Opus 音频编码格式

2020年3月15日

音频编码是将原始的音频采样数据，通过某种算法将其压缩编码成规定格式的二进制码流，从而方便存储和传输。Opus 就是由 Xiph.Org 基金会发明的一种音频编码格式。

Opus 编码简介

音频信号处理中将音频（Audio）分为两大类：语音（Speech）和音乐（Music）。语音一般由人声带发出，人说话时语音的频率一般在 300 ~ 3400 Hz 之间，频率比较低。而音乐包含各种乐器演奏的声音，频率范围更广，涵盖了人耳能够听到的 20 ~ 20 kHz。由于两类音频频率范围各有特点，因此一般会采用不同的技术来处理。

Opus 编码格式应用了两种技术：一个是**线性预测**（Linear Prediction，LP），另一个是**改进的离散余弦变换**（Modified Discrete Cosine Transform，MDCT）。线性预测技术在低频信号的编码上更加高效，适合处理语音数据。对于包含高频信号的音乐，改进的离散余弦变换这种域变换技术处理效率更高。

Opus 编码格式采用的技术不是全新的，它使用的线性预测技术来自于 Skype 发明的 SILK 编解码器，而改进的离散余弦变换技术来自于 CELT （Constrained-Energy Lapped Transform）。CELT 也是由 Xiph.Org 基金会早期发明一种音频编码格式，现在合并入 Opus 项目后，就不再有独立的 CELT 格式了。

为了对不同频率的音频应用不同的编码技术，Opus 音频频率带宽做了如下划分和命名：

缩写（全称）	音频带宽	应用采样率
NB (narrowband)	4 kHz	8 kHz
MB (medium-band)	6 kHz	12 kHz
WB (wideband)	8 kHz	16 kHz
SWB (super-wideband)	12 kHz	24 kHz
FB* (fullband)	20 kHz	48 kHz

> 根据奈奎斯特采样定理，应用 48 kHz 的采样率，实际上可以处理 24 kHz 以内的音频信号，但是即便如此，Opus 也不会处理超过 20 kHz 的音频，因为超过 20 kHz 的音频人耳已经很难听到了。

Opus 编码器在处理音频的时候，会将音频划分成多个**帧**（Frame）之后，针对每帧来处理。Opus 支持的帧长有：2.5ms、5ms、10ms、20ms、40ms、60ms。

Opus 工作在 SILK 模式时，支持 NB、MB、WB 频率带宽的音频，并且帧长在 10ms ~ 60ms 之间。工作在 CELT 模式时，支持 NB、WB、SWB、FB 音频带宽，并且帧长在 2.5ms ~ 20ms 之间。Opus 还可以工作在混合模式（Hybrid），也就是 SILK 和 CELT 同时起作用，这种情况下只支持 SWB 、FB 音频带宽，并且帧长为 10ms 或 20ms。

Opus 包结构

Opus 编码器处理原始数据输出一串包（Packet），一个包里面可能包含多个编码后的音频帧数据，只是这些音频帧的参数必须是一致的，例如：编码模式、音频带宽、帧大小以及声道数。

下面详细描述 Opus 包的结构。

TOC 字节

每个 Opus 包以一个 TOC （Table of Contents）字节开头，其结构如下：

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
| config  |s| c |
+-+-+-+-+-+-+-+-+
```

这个字节由三部分组成：配置数（config），立体声标志（s），帧数（c）。

前 5 位的配置数定义了 32 种编码配置，不同的编码模式、音频带宽和帧长度组成了这 32 种配置，如下表所示：

配置数（config）	编码模式	音频带宽	帧长度
0...3	SILK-only	NB	10, 20, 40, 60 ms
4...7	SILK-only	MB	10, 20, 40, 60 ms
8...11	SILK-only	WB	10, 20, 40, 60 ms
12...13	Hybrid	SWB	10, 20 ms
14...15	Hybrid	FB	10, 20 ms
16...19	CELT-only	NB	2.5, 5, 10, 20 ms
20...23	CELT-only	WB	2.5, 5, 10, 20 ms
24...27	CELT-only	SWB	2.5, 5, 10, 20 ms
28...31	CELT-only	FB	2.5, 5, 10, 20 ms

立体声标志位（s）取值 0 表示单声道，1 表示多声道立体声。

TOC 中最后两位（c）表示：

- 0：一个包中只有一帧音频。
- 1：一个包中有两帧音频，并且大小相同。
- 2：一个包中有两帧音频，但是大小不同。
- 3：一个包中有任意帧音频。

不同帧结构的包

根据一个包的 TOC 字节中帧数（c）的不同取值，我们把这个包命名为：**c 号包**。下面我们介绍这 4 种不同帧结构的包。

> **帧长度编码**
> 当一个包含有多个 VBR 的音频帧时，那么除了最后一个音频帧，前面几个帧都需要对帧的长度进行编码。存储帧长度的编码占用 1 ~ 2 个字节，其规则如下：
> - 第一个字节取值为 0：表示没有任何帧数据（这通常是非连续传输（DTX）或者音频包丢失）
> - 第一个字节取值为 1 ~ 251：表示第一帧的字节数
> - 第一个字节取值为 252 ~ 255：第二个字节也参与编码帧长度，第一帧的总字节数为：`(第二字节*4)+第一字节`
>
> 因此一个帧的最大长度为：255 * 4 + 255 = **1275** 字节。对于一个 20ms 的帧来说，这个长度代表 510 kbit/s 的码率。这个码率几乎是立体声音乐的有损压缩编码最高有效码率。超过这个码率，最好采用无损编码。这也是 MDCT 算法的最高有效码率，超过这个值，在增加码率进行编码，音频的质量也不会跟着提高。

0 号包: 一个包只包含一帧音频

其包结构如下，TOC 字节之后，紧跟着一帧音频的数据。

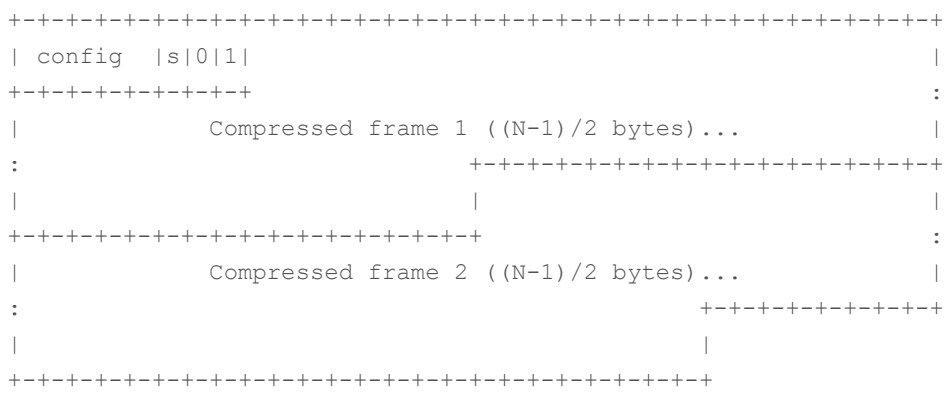
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| config  |s|0|0|                                                 |
+-+-+-+-+-+-+-+-+                                                 |
|                   Compressed frame 1 (N-1 bytes)...           :
:                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

后面这帧音频有可能是 0 个字节，这也是合法的 0 号包，这样的话这个包就只有一个 TOC 字节。

1 号包: 一个包里面含有两个大小相同的帧

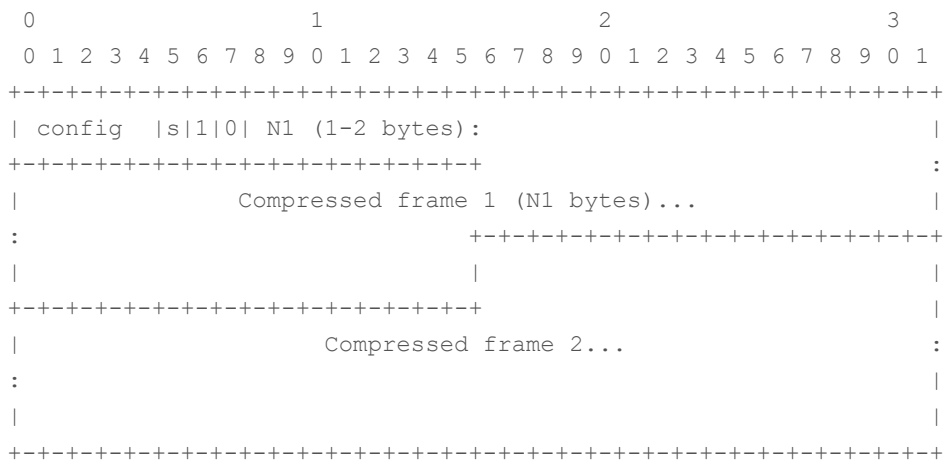
TOC 字节后，紧跟着两个帧的数据，两个帧的大小各占这个包剩下字节数的一半。由此可以看出，1 号包的大小必定为奇数。

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```



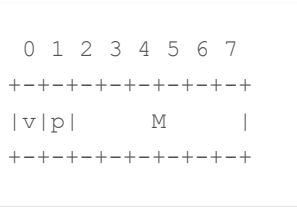
2 号包: 一个包里面含有两个大小不同的帧

这种情况下，因为一个音频包里面包含了两个大小不同的音频帧，因此需要对第一个帧的字节数编码，否则无法区分开两个帧。TOC 字节后面的 1 ~ 2 个字节为第一个帧的字节数，其规则如前面所述的**帧长度编码**。



3 号包: 一个包里面含有任意个帧

这种类型的包在 TOC 字节之后，有一个字节编码这个包里面的帧数量，这个字节的结构如下图所示。



帧数量字节包含三部分信息：

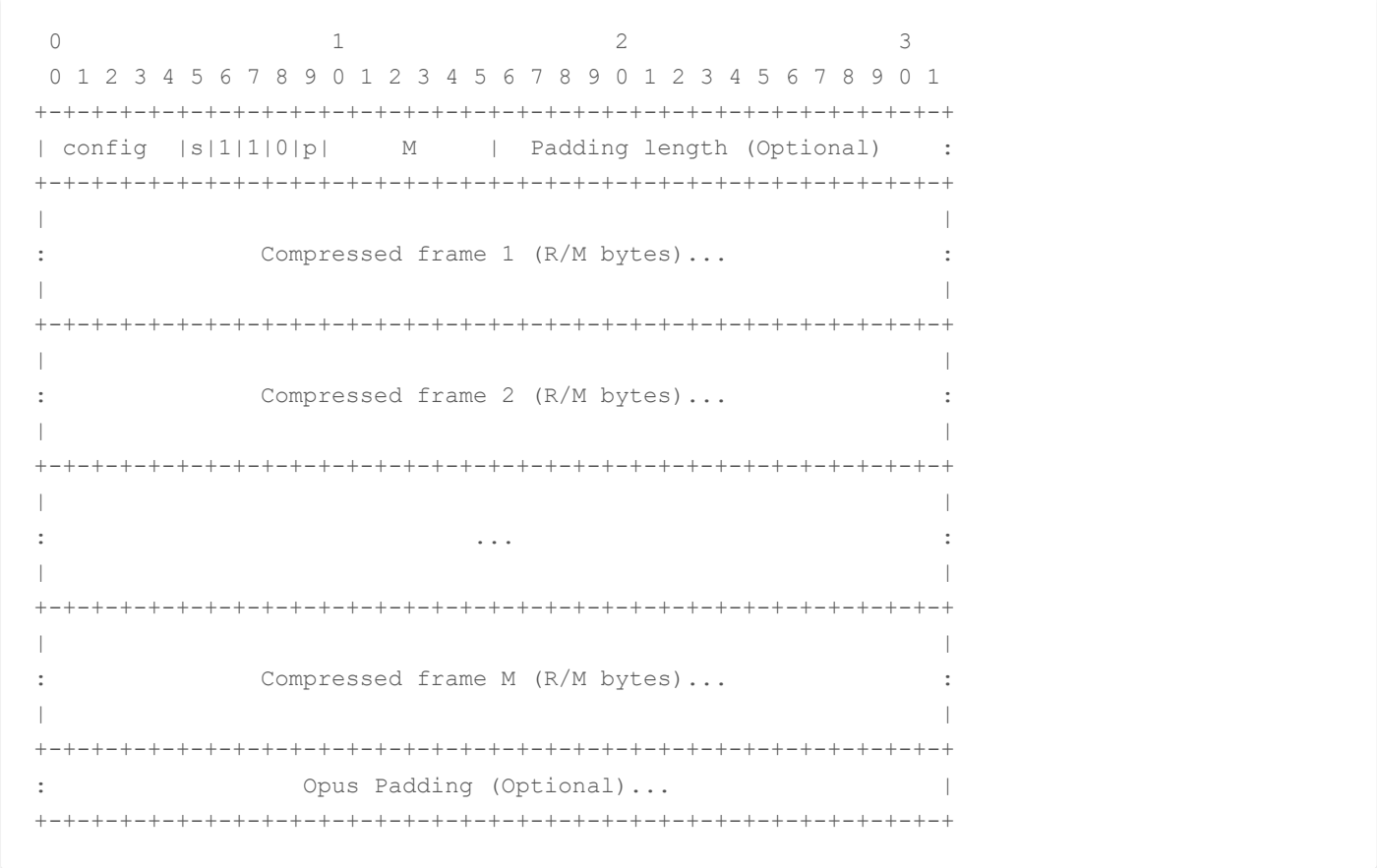
- v 等于 0 表示 CBR，等于 1 表示 VBR。
- p 等于 1 表示包里面含有填充字节。
- M 表示包里面含有的帧个数。

规定一个包所包含的音频长度不能超过 120ms，如果按最小的帧长 2.5ms 计算，一个包的音频包所含有的音频帧不会超过 48 个。

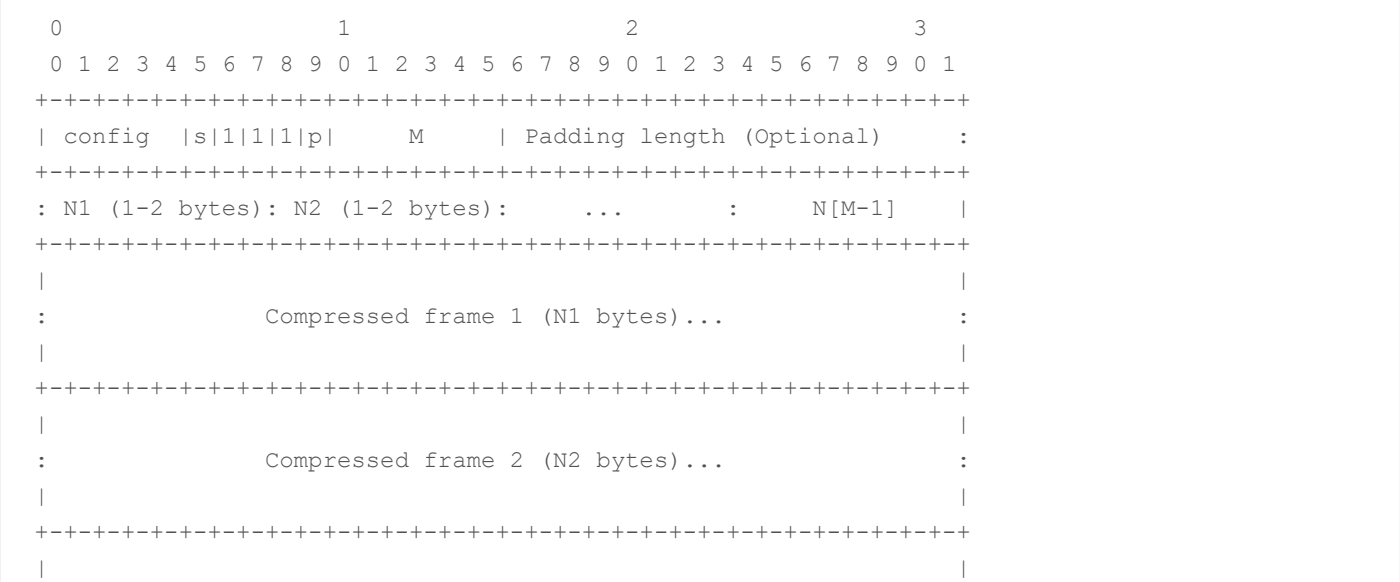
当音频包里面含有填充字节时，帧数量字节后面的字节用于编码填充字节的长度。如果帧数量字节后面的那个字节值为 N, 且 N 大于 0 小于等于 254，则表示包后面的填充字节数为 N 个字节。如果帧数量字节后面的那个字节值为 255，那么表示包后面的填充了254个字节，并且这个字节后面的一个字节编码了更多的填充字节数。以此类推，可以编码任意长度的填充字节数。

假如 P 表示填充字节的总数（包含编码填充字节长度的字节数），N 表示整个音频包的字节数。

那么对于 CBR 编码，R = N-2-P 就是有效音频数据的字节数。包里面每个音频帧的字节数量为 R/M，如下图所示。



对于 VBR 编码的情况，填充长度字节的后面跟上了 M - 1 个帧长度的编码，每个帧长度都会用一到两个字节做如前面所述的**帧长度编码**。其包结构如下所示：



```
:                               ...                               :
|                                                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                 |
:                     Compressed frame M...                       :
|                                                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                   Opus Padding (Optional)...                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

从上面 0 ~ 3 号包的结构可以看到，虽然有帧长度编码信息区分出包内的每个帧，但是包没有编码自己的总长度信息，因此如果编码器输出的多个包直接拼接存储在一起，那么没有办法区分出每个包的边界，编码器也就没有办法解码这种裸的编码流。为了让解码器能够处理 Opus 裸流，需要将其封装在一种容器格式中（例如 Ogg 格式），让容器格式提供分包的信息。

合法音频包的验证条件

根据前面描述，我们可以通过以下几个约束条件来判断一个 Opus 包是否合法：

1. 音频包至少包含一个字节，即 TOC 字节。
2. 一个音频帧的长度不能超过 1275 字节。
3. 1 号音频包的字节数必须是奇数，使得 `(N-1)/2` 计算出来是整数。
4. 2 号音频包的 TOC 字节后面必须有足够的字节数用于编码帧长度，而且帧长度不能大于音频包剩下的字节数。
5. 3 号音频包至少包含一个音频帧，但是总的音频长度不得超过 120ms。
6. CBR 的 3 号包至少包含 2 个字节。添加在音频包后面的填充字节数和表示填充长度的字节数之和 P 不能大于 N-2，而且 (N-2-P) 是帧个数 M 的倍数。
7. VBR 的 3 号包必须足够大到容纳所有的包头字节，以及对应的前 M-1 个帧的长度，和填充字节数。

带分界的音频包

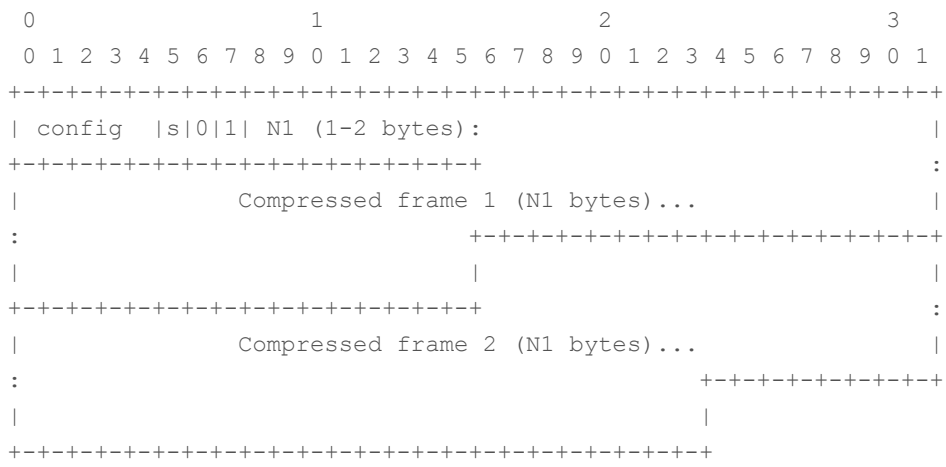
如前所述，Opus 包与包之间没有界限，需要额外的机制告诉解码器每个包的大小。不过除此之外，Opus 标准还定义了一种**带分界**（Self-Delimiting）的音频包，编码器拿到这种包可以直接推断出包的大小。

从前面描述的 0 ~ 3 号包结构来看，对于包含 CBR 的包，由于每个帧长度一样，只需要再添加一个帧的长度编码就可以确定这个包的总大小；对于 VBR 的包，由于包里面已经含有了除最后一个帧的长度编码，只需要再添加最后一帧的长度编码就可以算出包的总大小。不管哪种情况，这个长度编码都采用前面提到的 1 ~ 2 字节的**帧长度编码**。下面分别描述 0 ~ 3 号包是如何添加这个长度编码的。

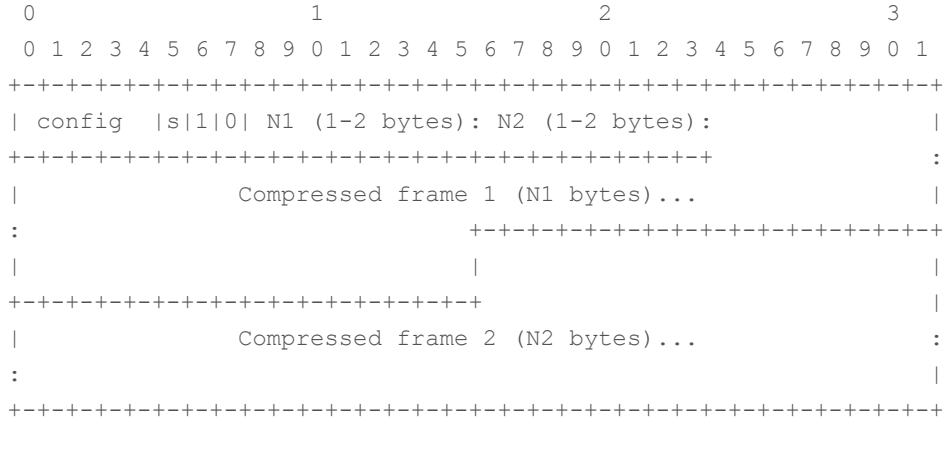
对于 0 号包，在 TOC 字节后面，添加帧长度编码，表示后面这个帧的大小。

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| config  |s|0|0| N1 (1-2 bytes):                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                                 |
|               Compressed frame 1 (N1 bytes)...                  :
```

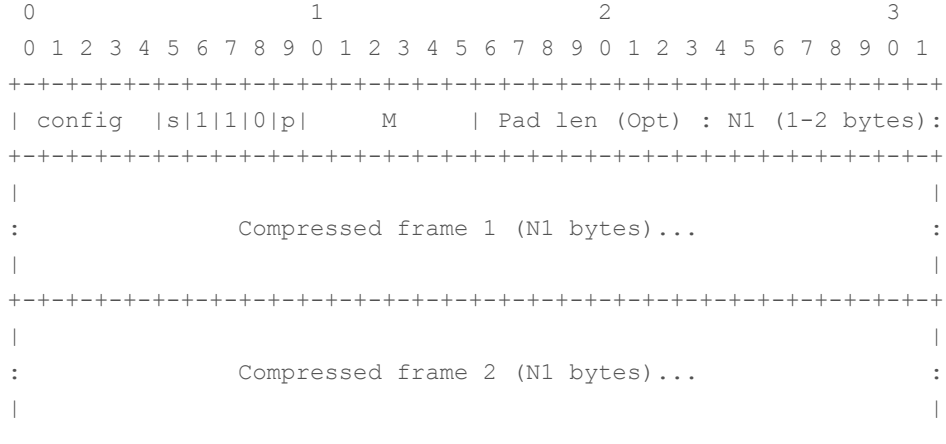
对于 1 号包，在 TOC 字节后面，添加帧长度编码，表示后面每个帧的大小。如果这个编码的值为 N1，那么后面两个帧的总大小为 2 * N1。

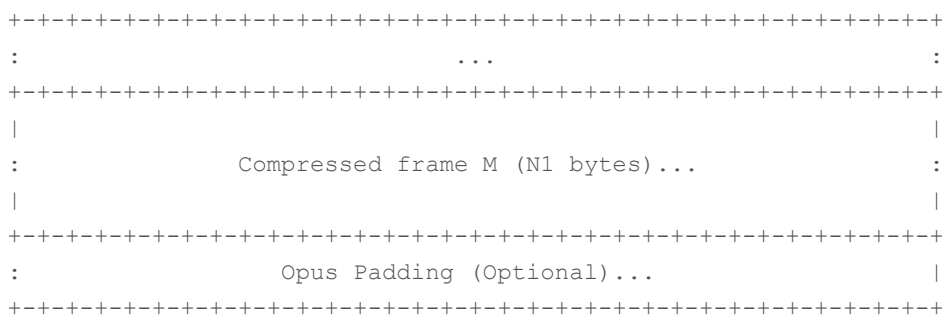


对于 2 号包，在第一帧数据前面，添加第二帧的长度编码，表示第二帧的大小。如果这个编码的值为 N2，那么两个帧的总大小为 N1 + N2。

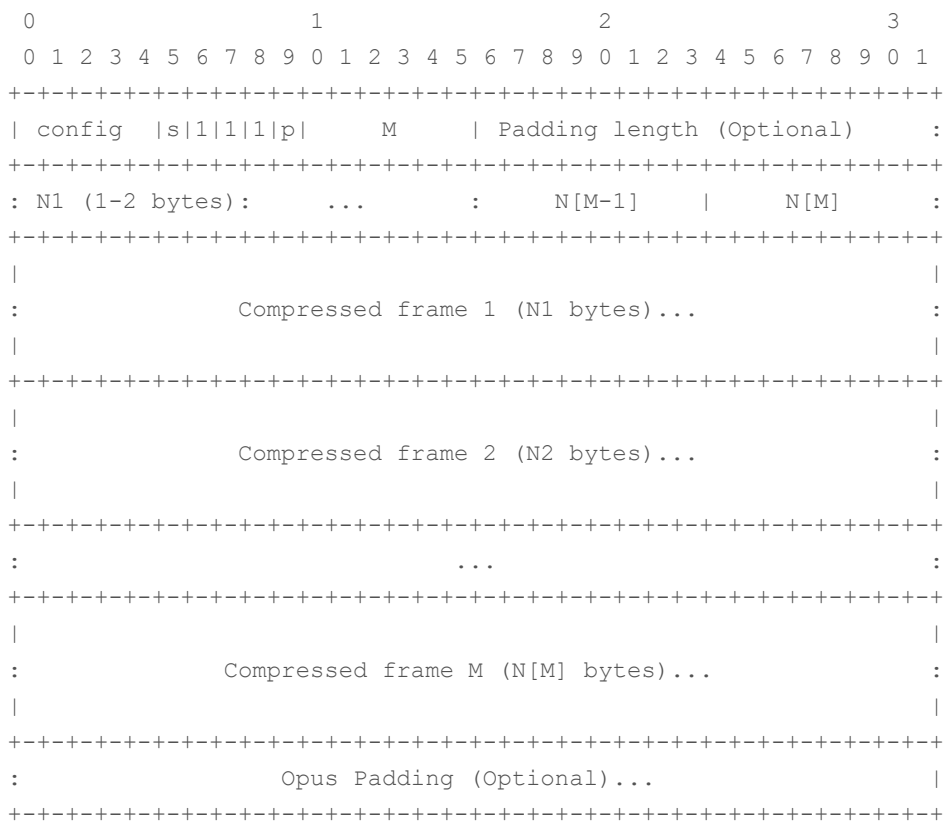


对于 CBR 的 3 号包，在第一帧前面，添加帧长度编码，表示后面每个帧的大小。如果这个编码的值为 N1，那么后面 M 个帧的总大小为 M * N1。





对于 VBR 的 3 号包，在第一帧前面，添加帧长度编码，表示最后一个帧的大小。如果这个编码的值为 N[M]，那么后面 M 个帧的总大小为 N1 + N2 + ... + N[M]。



从上面的描述我们可以看到，实际上这个额外的帧长度编码都是在原来的包结构基础上，插入到了第一帧数据的前面。

在一些场景下，带分界包有实际的用途。根据标准，Opus 只能编码单声道或者双声道立体声音频。因此一个 Opus 码流要不就是单声道的，要不就是双声道的。如果要传输或者存储多声道（大于2）的音频，就需要复合多个 Opus 流。例如 5.1 环绕立体声有 6 个声道，传输或者存储这样一个音频，可能需要组合多个单声道或双声道的 Opus 流。

假如 5.1 环绕立体声由 2 个双声道，2 个单声道的 Opus 流组成。那么需要将这 4 个 Opus 流复合成一个流，每次从 4 个 Opus 流中各取一个包，组成一个复合音频包。每个复合音频包就包含四个包，分别来自同一时刻不同的 Opus 流。为了让解码器能够顺利从复合音频包中识别出四个包，就需要让前三个包采用带分界格式的编码。一般来说复合包的总大小会通过容器格式或者传输协议信息告诉解码器，因此复合包里面最后一个子 Opus 包不需要采用带分界包格式，可以直接推断出来。

参考文献：

[1] Voice frequency: https://en.wikipedia.org/wiki/Voice_frequency
[2] CELT: https://en.wikipedia.org/wiki/CELT
[3] Definition of the Opus Audio Codec: https://tools.ietf.org/html/rfc6716
[4] 5.1 surround sound: https://en.wikipedia.org/wiki/5.1_surround_sound

上一篇 下一篇