Wireless Security Lab & Open BTS

Final Report
12/09/2013
Iowa State University

Senior Design, Team 13-14:

Xiaofei Niu
Thuong Tran
Matt Mallett
Yuqi Wang
Chris Van Oort
Muhammad Tahsinur Rahman Khan

Client/Advisor:

Dr. George Amariucai

Table of Contents

1) Executive Summary

This document details the final design and implementation by Senior Design team Dec 13-14 of a remotely-accessible laboratory environment for Computer Engineering 537: Wireless Network Security.

The course teaches concepts that may be difficult to understand without first-hand experimentation. By introducing a laboratory element, students would be able to gain a greater understanding of the concepts of the course, and would be able to perform privacy- and security-sensitive experiments in the safety of a controlled environment.

The system will consist of not only the hardware implementation to support wireless security experimentation but also laboratory experiments designed to complement the lecture material. Students will be able to experiment with the laboratory environment on their own, but will also be guided through structured laboratory procedures.

From a hardware perspective, virtualization will be used to support many machines on one physical host. This reduces costs and increases system efficiency. Machines will be grouped in twos, one machine creating legitimate traffic on a network while another performs any number of operations on it, ranging from listening to injecting data to causing transmission errors. Labs will be attempted to investigate vulnerabilities in Wi-Fi. Labs in other wireless standards will be enabled by utilizing software defined radios.

The lab must support multiple users, be accessible via GUI from a remote machine either on or off campus, and have ample documentation for both students and faculty administrators. It must also be reliable and cost-efficient.

2) Background Information and Concept

This project is designed to add additional learning opportunities for students of Computer Engineering 537: Wireless Network Security. This course introduces several common wireless communication standards including Wi-Fi, Bluetooth, and GSM, and then examines the vulnerabilities present in each communication system and the measures that can be taken to counter attacks on these vulnerabilities. Topics of discussion include cryptography and information theory.

Currently, the course material revolves primarily around homework. There is no laboratory element. However, the nature of the class material lends itself to hands-on student investigation and experimentation. Without a university sponsored environment for conducting this experimentation, the student is forced to source their own equipment; if this is unfeasible, the student may miss out of valuable learning opportunities.

A laboratory environment is therefore desired, to provide a sandbox in which students are provided with tools and machines which allow them to apply their knowledge to novel problems. This safe, isolated environment would give all students equal access to the equipment necessary to duplicate concepts discussed in lecture.

Ideally, the equipment would be housed in a physical laboratory, where students could work with it directly. However, many students enrolled in the course are Distance Education students, taking the class remotely over the Internet. Assembling a physical lab on campus would be useless to these students. The goal is therefore to make all aspects of the lab remotely-accessible, such that any student can access lab resources from any location at any time.

3) Conceptual System Description

The laboratory structure will be reconfigurable to suit the different wireless system architectures under study.

For experiments dealing with Wi-Fi, a two-node setup will be pursued, with each node equipped with a Wi-Fi interface. In this configuration, one node acts as a legitimate client, transferring data across a Wi-Fi network via a router. The second node is preloaded with security tools and will be capable of executing a variety of exploits and tests on the network, including packet sniffing and injection. This functionality will be implemented through a combination of compatible hardware and commercially- or freely-available software packages.

Students are given full access to their laboratory machines, including root credentials where necessary. Access is only restricted where it is infeasible to grant it due to the probability of generating problems that cannot be fixed remotely, such as in the wireless access points. This will simplify administration of the environment while granting maximum freedom to the students in designing and executing experiments.

To utilize the available hardware resources in the most efficient way possible, as well as to provide the faculty administrator of the laboratory environment with the greatest ease in configuration, a hypervisor-based virtualization scheme will be used to implement student machines. Virtualization allows one physical machine to appear as a large number of virtual machines, each able to host an independent operating system, share resources with the host and other virtual machines, and allow remote access for lab users. It also allows the user base to expand modestly without the need to purchase and configure additional hardware.

A caveat to the use of virtual rather than physical machines is that in order for many of the software tools relevant to the lab to function, they usually require low-level access to the wireless interface equipment. Because virtualization technology abstracts the physical hardware from the virtual machines, this issue must be taken into consideration. There are two ways to approach the problem, either using hardware and software which support low-level I/O virtualization, or leveraging the compatibility of devices on the Universal Serial Bus with many modern hypervisors. The hypervisor chosen for this project was VMware vSphere Hypervisor 5.1, a powerful enterprise-class host which has an agreeable licensing model for this type of implementation. VSphere Hypervisor supports both USB and low-level I/O redirection, which means either approach could be implemented on the same base software. In implementation, USB devices were ultimately chosen due to driver support and expandability.

In addition to clients and attackers, two other machines are necessary. One is a USRP Windows node, which views signals transmitted from the USRP that is acting as a base station. The second is a

web and script host, an intermediary between the user and the hypervisor, which allow the user to execute low-level commands such as powering on the machine.

All machines implemented in this laboratory run a distribution of Linux, with the exception of the USRP node. The choice of Linux will allow fine control over student access privileges on the machines. It also ensures compatibility with the most popular wireless security related software packages. Kali has been chosen as the distribution for the attack machines due to its preconfigured tool set. Ubuntu was chosen for the clients for ease of use. The USRP node runs Windows 7 due to its compatibility with LabVIEW, a software program that allows us to view wireless signals.

The documentation for the laboratory environment is comprehensive. It needs to contain step-by-step instructions for the proper administration of the environment, as well as recovery strategies for if and when machines become corrupted. In case of catastrophe, complete hard copy backups of the final state of all machines are provided with instructions for rebuilding the laboratory environment if necessary.

4) Functional Requirements

Remote access for both on campus and off campus students

Remote access to the lab is the most important requirement. No student will have physical access to the lab and thus the lab must be fully accessible from a remote session. Since many of the tools that will be used in the lab have a graphical user interface, the remote session must also be capable of having a graphical user interface. This will be accomplished by utilizing NoMachine, a remote machine controller software.

Support for at least four concurrent users

Having multiple users running experiments in the lab at the same time is another core requirement. Without support for multiple users a student would have to wait for the lab to be free. This would be very frustrating for a student with a very strict schedule and should be avoided as much as possible. Four users were chosen as a balance between number of users and available non-overlapping 802.11 channels.

Support for WiFi experiments, with expandability for others

The hardware is provided to support a number of different experiments. USB radios were chosen, to some extent for cost but moreover to allow greater expandability in design, as USB devices are available for a wide range of radio types. Additionally, by limiting all the hardware to a single type of interface, the backend coding is greatly simplified.

No interference between users

The lab is designed in such a way that the wireless traffic from unrelated nodes cannot interfere. In

this way, a user can feel free to conduct any desired experiment without fear of disrupting other users.

Comprehensive documentation for both administrators and students

The laboratory includes a large number of powerful tools and complex scripts. Comprehensive documentation is necessary to ensure both administrators and students can use and configure the environment with ease.

Physical Machines vs. Virtual Machines

Installing a large number of physical machines can be very expensive and the machines themselves will have to be configured separately. A virtual machine server can be a very effective solution, especially when a large number of machines need to be set up and accessed remotely. These virtual machines can also be configured and administered easily using a common console. They also provide the added benefit that they can be replaced with an existing image if something goes wrong or the machine gets corrupted.

Single Adapter vs. Multiple Adapters

A single wireless adapter can potentially be used by multiple users connected remotely for only a very few select specific experiments. Many wireless security experiments require direct access to the wireless interface to bypass certain protocols and carry out an attack. Thus, each wireless adapter can be accessible by only one user at one point of time. Also, although the virtual machines can share a single Ethernet adapter, they cannot share a single wireless adapter. Thus each virtual machine will need its own separate wireless adapter.

PCI Adapters vs. USB Adapters

PCI adapters have been available in the market for a long time now and have better support and troubleshooting for attack experiments. However, USB adapters are starting to be used more and more extensively nowadays and offer the added advantage that they can be connected to the VM server using extensible USB cords such that they can be spaced apart to limit interaction and interference between the radio waves. Additionally, server machines often do not come with a very large number of PCI slots and USB adapters may be necessary to support multiple adapters for the multiple VM's.

Windows OS vs. Linux OS

Windows OS has arguably better support for remote access and is the more widely used operating system. However, Linux OS offers better flexibility, has better support for wireless tools, can be easier to configure and is an open source solution. Windows users can take some time to get used to the Linux environment; however, Linux offers much better support for wireless tools and can therefore be considered the better option for carrying out wireless security experiments.

5) Non-Functional Requirements

User friendly access interface

The environment must be easy to use. It is pointless to design a very powerful set of tools that are impossible to get running. Therefore, both web interface and the machines themselves will be set up to be very easy to use, and self-configuring wherever possible.

Adequate network bandwidth

The system must be designed such that even at peak usage, it remains responsive for all users. Whereas eight or more remote graphical sessions may be running on the host at the same time, the network capabilities must be designed to ensure optimal usability.

Adequate system resources

Similar to bandwidth, the system should be fully usable and free of excessive lag even at peak loading from multiple users. The host must therefore be configured with enough resources that each machine can make use of what it needs to operate.

Real world network simulation

The environment must emulate a real network, or there is no purpose in running experiments on it. There are no software-based wireless network simulators that are usable in the way requested by the client, necessitating the design of this laboratory environment using real, physical networking equipment.

6) Low-Level System Architecture

The full-system architecture consists of three distinct but related sub-architectures: hardware, software, and network. The hardware architecture consists of all physical components necessary to properly implement the requirements of the project (e.g. computing equipment, radios, network interconnect hardware, and any custom-built hardware deemed necessary). The software architecture consists of all non-physical components of the project (e.g. virtualization environment, operating systems, exploit tools, administration services, user scripts, and device firmware for any modified or custom device in use). The network architecture contains some components of both hardware and software architecture, but abstracts these components to deliver a node-centric view of inter-device communication paths and protocols.

6.1) Hardware Architecture

The hardware architecture was designed with the intent of employing commodity hardware in creative ways in order to fulfill the unique needs of the project in a cost-effective way. Attempts were made to consolidate hardware wherever possible to trim costs and ease administration. The end result is robust yet easy to use and expandable should future need arise.

The backbone of the project consists of two commodities x86-based servers. These will serve as the sole compute nodes for all project-related services and users. The servers will be sufficiently powerful such that no student's use of the environment will noticeably impact its usability for any other student, nor will the necessary backend services disrupt or be disrupted by student usage. Due to materials on hand, a single server based on Intel's Xeon 5500 series processor technology and architecture was chosen as the initial test bed for this project. Xeon 5500 is a maturing series (which leads to lower costs), but still utilizes Intel's current-generation Nehalem core for an excellent power to cost ratio. Paired with triple- channel DDR-1333 and a mainboard supporting advanced Intel Virtualization Technologies, this is a very capable architecture for a multi-user, multi-service environment. The full technical specifications for the test server are listed below; however, it should be noted that the final implementation will utilize two machines of possibly lower performance, in order to spread the I/O load.

6.1.1) Base Testbed Specifications:

Processor:	Intel Xeon 5504, 4x2.0GHz, two CPUs
Mainboard:	Supermicro X8DTL-3F X58-chipset board, with IPMI
Memory:	12GB ECC DDR3-1333, 6x2GB
Storage:	Western Digital RE3, 500GB, two drives, JBOD

In addition to the core systems, peripherals are necessary in order to communicate with the wireless networks under test. The Xeon 5500 series architecture provides support for Intel Virtualization Technologies for Directed I/O (VT-d), which allows virtual machines to have direct access to the PCI

bus. This allows for the ability to access PCI- or PCI Express-connected Wi-Fi cards as virtualized hardware, which would not be possible on a platform which does not support VT-d. Using Atheros PCI-based interfaces enables the utilization of the very well-supported open-source driver package ath5k, which by virtue of being open may be able to be modified to generate novel attacks. Ultimately, however, the team has chosen to utilize USB-based dongles from Ralink, which has a less-supported but similarly open-source driver, and eliminates the need for a platform supporting VT-d or a motherboard with more than four PCI or PCI Express slots. Banking on the use of largely independent host controllers (motherboard controllers supplemented by PCI Express-based add-on card controllers), the USB devices should experience no more noticeable latency than the PCI solution, and provide a comparably priced, more convenient alternative.

In addition to base computing resources, several other hardware devices are necessary to complete the lab environment. A number of wireless access points are required in order for the devices to speak to one another and for many of the attacks to be properly carried out. Some method of analyzing the signals present in the laboratory – whether spectrum or waveform – was requested by the client as a means of better understanding the attacks taking place. Finally, hardware for any protocol being studied other than Wi-Fi – such as Bluetooth – would also need to be made available.

The choice of hardware type and interface, and equipment multiplicity, has been carefully weighed between price, performance, user experience, and implementation practicality. The following summarizes the pros and cons of several implementation scenarios for Wi-Fi hardware; similar tradeoffs exist with other interface options (such as Bluetooth).

6.1.2) Software Defined Radio

A software-defined radio system, or SDR, is new technology for implementing radio communications systems. Components that have been typically implemented in hardware are instead implemented by means of software on a personal computer. Implementations of radio communications in software have lots of advantages over traditional hardware implementation. Some of the advantages are:

-Makes communications systems reconfigurable (adapting to new standards), keeping the hardware configuration the same.

-Upgradable and modifiable. Whereas, traditional hardware with fixed design, can only provide limited implementation of filters.

-Flexible enough to avoid the limited spectrum assumptions.

-Ability to implement cognitive/smart radio.

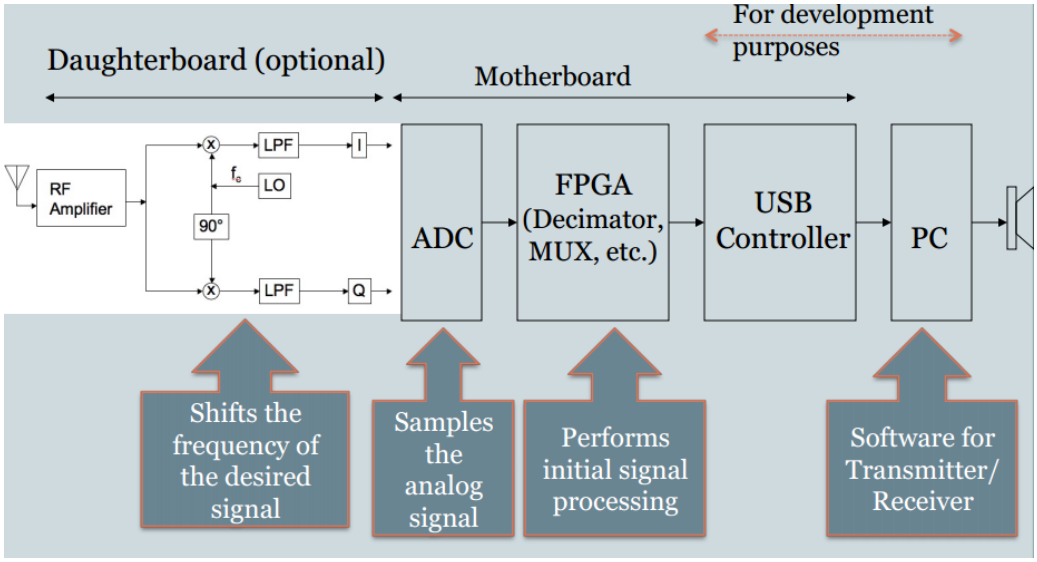
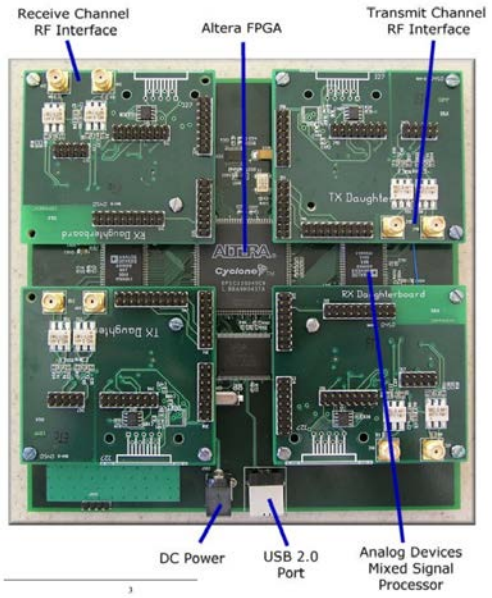


Figure4: Components of a SDR system

6.1.3) USRP

Universal Software Radio Peripheral products are computer hosted software radios. They can be connected to a host PC through a high-speed USB or Gigabit Ethernet Link and can turn the standard host PC into wireless prototyping platform. USRPs are commonly used with the GNU Radio/LabView software suite to create complex software-defined radio systems.

The USRP product family includes a variety of models that use a similar architecture. A motherboard provides the following subsystems: clock generation and synchronization, FPGA, ADCs, DACs, host processor interface, and power regulation. These are the basic components that are required for baseband processing of signals. A modular front-end, called a daughterboard, is used for analog operations such as up/down-conversion, filtering, and other signal conditioning. This modularity permits the USRP to serve applications that operate between DC and 6 GHz.



Figures5: Snapshot of an USRP

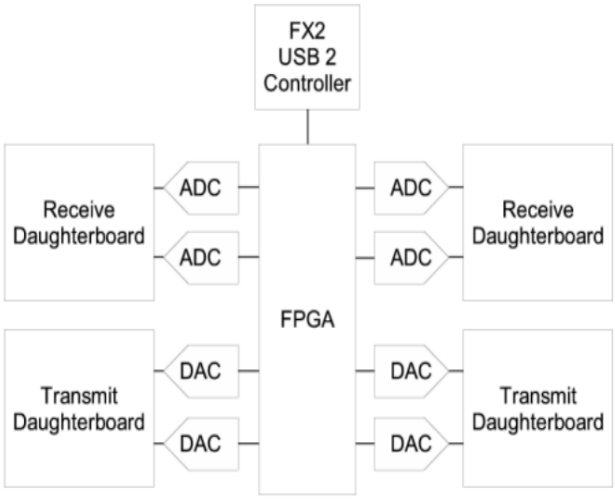


Figure6: Schematic diagram of the hardware layout:

6.1.4) BTS (base transceiver station)

A base transceiver station (BTS) is a piece of equipment that facilitates wireless communication between user equipment (UE) and a network. UEs are devices like mobile phones (handsets), WLL phones, computers with internet connectivity, WiFi and WiMAX devices and others. The network can be that of any of the wireless communication technologies like GSM, CDMA, Wireless local loop, WAN, WiFi, WiMAX, etc.

BTS works by regularly sending beacon signal in its coverage range, register the mobile station in its coverage and as soon as the mobile station invokes service, a free channel is assigned to it. MS (mobile station) sends its voice or data signal to BTS and BTS sends it to BSC (Base Station Controller) and BSC sends it to MSC (Mobile Switching Center)and MSC connects to the other side Mobile Station/PSTN phone/ or connects to SMSC (Short Message Service Center)if the service is for SMS or SGSN (Serving GPRS support node) for internet service. Thus BTS is the first contact for connection or release of a mobile service.

A BTS in general has the following parts: Transceiver (TRX), Power amplifier (PA), Combiner, Duplexer, Antenna, Alarm extension system, Control function and Baseband receiver unit (BBxx).

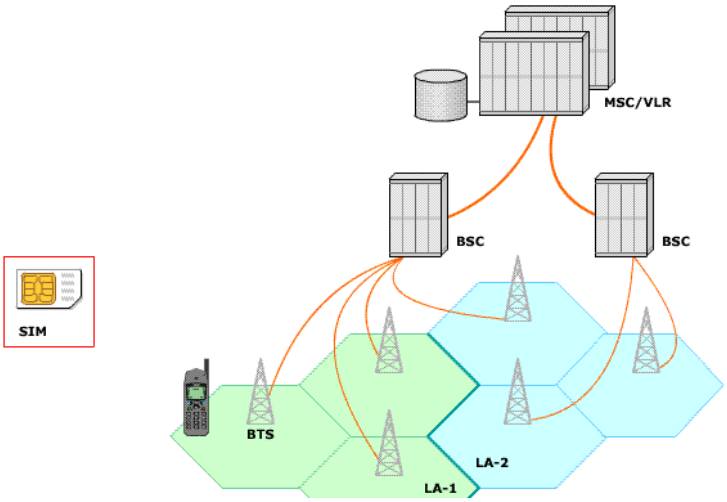


Figure7: Simplified diagram of cellphone network structure

6.1.5) OpenBTS (Open Base Transceiver Station)

OpenBTS (Open Base Transceiver Station) is a software-based GSM access point, allowing standard GSM-compatible mobile phones to be used as SIP (Session Initiation Protocol) endpoints in Voice over IP (VOIP) networks. OpenBTS replaces the conventional GSM operator core network infrastructure from layer 3 upwards. Instead of relying on external base station controllers for radio resource management, OpenBTS units perform this function internally. Instead of forwarding call traffic through to an operator's mobile switching center, OpenBTS delivers calls via SIP to a VOIP soft switch or PBX (private branch exchange).

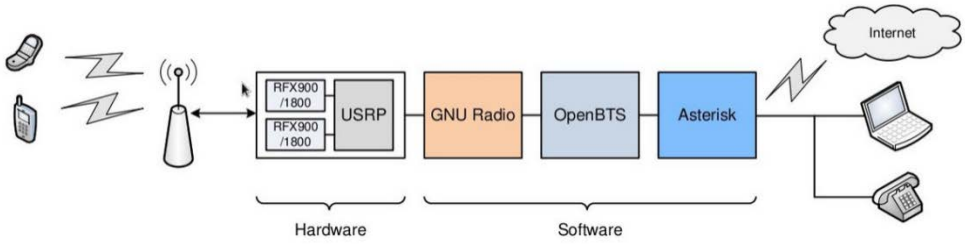


Figure8: One implementation of OpenBTS

In our senior design we tried to program the NI-USRP 2920 given to us using LabVIEW and GNURadio. Although we successfully programmed the USRP using LabVIEW, programming the USRP with GNURadio proved to be a challenge. The reason for our difficulty was: NI-USRP 2920 drivers, firmware and FPGA images are only compatible with Windows OS. The GNURadio and subsequent implementation of OpenBTS requires use of the USRP with Linux OS, because both GNURadio and OpenBTS are UNIX applications. We tried updating the drivers, firmware and FPGA images for the USRP, to make it interact with Ubuntu, but unfortunately it refused to respond to any Linux operation. The next thing we tried to do was install GNURadio using pre-built installers from Ettus Research. Although we were successful in getting everything installed in Windows OS, the installed GNURadio was missing a lot of libraries for

which there were no known fixes. The logical next step was to try using CYGWIN, a UNIX like shell for Windows, for installing and running GNURadio and OpenBTS. But the instructions and dependencies were not maintained for a long time, and as a result we were unsuccessful in that as well.
So, we concentrated on programming with LabVIEW and got our intended results. Using LabVIEW with the NI-USRP we could capture GSM signals during phone calls, and display the frequency spectrum and waveform of those GSM signals in a LabVIEW GUI we developed.

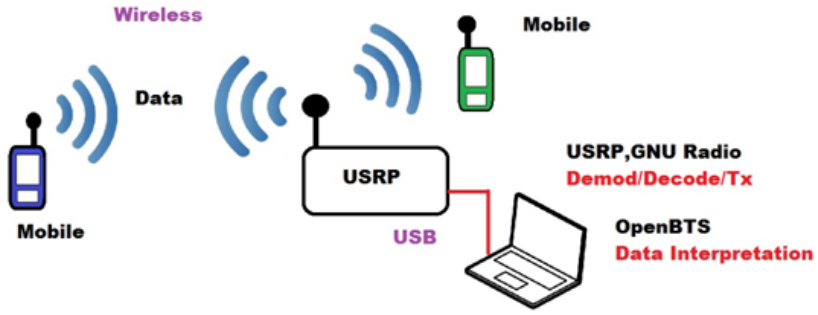


Figure9: Below we have tried to illustrate how we might go about implementing on what is required

6.2) Wi-Fi Hardware Scenarios:

Scenario 1: 1/1/1 Bridged

Transmitters:	1 Wi-Fi
Access Points:	1
Attack Nodes:	1 Wi-Fi (shared via Ethernet bridge)
Simultaneous users:	Many
Implementation:	One transmitter communicates over one access point. Attack node is accessible to users over a virtual Ethernet connection. Users send packets to virtual attack node, which relays them to the radio.
Pros:	Requires only two wireless interfaces and an access point, so cost is low. No worry about multi-channel interference. Assuming a sufficiently powerful host, would support every user simultaneously. Few privileges needed on the machine for use.

Cons:	Many attacks require driver-level access to the wireless interface, which is not accessible over a virtual Ethernet bridge, therefore this method severely limits the types of attacks which are possible. Users could write attacks which require unfair amounts of radio time, take down the network, or otherwise negatively impact other users' experiences.

Scenario 2: 1/1/1 Shared

Transmitters:	1 Wi-Fi
Access Points:	1
Attack Nodes:	1 Wi-Fi (shared)
Simultaneous users:	1 or Many
Implementation:	One transmitter communicates over one access point. Users simultaneously log on to attack node directly and generate attacks on local hardware.
Pros:	Requires only two wireless interfaces and an access point, so cost is low. No worry about multi-channel interference. Assuming a sufficiently powerful host, would support every user simultaneously. Local hardware allows driver-level access, so most attacks are feasible.
Cons:	Limited support for driver-level access by multiple simultaneous users; device locking by OS may allow only one user access to hardware at a time. If multiple users are allowed, users could write attacks that would require unfair amounts of radio time, take down the network, or otherwise negatively impact other users' experiences.

Scenario 3: 1/1/4 Independent

Transmitters:	1 Wi-Fi
Access Points:	1
Attack Nodes:	4 Wi-Fi
Simultaneous users:	4
Implementation:	One transmitter communicates over one access point. Four machines with radios are available for student use, one student per machine, first come first served.

Pros:	Requires five wireless interfaces and an access point; cost is fairly low. No worry about multi-channel interference. Potentially lowers the requirement for host power due to lower user count. Local hardware on single-user system uarantees unrestricted access to radio.
Cons:	Limited to four simultaneous users; others must wait for slots to open. A single target network means users can negatively impact other users' environment experience, which may be manifest in unpredictable results or unreliable environment operation.

Scenario 4: 4/4/4 Independent

Transmitters:	4 Wi-Fi
Access Points:	4
Attack Nodes:	4 Wi-Fi
Simultaneous users:	4
Implementation:	Four transmitters communicate over four access points. Four machines with radios are available for student use, one student per machine, first come first served.
Pros:	Each user may run any attack on the environment, including denial-of-service or jamming attacks, with little risk of disrupting other users. Local hardware on single-user system guarantees unrestricted access to radio.
Cons:	Limited to four simultaneous users; others must wait for slots to open. This volume of USB devices requires a hub or controller card. Cost is comparatively high. Interference between four independent networks may be a problem.

Given these choices, and the relatively low cost of the variable hardware being considered, it was ultimately decided to pursue a plan of independent networks with four transmitters, four access points and four malicious radios. This choice guarantees the greatest number of possible attacks and the best reliability for all users.

It is also desirable to be able to see waveforms of some of the radio activity to analyze network modulation schemes. This is accomplished by LabVIEW and the on the USRP node.

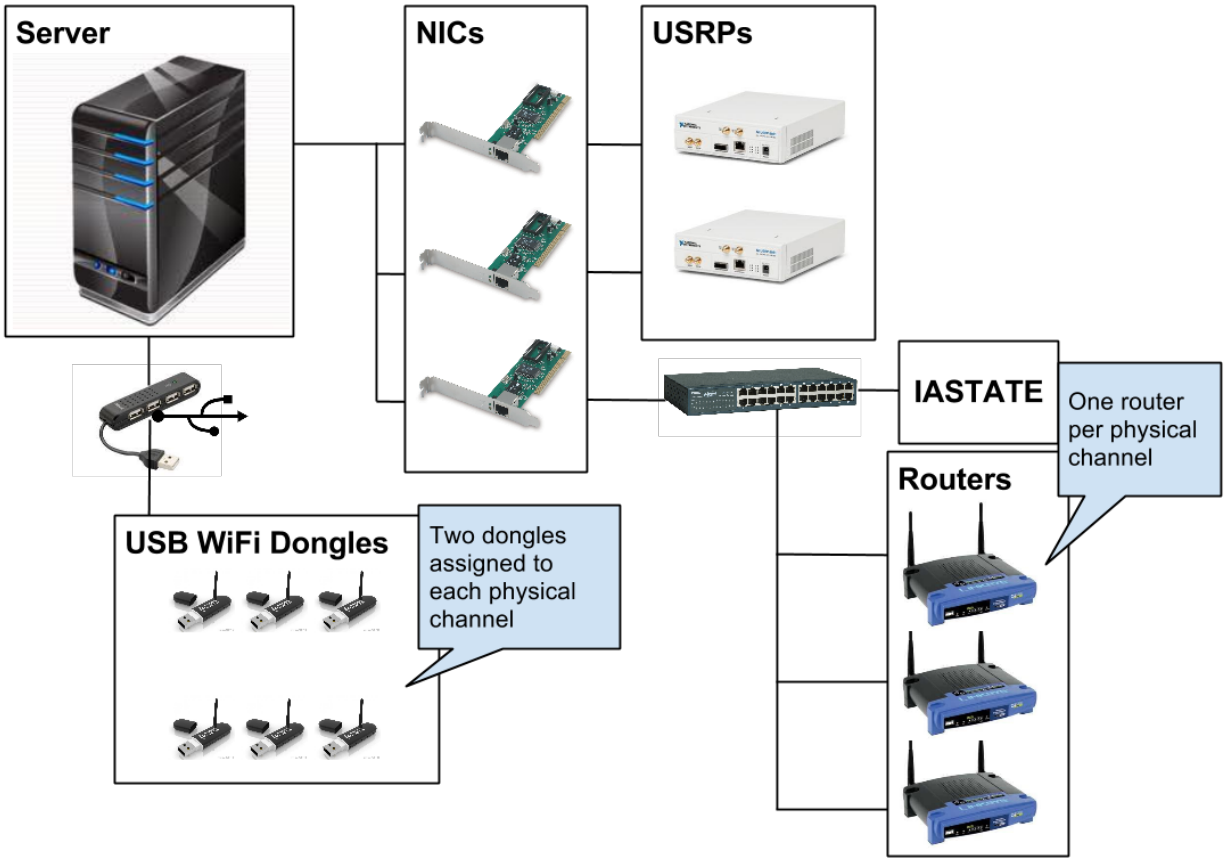


Figure 10: Full-system hardware architecture (server depicted as complete unit for simplicity)

The goal of the hardware section of this project is to implement a functional GSM OpenBTS (base transceiver station) network on the ISU network, using USRPs (universal software defined radio peripherals) from National Instruments. The project needs to implement a small-scale implementation of the OpenBTS (single USRP). The main functional requirement is that implementation of that the interfacing with the USRPs is done via LabVIEW.

6.2) Software Architecture

The software required to implement all features of this project is a mixture of operating systems, open-source utilities, custom shell scripts, web applications, and embedded firmware. Generally speaking, the operating systems and utilities are unmodified off-the-shelf products, while all of the supporting software is written by the team.

At the root of all software systems is the hypervisor. The hypervisor is installed as an operating system on the physical host, and emulates physical hardware to allow other operating systems to run

inside containers (“virtual machines”) within it. The software chosen for this project is VMware vSphere Hypervisor (formerly VMware ESXi), a mature, full-featured product which is licensed from VMware for no cost. It was chosen for its feature set, ease of management via vSphere Client, and a moderately strong team background in its use on top of the hypervisor, two sets of virtual machines are installed, one set per physical host. The first set is an array of machines running the Kali distribution of Linux, a security-centric distribution that comes pre-loaded with many of the tools necessary to run experiments on the lab environment. One Kali machine is created for each student that will be using the environment, to enable the student to create custom scripts, modify system files, and otherwise configure the system as much as they deem necessary to execute any particular attack, without affecting any other user. The second set is an array of transmitter nodes, again one per student. These will run the Ubuntu Linux distribution. A few additional machines will be configured in addition to the basic student-accessible machines.

Each machine that is designed to be single-user remotely accessed (all attack and transmitter nodes) will run NoMachine NX Server, which handles remote session serving. Alternately, users may log on via SSH using a terminal emulator if they do not require a GUI. The administration node will be given only SSH access. SSH access will also be enabled on the hypervisor.

The webserver node will act as the backbone for the operation of the environment. It acts as the bridge between the user’s requests and the hypervisor’s actions. The administration node runs an Apache web server, a MySQL database server and a PHP interpreter engine which hosts an information and control portal for the environment. The portal is the first step in a user’s access of the environment. After user authentication via PHP sessions, the user can perform a number of actions, such as power on or off virtual machines, launch the NX remote access plugin, reload a virtual disk from backup, examine system load and hardware usage, view and message users currently using lab resources, view lab-level data such as spectrum, and more. In order to accomplish hypervisor-level actions such as powering on or off machines, the portal executes console SSH commands via PHP.

A number of tools will be written for the hypervisor that facilitate remote user interaction as well as simplification of administration. The web server requests actions by calling these tools over a SSH connection to the server. Tools for user creation, deletion and modification for the portal, machine power state control, disk copy and virtual machine initialization, and all other necessary actions will be written as shell scripts and stored in the hypervisor’s file-system.

On the transmit machines, scripts will be written to automate the sending and receiving of data. This will negate the need for the student to use the machine directly, and add a level of repeatability and conformity to the environment. The machines will be scripted to log in and out of several common websites, transfer data to and from a remote machine, and perform other everyday user actions.

An overview of the backend software is shown in figure 11.

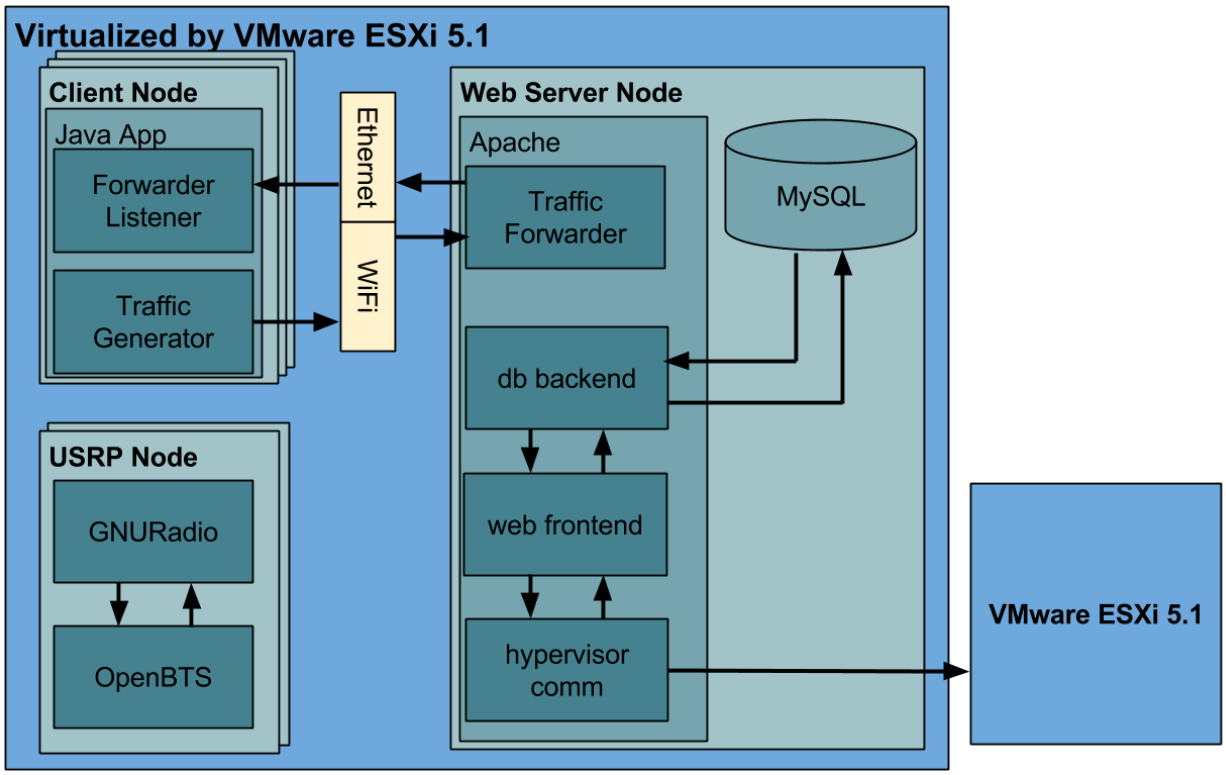


Figure 11: Overview of backend environment (duplicated host details omitted)

6.3) Network Architecture

The network architecture of the environment is designed to allow communication between machines where required, while segmenting separate network blocks and also allowing access to all machines from the outside world. This is accomplished using a combination of virtual and physical networking devices.

Each host is connected to the Iowa State network and thereby the Internet via a physical Ethernet interface. The physical Ethernet is connected to a virtual switch on the host, which in turn connects to all virtual machines. In the case of the attack nodes, this Ethernet interface is unrestricted and allows traffic on all ports; on the transmitter nodes, the Ethernet interface is restricted to only the ports required to contact the machine via SSH and NX via the firewall. Each machine also is connected to a physical USB WiFi device and by restricting traffic to SSH and NX on the transmitter, all HTTP and other traffic is forced across the wireless network for possible interception.

The array of routers is connected via a switch to a second physical Ethernet interface on the host. This is run through the firewall and eventually connects to the outside world. All administrative machines must also be given access to the Iowa State network in order to operate correctly, although they may be given access via the firewall machine with a lenient security profile implemented for some added security.

A simplified diagram of the network topology is depicted below.

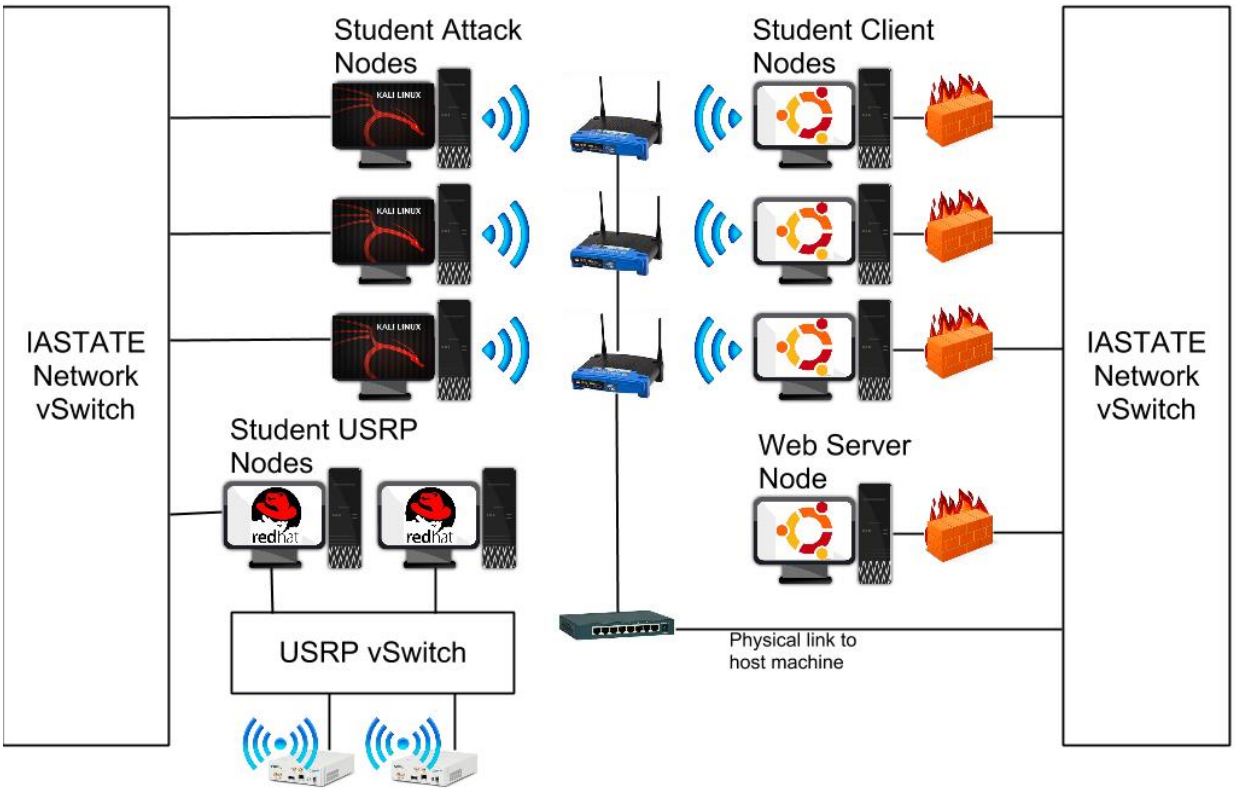


Figure 12: Full-system network topology

7) System Module Design

As much of the system is assembled from off-the-shelf components in standard configurations, extensive clarification is not necessary. As far as installation of physical components is concerned, brackets for secure wall, ceiling, or shelf mounting (depending on client specification) will be provided for radio components. The server itself will be installed either in a rack-mountable enclosure or a modular ATX case, again depending on final installation location and client request.

The virtual machines will be configured with a standard set of hardware using VMware's suggested settings for the guest OS, with the addition of wireless networking devices as necessary for the particular machine being designed.

The backend scripting framework utilizes standard Linux shell commands and VMware's own command-line tools to perform several hypervisor-specific functions. These commands are run on the hypervisor's console via SSH requests from the control node, which are generated using commands embedded in the PHP web interface to the environment and executed via PHP's shell_exec function. This function runs under privileges of the webserver user, but because the functions are relayed over SSH, the scripts are in fact run with the permissions of the remote host. A privileged user is created on the hypervisor to host these commands.

To handle authentication, the webserver user's RSA key is stored on the hypervisor. This prevents the server for asking for the privileged user password on every script execution.

The script architecture is designed to provide several important functions. First, it allows machines to be powered on and off. This is required to accommodate all users of the environment. With only four available radios, only four user machines can be connected to radios and powered on at any given time. The assignment of users to machines can be done in two ways; either four virtual machines can be created and all users can log on to these machines directly, or one virtual machine per user can be created and the radios dynamically assigned as users log on. The former scenario is much less complex, but users may be able to modify the environment in ways that affect other users' results. The latter requires more machines and some backend scripting, but allows each user a unique environment, which they can edit, at will, without affecting other users. This latter approach is what is attempted in this project.

When a user requests to use a machine, the scripts first determine which radios are currently in use. If there are none available, the user is not allowed to log on, or may log on without radios; if one or more is available, the user is allowed to choose a radio set to use. It patches the radio's configuration information into the virtual machine's configuration file and then powers on the machine. After the machine has booted, the user is able to use NX client, and they can begin to use the environment.

It is conceivable that users will switch radios from logon to logon. Boot scripts on each virtual machine will attempt to patch this by setting the interface MAC addresses and establishing environment variables to replace key radio-specific data. In this way, users' scripts can be written to run on any machine. Given the option to choose a radio set, users may alternately elect to wait until a specific radio is available in order to work on their scripts if problems should arise from changing hardware.

Because users will be given low-level access to the operating system, it will be possible for the user to damage their machine. The web interface will give the user the option to restore the virtual disk image from backup, using VMware's command-line disk cloning tools.

A Java application will run continuously on all transmitter machines. This application will continually run experiments such as logging in and out of common sites such as Gmail, Facebook and Twitter, transferring files, and connecting and disconnecting from the access point. These actions are performed in hopes of generating live, capturable data for lab users to experiment with. Some specific actions will be able to be requested by the user, such as switching encryption mechanisms, and the user will be

given access to the machine to generate custom data, but for the most part the transmitter nodes will act without user intervention for simplicity. The web server acts as a target node for this application, and will echo back traffic it receives through a simple php script. This gives students a view of what data made it across the network, and whether their messages had been tampered with.

A number of scripts will also be made available to the system administrator. These scripts will be responsible for initializing the lab for a set of users; user creation, modification and deletion; virtual disk and machine configuration; and other tasks. These may be run via direct SSH to the hypervisor, but will generally not be made available on the web interface for security.

8) User Interface Design

The backend of the site may actually run the environment behind the scenes, but the user interface is no less important. The tools and scripts will be wrapped into an attractive and easy to use website. The interface will be simple buttons and color-coded displays, and will be heavily documented. User authentication will be provided by PHP sessions in order to secure a user's own machine from use by others.

An important part of the project is the creation and walkthrough of common attack scenarios to carry out within the environment. These scenarios will tend to exemplify the use of one or a number of related tools, and will serve as tutorial examples that may be built upon by the user to create new and interesting experiments.

9) Use Cases

Login / Logout

When a user logs in the control machine looks up the password associated with the given username. It then hashes and salts the provided password and compares it with the value stored in the database. If the values match a PHP session is started and the user is then authorized to perform the actions defined by the type filed of the database table.

When a user logs out the PHP session will be destroyed and the user will have to log back in before he or she can perform any other actions.

Power on

The user will use the web interface to choose which USB radios to attach to each of their machines. Once the radios are selected the user will click the start environment button. This will cause the control machine to call the provision and boot script on the hypervisor. If the script returns success the users machines are powered on and the current time and date are stored in the lastsession field of the user table in the database. Each user is given a two hour session in which they will have sole use of the radios they selected when they booted their machines. In the last five minutes of the session the web interface displays a pop-up asking the user if they would like to renew their session. If they choose to

update their session then the lastsession field in the user table is updated. Otherwise, after five minutes the control machine with tell the hypervisor to power down the user's machines. This allows the control machine to automate the process of terminating idle users. Because the host machine has a limited number of radios, it is important to keep all resources not being used available to other users.

Change password

When the user clicks the change password link a form will be displayed with two fields, new password and confirm password. The user can then enter their new password in the first and second box. Once the user has submitted the passwords the control machine compares the two values. If the password contains any non alphanumeric values, if the passwords do not match, if the passwords are blank, or if the passwords are longer than 16 characters the user is shown a pop-up displaying the error. Otherwise the control machine will hash and salt the password and store the value in the database.

Add User

Users with admin credentials are allowed to add users to the lab. There are two different ways to accomplish this. Administrators can add one user at a time or upload a text file containing a list of user names and passwords. Each time a user is added the control machine makes sure the username and passwords contain only alphanumeric passwords, the username does not already exists in the database, and the username and password are between 1 and 16 characters. If any of these conditions are not met the user will not be added. Next the control machine invokes the 'verify and initialize' script on the hypervisor passing it the list of users. If the hypervisor response with success the list of usernames and passwords are written to a file on the control machine while the hypervisor begins the creation of the virtual machines for each user.

Remove User

The web interface is capable of displaying all the lab users currently in the system to an administrator. The administrator can then select with users to remove from the system. When the list of users is submitted to the control machine, the control machine invokes the destroy user virtual machines script on the hypervisor passing it the list of usernames. After the hypervisor is finished destroying each user's virtual machines the control machine removes the user from both the user table and the portdef table of the database.

Power Down

The web interface is also capable of display the state of all machines associated with each user in the lab. From this list an administrator can select which machines to power down. Once the list is submitted the control machine invokes the power off virtual machines script on the hypervisor for each selected user. The web interface then displays the result of the command to the administrator in a popup menu.

10) Testing

Creating Virtual Machines

Creating virtual machines for new users involves cloning an existing image, booting it, and configuring it over SSH.

Results – We are able to clone and build virtual machines on the fly over SSH on the ESXi server.

We are also able to perform power functions and queries (on/off/get status/delete).

Known Issues – Still working on getting and setting static IP information on the cloned virtual machines.

Adding/Removing Users

We ran several tests from both the hypervisor and web interface to test the functionality of adding/removing users and their virtual machines from the system.

Results – Users are able to be removed through the web interface. Their subsequent machines are also able to carefully be turned off, unregistered out of ESXi, VMDK's removed, and any associated files removed.

Known Issues – There are no known issues here.

Changing Account Passwords

We ran several tests with several different combinations of letters. The system does not allow alphanumeric characters, passwords longer than 16 characters, or blank passwords.

Results – The web interface allows users to change their passwords.

Known Issues – This may be vulnerable to cross site scripting injections, but more testing and security research needs to go into that to be 100% certain.

Powering Down Machines

Testing for this was done from both the hypervisor and the web interface.

Results – Machines are able to be powered down from the web interface as well as through SSH and the vSphere client.

Known Issues – There are no known issues.

Attaching Radios and Booting Machines

We tested attaching all possible combinations of radios to the virtual machines.

Results – We are able to add radios and hardware to the virtual machines manually through vSphere.

Known Issues – We are currently unable to dynamically allocate hardware to virtual machines. We are continuing to work on support for this.

Hardware testing with the USRP:

We programmed the NI-USRP 2920 to capture GSM signals. The carrier frequency range and I/Q rate was set at 1.85G and 1M for our final testing. We then made GSM phone calls near the USRP and the results were recorded in the spectrum and waveform graph, built into our LabVIEW GUI.

Results –

Fig 13: Without any (or negligible) GSM signals around the USRP

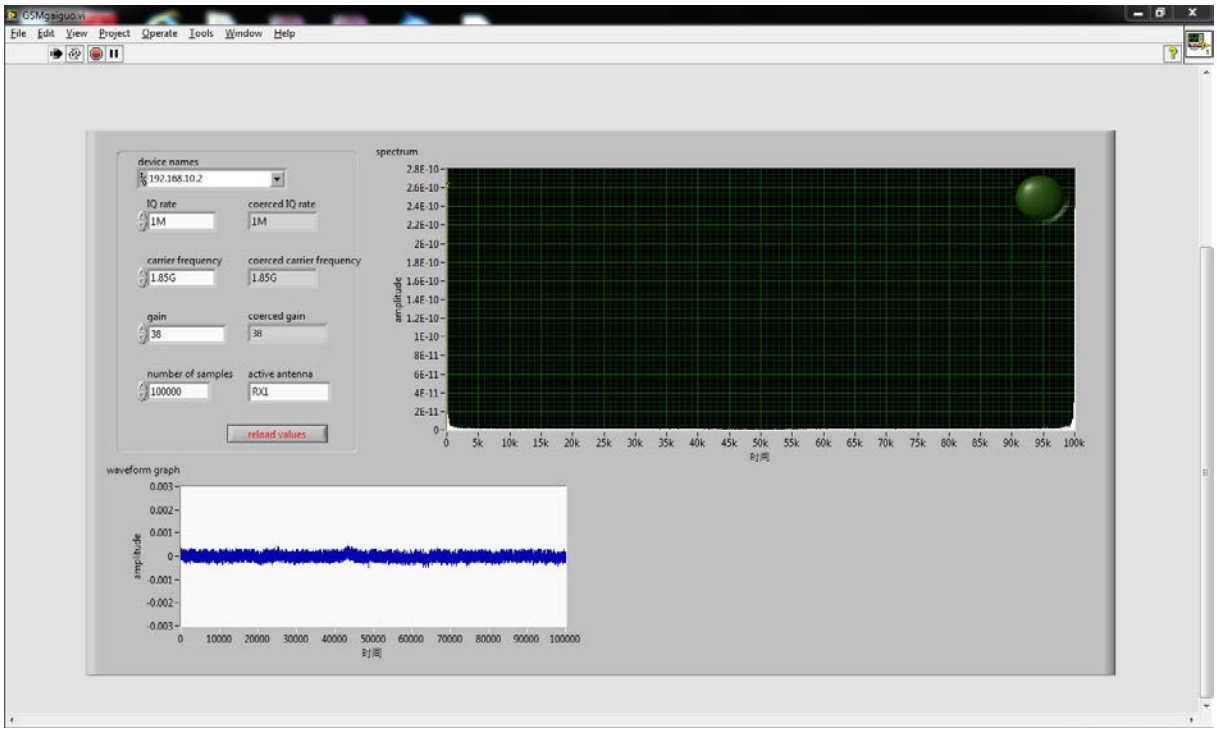
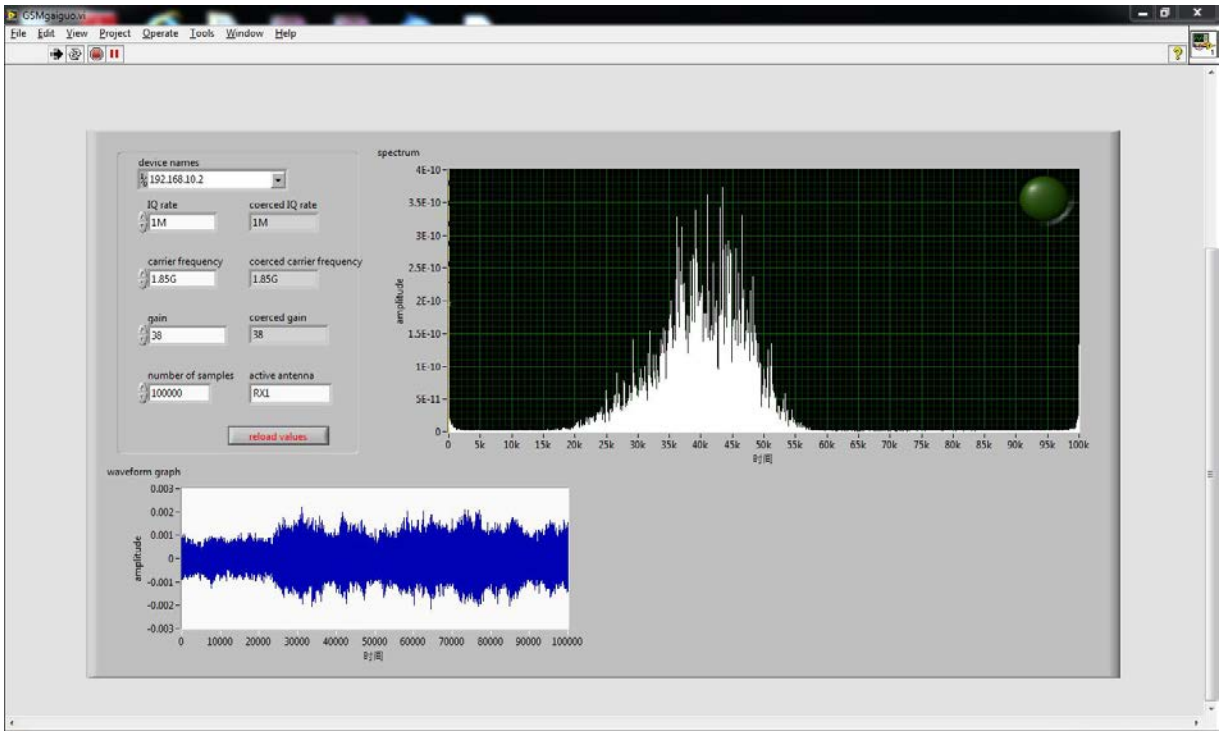


Fig 14: While making GSM calls nearby the USRP



Known Issues – We had to make sure to use the correct version of NI-USRP firmware and FPGA images. Earlier we uploaded the latest version of UHD firmware and FPGA images to the NI-USRP, in order to make it work with GNURadio and OpenBTS. That caused an error on the USRP boards, effectively making it irresponsive to our host machines. Then we booted the device into safe mode, manually added the device in the NI configuration utility, using the IP address and loaded the correct NI-USRP firmware and FPGA images. After that, our device started responding again and we could carry out our testing.

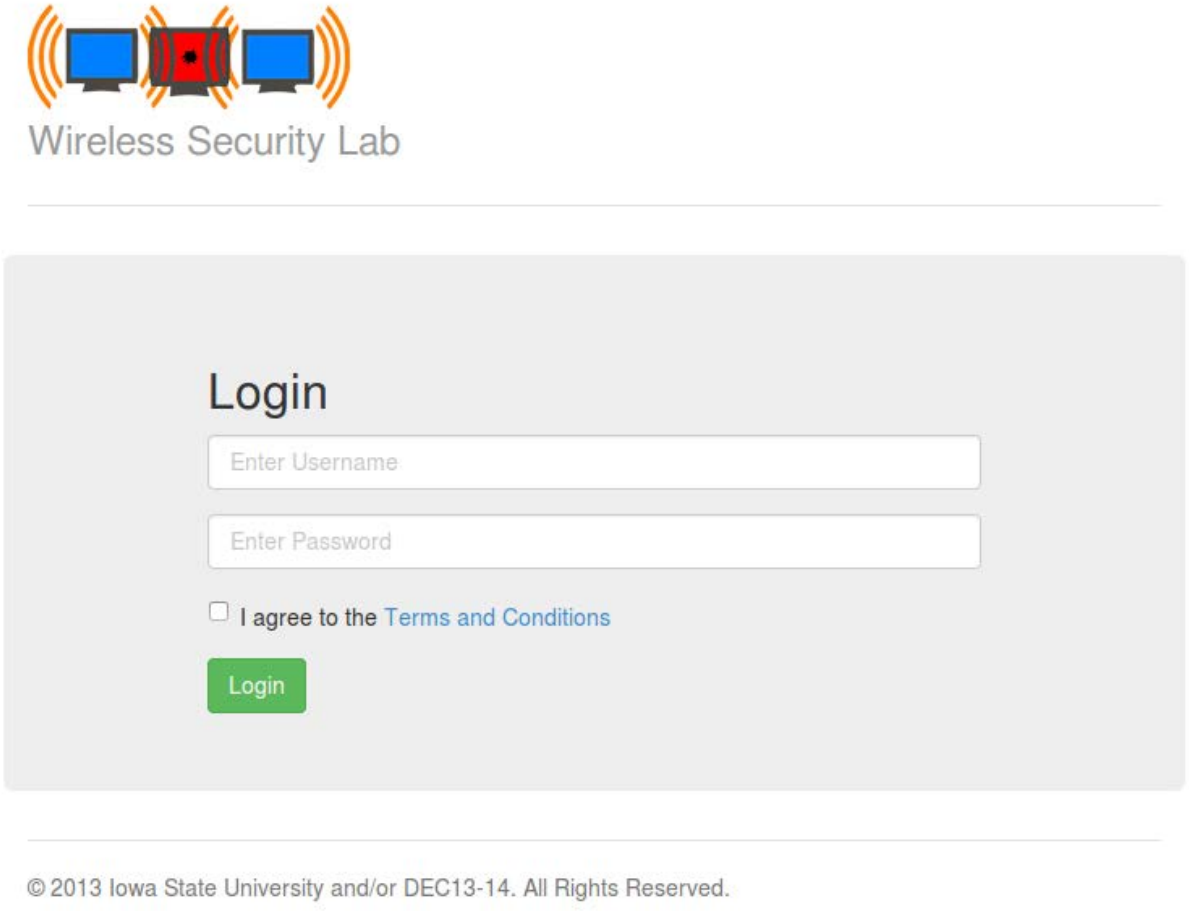
11) Conclusion

All of us came in with little to no understanding of the various software and hardware technologies that were presented in this project. We learned a lot about virtual machines and their management, as well as about how cell phone transmissions work. It was a good experience for all of in interacting with a client and trying to actualize their requirements. This project was grand and scope and is very worthy in being the capstone to our college careers. There is a great deal that has yet to be implemented, but the research and documentation that went into this project is quite extensive and thus would be easily continued by another Senior Design group. Overall, it was a great experience being involved with all aspects of a project from its inception to its completion.

Appendix
Operational Manual

Web Interface Instructions
The website contains two user modes: administrator and user. We will walk through all actions available to both.

Logging In
Navigate to https://129.186.215.201/wseclab [pictured below]
Enter credentials, click Log In



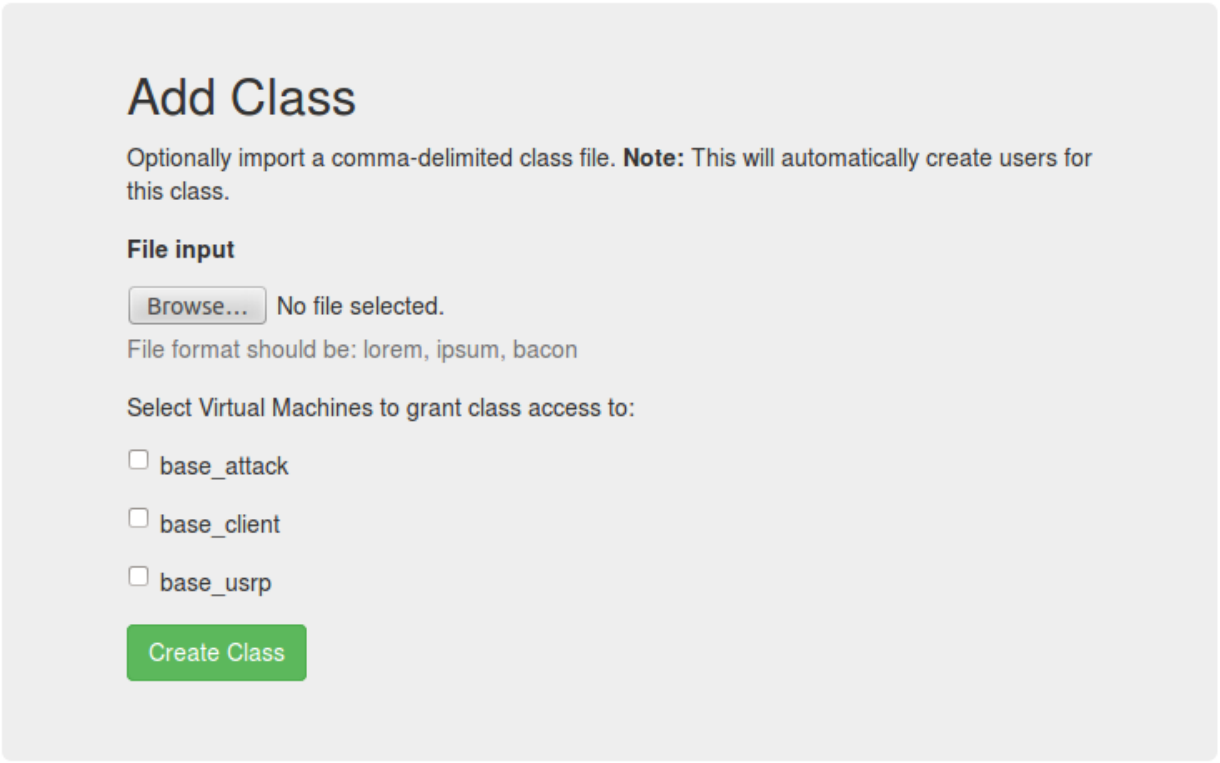
[Administrator] Managing Classes

After logging in, an administrator is automatically redirected to the class administration page. From here, an admin can add and delete classes, as well as perform class-wide actions such as removing all users from a class, and adding users to a class.

Section 1: Viewing/Deleting classes

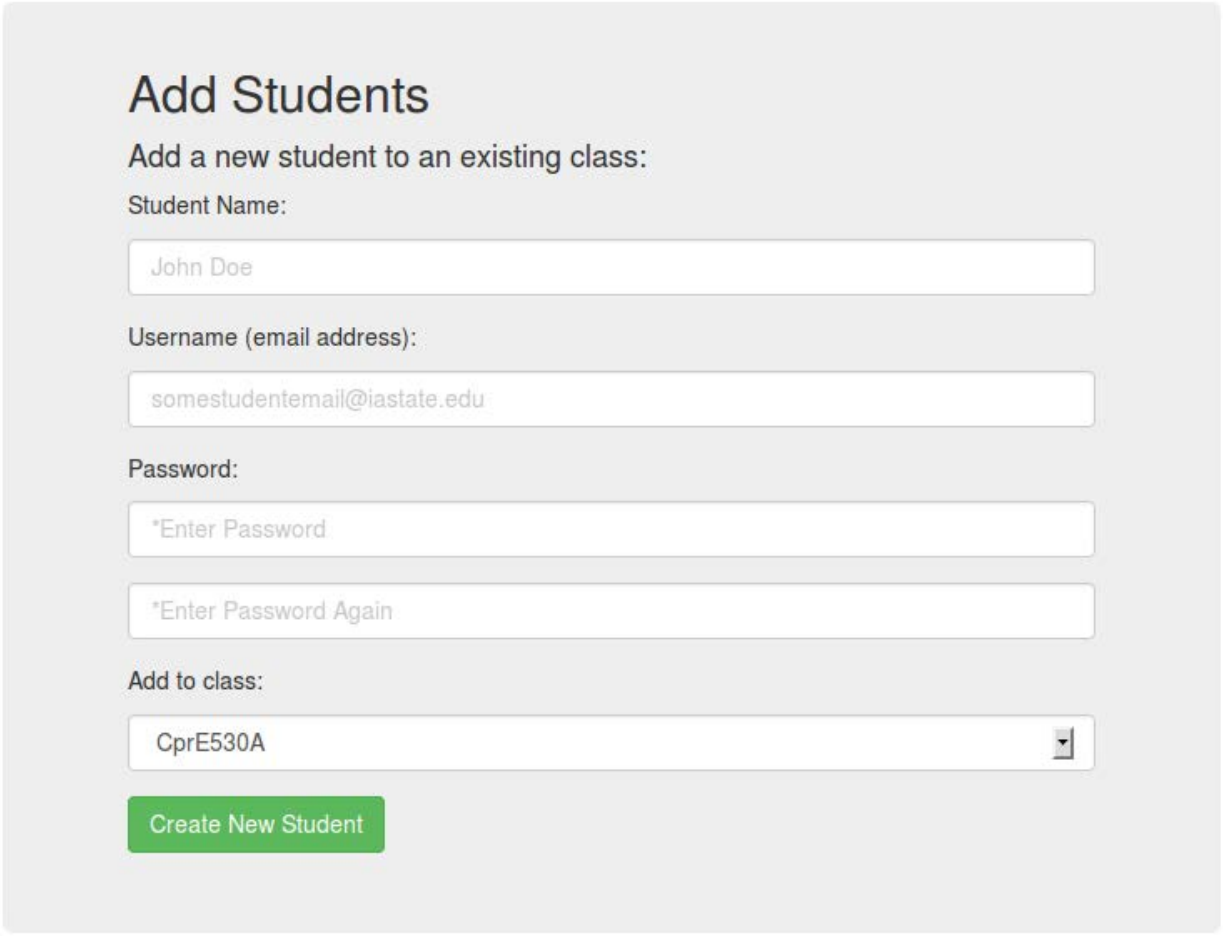
This section of the class manager page lists all classes. You can view more information on a class by clicking on its name, or delete the class and all associated user accounts and vms by clicking the corresponding Remove Class.

Section 2: Creating a new class



This section allows you to create a new class. The optional file input is a comma separated list of student names to be added as users in the system. Students will only have access to the virtual machines checked from the list of available types.

Section 3: Add Student to Existing Class

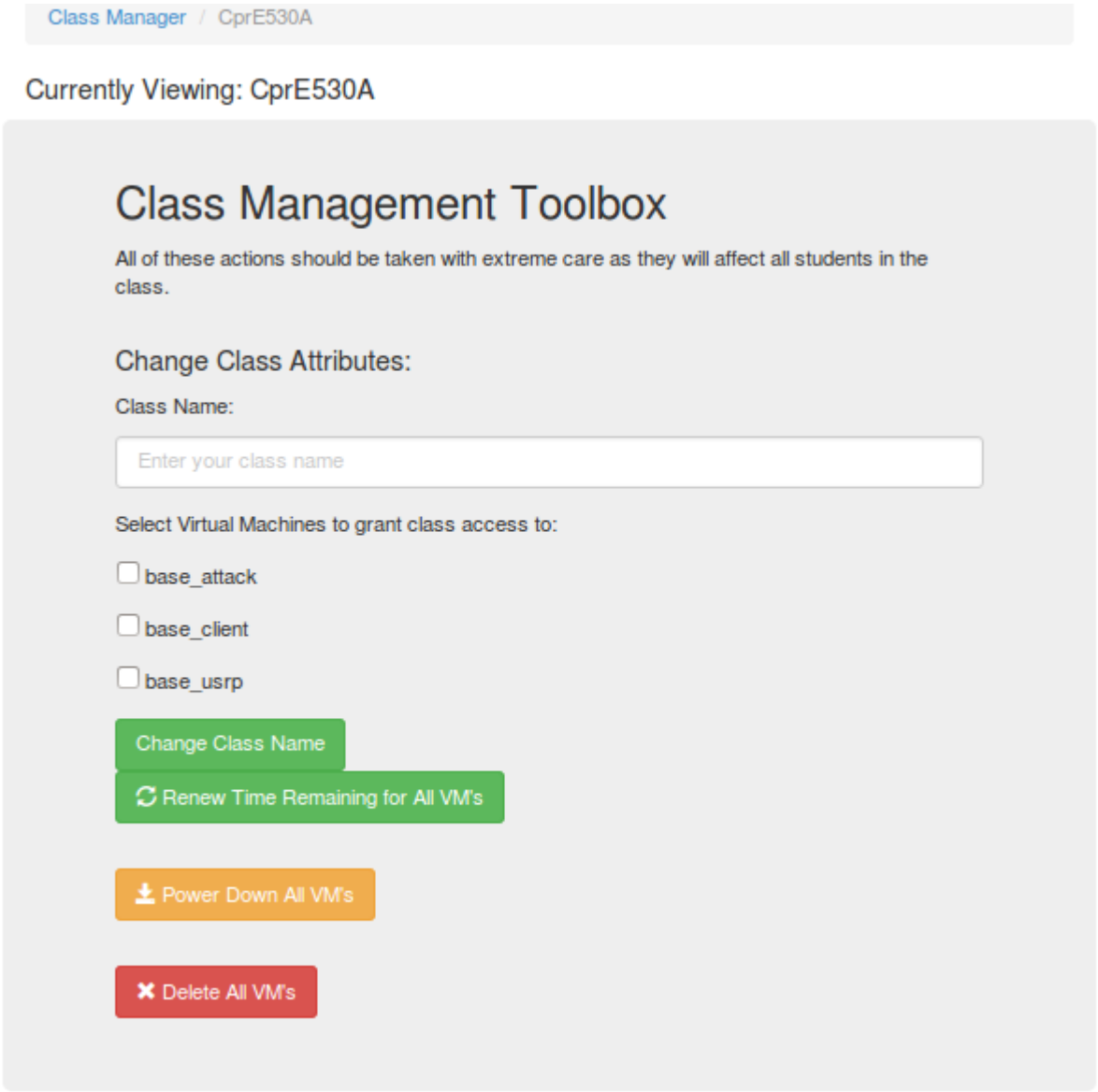


This section allows you to create a new user account and associate them with a specified class. Required fields are Student Name and Password.

[Administrator] Detailed Information on Individual Classes

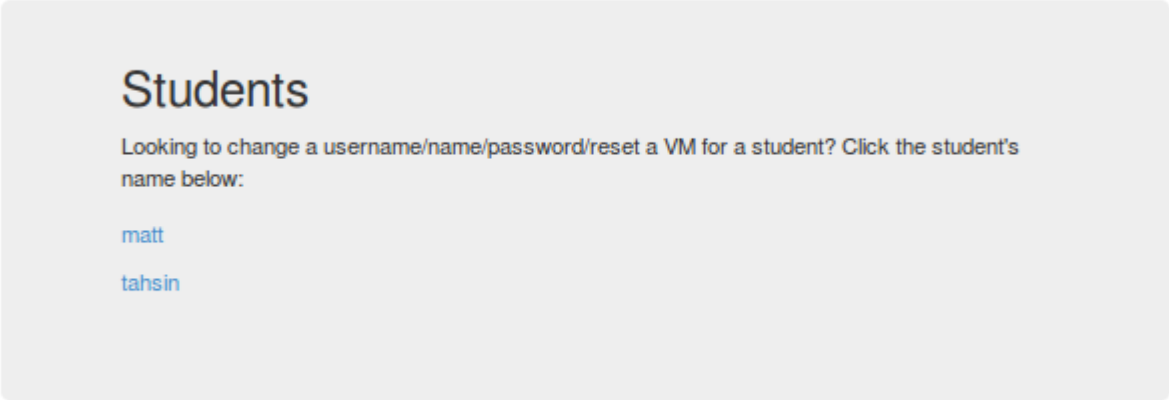
Clicking on a class from the list of all classes in the class administration page will lead you to the class view page. On this page you can view and edit detailed information on a specific class.

Section 1: Class Attributes and Class-wide Actions



In this section, the name and allowed machines of a class can be edited. In addition, buttons are provided to manage all vms of students associated with this class. An administrator can renew the time limit, power down, or delete all class vms.

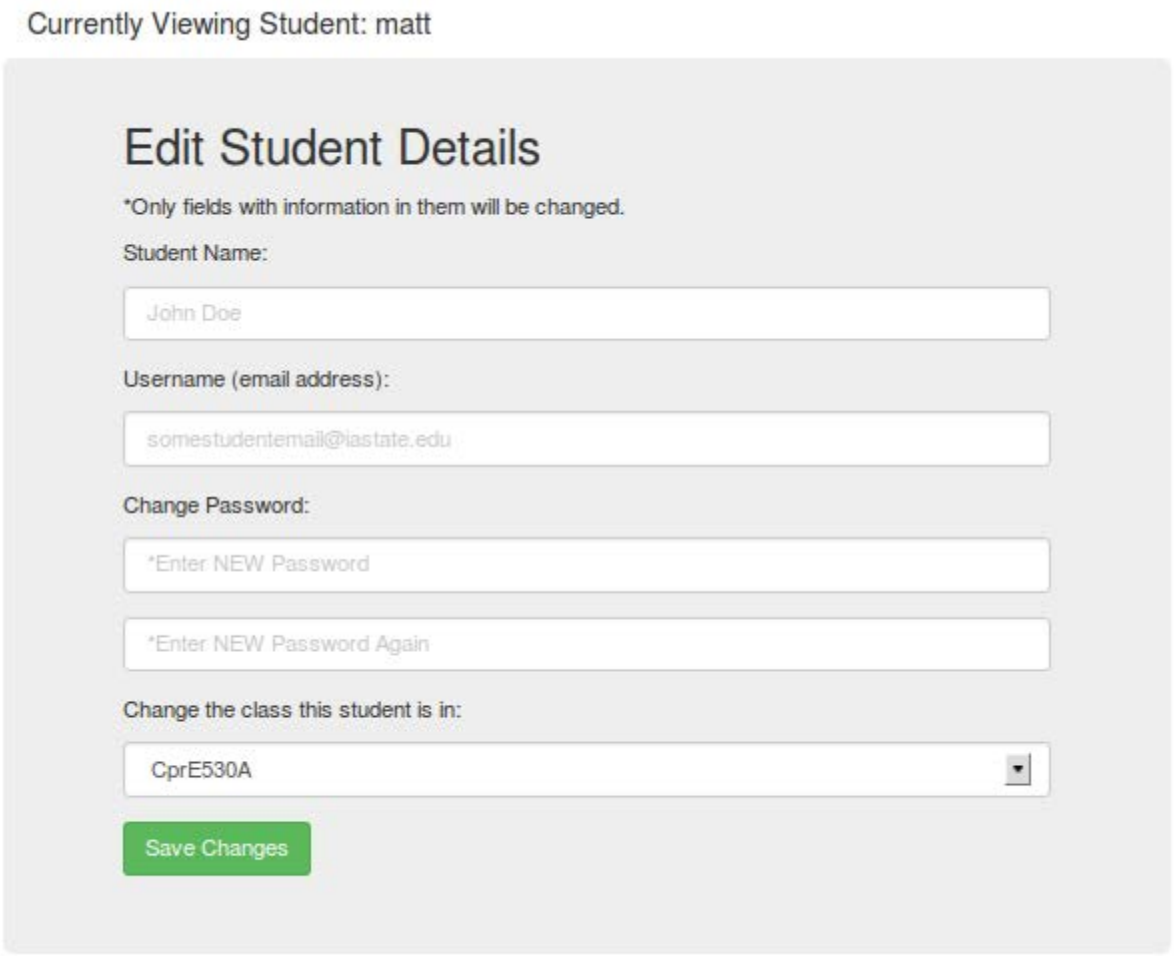
Section 2: List of Students in the Class



Here is a list of all students associated with this class. Clicking a student's name will open a page with detailed information on that student.

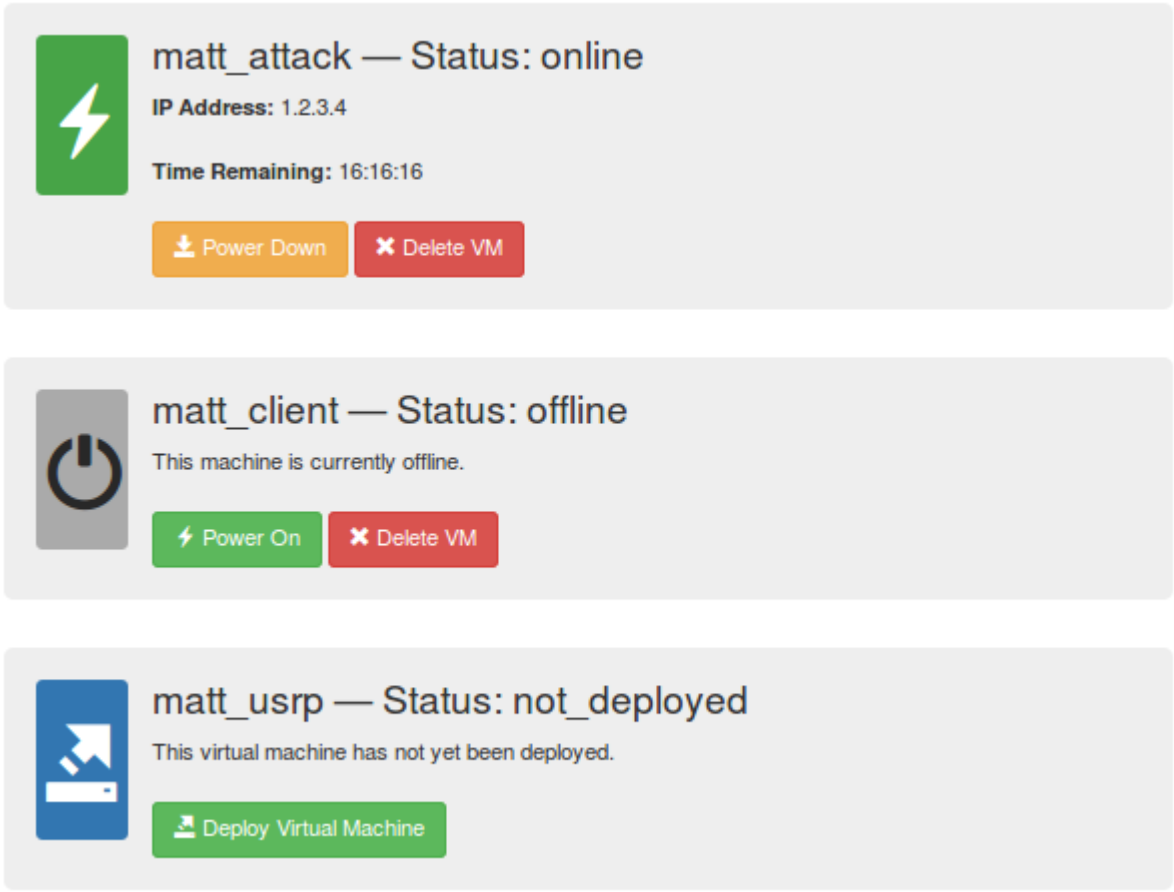
[Administrator] Detailed Information on a Student

Section 1: Student Details



This section allows an administrator to edit all information about a student.

Section 2: Manage a Student's Virtual Machines

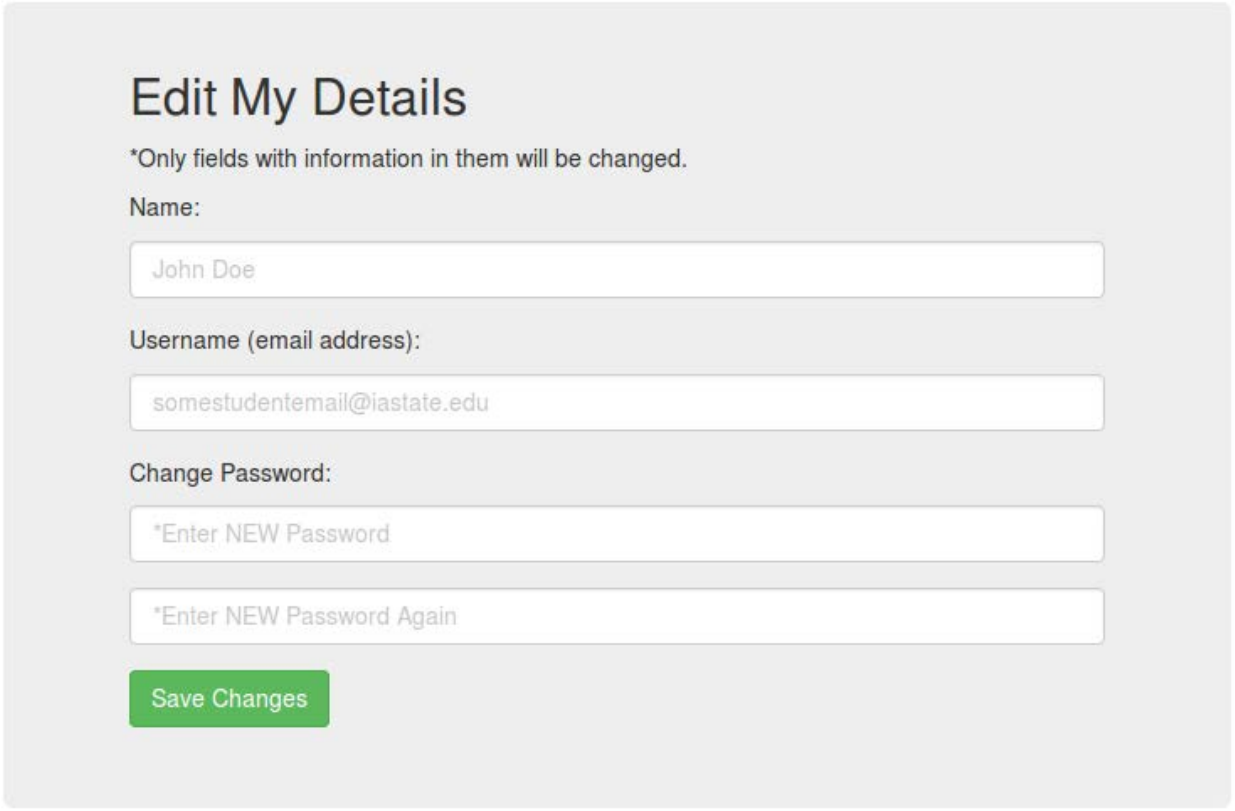


This section allows an administrator to create, manage and delete virtual machines for the student.

[Administrator] Manage Administrators

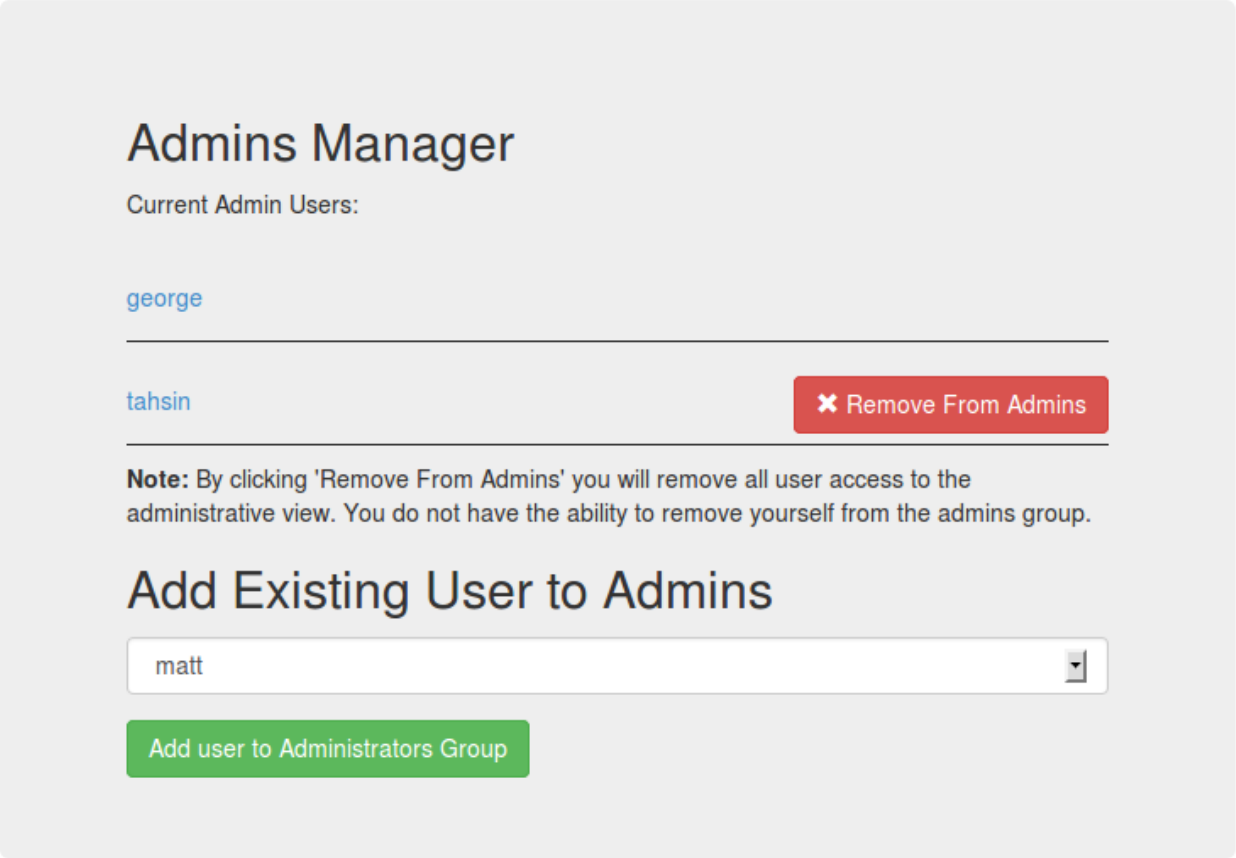
This page can be accessed by clicking the Admin Tools button in the top right menu.

Section 1: Edit My Details



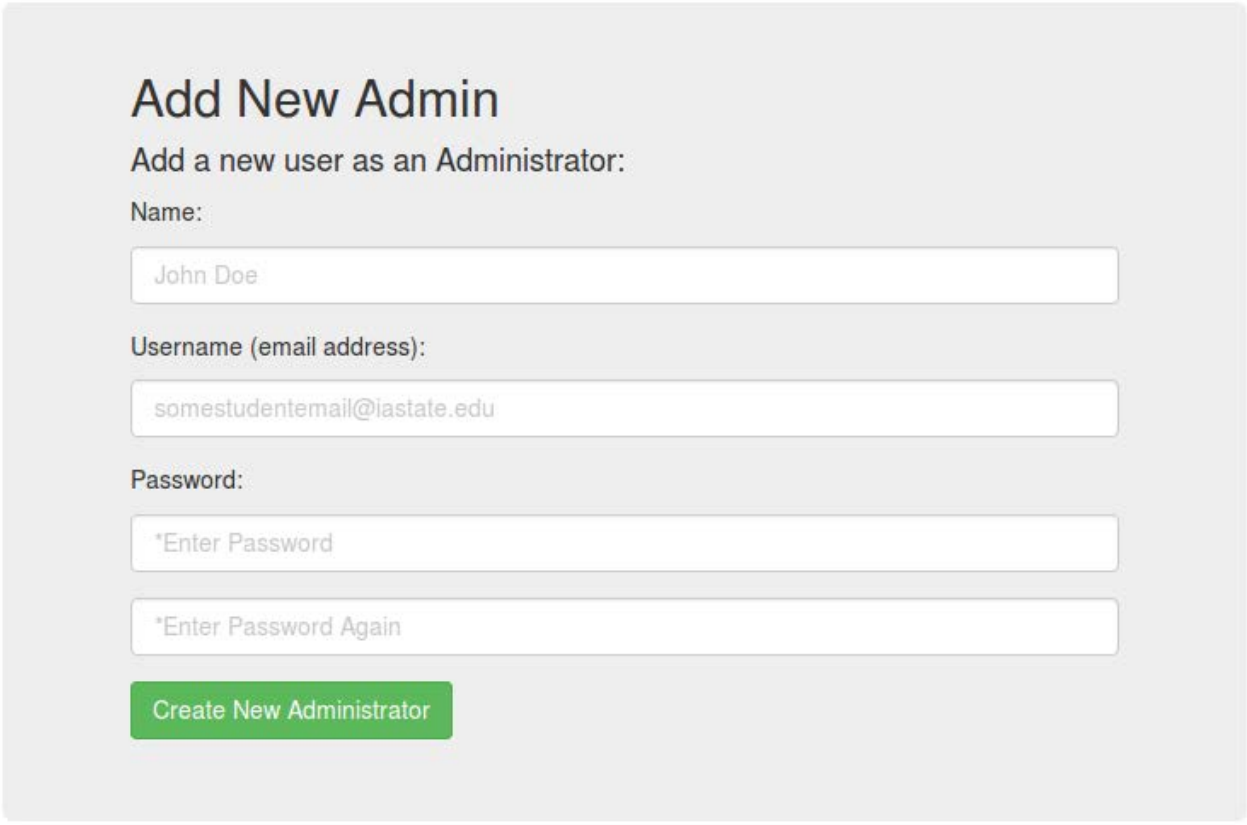
This section allows an administrator to manage their own user account information.

Section 2: Add/Remove Administrators



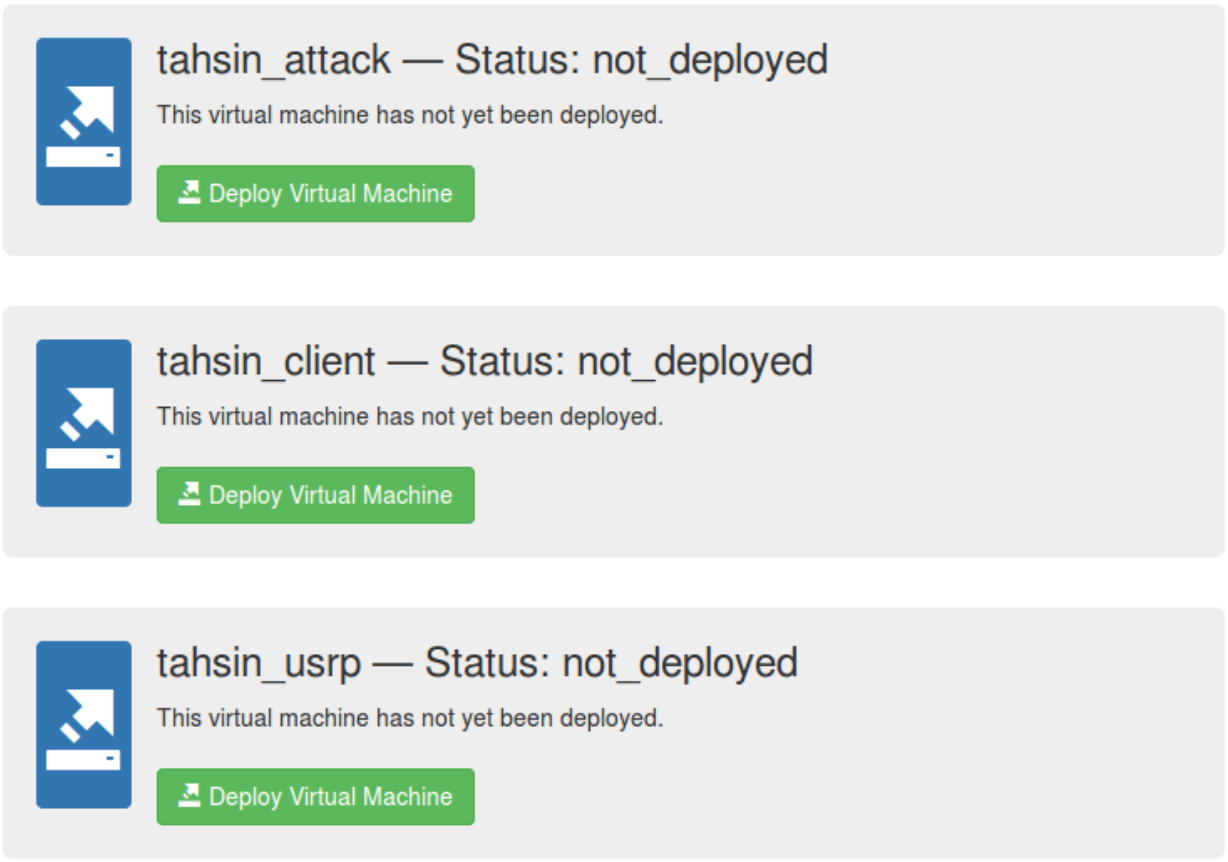
This section allows an admin to revoke administrative privileges from a current admin (you can't demote yourself). It also allows an admin to promote an existing user to an administrator.

Section 3: Create New Administrator



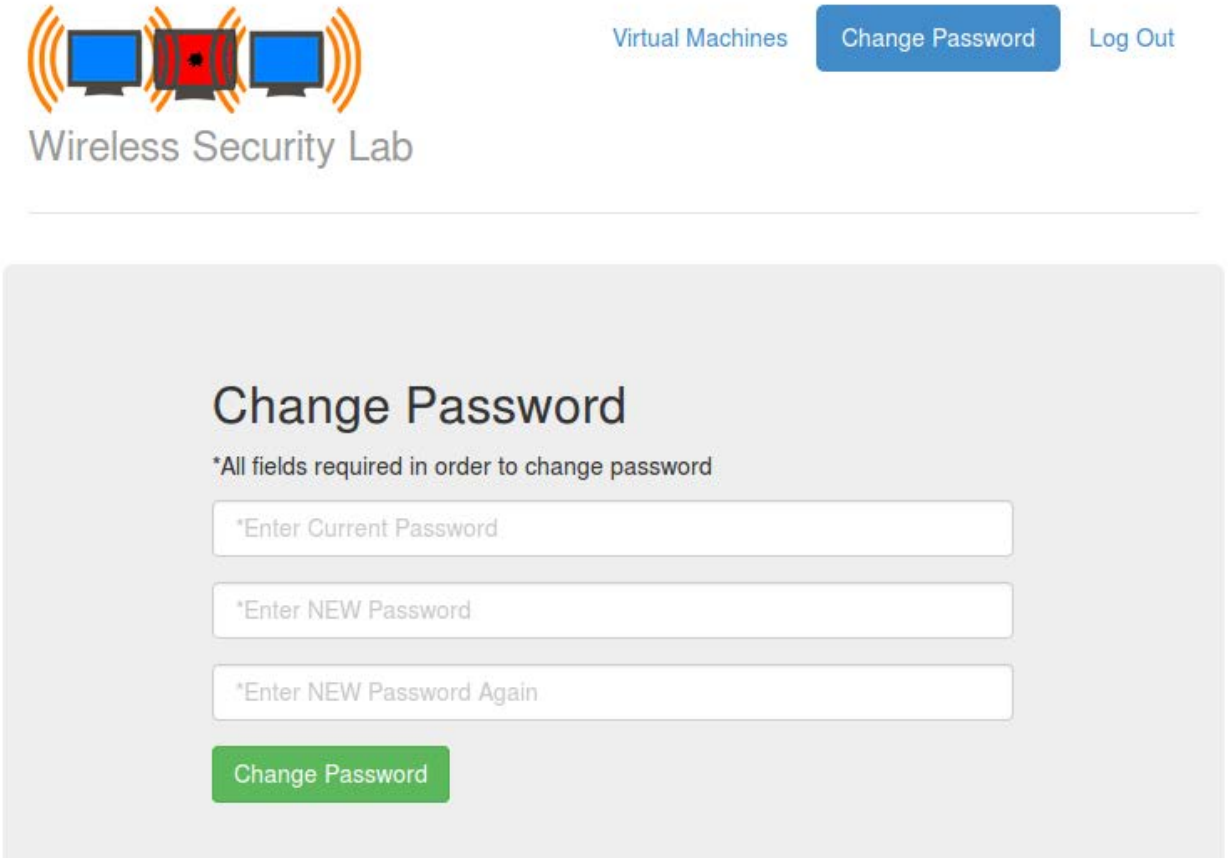
This section allows an administrator to create a new user with administrative privileges. The fields Name and Password are required.

[User] View/Manage My Virtual Machines



After a successful login, a user is redirected to this page. This page allows a user to view their virtual machine inventory. Users can deploy fresh images, delete their machines, or power on and off their machines. When a machine is powered on, the user will be able to see the machine's time until expiration and IP address.

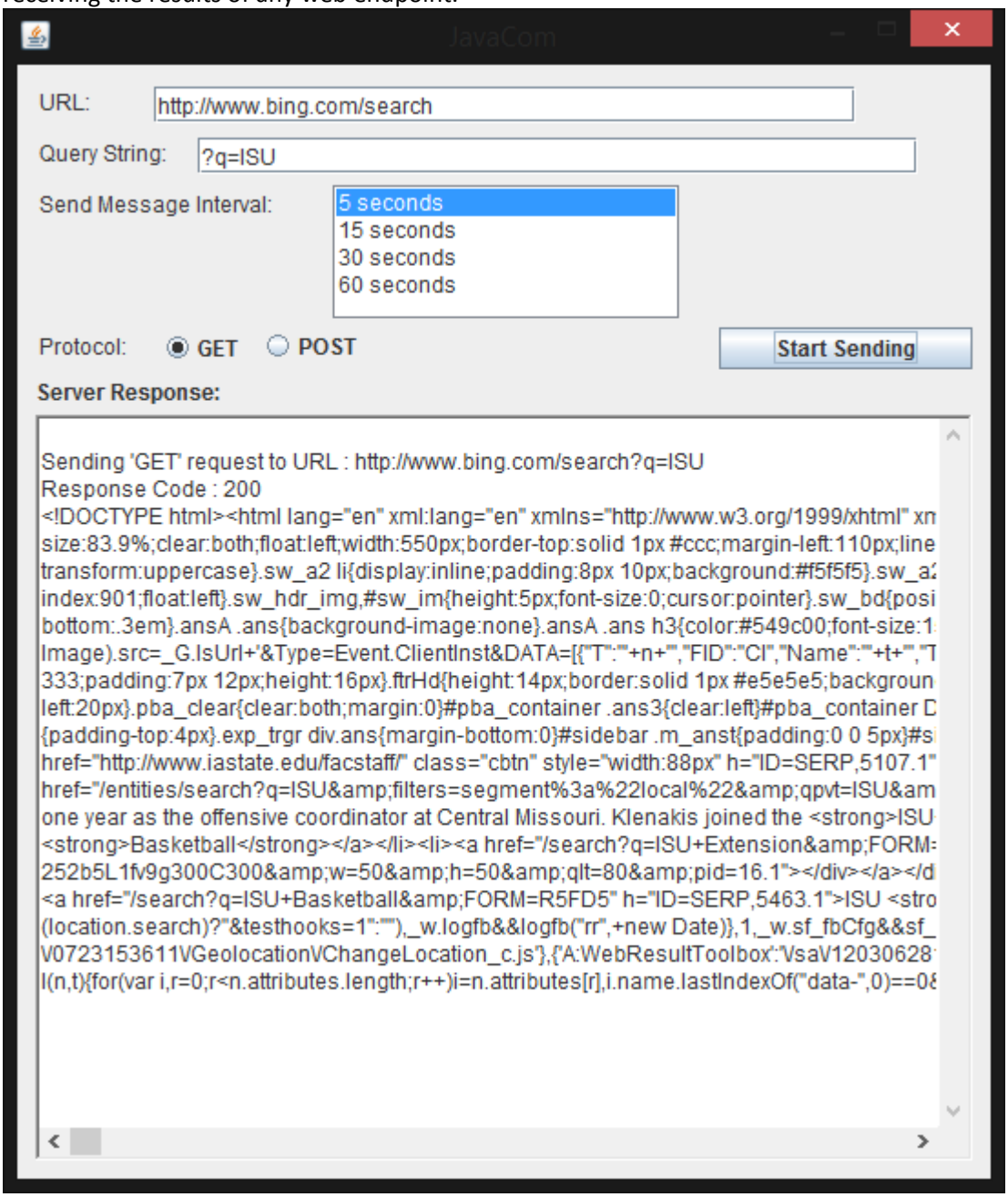
[User] Change My Password



This page can be accessed by clicking Change Password in the top right menu. From this page, a user can update their password.

Operating the Web Traffic Echoing Application

On the desktop of the Ubuntu Defender machine there is a file called JavaCom.jar. Run the application and you'll be presented with a window similar to the one below. JavaCom is an application that can send GET and POST requests to specify URL's on a specified interval. The application was originally intended to only send traffic to a page on the web server, but it's capable of sending traffic and receiving the results of any web endpoint.



In the Application you have several parameters to set: URL, query string, time interval and GET/POST. Each of these should be specified. When that information is specified, users can click 'Start Sending.' After that, the application will write out the results in the 'Server Response' text box.

Of note there is a copy of what was sent and to what URL, the server's HTTP response code, and the response text. All of this text can be copied out to notepad or the like in order to save and analyze it.

Our server endpoint is designed echo the traffic sent to it back to the application with the idea being that users can also use the Kali Linux attacker VM to intercept and modify the traffic. Currently the endpoint URL's are being setup and will be included in a later revision of this document.

LabVIEW with NI USRP-2920

We needed a computer with gigabit Ethernet port, LabVIEW 2011, NI-USRP 2920 and NI-USRP Driver 1.1. We made sure to install the LabVIEW on our machine before installing instrument drivers. Using an Ethernet cable, the USRP was connected to the computer and the device was powered up using the supplied power adapter. The computer used for this purpose, ran Windows 7 OS. We had to set up the network prior to communicating and programming the USRP using LabVIEW. From the Windows Network and Sharing Center, we clicked on the Ethernet connection and modified the Ethernet connection properties. The following steps must be followed in order to connect to the NI-USRP:

- Right click on the Ethernet connect
- Click on the properties button
- Under Networking Tab, select IPv4
- Right click on IPv4 and click on the properties
- Select the "Use the following IP address" option.
- Set the IP address to 192.168.10.2 and subnet mask to 255.255.255.0

Upon completion of the network setup, open the NI USRP Configuration Utility and the connected device was visible.

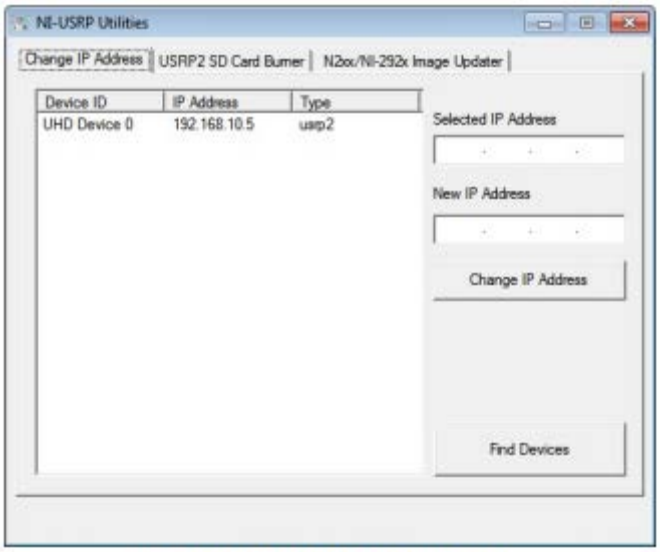


Figure 1. Confirm Network Connection

After the completion of the setup process, we opened up LabVIEW and started developing program in Graphical Programming Language that is provided by LabVIEW. Upon completing the program for capturing GSM signals, we ran it on LabVIEW development environment. The first part of the program opens up a GUI in which we can set up different sort of parameters, such as carrier frequency and Gain. These parameters utilize the functionality of USRP to capture any GSM signals. For testing, we ran program without any GSM interference around the USRP. As a result no waveforms or frequency spectrum could be

noticed in graphs of the GUI. Then we made calls using GSM phones, nearby the USRP. The USRP could pick up the GSM signals during the call and the resulting frequency spectrum and amplitude waveforms were plotted on the corresponding graphs.

Testing Result:

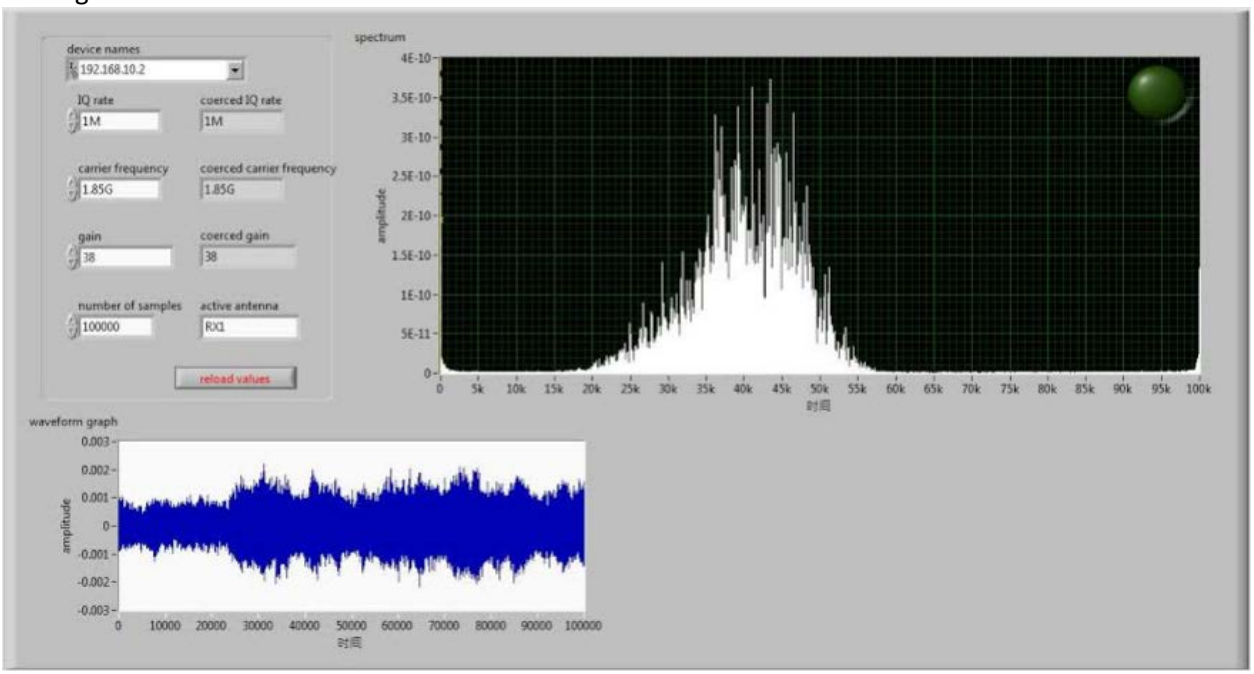


Figure: Captured GSM signals by NI USRP being displayed on LabVIEW GUI

OpenBTS – Network Design

By: Thuong Tran

Under the Supervision of
Professor George Amuracai

Abstract

This document is to be seen as a brief introduction to SDR, GSM Network, and the OpenBTS and as well as a guideline or a collection of notes for newbies to OpenBTS who struggle to get it working, or are lost in the wiki pages and wonder where to start.

Mostly, I detail here how I got it to work on my side, from step to step, with answers I found to a few issues I faced or my understanding of the problem. If you need more help, please refered to the OpenBTS wiki page and to subscribe to the mailing-list.

Table of Symbols and Abbreviations

Symbol	Abbreviation
SDR	Software defined Radio
RF	Radio Frequency
IF	Intermediate Frequency
TX	Transmitter
RX	Receiver
TRX	Transceiver
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
FPGA	Field Programmable Gate Array
DUC	Digital up Converter
DDC	Digital down Converter
USRP	Universal Software Radio Peripheral
UHD	USRP Hardware Driver
USB	Universal Serial Bus
GSM	Global System for Mobile Communication
PBX	Private Branch exchange
VOIP	Voice over Internet Protocol
GPRS	General Packet Radio Service
EDGE	Enhanced Data rates for GSM Evolution
MS	Mobile Station
SIM	Subscriber Identity Module
IMIE	International Mobile Equipment Identity
BTS	Base Transceiver Station
BSC	Base Station Controller
BSS	Base Station Subsystem
MSC	Mobile Switching Center
NSS	Network Station Subsystem
VLR	Visitor Location Register
HLR	Home Location Register
SIP	Session Initiation Protocol
TDM	Time Division Multiplexing
LAPDm	Link Access Procedures on the D channel
ETSI	The European Telecommunications Standards Institute
RR	Radio Resources
MM	Mobile Management
CC	Call Control
TCH	Traffic Channel
RACH	Random Access Channel

Chapter 1 - Introduction to OpenBTS

1.1 Software Defined Radio (SDR)

1.1.1 What is SDR?

Over the last decade as semiconductor technology has improved both in terms of performance, capability and cost, new radio technologies have emerged from military and research and development labs and become mainstream technologies. One of these technologies is software defined radio. Although much has been discussed in recent years, a good definition of software radio is difficult to generate. This is largely due to the flexibility that software defined radios offer, allowing them to take on many different forms that can be changed to suite the need at hand, but we can say that software defined radio is: **"Radio in which some or all of the physical layer functions are software defined"** In other words, Software Defined Radio (SDR) is a radio communication technology that is based on Software defined wireless communication protocols instead of hardwired implementations. Frequency band, air interface protocol and functionality can be upgraded with software download and update instead of a complete hardware replacement. Traditional hardware based radio devices limit cross-functionality and can only be modified through physical intervention. This physical intervention results in higher production costs and minimal flexibility in supporting multiple standards. By contrast, SDR technology provides an efficient and comparatively inexpensive solution to this problem, allowing multi-mode, multi-band and/or multi-functional wireless devices that can be enhanced using software upgrades.

1.1.2 Architecture

The software-defined radio (SDR) contains a number of basic functional blocks. The radio can be split into basic blocks, namely the front end, the IF section and the base band section as shown below. Each of the sections undertakes different types of functions.

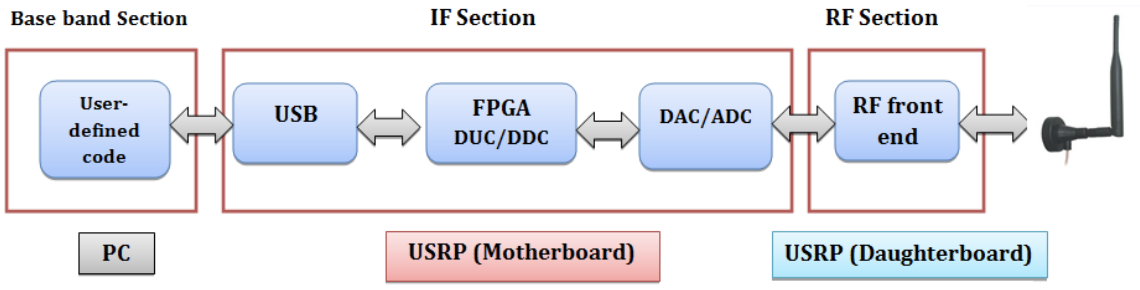


Figure 1 - USRP TX &RX Path

The front-end section uses analog RF circuit (Daugherboard) and it is responsible for receiving and transmitting the signal at the operational frequency. It also changes the signal to or from the intermediate frequency through up or down conversion.

The IF section performs the digital to and from analog conversions through (ADC/DAC). It also contains the processing (FPGA) that undertakes what may be thought of as the traditional radio processing elements, including filtering, modulation and demodulation and any other signal processing that may be required.

The Digital Up Converter (DUC), where the received signal from base band processor is modulated and conditioned as required. The Digital Down Converter (DDC), where the signal is processed and demodulated to provide the baseband signal for the baseband processor.

The final stage of the radio is the baseband processor (PC Processor). It is so obvious that the complexity level has been transferred to the PC part where the code is written to perform baseband processing.

1.1.3 Operation Concept

The ideal receiver scheme would be to attach an analog-to-digital converter to an antenna. A digital signal processor would read the converter, and then its software would transform the stream of data from the converter to any other form the application requires.

An ideal transmitter would be similar. A digital signal processor would generate a stream of numbers. These would be sent to a digital-to-analog converter connected to a radio antenna.

The ideal scheme is not completely realizable due to the actual limits of the technology. The main problem in both directions is the difficulty of conversion between the digital and the analog domains at a high enough rate and a high enough accuracy at the same time.

1.1.4 Advantages of SDR

SDR has expanded the idea of open-source and enabled amateur radio users and students to try and join the world of communications with very reasonable costs and without the need of complicated hardware, all what is needed is a Computer, a single transceiver and a software code that can be easily implemented or can be obtained from the internet, All this software enabled the prototyping to be faster and cheaper than hardware prototyping.

SDR has the ability to receive and transmit various modulation methods using the same set of hardware. The ability to alter functionality by downloading and running new software as well as the possibility of adaptively choosing an operating frequency and a mode best suited for prevailing conditions.

In other word SDR solves the two main challenges for a wireless system, which are compatibility and spectrum usage.

From the Vendors point of view, SDR enables the implementation of a family of radio products using a common platform architecture allowing the prototyping and so faster introduction of new products and the development costs will be dramatically low. Also the use of SDR would allow bug fixing over the air or other remote reprogramming thus reducing both time and cost associated with

operation and maintenance.

While for Operators, New features and capabilities could be added without requiring major modifications to the hardware as the old hardware could be used with simple modifications to the software to upgrade the whole system to work with the new features and services significantly reducing logistical support and operating expenditures.

A Software Defined Radio can easily be many different kinds of radio, often several different types at once. SDR has the potential to be a revolutionary technology that will dramatically impact the wireless technology industry.

1.1.5 SDR Application

Through the last two decades of open source developing, the SDR has about several hundreds of applications such as Cognitive Radio, RF-ID and OpenBTS which is our project subject and we will talk about it in details later.

1.2 USRP

1.2.1 What is USRP?

The Universal Software Radio Peripheral (USRP) is a computer-hosted software radios, developed by Matt Ettus and his team at the Ettus Research LLC.

The USRP product family is intended to be a comparatively inexpensive hardware platform for software radio, and is commonly used by research labs, universities, and hobbyists. The USRP family was designed for accessibility, all USRP products are controlled with the open source UHD driver.

The USRP is designed to allow general-purpose computers to function as high bandwidth software radios. In essence, it serves as a digital baseband and IF section of a radio communication system.

In addition, it has a well-defined electrical and mechanical interface to RF front-ends (daughterboards), which can translate between that IF or baseband and the RF bands of interest.

The USRP does all of the waveform specific processing on the host CPU like

- Modulation and Demodulation

All of the high-speed general-purpose operations are done on the FPGA like

- Digital Up Conversion (DUC).
- Digital Down Conversion (DDC).
- Decimation.

NI-USRP 2920

1.2.2 UHD

UHD is the "Universal Software Radio Peripheral" (USRP) Hardware Driver is the device driver provided by Ettus Research for use with the USRP product family. It works on all major platforms Linux, Windows, and Mac

The goal of UHD is to provide a host driver and API for current and future Ettus Research products. Users will be able to use the UHD driver standalone or with third-party applications such as:

- GNU Radio.
- LabVIEW.
- MATLAB.
- OpenBTS.

1.2.3 USRP Component

The USRP is made up of the motherboard, which has USB 2.0 interface for connection to the computer and the power connector and contains a FPGA section for high speed signal processing, and interchangeable daughterboard that cover different frequency ranges. In addition to ADC, DAC and one or more antennas.

Motherboard The motherboard provides the following subsystems:

- FPGA,
- ADCs, DACs,
- Host processor interface,
- Power regulation.
- Clock generation and synchronization

These are the basic components that are required for baseband processing of signals.

Daughterboard

Daughterboard turn USRP motherboard into a complete RF transceiver system. Just add an antenna, and you are ready for two-way, high bandwidth communications in many popular frequency bands, it is used for analog operations such as up/down-conversion, filtering, and other signal

conditioning. This modularity permits the USRP to serve applications that operate between DC and 6 GHz.

Xilinx Spartan 3A-1400 FPGA

FPGA plays a key role in the USRP system. Basically what it does is to perform high bandwidth math, and to reduce the data rates to something you can handle with USB2.0.
The FPGA connects to a USB2 interface chip, the Cypress FX2.

Cypress FX2

The Cypress FX2 interfaces between the FPGA and a USB 2.0 port. The USRP connects to a USB port on the host computer where modulation and demodulation is performed.

1.3 OpenBTS

1.3.1 Introduction and History

What is OpenBTS?

OpenBTS is a software-based GSM access point, allowing standard GSM-compatible mobile phones to make telephone calls without using existing telecommunication providers' networks. OpenBTS is notable for being the first free software implementation of the industry-standard GSM protocol stack.

OpenBTS is an open-source UNIX application that uses the Universal Software Radio Peripheral (USRP) to present a GSM air interface ("Um") to standard GSM handset and uses the Asterisk® software PBX to connect calls. The combination of the ubiquitous GSM air interface with VoIP backhaul could form the basis of a new type of cellular network that could be deployed and operated at substantially lower cost than existing technologies in Greenfields in the developing world. In other word OpenBTS = GSM + VOIP.

1.3.2 Traditional GSM networks

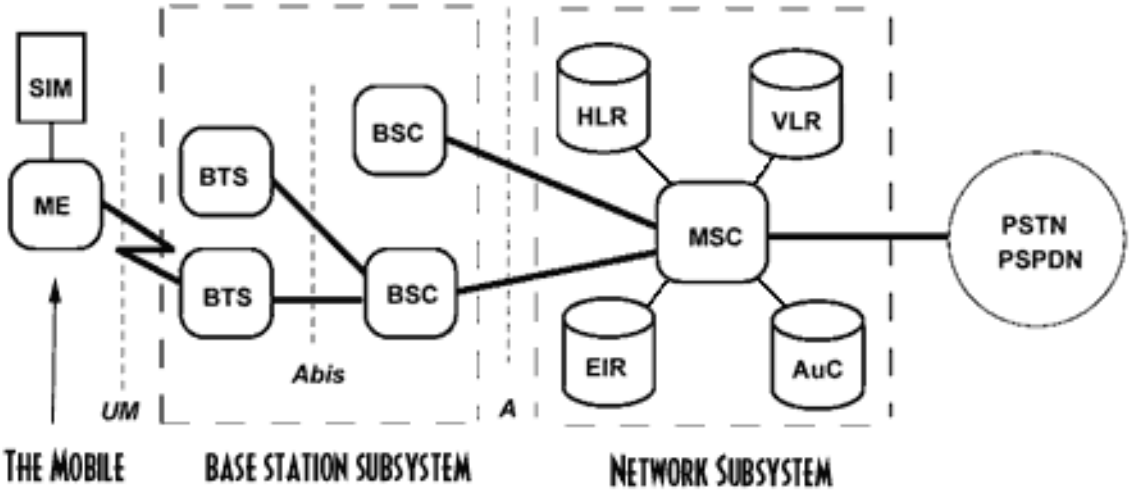
GSM, or Global System for Mobile Communications, is an European standard for the Mobile telecommunications and it is considered as one of the most popular standard worldwide, it is known as the second generation mobile telecommunications system “2G system”.

GSM operate in the 900MHz band, Uplink band from 890 to 915 MHz and downlink band from 935 to 960 MHz, the two bands are generally separated by 45MHz. GSM uses GMSK “Gaussian Minimum Shift Key” with a 270.833 kHz symbol rate. The channel is time-domain multiplexed into 8 timeslots, each with duration of 156.25 symbol periods.

The principle component groups of a GSM network are as follows:

- The Mobile Station (MS).
- The Base Station System (BSS).
- The Network Switching System (NSS).

The diagram below shows the GSM network Architecture:



Mobile Station (MS)

The MS consists of two parts, the Mobile Equipment (ME) and an electronic smart card called a Subscriber Identity module (SIM). The ME is the hardware used by the subscriber to access the network. The hardware has an identity number associated with it, which is unique for that particular device and permanently stored in it. This identity number is called the International Mobile Equipment Identity (IMEI) and enables the network operator to identify mobile equipment, which may be causing problems on the system.

The SIM is a card that plugs into the ME. This card identifies the MS subscriber and also provides other information regarding the service that subscriber should receive.
Five identity numbers identify the subscriber as follows:

Base Station System (BSS)

The GSM Base Station System is the equipment located at a cell site. It comprises a combination of digital and RF equipment. The BSS provides the link between the MS and the MSC. The BSS communicates with the MS over the digital air interface and with the MSC via 2 Mbit/s links.

The BSS consists mainly of:

1. The Base Transceiver Station – BTS

   The BTS contains the RF components that provide the air interface for a particular cell. This is the part of the GSM network, which communicates with the MS. The antenna is included as part of the BTS.

2. The Base Station Controller – BSC

   The BSC as its name implies provides the control for the BSS. The BSC communicates directly with the MSC. The BSC may control single or multiple BTSs.

The Network Switching System includes the main switching functions of the GSM network. It also contains the databases required for subscriber data and mobility management. Its main function is to manage communications between the GSM network and other telecommunications networks.

The components of the Network Switching System are listed below:

- Mobile Services Switching Centre – MSC.
- Home Location Register – HLR.
- Visitor Location Register – VLR.
- Equipment Identity Register – EIR.
- Authentication Centre – AUC.
- Interworking Function – IWF.
- Echo Canceller – EC.

We will focus on MSC, HLR and VLR.

Mobile Services Switching Centre (MSC)

The MSC is included in the GSM system for call-switching. Its overall purpose is the same as that of any telephone exchange.

However, because of the additional complications involved in the control and security aspects of the GSM cellular system and the wide range of subscriber facilities that it offers, the MSC has to be capable of fulfilling many additional functions.

The MSC will carry out several different functions depending upon its position in the network. When the MSC provides the interface between the PSTN and the BSSs in the GSM network it will be known as a Gateway MSC. In this position it will provide the switching required for all MS originated or terminated traffic.

Each MSC provides service to MSs located within a defined geographic coverage area, the network typically contains more than one MSC. One MSC is capable of supporting a regional capital with approximately one million inhabitants. An MSC of this size will be contained in about half a dozen racks.

The functions carried out by the MSC are listed below:

-- Call Processing
-- Operations and Maintenance Support
-- Internetwork Interworking
-- Billing

Home Location Register (HLR)

The HLR is the reference database for subscriber parameters. Various identification numbers and addresses are stored, as well as authentication parameters. The network provider enters this information into the database when a new subscriber is added to the system. The parameters stored in the HLR are listed opposite: The HLR database contains the master database of all the subscribers to a GSM PLMN.

The data it contains is remotely accessed by all the MSCs and the VLRs in the network and, although the network may contain more than one HLR, there is only one database record per

subscriber – each HLR is therefore handling a portion of the total subscriber database. Either the IMSI or the MSISDN number may access the subscriber data. The data can also be accessed by an MSC or a VLR in a different PLMN, to allow inter-system and inter-country roaming.

Visitor Location Register (VLR)

The VLR contains a copy of most of the data stored at the HLR. It is, however, temporary data that exists for only as long as the subscriber is "active" in the particular area covered by the VLR. The VLR database will therefore contain some duplicate data as well as more precise data relevant to the subscriber remaining within the VLR coverage.
The VLR provides a local database for the subscribers wherever they are physically located within a PLMN; this may or may not be the "home" system. This function eliminates the need for excessive and time-consuming references to the "home" HLR database.

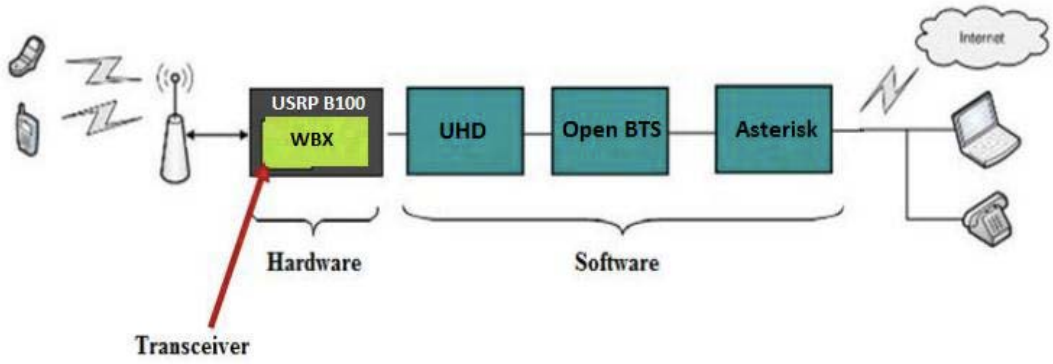
1.3.3 OpenBTS and Traditional GSM

In this section we know how OpenBTS replaced the GSM Network Component that we have mentioned previously

1. A USRP (Universal Software Radio Peripheral) as hardware. USRP can be readily adapted as a GSM transceiver (BTS) (i.e.: it transmits and receives the GSM signal to and from the mobile phone).

2. OpenBTS software code which generates with UHD an air interface that to a cell phone, looks just like any other GSM cellular network. On the network side, it's an Asterisk server (VoIP), used to connect calls. OpenBTS software code plays the role of MSC/VLR in processing all the calls incoming to, or originating from subscribers visiting the given switch area.

Using openBTS source code only creates a beacon signal such that openBTS network is created and a phone can register to this network But, cannot make a phone call with another registered phone except when asterisk is installed and configured in this system as Asterisk plays the role of HLR in the traditional GSM network which is the main database of permanent subscriber information for a mobile network (i.e.: it stores an IMSI for each subscriber, authentication key, subscriber status and the current location).



1.3.4 OpenBTS Advantages

The main advantage of the OpenBTS is the minimum cost as we can install the network at about 1/10 of the cost of current technologies, and still be compatible with most of the handsets that are already in the market. By replacing the GSM core network with commodity Hardware and open source Software. Also, OpenBTS allow bug fixing over the air or other remote reprogramming thus reducing both time and cost associated with operation and maintenance.

OpenBTS solves one of the toughest challenges for the Mobile Communication systems, which is the compatibility, as now it's about upgrading the software which is not comparable with Hardware replacement cost.

1.3.5 Hardware requirement for OpenBTS

For a small OpenBTS network with, the minimal hardware requirements are:

1) Unix Computer (Ubuntu): Basically, any computer should do the work. The only thing which is really required is a USB port to plug the USRP board, but all computers usually have that.

2) USRP (N210): This board (see Figure 3) can be purchased from NI, or similar product can be purchased from Ettus Research (https://www.ettus.com/product/details/USRP-PKG) for 700 USD. It includes an Altera Cyclone FPGA. OpenBTS also works with other boards such as USRP2, B100, N200, E100... See the wiki [Ran].

3) WBX daughterboard (as a Transceiver): Select the daughterboard you need according to the GSM band you want to use. RFX 900 for GSM 850/900, RFX 1800 for GSM 1800/1900. Price from Ettus: 275 USD (https://www.ettus.com/product/details/RFX1800)

4) Two antennas covering GSM range (one for TX and one for RX): 1 antenna per daughterboard. Be sure to select an antenna that matches your daughterboard. Can be purchased from Ettus for 35 USD.

5) Mobile phones. Obviously you need one at least. It must be unlocked. And you need to be able to manually select a network for that phone (see Figure 12).

6) SIM cards: One SIM card per mobile phone. It is possible to use a standard SIM card - the one you use in your own mobile phone[1], or you can buy a programmable SIM card (see Figure 1). Search for something like Super SIM, SIM MAX, Magic SIM, 12in1 or 16in1 SIM on the web. For each SIM card, you need to know its IMSI (section 7.1 explains how to get it). On eBay, such SIM cards are sold for approximately 1 USD.

7) Magic SIM card reader/writer: If you use Magic SIM cards, you need to card reader and writer to program the SIM card (see Figure 2). This usually costs only a few bucks.

1.3.6 Software requirement for OpenBTS

The major components you need for OpenBTS are:

- A Linux operating system. It might be portable to other Unix systems.
- GnuRadio

- A SIP PBX such as Asterisk, FreeSwitch or Yate
- OpenBTS

1.3.7 OpenBTS P2.8 Release

OpenBTS P2.8 is the latest version of public OpenBTS soft wre, it includes 4 main modules which is:

- Transceiver.
- GSM Stack.
- Control.
- SIP Switch.

1.3.8 OpenBTS Modules

Transceiver Module

The Transceiver is responsible for transmitting and receiving samples to and from the USRP, also it passes these samples in the form of raw bits to the GSM stack in case of reception or receives them from the GSM stack in case of transmission.
It interfaces with the GSM stack through UDP socket, and with the USRP through USB 2.0.
It performs the basic operations such as modulation, interleaving, correlation, etc.

GSM Module

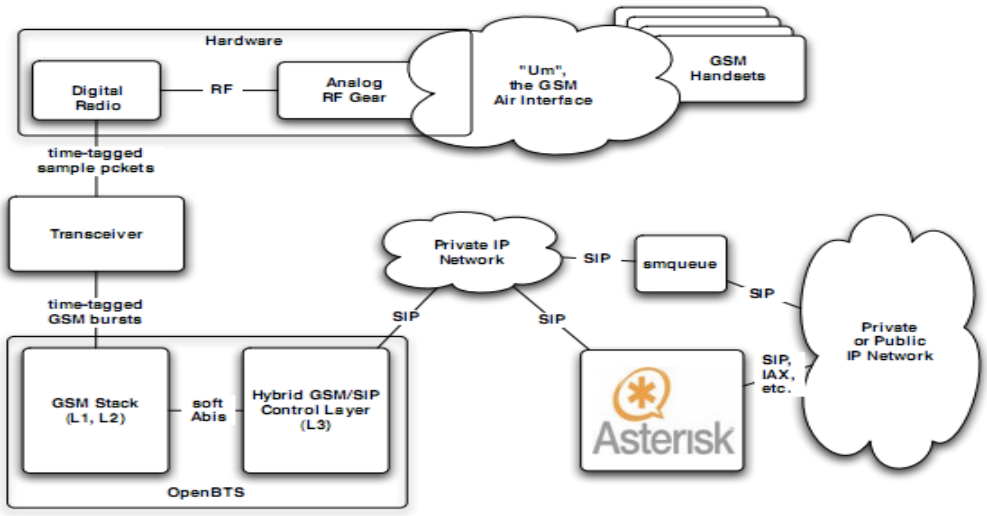
The GSM module implements the GSM stack above the radio modem, it implements the three layers found In the ETSI standards.
The interface between the control and the GSM layers is the L3 messages sent between them.

Control Module

Perform the signaling and connection management
L3 radio resource management functions
L3 GSM-SIP gateway for mobility management
L3 GSM-SIP gateway for call control

SIP Module

OpenBTS uses a SIP switch or PBX to perform the call control functions that would normally be performed by the mobile switching center in a conventional GSM network, although in most network configurations. This switching function is distributed over multiple switches. These switches also provide transcoding services.
In OpenBTS P2.8 the standard SIP switch is Asterisk 1.8.



1.3.9 OpenBTS Setup

Basically, the steps are:

1. **OpenBTS prerequisites**
   To properly compile and install OpenBTS install the following:

   autoconf
   libtool
   libosip2-dev
   libortp-dev
   libusb-1.0-0-dev
   g++
   sqlite3
   libsqlite3-dev (sipauthserve only)
   libboost-all-dev
   libreadline6-dev
   erlang (very important for asterisk OCBD realtime driver)

   These can be installed with the following command:

   sudo apt-get install autoconf libtool libosip2-dev libortp-dev libusb-1.0-0-dev g++ sqlite3 libsqlite3-dev libboost-all-dev libreadline6-dev erlang

   Install subversion as well with the same method. And also git is really useful for pulling repositories.

2. **Create Directory**
   Make a directory called OpenBts where you will place everything related to the OpenBTS system by running the following command, not as sudo in order to give access to all users

mkdir OpenBts
cd OpenBts

3. **Download OpenBTS**
   The best way to get OpenBTS is by pulling the code directly from the source code repository as an anonymous read-only user by running the following command (for version 2.8):
   cd OpenBts
   svn co http://wush.net/svn/range/software/public

4. **Download and install requirements**
   sudo apt-get install autoconf libtool libosip2-dev libortp-dev libusb-1.0-0-dev g++ sqlite3 libsqlite3-dev erlang libreadline6-dev libboost-all-dev

   if svn is not installed, first run
   sudo apt-get install subversion

5. **Building OpenBTS**
   To build OpenBTS fist cd into the appropriate folder:

   cd /OpenBts/public/openbts/trunk

   Then depending on what hardware radio you are using you can configure accordingly. **USRP2 and N200 series require host-based re-sampling support and use UHD drivers, so you would run the following commands:**

   autoreconf -i
   ./configure --with-uhd --with-resamp
   make

   Additional built time options for UHD devices include 10 MHz external reference support (we are not using this). For example, to use the front panel reference input on a N210, configure with the following options:

   autoreconf -i
   ./configure --with-uhd --with-resamp --with-extref
   make

   For USRP1 Single daughterboard configuration we use GNURadio driver and would run the following command:

   autoreconf -i
   ./configure --with-usrp1 --with-singledb
   make

   With the build resolved, you'll need to build and link the transceiver appropriate for your

hardware. For the USRP/UHD installs:

(from OpenBTS root)
cd Transceiver52M
make
cd ../apps
ln -s ../Transceiver52M/transceiver .

6. Configuring OpenBTS

With OpenBTS built, you now need to configure it to run correctly. There are key files that must be created for this to happen. For example /etc/OpenBTS/OpenBTS.db. OpenBTS.db is the database store for all OpenBTS configuration. It must be installed at /etc/OpenBTS, which likely does not exist. So, to create this file:

(from the OpenBTS directory)
sudo mkdir /etc/OpenBTS
sudo sqlite3 -init ./apps/OpenBTS.example.sql /etc/OpenBTS/OpenBTS.db ".quit"

This generates a lot of stock configuration options. You can find these listed here: https://wush.net/trac/rangepublic/wiki/openBTSConfig. Most of these only need to be tweaked if you are moving beyond a simple desktop setup. However, a few are required for basic operation. These are:

GSM.Radio.Band - Set this to the GSM band appropriate for your hardware.
GSM.Radio.C0 - This is the ARFCN. Set it to something appropriate for your band as described below.

Control.LUR.OpenRegistration - Set this to a regular expression of numbers matching the IMSIs of your test phones. This tells OpenBTS to not reject your handset just because your registration server (below) isn't responding. Useful for debugging and initializing the system.

To edit the OpenBTS.db file I recommend downloading and installing SQLite Database Browser which allows to visualize the entire database and its fields.

sudo apt-get install sqlitebrowser
Or another great program:
sudo apt-get install sqliteman

Then you can run SQLite Database Browser or SQLite Manager by calling it in terminal:

sudo sqlitebrowser
sudo sqliteman

Let's take for example setting up the system in the Netherlands, where there are specific channels and frequencies allowed and open for research purposes.

NOTE: The Dect Guardband is a frequency band between 1877 MHz and 1880 MHz (in duplex band between1782 MHz and 1785 MHz). The maximum capacity without a license is 200 mW ERP (Effective Radiated Power). Therefore the Radio can be programmed on the Channel 871, but any of these channels would work.
You can find the complete list of Channels here:
http://gnuradio.org/redmine/attachments/115/all_gsm_channels_arfcn.txt#L829

Or if you notice too much interference, you can try and use a different band that is not being used in the US.

These values can be modified, in the most recent version of OpenBTS, from OpenBTSCLI with the config command:

config GSM.Radio.Band 1800
config GSM.Radio.C0 880
*config Control.LUR.OpenRegistration **

7. **Subscriber Registry and Sipauthserv**
   OpenBTS depends on the installation of Sipauthserver the SIP authorization server. You'll need to build and install it before running OpenBTS.
   a. **Subscriber Registry**
      To setup the Subscriber Registry database you must first create the file path the db will reside in. By default, this is /var/lib/asterisk/sqlite3dir.

      (from svn root)
      cd subscriberRegistry/trunk/configFiles/
      sudo mkdir -p /var/lib/asterisk/sqlite3dir
      sudo sqlite3 -init subscriberRegistryInit.sql /var/lib/asterisk/sqlite3dir/sqlite3.db ".quit"
   b. **Sipauthserv**
      Sipauthserve is an aptly-named daemon providing SIP authentication services. The SIP.Proxy.Registration config variable in openbts should point to its hostname and port. To build Sipauthserve, you MUST HAVE ALREADY BUILT OPENBTS. This is a makefile hack, and will hopefully be fixed at some point in the future. To build Sipauthserve:

      (from svn root)
      cd subscriberRegistry/trunk
      make

      This will produce a sipauthserve executable.
      As with OpenBTS, you'll need to configure sipauthserve. We assume /etc/OpenBTS/ already exists.

      (from subscriberRegistry root)
      sudo sqlite3 -init sipauthserve.example.sql /etc/OpenBTS/sipauthserve.db ".quit"

   c. **Running sipauthserve**

Running sipauthserve will provide you with a registration server. To do so:

(from subscriberRegistry root)
sudo ./sipauthserve

sipauthserve does not have a CLI, so you'll only see a small output:

ALERT 139639310980928 sipauthserve.cpp:214:main: ./sipauthserve (re)starting

Remember, if you change any of the config variables, you'll need to restart sipauthserve for the changes to take effect.

8. Smqueue

Smqueue is the store-and-forward message service packaged with OpenBTS. Building and running is very similar to the process used for OpenBTS.

a. Build and Install

In the smqueue/trunk directory, run the following commands:

autoreconf -i
./configure
make

You should now have an smqueue executable in the smqueue/trunk/smqueue directory.

b. Configuring Smqueue

Similar to OpenBTS, Smqueue also depends on a configuration file, located at /etc/OpenBTS/smqueue.db. Smqueue creates an empty, nonfunctional version of this db if it is not available. That's of no use to anyone. Instead, do as we did with OpenBTS and run the following command:

(from the smqueue directory)
sudo sqlite3 -init smqueue/smqueue.example.sql /etc/OpenBTS/smqueue.db ".quit"

That will initialize /etc/OpenBTS/smqueue.db with default values. These configuration variables should work without modification, and are listed here:
https://wush.net/trac/rangepublic/wiki/smqueueConfig

c. Running Smqueue

Smqueue is run with the following command:

(from the smqueue directory)
cd smqueue sudo ./smqueue

Smqueue does not have a command-line interface, instead just reading configuration values and processing messages. So you'll only see a small output:

ALERT 140545832068928 smqueue.cpp:2421:main: smqueue (re)starting smqueue logs to syslogd facility LOCAL7, so there's not much to see here

Remember, if you change any of the variables, you'll need to restart smqueue for the changes to take effect.

1.4 GnuRadio

GNU Radio is the software used to communicate with the USRP device. If using a USRP1 model, then a version higher than 3.3 and lower than 3.5 is required.
http://gnuradio.org/redmine/projects/gnuradio/wiki/

If you are planning on using the USRP2 or N series devices, then you can use the latest version of GNU Radio, which can be installed using the script below. In fact, for the USRP2 or N series all you need GNU Radio for are the UHD drivers.

Copy and Paste or download the script from here:
http://www.sbrac.org/files/build-gnuradio

Place the build-gnuradio file inside your OpenBts folder where you will build GNU Radio. This script not only builds GNU Radio but also all the prerequisite libraries, thanks a lot to the developer Markus for making it so easy. Cd into the OpenBts folder and run the following commands not as root:

NOTE: *This process usually takes 3 or more hours! Let it run its course and monitor for errors. This script downloads and builds the latest version of GNU Radio.*

chmod a+x build-gnuradio
./build-gnuradio

1.5 Asterisk

1. **Installation:**
   Asterisk is the PBX software I prefer to use, but there are alternatives such as FreeSwitch. If you plan on seriously delving into Asterisk, then I suggest installing and building Asterisk from source for a complete 1.8 version. In that case I recommend following the instructions described at http://www.asterisk.org/. FreeSwitch is a great alternative, but not as documented. To install Asterisk from package manager in Ubuntu, run the following command:

   sudo apt-get install asterisk

   During configuration and install process you may be asked to supply your country code to set telephony preferences for your area. See here for the right codes:
   http://en.wikipedia.org/wiki/International_mobile_phone_codes

Example:
Australia 61
USA+Canada 1

In order to use Asterisk as PBX for OpenBTS, we will have to modify a few files and configure a couple of modules, I will explain those steps further below.

2. **Configuring the PBX (Asterisk 1.8)**
   Asterisk is the "standard" OpenBTS PBX. It's the easiest to set up, most documented, and generally simplest option. The steps for installing and configuring Asterisk to work with OpenBTS are located here: https://wush.net/trac/rangepublic/wiki/asteriskConfig

3. **Asterisk RealTime with ODBC Connector**
   The previous command should have downloaded the latest version (1.8) or you can build from source by following the instructions on the Asterisk wiki.
   https://wiki.asterisk.org/wiki/display/AST/Home

   For Asterisk to work with OpenBTS, Sipauthserve, Smqueue and all, we need to install and configure MySQL and ODBC. For a detailed series of steps (it would take a guide by itself to do!) please refer to the steps highlighted in **Chapter 16 and 18 of Asterisk, The Definitive Guide published by O'Reilly**: http://ofps.oreilly.com/titles/9780596517342/

   And then follow the steps mentioned here:
   http://wush.net/trac/rangepublic/wiki/sqlie3ODBC

1.5.1 Running It All

If you are not going to require WiFi or other networking capabilities for your machine while running OpenBTS you can disable networking, otherwise keep networking enabled. Plug the Radio power supply into an outlet and after the power lights come on, plug it into the computer through the Ethernet Cable. Now follow these easy steps to run the entire system:

- **Turn off firewall.** Ubuntu has a new firewall feature that conflicts, so we need to turn off by saying:
  sudo ufw disable

- **Give the port a static IP.** The Ubuntu network needs to be manually configured to talk to the Radio, so we use the ifconfig command to get it to bind to a specific address (you need to do it each time you start the system).
  sudo ifconfig eth0 192.168.10.1 netmask 255.255.255.0 promisc

  If it worked, and you run the ifconfig command again, you should see something like:
  eth0
  Link encap:Ethernet HWaddr 00:22:15:d3:7e:1f
  inet addr:192.168.10.1 Bcast:192.168.10.255 Mask:255.255.255.0
  inet6 addr: fe80::222:15ff:fed3:7e1f/64 Scope:Link
  UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1

RX packets:8752 errors:0 dropped:0 overruns:0 frame:0
TX packets:8993 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:2712616 (2.7 MB) TX bytes:2769249 (2.7 MB)
Interrupt:43 Base address:0x8000

- **Make sure the computer recognizes the UHD Device.** To test if this works you can find your USRP by issuing the command:
  uhd_find_devices
  If everything is plugged in together and installed correctly, in the terminal you will read:
  ~$ uhd_find_devices
  linux; GNU C++ version 4.6.1; Boost_104601; UHD_003.005.000-b1f34b4

  UHD Device 0

  Device Address:
  type: usrp
  addr: 192.168.10.2
  name:

- **Check the lights on the USRP.** The LED indicator should show you D and F lit as well as the left light on the ethernet port lit, the port green LED, if the Ethernet cable is plugged in.
  Run each individual executable in a separate terminal window as sudo. These are:
- **Asterisk:**
  (this is to run in verbose mode, so we get to read everything it does)
  sudo #asterisk -vvvvvr
  (this is what it will respond with:)
  CLI>

- **Smqueue:**
  (this is the directory to look for it in)
  ~OpenBts/public/smqueue/trunk/smqueue/
  sudo ./smqueue
  (if everything runs correctly this is what it should respond:)
  ALERT 3077551824 smqueue.cpp:2347:main: smqueue (re)starting
  smqueue logs to syslogd facility LOCAL7, so there's not much to see here

- **Sipauthserve:**
  (this is the directory to look for it into)
  ~OpenBts/public/subscriberRegistry/trunk/
  sudo ./sipauthserve
  (if everything runs correctly this is what it should respond:)
  ALERT 3078424272 sipauthserve.cpp:213:main: ./sipauthserve (re)starting

- OpenBTS:
  (this is the directory to look for it into)

~OpenBts/public/openbts/trunk/apps/
sudo ./OpenBTS
(if everything runs correctly this is what it should respond:)
ALERT 3079374544 OpenBTS.cpp:148:main: OpenBTS starting, ver P2.8TRUNK build date Feb 20 2012
1330365223.527164 3079374544:
OpenBTS
Copyright 2008, 2009, 2010, 2011 Free Software Foundation, Inc.
Copyright 2010 Kestrel Signal Processing, Inc.
Copyright 2011 Range Networks, Inc.
Release P2.8TRUNK formal build date Feb 20 2012
"OpenBTS" is a trademark of Range Networks, Inc.
Contributors:
Range Networks, Inc.:
David Burgess, Harvind Samra, Donald Kirker, Doug Brown
Kestrel Signal Processing, Inc.:
David Burgess, Harvind Samra, Raffi Sevlian, Roshan Baliga
GNU Radio:
Johnathan Corgan
Others:
Anne Kwong, Jacob Appelbaum, Joshua Lackey, Alon Levy
Alexander Chemeris, Alberto Escudero-Pascual
Incorporated GPL libraries and components:
libosip2 (LGPL), liportp2 (LGPL)
This program comes with ABSOLUTELY NO WARRANTY.
Use of this software may be subject to other legal restrictions,
including patent licsensing and radio spectrum licensing.
All users of this software are expected to comply with applicable
regulations and laws. See the LEGAL file in the source code for
more information.
1330365223.566186 3079374544:
Starting the system...
linux; GNU C++ version 4.6.1; Boost_104601; UHD_003.004.000-b1f34b4
1330365228.732672 3079374544:
Welcome to OpenBTS. Type "help" to see available commands.
OpenBTS>
If you now type help and hit return it will show a series of options and commands.

Congratulations! You now have the whole software package running properly and ready for phones to camp (aka register onto the network)!

- **Check the lights again.** You should now see that A C E D F LEDs on the N200 are lit and both the green and orange LEDs on the ethernet port are ON.

- **Let's camp some phones!** Turn on a phone with a GSM SIM card installed. It would be best if this SIM was NOT from a local carrier; then the phone will not immediately camp to one of

their towers in the area.

In most cases, on most phones, there is a way to select the specific network you wish to attach to by navigating the phone menu to scan the available networks. For this basic install, the network will most likely be announced as: 001 01 or a variation. Connect the phone to that network. Your BTS should reply with a Welcome message, allowing the phone to associate and send back a desired 7 digit unique number. If this fails, make sure you set the Control.LUR.OpenRegistration variable in OpenBTS.db.

- With these two tests passed, you can now test the connection to the network. Register two phones and see if you can call each other's numbers or send an SMS to each other. If that's working, congrats again! You now have a working system!
- **tmsis.** In the OpenBTS terminal screen you can type the command tmsis and this should show you the IMSI numbers of the SIM cards that are now associated and are camping on the network.

1.5.2 Notes

- Just as a recap, the local addresses and ports for all the applications to run on are these:

  Asterisk: 127.0.0.1:5060
  Smqueue: 127.0.0.1:5063
  OpenBTS: 127.0.0.1:5062
  SubscriberRegistry/Sipauthserve: 127.0.0.1:5064

- The extension numbers for the registered phones are saved into the sqlite3.db under the sip_buddies table. You can edit the entries in this table in order to erase some of the extensions or IMSI numbers that have been registered. The path for the SIP_BUDDIES table is: */var/lib/asterisk/sqlite3dir/*

- If in running OpenBTS you encounter this message:
  ortp-warning-Must catchup 51 miliseconds.
  ortp-warning-Must catchup 56 miliseconds.
  ortp-warning-Must catchup 57 miliseconds.
  ortp-warning-Must catchup 45 miliseconds.

  It's OK, it happens quite often. It just means that because of the computer hardware or the antennas, the CPU is running slightly behind but it's catching up. It might happen if you are trying to do other things while running the entire system (like checking email, or opening an extra terminal window). The system should keep running properly regardless and if not you can try running OpenBTS with real-time priority (using the `chrt` command).

- Asterisk can be very temperamental, so it's best to handle calls with extra care and allow for enough time to pass between initiating calls back and forth. The system, because of having just 1 daughter board, can handle innumerable amounts of SMS text messages, but it can only handle 2 voice calls (among 4 phones) at once. It was tested and it actually behaved quite well.

If you experience dropped calls it might be because of the actual GSM frequency getting spammed with other devices that might be on the same network.

If you get something similar to this Warning:
WARNING[997]: chan_sip.c:3551 retrans_pkt: Retransmission timeout reached on transmission 1453771899@127.0.0.1 for seqno 325 (Critical Response) -- See https://wiki.asterisk.org/wiki/display/AST/SIP+Retransmissions
Packet timed out after 32000ms with no response

It just means that there has been some issue and some asynchronous communication between the Radio and Asterisk. Be patient and try another handheld or try killing all running processes and restart the system.

- OpenBTS is an Open Source project released under Open Source licenses and therefore in constant development and evolution, I recommend sticking to a version that has been tested by multiple organizations and for which there is extended documentation, but if you feel brave and want to update to the latest and greatest, here is a link to do so:
https://wush.net/trac/rangepublic/wiki/UpdateOpenBTS

- **Useful Links**

https://wush.net/trac/rangepublic/wiki/BuildInstallRun
http://wush.net/trac/rangepublic/wiki/sqlie3ODBC
https://wush.net/trac/rangepublic/wiki/UpdateOpenBTS
http://gnuradio.org/redmine/projects/gnuradio/wiki/UbuntuInstall
http://code.ettus.com/redmine/ettus/projects/uhd/wiki
http://gnuradio.org/redmine/projects/gnuradio/wiki/OpenBTS
http://sqlitebrowser.sourceforge.net/
http://www.ubuntu.com/
http://www.asterisk.org/
https://github.com/ttsou/
http://www.ettus.com/products