8 RTCP详解

腾讯课堂 零声学院

FFmpeg/WebRTC/RTMP 音 视 频 流 媒 体 高 级 开 发 https://ke.qq.com/course/468797?tuin=137bb271

RTCP功能

Real-time Transport Control Protocol或RTP Control Protocol或简写RTCP）是实时传输协议（RTP）的一个姐妹协议。RTCP由RFC 3550定义（取代作废的RFC 1889）。RTP 使用一个 偶数 UDP port ； 而RTCP 则使用 RTP 的下一个 port，也就是一个奇数 port。RTCP与RTP联合工作，RTP实施实际数据的传输，RTCP则负责将控制包送至电话中的每个人。其主要功能是就RTP正在提供的服务质量做出反馈。

RTCP报文封装在UDP中进行传输，作用如下:

- 质量反馈
- 传输层标识（CNAME）
- 给参与者发送RTCP控制报文
- 最小会话控制消息（可选）

RTCP端口号 = RTP端口号 + 1

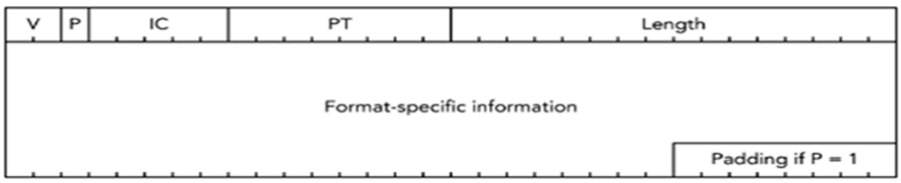
RTCP报文格式--报文类型

在RTP的规范（RFC 3550）中，一共定义了5种RTCP报告用来报告当前控制信息：

Packet Type值	分组类型	描述
200	SR（Sender report）	发送方报告
201	RR（Receiver report）	接收方报告

202	SDES（Source description）	源描述报告
203	BYE（Goodbye）	离开会话
204	APP（Application-defined）	应用定义

RTCP的5种报告：RR，SR，SDES，BYE和APP。他们使用共同的结构，但是在某些具体的地方有一些不同。
以下是RTCP报文基本结构：



其中比较重要的几个域及其意义如下：

域名	长度（bit）	含义
Version（V）	2	定义了RTP的版本，此协议定义的版本是2。
Padding（P）	1	如果填充位被设置为1，则一个或多个附加的字节会加在包头的最后， 附加的最后一个字节放置附加的字节数。填充可能用于某些具有固定长 度的加密算法，或者在底层数据单元中传输多个RTP包。
Item count（IC）	5	有些RTCP分组类型包含多个条目（item），IC用来计算有多少个条目。因为IC只有5个比特，所以最多31个item。如果需要的item超过31个，那么应用实现必须包含多个RTCP分组。如果IC为0表示空的item列表。分组如果不需要item列表，那么可以把IC字段用于其他目的。
Packet type（PT）	8	PT标识了分组中携带消息的类型。在RTP标准中定义了5种类型：RR，SR，SDES，BYE和APP。
Length（M）	16	分组长度，以4 bytes为单位，所以意味着RTCP分组必须是4字节对齐。该长度不包含32 bites固定头，也就是说length为0也是合理的，说明只有4字节的头部（这种情况IC也是0）。

header size = 2 + 1 + 5 +8 + 16 = 4个字节

```
Length: 6 (28 bytes)
Sender SSRC: 0xd801e570 (3624002928)
Timestamp, MSW: 3813460475 (0xe34cc9fb)
Timestamp, LSW: 4011499454 (0xef1a9fbe)
[MSW and LSW as NTP timestamp: Nov  4, 2020 06:34:35.933999999 UTC]
RTP timestamp: 2794561330
Sender's packet count: 0
Sender's octet count: 0
[RTCP frame length check: OK - 28 bytes]
```

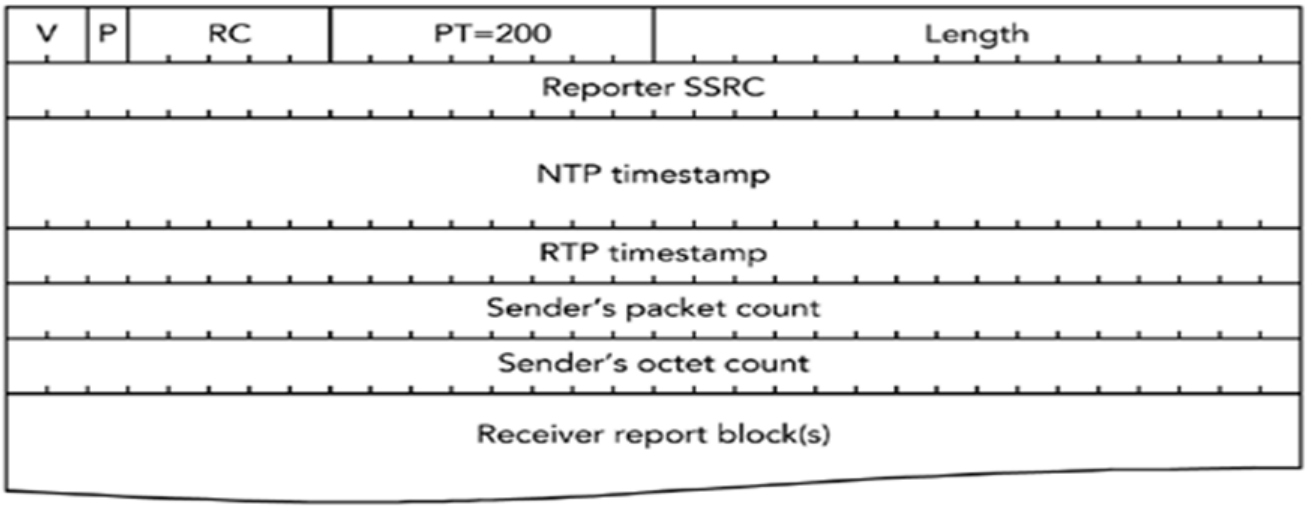
```
0  00 0c 29 11 82 79 9c 30  5b a4 43 9b 08 00 45 00   ··)··y·0 [·C···E·
0  00 38 6e e7 00 00 80 11  44 f4 c0 a8 02 df c0 a8   ·8n····· D·······
0  02 aa 7b 67 e8 51 00 24  54 eb 80 c8 00 06 d8 01   ··{g·Q·$ T·······
0  e5 70 e3 4c c9 fb ef 1a  9f be a6 91 9f 32 00 00   ·p·L···· ·····2··
0  00 00 00 00 00 00                                  ······
```

SR 包，总共28字节 = header 4字节 + 4字节*6

```
typedef struct _rtcp_header_t
{
    uint32_t v:2;       // version
    uint32_t p:1;       // padding
    uint32_t rc:5;      // reception report count
    uint32_t pt:8;      // packet type
    uint32_t length:16; /* pkt len in words, w/o this word */
} rtcp_header_t;
```

RTCP报文格式-- SR报文格式

为了补充接收者报告，RTP协议还规定了最近发送数据的参与者发送SR，该报告提供了发送的媒体的一些信息。主要用于接收端同步多媒体流，如语音和视频流。



其中域及其意义如下：

域名	长度（bit）	含义

NTP timestamp	64	64 bites，无符号整数。指出了该报告发出时的时间。该时间戳的高32 bites以NTP格式表示，从1900年1月1日开始计数，以秒为单位。低32 bites表示秒后面的小数。如果需要转化Unix时间到NTP时间，在Unix时间加上2,208,988,800即可。
RTP timestamp	/	RTP时间戳以RTP媒体流的时钟为单位，这个值通常不等于前面分组数据的RTP时间戳，因为时间会流逝。
Sender's packet count	/	表示这个同步源从这个会话开始到现在（发出RTCP报文时）发出的数据分组的个数。
Sender's octet count	/	表示同步源从这个会话开始到现在（发出RTCP报文时）发出的所有数据分组的字节数。如果发送者改变了SSRC，那么sender's packet count和sender's octet count被会被重置。

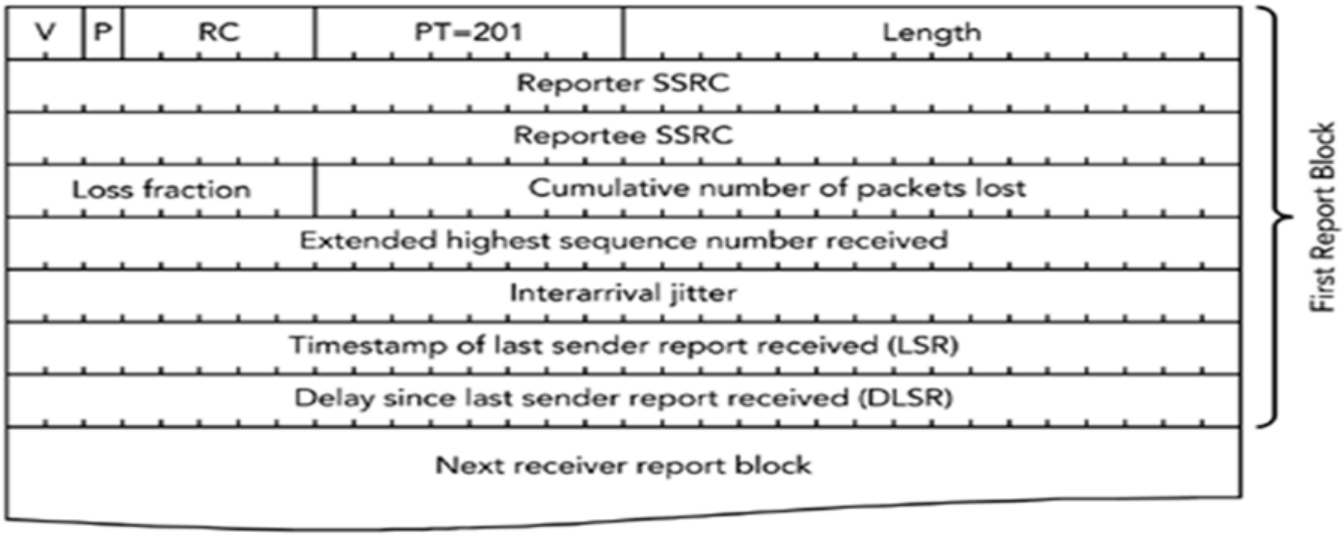
```
typedef struct _rtcp_sr_t // sender report
{
    uint32_t ssrc;
    uint32_t ntpmsw; // ntp timestamp MSW(in second)
    uint32_t ntplsw; // ntp timestamp LSW(in picosecond)
    uint32_t rtpts;  // rtp timestamp
    uint32_t spc;    // sender packet count
    uint32_t soc;    // sender octet count
} rtcp_sr_t;
```

以下是SR报文的实例:

```
Real-time Transport Control Protocol (Sender Report)
  [Stream setup by SDP (frame 218)]
      [Setup frame: 218]
      [Setup Method: SDP]
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0000 = Reception report count: 0
    Packet type: Sender Report (200)
    Length: 6 (28 bytes)
    Sender SSRC: 0x34ca36fd (885667581)
    Timestamp, MSW: 3585974789 (0xd5bda205)
    Timestamp, LSW: 1717987000 (0x666666b8)
    [MSW and LSW as NTP timestamp: Aug 20, 2013 08:06:29.400000 UTC]
    RTP timestamp: 191731761
    Sender's packet count: 53
    Sender's octet count: 9116
Real-time Transport Control Protocol (Source description)
  [RTCP frame length check: OK - 76 bytes]
```

RTCP报文格式-- RR报文格式

RTCP通过RR可以很好地保证传输质量，每个接收数据的参与者都要发出RR。



其中PT定义为201。接收者报告包含发送者的SSRC，跟随在由RC指定的（0个或多个）报告块之后。

其中域及其意义如下：

域名	长度(bit)	含义
Reportee SSRC（接收者同步源）	32	占用32 bites，表示这个报告反馈给谁，也就是谁适合接收这个报告。
Loss fraction（丢包率）	8	占用8 bites，定义为这个报告周期丢失的分组除以期待的分组数量。
Cumulative number of packets lost（丢包数量）	24	24 bites带符号整形，累计丢失的包的个数。计算方法是：期待接收的分组数目-实际接收的分组数目。所谓期待的分组数目是这样定义的：最后一个分组的序列号－初始化分组序列号。
Extended highest sequence number（扩展高位序列号）	32	占用32 bites，低16 bites取值为当前收到的RTP报文的序列号，高16 bites是扩展位，用于标识序列号周期的计数。
Interarrival jitter（抖动评估）	32	占用32 bites，数据分组传输的统计评估值，用于评估网络的抖动情况。表示方式和时间戳相同。
Last sender report (LSR)	32	占用32 bites，等于从reportee最近接收到SR分组的64 bits NTP 格式时间戳的中间32 bites，如果没有接收到SR分组，那么LSR置0。
Delay since last		表示接收到最近的SR到发出这个报告的时延，单位是1/65,536

sender report (DLSR)		秒。如果没有接收到SR，DLSR置0。

```
typedef struct _rtcp_rr_t // receiver report
{
    uint32_t ssrc;
} rtcp_rr_t;

typedef struct _rtcp_rb_t // report block
{
    uint32_t ssrc;
    uint32_t fraction:8; // fraction lost
    uint32_t cumulative:24; // cumulative number of packets lost
    uint32_t exthsn; // extended highest sequence number received
    uint32_t jitter; // interarrival jitter
    uint32_t lsr; // last SR
    uint32_t dlsr; // delay since last SR
} rtcp_rb_t;
```

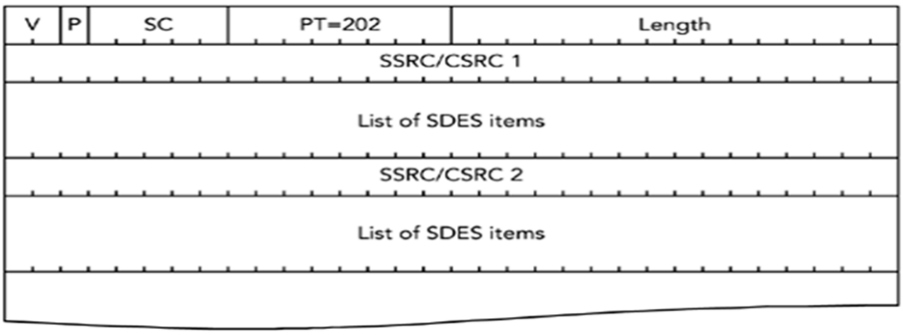
- 丢包率计算：表示方式：分母固定为256，分子是loss fraction表示的整数。所以如果想要表示1/4的报文丢失，那么loss fraction＝64。
- 丢包数量计算：cumulative number of packets lost不是以每个周期为计算范围，而是以整个会话为计算范围。所以0x7fffff是cumulative number of packets lost的最大值，因为它是带符号整数。
- 扩展高位序列号：随着会话时间增长，16比特长度的序列号可能会不够用，当序列号又回到初始化序列号时，为了表示这个环绕，在高16比特记录环绕的次数，也就是把序列号扩展了。

RTCP报文格式-- RR报文实例

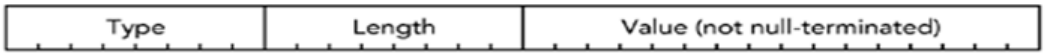
```
Real-time Transport Control Protocol (Receiver Report)
  [Stream setup by SDP (frame 252)]
    [Setup frame: 252]
    [Setup Method: SDP]
  10.. .... = Version: RFC 1889 Version (2)
  ..0. .... = Padding: False
  ...0 0001 = Reception report count: 1
  Packet type: Receiver Report (201)
  Length: 7 (32 bytes)
  Sender SSRC: 0x60f716c1 (1626805953)
  Source 1
    Identifier: 0x34ca36fd (885667581)
    SSRC contents
      Fraction lost: 141 / 256
      Cumulative number of packets lost: 74
    Extended highest sequence number received: 133
      Sequence number cycles count: 0
      Highest sequence number received: 133
    Interarrival jitter: 266143
    Last SR timestamp: 2718262886 (0xa2056666)
    Delay since last SR timestamp: 9437 (143 milliseconds)
Real-time Transport Control Protocol (Source description)
[RTCP frame length check: OK - 76 bytes]
```

RTCP报文格式-- SDES报文格式

IRTCP还可以通过传输SDES来详细描述源，如标识和一些辅助信息（地理位置，电话号码以及Email地址等信息）。一般来说SDES数据由用户输入，显示在应用的图形界面上。



- 一般来说SDES列表（list of SDES items）以被描述的源的SSRC开始。跟随一个或者多个描述项，描述项格式如下图：



- 如果描述项的type＝1，那么该描述称为CNAME item，为每个参与者提供了规范名（canonical name）。
- 这个规范名是稳固的永久的标识，独立于同步源标识。CNAME能同时用于一个参与者的多个会话。
- CNAME是唯一一个强制实现的SDES item，所有实现必须实现该描述。

RTCP报文格式-- SDES报文实例

SR的SDES报文案例

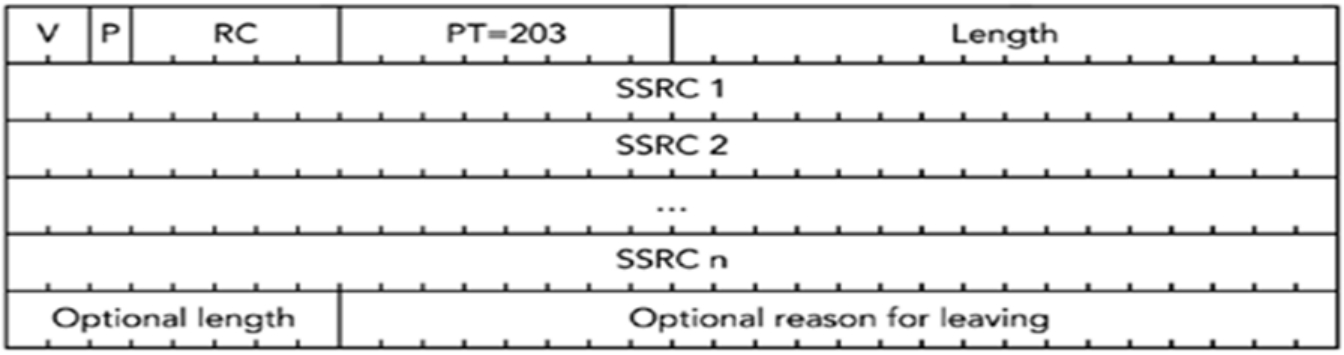
```
⊞ Real-time Transport Control Protocol (Sender Report)
⊟ Real-time Transport Control Protocol (Source description)
  ⊟ [Stream setup by SDP (frame 218)]
      [Setup frame: 218]
      [Setup Method: SDP]
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Source description (202)
    Length: 11 (48 bytes)
  ⊟ Chunk 1, SSRC/CSRC 0x34CA36FD
      Identifier: 0x34ca36fd (885667581)
    ⊟ SDES items
        Type: CNAME (user and domain) (1)
        Length: 36
        Text: fast_media_audio_668949a0@huawei.com
        Type: END (0)
  [RTCP frame length check: OK - 76 bytes]
```

RR的SDES报文案例

```
⊞ Real-time Transport Control Protocol (Receiver Report)
⊟ Real-time Transport Control Protocol (Source description)
  ⊟ [Stream setup by SDP (frame 252)]
      [Setup frame: 252]
      [Setup Method: SDP]
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Source description (202)
    Length: 10 (44 bytes)
  ⊟ Chunk 1, SSRC/CSRC 0x60F716C1
      Identifier: 0x60f716c1 (1626805953)
    ⊟ SDES items
        Type: CNAME (user and domain) (1)
        Length: 29
        Text: Administrator@WIN-PF391EB3R12
        Type: NOTE (note about source) (7)
        Length: 0
        Type: NAME (common name) (2)
        Length: 0
        Type: END (0)
  [RTCP frame length check: OK - 76 bytes]
```

RTCP报文格式-- BYE报文格式

RTCP协议可以通过BYE分组进行自由的成员控制，RTCP BYE标识离开会话的成员，或者成员改变SSRC。BYE分组可能在传输中丢失，而且应用实现不会再次产生BYE分组。所以接收者应该准备好对某些用户超时，而且没有接收到BYE分组。

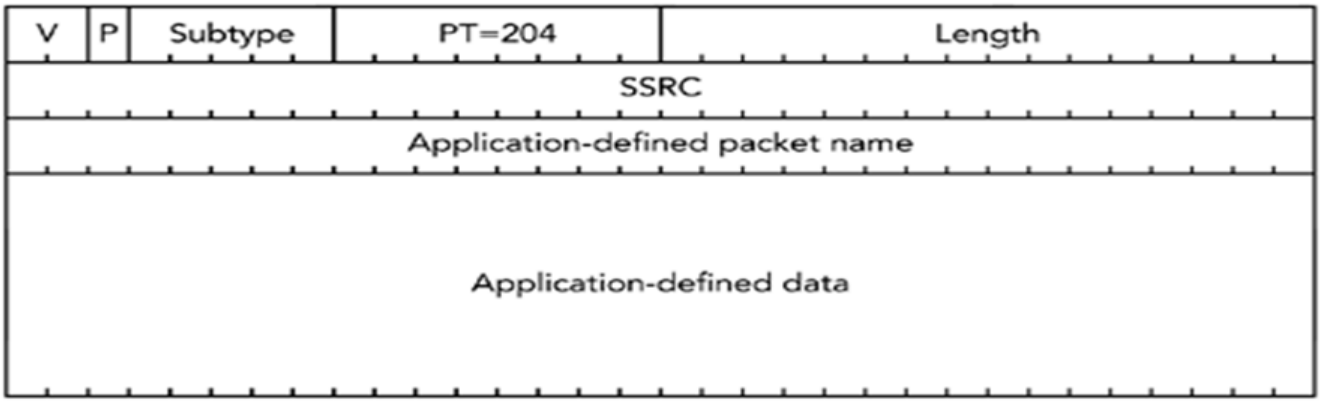


以下是BYE的报文实例

```
⊞ Real-time Transport Control Protocol (Receiver Report)
⊟ Real-time Transport Control Protocol (Goodbye)
  ⊟ [Stream setup by SDP (frame 252)]
      [Setup frame: 252]
      [Setup Method: SDP]
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 0001 = Source count: 1
    Packet type: Goodbye (203)
    Length: 1 (8 bytes)
    Identifier: 0x60f716c1 (1626805953)
  [RTCP frame length check: OK - 40 bytes]
```

RTCP报文格式-- APP报文格式

APP数据报文允许应用定义扩展。APP分组用来进行一些非标准RTCP扩展，或者进行一些新特性的试验，等到试验成熟，就可以注册成一种新的类型。应用实现对不认识的APP应该予以忽略。



RTCP报文格式-- APP报告实例

Application-defined packet name使用4个字符的标识符，唯一标识这个扩展。每个字符使用ASCII编码格式，区分大小写。

```
⊟ Real-time Transport Control Protocol (Application specific)
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    Subtype: 0
    Packet type: Application specific (204)
    Length: 44
    Identifier: 30508
    Name (ASCII): QTSS
    Application specific data: 40B402060000000772720004003O7FB8646C0002001E7072...
```

RTSP play同步

`RTP`包中的时间戳字段是说明数据包时间的同步信息，是数据能以正确的时间顺序恢复的关键。时间戳的值给出了分组中数据的第一个字节的采样时间。为了计算各个数据流的播放时间以及同步处理，仅有`RTP`包中的时间戳信息是不够的。

在整个播放过程中，包括这样几种时间:

1. `RTP`包中的`rtptime`
2. `PLAY`请求的`Response`中的`rtp time`和`npt`
3. `RTCP`的`SR`(`Sender Report`)中的`rtp`和`ntp`时间戳对

一、时间戳映射关系

首先介绍`PLAY`请求的`Response`里的两个域:

(1) `npt`

`npt`顾名思义`Normal PLay Time`，正常播放时间，指出了流相对与播放开始时的绝对位置。播放开始时的之间定义为`0.0s`。这个特殊的常量`now`被定义为现场事件的当前时刻。"`now`"常数允许客户端请求接收实时反馈而不是存储或者延时的版本。因为对于这种情况而言,既没有绝对时间,也没有`0`时间,所以需要该参数。

(2) `rtptime`

`rtptime`是发送`PLAY`请求后将收到的第一个`RTP`包的时间戳值。

`npt`和`rtptime`的区别在于`npt`是影片开始的相对时间，而`rtptime`是会话开始的相对时间。因此在`client`端，需要对这两者进行`map`处理。

在`client`端计算播放时间戳的公式如下:

nptUs = (current_rtpTime – start_rtptime) / scale + srart_npt

其中:

`start_rtptime`与**`start_npt`**分别是**`PLAY`**请求的**`Response`**中的`rtptime`和`npt`。

`current_rtpTime`是当前收到的`RTP`包头中的`rtptime`。

`scale`为`PLAY`请求的`Response`中的`scale`值。在正常播放的情况下为`1`，快速播放时大于`1`，当处于反向扫描模式时小于`-1`.

二、媒体间同步方法（不同设备的同步）

上面的处理仅仅实现了媒体内的同步，在实现媒体间同步时，还需要进行其他的处理工作。这就需要用到**`RTCP`**的**`SR`**(**`Sender Report`**)。在`SR`中包含一个**<`rtp`，`ntp`>**时间戳对，通过这个时间戳对可以将音频和视频准确的定位到一个绝对时间轴上。

<`rtp`，`ntp`>时间戳对的必要性在于不同流的`RTP`时间戳有不同的随机偏移量，因此无法直接进行同步。

<`rtp`，`ntp`>中的`ntp`是网络时间协议格式表示的绝对时间值。

<`rtp`，`ntp`>中的`RTP`与数据包中的**`RTP`**时间戳具有相同的单位和偏移量，但在大多数情况下此时间戳不等于任何临近的**`RTP`**包中的时间戳。

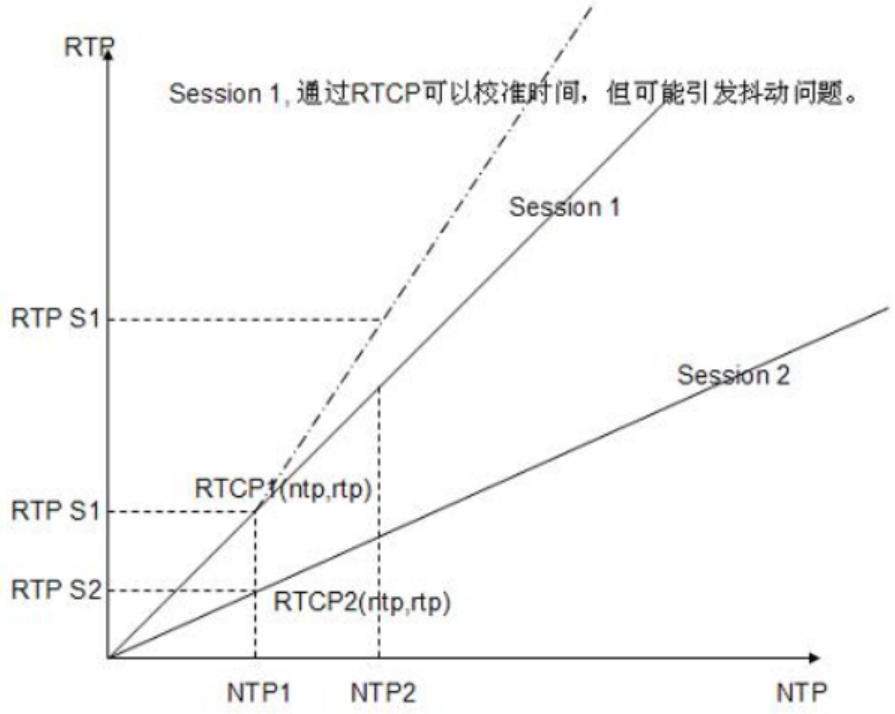
根据不同**`stream`**中的**<`rtp`，`ntp`>**时间戳对，可以将一个`stream`中的时间戳值映射为另一个**`stream`**的时间戳值，从而实现媒体间的同步。

RTCP同步

RTP支持传送不同codec的steaming，不同codec的clock rate的也不一样，不同的media之间需要依靠RTCP进行同步。这里简单介绍一下他们的机制。

在每个RTCP SR包中对应有一个RTP时间和一个NTP时间，它表达的意思很明确，那就是这个RTP时间对应的绝对时间， 不同media的RTP时间尽管不同，但可以通过NTP时间映射到同一个时间轴上，从而实现同步。

如下图所示，RTP session 1 send H264 使用90,000HZ，而RTP session 2 send G.711 使用8,000HZ:



也就是是说有3个时间轴，音频时间轴，视频时间轴，ntp时间轴。

音视频的时间轴的单位都是各自的采样率，需要除以采样率才能取得单位为秒的时间单位。

有两个rtcp流，分别为音/视频的，其中有一个当前的音频的timestamp和一个ntp的timestamp。这两个值是在不同轴上的相同时间点，即音/视频轴和ntp轴的重合点。使用这个值可以使音视频轴同步。

当拿到音频NTP时间 （Tan），音频RTP时间（Tar），视频NTP时间（Tvn），视频RTP时间（Tvr），就可以计算音视频时间轴的差距D:

假设使用音频为主轴，视频向音频对齐。D = (Tar–Tvr) – (Tan – Tvn);

新的视频时间戳为Tar = Tar + D;

在rtcp的sr单元中有32位的MSW和32位的LSW。MSW的单位为秒，而LSW的单位为232 picoseconds。1皮秒为1/10^12秒。LSW转为us的公式为(LSW*10^12/2^32)/1000000;