# RTP 协议

## 1.RTP介绍

数据传输协议RTP，用于实时传输数据。RTP全名是Real-time Transport Protocol（实时传输协议）。它是IETF提出的一个标准，对应的RFC文档为RFC3550。RFC3550不仅定义了RTP，而且定义了配套的相关协议RTCP（Real-time Transport Control Protocol，即实时传输控制协议）。RTP用来为IP网上的语音、图像、传真等多种需要实时传输的多媒体数据提供端到端的实时传输服务。RTP为Internet上端到端的实时传输提供时间信息和流同步，但并不保证服务质量，服务质量由RTCP来提供。

## 2.RTP内容

　　RTP标准定义了两个子协议 ，RTP和RTCP

　　数据传输协议RTP，用于实时传输数据。该协议提供的信息包括：时间戳（用于同步）、序列号（用于丢包和重排序检测）、以及负载格式（用于说明数据的编码格式）。

　　控制协议RTCP，用于QoS反馈和同步媒体流。相对于RTP来说，RTCP所占的带宽非常小，通常只有5%。

## 3.RTP的使用

**3.1.为什么要使用RTP**

　　一提到流媒体传输、一谈到什么视频监控、视频会议、语音电话（VOIP），都离不开RTP协议的应用，但当大家都根据经验或者别人的应用而选择RTP协议的时候，你可曾想过，为什么我们要使用RTP来进行流媒体的传输呢？为什么我们一定要用RTP？难道TCP、UDP或者其他的网络协议不能达到我们的要求么？

　　像TCP这样的可靠传输协议，通过超时和重传机制来保证传输数据流中的每一个bit的正确性，但这样会使得无论从协议的实现还是传输的过程都变得非常的复杂。而且，当传输过程中有数据丢失的时候，由于对数据丢失的检测（超时检测）和重传，会数据流的传输被迫暂停和延时。

　　或许你会说，我们可以利用客户端构造一个足够大的缓冲区来保证显示的正常，这种方法对于从网络播放音视频来说是可以接受的，但是对于一些需要实时交互的场合（如视频聊天、视频会议等），如果这种缓冲超过了200ms，将会产生难以接受的实时性体验。

**3.2.为什么RTP可以解决上述时延问题**

　　RTP协议是一种基于UDP的传输协议，RTP本身并不能为按顺序传送数据包提供可靠的传送机制，也不提供流量控制或拥塞控制，它依靠RTCP提供这些服务。

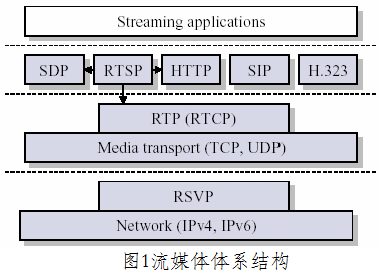
这样，对于那些丢失的数据包，不存在由于超时检测而带来的延时，同时，对于那些丢弃的包，也可以由上层根据其重要性来选择性的重传。比如，对于I帧、P帧、B帧数据，由于其重要性依次降低，

故在网络状况不好的情况下，可以考虑在B帧丢失甚至P帧丢失的情况下不进行重传，这样，在客户端方面，虽然可能会有短暂的不清晰画面，但却保证了实时性的体验和要求。

## 4.RTP的协议层次

**4.1.传输层的子层**

下图给出了流媒体应用中的一个典型的协议体系结构。



从图中可以看出，RTP被划分在传输层，它建立在UDP上。同UDP协议一样，为了实现其实时传输功能，RTP也有固定的封装形式。

RTP用来为端到端的实时传输提供时间信息和流同步，但并不保证服务质量。服务质量由RTCP来提供。

**4.2.应用层的一部分**

　　从应用开发者的角度看，RTP 应当是应用层的一部分。在应用的发送端，开发者必须编写用 RTP 封装分组的程序代码，然后把 RTP 分组交给 UDP 插口接口。在接收端，

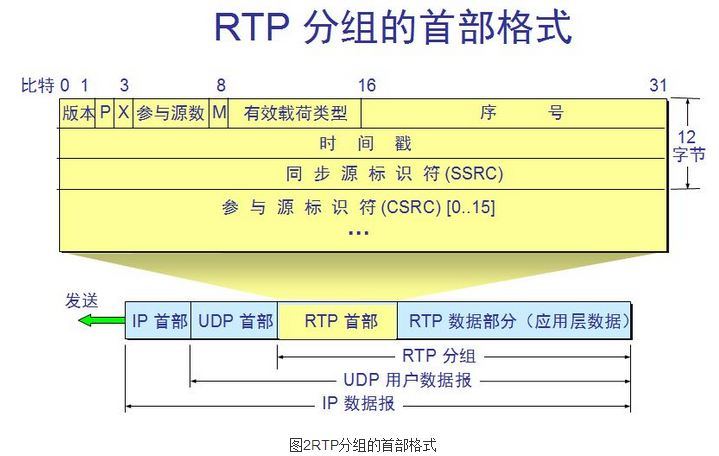
RTP 分组通过 UDP 插口接口进入应用层后，还要利用开发者编写的程序代码从 RTP 分组中把应用数据块提取出来。

## 5.RTP报文

**5.1.RTP分组**

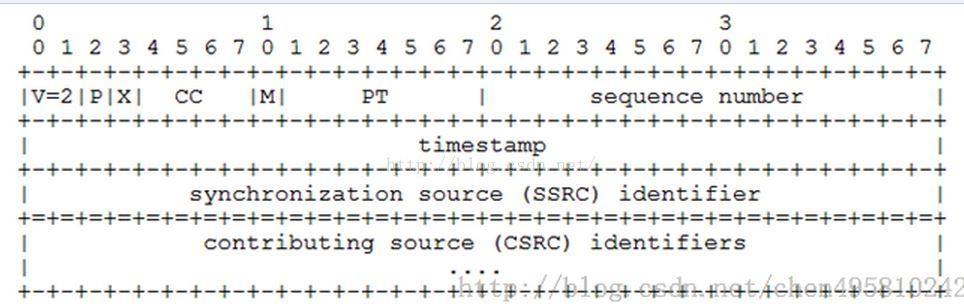
  RTP分组只包含RTP数据，而控制是由RTCP协议提供。RTP在1025到65535之间选择一个未使用的偶数UDP端口号，而在同一次会话中的RTCP则使用下一个奇数UDP端口号。

端口号5004和5005分别用作RTP和RTCP的默认端口号。RTP分组的首部格式如图2所示，其中前12个字节是必须的。



**5.2.RTP首部**

5.2.1.RTP报文头格式（见RFC3550 Page12）：



1).版本号（V）：用来标志使用的RTP版本，占2位，当前协议版本号为2

2).填充位（P）：填充标志，占1位，如果P=1，则该RTP包的尾部就包含附加的填充字节，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).扩展位（X）：扩展标志，占1位，如果X=1，则在RTP固定头部后面就跟有一个扩展头部

4).CSRC计数器（CC）：CSRC计数器，占4位，指示固定头部后面跟着的CSRC 标识符的个数

5).标记位（M）：标记，占1位，该位的解释由配置文档（Profile）来承担，不同的有效载荷有不同的含义，一般而言，对于视频，标记一帧的结束；对于音频，标记会话的开始。

6).载荷类型（PayloadType）： 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。

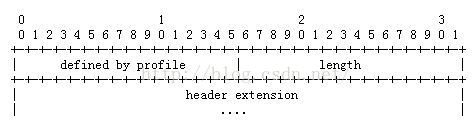
7).序列号（SN）：占16位，用于标识发送者所发送的RTP报文的序列号，每发送一个报文，序列号增1。接收端可以据此检测丢包和重建包序列，当下层的承载协议用UDP的时候，网络状况不好的时候可以用来检查丢包，同时出现网络抖动的情况可以用来对数据进行重新排序，序列号的初始值是随机的，同时音频包和视频包的sequence是分别记数的。

8).时间戳(Timestamp): 占32位，记录了该包中数据的第一个字节的采样时刻。在一次会话开始时，时间戳初始化成一个初始值。即使在没有信号发送时，时间戳的数值也要随时间而不断地增加（时间在流逝嘛）。时钟频率依赖于负载数据格式，并在描述文件（profile）中进行描述。一般而言，必须使用90 kHz 时钟频率。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).同步源标识符(SSRC)：占32位，用于标识同步信源，同步源就是指RTP包流的来源。在同一个RTP会话中不能有两个相同的SSRC值。该标识符是随机选取的，RFC1889推荐了MD5随机算法。

10).贡献源列表（CSRC List）：可以有0～15个，每个CSRC标识符占32位，用来标志对一个RTP混合器产生的新包有贡献的所有RTP包的源。由混合器将这些有贡献的SSRC标识符插入表中。SSRC标识符都被列出来，以便接收端能正确指出交谈双方的身份。

5.2.3.RTP扩展头结构



若 RTP 固定头中的扩展比特位置1（注意：如果有CSRC列表，则在CSRC列表之后），

则一个长度可变的头扩展部分被加到 RTP 固定头之后。头扩展包含 16 比特的长度域，指示扩展项中 32 比特字的个数，不包括 4 个字节扩展头(因此零是有效值)。

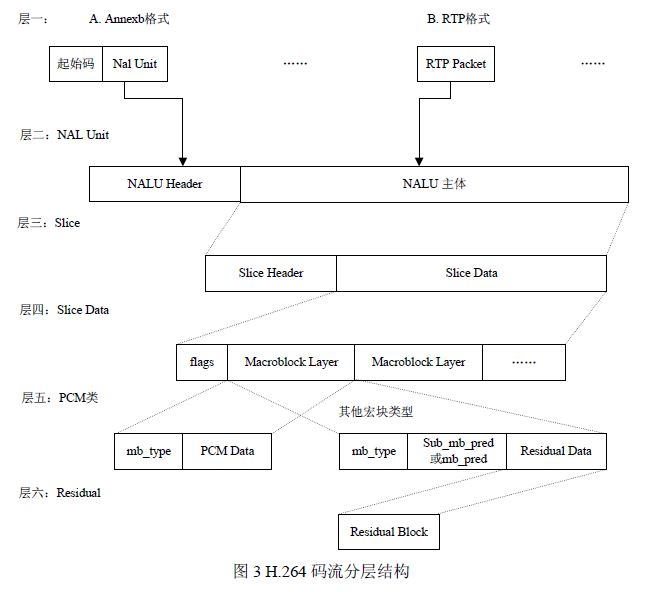
　　RTP 固定头之后只允许有一个头扩展。为允许多个互操作实现独立生成不同的头扩展，或某种特定实现有多种不同的头扩展,扩展项的前 16 比特用以识别标识符或参数。这 16 比特的格式由具体实现的上层协议定义。基本的 RTP 说明并不定义任何头扩展本身。

**5.3.RTP数据部分**

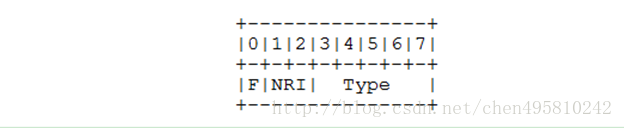
5.3.1.RTP荷载H264码流

1).码流结构

RFC3984是H.264的baseline码流在RTP方式下传输的规范，



2).荷载结构



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（如上图）识别荷载结构。

I、 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

II、 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

III、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

3).单个NAL包单元

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。

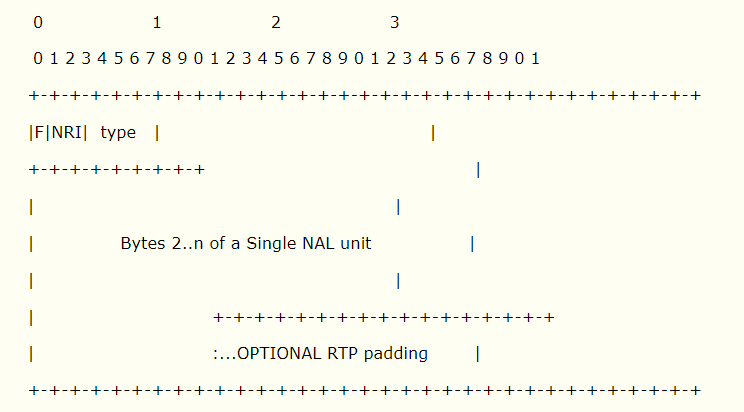
打包H264码流时，只需在帧前面加上12字节的RTP头即可，具体来说：

12字节的RTP头后面的就是音视频数据。一个封装单个NAL单元包到RTP的NAL单元流的RTP序号必须符合NAL单元的解码顺序。

对于 NALU 的长度小于 MTU 大小的包, 一般采用单一 NAL 单元模式.

对于一个原始的 H.264 NALU 单元常由[Start Code] [NALU Header] [NALU Payload]三部分组成, 其中 Start Code 用于标示这是一个 NALU 单元的开始, 必须是 "00 00 00 01" 或 "00 00 01", NALU 头仅一个字节, 其后都是 NALU 单元内容.

打包时去除 "00 00 01" 或 "00 00 00 01" 的开始码, 把其他数据封包的 RTP 包即可.



如有一个 H.264 的 NALU 是这样的:

[00 00 00 01 67 42 A0 1E 23 56 0E 2F ... ]

这是一个序列参数集 NAL 单元. [00 00 00 01] 是四个字节的开始码, 67 是 NALU 头, 42 开始的数据是 NALU 内容.

封装成 RTP 包将如下:

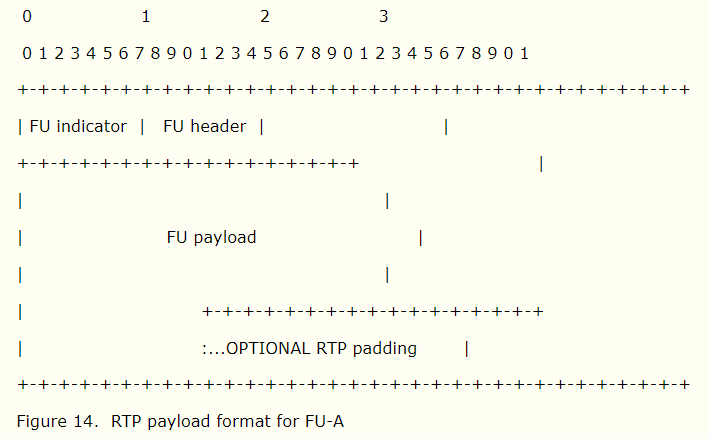
[ RTP Header ] [ 67 42 A0 1E 23 56 0E 2F ]

即只要去掉 4 个字节的开始码就可以了.

4).分片单元（FU-A）

数据比较大的H264视频包，被RTP分片发送。12字节的RTP头后面跟随的就是FU-A分片：

当 NALU 的长度超过 MTU 时, 就必须对 NALU 单元进行分片封包. 也称为 Fragmentation Units (FUs).

  Figure 14.  RTP payload format for FU-A

I、FU indicator有以下格式：

  +---------------+

  |0|1|2|3|4|5|6|7|

  +-+-+-+-+-+-+-+-+

  |F|NRI|  Type   |

  +---------------+

FU指示字节的类型域Type=28表示FU-A。。NRI域的值必须根据分片NAL单元的NRI域的值设置。

NAL单元荷载类型定义见下表

表1. 单元类型以及荷载结构总结

  .Type   Packet      Type name

  ---------------------------------------------------------

  0      undefined                                    -

  1-23   NAL unit    Single NAL unit packet per H.264

  24     STAP-A     Single-time aggregation packet

  25     STAP-B     Single-time aggregation packet

  26     MTAP16    Multi-time aggregation packet

  27     MTAP24    Multi-time aggregation packet

  28     FU-A      Fragmentation unit

  29     FU-B      Fragmentation unit

  30-31  undefined

II、FU header的格式如下：

  +---------------+

  |0|1|2|3|4|5|6|7|

  +-+-+-+-+-+-+-+-+

  |S|E|R|  Type   |

  +---------------+

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。

Type: 5 bits

III、打包和解包

打包：当编码器在编码时需要将原有一个NAL按照FU-A进行分片，原有的NAL的单元头与分片后的FU-A的单元头有如下关系：

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，

FU indicator与FU header的剩余位数根据实际情况决定。

解包：当接收端收到FU-A的分片数据，需要将所有的分片包组合还原成原始的NAl包时，FU-A的单元头与还原后的NAL的关系如下：

还原后的NAL头的八位是由FU indicator的前三位加FU header的后五位组成，即：

nal\_unit\_type = (fu\_indicator & 0xe0) | (fu\_header & 0x1f)

IV、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NAl 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。

V、注意

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。

**5).组合封包模式**

当 NALU 的长度特别小时, 可以把几个 NALU 单元封在一个 RTP 包中.

5.3.2.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：

a、        MPEG-4 视频流： 0x10；

b、        H.264 视频流： 0x1B；

c、        SVAC 视频流： 0x80；

d、        G.711 音频流： 0x90；

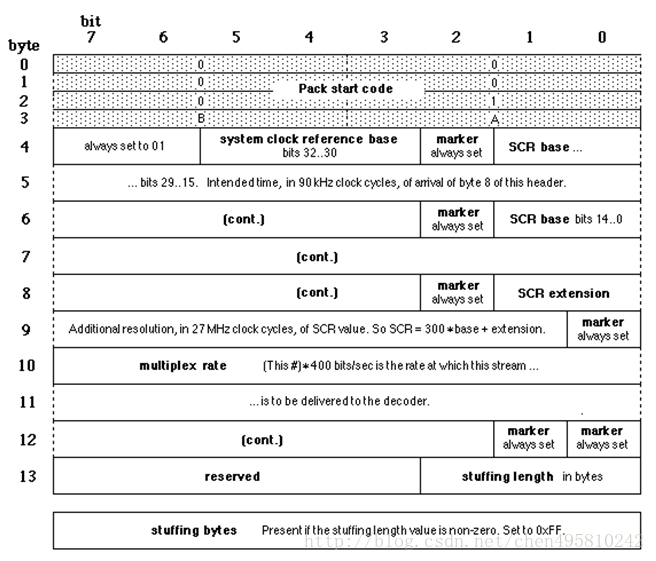
e、        G.722.1 音频流： 0x92；

f、        G.723.1 音频流： 0x93；

g、        G.729 音频流： 0x99；

h、        SVAC音频流： 0x9B。

1).PS包头



I、        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。

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

III、      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；

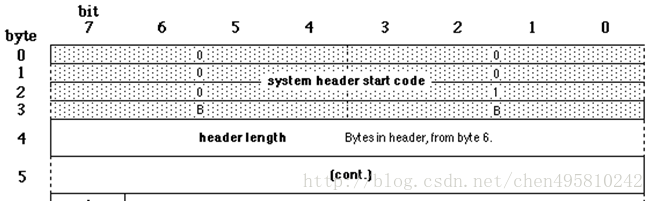
接下来的9位包括了SCR，SCRE，MUXRate，具体看上图

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.

2).系统标题



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).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，见上图。

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；

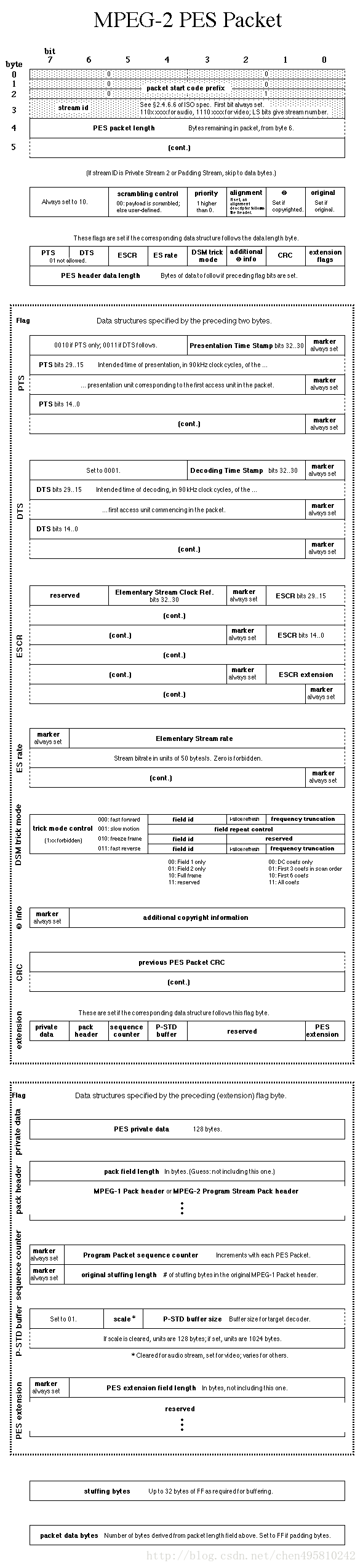
下一个字节是c0，代表音频流；

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。

4).PES分组头部



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] view plain copy

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

## 6.RTP的会话过程

当应用程序建立一个RTP会话时，应用程序将确定一对目的传输地址。目的传输地址由一个网络地址和一对端口组成，有两个端口：

一个给RTP包，一个给RTCP包，使得RTP/RTCP数据能够正确发送。RTP数据发向偶数的UDP端口，而对应的控制信号RTCP数据发向相邻的奇数UDP端口（偶数的UDP端口＋1），这样就构成一个UDP端口对。

RTP的发送过程如下，接收过程则相反。

   RTP协议从上层接收流媒体信息码流（如H.264），封装成RTP数据包；RTCP从上层接收控制信息，封装成RTCP控制包。

   RTP将RTP 数据包发往UDP端口对中偶数端口；RTCP将RTCP控制包发往UDP端口对中的接收端口，即奇数端口。

## 7.RTP的profile机制

        RTP为具体的应用提供了非常大的灵活性，它将传输协议与具体的应用环境、具体的控制策略分开，传输协议本身只提供完成实时传输的机制，开发者可以根据不同的应用环境，自主选择合适的配置环境、以及合适的控制策略。

  这里所说的控制策略指的是你可以根据自己特定的应用需求，来实现特定的一些RTCP控制算法，比如前面提到的丢包的检测算法、丢包的重传策略、一些视频会议应用中的控制方案等等（这些策略我可能将在后续的文章中进行描述）。

  对于上面说的合适的配置环境，主要是指RTP的相关配置和负载格式的定义。RTP协议为了广泛地支持各种多媒体格式（如 H.264, MPEG-4, MJPEG, MPEG），没有在协议中体现出具体的应用配置，而是通过profile配置文件以及负载类型格式说明文件的形式来提供。

对于任何一种特定的应用，RTP定义了一个profile文件以及相关的负载格式说明，相关的文件如下所示：

《RTP Profile for Audio and Video Conferences with Minimal Control》（RFC3551）

《RTP Payload Format for H.264 Video》（RFC3984）

《RTP Payload Format for MPEG-4 Audio/Visual Streams》（RFC3016）

等等，想了解更多可以点击这里：http://en.wikipedia.org/wiki/RTP\_audio\_video\_profile

说明：如果应用程序不使用专有的方案来提供有效载荷类型(payload type)、顺序号或者时间戳，而是使用标准的RTP协议，应用程序就更容易与其他的网络应用程序配合运行，

这是大家都希望的事情。例如，如果有两个不同的公司都在开发因特网电话软件，他们都把RTP合并到他们的产品中，这样就有希望：使用不同公司电话软件的用户之间能够进行通信。

## 8.RTCP的主要功能

服务质量的监视与反馈、媒体间的同步，以及多播组中成员的标识。在RTP会话期 间，各参与者周期性地传送RTCP包。RTCP包中含有已发送的数据包的数量、丢失的数据包的数量等统计资料，

因此，各参与者可以利用这些信息动态地改变传输速率，甚至改变有效载荷类型。RTP和RTCP配合使用，它们能以有效的反馈和最小的开销使传输效率最佳化，因而特别适合传送网上的实时数据。

## 9.RTCP的使用

RTCP 控制协议需要与RTP数据协议一起配合使用，当应用程序启动一个RTP会话时将同时占用两个端口，分别供RTP 和RTCP使用。RTP本身并不能为按序传输数据包提供可靠的保证，

也不提供流量控制和拥塞控制，这些都由RTCP来负责完成。通常RTCP会采用与 RTP相同的分发机制，向会话中的所有成员周期性地发送控制信息，应用程序通过接收这些数据，

从中获取会话参与者的相关资料，以及网络状况、分组丢失概率等反馈信息，从而能够对服务质量进行控制或者对网络状况进行诊断。

## 10.RTCP报文

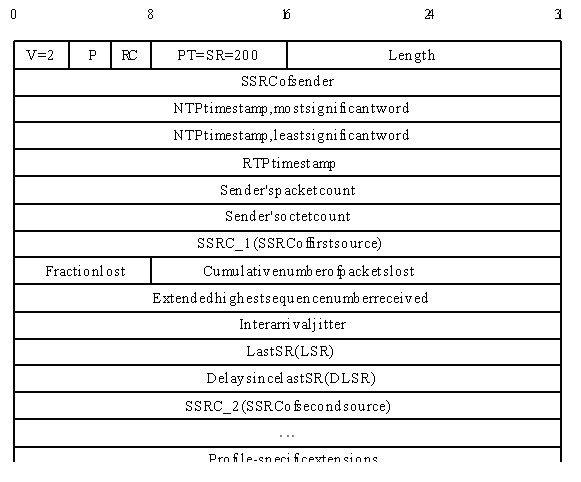
RTCP也是用UDP来传送的，但RTCP封装的仅仅是一些控制信息，因而分组很短，所以可以将多个RTCP分组封装在一个UDP包中。RTCP有如下五种分组类型。

| **类型** | **缩写表示** | **用途** |
| --- | --- | --- |
| 200 | SR（Sender Report） | 发送端报告 |
| 201 | RR（Receiver Report） | 接收端报告 |
| 202 | SDES（Source Description Items） | 源点描述 |
| 203 | BYE | 结束传输 |
| 204 | . APP | 特定应用 |

上述五种分组的封装大同小异，下面只讲述SR类型，而其它类型请参考RFC3550。

        发送端报告分组SR（Sender Report）用来使发送端以多播方式向所有接收端报告发送情况。SR分组的主要内容有：相应的RTP流的SSRC，RTP流中最新产生的RTP分组的时间戳和NTP，RTP流包含的分组数，RTP流包含的字节数。

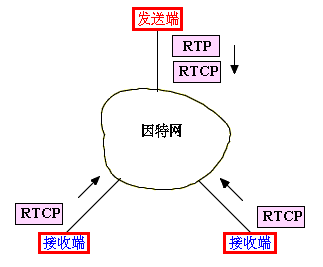
SR包的封装如下图所示。



# RTCP 协议

### RTCP概要

实时传输控制协议(Real-time ControlProtocol，RTCP)与RTP共同定义在1996年提出的RFC 1889中，是和 RTP一起工作的控制协议。**RTCP单独运行在低层协议上，由低层协议提供数据与控制包的复用。在RTP会话期间，每个会话参与者周期性地向所有其他参与者发送RTCP控制信息包，如下图所示。对于RTP会话或者广播，通常使用单个多目标广播地址，属于这个会话的所有RTP和RTCP信息包都使用这个多目标广播地址，通过使用不同的端口号可把RTP信息包和RTCP信息包区分开来。**



### RTCP功能：**监视传输统计信息和服务质量**

1、为应用程序提供会话质量或者广播性能质量的信息

**RTCP的主要功能是为应用程序提供会话质量或者广播性能质量的信息**。每个RTCP信息包不封装声音数据或者电视数据，而是**封装发送端（和 / 或者）接收端的统计报表**。这些信息**包括发送的信息包数目、丢失的信息包数目和信息包的抖动等情况**，这些反馈信息反映了当前的网络状况，对发送端、接收端或者网络管理员都非常有用。RTCP规格没有指定应用程序应该使用这些反馈信息做什么，这完全取决于应用程序开发人员。例如，发送端可以根据反馈信息来调整传输速率，接收端可以根据反馈信息判断问题是本地的、区域性的还是全球性的，网络管理员也可以使用RTCP信息包中的信息来评估网络用于多目标广播的性能。

2、确定 RTP用户源

RTCP为每个RTP用户提供了一个全局唯一的规范名称 (Canonical Name)标志符 CNAME，接收者使用它来追踪一个RTP进程的参加者。当发现冲突或程序重新启动时，RTP中的同步源标识符SSRC可能发生改变，接收者可利用CNAME来跟踪参加者。同时，接收者也需要利用CNAME在相关RTP连接中的几个数据流之间建立联系。当 RTP需要进行音视频同步的时候，接受者就需要使用 CNAME来使得同一发送者的音视频数据相关联，然后根据RTCP包中的计时信息(Network time protocol)来实现音频和视频的同步。

3、控制 RTCP传输间隔

由于每个对话成员定期发送RTCP信息包，随着参加者不断增加，RTCP信息包频繁发送将占用过多的网络资源，为了防止拥塞，必须限制RTCP信息包的流量，**控制信息所占带宽一般不超过可用带宽的 5%**，因此就需要调整 RTCP包的发送速率。由于任意两个RTP终端之间都互发 RTCP包，因此终端的总数很容易估计出来，**应用程序根据参加者总数就可以调整RTCP包的发送速率。**

4、传输最小进程控制信息

这项功能对于**参加者可以任意进入和离开的松散会话进程**十分有用，参加者可以自由进入或离开，没有成员控制或参数协调。

# ****RTCP信息包****

**RTCP也是用UDP来传送的**，但RTCP封装的仅仅是一些控制信息，因而分组很短，所以可以将多个RTCP分组封装在一个UDP包中。

类似于RTP信息包，每个RTCP信息包以固定部分开始，紧接着的是可变长结构单元，最后以一个32位边界结束。

**根据所携带的控制信息不同RTCP信息包可分为RR（接收者报告包）、SR（源报告包）、SEDS（源描述包）、BYE（离开申明）和APP（特殊应用包）五类5类：**

1. SR：发送端报告，由发送 RTP 数据报的应用程序或中端发出的。
2. RR：接收端报告，由接受但不发送 RTP 数据报的应用程序或中端发出。
3. SDES: 源描述，传递与会话成员有关的标识信息的载体，如用户名、邮件、电话等。
4. BYE: 通知离开，通知回话中的其他成员将退出会话。
5. APP: 由应用程序自己定义，作为 RTCP 协议的扩展。

**RTCP 协议如何实现媒体流的同步**

通过抓包分析 RTCP 发送端报告，RTP 的同步其实就靠这三个域：

1. sender SSRC ：SR 包发送的同步源标识符。与对应 RTP 包中的 SSRC 一样
2. NTP timestamp：SR 包发送时的绝对时间。用于同步不同的流。
3. RTP timestamp：与 NTP 时间戳对应，与 RTP 包中的时间戳具有相同的初始值。

# RTSP协议

**RTSP**（Real Time Streaming Protocol）是由Real Network和Netscape共同提出的如何有效地在IP网络上传输流媒体数据的应用层协议。RTSP对流媒体提供了诸如暂停，快进等控制，而它本身并不传输数据，RTSP的作用相当于流媒体服务器的**远程控制**。服务器端可以自行选择使用TCP或UDP来传送串流内容，它的语法和运作跟HTTP 1.1类似，但并不特别强调时间同步，所以比较能容忍网络延迟。

RTSP处理流时会根据端点间可用带宽大小，将音视频等数据切割成小分组（packet）进行传输，使得客户端在播放一个分组的同时，可以解压缓存中第二个甚至下载第三个分组。通过**缓存和多码率流**技术，用户将不会感觉到数据间存在停顿。至于RTSP的特性，则主要体现在如下方面：

 多服务器兼容 ：媒体流可来自不同服务器

 可协商：客户端和服务器可协商feature支持程度

 HTTP亲和性：尽可能重用HTTP概念，包括认证、状态码、解析等

 易解析：HTML或MIME解析器均可在RTSP中适用

 易扩展：新的方法或参数甚至协议本身均可添加或定制

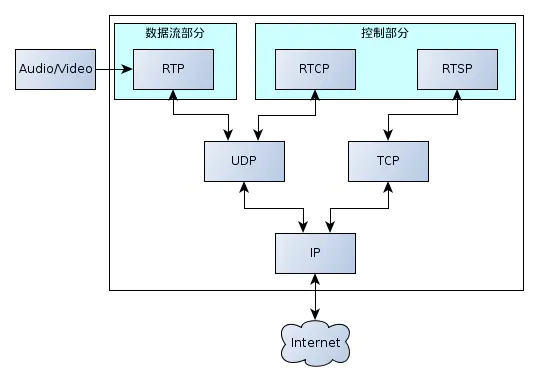
 防火墙亲和性：传输层或应用层防火墙均可被协议较好处理

 服务器控制：控制概念易于理解，服务器不允许向客户端传输不能被客户端关闭的流

 多场景适用：RTSP提供帧级别精度，适用于更多媒体应用场景

RTSP组合使用了可靠传输协议TCP（控制）和高效传输协议UDP（内容）来串流（streaming）内容给用户。它支持点播（Video-On-Demand）以及直播（Live Streaming）服务。

RTSP协议本身并不负责数据传输，**通常（非必须）**是通过RTP（Real-time Transport Protocol）配合RTCP（Real-time Control Protocol）完成数据流和控制命令（同步、QOS管理等）的传输。具体应用中，三者的关系如下图所示：



## RTSP和HTTP RTP(RTCP)的关系

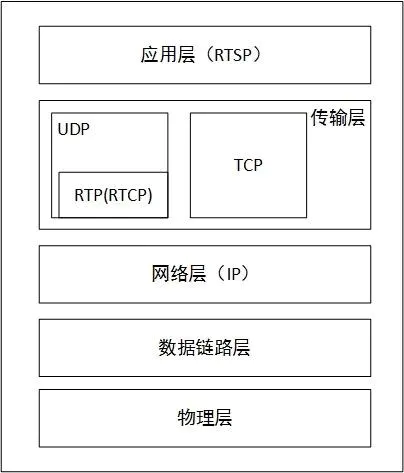
### RTSP和HTTP

* **联系**：两者都用纯文本来发送消息，且rtsp协议的语法也和HTTP类似。RTSP一开始这样设计，也是为了能够兼容使用以前写的HTTP协议分析代码 。
* **区别**：rtsp是有状态的，不同的是RTSP的命令需要知道现在正处于一个什么状态，也就是说rtsp的命令总是**按照顺序来发送**，某个命令总在另外一个命令之前要发送。RTSP不管处于什么状态都**不会断掉连接**。而http则不保存状态，协议在发送一个命令以后，**连接就会断开**，且命令之间是**没有依赖性**。rtsp协议使用554端口，http使用80端口。

### RTSP和RTP(RTCP)

* **RTP**：Realtime Transport Potocol 实时传输协议  
  RTP提供时间标志,序列号以及其他能够保证在实时数据传输时处理时间的方法。
* **RTCP**：Realtime Transport Control Potocol 实时传输控制协议  
  RTCP是RTP的控制部分,用来保证服务质量和成员管理。RTP和RTCP是一起使用的。
* **RTSP**：RealTime Streaming Potocol 实时流协议  
  RTSP具体数据传输交给RTP,提供对流的远程控制

RTP是基于 UDP协议的， UDP不用建立连接，效率更高；但允许丢包， 这就要求在重新组装媒体的时候多做些工作  
RTP只是包裹内容信息，而RTCP是交换控制信息的，QoS是通过RTCP实现的  
应用程序对应的是play, seek, pause, stop等命令，RTSP则是处理这些命令，在UDP传输时并使用RTP(RTCP)来完成。如果是TCP连接则不会使用RTP(RTCP)。



**RTSP的client连接server通过SDP（**[**会话描述协议**](https://link.jianshu.com/?t=http://blog.chinaunix.net/uid-24372973-id-293651.html)**）传递信息**

# 2. RTSP方法

RTSP中并没有连接的概念，而是通过会话（Session）进行管理。每个会话有对应的会话ID，会话中可能可能涉及一至多个流，会话生命周期中，客户端也可能切换连接（如TCP）来传递RTSP请求（request）。

## 2.1RTSP消息

RTSP的消息有两大类，一是请求消息(request)，一是回应消息(response)，两种消息的格式不同。  
**请求消息格式**：

方法 URI RTSP版本 CR LF  
消息头 CR LF CR LF  
消息体 CR LF

方法包括：OPTIONS、SETUP、PLAY、TEARDOWN DESCRIBE  
URI是接收方（服务端）的地址，例如：[rtsp://192.168.22.136:5000/v0](https://link.jianshu.com?t=rtsp:/192.168.22.136:5000/v0)  
每行后面的CR LF表示回车换行，需要接收端有相应的解析，消息头需要有两个CR LF。

DESCRIBE rtsp://192.168.1.211 RTSP/1.0

CSeq: 1

Accept: application/sdp

User-Agent: magnus-fc

**回应消息格式**：

RTSP版本 状态码 解释 CR LF  
消息头 CR LF CR LF  
消息体 CR LF

其中RTSP版本一般都是RTSP/1.0，状态码是一个数值，200表示成功，解释是与状态码对应的文本解释，详细请见：[SDP协议介绍。](#sdp_intro)

RTSP/1.0 200 OK

CSeq: 1

Server: GrandStream Rtsp Server V100R001

Content-Type: application/sdp

Content-length: 256

Content-Base: rtsp://192.168.1.211/0

v=0

o=StreamingServer 3331435948 1116907222000 IN IP4 192.168.1.211

s=h264.mp4

c=IN IP4 0.0.0.0

t=0 0

a=control:\*

m=video 0 RTP/AVP 96

a=control:trackID=0

a=rtpmap:96 H264/90000

m=audio 0 RTP/AVP 97

a=control:trackID=1

a=rtpmap:97 G726-16/8000

## 2.2简单的rtsp交互过程:

C表示rtsp客户端, S表示rtsp服务端

**step1:**  
C->S:OPTION request //询问S有哪些方法可用  
S->C:OPTION response //S回应信息中包括提供的所有可用方法

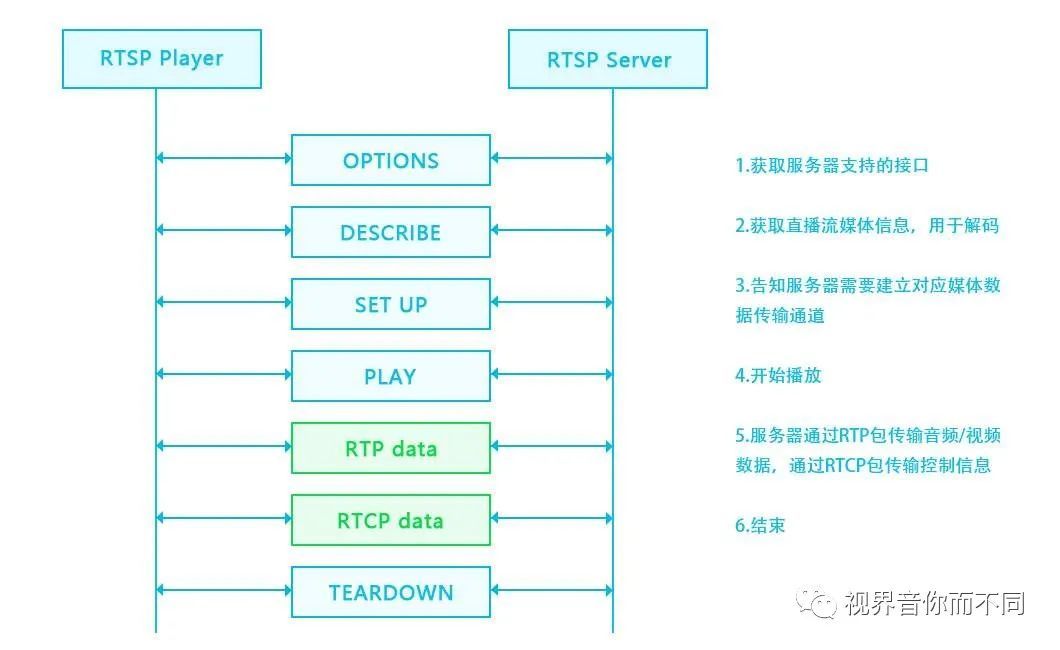
**step2:**  
C->S:DESCRIBE request //要求得到S提供的媒体初始化描述信息  
S->C:DESCRIBE response //S回应媒体初始化描述信息，主要是sdp

**step3:**  
C->S:SETUP request //设置会话的属性，以及传输模式，提醒S建立会话  
S->C:SETUP response //S建立会话，返回会话标识符，以及会话相关信息

**step4:**  
C->S:PLAY request //C请求播放  
S->C:PLAY response //S回应该请求的信息

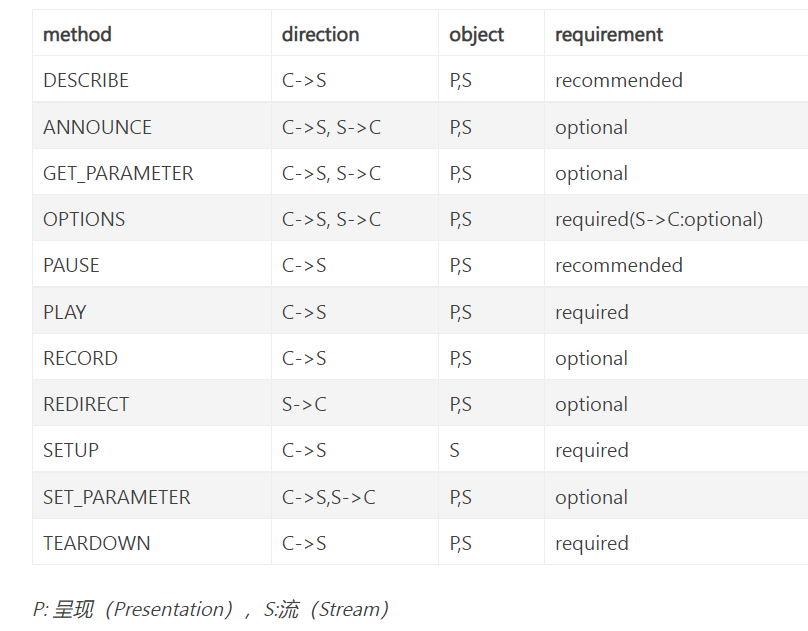
S->C:发送流媒体数据

**step5:**  
C->S:TEARDOWN request //C请求关闭会话  
S->C:TEARDOWN response //S回应该请求

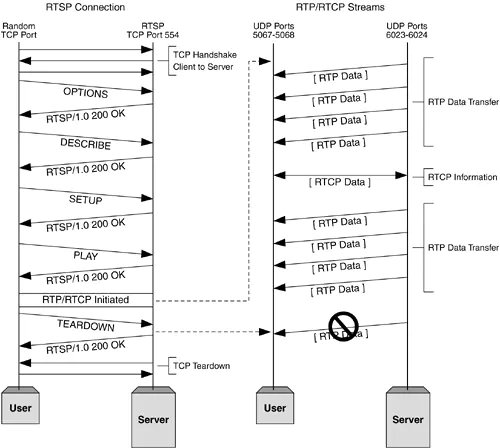


## 2.3 RTSP中常用方法

RTSP中并没有连接的概念，而是通过会话（Session）进行管理。每个会话有对应的会话ID，会话中可能可能涉及一至多个流，会话生命周期中，客户端也可能切换连接（如TCP）来传递RTSP请求（request）。



一个RTSP应用（如点播）生命周期内，通常会话（所有交互）过程如下图所示：



### OPTIONS

得到服务器提供的可用方法

OPTIONS rtsp://192.168.20.136:5000/xxx666 RTSP/1.0

CSeq: 1 //每个消息都有序号来标记，第一个包通常是option请求消息

User-Agent: VLC media player (LIVE555 Streaming Media v2005.11.10)

服务器的回应信息包括提供的一些方法,例如:

RTSP/1.0 200 OK

Server: UServer 0.9.7\_rc1

Cseq: 1 //每个回应消息的cseq数值和请求消息的cseq相对应

Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE, SCALE,GET\_PARAMETER //服务器提供的可用的方法

### DESCRIBE

C向S发起DESCRIBE请求,为了得到会话描述信息(SDP):

DESCRIBE rtsp://192.168.20.136:5000/xxx666 RTSP/1.0

CSeq: 2

token:

Accept: application/sdp

User-Agent: VLC media player (LIVE555 Streaming Media v2005.11.10)

服务器回应一些对此会话的描述信息([sdp](#sdp_intro)):

RTSP/1.0 200 OK

Server: UServer 0.9.7\_rc1

Cseq: 2

x-prev-url: rtsp://192.168.20.136:5000

x-next-url: rtsp://192.168.20.136:5000

x-Accept-Retransmit: our-retransmit

x-Accept-Dynamic-Rate: 1

Cache-Control: must-revalidate

Last-Modified: Fri, 10 Nov 2006 12:34:38 GMT

Date: Fri, 10 Nov 2006 12:34:38 GMT

Expires: Fri, 10 Nov 2006 12:34:38 GMT

Content-Base: rtsp://192.168.20.136:5000/xxx666/

Content-Length: 344

Content-Type: application/sdp

v=0 //以下都是sdp信息

o=OnewaveUServerNG 1451516402 1025358037 IN IP4 192.168.20.136

s=/xxx666

u=http:///

e=admin@

c=IN IP4 0.0.0.0

t=0 0

a=isma-compliance:1,1.0,1

a=range:npt=0-

m=video 0 RTP/AVP 96 //m表示媒体描述，下面是对会话中视频通道的媒体描述

a=rtpmap:96 MP4V-ES/90000

a=fmtp:96 profile-level-id=245;config=000001B0F5000001B509000001000000012000C888B0E0E0FA62D089028307 a=control:trackID=0 //trackID＝0表示视频流用的是通道0

### SETUP

客户端提醒服务器建立会话,并确定传输模式(用于请求URL使用指定传输格式，必须在PLAY前发出):

SETUP rtsp://192.168.20.136:5000/xxx666/trackID=0 RTSP/1.0

CSeq: 3

Transport: RTP/AVP/TCP;unicast;interleaved=0-1

User-Agent: VLC media player (LIVE555 Streaming Media v2005.11.10)

//uri中 带有trackID＝0，表示对该通道进行设置。Transport参数设置了传输模式，包的结构。接下来的数据包头部第二个字节位置就是 interleaved，它的值是每个通道都不同的，trackID＝0的interleaved值有两个0或1，0表示rtp包，1表示rtcp包，接收端根据interleaved的值来区别是哪种数据包。

服务器回应信息:

RTSP/1.0 200 OK

Server: UServer 0.9.7\_rc1

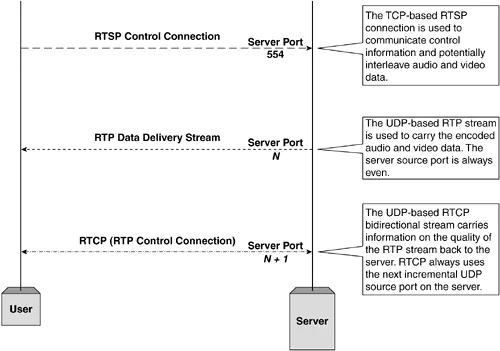
Cseq: 3

Session: 6310936469860791894 //服务器回应的会话标识符

Cache-Control: no-cache

Transport: RTP/AVP/TCP;unicast;interleaved=0-1;ssrc=6B8B4567

客户端请求中，指明了用于接收RTP数据（音视频）的本地端口5067，以及RTCP数据（元信息）的端口5068。这里图示说明下RTSP（554/8554）、RTP、RTCP端口关系。



可以看到，RTCP端口是基于RTP的，且始终为其端口值+1。服务器回复中，确认了客户端所请求的端口，并给出服务器端对应开辟的端口值6023/6024。

### PLAY

用于请求服务器使用SETUP中确认的机制开始传输数据，客户端不应在SETUP请求未被确认应答成功前发出PLAY请求。另外需要注意，PLAY请求是需要排队的，其中可携带Range域以指明区间。:

PLAY rtsp://192.168.20.136:5000/xxx666 RTSP/1.0

CSeq: 4

Session: 6310936469860791894

Range: npt=0.000- //设置播放时间的范围

User-Agent: VLC media player (LIVE555 Streaming Media v2005.11.10)

服务器回应信息:

RTSP/1.0 200 OK

Server: UServer 0.9.7\_rc1

Cseq: 4

Session: 6310936469860791894

Range: npt=0.000000-

RTP-Info: url=trackID=0;seq=17040;rtptime=1467265309

//seq和rtptime都是rtp包中的信息

### TEARDOWN

客户端发起关闭请求:

TEARDOWN rtsp://192.168.20.136:5000/xxx666 RTSP/1.0

CSeq: 5

Session: 6310936469860791894

User-Agent: VLC media player (LIVE555 Streaming Media v2005.11.10)

服务器回应:

RTSP/1.0 200 OK

Server: UServer 0.9.7\_rc1

Cseq: 5

Session: 6310936469860791894

除了上述提到的5种必选方法外，还有如下方法是可选的。

* ANNOUNCE  
  ANNOUNCE方法有两个用途：
  + 从服务器发送给客户端时：用于更新实时会话描述
  + 从客户端发送给服务器时：推送URL指定的呈现或媒体对象的描述

如果呈现**途中插入新流，应重发完整描述，而不仅仅是增量**。这样一来，也就支持了动态移除组件。

C->S: ANNOUNCE rtsp://video.foocorp.com:554/streams/example.rm RTSP/1.0

CSeq: 10

Session: 47112344

Content-Type: application/sdp

Content-Length: 332

v=0

o=mhandley 2890844526 2890845468 IN IP4 126.16.64.4

s=SDP Seminar

i=A Seminar on the session description protocol

u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps

e=mjh@isi.edu (Mark Handley)

c=IN IP4 224.2.17.12/127

t=2873397496 2873404696

a=recvonly

m=audio 3456 RTP/AVP 0

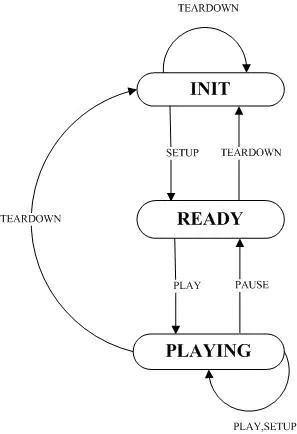
m=video 2232 RTP/AVP 31

S->C: RTSP/1.0 200 OK

CSeq: 10

* GET\_PARAMETER  
  用于请求URI指定呈现或流的参数值。回复的内容有待实现，不带任何实体主体的GET\_PARAMETER可用于测试客户端或服务器是否在线（类似“ping”程序）。
* S->C: GET\_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
* CSeq: 431
* Content-Type: text/parameters
* Session: 12345678
* Content-Length: 15
* packets\_received
* jitter
* C->S: RTSP/1.0 200 OK
* CSeq: 431
* Content-Length: 46
* Content-Type: text/parameters
* packets\_received: 10
* jitter: 0.3838
* SET\_PARAMETER  
  SET\_PARAMETER请求用于设置URI指定呈现或流的参数值。  
  每条请求应当只包含一个参数以允许客户端确定失败原因，如果请求中包含多个参数，服务器必须只在所有参数都能成功设置情况下生效。服务器应当允许同一参数多次设置同一值，但可拒绝设置为不同值。  
  *注意：媒体流传输参数必须通过SETUP请求来设置*
* C->S: SET\_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
* CSeq: 421
* Content-Length: 20
* Content-type: text/parameters
* barparam: barstuff
* S->C: RTSP/1.0 451 Invalid Parameter
* CSeq: 421
* Content-Length: 10
* Content-type: text/parameters
* barparam
* PAUSE  
  PAUSE请求会临时中断流传输，如果URL指定的是某一个流，则该流的播放和录制会被暂停。例如，对于音频而言，相当于静音操作。如果URL指定的是一组流，则呈现或组中所有活动流将会被暂停。当恢复播放或录制时，所有轨道的流必须进行同步。  
  此过程中，所有服务器资源均会保留，除非SETUP时，头中有参数指定延时，那么服务器可能在触发延时后关闭会话并释放资源。
* C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/1.0
* CSeq: 834
* Session: 12345678
* S->C: RTSP/1.0 200 OK
* CSeq: 834
* Date: 23 Jan 1997 15:35:06 GMT
* RECORD  
  RECORD方法用于开始录制当前呈现描述中的一段媒体数据，UTC格式时间戳包含开始点和结尾点。如未给出时间范围，则使用呈现描述中的开始点和结尾点。如会话已处于运行中，则立即开始录制。  
  服务器决定将录制数据保存在请求URI或其他URI，如使用其他URI，需回复“201（Created）”并包含实体以描述请求状态和新资源位置信息。  
  一个支持直播情境下录制的服务器必须支持clock格式，这里smpte格式并没有意义。
* C->S: RECORD rtsp://example.com/meeting/audio.en RTSP/1.0
* CSeq: 954
* Session: 12345678
* Conference: 128.16.64.19/32492374
* REDIRECT  
  REDIRECT请求用于提示客户端它必须连接至另一个服务器，其中强制包含了Location头，以指明新的服务器URL。其中还可能包含Range参数，指明何时重定向将生效。如果客户端希望向该URI发送和接收媒体，则客户端必须先对当前会话发出TEARDOWN请求，然后向目标主机发出SETUP请求新的会话。
* S->C: REDIRECT rtsp://example.com/fizzle/foo RTSP/1.0
* CSeq: 732
* Location: rtsp://bigserver.com:8001
* Range: clock=19960213T143205Z-

# 2.4 RTSP状态机



# RTMP 协议

Real Time Messaging Protocol(实时消息传送协议协议)是Adobe Systems公司为Flash Player和服务器之间音频,视频和数据传输开发的私有协议,adobe目前提供了一个并不完整的rtmp specification给大众使用,所以在使用rtmp协议时需要按flash player返回的包进行解析. 目前rtmp有以下几个变种:

* rtmp是工作在TCP之上的明文协议,默认使用1935端口
* rtmps是rtmp使用TLS/SSL连接
* rtmpe是adobe使用自己的加密机制对rtmp进行加密的,虽然加密机制是使用了行业标准,并且内部实现也是专有的,但rtmpe设计基本上错误的,它本身也不提供任何的安全性.
* rtmpt是对rtmp协议提供了一个http的封装,主要是为了防止防火墙对其进行拦截.

# 1. handshake

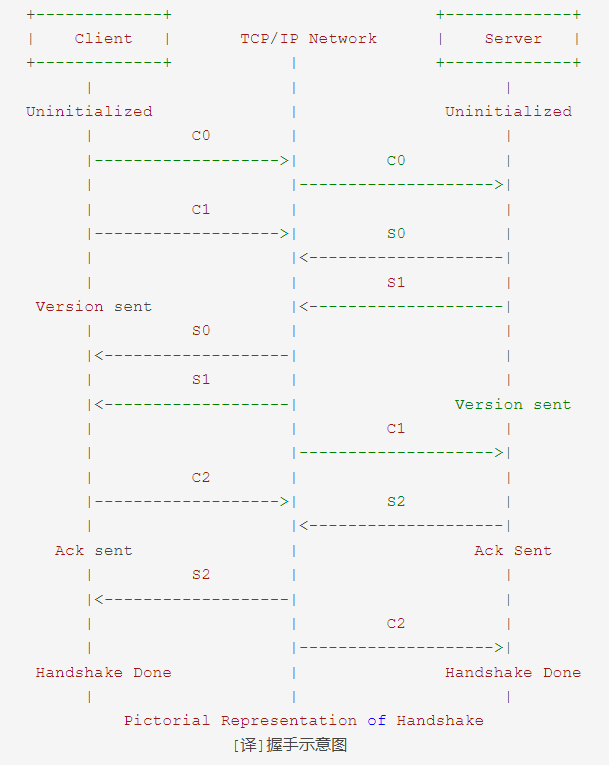
## 1.1 概述

rtmp 连接从握手开始。它包含三个固定大小的块。客户端发送的三个块命名为 C0,C1,C2；服务端发送的三个块命名为  
S0,S1,S2。

握手序列：

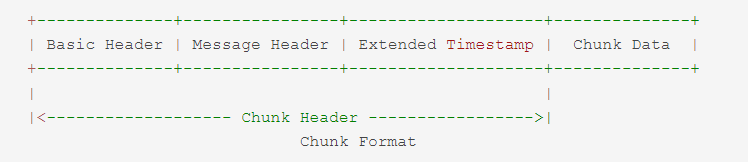
* 客户端通过发送 C0 和 C1 消息来启动握手过程。客户端必须接收到 S1 消息，然后发送 C2 消息。客户端必须接收到  
  S2 消息，然后发送其他数据。
* 服务端必须接收到 C0 或者 C1 消息，然后发送 S0 和 S1 消息。服务端必须接收到 C2 消息，然后发送其他数据。

##### 握手示意图



# 2. 组块

## 2.1 块格式



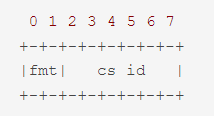
* 块的基本头（1-3字节）：这个字段包含块流ID和块类型。块类型决定了编码过的消息头的格式。这个字段是一个变  
  长字段，长度取决于块流ID。
* 消息头（0,3,7,11字节）：这个字段包含被发送的消息信息（无论是全部，还是部分）。字段长度由块头中的  
  块类型来决定。
* 扩展时间戳（0,4字节）：这个字段是否存在取决于块消息头中编码的时间戳。
* 块数据（可变大小）：当前块的有效数据，上限为配置的最大块大小。

## 2.2 Basic Header

包含 chunk stream ID（流通道id）和chunk type（即fmt），chunk stream id 一般被简写为CSID，用来唯一标识一个特定的流通道，chunk type决定了后面Message Header的格式。Basic Header的长度可能是 1，2，或 3 个字节，  
其中 chunk type 的长度是固定的（占2位，单位是bit），Basic Header 的长度取决于 CSID 的大小，在足够存储这两个字段的前提下最好用尽量少的字节从而减少由于引入Header增加的数据量。

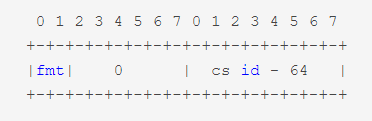
RTMP协议支持用户自定义 [3,65599] 之间的 CSID，0, 1, 2 由协议保留表示特殊信息。0 代表 Basic Header 总共要占用 2 个字节，CSID 在 [64,319] 之间; 1 代表占用 3 个字节，CSID 在 [64,65599] 之间; 2 代表该 chunk 是控制信息和一些命令信息。

### 2.2.1 Basic Header：1 byte



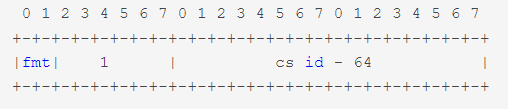
### 2.2.2 Basic Header: 2 byte , csid == 0

CSID占14bit，此时协议将于chunk type所在字节的其他bit都置为0，剩下的一个字节表示CSID - 64，这样共有8个bit来存储 CSID，8 bit 可以表示 [0,255] 个数，因此这种情况下 CSID 在 [64,319]，其中 319 = 255 + 64。



### 2.2.3 Basic Header: 3 bytes , csid == 1

CSID占22bit，此时协议将第一个字节的[2,8]bit置1，余下的16个bit表示CSID - 64，这样共有16个bit来存储CSID，16bit可以表示[0,65535]共 65536 个数，因此这种情况下 CSID 在 [64,65599]，其中65599=65535+64，需要注意的是，Basic Header是采用小端存储的方式，越往后的字节数量级越高，因此通过3个字节的每一个bit的值来计算CSID时，应该是: <第三个字节的值> \* 256 + <第二个字节的值> + 64.

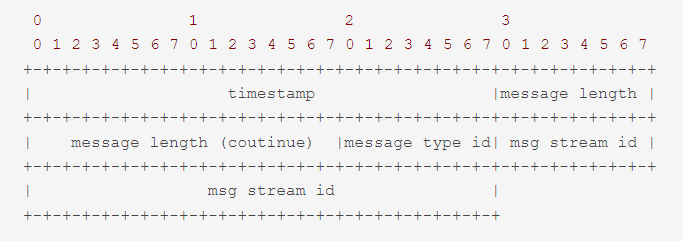


## 2.3 Message Header

包含了要发送的实际信息（可能是完整的，也可能是一部分）的描述信息。Message Header的格式和长度取决于Basic Header的chunk type，即fmt，共有四种不同的格式。其中第一种格式可以表示其他三种表示的所有数据，但由于其他三种格式是基于对之前chunk的差量化的表示，因此可以更简洁地表示相同的数据，实际使用的时候还是应该采用尽量少的字节表示相同意义的数据。下面按字节从多到少的顺序分别介绍这四种格式的 Message Header。

### Message Header 四种消息头格式。

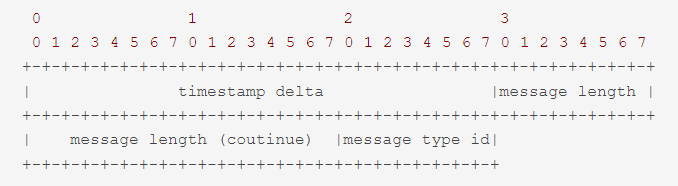
#### 一、Chunk Type(fmt) = 0：11 bytes



type=0时Message Header占用11个字节，其他三种能表示的数据它都能表示，但在chunk stream 的开始第一个chunk和头信息中的时间戳后退（即值与上一个chunk相比减小，通常在回退播放的时候会出现这种情况）的时候必须采用这种格式。

* timestamp（时间戳）：占用3个字节，因此它最多能表示到16777215=0xFFFFFF=2^24-1，当它的值超过这个最大值时，这三个字节都置为1，这样实际的timestamp会转存到 ExtendedTimestamp 字段中，接收端在判断timestamp字段24个位都为1时就会去Extended Timestamp中解析实际的时间戳。
* message length（消息数据长度）：占用3个字节，表示实际发送的消息的数据如音频帧、视频帧等数据的长度，单位是字节。注意这里是Message的长度，也就是chunk属于的Message的总长  
  度，而不是chunk本身data的长度。
* message type id(消息的类型id)：1个字节，表示实际发送的数据的类型，如8代表音频数据，9代表视频数据。
* message stream id(消息的流id)：4个字节，表示该chunk所在的流的ID，和Basic Header的CSID一样，它采用小端存储方式。

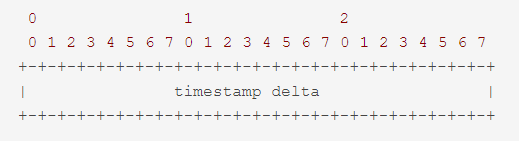
#### 二、Chunk Type(fmt) = 1：7 bytes



type为1时占用7个字节，省去了表示message stream id的4个字节，表示此chunk和上一次发的 chunk 所在的流相同，如果在发送端和对端有一个流链接的时候可以尽量采取这种格式。

* timestamp delta：3 bytes，这里和type=0时不同，存储的是和上一个chunk的时间差。类似上面提到的timestamp，当它的值超过3个字节所能表示的最大值时，三个字节都置为1，实际的时间戳差值就会转存到Extended Timestamp字段中，接收端在判断timestamp delta字段24个bit都为1时就会去Extended Timestamp 中解析实际的与上次时间戳的差值。
* 其他字段与上面的解释相同.

#### 三、Chunk Type(fmt) = 2：3 bytes



type 为 2 时占用 3 个字节，相对于 type = 1 格式又省去了表示消息长度的3个字节和表示消息类型的1个字节，表示此 chunk和上一次发送的 chunk 所在的流、消息的长度和消息的类型都相同。余下的这三个字节表示 timestamp delta，使用同type=1。

#### 四、Chunk Type(fmt) = 3: 0 byte

type=3时，为0字节，表示这个chunk的Message Header和上一个是完全相同的。当它跟在type=0的chunk后面时，表示和前一个 chunk 的时间戳都是相同。什么时候连时间戳都是相同呢？就是一个 Message 拆分成多个 chunk，这个 chunk 和上一个 chunk 同属于一个 Message。而当它跟在 type = 1或 type = 2 的chunk后面时的chunk后面时，表示和前一个 chunk  
的时间戳的差是相同的。比如第一个 chunk 的 type = 0，timestamp = 100，第二个 chunk 的 type = 2，timestamp delta = 20，表示时间戳为 100 + 20 = 120，第三个 chunk 的 type = 3，表示 timestamp delta = 20,时间戳为 120 + 20 = 140。

## 2.4 Extended Timestamp(扩展时间戳)

在 chunk 中会有时间戳 timestamp 和时间戳差 timestamp delta，并且它们不会同时存在，只有这两者之一大于3字节能表示的  
最大数值 0xFFFFFF ＝ 16777215 时，才会用这个字段来表示真正的时间戳，否则这个字段为 0。扩展时间戳占 4 个字节，  
能表示的最大数值就是 0xFFFFFFFF ＝ 4294967295。当扩展时间戳启用时，timestamp字段或者timestamp delta要全置为1，  
而不是减去时间戳或者时间戳差的值。

#### https://www.cnblogs.com/jimodetiantang/p/8974075.html

## 简略：

## 1. RTMP协议介绍

Real Time Messaging Protocol(实时消息传送协议协议)是Adobe Systems公司为Flash Player和服务器之间音频,视频和数据传输开发的私有协议,adobe目前提供了一个并不完整的rtmp specification给大众使用,所以在使用rtmp协议时需要按flash player返回的包进行解析. 目前rtmp有以下几个变种:

* rtmp是工作在TCP之上的明文协议,默认使用1935端口
* rtmps是rtmp使用TLS/SSL连接
* rtmpe是adobe使用自己的加密机制对rtmp进行加密的,虽然加密机制是使用了行业标准,并且内部实现也是专有的,但rtmpe设计基本上错误的,它本身也不提供任何的安全性.
* rtmpt是对rtmp协议提供了一个http的封装,主要是为了防止防火墙对其进行拦截.

## 2. 包结构

* rtmp消息包使用的是二进制数据流,它们使用AMF0/AMF3进行编码.与其它协议一样,rtmp消息也是也包括消息头与消息体,而消息头又可以分为basic header,chunk header,timestamp.
* basic header是此包的唯一不变的部分,并且由一个独立的byte构成,这其中包括了2个作重要的标志位,chunk type以及stream id.chunk type决定了消息头的编码格式,该字段的长度完全依赖于stream id,stream id是一个可变长的字段.
* message header该字段包含了将要发送的消息的信息(或者是一部分,一个消息拆成多个chunk的情况下是一部分)该字段的长度由chunk basic header中的trunk type决定. timestamp扩展时间戳就比较好理解的,就是当chunk message header的时间戳大于等于0xffffff的时候chunk message header后面的四个字节就代表扩展时间.

## 3. 握手

在rtmp连接建立后,服务端与客户端需要通过3次交换报文完成握手. 握手其他的协议不同,是由三个静态大小的块,而不是可变大小的块组成的,客户端与服务器发送相同的三个chunk,客户端发送c0,c1,c2 chunk,服务端发送s0,s1,s2 chunk.

## 4. 发送顺序

1. 握手开始时,客户端将发送c0,c1 chunk,此时客户端必须等待,直到收到s1 chunk,才能发送c2 chunk.
2. 此时服务端必须等待,直到已收到c0后才能发送s0和s1,当然也可能会等到接收c1后才发送.
3. 当服务器收到c2后才能再发送的其他数据,同理,当客户端收到s2后才能发送其它数据.

## 5. 握手包格式

* c0与s0格式 c0和s0包是一个1字节,可以看作是一个byte 目前rtmp版本定义为3,0-2是早期的专利产品所使用的值,现已经废弃,4-31是预留值,32-255是禁用值(这样做是为了区分基于文本的协议,因为这些协议通常都是以一个可打印的字符开始),如果服务端不能识别客户请求的版本,那么它应该发送3的响应,客户端这时可以选择下降到版本3,也可以放弃这次握手.
* c1与s1格式 c1与s1长度为1536个字节,它们由以下字段组成 时间戳:该字段占4字节,包含了一个时间戳,它是所有从这个端点发送出去的将来数据块的起始点,它可以是零,或是任意值,为了同步多个数据块流,端点可能会将这个字段设成其它数据块流时间戳的当前值. 0:此标记位占4字节,并且必须是0 随机数:该字段占1528字节,可以是任意值,因为每个端点必须区分已经初始化的握手和对等端点初始化的握手的响应,所以这个数据要足够的随机,当然这个也不需要密码级的随机或是动态值.
* c2与s2格式 c2和s2包长都是1536字节,几乎是s1和c1的回显.
* time1,time2,随机数回显 time1 该字段占4字节,包含有对方发送过来s1或c1的时间戳 time2 该字段占4字节,包含有对方发送过来的前一个包(s1或者c1)的时间戳 随机数回显 该字段占1528字节,包含有对方发送过来的随机数据字段,每个通信端点可以使用time和time2字段,以及当前的时间戳,来快速估计带宽和/或连接延时,但这个数值基本上没法用.

## 6. 握手状态

* 未初始化:在这个阶段,协议版本被发送,客户和服务端都是未初始化的,客户端在包c1中发送协议版本,如果服务端支持这个版本,它将会发送s0和s1作为响应,如果不支持,则服务端会用相应的动作来响应,在RTMP中这个动作是结束这个连接.
* 版本发送完成:客户端和服务端在未初始化状态之后都进入到版本发送完成状态,客户端等待包s1,而服务端等待包c1,在收到相应的包后,客户端发送包c2,而服务端发磅包s2,状态变成询问发送完成.
* 询问发送完成:客户端和服务端等待s2和c2.
* 握手完成:客户端和服务端开始交换消息.