码龄11年 暂无认证
1221 原创 | 2万+ 周排名 | 3137 总排名 | 262万+ 访问 | 等级
3万+ 积分 | 624 粉丝 | 759 获赞 | 236 评论 | 4361 收藏
私信 关注



转载 phymat.nico 于 2017-12-14 00:03:13 发布 14822 收藏 32

写在前面：RTP的解析，网上找了很多资料，但是都不全，所以我力图整理出一个比较全面的解析，

其中借鉴了很多文章，我都列在了文章最后，在此表示感谢。

互联网的发展离不开大家的无私奉献，我决定从我做起，希望大家支持。

## 1、RTP Header解析

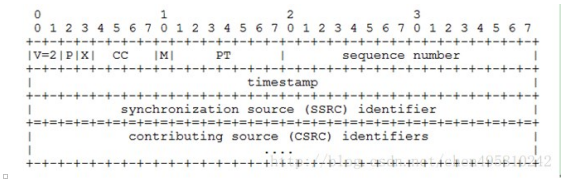



图1

1) V：RTP协议的版本号，占2位，当前协议版本号为2

2) P：填充标志，占1位，如果P=1，则在该报文的尾部填充一个或多个额外的八位组，它们不是有效载荷的一部分。

3) X：扩展标志，占1位，如果X=1，则在RTP报头后跟有一个扩展报头

4) CC：CSRC计数器，占4位，指示CSRC 标识符的个数

5) M: 标记，占1位，不同的有效载荷有不同的含义，对于视频，标记一帧的结束；对于音频，标记会话的开始。

6) PT: 有效荷载类型，占7位，用于说明RTP报文中有效载荷的类型，如GSM音频、JPEM图像等,在流媒体中大部分是用来区分音频流和视频流的，这样便于客户端进行解析。

7) 序列号：占16位，用于标识发送者所发送的RTP报文的序列号，每发送一个报文，序列号增1。这个字段当下层的承载协议用UDP的时候，网络状况不好的时候可以用来检查丢包。同时出现网络抖动的情况可以用来对数据进行重新排序，序列号的初始值是随机的，同时音频包和视频包的sequence是分别记数的。

8) 时戳(Timestamp)：占32位，必须使用90 kHz 时钟频率。时戳反映了该RTP报文的第一个八位组的采样时刻。接收者使用时戳来计算延迟和延迟抖动，并进行同步控制。

9) 同步信源(SSRC)标识符：占32位，用于标识同步信源。该标识符是随机选择的，参加同一视频会议的两个同步信源不能有相同的SSRC。

10) 特约信源(CSRC)标识符：每个CSRC标识符占32位，可以有0～15个。每个CSRC标识了包含在该RTP报文有效载荷中的所有特约信源。

注：基本的RTP说明并不定义任何头扩展本身，如果遇到X=1，需要特殊处理

取一段码流如下：

80 e0 00 1e 00 00 d2 f0 00 00 00 00 41 9b 6b 49 €?....??....A?kI

e1 0f 26 53 02 1a ff06 59 97 1d d2 2e 8c 50 01 ?.&S....Y?.?.?P.

cc 13 ec 52 77 4e e50e 7b fd 16 11 66 27 7c b4 ?.?RwN?.{?..f'|?

f6 e1 29 d5 d6 a4 ef3e 12 d8 fd 6c 97 51 e7 e9 ??)????>.??l?Q??

cfc7 5e c8 a9 51 f6 82 65 d6 48 5a 86 b0 e0 8c ??^??Q??e?HZ????
其中，
80 是V_P_X_CC
e0 是M_PT
00 1e 是SequenceNum
00 00 d2 f0 是Timestamp
00 00 00 00是SSRC
把前两字节换成二进制如下
1000 0000 1110 0000
按顺序解释如下：
10 是V；
0 是P；
0 是X；
0000 是CC；
1 是M；
110 0000 是PT；

排版不如word看的清晰，大家凑合着看吧。

原创不易，转载请附上链接，谢谢 http://blog.csdn.net/chen495810242/article/details/39207305

## 2、RTP荷载H264码流

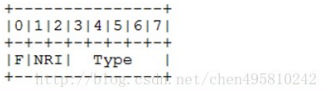



图2

荷载格式定义三个不同的基本荷载结构，接收者可以通过RTP荷载的第一个字节后5位（如图2）识别荷载结构。

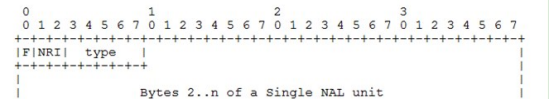
1）单个NAL单元包：荷载中只包含一个NAL单元。NAL头类型域等于原始 NAL单元类型,即在范围1到23之间

2）聚合包：本类型用于聚合多个NAL单元到单个RTP荷载中。本包有四种版本,单时间聚合包类型A (STAP-A)，单时间聚合包类型B (STAP-B)，多时间聚合包类型(MTAP)16位位移(MTAP16), 多时间聚合包类型(MTAP)24位位移(MTAP24)。赋予STAP-A, STAP-B, MTAP16, MTAP24的NAL单元类型号分别是 24,25, 26, 27

3）分片单元：用于分片单个NAL单元到多个RTP包。现存两个版本FU-A，FU-B,用NAL单元类型 28，29标识

常用的打包时的分包规则是：如果小于MTU采用单个NAL单元包，如果大于MTU就采用FUs分片方式。
因为常用的打包方式就是单个NAL包和FU-A方式，所以我们只解析这两种。

### 2.1、单个NAL单元包



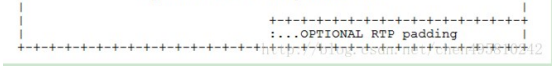



图3

定义在此的NAL单元包必须只包含一个。这意味聚合包和分片单元不可以用在单个NAL 单元包中。并且RTP序号必须符合NAL单元的解码顺序。NAL单元的第一字节和RTP荷载头第一个字节重合。如图3。

打包H264码流时，只需在帧前面加上12字节的RTP头即可。

## 2.2、分片单元（FU-A）

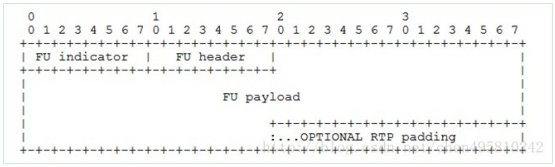



图4

分片只定义于单个NAL单元不用于任何聚合包。NAL单元的一个分片由整数个连续NAL单元字节组成。每个NAL单元字节必须正好是该NAL单元一个分片的一部分。相同NAL单元的分片必须使用递增的RTP序号连续顺序发送(第一和最后分片之间没有其他的RTP包）。相似，NAL单元必须按照RTP顺序号的顺序装配。

当一个NAL单元被分片运送在分片单元(FUs)中时，被引用为分片NAL单元。STAPs,MTAPs不可以被分片。 FUs不可以嵌套。 即, 一个FU 不可以包含另一个FU。运送FU的RTP时戳被设置成分片NAL单元的NALU时刻。

图 4 表示FU-A的RTP荷载格式。FU-A由1字节的分片单元指示（如图5），1字节的分片单元头（如图6），和分片单元荷载组成。


图 5　　图 6

S: 1 bit 当设置成1,开始位指示分片NAL单元的开始。当跟随的FU荷载不是分片NAL单元荷载的开始，开始位设为0。

E: 1 bit 当设置成1, 结束位指示分片NAL单元的结束，即, 荷载的最后字节也是分片NAL单元的最后一个字节。当跟随的 FU荷载不是分片NAL单元的最后分片,结束位设置为0。

R: 1 bit 保留位必须设置为0，接收者必须忽略该位

打包时，原始的NAL头的前三位为FU indicator的前三位，原始的NAL头的后五位为FU header的后五位。

取一段码流分析如下：
80 60 01 0f 00 0e 10 00 00 0000 00 7c 85 88 82 €`.....|???

00 0a 7f ca 94 05 3b7f 3e 7f fe 14 2b 27 26 f8 ...??.;.>.?.+'&?

89 88 dd 85 62 e1 6dfc 33 01 38 1a 10 35 f2 14 ????b?m?3.8..5?.

84 6e 21 24 8f 72 62f0 51 7e 10 5f 0d 42 71 12 ?n!$?rb?Q~._.Bq.

17 65 62 a1 f1 44 dc df 4b 4a 38 aa 96 b7 dd 24 .eb??D??KJ8????$前12字节是RTP Header
7c是FU indicator
85是FU Header
FU indicator（0x7C）和FU Header（0x85）换成二进制如下
0111 1100 1000 0101
按顺序解析如下：

| | |
|---|---|
| 0 | 是F |
| 11 | 是NRI |
| 11100 | 是FU Type，这里是28，即FU-A |
| 1 | 是S，Start，说明是分片的第一包 |
| 0 | 是E，End，如果是分片的最后一包，设置为1，这里不是 |
| 0 | 是R，Remain，保留位，总是0 |
| 00101 | 是NAI Type，这里是5，说明是关键帧（不知道为什么是关键帧请自行谷歌） |

打包时，FUindicator的F、NRI是NAL Header中的F、NRI，Type是28；FU Header的S、E、R分别按照分片起始位置设置，Type是NAL Header中的Type。

解包时，取FU indicator的前三位和FU Header的后五位，即0110 0101（0x65）为NAL类型。

## 3、RTP荷载PS流

针对H264 做如下PS 封装：每个IDR NALU 前一般都会包含SPS、PPS 等NALU，因此将SPS、PPS、IDR 的NALU 封装为一个PS 包，包括ps 头，然后加上PS system header，PS system map，PES header+h264 raw data。所以一个IDR NALU PS 包由外到内顺序是：PSheader| PS system header | PS system Map | PES header | h264 raw data。对于其它非关键帧的PS 包，就简单多了，直接加上PS头和PES 头就可以了。顺序为：PS header | PES header | h264raw data。以上是对只有视频video 的情况，如果要把音频Audio也打包进PS 封装，也可以。当有音频数据时，将数据加上PES header 放到视频PES 后就可以了。顺序如下：PS 包=PS 头|PES(video)|PES(audio)，再用RTP 封装发送就可以了。

GB28181 对RTP 传输的数据负载类型有规定（参考GB28181 附录B），负载类型中96-127

RFC2250 建议96 表示PS 封装，建议97 为MPEG-4，建议98 为H264

即我们接收到的RTP 包首先需要判断负载类型，若负载类型为96，则采用PS 解复用，将音视频分开解码。若负载类型为98，直接按照H264 的解码类型解码。

注：此方法不一定准确，取决于打包格式是否标准

PS 包中的流类型（stream type）的取值如下：

1) MPEG-4 视频流： 0x10；

2) H.264 视频流： 0x1B；

3) SVAC 视频流： 0x80；

4) G.711 音频流： 0x90；

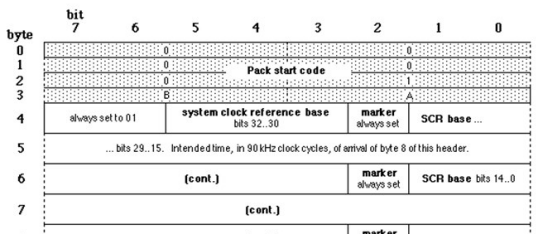
5) G.722.1 音频流： 0x92；

6) G.723.1 音频流： 0x93；

7) G.729 音频流： 0x99；

8) SVAC音频流： 0x9B。

## 3.1、PS包头



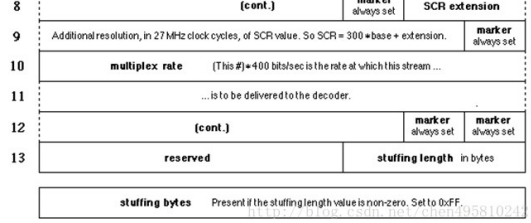



图7

1) Pack start code：包起始码字段，值为0x000001BA的位串，用来标志一个包的开始。

2) System clock reference base，system clock reference extenstion：系统时钟参考字段。

3) Pack stuffing length： 包填充长度字段，3 位整数，规定该字段后填充字节的个数

80 60 53 1f 00 94 89 00 00 0000 00 00 00 01 ba €`S..??.....?

7e ff 3e fb 44 01 00 5f 6b f8 00 00 01 e0 14 53 ~.>?D.._k?...?.S

80 80 05 2f bf cf bed1 1c 42 56 7b 13 58 0a 1e €€./????.BV{.X..

08 b1 4f 33 69 35 0453 6d 33 a8 04 15 58 d9 21 .?O3i5.Sm3?..X?!

9741 b9 f1 75 3d 94 2b 1f bc 0b b2 b4 97 bf 93 ?A??u=?+.?.?????
前12位是RTP Header，这里不再赘述；

000001ba是包头起始码；

接下来的9位包括了SCR，SCRE，MUXRate，具体看图7

最后一位是保留位（0xf8），定义了是否有扩展，二进制如下

1111 1000

前5位跳过，后3位指示了扩展长度，这里是0.

## 3.2、系统标题

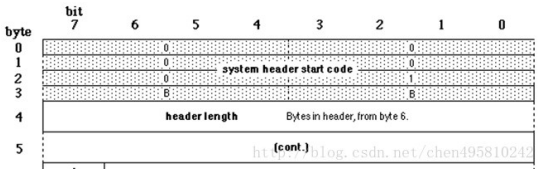



图8

Systemheader当且仅当pack是第一个数据包时才存在，即PS包头之后就是系统标题。取值0x000001BB的位串，指出系统标题的开始，暂时不需要处理，读取Header Length直接跳过即可。

## 3.3、节目映射流

Systemheader当且仅当pack是第一个数据包时才存在，即系统标题之后就是节目流映射。取值0x000001BC的位串，指出节目流映射的开始，暂时不需要处理，读取Header Length直接跳过即可。前5字节的结构同系统标题，见图8。

取一段码流分析系统标题和节目映射流

00 00 01 ba 45 a9 d4 5c 34 0100 5f 6b f8 00 00 ...?E??\4.._k?..

01 bb 00 0c 80 cc f5 04 e1 7f e0 e0 e8 c0 c0 20 .?..€??.?.?????

00 00 01 bc 00 1e e1 ff00 00 00 18 1b e0 00 0c ...?..?.....?..

2a 0a 7f ff 00 00 0708 1f fe a0 5a 90 c0 00 00 *.....??Z??..

00 00 00 00 00 00 01 e0 7f e0 80 80 0521 6a 75 .....?.?€€.!ju
前14个字节是PS包头（注意，没有扩展）；

接下来的00 00 01 bb是系统标题起始码；

接下来的00 0c说明了系统标题的长度（不包括起始码和长度字节本身）；

接下来的12个字节是系统标题的具体内容，这里不做解析；

继续看到00 00 01 bc，这是节目映射流起始码；

紧接着的00 1e同样代表长度；

跳过e1 ff，基本没用；

接下来是00 18，代表基本流长度，说明了后面还有24个字节；

接下来的1b，意思是H264编码格式；

下一个字节e0，意思是视频流；

接下里00 0c，同样代表接下的长度12个字节；

跳过这12个字节，看到90，这是G.711音频格式；

下一个字节是c0，代表音频流；

接下来的00 00同样代表长度，这里是0；

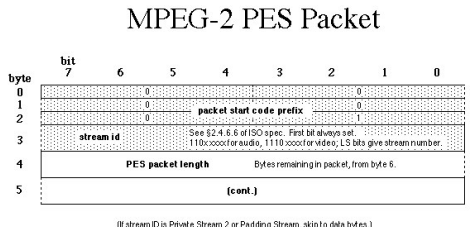
接下来4个字节是CRC，循环冗余校验。

到这里节目映射流解析完毕。（好累奋斗）。

原创不易，转载请附上链接，谢谢http://blog.csdn.net/chen495810242/article/details/39207305

好戏还在后头呢。生气

## 3.4、PES分组头部



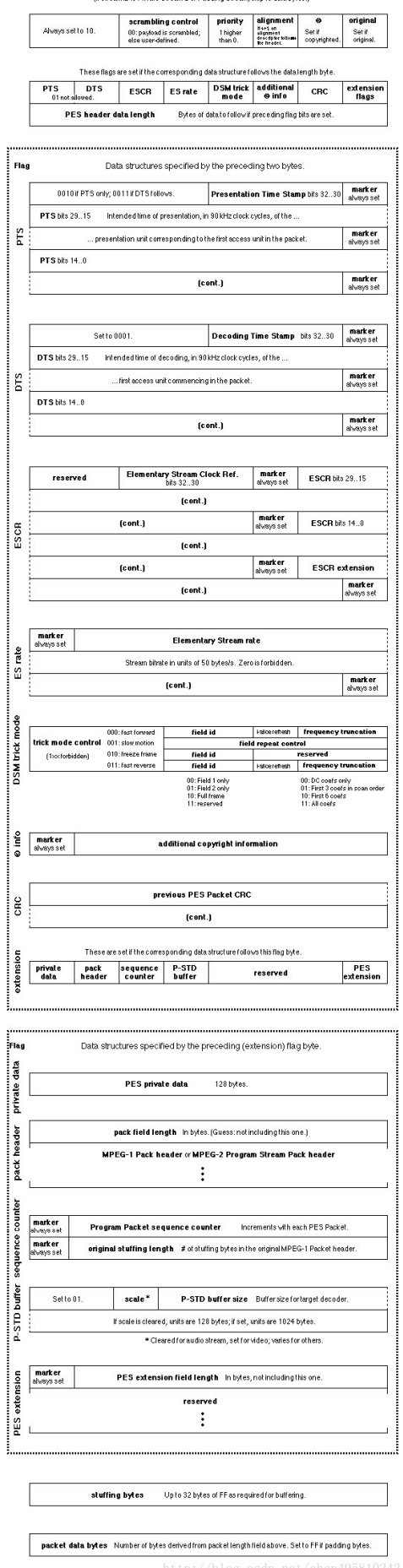





图9

别被这么长的图吓到，其实原理相同，但是，你必须处理其中的每一位。

1) Packet start code prefix：值为0x000001的位串，它和后面的stream id 构成了标识分组开始的分组起始码，用来标志一个包的开始。

2) Stream id：在节目流中，它规定了基本流的号码和类型。0x(C0~DF)指音频，0x(E0~EF)为视频

3) PES packet length：16 位字段，指出了PES 分组中跟在该字段后的字节数目。值为0 表示PES 分组长度要么没有规定要么没有限制。这种情况只允许出现在有效负载包含来源于传输流分组中某个视频基本流的字节的PES 分组中。

4) PTS_DTS：2 位字段。当值为'10'时，PTS 字段应出现在PES 分组标题中；当值为'11'时，PTS 字段和DTS 字段都应出现在PES 分组标题中；当值为'00'时，PTS 字段和DTS 字段都不出现在PES分组标题中。值'01'是不允许的。

5) ESCR：1位。置'1'时表示ESCR 基础和扩展字段出现在PES 分组标题中；值为'0'表示没有ESCR 字段。

6) ESrate：1 位。置'1'时表示ES rate 字段出现在PES 分组标题中；值为'0'表示没有ES rate 字段。

7) DSMtrick mode：1 位。置'1'时表示有8 位特技方式字段；值为'0'表示没有该字段。

8)　Additionalinfo：1 位。附加版权信息标志字段。置'1'时表示有附加拷贝信息字段；值为'0'表示没有该字段。

9)　CRC：1 位。置'1'时表示CRC 字段出现在PES 分组标题中；值为'0'表示没有该字段。

10)　Extensionflag：1 位标志。置'1'时表示PES 分组标题中有扩展字段；值为'0'表示没有该字段。

PES header data length：　8 位。PES 标题数据长度字段。指出包含在PES 分组标题中的可选字段和任何填充字节所占用的总字节数。该字段之前的字节指出了有无可选字段。

老规矩，上码流：

00 00 01 e0 21 33 80 80 05 2b 5f df 5c 95 71 84 ...?!3€€.+_?\?q?

aa e4 e9 e9 ec 40 cc17 e0 68 7b 23 f6 89 df 90 ?????@?.?h{#????

a9d4 be 74 b9 67 ad 34 6d f0 92 0d 5a 48 dd 13 ???t?g?4m??.ZH?.

00 00 01是起始码；
e0是视频流；

21 33 是帧长度；

接下来的两个80 80见下面的二进制解析；

下一个字节05指出了可选字段的长度，前一字节指出了有无可选字段；

接下来的5字节是PTS；

第7、8字节的二进制如下：

1000 0000 1000 0000

按顺序解析：

第7个字节：

10　　是标志位，必须是10；

00　　是加扰控制字段，'00'表示没有加密，剩下的01,10,11由用户自定义；

0　　是优先级，1为高，0为低；

0　　是数据对齐指示字段；

0　　是版权字段；

0　　是原始或拷贝字段。置'1'时表示相关PES分组有效负载的内容是原始的；'0'表示内容是一份拷贝；

第8个字节：

10　　是PTS_DTS字段，这里是10，表示有PTS,没有DTS；

0　　是ESCR标志字段，这里为0，表示没有该段；

0　　是ES速率标志字段，，这里为0，表示没有该段；

0　　是DSM特技方式标志字段，，这里为0，表示没有该段；

0　　是附加版权信息标志字段，，这里为0，表示没有该段；

0　　是PESCRC标志字段，，这里为0，表示没有该段；

0　　是PES扩展标志字段，，这里为0，表示没有该段；

本段码流只有PTS，贴一下解析函数

```cpp
unsigned long parse_time_stamp (const unsigned char *p)
{
    unsigned long b;
    //共33位，溢出后从0开始
    unsigned long val;

    //第1个字节的第5、6、7位
    b = *p++;
    val = (b & 0x0e) << 29;

    //第2个字节的8位和第3个字节的前7位
    b = (*(p++)) << 8;
    b += *(p++);
    val += ((b & 0xfffe) << 14);

    //第4个字节的8位和第5个字节的前7位
    b = (*(p++)) << 8;
    b += *(p++);
    val += ((b & 0xfffe) >> 1);

    return val;
}
```

其他字段可参考协议解析

ps：

遇到00 00 01 bd的，这个是私有流的标识

ps:

另外，有的hk摄像头回调然后解读出来的原始h.264码流，有的一包里只有分界符数据(nal_unit_type=9)或补充增强信息单元(nal_unit_type=6)，如果直接送入解码器，有可能会出现问题，这里的处理方式要么丢弃这两个部分，要么和之后的数据合起来，再送入解码器里，如有遇到的朋友可以交流一下:)

写在后面：

第一次发原创，在这里感谢 @cmengwei 的无私帮助，提供了很多帮助，非常感谢。

文档我都放在了我的资源里面，有1个下载积分，大家不要吝啬，绝对值得！

《RTP Payload Format for H.264 Video》

http://download.csdn.net/detail/chen495810242/7904367

《MPEG2-2(13818中文版)》

http://download.csdn.net/detail/chen495810242/7904401

RTP荷载H264的代码参考：

http://blog.csdn.net/dengzikun/article/details/5807694

RTP荷载PS流的代码参考：

http://www.pudn.com/downloads33/sourcecode/windows/multimedia/detail105823.html

http://www.oschina.net/code/snippet_99626_23737

请不要跟我要源码，参考我提供的这些，你足以写出一个可以正常运行的程序。

授人以鱼不如授人以渔。

其他参考：

http://blog.csdn.net/duanbeibei/article/details/1698183

http://blog.csdn.net/wwyyxx26/article/details/15224879

http://blog.csdn.net/chen495810242/article/details/39207305