8 November 2024

IETF 121

draft-ietf-mlcodec-opus-extension

Draft Status

- No change since Prague

Repeat These Extensions Proposal

- Use case: reduce overhead of using the same extensions for multiple Opus frames in the same packet
  - E.g., Hybrid Mode or CELT in 60 ms packets
- Benefits of this proposal
  - Can reduce overhead even with just 1 extension appearing in 2 frames
  - Savings scale with the number of frames and repeated extensions
  - Applies to any extension: no extra IDs to register or SDP signaling
  - Integrates well with non-repeated extensions (e.g., DRED)
  - Doing it later would be a breaking change to extension parsing
- Costs
  - Additional implementation complexity (entirely optional for encoder)
  - Extensions for a frame no longer guaranteed to be physically contiguous

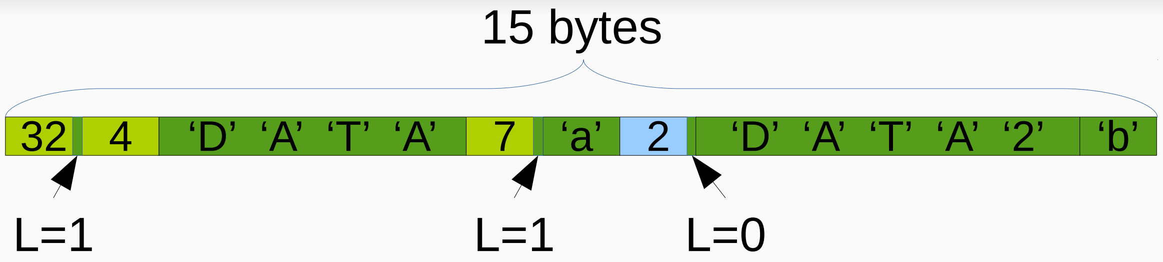
RTE Updates Since Vancouver

- Using RTE with L=0 with a long extension followed by short extensions can now elide the final length, even if those short extensions have a payload
- Using RTE with L=0 and no long extensions increments the current frame number
- Implementation at https://gitlab.xiph.org/xiph/opus/-/merge_requests/132

L=0 with Long, then Short Extensions

- Previously: the length of the final long extension was omitted if followed by zero or more short extensions with no payload
  - These extensions do not need any space, so the payload for the final long extension was simply the rest of the packet
- Now: the length of the final long extension is omitted if followed by zero or more short extensions, even if they have a payload
  - Have to track how much space is needed for those short extension payloads
  - Parsing can now fail if the implied length of the final long extension is negative
  - Some condition checks slightly simpler and easier to explain

L=0 with Long, then Short Extensions: Example



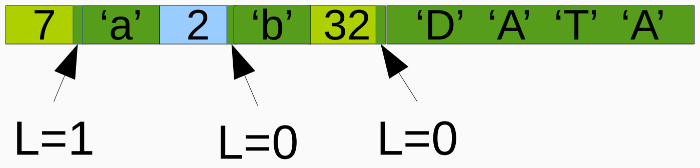
Decodes to:

ID	Frame	Length	Payload
32	0	4	"DATA"
7	0	1	"a"
32	1	5	"DATA2"
7	1	1	"b"

L=0 with Only Short Extensions

- L=0 on a long extension normally takes up the rest of the packet
- After the previous change, RTE with L=0 does this if any long extension is repeated
- But what do we do if only short extensions are repeated?
  - Non-repeated extension decoding continues from the frame *after* the current frame
  - Can save the 1 byte cost of a frame separator in some cases

L=0 with Only Short Extensions: Example



Decodes to:

ID	Frame	Length	Payload
7	0	1	“a”
7	1	1	“b”
32	1	4	“DATA”

Bitrate Savings

- Bitrate savings: (nb_repeated_extensions + 1)*(nb_frames – 1) + (up to one length) – (frame separators needed for non-repeated extensions) – 1
- Examples with 3×20ms frames:
  - 1 L=1 short extension per frame:
    - Savings: 9 bytes → 6 bytes
    - 200% overhead reduced to 100%
  - 2 repeated extensions + DRED in the first frame:
    - Savings: 2 frame separators + 4 extension IDs + DRED length – RTE byte
    - Total: 6 bytes / packet or 800 bps

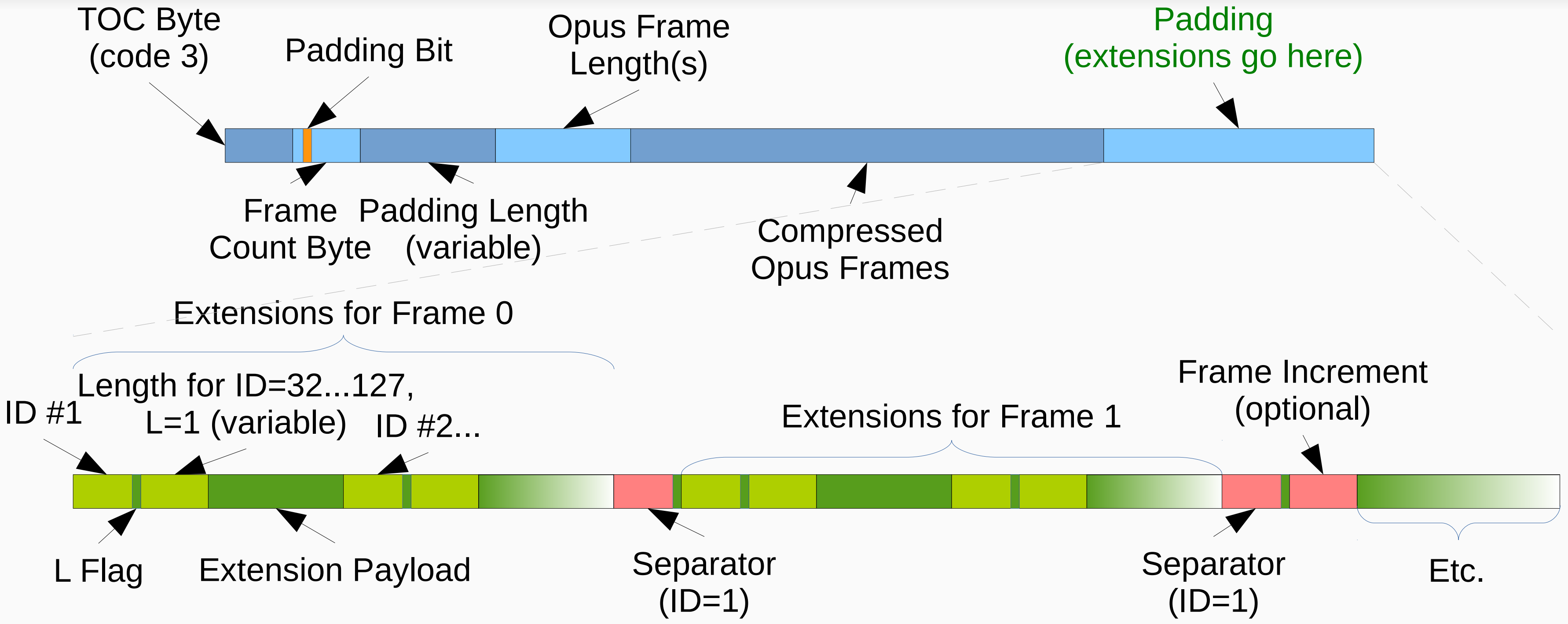
Code Complexity

```
if (iter->repeat_frame > 0) {
   /* We are in the process of repeating some extensions. */
   for (;iter->repeat_frame < iter->nb_frames; iter->repeat_frame++) {
      while (iter->src_len > 0) {
         const unsigned char *curr_data0;
         int repeat_id_byte;
         repeat_id_byte = *iter->src_data;
         iter->src_len = skip_extension(&iter->src_data, iter->src_len,
          &header_size);
         /* We skipped this extension earlier, so it should not fail now. */
         celt_assert(iter->src_len >= 0);
         /* Don't repeat padding. */
         if (repeat_id_byte <= 1) continue;
         /* If the "Repeat These Extensions" extension had L == 0 and this
             is the last repeated long extension, then force decoding the
             payload with L = 0. */
         if (iter->repeat_l == 0
          && iter->repeat_frame + 1 >= iter->nb_frames
          && iter->src_data == iter->last_long) {
            repeat_id_byte &= ~1;
         }
         curr_data0 = iter->curr_data;
         iter->curr_len = skip_extension_payload(&iter->curr_data,
          iter->curr_len, &header_size, repeat_id_byte,
          iter->trailing_short_len);
         if (iter->curr_len < 0) {
            return OPUS_INVALID_PACKET;
         }
         celt_assert(iter->curr_data - iter->data
          == iter->len - iter->curr_len);
         /* If we were asked to stop at frame_max, skip extensions for later
             frames. */
         if (iter->repeat_frame >= iter->frame_max) {
            continue;
         }
          if (ext != NULL) {
            ext->id = repeat_id_byte >> 1;
            ext->frame = iter->repeat_frame;
            ext->data = curr_data0 + header_size;
            ext->len = iter->curr_data - curr_data0 - header_size;
         }
         return 1;
      }
      /* We finished repeating the extensions for this frame. */
      iter->src_data = iter->repeat_data;
      iter->src_len = iter->repeat_len;
    }
   /* We finished repeating extensions. */
   iter->repeat_data_end = iter->repeat_data = iter->curr_data;
   /* If L == 0, advance the frame number to handle the case where we did
       not consume all of the data with an L == 0 long extension. */
   if (iter->repeat_l == 0) {
      iter->curr_frame++;
      /* Ignore additional padding if this was already the last frame. */
      if (iter->curr_frame >= iter->nb_frames) {
         iter->curr_len = 0;
      }
   }
   iter->repeat_frame = 0;
}
```

Questions?

- How do we get more reviews and feedback?

Opus Extension Format


TOC Byte (code 3)
Padding Bit
Opus Frame Length(s)
Padding (extensions go here)
Frame Count Byte
Padding Length (variable)
Compressed Opus Frames
Extensions for Frame 0
Length for ID=32...127, L=1 (variable)
ID #1
ID #2...
Extensions for Frame 1
Frame Increment (optional)
L Flag
Extension Payload
Separator (ID=1)
Separator (ID=1)
Etc.