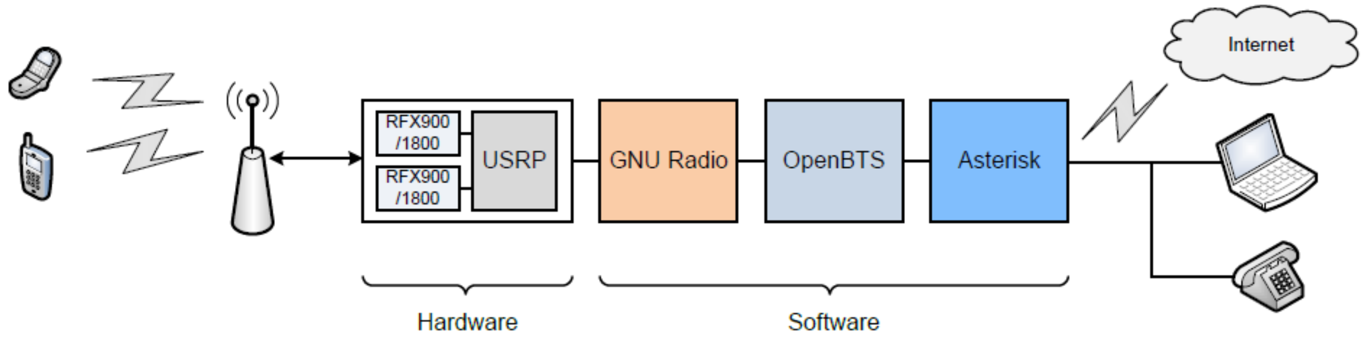
Introduction:

OpenBTS is an open-source UNIX application that uses the Universal Software Radio Peripheral (USRP) to present a GSM air interface to standard GSM handset and uses the Asterisk software PBX to connect calls.

A GSM network is a complex system composed by several components. The last mile of this system is the BTS (Base Transceiver Station). The BTS is responsible to transmit and receive the RF (Radio Frequency) signals to the user terminal (cell phone, PDA, modem, etc). The BTS's are controlled by a BSC (Base Station Controller) that is connected to the MSC/VLR (Mobile Switching Center/Visitor Location Register). Basically, the MSC/VLR is responsible to authenticate the user against the database (HLR - Home Location Register, AuC - Authentication Center).

The OpenBTS uses the USRP hardware to receive and transmit the GSM signaling. This is done by using the GNU Radio framework. The Asterisk is used to interface the GSM calls between the cellular phones under the OpenBTS network. Any other device that can be connected to the Asterisk can be also used. The following figure gives the overview of the system and how each component fits in the entire system.



The **GNU Radio** is a free software development toolkit that provides the signal processing runtime and processing blocks to implement software radios using readily-available, low-cost external RF hardware (in this case the USRP). The **USRP** (Universal Software Radio Peripheral) is a hardware designed by Ettus Research 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. There are several daughterboard's that can be used with the USRP covering from DC to 5.9 GHz. In our project we are using the RFX900 to cover the GSM 850 and 900 bands.

Problem Statement:

Task/ Open Issue 1: Send a SMS delivery report.

Send back a delivery report in L4 successfully through smqueue.

Task/Open issue 2: SMS Email Gateway Hack.

Send/receive SMS through e-mail with automatic return addresses that really work, even without carrier cooperation. Steps to make this work:

- Outgoing SMS gets an e-mail address <extension>@<server> prefixed to the message and then gets send through an HTTP gateway service.

- When the other party responds, most carriers will route the response to an e-mail address via SMTP.
- We have a server with an SMTP interface that receives the messages and echoes them as SIP to smqueue.
- Smqueue delivers the message to the handset as SMS.

Sub-tasks/Road to the solution:

Installation of the OpenBTS setup and to be able to make calls/exchange SMS is the primary task to be accomplished before proceeding with the other sub-tasks involved with both the targets.
Also, general study of the OpenBTS system including every component is required to get an idea about the different modules and also to troubleshoot inter-dependency bugs.

SMS delivery reports

1. Study the smqueue module in detail.
2. Study the interface between the smqueue module and OpenBTS.
3. Study the general message delivery mechanism to the MS in detail.
   a. Smqueue sends a SIP message to OpenBTS addressed to the recipient's IMSI.
   b. OpenBTS will attempt to deliver the message as the payload of an RP-DATA message.
   c. OpenBTS sends 200 OK to smqueue if delivery is successful.
   d. Once this is received, smqueue deletes the message from the messages table.
4. Necessary hooks should be coded in the smqueue module and the smqueue-OpenBTS interface so that a success delivery report is sent when smqueue receives the 200 OK messages from OpenBTS.
   a. Get the original sender's IMSI.
   b. Using the messaging process described in 3, send a delivery report to the sender.
   c. To extend this to support delivery failure messages, instead of capturing the 200 OK message, look for the SIP error message.
5. The above process should be done before the message is deleted from the messages table.

E-Mail gateway hack

1. Study the interface between SMTP and smqueue.
2. Study the message format from the SMTP interface to smqueue. This will be a SIP message.
3. Smqueue has to extract the details from this message,
   a. IMSI or MSISDN number
   b. The actual message – this is a reply to the SMS sent by the user as an email.
4. Smqueue should communicate with OpenBTS to send the message as a SMS to the mobile phone.
   a. This message will be a SIP message.
   b. The rest will be taken care by OpenBTS.

   The tasks listed are in the order of dependency. This has to be the flow of execution.

There are certain other dependencies since this is an open source project. We assume that the existing system supports SMS messaging. Also, the smqueue module should be functional. This is also a threat to the project, since existing failures in the above mentioned dependencies might delay the project cycle.

Timelines and Schedule:

Task	Description	Time
1	Learning and Understanding all the modules: OpenBTS (code and working), GSM, USRP, SIP and Asterisk	2 to 3 weeks
2	Basic Hardware and Software set up (install all the software components on the needed hardware and connect a call/exchange text message)	2 weeks (can happen in parallel with task1)
3	Understanding the SMS service in OpenBTS: smqueue component (code and functioning) and its interfacing with OpenBTS	2 weeks
4	In depth understanding of the general message delivery system; identify the smqueue modules which need to be modified in order to solve the problem.	2 weeks
5	Modify and test the code to implement the needed message delivery report (changes in smqueue and smqueue-OpenBTS interface)	1 to 2 weeks
6	Study the interface between SMTP and smqueue; Study the SIP message format from the SMTP interface to smqueue	1 week
7	Modify the code to extract the details from the SIP message and make smqueue interface with OpenBTS to deliver the message as SMS to the MS.	1 to 2 weeks
8	Compilation and Integration of all the changes; testing	2 weeks

References:

1. OpenBTS : http://openbts.org/
2. GitHub : https://github.com/RangeNetworks/dev