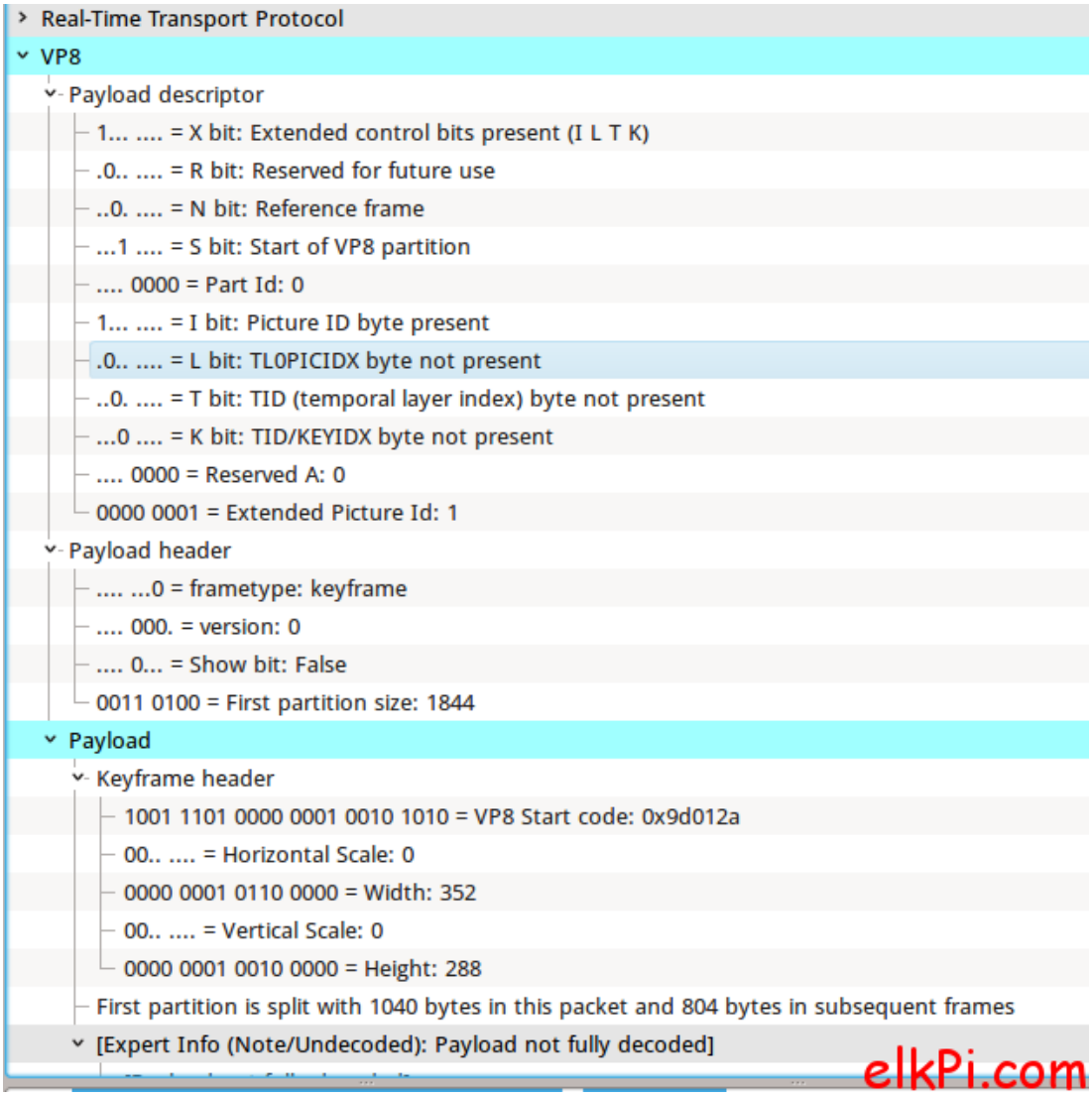
VP8 RTP负载格式

2014年12月13日 米鹿π 0 Comment 流处理

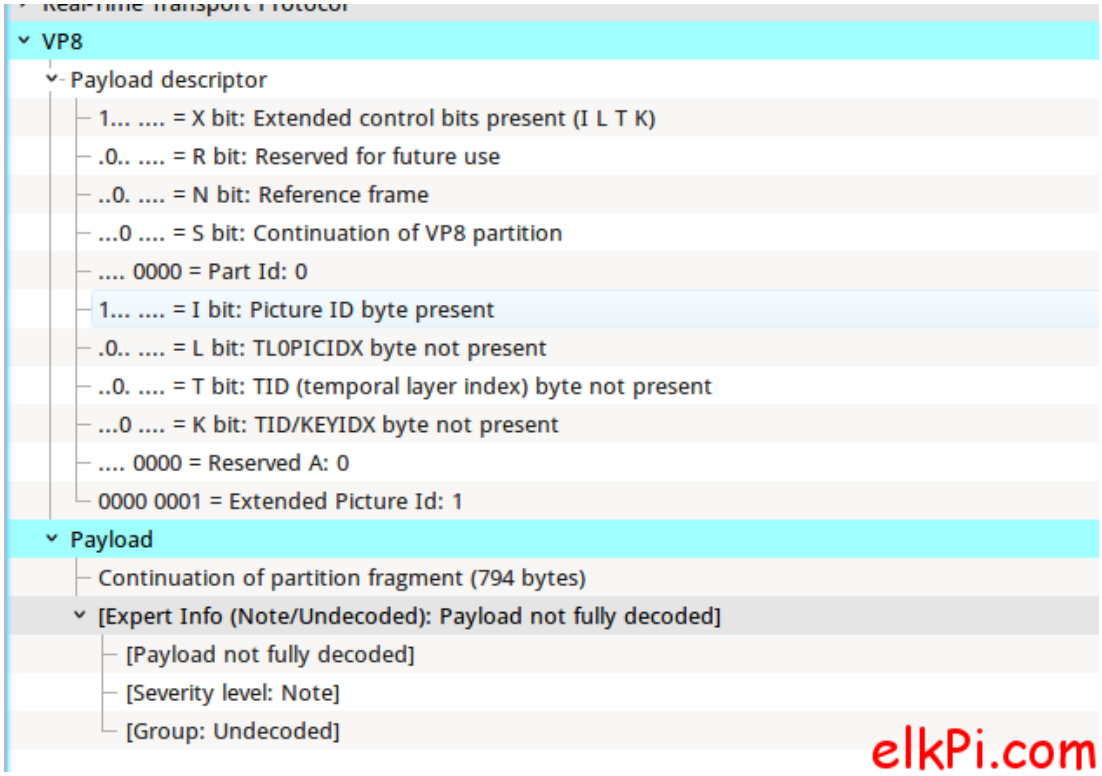
这几周挺忙的，都在搞webrtc的一些东西，webrtc用到的编码是VP8的。所以对VP8有了一些初步的了解，但是对其如何编解码还是不够深入了解，只是知道如果去解析RTP及构造RTP等一些粗浅的操作。这里顺便给记录一下。

首先，是需要有理论的知识，RTP Payload Format for VP8 Video draft-ietf-payload-vp8-05，这份ietf的文档，先看一些，这里有介绍关于VP8的rtp payload实现方式。其实这份文档分为好几个版本，05版本的并不是最新的，最新的是13版本的。为什么用05版本的，因为我自己之前有下载了ffmpeg 2.2.4的代码，代码中关于vp8 rtp的实现是基于05版本的，所以就看了这份了。谷歌对于这份说明更新还是很勤快的，现在还只是草案。不过开发下来，觉得VP8还是没H264的RTP负载实现来的好吧，虽然就只分为keyframes及interframes。

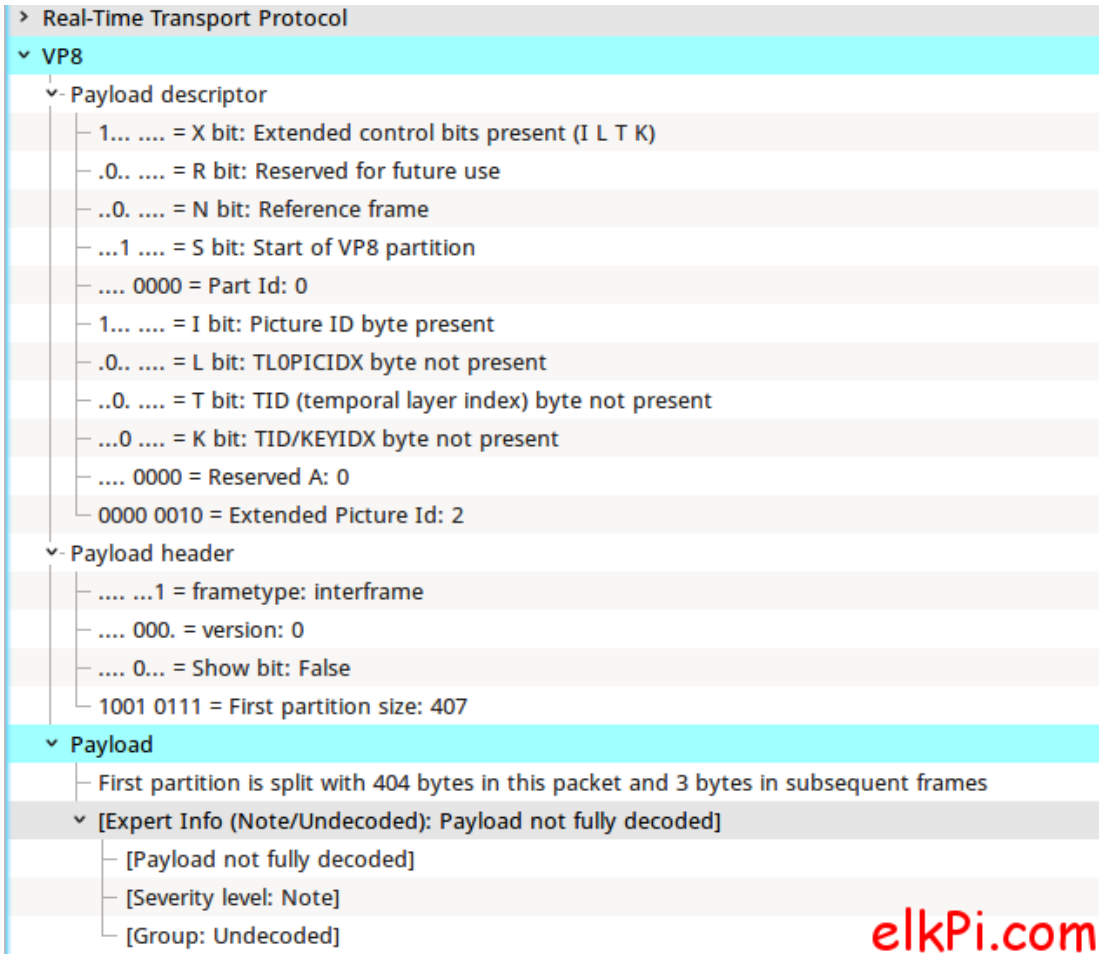
看完文档，对于RTP这种流处理，肯定少不了一些抓包工具，这里要说的工具就是wireshark了。而且是12及其之后的版本。为何使用12的版本，因为只有12之后的版本，才有对H264及VP8做一些解析，不过VP8的解析还是相对简单，但是绝对是够用。先看几张图：



上图中，是keyframe帧，在新版的wireshark可以使用vp8.hdr.frametype==keyframe或vp8.hdr.frametype==0来过滤。从keyframe header的解析上，可以看出视频流为cif分辨率。



接下来的这帧是continuation frame，这时候，除了VP8 payload descriptor就是payload了。还有就是interframe帧。如下：



interframe的话，比普通的continuation多了payload header。以上就是新版本wireshark提供给我们的一些信息了。通过这些信息可以初步的了解下

vp8 rtp payload的一些实现方式。但是具体的话，还是要通过RFC文档及阅读ffmpeg代码来深入了解了。

VP8 rtp必带的就是payload descriptor。RFC文档中给出的格式如下：

```
      0 1 2 3 4 5 6 7
     +-+-+-+-+-+-+-+-+
     |X|R|N|S|PartID | (REQUIRED)
     +-+-+-+-+-+-+-+-+
X:   |I|L|T|K| RSV   | (OPTIONAL)
     +-+-+-+-+-+-+-+-+
I:   |M| PictureID   | (OPTIONAL)
     +-+-+-+-+-+-+-+-+
L:   |   TL0PICIDX   | (OPTIONAL)
     +-+-+-+-+-+-+-+-+
T/K: |TID|Y| KEYIDX  | (OPTIONAL)
     +-+-+-+-+-+-+-+-+
```

前8位：

X：extended位，当该位为1时，后面这些 OPTIONAL（I,L,T,K）就需要进行解析了，如果为0的话，则直接忽略这些可选的项目。

R：reserved位。

N：Non-reference帧，当为1时，说明该帧可以被丢弃，就我目前抓到的报文，似乎还没有遇到被置为1的可丢弃的帧，大部分情况下，应该都是0的。当不知道该帧是否为参考时，必须被设置为0.

S：Start of VP8 partition，如果当前的帧为VP8 partition的起始，则该参数必须被置1,由于keyframe及interframe都是每个partition的起始，所以keyframe及interframe的话，S位肯定是1的，而continuation frame则不一定了。同一时间戳上的图片可能被分成多个的partition，这时候continuation frame中也就会出现S位为1的了。

PartID：partition index，如果S位为1,那么partid肯定是为1了。由于partition不可能会太大，所以这里只用了4位来表示，完全是够用的。

之后的ILTK都是需要X置1才有效。

I：picture id呈现标志位，置1时，必须在后面I所示行呈现picture id，一般从1开始依据图像顺序递增。目前大部分的软件都会把I置1.

L：TL0PICIDX呈现标志位，置1时，必须在后面L所示行呈现TL0PICIDX，目前

看来ffmpeg并没有使用该位。当T被置1时，L必须被置0!

T：TID呈现标志位.被置1时，可选的TID/KEYIDX部分必须被呈现。TID | Y部分必须在其之后。如果K被置1但T为0,TID/KEYIDX必须呈现出来，但是TID | Y必须被忽略。T或K都不为1时，TID/KEYIDX都不必呈现！*略为有点绕，不过好在ffmpeg直接把这几位都置0了。。。*

K：KEYIDX present，这个其实和T说明的差不多了。

RSV：预留位，必须全为0!

现在大部分的软件实现都会置将I置1,其余都置0,所以这里在之后最重要的就是解析picture id了。

PictureID：8位或16位的长度，其中首位为为1时，则为16位的长度，后15位为picture id，为0,则为8位的长度，后7位为picture id。

TL0PICIDX：8 bits temporal level zero index

TID：2 bits temporal layer index.

Y：1 layer sync bit.

KEYIDX：5 bits temporal key frame index.

从ffmpeg的代码上看，TL0PICIDX，TID，Y，KEYIDX都被忽略了，不过其实大部分编码也都不适用这几位。

以上就是VP8 payload descriptor的内容了，ffmpeg在函数 vp8_handle_packet (rtpdec_vp8.c)中有对这部分的解析代码。

```
int extended_bits, part_id,  start_partition;

extended_bits   = buf[0] & 0x80; // buf为RTP payload的起始指针
start_partition = buf[0] & 0x10;
part_id         = buf[0] & 0x0f;
```

代码上看，也是很是随意，哈哈，在取picture id的代码如下：

```
if (pictureid_present) {
        if (len < 1)
            return AVERROR_INVALIDDATA;
        if (buf[0] & 0x80) {
        if (len < 2)
```

```
            return AVERROR_INVALIDDATA;
        pictureid = AV_RB16(buf) & 0x7fff; // AV_RB16可以是htons，大小端
转换
        pictureid_mask = 0x7fff;
        buf += 2;
        len -= 2;
    } else {
        pictureid = buf[0] & 0x7f;
        pictureid_mask = 0x7f;
        buf++;
        len--;
    }
}
```

在descriptor之后，如果S位为1,则说明是起始的部分。这时候，还需要进一步的解析payload header，也就是VP8的头了，这部分长度为3字节。在libvpx中的结构体如下：

```
/* 24 bits total */
typedef struct
{
    unsigned int type: 1;
    unsigned int version: 3;
    unsigned int show_frame: 1;

    /* Allow 2^20 bytes = 8 megabits for first partition */

    unsigned int first_partition_length_in_bytes: 19;

#ifdef PACKET_TESTING
    unsigned int frame_number;
    unsigned int update_gold: 1;
    unsigned int uses_gold: 1;
    unsigned int update_last: 1;
    unsigned int uses_last: 1;
#endif

} VP8_HEADER;

#ifdef PACKET_TESTING
#define VP8_HEADER_SIZE 8
#else
#define VP8_HEADER_SIZE 3
#endif
```

默认是不开启testing的，所以长度为24位，3字节。

```
      0 1 2 3 4 5 6 7
     +-+-+-+-+-+-+-+-+
     |Size0|H| VER |P|
     +-+-+-+-+-+-+-+-+
     |     Size1     |
     +-+-+-+-+-+-+-+-+
     |     Size2     |
     +-+-+-+-+-+-+-+-+
     | Bytes 4..N of |
```

```
      | VP8 payload   |
      :               :
      +-+-+-+-+-+-+-+-+
      | OPTIONAL RTP  |
      | padding       |
      :               :
      +-+-+-+-+-+-+-+-+
```

P：类型，VP8只有两种，keyframe及interframe，分别为0和1.RFC6386中有定义。

VER：版本，内容如下：

```
            +---------+-------------------------+-------------+
            | Version | Reconstruction Filter   | Loop Filter |
            +---------+-------------------------+-------------+
            | 0       | Bicubic                 | Normal      |
            |         |                         |             |
            | 1       | Bilinear                | Simple      |
            |         |                         |             |
            | 2       | Bilinear                | None        |
            |         |                         |             |
            | 3       | None                    | None        |
            |         |                         |             |
            | Other   | Reserved for future use |             |
            +---------+-------------------------+-------------+
```

H：显示位。0的时候，不显示，其实我觉得很奇怪，因为我这边抓到的报文，该位都是0,总不能都不显示吧，显然又是被解码器忽略了！

Size：首个partition的长度，19位，很奇怪的设定，计算方式是这样的
1stPartitionSize =
Size0 + 8 * Size1 + 2048 * Size2

在之后，我构造VP8的报文的时候，一直很纠结于这个size的大小，因为我有修改了里面的一些长度，虽然失败了（对VP8编解码还是不够熟悉）。但是其实计算并不复杂，在libvpx代码中实现如下：

```c
{
    int v = (oh.first_partition_length_in_bytes << 5) |
            (oh.show_frame << 4) |
            (oh.version << 1) |
            oh.type;

    dest[0] = v;
    dest[1] = v >> 8;
    dest[2] = v >> 16;
}
```

因为为24位，普通情况下int为32位，显然是足够存储的，所以通过移位及或之后，就可以得到所需要的值，在依次向右移位来赋值，就达到构造该payload header的目的了。

最后一个需要解析的是关于keyframe的了，只有是keyframe才带有keyframe header。这个头里面携带了图像的大小，以及起始的校验值0x9d012a。rfc6386更是把代码直接贴出来了：

头校验，不符合0x9d012a显然非VP8的keyframe了。

```
unsigned char *c = pbi->source+3;

   // vet via sync code
   if (c[0]!=0x9d||c[1]!=0x01||c[2]!=0x2a)
       return -1;
```

之后就是取图像的宽高了，以及一些分量信息。

```
pc->Width      = swap2(*(unsigned short*)(c+3))&0x3fff;
pc->horiz_scale = swap2(*(unsigned short*)(c+3))>>14;
pc->Height     = swap2(*(unsigned short*)(c+5))&0x3fff;
pc->vert_scale  = swap2(*(unsigned short*)(c+5))>>14;
```

这两个垂直及水平分量的内容如下：

```
            +-------+----------------------------------------+
            | Value | Scaling                                |
            +-------+----------------------------------------+
            | 0     | No upscaling (the most common case).   |
            |       |                                        |
            | 1     | Upscale by 5/4.                        |
            |       |                                        |
            | 2     | Upscale by 5/3.                        |
            |       |                                        |
            | 3     | Upscale by 2.                          |
            +-------+----------------------------------------+
```

这里还涉及了大小端的问题，目测开发这个libvpx的时候，使用x86的设备（现在大部分PC都小端）。默认直接小端传输了。。看下面这个实现，因为ppc是大端的，网络字节序也是大端，没想到大端的ppc反倒要做转换。

```
#if defined(__ppc__) || defined(__ppc64__)
# define swap2(d)  \
  ((d&0x000000ff)<<8) |  \
  ((d&0x0000ff00)>>8)
```

```
#else
  # define swap2(d) d
#endif
```

至此，VP8的RTP负载的内容大体就是这样了，之后的一些负载内容，涉及的是编解码还有图像信息，wireshark也没有给直接的解析出来，其实这里还有个keyframe的Color Space and Pixel Type，目前还没搞懂。有时间搞懂了，在补充吧！

转载请注明： 转载自elkPi

本文链接地址: VP8 RTP负载格式

RTP VP8