DYSONSPHERE
LIMITED

TECHNICAL PAPER · VOIP MEDIA ENGINEERING

Keeping Opus In-Band FEC Alive

The keep-fec-enabled technique in FreeSWITCH mod_opus: how it works, what a decade of libopus changes did to it, and a version-aware reference implementation verified against real encoders.

Dragos Oancea
Original author of FreeSWITCH FS-7772 (2015)

Edition 1.0
October 2026

Keeping Opus In-Band FEC Alive
Technical Paper, Edition 1.0, October 2026
Author: Dragos Oancea
Published by DYSONSPHERE LIMITED

© 2026 DYSONSPHERE LIMITED. All rights reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and other non-commercial uses permitted by copyright law.

Reference implementation

The reference implementation in Chapter 10 and Appendix B (`keepfec.h`, `keepfec.c`) is © 2026 DYSONSPHERE LIMITED. Purchasers of this paper may use, modify and incorporate it in their own software, including commercial software, without attribution. Redistributing it as a standalone code listing or as part of a competing publication requires the publisher's permission.

Third-party material

Excerpts from FreeSWITCH (`src/mod/codecs/mod_opus/mod_opus.c`) are reproduced under the Mozilla Public License 1.1; the original source is available at github.com/signalwire/freeswitch. FreeSWITCH is © 2005–2014 Anthony Minessale II and contributors.

Excerpts from libopus are © 2001–2023 Xiph.Org, Skype Limited, Octasic, Jean-Marc Valin, Timothy B. Terriberry, CSIRO, Gregory Maxwell, Mark Borgerding, Erik de Castro Lopo, Mozilla and Amazon, and are reproduced under the BSD 3-Clause licence of the Opus codec (opus-codec.org).

Trademarks

FreeSWITCH, Opus and other product names used in this paper are trademarks of their respective owners. They are used here only to identify the software discussed. This paper is independent and is not endorsed by, sponsored by or affiliated with SignalWire, the Xiph.Org Foundation or the Opus project.

Disclaimer

This paper and its code are provided “as is”, without warranty of any kind. Measurements were made in the environment described in Chapter 8 and may differ in other environments, library builds and network conditions. Test any change in a staging system before deploying it to production.

Contents

	Executive summary	4
1	Introduction	5
2	Opus in brief	6
3	In-band FEC: how LBRR works	8
4	The FEC decision in libopus 1.1	9
5	How FreeSWITCH reacts to packet loss	11
6	The 2015 implementation, annotated	12
7	Evolution in FreeSWITCH	15
8	Does the model still hold? libopus 1.2 and later	16
9	What FEC buys: speech-quality measurements	18
10	A version-aware reference implementation	21
11	Operator guide	24
12	Conclusions	26
A	Original 2015 function (MPL 1.1)	27
B	Reference implementation source	28
C	Measurement data	31
D	References	33

© 2026 DYSONSPHERE LIMITED

OVERVIEW

Executive summary

Opus only sends in-band forward error correction when its internal rate is above a loss-dependent threshold. In 2015, FreeSWITCH gained a feature, `keep-fec-enabled`, that reproduces that hidden rule and nudges the bitrate just high enough to keep FEC on. This paper explains the technique, measures it against three generations of libopus, quantifies what it does for speech quality, and provides a corrected implementation for today's libraries.

1 · The 2015 model was exact for its library.

Measured against libopus 1.1, the bitrate predicted by the FreeSWITCH algorithm matched the real FEC switch-on point at all 20 measured operating points (10 loss levels, wideband and narrowband), to the 400 bit/s resolution of the test.

2 · FEC, not bitrate, delivers the gain.

With libopus 1.1 at 5–20 % random loss, keep-FEC lifted wideband PESQ scores by 0.80 to 1.28 MOS. Raising the bitrate by the same amount with FEC off gained at most 0.03 MOS.

3 · libopus 1.2 changed the rules.

Since June 2016 the FEC decision has lived in `opus_encoder.c`, uses a loss-penalised “equivalent rate” with hysteresis, and lowers the audio bandwidth to make room for FEC when loss exceeds 5 %. Against libopus 1.3.1 and 1.6.1, the 2015 constants are 1.6–4.0 kbit/s too low: at ≤ 5 % loss FEC never turns on, above 5 % it turns on only by narrowing wideband speech, and at 8 kHz it never turns on at all.

4 · Today's defaults can start calls without FEC.

FreeSWITCH's built-in defaults (keep-FEC on, 20 % initial loss, CBR unless `use-vbr` is set) produce no FEC at call start for narrowband calls on current libopus, and none at all in CBR on any version tested.

5 · A version-aware fix, verified.

The reference implementation in this paper models both libopus generations, VBR and CBR, every frame size and complexity. It achieved FEC at full bandwidth in 706 of 708 tested configurations; both exceptions are bandwidth choices libopus 1.1 makes for unrelated reasons.

6 · Know the limits.

LBRR protects isolated losses. With 10 % loss arriving in bursts averaging four packets, the FEC benefit fell from +1.37 to +0.24 MOS.

© 2026 DYSONSPHERE LIMITED

CHAPTER 1

Introduction

Every packet-switched voice call loses some packets. Wi-Fi contention, mobile handovers, congested access links and overloaded servers all drop or delay audio, and a packet that arrives after its playout time is as good as lost. Receivers have two broad tools to cope. Packet loss concealment (PLC) synthesises a plausible replacement from the audio around the gap. Forward error correction (FEC) sends redundant information in advance so the receiver can rebuild what was lost.

Opus, the IETF codec standardised in RFC 6716 and the default for WebRTC, offers both. Its in-band FEC is cheap and effective for speech: each packet can carry a low-bitrate copy of the previous frame, so a single lost packet can be rebuilt from the one that follows it. There is a catch that is easy to miss. Turning FEC on through the API is a request, not a command. The reference encoder decides frame by frame whether it can afford the redundancy, and at the moderate bitrates typical of telephony it often decides that it cannot. A system can believe it is protected while it is sending no FEC at all.

In 2015 the author added a feature to FreeSWITCH, the open-source telephony platform, that closes this gap. Called `keep-fec-enabled` and tracked as FS-7772, it watches the packet-loss rate reported by the far end, recomputes the encoder's private decision rule in the application, and raises the bitrate just enough that the encoder keeps adding FEC.

This paper is the full account of that technique. Chapters 2 and 3 cover the Opus background needed to follow the rest. Chapter 4 derives the decision rule in libopus 1.1, the version the feature was written against. Chapters 5 to 7 cover the FreeSWITCH side: the plumbing, an annotated walk through the original code, and how it evolved. Chapter 8 tests whether the model still holds in today's libopus, and finds that it does not. Chapter 9 measures what keep-FEC is worth in speech quality. Chapter 10 presents a version-aware reference implementation with its verification, and Chapter 11 turns the findings into configuration advice.

Who this paper is for

It is written for VoIP and WebRTC engineers who configure or develop media servers, for developers integrating libopus into their own products, and for operators who want to know whether their calls are actually protected. It assumes familiarity with RTP and RTCP and basic C. No prior knowledge of Opus internals is needed.

How the measurements were made

All numbers in this paper come from real encoders, not from the models being tested. libopus 1.1, 1.3.1 and 1.6.1 were compiled from the official release tags, a wideband speech recording was encoded under each configuration, and every output packet was inspected for LBRR data with `opus_packet_has_lbrr()` and for its coded bandwidth. Speech quality was scored with PESQ (ITU-T P. 862.2, wideband). Chapter 8 and Chapter 9 give the details.

© 2026 DYSONSPHERE LIMITED

CHAPTER 2

Opus in brief

Opus combines two codec technologies. **SILK**, originally developed by Skype, is a linear-prediction speech coder that is efficient at low bitrates. **CELT**, from Xiph.Org, is a transform coder suited to music and high bitrates. An Opus encoder runs in one of three modes: SILK-only, CELT-only, or hybrid, where SILK codes the band up to 8 kHz and CELT adds the band above it.

Bandwidths and modes

Opus codes one of five audio bandwidths, chosen by the encoder within the limits the application sets. Only SILK can produce in-band FEC, so the bandwidths SILK covers are the ones that matter here.

Bandwidth	Audio band	Internal rate	Codec layer	SILK LBRR base threshold
Narrowband (NB)	4 kHz	8 kHz	SILK	12 000 bit/s
Medium-band (MB)	6 kHz	12 kHz	SILK	14 000 bit/s
Wideband (WB)	8 kHz	16 kHz	SILK (or CELT)	16 000 bit/s
Super-wideband (SWB)	12 kHz	24 kHz	Hybrid or CELT	20 000 bit/s (≥1.2 only)
Fullband (FB)	20 kHz	48 kHz	Hybrid or CELT	22 000 bit/s (≥1.2 only)

At the bitrates used in telephony with FEC, roughly 12 to 32 kbit/s for mono speech, libopus normally runs SILK-only at narrowband or wideband. That is the regime this paper analyses. The API *sample rate*, the rate at which the application feeds PCM to the encoder, sets the highest bandwidth the encoder may use. It need not equal the internal SILK rate: an encoder fed at 48 kHz can still code wideband or narrowband speech.

Packets and the TOC byte

Every Opus packet begins with a one-byte table-of-contents (TOC) header that signals the mode, bandwidth, frame duration and frame count. Frames last 2.5, 5, 10, 20, 40 or 60 ms; SILK supports 10 ms and longer. Telephony nearly always uses 20 ms, giving 50 packets per second. The TOC byte matters later: it is overhead that the encoder removes before it hands its rate budget to SILK, and that detail is part of the decision rule.

Bitrate control

Applications set a target with `OPUS_SET_BITRATE` and choose variable (VBR, the default) or constant (CBR) bitrate with `OPUS_SET_VBR`. Two further controls drive FEC: `OPUS_SET_INBAND_FEC` asks the encoder to add FEC, and `OPUS_SET_PACKET_LOSS_PERC` tells it how much loss to expect. Both are needed: with an expected loss of 0 %, libopus never codes FEC.

Signalling in SDP

RFC 7587 defines the RTP payload format for Opus and a set of `fmtp` parameters through which each side states its preferences. Several of them bear directly on keep-FEC.

© 2026 DYSONSPHERE LIMITED

fmtp parameter (RFC 7587)	Meaning	Relevance to keep-FEC
`useinbandfec=1`	The receiver can decode in-band FEC	Without it, there is no point coding LBRR at all
`maxaveragebitrate`	Highest average bitrate the receiver wants	Hard ceiling for any FEC-driven increase
`cbr=1`	Receiver prefers constant bitrate	CBR raises the bitrate FEC needs
`maxplaybackrate`	Highest sample rate the receiver will play	Determines NB / WB and therefore the threshold
`usedtx`	Discontinuous transmission	Silence frames carry no LBRR
`ptime`, `maxptime`	Packet duration	Changes frame rate and the rate arithmetic

© 2026 DYSONSPHERE LIMITED

CHAPTER 3

In-band FEC: how LBRR works

Opus's in-band FEC is SILK's **Low Bit-Rate Redundancy** (LBRR). When it is active, each SILK packet carries, in addition to the current frame, a second encoding of the *previous* frame. The redundant copy is coded with the same SILK machinery but with coarser quantisation, controlled by a gain increase that grows as the expected loss falls, so it costs a fraction of a normal frame.

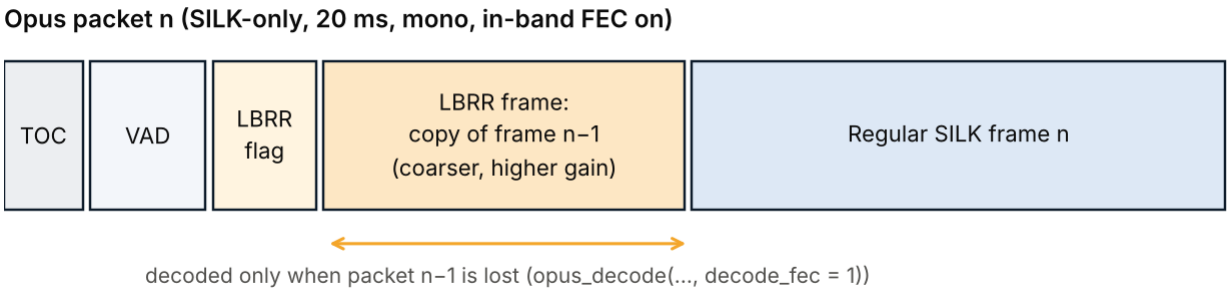


Figure 1. Layout of a SILK-only Opus packet with in-band FEC. The LBRR flag announces redundancy; the LBRR frame is a coarser copy of the previous frame, followed by the regular frame. Sizes are illustrative.

Using FEC at the receiver

FEC only helps if the decoder uses it. When packet n is missing but packet n+1 has arrived, the application calls `opus_decode()` on packet n+1 with `decode_fec = 1` and a frame size equal to the missing duration. The decoder extracts the LBRR copy of frame n and returns it in place of the lost audio. It then decodes packet n+1 normally. If packet n+1 carries no LBRR data, the call falls back to PLC.

Two practical consequences follow. First, the receiver must hold at least one packet in its jitter buffer so that packet n+1 is available when frame n is due; FreeSWITCH's `use-jb-lookahead` option exists for exactly this. Second, LBRR repairs a loss only if the following packet survives. Two consecutive losses leave the first one unrecoverable, which is why the protection weakens sharply under bursty loss (Chapter 9).

The cost

LBRR is not free. Within a fixed bitrate, the bits spent on redundancy come out of the primary frame, so quality on a clean link drops slightly. That is why libopus enables it only when it is told to expect loss and only when the rate is high enough that the primary frame can spare the bits. The rule that makes that judgement is the subject of the next chapter.

What FEC is not

LBRR is distinct from three related techniques. PLC conceals rather than repairs. Packet duplication or RED (RFC 2198) sends whole extra copies at the RTP layer. Opus 1.5 introduced Deep REDundancy (DRED), a neural-network technique that can cover much longer gaps; it is a separate, optional mechanism and is outside the scope of this paper.

© 2026 DYSONSPHERE LIMITED

CHAPTER 4

The FEC decision in libopus 1.1

In libopus 1.1, the release current when FS-7772 was written, the decision lives in `silk_setup_LBRR()` in `silk/control_codec.c`. It is short enough to quote in full.

```
if( psEncC->useInBandFEC && psEncC->PacketLoss_perc > 0 ) {
    if( psEncC->fs_kHz == 8 )       LBRR_rate_thres_bps = LBRR_NB_MIN_RATE_BPS;   /* 12000 */
    else if( psEncC->fs_kHz == 12 ) LBRR_rate_thres_bps = LBRR_MB_MIN_RATE_BPS;   /* 14000 */
    else                            LBRR_rate_thres_bps = LBRR_WB_MIN_RATE_BPS;   /* 16000 */
    LBRR_rate_thres_bps = silk_SMULWB( silk_MUL( LBRR_rate_thres_bps,
            125 - silk_min( psEncC->PacketLoss_perc, 25 ) ), SILK_FIX_CONST( 0.01, 16 ) );
    if( TargetRate_bps > LBRR_rate_thres_bps ) {
        psEncC->LBRR_enabled = 1;
        ...
```

libopus 1.1, silk/control_codec.c (excerpt; constants from silk/tuning_parameters.h). BSD licence.

The threshold

Three conditions must hold. FEC must be requested, the expected loss must be above zero, and SILK's target rate must exceed a threshold T that depends on SILK's internal bandwidth and the expected loss p:

$$T = \mathrm{SMULWB}(\ \mathrm{base} \times (125 - \min(p, 25)),\ 655\ ) \approx \mathrm{base} \times (1.25 - \min(p, 25) / 100)$$

The base is 12, 14 or 16 kbit/s for NB, MB and WB. `SMULWB` is SILK's fixed-point multiply, and 655 is 0.01 in Q16 format, which rounds slightly low. The threshold therefore runs from 1.24 × base at 1 % loss down to 1.00 × base at 25 % loss and above. Counter-intuitively, it *falls* as loss rises: the more loss there is, the more readily SILK pays for redundancy.

The rate SILK actually sees

The threshold is compared against `TargetRate_bps`, not against the bitrate the application set. Between the two sits `opus_encode_native()`, which converts the requested bitrate into a byte budget per frame, removes one byte for the TOC header, and converts back:

$$\mathrm{bytes} = \mathrm{floor}(B \times N / (Fs \times 8)) - 1 \quad R = 8 \times \mathrm{bytes} \times \mathrm{frame_rate}$$

where B is the requested bitrate, N the frame size in samples, Fs the API sample rate, and `frame_rate = Fs / N`, computed in integer arithmetic (16, not 16.67, for 60 ms frames). At 20 ms this simplifies to $R = 400 \times (\mathrm{floor}(B/400) - 1)$: the rate SILK sees is always at least 400 bit/s below the request, and moves in 400 bit/s steps.

In CBR mode two more adjustments apply. The byte budget is rounded to the nearest byte rather than floored, and SILK's initial target is lowered by a further 2 000 bit/s “to make it easier to reach the CBR rate”, in the words of the source comment. CBR therefore needs about 2 kbit/s more for the same FEC decision, an effect the 2015 feature did not model.

© 2026 DYSONSPHERE LIMITED

Worked example

Take a wideband call at 20 ms, VBR, with a configured bitrate of 16 000 bit/s, and suppose the far end reports 10 % loss.

Step	Computation	Result
Threshold	16 000 × (125 − 10) × 655 / 65 536	18 389 bit/s
SILK rate at 16 000	400 × (40 − 1)	15 600 bit/s · FEC off
Smallest passing request	400 × (47 − 1) = 18 400 > 18 389	B = 18 800 bit/s
Cost	18 800 − 16 000	+2.8 kbit/s

Without intervention, this call sends no FEC at all despite the encoder being told to expect 10 % loss. With 2.8 kbit/s more, every speech packet carries a repair copy of its predecessor.

© 2026 DYSONSPHERE LIMITED

CHAPTER 5

How FreeSWITCH reacts to packet loss

FreeSWITCH learns about loss from the far end's RTCP receiver reports, which carry a “fraction lost” field for the stream FreeSWITCH is sending. In `switch_rtp.c`, the RTP layer keeps a running average of that field (`loss_avg`) per report. When the average changes, it calls `switch_core_media_codec_control()` with the command `SCC_AUDIO_PACKET_LOSS` on the write (encoding) side of the audio codec.

The command reaches `switch_opus_control()` in `mod_opus.c`. There the module passes the new percentage to the encoder with `OPUS_SET_PACKET_LOSS_PERC` and, when `keep-fec-enabled` is set, runs the keep-FEC logic. When the separate `adjust-bitrate` option is on, the RTP layer also runs Kalman-filter estimators of loss and round-trip time and may send `SCC_AUDIO_ADJUST_BITRATE` commands to raise or lower the bitrate on a slow or congested link.

Two properties of this path shape the design. RTCP reports arrive every few seconds, so the loss value is coarse and lags the network. And the value describes loss in the *outgoing* direction, measured by the receiver. That is exactly the stream whose encoder needs to add FEC, so the feedback loop is the right one.

Configuration

The relevant parameters live in `autoload_configs/opus.conf.xml`: `keep-fec-enabled`, `packet-loss-percent` (the loss assumed at call start), `use-vbr`, `complexity`, `maxaveragebitrate`, `advertise-useinbandfec`, `use-jb-lookahead` and `adjust-bitrate`. Chapter 11 recommends values.

© 2026 DYSONSPHERE LIMITED

CHAPTER 6

The 2015 implementation, annotated

FS-7772 (commit `b3f453b0`, merged through pull request #327 on 3 July 2015) added 57 lines to `mod_opus.c`: a configuration flag, a hook in the codec-control handler, and the function `switch_opus_keep_fec_enabled()`. Figure 2 shows its control flow; the full listing is in Appendix A.

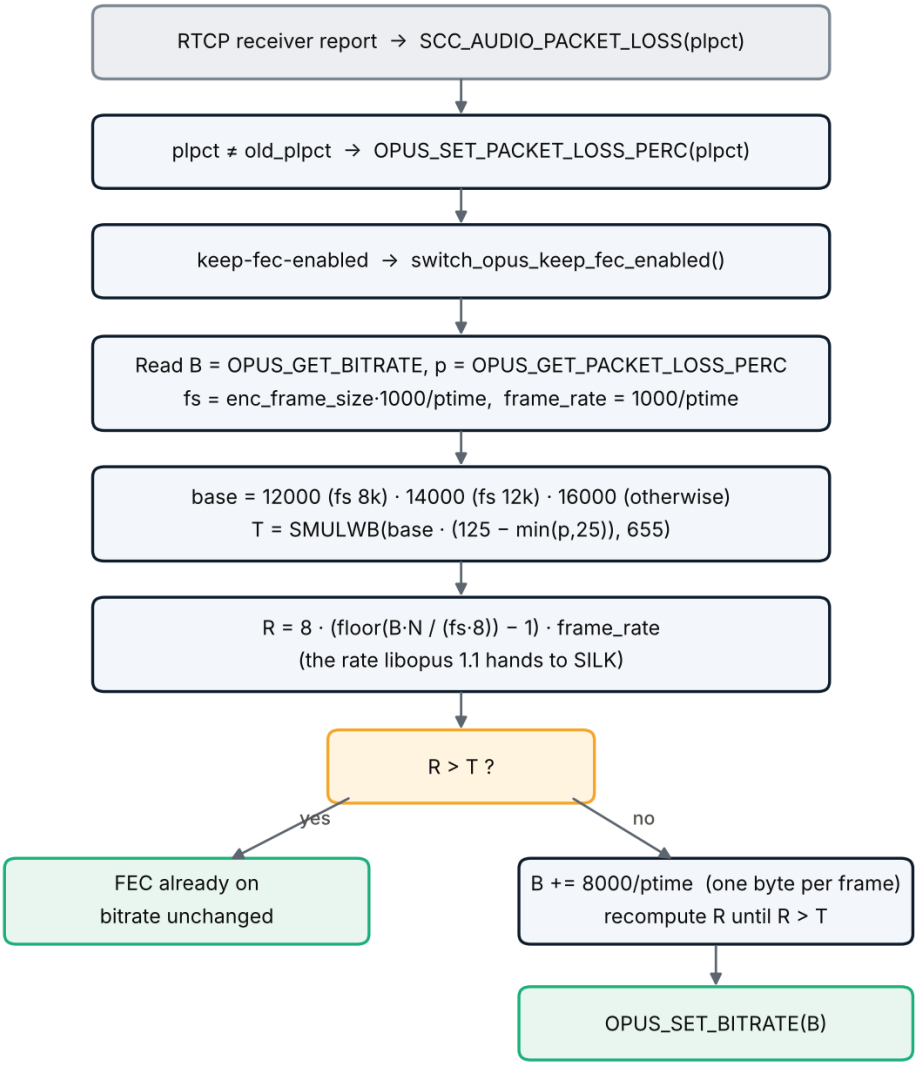


Figure 2. Control flow of the 2015 implementation, from an RTCP loss report to the bitrate decision.

The key design choice

libopus offers no API to ask whether FEC is active, or what bitrate it would need. The 2015 code therefore re-derives the encoder's private rule in the application: it reproduces SILK's threshold bit-for-bit, including the fixed-point multiply, and reproduces `opus_encode_native()`'s byte-and-TOC

© 2026 DYSONSPHERE LIMITED

arithmetic. It then steers the one public control that moves the outcome, the bitrate. The result is a precise, minimal intervention: the bitrate rises only when FEC would otherwise be off, and only by as much as needed.

Walking through the code

Inputs

The function reads the current bitrate and loss back from the encoder with `OPUS_GET_BITRATE` and `OPUS_GET_PACKET_LOSS_PERC`, so it always works from the encoder's real state. It derives the API sample rate from the encoder frame size and packet time, the frame rate as `1000 / ptime`, and a step of `8000 / ptime`: one byte per frame, expressed in bit/s.

```
uint32_t fs = context->enc_frame_size * 1000 / (codec->implementation->microseconds_per_packet /
1000);
float frame_rate = 1000 / (codec->implementation->microseconds_per_packet / 1000);
uint32_t step = 8000 / (codec->implementation->microseconds_per_packet / 1000);
```

The integer division in `frame_rate` looks like a precision bug for 60 ms packets, but it is not: libopus 1.1 itself computes `frame_rate = Fs / frame_size` in integers, so the module matches the library's own arithmetic.

The threshold

```
a32 = LBRR_rate_thres_bps * (125 - (((current_loss) < (25)) ? (current_loss) : (25)));
b32 = ((opus_int32)((0.01) * ((opus_int64)1 << (16)) + 0.5));
LBRR_threshold_bitrate = (a32 >> 16) * (opus_int32)((opus_int16)b32)
                       + (((a32 & 0x0000FFFF) * (opus_int32)((opus_int16)b32)) >> 16);
```

This is SILK's `silk_SMULWB` macro expanded by hand, with `SILK_FIX_CONST(0.01, 16)` evaluated to 655. Because it is the same arithmetic, the module's threshold equals libopus's to the last bit: 14 871 bit/s for narrowband at 1 % loss, not the 14 880 a floating-point calculation would give.

The effective rate and the search

The module computes the rate SILK will see, compares it with the threshold, and if FEC would be off, raises the bitrate one byte per frame until it would be on. Because each step adds exactly one byte, the loop stops at the smallest byte-aligned bitrate that keeps FEC.

Validation against libopus 1.1

We encoded speech with libopus 1.1 at every bitrate from 10 to 30 kbit/s in 400 bit/s steps, at 10 loss levels, at both wideband and narrowband, and recorded the lowest bitrate from which every speech packet carried LBRR data. At all 20 operating points the measured switch-on bitrate equalled the bitrate the 2015 algorithm chooses (Figure 3 and Appendix C). The 8 kHz start-of-call table that FS-8161 later hard-coded is the same set of numbers.

> **FINDING**
>
> For the library it was written against, the 2015 model is not an approximation. It reproduces libopus 1.1's FEC decision exactly in VBR mode.

© 2026 DYSONSPHERE LIMITED

Limitations of the original

- **One-way adjustment.** The function only raises the bitrate. Because the threshold rises as loss falls, falling loss pushes the bitrate up again, and rising loss leaves the extra bits in place. Over a long call the bitrate drifts towards the 1 % worst case.
- **Zero loss.** A report of 0 % still triggers the function, which raises the bitrate even though libopus never codes FEC at 0 % loss.
- **CBR not modelled.** In CBR mode libopus 1.1 lowers SILK's target by 2 kbit/s, so the chosen bitrate falls short and FEC stays off.
- **No ceiling.** The new bitrate is not checked against the `maxaveragebitrate` negotiated in SDP.
- **FEC request not checked.** The function does not verify that in-band FEC was requested and negotiated.
- **Bandwidth assumed from the API rate.** The module picks the threshold from the API sample rate, while SILK uses its internal rate. At 48 kHz it assumes wideband, the highest base and so the safe choice.
- **Coupled to library internals.** Correctness rests on private constants and formulas. Chapter 8 shows what happened when they changed.

© 2026 DYSONSPHERE LIMITED

CHAPTER 7

Evolution in FreeSWITCH

The feature has been refined several times since 2015, and the libopus it depends on changed underneath it a year after it shipped.

Change	Date	What it did
FS-7772 · b3f453b0	3 Jul 2015	Introduced `keep-fec-enabled` and `switch_opus_keep_fec_enabled()`.
FS-8161 · 6385fb86	17 Sep 2015	Calls the function only when loss exceeds 10 % (below that, PLC is preferred); adds a pre-computed start-of-call bitrate table; applies the configured initial loss at encoder init.
FS-8644 · 0e6e53f1	23 Oct 2015	Adds `adjust-bitrate`: Kalman-filter estimators of loss and RTT driving bitrate from RTCP, alongside keep-FEC.
libopus 2b011b64	21 Jun 2016	Upstream moves the FEC decision from SILK into `opus_encoder.c` (shipped in libopus 1.2).
FreeSWITCH master	Oct 2026	Function still present, with a 0 % reset branch and a float frame rate; 16 kHz start table added; `keep_fec`, initial loss 20 % and CBR are the built-in defaults.

FS-8161 stopped the function running below 10 % loss, on the reasoning that PLC handles light loss well. That also removed the wasted increase at 0 % loss. Current master adds a branch that resets the bitrate to `maxaveragebitrate` when loss is zero, but because the function is only called above 10 % loss, the branch is not reached from the loss path: once raised, the bitrate still does not come back down when loss subsides. The start-of-call tables FS-8161 introduced are the 2015 algorithm's output pre-computed for the configured initial loss.

Today's defaults

In current master the built-in defaults are `keep_fec = 1`, an initial loss of 20 %, and CBR unless the configuration sets `use-vbr` (the shipped vanilla configuration does set it). With FEC negotiated, a call therefore starts at 17 200 bit/s for wideband or 13 200 bit/s for narrowband, the table values for 20 % loss. We encoded speech under exactly those settings with each library.

libopus	Rate mode	WB call, 17 200 bit/s	NB call, 13 200 bit/s
1.1	VBR	FEC, full WB	FEC
1.1	CBR	no FEC	no FEC
1.3.1 / 1.6.1	VBR	FEC, bandwidth narrowed	no FEC
1.3.1 / 1.6.1	CBR	no FEC	no FEC

Only libopus 1.1 in VBR mode delivers what the feature intends. The next chapter explains why.

© 2026 DYSONSPHERE LIMITED

CHAPTER 8

Does the model still hold? libopus 1.2 and later

On 21 June 2016, libopus commit `2b011b64`, "Moving FEC decision to the Opus encoder level (from SILK level)", replaced the rule FS-7772 reproduces. It shipped in libopus 1.2 and remains, with the same constants, in every release up to 1.6.1. SILK's `silk_setup_LBRR()` now simply obeys a flag; the decision is made by `decide_fec()` in `src/opus_encoder.c`.

```
if (!useInBandFEC || PacketLoss_perc == 0 || mode == MODE_CELT_ONLY)
   return 0;
for (;;) {
   LBRR_rate_thres_bps = fec_thresholds[2*(*bandwidth - OPUS_BANDWIDTH_NARROWBAND)];
   hysteresis          = fec_thresholds[2*(*bandwidth - OPUS_BANDWIDTH_NARROWBAND) + 1];
   if (last_fec == 1) LBRR_rate_thres_bps -= hysteresis;
   if (last_fec == 0) LBRR_rate_thres_bps += hysteresis;
   LBRR_rate_thres_bps = silk_SMULWB( silk_MUL( LBRR_rate_thres_bps,
         125 - silk_min( PacketLoss_perc, 25 ) ), SILK_FIX_CONST( 0.01, 16 ) );
   if (rate > LBRR_rate_thres_bps)           return 1;
   else if (PacketLoss_perc <= 5)            return 0;
   else if (*bandwidth > OPUS_BANDWIDTH_NARROWBAND) (*bandwidth)--;
   else break;
}
```

libopus 1.6.1, src/opus_encoder.c, decide_fec() (excerpt). BSD licence. Thresholds: NB 12 000, MB 14 000, WB 16 000, SWB 20 000, FB 22 000, each with 1 000 bit/s hysteresis.

Three changes that matter

1. A different rate is compared

The `rate` passed in is an "equivalent rate" from `compute_equiv_rate()`, not the byte-and-TOC rate of 1.1. It starts from the requested bitrate and applies penalties.

```
equiv = bitrate;
if (frame_rate > 50) equiv -= (40*channels+20)*(frame_rate - 50);
if (!vbr)            equiv -= equiv/12;                 /* CBR penalty */
equiv = equiv * (90+complexity)/100;
if (mode == MODE_SILK_ONLY || mode == MODE_HYBRID) {
   if (complexity<2) equiv = equiv*4/5;
   equiv -= equiv*loss/(6*loss + 10);                  /* loss penalty */
}
```

libopus 1.6.1, src/opus_encoder.c, compute_equiv_rate() for SILK mode (excerpt). BSD licence.

The loss penalty is large: at 10 % loss the equivalent rate is 14 % below the request, at 25 % it is 16 % below. The CBR penalty takes another 8 %, and complexity below 10 takes 1 % per step.

2. Hysteresis

The threshold base is raised by 1 000 bit/s while FEC is off and lowered by 1 000 bit/s once it is on. Switching FEC on therefore takes about 2.5 kbit/s more than keeping it on.

3. Bandwidth fallback

Above 5 % loss, if the rate is too low for FEC at the current bandwidth, libopus lowers the bandwidth one step at a time until FEC fits, down to narrowband. In effect, libopus 1.2 and later contain their own version of keep-FEC, but one that pays for redundancy with audio bandwidth instead of bitrate. At 5 % loss or below there is no fallback: FEC is simply off.

© 2026 DYSONSPHERE LIMITED

Measured thresholds

To see the combined effect we repeated the bitrate sweep on libopus 1.3.1 and 1.6.1 (identical results; 1.3.1 is still the version shipped by many Linux distributions). We recorded, for each loss level, the lowest bitrate at which every speech packet carried LBRR at full bandwidth, and, for wideband, the lowest at which LBRR appeared at any bandwidth.

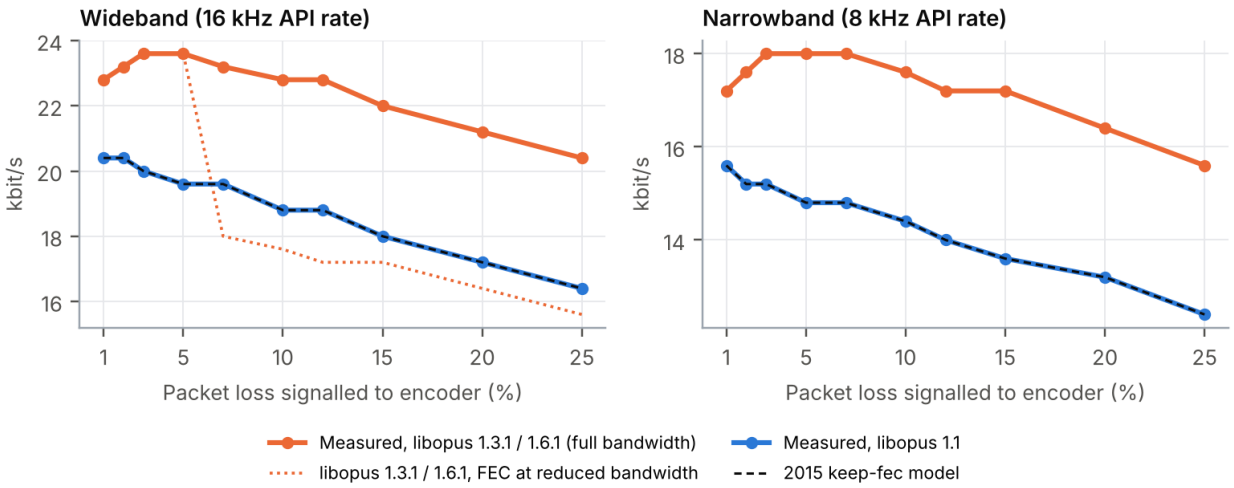


Figure 3. Lowest bitrate (VBR, 20 ms, complexity 10) at which every speech packet carries in-band FEC. Blue: libopus 1.1, matched exactly by the 2015 model (dashed). Orange: libopus 1.3.1 and 1.6.1. The dotted line shows where newer libraries switch FEC on by narrowing the bandwidth; narrowband cannot narrow further. Values in Table C.1.

The gap is 2.4–4.0 kbit/s at wideband and 1.6–3.6 kbit/s at narrowband. Read against the 2015 bitrates, it produces three regimes:

- **Wideband, loss ≤ 5 %.** The keep-FEC bitrate is below the new threshold and there is no fallback, so FEC stays off. The intervention spends extra bits for nothing.
- **Wideband, loss > 5 %.** FEC turns on, but because libopus narrows to medium-band or narrowband to make room for it. Callers lose the upper half of the speech band to gain redundancy.
- **Narrowband, any loss.** There is no narrower band to fall back to, so FEC stays off at every loss level.

> **FINDING**
>
> Against libopus 1.2 and later, the 2015 constants never achieve their goal of FEC at full bandwidth. Because FreeSWITCH links whatever libopus the system provides, most current deployments are in this situation.

© 2026 DYSONSPHERE LIMITED

CHAPTER 9

What FEC buys: speech-quality measurements

Method

A 3.1-second wideband English speech recording (the test file distributed with the open-source `python-pesq` project), repeated four times to 12.4 seconds, was encoded at a 16 kHz API rate, 20 ms frames, mono, complexity 10, VBR, with the expected loss set to the simulated loss rate. Packets were dropped at random (independent losses) using 20 different random seeds per condition. The decoder repaired a lost packet with FEC when the next packet had arrived and in-band FEC was in use, and used PLC otherwise. Each decoded output was scored against the original with PESQ in wideband mode (ITU-T P.862.2, MOS-LQO), and the 20 scores were averaged.

Four encoder configurations were compared at each loss rate:

- **A** · 16 kbit/s with FEC requested: what libopus does on its own at a modest bitrate.
- **B** · The bitrate chosen by the 2015 keep-FEC algorithm.
- **C** · The bitrate chosen by this paper's reference implementation (Chapter 10).
- **D** · C's bitrate with FEC off at both ends: a control that isolates the effect of the extra bits from the effect of FEC.

Results

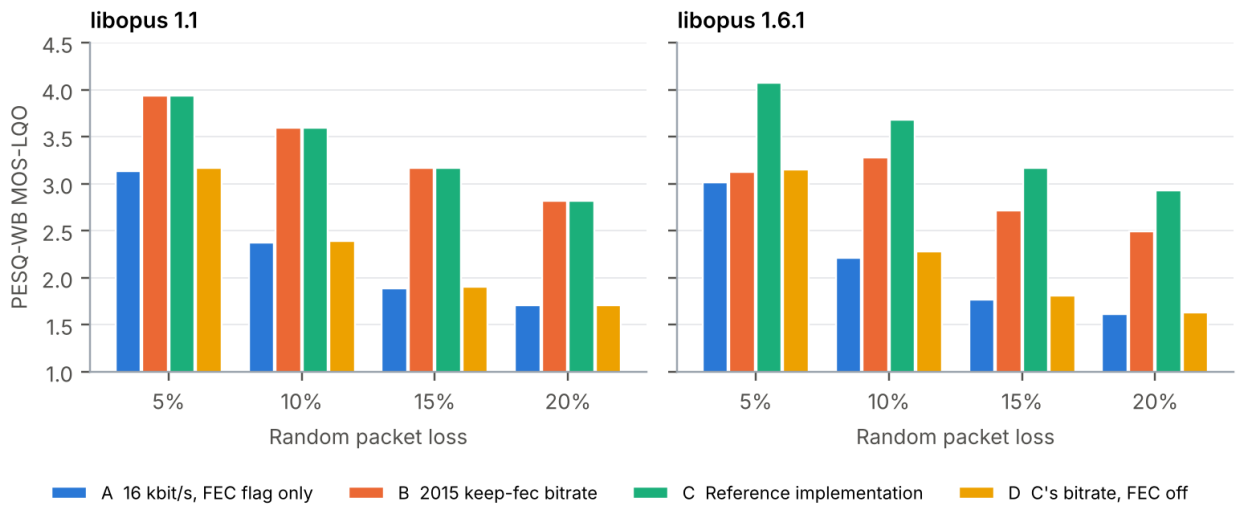


Figure 4. Wideband PESQ MOS-LQO, mean of 20 random-loss trials per bar. Left: libopus 1.1, where B and C choose the same bitrate. Right: libopus 1.6.1, where B's FEC is off at 5 % and narrowed above it, and C keeps full-bandwidth FEC. Values in Tables 9.1 and 9.2.

© 2026 DYSONSPHERE LIMITED

Table 9.1. libopus 1.1: MOS-LQO and measured bitrate.

Loss	A · 16 kbit/s, FEC flag	B · 2015 keep-fec	C · Reference impl.	D · C's rate, no FEC
5%	**3.14** 15.9 kb/s no FEC	**3.94** 19.3 kb/s	**3.94** 19.3 kb/s	**3.17** 19.3 kb/s no FEC
10%	**2.37** 15.9 kb/s no FEC	**3.59** 18.4 kb/s	**3.59** 18.4 kb/s	**2.39** 18.5 kb/s no FEC
15%	**1.88** 15.9 kb/s no FEC	**3.16** 17.5 kb/s	**3.16** 17.5 kb/s	**1.90** 17.8 kb/s no FEC
20%	**1.70** 15.9 kb/s no FEC	**2.82** 16.8 kb/s	**2.82** 16.8 kb/s	**1.71** 17.1 kb/s no FEC

Table 9.2. libopus 1.6.1: MOS-LQO and measured bitrate. "Narrowed" means FEC was on but libopus lowered the bandwidth to fit it.

Loss	A · 16 kbit/s, FEC flag	B · 2015 keep-fec	C · Reference impl.	D · C's rate, no FEC
5%	**3.02** 15.4 kb/s no FEC	**3.12** 19.0 kb/s no FEC	**4.08** 22.7 kb/s	**3.15** 23.0 kb/s no FEC
10%	**2.21** 15.4 kb/s no FEC	**3.28** 18.1 kb/s narrowed	**3.68** 21.9 kb/s	**2.28** 22.2 kb/s no FEC
15%	**1.76** 15.4 kb/s no FEC	**2.71** 17.2 kb/s narrowed	**3.17** 21.1 kb/s	**1.81** 21.4 kb/s no FEC
20%	**1.61** 15.5 kb/s no FEC	**2.49** 16.4 kb/s narrowed	**2.93** 20.2 kb/s	**1.63** 20.6 kb/s no FEC

Reading the results

The gain comes from FEC. In every case, configuration D, which has the same bitrate as C but no FEC, scores at most 0.13 MOS above A. A few extra kbit/s barely matter on their own; spent on redundancy, they lift scores by 0.8 to 1.4 MOS over D.

On libopus 1.1 the 2015 algorithm is optimal. B and C choose identical bitrates and produce identical results.

On libopus 1.6.1 the 2015 algorithm leaves quality on the table. At 5 % loss it adds 3.6 kbit/s that buy 0.1 MOS, because FEC never turns on. Above 5 % its FEC works, but only by narrowing the bandwidth, and it trails C by 0.41–0.45 MOS. C costs about 4 kbit/s more than B and recovers full-bandwidth speech with redundancy.

The same pattern held on libopus 1.3.1 (Appendix C).

Bursty loss

Real networks often lose packets in runs. We repeated the libopus 1.6.1 test at 10 % average loss with a two-state Gilbert–Elliott model, varying the mean length of a loss burst.

© 2026 DYSONSPHERE LIMITED

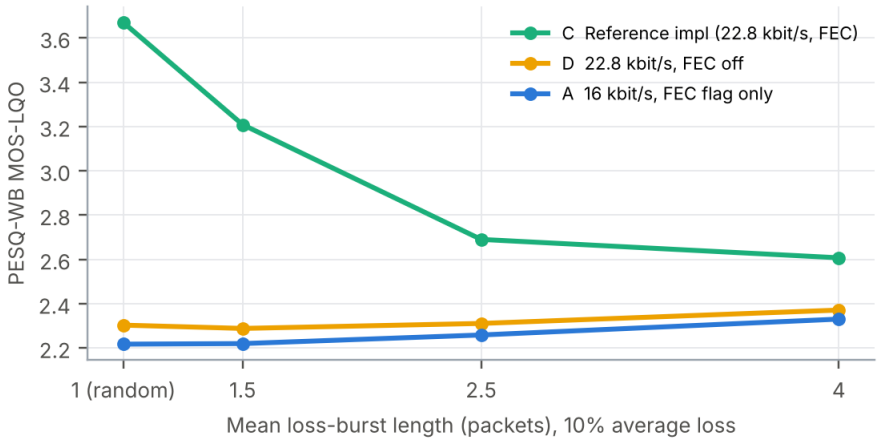


Figure 5. PESQ MOS-LQO at 10 % average loss as loss becomes burstier (libopus 1.6.1, 20 trials per point). FEC protects only a packet whose successor arrives, so its benefit shrinks as bursts lengthen.

Table 9.3. Burst-length results.

Mean burst (packets)	A · 16 kbit/s	D · 22.8 kbit/s, no FEC	C · 22.8 kbit/s, FEC	FEC gain (C − D)
1 (random)	2.22	2.30	3.67	**+1.37**
1.5	2.22	2.29	3.21	**+0.92**
2.5	2.26	2.31	2.69	**+0.38**
4	2.33	2.37	2.61	**+0.24**

With independent losses FEC is worth 1.37 MOS; with bursts averaging four packets it is worth 0.24. Keep-FEC is most valuable on links with scattered losses, such as busy Wi-Fi or lightly congested queues. Long outages need different tools: larger jitter buffers, RED-style duplication, or neural redundancy such as DRED.

Limits of this evaluation

PESQ is an objective estimate and does not replace listening tests. The results come from a single talker, a short recording, one frame size and simulated loss. The direction and rough size of the effects are robust across all conditions tested, but absolute MOS values will differ with other material.

© 2026 DYSONSPHERE LIMITED

CHAPTER 10

A version-aware reference implementation

The reference implementation keeps the 2015 idea, reproducing the encoder's rule and steering the bitrate, and fixes the limitations found in Chapters 6 to 8. It is a self-contained C module of about 140 lines that depends only on `<opus.h>`; the full source is in Appendix B.

Design goals

- **Version-aware.** Detect at runtime whether the linked libopus is 1.1 or 1.2+ and apply the matching model.
- **Complete model.** Cover VBR and CBR, every SILK frame size, channel count and complexity, including the CBR and low-complexity penalties.
- **Two-way.** Return to the base bitrate when FEC is not needed, instead of ratcheting upwards.
- **Respect negotiation.** Never exceed the SDP `maxaveragebitrate`, and do nothing if FEC was not negotiated.
- **Conservative by default.** Model the stricter switch-on threshold of libopus 1.2+ hysteresis, so FEC is on from the first packet after a change.

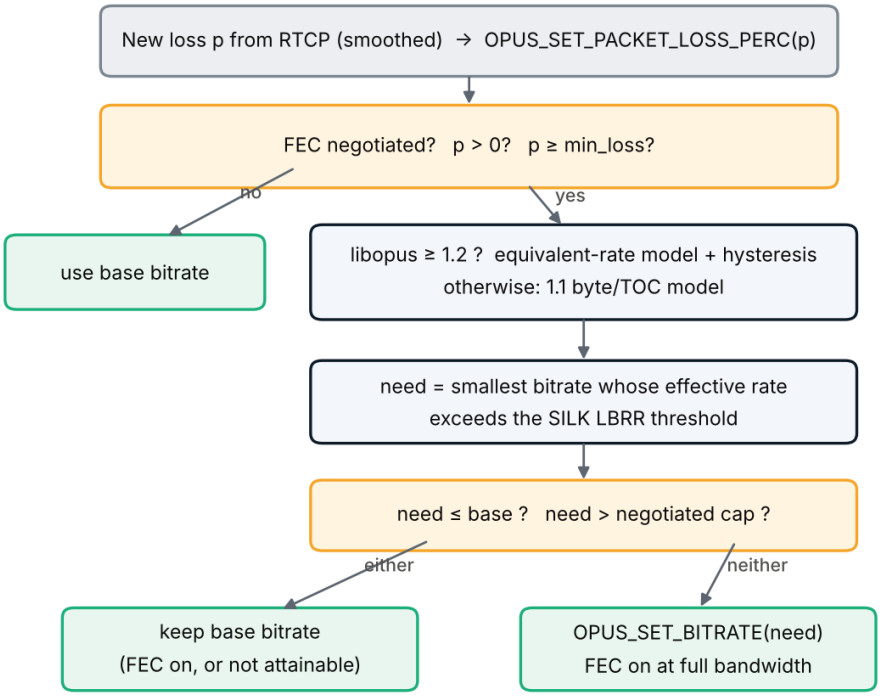


Figure 6. Decision flow of the reference implementation.

© 2026 DYSONSPHERE LIMITED

API

```c
typedef struct {
    int api_fs;          /* encoder API sample rate (8000..48000)            */
    int ptime_ms;        /* frame size: 10, 20, 40 or 60                      */
    int channels;        /* 1 or 2                                            */
    int complexity;      /* OPUS_SET_COMPLEXITY value, 0..10                  */
    int cbr;             /* 1 if OPUS_SET_VBR(0)                              */
    int fec_enabled;     /* useinbandfec negotiated and OPUS_SET_INBAND_FEC   */
    int base_bitrate;    /* bitrate wanted when FEC is not a concern          */
    int max_bitrate;     /* negotiated maxaveragebitrate, 0 = no cap          */
    int min_loss;        /* below this loss %, rely on PLC instead            */
    int libopus_ge_1_2;  /* from kf_detect_libopus_ge_1_2()                   */
} kf_params;

int kf_detect_libopus_ge_1_2(void);

/* Smallest bitrate (400 bit/s granularity) at which the linked libopus will
 * code LBRR at the protected bandwidth for this loss; 0 if none. */
int kf_fec_bitrate(const kf_params *p, int loss, int last_fec);

/* Bitrate to apply with OPUS_SET_BITRATE after OPUS_SET_PACKET_LOSS_PERC.
 * Moves up AND down: returns base_bitrate when FEC is unnecessary or cannot
 * fit under max_bitrate. *fec_expected reports whether FEC should be on. */
int kf_choose_bitrate(const kf_params *p, int loss, int last_fec, int *fec_expected);
```

A caller fills `kf_params` once per encoder and calls `kf_choose_bitrate()` whenever the loss estimate changes, then applies the result with `OPUS_SET_BITRATE`. `kf_fec_bitrate()` exposes the raw minimum for diagnostics and dashboards.

Verification

Each library was built from its release tag, the module was compiled against it, and for every configuration the bitrate returned by `kf_choose_bitrate()` was used to encode speech. A configuration passed if LBRR appeared in the speech packets and every packet stayed at the full SILK bandwidth for its API rate. The bitrate 400 bit/s lower was also tested, to measure how tight the choice is.

The VBR matrix covered NB, MB and WB API rates; 10, 20, 40 and 60 ms frames; complexity 10, 5 and 1; and loss of 1, 3, 5, 10, 15 and 25 %: 216 configurations per library. A CBR matrix added 20 configurations per library at NB and WB, 20 and 40 ms, and five loss levels.

Library	Configurations	FEC on, full bandwidth	Failures	Exact minimum	One step (400 bit/s) above minimum
libopus 1.1	236	234	2	236	0
libopus 1.3.1	236	236	0	158	78
libopus 1.6.1	236	236	0	158	78

All 472 configurations on libopus 1.3.1 and 1.6.1 passed. On libopus 1.1, 234 of 236 passed. The two exceptions (wideband, 10 ms frames, complexity 5 and 1, 25 % loss) carried FEC as intended, but libopus 1.1 chose medium-band coding for its own reasons at that frame size and complexity. For libopus 1.1 every choice was the exact minimum. For 1.2 and later, about one choice in three is 400 bit/s above the minimum, a consequence of evaluating thresholds at 400 bit/s granularity. In separate checks with the encoder complexity left at the libopus default of 9, all eight tested configurations also passed.

© 2026 DYSONSPHERE LIMITED

Integrating with mod_opus

The following sketch shows where the module plugs into FreeSWITCH. It is illustrative and is not part of any FreeSWITCH release.

```
/* At encoder init, after the encoder is configured: */
context->kf.api_fs         = enc_samplerate;
context->kf.ptime_ms       = codec->implementation->microseconds_per_packet / 1000;
context->kf.channels       = opus_codec_settings.stereo ? 2 : 1;
context->kf.complexity     = opus_prefs.complexity ? opus_prefs.complexity : 9; /* libopus default
*/
context->kf.cbr            = !opus_prefs.use_vbr;
context->kf.fec_enabled    = opus_codec_settings.useinbandfec;
context->kf.base_bitrate   = configured_bitrate;          /* what you would use without FEC */
context->kf.max_bitrate    = negotiated_maxaveragebitrate; /* 0 if none */
context->kf.min_loss       = 3;                           /* see Chapter 10, Tuning */
context->kf.libopus_ge_1_2 = kf_detect_libopus_ge_1_2();

/* In switch_opus_control(), case SCC_AUDIO_PACKET_LOSS: */
if (plpct != context->old_plpct) {
    int fec_on = 0;
    opus_encoder_ctl(context->encoder_object, OPUS_SET_PACKET_LOSS_PERC(plpct));
    if (opus_prefs.keep_fec) {
        int b = kf_choose_bitrate(&context->kf, plpct, 0, &fec_on);
        opus_encoder_ctl(context->encoder_object, OPUS_SET_BITRATE(b));
    }
    context->old_plpct = plpct;
}
```

Tuning

Minimum loss

`min_loss` sets the loss below which the module leaves the bitrate alone. FreeSWITCH master uses 10 %. Our measurements show clear FEC gains from 5 %, so a value between 3 and 5 % is a reasonable starting point where bandwidth allows.

Smoothing

RTCP loss figures are noisy. Smoothing them, for example with a short moving average or the existing Kalman estimators, before calling the module avoids bitrate flapping.

Hysteresis

Passing `last_fec = 1` after FEC is confirmed on lets the bitrate settle about 2 kbit/s lower on libopus 1.2 and later, matching the library's own hysteresis. This saves bandwidth, but it was not part of the verification above; the conservative default is 0.

© 2026 DYSONSPHERE LIMITED

CHAPTER 11

Operator guide

Know your libopus

Everything in this paper depends on the library version. FreeSWITCH builds against the system's `libopus-dev`. Check the version on each media server, for example with `pkg-config --modversion opus` or your package manager. If it is 1.2 or later, which is very likely, the 2015 keep-FEC constants do not achieve full-bandwidth FEC.

Recommended configuration

```
<configuration name="opus.conf">
  <settings>
    <param name="use-vbr" value="1"/>              <!-- avoid the CBR penalty -->
    <param name="complexity" value="10"/>
    <param name="keep-fec-enabled" value="1"/>
    <param name="advertise-useinbandfec" value="1"/>
    <param name="use-jb-lookahead" value="1"/>     <!-- let the decoder use FEC -->
    <param name="packet-loss-percent" value="10"/> <!-- initial assumption -->
    <param name="maxaveragebitrate" value="0"/>    <!-- or a cap >= the FEC rate -->
  </settings>
</configuration>
```

- **Use VBR.** CBR raises the FEC threshold by roughly 2 kbit/s in every libopus version. The shipped vanilla configuration sets `use-vbr`; make sure yours does too, because the built-in default without it is CBR.
- **Let the decoder use FEC.** Keep `advertise-useinbandfec` and `use-jb-lookahead` on, so the far end sends FEC and FreeSWITCH's decoder can reach it.
- **Leave headroom in caps.** If you set `maxaveragebitrate`, keep it above the FEC bitrate for your bandwidth (Table C.1): in VBR at 20 ms, about 24 kbit/s covers wideband at any loss on current libopus, and 18 kbit/s covers narrowband.
- **Choose the initial loss deliberately.** `packet-loss-percent` sets both the encoder's starting assumption and the start-of-call bitrate. On current libopus the built-in 20 % start-of-call bitrates (17 200 / 13 200 bit/s) are too low for full-bandwidth FEC.
- **Prefer 20 ms packets.** 10 ms frames add about 3 kbit/s of overhead in libopus 1.2+, raising every threshold.

Verifying FEC on the wire

Configuration is a hypothesis; the packets are the evidence. On a captured call, extract the Opus payloads and call `opus_packet_has_lbrr()` (libopus 1.5 and later) on each one, or decode the SILK LBRR flag yourself. In FreeSWITCH, `opus_debug on` in the console logs encoder decisions, including keep-FEC bitrate changes. A healthy protected call shows LBRR in nearly every speech packet once loss is reported, with the expected bandwidth in the TOC byte.

© 2026 DYSONSPHERE LIMITED

Checklist

1. Identify the libopus version on every media server.
2. Confirm both sides negotiate `useinbandfec=1`.
3. Set `use-vbr` to 1.
4. Check that `maxaveragebitrate`, yours and the far end's, leaves room for FEC.
5. Decide the minimum loss at which FEC is worth its bits.
6. On libopus 1.2+, replace or patch the 2015 constants with a version-aware model such as the one in Chapter 10.
7. Verify LBRR and bandwidth in captured packets under induced loss.

© 2026 DYSONSPHERE LIMITED

CHAPTER 12

Conclusions

The keep-FEC technique rests on a simple observation: an encoder told to add FEC may silently decline, and the decision depends on a bitrate threshold the application can compute. Reproducing that private rule exactly, and steering the public bitrate control to satisfy it, delivers full-bandwidth redundancy at minimum cost. For libopus 1.1 the 2015 implementation did exactly that, and our measurements confirm both its precision and its value: around a full MOS point of wideband speech quality at 10 % random loss, for under 3 kbit/s.

The technique's weakness is the same as its strength. Because it mirrors library internals, it is only as correct as its copy of them, and libopus rewrote those internals a year later. On every libopus released since 2017 the original constants miss, and the effects range from wasted bits to narrowed bandwidth to no FEC at all.

The fix keeps the idea and updates the model. A version-aware implementation that reproduces both generations of the rule, models CBR and complexity, respects negotiated limits and adjusts in both directions achieved full-bandwidth FEC in every configuration tested on current libraries. Operators can capture most of the benefit today by running VBR, leaving headroom in their bitrate caps, and checking their packets.

The broader lesson applies beyond Opus. When an application depends on a library's undocumented behaviour, it should detect the library version, test against each generation it supports, and verify the outcome on the wire rather than assume it.

© 2026 DYSONSPHERE LIMITED

APPENDIX A

Original 2015 function (MPL 1.1)

From FreeSWITCH commit `b3f453b02f62da450681c4b584bda838134c1317`, `src/mod/codecs/mod_opus/mod_opus.c`, by Dragos Oancea, 3 July 2015. Reproduced under the Mozilla Public License 1.1; tabs converted to spaces.

```
static switch_status_t switch_opus_keep_fec_enabled(switch_codec_t *codec)
{
    struct opus_context *context = codec->private_info;
    opus_int32 current_bitrate;
    opus_int32 current_loss;
    uint32_t LBRR_threshold_bitrate,LBRR_rate_thres_bps,real_target_bitrate ;
    opus_int32 a32,b32;
    uint32_t fs = context->enc_frame_size * 1000 / (codec->implementation-
>microseconds_per_packet / 1000);
    float frame_rate = 1000 / (codec->implementation->microseconds_per_packet / 1000);
    uint32_t step = 8000 / (codec->implementation->microseconds_per_packet / 1000); /* this works
for 10 ms, 20 ms and 40 ms frame sizes. not for 60 ms*/

    opus_encoder_ctl(context->encoder_object, OPUS_GET_BITRATE(&current_bitrate));
    opus_encoder_ctl(context->encoder_object, OPUS_GET_PACKET_LOSS_PERC(&current_loss));
    if( fs == 8000 ) {
        LBRR_rate_thres_bps = 12000; /*LBRR_NB_MIN_RATE_BPS*/
    } else if( fs == 12000 ) {
        LBRR_rate_thres_bps = 14000; /*LBRR_MB_MIN_RATE_BPS*/
    } else {
        LBRR_rate_thres_bps = 16000; /*LBRR_WB_MIN_RATE_BPS*/
    }
    /*see opus-1.1/src/opus_encoder.c , opus_encode_native() */
    real_target_bitrate =  8 * (current_bitrate * context->enc_frame_size / ( fs * 8 ) - 1) *
frame_rate ;
    /*check if the internally used bitrate is above the threshold defined in opus-1.1/silk/
control_codec.c  */
    a32 =  LBRR_rate_thres_bps * (125 -(((current_loss) < (25)) ? (current_loss) :  (25)));
    b32 =  ((opus_int32)((0.01) * ((opus_int64)1 << (16)) + 0.5));
    LBRR_threshold_bitrate =  (a32 >> 16) * (opus_int32)((opus_int16)b32) + (((a32 & 0x0000FFFF) *
(opus_int32)((opus_int16)b32)) >> 16);
    if ((!real_target_bitrate || !LBRR_threshold_bitrate)){
        switch_log_printf(SWITCH_CHANNEL_LOG, SWITCH_LOG_ERROR,"Opus encoder: error while
controlling FEC params\n");
        return SWITCH_STATUS_FALSE;
    }
    /* Is there any FEC at the current bitrate and requested packet loss ?
     * If yes, then keep the current bitrate. If not, modify bitrate to keep FEC on. */
    if (real_target_bitrate > LBRR_threshold_bitrate) {
        /*FEC is already enabled, do nothing*/
        if (globals.debug)
            switch_log_printf(SWITCH_CHANNEL_LOG, SWITCH_LOG_DEBUG,"Opus encoder: FEC is
enabled\n");
        return SWITCH_STATUS_SUCCESS;
    } else {
        while (real_target_bitrate <= LBRR_threshold_bitrate) {
            current_bitrate += step;
            real_target_bitrate =  8 * (current_bitrate * context->enc_frame_size / ( fs * 8 ) -
1) * frame_rate ;
        }
        opus_encoder_ctl(context->encoder_object,OPUS_SET_BITRATE(current_bitrate));
        if (globals.debug)
            switch_log_printf(SWITCH_CHANNEL_LOG,
SWITCH_LOG_DEBUG,"Opus encoder: increased bitrate to [%d] to keep FEC enabled\n",current_bitrate);
        return SWITCH_STATUS_SUCCESS;
    }
}
```

© 2026 DYSONSPHERE LIMITED

APPENDIX B

Reference implementation source

© 2026 DYSONSPHERE LIMITED. See the copyright page for the terms of use. Compile alongside your application and link with libopus.

keepfec.h

```h
#ifndef KEEPFEC_H
#define KEEPFEC_H

typedef struct {
    int api_fs;          /* encoder API sample rate (8000..48000)            */
    int ptime_ms;        /* frame size: 10, 20, 40 or 60                      */
    int channels;        /* 1 or 2                                            */
    int complexity;      /* OPUS_SET_COMPLEXITY value, 0..10                  */
    int cbr;             /* 1 if OPUS_SET_VBR(0)                              */
    int fec_enabled;     /* useinbandfec negotiated and OPUS_SET_INBAND_FEC   */
    int base_bitrate;    /* bitrate wanted when FEC is not a concern          */
    int max_bitrate;     /* negotiated maxaveragebitrate, 0 = no cap          */
    int min_loss;        /* below this loss %, rely on PLC instead            */
    int libopus_ge_1_2;  /* from kf_detect_libopus_ge_1_2()                   */
} kf_params;

int kf_detect_libopus_ge_1_2(void);

/* Smallest bitrate (400 bit/s granularity) at which the linked libopus will
 * code LBRR at the protected bandwidth for this loss; 0 if none. */
int kf_fec_bitrate(const kf_params *p, int loss, int last_fec);

/* Bitrate to apply with OPUS_SET_BITRATE after OPUS_SET_PACKET_LOSS_PERC.
 * Moves up AND down: returns base_bitrate when FEC is unnecessary or cannot
 * fit under max_bitrate. *fec_expected reports whether FEC should be on. */
int kf_choose_bitrate(const kf_params *p, int loss, int last_fec, int *fec_expected);

#endif
```

keepfec.c

```c
/*
 * keepfec.c - version-aware "keep FEC enabled" bitrate controller for libopus.
 *
 * Reference implementation accompanying the DYSONSPHERE LIMITED technical
 * paper "Keeping Opus In-Band FEC Alive". Depends only on <opus.h>.
 *
 * Copyright (c) 2026 DYSONSPHERE LIMITED. See the paper's copyright page
 * for the terms under which this code may be used.
 */
#include <string.h>
#include <stdlib.h>
#include <opus.h>
#include "keepfec.h"

/* SILK's fixed-point SMULWB(a, b) for b = 655 (0.01 in Q16). */
static int smulwb_001(int a)
{
    const int b = 655;
    return (a >> 16) * b + (((a & 0xFFFF) * b) >> 16);
}

int kf_detect_libopus_ge_1_2(void)
{
    /* "libopus 1.1.5", "libopus 1.3.1", "libopus v1.6.1", ... */
    const char *s = opus_get_version_string();
    const char *p = strpbrk(s, "0123456789");
    int major, minor;
```

```
    if (!p) return 1;
    major = atoi(p);
    p = strchr(p, '.');
    minor = p ? atoi(p + 1) : 0;
    return major > 1 || (major == 1 && minor >= 2);
}

/* SILK base threshold for the bandwidth that is to be protected. Only
 * SILK-coded bandwidths (NB/MB/WB) carry LBRR; above 16 kHz the target is
 * WB, which is what the encoder selects at the bitrates involved here. */
static int base_threshold(int api_fs)
{
    if (api_fs <= 8000)  return 12000;
    if (api_fs <= 12000) return 14000;
    return 16000;
}

/* Bitrate SILK compares against the threshold, given the user bitrate. */
static int effective_rate(const kf_params *p, int b, int loss)
{
    if (!p->libopus_ge_1_2) {
        /* libopus 1.1: opus_encode_native() floors to whole bytes per
         * frame, removes the TOC byte, and scales by an INTEGER frame
         * rate (Fs/frame_size: 16, not 16.67, at 60 ms). */
        int frame_rate = 1000 / p->ptime_ms;
        int bytes = (int)((long long)b * p->ptime_ms / 8000);
        if (p->cbr) {
            /* CBR rounds to the nearest byte and then lowers SILK's
             * target by 2 kbit/s "to make it easier to reach the CBR rate". */
            bytes = (b + 4 * frame_rate) / (8 * frame_rate);
            return 8 * (bytes - 1) * frame_rate - 2000;
        }
        return 8 * (bytes - 1) * frame_rate;
    } else {
        /* libopus >= 1.2: compute_equiv_rate() for SILK mode. */
        long long eq = b;
        int frame_rate = 1000 / p->ptime_ms;
        if (frame_rate > 50) eq -= (40LL * p->channels + 20) * (frame_rate - 50);
        if (p->cbr) eq -= eq / 12;
        eq = eq * (90 + p->complexity) / 100;
        if (p->complexity < 2) eq = eq * 4 / 5;
        eq -= eq * loss / (6 * loss + 10);
        return (int)eq;
    }
}

static int fec_threshold(const kf_params *p, int loss, int last_fec)
{
    int base = base_threshold(p->api_fs);
    int l = loss < 25 ? loss : 25;
    if (p->libopus_ge_1_2) base += last_fec ? -1000 : 1000;   /* hysteresis */
    return smulwb_001(base * (125 - l));
}

int kf_fec_bitrate(const kf_params *p, int loss, int last_fec)
{
    int t, b;
    if (loss <= 0) return 0;
    t = fec_threshold(p, loss, last_fec);
    /* Monotonic search in 400 bit/s steps (one byte per 20 ms frame). */
    for (b = 6000; b <= 128000; b += 400)
        if (effective_rate(p, b, loss) > t) return b;
    return 0;
}

int kf_choose_bitrate(const kf_params *p, int loss, int last_fec, int *fec_expected)
{
    int need, target = p->base_bitrate;
    if (fec_expected) *fec_expected = 0;

    /* Nothing to protect: FEC off, no loss, or loss low enough for PLC. */
    if (!p->fec_enabled || loss <= 0 || loss < p->min_loss) return target;

    need = kf_fec_bitrate(p, loss, last_fec);
```

© 2026 DYSONSPHERE LIMITED

```
    if (!need) return target;

    if (need <= target) {               /* base rate already carries FEC */
        if (fec_expected) *fec_expected = 1;
        return target;
    }
    if (p->max_bitrate && need > p->max_bitrate) {
        /* Cannot keep FEC within the negotiated cap. Stay at the base rate;
         * libopus >= 1.2 may still narrow the bandwidth to fit FEC. */
        return target;
    }
    if (fec_expected) *fec_expected = 1;
    return need;
}
```

© 2026 DYSONSPHERE LIMITED

APPENDIX C

Measurement data

C.1 FEC switch-on bitrates

Lowest bitrate from which every speech packet carried LBRR at full bandwidth (VBR, 20 ms, mono, complexity 10), searched in 400 bit/s steps from 10 to 30 kbit/s. “≥1.2” denotes libopus 1.3.1 and 1.6.1, which gave identical results. “Reduced bw” is the lowest bitrate at which LBRR appeared at any bandwidth. “Ref. impl” is the bitrate the reference implementation chooses for libopus ≥1.2.

Loss	Wideband, 16 kHz API rate (bit/s)					Narrowband, 8 kHz API rate (bit/s)				
	2015 model	1.1 meas.	≥1.2 meas.	≥1.2 reduced bw	Ref. impl	2015 model	1.1 meas.	≥1.2 meas.	≥1.2 reduced bw	Ref. impl
1%	20 400	20 400	22 800	22 800	22 800	15 600	15 600	17 200	—	17 200
2%	20 400	20 400	23 200	23 200	23 200	15 200	15 200	17 600	—	17 600
3%	20 000	20 000	23 600	23 600	23 600	15 200	15 200	18 000	—	18 000
5%	19 600	19 600	23 600	23 600	23 600	14 800	14 800	18 000	—	18 000
7%	19 600	19 600	23 200	18 000	23 200	14 800	14 800	18 000	—	18 000
10%	18 800	18 800	22 800	17 600	22 800	14 400	14 400	17 600	—	17 600
12%	18 800	18 800	22 800	17 200	22 800	14 000	14 000	17 200	—	17 200
15%	18 000	18 000	22 000	17 200	22 000	13 600	13 600	17 200	—	17 200
20%	17 200	17 200	21 200	16 400	21 200	13 200	13 200	16 400	—	16 400
25%	16 400	16 400	20 400	15 600	20 400	12 400	12 400	15 600	—	15 600

C.2 Speech quality, libopus 1.3.1

Loss	A · 16 kbit/s, FEC flag	B · 2015 keep-fec	C · Reference impl.	D · C's rate, no FEC
5%	**3.07** 15.4 kb/s no FEC	**3.11** 19.0 kb/s no FEC	**4.10** 22.9 kb/s	**3.15** 23.0 kb/s no FEC
10%	**2.28** 15.5 kb/s no FEC	**3.30** 18.4 kb/s narrowed	**3.76** 22.2 kb/s	**2.31** 22.3 kb/s no FEC
15%	**1.81** 15.5 kb/s no FEC	**2.72** 17.5 kb/s narrowed	**3.02** 21.4 kb/s	**1.84** 21.5 kb/s no FEC
20%	**1.63** 15.6 kb/s no FEC	**2.52** 16.7 kb/s narrowed	**2.80** 20.6 kb/s	**1.66** 20.7 kb/s no FEC

© 2026 DYSONSPHERE LIMITED

C.3 Test environment

Libraries	libopus 1.1, 1.3.1 and 1.6.1, built from the official git tags (floating-point build, x86-64, gcc -O2)
Encoder	OPUS_APPLICATION_VOIP, mono, OPUS_SIGNAL_VOICE, in-band FEC requested, expected loss set to the test loss
FEC detection	`opus_packet_has_lbrr()` and `opus_packet_get_bandwidth()` from libopus 1.6.1
Speech	3.1 s wideband English speech from the python-pesq project test set; 8 and 12 kHz versions by polyphase resampling
Quality metric	PESQ, ITU-T P.862.2 wideband, MOS-LQO, via the python-pesq implementation
Loss models	Independent (Bernoulli) loss; Gilbert–Elliott two-state model for bursts
Trials	20 random seeds per condition

© 2026 DYSONSPHERE LIMITED

APPENDIX D

References

1. J.-M. Valin, K. Vos, T. Terriberry. *Definition of the Opus Audio Codec.* IETF RFC 6716, September 2012.
2. J. Spittka, K. Vos, J.-M. Valin. *RTP Payload Format for the Opus Speech and Audio Codec.* IETF RFC 7587, June 2015.
3. Xiph.Org Foundation. *libopus source code*, releases 1.1 to 1.6.1. github.com/xiph/opus. In particular `silk/control_codec.c` and `src/opus_encoder.c`.
4. Xiph.Org Foundation. libopus commit 2b011b64, "Moving FEC decision to the Opus encoder level (from SILK level)", 21 June 2016.
5. FreeSWITCH. *mod_opus* and *switch_rtp.c* source code. github.com/signalwire/freeswitch. Commits b3f453b0 (FS-7772), 6385fb86 (FS-8161), 0e6e53f1 (FS-8644).
6. ITU-T Recommendation P.862.2. *Wideband extension to Recommendation P.862 for the assessment of wideband telephone networks and speech codecs.*
7. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications.* IETF RFC 3550, July 2003.
8. C. Perkins et al. *RTP Payload for Redundant Audio Data.* IETF RFC 2198, September 1997.

© 2026 DYSONSPHERE LIMITED



DYSONSPHERE LIMITED

Keeping Opus In-Band FEC Alive · Edition 1.0 · October 2026