

Mobile Netze

Man-In-The-Middle

Ausarbeitung
von
Attenberger, Bollenmiller, Schuster, Wilhelm
SS17 IG/GM
29. September 2017

Betreut von Prof. Dr. Wischhof

Inhaltsverzeichnis

1. Einleitung	2
2. Architektur des Osmocom Systems	3
2.1. Was ist OsmoNITB	3
2.2. Funktion der einzelnen Module	3
2.3. Funktionsweise von OsmoNITB	4
3. Architektur des OpenBTS Systems	6
3.1. Aufbau und Zusammenspiel	6
3.1.1. Bestandteile	6
3.1.2. Datenbanken	8
4. Inbetriebnahme eines Osmocom Systems	9
4.1. Vorinstallationen	9
4.1.1. Ubuntu 16.04.3	9
4.1.2. Git	10
4.1.3. Softwarevoraussetzungen	10
4.1.4. Aktivierung der Verbindung zum USRP2	10
4.2. Installation einzelner GSM Komponenten	11
4.2.1. OsmoTRX	11
4.2.2. OsmoBTS	11
4.2.3. OsmoNitb unter OpenBSC	13
4.3. Starten des Systems	13
4.4. Installation weiterer Komponenten	14
4.4.1. Osmo-sip-Connector	14
4.4.2. Asterisk	14
4.5. Verwendung einer GUI	16
5. Inbetriebnahme eines OpenBTS Systems	17
5.1. Vorinstallationen	17
5.1.1. Ubuntu 16.04.3	17
5.1.2. Git	17
5.1.3. Softwarevoraussetzungen	17
5.1.4. Aktivierung der Verbindung zum USRP2	17
5.2. Installation	17
5.3. Konfiguration	18
5.3.1. Datenbanken initialisieren	18
5.3.2. OpenBTS Einstellungen	19
5.3.3. Asterisk Einstellungen	20
5.4. Starten des Systems	20
5.5. Testen der Funktionen	21
6. Umsetzung des Projektziels	24
6.1. Abspeichern der Daten	24
6.2. Extrahieren und konvertieren der Daten	24
6.3. Automatisierung	25
6.4. Feature - Abhören der Aufnahme	26
6.5. Aufgetretene Probleme	28
7. Ergebnisse	29

8. Fazit	29
9. Projektaufteilung	29
A. osmo-bts.cfg	31
B. openbsc.cfg	32
C. Starten von OsmoBTS	34
D. Quellcode der GUI-Anwendung	35
E. Starten von OpenBTS	39
F. Installations- und Konfigurations-Skript	41
G. Skript zum Starten der Konvertierung	43

Abbildungsverzeichnis

1.	GSM Basisstation mit OsmoTRX & OsmoBTS	3
2.	GSM Systemarchitektur mit OsmoNITB	3
3.	OsmoNITB mit RTP proxy mode	5
4.	Unser Systemaufbau	5
5.	USRP N210 Software Defined Radio	7
6.	OpenBTS Systembestandteile	8
7.	OpenBTS System Diagramm	9
8.	Ansicht der Konsole nach dem Start des Transceivers	12
9.	Fehlermeldung bezüglich der Thread Priorität in osmoTRX	12
10.	Ansicht der Konsole nach dem Start der OsmoNitb	14
11.	Ansicht der Konsole nach dem Start des Osmo-sip-connectors	15
12.	Ansicht der Konsole nach Verbindung mit Asterisk	15
13.	GUI-Anwendung MitM	16
14.	Ansicht in SQLitebrowser zur Konfiguration der OpenBTS.db-Datenbank	19
15.	Ansicht der Konsole nach dem Start von SIPAuthServe	20
16.	Ansicht der Konsole nach dem Start von SMQueue	20
17.	Ansicht der Konsole nach dem Start von Asterisk	21
18.	Eingehende Nachricht nach erstmaliger Registrierung im Testnetz	21
19.	Tabelle DIALTABLE in sqlite3.db nach Registrierung mehrerer Endgeräte	22
20.	Tabelle SIP_BUDDIES in sqlite3.db nach Registrierung mehrerer Endgeräte	22
21.	Ausgabe in Asterisk nach Beenden eines Anrufs zwischen Mobilfunkteilnehmern	23
22.	Extrahierte SIP- & RTP-Pakete	25
23.	Zeile des incrontab	26
24.	Ansicht der Konsole nach dem Start der OsmoBTS	34
25.	Ansicht der Konsole nach dem Start von OpenBTS	40

1. Einleitung

Im Rahmen der Vorlesung Mobile Netze soll zunächst ein Mobilfunknetz in Betrieb genommen werden. Des Weiteren umfasst die Aufgabenstellung die Implementierung eines Features, welches vom Team in einer Projektvision definiert wird.

Als Projektziel des Teams J3A soll eine Man-In-The-Middle Funktionalität in einem GSM Netz integriert werden. Man-In-The-Middle bezeichnet eine Angriffsform, bei der ein Dritter den Datenverkehr zweier Gesprächspartner mitverfolgt. Dabei bleibt er unbemerkt und ist in der Lage alle Daten, die über die Kommunikationsverbindung gesendet werden, abzugreifen. Oft täuscht er den eigentlichen Gesprächspartner vor, sodass kein Verdacht geschöpft wird.

Der Fokus des Projekts ähnelt stark dem Prinzip des Man-In-The-Middle Angriffs. Das Ziel ist es ein Telefongespräch abzugreifen und abzuspeichern, der zwischen zwei Teilnehmern getätigt wird. Dabei sollen die abgespeicherten Daten in eine Form gebracht werden, die das Anhören lokal auf dem Rechner ermöglicht.

Die Umsetzung der Aufgabenstellung umfasst die Inbetriebnahme des GSM Netzes, das aus den Komponenten BTS, BSC und MSC besteht. Über Voice over IP soll der zweite Teilnehmer erreicht werden können. Alternativ zu Voice over IP war anfänglich auch die Verbindung zum zweiten Teilnehmer ebenfalls im GSM denkbar. Die minimale Anforderung besteht in der Manipulation der BTS/BSC dahingehend, dass diese die Daten abgreift und abspeichert. Das Hinterlegen einer Rufnummer, unter der der gespeicherte Anruf angehört werden kann, wurde als optionales Feature bezüglich des Projektzieles definiert.

2. Architektur des Osmocom Systems

2.1. Was ist OsmoNITB

Bei OsmoNITB handelt es sich um ein freies Softwarepaket aus dem Osmocom-Baseband-Projekt. Dessen Zweck ist das Aufspannen und Betreiben eines GSM/ GPRS-Netzes mittels nachgebauter Implementierungen des GSM-Protokolls. OsmoNITB bringt hierfür alle erforderlichen Bestandteile, die sich tiefer im Netz als die Basisstation (BTS) befinden.

Die Basisstation selbst wird mit OsmoBTS realisiert. Auch sie stammt vom OsmocomBB-Projekt. Die Signale hierfür werden mittels eines Software-Radiomodems (SDR) erzeugt und versendet. Wir verwendeten hierzu den USRP N210 von Ettus Research. Dieser ermöglicht es beliebige Signale bis zu einer Frequenz der Höhe 6GHz zu generieren. Um daraus überhaupt GSM-Signale zu Empfangen oder Senden zu können ist ein Transceiver notwendig. Dieses stammt ebenfalls vom OsmocomBB-Projekt und heißt OsmoTRX. Zwischen OsmoTRX und OsmoBTS werden für deren Kommunikation UDP-Nachrichten ausgetauscht. Die Anbindung dieser wiederum an OsmoNITB erfolgt durch das A-bis-Interface, einem GSM-spezifisches, standardisiertes Protokoll für den Austausch der Gespräche und Signalisierungsdaten. Es wird ebenso mittels einer freien Implementierung von OsmocomBB, namentlich "libosmo-abis"realisiert.

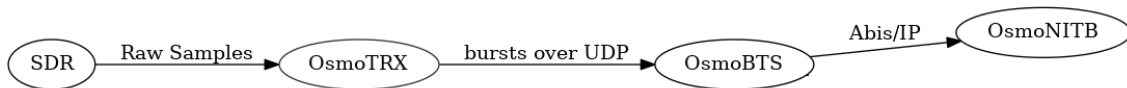


Abbildung 1: GSM Basisstation mit OsmoTRX & OsmoBTS

Der Basisstation-Controller (Base Station Controller, BSC) stellt hierbei die Gegenstelle dar und ist ebenso im OsmocomBB-Projekt enthalten.

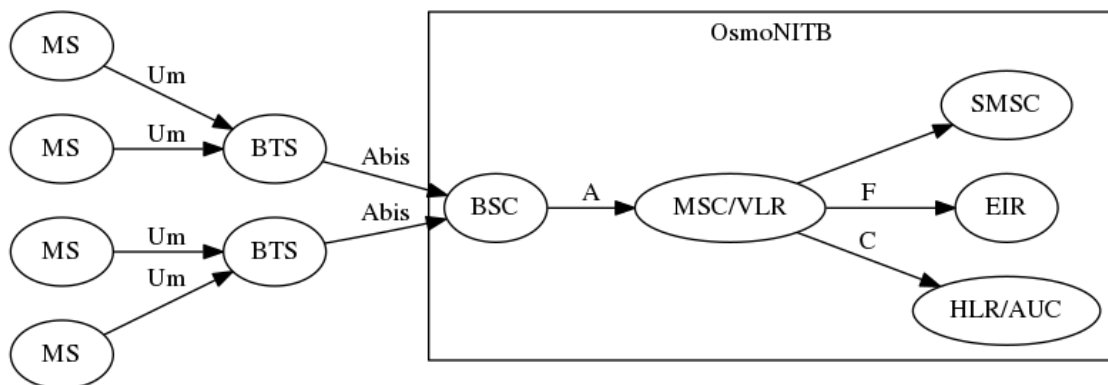


Abbildung 2: GSM Systemarchitektur mit OsmoNITB

Weiterhin existieren noch folgende weitere Bestandteile, wie MSC/VLR, SMSC, EIR und HLR/AUC, die im nächsten Kapitel erklärt werden.

2.2. Funktion der einzelnen Module

- **BSC**

Hierbei handelt es sich wie bereits oben erwähnt um den Basisstationcontroller. Er überwacht eine Mobilfunkverbindung und regelt die Leistung nach. Zudem löst er einen Zellenwechsel, dem sogenannten Handover aus. OsmoNITB nutzt hierzu die freie Implementierung OpenBSC.

- **MSC/VLR**

Die Mobile-service switching center stellt im GSM/GPRS-Netz die Vermittlungsstelle dar. Sie übernimmt die Anrufverwaltung, die Verwaltung der Authentifizierung und innerhalb eines kommerziellen Netzes auch die Gebührenerfassung. Das Visitor Location Register (VLR) würde zudem, die jeweilige Basisstation unter der ein Endgerät eingebucht ist oder zuletzt war, verwalten. Sie kennt damit den Ort des Mobilfunknutzers. Unser Projekt wird jedoch nur mit einer Basisstation durchgeführt, weshalb kein Handover nötig ist. Damit verbleibt das Mobiltelefon nur in einer Location.

- **SMSC**

Der SMSC ist ein Server für SMS-Dienste. Er kümmert sich um die Verarbeitung von Textmitteilungen. Hierunter fällt beispielsweise die Speicherung, Weiterleitung von SMS. Sie wird im weiteren Projektverlauf nicht benötigt.

- **EIR**

Dies ist ein optionales Modul und wird nicht zwingend für den Betrieb eines GSM-Netzes benötigt. Dabei handelt es sich um ein Equipment Identity Register (EIR). Hier werden die weltweit eindeutigen Seriennummern der Mobilgeräte (IMEI, International Mobile Equipment Identity) gespeichert. Ziel dieser Datenbank ist ein Sperren verlorener oder gestohlener Endgeräte zu ermöglichen. Auch sie wird in diesen Projekt nicht benötigt. Zudem werden auch im kommerziellen Betrieb nur wenige Mobilfunkbetreiber dieses, da die IMEIs oftmals verändert werden können und die Datenbank damit wirkungslos ist.

- **HLR/AUC**

HLR steht für Home Location Register und bildet die Datenbank in der die Nummer eines Mobiltelefons zu dessen eindeutigen Identifiers, der IMSI (International Mobile Subscriber Identity), hinterlegt ist. Zudem wird hier die TMSI aufgelöst. Eine temporäre IMSI, die den Nutzer eines Mobilgeräts durch Zuweisen einer veränderlichen Identifiers, besser vor Tracking schützen soll. Das Authentication Center (AUC) ist die Authentifizierungszentrale und damit der Ort, an dem der Authentifizierungsschlüssel Ki abgelegt ist. Hier wird die Authentifizierung der SIM-Karte gegenüber dem Mobilfunknetz durchgeführt.

OsmoNITB implementiert damit das Network Switching Subsystem (NSS), aber mit dem BSC auch Teile des Base Station Subsystems (BSS). Das NSS ist hierbei der Teil der inneren Infrastruktur des Netzes, welches sich mit der Verwaltung von Mobilgeräten und Gesprächen beschäftigt. Das BSS hingegen kümmert sich im wesentlichen nur um die Verwaltung und Belegung der Funkschnittstelle.

2.3. Funktionsweise von OsmoNITB

In der Standardeinstellung von OsmoNITB werden Steuerungssignale von der BTS an diese weitergereicht, während die Gesprächsdaten mit Hilfe des verbindungslosen Protokoll UDP direkt an die Ziel-BTS gerichtet werden. Dazu sendet OsmoNITB in Form einer A-bis-Meldung einen Lauschbefehl an die Ziel-BTS. Eine Schicht höher wird dieses als Multimedia Protokoll RTP interpretiert. In unseren Projekt wurde die Einstellung „RTP proxy mode“ verwendet. Dies bewirkt, dass die RTP-Daten zuerst von OsmoNITB registriert werden und von dieser ausgehend nochmals an die Ziel-BTS versendet werden. Dies ist notwendig, da wir nur eine BTS betreiben, die zugleich Start-BTS und Ziel-BTS ist. Damit wird erreicht, dass dennoch ein RTP-Datenstrom detektierbar ist. Nachfolgendes Bild zeigt den prinzipiellen Aufbau eines derartigen Aufbaus:

Weiterhin läuft auf unseren System die Telefonanlage Asterisk. Sie ist mittels Osmocom-Sip-Connector an OsmoNITB angeschlossen. Dieser generiert aus dem MNCC-Datenstrom, dem klassischen Anrufsteuerungsprotokoll von ISDN, aus OsmoNITB einen SIP-Datenstrom (Session Initiation Protocol). Mit Hilfe dieses Sitzungsprotokolls kann ein Gespräch für die Man-in-the-

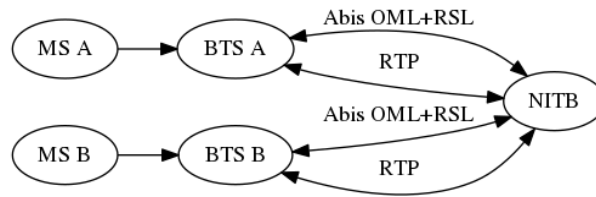


Abbildung 3: OsmoNITB mit RTP proxy mode

middle-Attacke abgefangen werden. Insgesamt ergibt sich somit für unser System folgender Aufbau:

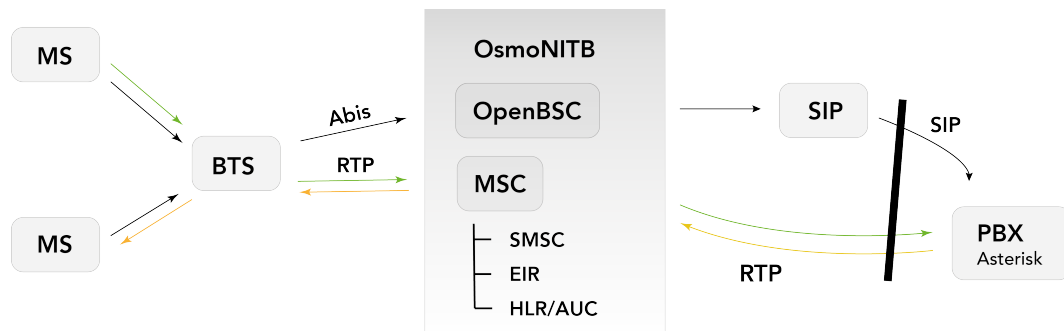


Abbildung 4: Unser Systemaufbau

Der Datenabgriff erfolgt in unserem Fall an der Stelle des schwarzen Balkens (s. Bild oben). Wird ein Gespräch aufgebaut, so signalisiert SIP dieses und der RTP-Stream kann abgefangen werden.

3. Architektur des OpenBTS Systems

OpenBTS (Open Base Transceiver Station) ist eine in C++ geschriebene, frei zugängliche Software-Suite des Unternehmens Range Networks. Zusammen mit einem Software Defined Radio (SDR) ist es möglich eine Basisstation für ein Mobilfunknetz mit GSM-Standard in Betrieb zu nehmen. Das Projekt nahm sich unter anderem zum Ziel die Kosten zur Inbetriebnahme eines GSM-Mobilfunknetzes so gering wie möglich zu halten, um in Gebieten eingesetzt werden zu können, in denen der Aufbau eines Mobilfunknetzes mit herkömmlichen GSM-Basisstationen nicht lukrativ genug ist.

Eine OpenBTS-Installation besteht dabei aus mehreren Softwarepaketen, welche die verschiedenen Funktionen eines Mobilfunknetzes implementieren und die unter der AGPLv3 Lizenz von Range Networks zur Verfügung gestellt werden. Diese Softwarepakete sind zuständig für die verschiedenen Bestandteile eines Mobilfunknetzes wie Schnittstellen, SMS-Versand, Teilnehmerauthentifizierung und Anrufvermittlung im eigenen Netz sowie, je nach Anbindung, zu VoIP- und Festnetzteilnehmern.

3.1. Aufbau und Zusammenspiel

Dieses Kapitel beschreibt zunächst die einzelnen Komponenten in der OpenBTS Architektur und deren Interaktion zueinander. Des Weiteren werden die Datenbanken, die ebenfalls teil der Infrastruktur sind, näher erläutert.

3.1.1. Bestandteile

Das vollständige OpenBTS-System beinhaltet zum Zeitpunkt des Projekts die folgenden Software-Komponenten:

- **OpenBTS**
Die eigentliche OpenBTS-Anwendung, die den Großteil des GSM-Stacks oberhalb des Radiomodems realisiert.
- **Transceiver**
Ein Software-Radiomodem sowie Hardware-Kontrollsystem, welches für die Anbindung eines Software Defined Radio (SDR) zuständig ist. In unserem Fall wurde das Universal Software Radio Peripheral (USRP) N210 SDR der Firma Ettus Research (siehe Abbildung 5) über das Netzwerk mit allen genutzten Computern verbunden.



Abbildung 5: USRP N210 Software Defined Radio

- **Asterisk**
Um eine Gesprächsvermittlung im Mobilfunknetz realisieren zu können, wird ein Private Branch Exchange (PBX) oder SIP Softswitch wie Asterisk benötigt, welcher somit die Hauptfunktionen eines klassischen Mobile Switching Center (MSC) übernimmt. Asterisk bietet die Möglichkeit sowohl innerhalb des eigenen Mobilfunknetzes, als auch ins Festnetz und zu VoIP-Services, Gespräche aufzubauen und unterstützt weitere Features wie Sprachdienste, Mailbox-Services oder Telefonkonferenzen.
- **SIPAuthServe**
SIPAuthserve verwaltet eine Subscriber Registry Datenbank, die die Teilnehmer-Informationen der im Netz registrierten Endgeräte enthält - unter anderem IMSI und Rufnummer. Diese Datenbank dient als Ersatz für das Home Location Register (HLR) eines klassischen GSM-Netzes sowie die SIP Registry von Asterisk.
- **SMQueue**
SMQueue ist ein RFC-3428 Store-and-Forward Message Service für die Übertragung und Speicherung von SMS-Nachrichten. Er verfügt über einen Shortcode-Handler, welcher ermöglicht den Inhalt von Textnachrichten als Eingabeargumente zu nutzen. So beinhaltet SMQueue standardmäßig einen Registrierungsprozess, der Benutzern dabei hilft eine gewünschte Rufnummer zu registrieren.

Die Beziehungen und Verbindungsprotokolle aller Komponenten eines OpenBTS-Systems werden in Abbildung 6 dargestellt. Dabei repräsentieren eckige Boxen Hardware-Komponenten, wohingegen abgerundete Boxen Software-Komponenten darstellen.

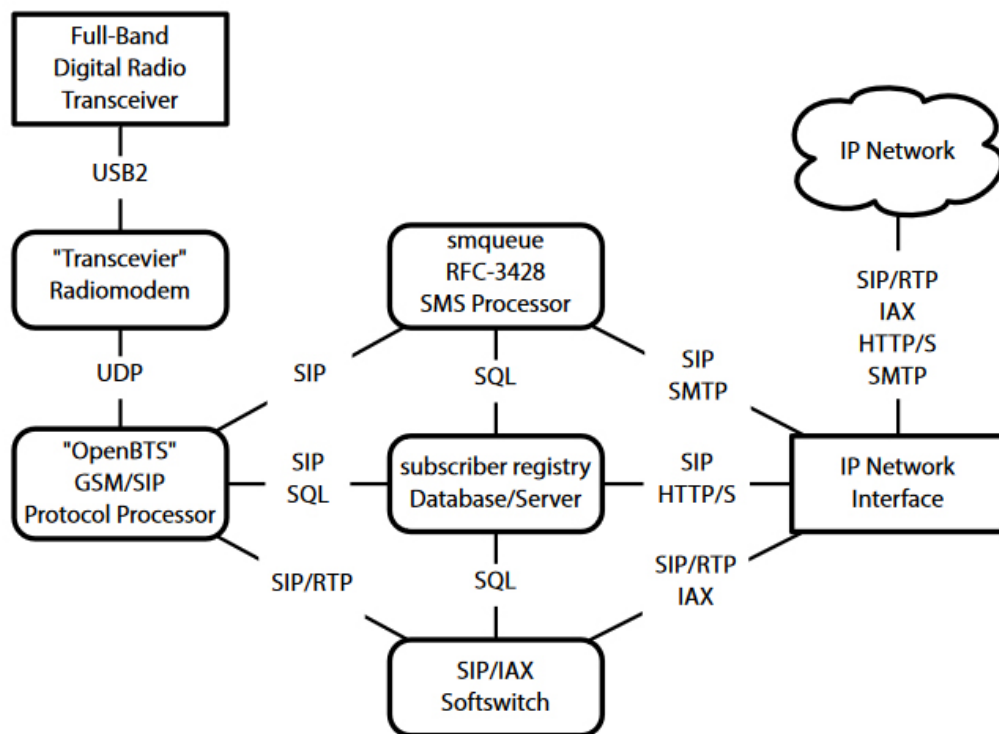


Abbildung 6: OpenBTS Systembestandteile

3.1.2. Datenbanken

Da in OpenBTS viele unterschiedliche Software-Komponenten miteinander interagieren, werden auch mehrere Datenbanken für Einstellungen oder die Kommunikation zwischen verschiedenen Komponenten benötigt. Die folgenden Datenbanken kommen dabei zum Einsatz:

Der Kommunikationsfluss der OpenBTS-Komponenten über die Datenbanken ist in Abbildung 7 dargestellt. **Schwarze** Pfeile bezeichnen SIP-Verbindungen, **rote** Pfeile stellen Datenbankzugriffe dar und der **blaue** Pfeil eine ODBC-Verbindung, die nicht standardmäßig in OpenBTS integriert ist und die zusätzlich für Asterisk Real-Time konfiguriert werden muss.

OpenBTS.db	Enthält alle Konfigurationseinstellungen des OpenBTS-Hauptprogramms. Die Einstellungen der für uns lizenzierten Frequenz, sowie Netzwerkparameter wie MCC, MNC oder Name können hier gesetzt werden.
TMSITable.db	Enthält die TMSI-IMSI Beziehungen der registrierten Endgeräte und wird vom OpenBTS-Hauptprogramm genutzt. Der Pfad wird als Parameter in der OpenBTS.db-Datenbank gesetzt.
ChannelTable.db	Enthält den Channel Status aller aktiven Channel und wird vom OpenBTS-Hauptprogramm genutzt. Der Pfad wird als Parameter in der OpenBTS.db-Datenbank gesetzt.
sipauthserve.db	Enthält alle Konfigurationseinstellungen von SIPAuthServe, unter anderem den Dateipfad zur Subscriber Registry (sqlite3.db).
smqueue.db	Enthält alle Konfigurationseinstellungen von SMQueue, unter anderem den Dateipfad zur Subscriber Registry (sqlite3.db).
sqlite3.db	Subscriber Registry, auf die von SIPAuthServe und SMQueue zugegriffen wird. Wenn Asterisk mit Real-Time Funktionen konfiguriert wird, greift auch Asterisk via ODBC auf die sqlite3-Datenbank zu.

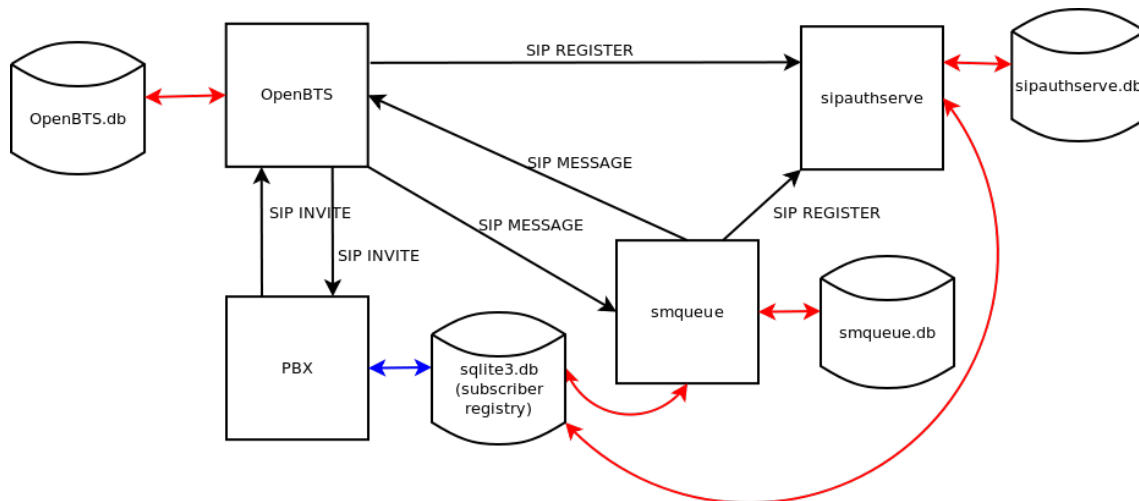


Abbildung 7: OpenBTS System Diagramm

4. Inbetriebnahme eines Osmocom Systems

Für Inbetriebnahme des GSM Netzes waren einige Vorinstallationen sowie das Einrichten von Ubuntu 16.04.3 nötig. Im Folgenden wird das Vorgehen zur Einrichtung des Systems sowie die Inbetriebnahme des GSM Netzes beschrieben.

4.1. Vorinstallationen

Dieses Kapitel behandelt die für die Installation und Inbetriebnahme notwendigen Softwarevoraussetzungen.

4.1.1. Ubuntu 16.04.3

Zunächst wurde wie bei der Installation von Osmocom das Betriebssystem Ubuntu 16.04.3 auf einem Labor-Rechner installiert und eingerichtet. Zusätzlich wurden alle Programme und Pakete mit den folgenden Befehlen aktualisiert.

```

1 sudo apt-get update
2 sudo apt-get upgrade

```

4.1.2. Git

Da die Open-Source Projekte von OsmocomBB auf Git-Repositories liegen, wurde zunächst Git eingerichtet. Zur Versionskontrolle und Verwaltung des Codes wurde außerdem ein Team-eigenes Git Repository angelegt.

```
1 sudo apt-get install git
```

4.1.3. Softwarevoraussetzungen

Osmocom empfiehlt zunächst die Einrichtung von einigen Bibliotheken und sonstigen, nötigen Abhängigkeiten als Voraussetzung für die Inbetriebnahme der GSM Komponenten. Diese wurden mittels Paketmanagers wie folgt installiert.

```
1 sudo apt-get install libpcsc-lite-dev libtalloc-dev libortp-dev libsctp-dev
2 libmnl-dev libdbi-dev libdbd-sqlite3 libsqlite3-dev sqlite3 libc-ares-dev
3 libdbi0-dev libdbd-sqlite3 build-essentials libtool autoconf automake pkg-
  config
4 libsqlite3-tcl sqlite-autoconf sqlite-autoconf
```

Die die Fehler bezüglich Bumpversion nicht behoben werden konnten, wurden sie ignoriert. Dies zog keinerlei Konsequenzen hinsichtlich der Inbetriebnahme der GSM Komponenten nach sich. Zusätzlich bedarf es der separaten Installation der Software Bibliotheken libosmo-abis, libosmo-core und libosmo-netif. Diese wurden von den entsprechenden Git Repositories heruntergeladen und nach analogem Vorgehen installiert.

```
1 git clone git://git.osmocom.org/<lib-source>
2 cd <lib-source>
3 autoreconf -fi
4 ./configure
5 make
6 make install
7 sudo ldconfig
```

Trotz der sorgfältigen Installation einiger Softwarevoraussetzungen traten zusätzliche Abhängigkeiten bei der Installation einzelner GSM Komponenten auf, welche in 4.2 beschrieben sind.

4.1.4. Aktivierung der Verbindung zum USRP2

Vor Installation des Treibers sollte zunächst die Netzwerkschnittstelle aktiviert werden. Die Default IP-Adresse des Ettus USRP2 ist 192.168.10.2. Dafür mussten wir zuerst erfahren, welche Bezeichnung die für den Ettus USRP2 verwendete Schnittstelle nutzt.

```
1 sudo ifconfig
```

Daraufhin konnten wir die IP-Adresse der Schnittstelle am PC so setzen, dass es möglich sein sollte mit dem Ettus USRP2 zu kommunizieren.

```
1 sudo ifconfig enp0s25 192.168.10.3
2 ping 192.168.10.2
```

Allerdings war zunächst keine Kommunikation mit dem USRP2 möglich, deshalb versuchten wir das Gerät zu finden, indem wir den kompletten IP-Bereich des Netzwerks anpingten sowie die automatische Suchfunktion von UHD-Geräten nutzten.

```
1 nmap -sP 192.168.10.0/24 | grep -oE '([[:digit:]]{1,3}\.){3}[[:digit:]
  :]]{1,3}'
2 uhd_find_devices
```

Nachdem auch diese beiden Befehle zu keinem Erfolg führten und das Gerät auch in anderen IP-Netzbereichen nicht gefunden werden konnte, brachten wir den N210 in den Safe-Mode, in

welchem es immer die IP-Adresse 192.168.10.2 besitzt. Dafür mussten wir den N210 aufschrauben und den Safe-Mode Knopf (S2) im Inneren drücken und bis zum erfolgreichen Neustart gedrückt halten. Nun konnten wir das Image des N210 aktualisieren und gleichzeitig die IP-Adresse außerhalb des Safe-Modus auf 192.168.10.2 setzen.

```
1 uhd_image_loader --args="type=usrp2,addr=192.168.10.2, reset"
2 sudo /usr/lib/uhd/uhd/uhd_recovery.py --ifc=enp0s25 --new-ip
   =192.168.10.2
```

Die automatische Suchfunktion zeigte nun endlich auch im normalen Modus eine Verbindung.

```
1 uhd_find_devices
2
3 linux; GNU C++ version 5.3.1 20151219; Boost_105800; UHD_003.009.002-0-
   unknown
4
5 -----
6 -- UHD Device 0
7 -----
8 Device Address:
9   type: usrp2
10  addr: 192.168.10.2
11  name:
12  serial: F449BF
```

4.2. Installation einzelner GSM Komponenten

OsmocomBB hält detaillierte Beschreibungen zur Installation der GSM Komponenten bereit, welche zur Inbetriebnahme des in Rahmen dieser Arbeit verwendeten GSM Netzes herangezogen wurden. Im Folgenden wird die Installation und Einrichtung der GSM Komponenten genauer erläutert.

4.2.1. OsmoTRX

Zur Kommunikation mit der Basisstation ist der osmoTRX Transceiver nötig. Osmocom bietet diesen - meist wie die anderen Komponenten - im Git Repository an. Die Installation des Transceiver erforderte die Bibliotheken *libusb-1.0-0-dev*, *uhd-host*, *libboost-dev* und *libuhd-dev*. Diese bieten die Suchfunktion *uhd_find_devices*. Dadurch lässt sich testen, ob die Basisstation gefunden wird. Mittels *osmo-trx -f* lässt sich der Transceiver nach der Installation starten.

```
1 git clone git://git.osmocom.org/osmo-trx
2 cd osmo-trx/
3 autoreconf -i
4 ./configure
5 sudo make -j8
6 sudo make install
7 osmo-trx -f
```

Zur Behebung der Warnung, die in Abbildung 9 zu sehen ist, wurde die Priorität in der Datei */etc/security/limits.conf* gesetzt.

```
1 @usrp - rtprio 50
```

4.2.2. OsmoBTS

Die Installation von OsmoBTS erfolgte analog zur Einrichtung der anderen Osmocom Komponenten.

```
1 git clone git://git.osmocom.org/osmo-bts.git
2 autoreconf -fi
3 cd osmo-bts
```

```

mitm@netpc02: ~/mitm/3.Osmocom/Startskripte
mitm@netpc02:~/mitm/3.Osmocom/Startskripte$ sudo ./startTransceiver.sh
Transceiver wird gestartet
linux; GNU C++ version 5.3.1 20151219; Boost_105800; UHD_003.009.002-0-unknown

opening configuration table from path :memory:
Info: SSE3 support compiled in and supported by CPU
Info: SSE4.1 support compiled in and supported by CPU
Config Settings
  Log Level..... NOTICE
  Device args.....
  TRX Base Port..... 5700
  TRX Address..... 127.0.0.1
  GSM Core Address.....127.0.0.1
  Channels..... 1
  Tx Samples-per-Symbol... 4
  Rx Samples-per-Symbol... 1
  EDGE support..... Disabled
  Reference..... Internal
  C0 Filler Table..... Dummy bursts
  Multi-Carrier..... Disabled
  Tuning offset..... 0
  RSSI to dBm offset..... 0
  Swap channels..... 0

-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
-- Detecting internal GPSDO.... Found an internal GPSDO
-- Setting references to the internal GPSDO
-- Transceiver active with 1 channel(s)

```

Abbildung 8: Ansicht der Konsole nach dem Start des Transceivers

```

UHD Warning:
  Unable to set the thread priority. Performance may be negatively affected.
  Please see the general application notes in the manual for instructions.
  EnvironmentError: OSError: error in pthread_setschedparam

```

Abbildung 9: Fehlermeldung bezüglich der Thread Priorität in osmoTRX

```

4 ./configure --enable-trx
5 sudo make -j8
6 sudo make install

```

Zur Konfiguration wurden zunächst die Pfade der folgenden Variablen angepasst.

```

1 PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
  games:/usr/local/games"
2 PKG_CONFIG_PATH="/home/netpc06/libosmo-abis/"
3 LIBOSMOTRAU_CFLAGS="/home/netpc06/libosmo-abis/"
4 LIBOSMOTRAU_LIBS="/home/netpc06/libosmo-abis/"

```

Weiterhin erforderte die Konfiguration von OsmoBTS Änderungen an der Datei `osmo-bts.cfg` (siehe Anhang A). Es wurde die Option *band* auf PCS1900 gesetzt, welche dem lizenzierten Frequenzband entspricht. Des Weiteren wurde die Signalstärke durch Angabe des *osmotrx rx-gain* auf 1 gesetzt. Eine letzte Änderung wurde an der lokalen und remote IP vorgenommen, welche auf 127.0.0.1 gesetzt wurde.

Die Konfigurationsdatei wurde unter dem Pfad `/home/netpc06/osmo-bts/src/osmo-bts-trx` abgelegt.

4.2.3. OsmoNITB unter OpenBSC

Weiterhin wurde OpenBSC installiert. Dazu wurde die Bibliothek *libssl-dev* (eigentlich unter *libcrypto* bekannt) vorausgesetzt.

```
1 sudo apt-get install libssl-dev
2 git clone git://git.osmocom.org/openbsc
3 cd openbsc/
4 cd openbsc/openbsc/
5 autoreconf -i
6 ./configure
7 sudo make -j8
8 sudo make install
```

Zur Konfiguration von OpenBSC wurde - analog zu OsmoBTS - eine Beispieldatei wie folgt angepasst (siehe Anhang B). *short name* und *long name* beschreiben den Name des Netzwerks und wurden umbenannt. Die Option *auth policy* wurde auf *accept-all* gesetzt, um alle Anfrage zur Registrierung im Netz zuzulassen. Andernfalls muss die IMSI der jeweiligen Mobilfunkstation (MS) in der Datenbank des Home Location Registers (HLR) hinterlegt sein. Da der erlaubte Frequenzbereich 1909,0/1989,0 MHz beträgt, wurde die Option *band* auf *PCS1900* gesetzt. Weiterhin wurde die *ipa.unit-id* an die der OsmoBTS angepasst (1901 0). Eine letzte Änderung wurde an der sogenannten Absolute Radio Frequency Channel Number (ARFCN) vorgenommen, die durch die Option *arfcn* angegeben wird. Dieser ergibt sich wie folgt:

$$ARFCN = Offset + \frac{f_{up} - f_{UplinkStart}}{f_{bandbreite}} \quad (1)$$

Der oben genannte Frequenzbereich lässt sich auf ein Netzwerk vom Typ PCS1900 zurückführen. Als *Offset* wurde ein Wert von 512 gewählt, der für diesen Frequenzbereich üblich ist. f_{up} wurde auf einen Wert von 1850,2 MHz gesetzt, da dieser der Startwert des Uplinks für das PCS1900 ist. In der Frequenzzuteilung der Bundesnetzagentur wurde eine Bandbreite von 0,2 MHz angegeben. Unter Verwendung dieser Werte ergibt sich ein ARFCN von 806.

Nach Änderung der Konfigurationsdatei *openbsc.cfg* wurde diese unter dem Pfad */home/netpc06/openbsc/openbsc/src/osmo-nitb* abgelegt.

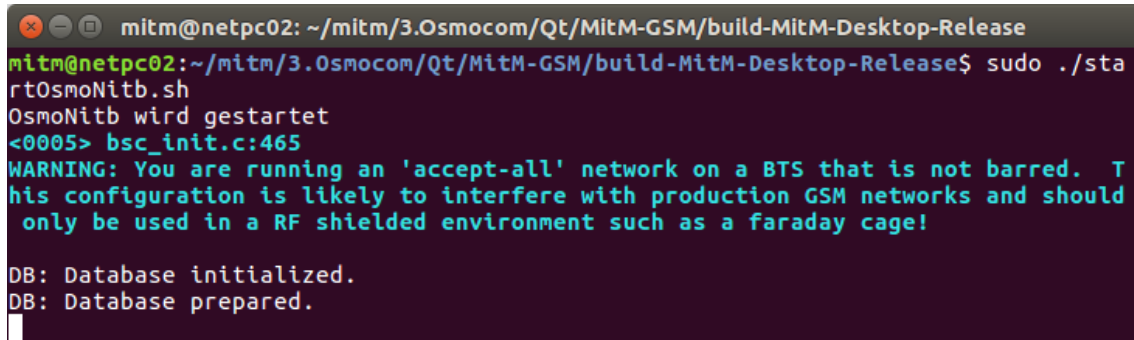
4.3. Starten des Systems

Nach Installation aller Komponenten wurde das System gestartet. Dabei wurden als Optionen die Pfade der Konfigurationsdateien angegeben. Da OsmoBTS den Transceiver fordert, musste dieser als erste Instanz gestartet werden. Die Reihenfolge der weiteren Komponenten spielt keine Rolle.

```
1 // Transceiver starten
2 cd /home/netpc06/osmo-trx
3 sudo osmo-trx -f
```

Als Option beim Start von Osmo-Nitb wurden sowohl der Pfad der Konfigurationsdatei als auch der der HRL Datenbank angegeben.

```
1 cd /home/netpc06/openbsc/openbsc/src/osmo-nitb
2 osmo-nitb -c /home/netpc06/openbsc/openbsc/src/osmo-nitb/openbsc.cfg -l /
  home/netpc06/openbsc/openbsc/src/osmo-nitb/hlr.sqlite3 -P -C --debug=
  DRLL:DCC:DMM:DRR:DRSL:DNM
```

```
mitm@netpc02: ~/mitm/3.Osmocom/Qt/MitM-GSM/build-MitM-Desktop-Release
mitm@netpc02:~/mitm/3.Osmocom/Qt/MitM-GSM/build-MitM-Desktop-Release$ sudo ./startOsmoNITB.sh
OsmoNITB wird gestartet
<0005> bsc_init.c:465
WARNING: You are running an 'accept-all' network on a BTS that is not barred. This configuration is likely to interfere with production GSM networks and should only be used in a RF shielded environment such as a faraday cage!
DB: Database initialized.
DB: Database prepared.
```

Abbildung 10: Ansicht der Konsole nach dem Start der OsmoNITB

OsmoBTS wurde wie folgt gestartet.

```
1 cd /home/netpc06/osmo-bts/src/osmo-bts-trx
2 sudo osmo-bts-trx -c osmo-bts.cfg
```

Konsolenausgabe siehe Anhang C.

4.4. Installation weiterer Komponenten

Neben den Komponenten zur Inbetriebnahme des GSM Netzes wurden außerdem weitere Installationen vorgenommen. Diese waren zur Umsetzung des Projektziels notwendig.

4.4.1. Osmo-sip-Connector

Zur Konvertierung von .wav-Dateien werden sowohl RTP als auch SIP Pakete benötigt. Um neben RTP und UDP Paketen auch SIP Pakete abfangen zu können, wurde der Osmo-sip-Connector wie folgt installiert.

```
1 git clone git://git.osmocom.org/osmo-sip-connector.git
2 cd osmo-sip-connector/
3 autoreconf -fi
4 ./configure
5 make
6 sudo make install
```

Die Konfigurationsdatei beinhaltet lediglich folgendes.

```
1 app
2 mncc
3 socket-path /tmp/bsc_mncc
4 sip
5 local 127.0.0.1 5069
6 remote 127.0.0.1 5060
```

Nach erfolgreicher Installation wurde der Osmo-sip-connector mit Angabe der Konfigurationsdatei gestartet.

```
1 osmo-sip-connector -c ./osmo-sip-connector.cfg
```

Zusätzlich muss nun beim Start von Osmo-NITB der Parameter *M tmpbsc_mncc* mit angehängt werden, sodass der Osmo-sip-Connector die Daten empfangen und konvertieren kann.

4.4.2. Asterisk

Den Aufbau von Gesprächen sowie deren Vermittlung wurde in dieser Arbeit der PBX Asterisk verwendet. Dieser wurde über den Paketmanager installiert. Die Einrichtung erforderte die Installation der Bibliothek *libsofia-sip-ua-glib-dev*.

```
mitm@netpc02: ~/mitm/3.Osmocom/Qt/MitM-GSM/build-MitM-Desktop-Release
mitm@netpc02:~/mitm/3.Osmocom/Qt/MitM-GSM/build-MitM-Desktop-Release$ sudo ./startSipConnector.sh
SipConnector wird gestartet
<0004> telnet_interface.c:102 telnet at 127.0.0.1 4256
<0001> mncc.c:879 Scheduling MNCC connect
su_source_port_create() returns 0x148e700
<0001> mncc.c:786 Reconnected to /tmp/bsc_mncc
<0001> mncc.c:699 Got hello message version 5
```

Abbildung 11: Ansicht der Konsole nach dem Start des Osmo-sip-connectors

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

Um Asterisk mit dem Osmo-sip-Connector zu verbinden muss die *sip.conf* im Verzeichnis */etc/asterisk* erweitert werden.

```
1 [GSM]
2 type=friend
3 host=127.0.0.1
4 dtmfmode=rfc2833
5 canreinvite=no
6 disallow=all
7 allow=gsm
8 context=gsmsubscriber
9 port=5069
```

Des Weiteren die *extensions.conf* im selben Verzeichnis erweitern.

```
1 [gsmsubscriber]
2 exten=>_XXXXX,1,Dial(SIP/GSM/\${EXTEN})
3 exten=>_XXXXX,n,Playback(vm-nobodyavail)
4 exten=>_XXXXX,n,HangUp
```

Nach der Installation startet Asterisk selbständig. Da mehrmals Änderungen an den Konfigurationsdateien von Asterisk vorgenommen wurden, wurde Asterisk mehrmals neu gestartet.

```
1 core restart gracefully
2 sudo asterisk -r
```

Nach mehreren Fehlschlägen bezüglich der Installation von Asterisk und folglichem Neuaufsetzen des gesamten Systems inklusive des Betriebssystems wurde Asterisk als erste Komponente vor der GSM Installation eingerichtet. Es stellte sich heraus, dass dieses Vorgehen zu einer erfolgreichen Installation von Asterisk und Inbetriebnahme des GSM Netzes führte.

```
mitm@netpc02: ~
mitm@netpc02:~$ sudo asterisk -r
Asterisk 13.1.0-dfsg-1.1ubuntu4.1, Copyright (C) 1999 - 2014, Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public License version 2 and other licenses; you are welcome to redistribute it under certain conditions. Type 'core show license' for details.
=====
Connected to Asterisk 13.1.0-dfsg-1.1ubuntu4.1 currently running on netpc02 (pid = 1400)
netpc02*CLI> core restart gracefully
```

Abbildung 12: Ansicht der Konsole nach Verbindung mit Asterisk

4.5. Verwendung einer GUI

Der Aufruf mittels Befehleingabe im Terminal zeigte sich als dauerhaft zu umständlich. Ständiges Wechseln der Pfade, gestaltete sich ebenso störend, wie die Eingabe der Befehle, die mal mit, mal ohne Root-Rechte gestartet werden konnten. Aus diesen Grund erstellten wir Shellskripte, die ein unnötiges hantieren im Terminal verhindern sollte. Aber auch diese Lösung gefiel uns nicht. Da wir teilweise an verschiedenen Rechnern verschiedene Konzepte umgesetzt hatten, aber jede Gruppe ihren Testbedarf hatte, musste unser GSM-Netz oftmals eingeschaltet und kurz darauf wieder abgeschaltet werden. Jeder Einschaltvorgang bestand dabei mindestens aus vier Terminalfenster in denen die Startskripte aufgerufen werden mussten. Zudem musste dies in richtiger Reihenfolge geschehen. Abhilfe schaffte hier eine einfache GUI-Anwendung, die mit Hilfe von Qt in C++ verfasst wurde.

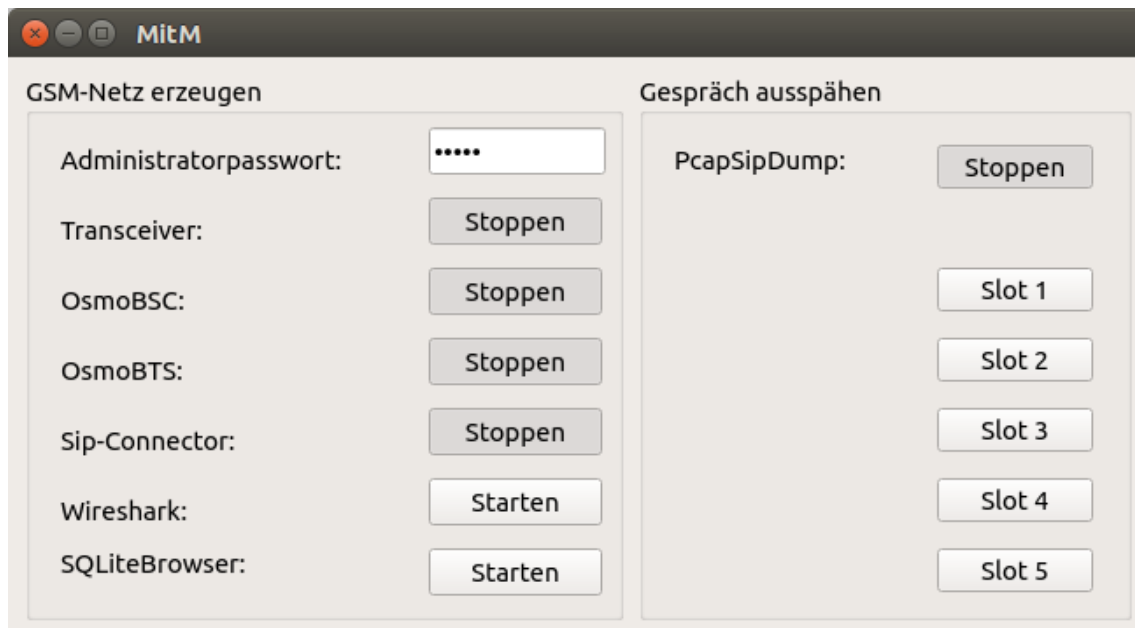


Abbildung 13: GUI-Anwendung MitM

Diese besteht im wesentlichen aus zwei Seiten. Die linke Seite beinhaltet zahlreiche Toggle-Buttons, mit denen die vorhandenen Shellskripte aufgerufen werden. Sie schalten die jeweilige Anwendung ein oder aus. Zudem ist ein Eingabefeld für das Root-Passwort vorhanden, da etliche Anwendungen Root-Rechte benötigen. Allerdings konnte dies nicht mehr fertiggestellt werden, so dass die Eingabe des Passworts derzeit wirkungslos bleibt. Stattdessen muss die GUI-Anwendung selbst mit Root-Rechten gestartet werden. Die linke Seite dient daher der Erzeugung eines GSM-Netzes. Auf der rechten Seite hingegen, befinden sich die Buttons mit denen ein Gespräch ausgespäht werden kann. Mit den Toggle-Button "PcapSipDump" wird das Shellskript gestartet, welches die Gespräche abfängt und in Audiostreams umwandelt. Die zugehörigen Buttons "Slot 1 -5" geben diese Gespräche wieder.

Quellcode siehe Anhang D.

5. Inbetriebnahme eines OpenBTS Systems

Die Inbetriebnahme des OpenBTS Systems gestaltete sich in den Vorinstallationen ähnlich wie beim Osmocom System.

5.1. Vorinstallationen

5.1.1. Ubuntu 16.04.3

Siehe Kapitel 4.1.1

5.1.2. Git

Auch Range Networks nutzt für OpenBTS Git-Repositories, weshalb auch für die Installation von OpenBTS zunächst Git eingerichtet wurde. Zur Versionskontrolle und Verwaltung des Codes wurde auch hier das Team-interne Git Repository genutzt.

```
1 sudo apt-get install software-properties-common python-software-properties
2 sudo add-apt-repository ppa:git-core/ppa
3 sudo apt-get update
4 sudo apt-get install git
```

5.1.3. Softwarevoraussetzungen

In vielen Installationsanleitungen werden zuerst eine Reihe von Bibliotheken und Paketen angegeben, die für die Installation von OpenBTS notwendig sein sollen. Range Networks hat die Installation dieser Pakete inzwischen ins Build-Skript integriert, sodass alle vorausgesetzten Abhängigkeiten später automatisch installiert werden sollten.

5.1.4. Aktivierung der Verbindung zum USRP2

Nachdem wir in Kapitel 4.1.4 die IP-Adresse des N210 erfolgreich zurücksetzen konnten, mussten wir in diesem Fall nur eine Verbindung zum USRP2-Gerät aufnehmen, indem wir die IP-Adresse des PCs innerhalb des gleichen Netzwerkbereichs setzten.

```
1 sudo ifconfig enp0s25 192.168.10.3
```

Dieses mal konnten wir mithilfe der automatischen Erkennungsfunktion direkt erkennen, dass eine Verbindung zum N210 bestand.

```
1 uhd_find_devices
```

Da uns im Laufe des Projekts die IP-Adresse der Schnittstelle am PC immer wieder zurückgesetzt wurde, speicherten wir die IP-Adresse daraufhin direkt in den Netzwerkeinstellungen.

5.2. Installation

Range Networks stellt eine detaillierte Anleitung zur Installation von OpenBTS inklusive aller zuvor beschriebenen Komponenten bereit, die zur Inbetriebnahme eines GSM-Netzes benötigt werden. Im Folgenden werden die Schritte zur Umsetzung auf der uns vorgelegenen Hardware genauer beschrieben.

Die aktuellste Version von OpenBTS kann direkt aus dem Git-Repository von Range Networks geladen werden.

```
1 sudo git clone https://github.com/RangeNetworks/dev.git
2 cd dev
3 sudo ./clone.sh
4 sudo ./switchto.sh master
```

Die Installation wird von Range Networks so leicht wie möglich gemacht, indem alle Komponenten zusammen aus ihren Git-Repositories heruntergeladen werden. Zugleich wird ein Build-Skript bereitgestellt, welches alle Build-Abhängigkeiten installiert und jede Komponente in ein installierbares Paket kompiliert.

Dafür muss lediglich das Build-Skript mit dem Namen des Transceivers als erstes Argument ausgeführt werden. Um einzelne Pakete zu kompilieren, kann zusätzlich optional der Name der Komponente als zweites Argument angegeben werden.

```
1 # (vom OpenBTS root)
2 sudo ./build.sh N210 [component-name]
```

Dieser Schritt dauerte bei uns eine ganze Weile - geschätzt eine halbe Stunde. Im Anschluss sollten alle kompilierten Pakete im Unterverzeichnis BUILDS/<timestamp> zu finden sein. Die beiden Libraries *libcoredumper* und *liba53* werden auch gleichzeitig installiert, alle restlichen Software-Pakete müssen im folgenden Schritt manuell installiert werden.

```
1 # (vom OpenBTS root)
2 sudo dpkg -i BUILDS/<timestamp>/*.deb
```

Um möglicherweise fehlende Abhängigkeiten aufzulösen, sollte noch folgender Befehl ausgeführt werden.

```
1 sudo apt-get -f install
```

5.3. Konfiguration

Nachdem alle Komponenten erfolgreich installiert wurden, muss der für den USRP N210 passende Transceiver verlinkt werden.

```
1 # (vom OpenBTS root)
2 cd apps
3 ln -s ../Transceiver52M/transceiver .
```

5.3.1. Datenbanken initialisieren

Nun mussten noch einige der Datenbanken erstellt werden. Für die Initialisierung stehen in den Programmverzeichnissen .sql-Dateien zur Verfügung. Zuerst haben wir den Ordner für die Datenbanken erstellt und dann die OpenBTS-Datenbank initialisiert.

```
1 # (vom OpenBTS root)
2 sudo mkdir /etc/OpenBTS
3 sudo sqlite3 -init ./apps/OpenBTS.example.sql /etc/OpenBTS/OpenBTS.db ".quit"
```

Mit folgendem Befehl kann daraufhin getestet werden, ob die Datenbank erfolgreich initialisiert wurde. Wenn eine ganze Reihe an Zeilen ausgegeben werden - eine leere Datenbank dürfte nur drei Zeilen ausgeben - so scheint alles geklappt zu haben.

```
1 sqlite3 /etc/OpenBTS/OpenBTS.db .dump
```

Die Datenbanken von SMQueue und SIPAuthServe wurden analog dazu initialisiert.

```
1 # (vom OpenBTS root)
2 sudo sqlite3 -init subscriberRegistry/sipauthserve.example.sql /etc/OpenBTS
  /sipauthserve.db ".quit"
3 sudo sqlite3 -init smqueue/smqueue/smqueue.example.sql /etc/OpenBTS/smqueue
  .db ".quit"
```

Auch hier sollte getestet werden, ob die Datenbanken erfolgreich erstellt und mit Daten gefüllt wurden.

```
1 sqlite3 /etc/OpenBTS/sipauthserve.db .dump
2 sqlite3 /etc/OpenBTS/smqueue.db .dump
```

5.3.2. OpenBTS Einstellungen

Um OpenBTS nun an unser Netz und die lizenzierten Frequenzen anzupassen, mussten wir hier einige Einstellungen vornehmen. Die Änderungen haben wir direkt in der OpenBTS.db Datenbank vorgenommen, welche wir mithilfe des Programms *SQLitebrowser* geöffnet haben. Dieses mussten wir zunächst installieren und daraufhin starten.

```
1 sudo apt-get install sqlitebrowser
2 sudo sqlitebrowser
```

Im Programm konnten wir dann die Datenbank unter `/etc/OpenBTS/OpenBTS.db` öffnen und so folgende Parameter anpassen.

Control.LUR.OpenRegistration	.*	// Registrierung für alle Teilnehmer erlauben
GSM.Identity.MCC	001	// 001 für Testnetze
GSM.Identity.MNC	050	// frei wählbar
GSM.Identity.ShortName	mitm.de	// frei wählbar
GSM.Radio.Band	1900	// GSM Frequenzband
GSM.Radio.C0	806	// ARFCN des Frequenzbandes
GSM.Radio.RxGain	8	// Empfängerverstärkung in dB.
		// Für Ettus Geräte zwischen 0-10 empfohlen.

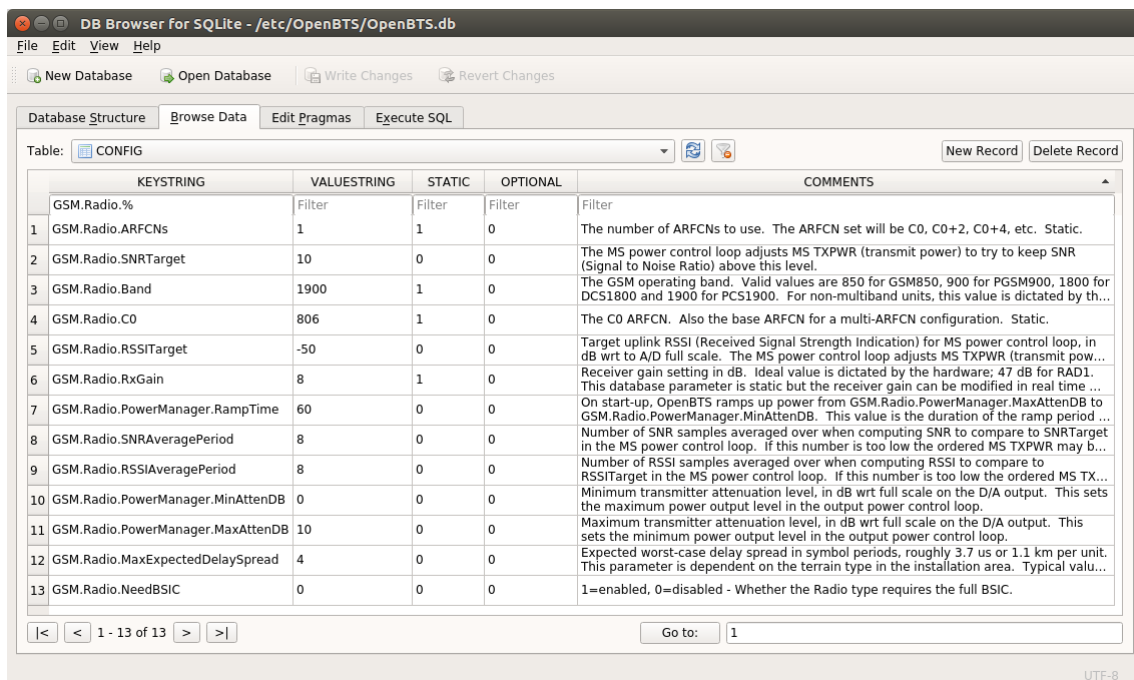


Abbildung 14: Ansicht in SQLitebrowser zur Konfiguration der OpenBTS.db-Datenbank

Da wir OpenBTS zunächst ohne vorherige Anpassung der Einstellungen am Radioband gestartet haben, bekamen wir folgende Fehlermeldung nach dem Start des Transceivers. Diese verschwand als wir die entsprechenden Einstellungen gesetzt hatten.

```
1 ALERT 2577:2596 2017-09-14T13:49:16.0 Transceiver.cpp:414:pullRadioVector:
   Clipping detected on RACH input
```

Damit waren die Grundeinstellungen für den Betrieb von OpenBTS gesetzt.

5.3.3. Asterisk Einstellungen

Da bei uns die Konfigurationsdateien von Asterisk nicht bei jeder Neuinstallation aktualisiert wurden, haben wir diese teilweise manuell zurückgesetzt.

```
1 # (vom OpenBTS root)
2 sudo cp asterisk-config/* /etc/asterisk
```

Nach Änderungen an den Konfigurationsdateien musste Asterisk neu gestartet werden bevor die Änderungen in Kraft getreten sind.

```
1 # (vom Userverzeichnis)
2 sudo ./asterisk -r           //so kann die CLI von Asterisk genutzt werden
3 sudo restart gracefully
```

5.4. Starten des Systems

Nach Installation aller Komponenten wurde das System gestartet. Die Reihenfolge spielt hierbei keine Rolle, da der Transceiver selbstständig von OpenBTS gestartet wird und alle Dienste über verschiedene Ports miteinander kommunizieren.

Um alle Programme später möglichst leicht starten zu können, wurden Links zu den ausführbaren Dateien im Userverzeichnis erstellt. Nur für OpenBTS wurde ein Link zum Überverzeichnis gesetzt, da das Starten ansonsten nicht funktionierte.

```
1 # (vom Userverzeichnis)
2 sudo ln -s dev/asterisk/asterisk-11.7.0/main/asterisk .
3 sudo ln -s dev/smqueue/smqueue/smqueue .
4 sudo ln -s dev/subscriberRegistry/apps/sipauthserve
5 sudo ln -s dev/openbts/apps/
6 sudo mv apps openbts
```

So konnten die einzelnen Programme nun jederzeit ohne größere Suche der Dateien gestartet werden.

```
1 # (vom Userverzeichnis)
2 sudo ./sipauthserve
3 sudo ./smqueue
4 sudo ./openbts/OpenBTS
5 sudo ./asterisk
```

Beim Start der verschiedenen Komponenten wurden uns folgende Zeilen ausgegeben.

Ansicht der OpenBTS Konsole siehe Anhang E.

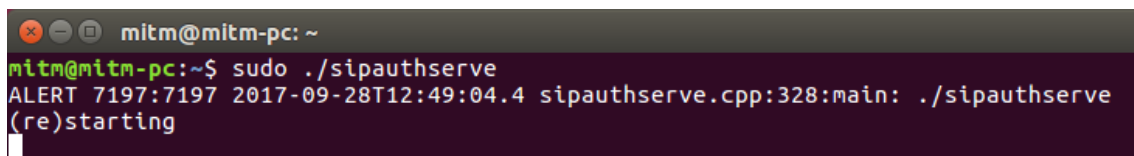


Abbildung 15: Ansicht der Konsole nach dem Start von SIPAuthServe

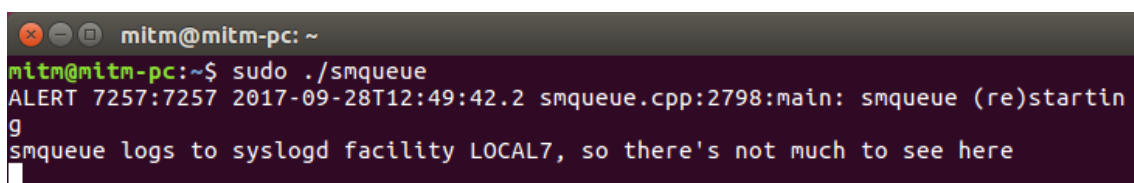


Abbildung 16: Ansicht der Konsole nach dem Start von SMQueue


```

mitm@mitm-pc: ~
mitm@mitm-pc:~$ sudo asterisk
Privilege escalation protection disabled!
See https://wiki.asterisk.org/wiki/x/1gKfAQ for more details.
mitm@mitm-pc:~$ sudo asterisk -r
Privilege escalation protection disabled!
See https://wiki.asterisk.org/wiki/x/1gKfAQ for more details.
Asterisk 11.7.0.5, Copyright (C) 1999 - 2013 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=====
Running as user 'asterisk'
Running under group 'www-data'
Connected to Asterisk 11.7.0.5 currently running on mitm-pc (pid = 9314)
mitm-pc*CLI>

```

Abbildung 17: Ansicht der Konsole nach dem Start von Asterisk

5.5. Testen der Funktionen

Nach dem Start aller Anwendungen war es uns nun endlich möglich unser eigenes Mobilfunknetz zu testen. Natürlich funktionierte nicht alles sofort, vor allem die Asterisk-Einstellungen stellten uns vor einige Probleme. Letztendlich konnten alle elementaren Funktionen in Betrieb gesetzt werden, nachdem alle Einstellungen für eine ODBC-Verbindung zwischen Asterisk und der sqlite3.db Datenbank entfernt wurden. Diese ODBC-Schnittstelle sollte für eine Real-Time Verbindung sorgen, die sichere Verbindungen über Asterisk ermöglicht, da die Verbindungen ansonsten über externe Schnittstellen übertragen werden.

Nach erfolgreicher Anmeldung im Testnetz erhielten wir sofort eine Nachricht von der Nummer 101, die wir per Datenbank hätten anpassen können. Antwortet man nun mit einer 7-10 stelligen Nummer, so wird dem Endgerät diese Telefonnummer zugewiesen und in der sqlite3.db Datenbank unter der zugehörigen IMSI abgespeichert.



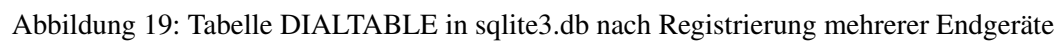
Abbildung 18: Eingehende Nachricht nach erstmaliger Registrierung im Testnetz

Das Senden von Nachrichten an verschiedene Teilnehmer im Netzwerk funktionierte schon von Beginn an einwandfrei. Daraufhin testeten wir einige der Nummern zu Testzwecken wie die des Echo-Calls. Diese Funktion wurde durch folgenden Code in den Asterisk-Einstellungen der extensions.conf umgesetzt.

```

1 [default] (+)
2 ...
3 exten => 2600,          1, Set (CDR(hangupdirection)=A)
4 same =>                n, Answer ()

```

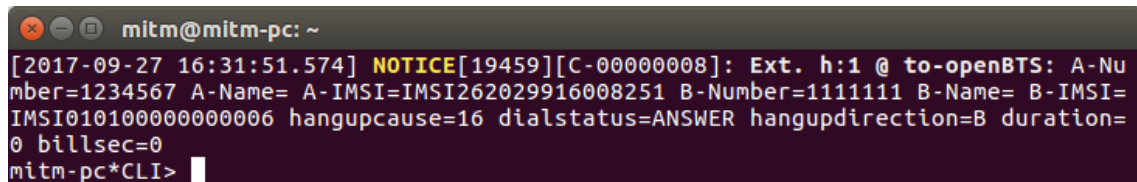



```

5 same =>          n,Echo()
6 same =>          n,Set(CDR(hangupdirection)=SYSTEM)
7 same =>          n,Hangup(16)
8 ...

```

Es dauerte eine Weile bis wir die richtigen Einstellungen für Asterisk gefunden hatten, sodass sowohl Anrufe zu Testnummern, als auch zwischen den Teilnehmern funktionierten. In Asterisk wurde nach einem Anruf eine Mitteilung ausgegeben, die folgendermaßen aussah und in der die IMSIs sowie Mobilfunknummern beider Teilnehmer aufgeführt wurden.



```

mitm@mitm-pc: ~
[2017-09-27 16:31:51.574] NOTICE[19459][C-00000008]: Ext. h:1 @ to-openBTS: A-Number=1234567 A-Name= A-IMSI=IMSI262029916008251 B-Number=1111111 B-Name= B-IMSI=IMSI010100000000006 hangupcause=16 dialstatus=ANSWER hangupdirection=B duration=0 billsec=0
mitm-pc*CLI>

```

Abbildung 21: Ausgabe in Asterisk nach Beenden eines Anrufs zwischen Mobilfunkteilnehmern

6. Umsetzung des Projektziels

Dieses Kapitel behandelt nun die konkrete Umsetzung des eigentlichen Projektziels "Man-In-The-Middle". Der Ansatz besteht darin die Gesprächsdaten auf der Strecke zwischen BTS, BSC und der Telefonvermittlungsanlage abzugreifen und zu speichern. Da alle Systeme softwareseitig auf einem PC ausgeführt werden und diese über Sockets miteinander kommunizieren, können die Daten überall abgegriffen werden.

Für die Installation und Einrichtung der benötigten Tools wurde ein Bash-Skript erstellt, welches die meisten Schritte automatisch durchführt. Die manuell noch auszuführenden Schritte werden in der Komandozeile ebenfalls ausgegeben (Skript siehe Anhang F).

6.1. Abspeichern der Daten

Mit Wireshark oder den beiden Komandozeilenanwendungen `tshark` und `tcpdump` können Daten von einem Interface analysiert werden und auch in eine `pcap`-Datei abgespeichert werden. Hierbei können bereits bei der Aufnahme Filter gesetzt werden, sodass nur die relevanten Daten gespeichert werden.

`Pcapsipdump` ist ein open-source Tool, welches auf `libpcap` basiert. Das Tool hört auf einem Interface den Netzwerkverkehr mit und speichert die SIP/RTP-Sessions als `pcap`-Datei ab. Diese Datei kann nun in `tcpdump`, Wireshark oder ähnlichem geöffnet, eingelesen und weiterverarbeitet werden. Eine hilfreiche Eigenschaft von `pcapsipdump` ist, dass das Tool pro Session selbstständig eine neue Datei anlegt. Das Tool läuft als Hintergrundprozess, sodass es nur einmal manuell gestartet werden muss. Alternativ kann `pcapsipdump` auch mit dem `systemd-Init`-Prozess automatisch gestartet werden, sofern man es nachträglich selbst konfiguriert. Da die Tools für das GSM-Netz, wie bereits erwähnt, alle auf demselben Rechner laufen, reicht es das Loopback-Interface abzuhören.

Vor der Installation des Programmes selbst muss Subversion und folgende Abhängigkeit installiert werden.

```
1 sudo apt-get install -y subversion libpcap-dev
```

Das Programm selbst vom SVN-Server herunterladen und installieren.

```
1 svn checkout https://svn.code.sf.net/p/pcapsipdump/code/trunk pcapsipdump-  
   code  
2 cd pcapsipdump-code  
3 sudo make  
4 sudo make install
```

Das Tool kann über das selbst erstellte `startingPcapsipdump.sh`-Skript oder alternativ auch manuell gestartet werden. Mit angegebenen Parametern lauscht es auf dem Loopback-Interface mit und speichert die Telefongespräche, sobald ein Anruf initiiert wird, in den angegebenen Ordner. Des Weiteren werden die Daten immer Paket gepuffert geschrieben, sodass die `pcap`-Datei immer konsistent ist.

```
1 sudo pcapsipdump -i lo -v 10 -d /home/all/wiresharkCalls/%Y%m%d-%H%M%S-%f-%  
   t-%i.pcap -U
```

6.2. Extrahieren und konvertieren der Daten

Zunächst wurden die von `pcapsipdump` extrahierten Daten mit Wireshark manuell analysiert. Darin sind nun nur noch die SIP- und RTP-Paket enthalten, wie in Abbildung 22 zu sehen. Wireshark erkennt den VOIP-Anruf und kombiniert die RTP-Packages korrekt. Allerdings konnte der Stream nicht direkt im Programm abgespielt werden. Der Grund hierfür ist, dass Wireshark gsm

nicht dekodieren kann. Jedoch gibt es einen Weg, wie die beiden Streams als .raw-Daten exportiert werden können. Hierfür wird ein beliebiges RTP-Packet ausgewählt. Über die Menüoption *Telefonie->RTP->Stream Analyse* kann der Stream analysiert werden. Nun kann der Hinweg und Rückweg als separate Datei gespeichert werden. Man muss jedoch als Datei-Typ .raw auswählen.

20170927-164237-30628-35366-f29af1b6-1e34-1236-b0a6-00265a686174.pcap

Anzeigefilter anwenden ... <Ctrl-/>

Ausdrucken...

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	SIP/SDP	684	Request: INVITE sip:35366@127.0.0.1:5060
2	0.000784	127.0.0.1	127.0.0.1	SIP	570	Status: 100 Trying
3	2.75274	127.0.0.1	127.0.0.1	SIP	506	Status: 180 Ringing
4	3.999479	127.0.0.1	127.0.0.1	SIP/SDP	824	Status: 200 OK
5	5.999772	127.0.0.1	127.0.0.1	SIP	353	Request: ACK sip:35366@127.0.0.1:5060
6	6.016068	127.0.0.1	127.0.0.1	RTP	87	PT=GSM 06.10, SSRC=0x16E55A6E, Seq=26163, Time=1077569409, Mark
7	4.016848	127.0.0.1	127.0.0.1	RTP	87	PT=GSM 06.10, SSRC=0x2296DD01, Seq=9545, Time=115475390
8	4.016862	127.0.0.1	127.0.0.1	RTCP	110	Sender Report Source description
9	4.934508	127.0.0.1	127.0.0.1	RTP	87	PT=GSM 06.10, SSRC=0x16E55A6E, Seq=26164, Time=1077569569
10	4.041646	127.0.0.1	127.0.0.1	RTP	87	PT=GSM 06.10, SSRC=0x2296DD01, Seq=9546, Time=115475550
11	4.058700	127.0.0.1	127.0.0.1	RTP	87	PT=GSM 06.10, SSRC=0x16E55A6E, Seq=26165, Time=1077569729
12	4.060112	127.0.0.1	127.0.0.1	RTP	87	PT=GSM 06.10, SSRC=0x2296DD01, Seq=9547, Time=1154755100
13	4.077254	127.0.0.1	127.0.0.1	RTP	87	PT=GSM 06.10, SSRC=0x16E55A6E, Seq=26166, Time=1077569889

> Frame 1: 684 bytes on wire (5472 bits), 684 bytes captured (5472 bits)

> Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> User Datagram Protocol, Src Port: 5069, Dst Port: 5060

> Session Initiation Protocol (INVITE)

0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00E.
0010 02 9e 91 ce 00 00 40 11 e8 7e 7f 00 00 01 7f 000..?
0020 00 01 13 cd 13 c4 02 8a 00 9e 49 4e 56 49 54 45INVITE
0030 20 73 69 70 3a 33 35 33 36 36 40 31 32 37 2e 30 sip:353 66@127.0
0040 2e 30 2e 31 3a 35 30 36 30 20 53 49 50 2f 32 2e .0.1:506 0 SIP/2.
0050 30 0d 0a 56 69 01 3a 20 53 49 50 2f 32 2e 30 2f 0.Via: SIP/2.0/
0060 55 44 50 20 31 32 37 2e 30 2e 30 2e 31 3a 35 30 UDP 127. 0.0.1:50
0070 36 39 3b 72 70 6f 74 3b 62 72 61 66 63 68 3d 69;rport;branch=

Pakete: 2874 · Anzeigzeit: 2874 (100.0%) · Ladezeit: 0:0:10 Profil:Default

Abbildung 22: Extrahierte SIP- & RTP-Pakete

Die .raw-Dateien können nun via folgendem Komandozeilenaufruf abgespielt werden.

```
1 padsp play -t gsm -r 8000 -c 1 example.gsm
```

Mit dem universellen Audiokonverter SoX können die Dateien in .wav convertiert werden, sodass diese auch mit jedem herkömmlichen Media Player abgespielt werden können.

```
1 sox -t gsm -r 8000 -c 1 example.raw exampleConverted.wav
```

Mit dem Bash-Skript pcap2wav von Git <https://gist.github.com/avimar/d2e9d05e082ce273962d742eb9acac16> können genau diese Schritte automatisiert durchgeführt werden.

6.3. Automatisierung

Das Abspeichern der Daten funktioniert bereits voll automatisiert und jeweils in eine extra Datei pro Session. Allerdings sollte das Konvertierungs-Skript noch automatisch ausgeführt werden. Hierfür kann das Linux-Tool *Incron* genutzt werden. Das Tool setzt auf das Kernel-Subsystem Inotify, um auf Dateisystem-Ereignisse zu reagieren. Dadurch kann ein Ordner überwacht werden und beim Erstellen einer neuen Datei die Ausführung des Konvertierungs-Skriptes getriggert werden. Incron ähnelt dabei in der Handhabung an das Standardwerkzeug *Cron*, welches Cron Jobs auf Basis von Zeitpunkten startet.

```
1 sudo apt-get install -y incron
```

Damit das Programm gestartet und konfiguriert werden kann, muss in der Datei `/etc/incron.allow` der Username eingetragen werden. Danach kann der Service gestartet werden.

```
1 systemctl start incron.service
```

Analog zu *Cron* werden über *incrontab* -e die Jobs angelegt und verwaltet.

```
1 /home/all/wiresharkCalls IN_CLOSE_WRITE /home/all/
  startPcap2wavqsmConversion.sh $@ $#
```

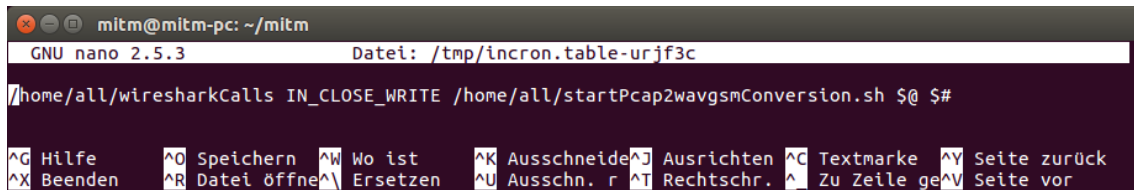


Abbildung 23: Zeile des incrontab

Der Eintrag besteht aus dem zu beobachtenden Verzeichnis-Pfad, dem auftretendem Ereignis und zuletzt dem auszuführenden Befehl beziehungsweise Skript. Über den Parameter `$@` wird das beobachtete Verzeichnis und mit `$#` der Name der Datei, welches das Ereignis getriggert hat, übergeben. Das Konvertierungs-Skript von 6.2 soll ausschließlich nach der Erstellung der Datei ausgeführt werden. Dazu wurde das Ereignis `IN_CLOSE_WRITE` genutzt, das in Abbildung 23 zu sehen ist. Sobald die Datei vollständig erstellt und geschlossen wurde, wird das auszuführende Skript (siehe Anhang G) getriggert. Dieses übergibt dem Konvertierungs-Skript von 6.2 die passenden Parameter und führt dieses aus.

Über den Status kann überprüft werden, ob ein Ereignis ausgelöst und das Skript getriggert wurde.

```
1 systemctl status incron.service
```

Nun werden also die Daten direkt von der Schnittstelle abgegriffen, gefiltert und gespeichert. Danach automatisch in `.wav` konvertiert, sodass die Gespräche lokal auf dem PC angehört werden können. Damit wäre das Projektziel bereits erreicht beziehungsweise mit der Konvertierung auch praktisch erweitert. Um nicht an den lokalen PC gebunden zu sein, wäre es nun auch möglich die Dateien über einen Webserver global zur Verfügung zu stellen.

6.4. Feature - Abhören der Aufnahme

Es soll das letzte oder die letzten Gespräche via einem Telefonanruf wiedergegeben werden können. Dies wurde zuerst mit einer, anschließend für mehrere speziell konfigurierte Nummern ermöglicht. Durch Änderungen und Erweiterungen in der der Konfigurationsdatei `extensions.conf` von Asterisk konnte jeweils eine Rufnummer hinterlegt werden, unter der ein Gespräch bereitgestellt wird. Das Abspielen einer Datei erfolgt durch die Methode *Playback*.

Da Asterisk kein `.wav` im GSM-Netz unterstützt, musste das Gespräch zusätzlich in `.gsm` konvertiert werden. Aus diesem Grund wurde das original Skript von 6.2 angepasst, sodass mit diesem beide Konvertierungen möglich waren.

Die Realisierung hierfür unterscheidet sich jedoch stark von dem verwendeten Architekturansatz. Dies liegt an den unterschiedlichen Asterisk-Versionen, die verwendet wurden. Ein Tausch beider Konfigurationen schlug fehl. Nachfolgend sind beide Ansätze aufgelistet:

Konfigurationsdatei für OsmoNITB-Ansatz

```
1 [gsmsubscriber]
2 exten=> 000001,1,Progress()
3 exten=> 000001,n,Wait(1)
4 exten=> 000001,n,Playback(/home/all/gsmCalls/recordedCall1.gsm_mixed)
5 exten=> 000001,n,Hangup()
6
7 exten=> 000002,1,Progress()
8 exten=> 000002,n,Wait(1)
9 exten=> 000002,n,Playback(/home/all/gsmCalls/recordedCall2.gsm_mixed)
10 exten=> 000002,n,Hangup()
11
```

```

12 exten=> 000003,1,Progress()
13 exten=> 000003,n,Wait(1)
14 exten=> 000003,n,Playback(/home/all/gsmCalls/recordedCall3.gsm_mixed)
15 exten=> 000003,n,Hangup()
16
17 exten=> 000004,1,Progress()
18 exten=> 000004,n,Wait(1)
19 exten=> 000004,n,Playback(/home/all/gsmCalls/recordedCall4.gsm_mixed)
20 exten=> 000004,n,Hangup()
21
22 exten=> 000005,1,Progress()
23 exten=> 000005,n,Wait(1)
24 exten=> 000005,n,Playback(/home/all/gsmCalls/recordedCall5.gsm_mixed)
25 exten=> 000005,n,Hangup()
26
27 exten=>_XXXXX,1,Dial(SIP/GSM/${EXTEN})
28 exten=>_XXXXX,n,HangUp

```

Insgesamt sind fünf Rufnummern (000001 - 000005) hinterlegt, sodass die letzten fünf Telefonate angehört werden können. Sie repräsentieren jeweils ein Mobiltelefon, das mit einer dieser Nummern verknüpft wurde. Diese Verknüpfung wird in der Datei Blots"hergestellt. Hier wird die IMSI einer Nummer zugeordnet. Nach Aufzeichnen eines Telefongesprächs und erneutem Anruf wird die erzeugte .gsm/wav Datei, des jeweiligen Slots überschrieben.

Konfigurationsdatei für OpenBTS-Ansatz

```

1 [default](+)
2 exten => 000001, 1, Set(CDR(hangupdirection)=A)
3 same => n, Answer()
4 same => n, GotoIf("${CDR(A-IMSI)}"!="IMSI01010000000006"?BUSY)
5 same => n, Playback(/home/all/gsmCalls/recordedCall1.gsm_mixed)
6 same => n, Set(CDR(hangupdirection)=SYSTEM)
7 same => n, Hangup(16)
8
9 same => n(BUSY, Set(CDR(hangupdirection)=SYSTEM)
10 same => n, Busy(5)
11
12 exten => 000002, 1, Set(CDR(hangupdirection)=A)
13 same => n, Answer()
14 same => n, GotoIf("${CDR(A-IMSI)}"!="IMSI01010000000006"?BUSY)
15 same => n, Playback(/home/all/gsmCalls/recordedCall2.gsm_mixed)
16 same => n, Set(CDR(hangupdirection)=SYSTEM)
17 same => n, Hangup(16)
18
19 same => n(BUSY, Set(CDR(hangupdirection)=SYSTEM)
20 same => n, Busy(5)
21
22 exten => 000003, 1, Set(CDR(hangupdirection)=A)
23 same => n, Answer()
24 same => n, GotoIf("${CDR(A-IMSI)}"!="IMSI01010000000006"?BUSY)
25 same => n, Playback(/home/all/gsmCalls/recordedCall3.gsm_mixed)
26 same => n, Set(CDR(hangupdirection)=SYSTEM)
27 same => n, Hangup(16)
28
29 same => n(BUSY, Set(CDR(hangupdirection)=SYSTEM)
30 same => n, Busy(5)
31
32 exten => 000004, 1, Set(CDR(hangupdirection)=A)
33 same => n, Answer()
34 same => n, GotoIf("${CDR(A-IMSI)}"!="IMSI01010000000006"?BUSY)
35 same => n, Playback(/home/all/gsmCalls/recordedCall4.gsm_mixed)
36 same => n, Set(CDR(hangupdirection)=SYSTEM)
37 same => n, Hangup(16)

```

```

38
39 same =>          n(BUSY, Set(CDR(hangupdirection)=SYSTEM)
40 same =>          n, Busy(5)
41
42 exten => 000005,   1, Set(CDR(hangupdirection)=A)
43 same =>          n, Answer()
44 same =>          n, GotoIf($["${CDR(A-IMSI)}"!="IMSI010100000000006"]?BUSY)
45 same =>          n, Playback(/home/all/gsmCalls/recordedCall5.gsm_mixed)
46 same =>          n, Set(CDR(hangupdirection)=SYSTEM)
47 same =>          n, Hangup(16)
48
49 same =>          n(BUSY, Set(CDR(hangupdirection)=SYSTEM)
50 same =>          n, Busy(5)

```

In diesem Ansatz wurde die IMSI eines Mobilgeräts direkt hinterlegt, sodass nur dieser die Gespräche abhören kann.

6.5. Aufgetretene Probleme

Vor der Installation des Osmo-sip-Connectors gab es zunächst Probleme beim Analysieren der Wireshark Daten, da die eigentlichen RTP-Pakete als UDP-Pakete erkannt wurden. Durch eine Einstellung in Wireshark lässt sich dies jedoch beheben. Dazu muss unter der Menüoption *Analyse->Protokolle aktivieren...* das Protokoll *rtp_udp* aktiviert werden. Nach neu laden des Files oder starten einer neuen Aufnahme werden die UDP-Pakete als RTP-Pakete erkannt. Allerdings haben nach wie vor die SIP-Pakete gefehlt, was wie beschrieben durch den Osmo-sip-Connector gelöst wurde.

Mit dem Tool in 6.2, welches für das Abspeichern der Daten zuständig ist, gab es Probleme. Denn es wurde kein Datei-Ereignis beziehungsweise erst nach sehr langer Verzögerung erkannt. Nach Lesen und Debuggen des Source-Codes, stellte sich heraus, dass das Tool die erstellte Datei sehr lange nicht schließt, obwohl bereits seit längerem die Session beendet ist. Der Übeltäter war ein Timer in der calltable-Klasse, welcher auf 5 Minute gestellt war. Nach Verändern des Source-Codes und Anpassen des Timers auf 5 Sekunden wurde auch die erstellte pcap-Datei kurz nach Ende der Session geschlossen und das Incron-Ereignis zum Start der Konvertierung getriggert.

```

211 ...
212 if (table[idx].is_used && (
213     (currttime - table[idx].last_packet_time > 5) ||
214     (currttime - table[idx].first_packet_time > opt_absolute_timeout))) {
215 ...

```

7. Ergebnisse

Insgesamt lässt sich sagen, dass die definierten Projektziele in vollem Umfang erreicht wurden. Die Inbetriebnahme des GSM Netzes wurde erfolgreich umgesetzt. Dabei wurden zwei unterschiedliche Architekturen parallel verfolgt, sodass letztendlich zwei verschiedene Netze in Betrieb genommen werden konnten. Aufgrund der geringen Einarbeitungszeit und der begrenzten Projektdauer konnte auf diese Weise sichergestellt werden, dass ein funktionierendes System zur weiteren Verfolgung des Projektzieles zur Verfügung stand. Des Weiteren wurde die Funktionalität des Abgreifens und Abspeicherns eines Telefongesprächs vollständig realisiert. Dabei werden die SIP/RTP Sessions durch das Tool `pcapsipdump` aufgezeichnet und im `.pcap` Format gespeichert. Anschließend wird die `.pcap` Datei in eine `.gsm` Datei und in das einfacher abspielbare `.wav` Format konvertiert. Dieses Vorgehen wurde schließlich automatisiert, sodass nach einem getätigten Anruf im Testnetz eine Aufnahme des Gesprächs lokal auf dem Rechner abgespielt werden kann.

Außerdem wurde im Projektziel ein optionales Feature formuliert, das ebenfalls vollständig umgesetzt werden konnte. Dieses umfasst die Hinterlegung einer bzw. mehrerer Rufnummern und das Abspielen des aufgezeichneten Telefongesprächs bei Anruf dieser Nummern. Es wurde pro Telefonnummer ein Slot erstellt, sodass immer die letzten Gespräche aller registrierten Teilnehmer abgehört werden können. Zusätzlich wurde nur eine festgelegte IMSI authentifiziert die Gespräche wieder abzuhören, sodass nicht jeder, wer jetzt die Nummern anruft, einfach die Gespräche abhören kann.

Beim Vergleich der beiden Architekturen lässt sich sagen, dass bei beiden zunächst einiges installiert und konfiguriert werden musste. Bei Osmocom gab es häufiger das Phänomen, dass beim Beenden eines Anrufs es länger dauerte bis der Anruf dann auch wirklich beendet wurde, sodass die Reaktionszeit bei OpenBTS geringer war. Zudem war die Sprachqualität ebenfalls bei OpenBTS besser bzw. klarer. Jedoch ist Osmocom mit den vielen einzelnen Bausteinen, welche auch der realen Welt entsprechen, flexibler, da bei OpenBTS vieles in dem Hauptmodul ist. Allerdings gibt es keinen klaren Gewinner oder Verlierer.

8. Fazit

Durch die effektive Zusammenarbeit sowie team-interne Aufgabenverteilung ergab sich ein rasches Vorankommen. Nichtsdestotrotz ergaben sich Probleme bei der Installation einzelner GSM Komponenten. Das Installieren und Konfigurieren zusätzlicher Abhängigkeiten erschwerte die Inbetriebnahme. So war anfänglich unklar, wie die Konfigurationsdateien von OpenBSC und OsmoBTS auf unsere Anforderungen angepasst werden mussten. Hinzu kamen Probleme beim Kompilieren dieser Dateien. Zur Umsetzung des Projektziels mussten zunächst Recherchen angestellt werden. Da die Vorgehensweise nicht immer eindeutig war, wurden zusätzliche Aufwände für das Ausprobieren mehrerer Möglichkeiten aufgebracht. Alles in allem wurde das Projektziel für alle Beteiligten zur vollkommenen Zufriedenheit zeitgerecht umgesetzt.

9. Projektaufteilung

Die Aufgaben zur Erreichung des Projektzieles wurden wie folgt verteilt:

- Inbetriebnahme des GSM Netzes (Attenberger, Bollenmiller, Schuster, Wilhelm)
- Realisierung der automatisierten Aufzeichnung und Abspeicherung eines Telefonats (Attenberger, Bollenmiller, Schuster, Wilhelm)

- Hinterlegung einer Rufnummer zur Anhörung des abgespeicherten Telefonats (Attenberger, Bollenmiller, Schuster, Wilhelm)
- Ausarbeitung und Dokumentation (Attenberger, Bollenmiller, Schuster, Wilhelm)

Das Projekt Man-In-The-Middle wurde präsentiert von der J3A Group. © J3A MITM

Appendix

A. osmo-bts.cfg

```

1 !
2 ! OsmoBTS () configuration saved from vty
3 !!
4 !
5 log stderr
6   logging color 1
7   logging timestamp 0
8   logging level rsl info
9   logging level oml info
10  logging level rll notice
11  logging level rr notice
12  logging level meas notice
13  logging level pag info
14  logging level llc info
15  logging level llp info
16  logging level dsp info
17  logging level abis notice
18 !
19 line vty
20   no login
21 !
22 phy 0
23   instance 0
24   osmotrx rx-gain 1
25   osmotrx ip local 127.0.0.1
26   osmotrx ip remote 127.0.0.1
27 bts 0
28   band PCS1900
29   ipa unit-id 1901 0
30   oml remote-ip 127.0.0.1
31   paging lifetime 0
32   gsmtap-sapi bcch
33   gsmtap-sapi ccch
34   gsmtap-sapi rach
35   gsmtap-sapi agch
36   gsmtap-sapi pch
37   gsmtap-sapi sdcch
38   gsmtap-sapi pacch
39   gsmtap-sapi pdtch
40   gsmtap-sapi sacch
41   trx 0
42   phy 0 instance 0

```

B. openbsc.cfg

```

1 !
2 ! OpenBSC configuration saved from vty
3 ! !
4 password foo
5 !
6 line vty
7 no login
8 !
9 e1\_input
10 e1\_line 0 driver ipa
11 e1\_line 0 port 0
12 network
13 network country code 262
14 mobile network code 99
15 short name mitm2
16 long name mitm2
17 auth policy accept-all
18 location updating reject cause 13
19 encryption a5 0
20 neci 1
21 paging any use tch 0
22 rrlp mode ms-based
23 mm info 1
24 handover 0
25 handover window rxlev averaging 10
26 handover window rxqual averaging 1
27 handover window rxlev neighbor averaging 10
28 handover power budget interval 6
29 handover power budget hysteresis 3
30 handover maximum distance 9999
31 timer t3101 10
32 timer t3113 60
33 timer t3122 10
34 dtx-used 0
35 subscriber-keep-in-ram 0
36 bts 0
37 type sysmobts
38 band PCS1900
39 cell\_identity 0
40 location\_area\_code 1
41 training\_sequence\_code 7
42 base\_station\_id\_code 63
43 ms max power 0
44 cell reselection hysteresis 4
45 rxlev access min 0
46 periodic location update 30
47 channel allocator descending
48 rach tx integer 9
49 rach max transmission 7
50 channel-description attach 1
51 channel-description bs-pa-mfrms 5
52 channel-description bs-ag-blks-res 1
53 ip.access unit\_id 1901 0
54 oml ip.access stream\_id 255 line 0
55 neighbor-list mode automatic
56 trx 0
57 rf\_locked 0
58 arfcn 806
59 nominal power 0
60 max\_power\_red 0
61 rsl e1 tei 0

```

```
62    timeslot 0
63      phys\_chan\_config CCCH+SDCCH4
64      hopping enabled 0
65    timeslot 1
66      phys\_chan\_config TCH/F
67      hopping enabled 0
68    timeslot 2
69      phys\_chan\_config TCH/F
70      hopping enabled 0
71    timeslot 3
72      phys\_chan\_config TCH/F
73      hopping enabled 0
74    timeslot 4
75      phys\_chan\_config TCH/F
76      hopping enabled 0
77    timeslot 5
78      phys\_chan\_config TCH/F
79      hopping enabled 0
80    timeslot 6
81      phys\_chan\_config TCH/F
82      hopping enabled 0
83    timeslot 7
84      phys\_chan\_config TCH/F
85      hopping enabled 0
```

C. Starten von OsmoBTS

```

mitm@netpc02: ~/mitm/3.Osmocom/Startskripte
<0000> rsl.c:2391 (bts=0,trx=0,ts=0,ss=0) Rx RSL BCCH_INFO
<0000> rsl.c:300 Rx RSL BCCH INFO (SI4, 23 bytes)
<0005> paging.c:523 Paging SI update
<0000> rsl.c:512 Rx RSL SACCH FILLING (SI5, 19 bytes)
<0005> paging.c:523 Paging SI update
<0000> rsl.c:516 Rx RSL Disabling SACCH FILLING (SI5bis)
<0005> paging.c:523 Paging SI update
<0000> rsl.c:516 Rx RSL Disabling SACCH FILLING (SI5ter)
<0005> paging.c:523 Paging SI update
<0000> rsl.c:512 Rx RSL SACCH FILLING (SI6, 13 bytes)
<0005> paging.c:523 Paging SI update
<0001> oml.c:973 OC=CHANNEL INST=(00,00,02) SET CHAN ATTR (TSC = 7)
<0006> scheduler.c:1355 Configuring multiframe with TCH/F+SACCH trx=0 ts=2
<0001> oml.c:335 OC=CHANNEL INST=(00,00,02) AVAIL STATE Dependency -> OK
<0001> oml.c:342 OC=CHANNEL INST=(00,00,02) OPER STATE Disabled -> Enabled
<0001> oml.c:303 OC=CHANNEL INST=(00,00,02) Tx STATE CHG REP
<0001> oml.c:973 OC=CHANNEL INST=(00,00,03) SET CHAN ATTR (TSC = 7)
<0006> scheduler.c:1355 Configuring multiframe with TCH/F+SACCH trx=0 ts=3
<0001> oml.c:335 OC=CHANNEL INST=(00,00,03) AVAIL STATE Dependency -> OK
<0001> oml.c:342 OC=CHANNEL INST=(00,00,03) OPER STATE Disabled -> Enabled
<0001> oml.c:303 OC=CHANNEL INST=(00,00,03) Tx STATE CHG REP
<0001> oml.c:973 OC=CHANNEL INST=(00,00,04) SET CHAN ATTR (TSC = 7)
<0006> scheduler.c:1355 Configuring multiframe with TCH/F+SACCH trx=0 ts=4
<0001> oml.c:335 OC=CHANNEL INST=(00,00,04) AVAIL STATE Dependency -> OK
<0001> oml.c:342 OC=CHANNEL INST=(00,00,04) OPER STATE Disabled -> Enabled
<0001> oml.c:303 OC=CHANNEL INST=(00,00,04) Tx STATE CHG REP
<0001> oml.c:973 OC=CHANNEL INST=(00,00,05) SET CHAN ATTR (TSC = 7)
<0006> scheduler.c:1355 Configuring multiframe with TCH/F+SACCH trx=0 ts=5
<0001> oml.c:335 OC=CHANNEL INST=(00,00,05) AVAIL STATE Dependency -> OK
<0001> oml.c:342 OC=CHANNEL INST=(00,00,05) OPER STATE Disabled -> Enabled
<0001> oml.c:303 OC=CHANNEL INST=(00,00,05) Tx STATE CHG REP
<0001> oml.c:973 OC=CHANNEL INST=(00,00,06) SET CHAN ATTR (TSC = 7)
<0006> scheduler.c:1355 Configuring multiframe with TCH/F+SACCH trx=0 ts=6
<0001> oml.c:335 OC=CHANNEL INST=(00,00,06) AVAIL STATE Dependency -> OK
<0001> oml.c:342 OC=CHANNEL INST=(00,00,06) OPER STATE Disabled -> Enabled
<0001> oml.c:303 OC=CHANNEL INST=(00,00,06) Tx STATE CHG REP
<0001> oml.c:973 OC=CHANNEL INST=(00,00,07) SET CHAN ATTR (TSC = 7)
<0006> scheduler.c:1355 Configuring multiframe with TCH/F+SACCH trx=0 ts=7
<0001> oml.c:335 OC=CHANNEL INST=(00,00,07) AVAIL STATE Dependency -> OK
<0001> oml.c:342 OC=CHANNEL INST=(00,00,07) OPER STATE Disabled -> Enabled
<0001> oml.c:303 OC=CHANNEL INST=(00,00,07) Tx STATE CHG REP
<000b> trx_if.c:170 No response from transceiver for phy0.0
<000b> trx_if.c:170 No response from transceiver for phy0.0
<000b> trx_if.c:170 No response from transceiver for phy0.0
<000b> trx_if.c:170 No response from transceiver for phy0.0
<000b> trx_if.c:170 No response from transceiver for phy0.0

```

Abbildung 24: Ansicht der Konsole nach dem Start der OsmoBTS

D. Quellcode der GUI-Anwendung

Inhalt von main.cpp

```

1  #include "mitm.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      MitM w;
8      w.show();
9
10     return a.exec();
11 }
```

Inhalt von mitm.h

```

1  #ifndef MITM_H
2  #define MITM_H
3
4  #include <QMainWindow>
5  #include <QProcess>
6
7  namespace Ui {
8  class MitM;
9  }
10
11 class MitM : public QMainWindow
12 {
13     Q_OBJECT
14
15 public:
16     explicit MitM(QWidget *parent = 0);
17     ~MitM();
18
19 private:
20     Ui::MitM *ui;
21     QString password;
22
23 public slots:
24     void checkInput();
25     void transceiverPressed();
26     void btsPressed();
27     void bscPressed();
28     void wiresharkPressed();
29     void sipConnectorPressed();
30     void sqlBrowserPressed();
31     void pcapSipDumpPressed();
32     void slot1Pressed();
33     void slot2Pressed();
34     void slot3Pressed();
35     void slot4Pressed();
36     void slot5Pressed();
37 };
38 #endif // MITM_H
```

Inhalt von mitm.cpp

```

1  #include "mitm.h"
2  #include "ui_mitm.h"
3
4  MitM::MitM(QWidget *parent) :
5      QMainWindow(parent),
6      ui(new Ui::MitM)
7  {
8      ui->setupUi(this);
9
10     connect(ui->lineEdit, SIGNAL(textEdited(QString)), this, SLOT(
        checkInput()));
11     connect(ui->pushButtonTransceiver, SIGNAL(clicked()), this, SLOT(
        transceiverPressed()));
12     connect(ui->pushButtonBSC, SIGNAL(clicked()), this, SLOT(bscPressed()));
13     ;
14     connect(ui->pushButtonBTS, SIGNAL(clicked()), this, SLOT(btsPressed()));
15     ;
16     connect(ui->pushButtonSipConnector, SIGNAL(clicked()), this, SLOT(
        sipConnectorPressed()));
17     connect(ui->pushButtonWireshark, SIGNAL(clicked()), this, SLOT(
        wiresharkPressed()));
18     connect(ui->pushButtonSQLBrowser, SIGNAL(clicked()), this, SLOT(
        sqlBrowserPressed()));
19     connect(ui->pushButtonPcap, SIGNAL(clicked()), this, SLOT(
        pcapSipDumpPressed()));
20     connect(ui->pushButtonSlot1, SIGNAL(clicked()), this, SLOT(slot1Pressed(
        )));
21     connect(ui->pushButtonSlot2, SIGNAL(clicked()), this, SLOT(slot2Pressed(
        )));
22     connect(ui->pushButtonSlot3, SIGNAL(clicked()), this, SLOT(slot3Pressed(
        )));
23     connect(ui->pushButtonSlot4, SIGNAL(clicked()), this, SLOT(slot4Pressed(
        )));
24     connect(ui->pushButtonSlot5, SIGNAL(clicked()), this, SLOT(slot5Pressed(
        )));
25 }
26
27 MitM::~MitM()
28 {
29     delete ui;
30 }
31
32 void MitM::checkInput() {
33     password = ui->lineEdit->text();
34 }
35
36
37 void MitM::transceiverPressed() {
38
39     QProcess* exec = new QProcess(this);
40
41     if (ui->pushButtonTransceiver->isChecked()) {
42         exec->start("./startTransceiver.sh");
43         ui->pushButtonTransceiver->setText("Stoppen");
44     }
45     else {
46         exec->close();
47         ui->pushButtonTransceiver->setText("Starten");
48     }
49 }

```

```

50
51
52
53 void MitM::bscPressed() {
54
55     QProcess* exec = new QProcess(this);
56
57     if (ui->pushButtonBSC->isChecked()) {
58         exec->start("./startOsmoNitb.sh");
59         ui->pushButtonBSC->setText("Stoppen");
60     }
61     else {
62         exec->close();
63         ui->pushButtonBSC->setText("Starten");
64     }
65 }
66
67
68 void MitM::btsPressed() {
69
70     QProcess* exec = new QProcess(this);
71
72     if (ui->pushButtonBTS->isChecked()) {
73         exec->start("./startOsmoBTS.sh");
74         ui->pushButtonBTS->setText("Stoppen");
75     }
76     else {
77         exec->close();
78         ui->pushButtonBTS->setText("Starten");
79     }
80 }
81
82
83
84 void MitM::wiresharkPressed() {
85
86     QProcess* exec = new QProcess(this);
87
88     if (ui->pushButtonWireshark->isChecked()) {
89         exec->start("./wireshark.sh");
90         ui->pushButtonWireshark->setText("Stoppen");
91     }
92     else {
93         exec->close();
94         ui->pushButtonWireshark->setText("Starten");
95     }
96 }
97
98
99
100 void MitM::sipConnectorPressed() {
101
102     QProcess* exec = new QProcess(this);
103
104     if (ui->pushButtonSipConnector->isChecked()) {
105         exec->start("./startSipConnector.sh");
106         ui->pushButtonSipConnector->setText("Stoppen");
107     }
108     else {
109         exec->close();
110         ui->pushButtonSipConnector->setText("Starten");
111     }
112 }

```



```

113
114
115
116 void MitM::sqlBrowserPressed() {
117
118     QProcess* exec = new QProcess(this);
119
120     if (ui->pushButtonSQLBrowser->isChecked()) {
121         exec->start("./startSQLBrowser.sh");
122         ui->pushButtonSQLBrowser->setText("Stoppen");
123     }
124     else {
125         exec->close();
126         ui->pushButtonSQLBrowser->setText("Starten");
127     }
128 }
129
130
131
132 void MitM::pcapSipDumpPressed() {
133
134     QProcess* exec = new QProcess(this);
135
136     if (ui->pushButtonPcap->isChecked()) {
137         exec->start("./startPcapSipDump.sh");
138         ui->pushButtonPcap->setText("Stoppen");
139     }
140     else {
141         exec->close();
142         ui->pushButtonPcap->setText("Starten");
143     }
144 }
145
146
147 void MitM::slot1Pressed() {
148     QProcess* exec = new QProcess(this);
149     exec->start("./slot1.sh");
150 }
151
152 void MitM::slot2Pressed() {
153     QProcess* exec = new QProcess(this);
154     exec->start("./slot2.sh");
155 }
156
157 void MitM::slot3Pressed() {
158     QProcess* exec = new QProcess(this);
159     exec->start("./slot3.sh");
160 }
161
162 void MitM::slot4Pressed() {
163     QProcess* exec = new QProcess(this);
164     exec->start("./slot4.sh");
165 }
166
167 void MitM::slot5Pressed() {
168     QProcess* exec = new QProcess(this);
169     exec->start("./slot5.sh");
170 }

```

E. Starten von OpenBTS

```

mitm@mitm-pc: ~/OpenBTS
mitm@mitm-pc:~/OpenBTS$ sudo ./OpenBTS
ALERT 6973:6973 2017-09-28T12:47:41.3 OpenBTS.cpp:595:main: OpenBTS (re)starting
, ver 5.0-master build date/time 2017-09-21T16:18:25
ALERT 6973:6973 2017-09-28T12:47:41.3 OpenBTS.cpp:596:main: OpenBTS reading conf
ig file /etc/OpenBTS/OpenBTS.db
1506595661.396800 139819790001984:

OpenBTS
Copyright 2008, 2009, 2010 Free Software Foundation, Inc.
Copyright 2010 Kestrel Signal Processing, Inc.
Copyright 2011, 2012, 2013, 2014 Range Networks, Inc.
Release 5.0-master+646bb6e79f CommonLibs:76b71d509b P formal build date 2017-09-
21T16:14:26
"OpenBTS" is a registered trademark of Range Networks, Inc.

Contributors:
  Range Networks, Inc.:
    David Burgess, Harvind Samra, Donald Kirker, Doug Brown,
    Pat Thompson, Kurtis Heimerl, Michael Iedema, Dave Gotwisner
  Kestrel Signal Processing, Inc.:
    David Burgess, Harvind Samra, Raffi Sevlia, Roshan Baliga
  GNU Radio:
    Johnathan Corgan
  Others:
    Anne Kwong, Jacob Appelbaum, Joshua Lackey, Alon Levy
    Alexander Chemeris, Alberto Escudero-Pascual
Incorporated L/GPL libraries and components:
  libortp, LGPL, 2.1 Copyright 2001 Simon MORLAT simon.morlat@linphone.org
  libusb, LGPL 2.1, various copyright holders, www.libusb.org
  libzmq, LGPL 3:
    Copyright (c) 2009-2011 250bpm s.r.o.
    Copyright (c) 2011 Botond Ballo
    Copyright (c) 2007-2009 iMatix Corporation
Incorporated BSD/MIT-style libraries and components:
  A5/1 Pedagogical Implementation, Simplified BSD License, Copyright 1998-1999 M
arc Briceno, Ian Goldberg, and David Wagner
  JsonBox, Copyright 2011 Anhero Inc.
  Google Core Dumper, BSD 3-Clause License, Copyright (c) 2005-2007, Google Inc.
Incorporated public domain libraries and components:
  sqlite3, released to public domain 15 Sept 2001, www.sqlite.org

This program comes with ABSOLUTELY NO WARRANTY.

Use of this software may be subject to other legal restrictions,
including patent licensing 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.

Note to US Government Users:
The OpenBTS software applications and associated documentation are "Commercial
Item(s)," as that term is defined at 48 C.F.R. Section 2.101, consisting of
"Commercial Computer Software" and "Commercial Computer Software Documentation,
"
as such terms are used in 48 C.F.R. 12.212 or 48 C.F.R. 227.7202, as
applicable. Consistent with 48 C.F.R. 12.212 or 48 C.F.R. Sections 227.7202-1
through 227.7202-4, as applicable, the Commercial Computer Software and
Commercial Computer Software Documentation are being licensed to U.S. Governmen
t
end users (a) only as Commercial Items and (b) with only those rights as are
granted to all other end users pursuant to the terms and conditions of
Range Networks' software licenses and master customer agreement.

1506595661.492904 139819790001984:
Starting the system...
ALERT 6973:6980 2017-09-28T12:47:46.5 OpenBTS.cpp:174:startTransceiver: starting
transceiver ./transceiver with 1 ARFCNs
linux; GNU C++ version 5.3.1 20151219; Boost_105800; UHD_003.009.002-0-unknown

Using internal frequency reference
-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
-- Detecting internal GPSDO.... Found an internal GPSDO
-- Setting references to the internal GPSDO
1506595671.976180 139819790001984:
system ready

1506595671.976201 139819790001984:
use the OpenBTSCLI utility to access CLI

1506595671.976319 139819790001984: OpenBTSCLI network socket support for tcp:493
00

OpenBTS>

```

Abbildung 25: Ansicht der Konsole nach dem Start von OpenBTS

F. Installations- und Konfigurations-Skript

```

1  #!/bin/bash
2
3  HOMEPATH=/home/all
4
5  #check if script called with root privileges
6  if [ `id -u` != 0 ];then
7      echo "You have to start this script with root privileges"
8      exit 1
9  fi
10
11
12 echo ">>creating folder with access of all user"
13 sudo mkdir -p $HOMEPATH/wiresharkCalls
14 sudo mkdir -p $HOMEPATH/wavCalls
15 sudo mkdir -p $HOMEPATH/gsmCalls
16 echo ""
17
18 echo ">>extending permissions"
19 sudo chmod a=rwx $HOMEPATH
20 sudo chmod a=rwx $HOMEPATH/wiresharkCalls
21 sudo chmod a=rwx $HOMEPATH/wavCalls
22 sudo chmod a=rwx $HOMEPATH/gsmCalls
23 echo ""
24
25 echo ">>copying conversionScript"
26 cp startPcap2wavgsmConversion.sh $HOMEPATH/.
27 chmod a=rwx $HOMEPATH/startPcap2wavgsmConversion.sh
28 cp pcap2wavgsm.sh $HOMEPATH/.
29 chmod a=rwx $HOMEPATH/pcap2wavgsm.sh
30 cp slots $HOMEPATH/.
31 chmod a=rw $HOMEPATH/slots
32 echo ""
33
34 echo ">>checking dependencies "
35 if ! which "tshark" > /dev/null
36 then
37     sudo apt-get install -y tshark sox
38 fi
39 echo ""
40
41 echo ">>installing pcapsipdump"
42 if ! which "svn" > /dev/null
43 then
44     sudo apt-get install -y subversion
45 fi
46 sudo apt-get install -y libpcap-dev
47 svn checkout https://svn.code.sf.net/p/pcapsipdump/code/trunk pcapsipdump-
   code
48 cd pcapsipdump-code
49 sudo cp ../calltable.cpp .
50 sudo make
51 sudo make install
52 cd ..
53 echo ""
54
55 sudo chmod +x startingPcapsipdump.sh
56 sudo ./startingPcapsipdump.sh $HOMEPATH
57 echo ""
58
59 echo ">>installing incron if not already installed.."
60 if ! which "incron" > /dev/null

```

```
61 then
62     sudo apt-get install -y incron
63 fi
64 echo ""
65 echo ""
66
67 echo ">>YOU have to do that manually:"
68 echo ">>append your username into '/etc/incron.allow'"
69
70 echo ">>starting service with 'systemctl start incron.service'"
71
72 echo ">>add job with 'incrontab -e' and append following line:"
73 echo "/home/all/wiresharkCalls IN_CLOSE_WRITE /home/all/
    startPcap2wavgsmConversion.sh \${@} \${#}"
74 echo ""
```

G. Skript zum Starten der Konvertierung

```

1  #!/bin/bash
2
3  DATE=""
4  datum() {
5      DATE=`date "+%Y-%m-%d %H:%M:%S"`
6  }
7
8  #test if another script is running and wait
9  while test `ps -e | grep startPcap2wavgsm | wc -l` > 2
10 do
11     sleep 1
12 done
13
14 LOGFILE=/home/all/startPcap2wavgsmConversion.log
15 GSMPATH=/home/all/gsmCalls
16 WAVPATH=/home/all/wavCalls
17
18 datum
19 echo "$DATE script started..." >> $LOGFILE
20
21 #check dir and file
22 if [ ! -d "$1" ]; then
23     datum
24     echo "$DATE directories '$1' does not exist!" >> $LOGFILE
25 fi
26 if [ ! -f "$1/$2" ]; then
27     datum
28     echo "$DATE file '$2' does not exist!" >> $LOGFILE
29 fi
30
31 #read calling numbers out of filenames
32 number_A="$(cut -d'-' -f3 <<<"$2")"
33 number_B="$(cut -d'-' -f4 <<<"$2")"
34
35 #filter numbers of slots
36 if [ "$number_B" = "000001" ] || [ "$number_B" = "000002" ] || [ "$number_B"
    = "000003" ] || [ "$number_B" = "000004" ] || [ "$number_B" = "000005"
    ]; then
37     exit 1
38 fi
39
40 #exit if no slots
41 if [ ! -f "/home/all/slots" ]; then
42     datum
43     echo "$DATE No slots reserved!" >> $LOGFILE
44     exit 1
45 fi
46
47 #check if calling numbers have a slot and convert calls to wav/gsm
48 while read -r line
49 do
50     if [ "$number_A" = "$(cut -d'=' -f1 <<<"$line")" ] || [ "$number_B" = "
    $(cut -d'=' -f1 <<<"$line")" ]; then
51         datum
52         echo "$DATE converting pcap to wav..." >> $LOGFILE
53         datum
54         echo "$DATE /home/all/pcap2wavgsm.sh -z wav \"$1/$2\" \"$WAVPATH/
    recordedCall$(cut -d'=' -f2 <<<"$line").wav\" >> $LOGFILE
55         /home/all/pcap2wavgsm.sh -z wav "$1/$2" "$WAVPATH/recordedCall$(cut
    -d'=' -f2 <<<"$line").wav"
56         datum

```

```
57     echo "$DATE converting pcap to gsm..." >> $LOGFILE
58     datum
59     echo "$DATE /home/all/pcap2wavgsm.sh -z gsm \"$1/$2\" \"$GSMPATH/
recordedCall$(cut -d'=' -f2 <<<"$line").gsm\"" >> $LOGFILE
60     /home/all/pcap2wavgsm.sh -z gsm "$1/$2" "$GSMPATH/recordedCall$(cut
-d'=' -f2 <<<"$line").gsm"
61     datum
62     echo "$DATE script ended..." >> $LOGFILE
63     echo >> $LOGFILE
64 fi
65 done < /home/all/slots
```