

Mobile Netze

Mobile Netze - Man-In-The-Middle

Ausarbeitung
von
Attenberger, Bollenmiller, Schuster, Wilhelm
SS17 IG
September 28, 2017

Betreut von Prof. Dr. Wischhof

Contents

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	5
3.1. Aufbau und Zusammenspiel	5
3.1.1. Bestandteile	5
3.1.2. Datenbanken	7
4. Inbetriebnahme eines Osmocom Systems	9
4.1. Vorinstallationen	9
4.1.1. Ubuntu 16.04.3	9
4.1.2. Git	9
4.1.3. Softwarevoraussetzungen	9
4.1.4. Aktivierung der Verbindung zum USRP2	9
4.2. Installation einzelner GSM Komponenten	10
4.2.1. OsmoTRX	10
4.2.2. OsmoBTS	11
4.2.3. OsmoNitb unter OpenBSC	12
4.3. Starten des Systems	12
4.4. Installation weiterer Komponenten	14
4.4.1. Osmo-sip-Connector	14
4.4.2. Asterisk	15
4.5. Verwendung einer GUI	16
5. Inbetriebnahme eines OpenBTS Systems	27
5.1. Vorinstallationen	27
5.1.1. Ubuntu 16.04.3	27
5.1.2. Git	27
5.1.3. Softwarevoraussetzungen	27
5.1.4. Aktivierung der Verbindung zum USRP2	27
5.2. Installation einzelner GSM Komponenten	27
5.3. Starten des Systems	27
6. Umsetzung des Projektziels	28
6.1. pcapsipdump	29
6.2. Abspeichern der Daten	29
6.3. pcap2wavgsm	30
6.4. Extrahieren der Daten	30
6.5. Vollautomatisieren aller Schritte	30
6.6. incron	30
6.7. Weiteres Feature	31
7. Ergebnisse	32
8. Fazit	32
9. Projektaufteilung	33

A. openbsc.cfg	34
B. osmo-bts.cfg	36

List of Figures

1.	GSM Systemarchitektur mit OsmoNITB	3
2.	USRP N210 Software Defined Radio	5
3.	OpenBTS Systembestandteile	6
4.	OpenBTS System Diagramm	7
5.	Ansicht der Konsole nach dem Start des Transceivers	11
6.	Fehlermeldung bezüglich der Thread Priorität in osmoTRX	11
7.	Ansicht der Konsole nach dem Start der OsmoNitb	13
8.	Ansicht der Konsole nach dem Start der OsmoBTS	14
9.	Ansicht der Konsole nach dem Start des Osmo-sip-connectors	15
10.	Ansicht der Konsole nach Verbindung mit Asterisk	15
11.	GUI-Anwendung MitM	16

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 ab zugreifen 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. OsmoNITB übernimmt hierbei die Kommunikation mit der Basisstation. Dies geschieht über das A-bis-Interface, einer standardisierten Schnittstelle für den Datenaustausch zwischen Basisstation und Basisstation-Controller (Base Station Controller, BSC), welches selbst Bestandteil dieses Pakets ist. Neben diesen existieren noch folgende weitere Bestandteile:

- MSC/VLR
- SMSC
- EIR
- HLR//AUC

Figure 1: GSM Systemarchitektur mit OsmoNITB

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 eine 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

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, Teilnehmer-authentifizierung und Anrufvermittlung im eigenen Netz sowie, je nach Anbindung, zu VoIP- und Festnetzteilnehmern.

3.1. Aufbau und Zusammenspiel

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 2) über das Netzwerk mit allen genutzten Computern verbunden.



Figure 2: 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. Es verfügt über einen Shortcode-Handler, welcher es 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 3 dargestellt. Dabei repräsentieren eckige Boxen Hardware-Komponenten, abgerundete Boxen Software-Komponenten.

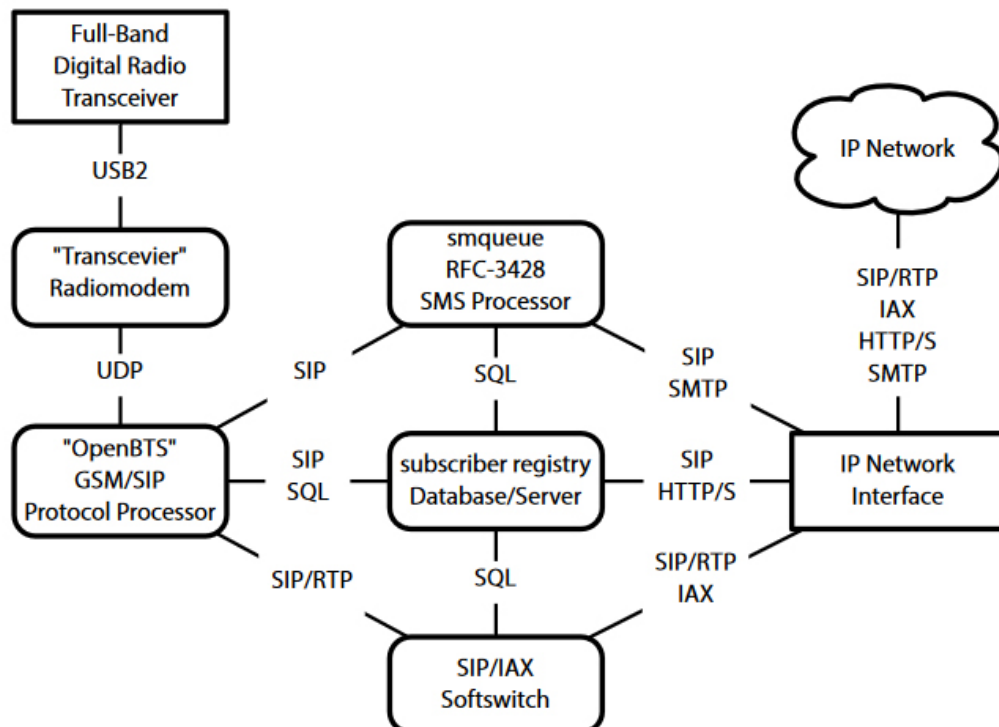


Figure 3: OpenBTS Systembestandteile

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:

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.

Der Kommunikationsfluss der OpenBTS-Komponenten über die Datenbanken ist in Abbildung 4 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.

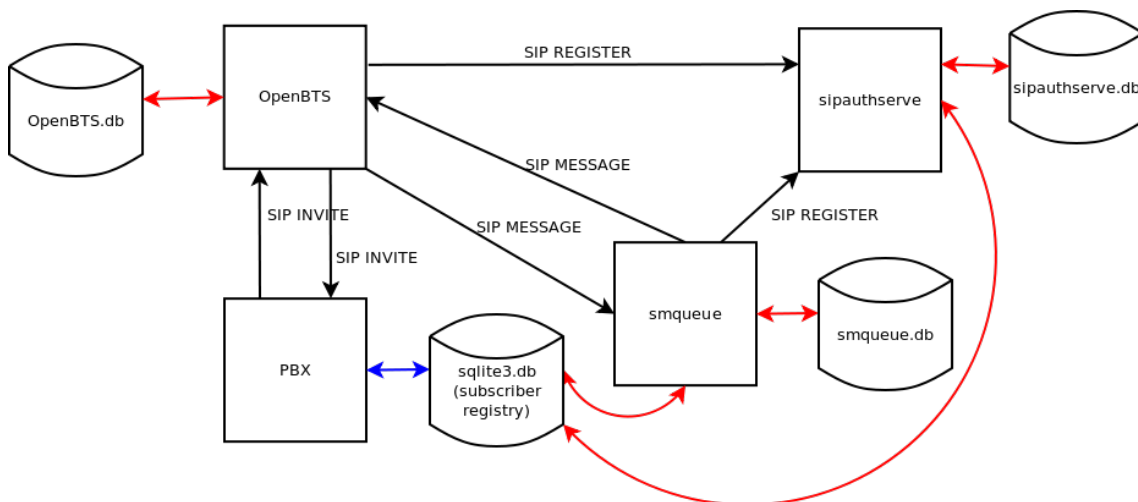


Figure 4: OpenBTS System Diagramm

3.2.

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

4.1.1. Ubuntu 16.04.3

Zunächst wurde das Betriebssystem Ubuntu 16.04.3 auf einem Labor-Rechner installiert und eingerichtet.

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-Modes 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* 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
```

```

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)

```

Figure 5: Ansicht der Konsole nach dem Start des Transceivers

Zur Behebung der Warnung, die in Abbildung 6 zu sehen ist, wurde die Priorität in der Datei `/etc/security/limits.conf` gesetzt.

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

```

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

```

Figure 6: Fehlermeldung bezüglich der Thread Priorität in osmoTRX

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
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 B). 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 A). *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 Frequenzuteilung 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.

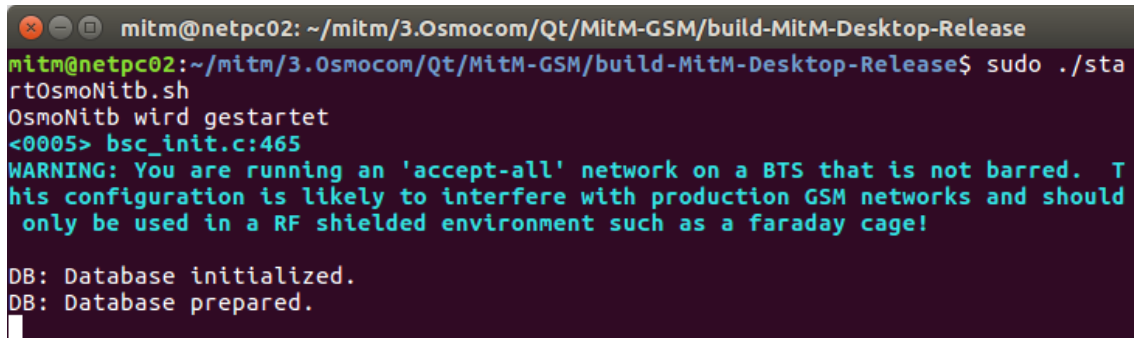
```

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
```

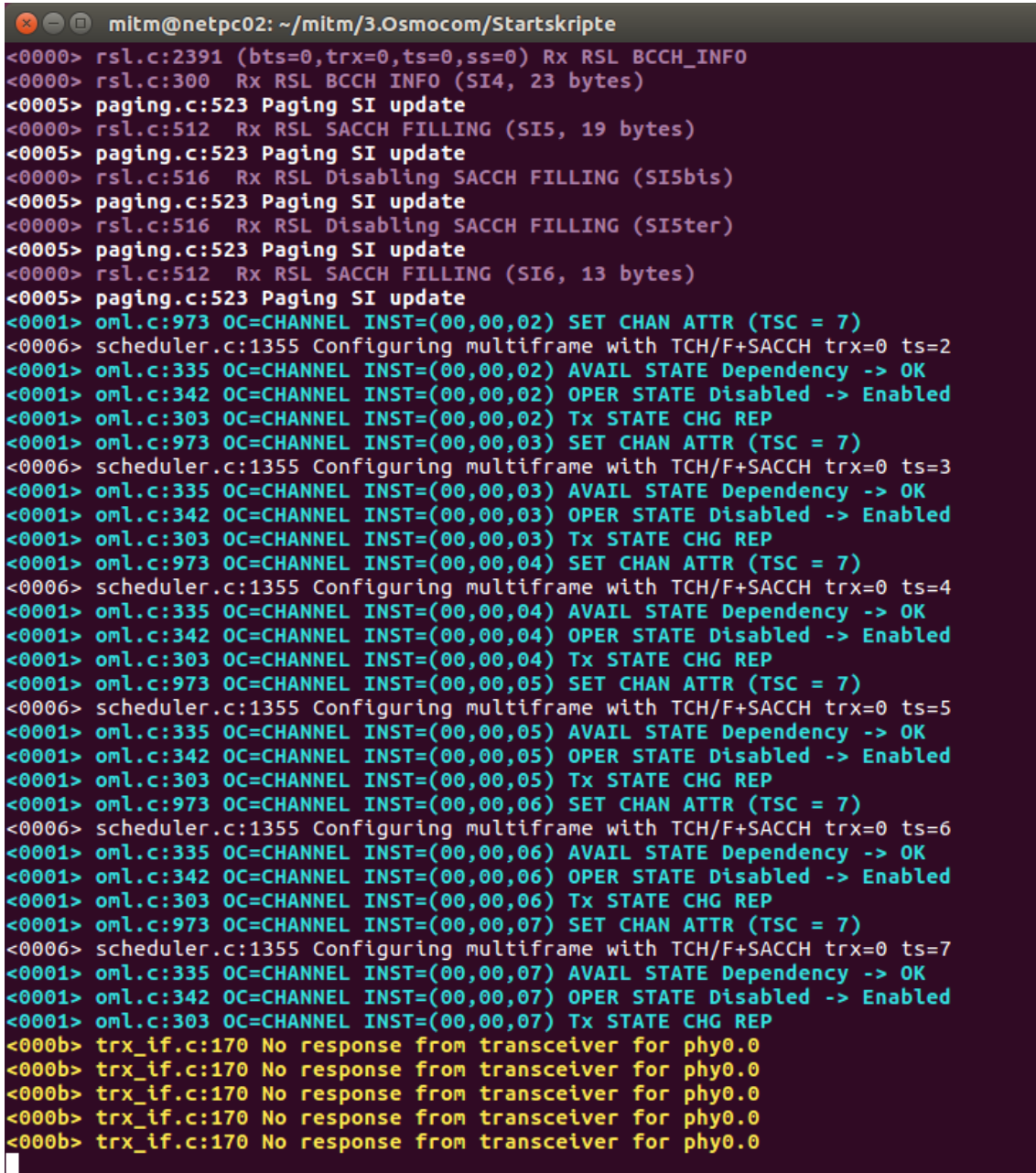


```
mitm@netpc02: ~/mitm/3.Osmocom/Qt/MitM-GSM/build-MitM-Desktop-Release
mitm@netpc02:~/mitm/3.Osmocom/Qt/MitM-GSM/build-MitM-Desktop-Release$ sudo ./startOsmoNitb.sh
OsmoNitb wird gestartet
<0005> bsc_init.c:465
WARNING: You are running an 'accept-all' network on a BTS that is not barred. This configuration 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.
```

Figure 7: 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
```



```
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
```

Figure 8: Ansicht der Konsole nach dem Start der OsmoBTS

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
```

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

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

```
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
```

Figure 9: Ansicht der Konsole nach dem Start des Osmo-sip-connectors

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.

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

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
```

Figure 10: 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.

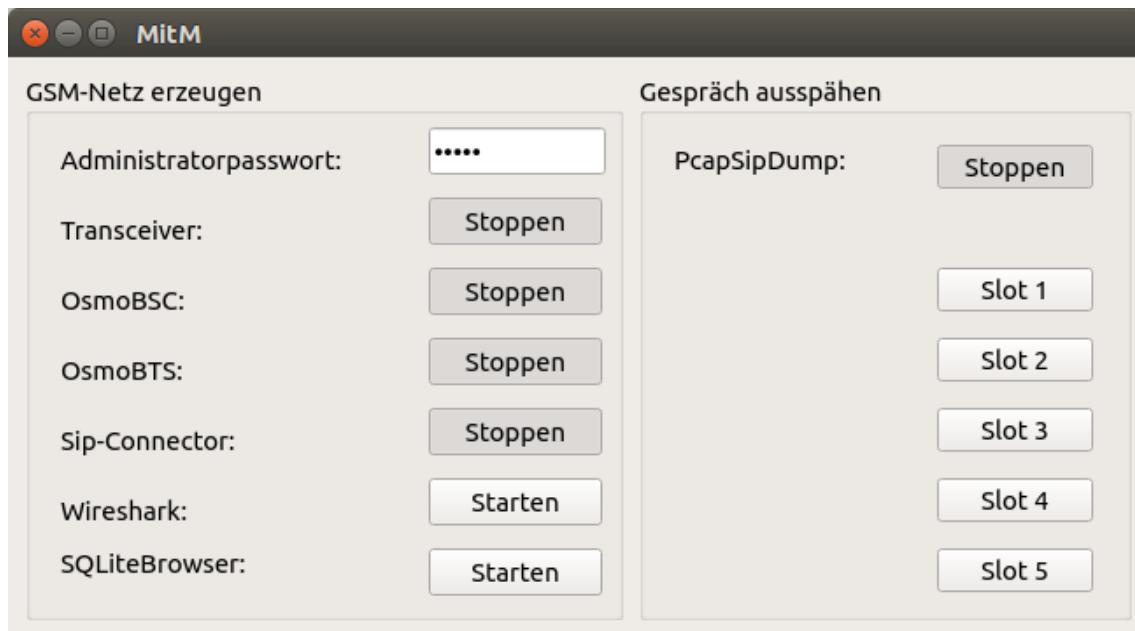


Figure 11: GUI-Anwendung MitM

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. Nachfolgend ist der Quellcode der GUI-Anwendung auf-

listet:

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(
11        checkInput()));
12    connect(ui->pushButtonTransceiver, SIGNAL(clicked()), this, SLOT(
13        transceiverPressed()));
14    connect(ui->pushButtonBSC, SIGNAL(clicked()), this, SLOT(bscPressed()))
15    ;
16    connect(ui->pushButtonBTS, SIGNAL(clicked()), this, SLOT(btsPressed()))
17    ;
18    connect(ui->pushButtonSipConnector, SIGNAL(clicked()), this, SLOT(
19        sipConnectorPressed()));
20    connect(ui->pushButtonWireshark, SIGNAL(clicked()), this, SLOT(
21        wiresharkPressed()));
22    connect(ui->pushButtonSQLBrowser, SIGNAL(clicked()), this, SLOT(
23        sqlBrowserPressed()));
24    connect(ui->pushButtonPcap, SIGNAL(clicked()), this, SLOT(
25        pcapSipDumpPressed()));
26    connect(ui->pushButtonSlot1, SIGNAL(clicked()), this, SLOT(slot1Pressed
27        ()));
28    connect(ui->pushButtonSlot2, SIGNAL(clicked()), this, SLOT(slot2Pressed
29        ()));
30    connect(ui->pushButtonSlot3, SIGNAL(clicked()), this, SLOT(slot3Pressed
31        ()));
32    connect(ui->pushButtonSlot4, SIGNAL(clicked()), this, SLOT(slot4Pressed
33        ()));
34    connect(ui->pushButtonSlot5, SIGNAL(clicked()), this, SLOT(slot5Pressed
35        ()));
36 }
37
38 MitM::~MitM()
39 {
40     delete ui;
41 }
42
43 void MitM::checkInput() {
44     password = ui->lineEdit->text();
45 }
46
47 void MitM::transceiverPressed() {
48
49     QProcess* exec = new QProcess(this);
50
51     if (ui->pushButtonTransceiver->isChecked()) {
52         exec->start("./startTransceiver.sh");
53         ui->pushButtonTransceiver->setText("Stoppen");
54     }
55     else {
56         exec->close();
57         ui->pushButtonTransceiver->setText("Starten");
58     }
59 }
60
61 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 }

```

Inhalt von mitm.ui

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">

```

```

3  <class>MitM</class>
4  <widget class="QMainWindow" name="MitM">
5    <property name="geometry">
6      <rect>
7        <x>0</x>
8        <y>0</y>
9        <width>648</width>
10       <height>330</height>
11     </rect>
12   </property>
13   <property name="windowTitle">
14     <string>MitM</string>
15   </property>
16   <widget class="QWidget" name="centralWidget">
17     <widget class="QGroupBox" name="groupBox">
18       <property name="geometry">
19         <rect>
20           <x>10</x>
21           <y>10</y>
22           <width>341</width>
23           <height>311</height>
24         </rect>
25       </property>
26       <property name="title">
27         <string>GSM-Netz erzeugen</string>
28       </property>
29       <widget class="QLabel" name="label">
30         <property name="geometry">
31           <rect>
32             <x>20</x>
33             <y>40</y>
34             <width>171</width>
35             <height>17</height>
36           </rect>
37         </property>
38         <property name="text">
39           <string>Administratorpasswort:</string>
40         </property>
41       </widget>
42       <widget class="QPushButton" name="pushButtonWireshark">
43         <property name="geometry">
44           <rect>
45             <x>230</x>
46             <y>230</y>
47             <width>99</width>
48             <height>27</height>
49           </rect>
50         </property>
51         <property name="text">
52           <string>Starten</string>
53         </property>
54         <property name="checkable">
55           <bool>true</bool>
56         </property>
57       </widget>
58       <widget class="QLabel" name="label_4">
59         <property name="geometry">
60           <rect>
61             <x>20</x>
62             <y>160</y>
63             <width>81</width>
64             <height>17</height>
65           </rect>

```



```

66     </property>
67     <property name="text">
68         <string>OsmoBTS:</string>
69     </property>
70 </widget>
71 <widget class="QLabel" name="label_6">
72     <property name="geometry">
73         <rect>
74             <x>20</x>
75             <y>240</y>
76             <width>91</width>
77             <height>17</height>
78         </rect>
79     </property>
80     <property name="text">
81         <string>Wireshark:</string>
82     </property>
83 </widget>
84 <widget class="QLabel" name="label_2">
85     <property name="geometry">
86         <rect>
87             <x>20</x>
88             <y>80</y>
89             <width>111</width>
90             <height>17</height>
91         </rect>
92     </property>
93     <property name="text">
94         <string>Transceiver:</string>
95     </property>
96 </widget>
97 <widget class="QLabel" name="label_3">
98     <property name="geometry">
99         <rect>
100             <x>20</x>
101             <y>120</y>
102             <width>91</width>
103             <height>17</height>
104         </rect>
105     </property>
106     <property name="text">
107         <string>OsmoBSC:</string>
108     </property>
109 </widget>
110 <widget class="QPushButton" name="pushButtonSipConnector">
111     <property name="geometry">
112         <rect>
113             <x>230</x>
114             <y>190</y>
115             <width>99</width>
116             <height>27</height>
117         </rect>
118     </property>
119     <property name="text">
120         <string>Starten</string>
121     </property>
122     <property name="checkable">
123         <bool>true</bool>
124     </property>
125 </widget>
126 <widget class="QPushButton" name="pushButtonSQLBrowser">
127     <property name="geometry">
128         <rect>

```

```

129         <x>230</x>
130         <y>270</y>
131         <width>99</width>
132         <height>27</height>
133     </rect>
134 </property>
135 <property name="text">
136     <string>Starten</string>
137 </property>
138 <property name="checkable">
139     <bool>true</bool>
140 </property>
141 </widget>
142 <widget class="QLineEdit" name="lineEdit">
143     <property name="geometry">
144         <rect>
145             <x>230</x>
146             <y>30</y>
147             <width>101</width>
148             <height>27</height>
149         </rect>
150     </property>
151     <property name="echoMode">
152         <enum>QLineEdit::Password</enum>
153     </property>
154 </widget>
155 <widget class="QPushButton" name="pushButtonTransceiver">
156     <property name="geometry">
157         <rect>
158             <x>230</x>
159             <y>70</y>
160             <width>99</width>
161             <height>27</height>
162         </rect>
163     </property>
164     <property name="text">
165         <string>Starten</string>
166     </property>
167     <property name="checkable">
168         <bool>true</bool>
169     </property>
170 </widget>
171 <widget class="QLabel" name="label_5">
172     <property name="geometry">
173         <rect>
174             <x>20</x>
175             <y>200</y>
176             <width>111</width>
177             <height>17</height>
178         </rect>
179     </property>
180     <property name="text">
181         <string>Sip-Connector:</string>
182     </property>
183 </widget>
184 <widget class="QPushButton" name="pushButtonBTS">
185     <property name="geometry">
186         <rect>
187             <x>230</x>
188             <y>150</y>
189             <width>99</width>
190             <height>27</height>
191         </rect>

```

```

192     </property>
193     <property name="text">
194         <string>Starten</string>
195     </property>
196     <property name="checkable">
197         <bool>true</bool>
198     </property>
199 </widget>
200 <widget class="QPushButton" name="pushButtonBSC">
201     <property name="geometry">
202         <rect>
203             <x>230</x>
204             <y>110</y>
205             <width>99</width>
206             <height>27</height>
207         </rect>
208     </property>
209     <property name="text">
210         <string>Starten</string>
211     </property>
212     <property name="checkable">
213         <bool>true</bool>
214     </property>
215 </widget>
216 <widget class="QLabel" name="label_7">
217     <property name="geometry">
218         <rect>
219             <x>20</x>
220             <y>270</y>
221             <width>121</width>
222             <height>17</height>
223         </rect>
224     </property>
225     <property name="text">
226         <string>SQLiteBrowser:</string>
227     </property>
228 </widget>
229 </widget>
230 <widget class="QGroupBox" name="groupBox_2">
231     <property name="geometry">
232         <rect>
233             <x>360</x>
234             <y>10</y>
235             <width>281</width>
236             <height>311</height>
237         </rect>
238     </property>
239     <property name="title">
240         <string>Gespr ch aussp hen</string>
241     </property>
242     <widget class="QLabel" name="label_8">
243         <property name="geometry">
244             <rect>
245                 <x>20</x>
246                 <y>40</y>
247                 <width>111</width>
248                 <height>17</height>
249             </rect>
250         </property>
251         <property name="text">
252             <string>PcapSipDump:</string>
253         </property>
254     </widget>

```

```

255 <widget class="QPushButton" name="pushButtonPcap">
256   <property name="geometry">
257     <rect>
258       <x>170</x>
259       <y>40</y>
260       <width>89</width>
261       <height>25</height>
262     </rect>
263   </property>
264   <property name="text">
265     <string>Starten</string>
266   </property>
267   <property name="checkable">
268     <bool>true</bool>
269   </property>
270 </widget>
271 <widget class="QPushButton" name="pushButtonSlot1">
272   <property name="geometry">
273     <rect>
274       <x>170</x>
275       <y>110</y>
276       <width>89</width>
277       <height>25</height>
278     </rect>
279   </property>
280   <property name="text">
281     <string>Slot 1</string>
282   </property>
283 </widget>
284 <widget class="QPushButton" name="pushButtonSlot2">
285   <property name="geometry">
286     <rect>
287       <x>170</x>
288       <y>150</y>
289       <width>89</width>
290       <height>25</height>
291     </rect>
292   </property>
293   <property name="text">
294     <string>Slot 2</string>
295   </property>
296 </widget>
297 <widget class="QPushButton" name="pushButtonSlot3">
298   <property name="geometry">
299     <rect>
300       <x>170</x>
301       <y>190</y>
302       <width>89</width>
303       <height>25</height>
304     </rect>
305   </property>
306   <property name="text">
307     <string>Slot 3</string>
308   </property>
309 </widget>
310 <widget class="QPushButton" name="pushButtonSlot4">
311   <property name="geometry">
312     <rect>
313       <x>170</x>
314       <y>230</y>
315       <width>89</width>
316       <height>25</height>
317     </rect>

```

```
318     </property>
319     <property name="text">
320         <string>Slot 4</string>
321     </property>
322 </widget>
323 <widget class="QPushButton" name="pushButtonSlot5">
324     <property name="geometry">
325         <rect>
326             <x>170</x>
327             <y>270</y>
328             <width>89</width>
329             <height>25</height>
330         </rect>
331     </property>
332     <property name="text">
333         <string>Slot 5</string>
334     </property>
335 </widget>
336 </widget>
337 </widget>
338 </widget>
339 <layoutdefault spacing="6" margin="11"/>
340 <resources/>
341 <connections/>
342 </ui>
```

5. Inbetriebnahme eines OpenBTS 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.

5.1. Vorinstallationen

5.1.1. 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.

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.

```
| sudo apt-get install git
```

5.1.3. Softwarevoraussetzungen

Um OpenBTS fehlerfrei installieren und in Betrieb nehmen zu können, wurden zunächst einige Bibliotheken und Pakete via Paketmanager installiert, um die vorausgesetzten Abhängigkeiten zu erfüllen:

```
| sudo apt-get install autoconf libtool libosip2-dev libortp-dev libusb-1.0-0-dev g++ sqlite3 libsqlite3-dev erlang libreadline6-dev libncurses5-dev
```

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.

```
| sudo ifconfig enp0s25 192.168.10.3
```

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

```
| 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 einzelner GSM Komponenten

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.

5.3. Starten des Systems

6. Umsetzung des Projektziels

Während es sich bei den beiden vorherigen Kapiteln um die Inbetriebnahme des GSM-Netzes selbst gehandelt haben, geht es nun um die konkrete Umsetzung des eigentlichen Projektziels "Man-In-The-Middle". Dabei wird versucht die Gesprächsdaten auf der Strecke zwischen BTS, BSC und PBX abzugreifen.

Zuerst versucht zwischen BTS und BSC?!? -> evtl Screenshot von Daten des ABIS-Interface?!

<https://osmocom.org/projects/osmo-sip-conector/wiki/Osmo-sip-connector> <http://ftp.osmocom.org/docs/latest/osmonusermanual.pdf> -> p.6/7

Abgreifen der Daten zwischen BTS und BSC (ABIS-Interface) schwierig -> Daten weiter analysiert und weitere Angriffspunkte ausmachen. -> Abgreifen der Daten vor der Telefonanlage (Asterisk (OPEN: IAX; Osmo: PBX)) -> dadurch die Daten im VOIP Format (SIP + RTP), welches via PC relativ gut zu handeln ist.

Zunächst eigenes analysieren der Daten. Suche nach praktischem Tool im Internet -> pcapspidump
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 ausgegeben.

Listing 1: Install and configure Script

```

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 pcapspidump"

```

```

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 $HOMEPAATH
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 ""

```

6.1. pcapsipdump

6.2. Abspeichern der Daten

Mit den beiden Komandozeilenanwendungen tshark und tcpdump können Daten von einem Interface in eine pcap-Datei abgespeichert werden. Hierbei können auch bereits schon beim aufnehmen Filter gesetzt werden, sodass nur die relevanten Daten gespeichert werden.

pcapsipdump ist open-source Tool, welches auf der libpcap basiert. Das Tool hört auf einem Interface die Daten 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. Das nette Feature dabei ist, dass das Tool selbstständig pro Session eine Datei anlegt. Das Tool läuft als Hintergrundprozess, sodass es nur einmal manuell gestartet werden muss. Alternativ kann das Tool auch mit dem systemd-Init-Prozess automatisch gestartet werden, sofern man es nachträglich selbst konfiguriert. Hören das Loopback-Interface ab -> da all Tools auf dem selben Rechner laufen und die Tools über diese Schnittstelle miteinander kommunizieren.

Abhängigkeiten des Programms installieren

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

Gestartet wird das Tool mit folgenden Parametern.

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


Im späteren Verlauf Probleme mit dem Tool, sodass letztendlich Source-Code angepasst wurde. Das Problem lag darin, dass das Tool die erstellte Datei lange nicht schließt, obwohl bereits seit längerem die Session beendet ist. Der Übeltäter war ein Timer in der callable-Klasse, welcher auf 5 Minute gestellt war. Nach Verändern des Timers auf 5 Sekunden wurde auch die erstellte pcap-Datei kurz nach Ende der Session geschlossen.

```

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         ...

```

6.3. pcap2wavgsm

6.4. Extrahieren der Daten

Zunächst wurden die von pcapsipdump extrahierten Daten mit Wireshark manuell analysiert. Darin sind nun wirklich nur noch die SIP- und RTP-Packet enthalten, wie auf fig XXXXX zu sehen. Mit Wireshark erkennt auch den VOIP-Anruf und kombiniert die RTP-Packages korrekt. Allerdings konnte der Stream nicht direkt im Programm abgespielt werden. Der Grund hierfür ist vermutlich, dass Wireshark gsm nicht dekodieren kann. Jedoch gibt es einen Weg, wie die beiden Streams als .raw-Daten exportiert werden können. Hierfür ein beliebiges RTP-Packet auswählen, über "Telefonie->RTP->Stream Analyse" den Stream analysieren. Nun kann der Hinweg und Rückweg als separate Datei gespeichert werden. Man muss jedoch als Datei-Typ .raw auswählen. Die .raw-Dateien können nun via folgendem Komandozeilenaufufr abgespielt werden

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

Mit dem universellen und sehr mächtigen Audiokonverter SoX können die Dateien über folgenden Komandozeilenaufufr 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 <https://gist.github.com/avimar/d2e9d05e082ce273962d742eb9acac16> können genau diese Schritte automatisiert ausgeführt werden.

6.5. Vollautomatisieren aller Schritte

6.6. incron

Das Abspeichern der Daten funktioniert bereits voll automatisiert und jeweils auch in eine extra Datei pro Session. Allerdings muss das Convertierungs-Skript noch automatisch getriggert bzw. 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 bei einer neuen Datei etwas getriggert werden, wie z.B. eben die Ausführung des Konvertierungs-Skriptes. Incron ähnelt dabei in der Handhabung an das Standardwerkzeug "Cron", welches Cron Jobs auf Basis von Zeitpunkten startet.

```

1 echo ">>installing incron if not already installed.."
2 if ! which "incron" > /dev/null
3 then
4     sudo apt-get install -y incron
5 fi
6 echo ""
7 echo ""
8
9 echo ">>YOU have to do that manually:"
10 echo ">>append your username into '/etc/incron.allow'"

```

```

11
12 echo ">>starting service with 'systemctl start incron.service'"
13
14 echo ">>add job with 'incrontab -e' and append following line:"
15 echo "/home/all/wiresharkCalls IN_CLOSE_WRITE /home/all/
    startPcap2wavgsmConversion.sh \${@} \${#}"
16 echo ""

```

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. Um nicht an den lokalen PC gebunden zu sein, wäre es möglich die Dateien über einen Webserver global zur Verfügung zu stellen.

6.7. Weiteres Feature

Es soll das letzte oder die letzten Gespräche via einem Telefonanruf wiedergegeben werden können. Dies wurde mit einer/mehreren speziell konfigurierten Nummern ermöglicht.

==> kann auch die Nummer beschränkt werden, sodass nur eine registrierte Nummer die Gespräche abhören kann?!???

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 zunächst in eine `.gsm` Datei und anschließend in das 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.

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.

9. Projektaufteilung

Projekt unterteilt in Aufgaben : dahinter jeweils alle Namen

Dokument unterteilt in Kapitel: dahinter jeweils alle Namen

Appendix

A. 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
```

B. 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

```