FreeSWITCH And The Opus Audio Codec

mod_opus

Document version 1.0

FreeSWITCH version 1.6.12

October 2016

Dragos Oancea
droancea@yahoo.com

Giacomo Vacca
giacomo.vacca@gmail.com

Table of Contents

Introduction

FreeSWITCH is an open source telephony platform supporting communication technologies such as SIP and WebRTC. Opus is an audio codec with high resilience to packet loss, thanks to its Forward Error Correction (FEC) techniques.
Opus encoding and decoding are supported in FreeSWITCH through its *mod_opus* module.

Opus is a required coded for WebRTC, thus guaranteeing interoperability across different implementations. Its adaptability makes it suitable for applications running on high-range laptops or smartphones, either standalone or integrated into web browsers.

FreeSWITCH support for Opus is independent from the endpoint in use, so it's available for both generic SIP endpoints and verto-based (WebRTC) clients.

Besides being a high quality and low latency audio codec, the main features of Opus for VOIP are FEC and the ability to negotiate asymmetry of audio streams.

The *mod_opus* module supports two variants of Opus: one at 48 KHz which can be used for WebRTC and it's present in the default configuration, and one which enforces the sampling rate at 8000 Hz and can be used for transcoding with widely used 8000 Hz codecs with as little CPU consumption as possible.

Using Opus in FreeSWITCH

Opus can be loaded just like the other codecs in FreeSWITCH.

The codec string can be one of these :

- opus@48000h@10i - Opus 48khz using 10 ms ptime (mono and stereo)
- opus@48000h@20i - Opus 48khz using 20 ms ptime (mono and stereo)
- opus@48000h@40i - Opus 48khz using 40 ms ptime
- opus@8000h@10i - Opus 8khz using 10 ms ptime (mono and stereo)
- opus@8000h@20i - Opus 8khz using 20 ms ptime (mono and stereo)
- opus@8000h@40i - Opus 8khz using 40 ms ptime
- opus@8000h@60i - Opus 8khz using 60 ms ptime
- opus@8000h@80i - Opus 8khz using 80 ms ptime
- opus@8000h@100i - Opus 8khz using 100 ms ptime
- opus@8000h@120i - Opus 8khz using 120 ms ptime

The format of the codec string is:

*CODEC_NAME@SAMPLE_RATE**h**@PTIME**i@**CHANNELS**c***

Note the "h" , "i" and "c" tokens, "h" for sample rate , "i" for ptime, "c" for number of channels.

Examples

<action application="set" data="codec_string=OPUS"/>

This is default in FreeSWITCH and it means OPUS@48000h@20i@1c . Its use is recommended.

<action application="set" data="codec_string=OPUS@8000h@20i"/>

Use this if you transcode to/from PCMA or PCMU or any other 8khz codec.

<action application="set" data="codec_string=OPUS@8000h@20i,OPUS@8000h@40i, OPUS@8000h@60i,OPUS@8000h@80i"/>

You can use this if your VOIP software wants one of those ptimes.

<action application="set" data="codec_string=OPUS@8000h”/>

This implies 20 ms ptime.

User Manual

Opus is configured through the config file *PATH_TO_CONFIG/autoload_configs/opus.conf.xml*.

There are FMTP params (they will be sent inside SDP to the remote peer), encoder only params, and decoder only params. Some of the decoder params are also FMTP params and FMTP encoder params are also FMTP params. The FMTP params are per RFC 7587.

The encoder params are:

- maxaveragebitrate (FMTP)
- sprop-maxcapturerate (FMTP)
- use-vbr (FMTP)
- use-dtx (FMTP)
- complexity
- keep-fec-enabled
- packet-loss-percent

The decoder params are:

- maxplaybackrate (FMTP)
- use-jb-lookahead

We are referencing the local encoder and the local decoder. Some of these params when sent to a peer (through SDP offer/answer) change their roles and a local encoder param will become a remote decoder param. The FMTP params are also marking characteristics of the local encoder and are they are shaping its functionality.

Configuration Parameters

We are showing the configuration parameters with the hardcoded values from mod_opus.c. If one of these params is not set in in the config file (opus.conf.xml), then it will have a default value as shown below. These values are not the ones from the default config file .

"use-vbr" setting - set OPUS in VBR mode - consume less network bandwidth . VBR is OPUS's native mode. This is a boolean value , it can be 0 or 1 . The hardcoded default is 0.

<param name="use-vbr" value="0"/>

"use-dtx" setting - do not use DTX because not necessarily the remote client/user app supports it. This is a boolean value, it can be 0 or 1. The hardcoded default is 1.

<param name="use-dtx" value="0"/>

"complexity" setting - this is a codec specific setting. The assumption is that the user has a powerful machine to do transcoding and the complexity (roughly degree of compression) can be 10 which is the highest.

<param name="complexity" value="10"/>

"maxaveragebitrate" setting (in bps) – this is the bitrate of the encoder , it can have any value between 6000 and 510000 bps. This is an average value and at times during encoding the real bitrate can vary around this value depending on the encoded information . See VBR.

Note that if the remote sends us a maxaveragebitrate of X and we have in our config file a value of Y, then we configure our encoder with the smallest of these values: min (X,Y).

For OPUS@8000h this bitrate should be chosen so that FEC is present in the payload whenever the Opus codec decides it can add it. There is a threshold bitrate for FEC activation in OPUS: 12400 bps. Lower than this bitrate , there will be no FEC.

<param name="maxaveragebitrate" value=" 30000"/>
for OPUS@48000h, or
<param name="maxaveragebitrate" value=" 14000"/>
for OPUS@8000h.

"maxplaybackrate" setting – this is the sample rate of encoder in Hz.
This will tell the remote party that it's pointless to send us higher rates, because FS will downsample anyway. Valid values are: 8000, 12000, 16000, 24000, 48000. If we use Opus@8000h it's better to tell the remote party to use 8000 Hz sample rate to spare us some CPU. Besides configuring the local encoder this value is also sent to the remote party as FMTP param.

Abbreviation	Audio bandwidth	Sample rate
NB (NARROWBAND)	4000 hz	8000 hz
MB (MEDIUMBAND)	6000 hz	12000 hz
WB (WIDEBAND)	8000 hz	16000 hz
SWB (SUPERWIDEBAND)	12000 hz	24000 hz
FB (FULLBAND)	20000 hz	48000 hz

FreeSWITCH (in mod_opus) supports only NARROWBAND and FULLBAND at initialization time because these are most widely used for VOIP. But if “maxplaybackrate” and “sprop_maxcapturerate” are adjusted (FTMP and/or config) and the “asymmetric-sample-rates” config param is enabled , then the user can change the sample rate and consequently the frame size of the encoder .
The OPUS@48000h decoder can play voice payloads encoded at any of the Opus sample rates.
The OPUS@8000h decoder will down sample by default to 8000 Hz but it will still decode any sample rate.

<param name="maxplaybackrate" value="48000"/>
for OPUS@48000h or
<param name="maxplaybackrate" value="8000"/>
for OPUS@8000h.

“sprop-maxcapturerate”: this is the maximum input sampling rate to tell to the remote receiver. It

can have values: 8000, 12000, 16000, 24000, 48000 in Hz .
We are not setting it if we use OPUS@48000h simply because the sender is assumed to have no limitations by default so it will just default to 48000 Hz.
For OPUS@8000h this is being set in FMTP and so we're telling the remote we don't plan any downsampling.

<param name="sprop-maxcapturerate" value="0"/>
for OPUS@48000h or
<param name="sprop-maxcapturerate" value="8000"/>
for OPUS@8000h.

"packet-loss-percent" setting – this is the initial packet loss that we pass to the encoder.
This value in conjunction with the maxaveragebitrate value will make the codec decide if it will use FEC or not.

<param name="packet-loss-percent" value="20"/>

"keep-fec-enabled" setting – this will enforce the encoder to use FEC regardless of the maxaveragebitrate supplied to it. In fact it will recalculate a new bitrate based on the packet loss percentage taken from RTCP arriving to us from the user app if the packet loss is higher than 10%.

You might expect that this bitrate would be lower than the maxaveragebitrate, but in reality the actual bitrate will vary with packet loss during the call - but only to keep FEC enabled. This will override the maxaveragebitrate value. But if there's no more packet loss during the call then the bitrate will go to the supplied maxaveragebitrate (the one we set from remote fmtp or the local config file). This is n-1 FEC.

This feature works in today's networks, although one can ask how come the call improves if we actually increase the bitrate under packet loss conditions. Here's an explanation. For example a congested WiFi router or core router will drop a 60 bytes packet (40 bytes IP/UDP/RTP header + 20 bytes voice payload) as easy as an 70 bytes packet , so if you increase the bitrate with 10 bytes per packet (let's say that this is the price to pay of having FEC enabled) will not matter that much for that congested router, it will continue to forward some packets and drop some packets in the same manner. It's a packet-loss pattern (many small gaps).

The usage of FEC will improve the audio quality because there will be redundant information in the packets eventually arriving at the decoder after the congestion adventure. This is just an example, don't take these values above exactly as they are (60, 70 , 10). Only 40 bytes for header overhead is a correct value and it's the only fixed value in calculating network bitrate. "maxaveragebitrate" is an average , and the actual network bandwidth used might be a little bit higher or a little bit smaller, depending on the content being encoded. The voice payload size varies (because Opus is VBR) and so does the FEC payload inside the voice payload.

<param name="keep-fec-enabled" value="1"/>

"use-jb-lookahead" setting – this enables FEC decoding in the decoder.
The jitter buffer must be enabled for FEC decoding to work. This setting has impact on decoding received FEC - it will improve the audio quality greatly under packet loss conditions. We decode incoming FEC by looking in the jitter buffer, checking if we have the next packet (only if the

current packet is missing due to packet loss) and then checking if it has FEC info in it. If it has FEC, then first we decode the FEC and then we decode the main payload.

<param name="use-jb-lookahead" value="1"/>

Add these in the dialplan to enable the jitter buffer:

<action application="set" data="jitterbuffer_msec=2p:25p:"/>
<action application="set" data="rtp_jitter_buffer_plc=true"/>
<action application="set" data="rtp_jitter_buffer_during_bridge=true"/>
<action application="set" data="suppress_cng=true"/>

"advertise-useinbandfec" setting - This is used to toggle "useinbandfec" parameter in fmtp.
We tell the remote party it's okay to send us voice payloads with FEC. If we don't have a jitter buffer activated we will just ignore the voice payloads with FEC, and play only the main part of the voice payload.

<param name="advertise-useinbandfec" value="1"/>

"asymmetric-sample-rates", when set to 1, will tell FreeSWITCH to evaluate the remote party's sprop-maxcapturerate and maxplaybackrate when deciding how to initialize the opus codec.

If we imagine an entity with local parameters:

maxplaybackrate=Da; sprop-maxcapturerate=Ea; maxaveragebitrate=Fa

and remote parameters:

maxplaybackrate=Db; sprop-maxcapturerate=Eb; maxaveragebitrate=Fb

then this entity can set the decoder at a sample rate of min(Da, Eb) and the encoder at a sample rate of min(Ea, Db) and bitrate at Fb.
Similarly and intuitively, the other entity involved can set the decoder at a sample rate of min(Db, Ea) and the encoder at a sample rate of min(Eb, Da) and bitrate Fa.

<param name="asymmetric-sample-rates" value="1"/>

"negotiate-bitrate", when set to 1, will tell FreeSWITCH to evaluate the remote party's maxaveragebitrate when deciding how to initialize the opus encoder. This is based on FMTP exchange.

<param name="negotiate-bitrate" value="1"/>

This is available from FreeSWITCH v1.6.12.

"adjust-bitrate " setting – this will enable a feedback loop using remote RTCP with the encoder. The encoder will adjust its bitrate based on what the other side reports (packet loss and RTT). It is a form of congestion control and it's based on estimators and detectors recently added in FreeSWITCH. It should work seamlessly with the other settings that can change the bitrate.

```
<param name=”adjust-bitrate” value=”0”/>
```

This is available starting from FreeSWITCH v1.6.11.

Examples of configuration

WebRTC configuration

For opus@48000h:

```
<configuration name="opus.conf">
   <settings>
      <param name="use-vbr" value="1"/>
      <param name="use-dtx" value="1"/>
      <param name="complexity" value="10"/>
      <param name="packet-loss-percent" value="15"/>
      <param name="keep-fec-enabled" value="1"/>
      <param name="use-jb-lookahead" value="1"/>
      <param name="maxaveragebitrate" value="64000"/>
      <param name="maxplaybackrate" value="48000"/>
      <param name="sprop-maxcapturerate" value="48000"/>
      <param name="adjust-bitrate" value="1"/>
   </settings>
</configuration>
```

Transcoding only configuration

For opus@8000h:

```
<configuration name="opus.conf">
   <settings>
      <param name="use-vbr" value="1"/>
      <param name="use-dtx" value="0"/>
      <param name="complexity" value="10"/>
      <param name="maxaveragebitrate" value="14400"/>
      <param name="maxplaybackrate" value="8000"/>
      <param name="packet-loss-percent" value="15"/>
      <param name="keep-fec-enabled" value="1"/>
      <param name="use-jb-lookahead" value="1"/>
      <param name="advertise-useinbandfec" value="1"/>
   </settings>
</configuration>
```

SDP Offer/Answer for Opus streams

This part refers generically to the SDP negotiation involved when dealing with Opus, but it's relevant to understand how FreeSWITCH works.

Both in standard SIP architectures and other WebRTC implementations, where the signaling system may be different than SIP, the offer/answer model is based on SDP ([RFC 4566]).

When negotiating an Opus session it's important to distinguish between the encoder and decoder properties. The caller ("offerer") will indicate its willingness to use Opus (potentially with a list of other supported codecs, with a specific priority) with some optional parameters related to the encoding or decoding phase, to optimize resource utilization. The callee ("answerer") will respond, potentially using some optional parameters to refine the overall session configuration.

Mandatory parameters

Media type: must be "*audio*"
Media subtype: defined as rtpmap, must be "*opus/48000/2*". 48000 refers to the sample rate. 2 refers to the number of channels. The media subtype doesn't change if the sample rate being negotiated is not 48000 or if there's no intention to use 2 channels. In other words that media subtype is always that string.

Optional parameters

Decoder parameters

ptime: media type parameter indicating the desired frame size (in msec) that the decoder is willing to receive. This refers to the initial negotiation only, as it may change dynamically during a call. When not indicated, it defaults to 20 msec. When present, ptime is indicated in an "a=" attribute.

maxptime: media type parameter indicating the maximum frame size (in msec) that the decoder is willing to receive. When not indicated, it defaults to 120 msec. When present, maxptime is indicated in an "a=" attribute.

maxaveragebitrate: media type parameter indicating the maximum average bit rate (in bps) that the decoder is willing to receive. Only values inside the Opus bitrate spectrum (6000 to 510000 bps) are allowed. When not indicated, the maximum (510000) is assumed. With this parameter the decoder indicates that optimal performance will be met if the received bitrate, on average, doesn't go beyond this value. The decoder though is assumed to be able to handle any legal bitrate. This is useful to ensure the receiving bandwidth is not saturated, e.g. in case of mobile apps with bad network conditions.

maxplaybackrate: media type parameter indicating the maximum playback sample rate (in samples/sec) the decoder is capable of handling. Encoding at sample rates higher than this parameter should not be done, as it's expected to impact negatively the bandwidth usage. For example, when an entity indicates maxplaybackrate at 8000, the encoder can save resources and network bandwidth by instantiating at 8000 Hz instead of 48000 Hz.

stereo: media parameter indicating the ability of the decoder to support stereo. It can be 0 or 1. When 0 (which is also the default value) is specified, then the encoder should not produce stereo audio, as it would waste processing and bandwidth resources.

cbr: media parameter indicating the preference from the decoder to receive a constant (CBR) or variable (VBR) bitrate. It can be 0 or 1, and it defaults to 0 (variable bitrate).

useinbandfec: media parameter indicating that the decoder is able to process in-band FEC packets. It can be 0 (impossible to handle FEC, default behaviour) or 1 (able to process FEC). When 0 is provided, the encoder should disable the production of FEC packets as it would impact negatively (it will use more network bandwidth without any gain on audio quality).

usedtx: media parameter indicating that the decoder prefers discontinuous transmission (DTX), i.e. the encoder should enable DTX and stop sending packets when source audio is absent. It can be 0 or 1, and defaults to 0 (continuous transmission).

Encoder parameters

sprop-maxcapturerate: media parameter indicating that the encoder won't capture at a sample rate higher than that value. This may hint the remote decoder to save computation resources, however the remote decoder is expected to handle correctly any legal sample rate.

sprop-stereo: media parameter indicating that the encoder won't produce stereo audio. This may hint the remote decoder to intialize at a lower complexity, as it doesn't need the full Opus capabilities, and hence save resources.

SDP offers examples

Generic offer

```
m=audio 54312 RTP/AVP 101
a=rtpmap:101 opus/48000/2
```

where 54312 is the candidate RTP port, 101 is the dynamic number assigned to opus, and opus is offered without any specific constraint. All the default values mentioned in the session about SDP negotiation apply.

Narrowband offer

Offer with 8 KHz sample rate, bitrate limitation on the decoder (max average bitrate requested 20 Kbps) and indications on the desired ptime (40 msec preferred and max):

```
m=audio 54312 RTP/AVP 101
a=rtpmap:101 opus/48000/2
a=fmtp:101 maxplaybackrate=8000; maxaveragebitrate=14000
a=ptime:40
a=maxptime:40
```

With such an offer incoming to FreeSWITCH, if negotiate-bitrate is set to 1 then FreeSWITCH will consider the offered value of maxaveragebitrate and use it to set the initial encoder bitrate if that value is smaller than the one configured in maxaveragebitrate configuration parameter in opus.conf.xml. All this will be anyway conditional to the packet-loss-percent value set in opus.conf.xml, since the call is of the narrowband type. So the initial bitrate for the encoder may be lower than the requested 14000 bps in this example.

Development

FEC on the decoder

The function *switch_opus_decode()* is called for each Opus frame to be decoded.

If the frame is missing , then the flag SFF_PLC is set by the Jitter Buffer, and the decoder either does FEC (if the next frame is present and it has FEC in it), or PLC.

Function *switch_opus_has_fec()* has to return correctly in order for FEC decoding to work.

This function uses API function *opus_packet_parse()* to break the frame in multiple Opus frames, if it's the case (eg: for ptimes higer than 60 ms, repacketization must be used, and we'll have frames with multiple Opus frames).

Inside an Opus frame there can be multiple SILK frames. Each of these frames can contain FEC information. This function looks at each SILK frame to see if there's FEC information in it , and it returns TRUE if it finds any. This is indispensable for the way the Opus (n-1) FEC works together with the FS jitter buffer because we'll know if we need to grab from the jitter buffer the same packet twice: once for playing FEC from it, and once to play the current voice payload itself.

FEC on the encoder

If packet loss is less or equal to 10% , we don't enforce FEC, but we still pass the packet loss percentage value to the encoder, so it would make some smart decisions on how to encode the voice further.

If packet loss is over 10% , we enforce FEC by increasing the bitrate slightly using code from the Opus encoder itself. Inside the Opus encoder, the decision to encode FEC is taken based on packet loss percentage and bitrate. If the bitrate is too low (under 12400 bps for Narrowband – which is a builtin threshold in Opus), there will be no encoded FEC. There are 25 threshold bitrates.

Function *switch_opus_keep_fec_enabled()* gets the current bitrate and the current packet loss via Opus API. Then it calculates the closest bitrate to the current one for which there's FEC at the given packet loss, and after that it sets this new bitrate back on the encoder.

There is also a function called *switch_opus_get_fec_bitrate()*, which contains a map, and it returns a value from the map, given a certain packet loss. The values from the map are pre-calculated because this function is only used at the beginning of the call, when the encoder is not instantiated yet and there is no information on how much packet loss the call will have so it's better to start it with FEC enabled. The map applies for NB only, but the function can be used with any valid rate for FEC (NB, MB, WB).

Physical explanation, on why this feature works: in case of network congestion, the network queues in remote devices start dropping entire packets, they would never drop just some bits or bytes from a packet. So we never lose bits or bytes, we lose entire packets. As long as we keep our packets small enough then a 20-30 bytes difference would not matter, since the remote packet queues will discard some of them anyway. But the packets that are not discarded will have some FEC information for the discarded ones, so part of the voice signal can be reconstructed by the decoder.

Codec Control (Congestion control)

There are two kinds of callbacks into *mod_opus*, from switch_rtp.c :

1. For passing packet loss to the codec

e.g.:

*switch_core_media_codec_control(rtp_session->session, SWITCH_MEDIA_TYPE_AUDIO, SWITCH_IO_WRITE, SCC_AUDIO_PACKET_LOSS, SCCT_INT, (void *)&rtp_session->rtcp_frame.reports[i].loss_avg, SCCT_NONE, NULL, NULL, NULL);*

This will actually call *switch_opus_control()*, with param SCC_AUDIO_PACKET_LOSS, so that it will adjust the packet loss percentage on the Opus encoder.

This is where *switch_opus_keep_fec_enabled()* is called (if enabled in the config file) and it will adjust bitrate to keep FEC enabled. Passing the actual packet loss to the Opus encoder, without adjusting the bitrate might make the encoder not to send FEC at all.

2. For passing commands to increase/decrease bitrate (the “increase” and “decrease” commands sent to the codec module) . We can also go to “default” or to “minimum” bitrate, depending on the network conditions.

e.g.:
*switch_core_media_codec_control(rtp_session->session, SWITCH_MEDIA_TYPE_AUDIO, SWITCH_IO_WRITE, SCC_AUDIO_ADJUST_BITRATE,
SCCT_STRING, "decrease", SCCT_NONE, NULL, NULL, NULL);*

This will call switch_opus_control(), with param SCC_AUDIO_ADJUST_BITRATE.

Debugging

Introduction

It is possible to log very detailed information on opus frames being decoded and encoded. Through *fsctl*, you can enable a debug mode with: “opus_debug on” (and disable it with “opus_debug off”).

For each frame FreesWITCH will log the size of data, whether FEC is present or not and whether there was PLC.

Warning: there will be one log line per frame, so with 20 msec packetization this means 50 lines of logs per second per direction. Do not enable it on a Production server or it may fill the log files very quickly.

How to work with the “opus_debug” CLI command

Example

```
2015-11-10 16:11:48.507949 [DEBUG] mod_opus.c:453 encode: opus_frames [1] samples [160] audio bandwidth [NARROWBAND] bytes [35] FEC[no] channels[1]
2015-11-10 16:11:48.507949 [DEBUG] mod_opus.c:453 decode: opus_frames [1] samples [160] audio bandwidth [NARROWBAND] bytes [42] FEC[yes] channels[1]
2015-11-10 16:11:48.507949 [DEBUG] mod_opus.c:453 encode: opus_frames [1] samples [160] audio bandwidth [NARROWBAND] bytes [36] FEC[no] channels[1]
2015-11-10 16:11:48.547947 [DEBUG] mod_opus.c:453 decode: opus_frames [1] samples [160] audio bandwidth [NARROWBAND] bytes [40] FEC[yes] channels[1]
```

You can see above 4 frames, 2 of them were just encoded (“encode:”) , and 2 of them will be decoded immediately (“decode:”). All 4 frames are Narrowband and they all have an actual 160 samples frame size, corresponding to 20 ms ptime. Since the opus codec is VBR they differ in size, they are around 30-40 bytes. Two of them have FEC, and they are all mono (1 channel).

Example with PLC

```
2015-11-13 07:19:19.525757 [DEBUG] mod_opus.c:453 decode: opus_frames [1] samples [320] audio bandwidth [NARROWBAND] bytes [80] FEC[yes] channels[1]
2015-11-13 07:19:19.585817 [DEBUG] mod_opus.c:764 Missing SEQ 397 Checking JB
2015-11-13 07:19:19.585817 [DEBUG] mod_opus.c:798 MISSING FRAME: PLC
2015-11-13 07:19:19.605788 [DEBUG] mod_opus.c:764 Missing SEQ 398 Checking JB
2015-11-13 07:19:19.605788 [DEBUG] mod_opus.c:798 MISSING FRAME: PLC
2015-11-13 07:19:19.645791 [DEBUG] mod_opus.c:764 Missing SEQ 399 Checking JB
```

You can see above one frame of 320 samples (corresponding to ptime 40) being decoded, then the next frames are missing, and PLC was done instead (there was no N+1 packet in the Jitter Buffer either).

Example with FEC

```
2015-11-13 07:19:23.005774 [DEBUG] mod_opus.c:453 decode: opus_frames [1] samples [320]
audio bandwidth [NARROWBAND] bytes [87] FEC[yes] channels[1]
2015-11-13 07:19:23.085842 [DEBUG] mod_opus.c:764 Missing SEQ 482 Checking JB
2015-11-13 07:19:23.085842 [DEBUG] mod_opus.c:769 Lookahead frame found: 1364160:483
2015-11-13 07:19:23.085842 [DEBUG] mod_opus.c:781 FEC info available in packet with SEQ:
483 LEN: 91
2015-11-13 07:19:23.085842 [DEBUG] mod_opus.c:798 MISSING FRAME: Look-ahead FEC
2015-11-13 07:19:23.085842 [DEBUG] mod_opus.c:453 FEC correction: opus_frames [1]
samples [320] audio bandwidth [NARROWBAND] bytes [91] FEC[yes] channels[1]
```

You can see above 2 frames, the first one is a normal frame to be decoded, the second one, where it says "FEC correction", is a frame that we just retrieved from the jitter buffer, because the RTP packet with SEQ 482 was missing due to packet loss.

An important aspect of codec negotiation is the FMTP we send or receive in the offer/answer.

How SDP in the 200 OK should look in order to reflect the settings in our codec_string and our opus.conf.xml file:

```
2015-11-12 16:53:02.487930 [DEBUG] mod_sofia.c:2312 Ring SDP:
v=0
o=FreeSWITCH 1447318678 1447318679 IN IP4 10.0.0.145
s=FreeSWITCH
c=IN IP4 10.0.0.145
t=0 0
m=audio 28504 RTP/AVP 96 101
a=rtpmap:96 opus/48000/2
a=fmtp:96 useinbandfec=1; maxaveragebitrate=14400; maxplaybackrate=8000
a=rtpmap:101 telephone-event/48000
a=fmtp:101 0-16
a=ptime:20
a=sendrecv
a=rtcp:28505 IN IP4 10.0.0.145
```

Note the *rtpmap* line with the Opus RTP profile opus/48000/2, then the FMTP values, and then the telephone-event clock rate which should be 48000 for Opus, even if inside our opus frames we have a signal sampled at different rate.

Statistics of Decoder

This is logged at the end of a call, and statistics about the total number of frames decoded, PLC performed and FEC frames decoded are reported:

```
0bd9616f-9867-4a75-8f15-5b703bbb4dda 2015-11-12 17:58:43.534824 [DEBUG]
mod_opus.c:661 Opus decoder stats: Frames[7038] PLC[2247] FEC[29]
```

Statistics of Encoder

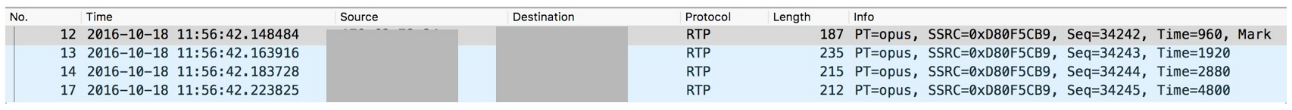
This is logged at the end of a call, with statistics about the number of frames that were encoded, and in particular the average bitrate of the encoded data.

2016-10-21 07:26:54.318880 [DEBUG] mod_opus.c:709 Opus encoder stats: Frames[1219] Bytes encoded[35490] Encoded length ms[24380] Average encoded bitrate bps[11830] FEC frames (only for debug mode) [755]

The number of encoded frames with FEC is also logged, but note that this is only taken into account when *opus_debug* is on. The reason behind this is to avoid unnecessary debug/log logic for each encoded frame (i.e. from the typical 50 times/sec for 20 msec ptime, to potentially higher frequencies for smaller ptimes).

Debugging with Wireshark

It's possible to see the details of an opus session with tools like Wireshark, after the packets have been captured with any of the available tools like tcpdump and the like.
The following Wireshark screenshot shows a few opus packets from a capture:

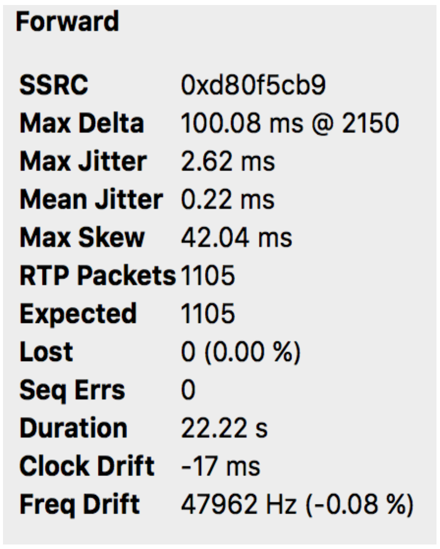


No.	Time	Source	Destination	Protocol	Length	Info
12	2016-10-18 11:56:42.148484	[redacted]	[redacted]	RTP	187	PT=opus, SSRC=0xD80F5CB9, Seq=34242, Time=960, Mark
13	2016-10-18 11:56:42.163916	[redacted]	[redacted]	RTP	235	PT=opus, SSRC=0xD80F5CB9, Seq=34243, Time=1920
14	2016-10-18 11:56:42.183728	[redacted]	[redacted]	RTP	215	PT=opus, SSRC=0xD80F5CB9, Seq=34244, Time=2880
17	2016-10-18 11:56:42.223825	[redacted]	[redacted]	RTP	212	PT=opus, SSRC=0xD80F5CB9, Seq=34245, Time=4800

You can see that Wireshark is interpreting those frames as Opus: this can be inferred by the SIP signaling preceding the session. To be noted that Opus doesn't have a dedicated IANA number, so the codec number is dynamic (e.g. could be 96, or 101, or something else).

The important part of this capture relates to the timing: every frame has an increment of 960 "time ticks", i.e. they refer to 20 msec at 48000 samples/sec. This is confirmed by the RTP sequence numbers, which are correctly incrementing without packets lost or out of order.
The length of the frames in this example show that it's surely not a narrowband session, where the length is expected to be about 30-40 Bytes, but refer to a larger bandwidth and a bitrate of about 45 Kbps.

Wireshark can also analyze the RTP stream and show some metrics related to them, as in the following screenshot:



Forward

SSRC	0xd80f5cb9
Max Delta	100.08 ms @ 2150
Max Jitter	2.62 ms
Mean Jitter	0.22 ms
Max Skew	42.04 ms
RTP Packets	1105
Expected	1105
Lost	0 (0.00 %)
Seq Errs	0
Duration	22.22 s
Clock Drift	-17 ms
Freq Drift	47962 Hz (-0.08 %)

APPENDIX – libopus

Installing libopus

Debian distributions:
apt-get install libopus

CentOS distributions:
yum install libopus

Building libopus

When libopus is not already available for your distribution, you can build it from source.
libopus is built in a standard way; you can either use the available tarball or checkout from git.

Prepare source from tarball

```
wget http://downloads.xiph.org/releases/opus/opus-1.1.3.tar.gz
tar xvf opus-1.1.3.tar.gz
cd opus-1.1.3
```

Prepare source from git

```
git clone https://git.xiph.org/opus.git
cd opus
git checkout v1.1.3
./autogen.sh
```

Build

```
./configure
make
make check
make install
```

Packaging libopus

Building libopus RPMs

```
yum install -y rpmdevtools libogg-devel gcc make wget
rpmdev-setuptree
cd ~/rpmbuild/SOURCES
wget http://downloads.xiph.org/releases/opus/opus-1.1.3.tar.gz
```

Copy an opus.spec file like the one in

https://freeswitch.org/confluence/display/FREESWITCH/Build+opus+1.1.1+RPMs+for+CentOS+7 into ~/rpmbuild/SPECS/opus.spec (check versions carefully).

```
cd  ~/rpmbuild/SPECS
rpmbuild -v -bb opus.spec
```

The packages will be in ~/rpmbuild/RPMS/x86_64/, and this procedure generates both libopus and libopus-devel packages.

APPENDIX – Building mod_opus

First install libopus-dev (or libopus-devel).

Ensure codecs/mod_opus is not commented in modules.cfg, e.g.:

```
...
#codecs/mod_mp4v
codecs/mod_opus
#codecs/mod_sangoma_codec
...
```

then build FreeSWITCH as usual.

Glossary

Audio Codec (audio enCOder/DECoder): Software that compresses and decompresses a digital audio signal. Codecs are generally understood to be various mathematical models used to digitally encode (and compress) audio information. Most sound codecs process audio signals that are already digitalized in time and amplitude. Many of these models take into account the human brain's ability to form an impression from incomplete information. The purpose of the various encoding algorithms is to strike a balance between efficiency and quality.

Audio Bandwidth: The range of audio frequencies which directly influence the fidelity of a sound. The higher the audio bandwidth, the better the sound fidelity. The highest practical frequency which the human ear can normally hear is 20 kHz.

Bitrate: The rate at which bits are transferred from one location to another. In other words, it measures how much data is transmitted in a given amount of time. Bitrate is commonly measured in bits per second (bps), kilobits per second (Kbps).
When encoding a speech signal, the bitrate is defined as the number of bits per unit of time required to encode the speech. It is measured in bits per second (bps), or generally kilobits per second. It is important to make the distinction between kilobits per second (kbps) and kilobytes per second (kBps).

FEC (Forward Error Correction): a method to address packet loss by including some information from previously transmitted packets in subsequent packets. By performing mathematical operations in a particular FEC scheme, it is possible to reconstruct a lost packet from information bits in neighboring packets.

Frame Size:The amount of data to be encoded or decoded at a given time by the encoding or decoding functions. It is measured in samples or time. e.g. 160 samples or 20 milliseconds.

Jitter: The measure of the variability over time of the latency across a network.
Real time communications (for example VoIP) usually have quality problems due to this effect. In general, it is a problem in slow-speed links or with congestion. It is hoped that the increase of QoS (quality of the service) mechanisms like priority buffers, bandwidth reservation or high-speed connections can reduce jitter problem. The best solution is to use jitter buffers.

Jitter Buffer: The jitter buffer, which is located at the receiving end of the voice connection, intentionally delays the arriving packets so that the end user experiences a clear connection with very little sound distortion. There are two kinds of jitter buffers, static and dynamic.

L16: L16 denotes uncompressed audio data samples, using 16-bit signed representation with 65535 equally divided steps between minimum and maximum signal level, ranging from -32768 to 32767.

Narrowband (Audio Bandwidth): in telephony, narrowband is usually considered to cover frequencies 300–3400 Hz. This spectrum allows for speech to traverse and nothing else.
This is the quality of voice achievable in landline PSTN phones (I.e. regular telephony) and in most of the cellular networks. The voice signal is sampled at 8,000 Hz.

Network bitrate: Codec Bitrate + network header overhead per packet (IP+UDP+RTP = 40 bytes) . Note that the bitrate values are only for one direction of the call.

PCM (Pulse code modulation): Modulation in which the peak-to peak amplitude range of the signal to be transmitted is divided into a number of standard values, each having its own code; each sample of the signal is then transmitted as the code for the nearest standard amplitude.

ptime (packetization time): The time between packets sent. It has usually a fixed length per RTP session. Same value as Frame Size , only that the term "ptime" is used in a RTP scenario. Frame Size has to do with encoding or decoding itself. You can decode or encode audio without sending it (e.g. you will not be putting it into network packets).
To improve network bandwidth utilization the obvious way to go is to send more frames in one RTP packet. However, as we do this, we also increase latency. If we decide to send, 100 milliseconds of audio in one packet, this means that we have added a latency of the same 100 milliseconds. Simply put, the first sample of the first frame arrives together with the last sample of the last frame in the packet, so the total delay is equal to the length of audio carried in the packet.

Sample rate: The number of samples of audio carried per second, measured in Hz or kHz (one kHz being 1000 Hz).

Sampling: In signal processing, sampling is the reduction of a continuous signal to a discrete signal. A sample is a value or set of values at a point in time and/or space.

VBR: Variable Bit-Rate allows a codec to change its bitrate dynamically to adapt to the "difficulty" of the audio being encoded. VBR can achieve lower bitrate for the same quality, or a better quality for a certain bitrate. Despite its advantages, VBR has two main drawbacks: first, by only specifying quality, there's no guarantee about the final average bitrate. Second, for some real-time applications like voice over IP (VoIP), what counts is the maximum bit-rate, which must be low enough for the communication channel.

References

https://en.wikipedia.org/wiki/Opus_(audio_format)
https://tools.ietf.org/html/rfc6716
https://freeswitch.org/confluence/display/FREESWITCH/mod_opus
https://tools.ietf.org/html/rfc7587
https://tools.ietf.org/html/rfc7584
https://tools.ietf.org/html/rfc4566
https://opus-codec.org

Recommended reads

“FreeSWITCH Cookbook”, https://www.packtpub.com/networking-and-servers/freeswitch-cookbook
“Mastering FreeSWITCH”, https://www.packtpub.com/networking-and-servers/mastering-freeswitch