Running FOSS Cellular Networks on Linux

Harald Welte <laforge@gnumonks.org>
What this talk is about

- Implementing GSM/GPRS network elements as FOSS
- Applied Protocol Archeology
- Doing all of that on top of Linux (in userspace)
- If you expected kernel stuff, you’ll be disappointed
Running your own Internet-style network

- use off-the-shelf hardware (x86, Ethernet card)
- use any random Linux distribution
- configure Linux kernel TCP/IP network stack
  - enjoy fancy features like netfilter/iproute2/tc
- use apache/lighttpd/nginx on the server
- use Firefox/chromium/konqueor/lynx on the client
- do whatever modification/optimization on any part of the stack
Running your own GSM network

Until 2009 the situation looked like this:

- go to Ericsson/Huawei/ZTE/Nokia/Alcatel/...
- spend lots of time convincing them that you’re an eligible customer
- spend a six-digit figure for even the most basic full network
- end up with black boxes you can neither study nor improve
  - WTF?
  - I’ve grown up with FOSS and the Internet. I know a better world.
Why no cellular FOSS?

- both cellular (2G/3G/4G) and TCP/IP/HTTP protocol specs are publicly available for decades. Can you believe it?
- Internet protocol stacks have lots of FOSS implementations
- cellular protocol stacks have no FOSS implementations for the first almost 20 years of their existence?
- it’s the classic conflict
  - classic circuit-switched telco vs. the BBS community
  - ITU-T/OSI/ISO vs. Arpanet and TCP/IP
Enter Osmocom

In 2008, some people started to write FOSS for GSM

- to boldly go where no FOSS hacker has gone before
  - where protocol stacks are deep
  - and acronyms are plentiful
  - we went from `bs11-abis` to `bsc_hack` to *OpenBSC*
  - many other related projects were created
  - finally leading to the *Osmocom* umbrella project
Classic GSM network architecture

Structure of a GSM network

- **ME**: Mobile Equipment
- **MS**: Mobile Station
- **UE**: User Equipment
- **SIM**: Mobile Equipment
- **BTS**: Base Transceiver Station
- **BSC**: Base Station Controller
- **GERAN**: GSM EDGE Radio Access Network
- **BSS**: Base Station System
- **AN**: Access Network
- **GPRS PS**: Packet Switched
- **CS**: Circuit Switched
- **MSC**: Mobile Switching Centre
- **VLR**: Visitor Location Register
- **HSS**: Home Subscriber Server
- **AuC**: Authentication Centre
- **EIR**: Equipment Identity Register
- **SMS-GMSC**: SMS Gateway MSC
- **GGSN**: Gateway GPRS Service Node
- **PSTN**: Public Switched Telephone Network
- **Internet**: Internet

Connections:
- **Um**: Interface between MS and BTS
- **Abis**: Interface between BTS and BSC
- **Gb**: Interface between GSN and GGSN
- **Gc**: Interface between CN and GGSN
- **Gb**: Interface between GSN and GGSN
- **Gd**: Interface between CN and GGSN
- **Gf**: Interface between CN and GGSN
- **Gn**: Interface between CN and GGSN
- **Gi**: Interface between CN and GGSN
- **PSTN**: Interface to PSTN
- **Internet**: Interface to Internet

GSM Acronyms, Radio Access Network

**MS**
Mobile Station (your phone)

**BTS**
Base Transceiver Station, consists of 1..n TRX

**TRX**
Transceiver for one radio channel, serves 8 TS

**TS**
Timeslots in the GSM radio interface; each runs a specific combination of logical channels

**BSC**
Base Station Controller
GSM Acronyms, Core Network

**MSC**
Mobile Switching Center; Terminates MM + CC Sub-layers

**HLR**
Home Location Register; Subscriber Database

**SMSC**
SMS Service Center
**GSM Acronyms, Layer 2 + 3**

**LAPDm**
- Link Access Protocol, D-Channel. Like LAPD in ISDN

**RR**
- Radio Resource (establish/release dedicated channels)

**MM**
- Mobility Management (registration, location, authentication)

**CC**
- Call Control (voice, circuit switched data, fax)

**CM**
- Connection Management
Osmocom GSM components
Includes functionality of
* BSC
* MSC/VLR
* HLR/AUC
* SMSC

Some support for
* Siemens
* Nokia
* Ericsson
* ip.access
Classic GSM network as digraph
Simplified OsmoNITB GSM network

which further reduces to the following minimal setup:

So our minimal setup is a **Phone**, a **BTS** and **OsmoNITB**.
Which BTS to use?

- Proprietary BTS of classic vendor
  - Siemens BS-11 is what we started with
  - Nokia, Ericsson, and others available 2nd hand

- *OsmoBTS* software implementation, running with
  - Proprietary HW + PHY (DSP): *sysmoBTS*, or
  - General purpose SDR (like USRP) + *OsmoTRX*

We assume a sysmoBTS in the following tutorial
OsmoBTS Overview

- Implementation of GSM BTS
- supports variety of hardware/PHY options
  - `osmo-bts-sysmo`: BTS family by sysmocom
  - `osmo-bts-trx`: Used with OsmoTRX + general-purpose SDR
  - `osmo-bts-octphy`: Octasic OCTBTS hardware / OCTSDR-2G PHY
  - `osmo-bts-litecell15`: Nutaq Litecell 1.5 hardware/PHY
Configuring Osmocom software

- all Osmo* GSM infrastructure programs share common architecture, as defined by various libraries *libosmo{core,gsm,vty,abis,netif,...}*
- part of this is configuration handling
  - interactive configuration via command line interface (**vty**), similar to Cisco routers
    - based on a fork of the VTY code from Zebra/Quagga, now *libosmovty*
- you can manually edit the config file,
- or use **configure terminal** and interactively change it
Configuring OsmoBTS

- *OsmoBTS* in our example scenario runs on the embedded ARM/Linux system inside the *sysmoBTS*
- we access the *sysmoBTS* via serial console or ssh
- we then edit the configuration file `/etc/osmocom/osmo-bts.cfg` as described in the following slide
Configuring OsmoBTS

```
bts 0
  band DCS1800 <1>
  ipa unit-id 1801 0 <2>
  oml remote-ip 192.168.100.11 <3>
```

1. the GSM frequency band in which the BTS operates
2. the unit-id by which this BTS identifies itself to the BSC
3. the IP address of the BSC (to establish the OML connection towards it)

**Note** All other configuration is downloaded by the BSC via OML. So most BTS settings are configured in the BSC/NITB configuration file.
Configuring OsmoNITB

- **OsmoNITB** is the `osmo-nitb` executable built from the `openbsc` source tree / git repository
- just your usual `git clone && autoreconf -fi && ./configure && make install`
  - (in reality, the `libosmo*` dependencies are required first...)
- **OsmoNITB** runs on any Linux system, like your speakers' laptop
  - you can actually also run it on the ARM/Linux of the `sysmoBTS` itself, having a literal *Network In The Box* with power as only external dependency
Configuring OsmoNITB

```
network
    network country code 1 <1>
    mobile network code 1 <2>
    short name Osmocom <3>
    long name Osmocom
    auth policy closed <4>
    encryption a5 0 <5>
```

1. MCC (Country Code) e.g. 262 for Germany; 1 == Test
2. MNC (Network Code) e.g. mcc=262, mnc=02 == Vodafone; 1 == Test
3. Operator name to be sent to the phone after registration
4. Only accept subscribers (SIM cards) explicitly authorized in HLR
5. Use A5/0 (== no encryption)
## Configuring BTS in OsmoNITB (BTS)

<table>
<thead>
<tr>
<th>Command</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>network</td>
<td>bts 0</td>
</tr>
<tr>
<td></td>
<td>type sysmobts</td>
</tr>
<tr>
<td></td>
<td>band DCS1800</td>
</tr>
<tr>
<td></td>
<td>ms max power 33</td>
</tr>
<tr>
<td></td>
<td>periodic location update 6</td>
</tr>
<tr>
<td></td>
<td>ip.access unit_id 1801 0</td>
</tr>
<tr>
<td></td>
<td>codec-support fr hr efr amr</td>
</tr>
</tbody>
</table>

1. **type of the BTS that we use** (must match BTS)
2. **frequency band of the BTS** (must match BTS)
3. **maximum transmit power** phones are permitted (33 dBm == 2W)
4. **interval at which phones should send periodic location update** (6 minutes)
5. **Unit ID of the BTS** (must match BTS)
6. **Voice codecs supported by the BTS**
Configuring BTS in OsmoNITB (TRX)

```bash
network
bts 0
  trx 0
    arfcn 871 <1>
    max_power_red 0 <2>
    timeslot 0
      phys_chan_config CCCH+SDCCH4 <3>
    timeslot 1
      phys_chan_config TCH/F <4>
      ...
    timeslot 7
      phys_chan_config PDCH <5>
```

1. The RF channel number used by this TRX
2. The maximum power reduction in dBm. 0 = no reduction
3. Every BTS needs need one timeslot with a CCCH
4. We configure TS1 to TS6 as TCH/F for voice
5. We configure TS6 as PDCH for GPRS
What a GSM phone does after power-up

- Check SIM card for last cell before switch-off
  - if that cell is found again, use that
  - if not, perform a network scan
    - try to find strong carriers, check if they contain BCCH
    - create a list of available cells + networks
    - if one of the networks MCC+MNC matches first digits of **IMSI**, this is the home network, which has preference over others

- perform **LOCATION UPDATE** (TYPE=IMSI ATTACH) procedure to network
- when network sends **LOCATION UPDATE ACCEPT**, **camp** on that cell

→ let’s check if we can perform **LOCATION UPDATE** on our own network
Verifying our network

- look at stderr of *OsmoBTS* and *OsmoNITB*
  - *OsmoBTS* will terminate if Abis cannot be set-up
  - expected to be re-spawned by init / systemd
- use MS to search for networks, try manual registration
- observe registration attempts logging level mm info
  → should show *LOCATION UPDATE* request / reject / accept
  - use the VTY to explore system state (*show * )
  - use the VTY to change subscriber parameters like extension number
Exploring your GSM networks services

- use *#100# from any registered MS to obtain own number
- voice calls from mobile to mobile
- SMS from mobile to mobile
- SMS to/from external applications (via SMPP)
- voice to/from external PBX (via MNCC)
- explore the VTY interfaces of all network elements
  - send SMS from the command line
  - experiment with silent call feature
  - experiment with logging levels
- use wireshark to investigate GSM protocols
Using the VTY

- The VTY can be used not only to configure, but also to interactively explore the system status (`show commands`)
- Every Osmo* program has its own telnet port

<table>
<thead>
<tr>
<th>Program</th>
<th>Telnet Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>OsmoPCU</td>
<td>4240</td>
</tr>
<tr>
<td>OsmoBTS</td>
<td>4241</td>
</tr>
<tr>
<td>OsmoNITB</td>
<td>4242</td>
</tr>
<tr>
<td>OsmoSGSN</td>
<td>4245</td>
</tr>
</tbody>
</table>

- ports are bound to `127.0.0.1` by default
- try tab-completion, `?` and `list` commands
Using the VTY (continued)

- e.g. `show subscriber` to display data about subscriber:

```
OpenBSC> show subscriber imsi 901700000003804
    ID: 12, Authorized: 1
    Extension: 3804
    LAC: 0/0x0
    IMSI: 901700000003804
    TMSI: F2D4FA0A
    Expiration Time: Mon, 07 Dec 2015 09:45:16 +0100
    Paging: not paging Requests: 0
    Use count: 1
```

- try `show bts, show trx, show lchan, show statistics, ...`
Extending the network with GPRS

Now that GSM is working, up to the next challenge!

- Classic GSM is circuit-switched only
- Packet switched support introduced first with GPRS
- GPRS adds new network elements (PCU, SGSN, GGSN)
- tunnel for external packet networks like IP/Internet
- tunnel terminates in MS and on GGSN
Extending the network with GPRS support

- **PCU**: Packet Control Unit. Runs RLC+MAC
- **SGSN**: Serving GPRS Support Node (like VLR/MSC)
- **GGSN**: Gateway GPRS Support Node (terminates tunnels)
GPRS Signalling basics

- GPRS Mobility Management (GMM)
  - just like GSM Mobility Management (MM)
    - *GPRS ATTACH, ROUTING AREA UPDATE, AUTHENTICATION*

- GPRS Session Management (SM)
  - establishment, management and tear-down of packet data tunnels
    - independent from IP, but typically IP(v4) is used
    - *PDP Context* (Activation | Deactivation | Modification)
GPRS Protocol Stack
GPRS User Plane

HTTP
TCP
IP

SNDCP
LLC
RLC
MAC

Physical Layer

TRAU Framing
Physical Layer
E1

TRAU Framing
RLC
MAC
NS
BSSGP

Frame Relay
E1
E1

Frame Relay
NS
Ethernet

IP
TCP
UDP
GTP-U

GGSN

BTS+CCU
A-bis
BSC+PCU
Gb
SGSN
Gn

MS
Um

GPRS Acronyms, Protocol Stack

- **Layer 3**
  - *SM*: Session Management (PDP contexts)
  - *GMM*: GPRS Mobility Management (like MM)

- **Layer 2**
  - *MAC*: Medium Access Control
  - *LLC*: Link Layer Control (segmentation, compression, encryption)
  - *RLC*: Radio Link Control
  - *SNDCP*: Sub-Network Dependent Convergence Protocol

  Scotty to the bridge: You have to re-modulate the sub-network dependent convergence protocols!
Simplified OsmoNITB network with GPRS

- **OsmoPCU** is co-located with **OsmoBTS**
  - connects over unix-domain PCU socket to BTS
- **OsmoSGSN** can run on any Linux machine
- **OpenGGSN** can run on any Linux machine
  - `tun` device is used for tunnel endpoints
- circuit-switched and packet-switched networks are completely separate

We need to configure those additional components to provide GPRS services.
Simplified OsmoNITB network with GPRS

Includes functionality of
* BSC
* MSC/VLR
* HLR/AUC
* SMSC
Configuring OsmoPCU

We assume we have obtained and compiled the osmo-pcu from git://git.osmocom.org/osmo-pcu

- **OsmoPCU** runs co-located with **OsmoBTS** to access/share the same PHY + Radio
- **OsmoPCU** is primarily configured from **OsmoBTS**
- **OsmoBTS** receives relevant config via A-bis OML
- **OsmoNITB** sends those OML messages to OsmoBTS
  - we thus need to set the PCU configuration in the NITB config file!
# BTS config for GPRS (in OsmoNITB)

```bash
bts 0
  gprs mode gprs <1>
gprs nsei 1234 <2>
gprs nsvc 0 nsvci 1234 <3>
gprs nsvc 0 local udp port 23000 <4>
gprs nsvc 0 remote ip 192.168.1.11 <5>
gprs nsvc 0 remote udp port 23000 <6>
```

1. enable `gprs` or `egprs` mode
2. NSEI for the NS protocol layer (unique for each PCU in SGSN)
3. NSVCI for the NS protocol layer (unique for each PCU in SGSN)
4. UDP port on PCU side of Gb connection
5. IP address of SGSN side of Gb connection
6. UDP port on SGSN side of Gb connection
Configuring OsmoSGSN (Gb and GTP)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SGSN-local IP address for Gb connection from PCUs</td>
</tr>
<tr>
<td>2</td>
<td>SGSN-local UDP port number for Gb connection from PCUs</td>
</tr>
<tr>
<td>3</td>
<td>SGSN-local IP address for GTP connection to GGSN</td>
</tr>
<tr>
<td>4</td>
<td>remote IP address for GTP connection to GGSN</td>
</tr>
<tr>
<td>5</td>
<td>GTP protocol version for this GGSN</td>
</tr>
<tr>
<td>6</td>
<td>route all APN names to GGSN 0</td>
</tr>
</tbody>
</table>

```
ns
encapsulation udp local-ip 192.168.100.11  <1>
encapsulation udp local-port 23000  <2>
sgsn
  gtp local-ip 127.0.0.2 <3>
  ggsn 0 remote-ip 127.0.0.1 <4>
  ggsn 0 gtp-version 1 <5>
  apn * ggsn 0 <6>
```
Configuring OsmoSGSN (subscribers)

OsmoSGSN (still) has no access to the OsmoNITB HLR, thus all IMSIs permitted to use GPRS services need to be explicitly configured.

```sh
sgsn
  auth-policy closed <1>
  imsi-acl add 262778026147135 <2>
```

1. only allow explicitly authorized/white-listed subscribers
2. add given IMSI to the white-list of subscribers
Setting up OpenGGSN

In `ggsn.cfg` we need to set:

```
listen 172.0.0.1 <1>
net 10.23.24.0/24 <2>
dynip 10.23.42.0/24 <3>
podns1 8.8.8.8 <4>
```

1. IP address to bind GSN to.
2. network/mask of `tun` device
3. pool of dynamic IP addresses allocated to PDP contexts
4. IP address of DNS server (communicated to MS via signalling)
Testing GPRS

- Check if `osmo-pcu`, `osmo-sgsn`, `openggsn` are running
- Check if NS and BSSGP protocols are UNBLOCKED at SGSN
  - If not, check your NS/BSSGP configuration
- Check for GPRS registration using `logging level mm info` in SGSN
Osmocom beyond GSM/GPRS RAN + NITB

- Smalltalk implementation of SIGTRAN + TCAP/MAP
- Erlang implementation of SIGTRAN + TCAP/MAP
- Lots of special-purpose protocol mangling
  - `bsc-nat` to introduce NAT-like functionality on A (BSSAP/BSSMAP)
  - `mgw-nat` to transparently re-write MAP/ISUP/SCCP
- GSMTAP pseudo-header for feeding non-IP protocols into wireshark
- SIM card protocol tracer hardware + software
- Lots of non-GSM projects from hardware to protocol stacks (TETRA, GMR, DECT, OP25)
- check [http://git.osmocom.org/](http://git.osmocom.org/) for full project list
So… I heard about OpenBTS?

- OpenBTS is completely unrelated to the Osmocom stack
- was independently developed by David Burgess & Harvind Simra
  - Kestrel Signal Processing → Range Networks
- doesn’t follow GSM system architecture at all
  - no Abis, BSC, PCU, SGSN, GGSN
- is a bridge of the GSM air interface (Um) to SIP
- Osmocom follows classic GSM interfaces / system architecture
- *OsmoTRX* forked *OpenBTS* SDR code to use *OsmoBTS* with SDR hardware
Outlook on FOSS 2.75G (EDGE)

- EDGE extends GPRS with higher data rates
  - 8PSK instead of GMSK modulation
  - lots of new MAC/RLC features (larger windows, incremental redundancy)
  - No changes required in OmsoSGSN and OsmoGGSN
- OsmoPCU is extended with EDGE support
- First working minimal subset published last week
Outlook on FOSS 3G (UMTS/WCDMA)

- UMTS very similar to GSM/GPRS in principle
  - still, almost every interface and protocol stack has changed
  - all elements have been renamed → more acronyms to learn

- UMTS is ridiculously complex, particular PHY + Layer 2
  - however, control plane L3 (MM/CC/CM/SM/GMM) mostly the same

- Implementing all of that from scratch is a long journey

- We’ve already reached *Peak 3G*

- Osmocom 3G support strategy
  - Implement Iu interface in NITB and SGSN
  - Implement HNB-GW to offer Iuh interface
  - Use existing femtocell / small cell hardware with proprietary PHY, RLC and MAC
  - Status: Started in October 2015, WIP. Overall completion > 50%.
Outlook on FOSS 4G (LTE)

- LTE has nothing in common with 2G/3G
- various FOSS activities
  - *OpenAirInterface* has some code for a software eNodeB
    - but they switched from GPLv3 to *non-free* license :(
  - *srsLTE* (main focus on UE side, but large parts usable for eNodeB side)
  - *OpenLTE* is another active FOSS project
- No Osmocom involvement so far
  - team is small, project scope of cellular infrastructure is gigantic
  - most customer funding currently still on GSM/GPRS/EDGE
  - if we’d start, we’d start implementing MME + S-GW and use existing LTE cells
The End

- so long, and thanks for all the fish
- I hope you have questions!
- have fun exploring mobile technologies using Osmocom
- interested in working with more acronyms? Come join the project!
- Check out http://openbsc.osmocom.org/ and openbsc@lists.osmocom.org
Thanks to

- Pablo for running netdevconf and inviting me
- the entire Osmocom team for what they have achieved
  - notably Dieter Spaar, Holger Freyther, Andreas Eversberg, Sylvain Munaut
- last but not least: CEPT for making the GSM specs English
  - (who’d want to read French specs anyway?)