FFmpeg4.3 系列 24

RTP 及 jrtplib 发送 H.264 码流实战

RTP 协议简介

RTP:(Real-time Transport Protocol)

是用于 Internet 上针对多媒体数据流的一种传输层协议.RTP 协议和 RTP 控制**协议 RTCP** 一起使用，而且它是建立在 UDP 协议上的.

RTP+RTCP::兄弟协议，他们两个端口是紧挨着的。

Rtp:10240, rtcp:10241

RTP 不像 http 和 ftp 可完整的下载整个影视文件，它是以固定的数据率在网络上发送数据，客户端也是按照这种速度观看影视文件，当影视画面播放过后，就不可以再重复播放，除非重新向服务器端要求数据。

实时传输协议 RTP（Real-Time Transport Protocol）：

RTP 是针对 Internet 上多媒体数据流的一个传输协议，由 IETF(Internet 工程任务组)作为 RFC1889 发布。RTP 被定义为在一对一或一对多的传输情况下工作，其目的是提供时间信息和实现流同步。RTP 的典型应用**建立在 UDP** 上，但也可以在 **TCP** 或 ATM 等其他协议之上工作。

RTP 本身只保证实时数据的传输，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。

RTP 工作机制

威胁多媒体数据传输的一个尖锐的问题就是不可预料数据到达时间。

但是流媒体的传输是需要数据的适时的到达用以播放和回放。

rtp 协议就是提供了时间标签,序列号以及其它的结构用于控制实时数据的流放。在流的概念中”时间标签”是最重要的信息。发送端依照即时的采样在数据包里隐蔽的设置了时间标签。在接受端收到数据包后,就依照时间标签按照正确的速率恢复成原始的适时的数据。不同的媒体格式调时属性是不一样的。

但是 rtp 本身并不负责同步，rtp 只是用于传输（音视频或其它数据）的协议，为了简化运输处理，提高该层的效率。将部分运输层协议功能（比如流量控制）上移到应用层完成。同步就是属于应用层协议完成的。它没有运输层协议的完整功能，不提供任何机制来保证实时地传输数据，不支持资源预留，也不保证服务质量。rtp 报文甚至不包括长度和报文边界的描述。同时 rtp 协议的数据报文和控制报文的使用相邻的不同端口，这样大大提高了协议的灵活性和处理的简单性。

rtp 协议和 udp 二者共同完成运输层协议功能。

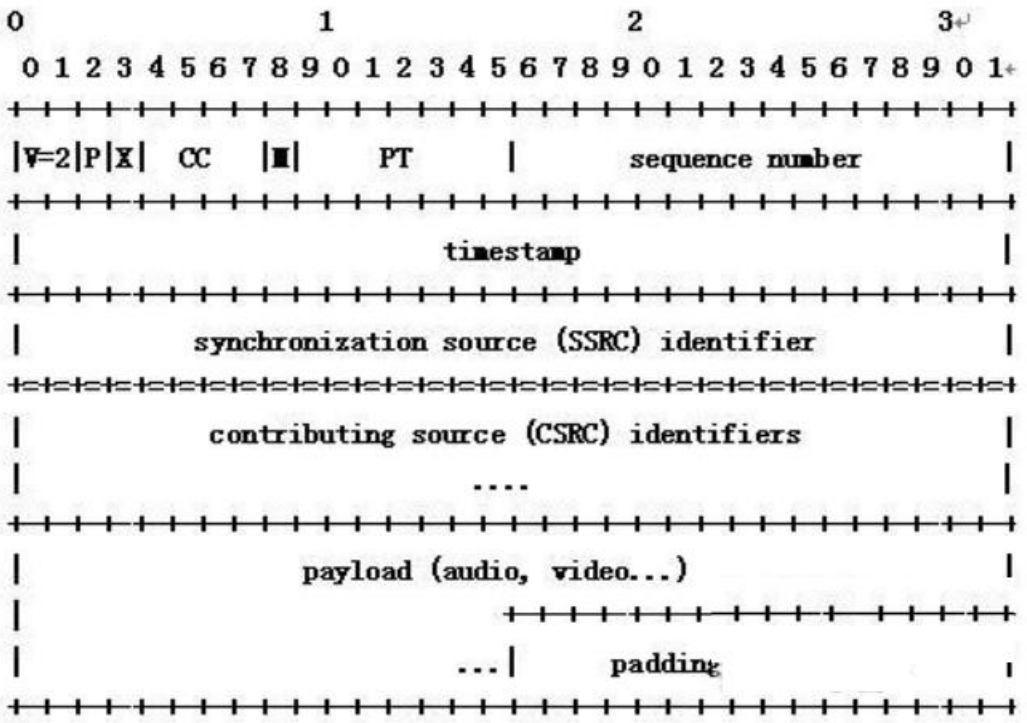
udp 协议只是传输数据包，不管数据包传输的时间顺序。

rtp 的协议数据单元是用 udp 分组来承载的。在承载 rtp 数据包的时候，有时候一帧数据被分割成几个包具有相同的时间标签，则可以知道时间标签并不是必须的。而 udp 的多路复用让 rtp 协议利用支持显式的多点投递，可以满足多媒体会话的需求。

rtp 协议虽然是传输层协议但是它没有作为 osi 体系结构中单独的一层来实现。rtp 协议通常根据一个具体的应用来提供服务，rtp 只提供协议框架，开发者可以根据应用的具体要求对协议进行充分的扩展。

RTP 协议的报文结构

RTP 头格式



开始 12 个八进制出现在每个 RTP 包中，而 CSRC 标识列表仅出现在混合器插入时。各段含义如下：

①版本（V）

2 位，标识 RTP 版本。

②填充标识（P）:padding

1 位，如设置填充位，在包尾将包含附加填充字，它不属于有效载荷。填充的最后一个八进制包含应该忽略的八进制计数。某些加密算法需要固定大小的填充字，或为在底层协议数据单元中携带几个 RTP 包。

③扩展（X）

1 位，如设置扩展位，固定头后跟一个头扩展。

④CSRC 计数（CC）

4 位，CSRC 计数包括紧接在固定头后 CSRC 标识符个数。

⑤标记（M）

1 位，标记解释由设置定义，目的在于允许重要事件在包流中标记出来。设置可定义其他标示位，或通过改变位数量来指定没有标记位。

⑥载荷类型（**PT**）

7 位，记录后面资料使用哪种 Codec ， receiver 端找出相应的 decoder 解碼出來。

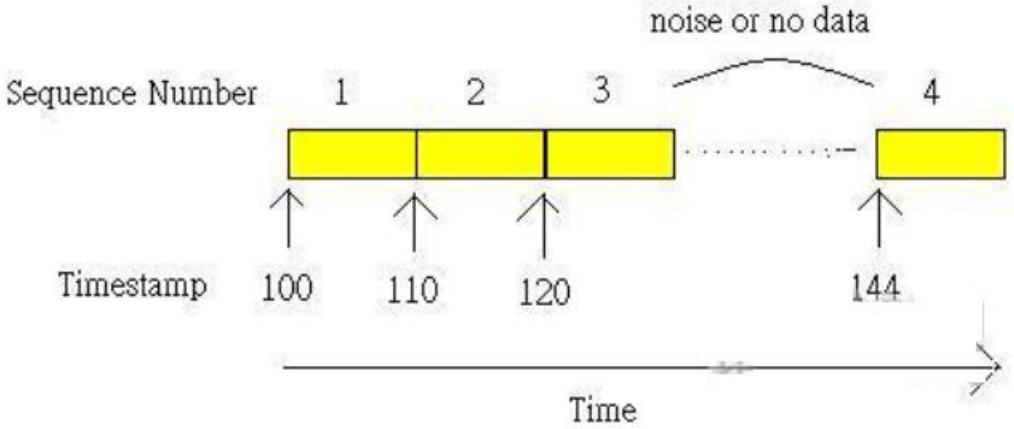
Payload Type	Codec
0	PCM μ -Law
8	PCM-A Law
9	G..722 audio codec
4	G..723 audio codec
15	G..728 audio codec
18	G..729 audio codec
34	G..763 audio codec
31	G..761 audio codec

⑦系列号(sn)

16 位，系列号随每个 RTP 数据包而增加 1，由接收者用来探测包损失。系列号初值是随机的，使对加密的文本攻击更加困难。

⑧时标(ts:timestamp）

32 位，时标反映 RTP 数据包中第一个八进制数的采样时刻，采样时刻必须从单调、线性增加的时钟导出，以允许同步与抖动计算。时标可以让 receiver 端知道在正确的时间将资料播放出来。



由上图可知，如果只有系列号，并不能完整按照顺序的将 data 播放出来，因为如果 data 中间有一段是没有资料的，只有系列号的话会造成错误，需搭配上让它知道在哪个时间将

data 正确播放出来，如此我们才能播放出正确无误的信息。

⑨SSRC

32 位，**SSRC 段标识同步源**。此标识不是随机选择的，目的在于使同一 RTP 包连接中没有两个同步源有相同的 SSRC 标识。尽管多个源选择同一个标识的概率很低，所有 RTP 实现都必须探测并解决冲突。如源改变源传输地址，也必须选择一个新 SSRC 标识以避免插入成环行源。

⑩CSRC 列表

0 到 15 项，每项 32 位。CSRC 列表表示包内的对载荷起作用的源。标识数量由 CC 段给出。如超出 15 个作用源，也仅标识 15 个。CSRC 标识由混合器插入，采用作用源的 SSRC 标识。提供信源用来标识对一个 RTP 混合器产生的新包有贡献的所有 RTP 包的源。是指当混合器接收到一个或多个同步信源的 RTP 报文后，经过混合处理产生一个新的组合 RTP 报文，并把混合器作为组合 RTP 报文的 SSRC，将原来所有的 SSRC 都作为 CSRC 传送给接收者，是接受者知道组成组合报文的各个 SSRC。

数据案例分析

2b: 1000 0000
2b:1110 0000
Vpx ccmpt

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 80 e0 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；

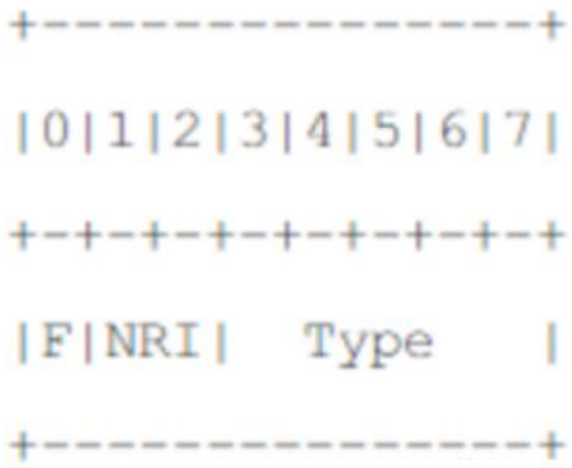
110 0000　　是 PT；

RTP 载荷 H264 码流

- **RTP 载荷 H264 码流：红色 RTP 协议头，黄色 H264 码流**

RTP头（12byte + N CSRC）	NAL头	RBSP

- **RTP 头后是 RTP 载荷，RTP 载荷第一个字节格式跟 NALU 头一样：**



- **F 和 NRI 也跟 NALU 头一样，只有 Type 有些不一样：拓展 24 － 31**

0	没有定义	
1-23	NAL 单元	单个 NAL 单元包
24	STAP-A	单一时间的组合包
25	STAP-B	单一时间的组合包
26	MTAP16	多个时间的组合包
27	MTAP24	多个时间的组合包
28	**FU-A**	**分片的单元（fragment unit ）**
29	FU-B	分片的单元
30-31	没有定义	

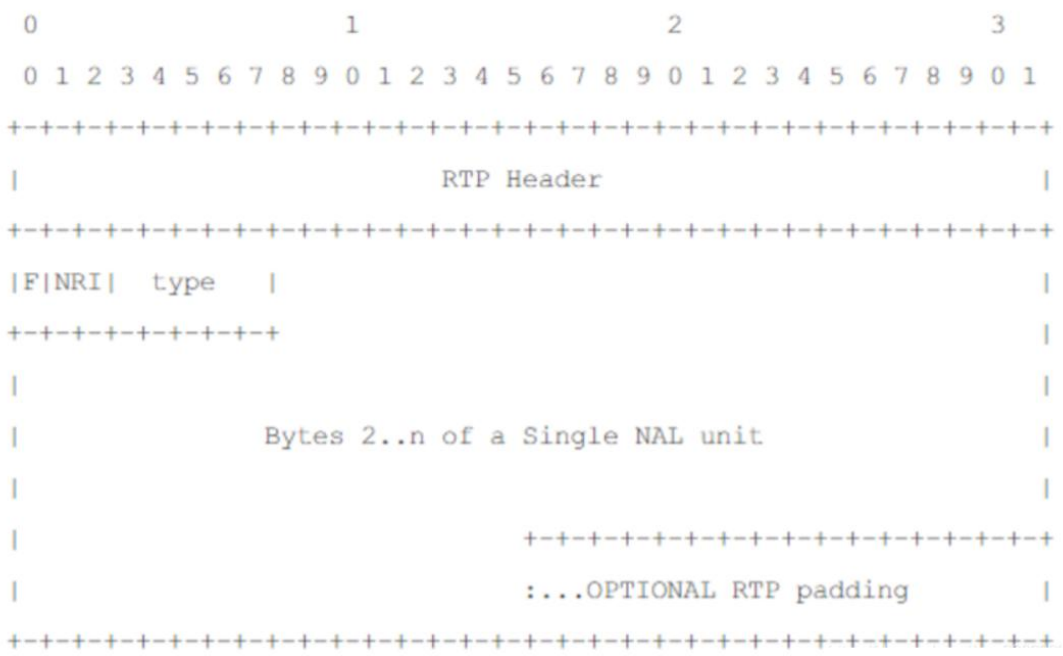
1)　单个 NAL 单元包：荷载中只包含一个 NAL 单元。NAL 头类型域等于原始 NAL 单元（NALU）类型，即 Type 在范围 1 到 23 之间。

2）组合包：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（Type）分别是 24 25 26 27

3）分片包：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（Type）28、 29 标识

常用的打包时的分包规则：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

单个 NAL 单元包格式



12（或更多）字节的 RTP 头后面的就是音视频数据，比较简单。
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. 以下即是一个被打包进 rtp 的 NALU 单元(不包含 rtp 头)，第一个字节是 NALU 头：

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

定义在此的 NAL 单元包必须只包含一个。RTP 序号必须符合 NAL 单元的解码顺序。这种情况下，NAL 单元的第一字节和 RTP 荷载头第一个字节重合。如上图所示。

对于一个原始 H264 的 NALU 单元常由[start code] [NALU Header] [NALU Payload]三部分组成，其中 start code 用于标识这是一个 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 单元载荷。

打包时去除“00 00 01”或“00 00 00 01”的开始码，把其他数据封装成 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 ...] ///H.264.Annexb

F、NRI、Type（1、2、5）：8bits：1 字节

0110 0111

Type--》 7：SPS、8：PPS

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 载荷.

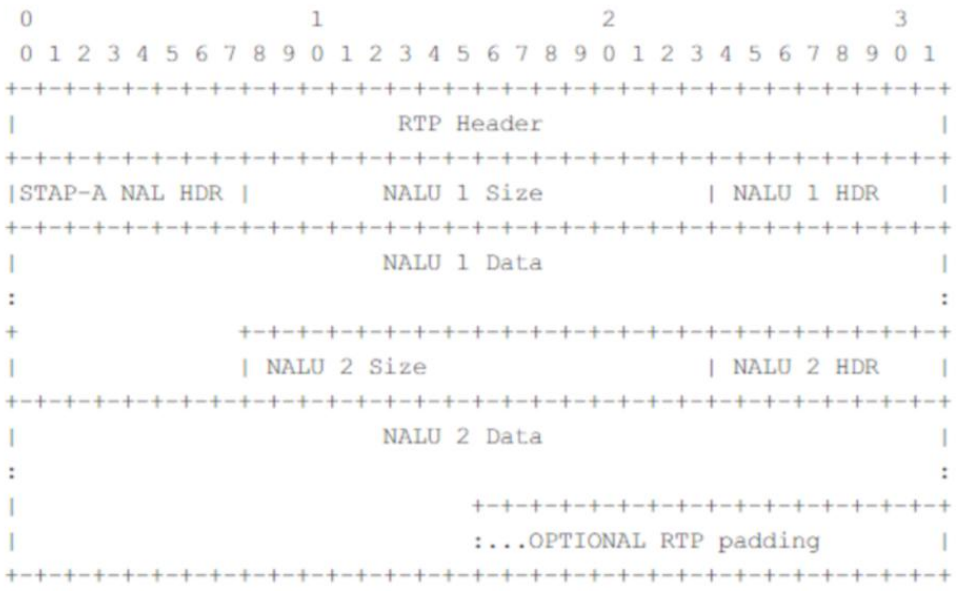
封装成 RTP 包将如下:

[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.

组合封包格式

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



这种模式下，有多个NALU载荷，多个NALU头。

例：

如有一个 H.264 的 NALU 是这样的:

[**00 00 00 01 67** 42 A0 1E 23 56 0E 2F ...] //67: sps

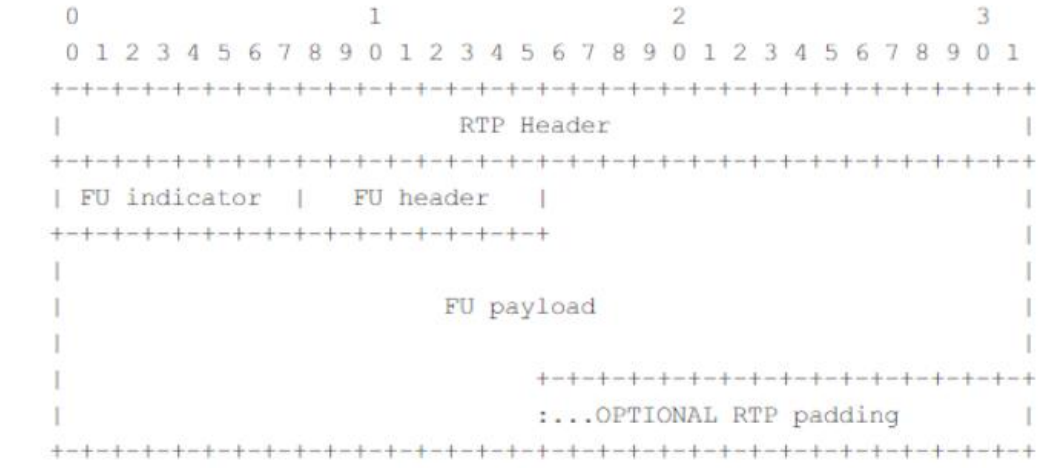
[**00 00 00 01 68** 42 B0 12 58 6A D4 FF ...]//68:pps

封装成 RTP 包将如下:

[RTP Header] [78 (STAP-A 头，占用 1 个字节)] [第一个 NALU 长度 (占用两个字节)] [67 42 A0

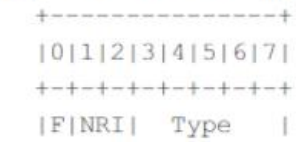
1E 23 56 0E 2F] [第二个 NALU 长度 (占用两个字节)] [68 42 B0 12 58 6A D4 FF ...]
学习网站：http://www.hellotongtong.com/
福优学苑：【 QQ 咨询：3212001984】
加微信可以提供远程服务,微信服务二维码(13661137824)：

分片单元（FU-A）

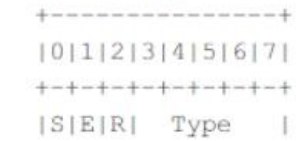


FU-A RTP 载荷格式

其中 FU indicator 8位格式为:

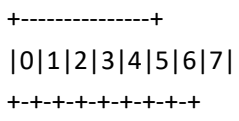


FU header 格式:



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)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。

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：FU indicator 有以下格式：



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。

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**

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	-

关于时间戳，需要注意的是 **h264 的采样率为 90000HZ**(被标准固定死的)，因此时间戳的单位为 1(秒)/90000，因此如果当前视频帧率为 25fps，那时间戳间隔或者说增量应该为 3600，如果帧率为 30fps，则增量为 3000，以此类推。

关于 h264 拆包，按照 FU-A 方式说明：

1）第一个 FU-A 包的 FU indicator：F 应该为当前 NALU 头的 F，而 NRI 应该为当前 NALU 头的 NRI，Type 则等于 28，表明它是 FU-A 包。FU header 生成方法：S = 1，E = 0，R = 0，Type 则等于 NALU 头中的 Type。

2）后续的 N 个 FU-A 包的 FU indicator 和第一个是完全一样的，如果不是最后一个包，则 FU header 应该为：S = 0，E = 0，R = 0，Type 等于 NALU 头中的 Type。

3）最后一个 FU-A 包 FU header 应该为：S = 0，E = 1，R = 0，Type 等于 NALU 头中的 Type。

因此总结就是：同一个 NALU 分包后的 **FU indicator 头**是完全一致的，FU header 只有 S 以及 E 位有区别，分别标记开始和结束，它们的 RTP 分包的序列号应该是依次递增的，并且它们的时间戳必须一致，而负载数据为 NALU 包去掉 1 个字节的 NALU 头后对剩余数据的拆分，这点很关键，你可以认为 NALU 头被拆分成了 FU indicator 和 FU header，所以不再需要 1 字节的 NALU 头了。

关于 SPS 以及 PPS，配置帧的传输我采用了先发 SPS，再发送 PPS，并使用同样的时间戳，或者按照正常时间戳增量再或者组包发送的形式处理貌似都可以，看播放器怎么解码了，另外提一下，如果我们使用 vlc 进行播放的话，可以在 sdp 文件中设置 SPS 以及 PPS，这样就可以不用发送它们了。

RTP 封装 H.264

H264 结构中，一个视频图像编码后的数据叫做一帧，一帧由一个片（slice）或多个片组成，一个片由一个或多个宏块（MB）组成。

h264 码流简介

H264 编码过程中的三种不同的数据形式：

- SODB　　数据比特串 ----＞最原始的编码数据，即 **VCL 数据**；
- RBSP　　原始字节序列载荷 ----＞在 SODB 的后面填加了**结尾比特**（RBSP trailing bits 一个 bit“1”）若干比特“0”,以便字节对齐；
- EBSP　　扩展字节序列载荷 ----＞ 在 RBSP 基础上填加了仿校验字节（**0X03**）它的原因是： 在 NALU 加到 Annexb 上时，需要添加每组 NALU 之前的开始码 StartCodePrefix，如果该 NALU 对应的 slice 为一帧的开始则用 4 位字节表示，ox00000001,否则用 3 位字节表示 ox000001（是一帧的一部分）。另外，为了使 NALU 主体中不包括与开始码相冲突的，在编码时，每遇到两个字节连续为 0，就插入一个字节的 0x03。解码时将 0x03 去掉。也称为**脱壳**操作。

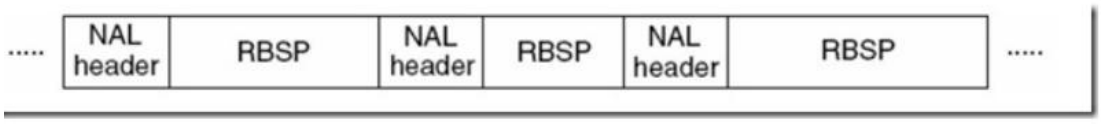
NAL 全称 Network Abstract Layer，即网络抽象层。在 H.264/AVC 视频编码标准中，整个系统框架被分为了两个层面：视频编码层面（VCL）和网络抽象层面（NAL）。

其中，前者负责有效表示视频数据的内容，而后者则负责格式化数据并提供头信息，以保证数据适合各种信道和存储介质上的传输。

NAL 单元是 NAL 的基本语法结构，它包含**一个字节的头信息**和一系列来自 VCL 的称为原始字节序列载荷（RBSP）的字节流。

- **帧格式**

H264 在网络传输的是 NALU，NALU 的结构是：NAL 头+RBSP，实际传输中的数据流如图所示：



H264 帧由 NALU 头和 NALU 主体组成。
NALU 头由一个字节组成,它的语法如下:

```
+---------------+
|0|1|2|3|4|5|6|7|
+-+-+-+-+-+-+-+-+
|F|NRI|  Type   |
+---------------+
```

- **F: 1 个比特**

  forbidden_zero_bit. 在 H.264 规范中规定了这一位必须为 0.

- **NRI: 2 个比特**

  nal_ref_idc. 取 00~11,似乎指示这个 NALU 的重要性,如 00 的 NALU 解码器可以丢弃它而不影响图像的回放,0～3，取值越大，表示当前 NAL 越重要，需要优先受到保护。如果当前 NAL 是属于参考帧的片，或是序列参数集，或是图像参数集这些重要的单位时，本句法元素必需大于 0。

- **Type: 5 个比特**

  标识 NAL 单元中的 RBSP 数据类型，其中，nal_unit_type 为 1， 2， 3， 4， 5 的 NAL 单元称为 VCL 的 NAL 单元，其他类型的 NAL 单元为非 VCL 的 NAL 单元。

nal_unit_type. 这个 NALU 单元的类型,1～12 由 H.264 使用，24～31 由 H.264 以外的应用使用,简述如下:

0	没有定义
1-23	NAL 单元 单个 NAL 单元包
1	不分区，非 IDR 图像的片
2	片分区 A
3	片分区 B
4	片分区 C
5	**IDR 图像中的片**
6	补充增强信息单元（SEI）
7	**SPS**
8	**PPS**
9	序列结束
10	序列结束
11	码流借宿

12 填充
13-23 保留

24 STAP-A 单一时间的组合包
25 STAP-B 单一时间的组合包
26 MTAP16 多个时间的组合包
27 MTAP24 多个时间的组合包
28 FU-A 分片的单元
29 FU-B 分片的单元
30-31 没有定义

由于 NAL 的语法中没有给出长度信息，实际的传输、存储系统需要增加额外的头实现各个 NAL 单元的定界。

其中，AVI 文件和 MPEG TS 广播流采取的是**字节流的语法格式**，即在 NAL 单元之前增加 **0x00000001 的同步码**，则从 AVI 文件或 MPEG TS PES 包中读出的一个 H.264 视频帧以下面的形式存在：

00 00 00 01 06 ... **00 00 00 01** 67 ... **00 00 00 01** 68 ... **00 00 00 01** 65 ...
SEI 信息 SPS PPS IDR Slice

而对于 MP4 文件，NAL 单元之前没有同步码，**却有若干字节的长度码**，来表示 NAL 单元的长度，这个长度码所占用的字节数由 MP4 文件头给出；此外，**从 MP4 读出来的视频帧不包含 PPS 和 SPS**，这些信息位于 **MP4 的文件头**中，解析器必须在打开文件的时候就获取它们。从 MP4 文件读出的一个 H.264 帧往往是下面的形式（假设长度码为 2 字节）：

00 19 06 [... 25 字节...] 24 aa 65 [... 9386 字节...]
SEI 信息 IDR Slice
学习网站：http://www.hellotongtong.com/
福优学苑：【 QQ 咨询：3212001984】
加微信可以提供远程服务,微信服务二维码(13661137824)：

ffmpeg -i test.mp4 -codec copy -bsf: h264_mp4toannexb -f h264 test.h264

h264 NALU 格式

网络抽象层单元类型 (NALU):

一个原始的 H.264 NALU 单元常由 [**Start Code**] [**NALU Header**] [**NALU Payload**] 三部分组成，形如 00 00 00 01 **67** 42 A0 1E 23， 则 67 是 nalu header, 42 之后的是 payload(有效的原始视频数据). 有些代码中的 nal_octet 变量指的就是 nalu header 字节。

Octet：字节、

NALU 头(NALU Header)由一个字节组成,它的语法如下:

```
+---------------+
|0|1|2|3|4|5|6|7|
+-+-+-+-+-+-+-+-+
|F|NRI|  Type   |
+---------------+
```

F: 1 个比特.
forbidden_zero_bit. 在 H.264 规范中规定了这一位必须为 0.

NRI: 2 个比特.
nal_ref_idc. 取 00~11,似乎指示这个 NALU 的重要性,如 00 的 NALU 解码器可以丢弃它而不影响图像的回放.

Type: 5 个比特.
nal_unit_type. 这个 NALU 单元的类型.简述如下:

0	没有定义	
1-23	NAL 单元	单个 NAL 单元包
24	STAP-A	单一时间的组合包
25	STAP-B	单一时间的组合包
26	MTAP16	多个时间的组合包
27	MTAP24	多个时间的组合包
28	**FU-A**	**分片的单元**
29	FU-B	分片的单元
30-31	没有定义	

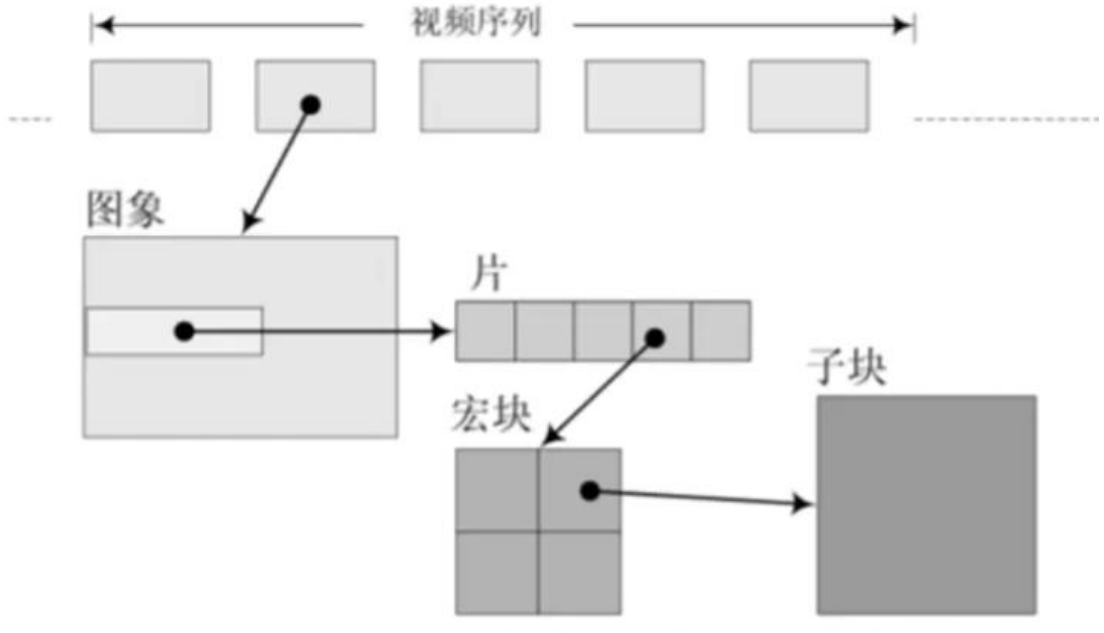
- **h264 仅用 1-23,**
- **24 以后的用在 RTP H264 负载类型头中(即 24 以后在 rtp 中使用)**

H.264 从 1999 年开始，到 2003 年形成草案，最后在 2007 年定稿有待核实。在 ITU 的标准里称为 H.264，在 MPEG 的标准里是 MPEG-4 的一个组成部分–MPEG-4 Part 10，又叫 Advanced Video Codec，因此常常称为 MPEG-4 AVC 或直接叫 AVC。
NALU（Network Abstract Layer Unit）全称为网络抽象层单元。
对于视频文件来说，视频由单张图片帧所组成，比如每秒 25 帧，但是图片帧的像素块之间存在相似性，因此视频帧图像可以进行图像压缩；H264 采用了 16*16 的分块大小，对视频帧图像进行相似比较和压缩编码。
H264 使用帧内压缩和帧间压缩的方式提高编码压缩率；H264 采用了独特的 I 帧、P 帧和 B 帧策略来实现，连续帧之间的压缩。
H264 帧的分类：

帧的分类	全称	作用
I 帧	帧内编码帧（intra picture）	I 帧通常是每个 GOP（MPEG 所使用的一种视频压缩技术）的第一个帧，经过适度地压缩，做为随机访问的参考点，可以当成图象。I 帧可以看成是一个图像经过压缩后的产物。自身可以通过视频解压算法解压成一张单独的完整的图片。
P 帧	前向预测编码帧（predictive-frame）	通过充分将低于图像序列中前面已编码帧的时间冗余信息来压缩传输数据量的编码图像，也叫预测帧。需要参考其前面的一个 I frame 或者 P frame 来生成一张完整的图片。
B 帧	双向预测帧（bi-directional interpolated prediction frame）	既考虑与源图像序列前面已编码帧，也顾及源图像序列后面已编码帧之间的时间冗余信息来压缩传输数据量的编码图像，也叫双向预测帧。则要参考其前一个 I 或者 P 帧及其后面的一个 P 帧来生成一张完整的图片。

- **H264 编码结构**

H264 每一帧的结构由图片组(GOP， group of pictures)、图片(pictrue)、片(Slice)、宏块(Macroblock)、子块(subblock)五个层次组成。



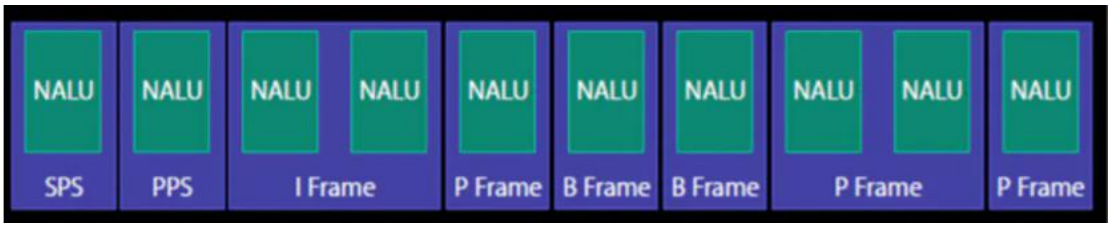
一个GOP视频序列图

- **一个 GOP 视频序列图**

GOP （图像组）主要用作形容一个 **IDR**（Instantaneous Decoding Refresh，即时解码刷新）帧 到下一个 IDR 帧之间的间隔了多少个帧。

一个图像组的第一个图像叫做 **IDR 图像**，IDR 图像都是 I 帧图像。

IDR 的核心作用是，是为了**解码的重同步**，当解码器解码到 IDR 图像时，立即将参考帧队列清空，将已解码的数据全部输出或抛弃，重新查找参数集，开始一个新的序列。这样，如果前一个序列出现重大错误，在这里可以获得重新同步的机会。IDR 图像之后的图像永远不会使用IDR 之前的图像的数据来解码。



NALU 结构图

- SPS:序列参数集（Sequence parameter set）SPS 中保存了一组编码视频序列(Coded video sequence)的全局参数。
- PPS：图像参数集(Picture parameter set) ，对应的是一个序列中某一幅图像或者某几幅图像的参数。
- I 帧：帧内编码帧，可独立解码生成完整的图片。
- P 帧: 前向预测编码帧，需要参考其前面的一个 I 或者 B 来生成一张完整的图片。
- B 帧: 双向预测内插编码帧，则要参考其前一个 I 或者 P 帧及其后面的一个 P 帧来生成一张完整的图片。
- **发 I 帧之前，至少要发一次 SPS 和 PPS。**

H.264 原始码流(裸流)是由一个接一个 NALU 组成，它的功能分为两层，VCL(video codec layer，视频编码层)和 NAL(Network Abstract Layer，网络抽象层)。

VCL：包括核心压缩引擎和块、宏块和片的语法级别定义，设计目标是尽可能地独立于网络进行高效的编码。

NAL：负责将 VCL 产生的比特字符串适配到各种各样的网络和多元环境中，覆盖了所有片级以上的语法级别。

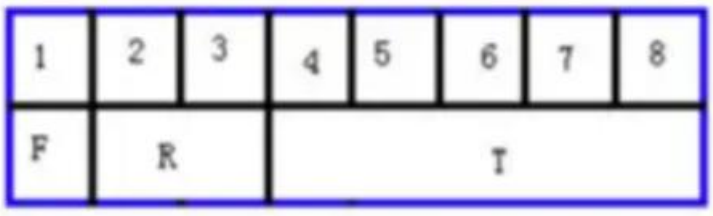
在 VCL 进行数据传输或存储之前，这些编码的 VCL 数据，被映射或封装进 NAL 单元（NALU）。一个 NALU = 一组对应于视频编码的 NALU 头部信息 + 一个原始字节序列负荷(RBSP，Raw Byte Sequence Payload).

NALU 结构单元的主体结构如下所示；一个原始的 H.264 NALU 单元通常由[StartCode] [NALU Header] [NALU Payload]三部分组成，其中 Start Code 用于标示这是一个 NALU 的开始，必须是"00 00 00 01" 或"00 00 01"，除此之外基本相当于一个 NAL header + RBSP;

FFmpeg 解复用后，MP4、FLV、TS文件读取出来的 packet 是不带 **startcode**,值得注意的是 TS 封装格式无 start code、SPS、PPS 也是可以正常播放的。

每个 NALU 是一个一定语法元素的可变长字节字符串，包括包含一个字节的头信息（用来表示数据类型），以及若干整数字节的负荷数据。

NALU 头信息（1Byte）：



NALU head

位域	位宽	说明
T	5 bit	负荷数据类型，表示 NALU 单元的类型,1～12 由 H.264 使用，24～31 由 H.264 以外的应用使用
R	2	重要性指示位，取值越大，表示当前 NAL 越重要，需要优先受到保护,如果当前 NAL 是属于参考帧的片、序列参数集或图像参数集这些重要的单位时，本句法元素必需大于 0
F	1	禁止位，H.264 规范中规定了这一位必须为 0.

H.264 标准指出，当数据流是储存在介质上时，在每个 NALU 前添加起始码：0x000001 或 0x00000001，用来指示一个 NALU 的起始位置。

在这样的机制下，在码流中检测起始码，作为一个 NALU 得起始标识，当检测到下一个起始码时，当前 NALU 结束

3 字节的 0x000001 只有一种场合下使用，就是一个完整的帧被编为多个 slice（片）的时候，包含这些 slice 的 NALU 使用3 字节起始码。其余场合都是 4 字节 0x00000001 的。

- **H264 annexb 模式**

H264 有两种封装格式：

annexb 模式，传统模式，有 startcode，SPS 和 PPS 是在 ES 中

mp4 模式，一般 mp4 mkv 都是 mp4 模式，container 封装中没有 startcode,SPS 和 PPS 以及其他信息，每一个 frame **前面 4 个字节**表示该 frame 的长度

ffmpeg -i test.mp4 -codec copy -bsf: h264_mp4toannexb -f h264 test.h264

开源库 jrtplib 的源码编译及环境搭建

ffmpeg -i test.mp4 -codec copy -bsf: h264_mp4toannexb -f h264 test.h264

源码下载地址：

https://research.edm.uhasselt.be/jori/page/CS/Jrtplib.html

https://github.com/j0r1/JRTPLIB

https://github.com/j0r1/JThread

Windows 编译：

Cmake:生成解决方案、编译、配置环境。

发现编译运行后在 jthread 项目下产生 **jthreadconfig.h**

发现编译运行后在 jrtplib 项目下产生 **rtpconfig**.h 和 **rtptypes**.h

Linux 编译：

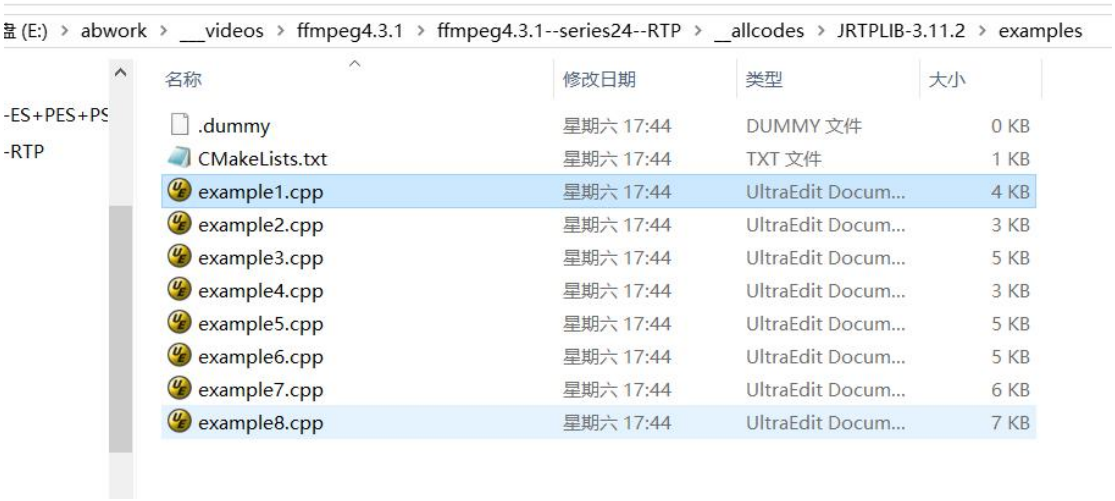
cmake CMakeLists.txt

make

make install

开源库 jrtplib 的官方案例详解与剖析

搭建 jrtplib 的开发环境



发现编译运行后在 jthread 项目下产生 jthreadconfig.h
发现编译运行后在 jrtplib 项目下产生 rtpconfig.h 和 rtptypes.h

- **坑：windows 下的网络库需要：手工添加头文件、库文件**

<winsock2.h>
ws2_32.lib

```
#pragma comment(lib, "ws2_32.lib")
```

用
LNK201 无法解析的外部符号 __imp__inet_addr@4，该符号在函数 _main 中被引用
LNK201 无法解析的外部符号 __imp__ntohl@4，该符号在函数 _main 中被引用
LNK201 无法解析的外部符号 __imp__WSAStartup@8，该符号在函数 _main 中被引用
LNK201 无法解析的外部符号 __imp__WSACleanup@0，该符号在函数 _main 中被引用
LNK112 4 个无法解析的外部命令

学习网站：http://www.hellotongtong.com/
福优学苑：【 QQ 咨询：3212001984】
加微信可以提供远程服务,微信服务二维码(13661137824)：

Example1:使用 jrtplib 发送数据包

在 jrtplib 的使用的时候，有下面的几点需要被别注意：

- **端口不能是奇数，否者运行时会出现错误：**

RTPSession

RTPUDPv4TransmissionParams

RTPSessionParams

RTPIPv4Address

RTPPacket

SendPacket(...)

BeginDataAccess(...)

GetNextPacket(...)

GotoNextSourceWithData(...)

EndDataAccess(...)

```
/** High level class for using RTP.
 *  For most RTP based applications, the RTPSession class will probably be the one to use. It handles
 *  the RTCP part completely internally, so the user can focus on sending and receiving the actual data.
 *  In case you want to use SRTP, you should create an RTPSecureSession derived class.
 *  \note The RTPSession class is not meant to be thread safe. The user should use some kind of locking
 *        mechanism to prevent different threads from using the same RTPSession instance.
 */
```

创建 UDP 程序，自己收包

```
// send the packet
status = sess.SendPacket((void *)"1234567890", 10, 0, false, 10);
```

socket(....)

bind(...)

receivefrom(....)

sendto(...)

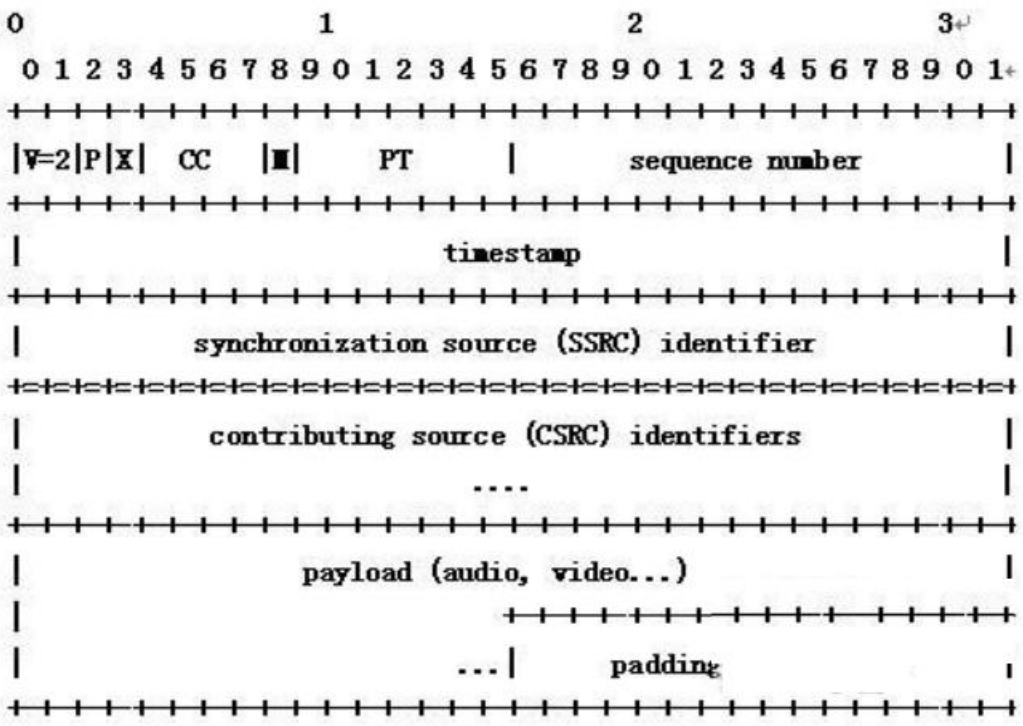
自己发送，自己接收，并分析 RTPPacket

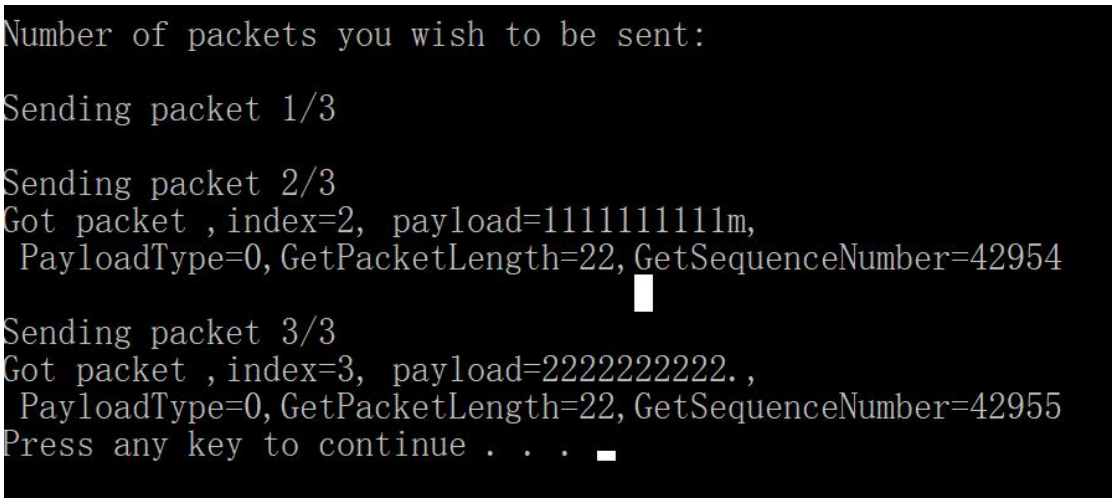
```
#include "jrtplib/rtppacket.h"
```

```
i, pack->Get    );
```

jrtplib::RTPPacket *pack

不允许指针指向不完整的类类型





使用 jrtplib 接收数据包

RTPSession

RTPUDPv4TransmissionParams
RTPSessionParams

Example2:设置更多参数

```
session.SetDefaultPayloadType(96);
session.SetDefaultMark(false);
session.SetDefaultTimestampIncrement(160);
```

Example3:自动添加目的地

```
class MyRTPSession : public RTPSession

protected:
    void OnNewSource(RTPSourceData *dat)
    OnBYEPacket
    OnRemoveSource
```

Example4:后台线程自动处理

```
class MyRTPSession : public RTPSession
{
protected:
    void OnPollThreadStep();
    void ProcessRTPPacket(const RTPSourceData &srcdat,const RTPPacket &rtppack);
};
/** Is called each time the poll thread loops.
     *  Is called each time the poll thread loops. This happens when incoming data was
     *  detected or when it's time to send an RTCP compound packet.
     */
    virtual void OnPollThreadStep();
```


Example5--8:案例分析

RTPMemoryManager

```
/** A memory manager. */
class JRTPLIB_IMPORTEXPORT RTPMemoryManager
{
public:
    RTPMemoryManager()                                  { }
    virtual ~RTPMemoryManager()                             { }
```

```
    /** Called to allocate \c numbytes of memory.
     *  Called to allocate \c numbytes of memory. The \c memtype parameter
     *  indicates what the purpose of the memory block is. Relevant values
     *  can be found in rtpmemorymanager.h . Note that the types starting with
     *  \c RTPMEM_TYPE_CLASS indicate fixed size buffers and that types starting
     *  with \c RTPMEM_TYPE_BUFFER indicate variable size buffers.
     */
    virtual void *AllocateBuffer(size_t numbytes, int memtype) = 0;

    /** Frees the previously allocated memory block \c buffer */
    virtual void FreeBuffer(void *buffer) = 0;
};



    /** Allows you to use an RTP packet from the specified source directly.
     *  Allows you to use an RTP packet from the specified source directly. If
     *  `ispackethandled` is set to `true`, the packet will no longer be stored in this
     *  source's packet list. Note that if you do set this flag, you'll need to
     *  deallocate the packet yourself at an appropriate time.
     *
     *  The difference with RTPSession::OnRTPPacket is that that
     *  function will always process the RTP packet further and is therefore not
     *  really suited to actually do something with the data.
     */
    virtual void OnValidatedRTPPacket(RTPSourceData *srcdat, RTPPacket *rtppack, bool
isonprobation, bool *ispackethandled);
```

开源库 jrtplib 发送 H264 码流并用 VLC 播放

RTP 与 H264 相关结构体

```
#define PACKET_BUFFER_END       (unsigned int)0x00000000
#define MAX_RTP_PKT_LENGTH      1360
#define H264                     96


#define SSRC            100
#define DEST_IP_STR    "127.0.0.1"
#define DEST_PORT      12500
#define BASE_PORT      9400


typedef struct ///自己手工修改：大小端
{
    /**//* byte 0 */
    unsigned char csrc_len:4;        /**//* expect 0 */
    unsigned char extension:1;        /**//* expect 1, see RTP_OP below */
    unsigned char padding:1;        /**//* expect 0 */
    unsigned char version:2;        /**//* expect 2 */
    /**//* byte 1 */
    unsigned char payload:7;        /**//* RTP_PAYLOAD_RTSP */
    unsigned char marker:1;        /**//* expect 1 */
    /**//* bytes 2, 3 */
    unsigned short seq_no;
    /**//* bytes 4-7 */
    unsigned  long timestamp;
    /**//* bytes 8-11 */
    unsigned long ssrc;            /**//* stream number is used here. */
} RTP_FIXED_HEADER;
```

```
typedef struct {
    //byte 0
     unsigned char TYPE:5;
    unsigned char NRI:2;
     unsigned char F:1;

} NALU_HEADER; /**//* 1 BYTES */

typedef struct {
    //byte 0
    unsigned char TYPE:5;
     unsigned char NRI:2;
     unsigned char F:1;
} FU_INDICATOR; /**//* 1 BYTES */

typedef struct {
    //byte 0
    unsigned char TYPE:5;
     unsigned char R:1;
     unsigned char E:1;
     unsigned char S:1;
} FU_HEADER; /**//* 1 BYTES */
typedef struct
{
  int startcodeprefix_len;      //! 4 for parameter sets and first slice in picture, 3 for
everything else (suggested)
  unsigned len;                 //! Length of the NAL unit (Excluding the start code, which
does not belong to the NALU)
  unsigned max_size;            //! Nal Unit Buffer size
  int forbidden_bit;            //! should be always FALSE
  int nal_reference_idc;        //! NALU_PRIORITY_xxxx
  int nal_unit_type;            //! NALU_TYPE_xxxx
  unsigned char *buf;                    //! contains the first byte followed by the EBSP
  unsigned short lost_packets;  //! true, if packet loss is detected
} NALU_t;




//static bool flag = true;
static int info2=0, info3=0;
RTP_FIXED_HEADER        *rtp_hdr;
```

```
FILE *bits = NULL;                //!< the bit stream file
NALU_HEADER       *nalu_hdr;
FU_INDICATOR *fu_ind;
FU_HEADER     *fu_hdr;
```

Jrtplib 发送 H264 码流

H264sender_jrtplib.cpp

```
    RTPSession session;
    RTPSessionParams sessionparams;
    sessionparams.SetOwnTimestampUnit(1.0/90000.0);

    RTPUDPv4TransmissionParams transparams;
    transparams.SetPortbase(BASE_PORT);

    int status = session.Create(sessionparams,&transparams);


NALU_HEADER       *nalu_hdr;
    FU_INDICATOR *fu_ind;
    FU_HEADER         *fu_hdr;
    char sendbuf[1500];
    char* nalu_payload;
```

- 解析 H264 码流 GetAnnexbNALU

- RTP 封包

关于 h264 拆包，按照 FU-A 方式说明：

1）第一个 FU-A 包的 FU indicator：F 应该为当前 NALU 头的 F，而 NRI 应该为当前 NALU 头的 NRI，Type 则等于 28，表明它是 FU-A 包。FU header 生成方法：S = 1，E = 0，R = 0，Type 则等于 NALU 头中的 Type。

2）后续的 N 个 FU-A 包的 FU indicator 和第一个是完全一样的，如果不是最后一个包，则 FU header 应该为：S = 0，E = 0，R = 0，Type 等于 NALU 头中的 Type。

3）最后一个 FU-A 包 FU header 应该为：S = 0，E = 1，R = 0，Type 等于 NALU 头中的 Type。

Vlc 播放

- **Show.sdp**

m=video 12500 RTP/AVP 96
a=rtpmap:96 H264
a=framerate:15
c=IN IP4 127.0.0.1