利用 VC++实现局域网实时视频传输

张勇 于金峰 蔡骅
（南京理工大学 自动化系，江苏 南京　210094）

摘　要　文中针对不同的局域网，提出一种通用的实时视频传输的解决方案。在使用 Divx 编解码的基础上，提出了从压缩、组帧、发送到接收、解压整个流程的思想，具体实施方案和 VC++实现核心源代码以及传输控制策略，有效地保证了高质量的实时视频传输。

关键词　客户/服务器；实时视频传输；Divx

1　引言

在局域网内部实时传输视频已经得到广泛应用。现在用以传输视频的局域网大多数是有线局域网，因为有线局域网技术成熟，传输速度快，稳定性好。但是视频数据量大，有线网络也会出现工作不稳定，引起数据堵塞，时间久了会导致严重的延迟现象；如果工作的环境不固定，要求移动性，那么就要采用无线网络，如今无线网卡的工作随环境的变化而变得不稳定，这样会导致视频传输的质量大幅度下降，容易引起画面的重影、抖动、花屏等现象。本文针对不同的局域网，提出一种通用的实时视频传输的解决方案，使用 VC++自封装的 Windows VFW SDK 软件开发包进行二次开发，通过 Divx 编解码，按照制定的传输策略，能够有效地解决由于网络的局部不稳定导致的视频图像重影、抖动、花屏等的问题。

2　局域网中实时视频传输存在的问题

为了在局域网上有效的、高质量的传输视频流，需要多种技术的支持，包括视频的压缩、编码技术，应用层质量控制技术等等[1]。

网络的带宽是有限的，所以需要压缩传输视频图像，MPEG－4 被广泛的应用于网络环境下的实时视频传输[1,2]，因为 MPEG－4 具有：可以达到很高的压缩比；具有灵活的编码和解码复杂性；基于对象的编码方式，允许视频、音频对象的交互；具有很强的容错能力等优点。本文采用 Divx 编解码器对视频进行编码、压缩，实际上 Divx＝（视频）MPEG－4＋（音频）MP3。

应用层质量控制技术现在采用的是 RTP/RTCP 协议[3,4]，以确保视频流在网络中低时延、高质量地传输。RTP 数据传输协议负责音视频数据的流化和负载，RTCP 负责 RTP 数据报文的传输控制。此协议是通过客户端（接收方）反馈网络的状况，服务器端（发送方）来调整信息采集、发送的速度和压缩率。但是，对于图像采集速度固定，需要软件进行压缩、解压，调整采集的速度会引起采集的数据来不及压缩而直接丢弃，调整编码器的压缩率需要重新设置编码器的参数，重启编码器，相应的解码器也要调整，这个过程中需要很长的时间，达不到实时的要求。所以本文没有采用 RTP/RTCP 协议，而是从发送端出发，实时判断网络状况，采用“停等”策略进行实时传输。

网络通信有两种协议 TCP 和 UDP，UDP 更适合于网络环境下的视频传输，但是它不提供检错和纠错功能，一旦网络出现堵塞时，大量的数据报文会丢失。对于 Divx 编解码技术，是以帧为单位进行编解码的，分为关键帧和非关键帧。在传输过程中，由于压缩率比较高，只要一帧中错一比特位，将影响其它几百甚至几千的比特位，直接造成图像的模糊、花屏等现象。只有等到下一次关键帧的到来才有可能恢复图像的清晰。为了保证传输的正确性，自己需要在应用层制定协议。如此一来，UDP 的优势荡然无存。所以本文选择使用 TCP 来进行网络通信。[5]综合使用 VFW 技术、流媒体技术，辅助以“停等”控制策略，较好的解决局域网中实时视频传输容易引起的重影、抖动、花屏的问题。

3　实时视频传输实现

为了达到视频传输的实时性，总的思想是最少的发送冗余信息，最大程度上发送最新的视频。

局域网实时视频传输采用服务器/客户机模式，利用 VC++实现。其工作流程如图 1 所示。

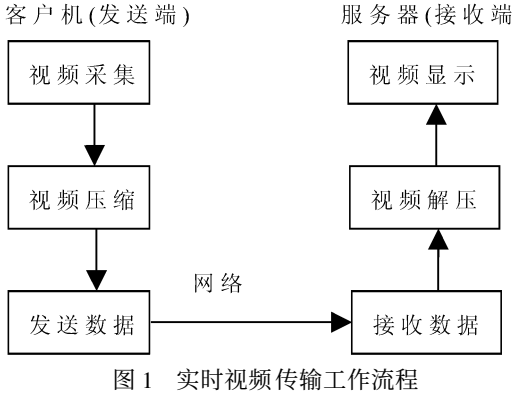

图 1　实时视频传输工作流程

视频采集采用 AVICap 从视频采集卡捕获视频图像，得到的是位图型式的视频帧，然后用 Divx 编码器进行压缩，通过 Winsock 实现压缩后的视频数据在局域网中的实时传输，接收完的数据交给 Divx 解码器解压，最后实现视频显示。

在 VC++中，采用 VFW 技术，客户端通过 capSetCallback-OnFrame()注册回调函数，当采集卡采集到一幅图像后，系统

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

就会自动调用回调函数，然后再回调函数中使用 ICSeqCompressFrame()函数进行压缩。然后再通过 Winsock 将压缩后的数据发送到服务器端。服务器端接收完一帧以后，交给 ICDecompress()解压，最后用 SetDIBitsToDevice()将图像显示出来。

3.1 视频帧的组建

视频采集的数据是位图型式的视频帧，Divx 编码器压缩以后形成以帧为格式的 Mpeg4 流。Divx 解码器也是以帧的格式解压。所以提出以帧为单位发送视频数据流。为了在接收端能够方便地提取出一帧，提出如图 2 所示的格式组建帧。

帧开始标志	帧大小	帧编号	帧类型	帧数据

图 2 视频帧格式

完整的一帧由 5 个字段组成，各个字段的意义如下：帧开始标志，标志着一帧地开始，占用 4 个字节的空间。不妨设为 0xffffffff。帧大小，表示整个帧的大小，包括 5 个字段的大小，占用 4 个字节的空间。帧编号，表示帧的顺序编号，占用 4 个字节的空间。帧类型，标志此帧是否是关键帧，占用 1 个字节的空间。帧数据，存放压缩后一帧的完整数据。

3.2 视频帧的发送

实时视频传输为了实时，要不断地将压缩好的数据发送到接受端。所以在发送端创建一个线程，专门用来发送数据。同时主线程仍然不停的采集数据并进行压缩。发送线程的工作流程如图 3 所示。

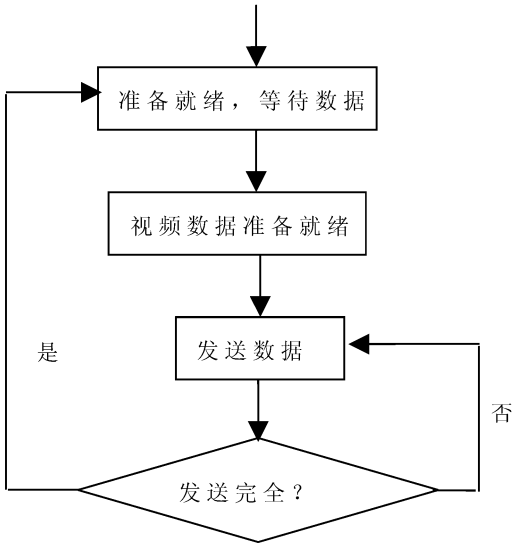

图 3 发送线程工作流程

不妨假设创建的线程名为 sendThread，核心代码实现如下：

```
while(1)
{
    isOK= true; //准备就绪
    SuspendThread(sendThread); // 挂起线程
    isOK= false; //线程正在发送数据
    int length= frameLength; //待发数据长度
    if(length< 50000) {//判断数据是否正常
      int n=0;
      int sendCount=0;
      while(length>0) {
        n= send(sock, (char *)imageBuf+ sendCount, length, 0); //发送数据,
        //imageBuf 是指针，指向待发数据帧
        if(n== SOCKET_ERROR) //网络出现异常，则退出线程
          break;
        length-= n;
        sendCount+= n;
      }
    }
}
```

线程中发送的数据帧是按照上一节中的方法组建好的数据帧。这种方法能够保证正在发送的当前帧能够完整地到达接收端。

注意此线程中刚开始或者每当发送完一帧以后，线程就转到挂起状态，等待外界唤醒。这个任务由回调函数完成，在回调函数中，判定如果发送线程准备就绪（处于挂起状态），则进行图像压缩，然后唤醒线程发送压缩完的数据，否则直接跳出，等待下一次调用回调函数，这种策略称之为“停等”策略，在后面有详细介绍。

3.3 视频帧的接收

接收端最重要的是从接受的数据流中提取出完整的一帧。方法的思想是：首先从数据流中寻找帧开始标志，再从紧挨后面的数据中提取出帧的大小，然后再从接收缓冲区中读入该帧剩余的数据。再寻找下一帧的开始标志，如此往复。图 4 是接收端的工作流程。

同样接收端创建一个线程专门用来执行数据接收。不妨假设线程名为 recThread，核心代码实现如下：

```
while(temp != SOCKET_ERROR)
{
  if(!isStart) {//帧数据是否开始，true 表示开始
    if(endNum> 3) // endNum 纪录当前接收未处理的数据
      endNum= 0;
    temp= recv(clisock, (char *)(recBuf+ endNum), 1000, 0);//从缓冲区读取数据
    startPos= serchStr(temp+ endNum); //查找帧开始标志
    if(startPos != -1) {
      isStart= true;
      endNum= temp+ endNum- startPos- 4;
      memcpy(imageBuf, recBuf+ startPos+ 4, endNum); //保存帧数据
    }
    else{
```

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

```
            memcpy(recBuf, recBuf+ temp+ endNum- 3, 3);//保存最后三个字节的数据
            endNum= 3;
          }
        }
      else{
        if(endNum< 4){//判定紧跟开始标志的数据, 如果小于 4 表示不能获得帧大小
          temp= recv(clisock, (char *)(recBuf), 1000, 0); //读入数据
          memcpy(imageBuf+ endNum, recBuf, temp);//保存数据
          endNum+= temp;
          if(endNum< 4)
            continue;
          frameSize= *((int *)imageBuf);//获得帧大小
          if(frameSize< 500 || frameSize> 50000){//异常处理(帧大小非法)
            isStart = false; //丢弃数据重新查找帧开始标志
            endNum = 0;
            continue;
          }
          frameSize-= endNum+4;
        }
        else{
          while(frameSize> 0&&temp!= SOCKET_ERROR){//获得完整帧的剩余数据
            temp= recv(clisock, (char *)(imageBuf+ endNum), frameSize, 0);
            endNum+= temp;
            frameSize-= temp;
          }
          if(frameSize<= 0){//帧结束置位, 解压
            isStart= false;
            endNum= 0;
            deCompress();//判断数据的有效性, 调用 ICDecompress 进行解压
          }
        }
      }
    }
```

以上程序执行的结果是将完整的一帧(除帧开始标志)保存在 imageBuf 中。

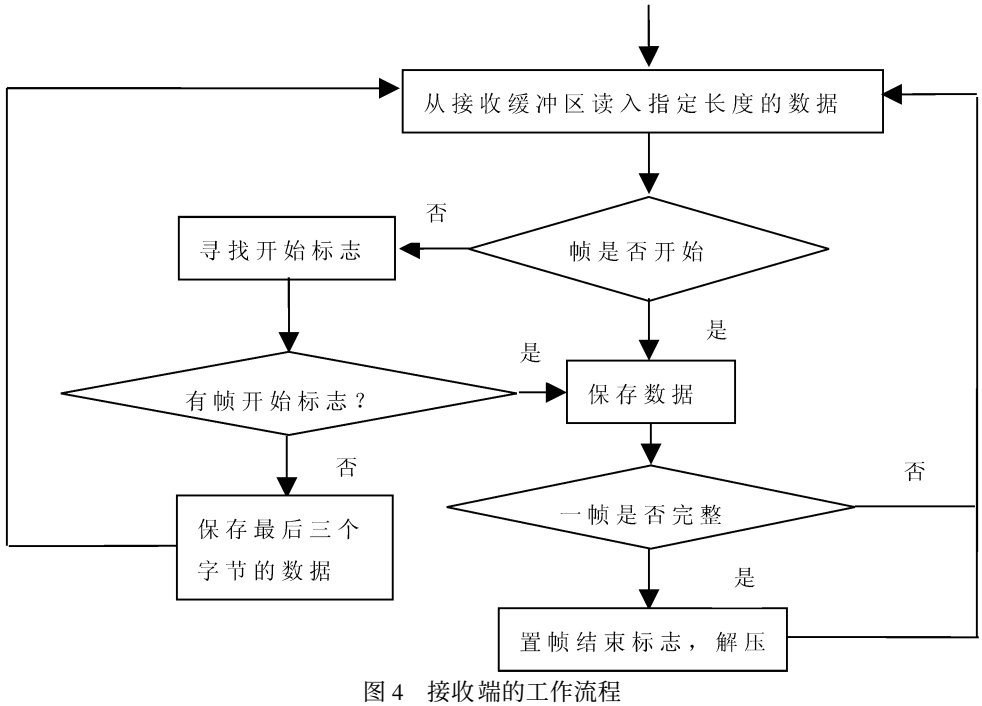

图 4　接收端的工作流程

3.4 “停等”控制策略

如果局域网通信速率很高, 而且工作稳定, 则按照以上说的方法进行实时视频传输, 不需要任何控制策略, 就可以达到非常好的效果。但是在很多情况下, 网络会出现异常, 这样会导致数据传输率明显下降, 造成发送端数据积压, 等待发送的数据不能正常发出去。此时就要采取一定的策略来控制发送端, 以达到实时性的要求。

上文发送程序中, 变量 isOK 是用来表示发送端当前帧有没有发完, 如果发完则置为 true, 同时也表示发送端准备就绪, 可以继续发送数据, 否则为 false。那么可以用 isOK 来通知视频采集和压缩线程, 如果 isOK 为 true, 则可以采集视频并且压缩, 然后唤醒发送线程继续发送新来的帧数据, 否则一直等待, 直到网络可以继续发送数据(isOK 为 true)。当然, 视频采集一直不停的进行, 那么当网络发生数据堵塞时, 只

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

要不让编码器进行压缩则可解决；当网络恢复正常时，继续进行压缩传输，换句话说，当网络发生堵塞时，直接抛弃等待发送的帧，保证一旦网络恢复时，发送最新的压缩帧。当然要保证一旦有一帧开始发送，就要将其完全发出。

按照这样的“停等”策略进行实时视频传输，只会带来一个问题：当网络质量差时，接收端画面中的移动目标会出现瞬间移动的现象。但是这种策略会保证不会出现重影，抖动，花屏等现象。

4 结论

本文提出的实时视频传输方案在 100M 的局域网、10M 局域网和 11M 无线局域网中进行了测试。测试时让一个目标在镜头前(发送端)移动，观察接收端视频的显示。在不同的局域网中进行了多次测试，每次测试时间从 10 分钟到 30 分钟不等，并且改变目标的运动速度进行实验。最后将数据汇总，得出统计结果。测试结果如表 1 所示。

表 1 不同局域网下的测试结果

	剧烈运动	正常运动	缓慢运动
100M 局域网	图像清晰，很流畅	图像清晰，很流畅	图像清晰，很流畅
10M 局域网	偶尔出现停顿，丢帧率 1%左右	图像清晰，人眼感觉流畅	图像清晰，很流畅
11M 无线局域网	经常出现停顿，丢帧率 5%－6%	经常出现停顿，丢帧率 2%－3%	偶尔出现停顿，丢帧率 1%左右

其中，丢帧率= $\frac{\text{采集但没有压缩的帧数}}{\text{采集帧总数}}$

注：11M 无线网卡是通过 USB1.0 接口和 PC 机连接的，如果采用 USB2.0 接口效果会更好。

从实际测试的结果看，效果是良好的，除了出现瞬间移动外，图像能够保持清晰，消除了由于网络质量差而导致的重影、抖动等现象，对于不同的局域网都能满足实时传输的要求。

参考文献

1 王海涛，张学平. 流媒体的关键技术及应用. 数据通信，2005(05)

2 谢长勇，陈念年，李波. 基于 MPEG－4 网络实时视频传输系统的设计. 攀枝花学院学报，2005(04)

3 朱鹏，李春文. 基于 RTP 的网络视频传输系统的设计与实现. 计算机工程与应用，2003(26)

4 李燕灵，马瑞芳，左力. 基于 RTP/RTCP 的实时视频数据传输模型及实现. 微电子学与计算机，2005(08)

5 王鹏，季桂树，张志武. 基于 TCP 协议控制的实时视频流传输系统. 微机发展，2005(10)

上接第 72 页

6 总结

本文通过对 portlet 和 servlet 结构上的对比分析阐明为什么 mvc 模式仍然适用于 portlet 应用的开发。给出保持 mvc 模式的 portlet 服务使用实现模式。通过 webservice 技术体系的特点指出引入缓存系统的必要性。提出了一种使用 oscache 和 jms 的 portlet 应用系统架构以提升系统性能。当然这里还有许多研究的细节需要在下一步工作中做继续探讨和研究。

参考文献

1 W. Clay Richardson, Donald Avondolio, Joe vitale, etc:《Professional Portal Development With Open Source Tools》, Wiley publishing inc, 2004；

2 《oasis－200304－wsrp－specification－1.0》；

3 《portlet－1.0－pfd－spec》；

4 “应用 OSCache 提升 J2EE 系统运行性能” http://gceclub.sun.com.cn/NASApp/sme/controller/teclist?tid=018&rid=0108#；

5 “Understanding JavaServer Pages Model 2 architecture”, http://www.javaworld.com/javaworld/jw－12－1999/jw－12－ssj－jspmvc.html；

6 oscache faq, http://www.opensymphony.com/oscache/wiki/FAQ.html#FAQ－objects.

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net