一、流媒体概念

流媒体包含广义和狭义两种内涵：广义上的流媒体指的是使音频和视频形成稳定和连续的传输流和回放流的一系列技术、方法和协议的总称，即流媒体技术；狭义上 的流媒体是相对于传统的下载-回放方式而言的，指的是一种从 Internet 上获取音频和视频等多媒体数据的新方法，它能够支持多媒体数据流的实时传输和 实时播放。通过运用流媒体技术，服务器能够向客户机发送稳定和连续的多媒体数据流，客户机在接收数据的同时以一个稳定的速率回放，而不用等数据全部下载完 之后再进行回放。

二、流媒体协议

实时传输协议（Real- time Transport Protocol，PRT）是在 Internet 上处理多媒体数据流的一种网络协议，利用它能够在一对一（unicast，单播）或者一对多 （multicast，多播）的网络环境中实现传流媒体数据的实时传输。RTP 通常使用 UDP 来进行多媒体数据的传输，但如果需要的话可以使用 TCP 或者 ATM 等其它协议，整个 RTP 协议由两个密切相关的部分组成：RTP 数据协议和 RTP 控制协议。实时流协议（Real Time Streaming Protocol，RTSP）最早由 Real Networks 和 Netscape 公司共同提出，它位于 RTP 和 RTCP 之上，其目的是希望通过 IP 网络有效地传输多媒体数据。

2.1RTP 数据协议

RTP 数据协议负责对流媒体数据进行封包并实现媒体流的实时传输，每一个 RTP 数据报都由**头部（Header）和负载（Payload）**两个部分组成，其中头部**前 12 个字节**的含义是固定的，而负载则可以是音频或者视频数据。RTP 数据报的头部格式如图 1 所示：

图 1 RTP 头部格式

其中比较重要的几个域及其意义如下：

CSRC 记数（CC） 表示 CSRC 标识的数目。CSRC 标识紧跟在 RTP 固定头部之后，用来**表**

示 RTP 数据报的来源，RTP 协议**允许在同一个会话中存在多个数据源**，它们可以 通过 **RTP 混合器**合并为一个数据源。例如，可以产生一个 CSRC 列表来表示一个电话会议，该会议通过一个 RTP 混合器将所有讲话者的语音数据组合为一个 RTP 数据源。

负载类型（PT） 标明 RTP 负载的格式，包括所采用的**编码算法、采样频率、承载通道**等。例如，类型 2 表明该 RTP 数据包中承载的是用 ITU G.721 算法编码的语音数据，采样频率为 8000Hz，并且采用单声道。

序列号 用来为接收方提供探测数据丢失的方法，但如何处理丢失的数据则是应用程序自己的事情，**RTP 协议本身并不负责数据的重传**。

时间戳 记录了**负载中第一个字节的采样时间**，接收方根据时间戳能够确定数据的到达是否受到了延迟抖动的影响，但具体**如何来补偿延迟抖动则是应用程序自己的事情**。

从 RTP 数据报的格式不难看出，它包含了传输媒体的类型、格式、序列号、时间戳以及是否有附加数据等信息，这些都为实时的流媒体传输提供了相应的基础。RTP 协议 的目的是提供实时数据（如交互式的音频和视频）的端到端传输服务，因此在 RTP 中没有连接的概念，它可以建立在底层的面向连接或面向非连接的传输协议之 上；RTP 也不依赖于特别的网络地址格式，而仅仅只需要底层传输协议支持组帧（Framing）和分段（Segmentation）就足够了；另外 RTP 本身还不提供任何可靠性机制，这些都要由传输协议或者应用程序自己来保证。在典型的应用场合下，RTP 一般是在传输协议之上作为应用程序的一部分加以实现的，如图 2 所示：

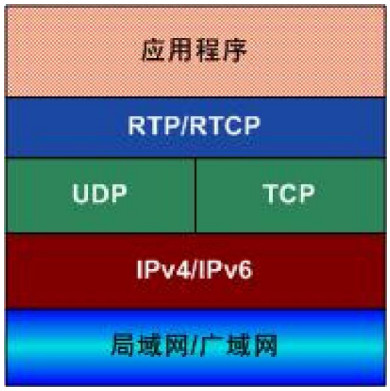


图 2 RTP 与各种网络协议的关系

2.2 RTCP 控制协议

RTCP 控制协议需要与 RTP 数据协议一起配合使用，**当应用程序启动一个 RTP 会话时将同时占用两个端口，分别供 RTP 和 RTCP 使用**。RTP 本身并不能为**按序**传输数据包提供**可靠**的保证，也不提供**流量控制和拥塞控制，这些都由 RTCP 来负责完成**。通常 RTCP 会采用与 RTP **相同的分发机制**，向会话中的**所有**成员**周期性**地发送控制信息，应用程序通过接收这些数据，从中获取会话参与者的相关资料，以及网络状况、分组丢失概率等反馈信息，从而能够对服

务质量进行控制或者对网络状况进行诊断。

RTCP 协议的功能是通过不同的 RTCP 数据报来实现的，主要有如下几种类型：

SR **发送端报告**，所谓发送端是指发出 RTP 数据报的应用程序或者终端，发送端同时也可以是接收端。

RR **接收端报告**，所谓接收端是指仅接收但不发送 RTP 数据报的应用程序或者终端。
SDES **源描述**，主要功能是作为会话成员有关标识信息的载体，如用户名、邮件地址、电话号码等，此外还具有向会话成员传达会话控制信息的功能。
BYE **通知离开**，主要功能是指示某一个或者几个源不再有效，即通知会话中的其他成员**自己**将退出会话。
APP 由应用程序自己定义，解决了 RTCP 的**扩展性**问题，并且为协议的实现者提供了很大的灵活性。

RTCP 数据报携带有**服务质量监控**的必要信息，能够对服务质量进行动态的调整，并能够对网络拥塞进行有效的控制。由于 **RTCP 数据报采用的是多播方式**，因此会话中的所有成员都可以**通过 RTCP 数据报返回的控制信息，来了解其他参与者的当前情况**。

在一个典型的应用场合下，发送媒体流的应用程序将周期性地产生发送端报告 **SR**，该 RTCP 数据报含有不同媒体流间的**同步信息**，以及**已经发送的数据报和字节的计数**，接收端根据这些信息可以估计出实际的数据传输速率。另一方面，接收端会向所有已知的发送端发送接收端报告 **RR**，该 RTCP 数据报含有**已接收数据报的最大序列号、丢失的数据报数目、延时抖动和时间戳**等重要信息，发送端应用根据这些信息可以估计出往返时延，并且可以根据数据报丢失概率和时延抖动情况**动态调整发送速率**，以改善网络拥塞状况，或者根据网络状况平滑地调整应用程序的服务质量。

2.3 RTSP 实时流协议

作为一个应用层协议，RTSP 提供了一个可供扩展的框架，它的意义在于使得实时流媒体数据的受控和点播变得可能。总的说来，RTSP 是一个流媒体表示协议， 主要用来控制具有实时特性的数据发送，但它本身并不传输数据，而是必须依赖于下层传输协议所提供的某些服务。RTSP 可以对流媒体提供诸如**播放、暂停、快进**等操作，它负责定义**具体的控制消息、操作方法、状态码**等，此外还**描述了与 RTP 间的交互操作**。

RTSP 在制定时较多地参考了 HTTP/1.1 协议，甚至许多描述与 HTTP/1.1 完全相同。RTSP 之所以特意使用与 HTTP/1.1 类似的语法和操作，在很大程度上是为了兼容现 有的 Web 基础结构，正因如此，HTTP/1.1 的扩展机制大都可以直接引入到 RTSP 中。

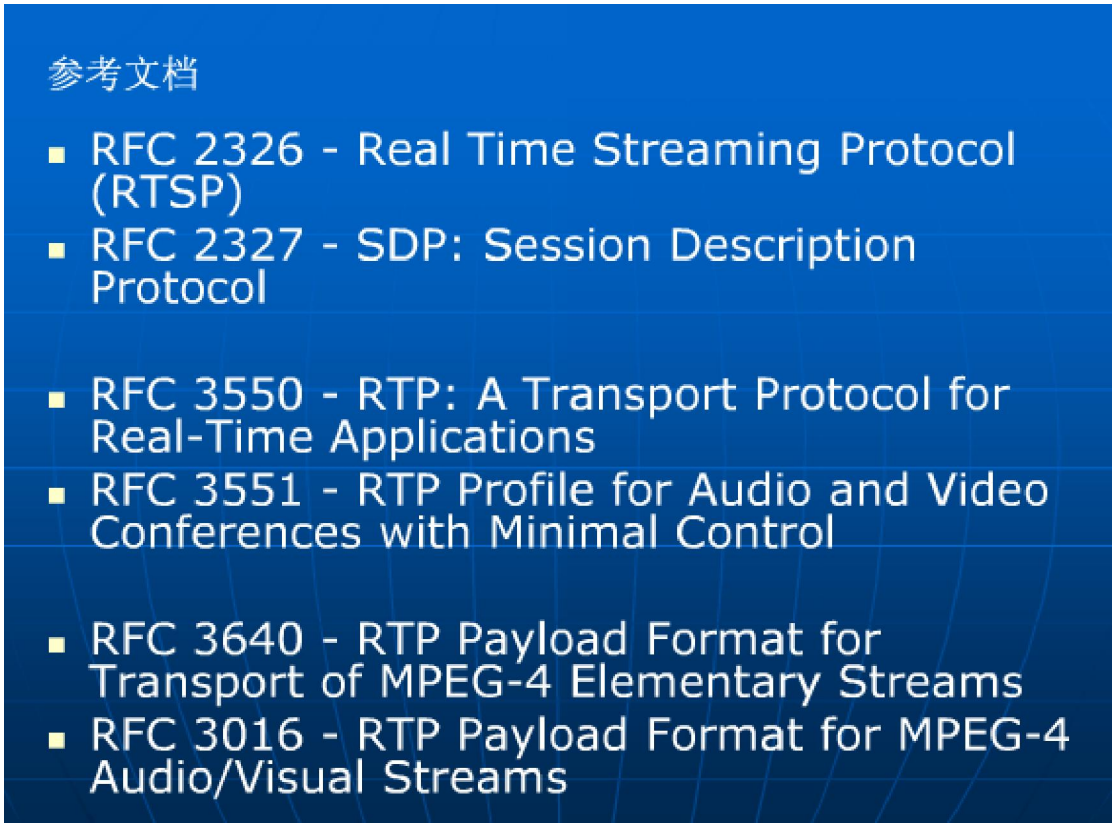
由 RTSP 控制的媒体流集合可以用表示描述 （Presentation Description）来定义，所谓表示是指流媒体服务器提供给客户机的一个或者多个媒体流的集合，而表示描述则包含了一个表示中各个媒体流的相关信息，如数据编码/解码算法、网络地址、媒体流的内容等。

虽然 RTSP 服务器同样也使用标识符来区别每一流连接会话（Session），但 RTSP 连接并没有被绑定到传输层连接（如 TCP 等），也就是说在整个 RTSP 连接期间，RTSP 用户可打开或者关闭多个对 RTSP 服务器的可靠传输连接 以发出 RTSP 请求。此外，RTSP 连接也可以基于面向无连接的传输协议（如 UDP 等）。

RTSP 协议目前支持以下操作：

检索媒体　　允许用户通过 HTTP 或者其它方法向媒体服务器提交一个表示描述。如表示是组播的，则表示描述就包含用于该媒体流的组播地址和端口号；如果表示是单播的，为了安全在表示描述中应该只提供目的地址。
邀请加入　　媒体服务器可以被邀请参加正在进行的会议，或者在表示中回放媒体，或者在表示中录制全部媒体或其子集，非常适合于分布式教学。
添加媒体　　通知用户新加入的可利用媒体流，这对现场讲座来讲显得尤其有用。与 HTTP/1.1 类似，RTSP 请求也可以交由代理、通道或者缓存来进行处理。

参考文档



三、JRTPlib3.7.1 在 windows 下的编译步骤

编译设置过程：
1.解压 jrtplib-3.7.1 和 jthread-1.2.1
2.用 VC 打开工程文件 jthread.dsw

3. 编译 jrtplib.lib 和 jthread.lib 需要注意 VC6 要求安装 Vs6sp6，在编译 jrtplib.lib 和 jthread.lib 前，在 project——settings——C/C++——Code generation:use run-time library 中，对于 debug，选择:Debug Multithreaded DLL，对于 release，则选择:Multithreaded DLL。
4.首先编译 jthread 库,然后将 jthread-1.2.1\src 内的"jmutex.h"和"jthread.h"两个头文件放入 jrtplib-3.7.1\src 目录下,然后将 jrtplib-3.7.1\src 文件夹下所有头文件中的<jmutex.h>和<jthread.h>语句修改为"jmutex.h"和"jthread.h",需要修改的文件为 rtpudpv4transmitter.h、rtpsession.h 和 rtppollthread.h。编译时注意编译方式和 jthread.lib 一致。
5.编译生成的 jthread.lib 和 jrtplib.lib 拷贝到系统目录:C:\Program Files\Microsoft Visual Studio\VC98\Lib 下，将 jrtplib-3.7.1\src 下所有的.h 头文件复制到 C:\Program Files\Microsoft Visual Studio\VC98\Include，以便以后使用。
6.现在我们就可以编译 jrtplib-3.7.1\examples 下的实例程序了。建立 VC 工程，打开 example1.c，在 Project Settings 的 link 页添加 jthread.lib jrtplib.lib ws2_32.lib，在 project——settings——C/C++——Code generation:use run-time library 中，对于 debug，选择:Debug Multithreaded DLL，对于 release，则选择:Multithreaded DLL。
7.编译源程序，运行就 OK 啦

四、JRTPLIB 库的使用方法及程序实现

（1）JRTPLIB 函数的使用

a、在使用 JRTPLIB 进行实时流媒体数据传输之前，首先应该**生成 RTPSession 类的一个实例**来表示此次 RTP 会话，然后调用 Create() 方法来对其进行初始化操作。JRTPLIB-3.7 中的 Create 方法为

Create (sessparams,&transparams)。其中的两个参数需要如下先定义：

```
RTPUDPv4TransmissionParams transparams;
RTPSessionParams sessparams;
sessparams.SetOwnTimestampUnit(1.0/8000.0);/*设置时间戳，1/8000 表示 1 秒钟采样 8000 次，即录音时的 8KHz*/
sessparams.SetAcceptOwnPackets(true);
transparams.SetPortbase(portbase);/*本地通讯端口*/
```

b、设置恰当的时戳单元，是 RTP 会话初始化过程所要进行的另外一项重要工作，这是通过调用 RTPSession 类的 SetTimestampUnit() 方法来实现的，前面已经提过。

c、当 RTP 会话成功建立起来之后，接下去就可以开始进行流媒体数据的实时传输了。首先需要**设置好数据发送的目标地址**，RTP 协议**允许同一会话存在多个目标地址**，这可以通过调用 RTPSession 类的 AddDestination()、DeleteDestination() 和 ClearDestinations() 方法来完成。例如，下面的语句表示的是让 RTP 会话将数据发送到本地主机的 6000 端口：

```
unsigned long addr = ntohl(inet_addr("127.0.0.1"));
sess.AddDestination(addr, 6000);
```

d、目标地址全部指定之后，接着就可以调用 RTPSession 类的 **SendPacket()** 方法，向所有的目标地址发送流媒体数据。SendPacket() 是 RTPSession 类提供的一个重载函数对于同一个 RTP 会话来讲，**负载类型、标识和时戳增量**通常来讲都是相同的，JRTPLIB 允许将它们**设置为会话的默认参数**，这是通过调用 RTPSession 类的 SetDefaultPayloadType()、SetDefaultMark() 和 SetDefaultTimeStampIncrement() 方法来完成的。为 RTP 会话设置这些默认参数的好处是可以简化数据的发送，例如，如果为 RTP 会话设置了默认参数：

```
sess.SetDefaultPayloadType(0);
sess.SetDefaultMark(false);
sess.SetDefaultTimeStampIncrement(10);
```

之后在进行数据发送时**只需指明要发送的数据及其长度**就可以了：

```
sess.SendPacket(buffer, 5);
```

在真正的语音传输中，上面的 buffer 就是我们录音时所得到的 buffer。使用上面的函数可以简单的发送，但无法真正的实现 RTP 传输，我们需要调用另一个接口：`sess.SendPacket((void *)buffer,sizeof(buffer),0,false,8000);`详细的说明可以查看 JRTPLIB 的说明文档。

e、对于流媒体数据的接收端，首先需要调用 RTPSession 类的 **Poll ()** 方法来**接收**发送过来的 RTP 或者 RTCP 数据报。

由于同一个 RTP 会话中允许有多个参与者（源），你既**可以通过调用 RTPSession 类的 GotoFirstSource() 和 GotoNextSource() 方法来遍历所有的源，也可以通过调用 RTPSession 类的 GotoFirstSourceWithData() 和 GotoNextSourceWithData() 方法来遍历那些携带有数据的源。**在从 RTP 会话中检测出有效的数据源之后，接下去就可以**调用 RTPSession 类的 GetNextPacket() 方法从中抽取 RTP 数据报**，当接收到的 RTP 数据报处理完之后，**一定要记得及时释放。**

JRTPLIB 为 RTP 数据报定义了**三种接收模式**，其中每种接收模式都具体规定了哪些到达的 RTP 数据报将会被接受，而哪些到达的 RTP 数据报将会被拒绝。通过调用 RTPSession 类的 **SetReceiveMode()** 方法可以设置下列这些接收模式：

RECEIVEMODE_ALL　　**缺省的接收模式**，所有到达的 RTP 数据报都将被接受；

RECEIVEMODE_IGNORESOME　　除了某些特定的发送者之外，所有到达的 RTP 数据报都将被接受，而被拒绝的发送者列表可以通过调用 AddToIgnoreList()、DeleteFromIgnoreList() 和 ClearIgnoreList() 方法来进行设置；

RECEIVEMODE_ACCEPTSOME　除了某些特定的发送者之外，所有到达的 RTP 数据报都将被拒绝，而被接受的发送者列表可以通过调用 AddToAcceptList ()、DeleteFromAcceptList 和 ClearAcceptList () 方法来进行设置。

下面是采用第三种接收模式的程序示例。

```
if (sess.GotoFirstSourceWithData()) {
do {
    sess.AddToAcceptList(remoteIP, allports,portbase);
          sess.SetReceiveMode(RECEIVEMODE_ACCEPTSOME);
   RTPPacket *pack;
   pack = sess.GetNextPacket();              // 处理接收到的数据
   delete pack;   }
while (sess.GotoNextSourceWithData());
}
```

完整的代码中，首先需调用 Poll（）方法接收 RTP 数据报，然后**在 BeginDataAccess()和 EndDataAccess()之间进行数据接收的操作**。此时，我们设定程序一直 do-while 等待并处理数据

（2）程序流程

发送：获得接收端的 IP 地址和端口号　创建 RTP 会话　指定 RTP 数据接收端 设置 RTP 会话 默认参数　发送流媒体数据

接收：获得用户指定的端口号 创建 RTP 会话 设置接收模式 接受 RTP 数据 检索 RTP 数据源 获取 RTP 数据报 删除 RTP 数据报

五、RTPSession 类介绍

对于大多数的 RTP 应用程序，RTPSession 类可能是 JRTPLIB 唯一使用的类。它能完全处理 RTCP 部份的数据包，所以用户可以把精力集中在真正的数据收发。

要知道 RTPSession 类在多线程下并不是安全的，因此，用户要通过某些锁同步机制来保证不会出现在不同线程当中调用同一个 RTPSession 实例。

RTPSession 类有如下的接口。

• RTPSession(RTPTransmitter::TransmissionProtocol proto = RTPTransmitter::IPv4UDPProto)

使用 proto 类型传输层创建一个 PRTSession 实例。如果 proto 使用用户自定义（user-defined）传输层，则相应的 NewUserDefinedTransmitter()函数必须实现。ps:这里默认就行了，默认就是 IPV4 网络。

• int Create(const RTPSessionParams &sessparams, const RTPTransmissionParams*transparams = 0)

使用 RTPSession 参数 sessparams 和 RTPTransmission 参数 transparams 真正创建一个 RTP 会话。如果 transparams 为 NULL，则使用默认的参数。ps:RTPSessionParams 我们可能要设得比较多，RTPTransmissionParams 参数就只要设置其中的端口就行了，端口一定要设对，不然进行组播时，这个进程将不接收数据。设置方式可以看 example.cpp。

• void Destroy()

离开一个会话但不向其它组成员发送 BYE 包。ps:我不推荐用这个函数除非是错误处理，正常离开我们应该用 ByeDestroy()。

• void BYEDestroy(const RTPTime &maxwaittime, const void *reason,size t reasonlength)

发送一个 BYE 包并且离开会话。在发送 BYE 包前等待 maxwaittime，如果超时，会不发送 BYE 包直接离开，BYE 包会包含你的离开原因 reason。相应的 reasonlength 表示 reason 长度。ps:因为 BYE 包是一个 RTCP 包，RTCP 不是要发就发的，它的发送时间是为了平衡带宽通过计算得出来的，那就很有可能到了要发的时候以经超过了 maxwaittime 时间了，作者可能认一直保留个这会话这么久没意义。当然，我常常把 maxwaittime 设得很大。

• bool IsActive()

看看这个 RTPSession 实例是否以经通过 Create 建立了真实的会话。

• uint32 t GetLocalSSRC()

返回我们的 SSRC。ps:至于什么是 SSRC，去看看 RFC3550 吧。我说过 JRTPLIB 只是 RTP 协议的包装，并没有做任何应用的事情。

• int AddDestination(const RTPAddress &addr)

添加一个发送目标。ps: 当然，如果我们使用组播，这里只用调用一次，把我们的组播地址写进去。这样，这组的全部人都能收到你发的包。但是组播可因特网的上设置很烦。而

且用组播测 试也很烦（组播必须 BIND 一个端口，如果你想在同一台机器上运行两个软件实例来没试，你就会发现同一个端口 BIND 两次，当然，后面那次会失败，也就是 说测试

不了，要测？找两台机器，或用虚拟机 ），如果组播不满足，我们就要把组播变在单播，这时就要返复调用这个函数把其它组成员的 IP 都加进来了。具体可以看看 example3.cpp。

• int DeleteDestination(const RTPAddress &addr)
从发送地址列表中删除一下地址。

• void ClearDestinations()
清除发送地址列表。

• bool SupportsMulticasting()
返回 JRTPLIB 是否支持组播。ps:这里指 JRTPLIB 本身，不是你的真实网络。编译 JRTPLIB 库时可能指定。
.

• int JoinMulticastGroup(const RTPAddress &addr)
加入一个组播组 addr。

• int LeaveMulticastGroup(const RTPAddress &addr)
离开一个组播组 addr。

• void LeaveAllMulticastGroups()
离开所有组播组。ps:我们可以同时加入多个组播组。.

• int SendPacket(const void *data, size t len)
• int SendPacket(const void *data, size t len, uint8 t pt, bool mark,uint32 t timestampinc)
• int SendPacketEx(const void *data, size t len, uint16 t hdrextID,const void *hdrextdata, size t numhdrextwords)
• int SendPacketEx(const void *data, size t len, uint8 t pt, boolmark, uint32 t timestampinc, uint16 t hdrextID, const void *hdrextdata,size t numhdrextwords)
上面的 4 个函数都是发送数据包的，我想如果你没有看 RTP 协议，我说了你也晕。如果你 RTP 协议看了，再看看 RTPSession.h 的注识，你就懂了。

• int SetDefaultPayloadType(uint8 t pt)
设定默认的 RTP PayloadType 为 PT。ps:和上面的第一个和第三个发送函数配套。至于应该设个什么数，如果你看 BAIDU 上乱七八糟的文章，当然的乱设就可能了。其实应该按 RFC3551，根据你要传输的媒体类型来设。

• int SetDefaultMark(bool m)
这设 RTP 数据包的 Mark 标识。ps:设为什么值好？这个，呵呵，连 RFC3550 也不能确定了。要看具体的 RTP Payload 规范，MPEG 的，H263 的都不一样。
MPEG2 www.ietf.org/rfc/rfc2250.txt
MPEG4 www.rfc-editor.org/rfc/rfc3016.txt

H263 www.ietf.org/rfc/rfc2190.txt

• int SetDefaultTimestampIncrement(uint32 t timestampinc)

设置默认的时间戳的增量。ps:也是和上的第一和第三个函数配套的。每发一个 RTP 数据包 timestamp 就会自动增加

• int IncrementTimestamp(uint32 t inc)

这个函数用来手工增加 Timestamp。有时我这很好用，例如，一个 RTP 数据包因为只含有静音数据，我们没有发送，这是我们就应手工增加 Timestamp 以便发下一个 RTP 数据包时它的 Timestamp 是正确的。

• int IncrementTimestampDefault()

这个函数用于增加由 SetDefaultTimestampIncrement 设定的值。有时候这很有用，例如，一个 RTP 数据包因为只含有静音数据，我 们没有发送。这时，这个函数就会被调用用来设置 Timestamp 以便下一个 RTP 包的 Timestamp 是正确的。

• int SetPreTransmissionDelay(const RTPTime &delay)

This function allows you to inform the library about the delay between sampling the first sample of a packet and sending the packet. This delay is taken into account when calculating the relation between RTP timestamp and wallclock time, used for inter-media synchronization.

• RTPTransmissionInfo *GetTransmissionInfo()

This function returns an instance of a subclass of RTPTransmissionInfo which will give some additional information about the transmitter (a list of local IP addresses for example). The user has to delete the returned instance when it is no longer needed.

• int Poll()

If you’re not using the poll thread, this function must be called regularly to process incoming data and to send RTCP data when necessary.

• int WaitForIncomingData(const RTPTime &delay,bool *dataavailable= 0)

Waits at most a time delay until incoming data has been detected. Only works when you’re not using the poll thread. If dataavailable is not NULL, it should be set to true if data was actually read and to false otherwise.

• int AbortWait()

If the previous function has been called, this one aborts the waiting. Only works when you’re not using the poll thread.

• RTPTime GetRTCPDelay()

Returns the time interval after which an RTCP compound packet may have

to be sent. Only works when you're not using the poll thread.

• int BeginDataAccess()

下面的函数（直到 EndDataAccess）要在 BeginDataAccess 和 EndDataAccess 之间被调用，BeginDataAccess 确保轮询(poll）线程不会在这期间访问 source table 。EndDataAccess 调用完成后，轮询(poll)线程会得到锁而继续访问。ps:首先，你里的 source table 中的每一个 source 表示参与会议中的每一个参与者的每一个独立的媒体流。我们会在下面用到他们，但同时，poll 线程也会轮询它们以正确处理和 RTCP 有关的内容。

• bool GotoFirstSource()

开始递归参与者的第一个流，如果找到了，就返回 tree,否则返回 false。ps：我们通过这个函数和下面的 GotoNextSource 遍历 source table 中的每一个 source。

• bool GotoNextSource()

设置当前的源（source）为 source table 中的下一个源。如果已经到尾部了就返回 false.

• bool GotoPreviousSource()

设置当前的源（source）为 source table 中上一个源。如果已经到头部了就返回 false.

• bool GotoFirstSourceWithData()

开始递归参与者中第一个有 RTP 数据的流，如果找到了，就返回 tree,否则返回 false。PS：在接收数据是我们常用的是这套函数，因为如果没有数据要来都没用。

• bool GotoNextSourceWithData()

设置当前的源（source）为 source table 中有 RTP 数据的下一个源。如果已经到尾部了就返回 false.

• bool GotoPreviousSourceWithData()

设置当前的源（source）为 source table 中有 RTP 数据的上一个源。如果已经到头部了就返回 false.

• RTPSourceData *GetCurrentSourceInfo()

返回当前参与者的当前源（source)的 RTPSourceData 实列。ps： 返回的这个 RTPSourceData 就是本进程从期它参与者的 RTCP 数据包中收集得到的信息，对我们来说其实很有用，只是作者的例程没有用上，国内的网络也没有提到。在 RFC3550 中有 关 RTCP 的东西都在这了，看过 RFC3550 的人都知到，里头谈得最多的就是 RTCP。这个类我们以后会专门说。

• RTPSourceData *GetSourceInfo(uint32 t ssrc)

返回由 ssrc 指定的 RTPSourceData ，或都 NULL（当这个条目不存在)。ps：这个函数也很有用。因为 GetCurrentSourceInfo 只有在 GotoFirstSource 等上下文当中才能用。如果我们

是在 RTPSource 子类的成员函数中，我们没有这个上下文，就只能用这个函数。

• RTPPacket *GetNextPacket()
得到当前参与者当前媒体流的下一个 RTP 数据包。

• int EndDataAccess()
请看 BeginDataAccess

• int SetReceiveMode(RTPTransmitter::ReceiveMode m)
Sets the receive mode to m, which can be one of the following:
– RTPTransmitter::AcceptAll
All incoming data is accepted, no matter where it originated from.
– RTPTransmitter::AcceptSome
Only data coming from specific sources will be accepted.
– RTPTransmitter::IgnoreSome
All incoming data is accepted, except for data coming from a specificset of sources.
Note that when the receive mode is changed, the list of addressed to be ignored or accepted will be cleared.

• int AddToIgnoreList(const RTPAddress &addr)
Adds addr to the list of addresses to ignore.

• int DeleteFromIgnoreList(const RTPAddress &addr)
Deletes addr from the list of addresses to ignore.

• void ClearIgnoreList()
Clears the list of addresses to ignore.

• int AddToAcceptList(const RTPAddress &addr)
Adds addr to the list of addresses to accept.

• int DeleteFromAcceptList(const RTPAddress &addr)
Deletes addr from the list of addresses to accept.

• void ClearAcceptList()
Clears the list of addresses to accept.

• int SetMaximumPacketSize(size t s)
Sets the maximum allowed packet size to s.

• int SetSessionBandwidth(double bw)
Sets the session bandwidth to bw, which is specified in bytes per second.

• int SetTimestampUnit(double u)

Sets our own timestamp unit to u. The timestamp unit is defined as a time interval divided by the number of samples in that interval: for 8000Hz audio this would be 1.0/8000.0.

• void SetNameInterval(int count)

在处理 source table 中的 sourcese 后，RTCP packet builder(我们不用理这个内部的东西）会检查是否有其它（non-CNAME)SDES 项目要发送。如果 count 为零或负数，则不发送，如果 count 为正数，则在 sources table 处理 count 次后会把 SDES name item 加到当前 RTCP 包中。ps: 其实每次处理 sources table 都会伴随都 SDES RTCP 数据包的发送，在这个数据包当中 CNAME 是必须的，但其它的项目不是必须的，这就函数确定了 NAME 项目发送的频度，如果为 1，则表不每个 SDES RTCP 数据包都带着它，如果为 2 则每两个 SDES 数据包就发送一次 NAME 项目，下面的 SetEMailInterval、 SetLocationInterval、SetPhoneInterval、SetToolInterval、SetNoteInterval 都是同一 原理。关于这个 ITEM 的描述，请看 RFC3550.老版本的 JRTPLIB 没有使用这套函数，而是用 EnableSendName()等函数。

• void SetEMailInterval(int count)

After all possible sources in the source table have been processed, the RTCP packet builder will check if other (non-CNAME) SDES items need to be sent. If count is zero or negative, nothing will happen. If count is positive, an SDES e-mail item will be added after the sources in the source table have been processed count times.

• void SetLocationInterval(int count)

After all possible sources in the source table have been processed, the RTCP packet builder will check if other (non-CNAME) SDES items need to be sent. If count is zero or negative, nothing will happen. If count is positive, an SDES location item will be added after the sources in the source table have been processed count times.

• void SetPhoneInterval(int count)

After all possible sources in the source table have been processed, the RTCP packet builder will check if other (non-CNAME) SDES items need to be sent. If count is zero or negative, nothing will happen. If count is positive, an SDES phone item will be added after the sources in the source table have been processed count times.

• void SetToolInterval(int count)

After all possible sources in the source table have been processed, the RTCP packet builder will check if other (non-CNAME) SDES items need to be sent. If count is zero or negative, nothing will happen. If count is positive, an SDES tool item will be added after the sources in the source table have been processed count times.

• void SetNoteInterval(int count)

After all possible sources in the source table have been processed, the RTCP packet builder will check if other (non-CNAME) SDES items need to be sent. If count is zero or negative, nothing will happen. If count is positive, an SDES note item will be added after the sources in the source table have been processed count times.

• int SetLocalName(const void *s, size t len)

设置 NAME SDES 项目，以遍会议的其它人员看到你的名称。下同。

• int SetLocalEMail(const void *s, size t len)

Sets the SDES e-mail item for the local participant to the value s with length len.

• int SetLocalLocation(const void *s, size t len)

Sets the SDES location item for the local participant to the value s with length len.

• int SetLocalPhone(const void *s, size t len)

Sets the SDES phone item for the local participant to the value s with length len.

• int SetLocalTool(const void *s, size t len)

Sets the SDES tool item for the local participant to the value s with length len.

• int SetLocalNote(const void *s, size t len)

Sets the SDES note item for the local participant to the value s with length len.

In case you specified in the constructor that you want to use your own transmission component, you should override the following function:

• RTPTransmitter *NewUserDefinedTransmitter()

The RTPTransmitter instance returned by this function will then be used to send and receive RTP and RTCP packets. Note that when the session is destroyed, this RTPTransmitter instance will be destroyed with a delete call.

By inheriting your own class from RTPSession and overriding one or more of the functions below, certain events can be detected:

• void OnRTPPacket(RTPPacket *pack, const RTPTime &receivetime, const RTPAddress *senderaddress)

如果有 RTPPacket 数据包来到，会调用这个函数处理。ps:这个函数在我们继承 RTPSession 类时很可能重载，这是获取 RTP 数据包除了上面所说的方法以外的另外一种方法，这个方法比较适合异步的情况。默认这个是一个空虚函数。除了这个函数以外，下面的几个函数了会经常重载。

• void OnRTCPCompoundPacket(RTCPCompoundPacket *pack, const RTPTime &receivetime, const RTPAddress *senderaddress)

Is called when an incoming RTCP packet is about to be processed.

• void OnSSRCCollision(RTPSourceData *srcdat, const RTPAddress *senderaddress, bool isrtp)

Is called when an SSRC collision was detected. The instance srcdat is the one present in the table, the address senderaddress is the one that collided with one of the addresses and isrtp indicates against which address of srcdat the check failed.

• void OnCNAMECollision(RTPSourceData *srcdat, const RTPAddress *senderaddress, const uint8 t *cname, size t cnamelength)

Is called when another CNAME was received than the one already present for source srcdat.

• void OnNewSource(RTPSourceData *srcdat)

当有一个新的条目加到 source table 时，调用这个函数。ps: 这也是一个比较重要的函数，因为这意味着很有可能有一个新的与会者加入。但令我很不高兴的是，这时候的 RTPSourceData 里头的 CNAME 和 NAME 等字段都还是无效的，这不是 RTCP 的责任，因为在这个 SDES RTCP 数据包中所有的信息都以经有了（通过抓包证实了这一点）。我们的函数被调用后，需要延时一会才能得到有关这个 Source 的 CNAME 和 NAME 等相关的信息。当然，如果你不想软件死掉，不能在这个函数体内以阻塞的方式延时。

• void OnRemoveSource(RTPSourceData *srcdat)

当有一个条目从 source table 中移除时调用这个函数。ps： 这通常意味着有一个与会者离开了，和 OnNewSource 不一样，这时的 CNAME 和 NAME 等都是有效的。用这个函数要注意，我们的“意味着两个字” 因为“加入”的可能不是一个新的与会者，而是一个现有与会者的一个新的媒体流。“离开”的也可能不是一个与会者，而只是其中一个与会者的其中一个媒体流，这两个函数只能给我们更新与会者提供一个触发条件而已。当 OnNewSource 调用时，我们要看看这个 CNAME 是不是以经在我们与会者名单中，如果不 是，那就是一个新与会者。同时，如果 OnRemoveSource 被调用，则我们要看看这个 CNAME 的与会者还有没有其它的 Source，如果没有了， 这个与会者才是真正离开。这么很麻烦？？那就对了，那就是现在的 H323 和 SIP 要做的事情——会话管理。

• void OnTimeout(RTPSourceData *srcdat)

Is called when participant srcdat is timed out.

• void OnBYETimeout(RTPSourceData *srcdat)

Is called when participant srcdat is timed after having sent a BYE packet.

• void OnBYEPacket(RTPSourceData *srcdat)

Is called when a BYE packet has been processed for source srcdat.

• void OnAPPPacket(RTCPAPPPacket *apppacket, const RTPTime &receivetime, const RTPAddress *senderaddress)

In called when an RTCP APP packet apppacket has been received at time receivetime from address senderaddress.

• void OnUnknownPacketType(RTCPPacket *rtcppack, const RTPTime &receivetime, const RTPAddress *senderaddress)

Is called when an unknown RTCP packet type was detected.

• void OnUnknownPacketFormat(RTCPPacket *rtcppack, const RTPTime &receivetime, const RTPAddress *senderaddress)

Is called when an unknown packet format for a known packet type was detected.

• void OnNoteTimeout(RTPSourceData *srcdat)

Is called when the SDES NOTE item for source srcdat has been timed out.

• void OnSendRTCPCompoundPacket(RTCPCompoundPacket *pack)

Is called when an RTCP compound packet has just been sent. Useful to inspect outgoing RTCP data.

• void OnPollThreadError(int errcode)

Is called when error errcode was detected in the poll thread.

• void OnPollThreadStep()

Is called each time the poll thread loops. This happens when incoming data was detected or when its time to send an RTCP compound packet.

六、环境搭建及编译方法

（1）Toolchain 的安装

首先找到 xscale-arm-toolchain.tgz 文件，假设该文件包放在/tmp/下

#cd /

#tar -zxvf /tmp/xscale-arm-toolchain.tgz

再设置环境变量

#export PATH=/usr/local/arm-linux/bin:$PATH

最后检查一下交叉编译工具是否安装成功

#arm-linux-g++ --version

看是否显示 arm-linux-g++的版本，如有则安装成功。

（2）JRTPLIB 库的交叉编译及安装

首先从 JRTPLIB 的网站（http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.htmll） 下载最新的源码包，此处使用的是 jrtplib-2.8.tar，假设下载后的源码包放在/tmp 下，执 行下面的命令对其解压缩：

#cd /tmp

#tar -zxvf jrtplib-2.8.tar

然后要对 jrtplib 进行配置和编译

#cd jrtplib-2.8
#./configure CC=arm-linux-g++ cross-compile=yes
修改 Makefile 文件
将链接命令 ld 和 ar 改为 arm-linux-ld 和 arm-linux-ar
#make
最后再执行如下命令就可以完成 JRTPLIB 的安装：
#make install
(3)程序编译
a、配置编译环境
可以用 export 来配置，也可以用编写 Makefile 的方法。这里采用 Makefile。
编写 Make*file&:*

```
INCL = -I/usr/local/include
CFLAGS = -pipe -O2 -fno-strength-reduce
LFLAGS = /usr/local/lib/libjrtp.a -L/usr/X11R6/lib
LIBS = -LX11 -LXext /usr/local/lib/libjrtp.a
CC = arm-linux-g++

main:main.o
$(CC) $(LFLAGS) $(INCL) -o main main.o $(LIBS)
main.o:main.cpp

clean:
rm -f main
rm -f *.o

.SUFFIXES:.cpp
.cpp.o:
$(CC) -c $(CFLAGS) $(INCL) -o $@ $<        /*  $@表示目标的完整名字      */
    /* $<表示第一个依赖文件的名字  */
```

b、编译
假设发送和接收程序分别放在/tmp/send 和/tmp/receive 目录下
#cd /tmp/send
#make
#cd /tmp/receive
#make

七、易出错误及注意问题

1、找不到一些标准的最 基本的一些头文件。
主要是因为 Toolchain 路径没安装对，要 严格按照步骤安装。
2、找不到使用的 jrtplib 库中的一些头文件。
在 jrtplib 的安装目录下，include 路径下不能再有别的目录。
3、recieve 函数接收数据包不能正确提出所要数据。
由于每一个 RTP 数据报都由头部（Header）和负载（Payload）两个部分组成，若使用 getrawdata()

是返回整个数据包的数据，包含传输媒体的类型、格式、序列号、时间戳以及是否有附加数据等信息。getpayload()函数是返回所发送的数据。两者一定要分清。
4、设置 RECEIVEMODE_ACCEPTSOME　　接收模式后，运行程序接收端不能接包。
IP 地址格式出了问题。iner_addr()与 ntohl()函数要用对，否则参数传不进去，接受列表中无值，当然接收不了数据包。
5、编译通过，但测试时接收端不能接收到数据。
可能是接收机防火墙未关闭。运行：
#iptables -F
也可能是 IP 地址没有设置好。运行：
#ifocnfig eth0　*.*.*.*　netmask *.*.*.*
6、使用 jrtolib 库时，在程序中 include 后最好加上库所在的路径。

八、程序

send:

```
#include <stdio.h>
#include <string.h>
#include "rtpsession.h"

// 错误处理函数
void checkerror(int err)
{
if (err < 0) {
 char* errstr = RTPGetErrorString(err);
 printf("Error:%s\\n", errstr);
 exit(-1);
}
}

int main(int argc, char** argv)
{
RTPSession sess;
unsigned long destip;
int destport;
int portbase = 6000;
int status, index;
char buffer[128];

if (argc != 3) {
 printf("Usage: ./sender destip destport\\n");
 return -1;
}
```

```
// 获得接收端的 IP 地址和端口号
destip = inet_addr(argv[1]);
if (destip == INADDR_NONE) {
 printf("Bad IP address specified.\\n");
 return -1;
}
destip = ntohl(destip);
destport = atoi(argv[2]);

// 创建 RTP 会话
status = sess.Create(portbase);
checkerror(status);

// 指定 RTP 数据接收端
status = sess.AddDestination(destip, destport);
checkerror(status);

// 设置 RTP 会话默认参数
sess.SetDefaultPayloadType(0);
sess.SetDefaultMark(false);
sess.SetDefaultTimeStampIncrement(10);

// 发送流媒体数据
index = 1;
do {
 sprintf(buffer, "%d: RTP packet", index ++);
 sess.SendPacket(buffer, strlen(buffer));
 printf("Send packet !\\n");
} while(1);

return 0;
}
```

receive:

```
#include <stdio.h>
#include "rtpsession.h"
#include "rtppacket.h"
```

```
// 错误处理函数
void checkerror(int err)
{
if (err < 0) {
 char* errstr = RTPGetErrorString(err);
 printf("Error:%s\\n", errstr);
 exit(-1);
}
}

int main(int argc, char** argv)
{
RTPSession sess;
int localport,portbase;
int status;
unsigned long remoteIP;
if (argc != 4) {
 printf("Usage: ./sender localport\\n");
 return -1;
}

// 获得用户指定的端口号

remoteIP = inet_addr(argv[1]);
localport = atoi(argv[2]);
portbase = atoi(argv[3]);
// 创建 RTP 会话
status = sess.Create(localport);
checkerror(status);

//RTPHeader *rtphdr;
unsigned long timestamp1;
unsigned char * RawData;
unsigned char temp[30];
int lengh ,i;
bool allports = 1;

sess.AddToAcceptList(remoteIP, allports,portbase);

  do {
//设置接收模式
    sess.SetReceiveMode(RECEIVEMODE_ACCEPTSOME);
sess.AddToAcceptList(remoteIP, allports,portbase);
```

```
 // 接受 RTP 数据
 status = sess.PollData();


// 检索 RTP 数据源
 if (sess.GotoFirstSourceWithData()) {
  do {

    RTPPacket* packet;
    // 获取 RTP 数据报
    while ((packet = sess.GetNextPacket()) != NULL) {
     printf("Got packet !\n");

timestamp1 = packet->GetTimeStamp();
lengh=packet->GetPayloadLength();
RawData=packet->GetPayload();

for(i=0;i<lengh;i++){
  temp[i]=RawData[i];
printf("%c",temp[i]);
}
temp[i]='\0';
printf("   timestamp: %d lengh=%d data:%s\n",timestamp1,lengh,&temp);
     // 删除 RTP 数据报

     delete packet;
    }
  } while (sess.GotoNextSourceWithData());
 }
} while(1);

return 0;
}
```