Implementation of OpenBTS Using NI 2901 and Ubuntu 16.04

Group Members

Ahsan Mehmood	BSEE15028
Ammar Ullah Mughal	BSEE15067
Muhammad Amir Altaf	BSEE15072

Course Instructor: Dr Mehboob Ur Rehman

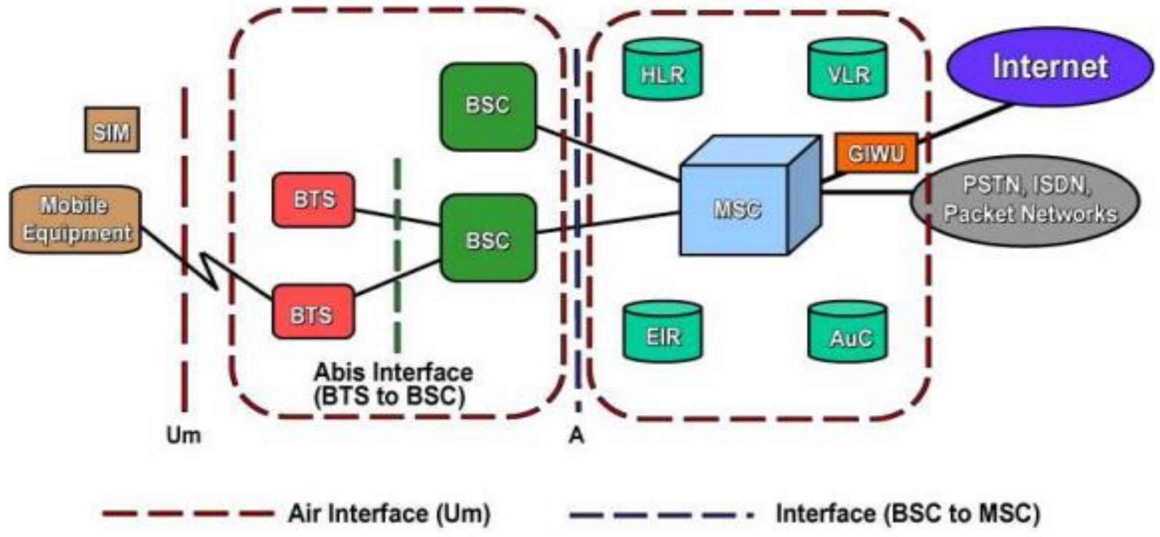
Information Technology university.

Introduction

OpenBTS

It is a software based GSM access point allowing standard GSM-compatible mobile phones to be used as SIP endpoints in Voice over IP (VOIP) networks. It has the same functionality as the BTS of a GSM Network. It is important to understand elements of BTS of GSM Network.

Architecture GSM



GSM Elements

MS (Mobile System)

It is composed by the Mobile Equipment (ME) and the SIM card. There are some important terms related to the mobile system that we need to know: IMEI, MSISDN, IMSI and TMSI.

BTS (Base Station)

The BTS contains the radio components that provide the RF air interface. Its functions are channel coding and decoding, rate adaptation, encryption, paging and uplink signal measurement.

BSC (Base Station Controller)

The BSC controls groups of BTS and manages the radio channels. It manages control messages from and to the MS. It also does encryption, paging, traffic measurement, authentication, location update and manages handover.

MSC (Mobile Switching Center)

it Is the telephone switching office for MS. Provides a service to mobiles located within a certain geographic coverage area. It is the interface to the BSS and to the PSTN. Controls call set up, routing procedures, collects billing data, compiles traffic statistics and controls the location registration and handover procedure

HLR (Home Location Register)

contains the current subscriber status, compose a register that contains data subscriber's data. It contains the IMSI of each MS, authentication parameters, services that each MS is subscribed to and special routing information. It also carries roaming number and the associated VLR

AuC (Authentication Center)

This entity works together with the HLR to perform MS authentication. It handles all the security associated with subscribers.

EIR (Equipment Identity Register)

It consists on a centralized database for validating the IMEI. EIR contains lists of IMEIs and classifies them in three ways: White List when IMEIs are valid, Black List when IMEIS are invalid (stolen) or Grey List when IMEI are suspicious or have problems.

Implementation Procedure

Hardware Required

NI2901 USPRP or any openBTS sported SDR with 2 antennas

Mobile Phone and Sim card

A PC with Ubuntu16.04 installed

Software and applications Required

Following commands can be used to install openBTS on a fresh PC.

Git Compatibility

```
$ sudo apt-get install software-properties-common python-software-properties

$ sudo add-apt-repository ppa:git-core/ppa

$ sudo apt-get update

$ sudo apt-get install git

$ git --version

git version xxxxxx
```

Now that you have Git installed, you can proceed to downloading the development scripts.

Downloading the Code

```
git clone https://github.com/RangeNetworks/dev.git
```

After this command go to home see there will be a folder named “dev”.

Now, to download all of the components, simply run the clone.sh script:

```
$ cd dev

$ ./clone.sh

$ ./switchto.sh master
```

The current version target can be listed for each component by using the state.sh script. This script also lists any outstanding local changes for each component

```
$ ./state.sh
```

Building the Code

Now connect USRP or SDR and using following command confirm the connectivity and “type” USRP.

```
$ uhd_find_devices
```

```
linux; GNU C++ version 4.6.3; Boost_104601; UHD_003.007.002-release

UHD Device 0

Device Address:

type: b200

addr: 192.168.10.2

name:

serial: XXXXXX
```

Now, we know the type of USRP. We will build the code for B200.

```
$ ./build.sh B200
```

ERROR:

If there occurs an error while running above command, try to reinstall Ubuntu and run commands from beginning.

if command runs successfully move forward.

Now, go to home/dev/BUILDS/ here will a folder. See the name of that folder e.g.

2014-07-29--20-44-51 use that in following command

```
$ ls dev/BUILDS/2014-07-29--20-44-51/*.deb

liba53_0.1_i386.deb                      range-asterisk-config_5.0_all.deb

libcoredumper1_1.2.1-1_i386.deb          range-configs_5.0_all.deb

libcoredumper-dev_1.2.1-1_i386.deb        sipauthserve_5.0_i386.deb

openbts_5.0_i386.deb                     smqueue_5.0_i386.deb

range-asterisk_11.7.0.4_i386.deb
```

Congratulations! You can now move on to installing and starting each component, as

well as learning what purpose each serves.

Installation

```
$ cd dev/BUILDS/2014-07-29--20-44-51/
```

Installing Dependencies

Execute the following commands to define an additional repository source for ZeroMQ, a library that all the components use:

```
$ sudo apt-get install software-properties-common python-software-properties

$ sudo add-apt-repository ppa:chris-lea/zeromq

$ sudo apt-get update
```

Coredumper library

OpenBTS uses the coredumper shared library to produce meaningful debugging information if OpenBTS crashes. Google originally wrote it and there are actually two

libcoredumper packages: libcoredumper-dev contains development files needed to

compile programs that utilize the coredumper library, and libcoredumper contains the

shared library that applications load at runtime:

```
$ sudo dpkg -i libcoredumper*.deb
```

A5/3 library

OpenBTS uses the A5/3 shared library to support call encryption. It contains cryptographic routines that must be distributed separately from OpenBTS:

```
$ sudo dpkg -i liba*.deb
```

System configuration

```
$ sudo dpkg -i range-configs*l.deb
```

Asterisk

```
$ sudo dpkg -i range-asterisk*.deb
$ sudo apt-get install -f
```

SIPAuthServe

$ sudo dpkg -i sipauthserve*.deb
$ sudo apt-get install –f

SMQueue

$ sudo dpkg -i smqueue*.deb
$ sudo apt-get install –f

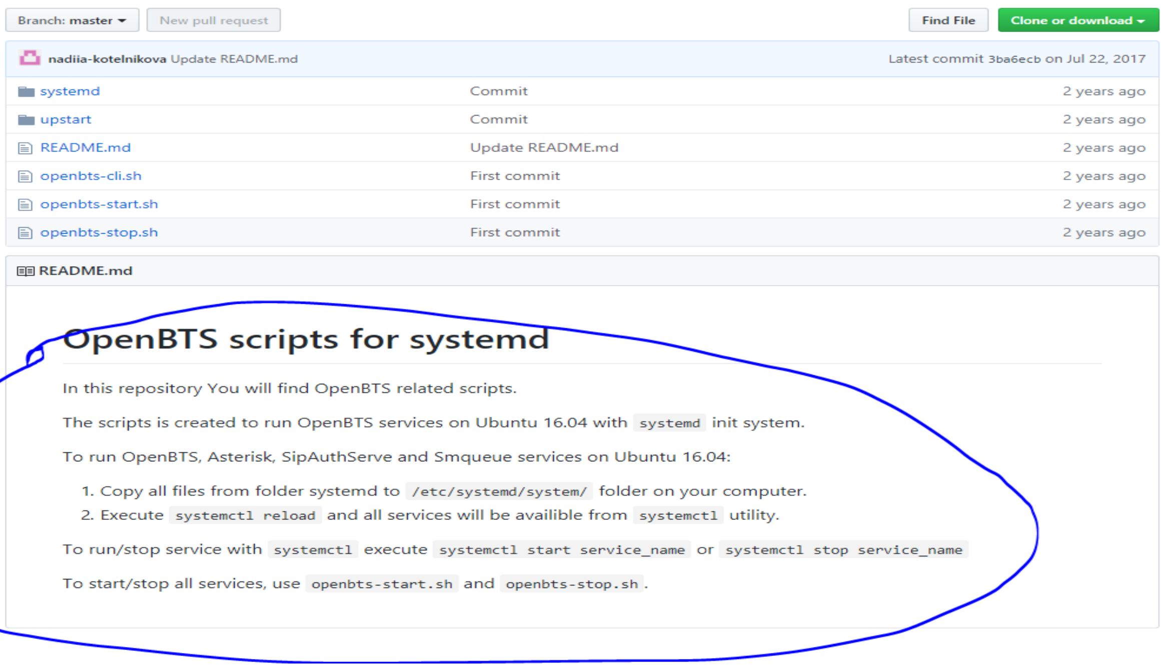
OpenBTS

$ sudo dpkg -i openbts*.deb
$ sudo apt-get install –f

Configuring the OpenBTS for systemd services

In newer versions of Ubuntu systemd is being used instead of upstart services. So, we need to configure Ubuntu to start services. For This, go to and follow the procedure highlighted

https://github.com/nadiia-kotelnikova/openbts_systemd_scripts


Branch: master ▾ | New pull request | Find File | Clone or download ▾

nadiia-kotelnikova Update README.md — Latest commit 3ba6ecb on Jul 22, 2017

systemd	Commit	2 years ago
upstart	Commit	2 years ago
README.md	Update README.md	2 years ago
openbts-cli.sh	First commit	2 years ago
openbts-start.sh	First commit	2 years ago
openbts-stop.sh	First commit	2 years ago

README.md

OpenBTS scripts for systemd

In this repository You will find OpenBTS related scripts.

The scripts is created to run OpenBTS services on Ubuntu 16.04 with `systemd` init system.

To run OpenBTS, Asterisk, SipAuthServe and Smqueue services on Ubuntu 16.04:

1. Copy all files from folder systemd to `/etc/systemd/system/` folder on your computer.
2. Execute `systemctl reload` and all services will be availible from `systemctl` utility.

To run/stop service with `systemctl` execute `systemctl start service_name` or `systemctl stop service_name`

To start/stop all services, use `openbts-start.sh` and `openbts-stop.sh`.

Possible error and its solution

If you are unable to copy and paste files as described in following picture. Run Following command in a new terminal window,

```
sudo add-apt-repository ppa:gnome3-team/gnome3
sudo apt-get update && sudo apt-get install nautilus
```

if you are able to copy and pate files in specified folder skip above step.

Start and stop services

To start all components, execute the following:

```
$ sudo systemctl start asterisk
$ sudo systemctl start sipauthserve
$ sudo systemctl start smqueue
$ sudo systemctl start openbts
```

Conversely, to stop all components, use:

Installation | 13

```
$ sudo systemctl stop openbts
$ sudo systemctl stop asterisk
$ sudo systemctl stop sipauthserve
$ sudo systemctl stop smqueue
```

Initial start

Now conform the connectivity of USRP or SDR using

```
$ uhd_find_devices
```

If USRP or SDR is connected run following commands

```
$ cd /OpenBTS
$ sudo ./transceiver
[sudo] password for (your pc name):
linux; GNU C++ version 4.6.3; Boost_104601; UHD_003.006.002-release
Using internal clock reference
-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
```

Now, transmitting and receiver indicator led on USRP must glow.

Starting Up the Network

```
$ sudo systemctl start openbts
```

The Configuration System and CLI

```
$ sudo /OpenBTS/OpenBTSCLI
```

```
OpenBTS> config GSM.Radio
GSM.Radio.ARFCNs 1 [default]
GSM.Radio.Band 900 [default]
GSM.Radio.C0 51 [default]
GSM.Radio.MaxExpectedDelaySpread 4 [default]
GSM.Radio.PowerManager.MaxAttenDB 10 [default]
GSM.Radio.PowerManager.MinAttenDB 0 [default]
GSM.Radio.RSSITarget -50 [default]
GSM.Radio.SNRTarget 10 [default]
```

The GSM.Radio.Band key shows that the 900 MHz band is being used and the GSM.Radio.C0 key indicates that ARFCN #51 in that band is currently selected.

Now run following command and resolve all errors and warnings

```
OpenBTS> audit
```

```
+-------------------------------------------------------------------+
| WARNING : Factory Radio Calibration [key current-value (factory)] |
| To use the factory value again, execute: rmconfig key |
+-------------------------------------------------------------------+
TRX.RadioFrequencyOffset "132" ("116")
```

In above response there is only one WARNING. If you see more errors or warnings resolve all errors and warning as follows

```
OpenBTS> rmconfig TRX.RadioFrequencyOffset
TRX.RadioFrequencyOffset set back to its default value
TRX.RadioFrequencyOffset is static; change takes effect on restart
```

Run above command for all errors and warnings.

Then,

For starters, set GSM.Radio.RxGain to 10:

```
OpenBTS> devconfig GSM.Radio.RxGain 10
GSM.Radio.RxGain changed from "52" to "10"
GSM.Radio.RxGain is static; change takes effect on restart
```

Searching for the Network

Now that the radio is calibrated and the settings are confirmed, you will use a handset to search for the newly created network. Each handset’ s menu is different but the item is usually similar to “Carrier Selection” or “Network Selection.” The process for manually selecting a different carrier on Android is detailed in Figure 2-1.

1. Launch the “Settings” application from the Android menu system.
2. Select “More.”
3. Select “Mobile networks.”
4. Select “Network operators.” This may or may not start a search. If it does not, select “Search networks.”
5. Once the search has finished, a list of available carrier networks is presented.

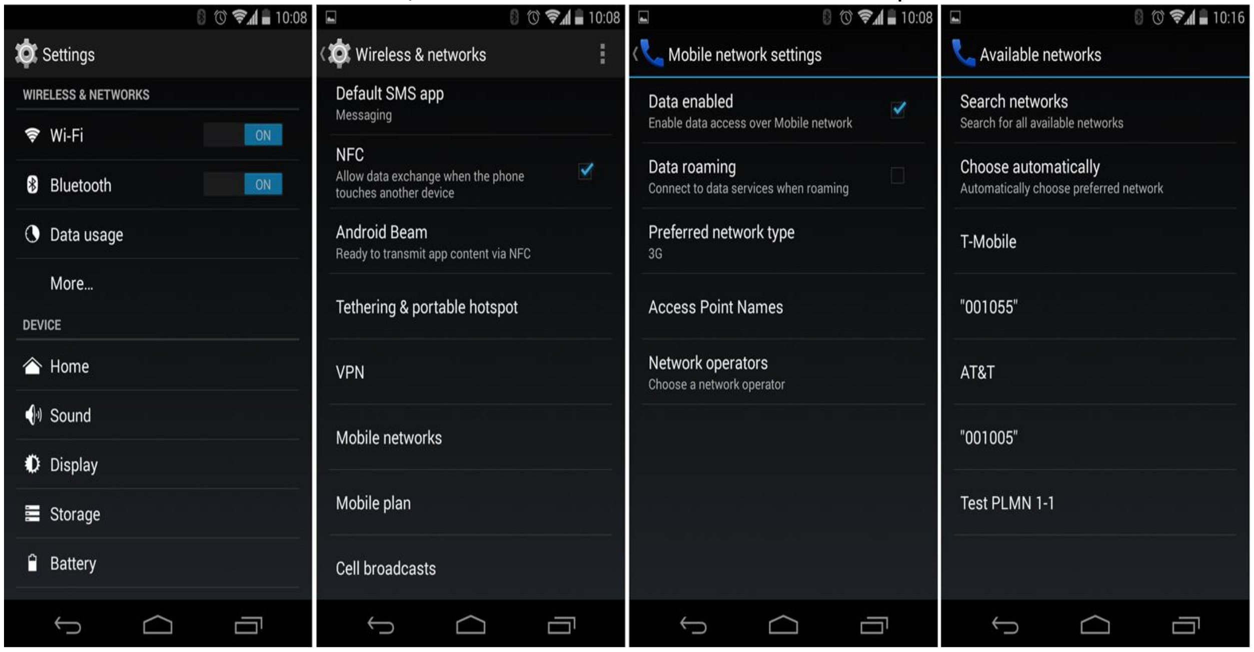
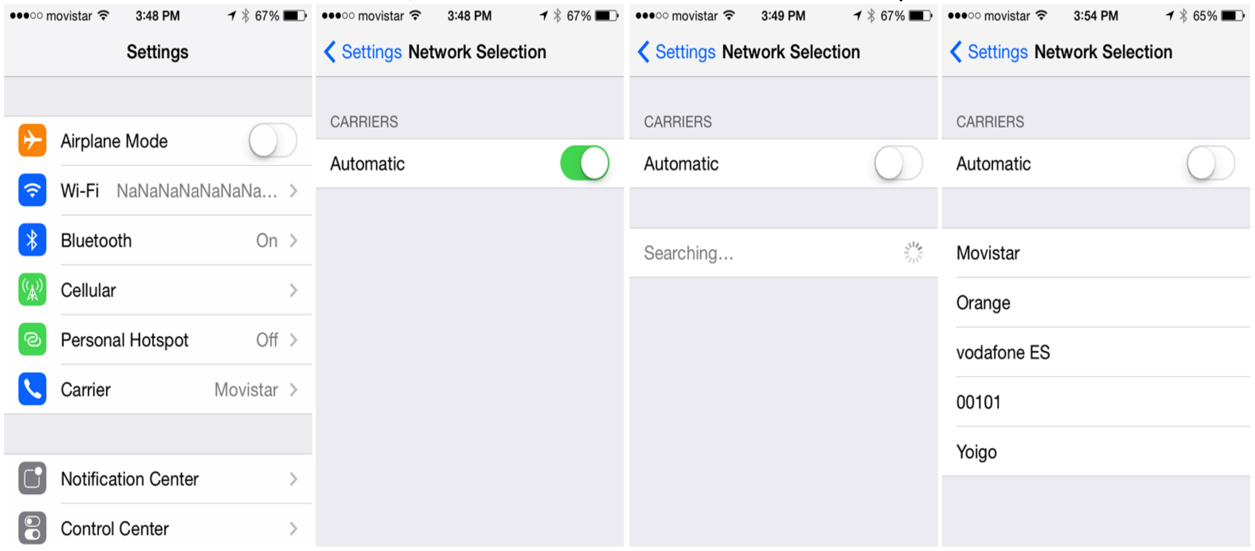


Figure 2-1. Android manual carrier selection

The process for manually selecting a different carrier on iOS 7 is detailed in Figure 2-2.

1. From the home screen, open the “Settings” app.
2. Select “Carrier.”

3. On the “Network Selection” screen, disable the automatic carrier selection.
4. The handset will now search for available carrier networks.
5. Once the search has finished, a list of available carrier networks is presented.



Here we see the test network in the list of selectable carriers. Depending on the handset model, firmware, and SIM used, the network ID will be displayed as “00101,” “001-01,” “Test PLMN 1-1,” or the GSM shortname of “OpenBTS.”

```
OpenBTS> noise
noise RSSI is -68 dB wrt full scale
MS RSSI target is -50 dB wrt full scale
INFO: the current noise level is acceptable.
OpenBTS> power
current downlink power 0 dB wrt full scale
```

First Connection

For connection we enable open registration using following command,

```
OpenBTS> config Control.LUR.OpenRegistration ".*"
OpenBTS> config Control.LUR.SendTMSIs "1"
OpenBTS> rxgain 20
-- courtesy Neel Pandeya and Ralph A. Schmid on the Openbts-discuss mailing list
```

This will enable open registration on the network and will enable TMSI assignment. You can now connect to this network (which will show up as 00101 or something on an Android phone) and call **2600**. **2600** is an echo service which enables you to check your OpenBTS setup.