基于流媒体技术的网络视频监控系统的研究与实现

摘要

随着网络技术和多媒体技术的不断发展，网络视频监控系统的应用越来越广泛，但其视频数据流在网络传输过程中的实时性和传输质量得不到很好的保证。流媒体技术的兴起较好的解决了这个问题，将其应用于网络视频监控系统中显然是数字监控领域的巨大突破。但目前的很多监控系统都跟具体的硬件相关，所以有必要开发一种具有通用性的基于流媒体技术的网络视频监控系统。

本文通过采用流媒体技术及其开发工具 DirectShow 完成了一个用 USB 摄像头作为视频采集设备，完全用软件的方法实现的网络视频监控系统。DirectShow 是微软公司开发的一套基于 Windows 平台的软件开发包，它为 Windows 平台上处理各种格式的媒体文件播放、音/视频采集等高性能要求的多媒体应用提供了完整的解决方案。本文的研究内容和成果如下：

1、研究了流媒体技术的原理及流媒体传输协议 RTP/RTCP 的实现机制和应用，以及视音频数据的压缩、解码标准；

2、选用了 Visual C++6.0 作为开发平台，运用微软的 DirectShow 技术进行了视音频的采集、播放、编码、解码、回放等开发编程；

3、对流媒体网络视频监控系统的总体结构和各功能模块进行了设计，并编程实现了系统服务器端和客户端软件功能，对网络传输模块的实现主要是建立了网络发送和网络接收两个过滤器，实现了视音频由发送端过滤器发送到网络，接收端可以顺利地回放视频图像和声音；

最后对完成的系统进行了相关的调试和实验，验证了系统方案的有效性，并对系统的进一步完善工作进行了展望。本系统软件原型具有较好的规范性和重用性，不仅可用于远程监控系统，还可用于多媒体远程教学、网络视频会议、视频聊天等软件系统，具有较好的应用前景

关键词：流媒体，视频监控系统，DirectShow，RTP/RTCP，Filter

RESEARCH AND REALIZATION OF NETWORK VIDEO MONITOR SYSTEM BASED ON STREAMING MEDIA TECHNOLOGY

ABSTRACT

With the development of network and multimedia technology, the application of the network video monitor system is more and more extensive. However, the network video monitor system often cannot adapt well to the high real-time requirement, and low-level efficiency of the video streaming transmission cannot satisfy the system requirement. The development of streaming media technology gives a chance to better solve this problem. Obviously, it is a great breakthrough in digital monitoring field that streaming media technology used for network video monitor system.But many monitor system relevant to concrete hardware, so it is necessary to develop a kind of commonality network video monitor system based on streaming media technoloy.

This paper completed a network monitor system using the USB camera as the video collecting equipment, which realized with pure software by adopting the streaming media technique and DirectShow.DirectShow is a software development kit, which Microsoft Corporation develops based on the Windows platform software.It provides a whole solution for the Windows platform to process the preview and the audio/video collection of the media document.The study contents and results in the paper are as follows:

1. Researched the principle of streaming media technology, the implementation mechanism and application of transport protocol RTP/RTCP, and the video and audio data compression, decoding standards;

2. Selected VC++6.0 and DirectShow tech as developing platform to make gathering, playing, encoding, decoding, replaying program;

3. Designed the overall structure of the streaming media network monitor system and each functional module, and realized the software function of both the server-end and client-end by programming. The network transmission module has mainly realized by establishing the network send filter and network receive filter, which can send the audio/video datas to the network and the receiver can playback the video images and sound smoothly.

At last, it has completed the relevant debugging and proved the effectiveness of the system , and the further improve work has been presented.With the quality of service and reusability as the design goals, the prototype can be applied not only in remote supervisory system but also in multimedia distanceeducation system, network video conference and so on.

Key Words：streaming media, video monitor system, DirectShow, RTP/RTCP, Filter

独创性声明

本人郑重声明：所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，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和撰写的研究成果，也不包含为获得华东交通大学或其他教育机构的学位或证书所使用过的材料。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了谢意。

本人签名_____日期_____

关于论文使用授权的说明

本人完全了解华东交通大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅。学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

保密的论文在解密后遵守此规定，本论文无保密内容。

本人签名_____导师签名_____日期_____

第一章　绪论

1.1　视频监控系统的发展

视频监控系统系统是安全防范系统的组成部分，它是一种防范能力较强的综合系统。随着社会的发展和技术的进步，人们对方便、快捷的视频监控的要求越来越高，涉及的领域也越来越广[1]。视频监控系统从最早的局限于金融业的应用，逐渐遍布各行各业，如：大型公共设施的安防、电力系统、交通领域、社区物业管理等。近年来，随着计算机、网络以及图像处理、传输技术的飞速发展，视频监控技术也取得了长足的发展。

视频监控系统发展大概经历了三个发展阶段：在 90 年代初以前，主要是以模拟设备为主的闭路电视监控系统，称为第一代视频监控系统，即模拟视频监控系统；90 年代中期，随着计算机处理能力的提高和视频技术的发展，人们利用计算机的高速数据处理能力进行视频的采集和处理，从而大大提高了图像质量，增强了视频监控的功能。这种基于多媒体计算机的系统称为第二代视频监控系统，即模拟与数字混合监控系统；90 年代末，随着网络带宽、计算机处理能力和存储容量的迅速提高，以及各种实用视频信息处理技术的出现，视频监控进入了全数字化的网络时代，称为第三代视频监控系统，即全数字视频监控系统或网络数字视频监控系统[2]。第三代视频监控系统以网络为依托，以数字视频的压缩、传输、存储和播放为核心，以智能实用的图像分析为特色，引发了视频监控行业的技术革命。

1.1.1　模拟视频监控系统

模拟视频监控系统一般是以摄像机、控制设备(云台、解码器)和中心设备(视频矩阵、画面分割器、磁带录像机)为核心，采用模拟方式传输信号，通过手动方式对各个被控站点的情况进行切换显示[2]。主要应用于小范围内的控制，监控图像一般只能在监控中心查看。其基本结构如图 1-1 所示：

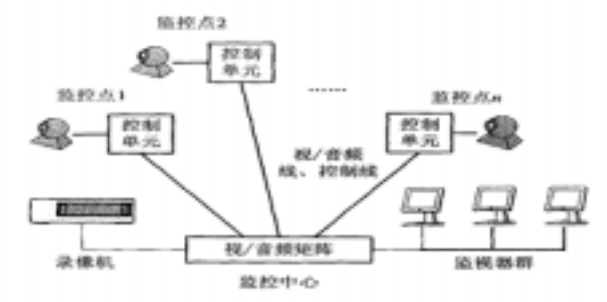

图 1-1　模拟视频监控系统示意图

Fig.1-1　Analog Video Monitor System Diagram

1.1.2　数字视频监控系统

数字视频监控系统一般是在监控现场，设置若干台摄像机、各种检测、报警探头与数据设备，它们通过各自的传输线路，汇聚到控制单元上，进行信号编码压缩，再以数字的方式经过传输系统到监控中心，监控中心通过信号解码进行实时监控。其核心就是将模拟的视频、音频和控制信号转化成数字信号，形成实现某种功能的数据流，有效地传输和控制[2]。由于需要建设大规模的传输系统，成本高，因此主要适用于近距离监控（如：大楼监控等）。其基本结构如图 1-2 所示：

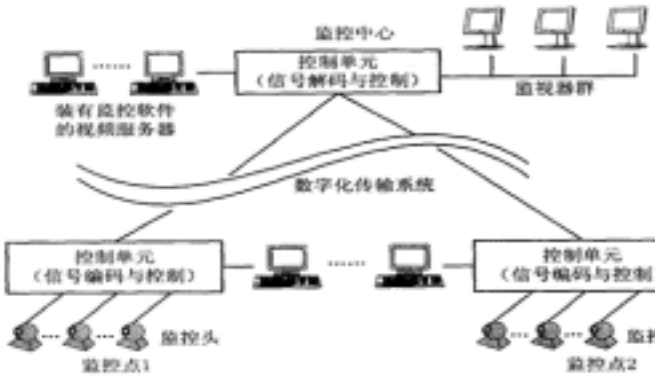

图 1-2　数字视频监控系统示意图

Fig.1-2　Digital Video Monitor System Diagram

1.1.3　IP 网络视频监控系统

IP 网络视频监控系统是在传统视频监控的基础上，通过以太局域网、PSTN 电话网、Internet 等网络完成数据传输，借助于数字光纤、数字微波、无线通信、ATM、DDN、ISDN、卫星通信等多种远程数字传输媒介，使用点对点、广播、组播等多种传输方式，使监控的图像在网络上进行传输，实现远程监控，完全不受空间的局限[3]。其基本结构如图 1-3 所示：

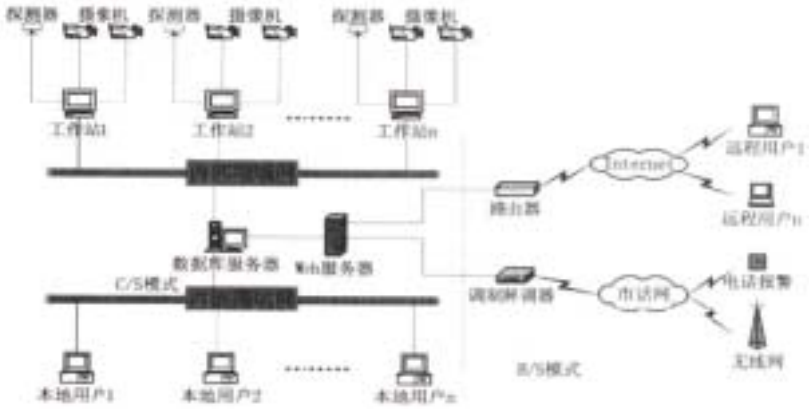

图 1-3　网络视频监控系统示意图

Fig.1-3　NetWork Video Monitor System Diagram

1.1.4　基于移动通信的视频监控系统

基于移动通信的视频监控系统是指运用现代移动通信技术（如：GPRS、CDMA），采用无线 Modem 将采集、编码压缩的视频图像通过通信网络传送到移动终端（手机、移动 PC 等），从而实现无线远程实时监控。其基本结构如图 1-4 所示[4]：

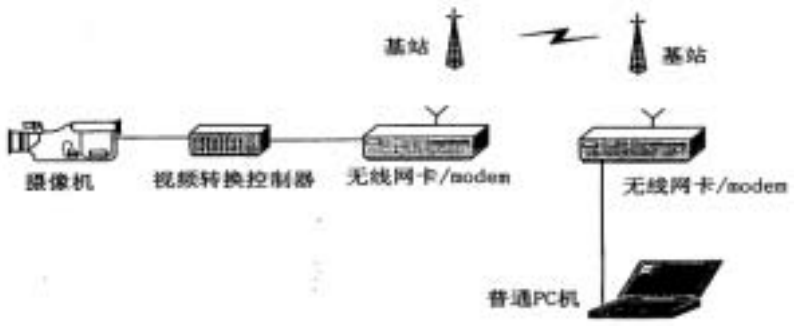

图 1-4　基于移动通信的视频监控系统示意图

Fig.1-4　Video Monitor System Based on mobile communication Diagram

1.1.5　智能视频监控系统

智能视频监控以数字化、网络化视频监控为基础，但又有别于一般的网络化视频监控，是一种更高端的视频监控应用，它区别于传统意义上的监控系统在于其智能性。智能视频监控系统以计算机视觉分析技术为主，能够识别不同的物体，发现监控画面中的异常情况，并能够以最快和最佳的方式发出警报和提供有用信息，从而能够更加有效的协助安全人员处理危机，并最大限度的降低误报和漏报现象。简单而言就是，不仅用摄像机代替人眼，而且用计算机代替人、协助人，来完成监视或控制任务，从而减轻人的负担。视觉监控以其强大的功能、灵活的使用方式、广泛的应用前景及巨大的潜在经济价值，从而激发了世界上广大科研工作者及相关商家的浓厚兴趣[5]。

在技术的不断前进发展中，视频监控系统必然沿着数字化、网络化和智能化的方向继续迅速发展。系统功能综合化、视频数字化、监控网络化、系统集成开放及标准化是视频监控系统发展的必然趋势。

1.2　流媒体技术概述

1.2.1　流媒体的发展现状

流媒体（Streaming Media）技术是从互联网上发展起来的一种多媒体应用技术。所谓流媒体，是指在 Internet/Intranet 中使用流式传输技术的连续时基媒体，如：音频、视频或多媒体文件[6]。流媒体不同于传统的多媒体，它的主要特点就是运用可变带宽技术，以流的形式进行数字媒体的传送。流媒体在播放前并不下载整个文件，数据流随时传送随时播放，只是在开始时有一些延迟，这样可以使人们在线欣赏到连续不断的

高品质的音频和视频节目，避免了用户必须等待整个文件全部从 Internet 上下载才能观看的缺点。流媒体技术起源于窄带互联网时期，当时互联网的通信网络还比较落后，用户仅能以非常低的连接速率，通过网络获得静态的图文。随着技术的发展，流媒体的定义已不再是单一的流式传输技术，它衍生出了适合流式传输的网络通信技术、多媒体数据采集技术、多媒体数据压缩技术、多媒体数据存储技术等更多的基础技术。现在的流媒体已经发展成为一个产业。互联网的发展更是决定了流媒体市场的广阔前景。商业网站利用流媒体播放新闻、音乐直播和点播，企业和一些机构采用点播和流媒体进行员工培训、信息发布、公司介绍等，可提高效率，节约开支。这样巨大的市场吸引越来越多的企业参与竞争。一个全球化的媒体市场和竞争格局已经初步形成，如何在这个市场取得份额，成为当前诸多企业关注的焦点。国内外厂商的纷纷涌入，将使我国的流媒体市场更加活跃，更加成熟，当然竞争也将更加激烈[7]。

目前，Internet 中最通用的流媒体系统包括 RealNetworks，Microsoft Window Media Player，Apple QuickTime 等，RealNetworks，Window Media Player 流媒体播放器甚至已成为 PC 标准配置[6]。

1.2.2　流媒体技术的应用

流媒体技术改变了传统互联网的呆板形象，丰富了互联网的功能，使之成为一种有强大吸引力的新媒体。流媒体技术正逐渐应用到社会的各行各业中，现在流媒体技术广泛应用在多媒体新闻发布、网络直播、电子商务、视频点播、视频会议、视频监控、远程教育、远程医疗等网络信息服务的各个方面，为提供人们的生活品质做出了巨大的贡献[8]。下面是几个典型的应用例子[9]：

在线直播(Live Video)：娱乐是流媒体的重要应用场合。用摄像机或其它装置获得视频信号后，就可以通过站点进行基于 Internet 的现场直播，或者保存为流媒体格式的文件，以供按需播放。需要在一台较高配置的 PC 机或服务器上安装上普通视频采集卡和声卡，然后通过视频采集卡输入视频和通过声卡输入声音信号就可以用实时编码工具来进行直播或录制成流媒体文件。在这种应用中可加入一定的计费手段，从而能够提供有偿多媒体内容服务。

远程教育(Remote Learning)：远程教学将为更多的人提供接受教育的机会。教学者事先在 Internet 上发出通知，听众在讲座开始前访问某个 URL 地址，当讲座开始时，听众可以看到演讲者的演讲画面并听到他的声音。整个讲座也可以流媒体文件的形式记录下来，用于以后按需播放。教学者事先把流媒体文件传给远程教学服务器，当听众需要听讲座时，同样访问相应的 URL 地址，请求获取服务器中的流媒体内容。媒体数据通过流式传输下载到用户的浏览器高速缓存中，由媒体播放器实时回放。

视频会议(Video Conference)：视频会议和远程教学有很多类似之处，但它对实时性的要求更高。在一个视频会议中，各个会议点用音视频采集设备得到多媒体内容信息，

经过数字化后用某种压缩方法进行压缩。压缩数据可以通过网络直接在各个会议点之间组播，或传到多点处理器经过合成或转换后再向各与会点组播。但不管采用哪种方式，都需要保证以尽量小的时延在各个点进行回放，这正是流媒体技术发挥作用的地方。

远程监控((Remote Surveillance)：流媒体技术也可以应用于远程监控。近年来，一些公司已经开发了一类所谓 web camera 的产品，基本上它就是把图像采集、压缩、web server 和 Internet 接入集成在一个设备中，使得远程监控可以通过 Internet 完成，这与传统的远程监控系统相比，在可控制性、监控距离、架设方便性等方面都有很强的竞争力。

1.2.3　流媒体技术的发展趋势

正如几年前的 IP 网络和 Web 技术，流媒体应用正处于高速持续增长时期。流媒体市场将呈现巨大的收入潜能，未来几年里，流媒体将会呈现出以下几个发展趋势[10]：

（1） MPEG-4 多媒体解码标准的发展势头会越来越猛；
（2） 通过电视观看流媒体；
（3） 拓展流媒体在通信领域内应用；
（4） 流媒体标准之争日渐白热化；
（5） 利用内容分发网络（CDN）来传输流媒体。

1.3　视频监控和流媒体技术结合

网络视频监控系统在其发展过程中遇到了许多的技术难题，最主要的就是视频数据流在网络传输过程中的实时性和传输质量不能得到很好的保证，这主要是因为在现阶段存在着网络带宽不足、数据的传输效率低下、传输过程中丢包率较高等因素。

如何在现有的网络带宽条件上实现远程监控，并且在价格与图像质量上求得最佳的平衡，成了目前迫切需要解决的问题。流媒体技术的兴起和广泛应用，为这个问题的解决提供了一个新的思路，它根据自身特性将连续媒体流压缩封装后按照一定的时序要求发给接收端，保证了接收端可以边接收数据边实时播放，从而提高了系统的实时性；同时，它的流量控制和网络拥塞控制机制也提高了网络传输效率的稳定性，进而保证了接收端的播放质量。而网络视频监控系统则以数字视频处理技术为核心，综合了网络技术、多媒体技术、图像压缩技术于一体，有效的克服了模拟监控的缺点；提高了数据传输的实时性、系统的并行处理能力和系统存储容量，并支持多种有线、无线传输介质，集视频切换、智能控制、远程传输等功能于一身。流媒体技术应用于网络视频监控系统显然是数字监控领域的巨大突破，能有效克服其他传输方式存在的局限性。

无论从技术角度还是市场角度来说，流媒体技术应用于网络视频监控系统都有其他技术无可比拟的优越性。特别对窄带远程监控尤为显著，用户不必等远端监控信息传输完毕即可实时、连续播放，有效克服其它方式播放的等待问题，且实时性较好。虽然，

该方式可能影响视频图像质量，但是，随着数字视频技术的发展、图像压缩质量的提高以及网络带宽的增大，流媒体技术的应用完全可以满足视频图像监控和记录的需求。因此，流媒体技术和视频监控技术的融合应用是它们发展的必然趋势，它们的综合应用在将来也必然越来越广泛。

1.4　论文研究内容及章节安排

1.4.1　课题相关领域的现状及发展趋势

在国内外市场上，主要推出了数字控制的模拟视频监控和数字视频监控两类产品。前者技术发展已经非常成熟、性能稳定，在实际工程中得到广泛应用，特别是在大、中型视频监控工程中的应用尤为广泛；后者是新近崛起的以计算机技术及图像视频压缩为核心的新型视频监控系统，该系统解决了模拟系统部分弊端，但仍需进一步完善和发展。视频监控应用具有巨大的市场前景，所以视频监控系统的研究和开发受到了学术界、产业界和使用部门的高度重视，研究成果和开发的产品层出不穷。

流媒体技术发展也非常迅猛，目前 Internet 上每周约有 45000 小时的广播节目，58 个美国电视台提供 Web 广播，34 个电视台提供点播服务。有近半数的跨国企业公司在内部使用流媒体实现 Web 广播。来自国际权威机构的调查，2000 年在网上访问流媒体的人数增加 65%,西方网络发达国家访问流媒体的人数已达到 1.1 亿人,约占网民的 1/3，在亚洲也迅速增加到 2500 万人，约占网民的 1/9。与用户增长相呼应，去年 Internet 上视频流媒体技术应用增长幅度达 251%。在美国已经有三分之一的 Internet 用户使用流媒体业务，有近百家的 ISP 提供流媒体业务，在 2000 年已有 40 亿左右的市场；在欧洲、亚洲等地这样的用户也有一定的规模[11]。

在欧美等发达国家，视频监控的发展比较快，应用也十分广泛。但是他们的产品价格较高、带宽要求较大、培训和维护也很不方便，而且从国外引进的产品本地化程度较差，在功能、接口、使用环境等方面均或多或少地存在一些与我国国情及现场需要不太吻合的地方。我国视频监控行业最初是由闭路电视监控(CCTV)逐渐发展起来的，已近二十年的历史，从简单模拟视频监控到现在的纯数字化网络视频监控，监控系统在国内目前应用已经相当广泛。国内长城集团、网通电子商务有限公司、北京微电子技术有限责任公司等也自主开发了国产的数字视频监控系统。国内高校也在监控系统的研发方而发挥着重要的作用。总的说来，流媒体服务以及流媒体技术应用在视频监控系统中在我国处于起步阶段，也有一些关键技术性问题还待于解决和完善。

1.4.2　课题研究的目的及意义

流媒体的应用广泛，如何充分应用这种优秀的新技术于传统的监控系统，让监控系统的应用范围更加广阔，并且满足网络化的需要，在现有的网络带宽条件上实现远程监

控，并且在价格与图像质量上求得最佳的平衡，是本论文着眼研究的课题。通过对本课题的关键技术的研究和应用，以达到能够设计并实现一种具有通用性的基于流媒体技术的网络视频监控系统，并且该监控系统不跟具体的硬件相关，完全用软件的方式来实现，系统采用模块化结构设计，易于更新和扩展，也可以有效降低系统集成的困难和成本，在工程应用中具有一定的实用价值。因此，本系统适应了监控系统数字化、网络化、集成化的发展趋势，具有良好的发展前景。

1.4.3　课题的主要内容及章节安排

本文的研究目标是：探索如何利用流媒体技术，实现基于流媒体技术的网络视频监控系统。该方案将使用 Visual C++ 6.0 为开发平台，服务器端软件采用 DirectShow 技术实现对 UBS 摄像头的视频采集、预览、MPEG-4 压缩、录像、回放等功能；客户端软件采用 DirectShow 技术实现对视频数据的解压，预览、录像、回放等功能。网络传输模块的重点是对网络发送过滤器和网络接收过滤器的设计和实现。采用 C/S 结构，完整构架了一个基于流媒体技术的小型网络视频监控系统。根据本课题的研究思路和设计模块划分，本论文共分为 7 章，各章安排如下：

第一章，绪论。阐述了课题研究背景，介绍了视频监控系统和流媒体技术的现状及发展、论文研究的内容和完成的工作。

第二章，介绍了流媒体相关技术，包括流媒体传输技术、流媒体原理，RTP/RTCP 协议等。

第三章，深入剖析了 DirectShow 技术，从 DirectShow 的结构，工作原理及内部实现机制等方面进行了分析和阐述。

第四章，详细阐述了系统的设计。首先，介绍了系统的整体设计思想，然后对服务器端和客户端各模块的设计，其后阐述了开发网络发送和接收过滤器的过程。

第五章，详细介绍了各种技术实现系统功能的具体过程，包括服务器端和客户端软件的各功能模块的实现，及网络模块的网络发送和接收过滤器的实现过程。

第六章，展示了系统的调试、运行与测试结果，测试主要在局域网中进行。

第七章，结束语。对本文的工作进行总结和展望。

第二章　流媒体相关技术介绍

2.1　流式传输基础

流媒体是指在 Internet/Intranet 中使用流式传输技术的连续时基媒体，如音频、视频或多媒体文件。流式媒体在播放前并不下载整个文件，只将开始部分内容存入内存，流式媒体的数据流随时传送随时播放，只是在开始时有一些延迟。流媒体实现的关键技术就是流式传输[6]。流式传输定义很广泛，现在主要指通过网络传送媒体（如视频、音频）的技术总称。实现流式传输有两种方法：实时流式(Real-Time Streaming)传输和顺序流式(Progressive Streaming)传输。一般说来，如视频为实时广播，或使用流式传输媒体服务器，或应用如 RTSP 的实时协议，即为实时流式传输。如使用 HTTP 服务器，文件即通过顺序流发送，这种传输方式就称为顺序流式传输。采用哪种传输方法依赖于用户的具体需求，当然，流式文件也支持播放前完全下载到硬盘后再播放[12]。

2.1.1　顺序流式传输

顺序流式传输是顺序下载，在下载文件的同时用户可以观看在线媒体。在给定时刻，用户只能观看已下载的那部分，而不能跳到还未下载的其他部分。顺序流式传输不像实时流式传输在传输期间根据用户连接的速度做调整。由于标准的 HTTP 服务器可发送这种形式的文件，也不需要其他特殊协议，它经常被称作 HTTP 流式传输[13]。顺序流式传输比较适合高质量的短片段，如片头、片尾和广告，由于该文件在播放前观看部分是无损下载的，所以这种方法能够保证电影播放的最终质量。但这就意味着用户在观看前，必须经历延迟，对较慢的连接尤其如此。

对通过调制解调器发布短片段，顺序流式传输显得很实用，它允许用比调制解调器更高的数据速率创建视频片段。尽管有延迟，毕竟可发布较高质量的视频片段。

顺序流式文件是放在标准 HTTP 或 FTP 服务器上，易于管理，基本上与防火墙无关。顺序流式传输小适合长片段和有随机访问要求的视频，如：讲座、演说与演示，它也不支持现场广播。严格说来，顺序流式传输其实是一种点播技术。

2.1.2　实时流式传输

实时流式传输是指保证媒体信号带宽与网络连接配匹，使媒体可被实时观看到。实时流与 HTTP 流式传输不同，它需要专用的流媒体服务器与传输协议。实时流式传输总是实时传送，特别适合现场事件，也支持随机访问，用户可快进或后退以观看前面或后面的内容。理论上，实时流一经播放就可不停止，但实际上，可能发生周期暂停[13]。

实时流式传输必须配匹连接带宽，这意味着在以调制解调器速度连接时图像质量较

差。而且，由于出错丢失的信息被忽略掉，网络拥挤或出现问题时，视频质量很差。如欲保证视频质量，顺序流式传输也许更好。实时流式传输需要特定服务器，如 QuickTime Streaming Server ，Real Server 与 Windows Media Server。这些服务器允许用户对媒体发送进行更多级别的控制，因而系统设置、管理比标准 HTTP 服务器更复杂。实时流式传输还需要特殊网络协议，如：RTSP（Real-Time Streaming Protocol）或 MMS（Microsoft Media Server）。这些协议在防火墙存在的情况下有时会出现问题，导致用户不能看到一些地点的实时内容。

2.2　流媒体技术原理

首先，流式传输的实现需要缓存。这是因为 Internet 是以包传输为基础进行断续的异步传输。数据在传输中它们要被分解为许多包，由于网络是动态变化的，各个包的选择路由可能不尽相同，故到达客户端的时间延迟也就不等。为此，要使用缓存系统来弥补延迟和抖动的影响，并保证数据包的顺序正确，从而使媒体数据能连续输出，而不会因为网络暂时拥塞使播放出现停顿。通常高速缓存所需容量并不大，因为高速缓存使用环形链表结构来存储数据，通过丢弃已经播放的内容，流可以重新利用空出的高速缓存空间来缓存后续尚未播放的内容。

再次，流式传输的实现需要合适的传输协议。WWW 技术是以 HTTP 为基础的，而 HTTP 又建立在 TCP 基础之上。由于 TCP 需要较多的开销，故不太适合传输实时数据。在流式传输的实现方案中，一般采用 HTTP/TCP 来传输控制信息，而用 RTP/ UDP 来传输实时视音频数据。

流式传输的过程一般是这样的：用户选择某一流媒体服务后，Web 浏览器与 Web 服务器之间使用 HTTP/TCP 交换控制信息，以便把需要传输的实时数据从原始信息中检索出来；然后客户机上的 Web 浏览器启动流媒体播放程序，使用 HTTP 从 Web 服务器检索相关参数对流媒体播放程序初始化。这些参数可能包括目录信息、A/ V 数据的编码类型或与 A/ V 检索相关的服务器地址。流式传输的过程如图 2-1 所示[14]。

图 2-1　流式传输基本原理

Fig.2-1　Fundamental principle of stream transmission

流媒体播放程序及 A/V 服务器之间运行实时流控制协议(RTSP)，以交换 A/V 传输所需的控制信息。与 CD 播放机或录像机所提供的功能相似，RTSP 提供了操纵播放、快进、快倒、暂停及录制等命令的方法。A/V 服务器使用 RTP/ UDP 协议将 A/V 数据传输给 A/V 客户程序(一般可认为客户程序等同于 Helper 程序)，一旦 A/V 数据抵达客户端，流媒体播放程序即可播放输出。

2.3　流媒体传输协议

2.3.1　RTP/RTCP 协议介绍

1996 年 IETF 的视频/音频工作组制订了 RTP/RTCP 协议，专门用于支持网络实时传输服务，提供数据实时传输的标准[15]。RTP/RTCP 是端对端的协议。在协议层次中，RTP/RTCP 虽然位于应用层，但多数应用还要在 RTP/RTCP 之上建立更符合应用要求的协议。在网络传输过程中，RTP/RTCP 不处理连接建立工作，其下层网络既可以是有连接的，也可以是无连接的。如果传输流媒体，RTP/RTCP 一般基于组播协议；它并不要求特定的地址格式，而仅要求下层提供相对好的分帧、合帧及分段、合段的工作；该协议不提供可靠性保障，作为应用层的一部分，该协议并不是操作系统的内容。RFC1889 中定义 RTP/RTCP 协议族由两个相关的协议构成：

1)实时传输协议 RTP 用来传输具有实时特点的数据。

2)实时传输控制协议 RTCP 用于统计、管理和控制 RTP 传输。RTP 和 RTCP 协同工作完成任务。

目前最新的版本是 2003 年公布的 RFC3550。RTP 是一个轻量级的网络协议，它没有对下一层的传输协议做硬性规定，在 IP 网络上，我们一般使用 UDP 协议作为下一层协议传输数据，图 2-2 显示了它在 TCP/IP 协议栈中的位置。

图 2-2　RTP 在 TCP/IP 协议栈中的位置

Fig.2-2　The position of RTP in TCP/IP protocol stack

2.3.2　RTP/RTCP 协议工作原理

RTP 和 RTCP 协作操作，完成数据的实时传输，为了控制会话，它还可能和其他的网络协议共同工作，比如在多媒体应用中的实时流协议 RTSP（Real-Time Streaming Protocol），如图 2-3 所示。

图 2-3　RTP/RTCP 协作工作图

Fig.2-3　Cooperation work chart of RTP/RTCP

在整个工作过程中它们各自完成自己的任务：

1）RTP 协议用来传输实时数据。在 RTP 包中包括数据包序列号、时间戳等信息。

2）RTCP 协议用来监控数据传输质量和统计参加在当前会话中的成员的一些信息。这些统计信息可能对那些“松散控制”的应用是足够的，比如像免费的网络直播等对成员没有具体限制的应用。但对于一些特殊要求，需要使用其他的非 RTP 手段来加以保证。

3）会话控制协议，比如 RTSP，用来传输和会话控制有关的数据，比如用户的管理、用户数据的设置和媒体的操作。

2.3.3　RTP/RTCP 传输流程

RTP 与 RTCP 传输视频流的工作流程如下：在视频服务器端，流媒体视频流按照 RTP 数据传输协议的数据包格式被装入 RTP 数据包的数据载荷段，并配置 RTP 数据包头部的时间戳、同步信息、顺序号等参数，即数据包被“流”化了；同时周期性地接收 RTCP 包，利用这些信息动态地改变自身参数设置。客户端收到数据包后先分析 RTP 包头，判断版本、载荷类型等信息的有效性，更新缓冲区的 RTP 信息，如收到的字节数、视频帧数、包数、顺序号等信息；按照 RTP 时间戳和顺序号等进行信源同步，整理 RTP 包顺序，重构视频帧；最后根据载荷类型标识进行解码，将数据放入缓存供解码器解码输出，同时客户端根据 RTP 包中的信息周期性回送包含服务质量(QoS)反馈控制信息的 RTCP 包(接收报告包 RR)到服务器以检测发送端和接收端数据的一致性。图 2-4 是基于 RTP/UDP/IP 的传输系统结构框图，在 UDP 协议的基础上，使用 RTP 协议传输视频流，使用 RTCP 协议进行传输控制。

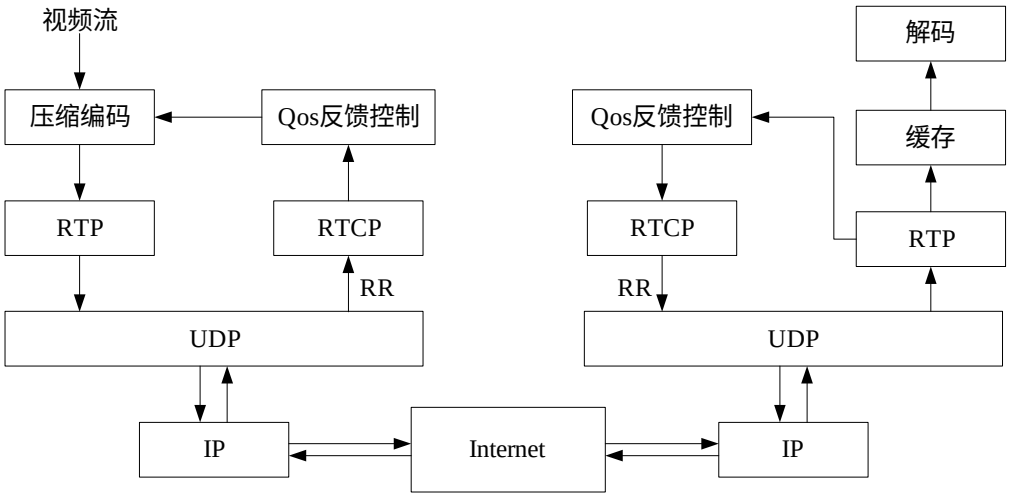

图 2-4　基于 RTP/UDP/IP 的传输系统结构框图

Fig.2-4　Structure chart of transmission system based on RTP/UDP/IP

2.3.3　RTP 协议

RTP 是英文 Real-Time Transport Protocol 的缩写，中文名称是实时传输协议，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。RTP 定义在 RFC1889 中[15]。其报文结构包含广泛用于多媒体的若干个域，包括声音点播、影视点播、Internet 电话和电视会议等。RTP 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。RTP 协议与 TCP 协议十分相似，只是当差错造成分组丢失时，不要求重发，同时 RTP 规范中还定义了实时传输控制协议 RTCP，用于提供 QoS 监视机制。RTP 协议位于传输层之上，它没有连接的概念，虽然它既可以建立在面向连接的协议上，也可以建立在面向无连接的协议上，但是一般来说，RTP 作为实时数据传输而设计的，而建立在 UDP 协议之上，RTP/RTCP/UDP 协议一起用于视频音频流的实时传输。RTP 用于 UDP 数据封装时的情景如图 2-5 所示。

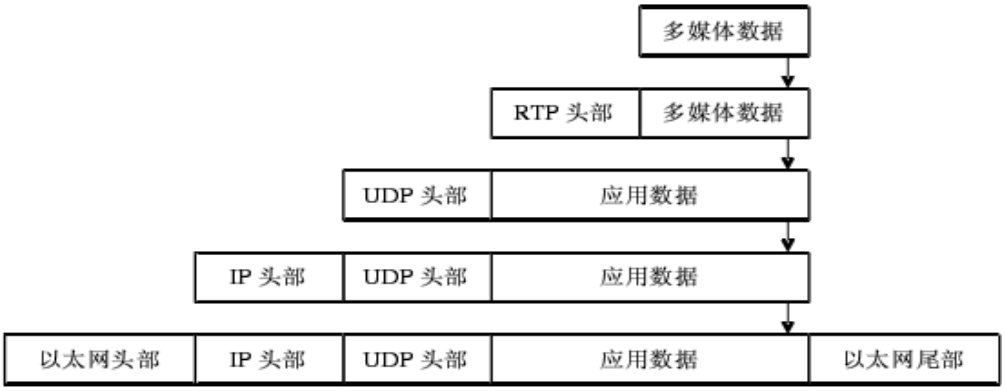

图 2-5　RTP 数据包的封装过程

Fig.2-5　Encapsulation Process of RTP Packet

RTP 协议对于实时多媒体数据的传输的特点有如下：

1）RTP 是一种轻型的传输协议，提供端到端的实时媒体传输功能，但并不提供机制来确保实时传输和服务质量。协议本身相对轻型、快捷，常常与具体应用结合在一起。

2）灵活性：RTP 协议将数据实时传输与控制策略分开。协议本身只提供实时传输机制，不具体规定控制策略。开发者可以根据不同的应用环境，选择实现效率较高的算法及控制策略。

3）独立性：RTP 协议与下层协议无关，可以在 UDP/IP、IPX、ATM 的 AAL 层上实现。

4）良好的扩展性：不仅支持单播，还支持组播。

RTP 协议的核心是其报文格式。报文是 RTP 对数据传输的封装单位，典型的报文由报头和负载组成，在协议中仅定义了报头的数据结构，而不限制负载的大小。RTP 报头由 16 个字节组成，其中最后 4 个字节 CSRC 域可选，格式如图 2-6 所示。

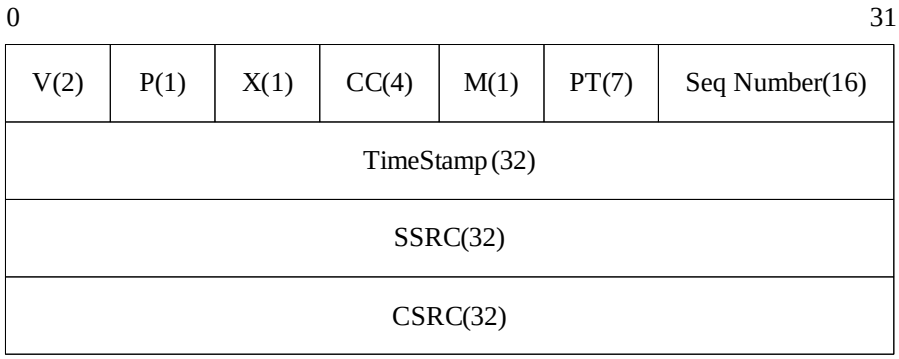

图 2-6　RTP 报头格式

Fig.2-6　Format of RTP Packet Header

V：RTP 版本号，2 位；

P：填充标识，1 位，置”1”表示用户数据最后有填充位，用户数据中最后一个字节是填充位计数，它表示一共加了多少个填充位。在两种情况下可能要填充，一是某些加密算法要求数据块大小固定；二是在一个低层协议数据包中装载多个 RTP 分组；

X：扩展位标识，1 位，置“1”表示 RTP 报头后紧随一个扩展报头；

CC：CSRC 计数，4 位，表示在定长的 RTP 报头后的 CSRC 标识符的数量；

M：标记，l 位，置“1”对于视频标识表示最后一帧；对于音频表示谈话开始。

PT(数据类型)：7 位，标识 RTP 报文内负载的数据类型；

SeqNumber(序列号)：2 字节，一个 RTP 传输会话中的所有 RTP 报文依次编号，其中第 1 个 RTP 包的编号可为 0 或为一个随机数。每发送一个 RTP 报文，序列号就加 1，接收端可以用它来检查报文是否有丢失并按顺序处理报文。

Timestamp(时间戳)：4 字节，标识 RTP 报文内负载的时间信息。它反映 RTP 数据信息中第 1 个字节的采样时刻(时间)。接收端可以利用这个时间戳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。

SSRC(同步源标识)：4 字节，用于标识数据源，该标识符在一次会话中是唯一的，如果有两个参与者的 SSRC 相同，就会产生冲突。

CSRC(混合器列表)：仅用于有混合器的情况下。可以有 0 到 15 项，每项 32 位，对一个 RTP 混合器(Mixer)产生的组合流有贡献的 RTP 分组源。CSRC 列表识别在此包中负载的有贡献源，识别符的数目在 CC 域中给定。若有贡献源多于 15 个，仅识别 15 个。

RTP 协议本身包括两部分：RTP 数据传输协议和 RTCP 传输控制协议。为了可靠、高效地传送实时数据，RTP 和 RTCP 必须配合使用，通常 RTCP 包的数量占所有传输量的 5%。RTP 实时传输协议主要用于负载多媒体数据，并通过包头时间参数的配置使其具有实时的特征。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 传输控制协议提供这些服务。

2.3.4　RTCP 协议

RTCP 是英文 Real-Time Transport Control Protocol 的缩写，中文名称是实时传输控制协议，是一种用来传输控制数据的协议，它主要向会话中的所有参与者定期发送控制数据报文，其发送的方式和 RTP 数据包相同[16]。RTCP 由多种不同的数据报文组成，每种报文传输不同的信息。这些报文包括以下 5 类：

发送者报告（SR）：用来发布数据发送者的传输信息；

接收者报告（RR）：用来发布数据接收者的数据接收质量统计信息，有时 RR 也包括在数据发送者报告中；

描述信息（SDES）：用来描述一个参与者；

离开报告（BYE）：用来通知会话中其他参与者自己要离开会话；

特殊应用报告（APP）：用来传输应用程序自定义的数据信息。

每个 RTCP 分组以一个固定首部开始，如图 2-7 所示，分组中其它字段的格式取决于分组首部中的类型字段。

图 2-7　RTCP 固定首部

Fig.2-7　Fixed Header of RTP Packet

RTCP 首部字段中，VER 表示版本号，P 字段指明是否填充，填充数据的最后一个字节表示填充数据的计数。分组的数据区包含一系列的报告记录，5 位的 RC 字段含有

报告计数。16 位的 length 字段用于说明分组的全部长度。8 位 P 字段说明了分组类型。RTCP 协议处理机制根据需要定义了四种类型的报文。这些类型决定了 RTCP 分组中报告的格式。

RTCP 协议的工作过程是：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，发送方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。RTCP 主要提供了四种功能：

首先，也是最主要的功能，RTCP 提供一种数据发布质量的反馈信息控制媒体数据的传送质量，也可用来监视网络和用来诊断网络中的问题，这是 RTP/RTCP 的核心功能，并和下层传输协议的流量控制等相关。

其次，由于端系统可能重新启动或在所有参与者中有可能发现 SSRC 冲突而改变自己的 SSRC 标志。数据接收者需要固定的标志来跟踪参与者。RTCP 传输的内容中包括一个固定的 CNAME 数据项，它是 RTP 源在传输层的标志，一般在会话中它是唯一并且固定的，在会话中可以使用它来标志不同的参与者，即使该参与者可能改变自己的 SSRC 标志。CNAME 标志也可以用来进行同步同一会话中的音视频等工作。

再次，由于参与者所发送的 RTCP 包，会话中所有参与者都可以接收的到，所以，一个参与者可以通过接收 RTCP 数据包估计会话中参与者的个数。根据参与者个数来调整发送 RTCP 数据的速度。

最后，一个可选特性，就是通过分析 RTCP 数据包描述会话成员，比如将他们的名字或 Email 地址显示在应用程序的用户界面上。

2.3.5 RTP 相关其他协议

在 RTP 协议或流媒体应用的运行过程中，有时还需要其它一些协议和 RTP 共同工作，来完成流媒体传输会话的管理和流媒体数据的传输。这些协议包括会话管理方面的 RTSP 协议和资源管理方面的 RSVP 协议等。

（1）RTSP 协议

实时流协议（Real-Time Streaming Protocol，RTSP）是一个应用层网络传输协议，由 IETF 发布，它主要用来对实时数据传输会话的控制[17]。RTSP 的一个主要功能是支持类似 VCR 的控制操作，如停止、暂停/重新开始、快进和快退，另一个主要功能是建立和控制在媒体服务器和客户机之间的连续音频/视频媒体流。RTSP 为实时数据的按需

和可控传输提供了一个可扩展的框架一般在设计的时候，与具体的应用相适应。进行传输的数据源可以是实时数据，也可以是存贮于介质上的历史数据。

RTSP 协议用于控制多路数据传输会话，并且这些流可以选择不同的下层传输协议，比如 UDP、组播 UDP 或 TCP 等，RTSP 为流的传输提供了选择下层传输协议的方法。在一般的流传输会话中，大都使用 RTP 作为下层传输协议，RTSP 自己并不传输媒体数据，只起一个控制传输的作用。它的很多操作和 HTTP 非常相似。HTTP 与 RTSP 相比，HTTP 传送 HTML，而 RTP 传送的是多媒体数据。HTTP 请求由客户机发出，服务器做出响应，使用 RTSP 时，客户机和服务器都可以发出请求，即 RTSP 可以是双向的。

（2）RSVP 协议

RSVP（Resource Reservation Protocol）是一种资源预留协议。它为 Internet 综合服务而设计的，提供了由接收方发起的面向组播或点对点数据传播的资源预留，保障了每一业务流都有足够的“ 独享 ”的带宽，克服了由于网络信包过多引起的拥塞、丢失和重传，提高了网络传输的 QoS 性能。

端系统（End System）利用 RSVP 协议为应用程序数据流向网络申请特定的 QoS，路由器等网络设备利用该协议传递 QoS 请求并建立和维护所需要的内部状态。RSVP 协议为单向的数据流进行资源的预留，所以对 RSVP 而言发送方和接收方并不相同，尽管某一进程可能同时是发送方和接收方。RSVP 协议处在 TCP/IP 协议栈中的传输层位置，虽然它并不传输任何实际的数据。RSVP 和 IGMP 或 ICMP 一样，是 Internet 的一种传输控制协议。

2.4　本章小结

本章首先对流媒体技术的概念与基本原理进行了简单介绍，包括流媒体的传输原理、流媒体的传输方式，然后对 RTP/RTCP/RTSP 协议分别进行了详细分析，为后文的流媒体网络传输实现奠定了理论基础。

第三章　DirectShow 的分析与研究

3.1　DirectShow 概述

3.1.1　DirectX 简介

Microsoft DirectX 是微软专门为基于 Windows 平台的游戏以及其它高性能多媒体应用程序的开发而提供的一套底层应用程序接口(API)函数，它支持三维图像和各类音效设备的输入。开发 DirectX 的主要目的是为了让运行在 Windows 平台的应用程序具有良好的表现力和实时的硬件访问能力。

DirectX 隐藏了复杂多变的硬件驱动及其相关的执行细节，因此，具有很好设备无关性，应用程序可以通过使用 DirectX 提供的编程接口直接对 HAL(硬件抽象层)进行操作，既充分发挥了计算机的各种优良性能，又简化了应用程序的编写[23]。

3.1.2　DirectShow 简介

DirectShow 是 Microsoft Windows 平台上处理流媒体的一套体系结构，是 Microsoft DirectX 家族的一员[18]。DirectShow 提供了高质量的多媒体数据采集和多媒体数据播放，它支持包括 ASF（Advanced Systems Format）、MPEG（Motion Picture Experts Group）、AVI（Audio-Video Interleaved）、MP3（MPEG AudioLayer-3）等在内的多种数据媒体格式，同时它也支持现在的 WDM 驱动的设备和过去 VFW 驱动的设备。DirectX 采用了 COM（Component Object Model）标准，而 DirectShow 是一套完全基于 COM 的应用系统。DirectShow 中的组件都是以 COM 对象的形式出现的，如果要开发一个 DirectShow 应用，我们应该必须知道如何使用 COM 组件，DirectShow 为用户提供了很多实用的组件来方便用户开发应用程序，但是，如果想扩展现有的 DirectShow 的功能来适应自己的应用，我们必须按照 COM 标准来实现我们的组件。

事实上，计算机应用领域中的很多模块都可以和 DirectShow 系统交互。也就是说，DirectShow 的应用范畴很广。单纯从本地系统来说，DirectShow 可以实现可以从本地机器中的采集设备采集音视频数据、压缩、保存、回放等功能。而从网络应用的角度来说，DirectShow 更可用于视频点播、视频会议、视频监视等领域。

3.1.2　组件对象模型（COM）基础知识

COM 作为 Win32 的一种基础架构，在 Windows 平台上的应用非常广泛，DirectShow 是一套完全基于 COM 标准的应用系统。如果要开发 DirectShow 应用，需要熟悉 COM 客户端应用开发的过程和方法；如果要开发 Filter 组件来扩展 DirectShow 的功能，就需

要熟悉 COM 组件的实现过程和方法，因为 Filter 本身就是一种 COM 组件。

在 COM 标准中，最重要的三个概念应该是 COM 组件、COM 对象和 COM 接口了。其中 COM 对象是一个非常活跃的元素，它是对外提供服务的主体；COM 组件为 COM 对象活动提供空间的一个容器，在 Windows 平台上可以是一个 DLL（Dynamic Linking Library）文件也可以是一个可执行的文件（EXE）；COM 对象以接口的方式向外界提供服务，这种接口被称作 COM 接口（COM Interface），COM 接口实际上是一个 COM 和 COM 客户程序之间的契约，它由一组被定义但是没有实现的函数原型所组成，也就是所谓的 C++中的纯虚类。一个 COM 组件中可以包含一个或几个 COM 对象，一个 COM 对象也可以实现一个或几个 COM 接口。图 3-1 简单说明了 COM 组件、COM 对象和 COM 接口之间的关系。

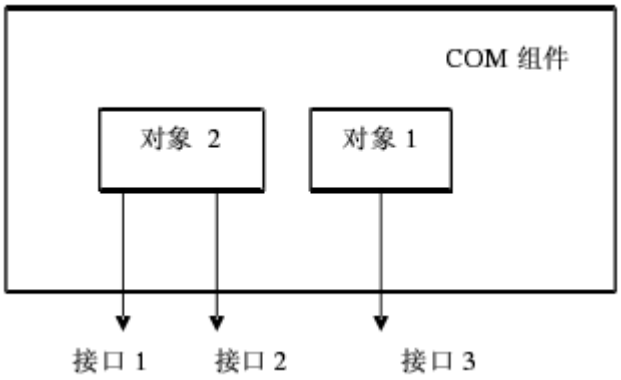

图 3-1　COM 组件、对象和接口之间的关系

Fig.3-1　The relations between COM Component、object and interface

COM 组件和 COM 接口都由一个 128 位的 GUID 来标识，组件的标识称为 Class ID（CLSID），而接口的标识被称为接口 ID（IID）。应用程序需要用这两个 GUID 来创建对象，以使用 COM 组件和组件中对象的服务。根据 COM 组件在客户端的加载方式不同，可以将 COM 组件分为三类：进程内组件、本地进程间组件和远程组件。DirectShow 中的 Filter 一般是一种进程内组件，以 DLL 的形式为应用提供服务。下面简单讲述进程内组件的应用和开发的关键过程。

当 COM 客户端调用 COM 的功能时，它首先创建一个 COM 对象或者通过其它途径获得 COM 对象，然后通过该 COM 对象所实现的 COM 接口调用 COM 对象所实现的服务。当所有的调用结束以后，如果客户端程序不再需要使用该 COM 对象，则它应该释放该对象所占有的资源和对象本身。此过程的具体实现可在讲述 COM 的文献上找到。

对于 Filter 的开发者来说，掌握 COM 组件、尤其是进程内组件的实现方法显得比较重要。对于普通开发人员来说，熟悉 COM 体系结构，开发 COM 组件是一件比较困难的事情，但是，DirectX SDK 为用户提供了相当丰富的基类，一般应用可以通过继承合适的基类来实现 Filter 的功能[22]。

3.2　DirectShow 结构

设计开发 DirectShow 的主要目的便是通过隔离应用系统和数据处理的复杂性、硬件系统的多样性和多个流之间同步的复杂性来简化在 Windows 平台上开发数字多媒体应用的过程，降低开发系统的难度为了适应数据媒体格式和硬件设备的多样性，DirectShow 使用一种子系统——过滤器（Filter）来处理软件和硬件之间的差异。图 3-2 显示了 DirectShow 应用程序、DirectShow 系统和 DirectShow 支持的软硬件之间的关系[18]。

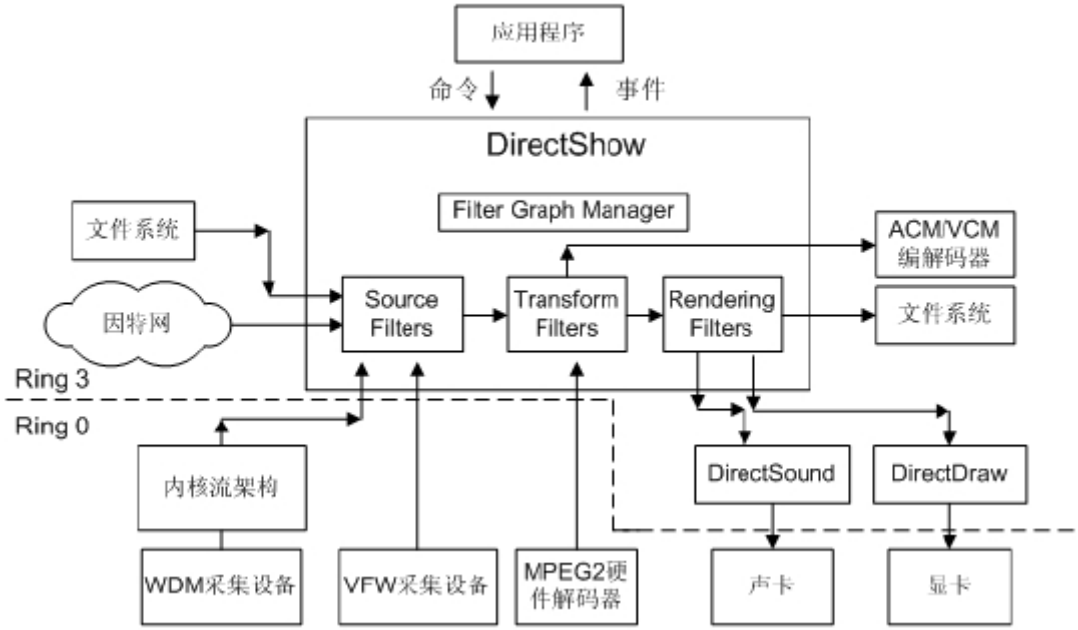

图 3-2　DirectShow 系统结构

Fig.3-2　DirectShow system structure

如图所示，中间方框以内是 DirectShow 系统，虚线以下是 Ring 0 级别的硬件设备及其驱动程序，虚线以上是 Ring3 级别的应用程序，DirectShow 位于应用程序层。DirectShow 数据处理过程一般由 Filter 来完成，所有这些 Filter 由一种被称为“Filter Graph”的模型来管理，使所使用的所有 Filter 根据先后顺序连接成一条生产线来协同工作。系统中的 Filter 分为几类，有些可以在系统以外的介质或设备上获得数据，有些可以以某种方式来处理这些数据，比如改变视频流的格式等，另外一些可以将处理好的数据送到系统的外界，比如写到文件里或在屏幕上显示图像等。DirectShow 通过这些不同的 Filter 和外界的物理设备交换数据，并控制这些设备，从而使 DirectShow 可以将应用程序和这些复杂的多种多样的设备隔离开来。

3.3　DirectShow 过滤器（Filter）

3.3.1　Filter 原理及应用

Filter 是一种 COM 组件，一般它都是进程内组件，它是 DirectShow 最基本的概念，DirectShow 中对数据的每一步操作都由 Filter 来完成。DirectShow 为用户提供了一组标准的 Filter，它们可以完成一些基本的数据操作。各个 Filter 之间是连接在一起的，这些 Filter 上的连接点也是一个 COM 对象，称作 Pin，各个 Filter 之间通过 Pin 组件传递数据流，这种多个 Filter 连接在一起的 Filter 组称作 Filter Graph。管理 Filter Graph 的管理者称作 Filter Graph Manager，也是一个 COM 对象。Filter Graph 是 Filter 的一个容器，Filter 是容器中的最小功能模块，作为 COM 组件，Filter 至少实现并暴露 IBaseFilter 接口给应用程序，而 Pin 则一定要暴露 IPin 接口。

根据 Filter 在 Filter Graph 中的位置不同或根据 Filter 上 Pin 的情况不同，可以将 Filter 分为三类，如图 3-3 所示[18]：

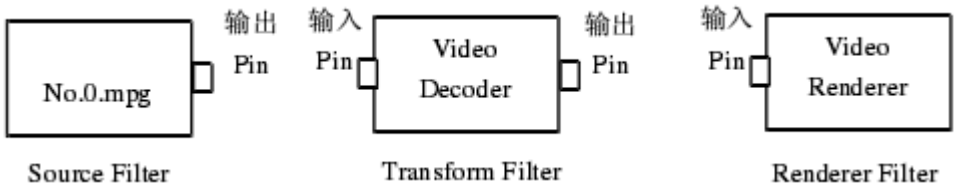

图 3-3　Filter 的种类

Fig3-3　Filter species

1）源过滤器（Source Filter），在 Graph 的开始位置，它的主要工作是从 Graph 外界获得数据，这些数据源的介质可以是硬盘文件、网络、其它数码设备。这些 Filter 一般只有输出 Pin（有些视频采集卡 Filter 上会有模拟的输入 Pin），不同的源 Filter 处理不同格式和形式的数据。

2）传输过滤器（Transform Filter），在 Graph 的中间位置，它们从上游的 Filter 获得数据，对数据进行一定的处理，比如编码、解码等操作，然后生成输出的数据流传输给下游的 Filter，这些 Filter 既有输入 Pin 又有输出 Pin。

3）渲染过滤器（Renderer Filter），它一般在 Graph 的末尾，从上游 Filter 接收数据，然后将数据以某种形式展现给用户，比如显示图像、播放声音或写入文件等，这些 Filter 一般只有输入 Pin，没有输出 Pin。

在使用过程中，Filter 有三种不同的状态：运行（Running）、暂停（Paused）和停止（Stopped）。其中运行状态表明 Filter 正在处理数据；停止状态是 Filter 停止处理数据的一种状态；而暂停状态一般用作在运行状态之前等待数据就绪的一个状态。一般来讲，Filter Graph 中的所有 Filter 的状态是统一改变的，它们的状态也是统一的，所以我们也可以称 Filter Graph 的状态是运行、暂停还是停止。

3.3.2　Filter 的连接

Filter 只有加入到 Filter Graph 中并且和其他 Filter 连接成完整的链路后，才会发挥作用。Graph 中的 Filter 要协同工作，应该首先按照一定的顺序连接在一起，Filter 的连接实际上是 Filter 上的 Pin 之间的连接。连接的方向一般是从上游（Upstream）Filter 的输出 Pin 指向下游（Downstream）Filter 的输入 Pin。 Pin 的连接是连接的两个 Pin 之间使用媒体类型的“ 协商 ”过程，Pin 通过 IPin 接口来完成相互之间的连接。在连接过程中，判断媒体类型是否合适和分配 Sample 分配器是两个非常重要的步骤。在 IPin 中分别有 CheckMediaType 和 DecideAllocator 两个方法来完成这两项功能。过滤器引脚之间的连接过程如图 3-4 所示，连接步骤大致如下：

(1)过滤器链路管理器调用过滤器输出引脚上的 IPin::Coilnect，并传入一个输入引脚指针参数；

(2)输出引脚一旦接受连接，则调用输入引脚的 IPin::ReceiveConnection 函数；

(3)如果输入引脚也接受连接，则连接成功。

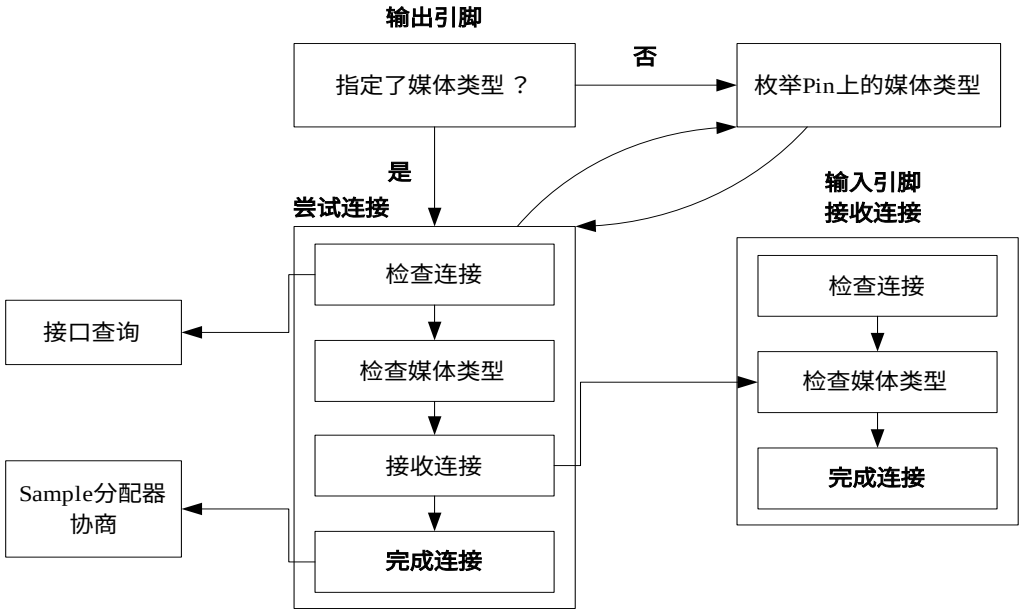

图 3-4　Pin 的连接过程

Fig.3-4　Pin connection process

3.3.3　Filter 之间数据传输原理

Filter 之间的成功连接为数据传输做好了准备。在 Pin 之间传输数据时，它们并不直接传输数据内存的地址，而是传输一个指向 COM 组件的指针。这个 COM 组件管理要传输的数据内存，我们称这个 COM 组件为 Sample，它实现了 IMediaSample 接口并暴露给应用程序。下游的 Filter 可以通过调用 IMediaSample 的接口方法获得媒体数据或 Sample 的描述信息。

Sample 中的数据由 Sample 对象来管理，Sample 对象也需要一个工具来管理，这种工具是另外一种 COM 组件称作分配器（Allocator）。Allocator 实现并暴露给应用程序 IMemAllocator 接口，此对象负责创建和管理 Sample。当 Filter 需要一个空闲 Sample 时就向 Allocator 申请，Allocator 返回给 Filter 一个指向 Sample 的指针，图 3-5[18]显示了数据传输和 Sample 分配管理的过程。

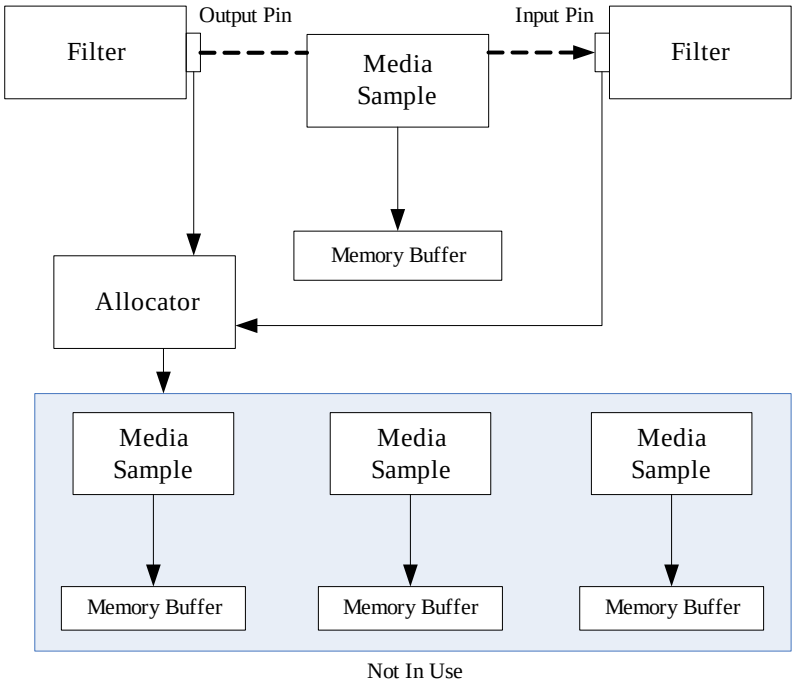

图 3-5　Media Sample 在 Filter 间传递

Fig.3-5　Media Sample’s transmit between Filers

Filter 之间数据传送的方式有两种：推模式和拉模式。其中推模式的数据传输是上游的 Filter 将自己所产生的 Sample 用专门的线程“推”给下游的 Filter；拉模式的数据传输是上游的 Filter 没有主动推数据的功能，需要下游的 Filter 向他申请数据，上游的 Filter 接收到请求后向下传递数据。

推模式：在源 Filter 后面的 Filter 输入 Pin 上，实现了一个 IMemInputPin 接口，数据正是通过上一级 Filter 调用这个接口的 Receive 方法进行传输。数据从输出 Pin 通过 Receive 方法调用传输到输入 Pin 上，并未进行内存拷贝，它只是一个相当于数据到达的“通知”。然后由上游的过滤器决定发送的数据并把数据“推”向下游的过滤器，下游过滤器被动接收并对数据进行处理。

推模式最典型的例子是用于实时多媒体数据源(Live Source)的发送，如图 3-6 所示。例如：视频采集卡等设备。这些设备以一种源过滤器的方式出现在过滤器图表中，它能够自己产生数据，并且使特定的线程将数据“推”向下一级过滤器。

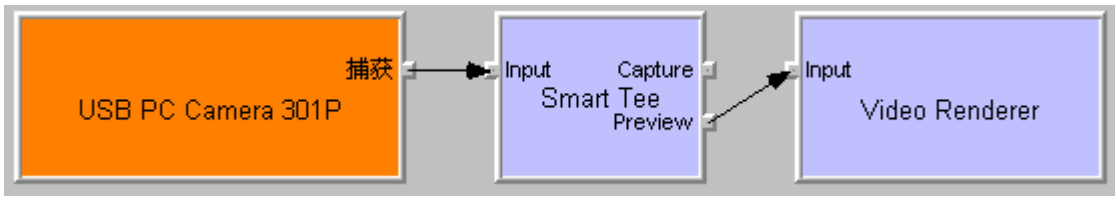

图 3-6　推模式图表

Fig.3-6　Graph of push mode

拉模式：源 Filter 的输出 Pin 上都实现了一个 IAsyncReader 接口，其后面的分离过滤器(Splitter Filter)，就是通过调用这个接口的 Request 方法或 SyncRead 方法来获得数据。分离过滤器像推模式一样，调用下一级 Filter 输入 Pin 上 IMemInputPin 接口 Receive 方法实现数据的往下传送。

这种数据传输模式由处于链路下游的过滤器主动向上游过滤器请求获取数据。虽然形式上媒体样本仍然是从上游的输出引脚传送至下游的输入引脚，但数据传输的主动权掌握在下游过滤器手中。媒体文件的回放就是拉模式方法最典型的应用，如图 3-7 所示。

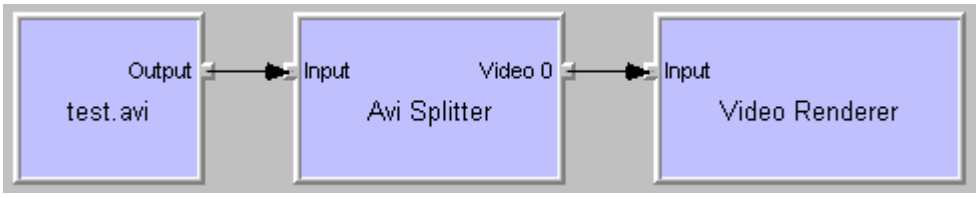

图 3-7　拉模式图表

Fig.3-7　Graph of poll mode

3.4　DirectShow 的内部机制

3.4.1　线程机制

DirectShow 应用程序至少包含两种重要的线程：应用程序主线程以及一个或多个媒体流线程。Filter 的状态变化在应用程序主线程发生，而媒体流线程处理 Sample 的传送。在应用程序主线程等待用户交互信息的同时，媒体流线程处理数据流在过滤器图中的传输。多线程处理则要注意线程同步问题，DirectShow 的解决方案是用临界区来保护敏感资源。每个过滤器的状态需要由一个临界区来保护。在应用程序主线程中使用的函数，必须使用 Filter 状态同步对象进行同步；在媒体流线程中使用的函数，必须使用媒体流同步对象进行同步。媒体流线程必须在 Filter Graph Manager 停止 Filter Graph 之前被关闭[20]。

3.4.2　事件机制

DirectShow 中应用程序与 Filter Graph 的交互控制可以通过事件通知机制(Event Notification)来实现。例如当数据回放完毕或是运行错误发生时，发出相应的事件，然后通过事件机制递交事件通知给 Filter Graph Manager 或者是应用程序处理。应用程序从消息队列中取得事件，并根据事件的类型做出相应的处理。DirectShow 中的事件通知机制跟 Windows 下的消息队列很类似。一个应用程序可以取消 Filter Graph Manager 对一个特定事件类型的默认反应，这个时候 Filter Graph Manager 把这种事件直接放入队列中让应用程序来处理[20]。

这种机制可以实现保证如下功能：

- Filter Graph Manager 和应用程序进行信息交互。
- Filter、应用程序和 Filter Graph Manager 之间进行信息交互。
- 由应用程序确定它自己要处理哪些事件和怎样处理这些事件。

常见的事件有 EC_COMPLETE，表示 Filter Graph 中所有的数据都已经回放完毕(此时 Filter Graph 不会自动转入 Stopped 状态，需要我们在应用程序中接收到这个事件后调用 Filter Graph Manager 的 IMediaControl::Stop 方法)；有 EC_ERRORABORT，表示运行时出错；有 EC_DEVICE_LOST，表示热插拔设备(典型的如 USB 设备、1394 接口设备等)脱离系统等等。（其他事件参见 DirectX 文档）。

Filter Graph Manager 提供以下三个接口(IMediaEventSink、IMediaEvent 和 IMediaEventEx)来支持事件通知，其中 IMediaEventSink 用在 Filter 内部，它的接口方法 Notify 用以向 Filter Graph Manager 发送事件通知。IMediaEvent 给应用程序提供了得到事件的方法。IMediaEventEx 是 IMediaEvent 的扩展，在应用程序一般使用这个接口处理 Filter Graph Manager 发出的事件。发出的事件。事件处理的大致过程如下：Filter Graph 中的 Filter 发出一个事件（运行出错了或者满足了一定的条件），接收者为 Filter Graph Manager；Filter Graph Manager 对一些特殊的事件有默认的处理方法，在接收到事件后，要么按默认方法直接处理这个事件，要么放到一个事件队列中等待上层的应用程序处理；应用程序在获知 Filter Graph Manager 有事件发出后，就可以使用 IMediaEventEx 接口从事件队列中读取事件，然后根据事件类型做出相应的处理。

3.4.3　时钟机制

DirectShow 结构最核心的部分是 Filter Graph Manager，它实现了两个功能：向下控制 Graph 中的所有 Filter，向上对应用程序提供编程接口。时钟机制是 Filter Graph Manager 的一个很重要的功能。时钟机制的引入主要是为了解决多媒体数据中音视频的同步问题。简单说就是选一个公共的参考时钟，并且要求给每个 Sample 都打上时间戳，Video Renderer 或 Audio Renderer 根据 Sample 的时间戳来控制播放。如果到达 Renderer 的 Sample 晚了，则加快 Sample 的播放；如果早了，则 Renderer 等待，一直到 Sample

时间戳的开始时间再开始播放。

Filter Graph Manager 在 Filter Graph 运行的时候自动地选择一个参考时钟，同一条链路的 Filter 参照同一个参考时钟。参考时钟的选择的算法如下[20]：

- 若 Filter Graph 设置了一个参考时钟，则直接使用这个参考时钟。
- 若 Filter Graph 中有支持 IReferenceClock 接口的 Live Source，则选择该 Live Source。Live Source 也称为推 Source，可以实时地接收数据，例如视频采集源就是一个典型的 Live Source。
- 若 Filter Graph 中没有 Live Source，则从 Renderer Filter 依次往上选择一个实现了 IReferenceClock 接口的 Filter。若连接着的 Filter 都不能提供参考时钟，则再从没有连接的 Filter 中选择。假如 Filter Graph 中含有一个播放音频的链路，由于声卡上一般都带有硬件定时器，则优先选择 Audio Renderer。
- 若以上方法都找不到适合的 Filter，则选取系统参考时钟。即创建一个 CLSID_SystemClock 的组件对象实例。

DirectShow 定义了两种时间：参考时间(Reference Time)和流时间(Stream Time)。前者取决于参考时钟的内部实现，是从参考时钟返回的绝对时间(调用 IReferenceClock::GetTime)。后者应用于 Filter Graph 内部的同步，是两次从参考时钟读取的数值的差值。流时间的取值算法如下：

- Filter Graph 运行时，取值为当前参考时钟时间减去 Filter Graph 启动时的时间；
- Filter Graph 暂停时，保持为暂停那一刻的流时间；
- 执行完一次随机定位（Seek）操作后，数值复位至零；
- Filter Graph 停止时，取值不确定。

3.4.4　质量控制机制

时钟机制是到对每个 Sample 打上时间戳，虽然 Renderer Filter 接收到 Sample 后，会根据 Sample 上的时间戳安排显示时机，但要彻底解决 Filter Graph 运行性能，DirectShow 设计了一种“自适应”的反馈机制：质量控制机制(Quality Control)。质量控制主要是调整整条链路运行状态时数据流传输的速率[20]。若 Renderer Filter 发现数据线程发送数据太快或太慢时，它就会发送一个质量消息，要求 Source Filter 调整数据流的速率。

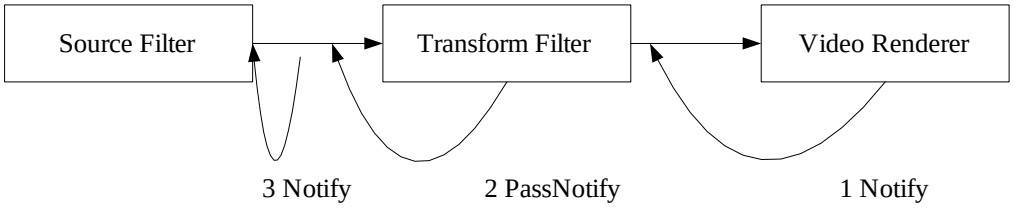

图 3-8　DirectShow 质量控制过程

Fig3-8　Quality control process of DirectShow

图 3-8[18]显示质量控制消息最初由 Video Renderer 发出，以“回溯”的方式逐个询问是否可进行质量的改善。

DirectShow 中使用如下的数据结构描述质量控制消息：

```
Typedef struct{
    QualityMessageType Type;
    Long Proportion;
    REFERENC_TIME Late;
    REFERENCE_TIME TimeStamp;
}Quality;
```

- 其中 Type 是一个枚举器，定了消息的型。

```
Typedef enum{
    Famine,          //表示当前数据发送太慢
    Flood            //表示当前数据发送太快
}QualityMessageType；
```

- Proportion（比例）：表示需要调整的码率与原先码率的比例关系。以 1000 为基线，150 表示调整 15%，1000 表示调整 100%；
- Late（延迟）：最新帧的延迟时间(到达时间与时间戳的差值)。
- Timestamp（时间戳）：最新帧的时间戳。

质量控制是通过 Quality Control 接口来处理的。该接口包含两种方法，Notify：任何实现 IQualityControl 的对象通过该方法接收质量消息，它可以处理该消息或者把它传递到另外一个质量控制对象；SetSink：指定质量管理器。

能够调整发送速度的 IQualityControl 接口一般在 Source Filter 上实现，Transform Filter 只是将 Quality Message 往上一级 Filter 传递。应用程序可以实现自己的 Quality Control Manager，然后通过调用 SetSink 方法设置给 Filter。

3.5 本章小结

本章主要介绍 DirectShow 框架的原理，深入剖析了 DirectShow 的内部机制，由于 DirectShow 框架是基于微软组建对象模型（COM）的，在 DirectShow SDK 中提供了封装好的 DirectShow 的 C++类库，这个类库已经实现了常用 Filter 上必需的接口，而且提供了基本的 Filter 框架，可以实现对多媒体数据流的强大的控制能力，因此解决了软件的重用性问题。本章内容为下面章节详细阐述的各功能模块设计和编程实现作好铺垫。

第四章　系统结构设计

4.1　系统总体结构设计

本监控系统架构建立在第三代数字视频监控系统的基础上，采用 C/S 结构，以流媒体技术为核心，运用 DirectShow 进行音频视频采集、压缩，并结合网络 RTP/RTCP 传输技术实现了主机间的数据通信，提出了一种在网络上应用流媒体技术实现音频视频实时传输的远程视频监控系统的设计。系统结构如图 4-1 所示。系统分为服务器端和客户端，两端的软件均在普通 PC 机上运行。服务器端 PC 接有 USB 摄像头，软件首先读出摄像头的设备名和音频设备名，在采集视频数据进行显示的同时，将视频数据采用 MPEG-4 压缩编码，可录像到硬盘上。当有客户端请求视音频数据时，将压缩后的数据通过 IP 网络发送出去，客户端软件接收到视音频数据后，存储到硬盘上或解压后进行播放。

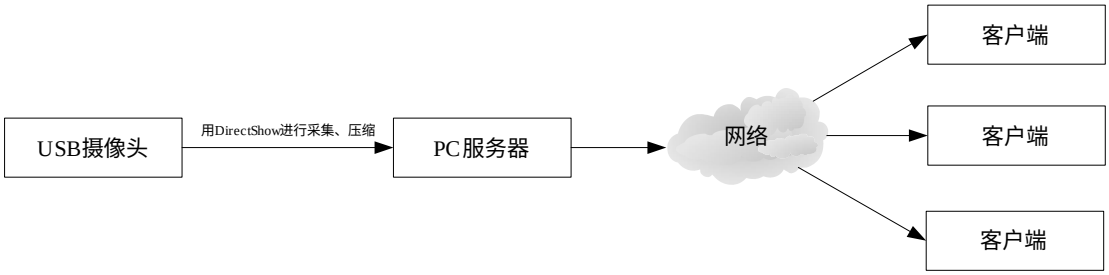

图 4-1　系统总体结构设计

Fig.4-1　Design of system total structure

本系统有以下功能：

（1）进行本地采集设备的自检，可检测到系统所采用的视频设备和音频设备名称；

（2）进行本地的视频图像预览显示；

（3）进行本地视频和音频采集参数调整；

（4）客户端可呼叫连接服务器请求视音频数据；

（5）实现客户端的视频录像、回放及抓图功能。

本系统设计的整套方案硬件上只需要市面上普通的 USB 摄像头和 PC 机，大大降低了硬件成本和开发成本，易于实现。虽然在视频效果上无法与专业的安防设备相比，但是 100 万像素的摄像头可以实现 320*240，甚至更高分辨率的画面，足以满足一般的小区、家庭监视的需要。本系统只需进行少许修改，还可以应用于远程教学，远程医疗中。

4.2　服务器端软件模块设计

服务器端软件分为五个模块，如图 4-2 所示

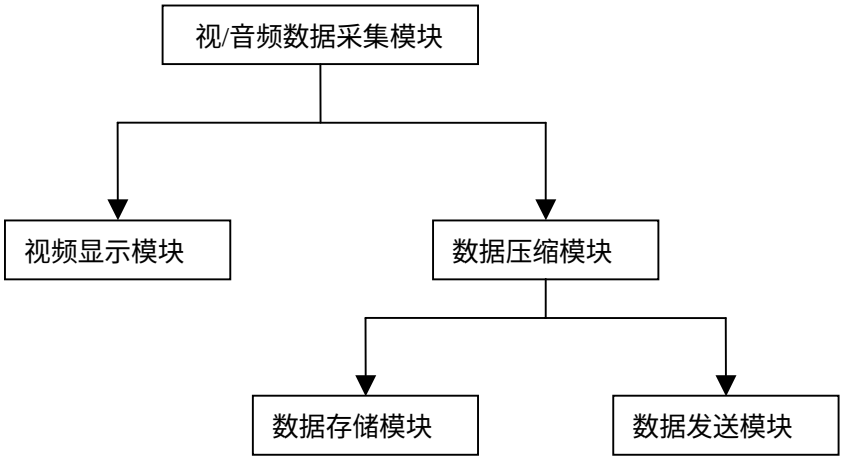

图 4-2　服务器软件模块组成图

Fig.4-2　Component graph of server software module

（1）视/音频数据采集模块：该模块从 USB 摄像头采集视频数据，从音频设备（声卡或麦克风）采集音频信号，以帧的形式分别发送到显示模块和压缩模块。

（2）视频显示模块：将采集模块送来的视频数据进行播放。

（3）数据压缩模块：将采集模块送来的视/音频数据压缩，减小数据量，并传送给数据存储模块和发送模块。

（4）数据存储模块：将压缩后的视/音频数据存储到本地硬盘上。

（5）数据发送模块：将压缩后的数据用 RTP/RTCP 协议发送到网络。

4.3　客户端软件模块设计

客户端软件分为五个模块，如图 4-3 所示

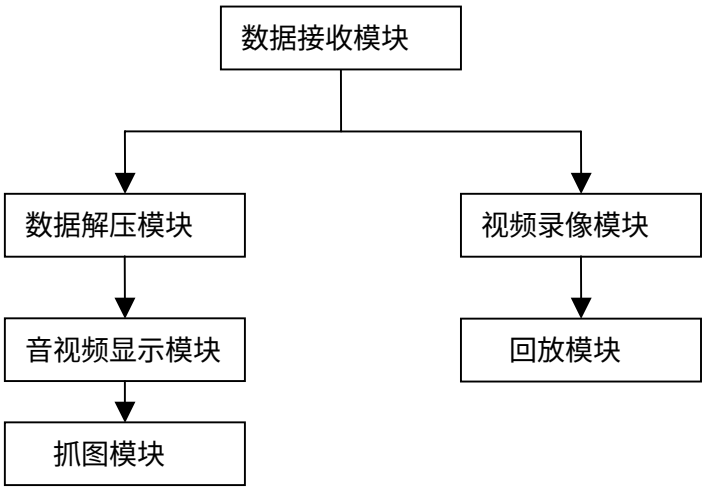

图 4-3　客户端软件模块组成

Fig.4-3　Component graph of client software module

（1）数据接收模块：该模块从网络中接收服务器端发送来的视/音频数据，并分别发送到数据存储模块和数据解压模块。

（2）数据存储模块：将接收模块送来的视频数据存储到本地硬盘上。

（3）数据解压模块：将接收模块送来的视频数据解压，并传送给视频显示模块。

（4）视音频显示模块：将解压后的数据进行播放。

（5）回放模块：回放视频文件。

4.4　网络传输模块的设计

网络传输部分是通过设计两个 Filter 来实现的，分别是网络发送 Filter（NetSend Filter）和网络接收 Filter（NetReceive Filter）。在传输中使用 RTP/RTCP 将视频数据和音频数据分别封装和传输。在方案中，我们将使用不同的端口进行视频和音频的传输。同时对视频和音频数据报使用统一的时间戳，以保证它们在客户端的同步播放。在正式开始媒体数据传输之前，发送端必须向接收端通知所传输的媒体类型。因为接收端需要根据媒体类型来确定所使用的解码器，以正确建立 Graph，而只有在 Graph 运行之后，接收端才能通过网络接收 Filter 接收媒体数据。发送端通知接收端的方式属于控制信息传输部分，实现方式与通常的网络传输相同。传输结构图如 4-4 所示

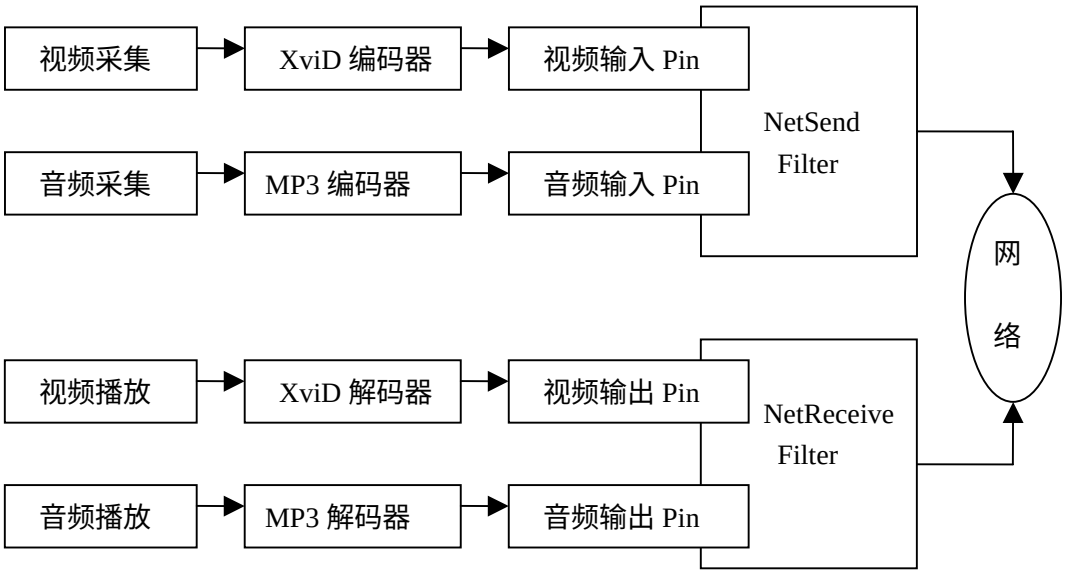

图 4-4　网络视频/音频传输结构图

Fig.4-4　Structure graph of web video/audio transmission

网络发送 Filter 的输入 pin 接收经过压缩编码的视音频数据后根据不同类型确定 RTP 数据报的负载类型（PT）和时间戳增量，封装好 RTP 数据分组后发送出去，网络接收 Filter 将接收到的数据经 RTP 数据分组分解，提取媒体实例样本，设置媒体实例的格式，主类型，采样频率和样本长度等，并将此媒体实例放入 Filter 的缓冲区中，准备传送至下一级解码器的输入 pin。

4.5　本章小结

本章对基于流媒体技术的网络视频监控系统的总体框架进行了设计，并规划了服务器端和客户端各功能模块的具体功能，并介绍了网络发送端和接收端过滤器的设计思想，为下一步的实现打下了坚实基础。

第五章　系统实现

5.1　开发环境与工具

5.1.1　开发环境

本系统的整个开发过程在局域网中实现。局域网中的 PC 服务器配置采用 Microsoft Windows 2000 Professional 操作系统，摄像头为市面上普通的 130 万像素摄像头，通过 USB 与计算机相连。客户端则是相同配置的 PC 机，采用 Windows XP 系统。

5.1.2　开发工具

系统采用 Microsoft Visual C++6.0 作为系统软件的编程语言和编译环境。Visua1 C++作为一个集成开发工具，为编程工作者提供了程序框架代码自动生成和可视化的资源编辑功能，从而使编程工作变得更为简单。并且 Microsoft 为 Visua1 C++提供了强大的基本类库 MFC(Microsoft Foundation Classes)，因此确立了 Visual C++在开发语言平台上的领先地位，它真正把 Windows 应用程序开发带入了一个面向对象的时代。Visual C++不仅仅是程序设计语言，而且也是一个非常全面的应用程序开发环境，使用它可以开发具有专业水平的 Windows 应用程序。

同时，由于各模块的实现都是使用微软公司的 DirectShow 技术，用于开发的计算机还需安装 Microsoft DirectX 9.0 SDK 程序、Microsoft DirectX 9.0 运行时库程序。借助 DirectX 底层驱动的支持，通过 COM 对象和模块化的方法，把多媒体数据的捕获、传输、处理与呈现从硬件设备的差异、同步、媒体数据格式等复杂问题中分离出来，提供统一的 API 和基类，从而简化了 Windows 平台下多媒体应用程序的开发。

5.2　服务器端软件实现

5.2.1　视频采集模块的实现

视频采集是实现系统的第一步，也是 DirectShow 最基本的应用之一。所谓采集，通常是指将模拟信号采样生成的数字信号，经过计算机处理后再现或存储到数字介质上。由于采集设备性能的差异以及兼容性问题，加上多媒体本身巨大的数据量，一般来说，采集的任务比较繁重，占用的系统资源也比较多。DirectShow 的出现使得采集过程变得较易实现，它对视频采集硬件的支持通过特定的包装 Filter 来实现，屏蔽了各种采集设备之间的差异，开发更为简单方便。DirectShow 可以支持多种接口的视频采集卡，它们

可以以 PCI 或 AGP 的方式直接插入 PC 机箱，也可以 USB 接口的方式外挂。

采集视频数据，首先要获取采集设备信息。系统可能运行在不同的计算机、不同的操作系统上，目标系统的硬件配置情况是不确定的，因此要采用系统枚举的方法来获取采集设备。只要采集设备正确安装，DirectShow 就能把它包装成一个 Filter，并且在一定的类型目录下注册。音频采集设备注册在 Audio Capture Sources 目录下，视频采集设备注册在 Video Capture Sources 目录下。应用程序只要枚举特定的类型目录，就能知道系统中安装有多少个、以及何种类型的采集设备。当我们执行 DirectX SDK 下的 GraphEdt 程序并选择插入 Filters 时，可在 Audio Capture Sources 和 Video Capture Sources 目录下找到代表安装在本地 PC 机上的音频和视频采集卡的 Filter。图 5-1 是本论文采用的音频和视频采集卡的 Filter 示意图。

图 5-1　视音频采集卡在 GraphEdt 下的示意图

Fig.5-1　Sketch map of Video/Audio Capture Card under GraphEdt

运用 DirectShow 实现视频采集模块的功能可分为以下四个步骤[21]：

（1）首先创建一个过滤器链路管理器（Filter Graph Manager）实例，并通过调用 IGraphBuilder 接口引出 IMediaControl（媒体控制），IVideoWindow（视频窗口控制），IMediaEventEx（媒体事件扩展）三个接口。应用程序通过调用 IVideoWindow 接口设置视频窗口的大小和位置；通过设定控制按钮调用 IMedidaControl 接口实现视频播放的开始和停止。核心代码如下：

```
BOOL CDXGraph::Create (void)
{
  if (!mGraph)
  {
```

```
            if (SUCCEEDED(CoCreateInstance(CLSID_FilterGraph, NULL,
          CLSCTX_INPROC_ SERVER, IID_IGraphBuilder, (void**) &mGraph)))
            {
                ……
                return QueryInterfaces ();
             }
             ……
     }
     return FALSE;
}
BOOL CDXGraph::QueryInterfaces (void)
{
   if (mGraph)
   {
          HRESULT hr=NOERROR;
          hr |= mGraph->QueryInterface(IID_IMediaControl,    (void
**) &mMediaControl);
          hr |= mGraph->QueryInterface(IID_IMediaEventEx,      (void
**) &mEvent);
          hr |= mGraph->QueryInterface(IID_IVideoWindow,       (void
**) &mVideoWindow);
          ……
          return SUCCEEDED(hr);
     }
     return FALSE;
}
```

（2）然后创建一个捕获链路（Capture Graph）实例，得到 ICaptureGraphBulider2 接口，DirectShow 专门提供了一个辅助组件 Capture Graph Builder(它的 CLSID 为 CLSID_CaptureGraphBuilder2)来简化这种 Filter Graph 的构建。通过调用接口方法 ICaptureGraphBuilder2::SetFiltergraph 设置 Filter Graph Manager 对象指针，与 IGraphBuilder 接口相连接，初始化 Filter Graph。核心代码如下：

```
hr=CoCreateInstance((REFCLSID)CLSID_CaptureGraphBuilder2, 0,
              CLSCTX_INPROC, (REFIID) IID_ICaptureGraphBuilder2,
              (void**)&mGraphBuilder);
pass=(mGraphBuilder!=NULL);
if (pass)
{
       mGraphBuilder->SetFiltergraph(mGraph->GetGraph());
       ……
}
```

（3）运用枚举系统设备方法，列举所有使用的音频采集设备并生成列表，用户可以选择任一设备，将其对应的 Capture Filter 加入到初始化后的 Filter Graph 中。

DirectShow 提供了一个专门的系统枚举组件（CLSID_SystemDeviceEnum），枚举的大致过程如下：

（1）使用 CoCreateInstance 函数创建一个系统枚举组件对象，并获得 ICreateDevEnmu 接口。

（2）使用 ICreateDevEnum 的方法 CreateClassEnmuerator 为指定的类型目录创建一个枚举器，并获得 IEnumMoniker 接口。

（3）使用 IEnumMoniker 的接口方法 Next 枚举指定类型目录下所有的设备标识(DeviceMoniker)。每个设备标识对象上都实现了 IMoniker 接口。

（4）调用 IMoniker 接口的 BindToStorage 函数之后就可访问设备标识的属性集，比如得到设备的显示名字（Display Name）、友好名字（Friend Name)等。

（5）调用 IMoniker 接口的 BindToObject 函数可以将设备标识绑定成一个 DirectShow Filter,随后调用 IFilterGraph::AddFilter 加入到 Filter Graph 中就可以参与工作了。核心代码如下：

```
hr=CoCreateInstance(CLSID_SystemDeviceEnum,NULL,CLSCTX_INPROC_SERVE
R,IID_ICreateDevEnum,(void**) &enumHardware);
                                                    //创建系统枚举组件对象
hr=enumHardware->CreateClassEnumerator(inCategory,&enumMoniker,0);
                              //指定枚举的类型目录，获得 IEnumMoniker 接口
if (hr==S OK)
 {
    IMoniker *pMoniker=NULL;
    if  (pEnum->Next(1,&pMoniker,NULL)==S_  OK)  //假设系统中只有一个采集设
备，故枚举到的第一个设备就是符合我们要求的采集设备
    {
      ……
      moniker->BindToStorage(0,0,IID_IPropertyBag,(void**)
                                  &propertyBag);
      ……
      hr=pMoniker->BindToObject(0,0,IID_IBaseFilter, (void
          **)&m_pCapture);//创建采集 Filter 实例
```

枚举到采集设备后，下一步的工作就是把它加入到 Filter Graph 中去。代码如下：

```
hr=m_pIGraphBuilder->AddFilter(m_pCapture, L"CaptureFilter");
```

（6）渲染捕获 Filter 的捕获输出 Pin（Capture Output Pin）或预览 Pin（Preview Pin），都可以播放捕获到的媒体流，实现实时预览功能。

音频采集模块和视频采集模块的原理和过程类似，这里不再重复说明，采集模块程序定义了两个类：CAudioDevices 和 CVideoDevices，分别用于维护音频采集设备和视频采集设备的列表。

5.2.2　视频显示模块的实现

上一节已经阐述了采集模块的实现过程，完成了采集应用 Filter Graph 的构造，由于我们采用的采集 Filter 只有一个捕获输出 Pin（如图 5-1 中所示），而采集 Filter 输出的视频数据流既要显示，又要压缩后存储或发送给客户端，因此必须在它后面连接一个 Smart Tee Filter，将数据流分成两路，一路用于预览，一路用于压缩后存储或进行网络发送。

Smart Tee Filter 的 GUDI 是 CLSD_SmartTee，调用 CoCreateInstance 函数创建一个 Smart Tee Filter，再用 IGraphBuilder 的 AddFilter 函数将它加入到 Filter Graph 中，最后将采集 Filter 的 Capture Pin 与 Smart Tee Filter 的 Input Pin 连接起来，Smart Tee Filter 得 Preview Pin 输出的数据用于显示图像，Capture Pin 输出数据用于视频压缩。Smart Tee Filter 如图 5-2 所示：

图 5-2　Smart Tee Filter

因此，实现视频显示模块的预览过滤器链路如图 5-3 所示：

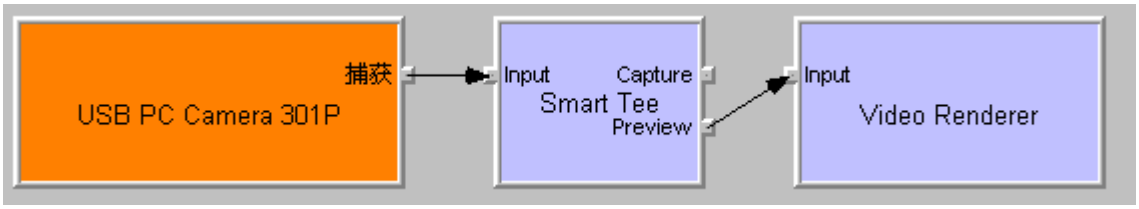

图 5-3　实现视频预览的 Filter Graph

Fig.5-3　Filter graph of video preview

5.2.3　视频压缩模块的实现

5.2.3.1 视频压缩标准

存储和网络传输是网络监控系统的基本功能，未经压缩的数字化图像和声音信号数据是非常庞大的。所以在视频监控系统中，必须对数字化多媒体信息进行压缩，用尽可能少的数据来表达信息，节省传输和存储的开销。

视频的压缩分帧内(Intraframe)压缩和帧间((Interframe)压缩。帧内压缩也称空间压缩((Spatial compression)，不考虑相邻帧之间的冗余信息，与静态图像压缩类似。帧间压缩是基于视频前后两帧之间具有很大的相关性，也称空间压缩(Temporal compression)，仅记录帧间的运动差值，可以大大减少数据量。帧间压缩也提高了视频编码对差错率的要求。

在编码标准方面，国际电信联盟 ITU 先后制定了 H.261,H.262,H.26L 等标准，国际标准化组织 ISO/IEC 先后制定了 MPEG-1，MPEG-2，MPEG-4 等标准。这些标准都是当前工业界和研究机构广泛使用并不断发展的标准系列。MPEG-4 标准的目标与以往 MPEG-1/2 标准有了很大的不同，应用前景也更为广阔。MPEG-4 的主要目标是：1、基于对象的压缩标准；2、具有可交互性；3、码率的带宽范围适应性(5k-l0Mbps)。其传输速率较低，在 4800-64000bps 之间，可以利用很窄的带宽，通过帧重建技术，压缩和传输数据，以求以最少的数据获得最佳的图像质量。这些特点使 MPEG-4 压缩后的视频更适合 Internet 上流媒体的传输要求[25]。

5.2.3.2 视频压缩方法的软件实现

在 DirectShow 中，集成了各种标准的编码解码器，它们都是以过滤器的形式存在，使用方法也和一般的过滤器一样，因此，实现视频图像的编码和解码只要在网络发送端过滤器的输入 Pin 和网络接收端过滤器的输出 Pin 进行媒体类型的定义，使之与所指定的媒体结构类型相匹配，即可实现编码解码功能。编码后的数据需要经过 RTP 打包进行流化处理后才能够成为适合流式传输的文件。

XviD MPEG-4 Codec 是以用 MPEG-4 技术的数字视频编解码器（Codec），本系统采用它对视频数据进行压缩，在客户端则采用 XviD MPEG-4 Decoder 对视频数据进行解压缩。两个编解码器的创建和使用方法基本相同。

创建 XviD MPEG-4 Codec Filter 同样要用枚举实现，枚举的方法和采集设备的枚举方法一直。视频压缩 Filter 一般存放在 Video Compressor 目录下，GUID 为 CLSID_VideoCompressorCategory，采用枚举的方法找到目录下的 XviD MPEG-4 Codec（安装了 DirectX 9.0 的计算机都有这个编码器），将其绑定为一个 Filter 后加入到 Filter Graph 中。XviD MPEG-4 Codec 有一个输入 Pin 和一个输出 Pin，需要将 5.2.2 中所创建的 Smart Tee Filter 的 Capture Pin 与 XviD MPEG-4 Codec Filter 的 Input Pin 连接起来。XviD MPEG-4 Codec Filter 如图 5-4 所示。程序中媒体类型的定义是在网络发送端 Filter 的输入 Pin 和网络接收端 Filter 的输出 Pin 的 GetMediaType()和 CheckMediaType()中进行设置的，此外还要对数据的 MEDIATYPE 和 MEDIASUBTYPE 进行设置。

图 5-4　视频压缩 Filter

Fig.5-4　Video compress filter

5.2.4　音频压缩模块的实现

5.2.4.1 音频压缩标准

对音频的编码主要目标就是保留原始音频的频率范围和其中变化的部分。如果带宽足够，这个目标容易满足。但在网络带宽条件有限的情况下，就必须对某些部分进行适当地折损。音频信号可分为电话质量的语音、调幅广播质量的音频信号和高保真立体声信号。语音信号的频率范围在 300Hz~3.4kHz。调幅广播质量音频信号的频率范围是 50Hz~7kHz，而高保真信号的频率范围是 10Hz~20kHz。随着频率带宽的增加，信号的真实度逐步改善，针对不同的音频信号，制定了相应的压缩标准。

目前比较流行的立体声音频压缩标准为“ MPEG 音频 ”。MPEG 是 ISO 提出的动态图像编码的国际标准，“ MPEG 音频 ” 是该标准的一部分。ISO/MPEG 音频压缩标准里包括了三个使用高性能音频数据压缩方法的感知方案。按照压缩质量和编码方案的复杂成都分为三个层次。目前最为流行的立体声音频压缩格式是 MP3，即 MPEG 音频第三层（MPEG Audio Layer-3）。MP3 的优点在于大幅降低数字声音文件的容量，而不是破坏原来的音质。MP3 音频的音质较好，采样率不高，完全适合在网络环境下传送。因此本文设计的音频传输链路中采用 MP3 为音频编码方式[26]。

5.2.4.2 音频压缩方法的软件实现

在 DirectShow 中集成了 MP3 编码、解码器过滤器，创建的方法和视频编解码 Filter 相同，通过 DirectShow 提供的过滤器接口直接选择集成好的编解码过滤器，并直接选择加入 Filter Graph 即可，不用专门编写压缩程序，但必须在网络发送过滤器的输入 Pin 和网络接收过滤器的输出 Pin 的 GetMediaType()和 CheckMediaType()中对音频格式进行设定，并将媒体类型的 MEDIASUBTYPE 和 MEDIATYPE 设置为 MEDIASUBTYPE_MPEGAudio 和 MEDIATYPE_Stream，以匹配与其直接连接的编解码器。MP3 编码器 Filter 如图 5-5 所示。

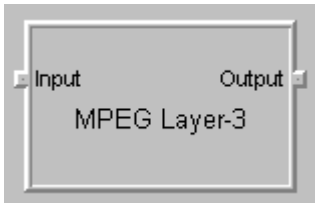

图 5-5　音频压缩 Filter

Fig.5-5　Audio compress filter

5.2.5　数据存储模块

视音频数据实现压缩后便可以保存起来，或是通过网络发送出去，这里我们主要实现视音频数据的存储，即将视音频内容采集到文件中保存起来。我们将其保存为 AVI 文件，DirectShow 提供了一个 AVI Mux Filter（如图 5-6 所示），视频输出 Pin 和音频输出 Pin 分别连接它的 Input Pin，即可以实现视频和音频数据的合成，然后调用 DirectShow

提供的 File write Filter（如图 5-7 所示），将合成的数据以文件的形式保存，这里我们将其保存在 sample.avi 文件中。整个服务器端实现过程的 Filter Graph 如图 5-8 所示。

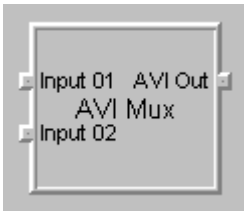

图 5-6　AVI Mux Filter

图 5-7　File write Filter

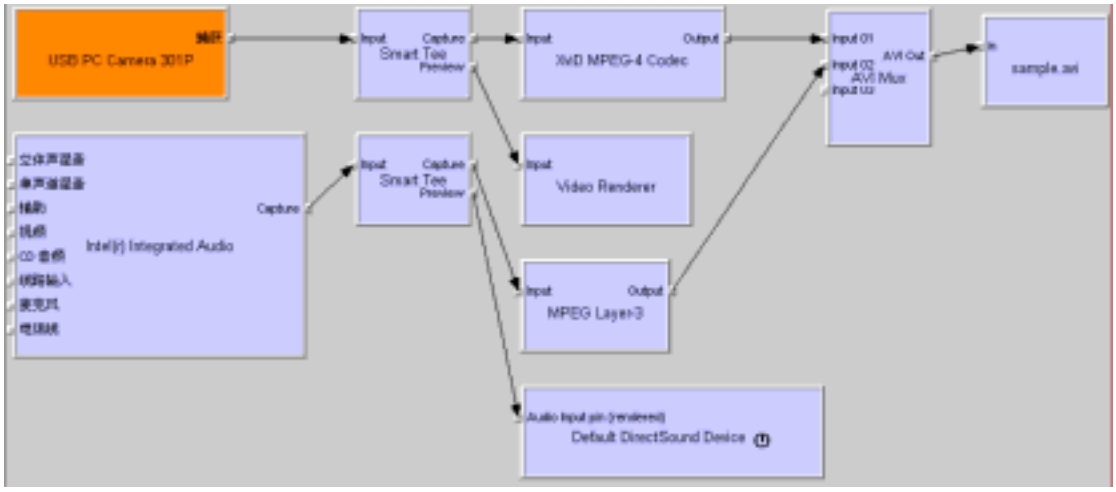

图 5-8　服务器端实现过程的 Filter Graph

Fig.5-8　Filter graph of server achieve process

5.3　网络传输的实现

服务器端的数据发送功能通过开发一个网络发送 Filter 来实现，命名为 NetSend Filter,它仅有输入 Pin，用于从上一级 Filter 获取 Sample 数据，在得到数据后对其进行封装，再向网络远程端发送。所以 NetSend Filter 是一个 Renderer Filter。

客户端的数据接收功能通过开发一个网络接收 Filter 来实现，命名为 NetReceive Filter，它的实现和 NetSend Filter 有些类似，都是一种应用程序内的 Filter 形式。但是 NetReceive Filter 的具体实现要比 NetSend Filter 复杂一点，NetReceive Filter 需要使用独立的线程进行网络数据的接收，当接收到第一个媒体格式数据后，必须马上通知上层应用程序完成 Filter Graph 的构建，然后继续接收媒体数据，并将这些数据以 Sample 的形式“ 推 ”给下一级 Filter。

5.3.1　服务器端网络发送的实现

5.3.1.1 网络发送 Filter 内部数据的流动

在 DirectShow 中的 Graph 内的 Filter 之间，数据的交换都是通过 MediaSample 来实现的。MediaSample 是 DirectShow 媒体数据的基本单位。Graph 中的媒体数据流动过程也就是 Filter 之间 MediaSample 的传递过程。NetSender Filter 属于 Renderer Filter，它只有一个 Input Pin，没有 Output Pin。它通过 Input Pin 从上方的 Filter 接收 MediaSample, 然后在 Filter 内对 MediaSample 进行分片，RTP 封装，然后通过 Net Sender 成员对象利用 RTP 打包发送。数据流动如图 5-9 所示

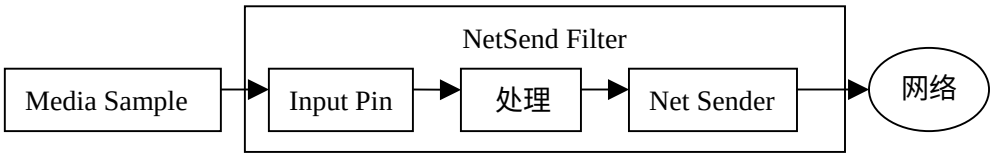

图 5-9　网络发送 Filter 内部数据流动

Fig.5-9　The data flow of Network Send Filter

5.3.1.2 网络发送 Filter 的设计

网络发送 Filter 的功能是接收来自上游的压缩视音频流，检查数据类型；对视音频流分别打包形成 RTP 包，发送至网络客户端。这些功能的实现主要依赖于三个类。首先是 CNetSendFilter 类，它是一个典型的 Renderer Filter，这个类是实现功能的主体框架类，它的最重要的一个父类是 CBaseFilter 基类，该类由 DirectShow SDK 提供，是为自行开发 Filter 而提供的 DirectShow 基类。CBaseFilter 是所有 Filter 的基类。而主类中的两个类的实例用以实现具体功能包括检查数据，打包，发送等。第二个类是 CNetSendInputPin，它主要作用是接收上游的压缩数据流，检查数据类型。系统提供了一个类 CBaseInputPin 帮助实现该类。第三个类是 CNetSender 类，该类负责对 MediaSample 的 RTP 打包和 UDP 发送。发送 Filter 类图如图 5-10 所示。

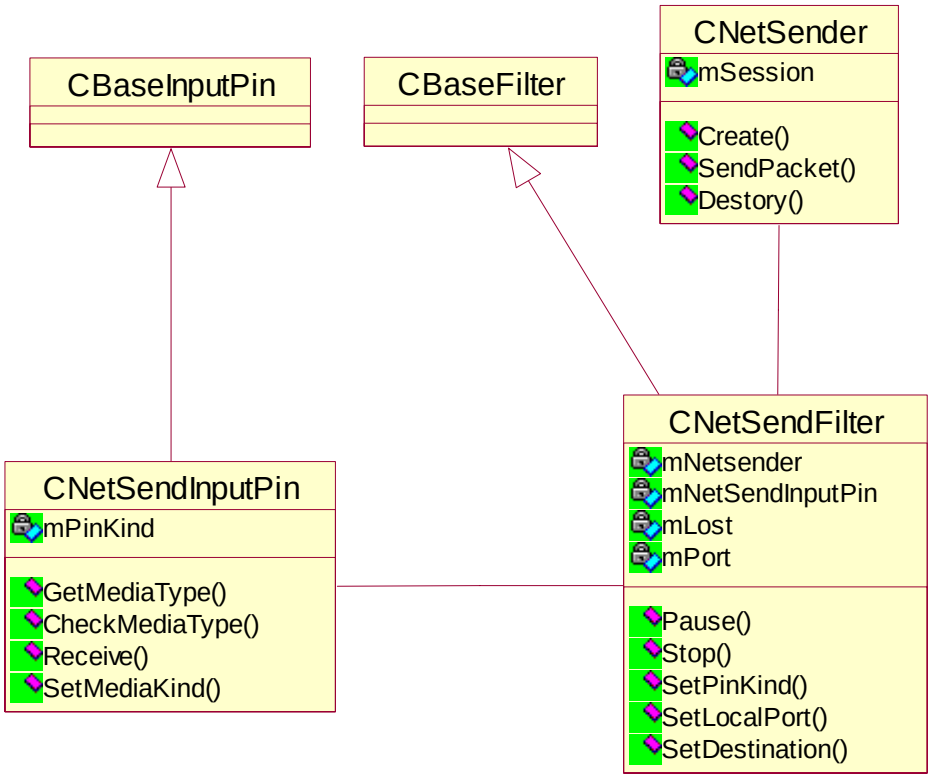

图 5-10　网络发送 Filter 类图

Fig.5-10　Class diagram of web transmit filter

下面对照图来分别就各个类的设计进行简单的阐述：

（1）CNetSendInputPin 的设计

CNetSendInputPin 需要完成的主要功能有两个：一是实现 Filter 之间所传输的媒体类型的协商，而是从上方的 Filter 接收 MediaSample。CNetSendInputPin 类主要实现的函数有四个：

GetMediaType 函数，用于获得该 Pin 的媒体类型是音频还是视频；

CheckMediaType 函数，该 Pin 连接过程中对连接用的媒体类型进行检查，仅接收 GetMediaType 获得的媒体类型。

Receive 函数，上级 Filter 通过该函数将数据传入，在这个函数中进行一定的 MediaSample 的选择工作，然后通过委托 Filter 上的 mNetSender 成员变量的 SendPacket 函数进行数据的打包发送。

SetMediaKind 函数，设定该 Pin 的媒体类型是音频还是视频。

主要的一个变量 mPinKind 用来存放媒体类型。NetSend Filter 通过 SetPindKind 函数来设置该值。

（2）CNetSender 类的设计

CNetSender 类用于完成基本的网络传输功能，主要是对接收到的媒体 Sample 进行 RTP 的打包和发送。CNetSender 的主要函数有：

Create 函数：主要作用是建立 RTP Session，它有两个参数。前一个参数指定了传输的一系列参数，主要有 RTCP 包的发送频率，RTP 包的缺省标记等。第二个参数指定了传输的协议，类中会根据它创建 Udpsocket 或者其他 Socket。

SendPacket 函数：发送数据。具体的发送动作调用的是 RTPSession 的 SendPacket 函数。

Destroy 函数：销毁一个 RTPSession。

CNetSender 变量 mSession 是一个 RTPSession 类的实例。

（3）CNetSendFilter 的设计

CNetSendFilter 类是网络发送 Filter 的主类，它的主要作用实际上就是上述两个类的组合，包括接收上游的 Sample，并将其打包发送。除此之外还包括对整个过程的总体控制。CNetSendFilter 类的继承来自 CBaseFilter 类，所以还包括 Pause 和 Stop 两个函数，由于 Graph 和 Filter 开始运行时，在 run 前必须经历 pause 状态，所以在 pause 中来创建一个 RTP 的 Session，从而启动相应的接收 RTP 的 RR 包的线程。它的函数主要有以下几个：

Pause 函数：用于在 Filter 从停止状态转到暂停状态是创建一个用于网络发送数据的 RTP 的 Session 同时指定其目标的 IP 地址和端口号，并启动接收 RTCP 的线程。通过 CNetSender 的 SetDestination 函数具体实现。

Stop 函数：Filter 转入停止状态时，通过 CNetSender::Destory 来结束 RTP 的会话。

SetPinKind 函数：设置该 Filter 的类型。具体实现要调用 InputPin 中的 SetMediaKind 函数。

SetLocalPort 函数：设置该 Filter 发送 Sample 时的所用的本地监听端口号。

SetDestination 函数：其租用是设置远程地址和端口号。

5.3.1.3 网络发送 Filter 的实现

（1）网络发送 Input Pin 的关键函数有：

- HRESULT CheckMediaType()：用于检查当前传输数据的数据格式，可以通过此函数限制某些格式数据的传输。它有一个参数，是一个媒体类型对象的指针。函数将依次将 Pin 所支持的媒体类型，与该媒体对象作比较，如果匹配的话，则成功返回，如果没有一个匹配的，则返回错误。该函数实现代码如下：

```
HRESULT CNetSendInputPin::CheckMediaType (const CMediaType * inMediaType)
{
    if (inMediaType->formattype == FORMAT_VideoInfo ||
        inMediaType->formattype == FORMAT_WaveFormatEx)
    {
```

```
            return S_OK;
        }
        return E_FAIL;
    }
```

- STDMETHODIMP Receive(IMediaSample *pSample) :该函数用于从上方的 Filter 接收 MediaSample。它有一个参数是一个 MediaSample 的指针。它的基本流程是：输入 Pin 与上一级 Filter 的输出 Pin 连接后，数据流会以 IMediaSample 结构一帧一帧传递过来。调用 IMediaSample::GetPoint() 和 IMediaSample::GetActuallDataLength()两个函数分别得到这一帧的有效负载数据和负载长度,最后调用 NetSend Filter 的 SendPacket 函数进行打包处理后发送到网络上。该函数实现代码如下：

```
STDMETHODIMP CNetSendInputPin::Receive (IMediaSample *pSample)
{
    CAutoLock lck (&mReceiveLock);
    HRESULT hr = CRenderedInputPin::Receive (pSample);
    if (SUCCEEDED(hr))
    {
        //判断 Sample 装的是否是媒体数据
        if (m_SampleProps.dwStreamId == AM_STREAM_MEDIA)
        {
            PBYTE pData;
            //从 Sample 中得到数据指针，以及有效数据长度
            pSample->GetPointer(&pData);
            long length = pSample->GetActualDataLength();
            int    bytes    = 0;
            //调用 SendPacket 函数
            ……
        }
    }
    return hr;
}
```

（2）网络发送 Filter 最主要的函数 SendPacket 的实现

网络发送 Filter 最主要的功能就是把接收到的 MediaSample 打包发送，而这功能由 Filter 的 SendPacket 函数来完成，其实最终就是由 CNetSender::SendPacket 来完成。。首先 NetSend Filter 的输入 Pin 接收经编码压缩后的音频、视频媒体实例（Sample），然后调用 CheckMediaType()函数，进行媒体类型的检查，根据不同的媒体类型进行分片和 RTP 封装之后，利用 NetSender 对象的 SendPacket 函数发送到网络。其发送基本流程如图 5-11 所示：

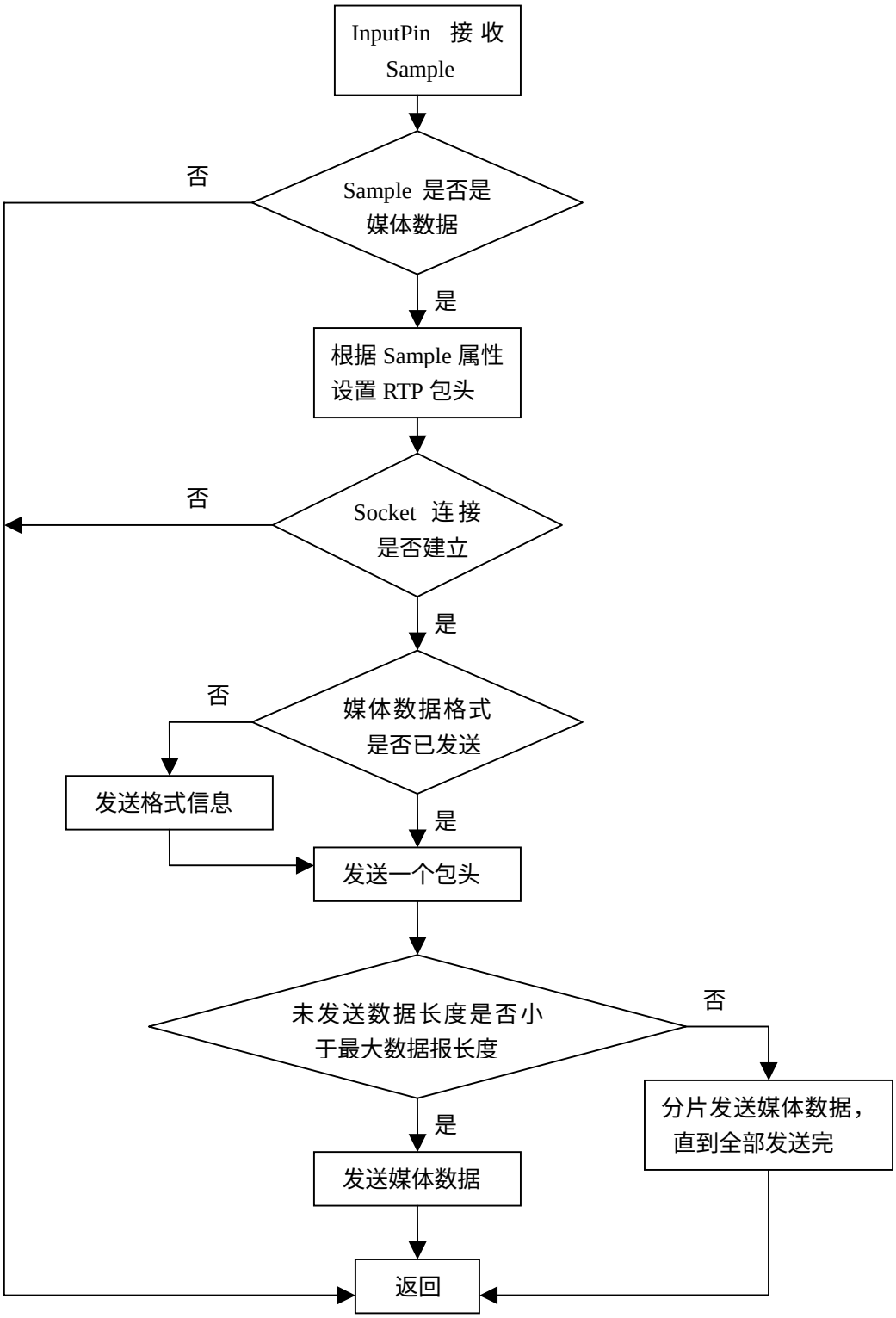

图 5-11　发送 Filter 的数据发送流程图

Fig.5-11　Flow chart of network send filter

我们把要发送的媒体数据先分片后发送，最大的报文长度可以根据用户的需求和网络环境来定，本系统设定的最大包产故未 1400bytes，所有大于 1400bytes 的包都采取分片发送，所有分片的包都打上相同的时间戳，在接收端碰到相同的时间戳的 RTP 包时进行重组，对最后一个包会打上 Mark 标记作为表示。系统中视频和音频数据采用不同的端口进行发送，但是视频和音频是同在一个 Graph 内处理的，都采用相同的时间戳，以保证他们在接收端能同步播放。

（3）RTP 传输的实现

目前国外有很多组织或者个人，提供了一些开源的 RTP 栈，如：GUNccRTP、RTPlib、librtp、ortp 和 JRTPLIB[35]等。本系统采用的是 JRTPLIB 来实现 RTP 传输。JRTPLIB 是一个用 C++语言实现 RTP 的开发源码库，目前已经可以运行在 Windows、Linux、FreeBSD、Solaris、Unix 和 Vxworks 等多种操作系统上。用户可通过编译源码中提供的库函数来实现 RTP/RTCP 的网络传输。它提供了若干个建立 RTP 应用的相关类，我们可以很方便的根据库提供相关类的方法建立 RTP 会话，发送 RTP 包和 RTCP 控制包，而不用担心 SSRC 的冲突。

具体实现过程如下：

1）初始化

在使用 JRTPLIB 进行实时流媒体数据传输之前，首先需要对 RTP 会话进行初始化，生成一个 RTPSession 类的实例，然后调用 Create()函数，来实现初始化操作。但此时函数并未制定 RTP/RTCP 的发送目标地址，由接下来的步骤完成。初始化实现的代码如下：

RTPSession session；

session.Create()；

如果 RTP 会话创建失败，Create()将会返回一个负数。JRTPLIB 采用了统一的错误处理机制，它提供的所有函数如果返回负数就表明出现了某种形式的错误，而具体的出错信息则可以通过调用 RTPGetErrorString()函数得到。RTPGetErrorString()函数将错误代码作为参数传入，然后返回该错误代码所对应的错误信息。

2）缺省参数设置

在 RTPSeesion 类中有一些函数用于缺省参数的设置，一般包括负载类型、标识、时间戳等。负载类型由 RTP 传输的数据类型决定，在具体实现是，可以采用缺省的方式。时间戳是 RTP 会话初始化过程所要进行的另一项重要工作，本系统通过调用的 SetTimeStampUnit()方法来实现的。

session.SetDefaultPayloadType();//负载类型设置

session.SetDefaultMark;//M 标识

seesion.SetDaulutTimeStampIncrement();//时间戳

3）建立连接

当 RTP 会话建立成功后，接下去就可以开始进行流媒体数据的实时传输了。首先获

取客户端的目标地址，RTP 协议允许同以会话存在多个目标地址，具体实现可通过调用 RTPSession 类的 AddDestination()来获取用户端的地址和端口号等信息。例如：

```
RTPIPv4Address rtpAddr (intIP, MACAST_PORT);
session.AddDestination (rtpAddr);
```

其中 RTPIPv4Adress 类的成员函数可以用来获取 IP 地址及端口好。通过以上步骤后，服务器端就可以向客户端发送流媒体数据了。

4）数据发送

通过调用 RPTSession 类的 SendPacket()方法，可以向客户端发送需要的数据。实现代码如下：

```
session.SendPacket ();
```

其中 SendPacket()是一个重载函数，它具有如下形式：

```
int SendPacket(void *data,int len)
int SendPacket(void *data,int len,unsigned char pt,bool mark, unsigned long
                timestampinc)
```

第一个参数 data 是载荷，第二个参数是指明载荷的长度，再往后依次是 RTP 负载类型、标识和时间戳量。这些参数通过刚才介绍的 SetDefaultPayloadType()、SetDefaultMark()和 SetDefaultTimeStampIncrement()来获取。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，对于同一个 RTP 会话来讲，负载类型、标识和时间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。如果用于发送的流媒体数据从缓存区读出，则调用函数：

```
seesion.SendPacket (buffer, bufsize)
```

5）控制信息

JRTPLIB 是一个高度封装后的 RTP 库，我们在使用它时很多时候并不用关心 RTCP 数据是如何被发送和接收的，因为这些都是可以由 JRTPLIB 自己来完成的。只要会话建立和数据发送成功，JRTPLIB 就能够自动对到达的 RTCP 数据报进行处理，并且还会在需要的时候发送 RTCP 数据报，从而能够确保整个 RTP 会话过程的正确性。

6）会话结束

当客户端完成流媒体数据的接收或者服务器要结束对客户端数据的发送，通过 RTPSession 类的 Destroy()就可以结束对话了。

```
session.Destroy ();
```

这便是实现网络传输的主流程，核心代码如下：

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

void checkerror(int err);
int main(int argc,char **argv)
{
        RTPSession session;
```

```
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=session.Create(portbase);
 checkerror(status);

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

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

//发送流媒体数据
index=1;
do{
       sprintf(buffer,"%d:RTP packet",index++);
       session.SendPacket(buffer,strlen(buffer));
       printf("Send packet!\\n");
}while(1);
     return 0;
```

5.3.2　客户端网络接收的实现

5.3.2.1 网络接收 Filter 内部数据的流动

NetReceive Filter 与 NetSend Filter 之间最大的区别就是 NetReceive Filter 需要自己实现 MediaSample 的管理。这是由于这两个 Filter 采用的是 PUSH 的数据流动方式。在这种数据流动方式中，都是由上方的 Filter 提供 Allocator，并进行 MediaSample 的管理。而 NetSend Filter 属于 Renderer Filter，它的下方已经没有 Filter 了，因此不需要提供 Allocator，也不需要管理 MediaSample。而 NetReceive Filter 通过 SamplePool 实现类似的功能。NetReceive Filter 的功能就是将网络上接收到的 RTP 包解包，对于同一个 Sample 的不同的包要进行重组，因此必须开辟一个缓存区去对这些包进行缓存。重组好 sample 后以“推”的模式传递到下一个 Filter。网络接收 Filter 内部数据流动如图 5-12 所示。

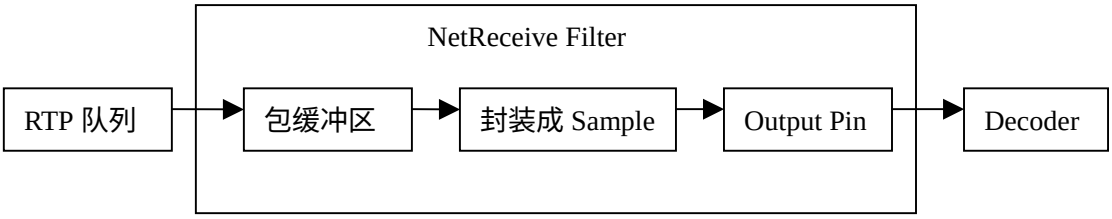

图 5-12　网络接收 Filter 内部的数据流动

Fig.5-12　Date flow in network receive filter

5.3.2.2 网络接收 Filter 的设计

网络接收 Filter 的主要功能是接收网络的媒体数据，经过 RTP 的解包，将数据以 MediaSample 的形式，传递给下方的 Filter。它位于 Graph 的起点，属于 Source Filter。这个 Filter 的功能实现主要通过三个类来实现。主要包括 CNetReceiveFilter 类，CNetReceiver 类和 CNetReceiveOutputPin 类。其中主类是 CNetReceiveFilter 类，它负责整个流程的控制，其父类是 CBaseFilter。CNetReceiver 类对应于发送端的 CNetSender，它的主要功能是实现网络数据的接收，然后以 Sample 的形式“推”给下一级 Filter。CNetReceiveOutputPin 类的主要作用是和下级的 Filter 进行媒体类型的协商，它的父类是 CBaseOutputPin。接收 Filter 的类图如图 5-13 所示。

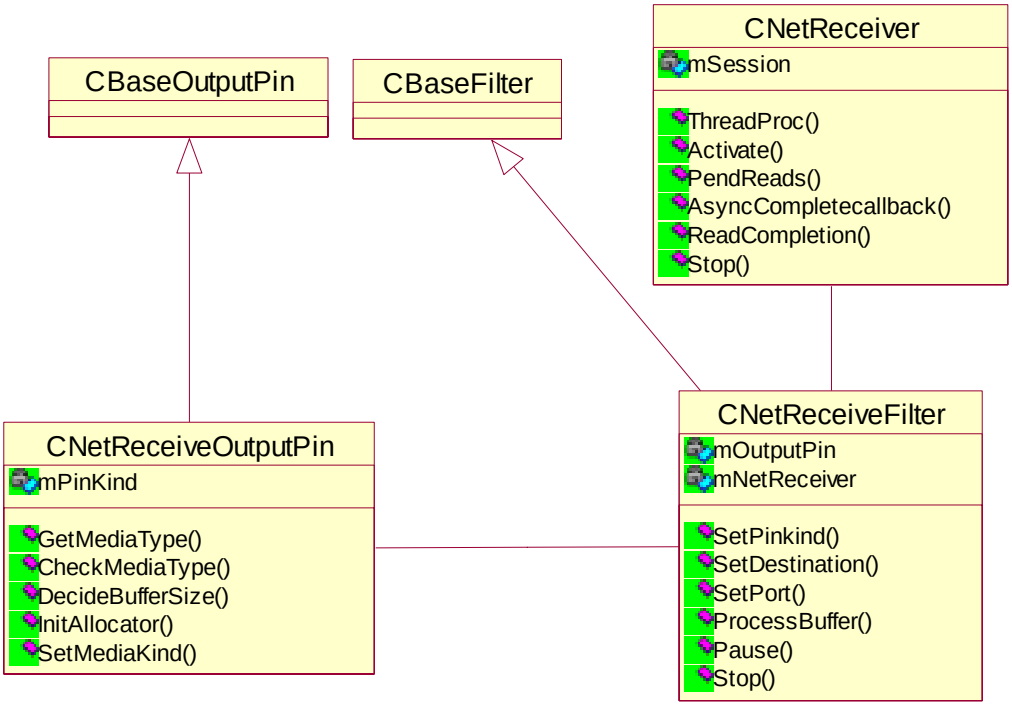

图 5-13　网络接收 Filter 类图

Fig.5-13　Class diagram of network receive filter

下面对照主要类的类图对类的设计进行简单的阐述：

（1）CNetReceiveOutputPin 类的设计

继承了 CBaseOutputPin 基类，主要的功能是与下方的 Filter 进行媒体类型的协商，还有就是向下方传递 MediaSample。所以它主要实现的函数就是两个：GetMdeiaType 和 CheckMediaType。这两个函数的作用和发送端 CNetSendInputPin 的两个函数的功能是一样的，实现也基本类似。

向下方传递 MediaSample 的功能有另外的两个函数 InitAllocator 和 DecideBufferSize 来实现。OutputPin 向下方传递传递 MediaSample，本质就是向下方 Filter 提供管理 MediaSample 访问的 Allocator（管理器）。InitAllocator 函数的作用就是获取创建完毕的 NetRecevAllocator 类的指针，实际的创建工作是由 Filter 对象来创建的。DecideBufferSize 函数的作用是决定该 Pin 使用的 Sample 管理器 NetRecvAllocator 的属性，如 Sample 内存大小、Sample 的个数等等。

（2）CNetReceiver 类的设计

CNetReceiver 的作用和 CNetSender 正好相反，它负责网络数据的接收，并将数据封装成 Sample，发到下一级 Filter 中。但是比起发送来，接收端的处理要复杂一些，它是一个异步接收过程。

Activate 函数，该函数作用是创建一个 RTP 的 Session，并启动控制线程。这个控制

线程就是一个网络数据接收然后封装为 Sample 并发送到下一级 Filter 的反复过程。如果线程启动失败，则销毁会话 Session。

Threadproc 函数，该函数是 NetRceciver 接收过程的主体，它没有参数和返回值。该函数是一个无限循环，它的主要流程是不断调用 WaitForMultipleObjectsEX 等待时间发生；如果接收到 Stop 时间，就跳出循环；如果超时，就调用 PendReads 函数获取 RTP 对列中的 RTP 包。

PendReads 函数，该函数的作用是将 RTP 队列中的同一个 Sample 的包接收并存放在一个 CBuffer 类中，记录下它的时间戳。可能同时有多个线程正在运行该函数，因此要加 Lock。

AsyncCompletionCallBack 函数，该函数是当前 PendReads 函数完毕时时的回调过程。它的主要作用是在每次接受完网络数据到 CBuffer 后，设置实际得到的 CBuffer 的长度，然后调用 ReadCompletion 函数完成最后的“推”的工作。

ReadCompletion 函数，该函数是一个网络数据接收请求完成的最后调用的函数，它将接收到的 Sample 的形式“推”给下一级 Filter 。具体的执行过程是：首先调用 PendReads 函数，进行新的接收，然后将 AsynCompleteCallBack 中传递的要求处理的 buffer 的指针交给 Filter。由 Filter 的 ProcessBuffer 函数进行实际的“推”处理。

Stop 函数，该函数主要作用是发出 EVENT_STOP 消息，以结束接收网络数据和传递的线程，并销毁 RTPSession。

（3）CNetReceiveFilter 类的设计

CNetReceiveFilter 与 CNetSendFiter 相同，都继承自 CBaseFilter 基类。CNetReceiveFilte 所需要重载的函数和 CNetSendFilter 相同，这里不再多做描述。它另外需要实现的一个重要成员函数是 ProcessBuffer，该函数的功能将接收到的网络数据以 Sample 的形式“推”给下一级 Filter。主要流程是从 MediaSample 池中去取出一个空闲的 MediaSample，并用这个 MediaSample 来包装进来的 CBuffer 类型的缓存对象，包括指定实际大小，打上时间戳等操作，然后调用 OutputPin 的 Deliver 函数，将这个 MediaSample 传递给下一级的 Filter，最后释放这个 MediaSample。

除了上面几个主要的类之外，为了完成 Filter 的网络接收和“推”数据到下一级的 Filter，程序自定义了几个工具类，如下：

CBuffer 自定义缓存类，封装了一定大小的内存，用于一块数据的读写，可以放到缓存池中使用。

CBufferPool 缓存池类，维护了一个包含一定数量的 CBuffer 类型的缓存对象的双向列表。这个列表中的每一个缓存对象都能被重用；外部调用者需要缓存时，从列表中取出一个空闲的缓存对象，然后使用，使用完之后，将该缓存对象收回列表中，等待下一次使用。

CMediaSample 类，因为 DirectShow 中以 Sample 的形式来传递数据，所以要定义这

个类来包装 CBuffer 类型的缓存对象，继承自 IMediaSample2。

CSamplePool 类，维护一个包含一定数量的 CMediaSample 类型的双向列表。列表中的每一个 Sample 对象都是可以重用的。

CNetRecvAlloc 类，继承自 IMemAllocator，主要功能就是实现对 Sample 的管理，它是 DirectShow 的标准设计，由于 CSamplePool 中已经实现 Sample 的管理功能，所以主要是调用 CSamplePool 进行实现。

5.3.2.3 网络接收 Filter 的实现

（1）Net Receive Output Pin 上几个重要函数的实现：

- SetupMediaType（）：当 Socket 上接收到视频格式数据后，重建输出 Pin 上使用的媒体类型，输出 Pin 必须进行格式设置才能正常运行。该函数实现代码如下：

```
void CNetReceiveOutputPin::SetupMediaType(long inType, char * inFormat, long inLength)
{
    if (inType == PT_VideoMediaType)
    {
        mPreferredMt.SetType (&MEDIATYPE_Video);
        mPreferredMt.SetFormatType (&FORMAT_VideoInfo);
        // 判断得出媒体类型的子类型
        VIDEOINFOHEADER * pvi = (VIDEOINFOHEADER *) inFormat;
        const GUID subtype = GetBitmapSubtype(&pvi->bmiHeader);
        mPreferredMt.SetSubtype (&subtype);
        //修正图像帧大小
        pvi->bmiHeader.biSizeImage =pvi->bmiHeader.biWidth*
              pvi->bmiHeader.biHeight*pvi->bmiHeader.bitBitCount/8;
        //设置格式数据
        mPreferredMt.SetFormat ((BYTE*) inFormat, inLength);
        mPreferredMt.SetSampleSize (pvi->bmiHeader.biSizeImage);
    }
    else   //音频格式
    {
        mPreferredMt.SetType (&MEDIATYPE_Audio);
        mPreferredMt.SetSubtype (&MEDIASUBTYPE_PCM);
        mPreferredMt.SetFormatType (&FORMAT_WaveFormatEx);
        // 设置格式数据
        WAVEFORMATEX * wave = (WAVEFORMATEX *) inFormat;
        wave->nAvgBytesPerSec = wave->nSamplesPerSec * wave->nChannels
            * wave->wBitsPerSample / 8;
        mPreferredMt.SetFormat ((BYTE*) inFormat, inLength);
        mFilter->SetAudioBytesPerSecond(wave->nAvgBytesPerSec);
    }
    mPreferredMt.SetTemporalCompression (FALSE);
}
```

- CheckMediaType()：用于输出 Pin 连接过程中媒体类型检查；实现代码如下：

```
HRESULT CNetReceiveOutputPin::CheckMediaType (const CMediaType * inMediaType)
{
    if (*inMediaType == mPreferredMt)
    {
        return NOERROR;
    }
    return E_FAIL;
}
```

- DecideBufferSize()：决定输出 Pin 上的 Sample 属性，实现代码如下：

```
HRESULT CNetReceiveOutputPin::DecideBufferSize (IMemAllocator * pAlloc, ALLOCATOR_PROPERTIES * pprop)
{
    ASSERT (pAlloc);
    ASSERT (pprop);
    HRESULT hr = NOERROR;

    // 决定 Sample 的内存大小
    if (mPreferredMt.formattype == FORMAT_VideoInfo)
    {
        //如果是视频数据，则 Sample 内存大小取值为图像帧大小
        VIDEOINFOHEADER * info = (VIDEOINFOHEADER *) mPreferredMt.pbFormat;
        pprop->cbBuffer = info->bmiHeader.biSizeImage;
    }
    else
    {
        //如果是音频数据，则 Sample 内存大小取值为 1 秒钟的数据量大小
        WAVEFORMATEX * info = (WAVEFORMATEX *) mPreferredMt.pbFormat;
        pprop->cbBuffer = info->nAvgBytesPerSec;
    }
    pprop->cBuffers  = 1;//使用 1 个 Sample
    pprop->cbAlign   = 1;//Sample 数据按 1 字节对齐

    ASSERT(pprop->cbBuffer);

    ALLOCATOR_PROPERTIES Actual;
    hr = pAlloc->SetProperties(pprop, &Actual);
    if (FAILED(hr))
    {
        return hr;
    }
```

```
    ASSERT (Actual.cBuffers == 1);

    if (pprop->cBuffers > Actual.cBuffers ||
        pprop->cbBuffer > Actual.cbBuffer)
    {
        return E_FAIL;
    }
    return NOERROR;
}
```

- GetMediaType()：得出输出 Pin 上推荐使用的媒体类型，实现代码如下：

```
HRESULT CNetOutPin::GetMediaType (int iPosition, CMediaType *pMediaType)
{
    if (iPosition == 0)
    {
        *pMediaType = mPreferredMt;
        return NOERROR;
    }
    return E_INVALIDARG;
}
```

其中对 CheckMediaType()函数和 GetMediaType()函数都做了特殊定义，只接收指定媒体类型的解码器的连接。

（2）Net Receiver 上重要函数的实现

- AsyncCompletionCallBack ()：用来得到 Pin 上一个空闲的 Sample 的数据内存地址，具体实现代码如下：

```
BOOL CNetReceiver:: AsyncCompletionCallBack (PBYTE * outBuffer)
{
    //将没有释放的 Sample 先释放掉
    if (mSample)
    {
        mSample->Release();
        mSample = NULL;
    }
    //在输出 Pin 上得到一个新的空闲的 Sample
    HRESULT hr = mOutPin->GetDeliveryBuffer (&mSample,NULL,NULL,0);
    if (mSample)
    {
        //得到 Sample 的数据内存地址
        mSample->GetPointer(outBuffer);
    }
    return SUCCEEDED(hr);
}
```

- ReadCompletion ()函数：将填好的数据的 Sample 发送给下一级 Filter。实现代码如下：

```
BOOL CNetReceiver:: ReadCompletion (long inSampleSize)
{
    if (mSample)
    {
        //设置 Sample 上的属性
        mSample->SetActualDataLength(inSampleSize);
        mSample->SetSyncPoint(TRUE);
        if (mIsVideo)
        {
            // 如果是视频帧，不使用时间戳（以最快速度播放）
            mSample->SetTime(NULL, NULL);
        }
        else
        {
            //以音频数据计算时间戳
            REFERENCE_TIME rtStart = mLastSampleTime;
            mLastSampleTime += (UNITS * inSampleSize / mAudioBytesPerSecond);
            REFERENCE_TIME rtEnd     = mLastSampleTime;
           //给 Sample 打上时间戳
            mSample->SetTime(&rtStart, &rtEnd);
        }
        //将 Sample 传送下去
        HRESULT hr = mOutPin->Deliver (mSample);
        //释放 Sample（让它回到空闲列表中去等候下一次使用）
        mSample->Release();
        mSample = NULL;
        return SUCCEEDED(hr);
    }
    return TRUE;
}
```

（3）Net Receiver Filter 上重要函数的实现

Net Receiver Filter 关键函数有：

- CBasePin *GetPin()：用于返回 Net Receiver Filter 的输出 Pin，在连接各个 Filter 时将会用到，具体代码如下：

```
CBasePin * CFilterNetReceiver::GetPin(int n)
{
    if (n == 0)
    {
        return mOutPin;
    }
```

```
        else
          {
              return NULL;
        }
      }
```

- SetAudioBytesPer()：用于设置音频每秒的数据量，在给音频 Sample 打时间戳的时候用于就是时间戳。实现代码如下：

```
void CFilterNetReceiver::SetAudioBytesPerSecond(long inBytes)
{
    mAudioBytesPerSecond = inBytes;
}
```

- 公共函数 SetupMediaType()：根据从网络上接收到的格式数据重建输出 Pin 上使用媒体类型，实现代码如下：

```
void CNetReceiveFilter::SetupMediaType(long inType, char * inFormat, long inLength)
{
   //判断将要接收的是视频数据还是音频数据
  if (inType == PT_VideoMediaType)
  {
      mIsVideo = TRUE;
  }
  else
  {
      mIsVideo = FALSE;
  }
  //为输出 Pin 重建媒体类型
  mOutputPin->SetupMediaType(inType, inFormat, inLength);
  // At last, notify the controller to build filter graph and running
  Broadcast(msg_MediaTypeReceived, &mIsVideo);
}
```

（4）数据接收过程的实现

网络接收 Filter 需要使用独立的线程进行网络数据的接收，此功能在 Net Receiver 中实现，数据接收流程如图 5-14 所示。

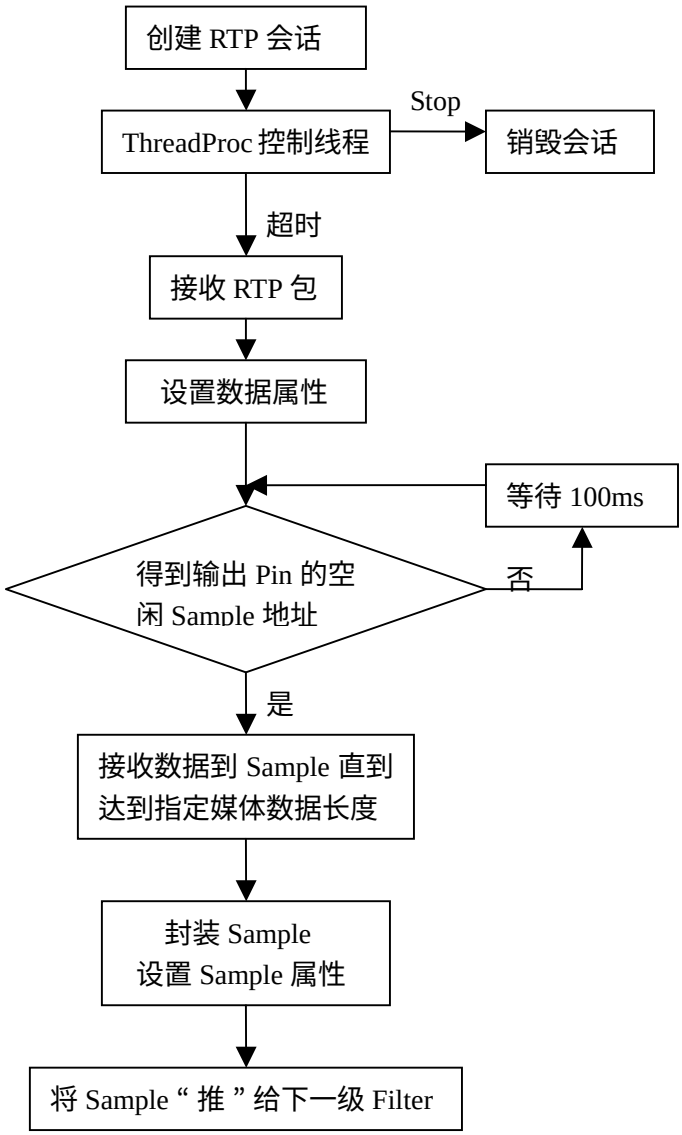

图 5-14　网络接收 Filter 数据接收流程图

Fig.5-14　Flow chart of network receive filter

与服务器端 RTP 数据传输的实现相对应，在客户端的数据接收也是通过调用 JRTPLIB 库的函数来实现。在此具体的网络数据异步接收的功能是在 PendReads()函数中实现，具体的流程图如图 5-15 所示。

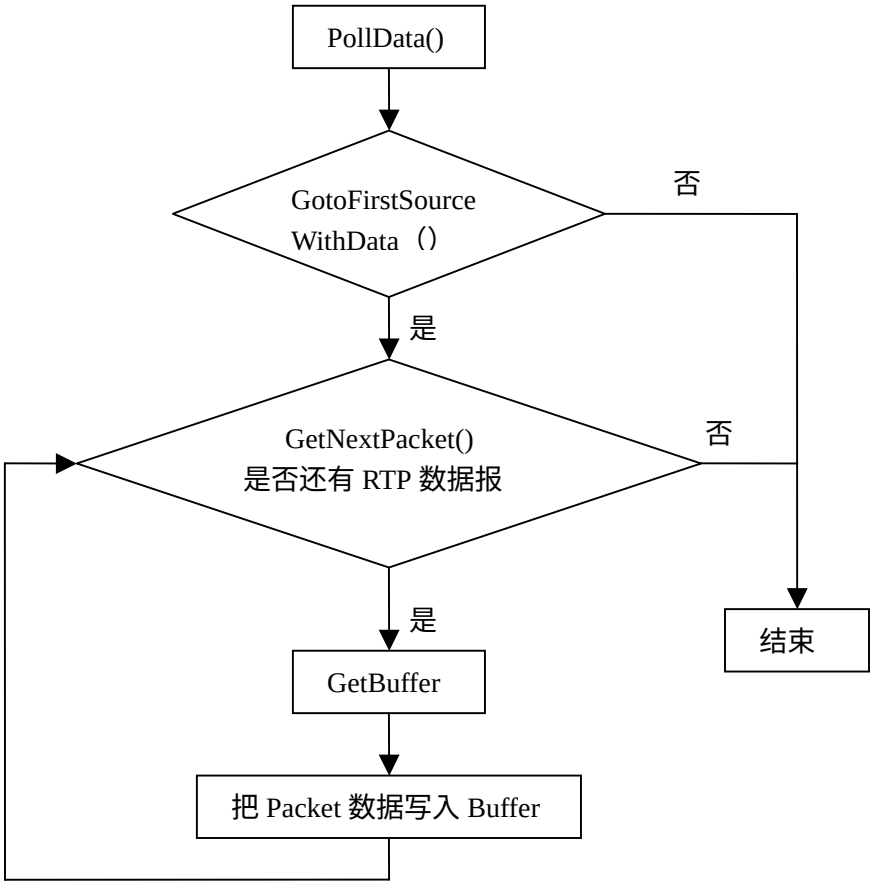

图 5-15　PendReads 函数实现的流程图

Fig.5-15　Flow chart of PendReads function

首先调用 RTPSession 类的 PollData()函数来接受发送过来的 RTP 或者 RTCP 数据报。接下来调用 RTPSession 类的 GotoFirstSourceWithData()和 GotoNextSourceWithData()来判断 RTP 数据源是否到达。从 RTP 会话中检测出有效的数据源之后，调用 RTPSession 类的 GetNextPacket()函数抽取 RTP 数据报，当接收到的 RTP 数据报处理完之后，需要及时释放缓存区空间。对接收到的 RTP 数据报进行处理的过程实现代码如下：

```
session.pollData ();
if(session.GotoFirstSourceWithData())
{     do
      {
        RTPPacket *pack;
        pack=session.GetNextPacket();
        delete pack;
        }
      While (sess.GotoNextSourceWithData());
  }
```

到此，网络发送 Filter 和接收 Fiter 都构建完毕，将这两个 Filter 分别加入到服务器端和客户端的 Filter Graph 中去便可实现网络发送和网络接收的传输功能。

5.4　客户端软件实现

5.4.1　数据解压模块的实现

该模块的实现和服务器端的压缩模块相对应。视频数据解压采用 XviD MPEG-4 Decoder 解码器实现，音频数据解压采用 MP3 解码器实现，它们的创建和使用方法与其对应的编码器相同，在 5.2.4 和 5.2.4 节中详细介绍过了，这里就不再累述。这两个解码器 Filter 分别如图 5-16、5-17 所示。

图 5-16　视频解压 Filter

Fig.5-16　Video decompression filter

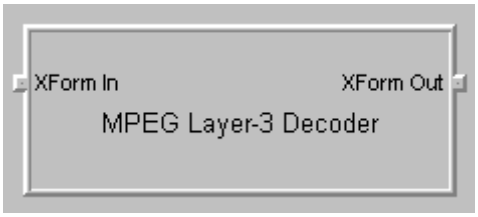

图 5-17　音频解压 Filter

Fig.5-17　Audio decompression filter

5.4.2　音视频显示模块

主要通过 Video Render Filter 和 Audio Render Filter 实现，它们位于 DirectShow Filter 目录下，CLSID 分别为 CLSID_VideoRenderer 和 CLSID_AudioRenderer。将解压缩后的数据传入它们的输入 Pin 就可以显示了。

5.4.3　抓图模块的实现

客户端在显示的同时可实现即时抓图功能，实现方法是使用 Filter Graph Manager 上的 IBasicVideo 接口，只要调用这个接口的 GetCurrentImage 函数就可以将当前帧抓下来，保存为静态的 BMP 文件。主要核心代码如下：

```
BOOL CDXGraph::GetCurrentImage (const char *inFile)
{
  //mBasicVideo 为 Filter Graph Manager 上获得 IBaseVideo 接口对象指针
  if(!mBasicVideo)
  {
      return FALSE;
  }
  long bitmapSize=0;
  //返回读取一帧需要的缓存大小
  if (SUCCEEDED(mBasicVideo->GetCurrentImage(&bitmapSize, 0)))
  {
      BOOL pass=FALSE;
      //申请足够大的缓存用于装载图象数据
      BYTE *buffer=new BYTE [bitmapSize];
```

```
        //返回图像数据（包括 BMP 信息头）
        if(SUCCEEDED(mBasicVideo->GetCurrentImage(&bitmapSize,(long*)buffer)))
        {
            BITMAPFILEHEADER            hdr;    //BMP 文件头
            LPBITMAPINFOHEADER          lpbi;   //BMP 信息头指针
            Lpbi=(LPBITMAPINFOHEADER)buffer;
            Int nColors=1<<lpbi->biBitCount;    //颜色数
            //如果颜色数超过 256，则该图像不使用调色板
            if(nColors>256)
            {
                nColors=0;
            }
            //初始化 BMP 文件头
            hdr. bfType=((WORD)('M'<<8)| 'B');//always is "BM"
            hdr. bfSize=bitmapSize+sizeof(hdr);
            hdr. bfReservedl=0;
            hdr. bfReserved2=0;
            hdr. bfOffBits=(DWORD) (sizeof(BITMAPFILEHEADER)+
                    lpbi->biSize +nColors*sizeof (RGBQUAD));
            //保存为一个 BMP 文件
            CFile bmpFile (inFile, CFile:: modeReadWrite| CFile::modeCreate
                    |CFile:: typeBinary);
            bmpFile.Write (&hdr, sizeof (BITMAPFILEHEADER)):
            bmpFile.Write (buffer, bitmapSize);
            bmpFile. Close ();
            pass=TRUE:
        }
        delete[] buffer;
        return pass;
    }
    return TRUE;
}
```

5.4.4　视音频录像模块的实现

该模块实现客户端以 MPEG-4 保存视频数据，MP3 保存音频数据，文件格式保存 AVI。用户可以在专门的回放窗口中观看录像文件，也可以通过 Windows Media Player 直接播放。写入文件的过程如 5.2.5 节中所述。

5.4.5　回放模块的实现

要在系统中回放一个录像文件，需要创建专门的回放 Filter Graph，如图 5-18 所示。在 Filter Graph 中要加入 MPEG-4 解压缩 Filter 和 MP3 解压缩 Filter ，数据经解压缩后，传递到 Video Renderer Filter 和 Audio Renderer Filter 进行播放。

图 5-18　文件回放的 Filter Graph

Fig.5-18　Filter graph of file playback

回放录像的播放器界面如图 5-19 所示。通过“ 打开 ”按钮，可以调入多媒体文件，“ 播放 ”、“ 暂停 ”、“ 停止 ”按钮分别控制媒体文件的播放、暂停和停止。“ 抓图 ”用于抓取播放过程中的当前图像帧，并保存为本地 BMP 文件。“ 全屏 ”用于媒体文件的全屏播放，Esc 或 Enter 键可以使视频回复到正常模式。下面简单介绍播放器的开发过程。

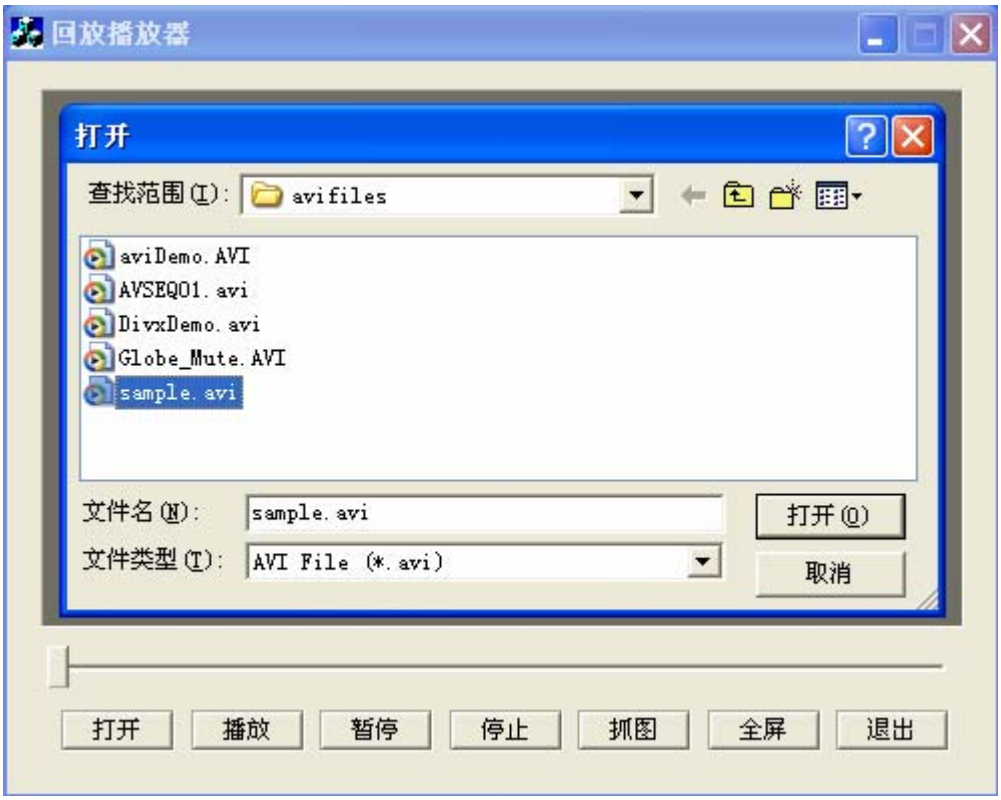

图 5-19　基于 DirectShow 的回放播放器

Fig.5-19　Playback player based on DirectShow

首先，编辑播放器界面并将它封装成类 PLAY,在 PLAY.h 中定义一个 CStatic 类型的视频窗口控制对象 mVideoWindow1。为了让视频窗口正常刷新，还需要修改视频窗口的风格，即在 PLAY.cpp 的初始化函数 OnInitDialog 中增加如下函数调用：

mVideoWindow1.ModifyStyle (0,WS_CLIPCHILDREN);

然后，添加对话框的 WM_ERASEBKGND 消息相应，代码如下：

```
BOOL PLAY::OnEraseBkgne (CDC* pDC)
{
    CRect rc;
    mVideWindow1.GetWindowRect(&rc);//获取播放窗口的屏幕坐标
    ScreenToClient(&rc);//将播放窗口的屏幕坐标转化为客户区坐标
    pDC->ExcludeClipRect(&rc);//调整图像大小以适应播放窗口大小
    return CDialog::OnEraseBkgnd(pDC);
    //调用积累 CDialog 的缺省处理函数处理 WM_ERASEBKGND 消息
}
```

我们将 Filter Graph 的各种相关操作，包括 Filter Graph Manager 的创建。其上各种控制接口的实现等封装成类 CDXGraph，并在 PLAY.h 中定义一个 CDXGraph 类型的指针 mFilterGraph。之后，根据用户选择的源文件，创建相应的回放 Filter Graph，核心代码如下：

```
void PLAY::CreateGraph(void)
{
    //创建新的 Filter Graph
    DestroyGraph ();
    mFilterGraph=new CDXGraph():
    if(mFilterGraph->Create())
    {
        mFilterGraph->RenderFile(mSourceFile);
        mFilterGraph->SetDisplayWindow(mVideoWindowl. GetSafeHwnd()):
        mFilterGraph->SetNotifyWindow(this->GetSafeHwnd());
        mFilterGraph->Pause();
     }
 }
```

最后，编写处理 Filter Graph 事件的代码。定义一个消息映射，并在头文件中声明消息相应函数。然后，在 cpp 文件中添加处理代码：

```
LRESULT PLAY::OnGraphNotify (WPARAM inWParam, LPARAM inLParam)
{
    IMediaEventEx *pEvent=NULL;
    if(mFilterGraph&&(pEvent=mFilterGraph->GetEventHandle()))
    {
        LONG eventCode=0, eventParaml=0, eventParam2=0;
        whi1e(SUCCEEDED(pEvent->GetEvent(&eventCode,&eventParaml ,
 &eventParam2, 0)))
        {
            pEvent->FreeEventParams(eventCode,eventParaml,eventParam2);
            switch (eventCode)
            {
            case EC_COMPLETE:
                OnButtonPause ();
```

```
                    mFilterGraph 一>SetCurrentPosition(0);
                    break;
                case EC_USERABORT:
                case EC_ERRORABOUT:
                    OnButtonStop ();
                    break;
                default:
                    break;
                }
            }
        }
        return 0;
    }
```

进度条的实现依赖 Filter Graph Manager 上的 IMediaSeeking 接口。通过一个定时器，不断调用 IMediaSeeking::GetCurrentPosition 接口方法来获取当前的播放事件，然后根据媒体的总时间长度，计算进度的百分比，并转化为进度控件的当前位置，然后通过调用 CsliderCtrl::SetPos 函数设置游标。

“播放”、“暂停”、“停止”按钮的事件代码中，分别调用了 CDXGraph 类的 Run()、Pause()、Stop()函数，实现代码如下：

- 暂停功能

```
void CGraPhBase::Pause()
{
m_pMediaControl->Pause ();
m_pMediaControl->StopWhenReady ();
}
```

- 停止功能

```
void CGraphBase::Stop()
{
m_pMediaControl->Stop ();
m_pMediaControl->StopWhenReady ();
}
```

- 播放功能

```
void CDxVideoGraphBase::play()
{
m_pMediaControl->Run ();
}
```

5.5 本章小结

作为本论文的重点，本章完整地给出了 C/S 结构下服务器端和客户端各模块功能的详细实现过程，以及网络传输的发送滤波器和接收滤波器的实现方法。到此也完成了整个系统的实现工作，对服务器端和客户端软件的运行界面和测试结果如下章所示。

第六章　系统的调试与运行

6.1　服务器端软件调试及运行界面

运行服务器端软件的计算机需要安装 DirectX9.0，外接 USB 摄像头，摄像头连入计算机后，系统启动是自动查找采集设备，服务器端界面如图 6-1 所示：

图 6-1　服务器运行界面

Fig.6-1　Server running interface

界面上设计了两个下拉列表，用来列举视频和音频设备，并设计了“ 视频设置 ”和“ 音频设置 ”两个按钮，前者用以跳出视频采集属性页，对视频采集参数进行设置，后者用以调出音频采集属性页，对音频采集参数进行设置。当按下“ 预览 ”按钮便开始显示视频画面，录像文件的路径为 C:\，文件名为：sample.avi。 ”“ 退出系统 ”按钮用于释放资源，结束系统的运行。

6.2　客户端软件调试及运行界面

运行客户端软件的计算机没有 XviD 解码器和 MP3 解码器则需要用户手动安装，所以客户端计算机最好也安装 DirectX9.0。客户端运行的界面如图 6-2 所示：

图 6-2　客户端运行界面

Fig.6-2　Client running interface

当按下“ 呼叫连接 ”按钮时会出现 IP 地址连接窗口，用以输入服务器主机的 IP 地址，连接到指定的服务器主机上后，便可以接收视音频数据。“ 视频调节 ”按钮可以对接收画面进行参数调节以适合最佳效果观看，“音频调节”按钮则可以对接收声音调节大小。按下“ 开始录像 ”可将多媒体文件保存到指定的文件中，供回放时观看。当按下“ 抓图 ”按钮时系统可把当前帧保存为位图文件。

系统的测试主要在研究所局域网（10M）中完成，其中 1 台 PC 作为服务器，连接一个 USB 摄像头，2 台 PC 作为客户机，获取前段视频流，对系统的各个功能进行了详细的测试，均达到了设计要求，服务器端视频采集功能正常，画面清晰，客户端连接和断开服务器功能正常，画面显示连贯清晰，延时小于 1 秒。

第七章　总结与展望

7.1　总结

本文研究和实现了基于流媒体技术的网络视频监控系统，通过对流媒体相关技术的研究和深入剖析 DirectShow 框架和机制，结合对监控系统整体构架了解的基础上，提出了系统的总体框架设计，并通过软件方式实现了网络视频监视系统的功能。重点完成了监控系统中服务器端和客户端软件各功能模块的实现，以及多媒体数据网络传输功能。

视频音频流的实时传输是目前多媒体领域的研究热点，DirectShow 以其灵活，高效的特点受到普遍关注，但完全运用 DirectShow 方法成功开发的远程监控软件并不多见。随着通讯网络宽带业务的拓展，网络传输能力将得到更进一步提高，网络服务质量也将得到有效的保证。原来依赖于硬件处理的编解码方式逐渐向软件转化方向发展，多媒体信息的大量采用将会使流媒体技术越来越多的应用于视频、音频传输系统中。

但目前在众多的多媒体音视频传输软件中，开发方法不规范，可重用性差一直是影响整个系统性能的主要因素。在本文的实践中，主要对以下方面进行了研究和开发：

（1）研究了视频监控技术理论体系，分析了流媒体技术的基本原理，并简要说明流媒体技术的出现对网络视频监控技术带来的技术革新。对流媒体技术应用于视频监控系统中的可行性进行了分析。

（2）在搭建系统前，详细分析了开发平台 DirectShow，从 DirectShow 的结构、工作原理、工作流程以及其内部实现机制进行了深入的分析和阐述，最后介绍了如何利用 DirectShow 开发应用程序。

（3）选用了 Visual C++6.0 作为开发平台，运用微软的 DirectShow 技术对视音频数据采集、压缩、解压、回放等功能进行开发编程，实现了服务器端和客户端软件的功能。

（4）重点研究了与系统网络传输模块相关的网络通信技术，并完成了网络传输模块的软件结构设计，通过开发网络发送 Filter 和网络接收 Filter 实现了服务器端网络发送和客户端网络接收功能。

（5）系统完成了局域网中的测试，取得了较好的效果，图像连贯，无明显马赛克，延时较小，满足了用户的需求。

随着社会的进步和技术的发展，视频监控技术总体上是朝着数字化、网络化以及功能综合化的方向发展，流媒体技术的兴起给网络视频监控技术的发展带来了新的契机。本论文是面向流媒体技术应用于网络监控领域的研究，由于时间仓促，且个人能力有限，对这一前沿技术研究的不够深入，设计的系统也只是在功能上进行了实现，一定存在许多需要改进和补充的地方。同时，围绕着本课题相关技术，今后仍然有许多可以值得研

究的工作可做：

（1）由于时间短暂，用户界面功能还不够完善，不能完全满足监控系统的要求，有待进一步完善，如增加多画面同时监控、动态报警等，以适合各种监视场合的需求。

（2）在设计过程中没有考虑安全性问题，任何安装了客户端软件的用户均可接收实时视频数据，在服务器端没有进行客户验证。

（3）因为是模拟的系统实现，没有对界面精雕细琢，也没有能在广域网上测试。如要作为推向市场的产品，还需在界面以及向广域网上拓展方面下一定的功夫。比如开发基于 B/S 模式下的客户端软件，从网页浏览和控制监控地点等。

7.2 展望

视频监控是目前国内外的一个热点研究领域，进入二十一世纪以来，数字化、网络化的步伐逐步加快，社会对监控领域的需求范围不断扩大，对监控系统本身的交互性、高效性、灵活性、安全性及稳定性也必然会提出更高的要求。然而随着技术的发展，视频监控技术并不只依靠计算机技术，而更多地是结合多媒体技术、网络通信技术、数字压缩技术、图像识别技术、流媒体技术等先进的技术，使得视频监控系统发展前景更为广阔。

本论文开发的小型网络监控系统，在实际的应用中，只需在现场安装一个摄像头，在企业的网络上应用本系统就能很方便的实现视频监控功能，对于那些不想在资金上投资很多，又想实现监控功能的企业来说，不失为一种很好的选择。虽然还有些监控功能没有实现，但是已经实现的这些功能是监控系统中的重要部分，该系统完善后的发展前景十分可观。

致谢

在此，向我的导师陈剑云教授表示衷心的感谢!本文从开题到完成都得到了陈老师悉心的指点。在攻读硕士的三年里，陈老师对我的指点使我受益匪浅。如果没有陈老师的帮助和教导，我无法取得新的进步，也无法完成硕士研究生的学业。

感谢研究所的各位老师，在我的研究生学习期间给予我无私的帮助和关怀，忠心地感谢他们！

感谢我的同学和朋友，陪伴我度过开心的研究生生活，在工作和学习生活中给予我大量的帮助和支持，在此向她们表示深深的谢意!

特别感谢我的父母和家人，他们对我的关心、鼓励和支持是我不断前进的动力!

最后，忠心地感谢在百忙之中为本论文评阅和参加答辩的各位老师！

参考文献

[1] 刘富强．数字视频监控系统开发及应用[M]．北京：机械工业出版社，2003．

[2] 张翔等．基于 IP 视频监控应用的 H.264 视频压缩[J]．工业控制计算机, 2003, Vol.16(12): 53-56.

[3] 潘勇强译. H.264 编码和基于 H.264 的移动服务[OL]. http://www.wx800.com, 2004-07．

[4] 吴辉．基于嵌入式 DVR/DVS 的网络视频监控系统的研究与实现[D].南昌：华东交通大学, 2005.

[5] 胡志刚．基于移动通信网络的视频监控系统设计与实现[D]．长沙：国防科技大学，2006．

[6] 吴国勇，邱学刚，万燕仔.网络视频流媒体技术与应用[M]．北京：北京邮电大学出版社，2001．

[7] 托匹克著，孔英会译.流媒体技术及商机揭秘[M]．北京：电子工业出版社，2004．

[8] 李炳林．流媒体技术及应用[J]．电力系统自动化, 2001:68-71.

[9] 于广．基于 DirectShow 的流媒体组件技术的研究与应用[D]．广州：华南理工大学，2003．

[10] 杨嫚，朱红.流媒体的发展现状及趋势[J]．情报科学，2003，Vol.21(12)：1246-1248．

[11] 袁尧．基于流媒体传输的视频监控平台的研究[D]．北京：北方交通大学，2006．

[12] 宋刚，杨显富．实时流媒体传输及其协议[J]．成都大学学报(自然科学版)，2005(3)．

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

[14] 张丽．流媒体技术大全（第一版）[M]．北京：中国青年出版社，2001．

[15] H.Schulzrinne et al. RTP: A Transport Protocol for Real-time Application[S]. IETF Audio/Video Transport Working Group, RFC 1889, 1996-01.

[16] H.Schulzrinne. RTP Profile for Audio and Video Conference with Minimal Control[S]. Internet ,RFC 1890, 1996(l).

[17] RFC 2326 Real-Time Steaming Protocol (RTSP) [S]. 1998.

[18] Microsoft DirectX9 online document[EB/OL]. http://msdn.microsoft.com．

[19] Microsoft DirectX9.0 SDK. Microsoft Corporation. 2004

[20] 陆其明．DirectShow 开发指南[M]．北京：清华大学出版社，2001．

[21] 陆其明．DirectShow 实务精选[M]．北京：北京科海电子出版社，2003．

[22] 潘爱民．COM 原理与应用[M]．北京：清华大学出版社，1999，11：11-14

[23] 袁红亮．基于 DirectShow 的流媒体实时传输的研究与实现[D]．大连：大连理工大学，2005．

[24] 刘忠贤．基于流媒体的网络监视系统的设计与实现[D]．西安：西安电子科技大学，2005．

[25] 张熙.基于 DirectShow 和 RTP 的网络视频监控系统设计与开发[D].四川：西南交通大学，2006．

[26] 张永刚．西变厂网络视频监控系统设计中的流媒体传输技术及应用[D].西安：长安大学，2006

[27] 汪理虎.基于流媒体的网络视频监控技术研究与系统实现[D].南京：南京航空航天大学，2006．

[28] 宋智．基于流媒体的网络视频监控系统研究与实现[J]．现代电子技术，2006(08)：66-70．

[29] 四维科技等．Visual C++视频/音频开发实用工程案例精选[M]．北京：人民邮电出版社,2004．

[30] 汪晓平等．Visual C++网络通信协议分析与应用实现[M]．北京：人民邮电出版社，2003

[31] 许先斌等．运用 RTP 协议实时传输 MPEG-4 流[J]．计算机工程与设计，2003(2)：57-59

[32] 赵进等基于 RTP 协议族的流媒体系统设计和实现[J]．计算机工程，2005(l)：195-197

[33] 刘毓敏，李剑琴，姚彬．数字视音频技术开发与应用[M]．北京：国防工业出版社，2003．

[34] 李燕灵，马瑞芳，左力．基于 RTP/RTCP 的实时视频数据传输模型及实现[J]．微电子学与计算机，2005，Vol.22(8)：138-140．

[35] http://research.edm.uhasselt.be/~jori/page/index.php?n=CS.Jrtplib.

[36] D.Wu, Y.T.Hou, W.Zhu.Streaming Video over the Internet: Approaches and Directions [J]. IEEE Transactions on Circuits and Systems for video Technology, March 2001, VOl.11 (13): 282-300.

[37] R.Leonardi. Intermedia synchronization for video conference over IP [J]. Image Communication 1999 (15):149-164.

[38] GoaYujni, Shi Feng, Zhang Yansu. Adaptive DirectShow Franework for Layer MPEG-4 Video Multicast [J]. Journal of Beijing Institute of Technology, 2003, Vol (12): 114-119

[39] C. Sacchi, F. Granelli & C. S. Regazzoni, A post-processing algorithm for performance enhancement of remote video-based monitoring systems [J]. IEEE, 1999:351-356.

[40] Michael Greiffnhagen, Dorin Comaniciu, Heinrich niemann and Visvanathan Ramesh, Design, analysis, and engineering of video monitoring systems, an approach and a case study [J]. Proceedings of the IEEE, 2001, Vol.89(10):1498-1517.

[41] Chao Huang, Jintao Li and Hongzhou Shi, An intelligent streaming media video service system [J]. IEEE, 2002:5-10.

[42] Damien Stolarz, Peer-to-peer streaming media delivery [J]. IEEE, 2002:48-52.

[43] Yuka Kato, DongMei Jiang and Katsuya Hakozaki, A proposal of a streaming video system adapting to various system environments and its implementation [J]. IEEE, 2004.

[44] Tetsuya Oh-ishi, Koji Sakai, Kazuhiro Kikuma and Akira Kurokawa, Study of the relationship between peer-to-peer systems and IP multicasting [J]. IEEE Communications Magazine, 2003:80-84.

[45] Zhaoyu Liu, Dichao Peng, Yuliang Zheng and Jeffrey Liu, Communication protection in IP-based video surveillance systems [J]. IEEE Computer Society, 2005.

[46] Lei Guo, Songqing Chen and Xiaodong Zhang, Design and evaluation of a scalable and reliable p2p assisted proxy for on-demand streaming media delivery [J], IEEE Computer Society, 2006:669-682.

[47] Jun Wang, Wei-Qi Yan, Mohan S.Kankanhalli, Ramesh Jain and Marcel J. T. Reinders, Adaptive monitoring for video surveillance [J]. IEEE, 2003:1139-1143.

个人简历　在读期间发表的学术论文

个人简历：

姜琴，女，1982 年 11 月生。

2004 年 7 月毕业于华东交通大学通信工程专业，获学士学位。

2005 年 9 月入华东交通大学读硕士研究生。

已发表论文：

[1] 姜琴，陈剑云.基于 CDMA 的变电站移动视频监控系统的设计[J].工业控制计算机，2008，第 4 期；

[2] 姜琴,王林,马勇.基于 P2P 技术的流媒体转发服务器的设计与实现[J]. 计算机系统应用,2008，4：p90-92；

[3] 马勇, 姜琴等. 基于禁忌搜索遗传算法的 PMU 布点配置[J]. 继电器, 2008, Vol. 36(2): 21-25.