电子科技大学

硕士学位论文

基于RTP协议流媒体服务器的研究

姓名：张洪宇

申请学位级别：硕士

专业：软件工程

指导教师：李毅;童亮

20070607

摘 要

流媒体技术的应用日益广泛，对流媒体技术的研究具有很大的实际意义,本文通过对 RTP/RTCP 协议的研究，分析流媒体服务器的一般功能和结构，给出构建一个基本的流媒体服务器的实现方案，实验证明可以同时满足多个实时和文件客户的要求。

实时传输协议（RTP）为数据提供了具有实时特征的端对端传送服务，如在组播或单播网络服务下的交互式视频音频或模拟数据。应用程序通常在 UDP 上运行 RTP 以便使用其多路结点和校验服务；这两种协议都提供了传输层协议的功能。但是 RTP 可以与其它适合的底层网络或传输协议一起使用。如果底层网络提供组播方式，那么 RTP 可以使用该组播表传输数据到多个目的地。

RTP 本身并没有提供按时发送机制或其它服务质量（QoS）保证，它依赖于低层服务去实现这一过程。 RTP 并不保证传送或防止无序传送，也不确定底层网络的可靠性。 RTP 实行有序传送， RTP 中的序列号允许接收方重组发送方的包序列，同时序列号也能用于决定适当的包位置，例如：在视频解码中，就不需要顺序解码。

RTP 由两个紧密链接部分组成：

- RTP — 传送具有实时属性的数据；
- RTP 控制协议（RTCP） — 监控服务质量并传送正在进行的会话参与者的相关信息。RTCP 第二方面的功能对于“松散受控”会话是足够的，也就是，在没有明确的成员控制和组织的情况下，它并不非得用来支持一个应用序的所有控制通信请求。

对于特定的应用，RTP 协议是可扩展的。所以 RTP 协议只是一个框架，并且有意被定义为如此。在实际应用时，RTP 协议的包头可以被修改来得到所需的功能，而不是像传统协议那样靠不断修改并使其统一来变得更完善。

关键字：RTP ，QoS， RTCP，UDP

Abstract

This memorandum specifies the real-time transport protocol (RTP), which provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. Those services include payload type identification, sequence numbering, timestamping and delivery monitoring. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services; both protocols contribute parts of the transport protocol functionality. However, RTP may be used with other suitable underlying network or transport protocols (see Section 10). RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network.

Note that RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. The sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence, but sequence numbers might also be used to determine the proper location of a packet, for example in video decoding, without necessarily decoding packets in sequence.

While RTP is primarily designed to satisfy the needs of multi-participant multimedia conferences, it is not limited to that particular application. Storage of continuous data, interactive distributed simulation, active badge, and control and measurement applications may also find RTP applicable.

This document defines RTP, consisting of two closely-linked parts:

o the real-time transport protocol (RTP), to carry data that has real-time properties.

o the RTP control protocol (RTCP), to monitor the quality of service and to convey information about the participants in an on-going session. The latter aspect of RTCP may be sufficient for "loosely controlled" sessions, i.e., where there is no explicit membership control and set-up, but it is not necessarily intended to support all of an application's control communication requirements. This functionality may be fully or partially subsumed by a separate session control protocol,

RTP is intended to be malleable to provide the information required by a particular application and will often be integrated into the application processing rather than being implemented as a separate layer. RTP is a protocol framework that is deliberately not complete.

Keywords: RTP , QoS, RTCP, UDP

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：张波宇　日期：2007年 6月 8日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：张波宇　导师签名：李敬

日期：2007年 6月 8日

引言

RTP(Real Time Transport Protocol:实时传输协议)是为支持实时多媒体通信而设计的传输层协议,是由 IETF(Internet Engineering Task Force:Internet 工程任务组)作为 RFC1889 而发布的。RTP 由两种协议组成:数据传送协议(RTP)和控制协议(RTCP)。RTP 负责多媒体数据的传送,RTCP 管理控制信息。

RTP 为数据提供了具有实时特征的端对端传送服务,如在组播或单播网络服务下的交互式视频音频或模拟数据。应用程序通常在 UDP 上运行 RTP 以便使用其多路结点和校验服务;这两种协议都提供了传输层协议的功能。但是 RTP 可以与其它适合的底层网络或传输协议一起使用。如果底层网络提供组播方式,那么 RTP 可以使用该组播表传输数据到多个目的地。

RTP 在多媒体数据的头部加上时标和符号化方式识别码后发送出去,接收端将加在头部的信息"再生"复原。RTCP 监视迟滞和带宽,并将其通知发送端。若可用带宽一旦变窄,RTCP 立刻将该信息通知发送端,发送端根据此信息,变更符号化方式和解像度,继续进行多媒体通信。RTP 是充分考虑所给定的网络,实现传送质量与给定网络相适应的多媒体通信。

第一章 技术背景

1.1 背景

随着 Internet 的日益普及，在网络上传输的数据已经不再局限于文字和图形，而是逐渐向声音和视频等多媒体格式过渡。目前在网络上传输音频/视频（Audio/Video，简称 A/V）等多媒体文件时，基本上只有下载和流式传输两种选择。通常说来，A/V 文件占据的存储空间都比较大，在带宽受限的网络环境中下载可能要耗费数分钟甚至数小时，所以这种处理方法的延迟很大。如果换用流式传输的话，声音、影像、动画等多媒体文件将由专门的流媒体服务器负责向用户连续、实时地发送，这样用户可以不必等到整个文件全部下载完毕，而只需要经过几秒钟的启动延时就可以了，当这些多媒体数据在客户机上播放时，文件的剩余部分将继续从流媒体服务器下载。

流（Streaming）是近年在 Internet 上出现的新概念，其定义非常广泛，主要是指通过网络传输多媒体数据的技术总称。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。

由于受网络带宽、计算机处理能力和协议规范等方面的限制，要想从 Internet 上下载大量的音频和视频数据，无论从下载时间和存储空间上来讲都是不太现实的，而流媒体技术的出现则很好地解决了这一难题。目前实现流媒体传输主要有两种方法：顺序流（progressive streaming）传输和实时流（realtime streaming）传输，它们分别适合于不同的应用场合。

顺序流传输

顺序流传输采用顺序下载的方式进行传输，在下载的同时用户可以在线回放多媒体数据，但给定时刻只能观看已经下载的部分，不能跳到尚未下载的部分，也不能在传输期间根据网络状况对下载速度进行调整。由于标准的 HTTP 服务器就可以发送这种形式的流媒体，而不需要其他特殊协议的支持，因此也常常被称作

HTTP 流式传输。顺序流式传输比较适合于高质量的多媒体片段，如片头、片尾或者广告等。

实时流传输

实时流式传输保证媒体信号带宽能够与当前网络状况相匹配，从而使得流媒体数据总是被实时地传送，因此特别适合于现场事件。实时流传输支持随机访问，即用户可以通过快进或者后退操作来观看前面或者后面的内容。从理论上讲，实时流媒体一经播放就不会停顿，但事实上仍有可能发生周期性的暂停现象，尤其是在网络状况恶化时更是如此。与顺序流传输不同的是，实时流传输需要用到特定的流媒体服务器，而且还需要特定网络协议的支持。

1.2 存在的问题

近来,电视会议系统、影视点播(VOD:Video On Demand)、动画数据库等多媒体应用,已成为网上业务的热点。作为多媒体应用环境,使采用 TCP/IP 协议的 Internet 显得"力不从心"。

Internet 对于目前的应用如 Telnet、FTP 等,可以说是"游刃有余",但是,一旦运行多媒体应用,原有应用的响应速度将会大幅度下降,而多媒体应用中的动画也会发生滞涩。这是由于采用 TCP/IP 协议的 Internet/Intranet 存在下述问题而引起的。

可靠性的协议并不适合用来传输对于时延很敏感的数据如实时语音或视频等，当发送者发现包丢失并且重传的时候，至少已经过了一个 RTT 的时间，而发送者要么必须等待重传的数据，造成声音的不连续或者不按照 TCP 协议而丢弃重发的数据，而标准的 TCP 实现都要求等待重发的数据并处理，所以延时不断增加，这对于实时传输是不利的；

- TCP 不支持组播；
- TCP 拥塞控制机制是在发现丢包的时候减小窗口，而对于语音或者视频来说，是不能够突然减小速率来饿死接受者的，比如对于 PCM 编码的语音数据来说，是不会超过 64kb/s 多少的。对于这些媒体的拥塞控制机制最好是更改编码的比特率，比方说根据 RTCP 接收报告分组。

1.3 解决方案

为了解决上述问题,开发出了支持多媒体通信的技术。

RTP(Real Time Transport Protocol:实时传输协议)是为支持实时多媒体通信而设计的传输层协议,是由 IETF(Internet Engineering Task Force:Internet 工程任务组)作为 RFC1889 而发布的。RTP 由两种协议组成:数据传送协议(RTP)和控制协议(RTCP)。RTP 负责多媒体数据的传送,RTCP 管理控制信息。

RTP 在多媒体数据的头部加上时标和符号化方式识别码后发送出去,接收端将加在头部的信息"再生"复原。RTCP 监视迟滞和带宽,并将其通知发送端。若可用带宽一旦变窄,RTCP 立刻将该信息通知发送端,发送端根据此信息,变更符号化方式和解像度,继续进行多媒体通信。RTP 是充分考虑所给定的网络,实现传送质量与给定网络相适应的多媒体通信。

RTP 由两个紧密链接部分组成:

实时传输协议 RTP (Realtime Transport Protocol) 是在因特网上实现点对点、点对多传送数据的一种传输协议,由 IETF(Internet 工程任务组)提出并标准化,其目的是提供时间信息和实现流同步。RTP 的典型应用建立在 UDP 上,但也可以在 TCP 或 ATM 等其他协议之上工作。但 RTP 本身只保证实时数据的传输,并不能为按顺序传送数据包提供可靠的传送机制,也不提供流量控制或拥塞控制,它依靠 RTCP 提供这些服务。RTP 是一种提供端对端传输服务的实时传输协议,用来支持在单目标广播和多目标广播网络服务中传输实时数据,而实时数据的传输则由 RTCP 协议来监视和控制。

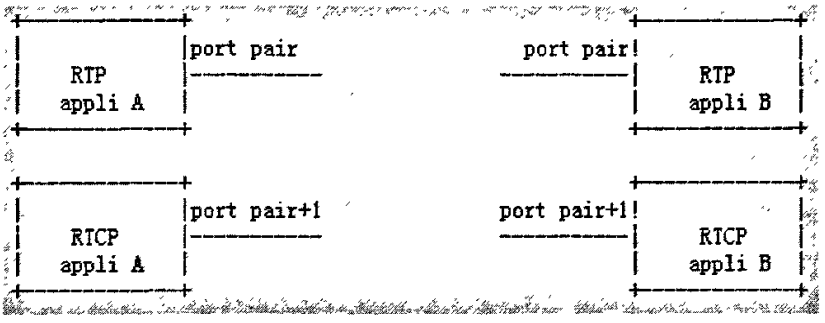
实时传输控制协议 RTCP (Realtime Transport Control Protocol):负责管理传输质量在当前应用进程之间交换控制信息。在 RTP 会话期间,各参与者周期性地传送 RTCP 包,包中含有已发送的数据包的数量、丢失的数据包的数量等统计资料,因此,服务器可以利用这些信息动态地改变传输速率,甚至改变有效载荷类型。RTP 和 RTCP 配合使用,能以有效的反馈和最小的开销使传输效率最佳化,故特别适合传送网上的实时数据。

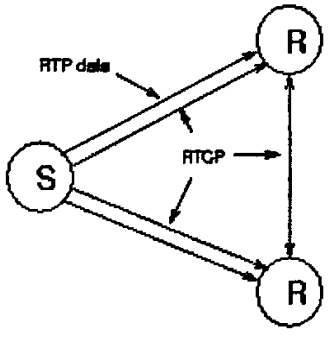
RTP 是一个单向协议,只能从服务器发送实况或存储流到客户。

RTP 一般和 RTCP 配合使用(图 1)。两个密切链接的部分(相继的 UDP 端口):

o 数据部分,传输 RTP 数据

o 控制部分 RTCP (Real Time Control Protocol),监视 QoS 和携带预期信息





(a) 用途

图 1-1　RTP 和 RTCP 的使用

第二章 RTP 协议之实时传输协议 RTP

2.1 RTP 简介

实时传输协议（RTP）定义在 RFC 1889 中。信息包的结构包含广泛用于多媒体的若干个域，包括声音点播(audio-on-demand)、影视点播(video on demand)、因特网电话(Internet telephony)和电视会议(videoconferencing)。RTP 的规格没有对声音和电视的压缩格式制定标准，它可以被用来传输普通格式的文件。例如，WAV 或者 GSM(Global System for Mobile communications)格式的声音、MPEG-1 和 MPEG-2 的电视，也可以用来传输专有格式存储的声音和电视文件。

RTP 为数据提供了具有实时特征的端对端传送服务，如在组播或单播网络服务下的交互式视频音频或模拟数据。应用程序通常在 UDP 上运行 RTP 以便使用其多路结点和校验服务；这两种协议都提供了传输层协议的功能。但是 RTP 可以与其它适合的底层网络或传输协议一起使用。如果底层网络提供组播方式，那么 RTP 可以使用该组播表传输数据到多个目的地。

RTP 本身并没有提供按时发送机制或其它服务质量（QoS）保证，它依赖于低层服务去实现这一过程。 RTP 并不保证传送或防止无序传送，也不确定底层网络的可靠性。 RTP 实行有序传送， RTP 中的序列号允许接收方重组发送方的包序列，同时序列号也能用于决定适当的包位置，例如：在视频解码中，就不需要顺序解码。

使用 RTP 协议的应用程序运行在 RTP 之上，而执行 RTP 的程序运行在 UDP 的上层，目的是为了使用 UDP 的端口号和检查和。如图 1 所示，RTP 可以看成是传输层的子层。由多媒体应用程序生成的声音和电视数据块被封装在 RTP 信息包中，每个 RTP 信息包被封装在 UDP 消息段中，然后再封装在 IP 数据包中。

<table>
<tr><td></td><td>TCP/IP 模型</td></tr>
<tr><td></td><td>应用层(application)</td></tr>
<tr><td rowspan="3">传输层</td><td>RTP</td></tr>
<tr><td>UDP</td></tr>
<tr><td>IP</td></tr>
</table>

	数据链路层(data link)
	物理层(physical)

表 3-1 RTP 是传输层上的协议

从应用开发人员的角度来看，可把 RTP 执行程序看成是应用程序的一部分，因为开发人员必需把 RTP 集成到应用程序中。在发送端，开发人员必需把执行 RTP 协议的程序写入到创建 RTP 信息包的应用程序中，然后应用程序把 RTP 信息包发送到 UDP 的套接接口(socket interface)，如表 1 所示；同样，在接收端，RTP 信息包通过 UDP 套接接口输入到应用程序，因此开发人员必需把执行 RTP 协议的程序写入到从 RTP 信息包中抽出媒体数据的应用程序。

TCP/IP 模型	
应用层(application)	
RTP	
—	套接接口
UDP	
IP	
数据链路层(data link)	
物理层(physical)	

表 3-2 RTP 和 UDP 之间的接口

现以用 RTP 传输声音为例来说明它的工作过程。假设音源的声音是 64 kb/s 的 PCM 编码声音，并假设应用程序取 20 毫秒的编码数据为一个数据块，即在一个数据块中有 160 个字节的声音数据。应用程序需要为这块声音数据添加 RTP 标题生成 RTP 信息包，这个标题包括声音数据的类型、顺序号和时间戳。然后 RTP 信息包被送到 UDP 套接接口，在那里再被封装在 UDP 信息包中。在接收端，应用程序从套接接口处接收 RTP 信息包，并从 RTP 信息包中抽出声音数据块，然后使用 RTP 信息包的标题域中的信息正确地译码和播放声音。

如果应用程序不使用专有的方案来提供有效载荷类型(payload type)、顺序号或者时间戳，而是使用标准的 RTP 协议，应用程序就更容易与其他的网络应用程序配合运行，这是大家都希望的事情。例如，如果有两个不同的公司都在开发

因特网电话软件，他们都把 RTP 合并到他们的产品中，这样就有希望：使用不同公司电话软件的用户之间能够进行通信。

这里需要强调的是，RTP 本身不提供任何机制来确保把数据及时递送到接收端或者确保其他的服务质量，它也不担保在递送过程中不丢失信息包或者防止信息包的次序不被打乱。的确，RTP 的封装只是在系统端才能看到，中间的路由器并不区分那个 IP 数据报是运载 RTP 信息包的。

RTP 允许给每个媒体源分配一个单独的 RTP 信息包流，例如，摄像机或者麦克风。例如，有两个团体参与的电视会议，这就可能打开 4 个信息包流：两台摄像机传送电视流和两个麦克风传送声音流。然而，许多流行的编码技术，包括 MPEG-1 和 MPEG-2 在编码过程中都把声音和电视图像捆绑在一起以形成单一的数据流，一个方向就生成一个 RTP 信息包流。

RTP 信息包没有被限制只可应用于单目标广播，它们也可以在一对多(one-to-many)的多目标广播树或者在多对多(many-to-many)的多目标广播树上传送。例如，多对多的多目标广播，在这种应用场合下，所有发送端通常都把他们的 RTP 信息包流发送到具有相同多目标广播地址的多目标广播树上。

2.2 RTP 数据报分析

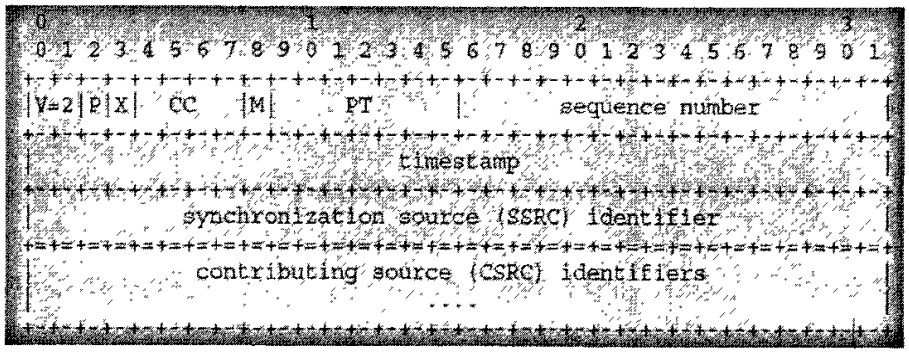


图 3-1 RTP 包

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。

前 12 个字节在每一个 RTP packet 中都存在，而一系列的 CSRC 标记只有存在 Mixer 时才有。

version (V): 2 bits

标明 RTP 版本号。协议初始版本为 0，RFC3550 中规定的版本号为 2。

padding (P): 1 bit

置 1 时表明末尾有非数据的填充字段。所有填充字段的最后一个字节为填充字段长度。该字段之所以存在是因为一些加密机制需要固定长度的数据块，或者为了在一个底层协议数据单元中传输多个 RTP packets。

extension (X): 1 bit

如果该位被设置，则在固定的头部后存在一个扩展头部。

CSRC count (CC): 4 bits

在固定头部后存在多少个 CSRC 标记。

marker (M): 1 bit

该位的功能依赖于 profile 的定义。profile 可以改变该位的长度，但是要保持 marker 和 payload type 总长度不变（一共是 8 bit）。

payload type (PT): 7 bits

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。如果接收方不能识别该类型，必须忽略该 packet。RTP 可支持 128 种不同的有效载荷类型。对于声音流，这个域用来指示声音使用的编码类型，例如 PCM、自适应增量调制或线性预测编码等等。如果发送端在会话或者广播的中途决定改变编码方法，发送端可通过这个域来通知接收端。表 1 列出了目前 RTP 所能支持的声音有效载荷类型。

表 3-3 目前 RTP 所能支持的声音有效载荷类型

有效载荷号	声音类型	采样率(kHz)	数据率(kb/s)
0	PCM mu-law	8	64
1	1016	8	4.8
2	G.721	8	32
3	GSM	8	32
6	DVI	16	64
7	LPC	8	2.4
9	G.722	8	48～64
14	MPEG Audio	90	-
15	G.728	8	16

对电视流，有效载荷类型可以用来指示电视编码的类型，例如 motion JPEG, MPEG-1，MPEG-2 或者 H.231 等等。发送端也可以在会话或者期间随时改变电视的编码方法。表 2 列出了目前 RTP 所能支持的某些电视有效载荷类型。

表 3-4 目前 RTP 所能支持的声音有效载荷类型

有效载荷号	电视格式
26	Motion JPEG
28	-
31	H.261
32	MPEG-1 video
33	MPEG-2 video

sequence number: 16 bits

序列号，每个 RTP packet 发送后该序列号加 1，接收方可以根据该序列号重新排列数据包顺序，其初始数值随机产生，以防止别人破译加密。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。例如，接收端的应用程序接收到一个 RTP 信息包流，这个 RTP 信息包在顺序号 86 和 89 之间有一个间隔，接收端就知道信息包 87 和 88 已经丢失，并且采取措施来处理丢失的数据。

timestamp: 32 bits

时间戳。它反映 RTP 数据信息包中第一个字节的采样时刻(时间)。接收端可以利用这个时间戳来去除由网络引起的信息包的抖动，并且在接收端为播放提供同步功能。该项用于时间同步计算和抖动控制，其精度必须足以满足上述两项要求。时钟频率和数据格式有关，不能使用系统时钟。对固定频率的音频来说，每次取样时戳时钟增 1。和包序号一样，时间戳的开始值也是随机的。如果多个连续的 RTP 包在逻辑上是同时产生的，那么它具有相同的时间戳。

SSRC: 32 bits

同步化源标识符。。它用来标识 RTP 信息包流的起源，在一个 RTP Session 其间每个数据流都应该有一个不同的 SSRC，所以当发送者传输地址改变时此值需重新生成，以防止形成循环。SSRC 不是发送端的 IP 地址，而是在新的信息包流开始时源端随机分配的一个号码。

CSRC list: 0 to 15 items, 32 bits each

标识贡献的数据源。只有存在 Mixer 的时候才有效。如一个将多声道的语音流合并成一个单声道的语音流，在这里就列出原来每个声道的 SSRC。

2.3 典型的 RTP 包传输流程:

RTP 利用混合器(Mixer)和翻译器(translator)完成实时数据的传输。混合器接收大自一个或多个发送方的 RTP 数据块，并把它们组合成一个新的 RTP 分组继续转发。这种组合数据块将有一个新的 SSRC 标识。具有新标识的特别发送方被作为特别信源加入到 RTP 数据块中。因为来自不同特别发送方的数据块可以非同步到达(它们可以经不同的路径经过这个网络),所以混合器改变了该媒体流的临时结构。与混合器不同，翻译器只改变数据块内容，而并不把媒体流组合在一起。翻译器只是对单个媒体流进行操作。可能进行编码转换或者协议翻译。一个典型的 RTP 包传输流程如下:

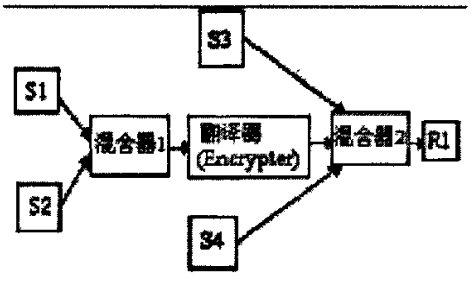


图 3-2 典型 RTP 包传输流程

其中 S1，S2，S3，S4 为数据源的发送端，R1 为最终 RTP 包流的接收端。 RTP 提供足够的机制来适应传输实时数据，例如用时间戳及其控制机制来对一些具有时间特质的数据流进行同步。为了更高层次的同步以及同步不是周期传输的媒体流,RTP 使用了一个线性单调的时钟，它的增长比媒体流的最小的数据块的增长还要快，例如音频的采样率。初始时钟值是随机的。一些串行的 RTP 包的时间戳如果在时间上是同时发生的，则应该相等。例如，那些属于同一个视频帧的包；一些串行的 RTP 包的时间戳也可以是非单调。如果它们不按顺序传送，例如，MPEG 的交插帧，但其顺序号必须是单调的。

2.4 复用 RTP 连接

为使协议有效运行，复用点数目应减至最小。RTP 中，复用由定义 RTP 连接的目的传输地址（网络地址与端口号）提供。例如，对音频和视频单独编码的远程会议，每个媒介被携带在单独 RTP 连接中，具有各自的目的传输地址。目标不在将音频和视频放在单一 RTP 连接中，而根据 SSRC 段载荷类型进行多路分解。

使用同一 SSRC ，而具有不同载荷类型的交叉包将带来几个问题：如一种载荷类型在连接期间切换，没有办法识别新值将替换那一个旧值。SSRC 定义成用于标识单个计时和系列号空间。如媒体时钟速率不同，而要求不同系列号空间以说明那种载荷类型有丢包，交叉复用载荷类型将需要不同计时空间。

RTCP 发送和接收报告可能仅描述每个 SSRC 的计时和系列号空间，而不携带载荷类型段。RTP 混合器不能将不兼容媒体流合并成一个流。在一个 RTP 连接中携带多个媒介阻止几件事：使用不同网络路径或网络资源分配；接受媒介子集。

对每种媒介使用不同 SSRC，但以相同 RTP 连接发送可避免前三个问题，但不能避免后两个问题。

2.5 对 RTP 头特定设置的修改

可以认为，现用 RTP 数据包头对 RTP 支持的所有应用类共同需要的功能集是完整的。然而，为维持 ALF 设计原则，头可通过改变或增加设置来裁剪，并仍允许设置无关监控和记录工具起作用。标记位与载荷类型段携带特定设置信息，但由于很多应用需要它们，否则要容纳它们，就要增加另外 32 位字，故允许分配在固定头中。包含这些段的八进制可通过设置重新定义以适应不同要求，如采用更多或更少标记位。如有标记位，既然设置无关监控器能观察包丢失模式和标记位间关系，我们就可以定位八进制中最重要的位。

其它特殊载荷格式（视频编码）所要求的信息应该携带在包的载荷部分。可出现在头，总是在载荷部分开始处，或在数据模式的保留值中指出。如特殊应用类需要独立载荷格式的附加功能，应用运行的设置应该定义附加固定段跟随在现存固定头 SSRC 之后。这些应用将能迅速而直接访问附加段，同时，与监控器和记录器无关设置仍能通过仅解释开始 12 个八进制处理 RTP 包。如证实附加功能是所有设置共同需要的，新版本 RTP 应该对固定头作出明确改变。

2.6 RTP 主要功能

RTP 数据协议的主要功能可归结如下：

（1）标识分组数据的编码算法、采样频率及承载通道；

（2）给各数据分组标明序列号，便于接收方探测分组丢失；

（3）提供时间戳来确定延迟抖动；

第三章 RTP 传输控制协议之——RTCP

3.1 RTCP 简介

实时传输控制协议(Real-time Control Protocol，RTCP)也定义在 1996 年提出的 RFC 1889 中。多媒体网络应用把 RTCP 和 RTP 一起使用，尤其是在多目标广播中更具吸引力。当从一个或者多个发送端向多个接收端广播声音或者电视时，也就是在 RTP 会话期间，每个参与者周期性地向所有其他参与者发送 RTCP 控制信息包。RTCP 用来监视服务质量和传送有关与会者的信息。对于 RTP 会话或者广播，通常使用单个多目标广播地址，属于这个会话的所有 RTP 和 RTCP 信息包都使用这个多目标广播地址，通过使用不同的端口号可把 RTP 信息包和 RTCP 信息包区分开来。RTCP 协议将控制包周期发送给所有连接者，应用与数据包相同的分布机制。低层协议提供数据与控制包的复用，如使用单独的 UDP 端口号。

RTCP 的主要功能是为应用程序提供会话质量或者广播性能质量的信息。每个 RTCP 信息包不封装声音数据或者电视数据，而是封装发送端和/或者接收端的统计报表。这些信息包括发送的信息包数目、丢失的信息包数目和信息包的抖动等情况，这些反馈信息对发送端、接收端或者网络管理员都是很有用的。RTCP 规格没有指定应用程序应该使用这个反馈信息做什么，这完全取决于应用程序开发人员。例如，发送端可以根据反馈信息来修改传输速率，接收端可以根据反馈信息判断问题是本地的、区域性的还是全球性的，网络管理员也可以使用 RTCP 信息包中的信息来评估网络用于多目标广播的性能。

3.2 RTCP 四大功能

(1) 主要是提供数据发布的质量反馈。RTCP 是作为 RTP 传输协议的一部分，与其他传输协议的流和阻塞控制有关。反馈对自适应编码控制直接起作用，但 IP 多播经验表明，从发送者收到反馈对诊断发送错误是至关重要的。给所有参加者发送接收反馈报告允许问题观察者估计那些问题是局部的，还是全局的。诸如 IP 多播等发布机制使网络服务提供商之类的团体可能接收反馈信息，充当第三方监控者来诊断网络问题。反馈功能由 RTCP 发送者和接收者报告执行。

(2) RTCP 带有称作规范名字(CNAME)的 RTP 源持久传输层标识。如发现冲突，或程序重新启动，既然 SSRC 标识可改变，接收者需要 CNAME 跟踪参加者。接收者也需要 CNAME 与相关 RTP 连接中给定的几个数据流联系。

(3)前两种功能要求所有参加者发送 RTCP 包，因此，为了 RTP 扩展到大规模数量，速率必须受到控制。让每个参加者给其他参加者发送控制包，就加大独立观察参加者数量。该数量用于计算包发送的速率。

(4)第四个可选功能是传送最小连接控制信息，如辨识参加者。最可能用在“松散控制”连接，那里参加者自由进入或离开，没有成员控制或参数协调，RTCP 充当通往所有参加者的方便通道，但不必支持应用的所有控制通讯要求。

在 IP 多播场合应用 RTP 时，前三个功能是必须的，推荐用于所有情形。RTP 应用设计人员必须避免使用仅在单播模式下工作的机制，那将导致无法扩展规模。

3.3 RTCP 协议工作模式

当应用程序开始一个 RTP 会话时将使用两个端口：一个给 RTP，一个给 RTCP。RTP 本身并不能为按顺序传送数据包提供可靠的传送机制，也不提供流量控制或拥塞控制，它依靠 RTCP 提供这些服务。在 RTP 的会话之间周期的发放一些 RTCP 包以用来传监听服务质量和交换会话用户信息等功能。RTCP 包中含有已发送的数据包的数量、丢失的数据包的数量等统计资料。因此，服务器可以利用这些信息动态地改变传输速率，甚至改变有效载荷类型。RTP 和 RTCP 配合使用，它们能以有效的反馈和最小的开销使传输效率最佳化，因而特别适合传送网上的实时数据。根据用户间的数据传输反馈信息，可以制定流量控制的策略，而会话用户信息的交互，可以制定会话控制的策略。

3.4 RTCP 数据报类型

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。

其中最主要的 RTCP 报文是 SR 和 RR。通常 SR 报文占总 RTCP 包数量的 25%，RR 报文占 75%。类似于 RIP 数据包，每个 RTCP 报文以固定的包头部分开始，紧接着的是可变的结构元素，但是以 32 位长度为结束边界。在 RTCP 报文中，不需要插入任何分隔符就可以将多个 RTCP 报文连接起来形成一个 RTCP 组合报文。由于需要底层协议提供整体民度来决定组合报文的结尾，所以在组合报文中没有单个 RTCP 报文的显式计数。

RTCP 控制报文的发送周期是变化的，与报文长度 L。用户数 N 和控制报文带宽 B 相关：周期 P=L*N / B。原因是 RTP 设计成允许应用自动扩展的模式，连接数可从几个到上千个。在一般的音频会议中，因为同一时刻一般只有两个人说话，所以数据流和控制流都是内在限制的，控制流不会对传输造成影响。而在组播发送模式下，给定连接数据率独立于用户数，仍是常数，但控制流量不是内在限制的。如果每个参加者以固定速率发送接收报告，控制流量将随参加者数量线性增长，因此，速率必须按比例下降。

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。

类似于 RTP 数据包，每个 RTCP 包以固定部分开始，紧接着的是可变长结构元

素，但以一个 32 位边界结束。包含安排要求和固定部分中长度段，使 RTCP 包可堆叠。不需要插入任何分隔符将多个 RTCP 包连接起来形成一个 RTCP 组合包，以低层协议用单一包发送出去。由于需要低层协议提供整体长度来决定组合包的结尾，在组合包中没有单个 RTCP 包显式计数。

组合包中每个 RTCP 包可独立处理，不需要根据包组合顺序。但为了执行协议功能，强加如下约束：

(1) 接收统计(在 SR 或 RR 中)应该经常发送，只要带宽允许，因此每个周期发送的组合 RTCP 包应包含报告包。

(2) 新接收者需要接收 CNAME，并尽快识别源，开始联系媒介进行同步，因此每个包应该包含 SDES CNAME。

(3) 出现在组合包前面的是包类型数量，其增长应该受到限制，以提高常数位数量，提高成功确认 RTCP 包对错误地址 RTP 数据包或其他无关包的概率。

因此，所有 RTCP 包至少必须以两个包组合形式发送，推荐格式如下：

①加密前辍(Encryption prefix)

仅当组合包被加密，才加上一个 32 位随机数用于每个组合包发送。

② SR 或 RR

组合包中第一个 RTCP 包必须总为一个报告包，方便头的确认。即使没有数据发送，也没有接收到数据，也要发送一个空 RR，即避免组合包中 RTCP 包为 BYE(终止标识）。

③ 附加 RR

如报告统计源数目超过 31，在初始报告包后应该有附加 RR 包。

④ SDES

包含 CNAME 项的 SDES 包必须包含在每个组合 RTCP 包中。如应用要求，其他源描述项可选，但受到带宽限制。

⑤ BYE 或 APP

其他 RTCP 包类型可以任意顺序排列，除了 BYE 应作为最后一个包发送，包类型出现可不止一次。

转换器或混合器从多个源组合单个 RTCP 包。如组合包整体长度超过网络路径量最大传输单元，可分成多个较短组合包用低层协议以单个包形式发送。注意，每个组合包必须以 SR 或 RR 包开始。附加 RTCP 包类型可在因特网分配号码机构（IANA, Internet Assighnedd Numbers Authority）处注册。

RTCP 组合包的长度有 UDP 包的长度决定。每个 RTCP 组合包必须而且至少包含

RR 和 SDES 这两个 RTCP 包。建议混合器和转换器将多个数据源传来的 RTCP 包合成一个组合包以节省头部的开支。如果 RTCP 组合包的长度超过网络的最大传输单元，则它必须分为几个部分来传输。

3.5 得到会话参加者数量

RTCP 包间隔的计算依赖于对参与会话的站点数目的估计。新站点被计入当它们被听到时，并且“应当”为每个新站点在以 SSRC 或 CSRC 标识符索引的表（见 8.2 节）中创建相应条目以跟踪它们。新条目“可以”被认为无效，直到收到多个包含新 SSRC 的报文被(见附录 A.1),或者收到一个包含对应 SSRC 的 CNAME 的 SEDS RTCP 包。当收到 RTCP BYE 包时，相应 SSRC 的条目“可以”被删除，除非此后一些散乱的数据包到达并导致重新创建此条目。替代地，“应当”标记此条目收到 BYE 并在适当延迟之后删除它。

参与者“可以”标记另一站点非活跃，或如果条目无效则删除它，如果几个 RTCP 报告间隔（“推荐”5s）内没有收到 RTP 或 RTCP 包。这为丢包提供了某种健壮性。所有站点必须使用相同的乘数且必须计算出大指向等的 RTCP 报告间隔，这个超时才能正常工作。因而，特定策略“应当”固定此乘数。

对大量成员参与的会话，维护一个存储着 SSRC 标识符和相应状态信息的表可能并不可行。一个实现“可以”按照[21]所述那样使用 SSRC 抽样（sampling）以减小存储需求。一个实现“可以”使用其他类似的算法。一个关键要求就是这样的算法“应当不”大体上低估组规模，然而“可以”高估。

3.6 源服务器描述的带宽分配

RTCP 包中的 SDES 包内包含了对于源服务器的描述。描述内容除了 CNAME 项是必须的，其余内容由应用程序的需要决定。由于发送额外内容需要占用网络带宽，所以发送不同内容的包具有优先级的差别。

3.7 RTCP 传输间隔

RTP 设计成允许应用自动扩展，连接数可从几个到上千个。例如，音频会议中，数据流量是内在限制的，因为同一时刻只有一两个人说话；对多播，给定连接数

据率仍是常数，独立于连接数。但控制流量不是内在限制的。如每个参加者以固定速率发送接收报告，控制流量将随参加者数量线性增长，因此，速率必须按比例下降。

对每个连接，假定数据流量受到“连接带宽”的总量限制。选择连接带宽是根据费用或网络带宽的先验知识，与媒体编码无关，但编码选择会受到连接带宽的限制。当调用一个媒体应用时，连接带宽参数由连接管理应用提供，但媒体应用也可根据单个发送者数据带宽设置缺省值。应用可根据多播范围规则或其他标准强制限制带宽。由于这是资源预订系统需要知道的，对控制与数据流量的带宽计算包括低层传输与网络协议。应用也需要知道在使用哪个协议。在传输途中，因为包将包含不同连接层的包头，所以连接层的包头不计算在内。控制流量应限制为连接带宽中已知的一小部分：小，传递数据的传输协议主要功能才不致受损；已知，控制流量才能包含在所给资源预订协议的带宽规范中，这样，每个参加者就可单独计算其共享量。建议分配给 RTCP 的连接带宽固定为 5%，而这个数值与间隔计算中其他常量并不重要，连接中所有参加者必须使用相同数值，因此，使用相同间隔计算。

计算 RTCP 包间隔依赖连接中地址加入数量的估计。当新地址被监听到，就加到此计数，并在以 SSRC 或 CSRC 标识索引的表中为之建立一个条目，用来跟踪它们。直到收到带有多个新 SSRC 包，新条目才有效。当收到具有相应 SSRC 标识的 RTCP 的 BYE 包，条目可从表中删除。如果对少量 RTCP 报告间隔没有接收到 RTP 或 RTCP 包，参加者可能将另外一个地址标记成不活动，如还未生效就删除掉。这为防止包丢失提供强大支持。为了“超时”正常工作，所有地址必须对 RTCP 报告间隔记入大致相同的数值。

一旦确认地址有效，如后来标记成不活动，地址的状态应仍保留，地址应继续计入共享 RTCP 带宽地址的总数中，时间要保证能扫描典型网络分区，建议为 30 分钟。注意，这仍大于 RTCP 报告间隔最大值的五倍。

这个规范定义了除必需的 CNAME 外的几个源描述项，如 NAME(人名)和 E-mail(电子邮件地址)。它也是定义新特定应用 RTCP 包类型的途径。给附加信息分配控制带宽应引起注意，因为它将降低接收报告和 CNAME 发送的速率而损害协议的性能。建议分配给单个参加者用于携带附加信息的 RTCP 带宽不要超过 20%。而且并没有有意让所有 SDES 项包含在每个应用中。

一个实现可以把最小 RTCP 间隔变得更小以适合会话带宽参数，有以下限制：

- 对多播会话，仅活跃数据发射机“可以”使用更小的最小值来计算符合 RTCP 包传输间隔。

- 对单播会话，非活跃数据发射机也“可以”使用更小的值，发送第一个复合 RTCP 包之前的延迟“可以”是 0。

- 对所有会话，计算参与者超时间隔时“应当”使用上述固定的最小值，这样那些不使用更小值的实现就不会被其他参与者过早地认为超时。

- 更小值（以秒计）“推荐”采用 360 除以会话带宽（单位 kb/s）。这个值在带宽超过 72kb/s 时小于 5 秒。

算法符合本节提出的目标。此算法计算出发送 RTCP 复合包的间隔，并在所有参与者间分配允许的控制流量带宽。这允许应用在如标识所有参与者是重要的这样的小会话能快速响应，并且能自动调节到更大会话。此算法具有如下特征：

- 计算出的 RTCP 包间隔随组内成员数目线性增涨。正是这个线性使得所有成员的控制流量总和为常量。

- 实际 RTCP 包间隔在计算值[0.5,1.5]倍范围内随即选取，以避免不期望的所有参与者同步[20]。加入会话后的第一个 RTCP 包延迟是最小 RTCP 间隔[0,0.5]倍范围内随机值。

- 计算出复合 RTCP 包的平均尺寸的估计值，包括所有发送和接收的，以自动适应承载的控制信息量的变化。

- 既然计算出的间隔依赖于观察到的组成员数目，可能存在不希望的启动效应：在一新用户加入一正退出的会话，或多个用户同时加入一新会话。这些新用户刚开始对组规模估计不正确，这样它们的 RTCP 传输间隔非常短。这个问题在多个用户同时加入会话时比较明显。为处理这种情况，应用了一个被称为“timer reconsideration”的算法。此算法实现了一个简单的补偿(back-off)机制使得用户在组规模增长时阻止发送 RTCP 包。

- 当用户离开会话，以 BYE 或超时，组规模减小，这样计算出的间隔应当减小。“reverse reconsideration”算法能够使得成员更快地减小其间隔以对组规模减小作出反应。

- BYE 包与其他 RTCP 包处理不同。当用户离开组时并希望发送 BYE 包时，它可以在其安排的下一个 RTCP 包之前发送。但是，BYE 包发送遵循一个补偿算法以避免大量成员同时离开会话时 BYE 包的泛滥。

这个算法可以用于那些允许所有参与者发送数据的会话。在此情形下，会话带宽参数是单个发送机带宽乘以参与者总数，并且 RTCP 带宽占其中 5%。

3.8 RTCP 的五种分组

RTCP 共定义了五种不同的分组：发送方报告（SR），接收方报告（RR），源描述（SDES），BYE，APP。

3.8.1 发送方报告（SR）

RTP 包的接受者发送 REPORT 的 RTCP 包来反馈服务质量。REPORT 包分为 2 种，RR 和 SR。两者的唯一区别是，发送后者的站点不仅是包的接受者同时也是活跃的包发送者。所以 SR 比 RR 要多 20 字节，用来携带发送者信息。

RTCP 共定义了五种不同的 RTCP 分组：发送方报告（SR），接收方报告（RR），源描述（SDES），BYE，APP。

一个会话双方进行 RTP 分组时使用 SR。格式如下：

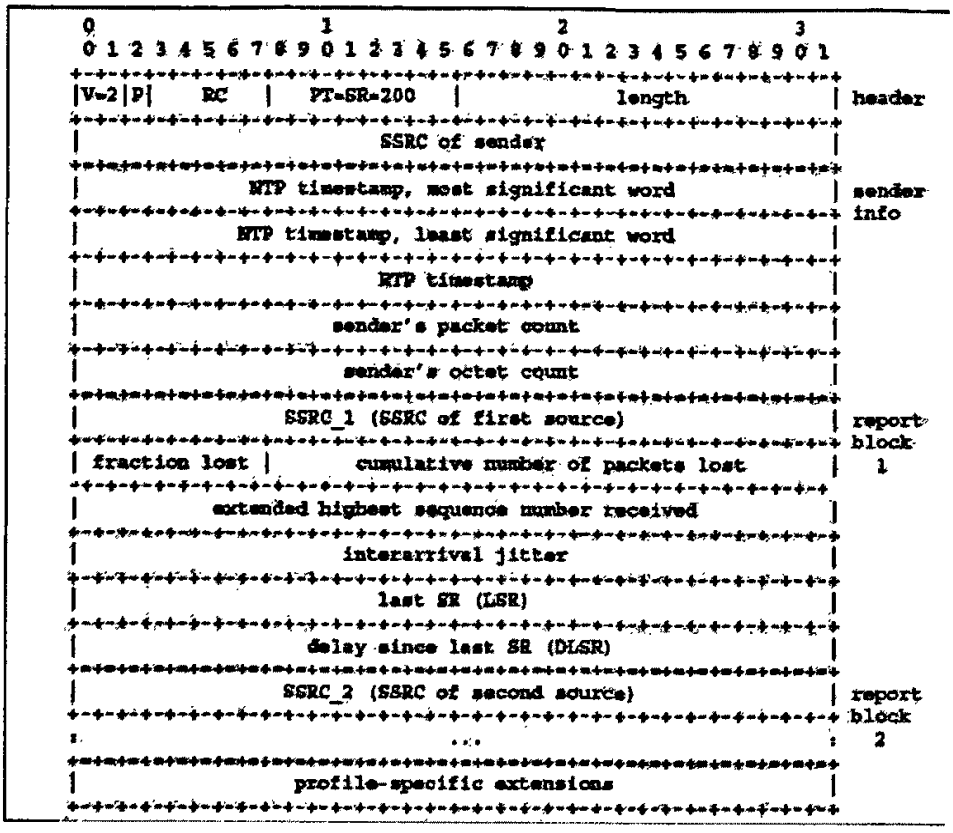


图 5 SR 包

SR 包由 3 部分组成，如果配置文件指定的话，可能会有额外的信息附加在第三部分之后。这三部分为：

1. HEADER
2. SENDER INFO
3. REPORT BLOCKS

HEADER:

version (V): 2 bits

RTP 协议所用版本

padding (P): 1 bit

置 1 时表明末尾有非数据的填充字段。填充字段的最后一个字节为填充字段长度。填充字段常被用于需加密的场合。 多个 RTCP 包被放在一个下层协议包中传送时，只有最后一个包有填充字段，因为下层协议包是整个被加密的。

Reception report count (RC): 5 bits

RTCP包中对于各个站点的RECEPTION REPORT项的数量，可以为0。

packet type (PT): 8 bits

RTCP包类型，作为SR包的值为常量200。

length: 16 bits

RTCP包长度，以32位字长为单位，包括包头和填充，然后减1，即为此项值。所以0也是合法长度。

SSRC: 32 bits

发送此包的站点的SSRC标识符

SENDER INFORMATION:

该部分共长20字节，其中包含了SENDER所要发送的信息，其中字段结构如下：

NTP时戳 (NTP timestamp): 64 bit

绝对时戳。在测量环路时延时可在对方的RR报文中带回；如果发送方不具有绝对时钟的能力，则可以用通话开始时间作为时钟0点或将此域置0。(在NTP格式中，64位的前32位是从1900年1月1日0时开始到现在的以秒为单位的整数部分，后32位是此时间的小数部分)

RTP timestamp: 32 bits

与上面的NTP timestamp对应，用来估算RTP的时钟周期

sender' s packet count: 32 bits

从传输开始到此RTCP包发送时刻为止，站点共发送的RTP数据包数量。

sender' s octet count: 32 bits

从传输开始到此RTCP包发送时刻为止，站点共发送的RTP数据量，即PAYLOAD的数量。当站点的SSRC改变时，此项清零。该项常被用于估计实际数据传输率。

REPORT BLOCKS

该项可以有0或者多项信息，取决于从上次发包到当前包的时间间隔内该服务器收到的数据包的源的数量。每项都传递了发送给当前站点数据包的SSRC的接收信息。该项包含以下信息：

SSRC_n (source identifier): 32 bits

当前BLOCK中的信息所对应的SSRC标识符

丢包率 (fraction lost): 8 bits

从上次发送RR或SR到本次发送期间，从该项对应的SSRC发送到本站点的包

中丢失的包的数量与应收到包的数量的比例。

累计的包丢失数（cumulative number of packets lost）：24 bits

从该项对应的 SSRC 到本站点所发送包的总丢包率。

接收到的扩展的最高序列号（extended highest sequence number received）：32 bits

低 16 位记录了从对应 SSRC 站点收到的包的最大序号，高 16 位记录了总的循环数。

到达间隔抖动（interarrival jitter）：32 bits

抖动控制字段。每两个 RTP 包的抖动可以用其 RTP 包中的 RTP 时戳和接收的时刻进行计算，计算公式如下：设 Rj 代表第 j 个包的到达时刻，Sj 代表第 j 个包的 RTP 时戳值，则第 i 个 RTP 报文与第 j 个 RTP 报文间的抖动为 D（i，j）：

D（i，j）=（Rj - Ri）-（Sj - Si）=（Rj - Sj）-（Ri - Si）

在生成 RTCP 报文时，其应当传送的时延抖动的值可用如下公式进行递推计算：

J=J+（D（I-1，I）-J）/16

其中，J 为要传送的时延抖动值。对后一项除以 16 是为了消除连带噪声。

上一 SR 报文时戳（LSR）：32 bit

收到的最近一个 SR 报文的 NTP 时戳的中间 32 位。

自上一 SR 的时间（DLSR）：32 bit

在收到上一个 SR 报文与此次发送的报文之间的时间。以 1/65536s 记。如果还没有收到任何 SR 报文，此值置 0。

3.8.2 接受者报告 RTCP 包(RR)

其结构与 SR 包基本相同，区别仅在于少了 20 字节的发送者信息，同时 PACKET TYPE 字段值为 201，而不是 SR 中的 200。

RTCP 接收方报告(RR)，格式如下。它由一个会话参与者发出，它接收了 RTP 分组，但自己还没有发送过 RTP 包到任何一个其他的参与者。

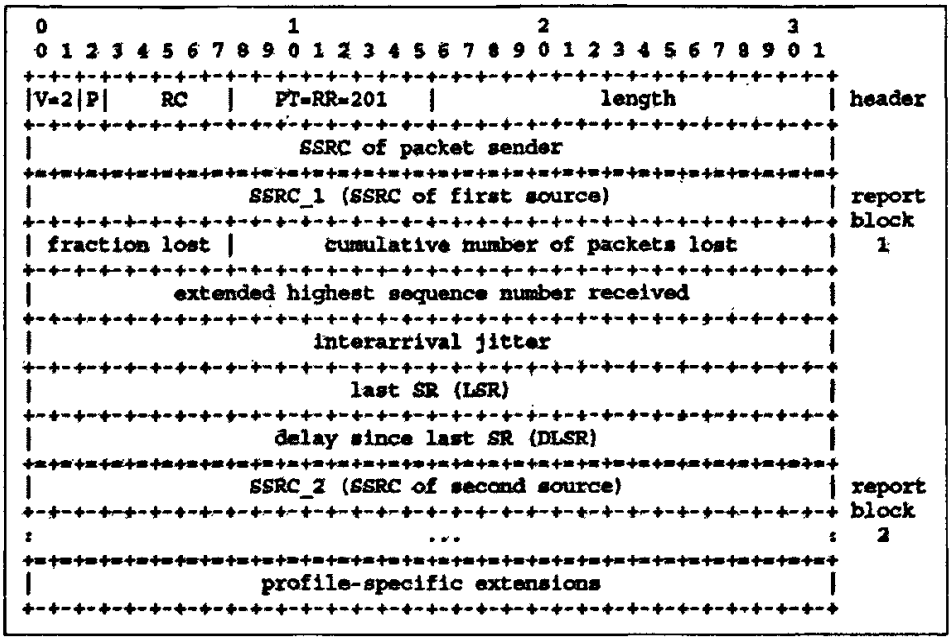


图 6 RR 包

如接收者或发送者有附加信息要有规律地报告,应对接收报告与接收者定义设置或应用扩展。如需要附加发送者信息，应首先包含在发送者报告的扩展中，但不出现在接收者报告内。如要包括接收者信息，数据结构应设块数组，与接收报告块数组类似，即块数量由 RC 段表征。

人们总是希望接收质量反馈不仅对发送者有用，而且对其他接收者和第三方监控都有用。发送者可根据反馈改变发送，接收者决定问题是出现在本地、区域还是全局。网络管理员可仅使用接收 RTCP 包、而不是 RTP 数据包的设置无关监控器来估计网络多播的性能。

累计计数用在发送信息与接收者报告块中，可考虑长期和短期测量与提供报告丢失恢复能力之间的差别。后两个接收到报告之间的差别可用来估计近来发送的质量。将 NTP 时标包含在内，从两个报告间隔的差别考虑速率。由于时标独立于数据编码时钟速率，很可能实现编码与设置无关的质量监控器。

丢失包累计数差别给出间隔期间丢掉的数量，而所收到扩展的最后一个系列号的差别则给出间隔期间希望发送的包数量，两者之比等于间隔期间包丢失百分比。如两报告连续，比值应该等于丢失段部分；否则，就不等。每秒包丢失率可通过 NTP 时标差除以丢失部分得到。

根据发送者信息，第三方监控器可计算出载荷平均数据速率与没收到数据间隔的平均包速率，两者比值给出平均载荷大小。如假设包丢失与包大小无关，那么特殊接收者收到的包数量给出此接收者收到的表观流量。

3.8.3 RTCP 源描述分组(SDES)

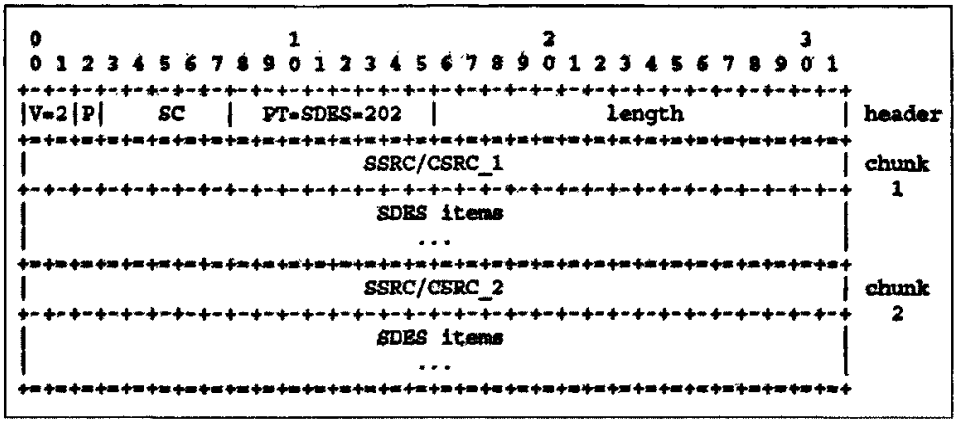


图 7 SDES 包

SDES 提供有关会话参与者的身份认证和会话参与者的其他信息。

version (V), padding (P), length:

与 SR 包定义相同

packet type (PT): 8 bits

值为常量 202

source count (SC): 5 bits

包中描述的 SSRC/CSRC 的数目。包头后的每一项都是对一个 SSRC/CSRC 的描述，其中包含一系列子项。每个子项都具有下列格式：

1. 8 位子项类型标识
2. 8 位该子项的描述文本长度，不包括子项头这 2 个字节
3. 描述文本，以 UTF-8 方式编码，最大长度 255 字节

2) 源描述项

源描述项内容如图 14-02-7 所示。

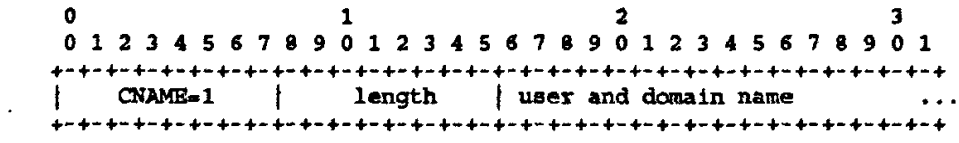


图 8 源描述项

① CNAME

规范终端标识 SDES 项。CNAME 标识属性如下：

- 如发生冲突或重启程序，由于随机分配的 SSRC 标识可能发生变化，需要 CNAME 项提供从 SSRC 标识到仍为常量的源标识的绑定。
- 像 SSRC 标识，CNAME 标识在 RTP 连接的所有参加者中应是惟一的。
- 为了提供一套相关 RTP 连接中某个参加者所采用的跨多媒体工具间的绑定，CNAME 应固定为那个参加者。
- 为方便第三方监控，CNAME 应适合程序或人员定位源。

② NAME

用户名称 SDES 项，这是用于描述源的真正的名称，如"John Doe, Bit Recycler, Megacorp"，可以是用户想要的任意形式（图 14-02-8）。对诸如会议应用，这种名称也许是参加者列表显示所最适宜的形式，它将是除 CNAME 外发送最频繁的项目。设置可建立这样的优先级别。NAME 值至少在连接期间仍希望保持为常数。它不该成为连接的所有参加者中惟一依赖。

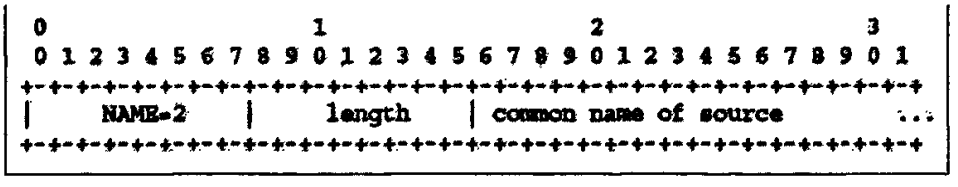


图 9 NAME

③ E-mail

电子邮件地址 SDES 项。邮件地址格式由 RFC822 规定，如"John.Doe@megacorp.com"。连接期间，电子邮件仍希望保持为常数。

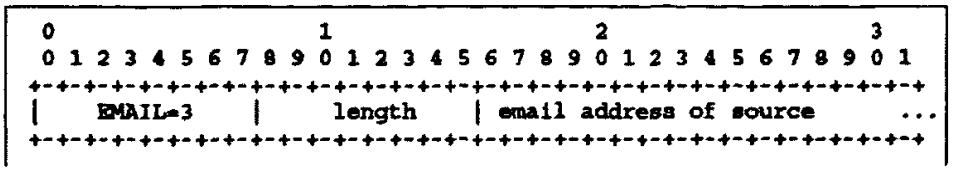


图 10 EMAIL

④ PHONE

电话号码 SDES 项。电话号码应带有加号，代替国际接入代码，如“+86 28 8541 1212”即为中国电话号码。

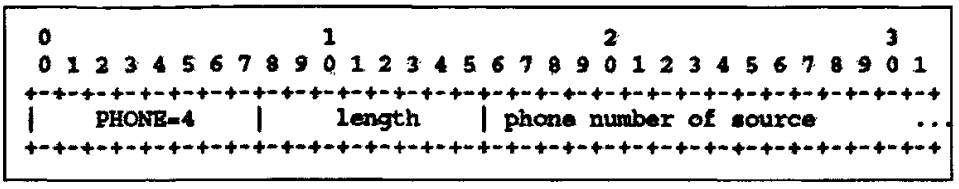


图 11 PHONE

⑤ LOC

用户地理位置 SDES 项。根据不同应用，此项具有不同的详细程度。对会议应用，字符串如"Murray Hill, New Jersey"就足够了。然而，对活动标记系统，字符串如“Room 2A244, AT&T BL MH”也许就适用。细节留给实施或用户，但格式和内容可用设置指示。在连接期间，除移动主机外，LOC 值期望仍保留为常数。

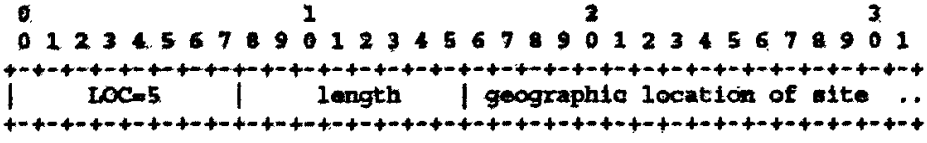


图 12 LOC

⑥ TOOL

应用或工具名称 SDES 项，是一个字符串，表示产生流的应用的名称与版本，如“video tool 1.2”。这部分信息对调试很有用，类似于邮件或邮件系统版本 SMTP 头。TOOL 值在连接期间仍保持常数。

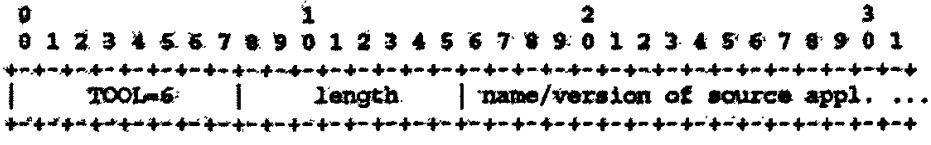


图 13 TOOL

⑦ NOTE

通知 / 状态 SDES 项。推荐的该项语法如下所述，但这些或其他语法可在设置中显式定义。NOTE 项旨在描述源当前状态的过渡信息，如"on the phone, can't talk"，或在讲座期间用于传送谈话的题目。它应该只用于携带例外信息，而不应

包含在全部参加者中，因为这将降低接收报告和 CNAME 发送的速度，因此损害协议的性能。特殊情况下，它不应作为用户设置文件的项目，也不是自动产生。

当其为活动时，由于 NOTE 项对显示很重要，其他非 CNAME 项(如 NAME)传输速率将会降低，结果使 NOTE 项占用 RTCP 部分带宽。若过渡信息不活跃，NOTE 项继续以同样的速度重复发送几次，但以一个串长为零的字符串通知接收者。然而，如对小倍数的重复或约 20~30 倍 RTCP 间隔也没有接收到，接收者也应该考虑 NOTE 项是不活跃的。

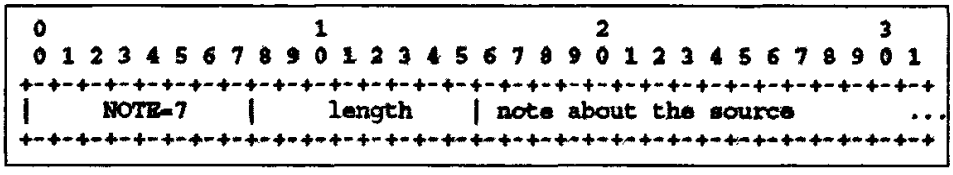


图 14 NOTE

⑧ PRIV

专用扩展 SDES 项。该项用于定义实验或应用特定的 SDES 扩展，它包括由长字符串对组成的前缀，后跟填充该项其他部分和携带所需信息的字符串值。前组长度段为 8 位。前缀字符中是定义 PRIV 项人员选择的名称，惟一对应应用接收到的其他 PRIV 项。应用实现者可选择使用应用名称，如有必要，外加附加子类型标识。另外，推荐其他人根据其代表的实体选择名称，然后，在实体内部协调名称的使用。

注意，前缀消耗了总长为 255 个八位组项的一些空间，因此，前缀应尽可能的短。这个设备和受到约束的 RTCP 带宽不应过载，其目的不在于满足所有应用的全部控制通讯要求。SDES PRIV 前缀没在 IANA 处注册。如证实某些形式的 PRIV 项具有通用性，IANA 应给它分配一个正式的 SDES 项类型，这样就不再需要前缀。这简化了应用，并提高了传输的效率。

3.8.4 断开 RTCP 包（BYE）

如混合器接收到一个 BYE 包，混合器转发 BYE 包，而不改变 SSRC / CSRC 标识。如混合器关闭，它也应该发出一个 BYE 包，列出它所处理的所有源，而不只是自己的 SSRC 标识。作为可选项，BYE 包可包括一个八位组计数，后跟表示离开原因的很多八位组文本，如："camera malfunction"或"RTP loop detected"。字符

串具有同样的编码，如在 SDES 中所描述的。如字符串填充包至下 32 位边界，字符中就不以空结尾；否则，BYE 包以空八位组填充。

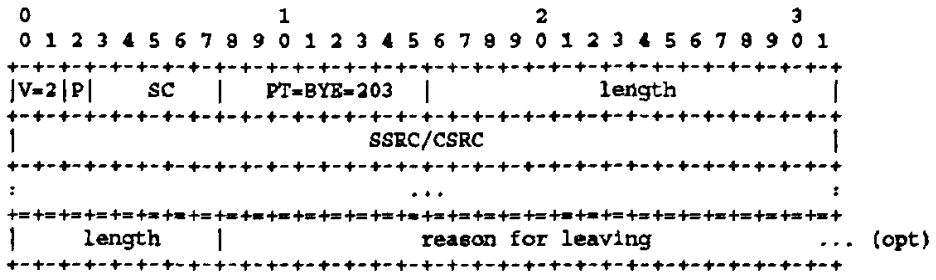


图 15 BYE

（1）版本(V)、填充(P)、长度（length）：如 SR 包中所描述。

（2）包类型(PT)：8 位，包含常数 203，标识 RTCP BYE 包。

（3）源计数(SC)：5 位，包含在 SDES 包中的 SSRC / CSRC 块数量，零值有效，但没有意义。

3.8.5 定义应用的 RTCP 包（APP）

APP 包用于开发新应用和新特征的实验，不要求注册包类型值。带有不可识别名称的 APP 包应被忽略掉。测试后，如确定应用广泛，推荐重新定义每个 APP 包，而不用向 IANA 注册子类型和名称段。

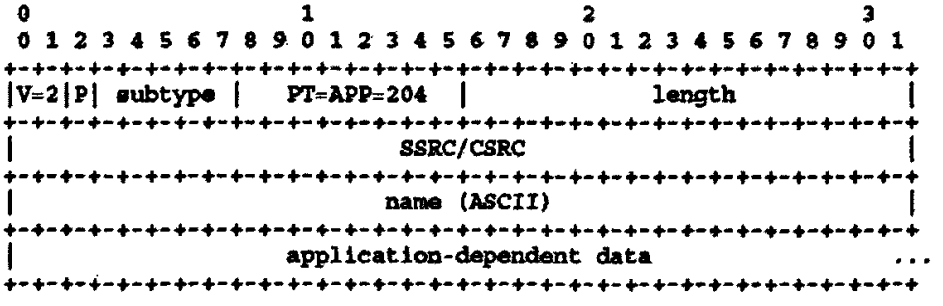


图 16 APP

版本(V)、填充(P)、长度（length）：如 SR 包中所描述。

包类型(PT)：8 位，包含常数 204，标识 RTCP APP 包。

第四章 RTP混合器和转换器

4.1 整体描述

（1）RTP MIXER/TRANSLATOR连接2个或多个所谓的“network cloud”。每个CLOUD由各自所用的网络层和传输层协议，传输目的地定义（地址＋端口）。1个站点可以在多个RTP会话中担当MIXER或TRANSLATOR的角色，但彼此之间是独立的实体。

（2）当一台机器被定义为 TRANSLATOR/MIXER 时，为防止出现循环，应遵循下列原则：

1. 每个CLOUD都在所使用的协议，地址和端口这几项上必须至少有一项存在不同。

2. 多个TRANSLATOR/MIXER不能并行地连接相同的网络，除非特殊需要。

3. 根据不同的目的，可以定义多种TRANSLATOR/MIXER，比如加密解密数据包，改变数据编码格式和协议等。

4.2 两者的区别

除了终端系统以外，RTP支持“转换器”和“混合器”，它们可以当作RTP层次的“过渡系统”。虽然引进这样的支持会给协议带来复杂性，但是实验证明是有用的。使用转换器和混合器是由于防火墙和与低带宽网络的连接引起的。

针对着不同的目的和应用有多种不同的转换器和混合器，转换器和混合器的区别是：转换器传送来自不同源的数据流，而混合器则将它们合成成一个新的数据流在转送。

转换器:转送 RTP 数据包而保持它们的 SSRC 不变，这使得即使所有源的数据包都经过同一个转换器且携带该转换器的网络地址的情况下，接收方还可以知道它们各自的源。一些转换器会原封不动传递数据，而另外一些则可能会改变数据的编码甚至 RTP 数据类型和时戳。除了通过其它方法知道原始数据源所使用的净负荷类型或传输地址外，接收方不能觉察到转换器的存在。

混合器：接收从一个或多个数据源来的 RTP 分组流，可能会改变数据格式，以某种方式合成这些数据流再传出去。由于多个输入的数据流之间一般不会同步，混合器会调整各个流的时钟，从而产生合成的数据流自己的时钟，因此混合器也是一个数据源。所有从混合器出来的数据包都应标上混合器自己的 SSRC，为了能继续知道数据各自的原始源，混合器应把它们的 SSRC 存在 CSRC 列表中。MIXER 相对于 TRANSLATOR 的好处在于较适合传输数据到低带宽网络，缺点在于接受方无法对 SSRC 进行有效控制。

下面所附图为 MIXER 和 TRANSLATOR 的工作方式，原理如下：M 代表 MIXER，T 代表 TRANSLATOR，E 代表 END SYSTEM，E1 带有 SSRC 值 17，E2 带有 SSRC 值 1，通过 M1 时，包的 SSRC 被换成 M 的 SSRC48, 而 17 和 1 被添加到 CSRC 列表。

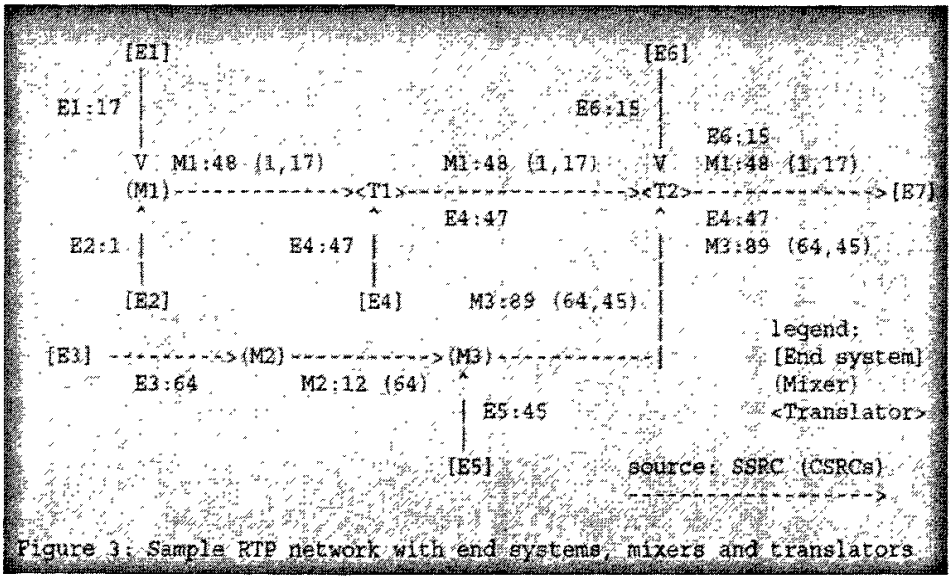


Figure 3: Sample RTP network with end systems, mixers and translators

4.3 两者对 RTCP 包的处理

转换器：

不改变数据的转换器，如只是在组播地址和单播地址之间进行复制，可以只是简单的转送 RTCP 包而不做任何变化。而改变数据的转换器，必须对 SR 和 RR 做出相应的变化，以至还能反映数据特性和接收质量。一般说来，转换器不应将不同源的 SR 和 RR 合到一个分组中，因为那样将降低通过 LSR 和 DLSR 来计算传播延时的精确性。

SR：如数据的编码变了，则发送的字节数要改变。

如将几个数据包合成一个数据包，则发送的分组个数要改变。

如果时间频率变了，则 RTP 时也要进行相应的变化。

SR/RR 的接收方报告块：如转换器将几个数据包合成一个数据包，则“收到的最高序号扩展”也要变化。

SDES：一般不变，但由于传输带宽的原因，也可以进行变化。

BYE：不变。

APP：不边。

混合器：

混合器由于重新生成自己的数据流，所一它的许多 RTCP 信息也要新生成。

SR/RR：重新生成自己的。

SDES：一般不变，但由于传输带宽的原因，也可以进行变化。

BYE：重新生成自己的。

APP：(特定的应用)。

第五章　构建流媒体服务器

5.1　流媒体服务器的基本功能和服务方式

5.1.1　主要功能

流媒体服务器的主要功能：

（1）响应客户的请求，把媒体数据传送给客户。流媒体服务器在流媒体传送期间必须与客户的播放器保持双向通信（这种通信是必需的，因为客户可能随时暂停或快放一个文件）。

（2）响应广播的同时能够及时处理新接收的实时广播数据，并将其编码。

（3）可提供其他额外功能，如：数字权限管理（DRM），插播广告，分割或镜像其他服务器的流，还有组播。

5.1.2　服务方式

流媒体服务器的服务方式

（1）单播。在客户端与媒体服务器之间建立一个单独的数据通道，从1台服务器送出的每个数据包只能传送给1个客户机。

（2）组播。在以组播技术构建的网络上，允许路由器一次将数据包复制到多个通道上。

（3）点播与广播。点播连接是客户端与服务器之间的主动的连接，在点播连接中，用户通过选择内容项目来初始化客户端连接，用户可以开始、停止、后退、快进或暂停流。广播指的是用户被动地接收流，在广播过程中，数据包的单独一个拷贝将发送给网络上的所有用户，客户端接收流，但不能控制流。

5.2　服务器的算法

服务器软件模型主要有两种，即并发服务器和循环服务器。循环服务器(Iterative Server)是指在一个时刻只处理一个请求的服务器。并发服务器(Concurrent Server)是指在一个时刻可以处理多个请求的服务器。事实上，多数

服务器没有用于同时处理多个请求的冗余设备，而是提供一种表面上的并发性，方法是依靠执行多个线程，每个线程处理一个请求，从客户的角度看，服务器就像在并发地与多个客户通信。

由于流媒体服务时间的不定性和数据交互实时性的请求，流媒体服务器一般采用并发服务器算法。本文构建了一个基本的流媒体服务器，能够同时响应多个用户的请求，把本地硬盘流媒体文件或实时数据流（H.263 格式）发送给用户。在应用中，把客户分为请求实时数据的实时客户和请求文件数据的文件客户两类。主要算法为：

（1）打开设备，分配资源。当设备准备好时，创建一个 RTP 实时服务线程和一个 RTCP 实时服务线程。

（2）创建一个 UDP 套接字并将其绑定到所提供服务的地址之上。

（3）反复调用接收模块，接收来自客户的 RTCP 报告，根据其类型做出响应。对新实时客户的请求，把客户地址添加到实时服务的客户列表中，对新文件客户的请求，则创建一个新 RTP 文件服务线程和一个新 RTCP 文件服务线程；对已经在服务中的客户则根据 RTCP 报告的内容调整服务。

RTP 实时服务线程 1：初始化客户列表和 RTP 首部。

RTP 实时服务线程 2：从设备读取媒体数据，把数据发送给实时服务列表中的客户。

RTP 实时服务线程 3：更新 RTP 首部和统计数据。

RTP 实时服务线程 4：计算延时，重复第二步。

RTCP 实时服务线程 1：初始化 RTCP 首部。

RTCP 实时服务线程 2：发送发送方报告给实时服务列表中的客户。

RTCP 实时服务线程 3：计算延时，重复第二步。

RTP 文件服务线程 1：初始化 RTP 首部。

RTP 文件服务线程 2.：从文件读取媒体数据，把数据发送给客户。

RTP 文件服务线程 3：更新已发送数据的统计信息，为生成发送方报告做准备。

RTP 文件服务线程 4：计算延时，调整发送速度，正常情况下开始重复第二步。

RTCP 文件服务线程 1：初始化 RTCP 首部，发送一个源描述(SDES)报文给客户。

RTCP 文件服务线程 2：根据已发送数据的统计信息生成发送方报告，发送给客户。

RTCP 文件服务线程 3：计算延时，正常情况下开始重复第一步。

5.3 流媒体服务器实现

1. 会话和流的两级分用

一个 RTP 会话(Session)包括传给某个指定目的地对(Destination Pair)的所有通信量，发送方可能包括多个。而从同一个同步源发出的 RTP 分组序列称为流(Stream)，一个 RTP 会话可能包含多个 RTP 流。一个 RTP 分组在服务器端发送出去的时候总是要指定属于哪个会话和流，在接收时也需要进行两级分用，即会话分用和流分用。只有当 RTP 使用同步源标识(SSRC)和分组类型(PTYPE)把同一个流中的分组组合起来，才能够使用序列号(Sequence Number)和时间戳(Timestamp)对分组进行排序和正确回放。

多线程的管理

并发服务器模式要求用多线程来提供服务，所以多线程的管理十分重要。在本文构建的服务器中，不同客户的请求和反馈都由服务器的主线程处理，由于实时数据的独有性，不同实时客户可以共用一个 RTP 实时服务线程和一个 RTCP 实时服务线程，这样可以大大减小服务器的负担，而每个文件客户由于请求的文件不同，相应地对速度和开始时间的要求都可能不同，所以需要有自己独有的 RTP 文件服务线程和 RTCP 文件服务线程。

RTP 服务线程负责把实时数据流发送给客户，RTCP 服务线程根据 RTP 线程的统计数据，产生发送方报告给客户。RTP 线程和 RTCP 线程之间通过一段共享内存交互统计数据，对共享内存必须设置互斥体进行保护，防止出现错误读写。在这种方式下，服务器可以根据每个用户的不同请求和具体情况方便地提供不同的服务。

时间戳的处理

时间戳字段是 RTP 首部中说明数据包时间的同步信息，是数据能以正确的时间顺序恢复的关键。时间戳的值给出了分组中数据的第一个字节的采样时间(Sampling Instant)，要求发送方时间戳的时钟是连续、单调增长的，即使在没有数据输入或发送数据时也是如此。在静默时，发送方不必发送数据，保持时间戳的增长，在接收端，由于接收到的数据分组的序号没有丢失，就知道没有发生数据丢失，而且只要比较前后分组的时间戳的差异，就可以确定输出的时间间隔。

RTP 规定一次会话的初始时间戳必须随机选择，但协议没有规定时间戳的单位，也没有规定该值的精确解释，而是由负载类型来确定时钟的颗粒，这样各种应用类型可以根据需要选择合适的输出计时精度。

在 RTP 传输音频数据时，一般选定逻辑时间戳速率与采样速率相同，但是在传输视频数据时，必须使时间戳速率大于每帧的一个滴答。如果数据是在同一时刻采样的，协议标准还允许多个分组具有相同的时间戳值。

媒体数据发送速度的控制

由于 RTP 协议没有规定 RTP 分组的长度和发送数据的速度，因而需要根据具体情况调整服务器端发送媒体数据的速度。对来自设备的实时数据可以采取等时间间隔访问设备缓冲区，在有新数据输入时发送数据的方式，时间戳的设置相对容易。对已经录制好的本地硬盘上的媒体文件，以 H.263 格式的文件为例，由于文件本身不包含帧率信息，所以需要知道录制时的帧率或者设置一个初始值，在发送数据的时候找出发送数据中的帧数目，根据帧率和预置值来计算时延，以适当的速度发送数据并设置时间戳信息。

多种流同步

RTCP 的一个关键作用就是能让接收方同步多个 RTP 流，例如：当音频与视频一起传输的时候，由于编码的不同，RTP 使用两个流分别进行传输，这样两个流的时间戳以不同的速率运行，接收方必须同步两个流，以保证声音与影像的一致。为能进行流同步，RTCP 要求发送方给每个传送一个唯一的标识数据源的规范名(Canonical Name)，尽管由一个数据源发出的不同的流具有不同的同步源标识(SSRC)，但具有相同的规范名，这样接收方就知道哪些流是有关联的。而发送方报告报文所包含的信息可被接收方用于协调两个流中的时间戳值。发送方报告中含有一个以网络时间协议 NTP(Network Time Protocol)格式表示的绝对时间值，接着 RTCP 报告中给出一个 RTP 时间戳值，产生该值的时钟就是产生 RTP 分组中的 TimeStamp 字段的那个时钟。由于发送方发出的所有流和发送方报告都使用同一个绝对时钟，接收方就可以比较来自同一数据源的两个流的绝对时间，从而确定如何将一个流中的时间戳值映射为另一个流中的时间戳值。

5.4 服务器架构

流媒体服务器是由一个父进程构成的，这个父进程分叉出一个子进程，该子进程就是核心服务器。父进程会等待子进程的退出。如果子进程错误退出，则父进程就会分叉出一个新的子进程。

核心服务器的作用是充当网络客户和服务器模块的接口，其中网络客户使用RTP和RTSP协议来发送请求和接收响应，而服务器模块则负责处理请求和向客户端发送数据包。核心服务器通过创建四种类型的线程来完成自己的工作，具体如下：

- 服务器自己拥有的主线程（Main Thread）。这个线程负责检查服务器是否需要关闭，记录状态信息，或者打印统计信息。
- 空闲任务线程（Idle Task Thread）。空闲任务线程管理一个周期性的任务队列。该任务队列有两种类型：超时任务和套接口任务。
- 事件线程（Event Thread）。事件线程负责侦听套接口事件，比如收到RTSP请求和RTP数据包，然后把事件传递给任务线程。

一个或者多个任务（Task）线程。任务线程从事件线程中接收RTSP和RTP请求，然后把请求传递到恰当的服务器模块进行处理，把数据包发送给客户端。缺省情况下，核心服务器为每一个处理器创建一个任务线程。

图 17 总结了客户端，核心服务器的线程，和服务器模块之间的关系。

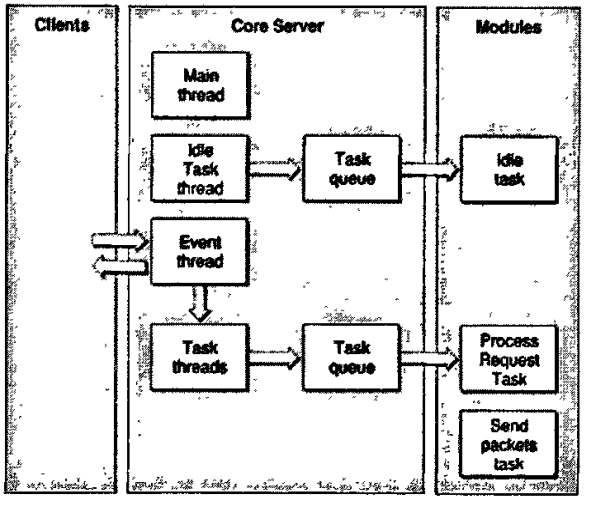


图 17 服务器架构

由于服务器很大程度上是异步的，所以需要为不同的事件提供一个通讯机制。举例来说，当某个用于RTSP连接的套接口得到数据时，需要通知某些组件，数据才能被处理。Task（任务）对象就是执行这种通讯的一般性的机制。

每一个 Task 对象都有两个主要的方法：即 Signal 和 Run。服务器调用 Signal 方法来把一个事件发送给 Task 对象，而 Run 方法则用来为 Task 对象指定处理事件的时机。

每个 Task 对象的目标都是用小的非阻塞的时间片来实现服务器的功能。Run 是一个纯虚函数，当 Task 对象有事件需要处理时被调用。在 Run 函数的内部，Task 对象可以调用 GetEvents 函数来接收当前的和先前已经用信号通知过的事件，并自动使该事件退出队列。Run 函数是永远不可重入的：如果一个 Task 对象在其 Run 函数中调用 GetEvents 函数，然后在 Run 函数完成之前又发出信号，则该 Run 函数只有在当前的函数实例退出之后，才能（因为那个新的事件）再次被调用。事实上，Task 对象的 Run 函数会被反复调用，直到该对象的所有事件全部被 GetEvents 函数清除掉为止。

这个事件触发任务的核心概念被集成到几乎每一个流媒体服务器的子系统中。举例来说，一个 Task 对象可能和一个套接口(Socket)对象相关联。如果 Socket 对象得到一个事件（通过 select()通知，或者通过 Mac OS X 的时间队列），则相应的 Task 对象就会得到信号通知。在这种情况下，Run 函数的函数体中就会包含相应的代码，来处理在该 Socket 对象上接收到的任何事件。

Task 对象使得流媒体服务器用单独一个线程来运行所有连接的做法成为可能，这正是流媒体服务器在单处理器系统上的缺省设置。

5.5 流媒体服务器操作概述

流媒体服务器通过调用模块的特定角色，来处理客户端的请求。每个角色都用来处理一个特定任务。这个部分的内容将描述在服务器启动和关闭，以及处理客户请求的时候，是如何和模块协同工作的。

图 5-1 显示的是服务器在启动和关闭的时候如何和 Register（登记），Initialize（初始化），和 Shutdown（关闭）这些角色协同工作的。

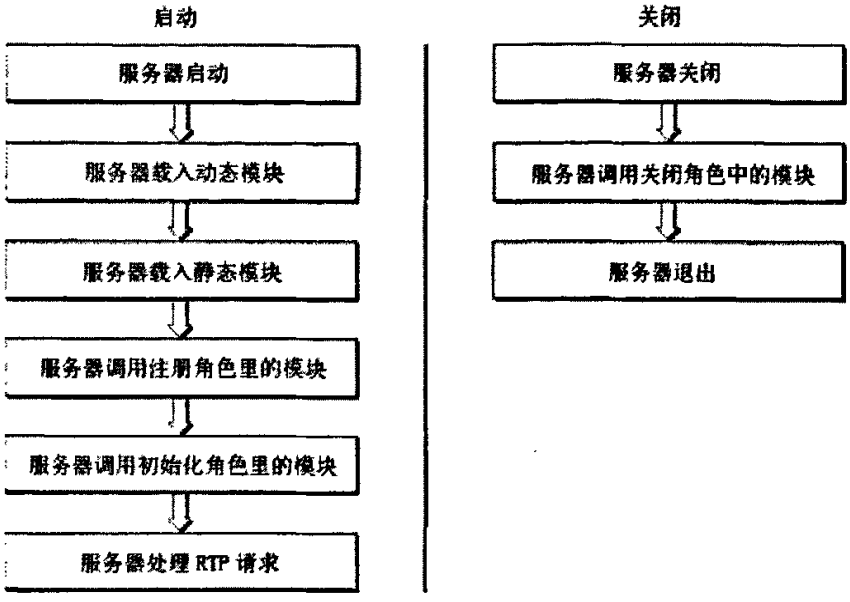


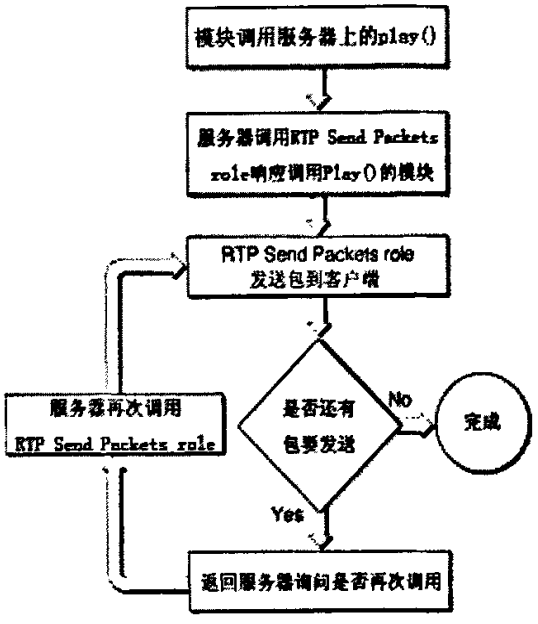
图 5-1 流媒体服务器的启动和关闭

服务器在启动的时候，会首先装载没有被编译到服务器里面的模块（即动态模块），然后再装载编译为服务器一部分的模块（即静态模块）。如果您正在书写模块来代替现有的服务器功能，则请将它编译为动态模块，使它率先被服务器装载。

在模块装载完成之后，服务器会调用每个模块的 Register（注册）角色，每个模块都必须支持这个角色。在 Register 角色中，模块会调用 AddRole()函数来指定自己支持的其它角色。

接下来服务器就以 Initialize（初始化）角色调用每一个注册支持该角色的模块。Initialize 角色执行模块所需要的任何初始化任务，比如分配内存和初始化全局的数据结构。

在服务器关闭的时候，服务器以 Shutdown 角色调用每个注册了该角色模块。在处理 Shutdown 角色的时候，模块应该执行一些收拾现场的任务，并释放全局的数据结构。



RTP Send Packets 角色的总结

RTP Send Packets 角色调用 Write（）函数，在 RTP 会话的基础上向客户发送数据。当 RTP Send Packets 角色发送完成一些数据包之后，就会把控制权返回给服务器，并指定服务器下次调用模块的 RTP Send Packets 角色的间隔时间。这个周期会一直重复，直到所有的媒体数据包被发送完成，或者由于客户请求的原因需要暂停或中止客户会话为止。

5.6　模块规则和流引用

模块可以创建线程，使用互斥锁，并且可以完全自由地使用任何操作系统工具。

流媒体服务器是完全多线程的，因此模块必须做好被别的线程抢入的准备。全局的数据结构以及代码中的关键部分应该用互斥锁进行保护。除非有特别的说明，否则我们只能假定线程的抢入是随时可能发生的。

服务器通常在很少的几个线程，或者可能是某个单一的线程上进行所有的活动，这就要求服务器尽可能使用异步 I/O（服务器实际的行为取决于具体的平台和管理员配置服务器的方法）。

模块应该遵循下面的规则：

尽可能快地执行任务，并将控制权返回给服务器。快速地返回使得服务器可以在大量的客户端之间进行负载均衡。

在执行流 I/O 的时候，需要做好处理 WouldBlock 错误的准备。如果发出的 I/O 请求被阻塞，则 Write（）和 Read（）这些回调函数就返回 WouldBlock。

尽可能避免使用同步 I/O。一个 I/O 操作被阻塞，可能会影响其它客户的流的品质。

编程接口中提供了一些流引用，作为一般化的流抽象。流可以用于读取或者写入很多类型的 I/O 资源，包括（但不限于）文件，错误日志，以及通过 RTP 进行通讯的套接口。

如果没有特殊的说明，所有的流都是异步的。在使用异步的文件系统回调函数时，模块应该做好接收 WouldBlock 结果码的准备，遵守本部分描述的各种类型的流的限制和规则。在需要阻塞当前线程才能完成被请求的操作时，流的回调函数就会返回 WouldBlock 错误。举例来说，如果某个套接口当前受到流量控制的约束，则对该套接口进行操作的 Write（）函数就会返回 WouldBlock。

当模块接收到 WouldBlock 返回码时，调用 RequestEvent（）回调函数，请求服务器在指定的流可以进行 I/O 的时候发出通知。调用 RequestEvent()函数之后，模块应该马上将控制权返回给服务器。这样当指定的流可以进行 I/O 时，服务器将会以完全相同的状态再次调用模块的同一角色。

所有流引用的类型都是 StreamRef。编程接口使用下面这些流类型：

ErrorLogStream

用于将二进制数据写入到服务器的错误日志中。服务器进程中只有一个这种流类型的实例，被作为参数传递给模块的初始化角色。在数据写入到这个流的时候，注册了 Error Log 角色的模块将会被调用。所有对于这种流类型的操作都是同步的。

FileStream

代表一个文件，可以通过调用 OpenFileStream()函数得到。如果打开文件流的时候使用了 FileStreamAsync 标志，则调用者在调用 Read()，Write()函数时应该考虑处理 WouldBlock 返回码。

RTPStreamStream

用于将数据写入到 RTP 客户端。在向这种类型的流写数据时，一个写调用对应一个完整的 RTP 数据包，包括报头。目前不可能用 RequestEvent 回调函数来接收这种流的事件，因此如果 Write()函数返回 WouldBlock，则模块必须周期性地检测阻塞状态是否被消除。这种流引用是 RTPStreamObject 对象的一个属性。

SocketStream

表示一个套接口。这种流类型允许模块使用流事件机制（RequestEvent）来进行 raw 套接口 I/O（事实上，RequestEvent 回调是唯一一个可用于这种流类型的流回调函数）。模块应该异步地读取套接口，而且应该使用操作系统的套接口函数来读写套接口。当那些例程处于阻塞状态时，模块可以通过 RequestEvent()函数来获得阻塞状态被清除的通知。

5.7 模块角色

角色（Role）为模块提供一个定义良好的状态，以执行特定类型的处理。系统用类型为 Role 的选择器（selector）来定义每个角色，表示服务器的内部处理状态及服务器数据的数字，以及服务器数据的可访问性和正当性。根据角色的不同，服务器可能把一个或者多个对象传递给模块。通常情况下，服务器使用对象来和模块进行信息交换。

表 5-1 列出流媒体服务器支持的部分角色。

名称	常数	任务
Register（注册）角色	Register_Role	注册模块支持的角色。
Initialize（初始化）角色	Initialize_Role	执行模块的初始化任务。
Shutdown（关闭）角色	Shutdown_Role	执行清除现场任务。
Reread Preferences（再次读取预置信息）角色	RereadPrefs_Role	重新读取模块的预置信息。
Error Log（错误纪录）	ErrorLog_Role	记录错误信息。

角色		
RTP Send Packets（发送数据包）角色	RTPSendPackets_Role	发送数据包。
Client Session Closing（客户会话关闭）角色	ClientSessionClosing_Role	在客户会话关闭时执行任务。
RTCP Process（处理）角色	RTCPProcess_Role	处理 RTCP 接受方的报告。
Open File Preprocess（打开文件预处理）角色	OpenFilePreProcess_Role	处理打开文件的请求。
Open File（打开文件）角色	OpenFile_Role	处理没有被 Open File Preprocess 角色处理过的打开文件请求。
Read File（读取文件）角色	ReadFile_Role	读取文件。
Request Event File（请求事件文件）角色	RequestEventFile_Role	客户端可能请求接收当文件变为可供读取时发出的通知，这个角色负责处理这些请求。
Close File（关闭文件）角色	CloseFile_Role	关闭先前打开的文件。

除了 Register，Shutdown，和 Reread Preferences 这三个例外角色，服务器在调用模块的其它角色时，会根据角色的不同向模块传入特定的结构。该结构包含模块执行相应角色时需要使用的信息，或者为模块提供一种向服务器返回信息的途径。

Register 角色

模块在 Register 角色中调用 AddRole()函数，来向服务器报告自己支持的角色。

模块还可以在 Register 角色中调用 AddService()函数来注册服务，以及调用 AddStaticAttribute()函数来为对象类型增加静态属性。

服务器在启动的时候，会调用模块的 Register 角色一次。Register 角色总是服务器调用的第一个角色。

如果模块的 Register 角色的返回值不是 NoErr，则不会被装载到服务器中。

Initialize 角色

在调用了所有模块的 Register 角色之后，服务器接着调用各个模块的 Initialize 角色，如果这些模块注册了该角色的话。模块通过 Initialize 角色来初始化全局的和私有的数据结构。

服务器会向每个模块的 Initialize 角色传入一些对象，这些对象可以用于获取服务器的全局属性，预置信息，以及文本错误信息。服务器还会传入错误记录流的引用，可以用于书写错误纪录。所有这些对象都是全局的，因此在这个服务器的运行期间是有效的，任何时候都可以进行访问。

在调用 Initialize 角色时，模块会接收到一个 Initialize_Params 结构，其定义如下：

```
typedef struct
{
ServerObject    inServer;
PrefsObject     inPrefs;
TextMessagesObject  inMessages;
ErrorLogStream inErrorLogStream;
ModuleObject    inModule;
} Initialize_Params;
```

inServer

这是一个 ServerObject 对象，包含服务器的全局属性，以及一个含有当前运行的服务器的所有模块信息的属性。

inPrefs

这是一个 PrefsObject 对象，包含服务器的预置信息。

inMessages

这是一个 TextMessagesObject 对象，模块可以用这个对象来提供本地化文本字符串。

inErrorLogStream

这是一个 ErrorLogStream 流的引用，模块可以用这个引用来输出服务器的错误纪录。往这个流输出数据将导致相应模块的 Error Log 角色被调用。

inModule

这是一个 ModuleObject 对象，模块可以用它来存储自身的信息，包括模块的名称，版本号，以及模块功能描述。

希望自己的 Initialize 角色被调用的模块，必须在其 Register 角色中调用 AddRole 函数，并将角色的实参指定为 Initialize_Role 常数。

如果模块的 Initialize 角色的返回值不为 NoErr，则不会被装载到服务器中。

Shutdown 角色

当服务器准备好关闭时，就会调用各个模块的 Shutdown 角色—如果这些模块注册了该角色的话。

服务器在调用模块的 Shutdown 角色时，没有传入任何参数。

模块可以在 Shutdown 角色中删除之前创建的数据结构，以及执行其它清除现场的工作。

希望自己的 Shutdown 角色被调用的模块必须在其 Register 角色中调用 AddRole 函数，并将角色的实参指定为 Shutdown_Role 常数。

在完成这个角色的处理之后，模块通常应该返回 NoErr。

服务器保证 Shutdown 角色的调用是在关闭自身之前最后一次调用该模块。

RTP Send Packets 角色

服务器在调用 Play 函数的时候就会调用模块的 RTP Send Packets 角色。RTP Send Packets 角色的责任是向客户端发送媒体数据，并告诉服务器什么时候模块的 RTP Send Packets 角色应该再次被调用。

在被调用的时候，RTP Send Packets 角色就会收到一个 RTPSendPackets_Params 结构，该结构定义如下：

```
typedef struct
{
    ClientSessionObject    inClientSession;
    SInt64                 inCurrentTime;
```

```
    TimeVal                 outNextPacketTime;
} RTPSendPackets_Params;
```

inClientSession

这是一个 ClientSessionObject 对象，代表客户会话。

inCurrentTime

当前时间，以服务器的时间单位计算。

outNextPacketTime

间隔时间，以毫秒为单位。在这个角色返回之前，模块需要设定一个合适的 outNextPacketTime 值，这个值是当前时刻和服务器再次为当前会话调用 RTP Send Packets 角色的时刻之间的时间间隔。

任何时候，只要模块为客户会话调用 Play 函数，就会激活该会话的 RTP Send Packets 角色。模块调用 Write 来向客户发送数据。

在模块处理 RTP Send Packets 角色的时候，服务器保证不会调用该模块中任何引用到客户会话的角色，这里的客户会话由 inClientSession 来表示。

希望自己的 RTP Send Packets 角色被调用的模块必须在其 Register 角色中调用 AddRole 函数，并将角色的实参指定为 RTPSendPackets_Role 常数。

在处理完成这个角色之后，模块通常应该返回 NoErr。

RTCP Process 角色

服务器一旦从客户端接收到 RTCP 接收方报告，就会调用模块的 RTCP Process 角色。

RTCP 接收方报告中包含客户端对流品质的反馈信息。这个反馈信息包括丢包率，音频脱包（run dry）的次数，以及每秒的帧数。RTPStreamObject 中的很多属性直接和接收方报告中的各个字段相对应。

在被调用的时候，RTP Process 角色会收到一个 RTCPProcess_Params 结构，该结构定义如下：

```
typedef struct
{
    RTPStreamObject         inRTPStream;
    ClientSessionObject     inClientSession;
```

```
    void*                   inRTCPPacketData;
    UInt32                  inRTCPPacketDataLen;
} RTCPProcess_Params;
```

inRTPStream

这是一个 RTPStreamObject 对象，代表当前这个 RTCP 包所属的 RTP 流。

inClientSession

这是一个 ClientSessionObject 对象，代表客户会话。

inRTCPPacketData

这是一个指针，指向含有即将被处理的数据包缓冲区。

inRTCPPacketDataLen

inRTCPPacketData 指向的缓冲区中正当数据的长度。

处理 RTCP Process 角色的模块通常会监视连接的状态。举例来说，它可能需要跟踪每个连接客户的丢包率，以及更新相应的计数器。

在模块处理 RTCP Process 角色的时候 服务器保证不会调用该模块中引用 RTP 流的其它角色，这里的 RTP 流由 inRTPStream 来表示。

希望自己的 RTCP Process 角色被调用的模块必须在其 Register 角色中调用 AddRole 函数，并将角色的实参指定为 RTCPProcess_Role 常数。

在处理完成这个角色之后，模块通常应该返回 NoErr。

Client Session Closing 角色

服务器对模块的 Client Session Closing 角色的调用，使得模块可以在客户会话的关闭时进行必要的处理。

在被调用的时候，Client Session Closing 角色会收到一个 ClientSessionClosing_Params 结构，该结构定义如下：

```
typedef struct
{
ClientClosing          inReason;
ClientSessionObject    inClientSession;
} ClientSessionClosing_Params;
```

inReason

描述会话关闭的原因。会话被关闭的原因可能是因为客户端发送一个 RTSP 拆卸（tear down）请求（CliSesClosClientTeardown），也可能是因为会话超时，或者客户端没有发出拆卸请求（CliSesClosClientDisconnect）的情况下断开连接。

inClientSession

这是一个 ClientSessionObject object 对象，表示正在关闭的客户会话。

在任何时候，只要由 inClientSession 指定的客户会话即将被拆卸，则 Client Session Closing 角色就会被调用。

在模块处理 Client Session Closing 角色时，服务器保证不调用该模块中其它引用到客户会话的角色，这里的客户会话角色由 inClientSession 表示。

希望自己的 Client Session Closing 角色被调用的模块必须在其 Register 角色中调用 AddRole 函数，并将角色的实参指定为 ClientSessionClosing_Role 角色。

在处理完成这个角色之后，模块通常应该返回 NoErr。

Read File 角色

当模块（或者服务器）为某个文件对象调用 Read()回调函数，以读取相应的文件时，服务器会以 Read File 角色调用各个模块来进行响应。

Read File 角色在被调用的时候会接收到一个 ReadFile_Params 结构，该结构定义如下：

```
typedef struct
{
Object      inFileObject;
UInt64      inFilePosition;
void*       ioBuffer;
UInt32      inBufLen;
UInt32*     outLenRead;
} ReadFile_Params;
```

inFileObject

这是一个文件对象，表示即将被读取的文件。文件系统模块通过这个文件对象来确定与之前调用的 Read() 函数相关联的文件。

inFilePosition

相对于文件开头的偏移量，以字节数记，表示即将被读取部分的开头。服务器对文件的位置进行维护，将它作为文件对象的属性进行存储，因此文件系统模块并不一定要在内部对文件的位置进行缓存，而是在任何时候都可以得到这个位置信息。

ioBuffer

这是一个缓冲区的指针，文件系统模块将读到的数据放置到这个缓冲区上。

ioBufLen

ioBuffer 指向的缓冲区的长度。

outLenRead

实际读取到的字节数。

文件系统模块应该尽可能将文件数据填充到由 ioBuffer 参数指向的缓冲区，这些数据是从文件中读出来的，其在文件中的起始位置由 inFilePosition 参数来指定。

如果文件是以 OpenFileAsync 标识打开的，则在读取数据可能导致线程阻塞的时候，模块应该返回 WouldBlock。否则，模块将会阻塞线程，直到得到所有的数据。当 ioBuffer 指向的缓冲区满，或者到达文件的尾部，文件系统模块应该将 outLenRead 参数设置为已经读取到的字节数，并返回 NoErr。

如果因为某种原因导致读操作失败，则处理这个角色的文件系统模块应该返 RequestFailed。

文件系统模块不需要显式注册这个角色。

Open File Preprocess 角色

服务器会调用 Open File Preprocess 角色，来响应通过 OpenFileObject 回调函数打开文件的模块。模块有义务对这个角色进行处理，以确定是否处理指定需要打开的文件类型。如果要对该文件类型进行处理，而且该文件存在，则模块就打开该文件，更新由服务器提供的文件对象，然后返回 NoErr。

Open File Preprocess 角色在被调用的时候会收到一个 OpenFile_Params 结构，该结构定义如下：

```
typedef struct
{
char*   inPath;
OpenFileFlags  inFlags;
}OpenFile_Params;
```

inPath

这是一个以 null 结尾的字符串，包含将要被打开的文件的全路径。

inFlags

这是一个文件打开的标志参数，指定调用 OpenFileObject 的模块是否可以处理异步的度操作，或者是否希望以从头到尾的顺序读取文件。

如果指定的文件是模块要处理的文件，则模块应该做好打开和配置文件所需要的一切必工作。如果指定的文件是模块要处理的，但是在准备配置文件的过程中发生错误，则模块应该返回 RequestFailed。

希望被以 Open File Preprocess 角色调用的模块必须在其 Register 角色中调用 AddRole()函数，并指定 OpenFilePreprocess_Role 作为角色参数。注册为这个角色的模块也必须处理下面这些角色，但是不必显式地加以注册： Read File 和 Close File。

Close File 角色

当模块（或者服务器）为某个文件对象调用 CloseFile 回调函数，来关闭某个已经打开的文件时，服务器会调用各个模块的 Close File 角色进行相应。

Close File 角色在被调用的时候会接收到一个 CloseFile_Params 结构，该结构定义如下：

```
typedef struct
{
Object          inFileObject;
}CloseFile_Params;
```

inFileObject

这是一个文件对象，表示即将被关闭的文件。文件系统模块使用这个文件对象来确定与之前调用的 Close() 函数相关联的文件。

处理这个角色的模块应该释放之前为这个即将被关闭的文件创建的任何数据结构。对于任何给定的文件对象，这个角色是文件系统模块中最后一个被调用的角色。文件系统模块不需要显式注册这个角色。

模块在处理完成这个角色之后，应该总是返回 NoErr。

5.8 对象定义

对象为模块和服务器之间的数据交换提供了一种途径。对象是由一些属性组成的，这些属性用于存储数据。每个属性都有一个名称，一个属性 ID，一个数据类型，以及一个读写属性值的权限。内置的属性是服务器在通常情况下为某个对象类型定义的属性。

类型为 RTPStreamObjectType 的对象是由一些属性组成的，这些属性描述特定的 RTP 流是音频，视频，或者是文本流。RTP 流对象（RTPStreamObject）就是这种对象类型的一个实例，通过调用 AddRTPStream 函数来创建。一个 RTP 流对象必须和一个客户会话对象（ClientSessionObject）相关联；而一个客户会话对象则可以和任何数目的 RTP 流对象相关联。这些属性对于所有角色都是正当的，只要它们操作的对象属性是从服务器传入的结构中的 RTPStreamObject 成员得到的。表 3 列举了 RTPStreamObjectType 对象的属性。

请注意：所有这些属性对于抢占访问都是安全的，因此可以通过调用 GetValue，GetValueAsString，或者 GetValuePtr 函数来读取。

表 3 RTPStreamObjectType 对象的属性

属性的名称及其描述	访问	数据类型
RTPStrTrackID 标识每个 RTP 流的唯一 ID。	可读，可写，抢占访问安全	UInt32
RTPStrSSRC 由服务器产生的同步源（Synchronization source，简称 SSRC）。服务器保证 SSRC 在同一个会话的所有流之间是唯一的。在所有由服务器产生的 RTCP 发送方报告中都包含有 SSRC。	可读，抢占访问安全	UInt32

RTPStrPayloadName 当前流的媒体名称。如果模块没有显式设定，则这个属性为空值。	可读，可写，抢占访问安全	char
RTPStrPayloadType 当前流的媒体净负荷（Payload）类型。如果模块没有将这个属性设定为 VideoPayloadType 或者 AudioPayloadType，则其值为 UnknownPayloadType。	可读，可写，抢占访问安全	RTPPayloadType
RTPStrFirstSeqNumber 最新的 PLAY 请求发出之后首个数据包的系列号。如果模块知道这个系列号，则必须在调用 Play 函数之前对这个属性进行设定。服务器通过这个属性生成一个正确的 RTSP PLAY 的响应。	可读，可写，抢占访问安全	SInt16
RTPStrFirstTimestampRTP 最新的 PLAY 请求发出之后为当前流生成的首个 RTP 数据包的时间戳。如果模块知道这个值，则必须在调用 Play 函数之前对这个属性进行设定。服务器通过这个属性生成一个正确的 RTSP PLAY 的响应。	可读，可写，抢占访问安全	SInt32
RTPStrTimescale 轨道的时间比例(Timescale)。如果知道这个值，则必须在调用 Play 函数之前对这个属性进行设定。	可读，可写，抢占访问安全	SInt32
RTPStrBufferDelayInSecs 客户缓冲区的尺寸。服务器将这个属性设置为 3 秒。然而模块需要负责确定缓冲区的大小，并据此对该属性进行设定。	可读，抢占访问安全	Float32
RTPStrNetworkMode RTP 流的网络模式。可能的值有 RTPNetworkModeDefault，RTPNetworkModeMulticast，和 NetworkModeUnicast。	可读，抢占访问安全	UInt32
下面这些属性的值来自媒体流中最新的 RTCP 数据包。如果最新的 RTCP 数据包中的某个域是空白的，则服务器会将其对应的属性值设置为 0。		

RTPStrFractionLostPackets 当前流丢失的数据包碎片。	可读，抢占访问安全	UInt32
RTPStrTotalLostPackets 当前流丢失的数据包总数。	可读，抢占访问安全	UInt32
RTPStrJitter 当前流的累计 jitter。	可读，抢占访问安全	UInt32
RTPStrRecvBitRate 客户端接收到的平均位率，以每秒的位数计。	可读，抢占访问安全	UInt32
RTPStrAvgLateMilliseconds 客户端接收到的延迟数据包的平均延迟时间，以毫秒计。	可读，抢占访问安全	UInt16
RTPStrPercentPacketsLost 当前流的丢包百分比。	可读，抢占访问安全	UInt16
RTPStrAvgBugDelayInMsec 平均的缓冲区延迟，以毫秒计。	可读，抢占访问安全	UInt16
RTPStrGettingBetter 如果客户端报告其接收的流品质正在变好，则这个属性的值为非零。	可读，抢占访问安全	UInt16
RTPStrGettingWorse 如果客户端报告其接收的流品质正在变坏，则这个属性的值为非零。	可读，抢占访问安全	UInt16
RTPStrNumEyes 连接到当前流的客户数。	可读，抢占访问安全	UInt32
RTPStrNumEyesActive 播放当前流的客户数。	可读，抢占访问安全	UInt32
RTPStrNumEyesPaused 当前处于连接状态，但又处于暂停状态的客户数目。	可读，抢占访问安全	UInt32
RTPStrTotPacketsRecv 客户端收到的数据包总数。	可读，抢占访问安全	UInt32
RTPStrTotPacketsDropped 被客户端丢弃的数据包数目。	可读，抢占访问安全	UInt16
RTPStrTotPacketsLost 丢失的数据包总数。	可读，抢占访问安全	UInt16

RTPStrClientBufFill 十分之一秒内客户缓冲区的充满程度。	可读，抢占访问安全	UInt16
RTPStrFrameRate 当前的帧率，以每秒的帧数计。	可读，抢占访问安全	UInt16
RTPStrExpFrameRate 期待的帧率，以每秒的帧数计。	可读，抢占访问安全	UInt16
RTPStrAudioDryCount 音频数据不足以流畅播放的次数。	可读，抢占访问安全	UInt16
RTPStrIsTCP 如果当前这个 RTP 流是基于 TCP 上发送的，则这个属性为真；如果是基于 UDP 之上发送，则为假。	可读，抢占访问安全	Bool16
RTPStrStreamRef StreamRef 对象用于将 RTP 或 RTCP 数据包发送给客户端。通过 WriteFlags 来指定被发送的数据包是 RTP 还是 RTCP 包。	可读，抢占访问安全	StreamRef
RTPStrTransportType 传输类型。	可读，抢占访问安全	RTPTransportType

类型为 ClientSessionObjectType 的对象包括一些描述客户会话的属性，这里所说的客户会话定义为一个客户流的表示。客户会话对象（ClientSessionObject）就是这种对象类型的一个实例。客户会话对象的属性对于所有的角色都是正当的，只要该角色接收到的 ClientSessionObject 类型的值是取自服务器传来的结构。所有这些属性对于抢占式访问都是安全的。

表 4ClientSessionObjectType 类型的对象的属性

属性的名称和描述	访问
CliSesStreamObjects 可重复访问的属性，包含属于当前会话的所有 RTP 流的引用（RTPStreamObject）。	可读，抢占访问安全
CliSesCreateTimeInMsec 会话创建的时间，以毫秒计。	可读，抢占访问安全

CliSesFirstPlayTimeInMsec 首次调用 Play()的时间，以毫秒计。	可读，抢占访问安全
CliSesPlayTimeInMsec 最近一次调用 Play()的时间，以毫秒计。	可读，抢占访问安全
CliSesAdjustedPlayTimeInMsec 向前调整过的最近一次播放请求的发出时间，这是为了推迟发送数据包，直到播放请求的响应被发出。这个时间以毫秒计。	可读，抢占访问安全
CliSesRTPBytesSent 当前会话已经发送的 RTP 字节数。	可读，抢占访问安全
CliSesRTPPacketsSent 当前会话已经发送的 RTP 数据包的数目。	可读，抢占访问安全
CliSesState 当前会话的状态。可能的值是 PausedState 和 PlayingState。	可读，抢占访问安全
CliSesMovieDurationInSecs 当前会话的电影时长，以秒计。如果模块没有进行设定，则该值为 0 。	可读，可写，抢占访问安全
CliSesMovieSizeInBytes 电影的尺寸，以字节计。如果模块没有进行设定，则该值为 0 。	可读，可写，抢占访问安全
CliSesMovieAverageBitRate 每秒钟的平均位率，这是由总共的 RTP 位除以电影时长得到。如果模块没有进行设定，则该值为 0 。	可读，可写，抢占访问安全
CliSesHostName 当前会话的主机名称。	可读，抢占访问安全
CliRTSPSessRemoteAddrStr 客户端的 IP 地址，以带点的十进制数格式表示。	可读，抢占访问安全
CliRTSPSessLocalDNS 当前 RTSP 连接中的本地 IP 地址对应的 DNS 名称。	可读，抢占访问安全

CliRTSPSessLocalAddrStr 当前 RTSP 连接中的本地 IP 地址，以带点的十进制数格式表示。	可读，抢占访问安全
CliRTSPSesUserName 发起最新请求的用户名。	可读，抢占访问安全
CliRTSPSesURLRealm 最新请求的范围信息。	可读，抢占访问安全
CliRTSPReqRealStatusCode 最新请求的状态信息（和 RTSPReqRealStatusCode 会话一样）。	可读，抢占访问安全
CliTeardownReason 连接断开的原因。如果不是客户端请求，则模块必须调用 Teardown()函数来设置连接断开的原因。	可读，可写，抢占访问安全
CliSesReqQueryString 创建当前会话的客户请求的请求字符串。	可读，抢占访问安全
CliRTSPReqRespMsg 当响应最近一次客户请求发生错误时发送给客户的错误信息。	可读，抢占访问安全
CliSesCurrentBitRate 电影的位率。	可读，抢占访问安全
CliSesPacketLossPercent 丢失数据包的百分比。例如，.5 = 50%	可读，抢占访问安全
CliSesTimeConnectedinMsec 客户会话的连接时长，以毫秒计。	可读，抢占访问安全
CliSesCounterID 会话的唯一 ID，基于计数器机制。	可读，抢占访问安全

5.9 函数定义

AddRole

增加一个角色。AddRole()函数告诉服务器可以调用您的模块中由 inRole 参数指定的角色。AddRole 函数只能在模块的 Register 角色中调用。

Error AddRole (Role inRole);

参数描述

inRole

输入参数，是一个类型为 Role 的值，指定即将被加入的角色。

result

结果码。如果 AddRole()函数在除了 Register 之外的角色中被调用，可能的返回值为 NoErr 和 OutOfState；如果模块正在注册 RTSP Request 角色，而已经有一个模块注册了该角色，则返回 RequestFailed；如果指定的角色不存在，则返回 BadArgument。

Read

从一个流中读取数据，放入缓冲区。

Int Read(StreamRef inRef, void* ioBuffer, UInt32 inBufLen, UInt32* outLengthRead);

参数描述：

inRef

输入参数，是一个类型为 StreamRef 的值，指定即将读取数据的流。调用 OpenFileObject 函数可以得到您希望读取的文件的流引用。

ioBuffer

输入参数，是一个指针，指向存放读取数据的缓冲区。

inBufLen

输入参数，是一个类型为 UInt32 的值，指定 ioBuffer 缓冲区的长度。

outLenRead

输出参数，是一个指针，指向类型为 UInt32 的值，表示读取的字节数。

result

结果码。可能的返回值之一是 NoErr；如果参数不正当，返回 BadArgument；如果读操作被阻塞，则返回 WouldBlock；如果读操作失败，则返回 RequestFailed。

Write

向一个流写入数据。将一个缓冲区中的数据写入到一个流。

```
Int  Write( StreamRef  inRef, void* inBuffer, UInt32 inLen, UInt32*
outLenWritten, UInt32 inFlags);
```

参数描述：

inRef

输入参数，是一个类型为 StreamRef 的值，指定即将写入数据的流。

inBuffer

输入参数，是一个缓冲区指针，指向即将写入的数据。

inLen

输入参数，是一个类型为 UInt32 的值，指定 inBuffer 缓冲区中数据的长度。

outLenWritten

输出参数，是一个指针，指向类型为 UInt32 的值，表示即将写入的字节数。

inFlags

输入参数，是一个类型为 UInt32 的值，可能的值请参见下面部分。

result

结果码。可能的返回值之一是 NoErr；如果参数不正当，返回 BadArgument；如果流的接受方已经不再处于连接状态，返回 NotConnected；如果流的缓冲数据在这个时刻不能完全被写入，则返回 WouldBlock。

Write 回调例程将一个缓冲区中的数据写入到一个流。

下面的枚举定义了 inFlags 参数可能的常数值：

```
enum
{
qtssWriteFlagsIsRTP = 0x00000001,
qtssWriteFlagsIsRTCP= 0x00000002
};
```

在向 RTP 流写入数据的时候，这些标志是相互关联的，它们告诉服务器写入的数据应该通过 RTP 通道(WriteFlagsIsRTP)发送，还是应该通过指定 RTP 流的 RTCP 通道发送（WriteFlagsIsRTCP）。

OpenFileObject

打开指定的文件并返回相应的文件对象。文件对象的属性之一是一个流引用，可以传递给流回调例程，以便读写文件数据，或者执行其它文件操作。

```
Int OpenFileObject( char* inPath, OpenFileFlags  inFlags, Object *
outFileObject);
```

参数描述：

inPath

输入参数，是一个以null结尾的字符串，表示即将被打开的文件在本地文件系统中的全路径。

inFlags

输入参数，是一个类型为OpenFileFlags的值，指定描述如何打开文件的标志。

outFileObject

输出参数，是一个指针，指向类型为Object的值，打开后的文件对应的文件对象将存放在这里。

result

结果码。可能的返回值之一是NoErr；如果参数不正当，返回BadArgument；如果指定的文件不存在，则返回FileNotFound。

CloseFileObject

用于关闭指定的文件。

```
Int CloseFileObject(Object inFileObject);
```

参数描述：

inFileObject

输入参数，是一个类型为Object的值，表示即将被关闭的文件。

result

结果码。可能的返回值之一是NoErr；如果参数不正当，返回BadArgument。

AddRTPStream

使一个模块可以向客户端发送 RTP 数据包，以响应 RTSP 请求。多次调用 AddRTSPStream 函数可以向会话中加入多个流。

如果希望开始播放一个流，可以调用 Play 函数。

Int AddRTPStream(ClientSessionObject inClientSession, RTSPRequestObject inRTSPRequest,
RTPStreamObject* outStream, AddStreamFlags inFlags);

参数描述

inClientRequest

输入参数，是一个类型为 ClientSessionObject 的值，标识 RTP 数据包的发送功能即将被打开的客户会话。

inRTSPRequest

输入参数，是一个类型为 RTSPRequestObject 的值。

outStream

输出参数，是一个指针，指向类型为 RTPStreamObject 的值，该值含有新创建的流。

inFlags

输入参数，是一个类型为 AddStreamFlags 的值，指定流的选项。

result

结果码。可能的返回值之一是 NoErr；如果 RTPStreamObject 对象不能被创建，返回 RequestFailed；如果参数不正当，则返回 BadArgument。

Play

Int Play(ClientSessionObject inClientSession, RTSPRequestObject inRTSPRequest, PlayFlags inPlayFlags);

开始播放与客户会话相关联的流。只有调用了 AddRTPStream()函数的模块才能调用 Play()函数。 在调用 Play()函数之前，模块应该为当前这个 RTP 流设置下面这些 RTPStreamObject 对象的属性：

- RTPStrFirstSeqNumber，这个属性应该设置为最后一个 PLAY 请求发出之后的第一个数据包的序列号。服务器通过这个序列号生成一个正确的 RTSP PLAY 响应。

- RTPStrFirstTimestamp，这个属性应该设置为最后一个 PLAY 请求发出之后为这个流生成的第一个 RTP 数据包的时间戳。服务器通过这个时间戳生成一个正确的 RTSP PLAY 响应。
- RTPStrTimescale，这个属性应该设置为轨道的时间比例尺。

在调用 Play()函数之后，模块的 RTP Send Packets 角色就会被调用。可以调用 Pause()函数来暂停流的播放，或者调用 Teardown()来关闭客户会话。

参数描述：

inClientSession

输入参数，是一个类型为 ClientSessionObject 的值，标识一个客户会话，该会话的 RTP 数据包的发送功能已经在之前调用 AddRTPStream()函数打开。

inRTSPRequest

输入参数，是一个类型 RTSPRequestObject 的值。

inPlayFlags

输入参数，是一个类型为 PlayFlags 的值。将 inPlayFlags 参数设置为 PlaySendRTCP 常数会使服务器在播放的过程中自动产生 RTCP 发送方报告。否则，模块会负责产生具体描述播放特征的发送方报告。

result

结果码。可能的返回值之一是 NoErr；如果参数不正当，返回 BadArgument；如果会话中没有加入流，则返回 RequestFailed。

Pause

暂停一个正在播放的流。只有调用了 AddRTPStream()函数的模块才能调用 Pause()函数。

Int Pause(ClientSessionObject inClientSession);

参数描述：

inClientSession

输入参数，是一个类型为 ClientSessionObject 的值，标识即将被暂停的客户会话。

result

结果码。可能的返回值之一是 NoErr；如果参数不正当，则返回 BadArgument。

Teardown

关闭一个客户会话。只有调用了 AddRTPStream()函数的模块才能调用 Teardown()函数。调用 Teardown()会使上层模块的 Client Session Closing 角色被调用，调用时传入由 inClientSession 参数标识的客户会话。

Int Teardown(ClientSessionObject inClientSession);

参数描述：

inClientSession

输入参数，是一个类型 ClientSessionObject 的值，标识即将被关闭的客户会话。

result

结果码。可能的返回值之一是 NoErr；如果参数不正当，则返回 BadArgument。

5.10 算法设计

我们提供例子来展现RTP发送器和接收器算法。可能有其他执行方法在特别的操作环境里更快或者有其他优势。以下的定义用于所有的例子；结构的定义只有32位“big-endian”（最重要的是前8位）是有效的结构。假设位的填充严格按照“big-endian”的位顺序，没有额外的填充。位的填充在当需要构建便携式实施时需要修改。

```
#include <sys/types.h>
typedef unsigned char u_int8;
typedef unsigned short u_int16;
typedef unsigned int u_int32;
typedef short int16;
#define RTP_VERSION 2
#define RTP_SEQ_MOD (1<<16)
```

```
#define RTP_MAX_SDES 255 /* maximum text length for SDES */
typedef enum {
RTCP_SR = 200,
RTCP_RR = 201,
RTCP_SDES = 202,
RTCP_BYE = 203,
RTCP_APP = 204
} rtcp_type_t;
typedef enum {
RTCP_SDES_END = 0,
RTCP_SDES_CNAME = 1,
RTCP_SDES_NAME = 2,
RTCP_SDES_EMAIL = 3,
RTCP_SDES_PHONE = 4,
RTCP_SDES_LOC = 5,
RTCP_SDES_TOOL = 6,
RTCP_SDES_NOTE = 7,
RTCP_SDES_PRIV = 8
} rtcp_sdes_type_t;
/*
* RTP data header
*/
typedef struct {
unsigned int version:2;
unsigned int p:1;
unsigned int x:1;
unsigned int cc:4;
unsigned int m:1;
unsigned int pt:7;
u_int16 seq;
u_int32 ts;
u_int32 ssrc;
```

```
u_int32 csrc[1];
} rtp_hdr_t;
/*
* RTCP common header word
*/
typedef struct {
unsigned int version:2;
unsigned int p:1;
unsigned int count:5;
unsigned int pt:8;
u_int16 length;
} rtcp_common_t;
/*
* Big-endian mask for version, padding bit and packet type pair
*/
#define RTCP_VALID_MASK (0xc000 | 0x2000 | 0xfe)
#define RTCP_VALID_VALUE ((RTP_VERSION << 14) | RTCP_SR)
/*
* Reception report block
*/
typedef struct {
u_int32 ssrc;
unsigned int fraction:8;

int lost:24;
u_int32 last_seq;
u_int32 jitter;
u_int32 lsr;
u_int32 dlsr;
} rtcp_rr_t;
/*
* SDES item
```

```
*/
typedef struct {
u_int8 type;
u_int8 length;
char data[1];
 } rtcp_sdes_item_t;
/*
* One RTCP packet
*/
typedef struct {
rtcp_common_t common; /* common header */
union {
/* sender report (SR) */
struct {
u_int32 ssrc; /* sender generating this report */
u_int32 ntp_sec; /* NTP timestamp */
u_int32 ntp_frac;
u_int32 rtp_ts; /* RTP timestamp */
u_int32 psent; /* packets sent */
u_int32 osent; /* octets sent */
rtcp_rr_t rr[1]; /* variable-length list */
} sr;
/* reception report (RR) */
struct {
u_int32 ssrc; /* receiver generating this report */
rtcp_rr_t rr[1]; /* variable-length list */
} rr;
/* source description (SDES) */
struct rtcp_sdes {
u_int32 src; /* first SSRC/CSRC */
rtcp_sdes_item_t item[1]; /* list of SDES items */
} sdes;
```

```
/* BYE */
struct {
u_int32 src[1]; /* list of sources */

/* can' t express trailing text for reason */
} bye;
} r;
} rtcp_t;
typedef struct rtcp_sdes rtcp_sdes_t;
/*
* Per-source state information
*/
typedef struct {
u_int16 max_seq;
u_int32 cycles;
u_int32 base_seq;
u_int32 bad_seq;
u_int32 probation;
u_int32 received;
u_int32 expected_prior;
u_int32 received_prior;
u_int32 transit;
u_int32 jitter;
/* ... */
} source;
```

5.10.1 RTCP 报头有效性检查

下列检查能用运用到RTCP包：

1. RTP版本字段必须=2
2. 第一个复合包里的RTCP包的负载类型字段必须等于SR或RR.
3. 第一个复合RTCP包填充位（P位）应该是0，因为只有最后的位有可能需要

填充。

4. 单独的RTCP包的长度字段必须达到接收到的复合RTCP包的所有长度。这是一个相当强的有效性检验。

以下的代码段实现了所有的这些检查。因为未知包类型或许会被呈现并且应该忽略，所以并没有为后来的包检查包类型。

```
u_int32 len; /* length of compound RTCP packet in words */
rtcp_t *r; /* RTCP header */
rtcp_t *end; /* end of compound RTCP packet */
if ((*(u_int16 *)r & RTCP_VALID_MASK) != RTCP_VALID_VALUE) {
/* something wrong with packet format */
}
end = (rtcp_t *)((u_int32 *)r + len);
do r = (rtcp_t *)((u_int32 *)r + r->common.length + 1);
while (r < end && r->common.version == 2);
if (r != end) {
/* something wrong with packet format */
}
```

5.10.2 定义 RTP 包的预计和丢失数

为了计算包的遗失率，从每个源接收到的包的预计数和实际数目都必须可知，使用每个源的状态信息定义源结构请参考下列代码中的S指针。接收到的包的数量只在当包达到时计算，包括任何延迟的和重复的包。由于接收到的最高序列号与接收到的第一个序列号之间的差异，包的预计数量能被接收器计算。由于序列号只有16位并且被围绕，因此有必要扩展最高的序列号和序列号反转的（ Shifted）计数器。

```
extended_max = s->cycles + s->max_seq;
expected = extended_max - s->base_seq + 1;
```

丢失包的数量被定义为预计的数字减去实际接受的数字：

```
lost = expected - s->received;
```

由于该数字是24位，所以应该是“0xffffff”填充，而不是0。

在最后报告间隔（由于之前SR或RR包的发送）过程中丢失的包片段被从间隔期间预计的和接收到的包的差异中计算出来， 当产生之前的报告时存储expected_prior 和received_prior值。

```
expected_interval = expected - s->expected_prior;
s->expected_prior = expected;
received_interval = s->received - s->received_prior;
s->received_prior = s->received;
lost_interval = expected_interval - received_interval;
if (expected_interval == 0 || lost_interval <= 0) fraction = 0;
else fraction = (lost_interval << 8) / expected_interval;
```

The resulting fraction is an 8-bit fixed point number with the binary point at the left edge.

5.10.3 产生 RTCP SDES 包

这个功能是在buffer b中建立SDES块组成参数，提供数组类型、值和b的长度.

```
char *rtp_write_sdes(char *b, u_int32 src, int argc, rtcp_sdes_type_t type[], char *value[],int length[])
{
rtcp_sdes_t *s = (rtcp_sdes_t *)b;
rtcp_sdes_item_t *rsp;
int i;
int len;
int pad;
/* SSRC header */
s->src = src;
rsp = &s->item[0];
/* SDES items */
for (i = 0; i < argc; i++) {
rsp->type = type[i];
len = length[i];
if (len > RTP_MAX_SDES) {
```

```
/* invalid length, may want to take other action */
len = RTP_MAX_SDES;
}
rsp->length = len;
memcpy(rsp->data, value[i], len);
rsp = (rtcp_sdes_item_t *)&rsp->data[len];
}
/* terminate with end marker and pad to next 4-octet boundary */
len = ((char *) rsp) - b;
pad = 4 - (len & 0x3);
b = (char *) rsp;
while (pad--) *b++ = RTCP_SDES_END;
return b;
}
```

5.10.4 解析 RTCP SDES 包

本函数解析了一个SDES包，调用find_member()函数为会话成员来寻找到信息的指针给SSRC标志符和用来为该成员存储新SDES信息的member_sdes()函数。本函数期望一个到RTCP包报头的指针。

```
void rtp_read_sdes(rtcp_t *r)
{
int count = r->common.count;
rtcp_sdes_t *sd = &r->r.sdes;
rtcp_sdes_item_t *rsp, *rspn;
rtcp_sdes_item_t *end = (rtcp_sdes_item_t *)
((u_int32 *)r + r->common.length + 1);
source *s;
while (--count >= 0) {
rsp = &sd->item[0];
if (rsp >= end) break;
```

```
s = find_member(sd->src);
for (; rsp->type; rsp = rspn ) {
rspn = (rtcp_sdes_item_t *)((char*)rsp+rsp->length+2);
if (rspn >= end) {
rsp = rspn;
break;
}
member_sdes(s, rsp->type, rsp->data, rsp->length);
}
sd = (rtcp_sdes_t *)
((u_int32 *)sd + (((char *)rsp - (char *)sd) >> 2)+1);
}
if (count >= 0) {
/* invalid packet format */
}
}
```

结论

流媒体技术的应用日益广泛，对流媒体技术的研究具有很大的实际意义,本文通过对 RTP/RTCP 协议的研究，分析流媒体服务器的一般功能和结构，给出构建一个基本的流媒体服务器的实现方案，实验证明可以同时满足多个实时和文件客户的要求。

随着多媒体数据在 Internet 上所承担的作用变得越来越重要，需要实时传输音频和视频等多媒体数据的场合也将变得越来越多，如 IP 电话、视频点播、在线会议等。RTP 是用来在 Internet 上进行实时流媒体传输的一种协议，目前已经被广泛地应用在各种场合。

致谢

在这篇论文即将完成之际，我首先要向我的导师李毅教授表示最诚挚的感谢！本论文的顺利完成，与他的引导是分不开的。在工程硕士第一年的学习过程中，他对于技术前沿的敏锐把握和研究方向的指导，都给予我极大的帮助。最重要的是让我找准了自己的发展方向。我很庆幸能在关键的时候得到李老师的帮助。在他的帮助下，我能够更迅速的领悟复杂的系统软件和层出不穷的应用软件技术，同时在他的启发下我的技术视野不仅仅局限于软件技术。这使我在技术上更加全面，在将来的产品开发中能够更有效的解决问题。

参考文献

[1]RFC 1889- 1996, RTP: ATransport Protocol for Real- Time Applications[S].

[2]RFC 1890- 1996, RTP Profile for Audio and Video Conferences with Minimal Control[S].

[3]张占军,韩承德,杨学良.多媒体实时传输协议 RTP[J].计算机工程与应用, 2001, (4): 9～11.

[4]陈化,游志胜,洪玫,等.基于 RTP 协议的实时语音传输性能优化方案[J]. 计算机应用, 2003,(10):18～20.

[5]刘涛,周兵,李笑佳.基于 RTP 的多媒体可靠实时传输[J].华中科技大学学报(自然科学版),2003,(1):17～19.

[6]蒋东兴.Windows Sockets 网络程序设计大全[M].北京:清华大学出版社,1999.

[7]张延军,杨静飞.自组织网中的 IP/UDP/RTP 头压缩算法[J].华中科技大学学报(自然科学版),2003,(1):10～12.

[8] J.-C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the internet," in SIGCOMM Symposium on Communications Architectures and Protocols , (London, England), pp. 58--67, ACM, Aug. 1994.

[9] I. Busse, B. Deffner, and H. Schulzrinne, "Dynamic QoS control of multimedia applications based on RTP," Computer Communications , Jan. 1996.

[10] S. Floyd and V. Jacobson, "The synchronization of periodic routing messages," in SIGCOMM Symposium on Communications Architectures and Protocols (D. P. Sidhu, ed.), (San Francisco, California), pp. 33--44, ACM, Sept. 1993. also in [25].

[11] J. A. Cadzow, Foundations of digital signal processing and data analysis New York, New York: Macmillan, 1987.

[12] International Standards Organization, "ISO/IEC DIS 10646-1:1993 information technology -- universal multiple-octet coded character set (UCS) -- part I: Architecture and basic multilingual plane," 1993.

[13] The Unicode Consortium, The Unicode Standard New York, New York: Addison-Wesley, 1991.

[14] Mockapetris, P., "Domain Names - Concepts and Facilities", STD13, RFC 1034, USC/Information Sciences Institute, November 1987.

[15] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, USC/Information Sciences Institute, November 1987.

[16] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, Internet Engineering Task Force, October 1989.

[17] Rekhter, Y., Moskowitz, R., Karrenberg, D., and G. de Groot, "Address Allocation for Private Internets", RFC 1597, T.J. Watson Research Center, IBM Corp., Chrysler Corp., RIPE NCC, March 1994。

[18] Lear, E., Fair, E., Crocker, D., and T. Kessler, "Network 10 Considered Harmful (Some Practices Shouldn't be Codified)", RFC 1627, Silicon Graphics, Inc., Apple Computer, Inc., Silicon Graphics, Inc., July 1994.

[19] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[20] W. Feller, An Introduction to Probability Theory and its Applications, Volume 1 , vol. 1. New York, New York: John Wiley and Sons, third ed., 1968.

[21] Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", RFC 1423, TIS, IAB IRTF PSRG, IETF PEM WG, February 1993.

[22] V. L. Voydock and S. T. Kent, "Security mechanisms in high-level network protocols," ACM Computing Surveys , vol. 15, pp. 135--171, June 1983.

[23] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.

[24] S. Stubblebine, "Security services for multimedia conferencing," in 16th National Computer Security Conference , (Baltimore, Maryland), pp. 391--395, Sept. 1993.

[25] S. Floyd and V. Jacobson, "The synchronization of periodic routing messages," IEEE/ACM Transactions on Networking , vol. 2,pp. 122-136, April 19

个人简历

张洪宇，男，1979 年 8 月 19 日出生，2002 年 7 月毕业于四川大学，2004 年 9 月进入电子科技大学软件学院学习。

研究生期间参与科研与研发项目：

1、RTP 协议分析

2、流媒体服务器软件设计

基于RTP协议流媒体服务器的研究

作者： 张洪宇
学位授予单位： 电子科技大学

相似文献(10条)

1.学位论文 翁伟东 移动流媒体服务器系统在3G上实现 2005

流媒体最先出现在固定互联网络。目前在网络上传输音视频等多媒体信息的方法主要采用下载和流式两种传输方案。对于下载方案而言，由于大文件往往需要大量的存储容量，同时受到网络带宽的限制，下载时间长，延迟很大。对于流式传输而言，文件边下载边播放，当媒体在客户机上播放时，文件的剩余部分在后台从服务器内继续下载，不仅启动延时缩短，而且也不需要太大的缓存容量，避免了用户必须等待整个文件全部下载后才能观看的缺点。在无线网络，流媒体业务主要面向cdma2000、GPRS/EDGE、UMTS等提供较高带宽(100kbps以上)的无线分组网络，可根据流媒体源的不同分为视频业务和音频业务。目前，基于有线的VOD、AOD业务，已经受到了用户的广泛欢迎。在无线网络，空中接口带宽的增加为流媒体业务的开展提供了良好的基础，结合无线系统不受时间、地点限制的特点，使得移动流媒体业务更具吸引力。本文主要从以下两个方面展开研究工作：

1.熟悉移动流媒体传输的基本原理和流媒体系统的基本结构，实现一个移动流媒体传输系统。系统采用RTSP，RTP等传输协议栈实现流媒体的直播和点播服务。

2.针对无线环境下流媒体传输中存在的带宽不足、接入多样性等问题，详细探讨了流媒体传输的自适应控制技术。基于RTP协议研究端到端的自适应视频传输控制方法。能够根据网络带宽变化调整速率，减小丢包率，实现较高的网络带宽利用率。

2.学位论文 史轶 流媒体应用及其QoS控制机制的研究 2002

该文在Windows Media基础上设计并实现了一个完整的流媒体服务系统,主要包括流媒体服务器,存储系统,播放器,节目和用户管理等部分,可应用于视频点播.现场直播,多媒体广告等领域.在局域网这类带宽充足网络环境中,流媒体应用的关键是流媒体服务器的性能.该文通过对三套视频点播系统进行性能评测.确定了流媒体服务器的磁盘外部交换率,带宽,CPU构成流媒体应用的硬件瓶颈.以及流媒体服务器的调度程序构成流媒体应用的软件瓶颈,这也是性能优化的主要问题.在Internet这类无法可靠保证服务质量（Quality of Service）的网络环境中,拥塞和带宽不足成为流媒体应用的主要瓶颈,因此Internet中流媒体应用关键是QoS控制机制.该文设计并实现了一个基于RTP协议和MEPG-4可扩展编码的QoS控制机制.这是Internet智能流媒体服务系统研究的一部分.

3.学位论文 张静 基于嵌入式系统的流媒体服务器技术研究与实现 2005

计算机技术和高速宽带网络的发展，为多媒体技术的应用提供了广阔的空间，网络多媒体是电子技术、计算机技术、通信技术相互结合的产物。稳定可靠的网络多媒体系统需要解决视/音频编解码、网络传输、远端设备控制等技术问题，同时还要考虑系统的灵活性和精简性，将嵌入式系统和流媒体技术相结合，构建一个灵活高效、扩展性强、可靠性高的系统是当前网络多媒体技术的发展趋势。嵌入式的视频会议、视频点播、远程教育、无人监控、数字家庭等分布式多媒体应用逐渐走入人们的生活，给人们的工作和生活带来了极大的便利和无穷的乐趣。

该文所研究的内容是由网络多媒体应用中的音视频传输需求提出，论文的论述从深度上分为研究和实现两个层面，在广度上，从嵌入式系统技术和流媒体技术两大热门科学入手：旨在研究基于实时传输协议(RTP)的流媒体传输方法，研究流媒体服务器的相关技术，研究资源预留协议(RSVP)；实现基于ARM9的嵌入式Linux系统开发，实现H.263视频流和G.729音频流实时传输功能，实现流媒体服务器集成到嵌入式系统。该文提出的主要研究内容和成果：

1、IETF(InternetEngineeringTaskForce)制定的实时传输协议(RTP)，它提供端到端的实时数据传输服务，其中的实时传输控制协议(RTCP)提供的控制功能可以有效的监控视频信息的传输。RTP协议非常适合流媒体在互联网上的传输，本文深入分析了RTP协议的特点、内容和在网络流媒体系统中的工作过程，实现了RTP协议在流媒体服务器上的应用，并对RTP时间戳的处理、RTP封装方法、RTCP自定义应用包扩展功能、多媒体同步机制和RTCP包为服务质量提供控制信息等方面提出了自己的设计方案。

2、该文比较了多种服务器模型的算法，针对本系统的要求，采用了基于并发多路复用技术和Linux网络编程技术的UDP并发服务器模型；还探讨了流媒体服务器的多媒体同步技术、服务质量相关参数计算和如何通过RTCP控制信息提供服务质量保证的方案。

3、该文介绍了课题所研究的基于嵌入式Linux的网络多媒体系统的工作原理、系统框图以及所选用的音视频压缩编码技术，详尽的阐述了嵌入式Linux系统开发平台的实现，在Flash设备上实现双文件系统的技术和使用网络文件系统来进行应用程序开发方案上都取得了突破性的成果。

4、对流媒体服务器软件模块化，从功能上划分为四大模块：主控制模块、RTP实现模块、RTCP实现模块以及RSVP实现模块，并通过Linux多线程编程和网络编程技术实现了各个模块的代码化，论文给出了各个模块实现的关键技术和算法流程。

5、该课题研究的最大的新点是流媒体服务器软件应用于嵌入式系统上，论文给出了流媒体服务器移植到嵌入式系统的详细过程，分析了此过程中的关键技术，并给出了解决的方案。最后，还测试了系统功能。测试结果表明，流媒体服务器能够在宿主机上成功编译，能在目标板上正常运行，能实时的传送音视频数据给客户端，并具备RSVP资源预留功能。

该文所研究的基于嵌入式Linux的网络流媒体系统可应用于手持PDA、3G移动终端、视频会议、视频点播以及远程视频监控，具有很强的实用价值，同时也对嵌入式系统研究和网络多媒体技术研究起到一定的参考作用。

4.学位论文 赵进 基于RTP协议族的流媒体传输系统的软件设计与实现 2004

多媒体数字技术的飞跃发展和个人计算机的日益普及,给人们的工作,学习,生活和娱乐带来了深远的影响.多媒体使计算机能够综合处理声音、文字、图像和视频,改变了人们使用计算机的方式.随着网络宽带化的发展趋势,人们希望有更直观、更丰富的新一代信息的表现形式.作为多媒体和网络的交叉学科流媒体(media streaming或streaming media)技术由此应运而生.传统的多媒体文件需要从服务器下载完才能播放,而流媒体的主要特点就是边下载边观看,以流的形式进行数字媒体的传送,从而使人们可以在线欣赏到连续不断的多媒体节目.该文主要研究关于流媒体传输的RTP协议族和实时流媒体数据的传输和播放系统的设计和实现.主分内容有:对RTP协议族进行深入研究,特别是H.263格式的视频数据的网络传输;分析了流媒体服务器的基本功能,工作流程以及不同模块的划分;分析了流媒体客户端接收的技术特点,介绍了播放器的结构和各模块的功能;对多播,RSVP等技术的应用进行研究.用GCC和VC++实现一个PC级的实时流媒体传输和播放系统,该系统包含有基于Linux平台的流媒体服务器和基于Windows平台的使用ActiveX技术的播放控件,能够完成实时流媒体数据的传输和播放以及简单的点播服务,并且能够提供多播,RSVP等手段提高服务性能.最后介绍了在Redhat8.0和Windows2000平台下对软件系统进行调试的情况,以及发现问题和解决的办法等,并进行了一定的性能分析.该技术可应用于视频会议、视频点播(Vod)系统和远程监控系统等领域.从事多媒体技术理论研究人员,视频软件的开发人员及有关技术人员可以从该文中获得有益的参考与启发.该论文工作得到广东省工业攻关项目[2002A1030405]的资助.

5.学位论文 余芳庭 VCR流媒体服务器的研究与设计 2008

移动通信和互联网是当今信息产业发展的两个热点，两者融合产生的移动互联网及其应用，为信息产业带来巨大商机。当今社会，人们不再满足于仅有的文本、声音、图像，而是希望得到声、文、图及视频流媒体信息，从而推动了移动流媒体的发展，提供流媒体服务的服务器系统也就成为人们关注的对象，正是基于此本文对这个课题做了深入研究。

由于硬件设施的有限性，本移动流媒体服务器的VCR功能通过一个简单的客户端在有线网络中得到了有效验证，为流媒体技术的发展与应用提供了可靠的理论依据和实践经验，为今后流媒体服务器的研究打下了坚实的基础，有广泛的应用前景。

本文从移动流媒体的发展背景着手，通过对其国内外的发展状况和流媒体服务器的发展现状的了解，首先明确了个人所需完成的任务；然后详细分析了实现流媒体服务器的四个重要协议：会话描述协议SDP、实时流协议RTSP、实时传输协议RTP、实时传输控制协议RTCP。通过对协议功能的了解，分析了协议具体内容。其中，RTSP协议主要分析了其OPTIONS、DESCRIBE、ETUP、PLAY、PAUSE、TEARDOWN这6个请求；RTP协议主要分析了RTP固定包头、RTP头部扩展、RTP协议子集、RTP连接复用；RTCP协议主要分析了RTCP包格式、RTCP发送间隔、保持会议成员的数目、源描述带宽的分配、发送和接收报告。

接着通过需求分析，考虑到实际限制，根据所定系统的具体需求和任务，设计出服务器系统的框架和结构，确定服务器关键技术：媒体内同步、媒体间同步、多线程的管理、流媒体服务器VCR功能的实现，然后根据所设计的服务器算法进行程序流程模块的实现。

最后，通过另外的测试系统对服务器进行调试与测试，进一步完善服务器的功能。根据测试系统所得数据，论文最后对系统的带宽利用率、服务响应时间、丢包率进行了统计，并且做出统计图形。

根据测试得出结论可知系统运行平稳，完满地实现了设计功能，证明整个设计方案是可行的，最终解决论文之初所困扰的一些问题。

6.学位论文 岑慧 RTP/RTCP协议在3G多媒体移动通信中的应用研究 2008

3G（The3rd—Generation Communication System），即第三代移动通信系统，她采用智能信号处理技术，除了能够支持更高速率的移动多媒（体）业务外，还能提供更高的频谱效率和服务质量，实现基于话音业务为主的多媒体数据通信，并将具有更强的多媒体业务服务能力。随着移动通信与信息家电、消费类电子产品的结合，第三代移动通信系统将实现宽带和综合多种业务需求，不仅能提供高质量的语音业务，而且能提供高速率的数据传输业务。

RTP/TRCP，前者为传输音频、视频、模拟数据等实时数据的传输协议，而后者是设计和RTP一起使用的进行流量控制和拥塞控制的服务控制协议。从上世纪七十年代出现并不断改进至今已形成国际化标准，目前众多大公司（如Netscape、Microsoft等）在多媒体实时通信中都支持RTP/RTCP协议。

本文首先详细介绍了RTP/RTCP协议，包括其协议组成、控制流程以及简单应用方面的内容。其次本文对多媒体网络通信技术作了详细阐述，引入了目前最流行的无线多媒体网络通信这个概念，并简单分析了目前一些基于无线多媒体通信技术的应用和研究。再次，本文引入3G无线通信系统概念，着重分析了3G环境下的无线多媒体通信技术的原理与实现过程，并与传统的多媒体网络通信作了详细的比较分析，从中提炼出3G多媒体通信的特点。最后，本文研究了一种基于RTP/RTCP的3G移动流媒体服务器关键技术，详细分析了该模型的设计原理与实现方式。本文还对相关领域的研究前景作了一些深入的展望。

7.期刊论文 季帮国.王翠荣.李焱.赵煜辉.JI Bang-guo.WANG Cui-rong.LI Yan.ZHAO Yu-hui 基于模糊控制理论的流媒体服务器设计 -计算机工程2008,34(14)

针对IP网络中传输多媒体信息存在较大延时和抖动以及带宽有限性等特征,结合RTP协议的特征设计与实现一种基于模糊控制理论的流媒体服务器质量控制系统.该系统能根据网络状况自适应地改变发送策略,结合编码后数据的特点,丢弃一些不必要的数据,这样不仅能节约一定的带宽,也保证了关键数据的传输和良好的服务质量.

8.学位论文 高磊 基于JAVA的移动流媒体增值业务研究 2006

计算机技术、移动通信技术和互联网技术的飞速发展，为移动互联网增值业务创造了广阔的发展空间。移动通信和互联网是当今信息产业发展的两个热点，两者融合产生的移动互联网及其应用，为信息产业带来巨大商机，同时，J2ME平台植入手机的举动使得手机的功能拓展成为了一个随身携带的微型计算机。使得手机用户随时随地的享受互联网的服务。

本文简要介绍了移动流媒体平台搭建，及在此平台上运行的移动终端、网关及源服务器端三个方面的功能。针对目前主流手机只支持一、两种多媒体格式甚至不支持流媒体格式的缺陷，通过研究实时流协议、MPEG-4压缩编码和ASF文件结构，设计开发了基于MPEG-4的ASF流媒体解码播放的应用。该应用主要功能是实时的接收流媒体数据，并能和流媒体服务器实时交互，对下载的流媒体数据进行分析，重新组合数据包使其按流文件原有的顺序排列，同时对排好的数据包进行拆分、重组，根据ASF文件格式和MPEG-4压缩特点，对现有数据流拆分、解码，最后将图像绘制到屏幕上。平台主要功能有基于WAP的浏览，为手机用户提供应用索引信息服务；实现应用程序的OTA下载以及提供流媒体内容服务并实时的按客户端要求传输数据。

9.学位论文 李鑫 流媒体服务器在网络摄像机中的嵌入式实现 2008

计算机技术和高速宽带网络的发展，为多媒体技术的应用提供了广阔的空间，网络多媒体是电子技术、计算机技术、通信技术相互结合的产物。稳定可靠的网络多媒体系统需要解决音视频编解码、网络传输、远端设备控制等技术问题，同时还要考虑系统的灵活性和精简性，将嵌入式系统和流媒体技术相结合，构建一个灵活高效、扩展性强、可靠性高的系统是当前网络多媒体技术的发展趋势。嵌入式的视频会议、视频点播、远程教育、无人监控、数字家庭等分布式多媒体应用逐渐走入人们的生活，给人们的工作和生活带来了极大的便利和无穷的乐趣。

本文以一个实际的嵌入式视频监控系统一---网络摄像机开发为背景，对嵌入式网络摄像机实现的关键技术，系统的软硬件设计及实现方法都进行了研究。最后，重点结合流媒体服务器的原理，实现了其在网络摄像机的移植，并给出了系统测试步骤及结果。

通过研究国际上流行的嵌入式流媒体通信系统结构框架，结合本系统设计时所要求的性能指标，在硬件平台的选择上，选用了美国Micronas公司推出的基于MIPS架构的CYPHER7108处理芯片。7108是一块单片型的、实时流式音视频压缩的SOC芯片，具有较强的实时多格式编码能力和多媒体应用上的诸多特点。在软件平台上我们采用源代码开放完全免费的Linux2.6操作系统。

文中对IETF(Internet Engineering Task Force)制定的多种流媒体传输协议做了深入研究，其中RTP协议提供端到端的实时数据传输服务，RTCP协议提供对RTP的控制功能，可以有效的监控视频信息的传输，RTSP协议定义了如何高效的实现一对多的多媒体数据传输等等。在对这些协议充分了解的基础上，才能更加灵活的进行流媒体服务器软件的移植和修改工作。分析了Spook的部分源代码，并成功移植到目标机中运行，实现了视频数据的低误码、低延迟、无明显抖动的传输。

本课题最主要的创新之处是将流媒体服务器软件Spook应用于嵌入式系统上，Spook是一个能捕捉实时音视频流并转换成流格式、通过IP网络传输的Linux服务器应用程序。文中给出了流媒体服务器移植到嵌入式系统的详细过程，分析了此过程中的关键技术，并给出了解决方案。最后，还测试了系统功能。测试结果表明，流媒体服务器能够在宿主机上成功编译，能在目标板上正常运行，能实时的传送视频数据给客户端。

基于本系统，授权的用户可以通过浏览器直接访问服务器，实现了实时视频浏览以及远程控制等功能。基于流媒体服务器的网络摄像机系统体系结构清晰，可方便地进行移植和扩展，利用本系统实现的视频监控，具有小型化，低功耗，稳定可靠等特点。

10.学位论文 梁毅宏 基于达芬奇系统的嵌入式实时流媒体服务器的研究与实现 2008

数字多媒体技术和宽带网络的飞速发展,使得网络资讯越来越丰富,各种音视频业务需求的急速增长和数字信号处理技术及相关器件的不断推陈出新,促使了嵌入式流媒体技术的诞生与发展。以流媒体应用为核心的系统要获得实际的推广应用,除了要解决音视频编码、网络传输等技术问题外,同时还要考虑系统的成本、功耗、便携性等,因此,结合流媒体技术和嵌入式系统的优点,研制开发性价比高的嵌入式流媒体系统成为了产业界和科研院所密切关注的课题。

本研究课题根据流媒体应用中大量的实时视频传输需求提出,设计并实现了一个基于达芬奇系统,具有实时视频采集、编码、流化处理及网络传输等完整功能的嵌入式实时流媒体服务器。本文首先简单介绍了达芬奇系统的软硬件资源,给出了流媒体服务器的系统框图及其Linux线程模型,然后重点阐述了服务器功能的设计与实现,并对相关的关键技术进行了深入探讨与研究。本系统的实时视频信号采集基于V4L2视频驱动,同时利用FrameBuffer设备驱动实现了视频预览功能；基于TI编解码引擎(Codec Engine)软件框架和双核处理器通信机制DSP/BIOS Link,通过H.264编码器实现对视频信号的压缩编码；H.264视频流封装复用为MPEG-2传输流后,作为实时流式视频源；实时流式数据的传输采用标准的RTP协议,统计、控制等反馈信息的传输机制依据RTCP协议构建,服务器与客户间的信令交互基于RTSP协议实现。最后,通过系统综合测试对系统功能进行了验证。

本研究实现了一个完整规范的嵌入式流媒体服务器,所形成的技术可应用于视频监控、在线直播等领域,具有很强的实用价值,同时对其它嵌入式网络应用系统的技术研究和开发也会起到很好的参考作用。