Osmo-CC Endpoint Guide

(jolly@eversberg.eu)

Index

Table of Contents

1. Osmo-CC

1.1 What is Osmo-CC

Osmo-CC is a telephony interface to connect telephone applications and interfaces. Each application or interface is called '**endpoint**'. The applications may run on a local machine or local IP network, or over wide area networks using tunnel or other security features.

It was designed as a replacement for the '**MNCC**' interface that was used by '**openbsc**' software to interconnect with a simple SIP endpoint or '**Linux-Call-Router**' (ISDN and SIP gateway software). The '**MNCC**' interface was re-used for other projects like '**OsmocomBB**' (mobile phone stack) and '**osmocom-analog**' (classic 1G base station emulator). The '**MNCC**' interface was designed for GSM. The messages and state machine is different for network and mobile side, so that upper layers are not compatible. It does not support overlap dialing, so that it is restricted to be used in the mobile world, where no overlap dialing exists. Information elements are restricted to GSM codecs, hacks were used to add linear audio for '**osmocom-analog**'.

Osmo-CC is made to interface different telephone applications, like ISDN interfaces, POTS interfaces, SIP interfaces, GSM components and more. Instead of using different application parts as in Signaling System No. 7, Osmo-CC provides a union off most messages, call states and information elements for all telephone networks. The application that uses Osmo-CC does not need to consider what network is used. Also it does not need to consider what mode the network protocol uses, like NT or TE, User or Network, FXS or FXO, ect.

1.2 Features of Osmo-CC

The Osmo-CC interface can be expanded with new messages and information elements that may be required by endpoints for new networks. Older existing application will simply ignore these information.

Osmo-CC supports overlap dialing, so that analog phones (FXS) can be used without dial timeout. But it is not required that an endpoint supports overlap sending or receiving or both.

Osmo-CC supports early audio until the call is connected. This way it is possible to hear announcements or tones. Late audio (after the call has been disconnected) is also supported. It is not required that endpoints support sending/receiving early/late audio.

Osmo-CC can be used as an interface inside an application with an endpoint, as well as between endpoints using a TCP/IP socket interface.

1.3 What Osmo-CC does not support

There is no security at Osmo-CC socket interface. This means that no authentication nor encryption is used. This is not a problem, if all applications (endpoints) that use Osmo-CC run on the same machine. If connecting application (endpoints) via Osmo-CC over an insecure network (internet), the network operator needs to add some security, like firewall or encrypted tunneling.

Classical telephone networks allow calls to be suspended or resumed at application layer. Osmo-CC does not support any messages for this. Instead, the endpoint that serves a network must handle suspend and resume of calls itself. Still it is possible to notify other endpoints via Osmo-CC messages that the call has been suspended or resumed.

Call forwarding or redirection is not supported by Osmo-CC. Application that require call forwarding must handle it themselves. Similar to changing the destination of a call, changing the audio codec is also not supported.

1.4 Technical detail

Osmo-CC is the interface that connects the network layer of an endpoint to higher layer. Also it can be used to bridge network layers of multiple endpoints via TCP/IP socket without the requirement of any PBX or router.

All messages start with a single byte of message type, followed by two bytes of length (network order) of all information elements, followed by the information elements. Each information element starts with a single byte of information element type, followed by two bytes of length (network order) of the information element, followed by information element data. Information elements may be used as a vector of information element. This means they may be omitted, included once or repeated multiple times within a message.

The interface is almost symmetrical. A release towards lower layer must be confirmed, a release from lower is not responded. (This way the application knows when resources like audio channels are free to be used for new calls.) Also the numbering of caller IDs and dialing can be different in each direction.

A TCP/IP socket, that is implemented in '**libosmocc**' with help of a little processing makes the Osmo-CC interface completely symmetrical. Also '**libosmocc**' implements some '**screening**' lists, to convert caller IDs and dialed numbers. Running two applications on the same machine does not require any IP/port configuration. In this case auto-configuration is used.

1.5 Audio streaming

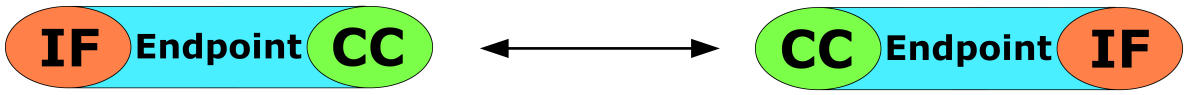
The actual audio is not streamed via Osmo-CC socket interface. Instead, '**RTP**' is used to transfer audio between two endpoints. This means that each endpoint must be capable of transfer '**RTP**'. Also it must support at least '**a-Law**' and '**mu-Law**' codec. In special setups

where both endpoints are designed to use different codec, it is not required to support '**a-Law**' and '**mu-Law**' codecs.

To negotiate codecs between endpoints, '**SDP**' is used. Osmo-CC just forwards '**SDP**' information inside its messages. This means that each endpoint must support '**SDP**' also. The benefit is that codec negotiation and audio transfer is end-to-end, especially when using SIP endpoints. New codecs do not require an update of Osmo-CC. Even video and other media could be used.

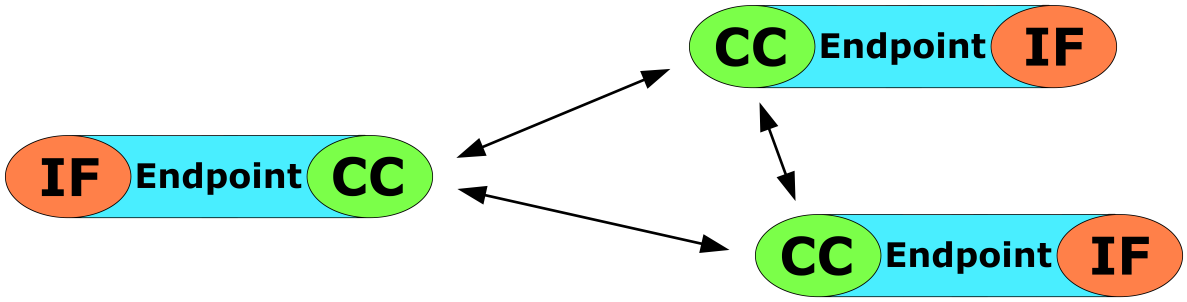
2. Osmo-CC endpoints

2.1 Overview of endpoints



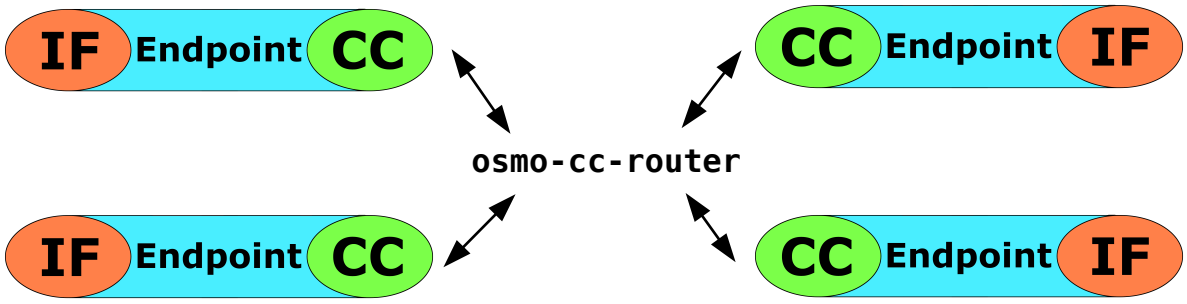
Two endpoints directly connected to each other

An Osmo-CC endpoint is a software that implements a telephony interface, like ISDN or SIP. It uses Osmo-CC socket interface to connect to another endpoint. The figure above shows two endpoints connected to each other via socket interface, no PBX is in between. This means that there are no routing or other features. All calls from one endpoint are forwarded to the other endpoint. Still it is possible to change caller ID and dialed number. This feature is called “screening”.



Three or more endpoints directly connected to each other

It is possible to have more than two endpoints connected to each other without a router. The “screening” feature allows to use caller ID or dialed number to define the destination socket interface for each call.



Multiple endpoints connected via “osmo-cc-router”

The most flexible way is to use ‘**osmo-cc-router**’. It is a script based router.

2.2 osmo-cc-alsa-endpoint

This endpoint can be used to make or receive calls with headset. It can be controlled via console interface. The audio uses ‘**ALSA**’ driver. ‘**ALSA**’ is disabled by default, so that it can be used to make test calls only. Use this as a test utility to test other endpoints.

2.2.1 Installation

```
$ cd ~
$ git clone https://gitea.osmocom.org/cc/osmo-cc-alsa-endpoint
Cloning into 'osmo-cc-alsa-endpoint'…
```

Change to directory:

```
$ cd osmo-cc-alsa-endpoint
```

Configure and compile:

```
$ sudo apt install gcc # in case you don’t have ‘gcc’ installed
$ sudo apt install make # in case you don’t have ‘make’ installed
$ sudo apt install automake # in case you don’t have ‘automake’ installed
$ sudo apt install libasound2-dev # in case you don’t have ALSA development library installed
$ autoreconf –if
$ ./configure
$ make clean
$ make
```

Install on your system:

```
$ sudo make install
```

2.2.2 Example: Call between two ALSA endpoints

Open a first terminal and run ‘osmo-cc-alsa-endpoint’ with given caller ID ‘0815’:

```
$ osmo-cc-alsa-endpoint -I 0815
…
on hook: ............................... (press digits 0..9 or d=dial)
```

Then open a second terminal on the same machine and run ‘osmo-cc-alsa-endpoint’ with some other caller ID or no caller ID:

```
$ osmo-cc-alsa-endpoint
…
on hook: ............................... (press digits 0..9 or d=dial)
```

Now enter some phone number on the first terminal:

```
on hook: 123............................ (press digits 0..9 or d=dial)
```

Then press ‘d’ to dial:

```
connected: 123 (press h=hangup)
```

Now you see the incoming call on the first terminal:

```
connected: 0815->123 (press h=hangup)
```

Press ‘h’ to hangup on either side, to return into ‘on hook’ state.

2.2.3 Using a Headset

To use audio, you need to find out the ‘**ALSA**’ device of your headset. To find out, use ‘arecord –l', which shows all sound devices that have input capability:

```
$ arecord -l
**** List of CAPTURE Hardware Devices ****
card 1: Generic [HD-Audio Generic], device 0: ALC887-VD Analog [ALC887-VD Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: Generic [HD-Audio Generic], device 2: ALC887-VD Alt Analog [ALC887-VD Alt Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

The headset on my machine is connected to the first entry in the list above. It is connected to ‘card 1’, device ‘0’. This means that the ‘**ALSA**’ device name is ‘hw:1,0’, so I add this option:

```
$ osmo-cc-alsa-endpoint –I 0815 -a hw:1,0
```

2.3 osmo-cc-sip-endpoint

This endpoint can be used to transfer calls via SIP protocol. It can be used without registering/authentication (peer-to-peer), but also with registering/authentication in two directions. This way it is possible to register to a SIP proxy, as well as being a SIP proxy that some device registers to.

The endpoint requires Sofia-SIP stack to be installed. It can be downloaded from http://sofia-sip.sourceforge.net/download.html or alternatively from http://download.eversberg.eu/sofia/sofia-sip-1.12.11.tar.gz.

Note: Compiling with regular optimization causes corrupt SIP messages.

2.3.1 Installation

Installation of Sofia-Sip:

```
$ cd ~
$ wget http://download.eversberg.eu/sofia/sofia-sip-1.12.11.tar.gz
```

Unpack and change to directory:

```
$ tar xvf sofia-sip-1.12.11.tar.gz
$ cd sofia-sip-1.12.11/
```

Compile without optimization:

```
$ sudo apt install gcc # in case you don’t have ‘gcc’ installed
$ sudo apt install make # in case you don’t have ‘make’ installed
$ ./configure CFLAGS=-O0
$ make clean
$ make
```

Install on your system. Be sure that you remove other versions of Sofia-Sip from your system, if any:

```
$ sudo make install
$ sudo ldconfig
```

Installation of osmo-cc-sip-endpoint:

```
$ cd ~
$ git clone https://gitea.osmocom.org/cc/osmo-cc-sip-endpoint
Cloning into 'osmo-cc-sip-endpoint'...
```

-

```
$ cd osmo-cc-sip-endpoint
```

Configure and compile:

```
$ sudo apt install automake # in case you don’t have ‘automake’ installed
$ autoreconf –if
$ ./configure
$ make clean
$ make
```

Install on your system:

```
$ sudo make install
```

2.3.2 Wardialing

Before running SIP endpoint that is reachable from everywhere on the Internet, note that there are many bots that scan the Internet and try to find bad configured SIP devices. Once found, they can be abused to make expensive long-distant calls. If you don't plan to link your SIP endpoint to the public telephone network, the bots might still be annoying and make your phone (or whatever is linked to your SIP endpoint) ring all the time.

Look at this example, where my router receives SIP messages from time to time. (at least the upper three)

```
$ tcpdump -n -i ppp0 udp port 5060
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ppp0, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
08:19:28.226120 IP 167.114.117.174.5086 > 82.139.198.227.5060: SIP, length: 419
08:25:29.389918 IP 193.107.216.17.5189 > 82.139.198.227.5060: SIP, length: 415
09:42:49.046676 IP 45.134.144.4.5071 > 82.139.198.227.5060: SIP, length: 405
09:59:50.361642 IP 101.37.32.76.26242 > 82.139.198.227.5060: SIP, length: 16
```

If you think that authentication prevents bots from making calls, you are wrong. If you register and make calls with authentication to some SIP proxy, you are not safe. If there is an incoming call from that proxy, no authentication is used in this direction. You need to prevent bots from reaching your SIP endpoint. The best way is to use a firewall to block all incoming SIP messages, except for messages from the proxy.

By default, local SIP port is 5060. If you run multiple SIP endpoints on one machine, you need other port numbers. Changing the port number instead of blocking IPs is not enough. I have seen 'bots' that scan other ports also.

If you use NAT, you are safe, because NAT will only forward packets of connections that have been previously initiated by the SIP endpoint.

If you use port forwarding, limit that to the IP of the proxy or any other remote SIP peer you want to connect with.

If you have direct connection without NAT and without port forwarding, limit incoming packets to established connections. Configuring your firewall is beyond the scope of this document.

2.3.3 Example: peer-to-peer connection via SIP

For better debugging SIP messages, use '**sngrep**'. It will show all SIP sessions that run via local machine. Install it on your machine with SIP endpoint or on your router, if it runs on Linux. After start it will capture all SIP packets. The first screen displays all session. Use the cursor and enter to select a session and switch to the second screen. The seconds screen shows a session with all messages. Use the cursor and enter to select a message and switch to the third screen. On the third screen the message is shown. Use Escape to go back one screen and to end program.

Run ‘**osmo-cc-alsa-endpoint**‘ on machine 1. The machine has local IP address ‘**10.0.0.28**’, so we need to tell the RTP process our local IP. We set our caller ID to ‘**0815**’. We also add a phone number, so that we don’t need to enter after running the endpoint.

```
$ ‘osmo-cc-alsa-endpoint’ -a default --cc "rtp-peer 10.0.0.28" -I 0815 123
...
options.c: 282 info   : Command line option '--cc', parameter 'rtp-peer 10.0.0.28'
options.c: 268 info   : Command line option '-I' ('--caller-id'), parameter '0815'
socket.c: 381 error  : OsmoCC-Socket failed, socket cause 3.
endpoint.c: 923 info   : Handle message CC-REL-REQ at state ATTACH-SENT (callref 1)
endpoint.c: 314 info   : Attachment to remote peer "127.0.0.1:4201" failed, retrying.
...
```

Also run ‘**osmo-cc-sip-endpoint**’ in different terminal on machine 1. The remote machine has IP address ‘**10.0.0.157**’. We tell SIP endpoint our local and the remote IP.

```
$ osmo-cc-sip-endpoint -l 10.0.0.28 -r 10.0.0.157
...
options.c: 268 info   : Command line option '-l' ('--local'), parameter '10.0.0.28'
options.c: 268 info   : Command line option '-r' ('--remote'), parameter '10.0.0.157'
endpoint.c: 923 info   : Handle message CC-ATTACH-RSP at state ATTACH-SENT (callref 1)
endpoint.c: 299 info   : Attached to remote peer "127.0.0.1:4200".
endpoint.c: 923 info   : Handle message CC-ATTACH-REQ at state IDLE (callref 2)
endpoint.c:  381 info     :  Remote  peer  with  socket  address  '127.0.0.1'  and  port  '4200'  and
interface 'alsa' attached to us.
endpoint.c: 385 info   : Changing message to CC-ATTACH-CNF.
...
```

On machine 2, run ‘**omso-cc-alsa-endpoint**’. Note that our local machine has local IP ‘**10.0.0.157**’.

```
$ osmo-cc-alsa-endpoint -a default --cc "rtp-peer 10.0.0.157"
...
options.c: 282 info   : Command line option '--cc', parameter 'rtp-peer 10.0.0.157'
options.c: 268 info   : Command line option '-I' ('--caller-id'), parameter '0815'
socket.c: 381 error  : OsmoCC-Socket failed, socket cause 3.
endpoint.c: 923 info   : Handle message CC-REL-REQ at state ATTACH-SENT (callref 1)
endpoint.c: 314 info   : Attachment to remote peer "127.0.0.1:4201" failed, retrying.
...
```

Also run ‘**osmo-cc-sip-endpoint**’ in different terminal on machine 2. Note that the remote machine has IP address ‘**10.0.0.28**’.

```
$ osmo-cc-sip-endpoint -l 10.0.0.157 -r 10.0.0.28
...
options.c: 268 info   : Command line option '-l' ('--local'), parameter '10.0.0.157'
options.c: 268 info   : Command line option '-r' ('--remote'), parameter '10.0.0.28'
endpoint.c: 923 info   : Handle message CC-ATTACH-RSP at state ATTACH-SENT (callref 1)
endpoint.c: 299 info   : Attached to remote peer "127.0.0.1:4200".
endpoint.c: 923 info   : Handle message CC-ATTACH-REQ at state IDLE (callref 2)
endpoint.c:  381 info     :  Remote  peer  with  socket  address  '127.0.0.1'  and  port  '4200'  and
interface 'alsa' attached to us.
endpoint.c: 385 info   : Changing message to CC-ATTACH-CNF.
...
```

To debug a SIP call, run ‘**sngrep**’ on one of the machines in another terminal. Now press ‘d’ to dial the on machine 1 at console of ‘**osmo-alsa-endpoint**’.

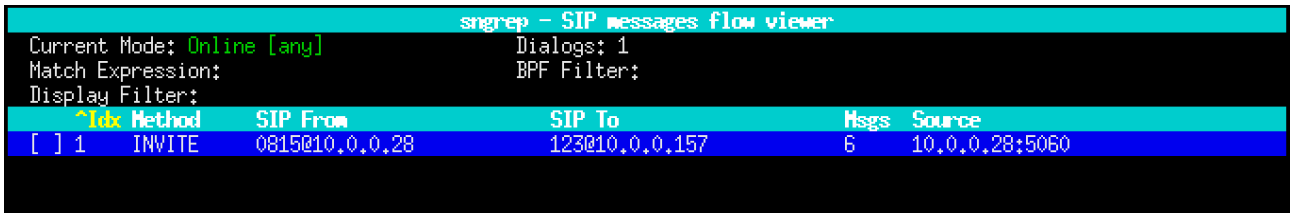
```
telephone.c: 642 info   : Outgoing call from '0815' to '123'
endpoint.c: 923 info   : Handle message CC-SETUP-IND at state IDLE (callref 361)
endpoint.c: 923 info   : Handle message CC-PROC-REQ at state INIT-IN (callref 361)
telephone.c: 484 info   : Incoming call acknowledged
```

```
endpoint.c: 923 info   : Handle message CC-SETUP-RSP at state PROCEEDING-IN (callref 361)
telephone.c: 503 info   : Incoming call acknowledged
endpoint.c: 923 info   : Handle message CC-SETUP-COMP-IND at state CONNECTING-IN (callref 361)
connected: 123 (press h=hangup)
```

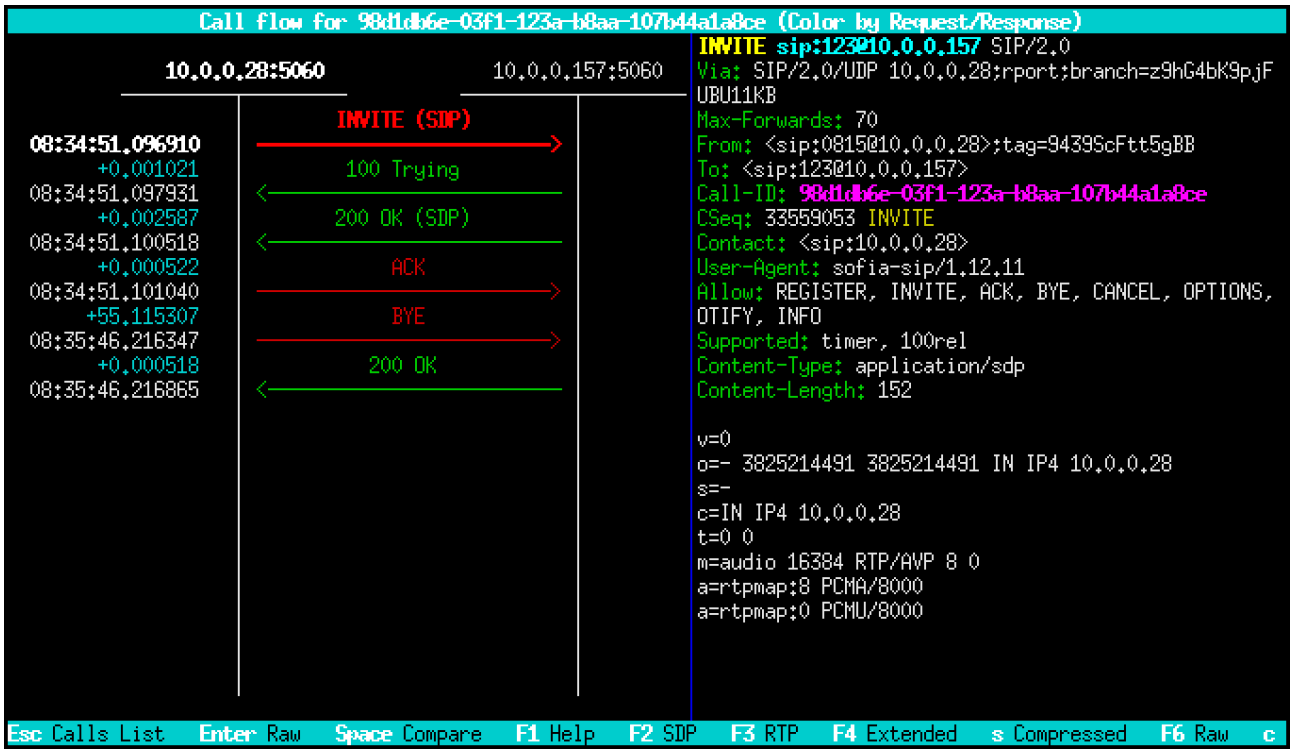
Then press ‘h’ to end the call.

```
telephone.c: 686 info   : Call hangup
endpoint.c: 923 info   : Handle message CC-REL-IND at state ACTIVE (callref 361)
endpoint.c: 740 info   : Changing message to CC-DISC-IND.
endpoint.c: 923 info   : Handle message CC-REL-REQ at state DISCONNECTING-IN (callref 361)
on hook: 123.............................. (press digits 0..9 or d=dial)
```

Watch the debug output on all four terminals. Also look at the ‘**sngrep**’ terminal:



Press enter to see a detailed graph of the call you just made:



You can see in the ‘**INVITE**’ message your local and remote IP, as well as local and remote phone number. The SDP body attached to the ‘**INVITE**’ message contains the supported codecs (PCMU and PCMA), the RTP IP ‘**10.0.0.28**’ and port ‘**16384**’ of machine 1. Use cursor keys to view the other messages.

2.3.4 Example: Register to a SIP proxy

Before we start, we need an account. In this example I register to my '**sipgate basic**' account and make a call. The settings are:

- User-ID: **160XXXX**
- Registrar: **sipgate.de**
- SIP Proxy: **sipgate.de**
- Password: **paSSword**

In this example, we use '**osmo-cc-alsa-endpoint**' to make a test call. Because we need send and receive RTP from/to different host (sipgate), we need to tell '**libosmocc**' what RTP peer we use. On the local machine we define it with '**--cc "rtp-peer 10.0.0.28"**'. Also do that for any other endpoint you want to use instead.

```
$ osmo-cc-alsa-endpoint -a default --cc "rtp-peer 10.0.0.28"
...
options.c: 268 info   : Command line option '-a' ('--audio-device'), parameter 'default'
options.c: 282 info   : Command line option '--cc', parameter 'rtp-peer 10.0.0.28'
socket.c: 381 error  : OsmoCC-Socket failed, socket cause 3.
endpoint.c: 923 info   : Handle message CC-REL-REQ at state ATTACH-SENT (callref 1)

endpoint.c: 314 info   : Attachment to remote peer "127.0.0.1:4201" failed, retrying.
on hook: ............................... (press digits 0..9 or d=dial)
```

Because my User-ID is '**160XXXX**' and the local machine has the IP address '**10.0.0.28**', we set our local identity with '**-l 160XXXX@10.0.0.28**'.

We set remote identity to '**sipgate.de**' only, because we want the dialed number to be added automatically. We do it with '**-r sipgate.de**'.

We register, because we want to tell the remote SIP gateway where to reach us in case of an incoming call. We use our User-ID '**160XXXX**' and the Registrar '**sipgate.de**' and do it with '**-R 160XXXX@sipgate.de**'

Also we need to authenticate ourself while registering or making a call, so we use our User-ID '**160XXXX**' and password '**paSSword**' and do it with '-**A 160XXXX 'paSSword' dummy**' The word '**dummy**' is not used when we authenticate towards a remote SIP peer, but it must be added, because this option requires three parameters.

Because the local machine is behind a NAT firewall, our local identity must be translated in all messages sent to the remote peer. We do it with '**-P 82.139.198.227**', if this is the public IP.

```
$ osmo-cc-sip-endpoint -l 160XXXX@10.0.0.28 -r sipgate.de -R 160XXXX@sipgate.de -A 160XXXX
'paSSword' dummy -P 82.139.198.227
...
options.c: 268 info   : Command line option '-l' ('--local'), parameter '10.0.0.28'
options.c: 268 info   : Command line option '-r' ('--remote'), parameter '160XXXX@sipgate.de'
options.c: 268 info   : Command line option '-R' ('--register'), parameter '160XXXX@sipgate.de'
options.c: 268 info   : Command line option '-A' ('--auth'), parameter '160XXXX' 'paSSword'
'dummy'
options.c: 268 info   : Command line option '-P' ('--public-ip'), parameter '82.139.198.227'
sip.c:1870 info   : Sending REGISTER
endpoint.c: 923 info   : Handle message CC-ATTACH-RSP at state ATTACH-SENT (callref 1)
endpoint.c: 299 info   : Attached to remote peer "127.0.0.1:4200".
sip.c: 926 info   : Received REGISTER response: 401 Unauthorized (registration)
```

```
sip.c: 926 info   : Received REGISTER response: 200 OK (registration)
endpoint.c: 923 info   : Handle message CC-ATTACH-REQ at state IDLE (callref 2)
endpoint.c: 381 info   : Remote peer with socket address '127.0.0.1' and port '4200' and
interface 'alsa' attached to us.
endpoint.c: 385 info   : Changing message to CC-ATTACH-CNF.
```

If we have dynamic public IP, we don't want to re-configure our SIP endpoint all the time. Instead we use a '**STUN**' server to resolve our public IP for us. We do it with '**-S stun.sipgate.net**' The procedure is almost the same, except that registration is performed when the public IP is resolved or when it changes.

```
osmo-cc-sip-endpoint -l 160XXXX@10.0.0.28 -r sipgate.de -R 160XXXX@sipgate.de -A 160XXXX
'paSSword' dummy -S stun.sipgate.net
...
options.c: 268 info   : Command line option '-l' ('--local'), parameter '160XXXX@10.0.0.28'
options.c: 268 info   : Command line option '-r' ('--remote'), parameter 'sipgate.de'
options.c: 268 info   : Command line option '-R' ('--register'), parameter '160XXXX@sipgate.de'
options.c: 268 info   : Command line option '-A' ('--auth'), parameter '160XXXX' 'paSSword'
'dummy'
options.c: 268 info   : Command line option '-S' ('--stun-server'), parameter 'stun.sipgate.net'
sip.c:1835 info   : STUN resolving for public IP
assign_socket: local socket is bound to 0.0.0.0:51569
endpoint.c: 923 info   : Handle message CC-ATTACH-RSP at state ATTACH-SENT (callref 1)
endpoint.c: 299 info   : Attached to remote peer "127.0.0.1:4201".
sip.c:1592 info   : STUN resolved!
sip.c:1870 info   : Sending REGISTER
sip.c: 926 info   : Received REGISTER response: 401 Unauthorized (registration)
sip.c: 926 info   : Received REGISTER response: 200 OK (registration)
endpoint.c: 923 info   : Handle message CC-ATTACH-REQ at state IDLE (callref 2)
endpoint.c: 381 info   : Remote peer with socket address '127.0.0.1' and port '4201' and
interface 'alsa' attached to us.
endpoint.c: 385 info   : Changing message to CC-ATTACH-CNF.
```

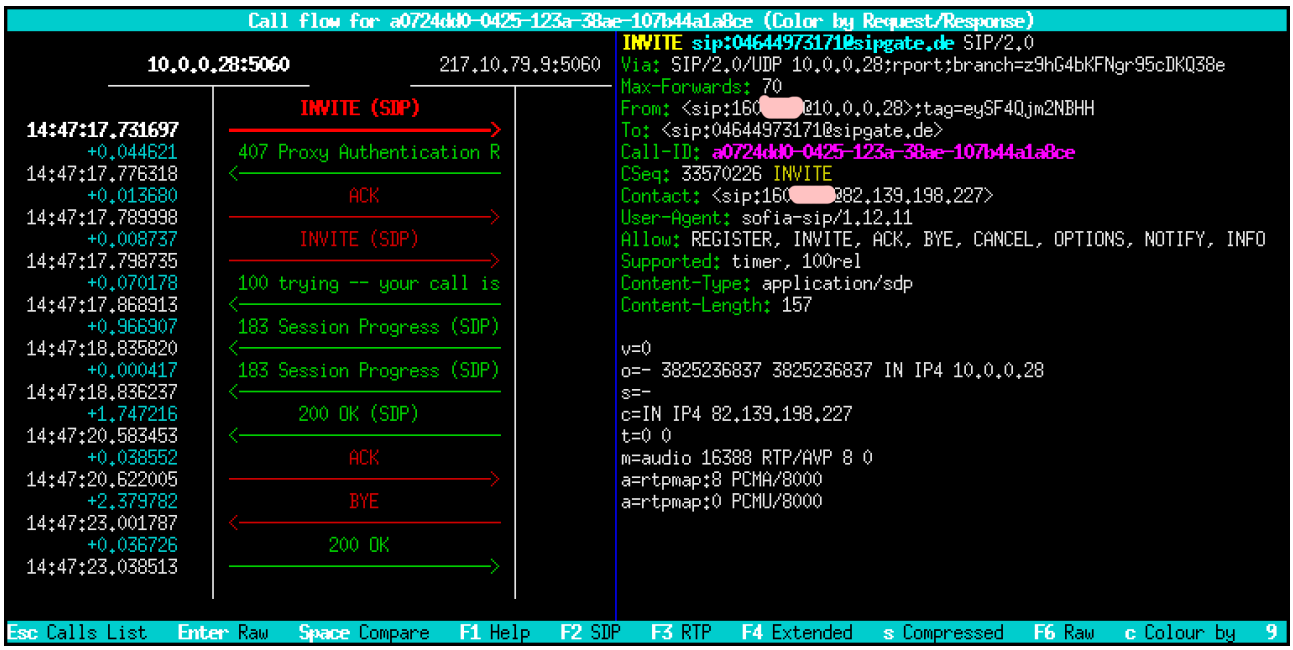
Now we can do a call by entering some number to the console of 'osmo-cc-alsa-endpoint'. We should start '**sngrep**' first to watch our call.

```
...
endpoint.c: 923 info   : Handle message CC-SETUP-REQ at state IDLE (callref 3)
sip.c: 465 info   : Sending INVITE (callref 3)
endpoint.c: 923 info   : Handle message CC-PROC-IND at state INIT-OUT (callref 3)
sip.c:1154 info   : Received INVITE response: 407 Proxy Authentication Required (callref 3)
sip.c:1154 info   : Received INVITE response: 183 Session Progress (callref 3)
nua(0x6782ad0): INVITE: ignoring duplicate SDP in 183 Session Progress
endpoint.c: 923 info   : Handle message CC-PROGRESS-IND at state PROCEEDING-OUT (callref 3)
sip.c:1154 info   : Received INVITE response: 183 Session Progress (callref 3)
endpoint.c: 923 info   : Handle message CC-PROGRESS-IND at state PROCEEDING-OUT (callref 3)
nua(0x6782ad0): INVITE: ignoring duplicate SDP in 200 OK
sip.c:1154 info   : Received INVITE response: 200 OK (callref 3)
endpoint.c: 923 info   : Handle message CC-SETUP-CNF at state PROCEEDING-OUT (callref 3)
endpoint.c: 923 info   : Handle message CC-SETUP-COMP-REQ at state CONNECTING-OUT (callref 3)
```

Finally the called party disconnects.

```
sip.c:1351 info   : Received BYE (callref 3)
sip.c:1358 info   :  -> ISDN cause: 16
sip.c:1366 info   : Sending BYE response: 200 OK (callref 3)
endpoint.c: 923 info   : Handle message CC-REL-IND at state ACTIVE (callref 3)
endpoint.c: 740 info   : Changing message to CC-DISC-IND.
endpoint.c: 923 info   : Handle message CC-REL-REQ at state DISCONNECTING-IN (callref 3)
```

We can verify our registration or call:



The graph shows our identity with our local IP in the '**From:**' line. Because we cannot be reached there, a '**Contact:**' line is added with our identity with our public IP. The dialed number is added to the '**To:**' line. The SDP message uses public IP, so that the remote peer knows where to send RTP to.

2.3.5 Example: Configure as SIP proxy

In this example we want a SIP phone (client) register to '**osmo-cc-sip-endpont**'. The machie it runs on has IP '**10.0.0.28**', so this is our Registrar and SIP Proxy. We will accept a user '**jolly**' and a password '**paSSword**'. The following configuration must be done on the client side:

- User-ID: **jolly**
- Registrar: **10.0.0.28**
- SIP Proxy: **10.0.0.28**
- Password: **paSSword**

Again, we use '**osmo-cc-alsa-endpoint**' to make and receive test calls. Because we need send and receive RTP from/to different device (our phone), we need to tell '**libosmocc**' what RTP peer we use. On the local machine we define it with '**--cc "rtp-peer 10.0.0.28"**'. Also do that for any other endpoint you want to use instead.

```
$ osmo-cc-alsa-endpoint -a default --cc "rtp-peer 10.0.0.28"
...
options.c: 268 info    : Command line option '-a' ('--audio-device'), parameter 'default'
options.c: 282 info    : Command line option '--cc', parameter 'rtp-peer 10.0.0.28'
socket.c: 381 error   : OsmoCC-Socket failed, socket cause 3.
endpoint.c: 923 info    : Handle message CC-REL-REQ at state ATTACH-SENT (callref 1)

endpoint.c: 314 info    : Attachment to remote peer "127.0.0.1:4201" failed, retrying.
on hook: ................................ (press digits 0..9 or d=dial)
```

Our local IP is '**10.0.0.28**', so we set our local peer with '**-l 10.0.0.28**'.

Because we don’t know the remote peer until it has registered, we don’t specify a remote peer.

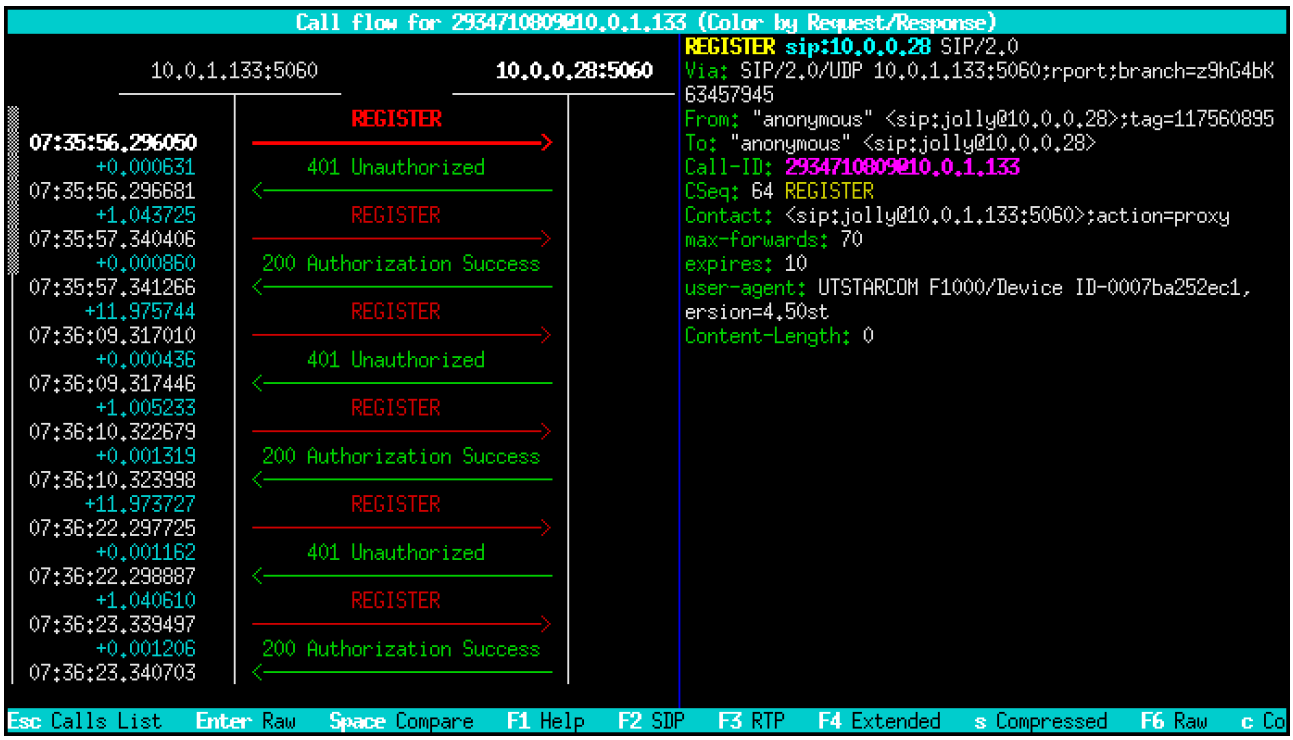
To allow the remote peer to register to us, we use command line option ‘**--local-register**’.

```
$ osmo-cc-sip-endpoint -l 10.0.0.28 –local-register
...
options.c: 268 info   : Command line option '-l' ('--local'), parameter '10.0.0.28'
options.c: 284 info   : Command line option '--local-register'
endpoint.c: 923 info   : Handle message CC-ATTACH-RSP at state ATTACH-SENT (callref 1)
endpoint.c: 299 info   : Attached to remote peer "127.0.0.1:4200".
endpoint.c: 923 info   : Handle message CC-ATTACH-REQ at state IDLE (callref 2)
endpoint.c: 381 info   : Remote peer with socket address '127.0.0.1' and port '4200' and
interface 'alsa' attached to us.
endpoint.c: 385 info   : Changing message to CC-ATTACH-CNF.
sip.c: 859 info   : Received REGISTER (registration)
sip.c: 926 info   : Sending REGISTER response: 200 Authentication not required (registration)
```

Because we did not set any authentication, any phone that registers to our peer will be accepted. This is probably not what we want. To restrict this, we need to enable authentication, so we do it with ‘**--local-auth**’. Also we must give credentials, so we do it with ‘**-A jolly ‘paSSword’ hello**’. ‘**hello**’ is our ‘**realm**’, that is used with the authentication process. In case the client wants a ‘**realm**’, make it match with our authentication parameter.

```
$ osmo-cc-sip-endpoint -l 10.0.0.28 –local-register --local-register --local-auth -A jolly
'paSSword' hello
...
options.c: 268 info   : Command line option '-l' ('--local'), parameter '10.0.0.28'
options.c: 284 info   : Command line option '--local-register'
options.c: 284 info   : Command line option '--local-auth'
options.c: 268 info   : Command line option '-A' ('--auth'), parameter 'jolly' 'pa$$word' 'hello'
endpoint.c: 923 info   : Handle message CC-ATTACH-RSP at state ATTACH-SENT (callref 1)
endpoint.c: 299 info   : Attached to remote peer "127.0.0.1:4200".
endpoint.c: 923 info   : Handle message CC-ATTACH-REQ at state IDLE (callref 2)
endpoint.c: 381 info   : Remote peer with socket address '127.0.0.1' and port '4200' and
interface 'alsa' attached to us.
endpoint.c: 385 info   : Changing message to CC-ATTACH-CNF.
sip.c: 859 info   : Received REGISTER (registration)
sip.c: 926 info   : Sending REGISTER response: 401 Unauthorized (registration)
sip.c: 859 info   : Received REGISTER (registration)
sip.c: 926 info   : Sending REGISTER response: 200 Authorization Success (registration)
```

We can verify our registration or call using ‘**sngrep**’ again:



Because my '**UTSTARCOM F1000**' registers with user '**jolly**' to the Registrar '**10.0.0.28**', it used URI '**sip:jolly@10.0.0.28**' in '**From:**' and '**To:**' lines. The '**Contact:**' line is used for inbound calls.

The first of two registrations fail, because authentication is requested with the "**401 Unauthorized**" failure message. Se second registration is successful, because the credentials match.

2.3.6 Additional Options

--send-ner

When a call is received, the SIP endpoint may send audio prior answer of the call. In case that audio is available pior answer, response to incoming call is "183 Session Progress" prior ringing ("180 Ringing"). If the remote has trouble with extra ringing message, disable it with "--send-ner".

--receive-ner

When an outgoing SIP call is made, the remote SIP endpoint may respond with "183 Session Progress", to indicate that audio is available prior answer. If it does not respond with ringing afterwards ("180 Ringing"), use "--receive-ner" to treat "183 Session Progress" as if the call is ringing.

--asserted-id user@host

If you register to a proxy, you might have a user name and a host name to register to. Then if you make calls, you might want to replace your source user name with your caller ID. This is useful if you have the ability to give any caller ID or if you have a set of phone

numbers or extensions. The proxy will not know who you are, so it cannot associate your call with your identity. To solve this, use “--asserted-id” with you identity.

--remote-nat-sip

If the remote uses NAT and source SIP address or port has changed due to NAT process: Ignore the address and port from the SIP contact. Use the address and port from where the SIP massasge was sent.

--remote-nat-rtp

Ignore the RTP address from the SDP message. Use the RTP address from the remote peer. (untested)

--public-ip <ip>

If you your SIP endpoint is located behind NAT firewall and your remote SIP peer is outside your network, you can specify the public IP that your firewall will translate local IP addresses to. If you don’t know your IP, use “--stun-server” instead.

This will not only add your contact with public IP to SIP messages, but also change contact inside SDP message. Then the remote knows where it can reach you and where to send audio data to.

--stun-server <server ip>

If you are behind a NAT firewall and have dynamic IP, use a stun server to obtain public IP address. SIP and SDP messages then use this IP as public IP address.

--register-interval <seconds>

Define interval for registering. If you experience that you unreachable between two register intervals, use shorter value. Lower value will also prevent NAT firewall from aging out.

--options-interval <seconds>

Same as above, but for option messages during a call.

--stun-interval <seconds>

Interval to ask STUN server for your current public IP.

--expires <seconds>

Set session expire timer. This will cause a re-invite. Turn off using ‘0’, which is the default value.

--user-agent <string>

Set user agent name. This string is uses as User-Agent tag in SIP messages.

--sofia-debug 9

Enables full debugging of Sofia stack. Use it, if you like to know more about why some message is transferred the way it is.

2.4 osmo-cc-misdn-endpoint

What do you want to connect to an ISDN card? If it is a phone line, a PBX or some kind of box from you internet provider, you run it in ‘TE’ mode. If you want to connect a phone, a analog converter or the external line of a PBX, you run it in ‘NT’ mode. In case read the Appendix A.

2.4.1 Installation

Installation of mISDN library ‘mISDNuser’:

```
$ cd ~
$ git clone https://github.com/ISDN4Linux/mISDNuser.git
Cloning into 'mISDNuser'...
```

Change to directory:

```
$ cd mISDNuser
```

Configure and compile:

```
$ sudo apt install automake # in case you don’t have ‘automake’ installed
$ sudo apt install gcc # in case you don’t have ‘gcc’ installed
$ sudo apt install make # in case you don’t have ‘make’ installed
$ autoreconf -if
$ ./configure
$ make clean
$ make
```

If you get ‘configure.ac:17: error: possibly undefined macro: AC_PROG_LIBTOOL’:

```
$ sudo apt install libtool
```

If you get ‘WARNING: 'flex' is missing on your system.’:

```
$ sudo apt install flex
```

Then configure and compile again. **NOTE: Don’t forget to ‘make distclean’ before you configure and compile again!**

If you get ‘bridge.c:159:2: error: array subscript 2 is outside array bounds of 'unsigned char[16]'...’, edit ‘bridge/bridge.c’:

```
$ nano bridge/bridge.c
```

Replace any ‘unsigned long’ with ‘unsigned int’. Then run ‘make’ again.

Install on your system:

```
$ sudo make install
$ sudo ldconfig
```

Installation of osmo-cc-misdn-endpoint:

```
$ cd ~
$ git clone https://gitea.osmocom.org/cc/osmo-cc-misdn-endpoint
Cloning into 'osmo-cc-misdn-endpoint'...
```

Change to directory:

```
$ cd osmo-cc-misdn-endpoint
```

Configure and compile:

```
$ autoreconf –if
$ ./configure
$ make clean
$ make
```

Install on your system:

```
$ sudo make install
```

2.4.2 Find ISDN card

Use ‘**lspci**’ to see if your ISDN card was detected. Then list your card using ‘**misdn_info**’ tool. In this case it is a ‘HFC-4S’:

```
$ lspci
...
00:04.0 ISDN controller: Cologne Chip Designs GmbH ISDN network Controller [HFC-4S] (rev 01)
...

$ sudo misdn_info

Found 4 ports
  Port  0 'hfc-4s.1-1':      TE/NT-mode BRI S/T (for phone lines & phones)
                             2 B-channels: 1-2
                               B-protocols: RAW HDLC X75slp L2:DSP L2:DSPHDLC
  --------
  Port  1 'hfc-4s.1-2':      TE/NT-mode BRI S/T (for phone lines & phones)
                             2 B-channels: 1-2
                               B-protocols: RAW HDLC X75slp L2:DSP L2:DSPHDLC
  --------
```

```
  Port  2 'hfc-4s.1-3':      TE/NT-mode BRI S/T (for phone lines & phones)
                              2 B-channels: 1-2
                                B-protocols: RAW HDLC X75slp L2:DSP L2:DSPHDLC
  --------
  Port  3 'hfc-4s.1-4':      TE/NT-mode BRI S/T (for phone lines & phones)
                              2 B-channels: 1-2
                                B-protocols: RAW HDLC X7
```

If you have a rare USB ISDN adapter, do this:

```
$ lsusb
...
Bus 001 Device 003: ID 0742:2008 Stollmann ISDN TA [HFC-S]
...

$ sudo misdn_info

Found 1 port
  Port  0 'HFC-S_USB.1':     TE/NT-mode BRI S/T (for phone lines & phones)
                              2 B-channels: 1-2
                                B-protocols: RAW HDLC X75slp
```

2.4.3 Example: Run in ‘TE’ mode

Because the ISDN port we want to use is ‘**0**’ or ‘**hfc-4s.1-1**’, we define it with ‘**-p 0**’ or ‘**-p hfc-4s.1-1**’. Because it is the fist port, we can just omit this.

Then our line we want to connect to uses ‘**multipoint**’ configuration. If it is a PBX line with DDI, we need to configure our endpoint as ‘**point-to-point**’, so we define it with ‘**--ptp**’

```
$ osmo-cc-misdn-endpoint
...
mlayer3.c:  98 notice : mISDN kernel version 1.01.29 found
mlayer3.c:  99 notice : mISDN user   version 1.01.31 found
mlayer3.c: 100 notice : mISDN library interface version 2 release 5
socket.c: 381 error  : OsmoCC-Socket failed, socket cause 3.
endpoint.c: 922 info   : Handle message CC-REL-REQ at state ATTACH-SENT (callref 1)
endpoint.c: 314 info   : Attachment to remote peer "127.0.0.1:4201" failed, retrying.
socket.c: 381 error  : OsmoCC-Socket failed, socket cause 3.
```

For testing this, we use ‘**osmo-cc-alsa-endpoint**’. We want to use sound card, so we define ‘**-a default**’ Because our phone number is ‘**123456**’, we define ‘--caller-id **123456**’:

```
$ osmo-cc-alsa-endpoint -a default -I 123456
...
options.c: 268 info   : Command line option '-a' ('--audio-device'), parameter 'default'
options.c: 268 info   : Command line option '-I' ('--caller-id'), parameter '123456'
endpoint.c: 922 info   : Handle message CC-ATTACH-RSP at state ATTACH-SENT (callref 1)
endpoint.c: 299 info   : Attached to remote peer "127.0.0.1:4201".
endpoint.c: 922 info   : Handle message CC-ATTACH-REQ at state IDLE (callref 2)
endpoint.c: 381 info     : Remote peer with socket address '127.0.0.1' and port '4201' and
interface 'HFC-S_USB.1' attached to us.
endpoint.c: 385 info   : Changing message to CC-ATTACH-CNF.
on hook: ............................... (press digits 0..9 or d=dial)
```

Now try to make or receive a call.

2.4.4 Example: Run in ‘NT’ mode

NOTE: If you want to change mode for USB, you must reload the module ‘hfcsusb’ or reboot!

Because the ISDN port we want to use is ‘**0**’ or ‘**hfc-4s.1-1**’, we define it with ‘**-p 0**’ or ‘**- p hfc-4s.1-1**’. Because it is the fist port, we can just omit this.

We want set ‘**NT**’ mode, so we define ‘**--nt**’. This only works if crossover cable an termination is used. See Appendix A for more.
The remote Osmo-CC endpoint may or may not send early/late audio. Since we always want to hear some ringback or busy tones in the phone, we must enable locally generated tones, we do it with ‘**--local-tones american**’.

```
$ osmo-cc-misdn-endpoint –nt –local-tones american
...
options.c: 284 info   : Command line option '--nt'
options.c: 284 info   : Command line option '--local-tones', parameter ‘american’
mlayer3.c:  98 notice : mISDN kernel version 1.01.29 found
mlayer3.c:  99 notice : mISDN user   version 1.01.31 found
mlayer3.c: 100 notice : mISDN library interface version 2 release 5
socket.c: 381 error  : OsmoCC-Socket failed, socket cause 3.
endpoint.c: 922 info   : Handle message CC-REL-REQ at state ATTACH-SENT (callref 1)
endpoint.c: 314 info   : Attachment to remote peer "127.0.0.1:4201" failed, retrying.
```

For testing this, we use ‘**osmo-cc-alsa-endpoint**’. We want to use sound card, so we define ‘**-a default**’:

```
$ osmo-cc-alsa-endpoint -a default
...
options.c: 268 info   : Command line option '-a' ('--audio-device'), parameter 'default'
endpoint.c: 922 info   : Handle message CC-ATTACH-RSP at state ATTACH-SENT (callref 1)
endpoint.c: 299 info   : Attached to remote peer "127.0.0.1:4201".
endpoint.c: 922 info   : Handle message CC-ATTACH-REQ at state IDLE (callref 2)
endpoint.c:  381 info     :  Remote  peer  with  socket  address  '127.0.0.1'  and  port  '4201'  and
interface 'HFC-S_USB.1' attached to us.
endpoint.c: 385 info   : Changing message to CC-ATTACH-CNF.
on hook: ............................... (press digits 0..9 or d=dial)
```

Now pick up the phone an see, if call works. Hang up and dial phone number of ISDN phone on ‘**omso-cc-alsa-endpoint**’ and call the phone.

2.4.5 Additional Options

--ulaw

Use U-LAW for b-channel coding instead of alaw. This is used in the USA.

--ptp

Use point-to-point configuration only if the remote side is a PBX line/device with DDI. If you use E1 interface, it is selected automatically, because E1 does not support multipoint.

--msn <msn1> [--msn <msn2> ...]

If using NT-Mode, you may also limit the number of phones (their numbers) by giving one or several MSN numbers. If the phone uses a number not listed or if the phone is called with an unlisted number, the first MSN is used.

--layer-1-hold 0 | 1
--layer-2-hold 0 | 1

On multipoint configuration, these layers are turned off when no call is present. For point-to-point they are kept up. Only change this, if you know what you are doing.

--channel-out [force,][<number>][,...][,free][,any][,no]

Channel selection list for all outgoing calls to the interface. A free channels is searched in order of appearance. Only change this, if you know what you are doing.

- force - Forces the selected port with no acceptable alternative. This will be automatically set for multipoint NT-mode ports.
- <number>[,...] - List of channels to search.
- free - Select any free channel
- any - On outgoing calls, signal 'any channel acceptable'. (see DSS1)
- no - Signal 'no channel available' aka 'call waiting'. (see DSS1)

--channel-in [<number>][,...][,free]

Give list of channels to select for calls from ISDN. Channel selection list for all incoming calls from the interface. A free channels is accepted if in the list. If any channel was requested, the first free channel found is selected. Only change this, if you know what you are doing.

- <number>[,...] - List of channels to accept.
- free - Accept any free channel

--timeouts <setup>,<overlap>,<proceeding>,<alerting>,<disconnect>

Alter ISDN protocol times. The default is 120 seconds for all states. Use 0 to disable. Change it only for NT-Mode, if you want to shorten the alerting duration.

--tx-delay <ms>

Give a delay in milliseconds. This is required for modem/fax. Audio toward ISDN interface is buffered with the given delay. This feature turns off the dejittering.

--tx-gain <dB>

Changes gain of audio towards ISDN interface. Give Gain in steps of
6 dB. (-48 .. 48)

--rx-gain <dB>

Changes gain of audio coming from ISDN interface. Give Gain in steps
of 6 dB. (-48 .. 48)

--pipeline <string>

mISDN allows to use echo cancellation modules. See mISDN documentation.

--dtmf 1 | 0

Turns DTMF detection on or off (default is 1).

--local-tones german | oldgerman | american

This is essential, if you want to hear tones in you phone, but the other Osmo-CC endpoint does not send early/late audio. Only useful for NT-Mode (or TE-Mode with early audio support).

--debug-misdn

Enables mISDN stack debugging.

--serving-location (see Q.931)

There is actually no use for it, except that the remote ISDN user knows where a call was disconnected. 0 = user, 1 = private network serving local user (default=1)

2.5 osmo-cc-ss5-endpoint

TBD
kurzinfo

2.6 Osmocom-Analog

TBD
- don’t use -a for console
- use -o to use socket
- use -x to connect to itself (cross connet calls)

-Example zeitansage
-Example: anruf per alsa.

3. Routing & Screening

TBD
- dialing screening
- both directions
- caller id screening
- both directions, connected id
- routing (local/remote)
- routing via screening

4. osmo-cc-router

TBD
- script erklaeren, wann und wie und bash
- script endet, neustart beim overlap
- script endet nach call, call geht weiter
- echo stdout
- echo stderr

commands:
- rtp-proxy (DTMF/transcoding/call-recording/announcements)
 what is default
 orig-codecs
 term-codecs
  PCMA PCMU L16 telephone-event

- play <name> (rtp-proxy)
  volume (factor)
  loop
- play-stop
- record <name> (rtp-proxy)
  volume (factor)
- record-stop
- tx-gain <db>, rx-gain <db>
- call

```
                if (value_of_param(argv[i], "interface", &interface));
        else if (value_of_param(argv[i], "bearer-coding", &bearer_coding));
        else if (value_of_param(argv[i], "bearer-capability", &bearer_capability));
        else if (value_of_param(argv[i], "bearer-mode", &bearer_mode));
        else if (value_of_param(argv[i], "calling", &calling));
        else if (value_of_param(argv[i], "calling-type", &calling_type));
        else if (value_of_param(argv[i], "calling-plan", &calling_plan));
        else if (value_of_param(argv[i], "calling-present", &calling_present));
        else if (value_of_param(argv[i], "calling-screen", &calling_screen));
        else if (value_of_param(argv[i], "no-calling", &no_calling));
        else if (value_of_param(argv[i], "calling2", &calling2));
        else if (value_of_param(argv[i], "calling2-type", &calling2_type));
        else if (value_of_param(argv[i], "calling2-plan", &calling2_plan));
        else if (value_of_param(argv[i], "calling2-present", &calling2_present));
        else if (value_of_param(argv[i], "calling2-screen", &calling2_screen));
        else if (value_of_param(argv[i], "no-calling2", &no_calling2));
        else if (value_of_param(argv[i], "redirecting", &redirecting));
        else if (value_of_param(argv[i], "redirecting-type", &redirecting_type));
        else if (value_of_param(argv[i], "redirecting-plan", &redirecting_plan));
        else if (value_of_param(argv[i], "redirecting-present", &redirecting_present));
        else if (value_of_param(argv[i], "redirecting-screen", &redirecting_screen));
        else if (value_of_param(argv[i], "redirecting-reason", &redirecting_reason));
        else if (value_of_param(argv[i], "no-redirecting", &no_redirecting));
        else if (value_of_param(argv[i], "dialing", &dialing));
        else if (value_of_param(argv[i], "dialing-type", &dialing_type));
```

```
                else if (value_of_param(argv[i], "dialing-plan", &dialing_plan));
                else if (value_of_param(argv[i], "keypad", &keypad));
```

- call-stop
- overlap, proceeding, alerting, answer
- disconenct, release
  cause
- dtmf
  → stdout: dtmf <digit>
- dtmf stop
- error
  → debug message

telephone-event and dtmf:
 dtmf x

Appendix

A. ISDN-Hardware

A.1 ISDN cards

The first thing needed for the PBX, is of course at least one ISDN card. At least two cards are required to interconnect internal telephones with external telephone lines. Any card can be used that is supported by the kernel's **mISDN driver**:

- HFC-S PCI based cards (tested)
- Fritzcard PCI (tested)
- Sedlbaur FAX
- Winbond
- HFC-4s / HFC-8s based cards (tested)
- HFC-E1 based cards (tested)
- HFC-USB
- …

Please mail your experience with untested cards, when using the kernel's mISDN driver.

In order to connect telephones directly to an ISDN card, it must support **NT-Mode**. Then it is possible to use it as internal ISDN port. There currently there are these type of chip sets, that support NT-Mode hardware layer:

- HFC-S PCI based cards
- HFC-4s / HFC-8s chip sets
- HFC-E1 chip set
- HFC-USB based adapters

All of these chips have a small 'Cathedral of Cologne' on the top:

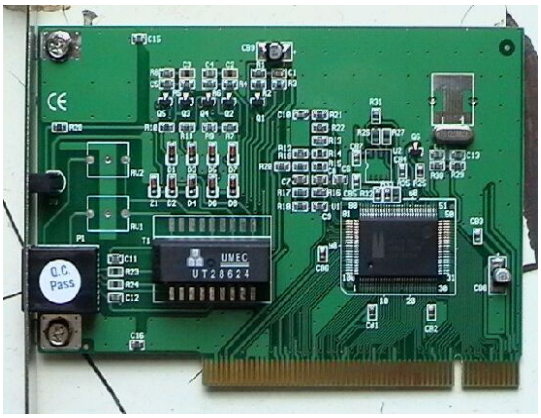
Cologne Chip

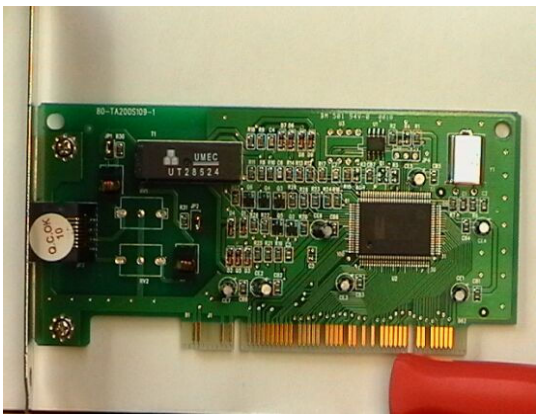
The following cards **do** have the required chip set capable of NT-Mode:

- Creatix ISDN-S0/PCI
- Trust PCI-Modem
- Acer ISDN 128 Surf PCI
- D-Link DMI-128I+

- Billion/Asuscom (Asuscom/Askey) (Beware of "BIPAC PCI SE", it has a W6692 chip that DOESN'T WORK)
- HFC cards from .Conrad Elektronik. (Bestellnummer 95 50 78)
- Neolec FREEWAY ISDNPCI (available at Reichelt.de)
- ISDN 128Kbs TA Card (TAS106H)
- HFC-4S / 8S / E1 OEM
- Junghanns Asterisk boards (HFC-4S / 8S / E1) www.junghanns.net
- Beronet Asterisk boards (HFC-4S / 8S / E1) www.beronet.com
- PBX4Linux boards (HFC-4S / 8S / E1)
- Typhoon ISDN Quick Com 128 PCI
- Arowana ISDN 128k
- Conceptronic C128i(R) (snogard.de)
- Conceptronic B128P Bulk ISDN
- Telekom Teledat 220 PCI
- Arowana 128PBS PCI
- mps Software ISLINEuni/ISLINEpro/ISLINEnote
- Micronet 128K ISDN TA Card
- Sitecom DC-105 - PCI ISDN Card
- Microcom ISDN Porte Internal (Czech site, I think)
- E-Tech PCTA128 PCI ISDN board (sold in Holland)
- Dynalink 128P PCI ISDN
- Longshine LCS-8051A (Reichelt)
- Stollmann TA USB

More cards are listed at www.mISDN.org. Please mail me other cards you know, that have an NT-Mode capable chip sets. The card will look like:





(Figure: Cards with HFC-PCI, thanx to Ulrich for the pics)

A.2 Connect ISDN telephones to your ISDN card.

Normally, cards run in **TE-Mode**, which means '**Terminal Equipment'**. Terminal equipment is e.g. an ISDN telephone, an ISDN card, a fax machine, or any other terminal to places calls, dial into the internet, or send and receive faxes. If the card runs in **NT-Mode**, it is capable of connecting terminal equipment to it. The **NT** is the small box where all ISDN terminals are connected, which means '**Network Termination'**. It is also known as 'NTBA' (Germany) or 'NT1' (everywhere else).

What must be done, to connect telephones to an ISDN card? Don't be shocked when you read the list. It is much easier, as you will find out later in the text. Here is a list of things to do:

- Of course, the card must support NT-Mode hardware layer. (all HFC chips do.)
- A driver capable of NT-mode must be installed. (part of mISDN)
- The ISDN telephone must be connected cross-over to the card.
- Power must be supplied to the ISDN bus, in case the telephone(s) has/have no own power supply.
- The ISDN bus must be terminated with resistors of 100 Ohm. (better 50 Ohm)

It is **not** possible to use just a cross-over cable for Ethernet. ISDN uses different wires in the cable than Ethernet and Fast Ethernet. Additionally, terminals and cards have no termination, because termination is normally done by NT1 or ISDN wall plug. ISDN cross connection is done by connecting the inner twisted wires (pin 4 and 5) with the twisted wires around them (pin 3 and 6). Pins 1, 2, 7 and 8 are not used. Some special PBX telephones can use these pins for extra power supply, but I don't know any telephone that supports it. Termination is essential, even if the ISDN cable would have almost zero length. All this can be done by using an old (even broken) NT1 (NTBA). Telephone companies throw them away when they are broken or customers have problems with it. Most times the power supply still works (about 9 out of 10). If you measure about 40 volts between pin 3 and 4, as well as pin 5 and 6, you know that your NT1 is good enough for using it as a power supply. Be sure that the terminator switches inside are turned on. (This is the factory default.)

An NT1 (NTBA) has three types of connectors. The first connects to the underground wire that is connected to the telephone network.[1] The second connects to the ISDN telephone(s). The third is connected to the power line. Follow the steps to get an NT1 connected to an ISDN card instead of the telephone network:

The VERY SIMPLE way:

Take an ISDN or Ethernet cable and cut it in the middle. Reconnect the inner pins (4 and 5) of one end with the pins around them (3 and 6) of the other end. Do it vice versa. (Connect 3 with 4, 4 with 3, 5 with 6, 6 with 5) Now you have an ISDN cross over cable, that is different to an Ethernet cable. Just connect the ISDN card with the cross over to one plug of your NT1, and a telephone with a normal cable (not crossed) to the other plug. Connect the power line of your NT1 or telephone and **you are done**. You will be able to connect only one telephone, because both plugs are used. Don't use the cross-over cable to directly connect your card to your telephone, unless your card or your cable has termination and your telephone has own power supply.

The COMFORTABLE way:

Step 1: Open the small door to get access to the wire clips. If it has no door, just take a screw driver and open the case. Be aware of high voltage at the switched power supply, even if it is not connected to the power line. Capacitors may hold high voltage for a long time. Inside are 4 clips to connect an ISDN cable directly to it, rather than connecting it to one of the RJ45 plugs. Each NT on the picture has six yellow clips: One pair (to the left)

[1] On the other end of the phone line, the telephone company uses something called LT (Line Termination). It works almost like the NT1, but it is connected together with many other LTs to one or more 2-mbit-interfaces. This is called the 'Access Network'. This network is connected to the public exchange.

are used to connect the U-Bus (underground wire), the two pairs (to the right) connect to the S-Bus (telephone). The S-Bus is directly connected in parallel to the two rj45 plugs.

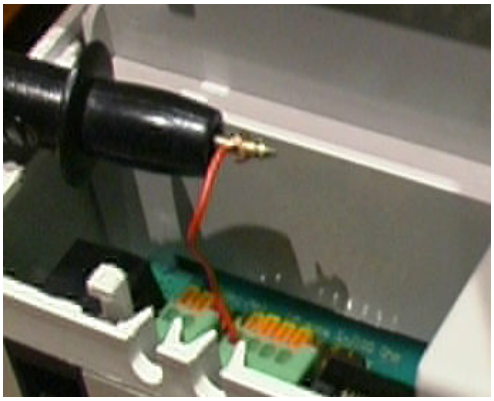
(Figure: NT1 with door open)

Step 2: Find out which clip inside the NT1 is connected to which pin on the rj45 plug. Therefore connect an ISDN cable to one of the two ISDN jacks. Take an Ohm-Meter and find out which clip is connected to which pin of the ISDN cable. Use a small piece of wire, to help the meter's probe get into the hole of the clip. Pin 4 and 5 as well as 3 and 6 of the ISDN cable, is connected through the coil inside the NT1. The coil has low resistance, so your Ohm-Meter will show that they are connected (almost 0 Ohms). So you cannot determine between pin 4 and 5 as well as between pin 3 and 6. There are two clips connected to pin 4 and 5, and another two connected to 3 and 6. It doesn't matter which clip is connected to pin 4 and which is connected to pin 5, since the **polarity doesn't matter**. (pin 3 and 6 respectively)

Most NT1 in Europe name the 4 pins: A1 and B1, A2 and B2[2]. Do the following connections for crossing send and receive pair:

- A1 -> pin 3
- B1 -> pin 6
- A2 -> pin 4
- B2 -> pin 5

[2] German Telekom (former German state-owned telephone company) counted their twisted pairs in their cables. A1 mens, the wire A of twisted pair 1. The white or the cable with less black rings is wire A, the other is wire B.

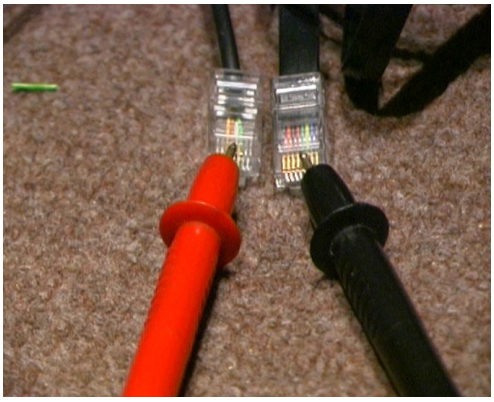



(Figure: Find clips)

Step 3: Take another ISDN cable or Ethernet cable and cut off one end. Take the inner pair of the cable (pin 4 and 5) and connect them to the clips, that are connected to the outer pins of the ISDN jacks (pin 3 and 6). Do this with the inner pins of the cable respectively. Again, polarity of a cable pair doesn't matter for the ISDN bus.


(Figure: cable connected to the clips)

Step 4: Use the Ohm-Meter again and verify the connection between the cable, that is connected to the clips, and the cable connected to one of the jacks. Pin 4 and 5 should now connect to pin 3 and 6 and vice versa.

(Figure: Verify cross connection)

Step 5: Connect the cable, which is connected to the clips, to the ISDN card, that should run in NT-Mode. Be sure that termination is switched on. Inside the NT1 are two switches, that must be ON. This is the default.

(Figure: cable connected to the ISDN card)

Step 6: Connect an ISDN telephone to the ISDN cable, that is connected to the ISDN jack. Also connect the power cable.

(Figure: complete setup with one phone)

Since there is nothing connected to the U-Bus (underground wire), the internal electronics is disabled[3], because it gets its power from the U-Bus (not from the power line). Only the power supply is connected to the coils (transformer) inside the NT1 providing power for telephones, as well as termination. This is why the NT1 can be broken. Only the power supply must work.

[3] Some new NT1 will also run their electronics with the integrated power supply, so no power feeding is required. (I found this on Sphairon's NT1 + Split boxes.) In this case, I would suggest to disconnect the internal coil from the NT1 controller. If you are not into electronics like that, just use an old NT1, it will work.

NOTE: If you like to connect more than one telephone, you might experience clicks and other audio problems. Use 50 ohm instead of 100 ohm termination. This can be done by adding another 100 ohm resistors in parallel with the ones in the NT1. Two 100 ohm resistors in parallel become 50 ohm! One resistor must be connected between pin 3 and 6, and the other between 4 and 5. You must figure out yourself how to connect the cable plus the resistors. Maybe you use a soldering iron or stick both into the holes of the clips.

Alternatively you may get an ISDN termination plug. Use the plug between one jack of the NT1 and the isdn phone or between NT1 and ISDN card. It will also lower the ISDN termination down to 50 ohm.