Hochschule München
Fakultät für Mathematik und Informatik

Projektdokumentation Mobile Netze

Handover mit OpenBSC und OpenBTS

Autoren: Max Eschenbacher
Stefan Giggenbach
Thomas Waldecker

Abgabe: 11.03.2012

betreut von: Prof. Dr. Zugenmaier

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Handover . . . 3
1.2 Projektziel und -durchführung . . . 5

2 OpenBSC **6**
2.1 Überblick . . . 6
2.2 Installation und Konfiguration . . . 6

3 OpenBTS **8**
3.1 Aufbau und Zusammenspiel . . . 9
3.1.1 Komponenten . . . 9
3.1.2 Datenbanken . . . 12
3.1.3 GSM/SIP-Abläufe . . . 12
3.2 Installation . . . 15
3.2.1 GNUradio . . . 15
3.2.2 OpenBTS . . . 16
3.2.3 Subscriber Registry und Sipauthserve . . . 17
3.2.4 Smqueue . . . 17
3.3 Konfiguration . . . 18
3.3.1 OpenBTS . . . 18
3.3.2 Sipauthserve . . . 19
3.3.3 Smqueue . . . 19
3.3.4 Asterisk . . . 19
3.4 Benutzung von OpenBTS . . . 22
3.4.1 Start der Dienste . . . 22
3.4.2 Command Line Interface (CLI) . . . 23
3.4.3 Registrierung einer MS an OpenBTS . . . 24

4 OpenBTS Software-Architektur **24**
4.1 Überblick . . . 24
4.2 Verwendete Klassen . . . 25

5 Erweiterung von OpenBTS **25**
5.1 Measurement Report 25
5.2 Handover Modul 29
5.3 Inter OpenBTS Handover 29

6 Analyse **29**
6.1 Vorbereitung 29
6.1.1 Tracen auf dem Um Interface mit einem Nokia 3310 . 29
6.1.2 Patchen von Wireshark 30
6.2 Intra BSC Handover mit OpenBSC und zwei nanoBTS 30
6.2.1 Aufbau 30

A Anhang **31**
A.1 Literaturverzeichnis 31

1 Einleitung

Von: Stefan Giggenbach

Im Rahmen des Moduls Mobile Netze im Masterstudiengang Informatik wird mit der Durchführung einer Projektarbeit, dass in der Vorlesung vermittelte Wissen praxisnah vertieft und ergänzt. In der vorliegenden Projektarbeit wurde die Handover-Funktionalität in einem GSM-Netzwerk näher untersucht. In diesem Kapitel wird nach einer theoretischen Einführung in die Handover-Thematik das Projektziel und die entsprechende Vorgehensweise beschrieben.

1.1 Handover

Der Handover stellt in einem GSM-Netzwerk eine wichtige Aufgabe des Mobility Management dar. Ändert ein Teilnehmen bei aktiver Verbindung seinen Standort, ist es möglich das er den von einer Funkzelle abgedeckten Bereich verlässt. In einem solchen Fall wird die Verbindung durch den Wechsel zu einer benachbarten Funkzelle (Handover) aufrecht erhalten. Grundsätzlich unterscheidet man in einem GSM-Netzwerk folgende Handoverszenarien:

- Intra BSC Handover
- Inter BSC Handover
- Inter MSC Handover
- Subsequent Inter MSC Handover

Die einzelnen Szenarien werden in [1] ausführlich beschrieben. Im folgenden wird nur der Ablauf des Intra BSC Handover näher betrachtet, der für das Verstädnis dieser Arbeit entscheidend ist. Aus Sicht der Mobile Station unterscheiden sich die genannten Handoverszenarien nicht.

Abbildung 1 zeigt das Ablaufdiagramm eines Intra BSC Handover. Während einer aktiven Verbindung wird der BSC in regelmäßigen Zeitabständen über die Signalqualität im Up- und Downstream informiert. Zu diesem Zweck sendet die Mobile Station über den SACCH sogenannte Measurement Reports, die anschließend im BSC zur Bestimmung der Downstream-Signalqualität ausgewertet werden. In den Measurement Reports sind neben Messergebnissen zur aktuell verwendeten BTS auch Messergebnisse zu benachbarten BTS, die auf Anweisung des BSC während den Sendepausen von der Mobile Station ermittelt werden. Die Signalqualität des Upstreams wird durch Messergebnisse aus der entsprechenden BTS ebenfalls im BSC berechnet. Der BSC kann aufgrund der eingehenden Measurement Reports zu dem Ergebnis kommen, dass ein Handover zwischen

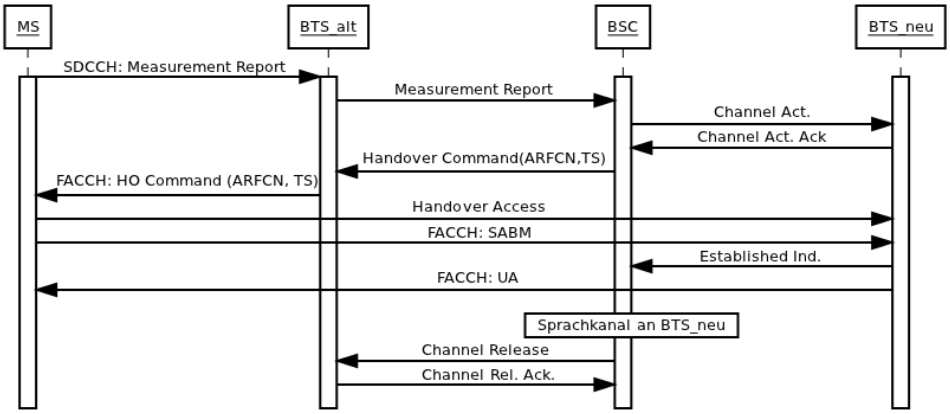

Abbildung 1: Ablaufdiagramm eines Intra BSC Handover

zwei benachbarten BTS notwendig ist, um einen Abbruch der Verbindung zu verhindern. Nach der Entscheidung des BSC einen Handover durchzuführen, wird im ersten Schritt ein TCH in der neuen BTS aufgebaut. War dieser Vorgang erfolgreich, wird der Mobile Station über den FACCH der bestehenden Verbindung ein Handover Command übermittelt. Das Handover Command enthält als Parameter die Frequenz und den Timeslot des TCH der neuen BTS. Nach der Synchronisation der Mobile Station mit der neuen BTS, sendet es in vier aufeinanderfolgenden Bursts eine Handover Access Message und anschließend eine Set Asynchronous Balanced Mode Message. Die neue BTS quittiert den erfolgreichen Handover mit einem Established Indicator gegenüber dem BSC und einer UA Massage gegenüber der Mobile Station. Nachdem der BSC die Verbindung auf den neuen TCH umschaltet, wird der TCH in der alten BTS abgebaut. Der Handover Vorgang ist damit abgeschlossen.

Die wichtigesten Punkte des Ablaufs für die Analyse und Impelmentierung einer Handover-Funktionalität sind damit:

- Erfassung und Auswertung der Measurement Reports
- Logik für die Entscheidungsfindung eines Handover
- Inter BTS Kommunikation zum Aufbau eines neuen TCH
- Erzeugen und Senden eines Handover Command
- Umschalten der bestehenden Verbindung und Abbau des alten TCH

1.2 Projektziel und -durchführung

Ziel der Projektarbeit ist die Integration der in Abschnitt 1.1 eingeführten Handover-Funktionalität in die Open Source Software OpenBTS. Das OpenBTS Projekt ermöglicht zusammen mit einer entsprechenden Radio-Hardware und zusätzlichen Software-Komponenten (GNURadio und Asterisk), den Betrieb eines GSM-Netzwerks. Mit der kommerziell vertriebenen Version der Software ist ein Handover zwischen zwei BTS bereits möglich. Die Vorraussetzungen für eine erfolgreiche Integration eines Handover-Moduls sind somit gegeben. Die Architektur, Installation und Konfiguration des im Anschluss für die Implementierung verwendeten OpenBTS-Systems werden in Kapitel 3 ausführlich beschrieben.

Noch vor der Integrations- und Implementierungsphase wird der Ablauf eines Handover genauer analysiert. Zu diesem Zweck wird ein OpenBSC-System verwendet, mit dem das in Abschnitt 1.1 eingeführte Handoverszenario reproduziert werden kann. Der Aufbau, sowie die Installation und Konfiguration des OpenBSC-Systems für die Durchführung eines Intra BSC Handover werden in Kapitel 2 behandelt.

Da brauchbare Dokumentationen zur Implementierung von OpenBTS nicht vorhanden sind, enthält Kapitel 4 einen Überblick zur Software-Architektur und detaillierte Beschreibungen zu verwendeten Klassen des OpenBTS-Quellcodes. Dieses Kapitel fasst die Ergebnisse der sehr aufwändigen Code-Analyse zusammen und soll zukünftigen Projektgruppen einen einfacheren Einstieg in die Weiterentwicklung der OpenBTS-Software ermöglichen.

Der Architekturentwurf für die Integration und die durchgeführten Implementierungsarbeiten des Handover-Moduls werden in Kapitel 5 behandelt. Die Arbeiten konnten in dem zur Verfügung stehenden zeiltichen Rahmen nicht vollständig abgeschlossen werden. Der aktuelle Entwicklungsstand wird in sich abgeschlossen festgehalten und Lösungsansätze für weitere Arbeiten werden aufgezeigt.

Die Analyse der durchgeführten Handover (sowohl mit OpenBSC als auch mit OpenBTS) wird am Ende der Arbeit in Kapitel 6 beschrieben. Dabei werden zum einen die vorbereitenden Arbeiten und eingesetzten Werkzeuge zur Erzeugung von Traces auf der Um- und Abis-Schnittstelle behandelt. Zum anderen werden die daraus gewonnen Erkenntnisse, die während der gesamten Entwicklungsarbeiten immer wieder zur Verifikation verwendet wurden, beschrieben.

2 OpenBSC

Von: Stefan Giggenbach

2.1 Überblick

Bei OpenBSC handelt es sich wie bei OpenBTS um ein Open Source Projekt. Die Entwicklung erfolgt vollständig in der Sprache C und hat keinen direkten Bezug zum OpenBTS Projekt. Der große Vorteil von OpenBSC liegt in der *network in the box* (nitb) genannten Version, die ohne zusätzliche Software-Komponenten den Betrieb eines GSM-Netzwerks ermöglicht. Mit OpenBSC wird zu einem sehr frühen Zeitpunkt im Projekt ein GSM-Netzwerk mit Handover-Funktionalität betrieben und die entsprechenden Abläufe können analysiert werden (siehe Kapitel 6).

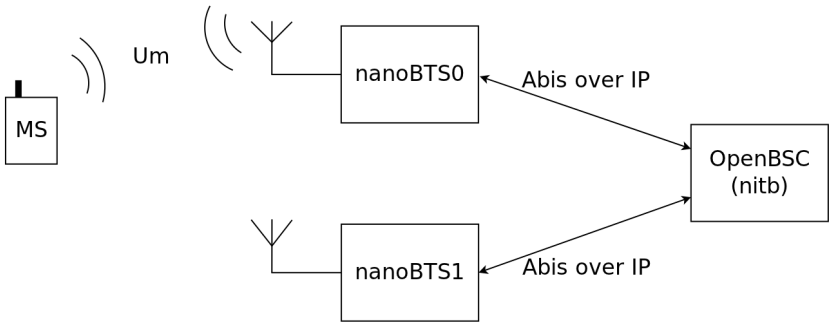

Abbildung 2: OpenBSC Versuchsaufbau

Abbildung 2 zeigt den im Projekt verwendeten Versuchsaufbau. OpenBSC übernimmt nicht nur die Aufgaben des BSC, sondern auch die des MSC. Die Teilnehmer Datenbanken HLR und VLR werden mit einer SQLite3 Datenbank realisiert. Wie in Abbildung 2 dargestellt werden zwei nanoBTS der Firma ip.access verwendet. Diese werden über getrennte Abis-over-IP-Schnittstellen an OpenBSC angebunden. Mit dem dargestellten Versuchsaufbau ist somit die Durchführung eines in Abschnitt 1.1 beschriebenen Intra BSC Handover mit verhältnismäßig geringem Installations- und Konfigurationsaufwand möglich.

2.2 Installation und Konfiguration

Die Installation von OpenBSC ist ausführlich im Wiki des Projekts [2] dokumentiert. In diesem Abschnitt werden deshalb nur die wichtigsten Punkte der Installation und die Konfiguration des Systems für den Multi-BTS-Betrieb behandelt.

OpenBSC (nitb) besteht aus insgesamt drei Komponenten:

- *libosmocore* - Die Kernbibliothek, die auch für andere Projekte (z. B. OsmoBTS) verwendet wird.
- *libosmo-abis* - Die Bibliothek zur Umsetzung der Abis- und Abis-over-IP-Schnittstellen.
- *openbsc* - Die eigentliche OpenBSC-Software, welche auch die nitb Version enthält.

Nach der Kompilierung und Installation dieser drei Komponenten können die beiden nanoBTS, die sich im selben IP-Netzwerk befinden müssen, konfiguriert werden. Dazu werden die zwei Anwendungen `./ipaccess-find` und `./ipaccess-config` im Verzeichnis `<openbsc>/src/ipaccess` benötigt. Die Verwendung der beiden Anwendungen und die benötigten Parameter zur Konfiguration werden ebenfalls im Wiki des Projekts [3] erläutert. Die bei der Konfiguration der nanoBTS vergebene UnitID wird dabei für die im Folgenden beschriebene Konfiguration von OpenBSC benötigt.

Um den Betrieb beider nanoBTS und die Handover-Funktionalität von OpenBSC zu aktivieren, muss die Konfigurationsdatei von OpenBSC modifiziert werden. Als Grundlage wird die Beispielkonfiguration `<openbsc>/doc/examples/osmo-nitb/nanobts/openbsc.cfg` verwendet. Listing 1 zeigt auszugsweise die wichtigsten Inhalte der modifizierte Konfigurationsdatei.

Listing 1: OpenBSC Konfigurationsdatei (Auszug)

```
!
! OpenBSC (0.10.1.40-2935) configuration saved from vty
.
network
 network country code 262
 mobile network code 98
 .
 handover 1
 .
 bts 0
  type nanobts
  band DCS1800
  cell_identity 0
  .
  ip.access unit_id 42 0
  .
  trx 0
   rf_locked 0
   arfcn 846
   nominal power 23
   max_power_red 22
   .
 bts 1
```

```
24   type nanobts
25   band DCS1800
26   cell_identity 1
27   .
28   ip.access unit_id 43 0
29   .
30   trx 0
31    rf_locked 0
32    arfcn 867
33    nominal power 23
34    max_power_red 22
35    .
```

Neben dem Network Country Code und dem Mobile Network Code (Zeilen 5 und 6) muss in den Netzwerkeinstellungen die Handover-Funktionalität gesetzt werden (Zeile 8). Die Konfiguration der beiden nanoBTS beschränkt sich im wesentlichen auf die Vergabe der eindeutigen CellIDs (Zeilen 13 und 26), der vorher festgelegten UnitIDs (Zeilen 15 und 28) und den beiden Frequenzen (ARFCN in Zeilen 19 und 32).

Nach der Modifikation der Konfigurationsdatei wird diese im Verzeichnis `<openbsc>/src/osmo-nitb` gespeichert. Anschließend kann das System mit dem Befehl `./openbsc` im selben Verzeichnis gestartet werden. Die Bedienung von OpenBSC erfolgt nach dem Start über eine Telnetsitzung auf Port 4242. Mit Hilfe des Command Line Interfaces der Telnetsitzung ist auch die Administration der Teilnehmerdatenbank möglich. Die Verwendung des CLI ist aufgrund der interaktiven Eingabe selbsterklärend.

Um einen Handover auszulösen, kann bei aktiver Verbindung entweder die Position einer Mobile Station verändert werden oder die Signalqualität wird durch entsprechende Schirmung der Geräte in ausreichendem Umfang reduziert. Die Analyse der mit OpenBSC durchgeführten Handover wird in Kapitel 6 detailliert beschrieben.

3 OpenBTS

Von: Max Eschenbacher

OpenBTS (Open Base Transceiver Station) ist eine freie Unix-Applikation die in C++ entwickelt wurde. Mit Hilfe eines Software Radios und der entsprechenden Hardware kann OpenBTS die GSM Luftschnittstelle *Um* simulieren. In Verbindung mit einer Private Branch Exchange (PBX) können die Mobilfunkteilnehmer untereinander, sowie, je nach Anbindung, mit VoIP- bzw. Festnetzteilnehmern telefonieren.

3.1 Aufbau und Zusammenspiel

3.1.1 Komponenten

Ein funktionsfähiges OpenBTS System besteht aus folgenden Komponenten:

- **OpenBTS**
  Die eigentliche Kernsoftware, welche (fast) den gesamten GSM-Stack oberhalb des Radios realisiert.

- **Transceiver**
  Kombination aus Radio-Hardware (USRP1) und der ansprechenden Software (GNUradio). Dadurch wird der gesamte Physical Layer der GSM *Um* Luftschnittstelle realisiert. Die eingesetzte Universal Software Radio Peripheral (USRP) ist von der Firma Ettus Research (siehe Abbildung 3) und wird über einen FA-Synthesizer (siehe Abbildung 4) mit 52MHz betrieben.

Abbildung 3: GSM-Radio USRP1

Abbildung 4: FA-Synthesizer

- **Asterisk**
  OpenBTS benutzt eine herkömmliche PBX um die Gesprächsvermittlung zu realisieren und damit das klassische Mobile Switching Center (MSC) zu ersetzen. Wir setzen dabei die freie Software namens Asterisk ein, die in OpenBTS unterstützt wird und neben der Hauptaufgabe, der Gesprächsvermittlung, weitere Features wie beispielsweise Mailbox-Services enthält.

- **Subscriber Registry**
  Die Subscriber Registry ist eine Datenbank die OpenBTS für die Subscriber Informationen nutzt. Sie ersetzt zum einen das klassische GSM Home Location Register (HLR) und zum anderen die SIP Registry von Asterisk.

- **Smqueue**
  Smqueue ist für den Versand bzw. die Speicherung von SMS Nachrichten zuständig. Darüber hinaus verfügt es über „Short Code"-Funktionalität, die es erlaubt, den Inhalt von Textnachrichten als Eingabeargumente für selbst entwickelte lokale Anwendungen zu benutzen oder an eine E-Mail-Adresse weiterzuleiten. Smqueue hat bereits eine solche Short-Code-Anwendung namens „register" integriert. Hierbei handelt es sich um eine interaktive Registrierung, bei der der Benutzer eine SMS mit der gewünschten Rufnummer als Inhalt an die BTS sendet. Wird diese Nummer noch nicht verwendet, trägt „register" diese zusammen mit der IMSI in die Subscriber Registry ein.

(Die Komponenten Smqueue und Subscriber Registry (sipauthserve) sind keine eigenständigen Projekte, sondern Bestandteile von OpenBTS.)

In Abbildung 5 sind die Beziehungen, sowie die Verbindungsprotokolle der einzelnen Komponenten untereinander ersichtlich.

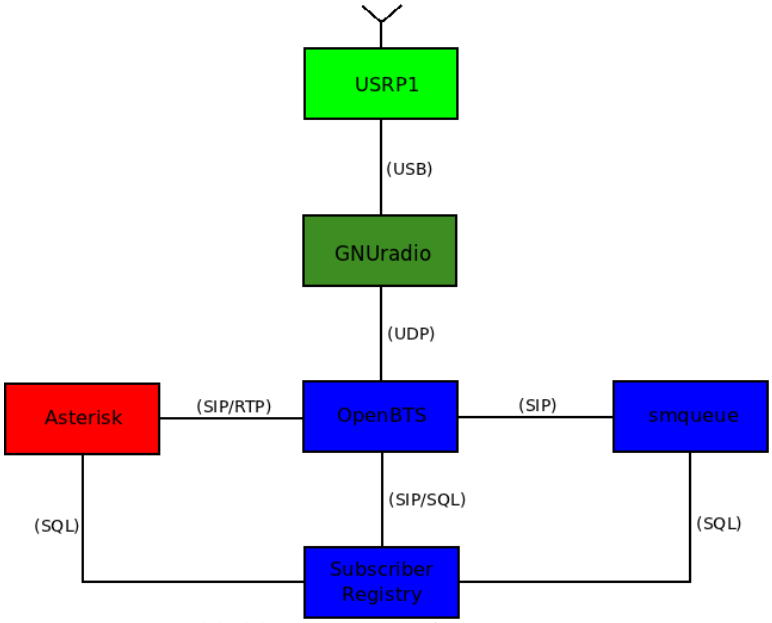

Abbildung 5: Systemkomponenten

In Abbildung 6 ist noch einmal der Kommunikationsfluss im Hinblick auf SIP aufgezeichnet. Dabei kennzeichnen die **schwarzen** Pfeile SIP-Verbindungen, die **roten** Pfeile Datenbankabfragen und der **blaue** Pfeil eine ODBC-Verbindung.

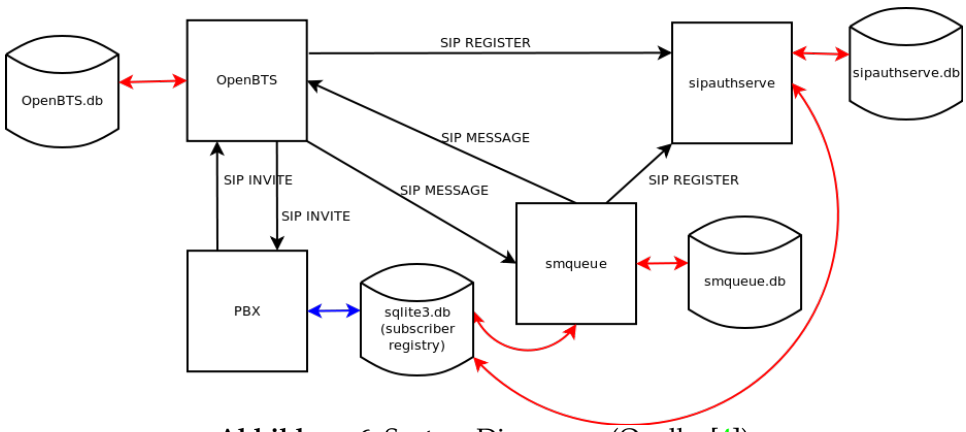

Abbildung 6: System Diagramm (Quelle: [4])

3.1.2 Datenbanken

Im System Diagramm findet man vier Datenbanken vor, welche für folgende Zwecke benutzt werden:

OpenBTS.db	Alle Konfigurationsparameter von OpenBTS werden seit der Version P2.8 nicht mehr in einzelnen Konfigurationsdateien hinterlegt, sondern in einer zentralen SQL-Datenbank verwaltet.
sipauthserve.db	Hier werden die angemeldeten MS-Teilnehmer registriert.
sqlite3.db	Von Asterisk benutzte Datenbank zur SIP User Registrierung, dessen Einträge von *sipauthserve* erzeugt werden.
smqueue.db	Enthält alle Konfigurationsparameter von *smqueue*.

3.1.3 GSM/SIP-Abläufe

Einer der Hauptmerkmale von OpenBTS ist es, das Mobile Switching Center (MSC) durch einen herkömmlichen VoIP-Switch (Asterisk) zu ersetzen. Dabei ist jede MS aus Sicht des Asterisk-Servers ein SIP-Endpunkt. Dieser SIP-Endpunkt wird von OpenBTS realisiert und verwaltet. Dabei wird jeder MS ein eigener SIP-Benutzername in der Form „IMSIxxxxxxxxxxxxxx“ zugeordnet, wobei die „x“ der IMSI-Nummer der MS entsprechen. Die IP-Adresse jedes SIP-Endpunktes ist immer die gleiche und zwar die der BTS, also dort, wo der OpenBTS-Dienst läuft. OpenBTS ist gegenüber Asterisk transparent, d.h. Asterisk sieht nur die MS als SIP-Endpunkte. Eine aktive Sprachverbindung besteht in diesem Kontext somit aus zwei Teilstrecken. Einmal die Strecke Asterisk $\longleftrightarrow$ OpenBTS, in der SIP/RTP-Pakete die Sprachübertragung erledigen, und zum anderen die Strecke OpenBTS $\longleftrightarrow$ MS, bei der ein GSM Sprach- bzw. Verkehrskanal (*TCH*) auf der Luftschnittstelle existiert. Die SIP-Verbindung terminiert also bei OpenBTS.

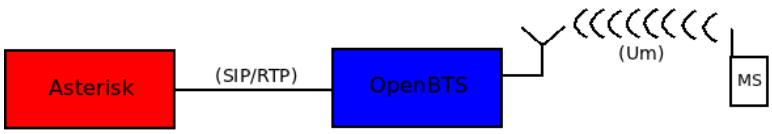

Abbildung 7: Terminierung der SIP-Verbindung an OpenBTS

Die bereits oben erwähnte Komponente *sipauthserve* ist dabei für die Registrierung der MS zuständig und trägt dazu die Teilnehmer in die Datenbank `sqlite3.db` ein.
Nachfolgend werden drei wesentliche GSM-Szenarien beschrieben, um das

Zusammenspiel aus GSM- und SIP, wie in Abbildung 6 gezeigt, zu verdeutlichen:

- **Registrierung (Location Update)**
  Wenn die MS eingeschaltet wird oder eine neue Location Area betritt, führt sie einen *LOCATION UPDATE REQUEST* (LUR) aus. Zudem ist es möglich, dass die BTS die MS periodisch dazu auffordert ein LUR auszuführen. Bei OpenBTS wird ein GSM LUR in Form eines *SIP REGISTER* durchgeführt (siehe Abb. 8). Zuerst findet der allgemeine GSM Ablauf der Kanalanforderung (ausgehend von der MS) statt. Nun folgt das eigentliche Location Update. Dazu schickt die MS einen *LOCATION UPDATE REQUEST* an OpenBTS, welche daraufhin ein *SIP REGISTER* an *sipauthserve* sendet. *sipauthserve* erstellt nun einen entsprechenden Eintrag in der SQL-DB `sqlite3.db` (siehe Abb. 6). Zum Schluss wird der zugewiesene GSM Kanal wieder abgebaut. Von nun an ist die MS im Netz registriert und kann durch einen *PAGING REQUEST* „angesprochen“ werden.

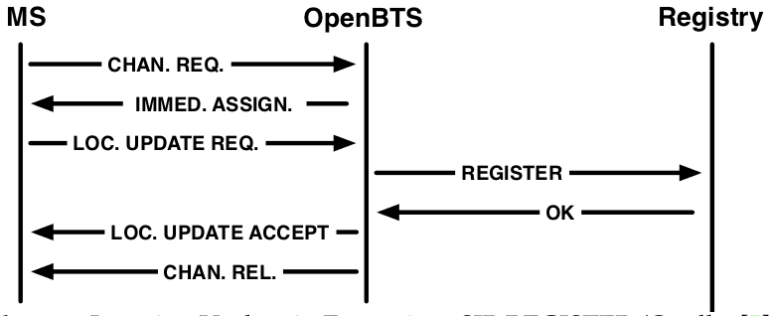

Abbildung 8: Location Update in Form eines SIP REGISTER (Quelle: [5](S.47))

- **Gesprächsaufbau (MS → SIP-Switch)**
  Der Gesprächsaufbau, ausgehend von der Mobile Station, ist in Abbildung 9 beschrieben. Die MS beantragt als erstes einen Kanal bei der BTS (*CHANNEL REQUEST* auf *RACH*). Nun weist die BTS der MS einen freien Kanal zu (*IMMEDIATE ASSIGNMENT*), woraufhin dann diese eine Anfrage zum Verbindungsaufbau (*CM SERVICE REQUEST*) an die BTS sendet. Akzeptiert die BTS die Anforderung, so folgt nun der eigentliche Aufbau des Gesprächs (*SETUP*). OpenBTS sendet dazu einerseits einen *SIP INVITE* an die PBX um eine SIP Session aufzubauen und andererseits ein *CALL PROCEEDING* zur Signalisierung an die MS. Wenn die SIP-Gegenstelle erreichbar ist (*Status: 200 OK*) bekommt die MS dies als Läutezeichen (*ALERTING*) mitgeteilt. Nimmt zudem die Gegenstelle das Gespräch an, so ist der Verbindungsaufbau komplett. Zwischen OpenBTS und PBX besteht nun eine RTP-Verbindung über die

die Sprachpakete transportiert werden und zwischen OpenBTS und MS besteht eine herkömmliche Sprachverbindung (*TCH*) auf der GSM Luftschnittstelle.

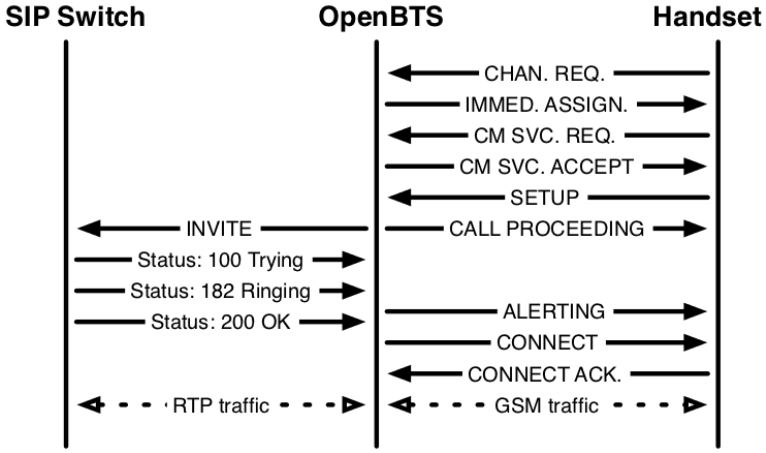

Abbildung 9: Gespräch ausgehend von der Mobile Station (Quelle: [5](S.48))

- **Gesprächsaufbau (SIP-Switch → MS)**
  Zur Vollständigkeit sei in Abbildung 10 der Gesprächsaufbau ausgehend vom SIP-Switch erwähnt. Diesmal geht der *SIP INVITE* von der PBX aus. OpenBTS versucht nun die MS zu erreichen (*PAGING REQUEST*). Bei Erfolg fordert die MS wiederum einen Kanal bei der BTS an (siehe Punkt vorher), das GSM Gespräch wird aufgebaut (*SETUP*) und falls der MS-Benutzer nun endgültig das Gespräch annimmt, ist der Verbindungsaufbau komplett. Nun besteht wieder eine RTP-Verbindung zwischen PBX und OpenBTS und eine Sprachverbindung (*TCH*) auf der GSM Luftschnittstelle zwischen OpenBTS und MS.

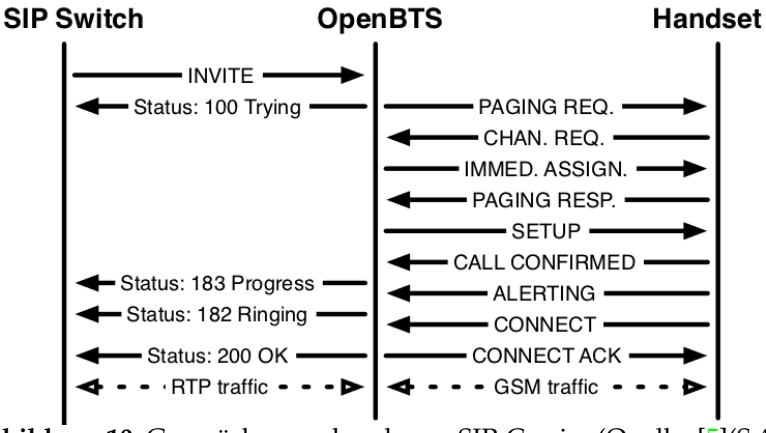

Abbildung 10: Gespräch ausgehend vom SIP-Carrier (Quelle: [5](S.48))

3.2 Installation

Die Installation von **GNUradio, OpenBTS samt smqueue und sipauthserve**, sowie **Asterisk**, erfolgte auf einem Ubuntu Linux 10.04.2 LTS. Dabei wurden die folgenden Schritte bei der Installation des jeweiligen Softwarepakets durchgeführt.

Paket	Version
GNUradio	3.4.2
OpenBTS	P2.8 (SVN)
smqueue	P2.8 (SVN)
sipauthserve	P2.8 (SVN)
Asterisk	1.6.2.9 (Ubuntu Repository)

Auf Paketabhängigkeiten wurde größtenteils Rücksicht genommen. Sollte es trotzdem zu ungelösten Abhängigkeiten kommen, so können fehlende Pakete bspw. mittels `apt-get` *aus entsprechenden Distributions-Repositories nachinstalliert werden.*

3.2.1 GNUradio

GNUradio wurde mit USRP1-Unterstützung kompiliert und installiert.

1. Fehlende Abhängigkeiten installieren:

```
sudo apt-get -y install git-core autoconf automake \
libtool g++ python-dev swig pkg-config \
libboost-all-dev libfftw3-dev libcppunit-dev \
libgsl0-dev libusb-dev sdcc libsdl1.2-dev \
python-wxgtk2.8 python-numpy python-cheetah \
python-lxml doxygen python-qt4 python-qwt5-qt4 \
libxi-dev libqt4-opengl-dev libqwt5-qt4-dev \
libfontconfig1-dev libxrender-dev
```

2. GNUradio mit USRP1-Unterstützung kompilieren und installieren:

```
wget http://gnuradio.org/redmine/attachments/\
download/279/gnuradio-3.4.2.tar.gz
tar xzf gnuradio-3.4.2.tar.gz
cd gnuradio-3.4.2/
./configure -with-usrp1
make
make check
sudo make install
```

3. Cache des Runtime Linkers aktualisieren:

```
export LD_LIBRARY_PATH=/usr/local/lib
sudo ldconfig
```

3.2.2 OpenBTS

Für die Installation von **OpenBTS, smqueue und sipauthserve** wurde die aktuellste Version aus dem SVN-Repository verwendet.

1. Fehlende Abhängigkeiten installieren:

```
sudo apt-get install autoconf libtool libosip2-dev \
libortp-dev libusb-1.0-0-dev g++ sqlite3 \
libsqlite3-dev erlang
```

2. Sourcecode aus SVN-Repository kopieren:

```
mkdir openbts
cd openbts
svn co http://wush.net/svn/range/software/public
```

3. In OpenBTS-Source-Verzeichnis wechseln und OpenBTS mit USRP1-Unterstützung kompilieren:

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

4. Da wir die USRP1 mit einem 52MHz Takt betreiben, muss ein Link zur entsprechenden Transceiver-Binary erstellt, sowie die passende „inband“-Tabelle kopiert werden:

```
(from OpenBTS root)
cd apps/
ln -s ../Transceiver52M/transceiver .
sudo mkdir -p /usr/local/share/usrp/rev4/
sudo cp ../Transceiver52M/std_inband.rbf \
/usr/local/share/usrp/rev4/
cd ..
```

5. Seit Version 2.8 wird die Konfiguration von OpenBTS nun nicht mehr in einer einzelnen großen Konfigurationsdatei gespeichert, sondern in einer SQL-Datenbank verwaltet. Diese erzeugen wir mit Hilfe des bereitgestellten Templates :

```
(from the OpenBTS directory)
sudo mkdir /etc/OpenBTS
sudo sqlite3 -init ./apps/OpenBTS.example.sql \
/etc/OpenBTS/OpenBTS.db
( .exit zum verlassen von sqlite3)
```

3.2.3 Subscriber Registry und Sipauthserve

1. Zuerst sollte **Asterisk** installiert werden, damit die entsprechenden Verzeichnisse existieren:

   ```
   sudo apt-get install asterisk
   ```

2. Im SVN-Repository befindet sich ein SQL-Skript das nun für die Erstellung der Subscriber Registry benutzt wird:

   ```
   (from svn root)
   cd public/subscriberRegistry/trunk/configFiles/
   sudo mkdir /var/lib/asterisk/sqlite3dir
   sudo sqlite3 -init subscriberRegistryInit.sql \
   /var/lib/asterisk/sqlite3dir/sqlite3.db
   ( .exit zum verlassen von sqlite3)
   ```

3. Nun wird **sipauthserve** kompiliert und dessen Konfigurationsdatenbank in `/etc/OpenBTS/` erzeugt:

   ```
   (from svn root)
   cd subscriberRegistry/trunk
   make
   sudo sqlite3 -init sipauthserve.example.sql \
   /etc/OpenBTS/sipauthserve.db
   ( .exit zum verlassen von sqlite3)
   ```

3.2.4 Smqueue

1. **Smqueue** kompilieren:

   ```
   (from svn root)
   cd smqueue/trunk/
   autoreconf -i
   ./configure
   make
   ```

2. Konfigurationsdatenbank von smqueue in `/etc/OpenBTS/` erzeugen:

   ```
   sudo sqlite3 -init smqueue/smqueue.example.sql \
   /etc/OpenBTS/smqueue.db
   ( .exit zum verlassen von sqlite3)
   ```

3.3 Konfiguration

3.3.1 OpenBTS

Die Datenbank `/etc/OpenBTS/OpenBTS.db` enthält sämtliche Konfigurationsparameter für OpenBTS. Eine Liste aller Parameter, sowie dessen Beschreibung, findet man unter https://wush.net/trac/rangepublic/wiki/openBTSConfig. Die Parameter können entweder direkt in der Datenbank `OpenBTS.db` (z.B. mittels `sqlite3 /etc/OpenBTS/OpenBTS.db`) oder am OpenBTS-Prompt mit den Befehlen `config` bzw. `unconfig` geändert werden.

Für die meisten Parameter eignen sich die bereits eingetragenen Standardwerte, jedoch sind einige grundlegende Einstellungen vorzunehmen:

- **GSM.Radio.Band**
  Bestimmt das benutzte GSM-Band, bei uns: **1800**
- **GSM.Radio.C0**
  Die ARFCN Nummer, bei uns: **867**
- **GSM.CellSelection.Neighbors**
  ARFCN der Nachbarzellen, bei uns: **846**
- **Control.LUR.OpenRegistration**
  Mittels eines regulären Ausdrucks werden die IMSIs definiert, die sich an der BTS registrieren dürfen. Für Testzwecke ist es jedoch sinnvoll eine offene Registrierung zu verwenden, d.h. es darf sich jede MS verbinden. Dazu ist der Wert auf **Null** zu setzen.
- **GSM.Identity.MCC**
  Der Mobile Country Code bestimmt die Länderkennung, bei uns: **262** für Deutschland.
- **GSM.Identity.MNC**
  Der Mobile Network Code ist eine zweistellige Nummer, die den Mobilfunkanbieter kennzeichnet. In Deutschland besitzt T-Mobile beispielsweise die 01, E-Plus die 03 und O2 die 07. Wir haben den Wert **99** eingestellt, da sich dieser offiziell nicht in Gebrauch befindet.
- **GSM.Identity.ShortName**
  Kurze Bezeichnung des Netzwerks; wird bei manchen Mobilfunktelefonen im Display angezeigt (überwiegend neuere Modelle); bei uns **OpenBTS HM**
- **GSM.RACH.AC**
  Die Access Class Flags sollten bei einem nicht funktionstüchtigen Netz auf `0x0400` gesetzt werden, um dem Benutzer mitzuteilen, dass keine Notrufunterstützung existiert.

3.3.2 Sipauthserve

In der Regel müssen keine weiteren Einstellungen vorgenommen werden, solange man den Standardpfad für die `sqlite3.db` eingehalten hat. Hat man dies nicht, so ist der neue Dateipfad (Feld: `SubscriberRegistry.db`) in der Konfigurationsdatenbank (`/etc/OpenBTS/sipauthserve.db`) anzugeben. Hilfreich könnte auch das Hochsetzen des `Log.Level` sein, welches standardmäßig auf *WARNING* eingestellt ist und unter *DEBUG* deutlich mehr Hinweise bzgl. der erstellten Registry-Einträge offenbart.

3.3.3 Smqueue

Auch an der Konfiguration von *smqueue* muss zwangsläufig keine Änderung vorgenommen werden. Allerdings enthält die Konfigurationsdatenbank (`/etc/OpenBTS/smqueue.db`) weitaus mehr Einträge als die von sipauthserve. Gut die Hälfte dieser Parameter beziehen sich auf die „Short Code“-Funktionalität.

3.3.4 Asterisk

Die Konfiguration von Asterisk beschränkt sich in diesem Dokument auf die Intra-BTS-Kommunikation, d.h. es können nur Gespräche zwischen MS und MS bzw. MS und Asterisk-Diensten (Echo-Test, VoiceMail) stattfinden, nicht jedoch von oder nach außerhalb (Festnetz, andere SIP-Teilnehmer) telefoniert werden.

- **Teilnehmereintrag**
  Damit die MS untereinander telefonieren können, müssen die Asterisk-Konfigurationsdateien `sip.conf` und `extensions.conf` im Verzeichnis `/etc/asterisk/` angepasst werden. Als Beispiel wird die **IMSI 001010000000000** verwendet und dieser die Teilnehmerrufnummer **2101** zugeordnet.

In `sip.conf` muss folgender Eintrag hinzugefügt werden:

```
...
[IMSI001010000000000]
callerid=2101          ; Teilnehmerrufnummer (diese
                       ; sieht auch die Gegenstelle)
canreinvite=no         ; Asterisk ist Mittelsmann
                       ; zwischen MS und Gegenstelle
type=friend            ; MS ruft uns bzw. wir rufen MS an
context=sip-external   ; zugeordneter Kontext
allow=gsm              ; Sprachcodec 'gsm' erlauben
host=dynamic           ; dynamischer Hostname
dtmfmode=rfc2833       ; DTMF-Töne nach Standard RFC2833
...
```

In `extensions.conf` muss je IMSI ein Eintrag in der ihr zugeordneten Extension (hier `sip-external`) hinzugefügt werden:

```
...
[sip-external]
exten => 2101,1,Dial(SIP/IMSI001010000000000@127.0.0.1:5062)
...
```

- **Echo-Test**
  Zu Testzwecken empfiehlt es sich einen sog. „Echo-Test“ unter Asterisk zu konfigurieren. Bei einem Echo-Test wird alles was man sagt als Echo zurückgeschickt. Zum einen erkennt man dadurch, welche Latenzzeit zwischen MS und Asterisk besteht und zum anderen lässt sich ein Sprachkanal ohne die Notwendigkeit einer zweiten MS einfach aufbauen. Um den Echo-Test zu aktivieren, werden folgende Zeilen in die `extensions.conf` innerhalb des Kontext `[sip-external]` eingefügt:

```
[sip-external]
...
exten => 600,1,Answer()
exten => 600,2,Playback(demo-echotest)
exten => 600,3,Echo()
exten => 600,4,Playback(demo-echodone)
exten => 600,5,Hangup()
...
```

Nun ist der Echo-Test unter der Rufnummer 600 zu erreichen.

In der Praxis zeigte sich, dass die Sprachverzögerung um die 0.5 Sekunden liegt, was mit hoher Wahrscheinlichkeit an OpenBTS liegt. Ein zweiter Test zwischen Asterisk und einem herkömmlichen SIP-Client (Software), über eine direkte VoIP-Verbindung, wies nämlich keine nennenswerte Latenz auf.

- **VoiceMail**
  Jedem eingetragenen Teilnehmer kann eine Mailbox zugeordnet werden, welche aktiv wird, wenn der Teilnehmer entweder nicht erreichbar ist (im Netz nicht registriert) oder nach einer einstellbaren Zeit den Anruf nicht entgegennimmt. Für die Einrichtung einer Mailbox sind die folgenden Schritte notwendig:

  1. Neuen Kontext (im Bsp. `[mailbox]` genannt) und Benutzer in `/etc/asterisk/voicemail.conf` hinzufügen:

```
...
[mailbox]
2101 => 1234, 2101 ; #Rufnummer => #Passwort, #Benutzer
;weitere VoiceMail-Benutzer
...
```

  2. Regelwerk in `/etc/asterisk/extensions.conf` anpassen und VoiceMail-Abfrage hinzufügen:

```
...
[sip-external]
exten => 2101,1,Dial(SIP/IMSI001010000000000
         @127.0.0.1:5062,20)
exten => 2101,2,VoiceMail(2101@mailbox)
exten => 2101,3,PlayBack(vm-goodbye)
exten => 2101,4,HangUp()

exten => 8888,1,VoiceMailMain(s${CALLERID(num)}@mailbox)
; Abfrage ohne Auth.
exten => 9999,1,VoiceMailMain(@mailbox) ; Abfrage mit Auth.
...
```

     Das obige Regelwerk des Benutzers mit der Teilnehmerrufnummer 2101 besagt Folgendes:
     Zuerst wird versucht den Benutzer **IMSI001010000000000** über SIP zu erreichen (siehe Kapitel 3.1.3). Falls er im Netz registriert ist

und die MS über *PAGING* erreichbar ist, wird man nun maximal 20 Sekunden lang anläuten und nach Ablauf dieser Zeit zu Regel 2 springen. Ist der Benutzer erst gar nicht im Netz registriert, so wird unmittelbar zu Regel 2 gesprungen. Regel 2 definiert den eigentlichen Mailbox-Befehl. Der Anrufende hört die Mailboxansage und hat die Möglichkeit eine Nachricht zu hinterlassen. Hat er dies getan, so kann er entweder direkt auflegen oder mittels der Rautetaste die Aufnahme beenden. Regel 3 spielt daraufhin ein „Auf Wiedersehen"-Soundfile ab und Regel 4 beendet letztendlich den Anruf.

Zur Abfrage seiner Mailbox hat man zwei Möglichkeiten. Ruft man, in unserem Beispiel die Nummer **8888** an, so gelangt man direkt, d.h. ohne interaktive Authentifizierung, zu seinem Mailbox-Menü. Dort kann man aufgenommene Nachrichten abspielen, verwalten, löschen und weitere Mailbox-Optionen treffen. Unter der **9999** muss man sich erst mit seiner Mailbox-Kennung (hier im Beispiel ist das die 2101) und seinem Passwort (1234) gegenüber der Mailbox-Abfrage authentifizieren.

In der Praxis kam es zu Problemen bei der interaktiven Mailbox-Abfrage. Die Kennung und/oder das Passwort wurden von Asterisk als falsch betrachtet und eine erfolgreiche Authentifizierung war nicht möglich. Der Grund hierfür dürfte an einem nicht unterstützten bzw. nicht konfigurierten DTMF-Modus liegen, mittels dem die Tasteneingabe (Tastentöne) übertragen werden. Aus Zeitgründen und weil die Mailbox-Abfrage für unser Projektziel nicht relevant war, wurde der Sache nicht weiter auf den Grund gegangen.

3.4 Benutzung von OpenBTS

3.4.1 Start der Dienste

Damit die Registrierung der Teilnehmer und der Versand von Textnachrichten möglich ist, müssen neben OpenBTS auch die beiden Dienste `sipauthserve` sowie `smqueue` gestartet werden.

```
(from svn root)
./smqueue/trunk/smqueue/smqueue &
./subscriberRegistry/trunk/sipauthserve &
./openbts/trunk/apps/OpenBTS
```

3.4.2 Command Line Interface (CLI)

Konnte OpenBTS erfolgreich gestartet werden, sieht man nun das Command Line Interface (CLI) vor sich:

```
OpenBTS>
```

Nachfolgend einige Beispiele von hilfreichen CLI-Befehlen:

- **Befehl:** `calls`
  Listet aktive Gesprächs- bzw. SMS-Aktivitäten auf.

```
OpenBTS> calls
2060207953 C0T1 TCH/F IMSI=001010000000000 L3TI=8
SIP-call-id=1811340387 SIP-proxy=127.0.0.1:5060 MOC
to=600 GSMState=active SIPState=Active (5 sec)
```

  Dabei ist `2060207953` die Transaktions-ID, `C0T1` die C0-ARFCN und der dabei verwendete Zeitschlitz (*Nr. 1*), `TCH/F` die Kanalart (*Full-Rate Traffic Channel*) und `IMSI=001010000000000` die IMSI des Teilnehmers. Die restlichen Angaben beziehen sich auf die SIP-Verbindung (*ID, SIP-Status und Verbindungsdauer*).

- **Befehl:** `chans`
  Listet aktive Kanäle und die dazugehörigen Leistungswerte auf.

```
OpenBTS> chans
CN TN chan     transaction UPFER RSSI TXPWR TXTA DNLEV DNBER
CN TN type     id          pct    dB   dBm  sym   dBm   pct
 0 1    TCH/F  1247828231  0.54  -53    30    1   -48  0.00
```

  Im obigen Beispiel handelt es sich um einen Verkehrskanal im Full-Rate Modus (`TCH/F`) der den ersten Zeitschlitz (`TN=1`) belegt und der Transaktions-ID `1247828231` zugeordnet ist. Die MS sendet mit `30 dBm` (`TXPWR`) und besitzt einen Timing Advance (`TXTA`) Wert von `1`. Die beiden Angaben `UPFER` und `RSSI` beziehen sich auf den Uplink. `UPFER` gibt dabei die Fehlerrate der Frames an, die im Beispiel bei `0,54` Prozent liegt, und `RSSI` die Stärke des Sendesignals der MS, hier `-53 dBm`. Für den Downlink gibt `DNBER` die Fehlerrate der Bits an und `DNLEV`, das Pendant zu RSSI im Uplink, gibt die Empfangssignalstärke der MS, hier `-48 dBm`, an.

- **Befehl:** `tmsis`
  Listet die aktuelle TMSI-Tabelle auf.

```
OpenBTS> tmsis
TMSI       IMSI             age   used
1          001010000000000  138s  138s
```

  Im Beispiel wurde der IMSI `001010000000000` die TMSI `1` zugeordnet. Dieser Eintrag wurde vor 138s (`age`) erstellt. Ebenfalls 138s ist diese TMSI in Gebrauch (`used`).

Eine Liste aller möglichen OpenBTS-Befehle sowie dessen Beschreibung befindet sich unter https://wush.net/trac/rangepublic/wiki/cli oder im Benutzerhandbuch von OpenBTS [5](Kapitel 5.5).

3.4.3 Registrierung einer MS an OpenBTS

Mit Hilfe der bereitgestellten Mobiltelefone vom Typ „Nokia 3330“ und der programmierbaren SIM-Karten von Giesecke & Devrient konnte nun eine Registrierung in unserem GSM Testnetz namens „OpenBTS HM“ stattfinden. Dank der offenen Registrierung (`Control.LUR.OpenRegistration`) kann man sich einfach am Netz registrieren und erhält zudem eine SMS, dessen Inhalt über den Konfigurationsparameter `Control.LUR.OpenRegistration.Message` bestimmbar ist. Wichtiger Bestandteil dieser Kurzmitteilung ist die IMSI der SIM-Karte, mit der sich die MS an der BTS registriert hat. Nun kann ein Teilnehmereintrag in Asterisk, wie in Kapitel 3.3.4 beschrieben, erfolgen. Folgendes Beispiel zeigt den SMS-Inhalt für die IMSI 262071111111111:

```
Welcome to the GSM test network. Your IMSI is
IMSI:262071111111111
```

4 OpenBTS Software-Architektur

Von: Stefan Giggenbach

4.1 Überblick

OpenBTS wurde vollständig mit der objektorientierten Sprache C++ implementiert. Dabei wurde ein modularer Aufbau verwendet, der die wichtigsten Bestandteile des Systems zusammenfasst. Folgende Auflistung zeigt die einzelnen Module von OpenBTS und entspricht dabei der Ordnerstrutkur im Verzeichnis `<openbts>/src/openbts/trunk`.

- *apps* - OpenBTS Applikation (main-File)
- *CLI* - Command Line Interface
- *CommonLibs* - Standardfunktionen wie BitVector, Sockets, Thread, etc.
- *Control* - Funktionen für GSM Call Control, Mobility Management und SIP
- *Globals* - Deklaration der globalen Variablen
- *GSM* - GSM Stack
- *SIP* - SIP State Machine die vom Control Modul verwendet wird
- *SMS* - SMS Stack
- *sqlite3* - SQLite3 Zugriffsfunktionen
- *SR* - Subscriber Registry
- *Transceiver* - Software Transceiver mit Ünterstützung des USRP1
- *TRXManager* - Schnittstelle zwischen GSM Stack und Software Transceiver

Das *apps*-Modul stellt die eigentliche OpenBTS Applikation dar. Neben dem ausführbaren Binary befindet sich in diesem Verzeichnis mit der Datei `OpenBTS.cpp` das main-File des Projekts. In dieser Datei werden alle verwendeten globalen Objekte instanziiert, die Konfiguration der benötigten Ressourcen durchgeführt und die am Programmablauf beteiligten Threads gestartet. Neben dem *apss*-Modul werden sowohl das *CLI*- als auch das *GSM*-Modul, für die Erweiterung der Software um die Handover-Funktionalität, modifiziert. Die restlichen Module und deren Funktion wurden nur aus Gründen der Vollständigkeit aufgeführt.

Im folgenden Abschnitt werden zwei Klassen des GSM-Moduls, die bei der Implementierung der Handover-Funktionalität eine wichtige Rolle spielen, genauer betrachtet.

4.2 Verwendete Klassen

5 Erweiterung von OpenBTS

5.1 Measurement Report

Von: Max Eschenbacher

Measurement Reports enthalten Messwerte bzgl. der Empfangsleistung, Empfangsqualität, sowie Informationen zu Nachbarzellen. Sie werden

beim Einbuchen in das Netzwerk und während eines Gesprächs (ca. 2 mal in der Sekunde) von der MS an die BTS gesandt. Measurement Reports sind im RR-Sublayer (*Radio Resource*) angesiedelt und mit dem Nachrichtentyp *MEASUREMENT REPORT* gekennzeichnet. Die Messwerte sind für das Weiterreichen (Handover) der MS von großer Bedeutung.

OpenBTS verwaltet diese Messwerte intern in einer eigenen Klasse, bietet aber auch die Möglichkeit, diese in eine externe SQL-Datenbank abzulegen. Mit der OpenBTS-Option `Control.Reporting.PhysStatusTable` kann der Pfad der Datenbank angegeben werden:

```
OpenBTS> config Control.Reporting.PhysStatusTable \
/etc/OpenBTS/phystatus.db
```

Leider werden keinerlei Informationen bzgl. der Nachbarzellen in die Datenbank eingetragen. Deshalb musste die Tabelle `PHYSTATUS` in der Datenbank um zusätzliche Felder für die Nachbarzellen erweitert und die Methode `PhysicalStatus::setPhysical()` in der C++-Quelldatei `<OpenBTS-DIR>/GSM/PhysicalStatus.cpp` angepasst werden. Zusätzlich wurde ein neuer CLI-Befehl namens `showmr` implementiert, welcher den Inhalt der „Measurement Report Datenbank“ entsprechend formatiert und im OpenBTS-Terminal auflistet. Dazu wurde eine bereits bestehende, aber auskommentierte Methode (`PhysicalStatus::dump()` ebenfalls in `<OpenBTS-DIR>/GSM/PhysicalStatus.cpp`), von OpenBTS an die neuen Bedürfnisse (erweitertes Tabellenlayout) angepasst und der eigentliche CLI-Befehl in der Datei `<OpenBTS-DIR>/CLI/CLI.cpp` hinzugefügt.

Es sei erwähnt, dass das Handover-Modul (siehe Kapitel 5.2) auf die Measurement-Daten in der SQL-Tabelle aus Performancegründen komplett verzichtet und sich die benötigten Messwerte direkt über den Aufruf der entsprechenden Getter-Methoden des Measurment-Objekts besorgt. Somit dient die SQL-Tabelle rein dem CLI-Befehl `showmr`, damit dieser nicht nur den aktuell vorliegenden Messbericht anzeigt, sondern auch Auskunft über zurückliegende Reports geben kann.

Nachfolgend eine Beispielausgabe von `showmr`:

```
OpenBTS> showmr
############################################################
                Measurement Report:
############################################################
CN_TN_TYPE_OFFSET                =       C0T0 SDCCH/4-0
ARFCN                            =       867
ACCESSED                         =       1330702677
RSSI                             =       -63.750000
TIME_ERR                         =       -0.222656
TIME_ADVC                        =       1
TRANS_PWR                        =       30 dBm
FER                              =       0.000000
RXLEV_FULL_SERVING_CELL          =       -48 dBm
RXLEV_SUB_SERVING_CELL           =       -48 dBm
RXQUAL_FULL_SERVING_CELL_BER     =       0.181000 dBm
RXQUAL_SUB_SERVING_CELL_BER      =       0.181000 dBm
NO_NCELL                         =       1
RXLEV_CELL_1 = 0, BCCH_FREQ_CELL_1 = 0, BSIC_CELL_1 = 0
RXLEV_CELL_2 = 0, BCCH_FREQ_CELL_2 = 0, BSIC_CELL_2 = 0
RXLEV_CELL_3 = 0, BCCH_FREQ_CELL_3 = 0, BSIC_CELL_3 = 0
RXLEV_CELL_4 = 0, BCCH_FREQ_CELL_4 = 0, BSIC_CELL_4 = 0
RXLEV_CELL_5 = 0, BCCH_FREQ_CELL_5 = 0, BSIC_CELL_5 = 0
RXLEV_CELL_6 = -33, BCCH_FREQ_CELL_6 = 63, BSIC_CELL_6 = 1
############################################################
CN_TN_TYPE_OFFSET                =       C0T1 TCH/F
ARFCN                            =       867
ACCESSED                         =       1330696371
RSSI                             =       -57.250000
TIME_ERR                         =       0.263672
TIME_ADVC                        =       1
TRANS_PWR                        =       30 dBm
FER                              =       0.042869
RXLEV_FULL_SERVING_CELL          =       -48 dBm
RXLEV_SUB_SERVING_CELL           =       -48 dBm
RXQUAL_FULL_SERVING_CELL_BER     =       0.000000 dBm
RXQUAL_SUB_SERVING_CELL_BER      =       0.000000 dBm
NO_NCELL                         =       7
RXLEV_CELL_1 = 0, BCCH_FREQ_CELL_1 = 0, BSIC_CELL_1 = 0
RXLEV_CELL_2 = 0, BCCH_FREQ_CELL_2 = 0, BSIC_CELL_2 = 0
RXLEV_CELL_3 = 0, BCCH_FREQ_CELL_3 = 0, BSIC_CELL_3 = 0
RXLEV_CELL_4 = 0, BCCH_FREQ_CELL_4 = 0, BSIC_CELL_4 = 0
RXLEV_CELL_5 = 0, BCCH_FREQ_CELL_5 = 0, BSIC_CELL_5 = 0
RXLEV_CELL_6 = 0, BCCH_FREQ_CELL_6 = 0, BSIC_CELL_6 = 0

############################################################
```

Erläuterungen zur Beispielausgabe:

Der erste Measurement Report wurde um `1330702677` (Unix-Time, entspricht 02.03.2012 - 16:37:57 Realzeit) im `SDCCH` (*Standalone Dedicated Control Channel*) mit der Nummer `0` (von 4 Möglichen) auf der `ARFCN 867` gesendet. Die empfangene Signalstärke (`RSSI` = *Received Signal Strength Indication*) betrug `-63.75 dBm`. Der zugeordnete *Timing Advance* Parameter der MS betrug `1` Symbolperiode und wies einen Fehler (`TIME_ERR`), d.h. einen Zeitversatz, von `-0.222656` Symbolperioden auf.

Die Sendeleistung der MS betrug `30 dBm` und hatte bis dato eine `Uplink-FER` (= *Frame Erasure Rate*; gibt das Verhältnis zwischen verworfenen (defekten) Frames und der Gesamtanzahl der Frames an) von `0`. Der Empfangspegel der verwendeten Zelle (`RXLEV_FULL_SERVING_CELL`) betrug `-48 dBm` und die Empfangsqualität `RXQUAL_FULL_SERVING_CELL_BER = 0.181000 dBm`. Die Angaben `SUB` und `FULL` bei der Empfangsleistung und -qualität beziehen sich auf die Verwendung von *DTX* (*Discontinuous Transmission*). `FULL` bezieht dabei alle Frames mit ein, also auch die zu dessen Zeitpunkt keine Sprache gesendet wurde. `SUB` bezieht hingegen nur die effektiven „Sprachframes“ mit ein. Da jeweils beide Werte im obigen Beispiel gleich sind, kann davon ausgegangen werden, dass kein DTX verwendet wurde.

Die restlichen Angaben beziehen sich auf die Nachbarzellen. `NO_NCELL` gibt die Anzahl der sichtbaren Nachbarzellen an. Dabei gibt es zwei Sonderfälle: `NO_NCELL=0` - es existieren keine Messwerte, `NO_NCELL=7` - es existieren keine Nachbarzellen. Im obigen Beispiel sieht die MS eine Nachbarzelle (Nr. 6) mit der ID `1` (*Base Station ID Code*) und einer Empfangsleistung von `-33 dBm`. Der *Broadcast Control Channel* (`BCCH`) liegt dabei auf Frequenz `63`.

Analog zum ersten Eintrag ist auch der zweite Eintrag im Measurement Report zu interpretieren. Dieser bezieht sich auf einen *Traffic Channel* im Zeitschlitz Nr. 1, bei dem die MS keine Informationen bzgl. der Nachbarzellen (`NO_NCELL=7`) besitzt.

5.2 Handover Modul
Von: Stefan Giggenbach

5.3 Inter OpenBTS Handover
Von: Thomas Waldecker

6 Analyse
Von: Thomas Waldecker

6.1 Vorbereitung

6.1.1 Tracen auf dem Um Interface mit einem Nokia 3310

Das Um Interface kann mit einem Nokia 3310 Mobiltelefon und einem dazugehörigen Adapter, der zwischen den Akku und das Telefon gesteckt wird und auf die vier Kontakte des internen Telefonbus zugreift getraced werden.

Im Adapterkabel ist ein USB-zu-Seriell Wandler integriert. In die Konfigurationsdatei muss deshalb das Device des USB-Seriell Wandlers eingetragen werden (Siehe Listing 2).

Listing 2: Konfigurationsdatei für gammu und dem verwendeten Adapter

```
1 [gammu]
2
3 port = /dev/ttyUSB0
4 model = 6110
5 connection = mbus
6 synchronizetime = yes
7 logfile =
8 logformat = nothing
9 use_locking = yes
10 gammuloc =
```

Das Tracen kann nach der weiteren Konfiguration, die in [6] beschrieben ist mit folgenden Kommando gestartet werden (Listing 3).

Listing 3: Aufruf von Gammu

```
1 sudo gammu --nokiadebug nhm5_587.txt v20-25,v18-19
2 Debug Trace Mode -- wumpus 2003
3 Loading
4 Activating ranges:
5   20-25 verbose=1
6   18-19 verbose=1
7 Debug Trace Enabled
8 Press Ctrl+C to interrupt...
9 <1805> MDI:m2d/FROM_MCU_TO_FBUS
10 t=0002 nr=0f: D 05: 1e 0c 00 40 00 06 01 01 70 01 01 47
11 <198E> MDI:d2m/FROM_FBUS_TO_MCU
12 t=0003 nr=10: D 8E: 1e 00 0c 7f 00 02 40 07
```

6.1.2 Patchen von Wireshark

6.2 Intra BSC Handover mit OpenBSC und zwei nanoBTS

6.2.1 Aufbau

In einem Raum (Mobile Netze Labor) wurden an zwei Ecken jeweils eine nanoBTS aufgestellt und mit dem Laborrechner verbunden.

A Anhang

A.1 Literaturverzeichnis

[1] Martin Sauter: *Grundkurs Mobile Kommunikationssysteme*, Vieweg+Teubner Verlag, Wiesbaden 2011

[2] Building OpenBSC: http://openbsc.osmocom.org/trac/wiki/Building_OpenBSC Abgerufen am 09.03.2012

[3] ipaccess-config (Konfiguration der nanoBTS): http://openbsc.osmocom.org/trac/wiki/ipaccess-config Abgerufen am 09.03.2012

[4] OpenBTS System Diagramm: https://wush.net/trac/rangepublic/attachment/wiki/BuildInstallRun/openbts_system_diagram.png, Abgerufen am 03.03.2012

[5] Range Networks Inc.: *OpenBTS P2.8 Users Manual Doc. Rev. 1*, Range Networks Inc. 2011

[6] AirProbe Wiki - tracelog: https://svn.berlin.ccc.de/projects/airprobe/wiki/tracelog, Abgerufen am 09.03.2012