Exploring OpenBTS: Lab Write-up

In this lab session, we explored the functionalities of OpenBTS, an open-source GSM base station, and its integration with mobile devices. The objective was to understand the setup process, customize the base station, establish a connection with a mobile device, and confirm connectivity by sending an SMS message.

This lab takes place inside the OpenBTS virtual machine on ialab.dsu.edu

1 SET UP

Since the remainder of the lab relies on the presence of the radio device, I began the lab by executing the `lsusb` command to confirm the connection status of the OpenBTS radio.

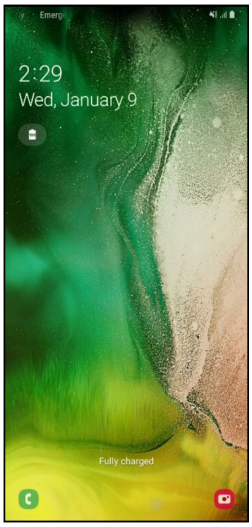
```
dsu@ubuntu:~$ lsusb
Bus 002 Device 002: ID 2500:0020
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 013: ID 04e8:6860 Samsung Electronics Co., Ltd Galaxy (MTP)
Bus 001 Device 012: ID 04e8:6860 Samsung Electronics Co., Ltd Galaxy (MTP)
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Next, I executed the `adb devices -l` command in the terminal in order to view connected Android devices and obtain their serial numbers.

```
dsu@ubuntu:~$ adb devices -l
List of devices attached
R58MA5S9CQZ            device usb:1-1.3 product:a10ub model:SM_A105M device:a10
R9WN50YW86J            device usb:1-1.4 product:a10sub model:SM_A107M device:a10s
```

By selecting a device and extracting the serial number (R9WN50YW86J), I initiated the connection with the phone by executing the command `scrcpy -s R9WN50YW86J`. This command allowed me to mirror the phone's screen onto my desktop, which granted access and control of the device's interface and functionalities directly from my computer.

```
dsu@ubuntu:~$ scrcpy -s R9WN50YW86J
INFO: scrcpy 1.12.1 <https://github.com/Genymobile/scrcpy>
502 KB/s (26238 bytes in 0.050s)
INFO: Initial texture: 720x1520
```



Next, I launched OpenBTS by navigating to the appropriate directory and executing `sudo ./OpenBTS/`. This interface serves as the platform for executing various commands, including configuration changes, which are processed by OpenBTS to customize and manage the behavior of the base station. [1]

```
dsu@ubuntu:/OpenBTS$ sudo ./OpenBTS
ALERT 12184:12184 2024-03-08T00:31:19.7 OpenBTS.cpp:595:main: OpenBTS (re)starting, ver 5.0-master
build date/time 2018-01-04T07:13:10
ALERT 12184:12184 2024-03-08T00:31:19.7 OpenBTS.cpp:596:main: OpenBTS reading config file /etc/Open
BTS/OpenBTS.db
1709886679.717282 140432561239872:
```

2 CUSTOMIZATION

Once inside the BTS command line, I modified the Mobile Network Code (MNC) key using the command config `GSM.Identity.MNC 17`, which resulted in the network being identified as 00117.

```
OpenBTS> config GSM.Identity.MNC 17
GSM.Identity.MNC changed from "11" to "17"
```

Next, I set the short name for the network to "DontBlink" using the command `config GSM.Identity.ShortName DontBlink`. Assigning a unique short name to the network is important as it facilitates easy identification within the connected mobile device, especially as it moves within the coverage area. [1]

```
OpenBTS> config GSM.Identity.ShortName DontBlink
GSM.Identity.ShortName changed from "Range" to "DontBlink"
```

Before connecting the device, I ensured that Open Registration was enabled within the OpenBTS system. This step is important as it allows devices to join the network and use its services even if it has not been assigned a phone number yet. While some services may be limited until the setup process is complete, devices setup in this manner can receive text messages. This ensures that devices can stay connected and receive important communications while they complete the setup process. [1] Since this feature is disabled by default, I enabled it using the command `config Control.LUR.OpenRegistration .*`.

```
OpenBTS> config control
control - no keys matched, developer/factory keys can be accessed with "devconfig".
OpenBTS> config Control
Control.GSMTAP.GPRS 0     [default]
Control.GSMTAP.GSM 0     [default]
Control.GSMTAP.TargetIP 127.0.0.1     [default]
Control.LUR.404RejectCause 0x04     [default]
Control.LUR.AttachDetach 1     [default]
Control.LUR.FailMode ACCEPT     [default]
Control.LUR.FailedRegistration.Message Your handset is not provisioned for this network.
      [default]
Control.LUR.FailedRegistration.ShortCode 1000     [default]
Control.LUR.NormalRegistration.Message (disabled)     [default]
Control.LUR.NormalRegistration.ShortCode 0000     [default]
Control.LUR.OpenRegistration (disabled)     [default]
Control.LUR.OpenRegistration.Message Welcome to the test network.  Your IMSI is      [de
fault]
Control.LUR.OpenRegistration.Reject (disabled)     [default]
Control.LUR.OpenRegistration.ShortCode 101     [default]
Control.LUR.QueryClassmark 0     [default]
Control.LUR.QueryIMEI 0     [default]
Control.LUR.RegistrationMessageFrequency FIRST     [default]
Control.LUR.SendTMSIs 0     [default]
Control.LUR.UnprovisionedRejectCause 0x04     [default]
Control.Reporting.PhysStatusTable /var/run/ChannelTable.db     [default]
Control.Reporting.StatsTable /var/log/OpenBTSStats.db     [default]
Control.Reporting.TMSITable /var/run/TMSITable.db     [default]
Control.Reporting.TransactionTable /var/run/TransactionTable.db     [default]
Control.SMSCB.Table (disabled)     [default]
Control.TMSITable.MaxAge 576     [default]
Control.VEA 0     [default]
GPRS.ChannelCodingControl.RSSI -40     [default]
OpenBTS> config Control.LUR.OpenRegistration .*
Control.LUR.OpenRegistration changed from "" to ".*"
```

3 CONNECTION

After these initial configurations, I attempted to connect to the mobile device. However, my assigned network identity (00117) did not show up as expected. To resolve this issue, I began by adjusting the Absolute Radio Frequency Channel Number (ARFCN). It is important for each tower to have its own unique frequency because if two nearby towers use the same frequency, their signals can interfere with each other and cause disruptions - similar to trying to listen to two radio stations broadcasting on the same frequency simultaneously. [1] Therefore, to resolve any possible network connectivity issues, I set the ARFCN to channel 10 by using the command `config GSM.Radio.C0 10`.

```
OpenBTS> config GSM.Radio.C0 10
GSM.Radio.C0 changed from "50" to "10"
GSM.Radio.C0 is static; change takes effect on restart
```

Next, I updated the gain settings within OpenBTS by setting it to the suggested value of '10' by using the command `rxgain 10`. This adjustment helps ensure that the signals transmitted and received by OpenBTS are set to an appropriate level in order to achieve clear and reliable communication. [1] Therefore, by lowering the gain from "47" to "10", I aimed to reduce the sensitivity of the BTS receiver to external signals and make it less susceptible to interference.

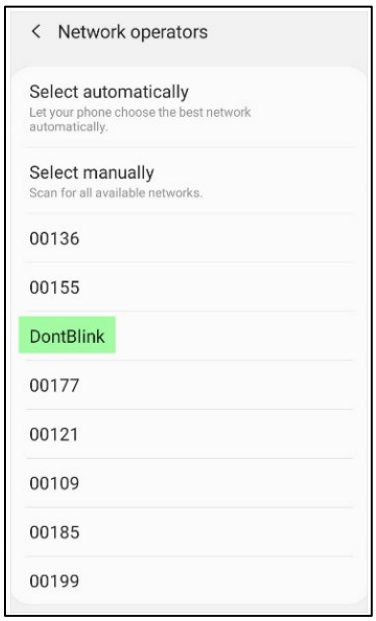
```
OpenBTS> rxgain 10
current RX gain is 47 dB
new RX gain is 10 dB
```

After restarting to apply these changes, I encountered another challenge as OpenBTS failed to connect due to a binding issue on port 405060. To address this issue, I executed the command `sudo netstat -tulnlp` to identify the process ID (PID) responsible for occupying the port. After locating the associated OpenBTS process (PID 11636), I was able to terminate the process and free up the necessary resources. These steps resolved the binding issue and enabled OpenBTS to successfully initialize and reestablish its required connections.

```
1709886539.869909 140205278746432:
Starting the system...
ALERT 12042:12042 2024-03-08T00:28:59.8 TRXManager.cpp:492:setTSC: SETTSC failed with status 1
EMERG 12042:12042 2024-03-08T00:29:00.2 NodeManager.cpp:89:start: Could not bind commandsAddress:
 tcp://127.0.0.1:45060 (possibly in use?)  Exiting...
```

```
dsu@ubuntu:/OpenBTS$ sudo kill 11636
```

Finally, I was able to restart OpenBTS and establish a connection to the mobile device. By accessing the Network Operators menu and manually scanning for available networks, I identified my network by its short name, "DontBlink." After selecting this option, the mobile device successfully connected to OpenBTS.



4 CONNECTION VERIFICATION AND COMMUNICATION

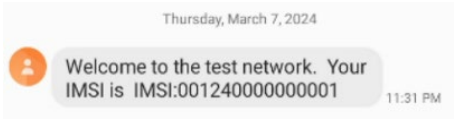
To confirm the success of the connection, I executed the command `tmsis`. The output revealed only one device, which made it easy to match the International Mobile Equipment Identity (IMEI) with the new device.

```
OpenBTS> tmsis
IMSI            TMSI IMEI            AUTH CREATED ACCESSED TMSI_ASSIGNED
001240000000001 -    353414117636670 2    10s     10s      0
```

However, to ensure accuracy, I also accessed the 'About' section of the Galaxy A10s to retrieve the IMEI number listed on the device.

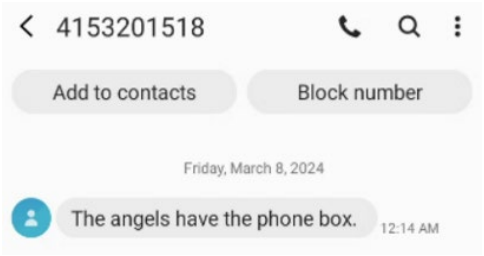
As expected, the IMEI values matched, verifying the IMEI number as 353414117636676. Therefore, with these matching IMEI values, I can confidently assert that the corresponding International Mobile Subscriber Identity (IMSI) listed in the `tmsis` output is the IMSI for the new device.

Additionally, the default Open Registration Message, managed by `Control.LUR.OpenRegistration.Message`, welcomes new devices to the network by broadcasting their IMSI. This message serves as an additional confirmation of the unique identifier, and can be observed through the SMS messages on the device. However, it is crucial to emphasize that broadcasting the IMSI via a message is highly unusual and specific to this lab environment.



Lastly, I utilized the previously discovered IMSI to send a message and test the communication capabilities of the system. Using the `'sendsms'` command followed by the IMSI, I initiated the message transmission process within OpenBTS.

```
OpenBTS> sendsms 001240000000001 4153201518 The angels have the phone box.
message submitted for delivery
```



References

[1] M. Iedema, "Getting Started with OpenBTS," O'Reilly Media, Inc., 2015.