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 that use Osmo-CC run on the same machine. In cause connecting application via Osmo-CC over a network, 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. Endpoints

2.1 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.

2.1.1 Installation

```
$ cd ~
$ git clone git://git.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
```

2.1.2 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.

2.1.3 Using a Headset

To use audo, 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
```

2.2 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.

2.2.1 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 git://git.osmocom.org/cc/osmo-cc-sip-endpoint
Cloning into 'osmo-cc-sip-endpoint'...
```

Change to directory:

```
$ 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
```

2.2.2 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.

2.2.3 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.
...
```

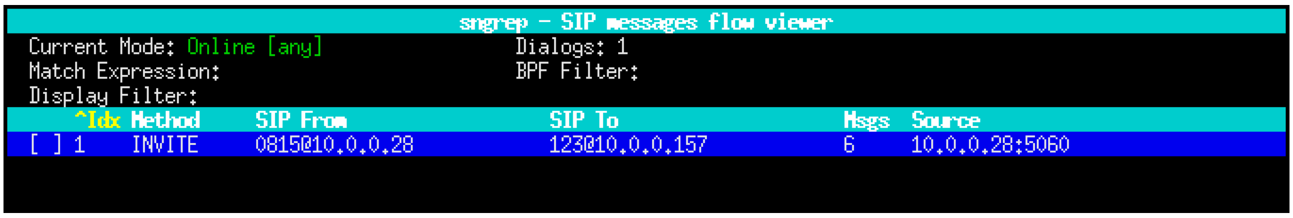
To debug a SIP call, run ‘**sngrep**’ on one of the machines in another terminal. Now press ‘d’ to dial the on machine 1 at console of ‘**osmo-alsa-endpoint**’.

```
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.

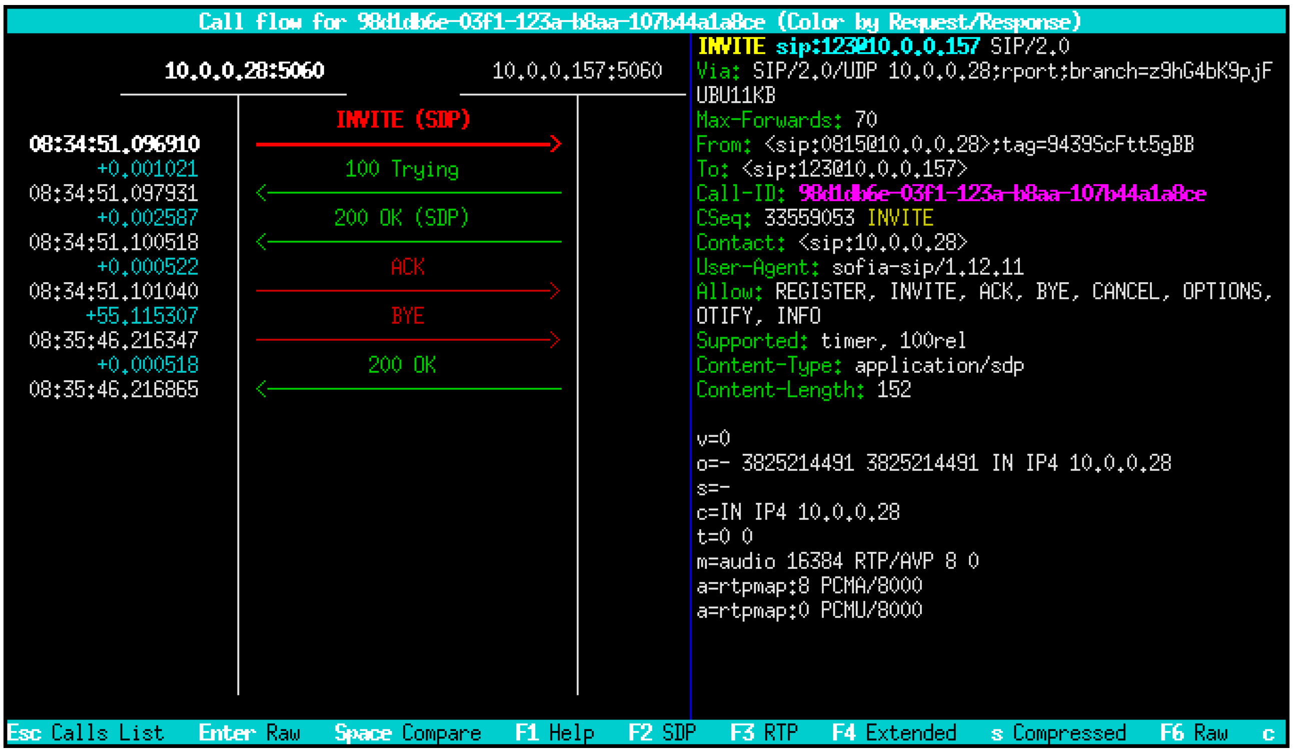
```
telephone.c: 686 info   : Call hangup
endpoint.c: 923 info   : Handle message CC-REL-IND at state ACTIVE (callref 361)
endpoint.c: 740 info   : Changing message to CC-DISC-IND.
endpoint.c: 923 info   : Handle message CC-REL-REQ at state DISCONNECTING-IN (callref 361)
on hook: 123..............................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

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



```
                          sngrep - SIP messages flow viewer
 Current Mode: Online [any]                     Dialogs: 1
 Match Expression:                              BPF Filter:
 Display Filter:
     ^Idx Method     SIP From                 SIP To                   Msgs  Source
 [ ] 1    INVITE     0815@10.0.0.28           123@10.0.0.157           6     10.0.0.28:5060
```

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



```
              Call flow for 98d1db6e-03f1-123a-b8aa-107b44a1a8ce (Color by Request/Response)
                                                            INVITE sip:123@10.0.0.157 SIP/2.0
              10.0.0.28:5060            10.0.0.157:5060     Via: SIP/2.0/UDP 10.0.0.28;rport;branch=z9hG4bK9pjF
                                                            UBU11KB
                              INVITE (SDP)                  Max-Forwards: 70
 08:34:51.096910        ──────────────────────>             From: <sip:0815@10.0.0.28>;tag=9439ScFtt5gBB
       +0.001021              100 Trying                    To: <sip:123@10.0.0.157>
 08:34:51.097931        <──────────────────────             Call-ID: 98d1db6e-03f1-123a-b8aa-107b44a1a8ce
       +0.002587              200 OK (SDP)                  CSeq: 33559053 INVITE
 08:34:51.100518        <──────────────────────             Contact: <sip:10.0.0.28>
       +0.000522                 ACK                        User-Agent: sofia-sip/1.12.11
 08:34:51.101040        ──────────────────────>             Allow: REGISTER, INVITE, ACK, BYE, CANCEL, OPTIONS,
      +55.115307                 BYE                        OTIFY, INFO
 08:35:46.216347        ──────────────────────>             Supported: timer, 100rel
       +0.000518                200 OK                      Content-Type: application/sdp
 08:35:46.216865        <──────────────────────             Content-Length: 152

                                                            v=0
                                                            o=- 3825214491 3825214491 IN IP4 10.0.0.28
                                                            s=-
                                                            c=IN IP4 10.0.0.28
                                                            t=0 0
                                                            m=audio 16384 RTP/AVP 8 0
                                                            a=rtpmap:8 PCMA/8000
                                                            a=rtpmap:0 PCMU/8000

Esc Calls List   Enter Raw   Space Compare   F1 Help   F2 SDP   F3 RTP   F4 Extended   s Compressed   F6 Raw   c
```

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.

2.2.4 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**
Outbound Proxy:	**sipgate.de**
Password:	**pa$$word**

In this example, we use ‘**osmo-cc-alsa-endpoint**’ to make a test call. Because we need send and receive RTP from 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 the remote SIP gateway to know where to reach us. 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 ‘**pa$$word**’ and do it with ‘-**A 160XXXX pa$$word 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 pa$$word
dummy -P 82.139.198.227
...
options.c: 268 info   : Command line option '-l' ('--local'), parameter '10.0.0.28'
options.c: 268 info   : Command line option '-r' ('--remote'), parameter '160XXXX@sipgate.de'
options.c: 268 info   : Command line option '-R' ('--register'), parameter '160XXXX@sipgate.de'
options.c: 268 info   : Command line option '-A' ('--auth'), parameter '160XXXX' 'pa$$word'
'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 pa$$word
dummy -S stun.sipgate.net
...
options.c: 268 info   : Command line option '-l' ('--local'), parameter '160XXXX@10.0.0.28'
options.c: 268 info   : Command line option '-r' ('--remote'), parameter 'sipgate.de'
options.c: 268 info   : Command line option '-R' ('--register'), parameter '160XXXX@sipgate.de'
options.c: 268 info   : Command line option '-A' ('--auth'), parameter '160XXXX' 'pa$$word'
'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.
```

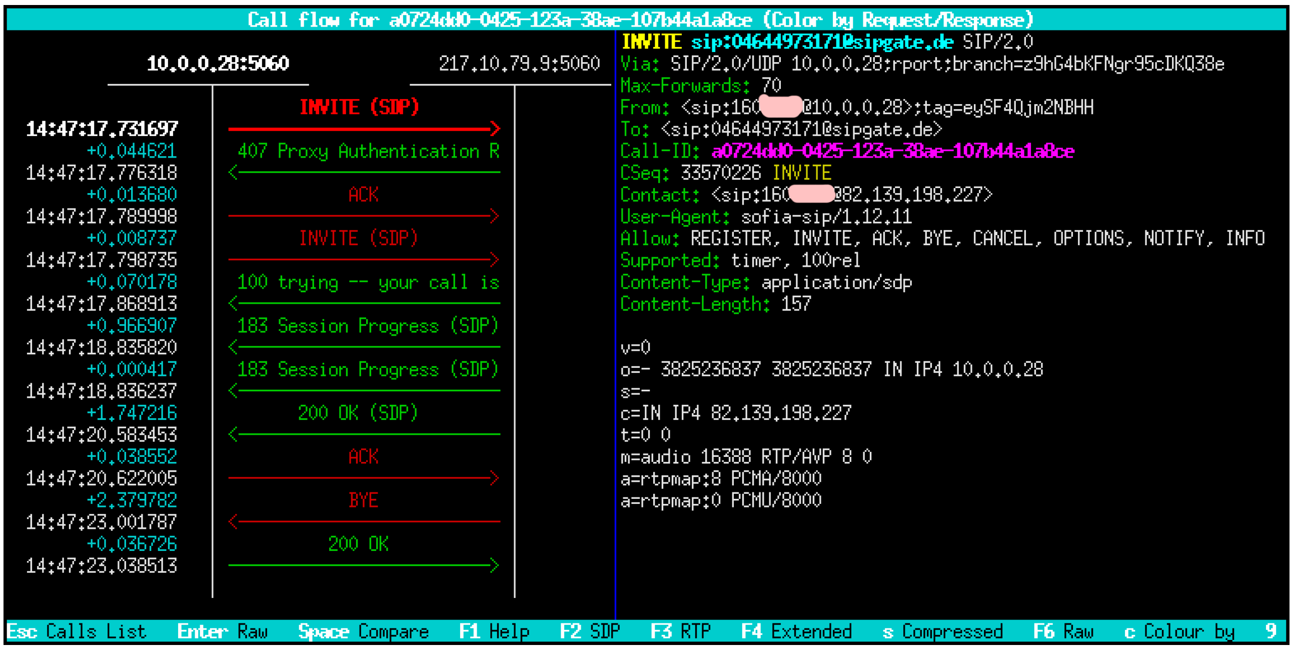
Now we can do a call by entering some number to the console of ‘osmo-cc-alsa-endpoint’. We should start ‘**sngrep**’ first to watch our call.

```
...
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:



```
Call flow for a0724dd0-0425-123a-38ae-107b44a1a8ce (Color by Request/Response)

          10.0.0.28:5060            217.10.79.9:5060
                      INVITE (SDP)
14:47:17.731697   ──────────────────────────────>
       +0.044621   407 Proxy Authentication R
14:47:17.776318   <──────────────────────────────
       +0.013680          ACK
14:47:17.789998   ──────────────────────────────>
       +0.008737      INVITE (SDP)
14:47:17.798735   ──────────────────────────────>
       +0.070178   100 trying -- your call is
14:47:17.868913   <──────────────────────────────
       +0.966907   183 Session Progress (SDP)
14:47:18.835820   <──────────────────────────────
       +0.000417   183 Session Progress (SDP)
14:47:18.836237   <──────────────────────────────
       +1.747216       200 OK (SDP)
14:47:20.583453   <──────────────────────────────
       +0.038552          ACK
14:47:20.622005   ──────────────────────────────>
       +2.379782          BYE
14:47:23.001787   <──────────────────────────────
       +0.036726         200 OK
14:47:23.038513   ──────────────────────────────>
```

```
INVITE sip:04644973171@sipgate.de SIP/2.0
Via: SIP/2.0/UDP 10.0.0.28;rport;branch=z9hG4bKFNgr95cDKQ38e
Max-Forwards: 70
From: <sip:160[redacted]@10.0.0.28>;tag=eySF4Qjm2NBHH
To: <sip:04644973171@sipgate.de>
Call-ID: a0724dd0-0425-123a-38ae-107b44a1a8ce
CSeq: 33570226 INVITE
Contact: <sip:160[redacted]@82.139.198.227>
User-Agent: sofia-sip/1.12.11
Allow: REGISTER, INVITE, ACK, BYE, CANCEL, OPTIONS, NOTIFY, INFO
Supported: timer, 100rel
Content-Type: application/sdp
Content-Length: 157

v=0
o=- 3825236837 3825236837 IN IP4 10.0.0.28
s=-
c=IN IP4 82.139.198.227
t=0 0
m=audio 16388 RTP/AVP 8 0
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
```

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.

2.2.5 Example: Configure as SIP proxy

TBD

2.2.6 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 number 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”.

--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.

2.3 osmo-cc-isdn-endpoint

TBD
- verweis Siehe anhang
- nt or te
- ptp or ptmp
- ptmp->msn
- ptp->l1 / l2 hold
- fax? –tx-delay

2.4 osmo-cc-ss5-endpoint

TBD
kurzinfo

2.5 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

- 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

TBD