

# **Mobile Netze**

## Mobile Netze - Man-In-The-Middle

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

## Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
<b>2. Architektur des Osmocom Systems</b>	<b>2</b>
<b>3. Architektur des OpenBTS Systems</b>	<b>3</b>
3.1. Aufbau und Zusammenspiel . . . . .	3
3.1.1. Bestandteile . . . . .	3
3.1.2. Datenbanken . . . . .	5
<b>4. Inbetriebnahme eines Osmocom Systems</b>	<b>7</b>
4.1. Vorinstallationen . . . . .	7
4.1.1. Ubuntu 16.04.3 . . . . .	7
4.1.2. Git . . . . .	7
4.1.3. Softwarevoraussetzungen . . . . .	7
4.1.4. Aktivierung der Verbindung zum USRP2 . . . . .	7
4.2. Installation einzelner GSM Komponenten . . . . .	8
4.2.1. OsmoTRX . . . . .	8
4.2.2. OsmoBTS . . . . .	8
4.2.3. OsmoNitb unter OpenBSC . . . . .	9
4.3. Starten des Systems . . . . .	9
4.4. Installation weiterer Komponenten . . . . .	10
4.4.1. Asterisk . . . . .	10
4.4.2. Osmo-sip-Connector . . . . .	10
<b>5. Inbetriebnahme eines OpenBTS Systems</b>	<b>11</b>
5.1. Vorinstallationen . . . . .	11
5.1.1. Ubuntu 16.04.3 . . . . .	11
5.1.2. Git . . . . .	11
5.1.3. Softwarevoraussetzungen . . . . .	11
5.2. Installation einzelner GSM Komponenten . . . . .	11
5.3. Starten des Systems . . . . .	11
<b>6. Umsetzung des Projektziels</b>	<b>12</b>
6.1. pcapsipdump . . . . .	13
6.2. Abspeichern der Daten . . . . .	13
6.3. pcap2wavgsm . . . . .	14
6.4. Extrahieren der Daten . . . . .	14
6.5. Vollautomatisieren aller Schritte . . . . .	14
6.6. incron . . . . .	14
6.7. Weiteres Feature . . . . .	15
<b>7. Ergebnisse</b>	<b>16</b>
<b>8. Projektaufteilung</b>	<b>17</b>
<b>A. openbsc.cfg</b>	<b>18</b>
<b>B. osmo-bts.cfg</b>	<b>20</b>

## **1. Einleitung**

Was ist GSM? Was ist OpenBTS, OsmoBTS, OpenBSC, Osmo-Nitb etc?

- eigenes weiteres Kapitel – Was ist das Ziel des Projektes - UseCases
- Architektur dann als eigenes Kapitel - Beschreibung einzelner Komponenten sowie deren Funktion
- Systemspezifikation

## **2. Architektur des Osmocom Systems**

Evtl unterteilung in Kapitel: Überblick Beschreibung der einzelnen Komponenten

### 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

##### 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 1) über das Netzwerk mit allen genutzten Computern verbunden.



Abbildung 1: 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 2 dargestellt. Dabei repräsentieren eckige Boxen Hardware-Komponenten, abgerundete Boxen Software-Komponenten.

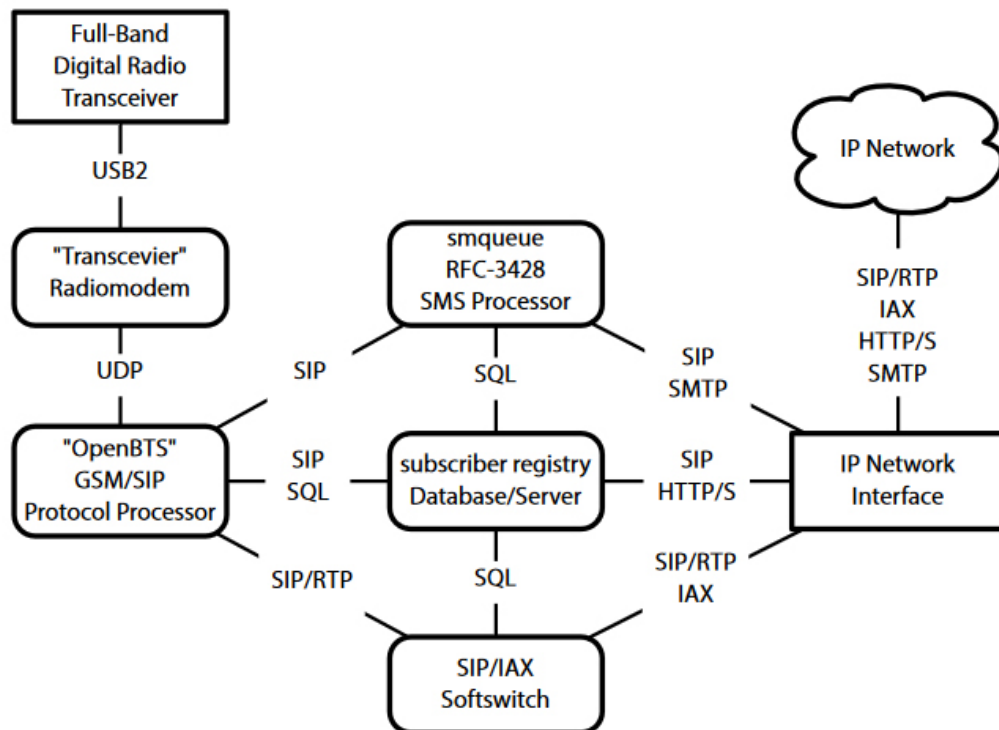


Abbildung 2: OpenBTS Systembestandteile

### 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:

<b>OpenBTS.db</b>	Enthält alle Konfigurationseinstellungen des OpenBTS-Hauptprogramms. Die Einstellungen der für uns lizenzierten Frequenz, sowie Netzwerkparameter wie MCC, MNC oder Name können hier gesetzt werden.
<b>TMSITable.db</b>	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.
<b>ChannelTable.db</b>	Enthält den Channel Status aller aktiven Channel und wird vom OpenBTS-Hauptprogramm genutzt. Der Pfad wird als Parameter in der OpenBTS.db-Datenbank gesetzt.
<b>sipauthserve.db</b>	Enthält alle Konfigurationseinstellungen von SIPAuthServe, unter anderem den Dateipfad zur Subscriber Registry (sqlite3.db).
<b>smqueue.db</b>	Enthält alle Konfigurationseinstellungen von SMQueue, unter anderem den Dateipfad zur Subscriber Registry (sqlite3.db).
<b>sqlite3.db</b>	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 3 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.

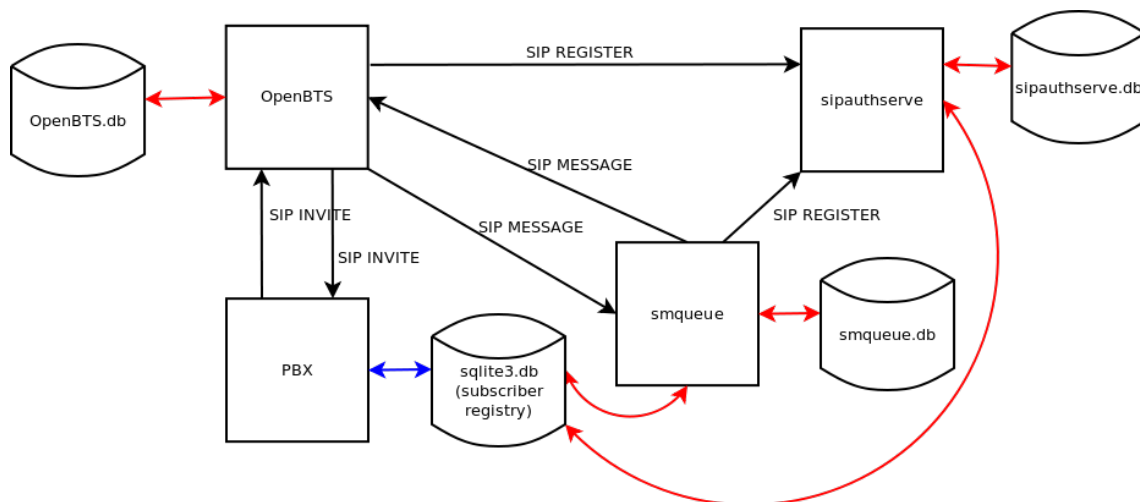


Abbildung 3: OpenBTS System Diagramm

### 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 wie folgt aktiviert werden. Die default IP Adresse des Ettus USRP2 ist 192.168.10.2.

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

## 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.

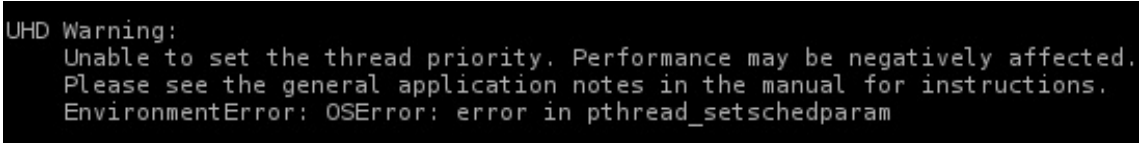
### 4.2.1. OsmoTRX

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
```

Zur Behebung der Warnung, die in Abbildung 4 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
```

Abbildung 4: 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. OsmoNtb 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.

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

Die Reihenfolge der weiteren Komponenten spielt keine Rolle. OsmoBTS wird wie folgt gestartet.

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

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
```

## **4.4. Installation weiterer Komponenten**

### **4.4.1. Asterisk**

### **4.4.2. Osmo-sip-Connector**

## **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**

Verweis machen, wenn wie bei Osmocom

#### **5.1.2. Git**

Verweis machen, wenn wie bei Osmocom

#### **5.1.3. Softwarevoraussetzungen**

selbst

### **5.2. Installation einzelner GSM Komponenten**

selbst

### **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 Kommandozeile 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 $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 ""

```

## 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 $HOMEPATH/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 caltable-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         (currrtime - table[idx].last_packet_time > 5) ||
214         (currrtime - 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 Komandozeilenaufwurf 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 Komandozeilenaufwurf 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**

vielleicht unterteilen in: Fazit/ Ergebnisse/ Was wurde umgesetzt, was nicht Probleme Lessons Learned

## **8. 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

```