H264 关于 RTP 协议的实现

2010-07-22 13:35

完整的 C / S 架构的基于 RTP / RTCP 的 H．264 视频传输方案。此方案中，在服务器端和客户端分别进行了**功能模块设计**。**服务器端**：RTP 封装模块主要是对 H．264 码流进行打包封装；RTCP 分析模块负责产牛和发送 RTCP 包并分析接收到的 RTCP 包；QoS 反馈控制模块则根据 RR 报文反馈信息动态的对发送速率进行调整；发送缓冲模块则设置端口发送 RTP、RTCP 包。**客户端**：RTP 模块对接收到的 RTP 包进行解析判断；RTCP 模块根据 SR 报文统计关键信息，产牛并发送 RR 包。然后，在 VC++6．0 下用 Socket 编程，完成基于 RTP / UDP / IP 的 H．264 视频传输，并在局域网内运行较好。

基于 RTP / UDP / IP 的 H．264 视频传输结构设计

对于 H．264 视频的实时传输应用来说，TCP 的重传机制引入的时延和抖动是无法容忍的，因此我们采用 UDP 传输协议。但是 UDP 协议本身是面向无连接的，不能提供质量保证。而基于 UDP 之上的高层协议 RTP / RTCP 可以一起提供**流量控制和拥塞控制**服务。图给出了基于 RTP / UDP / IP 的 H．264 视频传输的框架。

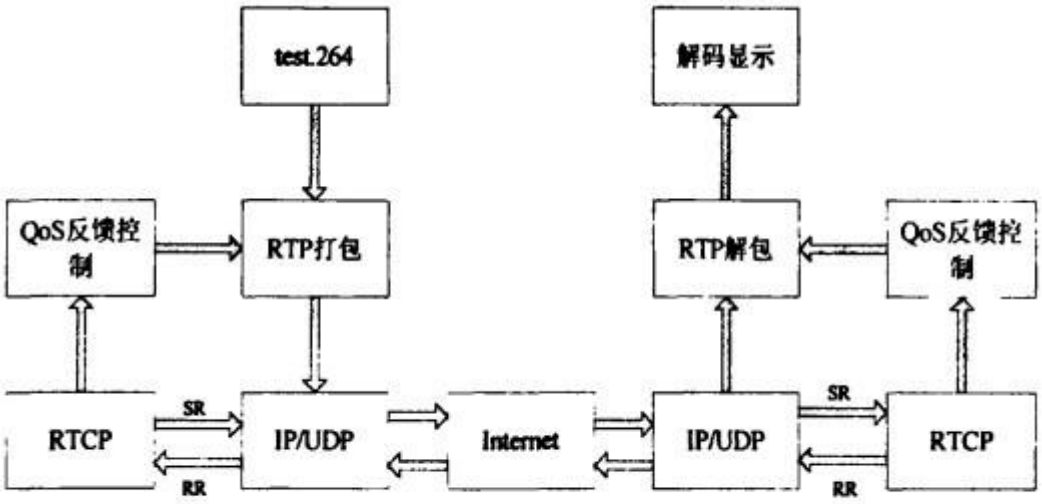

图4-1 基于RTP/UDP/IP的H.264视频传输框图

H．264 视频流的 RTP 封装策略

从图 4—1 可以看出，H．264 视频数据**首先经 RTP 进行封装**，打包成适合网络传输的**数据包才能进行传输**。所以，如何设计合适的 RTP 封装策略对 H．264 视频数据进行封装是十分重要的。一般来说，在 H．264 中，RTP 封装应该遵循几个设计原则：

1、较低的开销，因此 MTU 的尺寸应该限制在 100—64K 字节范围内。

2、易于区分分组的重要性，而不必对分组内的数据解码。

3、应能检测到数据的类型，而不需解码整个数据流，并能根据编码流之间的相关性丢弃无用数据，如网关应能检测 A 型分割的丢失，并能丢弃相应的 B 型和 C 型分割。

4、应支持将一个 NALU 拆分为若干个 RTP 包：不同大小的输入图片决定了 NALU 的长度可能会大于 MTU，只有拆分后才会避免 IP 层在传输时出现分片。

5、支持将多个 NALU 汇集在一个 RTP 分组中，即在一个 RTP 包中传输超过一个 NALU，当多个图片的编码输出小于 M1IU 时就考虑此模式，以提高网络传输效率。

RTP 载荷封装设计

本文的网络传输是基于 IP 协议，所以**最大传输单元(MTU)**最大为 1500 字节，在使用 IP / UDP / RTP 的协议层次结构的时候，这其中包括至少 **20 字节的 IP 头，8 字节的 UDP 头**，以及 **12 字节的 RTP 头**。这样，头信息至少要占用 40 个字节，那么 RTP 载荷的最大尺寸为 1460 字节。

IP报头 （20字节）	UDP报头 (8字节)	RTP报头 (12字节)	有效载荷

表4-2 网络传输中H.264的封装格式

一方面，如果每个 IP 分组都填满 1500 字节，那么协议头的开销为 2．7%，如果 RTP 载荷的长度为 730 字节，协议头的开销仍达到 5．3%，而假设 RTP 载荷的长度不到 40 字节，那么将有 50%的开销用于头部，这将对网络造成严重资源浪费。另一方面，如果将要封装进 RTP 载荷的数据大于 1460 字 节，并且我们没有在应用层数据装载进 RTP 包之前进行**载荷分割**，将会产生大于 MTU 的包。**在 IP 层其将会被分割成几个小于 MTU 尺寸的包**， 这样将会无法检测数据是否丢失。因为 IP 和 UDP 协议都没有提供分组到达的检测，如果分割后第一个包成功接收而后续的包丢失，由于只有第一个包中包含有完 整的 RTP 头信息，而 RTP 头中没有关于载荷长度的标识，因此判断不出该 RTP 包是否有分割丢失，只能认为完整的接收了。并且在 IP 层的分割无法在应用层 实现保护从而降低了非平等包含方案的效果。由于 UDP 数据分组小于 64K 字节，而且一个片的长度对某些应用场合来说有点太小，所以**应用层的打包**也是 RTP 打包机制的一个必要部分。最新的 RFC3984 标准中提供了针对 H．246 媒体流的 RTP 负载格式，主要有三种：
单个 NAL 单元分组、聚合分组、片分组。

NAL 单元单一打包

将一个 NAL 单元封装进一个包中，也就是说 RTP 负载中只包含一个 NAL 单元，NAL 头部兼作 RTP 头部。RTP 头部类型即 NAL 单元类型 1-23，如下图所示：

NAL单元类型	NAL单元类型名称
0	未使用
1	不分区，非IDR图像的片
2	A型分割
3	B型分割
4	C型分割
5	IDR图像中的片
6	增强信息SEI
7	序列参数集
8	图像参数集
9	分界符
10	序列结束
11	码流结束
12	填充
13-23	保留
24	单一时间聚合分组STAP-A
25	单一时间聚合分组STAP-B
26	多时间聚合分组MTAP16
27	多时间聚合分组MTAP24
28	片分组方式FU-A
29	片分组方式FU-B
30-31	为定义

表 4-3 NAL 单元类型及其名称

NAL 单元的重组

此分组类型用于将**多个 NAL 单元聚合在一个 RTP 分组**中。一些 H. 264 的 NAL 单元的大小，如 **SEI NAL 单元**、参数集等都非常小，有些只有几个字节，因此

应该把它们组合到一个 RTP 包中，将会有利于减小头标(RTP / UDP / IP)的开销。目前存在着两种类型聚合分组：

NAL 单元的分割

将一个 NAL 单元分割，使用**多个 RTP 分组**进行传输。共有两个类型 FU—A 和 FU—B，单元类型中分别为 28 和 29。根据 IP 层 MTU 的大小，对大尺寸的 NALU 必须要进行分割，可以在分别在两个层次上进行分割：

1)视频编码层 **VCL 上的分割**

为了适应网络 MTU 的尺寸，可以使用编码器来选择**编码 Slice NALU** 的大小，从而使其提供较好的性能。一般是对编码 Slice 的大小进行调整，使其小于 **1460 字节**，以免 IP 层的分割。

2)网络提取层 **NAL 上的分割**

在网络提取层上对 NALU 的分割主要是采用**分片单元方案**，H. 264 标准中提出了分割机制，可以使 NAL 单元的尺寸小于 1460 字节。注意：此方式是针对**同一个 NAL 单元进行分割的**，不适用于聚合分组。一个 NAL 单元采用分割分组后，每个 RTP 分组序列号依次递增 l，**RTP 时间戳相同且惟一**。NAL 单元的分割是 RTP 打包机制的一个重要环节，总结其分割机制主要有如下几个特点：

①分割 NALU 时，是以 RTP 次序号**升序进行**传输。在序列号不循环的前提下，属于前一帧图像的所有图像片包以及 A / B / C 数据分割包的序列号要小于后帧图像中的图像片及数据分割包的序列号。

②一个符号机制来标记一个分割的 NALU 是第一个还是最后一个 NAL 单元。

3.存在另外一个符号机制用来检测是否有丢失的分块。

④辅助增强信息包和头信息包可以任意时间发送。

⑤同一帧图像中的图像片可以以任意顺序发送，但是对于低时延要求的网络系统，最好是以他们原始的编码顺序来发送。

1)**单一时间聚合分组**(STAP)：包括单一时间聚合分组 A(STAP—A)和单一时间聚合分组 B(STAP—B)，按时间戳进行组合，他们的 NAL 单元具有相同的时间戳，一般用于低延迟环境。STAP—ASTAP—B 的单元类型分别为 24 和 25。

2)**多时间聚合分组**(MTAP)：包括 16 比特偏移多时间聚合分组(MTAPl6)和 24 比特偏移多时间聚合分组 (MTAP24)不同时间戳也可以组合，一般用于高延迟的网络环境，比如流媒体应用．它的打包方案相对复杂，但是大大增强了基于流媒体的 H．264 的性 能。MTAPl6 MTAP24 的单元类型分别为 26 和 27。

RTP 包的封装流程设计

根据 H. 264NAL 单元的分割重组的性质以及 RTP 打包规则，本文实行的对 **RTP 打包的设计**如下：

1、若接收到的 NAL 单元小于 MAX—SIZE(此时 MAX-sIZE 为设定的**最大传输单元**)，则对它进行单一打包，也就是将此 NAL 单元直接放进 RTP 包的载荷部分，生成一个 RTP 包。

2、若接收到的 NAL 单元大于 MAx—SIZE 字节，则对它进行分割，然后对分割后的 NAL 单元进行步骤 1 方式打包。分割方案如下：

```
If ((Nsize%MAX_SIZE)>0)
  K=( Nsize / MAX_SIZE)+1
else
  K= Nsize / MAX_SIZE
 if (Nsize %K>0)
   N= Nsize %K+1
 else
   N= Nsize /K
```

其中 **Nsize 是分割前的 NAL 单元大小**，N 是**分割后 NAL 单元的大小**。**K 分割后的单元数**。分割后最后一个单元的大小可能会小于 N，这时必须使用 **RTP 载荷填充**是其同前面的分块大小相同，此时 RTP 头中的**填充标识位值为** 1。

3、对 SEI，参数集等小 NAL 单元重组，将**它们合并到一个 RTP** 包中。虽然步骤 3 中的重组方案可以减小 IP / UDP / RTP 头部开销，但是对于包丢失率比较高的网络环境，这意味着一个 RTP 包的丢失可能会导致多片的丢失，往往一个片中就有一个 P 图像，解码后的视频质量必然会严重下降。因此，在丢失率的网络中可以采用 **NAL 单元的重组方案**，而在高丢失率的网络环境中采用 NAL 单元重组时要进行有效的差错控制．在本文中不使用重组方案。

RTP / RTCP 包的封装实现

RTP 包封装设计

将 H.264 码流进行 RTP 数据封装，下面是 H.264 中的 RTP 包头的数据结构：

```
Typedef struct
 {
  unsigned int v;              // 版本号，2 比特，值为 0X2
  unsigned int p;              // 填充标识，1 比特，值为 0
  unsigned int x;              // 扩充标识，1 比特，值为 0
  unsigned int cc;             // CSRC 计数，4 比特，值为 0
  unsigned int m;              // 标识位，1 比特
  unsigned int pt;             // 标识负载类型，7 个比特
  unsigned int seq;            // RTP 序列号,16 个比特
  unsigned int timestamp;      // 时间戳,32 个比特
  unsigned int ssrc;           // 同步源标识符，32 个比特
  byte*  payload;              // 载荷类型
  unsigned int paylen;         // 载荷的长度  NALU Length
  byte*  packet;              // 把头部和载荷打包
  unsigned int packlen;       // 包的长度
 }RTP_Packet;
```

RTcP 包的封装设计

RTCP 报文封装在 UDP 数据报中进行传输，发送时使用比它所属的 RTP 流的端口号大 1 的协议号(RTP 使用偶数号，RTCP 使用奇数号)。以下是 RTCP 头部数据结构：

```
Typedef struct
{
  unsigned int v;            // 版本号，2 比特，值为 0X2
  unsigned int p;            // 填充标识，1 比特，值为 0
  unsigned int rc;           // 接收报告计数，5 比特
  unsigned int pt;           // RTCP 包的类型，8 比特
  unsigned int length;       // 包的长度
  }RTCP_Header;
```

RTCP使用5个基本报文类型允许发送端和接收端交换有关会话信息。

```
 Typedef enum
 {
   RTCP_SR=200;              // 发送端报告类型为 200
   RTCP_RR=201;              // 接收端报告类型为 201
   RTCP_SDES=202;            // 源端描述报文类型为 202
   RTCP_BYE=203;             // 结束报文类型为 203
   RTCP_APP=204;             // 应用程序特定报文类型为 204
 }RTCP_Type
```

RTCP 接收者报文 RR 的封装数据结构：

```
Typedef struct
{
  unsigned int ssrc;                  // 同步源标识符
  unsigned int LostPacket;            // 丢包数
  unsigned int LostRate;              // 丢包率
  unsigned int MaxSeq;                // 最大序列号
  unsigned int Jitter;                // 到达间隔抖动
  unsigned int Lsr;                   // 最后到达的 SR 的时间戳
```

```
 unsigned int Dlsr;                    // 从最后一个发送端报告之后的延迟
}RTCP_Receiver;
```

RTCP 发送者报文 SR 的封装数据结构：

```
Typedef struct
{
 unsigned int TimeStamp;               // 时间戳
 unsigned int PacketCount;             // 发送包数
 unsigned int ssrc;                    // 同步源标识符
 unsigned int LostRate;                // 丢包率
 unsigned int MaxSeq;                  // 最大序列号
 unsigned int Jitter;                  // 到达间隔抖动
 unsigned int Lsr;                     // 最后到达的 SR 的时间戳
 unsigned int Dlsr;                    // 从最后一个发送端报告之后的延迟
}RTCP_Sender;
```

H264 实时编码及 NALU，RTP 传输（ZZ）

2010-07-25 11:46