【信息科学与控制工程】　　doi：10.11809/scbgxb2014.09.030

基于 ffmpeg 内核的 H.264 实时视频解码器开发

陈　阳

（中国兵器工业第 58 研究所军品部，四川 绵阳　621000）

摘要：针对高清视频编码压缩传输特点，提出基于 ffmpeg 内核的 H.264 实时解码方法，实现对 RTSP 发送的 H.264 实时视频流进行实时解码，解码后图像稳定、流畅、延时小；本方法设计简洁，调用方便，适用 H.264 格式的网络实时数据的采集和处理。

关键词：ffmpeg；RTSP；RTP；H.264；解码器

本文引用格式：陈阳. 基于 ffmpeg 内核的 H.264 实时视频解码器开发[J]. 四川兵工学报，2014(9)：102－104.

中图分类号：TP311.1　　**文献标识码**：A　　**文章编号**：1006－0707(2014)09－0102－03

Real-time Video Decoder of H.264 On ffmpeg Kernel

CHEN Yang

(Department of Military Products, No. 58 Research Institute of China Ordnance Industries, Mianyang 621000, China)

Abstract: Aiming at the feature of high precision video ecoding compression transmission, this paper puts forwards H.264 real-time decoding method based on ffmpeg core. It realizes the decoding of H.264 real-time video stream sent by RTSP. After decoding, the image is stable and smooth with little time delay. The design is simple and easy to call, and can be applied to H.264 network realtime data collection and treatment.

Key words: ffmpeg; RTSP; RTP; H.264; encoder

Citation format: CHEN Yang. Real-time Video Decoder of H.264 On ffmpeg Kernel[J]. Journal of Sichuan Ordnance, 2014(9): 102－104.

ffmpeg 是一个开源跨平台的视频和音频流方案，它支持超过 90 种解码器和协议，提供了录制、转换以及流化音视频的完整解决方案。目前，关于 ffmpeg 对 H.264 视频文件解码的相关介绍比较多，但对 ffmpeg 怎样处理实时 H.264 码流的相关介绍较少，笔者根据实时流媒体的传输和压缩特点，对基于 ffmpeg 内核的 H.264 实时视频解码器进行研究。

1　开发环境

在本设计中，使用高性能 pc 机，采用 windowsXP 操作系统，开发工具为 visual C ++6.0，H.264 解码内核采用 ffmpeg 5.0，视频的实时传输采用 RTSP 协议。RTSP（Real-Time Stream Protocol）是一种基于文本的应用层协议，它定义了如何有效的通过 IP 网络传送多媒体数据，使得实时流媒体数据的受控和点播变得可能。

2　开发步骤

2.1　实时数据的接收

H.264 实时视频解码器首先要获取实时视频流数据。在开发环境简介中我们介绍了网络传输采用 RTSP 协议，完成一次 RTSP 点播和停止操作过程有以下几步：客户端连接到流服务器并发送一个 RTSP 描述命令（DESCRIBE）；客户端接收流服务器的 SDP 反馈信息，包括流数量、媒体类型等信息；客户端分析该 SDP 描述后，根据其参数为会话中的流发送一个 SETUP 命令，该命令用于告诉服务器客户端接收媒体数据的端口；流媒体连接建立完成后，客户端发送一个播放命令（PLAY），服务器就开始在 UDP 上传送媒体流（RTP 包）到客户端；客户端可发送一个终止命令（TERADOWN）来结束流媒体会话。建立了 RtspRequest 类，专门用于负责

收稿日期：2014－06－27

作者简介：陈阳（1979—），女，工程师，主要从事智能安防、态势感知技术研究。

RTSP 的通讯,通过调用 RtspRequest 类中的函数建立 RTSP 连接,主要代码如下:

```
g_RtspRequest. Open( m_url,"192. 168. 1. 100",0);
g_RtspRequest. RequestOptions( );
g_RtspRequest. RequestDescribe( &sdp);
g_RtspRequest. RequestSetup( setupName,1, m_rtpPort, rtcpPort,&sess);
g_RtspRequest. RequestPlay( );
handle_rec = CreateThread( NULL,4, RcvVideoData, this, 0,NULL);
```

其中 m_url 存储的是服务器的地址,由于媒体流信息中的视频和音频是分开传送的,通过 setupName 来向服务器申请获取流的类型,要获取视频流则 setupName = "track1",要获取音频流则 setupName = "track2"。通过 RTSP 通讯,服务器端开始通过 RTP 向客户端发送视频流数据,因此笔者创建接收 RTP 数据线程来获取实时发送的视频数据。

2.2　获取 RTP 数据报文

RTP 即实时传输协议,用于 Internet 上针对多媒体数据流的传输。它通常使用 UDP 协议来传送数据。RTP 协议主要完成对数据包进行编号,加盖时 戳,丢包检查,安全与内容认证等工作。通过这些工作,应用程序会利用 RTP 协议的数据信息保证流数据的同步和实时传输。目前开源 JRTPLIB 库是一个用 C ++ 语言实现的 RTP 库,包括 UDP 通讯,因此笔者在工程中通过调用 JRTPLIB 库函数来实现 RTP 数据流的接收。下载 jrtplib - 3. 7. 1. rar 后,首先将其解压到一个临时文件夹中,编译后获得 jthread. lib 和 jrtplib. Lib 两个 lib 文件,将它们复制到 VC6 的 lib 文件夹下,将两个库文件的头文件拷贝到项目的 include 文件夹中,在接收 RTP 数据的 CPP 文件中调用#include 命令引用 JRTPLIB 的头文件,通过#pragma comment 命令调用两个库文件 jthread. lib 和 jrtplib. Lib。做完以上操作,就可调用相关函数来获取 RTP 数据了。主要代码如下:

```
//创建 RTP 对话代码
RTPSessionParams sessionparams;
  RTPUDPv4TransmissionParams transparams;
  sessionparams. SetOwnTimestampUnit(1. 0/90000. 0 );
  transparams. SetPortbase( m_rtpPort );
  status = rtpsess. Create( sessionparams, &transparams);
//在线程 RcvVideoData 中获取 RTP 数据代码
  dlg - > rtpsess. Poll( );
  dlg - > rtpsess. BeginDataAccess( );
if( dlg - > rtpsess. GotoFirstSourceWithData( ) )
  {
    do
    {
  RTPPacket * rtppack;
  while( ( rtppack = dlg - > rtpsess. GetNextPacket( ) ) ! = NULL )
    {
recvdata = rtppack - > GetPacketData( );
recvsize = rtppack - > GetPayloadLength( );
if( recvsize > 0)
{
//此处将接收的 RTP 数据包进行解析
}
dlg - > rtpsess. DeletePacket( rtppack );
  }
}while( dlg - > rtpsess. GotoNextSourceWithData( ) );
}
dlg - > rtpsess. EndDataAccess( );
```

2.3　解析 RTP 数据

RTP 接收的视频编码格式为 H. 264,因此必须先了解 RTP 包和 H. 264 的相关知识和格式。RTP 包头格式如图 1 所示。

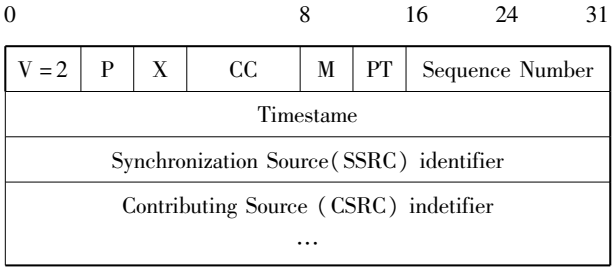

图 1　RTP 包头格式

负载类型 Payload type (PT):7 bits;序列号 Sequence number (SN):16 bits;时间戳 Timestamp:32 bits;对于 H. 264 来说说 PT = 96。H. 264 分为视频编码层(VCL)和网络抽象层(NAL),VCL 包含 Codec 的信令处理功能如转换、量化、运动补偿预测机制以及循环过滤器,(NAL)封装 VCL 编码器输出的片断到网络抽象层单元(NAL units),它适合于通过包网路传输或用于面向包的多路复用环境,用与实际的传输。所有 NAL 单元有一个单个 NAL 单元类型字节,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(forbidden_zero_bit):1 位,在 H. 264 规范中规定了这一位必须为 0;NRI(nal_ref_idc):2 位,取 00 ~ 11,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,一般情况下不太关心这个属性;Type(nal_unit_type):5 位,NALU 单元的类型,类型描述如表 1 所示。

对于 NALU 的长度小于 MTU(最大传输单元值为 1 500 字节)大小的包,一般采用单一 NAL 单元模式,即一个 RTP 包仅由一个完整的 NALU 组成. 这种情况下 RTP 包内 NAL 头类型字段和原始的 H. 264 的 NALU 头类型字段是一样的。对于一个原始的 H. 264 NALU 单元常由[Start Code][NALU Header][NALU Payload] 3 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 00 01",NALU 头仅一个字节,其后都是 NALU 单元内容。

表1 NALU 单元的类型

类型	定义	备注
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	没有定义	

对于对于一些分辨率较高的实时视频传输,大多数时候是一个 NALU 单元封装成多个 RTP 包,因此需要对接收的 RTP 数据进行分析,将多个 RTP 中的传输的数据组合成一个完整的 NALU 单元,传入到 FFmpeg 中进行解码。笔者通过抓包分析发现,将接收的 RTP 包去掉 RTP 包头 12 个字节后,判断分片包的类型只需分析第一个字节,如果第一个字节 0 位为 1,为首包,第一个字节 1 位为 1 则为尾包,其余为分片包的中间包,组合时在分片包的首包前面加上[00 00 00 01]即得到完整的 NALU 单元。主要代码如下:

```
if ( PayloadType ! = 28 ) // whole NAL
{
    *(DWORD*)m_pBuf=0x01000000;
    memcpy(m_pBuf+4,pPayload,PayloadSize);
    *outSize=PayloadSize+4;
    m_bAssemblingFrame=false;
    return m_pBuf;
}
else // FU_A
{
    if ( pPayload[1] & 0x80 ) // FU_A start
    {
    pPayload[1] = (pPayload[0]&0xe0) + (pPayload[1]
&0x1f);
    *(DWORD*)m_pBuf=0x01000000;  CopyMemory(m_
pBuf+4,pPayload+1,PayloadSize-1); m_dwSize=PayloadSize
+3;
    return NULL;
}
else if(pPayload[1]&0x40)//FU_A end
{
    CopyMemory(m_pBuf+m_dwSize,pPayload+2,PayloadSize
-2);
    *outSize=m_dwSize+PayloadSize -2;
    return m_pBuf;
}
else//FU_A middle
{
    CopyMemory(m_pBuf+m_dwSize,pPayload+2,PayloadSize
-2);
    m_dwSize=m_dwSize+PayloadSize -2;
    return NULL;
}
}
```

2.4 H.264 解码

通过以上几个步骤获取了 H.264 帧数据后,通过调用 ffmpeg 来进行视频解码显示。Ffmpeg 在 VC 中的调用方式与 JRTPLIB 类似,此处就不再详述,将对 ffmpeg 的调用过程进行描述。整个解码过程分为以下几步:

(1)初始化 ffmpeg 和注册库中所有可用的文件格式和编码器,调用 avcodec_init 和 av_register_all 函数实现此功能;

(2)使用 avcodec_find_decoder 函数在注册库中寻找 H.264 解码器,m_ffvpCodecCtx->flags|=CODEC_FLAG_TRUNCATED 设置解码器能够处理截断的数据帧,后通过 avcodec_open 打开解码器;

(3)因为需要把帧的格式从原来的转换为 RGB,所以需要调用两次 avcodec_alloc_frame 为解码后的原始帧和转换后的 RGB 帧分配内存;

(4)将完整的 NALU 单元传入函数 avcodec_decode_video 进行解码,将解码后获取的帧放入已分配好的内存空间中;

(5)调用 sws_getContext、avpicture_fill、sws_scale 等函数将原始图像转换为 RGB。

完成以上步骤后即可获取解码后的 RGB 数据,实现解码功能。

3 结论

目前,该设计已成功应用于多个项目的前端视频数据采集中。测试结果表明:基于 ffmpeg 内核的 H.264 实时解码器,能够接受通过 RTSP 发送的 H.264 实时视频流并实时解码,设计简洁,调用方便,从显示效果来看,解码后图像稳定、流畅,延时小,适用 H.264 格式的网络实时数据的采集和处理。

参考文献:

[1] 张前进.基于 RTP 的 h264 实时传输系统的设计与实现[J].企业技术开发,2011(23):12-15.

[2] 赵臣兵,刘立柱.基于 RTP 协议的视频实时采集与传输的研究[J].微计算机信息,2006(16):57-61.

[3] 刘马飞,曾学文,倪宏.windows 平台下应用 ffmpeg 实现 h.264 视频回放[J].微计算机应用,2008(11):101-104.

[4] 盛先刚.基于 RTP 的 h264 视频传输系统研究[D].西安:西安电子科技大学,2006.

[5] 肖娅.H.264 视频流实时传输系统的研究与应用[D].北京:北京邮电大学,2008.

(责任编辑 周江川)