利用 JAVA实现局域网的跨平台实时视频传输*

吴良斌
（福建信息职业技术学院，福建 福州 350003）

摘 要：通过分析在局域网中视频传输的特点和模型及存在的问题，并针对不同的平台，提出一种通用的实时视频传输的解决方案，即在 SUN公司提出的 JMF多媒体框架基础上进行扩展，实现一个与平台无关的基于 RTP/RTCP协议的流式媒体传输模型的系统设计。

关键词：JAVA；服务器；RTP/RTCP；实时视频传输；JMF
中图分类号：TP393.1 文献标识码：A 文章编号：1124（2007）03－0017－05

0 引 言

目前在局域网内部实时传输视频已经得到广泛应用，且传输视频的局域网大多数是有线局域网，主要因为有线局域网技术成熟，传输速度快，稳定性好。但在视频数据量大时，也会出现工作不稳定、数据堵塞、甚至于严重的延迟现象。基此，本研究拟采用 RTP/RTCP协议，并利用 JMF的多功能，扩展其原有接口，以实行局域网的跨平台实时视频传输。

1 局域网中视频传输的典型模式

在局域网中视频传输采用的是服务器/客户机模式，一个视频系统的功能模块如图 1所示，它包括 5个功能模块：

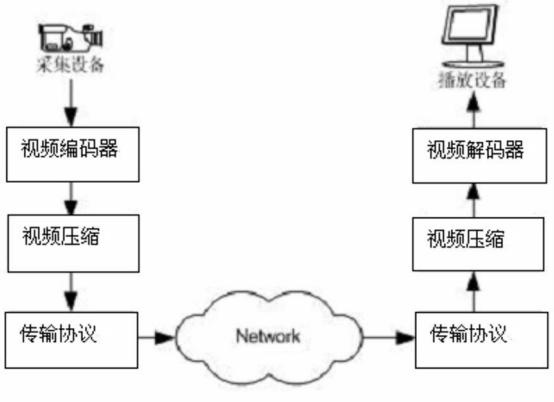

图 1 实时视频传输工作流程

1）视频采集和视频显示

视频采集设备负责将视频信息输入计算机，如数码摄像头等。而视频显示负责播放视频信息，如一般的显示终端。

2）编码和解码工具

编码软件负责将视频采集设备传送过来的视频信号编码成流媒体格式，而解码软件与之相反负责将媒体数字信号转化成终端可识别的信息。

3）视频压缩和视频解压

视频媒体数据量巨大必须将编码后的视频媒体数据进行高效的压缩，然后将压缩后的数据在网络上传输。其相应的解压模块负责将数据还原成为原来的流媒体格式。

4）传输协议

流媒体数据由于自身的特点，在网络上传输时将采用和过去诸如文字，静态图像等不同的网络协议来进行传输。

5）传输网络

一般来说视频传输可以在现有任何网络上进行传输，更高的带宽将有利于视频传输效率的提高，也能够保证其播放质量。

在图 1中，可以看到作为整个传输模式的底层，传输协议有着至关重要的作用。本文拟实现的

* 收稿日期：2007－03－22
作者简介：吴良斌（1977— ），男，福建三明人，助教，研究方向：网络编程、网络安全。

方案,主要采用 RTP/RTCP协议(该协议是由 IETF为视音频的实时传输而设计的传输协议)。当前的视频传输系统存在的问题主要有:可扩展性差、格式不统一等,我们则依据 JAVA具有平台无关性的特点,利用 JMF(Java Media Framework)的多功能,并对其原有接口加以扩展,来实现基于 RTP/RTCP的流媒体传输模型。

2 基于 JMF架构实现 RTP/RTCP传输模型

JMF(Java Media Framework)是 Sun公司提出的 Java媒体架构,它是对应 Java2平台标准版(J2SE)的一种可选用的应用编程接口(API)。JMF的源代码通过 SCSL(Sun社团源代码许可模式)发布。下面给出本文实现视频传输系统的结构及其有关的代码。

2.1 结构介绍

2.1.1 服务器

接受多台客户端连接,传送客户端信息。

VideoServer.java:定义服务器 Socket和输入输出流。

ServerFrame.java:定义服务器界面。

2.1.2 客户端

通过服务器,互相浏览视频、收听音频和文字交流。

RTPTransmit.java:定义 RTP视音频数据传送。

RTPReceive.java:定义 RTP视音频数据接收。

VFWAuto.java:调用 JMF视音频设备接口。

Client.java:定义客户端 Socket和控制 RTP传输。

MainFrame.java:定义客户端主界面和功能。

2.1.3 其他类

服务器和客户端共享包。

CustInfo.java:客户端信息类。

Customer.java:客户端请求包,序列化。

Msg.java:客户端文字聊天记录。

2.2 关键组件的实现

2.2.1 视频、音频设备捕获

在“系统设置”窗口中调用 JMF的设备搜索接口,调用代码如下:

VFWAuto.java部分代码:

```
public VFWAuto() {
Vector devices = (Vector) CaptureDeviceMan-
ager.getDeviceList(null).clone();
Enumeration enum1 = devices.elements();
while(enum1.hasMoreElements()) {
CaptureDeviceInfo cdi = (CaptureDeviceInfo)
enum1.nextElement();
String name = cdi.getName();
if(name.startsWith("vfw"))
CaptureDeviceManager.removeDevice(cdi);
}
int nDevices = 0;
for(int i = 0; i < 10; i++) {
String name = com.sun.media.protocol.vfw.
VFWCapture.capGetDriverDescriptionName(i);
if(name! = null&& name.length() > 1) {
System.err.println("Found device " +
name);
System.err.println("Querying device. Please
wait...");
com.sun.media.protocol.vfw.VFWSourceS-
tream.autoDetect(i);
nDevices++;
}
}
```

VFWAuto.java调用了 JMF的视频设备搜索接口,它返回一个 Vector数组,里面存放包含设备信息的 CaptureDeviceInfo类对象。

获取本地视音频数据

```
public static javax.media.Player player = null;
public static MediaLocator audioml = null,
videoml = null;
DataSource[] dataSources = new DataSource
[2]; //建立混合数据源
dataSources[0] = Manager.createDataSource
(videoml);
dataSources[1] = Manager.createDataSource
(audioml);
DataSource ds = Manager.createMergingData-
```

```
Source( dataSources);
    player = Manager.createRealizedPlayer
( ds); //建立媒体播放器
    player.start();
    Component comp;
    if((comp = player.getVisualComponent()) !
= null){
    //comp.setSize(localVideoPanel.WIDTH, lo-
calVideoPanel.HEIGHT);
    localVideoPanel.removeAll();
    localVideoPanel.add(comp); //放置视频组件
    localVideoActive= true;
    }
```

2.2.2 视音频数据实时传输

步骤一: 建立 2个数据源, 分别存储音频数据和视频数据。

```
    dataSources[0] = Manager.createDataSource
(MainFrame.vidoml);
    dataSources[1] = Manager.createDataSource
(MainFrame.audioml);
    //vidoml和 audioml是 MediaLocator类实例,
是主窗口的视频、音频数据地址。
    DataSource ds = Manager.createMergingData-
Source(dataSources);
    //组合视频音频数据, 建立新的数据源 ds
    Processor processor = Manager.createProcessor
(ds);
    //利用参数 ds, 建立数据处理器 processor
```

步骤二: 检查视频数据格式是否合法。

```
    Format checkForVideoSizes(Format original,
Format supported){
    int width, height;
    Dimension size = ((VideoFormat) original).
getSize();
    Format jpegFmt = new Format(VideoFormat.
JPEG_RTP);
    Format h263Fmt = new Format(VideoFormat.
H263_RTP);
    if(supported.matches(jpegFmt)){
    width= size.width% 8 == 0? size.width:
((int)(size.width/8) * 8);
    height= size.height% 8 == 0? size.height
: ((int)(size.height/8) * 8);
    }
    else if(supported.matches(h263Fmt)){
    if(size.width<= 128){
    width= 128; height= 96;
    }else if(size.width<= 176){
    width= 176; height= 144;
    }else{
    width= 352; height= 288;
    }
    }else
    return supported;
    return (new VideoFormat(null, new Dimension
(width, height),
    Format.NOT_SPECIFIED, null,
    Format.NOT_SPECIFIED)).intersects(suppor-
ted);
    }
```

返回 Format类型

步骤三: 传输视频音频数据。

```
    private RTPManager rtpMgrs[];
    private String createTransmitter(){
    PushBufferDataSource pbds= (PushBufferData-
Source) dataOutput;
    PushBufferStream pbss[] = pbds.getStreams
();
    rtpMgrs= new RTPManager[pbss.length];
    for(int i= 0; i< pbss.length; i++){
    try{
    rtpMgrs[i] = RTPManager.newInstance();
    int port= portBase+ 2 * i;
    InetAddress ipAddr= InetAddress.getByName
(ipAddress);
    SessionAddress localAddr= new SessionAddress
(InetAddress.getLocalHost(), port);
    SessionAddress destAddr= new SessionAddress
```

```
( ipAddr port);
    rtpMgrs[ j]. initialize( localAddr);
    rtpMgrs[ j]. addTarget( destAddr);
    System. err println( "Created RTP session: "
+ ipAddress + " " + port);

    SendStream sendStream = rtpMgrs[ j]. cre-
ateSendStream( dataOutput, i);
    sendStream. start();
    } catch ( Exception e) { return e. getMessage
(); }
    }
    return null;
    }
```

要完成项目1应把握住3个关键之处:跨平台、视音频多轨传输和实时性。

①跨平台。

这里的平台不单指操作系统,也指摄像头的硬件平台。这方面JAVA提供了良好的操作系统跨平台性,它提供了对几乎所有摄像头硬件平台的支持,并且提供搜索设备和检测设备的API。

②视音频多轨传输

JMF提供了先进的媒体处理能力,从而扩展了Java平台的功能。JMF的API主要由一些接口组成,这些接口定义了用于捕获、处理和播放基于时间的媒体对象行为和相互作用的过程。JMF的媒体播放器利用数据源(DataSources)对象来进行媒体内容的传输,而数据源对象封装了该媒体的位置信息和能够播放该媒体的软件和相关协议信息。数据源通常用2种方式来定义,媒体定位器或URL,媒体定位器类似于URL而且可以创建自一个URL,但是必须在系统上安装能够识别URL的协议。数据源可以管理一组源数据流对象。标准的数据源是以一定数量字节作为一个传输单位的。而缓冲数据源(Buffer Data Source)用一个缓冲对象作为传输单位。JMF提供的一种特殊数据源——合并数据源(merging data sources),它可以将来自于多个数据源的源数据流合并为一个数据源,可以对一系列得数据源进行统一管理,可以调用管理器(manager)的createMergingDataSource方法并传递相应的数据源来创建一个合并数据源。

③实时性

要通过在JMF中实现RTP协议。JMF可以实现RTP媒体流的回放和传输,这主要由javax.media.rtp、javax.media.rtp.event和javax.media.rtp.rtcp包中定义的API完成。JMF可以通过标准的JMF plug-in机制来实现支持特定的RTP格式和动态负载。

以上是程序设计中的一些结构和关键组件实现的简要说明,图2.3.4.5是该项目测试运行的部份截面图。

图2 服务器运行界面

图3 客户端网络端口设置

图 4　客户端视音频设备设置

图 5　用户列表及文字聊天界面

3　结论

本项目基本上实现了局域网内的跨平台视音频实时传输,但还存在几个问题需要继续研究: 虽然 JAVA保证了项目的跨平台性,但由于硬件设备的差异 JMF却不完全是跨平台的,致使本项目的跨平台性并不纯粹; 基于 JMF的视音频录像功能还未实现; 项目健壮性有待提高,特别是在反复开关视频数据源时,由于流程复杂以及在多线程处理上的漏洞,使得程序运行的稳定性有待提高。

尽管如此,本项目仍是具有较大的前景和意义的。今后电子办公、视频会议必将得到广泛的应用,而且一个单位里的机器型号样式会越来越多样化,但基于 JMF的跨平台视频会议系统将会满足这一需求; 并且随着软硬件配置的提升, JAVA视频会议系统的视频采集、传输等的性能将逐步赶上基于 C开发的视频会议系统。在流媒体的领域里,重点不应是只放在几个孤立的关键技术上,而是应该把流媒体当作一个系统工程,编码、传输、分享、网络以及设备都是互相联系的一个整体。如何在这样一个系统里,最有效地将流媒体以一种最适合用户终端设备的形式传送给用户,并且不增加服务器和网络负担,将是能否在流媒体领域的竞争中立于不败之地的根本。

参考文献:

[1] 兰荪, 卓力. 小波编码与网络视频传输[M]. 北京: 科学出版社, 2005

[2] 朱鹏, 李春文. 基于 RTP的网络视频传输系统的设计与实现[D]. 计算机工程与应用, 2003

[3] 钟玉琢, 向哲, 沈洪. 流媒体和视频服务器[M]. 北京: 清华大学出版社, 2003

[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[J]. 微电子学与计算机, 2005(8): 138－140