



华南理工大学

South China University of Technology

硕士学位论文

基于 RTSP 的 H.264 实时流媒体传输方案
的研究与实现

作者姓名	李罗涛
学科专业	电路与系统
指导教师	吴宗泽副教授
所在学院	电子与信息学院
论文提交日期	2014 年 5 月

Research and Realization of H.264 Real-time Streaming Media Mechanism based on RTSP

A Dissertation Submitted for the Degree of Master

Candidate: Li luotao

Supervisor: Prof. Wu zongze

South China University of Technology

Guangzhou, China

分类号：TN393.5

学校代号：10561

学 号：201120107529

华南理工大学硕士学位论文

基于 RTSP 的 H. 264 实时流媒体传输方案 的研究与实现

作者姓名：李罗涛

指导教师姓名、职称： 吴宗泽副教授

申请学位级别：工学硕士

学科专业名称：电路与系统

研究方向：图像技术与智能系统

论文提交日期： 2014 年 5 月 20 日

论文答辩日期： 2014 年 6 月 7 日

学位授予单位：华南理工大学

学位授予日期： 年 月 日

答辩委员会成员：

主席：周智恒 副教授_____

委员：傅予力 教授 吴宗泽 副教授 李波 副教授 向友君副教授_____

华南理工大学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：李罗清

日期：2014年6月7日

学位论文授权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属华南理工大学。学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许学位论文被查阅（除在保密期内的保密论文外）；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。本人电子文档的内容和纸质论文的内容相一致。

本学位论文属于：

保密，在年解密后适用本授权书。

不保密，同意在校园网上发布，供校内师生和与学校有共享协议的单位浏览；同意将本人学位论文提交中国学术期刊(光盘版)电子杂志社全文出版和编入 CNKI《中国知识资源总库》，传播学位论文的全部或部分内容。

(请在以上相应方框内打“√”)

作者签名：李罗清
指导教师签名：李罗清

日期：2014-6-7

日期：2014-6-7

摘 要

随着互联网的广泛普及和多媒体技术的迅速发展，基于网络的流媒体传输技术得到了广泛的应用。如视频通话、视频监控、视频点播、网络直播、远程医疗等。但互联网只提供一种“尽力而为”的服务，在视频传输的过程中，由于网络的延时，带宽的不稳定，编码效率的低下，很容易造成数据包的丢失，从而导致视频播放的失真等问题，因此很有必要研究一个有效的实时流媒体传输方案来解决这些问题。本文在对 live555 进行二次开发的基础上，利用 FFmpeg 和 H.264 的高效编解码技术以及 RTSP 和 RTP/RTCP 的高效网络传输策略来实现了一个基于 RTSP 协议的 H.264 实时流媒体传输方案。

本文先是对流媒体的相关技术进行了介绍，对网络传输协议进行了深入的研究，选择了 RTSP、RTP、RTCP 协议作为网络传输和控制协议，H.264 和 AAC 作为主要的视频、音频编解码标准。

其次对流媒体方案进行了比较，选择了 live555 和 FFmpeg 作为主要的技术框架。在对 live555 库、FFmpeg 库和 Android 系统的架构进行了简要地分析的基础上，提出了系统的总体框架，并对服务器和客户端的主要模块进行了简要介绍。

接着详细地分析了实时流媒体传输系统的服务器方案，利用 RTCP 技术解决了实时传输的拥塞控制问题，针对 Live555 不支持客户端上传、MP4 文件下发、实时转发等问题，对 Live555 进行了二次开发，增加了上述功能模块，并进行了多进程扩展。

然后对流媒体客户端的方案进行了详细介绍，通过在 Android 平台上移植 FFmpeg，结合 Android API 开发了一个既支持 RTSP 上传又支持 RTSP 播放的手机客户端，通过采用缓冲队列和时间戳来解决音视频同步和播放等问题。此外还介绍了音视频采集模块，音视频编码模块，客户端 MP4 文件解析模块、音视频解码模块，音视频播放模块、音视频同步模块的具体实现。

最后对系统的硬件和软件环境进行了介绍，并对流媒体服务器和客户端进行了相关测试，对测试结果进行分析表明，系统具有较好的实时性和传输质量。

关键字： 流媒体；Live555；FFmpeg；实时转发；RTSP/RTP/RTCP；H264

Abstract

With the rapid development of the Internet and the widespread popularity of multimedia technology, streaming media transmission technology based on network has been widely used. Such as video calls, video surveillance, video on demand, webcast, telemedicine and so on. However, the Internet is only a "best effort " service, in the course of the video transmission, because of the network delay, unstable bandwidth and the low encoding efficiency, it is easy to cause the loss of data packets, resulting video playback distortion and other problems, so it is necessary to research an effective real-time streaming media solutions to solve these problems. Based on the live555 secondary development, this theis realizes a RTSP-based H.264 protocol real-time streaming media transmission system, on the use of FFMPEG and efficient H.264 codec technology as well as RTSP and RTCP/RTP efficient network transmission strategy.

First, this article introduces the related technologies of streaming media and researchs the network transport protocol deeply, chooses RTSP, RTP, RTCP protocol as the network transport and control protocols, H.264 and AAC as the main video, audio codec standard.

Second, streaming media solutions are compared to select the live555 and FFMPEG as the main technical framework. On the basis of the brief analysis of live555 library, FFMPEG library and the Android system architecture, this article proposes a general framework for the system, and introduces the main modules of server and client briefly.

Third, a detailed analysis of real-time streaming media server scheme is achieved using RTCP technology to solve the problem of congestion control in real-time transmission process, due to Live555 does not support client upload, MP4 document issued, real-time forwarding and other issues, carring on a secondary development on Live555, completing the function of these additional modules, and a multi-process extension.

Then streaming media client programs are described in detail, by transplanting FFMPEG on the Android platform, combined with Anroid API, develop a mobile client that supports both RTSP uploading and RTSP playback, through the use of buffer queue and timestamps to resolve audio and video synchronization and playback problems. It also introduces the specific implementation of the audio and video capture module, audio and video coding module, the MP4 file parsing module of client, audio and video decoding module, audio and video playback module, audio and video synchronization module.

Finally, the hardware and software environment of the system are introduced, and related

tests are done on streaming media server and client, test results show that the system has good real-time performance and transmission quality.

Keywords: Streaming Media; Live555; FFMPEG; Real-time Forwarding; RTSP /RTP / RTCP; H264

目 录

摘 要.....	I
Abstract.....	I
目 录.....	I
第一章 绪论	1
1.1 选题背景与意义.....	1
1.2 国内外研究现状.....	1
1.3 本文的组织结构.....	2
第二章 流媒体相关技术概述	4
2.1 流媒体传输协议介绍.....	4
2.1.1 RTSP 协议.....	4
2.1.2 RTP 协议.....	5
2.1.3 RTCP 协议.....	5
2.1.4 SDP 协议.....	7
2.2 H.264 相关技术.....	8
2.2.1 网络提取层 (NAL)	8
2.2.2 RTP 包头介绍.....	11
2.2.3 H264 的 RTP 传输.....	11
2.3 流媒体的播放方式.....	13
2.3.1 单播.....	13
2.3.2 组播.....	13
2.3.3 点播与广播.....	13
2.3.4 P2P 播放.....	14
2.4 本章小结.....	14
第三章 实时流媒体总体方案介绍	15
3.1 方案选型.....	15
3.2 方案中开源库的介绍.....	15
3.2.1 live555 库.....	15
3.2.2 FFMPEG 库.....	17

3.2.3 Android 介绍.....	18
3.3 总体方案设计.....	22
3.3.1 总体框架.....	22
3.3.2 流媒体服务器端.....	22
3.3.3 流媒体客户端.....	23
3.4 本章小结.....	23
第四章 实时流媒体服务器方案实现.....	24
4.1 流媒体服务器整体方案.....	24
4.2 RTSP 服务器的主要逻辑.....	25
4.3 RTP 打包与发送.....	31
4.4 扩展 RTSPServer 支持处理客户端的上传.....	34
4.5 扩展 RTSPServer 支持 MP4 视频格式.....	35
4.6 扩展 RTSPServer 支持实时转发.....	37
4.7 使 RTSPServer 支持多进程.....	39
4.8 本章小结.....	40
第五章 实时流媒体客户端方案实现.....	41
5.1 流媒体客户端整体方案.....	41
5.2 FFmpeg 在 Android 上的移植.....	42
5.3 客户端音视频采集.....	45
5.3.1 音频采集实现.....	45
5.3.2 视频的采集实现.....	46
5.4 客户端音视频编码.....	48
5.5 客户端 MP4 文件的解析.....	48
5.6 客户端音视频解码.....	50
5.7 客户端音视频播放.....	51
5.8 客户端音视频同步.....	52
5.9 本章小结.....	53
第六章 系统测试.....	54
6.1 硬件环境.....	54
6.2 软件环境.....	54

6.2.1 LVS 环境搭建	54
6.3 系统测试	56
6.3.1 边拍边传功能测试	56
6.3.2 播放 rtsp 视频功能测试	56
6.3.3 播放本机视频功能测试	57
6.3.4 实时转发功能测试	58
6.3.5 服务器性能测试	59
6.4 测试结果分析	61
6.5 本章小结	61
第七章 总结与展望	62
7.1 总结	62
7.2 展望	63
参考文献	64
攻读硕士学位期间取得的研究成果	66
致 谢	67

第一章 绪论

1.1 选题背景与意义

当前，计算机网络和流媒体技术迅速发展，人们已不再满足于网络上传输文件、网页浏览等简单的功能，希望能够在网络上享受更加丰富的多媒体实时交互功能，比如在网上观看直播视频，在手机端实现视频通话，在教育上实现远程教育，在交通、超市等地实现视频监控等功能^[1]。分析这些业务可以发现，它们都有一个共同特点，那就是包含的数据信息数据量非常大，对网络的实时性要求非常高。如何能够研究出高效的音视频压缩算法，在保证质量的基础上尽量地降低数据量，以达到最高的压缩比，使用可靠的传输协议来实现音视频数据在网络上的可靠的传输，有效地解决网络的信道带宽不稳定，网络中传输数据的信道误码率，保证音视频传输的实时性要求，这些都是一直以来研究的技术热点。自此，视频压缩编码技术和网络传输协议的研究得到了迅速的发展。

当今世界计算机网络迅速发展，流媒体技术已经在无形中改变着我们的生活方式，现在人们几乎已经离不开网络，如何研究出先进的技术并用于实践使我们的生活变得更加丰富、方便，一直是我们努力的目标。所谓流媒体就是应用流技术在网络上传输的多媒体文件。网络上采用的传输媒体的方式主要有两种：下载和流式传输。而流式传输技术是指将音视频等资源应用流技术在网络上进行传输，可以实现边下载边播放，而不需要花费很长的时间将整个媒体文件先下载下来再进行播放，用户只需要花费几秒钟的时间将音视频资源先放到缓冲区，得到必要的播放数据，然后就可以将下载和播放并行，这样可以很大程度上减少用户的等待时间，提高用户体验。

基于 RTSP 的 H.264 的实时流媒体传输方案，应用范围十分广泛，可用于视频通话、视频短信、视频会议、远程教育、远程医疗、视频监控、视频直播、点播等等领域，意义重大。本论文是基于实验室的横向课题——视信，视信是一个能够实现即拍即传的视频短信系统，本文的实时流媒体传输方案即是视信系统里面关于视频通道的研究，视信系统还包括了 web 服务器、请求服务器等功能。

1.2 国内外研究现状

在流媒体领域，日本和韩国发展得比较早，视频业务的发展也一直保持着领先地位。韩国电讯 SK 推出了“NATEON”服务，提供了音视频点播、游戏、网络电视等内容。日

本的 NTT 和 DoCoMo 在流媒体方面表现也十分出色，推出了 i-mode 类型的业务模式。

在国内也出现了很多流媒体的应用，如腾讯视频、优酷网、搜狐视频、乐视、ppstream 等，我国流媒体业务发展迅速，已经进入了先进行列。

流媒体技术主要解决三个方面的问题：一是使用压缩率高的音视频压缩算法，创造一个高效的编码器，二是需要有一个高性能的流媒体服务器，以满足高并发的要求，三是要使用一个可靠的流媒体传输协议来确保流媒体的实时高效传输^[23]。

编码及压缩技术：当前网络上的流媒体格式主要包括三种，Real Networks 公司的 RealMedia, 苹果公司的公司的 QuickTime, 以前微软公司的 ASF。这三个公司都有自己的压缩编码标准并享有专利权，不利于我们的开发。后来人们都重点集中在国际上统一的视频编码标准——MPEG，它不涉及专利问题，更适合开发。在众多的标准当中，H.264 代表当前最先进的压缩标准，如果以 MPEG-2 作为基准的话，达到相同的播放效果 MPEG-4 只要 60% 的带宽，而 H.264 却只要 40%。H.264 作为一种先进的编码技术，能够很好地进行速率调节，适合网络速率的变化。

流媒体服务器的性能：随着流媒体规模的不断扩大，对流媒体服务器也提出了更高的要求。限制服务器性能的三个主要性能指标是并发量，磁盘 I/O 瓶颈，网络带宽。因此需要针对这些特点选择一个合适的系统框架，注重流媒体协议的扩展保证流媒体的传输质量，采用集群策略和良好的文件存储策略来解决并发量和存储的问题。

流媒体传输协议：流媒体技术还要有一个可靠的传输协议来支持音视频资源的传输，解决带宽、丢包和延时的问题。由于 TCP 比较占用资源，传输控制信息如 RTSP 交互连接一般采用 TCP 的方式，而传输数据信息则采用 RTP/UDP 的方式。传输协议主要有三种：RTSP 实时流协议，RTP 实时传输协议，RTCP 实时控制协议。

1.3 本文的组织结构

基于本课题的研究目的，本文在研究了流媒体的背景之下，通过熟悉各种流媒体协议和开源的各种框架，提出了一种基于 RTSP 的 H.264 实时流媒体传输方案，本文分为七章来介绍系统的研究背景、流媒体的相关技术、系统方案研究、服务器端和客户端的实现以及测试结果。内容结构如下：

第一章介绍了课题研究的背景和意义，分析了国内外针对本课题的研究现状，并给出了本论文的组织结构。

第二章概述了流媒体的相关技术，介绍了各种流媒体传输协议，如 RTSP 协议、RTP

协议、RTCP 协议以及 SDP 协议。还介绍了 H. 264 协议的网络提取层 (NAL) 的具体工作原理, RTP 的包头和传输以及流媒体的主要播放方式。

第三章在对流媒体方案进行了比较和选型, 介绍了本系统中用到的两个开源库, live555 库和 FFmpeg 库, 然后对 Android 的框架进行介绍的基础上提出了实时流媒体总体方案, 还简要介绍了服务器端和客户端的主要部分的功能。

第四章是本论文的核心章节, 提出了流媒体服务器的整体方案, 介绍了 live555 中 RTSP 服务器的主要逻辑和 RTP 的打包和发送, 接着对 live555 进行二次开发, 使其支持处理客户端的上传, 支持 MP4 格式文件的下发, 支持实时转发、进行多进程的扩展。

第五章介绍了实时流媒体客户端方案的实现, 包括音视频的采集、音视频的编码、MP4 格式的分析、客户端音视频的解码、客户端音视频的同步、以及播放。

第六章介绍了系统的测试, 包括环境部署和服务器、客户端的测试。

第七章是总结和展望, 通过对项目的现状进行分析, 总结了一些不足之处, 并提出了一些需要改善的地方。

第二章 流媒体相关技术概述

2.1 流媒体传输协议介绍

2.1.1 RTSP 协议

RTSP (Real-Time Streaming Protocol) [4], 实时流协议, 是一种应用层的协议, 用于实时地控制数据的传输。RTSP 提供一个可扩展的架构来实现控制实时媒体资源的在线点播, 如音频或视频内容。数据源可以是直播信号也可以是制作好的音视频文件。RTSP 能够同时控制多个 RTP 的会话过程。

RTSP 实时流媒体协议[5], 是一种实时流媒体控制协议, 它本身并不进行数据流的传输, 而只是为对数据流的传输进行控制, 它包括了对 RTSP 连接进行建立以及类似于 VCR 的会话控制功能, 包括播放、暂停、快进、快退等。RTSP 可以选择使用 TCP 或 UDP 进行连接, 因为 RTSP 只是发送控制信息, 一般使用 TCP 的方式, 而 RTP(流媒体传输协议)一般使用 UDP 的方式。RTSP 与 HTTP 类似, 但 HTTP 是一种无状态的协议而 RTSP 是一种有状态的协议, 客户端和服务端都可以互相发送 RTSP 请求。RTSP 的应用场景一般包括三个方面:

从媒体服务器上播放资源。客户端可以通过 RTSP 协议点播服务器上的音视频资源, 通过 RTP 协议拉取数据包进行解码播放。

媒体服务器加入会议中。媒体服务器可以加入正在进行的会议, 并且可以进行会议的回放, 录制等。

将媒体加入讲座中。RTSP 服务器可以告诉用户有附加资源加入了讲座中, 客户可以进行获取。

RTSP 协议支持的操作如下:

RTSP 请求报文的方法包括: OPTIONS、ANNOUNCE、DESCRIBE、SETUP、TEARDOWN、PLAY、RECORD、PAUSE、GET_PARAMETER 和 SET_PARAMETER 等。

下表 2-1 是 RTSP 的方法的总览, 其中方向代表是由哪一方主动请求哪一方, 对象指当前 RTSP 方法用来操作的是表示还是流, 而要求是指为了实现 RTSP 会话, 当前方法是否必要。

表 2-1 RTSP 方法介绍

方法	方向	对象	要求
OPTIONS	C->S, S->C	P,S	必要
DESCRIBE	C->S	P,S	推荐
ANNOUNCE	C->S, S->C	P,S	可选
SETUP	C->S,	S	必要
PLAY	C->S	P,S	必要
PAUSE	C->S	P,S	推荐
RECORD	C->S	P,S	可选
REDIRECT	C->S	P,S	可选
SET_PARAMETER	C->S, S->C	P,S	可选
GET_PARAMETER	C->S, S->C	P,S	可选
TEARDOWN	C->S,	P,S	必要

项目中的 RTSP Stream Server 是基于开源项目 live555 进行开发的，但 live555 中的 RTSP Server 不支持处理 ANNOUNCE 和 RECORD 方法，所以我们需要自己扩充。

2.1.2 RTP 协议

实时传输协议（Real-time Transport Protocol）RTP^[6 7]是一种网络传输协议，它负责网络上流媒体数据包的实时发送与接收。因为 RTP 传输的是音视频数据，数据量很大，允许少量的丢包，所以传输层一般使用 UDP 协议，以降低资源的消耗。RTP 协议只负责数据的传输而不保证质量，需要通过 RTCP 实时控制协议进行流量控制和拥塞控制来保证传输质量。在流媒体数据传输中的一个重要问题就是不能预料数据包的到达时间，所以在 RTP 的协议中包含了时间标签，发送端在 RTP 打包时可以设置时间标签，接收端就可以按照时间标签来以正确的速率来还原原始的数据。

2.1.3 RTCP 协议

RTCP 概要

实时控制协议(Real-time Control Protocol) 与 RTP 一起共同定义在 1996 年提出的 RFC 1889 中，它是和 RTP 协议一起协作的控制协议。RTCP 单独运行在底层的协议上，由底层协议提供数据包与控制包的复用。在整个 RTP 的会话当中，每个会话参与者会周期性地向所有其他参与者发送 RTCP 控制信息包。RTCP 包含了五种类型信息包：(1)

SR: 发送报文, 表示当前情景发送者的发送、接收统计; (2)**RR**: 接收报文, 表示非情景发送者的接收统计; (3) **SDES**: 源描述项报文, 包括 **CNAME** 等信息; (4) **BYE**: 结束报文; (5)**APP**: 表示特定的函数, 用户可以自己定义, 利于 **RTCP** 协议的扩展。其中最主要的报文是 **SR** 报文和 **RR** 报文。通常对于 **RTP** 会话或者广播来说, 会使用单个多目标广播地址, 属于所在会话的两种信息包都会使用这个地址, **RTP** 信息包和 **RTCP** 信息包可以通过使用不同的端口号区分开来。

RTCP 功能

1、为应用程序提供当前的通信质量或者广播性能服务质量的信息

RTCP 的主要功能是保证 **RTP** 的传输质量, 在客户端和服务端之间发送控制信息, 为 **RTP** 提供流量控制和拥塞控制。每个 **RTCP** 数据包并不封装音频数据或者视频数据, 而是封装已经发送的数据包的数量、丢失的数据包的数量以及数据包的抖动情况信息的统计信息报表。这些非常有用的反馈信息说明了当前的网络状况, 对发送端、接收端都非常有价值。应用程序开发人员可以根据这些反馈信息进行相应的处理。比如, 发送者可以根据相应的反馈信息来调整发送速率, 以防止网络拥塞等问题, 接收者则可以根据相应的反馈信息来判断问题的来源, 是本地的还是远端的问题, 网络管理者通过对 **RTCP** 信息包中信息进行分析也可以得知所在网络的多目标广播性能。

2、确定 RTP 用户源

RTCP 协议为每一个 **RTP** 用户提供一个全球唯一的规范名称 (**Canonical Name**) 的标识符 **CNAME**^[8], 接收端可以使用它来得知在一个 **RTP** 进程有哪些参与者。当程序重新启动或者发现冲突时, **RTP** 包头的 **SSRC** 同步源标识符可能改变, 接收端则可以使用 **CNAME** 来跟踪参与的用户。同时, 利用 **CNAME**, 接收者也可以在几个相关的 **RTP** 连接中建立数据流的联系。此外, 接收者也可以利用 **CNAME** 来进行音视频同步, 把来自同一个发送者的音视频数据关联起来。

3、控制 RTCP 传输间隔

由于参与者越来越多, 而每一个会话都会定时地向对方发送 **RTCP** 控制信息包, **RTCP** 控制信息包如果发送太频繁, 将会占用过多的网络资源, 为了防止网络拥塞, 我们必须限制 **RTCP** 包的发送频率, 控制信息的带宽一般不要超过可用带宽的 5%。为了对 **RTP** 的发送进行控制, 任意两个 **RTP** 用户之间都会互发 **RTCP** 信息, 因此我们很容易估算出 **RTP** 用户的数量, 而程序可以根据会话中的用户量来动态调整 **RTCP** 分组的传输速率。

4、传输最小进程控制信息

通过这项功能，用户可以随意进入或离开松散的会话进程，参与人员进入离开都很自由，不会对成员进行控制或参数协调。

2.1.4 SDP 协议

SDP(Session Describe Protocol) 即会话描述协议^[9]，它可以为会话通知、会话邀请以及其它形式的多媒体会话目的提供多媒体的会话描述。

SDP 即用于将相关的设置信息传输到接收端，SDP 只是一种会话描述格式，它不属于传输协议，它只是用于在各种传输协议中用于对会话进行描述，包括 SAP(session Announcement Protocol) 会话通知协议、RTSP 实时流协议、SIP(Session Initiation Protocol) 会话初始协议、以及 HTTP 超文本传输协议。

SDP 当初的设计目的是通用性，它可以应用于各种网络环境和应用程序，而不只是局限于组播，但 SDP 并不支持采用哪种会话内容或媒体编码。

SDP 信息包通常由会话信息与媒体信息组成。

a) 会话信息

会话信息由会话名称、目的以及会话的活动时间组成。通常由于参与会话的资源有限，需要包括这些有用的附加信息：会话的联系人信息和会话的所占带宽。

b) 媒体信息

媒体信息包括：媒体类型，例如音频、视频、字幕；传输协议，例如 TCP/UDP/RTP/IP；媒体格式，例如 H.264 视频和 AAC 音频；多播组地址以及媒体的传输端口(IP 多播通信)；对于 IP 单播而言，则包括单播地址和端口号。

SDP 描述由许多文本行组成，文本行的格式为<类型>=<值>，<类型>是一个字母，<值>是结构化的文本串，其格式依<类型>而定。

为了减少 SDP 描述所用的开销，便于差错检测，采用了紧凑型编码。构成 SDP 会话描述的文本行的格式为：<type>=<value>，一般由多个字段组成，各字段由一个空格符分隔。

通常一个 SDP 描述包括一个会话描述和 0 个或者多个媒体描述^[10]。媒体可以是音频、视频、字幕等。会话级部分是从‘v=’行开始，后面则是 0 个以上的媒体级部分。媒体级描述由‘m=’行开始，后面则跟随下一个媒体描述，或者是整个会话描述的结尾。

2.2 H.264 相关技术

2.2.1 网络提取层 (NAL)

在网络传输中，编码器将各个 NAL 单元独立开来，完整地放入分组当中，由于每个分组都有头部因此解码器可以很容易地将 NAL 单元进行分离，并传给解码器进行解码。解码器并不能区别 NAL 的边界，因此需要合适的机制来解决这个问题。

针对此，H.264 将 NAL 单元进行了如下定义：NAL 单元由一组用于视频编码数据的 NAL 头部信息和用于数据封装的原始字节载荷 (RBSP) 构成^[1]。在 H264 的规定中，RBSP 的类型分为两种，一种封装的是音视频的编码数据，另一种是封装的是控制信息。

在 NAL 单元的头部信息中包含有一个是否可以告戒的标记，指示着告戒此信息是否会导致错误，在一般情况下，如下这个信息不是用于构建图像则可以丢弃。送到解码器中的 NAL 单元必须保持严格的顺序，如果错序的话则必须有一种恢复顺序的方法。

NAL 单元序列的结构如下图 2-1:

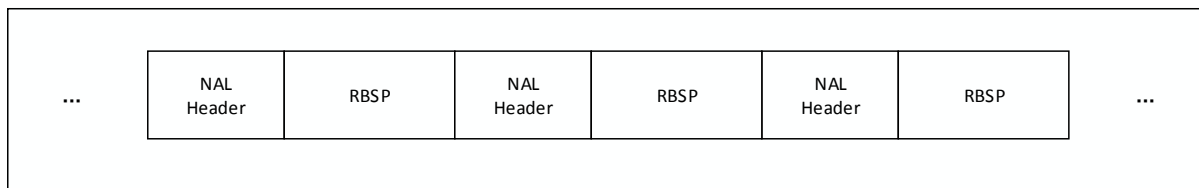


图 2-1 NAL 单元序列

RBSP 的类型介绍如下:

SPS：序列参数集包含了一系列视频编码序列的各种参数信息，如标识符序列参数集 ID、参考帧的数目、解码图像尺寸大小、帧数、视频格式和 POC 的约束等等。

PPS：即图像参数集，对应着序列中的一个图像或几个图像，参数包括标识符图像参数集 id、可选的序列参数集 id、片组数目、初始量化参数等等。

数据分割：在片的编码数据中，存放着三个独立的数据分割，分割 A、分割 B 和分割 C。A 中含有片头和各个宏块的头数据，B 中含有 SI 片宏块信息和帧内的残差数据，C 中则含有帧间的残差数据。这些分割信息可以用于错误恢复编码。

编码片：编码片包括片的头部信息和编码数据信息。

增加信息 SEI：可以用于视频解码时，提供增强信息。

图像定界符 PD：代表着图像的边界。

序列结束符：表明下一幅为 IDR 图像。

流结束符：表明当前流中已经没有图像。

填充数据：没有信息意义的哑无数据。

编码器将各个 NAL 单元独立而完整地放入一个数据分组当中，由于每个分组都有头部，解码器可以很容易地检测出 NAL 的边界，并依次取出 NAL 信息进行解码。NAL 包含着一个起始码 0x000001，当遇到下一个起始码说明前一个 NAL 单元已经结束。0x000000 作为一个特殊的分界，它被检测到时也意味着前一个 NAL 单元的结束。

但是当 NAL 单元的数据信息中包含 0x000001 或者 0x000000 时，就会出现严重的问题，因此 H.264 引用了一种防止竞争的机制，当编码器检测到 NAL 数据信息中包含有 0x000000、0x000001、0x000002 以及 0x000003 其中的任何一个时，会在最后一个字节前加入 03，如下表 2-2 所示：

表 2-2 NAL 特殊数据的修改

0x000000	0x00000300
0x000001	0x00000301
0x000002	0x00000302
0x000003	0x00000303

这样在检测到上次四种类型的数据信息时就会把里面的 0x03 删掉，并将数据信息传入解码器进行解码播放。

NALU 类型

表明 NAL 单元中的 RBSP 的数据类型，其中 NAL 单元类型为 1，2，3，4，5 及 12 的 NAL 单元称为 VCL 的 NAL 单元，其他类型的 NAL 单元为非 VCL 的 NAL 单元。

NAL 单元的类型具体定义如下表 2-3 所示：

表 2-3 NAL 单元语义

NAL 单元类型	意义
0	没有定义
1	不分区，非 IDR 图像中不使用数据划分的片段
2	非 IDR 图像中片分区 A 的片段
3	非 IDR 图像中片分区 B 的片段
4	非 IDR 图像中片分区 B 的片段
5	IDR 图像中的片段
6	补充增强信息
7	SPS 序列参数集

8	PPS 图像参数集
9	分割符，需在 SEI 之前
10	NAL 序列结束符
11	流结束符
12	数据填充
13-23	保留未使用
24-31	没有定义

NALU 的顺序要求

解码器对 NAL 单元的顺序要求非常严格，如果 NAL 单元的顺序有误的话，解码器想要正确地解码，需要将其按照 H.264/AVC 的标准进行重新组织后再传给解码器。

1.序列参数集 (SPS) NAL 单元必须在传送所有以其为参数的 NAL 单元前发送，不过也允许出现重复的 SPS 序列参数集单元。

2.图像参数集 (PPS) NAL 单元必须在传送所有以其为参数的 NAL 单元前发送，，不过也允许出现重复的 PPS 序列参数集单元，与上述的 SPS 序列参数集相同。

3.不同的编码片单元和数据分段单元在顺序不能相互交叉。即不允许某一图像的编码片和数据片单元中出现在另一图像的编码片和数据片单元中^[4]。

4.参考图像的影响：如果一幅图像参考另一幅图像，则后者的所有编码片和数据片必须在属于前者的编码片和数据片之前。

5.基本编码图像的所有片段 (slice) 单元和数据划分片段 (data partition) 单元必须在属于相应冗余编码图像的片段 (slice) 单元和数据划分片段 (data partition) 单元之前。

6.如果数据序列中有连续的没有参考的基本编码图像，那么图像序号小的必须在前面。

7.如果 arbitrary_slice_order_allowed_flag 参数为 1，则一个基本编码图像中的编码片段和数据片段顺序可以随意，如果 arbitrary_slice_order_allowed_flag 参数为 0，则必须要以片段中第一个宏块的位置来确定所有片段的顺序，倘若使用数据划分的话，则 A 类数据片必须在 B 类数据片段之前，而 B 类数据片需要在 C 类数据片之前，而且不同片的数据片还不能相互交叉。

8. SEI (补充增强信息) 单元也必须在与它相关的基本编码图片的片段单元和数据划分片段单元的前面，也必须紧跟着上一个基本编码图片的所有片段单元和数据划分片

段单元的后边。

2.2.2 RTP 包头介绍

RTP 包头分析，包头如表 2-4 所示：

表 2-4 RTP 包头

1	2	3	8	9	16bit
V	P	X	CSRC Count	M	Payload Type
Sequence number				Timestamp	
SSRC				CSRC (variable 0 – 15 items 32bits each)	

V — 版本。识别 RTP 版本。

P — 间隙 (Padding)。设置时，数据包可以包含一个或多个附加间隙组，但是这部分并不属于有效载荷^[12]。

X — 扩展位。设置在固定头的后面，根据特定格式可以设置一个扩展头。

CSRC Count — 包含了 CSRC 标识符的编号。

M — 标记。标记由 Profile 文件定义。用于在数据包中标记某些重要事件如帧边界。

Payload Type — 即负载类型，是识别 RTP 有效载荷的格式，应用程序可以对其进行解释。Profile 文件规定了从 Payload 编码到 Payload 格式的缺省静态映射。另外的 Payload Type 编码可能通过非 RTP 方法实现动态定义。

Sequence Number — RTP 包的序列号，每次发送一个 RTP 数据包，序列号都会自增 1。通过这个参数，接收方可以统计数据包是否丢失并尝试恢复数据包序列。

Timestamp — 反映了 RTP 传输的数据包中第一个八位组的采样时间。通过时间频率的计算来表示时间，可以支持同步和抖动计算。

SSRC — 同步源。该标识符上计算机随机选择，是为了确保在同一个 RTP 的会话期间两个同步源的 SSRC 不同。

CSRC — 贡献源标识符。用于识别该数据包中的有效载荷的贡献源。

2.2.3 H264 的 RTP 传输

为了降低接收端的 RTP 包丢包率，RTP 必须对重要的信息进行备份，它可以通过发送冗余信息的方式来实现，但这样会牺牲一点时延，多媒体的传输规范如下：

(1) 对重要的数据进行多次重发, 保证接收端可以接收到一份正确的数据, 分组复制多次重发, 发送端对重要的数据进行多次重发, 以保证接收端可以正确地接收到一份数据, 同时接收端还要对已接收的数据包的复制包进行丢弃。

(2) 对分组进行前向纠错, 可以先对被保护的分组进行异或处理, 再将运算结果作为冗余信息发送回接收端。由于时延, 这种方式并不适合于对话应用, 但可用于流媒体传输。

(3) 对音频进行冗余编码, 可以用来保护类似音视频的数据流。每个分组由头标 header、载荷类型(payload)以及上一个分组的载荷类型(payload)组成。

RTP 的封装规范总结如下:

(1) 尽量减少额外开销, MTU(最大传输单元)的尺寸要限制在 100 字节到 64 千字节以内;

(2) 对分组容易区分, 而不需要对分组内的数据进行解码;

(3) 在不需要解码整个数据流的前提下, 应能检测到数据包的类型, 并能根据码流间的相关性去除一些无用数据。

(4) 支持将 NALU 分割成若干个 RTP 包, 这样才能避免在 IP 层进行分片。

(5) 支持将多个 NALU 汇集到一个 RTP 分组中, 当 NALU 比较小时, 一个 RTP 包可能包括多个 NALU, 这样可以提高网络传输效率。

H.264 采用比较简单的打包方案, 在一个 RTP 的分组里只放入一个 NAL 单元, 将 NAL 单元放入到 RTP 的 Payload(载荷)中, 设置 RTP 头标值。在比较理想的情况下,, 为了避免 IP 层的拆包, VCL(视频编码层)不会产生超过最大传输单元尺寸的 NAL 单元。接收端可以通过 RTP 序列号信息来识别当前是否为复制包, 并复制包丢弃而只取出有效的 RTP 包中的 NAL 单元数据。

因为各种网络的 MTU 的大小不一致, 而编码器又不了解, 这样就会导致产生大于 MTU 尺寸的 NAL 单元。这样就会对 NAL 单元进行分割和合并。

(1) NALU 的分割

NAL 单元可能会大于最大传输单元的尺寸限制, 虽然 IP 层的数据包拆分可以使数据块的大小在 64 千字节以内, 但在应用层上无法得到保护, 这样就会影响非平等保护的质量。拆分方案应当符合如下特点:

a) NAL 单元的各个分块应当按照 RTP 的序号的升序在网络上进行传输。

b) 可以对最开始和最后的 NAL 单元分块进行标记。

c) 可以对丢失的分块进行检测。

(2) NALU 的合并

为了减少 RTP 发送的开销, 我们可以合并一些比较小的 NAL 单元如 SEI、SPS、PPS 参数集等信息。下面是两种集合分组: ①单时刻聚合包分组 (STAP), 按时间戳进

行组合；②多时刻聚合包分组（MTAP），可以组合不同的时间戳，一般用于对时延要求不高的环境。

2.3 流媒体的播放方式

2.3.1 单播

单播是指在客户端和服务端之间需要建立一条单独的数据通道，从一台服务器送出的每个数据都只能传输到一台客户机，这种方式称为单播。每个用户都必须对服务器发送单独的查询，而服务器也必须向每个客户端发送申请的数据包拷贝。这种冗余会给服务器带来沉重的负担，响应也需要很长的时间，以至停止播放；管理人员为了保证服务质量，只有被迫购买硬件和带宽。

2.3.2 组播

组播是指在网络中将数据包尽力地发送到某一个确定的集合（即组播组），基本思想是：源主机只发送出一份数据，它的目的地址为组播组地址^[13]；加入组播组中的所有成员都能接收到同样的数据拷贝，而其他主机则接收不到。

组播技术非常有效地解决了单点发送、多点接收的问题，实现了 IP 网络中单点到多点的数据高效传送，大大地节约了网络带宽和网络负载。更为重要的是，利用组播的这一特性可以方便地提供很多增值服务，比如在线直播、远程教育、网络电台、网络电视、视频会议等信息服务领域。

2.3.3 点播与广播

点播是客户端与服务端之间的主动连接。在点播连接中，用户可能通过选择内容来初始化客户端的连接。用户可以开始、暂停、快进、后退或停止流。点播提供了对媒体流的最高控制，但这种方式需要每个客户端连接服务器，所以会迅速地消耗网络带宽。

而广播指的是用户被动地接收流。在广播过程中，客户端只能接收流不能控制流。比如，用户不能暂停、前进或后退流。在广播方式中，数据包的拷贝将发送给网络中的所有用户。而在单播中，会将数据包复制多个拷贝，以多个点对点的方式来分别发送给需要的用户，而使用广播发送，则不管用户是否需要，数据包的一个拷贝将会发送给网络中的所有用户，所以不管是单播还是广播都非常浪费网络带宽。而组播吸收了上述两种发送方式的优点，克服其弱点，只将数据包的一个单独拷贝发送给网络中的需要的客

户。组播不会像单播一样复制数据包的多个拷贝传输至网络中，也不会像广播一样发送给不需要的用户，从而保证在网络中只占用最小的带宽。

2.3.4 P2P 播放

P2P 播放是基于 P2P 技术的流媒体播放方式之一。在使用的时候，每一个客户端既是客户机又是服务器。你从别人那里下载需要播放的那一片段的同时，也在给别人提供下载另一个片段。因此如果在线人数越多，播放反而会更流畅。但是这种播放方式比较占用网络带宽和资源。

2.4 本章小结

本章首先介绍了流媒体协议栈的各种协议，包括 RTSP(Real Time Streaming Protocol) 实时流传输协议，RTCP(RTP Control Protocol)控制协议，RTP(Real-time Transport Protocol) 实时传输协议以及 SDP(Session Description Protocol)会话描述协议。其次介绍了 H264 的网络提取层(NAL)，RTP 包头的封装方式，H264 的 RTP 传输以及流媒体的主要播放方式。

第三章 实时流媒体总体方案介绍

实时流媒体传输方案主要包括流媒体服务器和流媒体客户端两部分。流媒体服务器需要实现处理客户端的视频文件上传，客户端的点播以及实时转发等功能，而流媒体客户端则需要实现本地和服务端上视频的点播和上传功能。本章在第一章和第二章对项目背景和流媒体技术进行分析的基础之上，对实时流媒体传输方案进行了比较和选型，介绍了本系统中用到的开源库 live555 和 FFmpeg 以及 Android 的系统开发框架，提出了 RTSP 实时流媒体传输总体方案。

3.1 方案选型

由于需要对服务器进行二次开发，市面上开源的流媒体服务器主要有 live555 和 Darwin，下表 3-1 对它们进行比较：

表 3-1 Live555 与 Darwin 比较

	Live555	Darwin
简单介绍	Live555 是实现 RTSP 流媒体服务的极简代码，用 C++语言实现。	由苹果公司开发的一套流媒体方案
优点	代码结构精简易于二次开发，项目活跃。	支持的视频格式较多。
缺点	不支持上传、视频格式支持不全。	项目复杂、不活跃，资料少。

通过对 live555 和 Darwin 的分析比较，选择了易于二次开发的 live555 作为本方案中的流媒体服务器。

而手机客户端由于要实现边拍边传，而且 Android 自带的 API 编解码库比较少，支持的格式不多，所以在客户端移植了 FFmpeg，用于编解码、MP4 文件的解析和封装和 RTSP 协议的支持。

3.2 方案中开源库的介绍

3.2.1 live555 库

随着流媒体技术的日益发展，各种实现流媒体技术的开源库也不断涌现，其中 Live555 是实现了 RTSP 流媒体协议栈的最精简的代码，而且可以方便地应用于嵌入式

实践当中。流媒体服务器在流媒体传输系统中起着至关重要的作用，它负责从手机客户端或者音视频采集终端接收封装媒体文件，或者通过流媒体的传输协议将媒体资源传送到其他地方。Live555 精简地实现了 RTSP、RTP、RTCP 协议，此外 Live555 还实现了对各种播放方式的支持包括单播、组播、源特定多播、广播。

Live555 中实现了对各种音视频格式的流化、接收、封装的支持，包括 H264、DV、AC3、MP4V-ES、MPV、T140 等多种音视频编码。Live555 因为设计良好，也非常容易扩展对其他编码格式的支持^[14]。

Live555 的整体代码框架如图 3-1 所示：

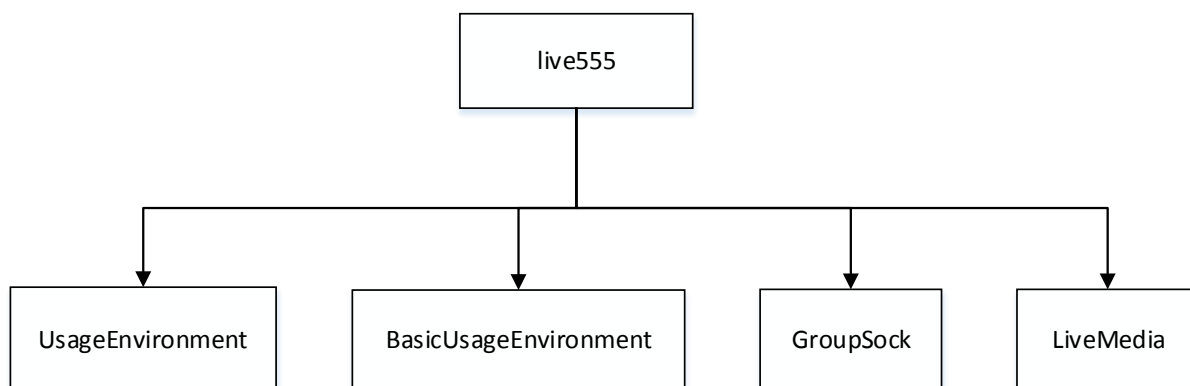


图 3-1 Live555 的整体代码框架图

在 UsageEnvironment 中主要包括了 UsageEnvironment 类、TaskScheduler 类和 HashTable 类。UsageEnvironment 用于输入输出操作和错误信息的处理。TaskScheduler 类用于事件的任务调度，包括异步事件的处理以及回调函数的注册。TaskScheduler 类通过 DelayQueue 来实现事件的延时调度，以控制好数据包的发送速率。

BasicUsageEnvironment 模块是 UsageEnvironment 模块的一个具体实现，它实现了具体的输入输出操作以及任务调度的处理操作。

GroupSock 里面封装了一系列的网络接口，包括网络地址类、数据包的发送类等等，它原本是为了支持组播，但也可以支持单播功能。

LiveMedia 是 Live555 的核心部分，该模块里面抽象了一个 Medium 类，其他所有类都是派生自这个类。LiveMedia 中实现了 source，包括 RTPSource、FileSource 等各种提供资源的类，也实现了 Sink,包括 RTPSink、FileSink 等消费资源的类，同时也实现了 RTP、RTCP、RTSP 客户端和 RTSP 服务器的各种处理操作，另外还实现了针对于各种音视频编码格式的 Source 和 Sink 的扩展。

3.2.2 FFMPEG 库

FFMPEG^[15 16]是一个集音视频采集、编码、解码、格式转换为一体的完全开源的流媒体解决方案。它包括了目前领先的音视频编解码库 libavcodec。FFMPEG 是在 linux 下开发出来的,但也能在 windows 在进行编译。FFMPEG 实现了各种音视频格式的编解码,各种多媒体文件的封装解封装,复用解复用,同时还可以实现音视频的同步,视频分辨率的改变,音视频文件的缩放、快进以及快退等功能。总之 FFMPEG 是一个非常强大的音视频处理库,广泛地应用于市面上的各种播放器如暴风影音、VLC、Mplayer 中。

FFMPEG 包含了以下各个模块:

libavformat: 用于解析和生成各种音视频编码和解码的格式信息,并根据解码信息生成解码上下文信息,同时还存放复用 Muxer 和解复用 Demuxer 的库。Muxer 可以将音视频、字幕格式合并成一个文件,而 demuxer 的作用则相反。

libswscale: 用于对视频进行缩放显示,它需要我们确认视频源的格式和转换格式以及宽度和高度等信息。

libpostproc: 这个模块是用于后期的效果处理。

libavcodec: 里面包括各种 encode/decode 模块,用于各种格式的音频/图像的编解码;

libavutil: 是一个公共工具的函数库,包括一些内存操作、CRC 的检验、内存分配、大小端序列的转换等功能,被其他各个模块所使用。

libavdevice: 用于对输入输出设备的操作支持

ffsever: 是一个实现了 HTTP 和 RTSP 的流媒体服务器。

ffplay: 是一个简单的播放工具,通过 FFMPEG 进行解复用和解码,并使用 SDL 库进行渲染播放。

ffmpeg: 这是该项目提供的一个工具,可以通过相关参数进行音视频的格式转换、解码等功能。

FFMPEG 的主要数据结构:

a) AVFormatContext

这个数据结构包含了一个流的格式信息,它包含了 AVInputFormat 或 AVOutputFormat、AVStream、AVPacket 这几个重要的数据结构以及一些其他的相关信息,比如媒体文件的名称、作者、版权、文件长度、比特率等信息。

b) AVOutputFormat

这个数据结构指示着编解码器采用哪种类型，它可以通过文件得知文件的类型，申请对应的编解码器。

c) AVStream

AVStream 在 FFmpeg 中是一个频繁使用的类，它包含了流的一些数据信息，包括帧率、帧数、持续时间、时间基数单位等。

d) AVCodecContext

这个数据结构对编码器的上下文信息进行了描述，包含了各种编码器需要的信息，包括视频的宽度和高度、音频的采集率和通道数、音频的原始采样格式、时间基数单位、编码器的信息等。

e) AVPacket

FFmpeg 作用这个数据结构来暂时存储解复用后，解码之前的多媒体数据，比如音视频或者字幕数据。其中 PTS 代表显示时间戳，DTS 代表解码时间戳，stream_index 是媒体流的索引，duration 代表数据包的时间长度，data 是指缓冲区的指针地址，size 指数据包的长度。

3.2.3 Android 介绍

3.2.3.1 Android 系统架构

Android 的系统架构如图 3-2 所示。

Android 系统架构图

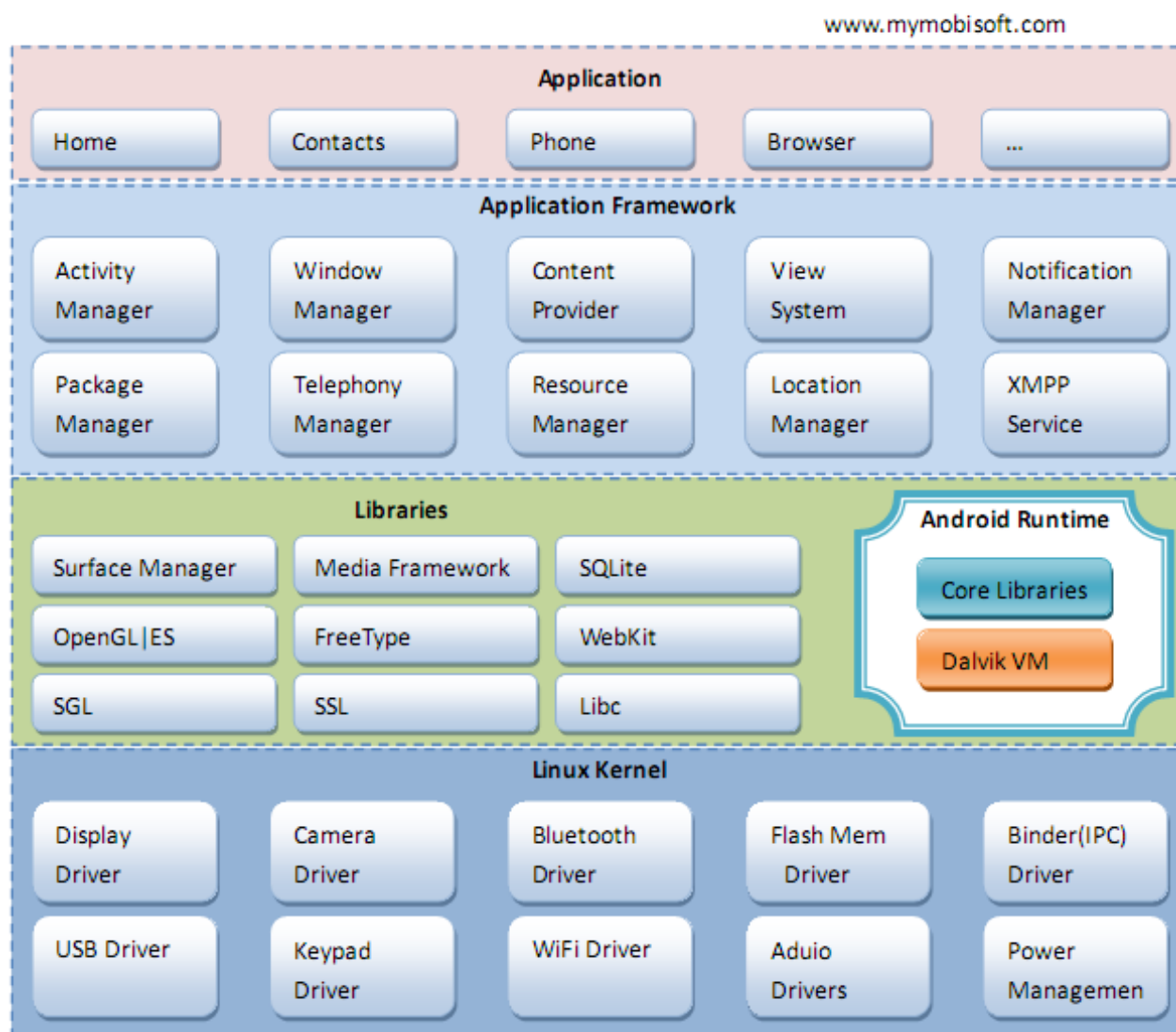


图 3-2 android 框架组成

Android 系统从下往上看，包括了四个层次，分别为 Linux 内核层、系统运行库层、应用框架层和应用层。下面分别对这四个层次进行简要的介绍：

a) Linux 内核及硬件驱动层

Android 是基于 linux2.6 内核的，它的许多核心服务如内存管理、进程调度、安全等都需要依赖于 linux 内核，内核层包括了很多硬件驱动如显示驱动、Bluetooth 驱动、闪存驱动、USB 驱动、Keypad 驱动、Wifi 驱动、Audio 驱动以及电源管理等。

b) 系统运行库层

从上图可以看出，系统运行库层包括两部分：系统库和 Android 运行时库。其中，系统库主要如 C 或 C++ 语言编写，通过调用 linux 内核的模块编译成相应的动态链接库以便上层应用框架层进行调用，起到了连接内核层和应用框架层的纽带作用。在系统库

层中包括 Surface Manager 库：用于管理和显示子系统，同时也包括 2D 和 3D 图像的合成；多媒体库：主要包括两部分的功能，即音视频的播放和录制；SQLite 库：是一种通用的小型嵌入式关系数据库；OpenGL 库：主要用于 3D 图像的绘制、FreeType 库：实现了点阵字和矢量字体的显示；WebKit 库：网络浏览器的核心引擎；SGL 库：2D 图形渲染引擎；SSL 库：是 TCP/IP 层和应用层之间的一种协议，可以为 Android 通信进行握手，进行安全支持等。Android 开发人员可以将自己的程序或者开源库移植到系统库中。

Android 运行时库分为 JAVA 核心库和 Dalvik 虚拟机两部分。JAVA 的核心库包含了 JAVA 的各种核心 API，提供了 JAVA 的大部分功能，包括网络 API android.net，多媒体 API android.media 等。在 Android 中，每个程序都会有一个专有的进程，这一点和 J2ME 不同，而且每一个进程都会拥有一个独立的 Dalvik 虚拟机实例，而且多个 Dalvik 虚拟机实例可以在多个进程中独立而有效地运行^[17]。与 JAVA VM 虚拟机不同，Dalvik 虚拟机并不是基于栈而是基于寄存器的一种虚拟机，它执行的不是 JAVA 的标准字节码而是经过 SDK 的 dx 工具转化为的后缀名为 dex 的可执行文件，这种格式的文件的好处在于可以在移动设备中实现更高效的优化，节省内存空间。

c) 应用程序框架层

应用程序框架层是我们开发 Android 的基础，它使组件的重用变得更加简单，程序员可以根据需要进行高效开发，主要包括了十个部分的功能——活动管理器：管理整个应用程序的生命周期；窗口管理器：管理所有的窗口操作；内容提供者：可以实现程序间的内容分享；视图系统：为应用程序提供基本组件；通知管理器：开发人员可以在状态栏自定义状态信息；包管理器：对 Android 系统内的程序和包进行管理；电话管理器：管理移动设备的功能；资源管理器：向 Android 程序提供各种非代码的资源，如图片、布局文件等；位置管理器：可以提供位置服务；XMPP 服务。

d) 应用程序层

Android 系统中不仅包含平台也包括了各种应用程序包括联系人、通话软件、网络浏览器、视频播放器、图片浏览器等，这一层中的应用程序都可以被开发人员重新编写，从而定制各种不同类型的 Android 手机 ROM。

3.2.3.2 Android 应用层开发

Android 的应用开发是基于 JAVA 语言的，所有在 Android 开发中自然可以使用各种 JAVA 的开发库。Android 应用层开发所用要到的 API 主要由下面几部分组成：

Java 标准 API。这部分是 java 的核心部分，它以 java.X 形式进行命名，包括了很多 JAVA 的基础库，如输入输出库(java.io)等。

Java 扩展 API。这部分是 JAVA API 的扩展部分，它以 javax.X 形式进行命名，丰富了 JAVA API 的功能，它是为了兼容原来的版本。JAVA 扩展 API 中包括如 javax.security(安全相关)、javax.sql(数据库操作)等。

企业和组织提供的 java 类库。这一部分以 org.X 的形式进行命名，比如 org.json 可以用于快速构造和解析 JSON 字符串，这一部分的类库大多都不是 SUN 公司开发的，但可以直接使用。

Android 的各种类库。首先包括 OS 相关如 android.os 和 VM 相关如 dalvik.system 的系统级的 android 库；其次包括一些程序的框架库，如 android.content.xxx、android.view.xxx、android.app.xxx 等；然后包括一些系统内置的服务，比如多媒体库 android.media.xxx、android.opengl、android.animation(动画系统)等，包括链路库、传输层以及应用层的网络库，数据库 android.database.xxx，XML 解析库 android.sax 等等。

3.2.3.3 基于 Android 音视频录制播放技术研究

MediaRecorder 是一个用于录制音视频的 API，android 上视频和音乐的录制都是靠这个 API 实现。MediaRecorder 底层调用的是 OpenCore 库来实现的，它的上层还使用了 Binder(IPC)进程间通信的功能。MediaRecorder 的下层接口是 PVMediaRecorder.h，具体调用了 OpenCore,实现音频的输入、视频的输入和预览；上层接口是 mediarecorder.h；服务部分实现接口是 ImediaRecorder.h。

MediaPlayer 播放音视频，正如 MediaRecord 类似，MediaPlayer 可以用于音频和视频的解码播放。MediaPlayer 可以播放很多种格式的音视频文件，包括 MP4、3GP、MP3、AAC 等。MediaPlayer 会在框架层创建音视频解码器。MediaPlayer 的底层也是调用 OpenCore 来实现的，它的使用也非常简单，通过创建 MediaPlayer 实例，设置播放源再进行相应播放控制即可。

AudioTrack 也能播放音频，和 MediaPlayer 不同的是，它并不会创建解码器，所以只能播放解码后的 PCM 码流如 wav 文件，它和 MediaPlayer 是紧密联系的，在 MediaPlayer 中也会创建 AudioTrack 的实例。

3.3 总体方案设计

3.3.1 总体框架

下图 3-3 是整个实时流媒体传输方案的整体框架，为了增加并发量，流媒体服务器采用 LVS 集群方式进行扩展，并将视频存储至磁盘阵列之中，以解决磁盘 I/O 瓶颈，客户端用的是 Android 手机，通过在 Android 手机上移植 FFmpeg 库和 X264 库来实现音视频的软编码，通过 RTSP 协议来进行音视频的传输。

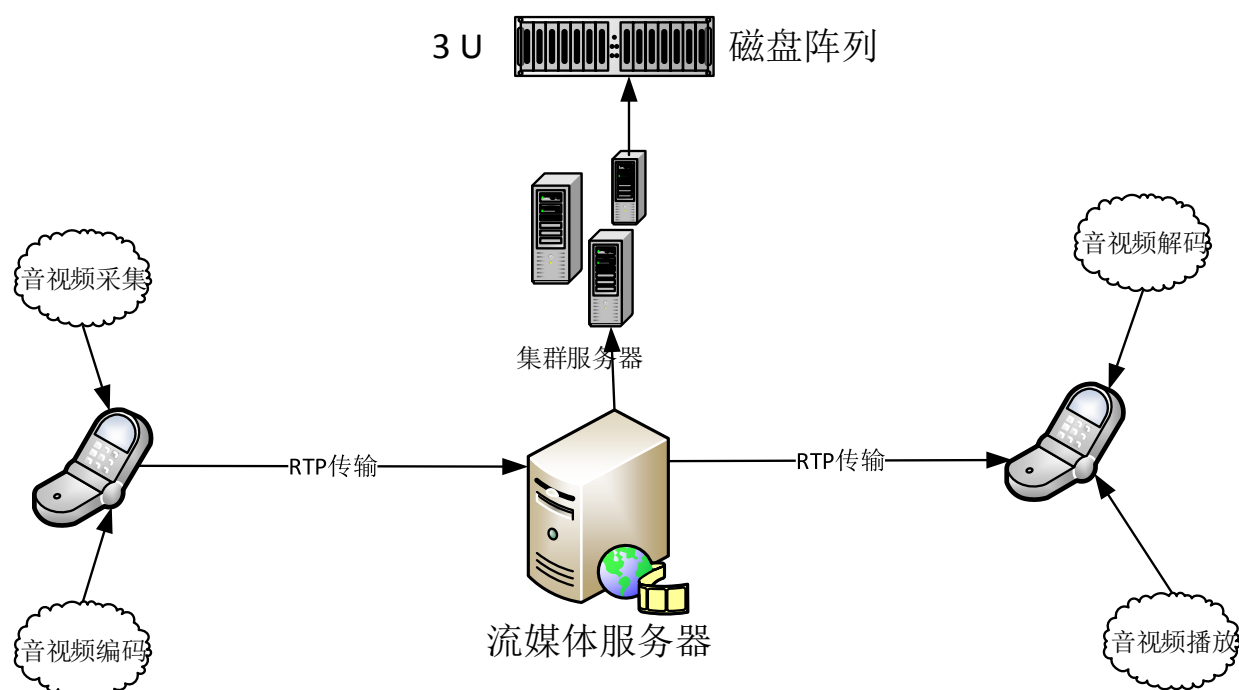


图 3-3 流媒体方案总体框架

3.3.2 流媒体服务器端

流媒体服务器采用的是 Live555+LVS 的架构模式进行系统搭建的。Live555 采用的是单进程的 select 模型，并发量不高，所以用多进程和集群的方式来进行扩展，充分利用多核 CPU 的能力，提高服务器的并发量。Live555 服务器不支持手机客户端的上传，支持的视频格式也有限，还不支持实时转发，本系统比较好地解决了上述问题。流媒体服务器的整体架构在第四章有详细介绍。

3.3.3 流媒体客户端

1、客户端采集上传与编码模块

图 3-4 显示了客户端采集上传编码的大体流程图。客户端音视频采集调用 Android 系统 API，主要用到 Camera 和 AudioRecord 类来完成音视频的采集，并在手机上移植了 FFmpeg、X264 库，对采集后的视频进行格式转换，之后分别对视频和音频进行编码，再对其进行 RTP 打包，通过 RTP 传输协议将音视频流传至流媒体服务器。

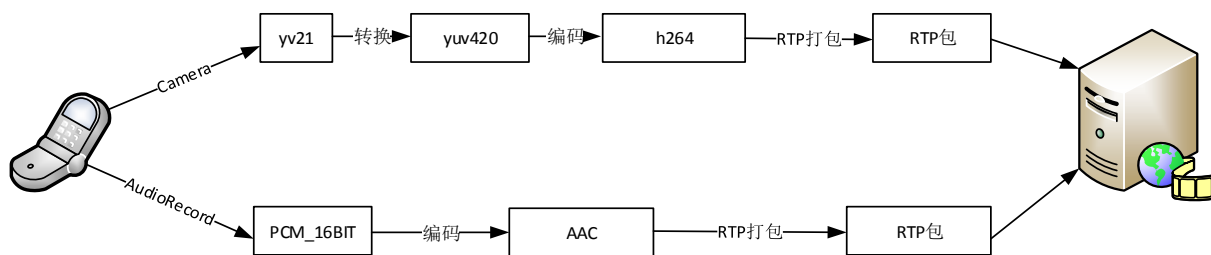


图 3-4 客户端采集编码上传

2、客户端解码播放模块

客户端解码播放模块是采集上传模块的逆过程，先将接收到的 RTP 包进行拆包处理，然后进行音视频解码，解码成一帧帧的图片和 PCM 语音信息，然后根据 NTP 的时间戳和 RTCP 的 SR 包和 RR 包的同步信息进行音视频同步。解码播放过程如下图 3-5 所示：

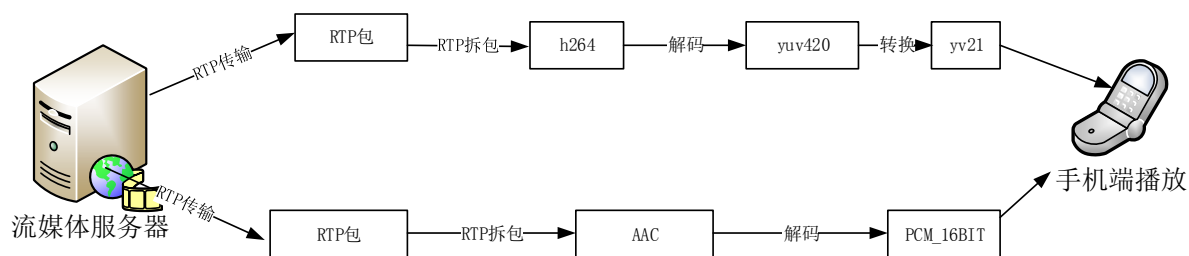


图 3-5 客户端解码播放

3.4 本章小结

本章首先对开源流媒体服务器的方案进行比较和选型，接着介绍了两个开源库 live555 和 FFmpeg，然后介绍了 Android 的基本开发框架，在此基本之上提出了一套实时流媒体的传输方案，还简要介绍了流媒体服务器端和手机客户端的各个基本模块。

第四章 实时流媒体服务器方案实现

实时流媒体服务器既要求保证功能的完整性，又需要保证可靠的流媒体视频质量，同时还需要具备高效的性能。本章正是基于上述三方面，扩展了 live555 流媒体服务器支持客户端的上传，MP4 文件的点播下发和实时转发功能；采用 RTCP 处理技术来保证传输的质量；采用多进程技术和 LVS 集群来提高流媒体服务器的并发量。具体的介绍如下。

4.1 流媒体服务器整体方案

流媒体服务器采用的是基于 live555 进行的二次开发，live555 是一个极其精简的流媒体服务器框架，易于扩展对各种多媒体格式的支持。Live555 不支持处理客户端视频文件的上传，也不支持实时转发，同时也不支持 MP4 文件格式的下发。所以在此基础上，对其进行二次开发，支持上述功能。流媒体服务器中的支持上传模块接收客户端发送过来 RTP 音视频流，首先进行 RTSP 交互建立 RTP 和 RTCP 传输连接，然后在传输层进行 TCP 或 UDP 头的解封装，在 IP 层进行 IP 头的解封装，接着对 RTP 包进行拆包并创建 FileSink 将视频流以 MP4 文件格式进行存储，当中通过 RTCP 的 SR 包和 RR 包的处理，来动态地计算出网络中数据包的接收速率，从而达到网络拥塞控制的目的。MP4 文件下发模块则用于向客户提供 MP4 播放功能，其过程则是上传的逆过程，首先对 MP4 文件进行解复用，分离出 H264 和 AAC 帧数据，然后创建 RTPSink 进行 RTP 打包与发送，同时 RTCP 用于控制发送的速率。实时转发模块则是这两个模块部分功能的结合，实时转发的源来自于客户端上传的 RTPSource，进行通过创建 RTPSink 将音视频流传送给另一个客户端。此外为了增加并发量，服务器进行了多进程扩展和负载均衡 LVS 的搭建。流媒体服务器的架构如下图 4-1 所示：

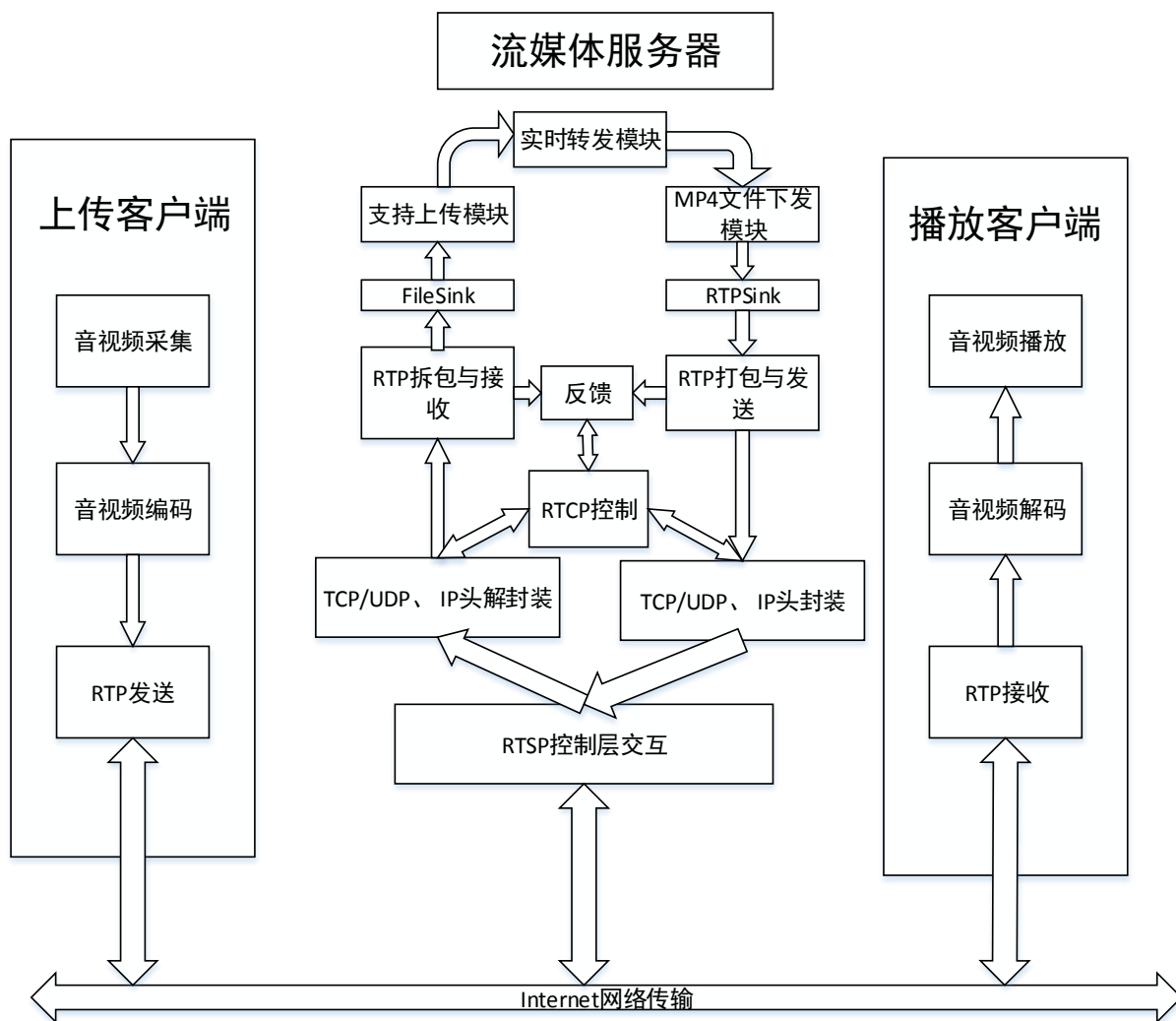


图 4-1 流媒体服务器架构图

4.2 RTSP 服务器的主要逻辑

在 live555 服务器中，包含了四个基本的模块，即 UsageEnvironment，BasicUsageEnvironment，GroupSocck 和 liveMedia。其中 UsageEnvironment 是基本的错误输入输出操作基类和延时任务、触发任务和网络任务以及任务大循环的基类，而 BasicUsageEnvironment 则是 UsageEnvironment 的基本实现。下面我先介绍一下，搭建 RTSP 服务器所需要的大致过程。

1、第一步是建立 BasicTaskScheduler 和 BasicUsageEnvironment。

```
//create task scheduler
scheduler = BasicTaskScheduler::createNew();
```

BasicTaskScheduler 可以用于服务器设置各种调度任务，将任务加入延时队列当中，选择在合适的时间进行调度以保证 RTP 的正确传输。

```
//create interactive environment
env = BasicUsageEnvironment::createNew(*scheduler);
```

将 scheduler 以引用的方式转给 BasicUsageEnvironment 对象，这样 BasicUsageEnvironment 对象就可以控制任务类的执行。这两个类构建起了整个 live555 的框架。

2、建立 RTSPServer

```
//create RTSP server
rtspServer = MyRTSPServer::createNew(*env, 554); //554 port
if(rtspServer == NULL)
{
    rtspServer = MyRTSPServer::createNew(*env, 8554); //if 554 be used, use 8554
    port
}
```

这里创建了 RTSPServer 类，并选用了标准的 554 和 8554 端口。

createNew 函数中调用了

```
int ourSocket = setUpOurSocket(env, ourPort);
```

它为 RTSP 的连接创建了 RTSP 套接字，设置了地址端口复用，非阻塞，设置合适的发送 Buffer，并且进行绑定监听套接字。

其次还调用了下面这个函数

```
env.taskScheduler().turnOnBackgroundReadHandling(fRTSPServerSocket,
(TaskScheduler::BackgroundHandlerProc*)&incomingConnectionHandlerRTSP, this);
```

其中 fRTSPServerSocket 为上面的 ourSocket 传给它的，并赋予了一个回调函数 incomingConnectionHandlerRTSP 来处理这个套接字上的相应请求。把套接字上的请求设置为后台大循环中的相应任务，将它和后台联系起来。

3、进入大循环处理事件

```
//enter event loop
env->taskScheduler().doEventLoop();
```

里面实际调用的是 BasicTaskScheduler 中的 SingleStep 函数。函数中转给 select 函数的有三种套接字，即读套接字，写套接字和异常套接字，并从延时队列中取出下一个任务的延时时间作为 select 函数的延时时间，select 响应相应的套接字事件，并调用相应的回调函数进行处理。

4、建立 RTSPClientSession

在 RTSPServer 的回调函数 `incomingConnectionHandlerRTSP` 中会处理相应的 RTSP 请求，并 `accept` 相应的套接字，并使用这个套接字来建立一个 `RTSPClientSession` 来单独用于具体的某一个客户请求的处理。`RTSPClientSession` 中的 `handleRequestBytes` 函数负责处理响应客户端的各种 RTSP 请求操作，包括 `Describe`, `Setup`, `Play`, `Teardown` 等等。其中 `Describe` 是产生客户请求的资源的 SDP 信息，`Setup` 是建立起音视频的 RTP 传输通道，`Play` 是触发开始 RTP 传输，`Teardown` 是终止会话。

下面是以是一个整合来详细介绍 RTSP 的交互流程。

a) OPTIONS 方法

首先客户端向服务器端发送 RTSP OPTION 命令询问服务器支持哪些 RTSP 方法，一般包括 `Describe`, `Setup`, `Play`, `Pause`, `Teardown` 等，它在 RTSP 会话的任何时候都可能发生，但它不会影响服务器的状态。

示例：

```
C->S: OPTIONS rtsp://218.192.170.198/test.mpg RTSP/1.0
```

```
CSeq: 1
```

```
User-Agent: openRTSP.exe (LIVE555 Streaming Media v2009.11.27)
```

```
S->C: RTSP/1.0 200 OK
```

```
CSeq: 1
```

```
Date: Sun, Jul 08 2012 09:05:33 GMT\r\n
```

```
Public: OPTIONS, ANNOUNCE, DESCRIBE, SETUP, TEARDOWN, PLAY, RECORD,  
PAUSE, GET_PARAMETER, SET_PARAMETER
```

b) DESCRIBE 方法

`DESCRIBE` 方法用于客户端通过媒体资源 URL 的方式向服务器请求所选资源的描述信息，服务器会分析解析客户端选择的资源的会话描述信息，进行响应。

示例如下：

```
C->S: DESCRIBE rtsp://218.192.170.143/test.264 RTSP/1.0
```

```
CSeq: 2
```

```
Accept: application/sdp
```

```
User-Agent: openRTSP.exe (LIVE555 Streaming Media v2009.11.27)
```

```
S->C: RTSP/1.0 200 OK
```

```
CSeq: 2
```

Date: Sun, Jul 08 2012 09:05:33 GMT

Content-Base: rtsp://218.192.170.143/test.264/

Content-Type: application/sdp

Content-Length: 523

v=0

o=- 1341738333186829 1 IN IP4 218.192.170.143

s=H.264 Video, streamed by the LIVE555 Media Server

i=test.264

t=0 0

a=tool:LIVE555 Streaming Media v2012.02.29

a=type:broadcast

a=control:*

a=range:npt=0-

a=x-qt-text-nam:H.264 Video, streamed by the LIVE555 Media Server

a=x-qt-text-inf:test.264

m=video 0 RTP/AVP 96

c=IN IP4 0.0.0.0

b=AS:500

a=rtpmap:96 H264/90000

a=fmtp:96packetization-mode=1;profile-level-id=64001E;sprop-parameter-sets=

Z2QAHzRAKA9sBEAAAMAAQAAAwA8DxYtEg==,aOvssiw=

a=control:track1

c) ANNOUNCE 方法

ANNOUNCE 方法作为一种可选方法，本项目中应用的 live555 开发库就没有实现，但是它主要有两种作用：一种是用于客户端上传资源给服务器时，客户端会生成相应的媒体信息给服务器，服务器根据 SDP 会话描述信息进行相应的处理。另一种是当服务器想要修改当前资源的 SDP 信息时，也可以向客户端请求 ANNOUNCE 方法，更新当前的媒体资源信息。

示例：

C->S: ANNOUNCE rtsp://218.192.170.143:554/test.sdp RTSP/1.0

CSeq: 2

Content-type: application/sdp

```

Content-length: 499
Date: Mon, 02 Jul 2012 04:34:40 GMT
v=0
o=- 14650 14650 IN IP4 127.0.0.1s= testMPEG4VideoToDarwin.exe
i=LIVE555 Streaming Media
c = IN IP4 218.192.170.143
a=x-qt-text-nam: testMPEG4VideoToDarwin.exe
a=x-qt-text-inf: LIVE555 Streaming Media
a=x-qt-text-cmt: source application:testMPEG4VideoToDarwin.exe
a=x-qt-text-aut:
m=audio 32 RTP/AVPP 0
m=video 0 RTP/AVP 96
S->C: RTSP/1.0 200 OK
CSeq: 2

```

d) SETUP 方法

SETUP 方法用于确定服务器和客户端之间的传输机制，包括 RTP 使用 TCP 还是 UDP 传输，播放方式是单播、组播还是广播，同时还要确定客户端和服务器的详细地址和端口信息，以便正确地建立套接字连接。在流媒体资源的传输过程中，也可以发送 SETUP 请求，服务器可以同意更改也可以不同意，不同意需要发送客户端对应的响应信息。

SETUP 方法示例如下：

```
C->S: SETUP rtsp://218.192.170.198/test.mpg/track1 RTSP/1.0
```

```
CSeq: 2
```

```
Transport: RTP/AVP/TCP; unicast; interleaved=0-1
```

```
Date: 2012 05:34:23 GMT
```

```
S->C: RTSP/1.0 200 OK
```

```
CSeq: 2
```

```
Date: 2012 05:34:34 GMT
```

```
Session: 10
```

```
Transport:RTP/AVP/TCP;unicast;destination=218.192.170.36;source=218.192.170.198;interleaved=0-1
```

e) PLAY 方法

PLAY 方法要求服务器在通过 SETUP 方法建立的传输机制进行数据的传输，客户

端不允许在 SETUP 方法被服务器返回成功之前请求 PLAY 方法。PLAY 请求可以根据某个范围进行播放，而不全部从头播放媒体资源。客户端的 PLAY 方法可以通过队列的方式进行请求，服务器可以依次处理相应的请求。如下个例子所示，没有范围的播放请求是合法的，它代表从头开始播放。

PLAY 方法的示例如下所示：

```
C->S: Request: PLAY rtsp://218.192.170.143/test.264/ RTSP/1.0
CSeq: 3
Session: 6A35C6D0
Range: npt=0.000-
User-Agent: testMPEG4VideoToDarwin.exe (LIVE555 Streaming Media v2009.11.27)
S->C: RTSP/1.0 200 OK
CSeq: 4
Date: Sun, Jul 08 2012 09:05:33 GMT\r\n
Range: npt=0.000-
Session: 6A35C6D0
RTP-Info: url=rtsp://218.192.170.143/test.264/track1 ;seq=56804;rtptime=135074508
```

f) PAUSE 方法

PAUSE 请求方法可以使流传输被临时中断。对于视频来说，视频画面会暂停，而对于音频，则会静音。如果 URL 指向的是一组流，则正在活动中的流的传输都会被中断，当恢复播放时，组流中的各个音视频必须进行同步。

在 PAUSE 请求中也可以包含一个 range 头部，表明 PAUSE 请求在什么时候生效。通常我们把它取叫 PAUSE Point。我们必须精确地给出这一头部而不能只是一个范围。PAUSE 请求比 PLAY 请求更有优先权，服务器在收到 PAUSE 请求之后，会将 PLAY 请求队列丢弃。

```
C->S: PAUSE rtsp://218.192.170.198/test.mpg RTSP/1.0
CSeq: 6
Session: 10
Date: 2012 05:34:23 GMT
S->C: RTSP/1.0 200 OK
CSeq: 6
Date: Tue, Mar 20 2012 05:34:34 GMT
Session: 10
```


g) Teardown 方法

TEARDOWN 方法将会终止所请求的 URL 的媒体资源的传输，并释放服务器上的相关资源。服务器与客户端建立的传输机制将不再有效，除非在 SDP 信息中包含了所有的传输参数信息，否则想再次请求播放之前的 URL 资源，必须通过 SETUP 方法建立传输信息。

示例：

```
C->S: Request: TEARDOWN rtsp://218.192.170.198/test.mpg RTSP/1.0
```

```
CSeq:7
```

```
Session: 10
```

```
Date: 2012 05:34:23 GMT\r\n
```

```
S->C: RTSP/1.0 200 OK
```

```
CSeq: 7
```

```
Date: Tue, Mar 20 2012 05:34:34 GMT
```

4.3 RTP 打包与发送

为了实现音视频的实时传输，需要对编码后的 H264 和 AAC 帧进行打包处理。RTP 的传输开始于客户端向服务器点播资源调用完 RTSP 方法 Play 之后，里面调用的是 MediaSink 的 startPlaying()方法，从 source 中拿数据。startPlaying()方法里面调用一个内部函数 continuePlaying()，内部调用打包函数 buildAndSendPacket()设置好 RTP 头，然后调用帧填充函数 packFrame()，进而调用一个 afterGettingFrame1()函数处理好帧的分片，经过一系统调用，最后调用 sendPacketIfNecessary()来真正实现 RTP 包的传输。具体的流程如下：

1) startPlaying()

当客户端调用 Play 方法时，服务器端会调用 handleCmd_Play()方法来响应请求，startPlaying()方法就是在 handleCmd_Play()方法中进行调用的，开始触发 RTP 的传输。函数里面还设置了 source 源，以便从其中获取数据。

2) continuePlaying()

```
MultiFramedRTPSink::continuePlaying() {
    buildAndSendPacket(True);//发送第一个包。
}
```

MediaSink的continuePlaying()函数是一个纯虚函数，需要由子类来具体实现。在系统中用到了是H264的视频编码方式，所以具体实现的函数是MediaSink的子类H264VideoRTPSink::continuePlaying()

代码如下:

```
H264VideoRTPSink::continuePlaying()
    {fOurFragmenter = new H264FUAFragmenter(envir(), fSource,
        OutPacketBuffer::maxSize,
        ourMaxPacketSize() - 12 //RTP 头的大小);

    fSource = fOurFragmenter;
    return MultiFramedRTPSink::continuePlaying();}
```

从代码中可以看出, 里面创建了一个 H264FUAFragmenter 类, 这个辅助类用于 H264 的 RTP 封包中的 NAL 单元的分片, 因为我们只讨论 RTP 的打包共性, 不对其详细分析。

3) buildAndSendPacket()

MultiFramedRTPSink 是与帧有关的类, 其实它要求每次必须从 source 获得一个帧的数据。可以看到 continuePlaying() 完全被 buildAndSendPacket() 代替。代码如下:

```
MultiFramedRTPSink::buildAndSendPacket(Boolean isFirstPacket) {
    // 设置 RTP 头信息
    unsigned rtpHdr = 0x80000000; // RTP 的第二个版本, 'M'位没有设置
    rtpHdr |= (fRTPPayloadType<<16); //RTP 载荷
    rtpHdr |= fSeqNo; // RTP 序列号
    fOutBuf->enqueueWord(rtpHdr); //向包中加入一个字
    ...
    packFrame(); //头准备好了, 再打包帧数据
}
```

先判断是否是第一帧, 如果是第一帧的话就以当前时间作为发送时间, 之后填充 RTP 头, RTP 包头包括版本号、负载类型、时间戳、标志位、序列号和同步源 SSRC 等 12 个字节组成。函数中设置了版本号、序列号、负载类型和 SSRC 等信息, 预留了时间戳和标志位的位置, 它们的填充将由 doSpecialFramehandling() 函数来完成。

4) packFrame()、getNextFrame()

packFrame() 用于填充真正的数据帧。

```
if (fOutBuf->haveOverflowData()) {
    afterGettingFrame1();
} else {
    fSource->getNextFrame()
}
```

首先判断上次打包的 `buffer` 中是否有剩下的帧数据，因为一个包可能容纳不下一个帧，有则使用之，并调用 `afterGettingFrame1()` 函数向包中填充数据。如果没有的话就跟 `source` 要数据，调用 `getNextFrame()` 再去取数据。

5) `afterGettingFrame1()`

`afterGettingFrame1()` 函数负责向包中填充数据，如何一帧的缓冲不够大，就会发生截断一帧数据的现象，如果包已经打入帧数据了，并且不能再向这个包中加数据了，就要把新获得的帧数据保存，此外还要计算获取的帧中有多少数据可以打到当前包中，并且把剩下的数据作为 `overflow` 数据保存。对一些负载格式还需要做特殊处理。

代码如下：

```
MultiFramedRTPSink::afterGettingFrame1() {
    doSpecialFrameHandling();
    sendPacketIfNecessary();
}
```

6) `sendPacketIfNecessary()`

```
MultiFramedRTPSink::sendPacketIfNecessary() {
    fRTPInterface.sendPacket(fOutBuf->packet()
        ,
        fOutBuf->curPacketSize()); //fRTPInterface 是创建的 TCP 或 UDP 套接字。
    nextTask() = envir().taskScheduler().scheduleDelayedTask(uSecondsToGo,
        (TaskFunc*) sendNext, this);
}
```

如果 `packet` 中有数据则调用 `sendPacket()` 将数据发出，其中 `sendPacket` 函数里面有两种方式来发送 RTP 包，一种是通过 UDP 的方式，另一种是通过 TCP 的方式，里面调用了 `SendRTPOverTCP()` 函数。接下来调用任务调度函数，`uSecondsToGo` 是下一次 RTP 包的发送延时时间，`sendNext` 指向了下一次任务的回调函数指针，而 `sendNext` 函数里面又调用了 `buildAndSendPacket()` 函数，这样 RTP 数据就可以不断地发出来，完成数据的传输。

RTP 打包与发送的流程图如图 4-2 下：

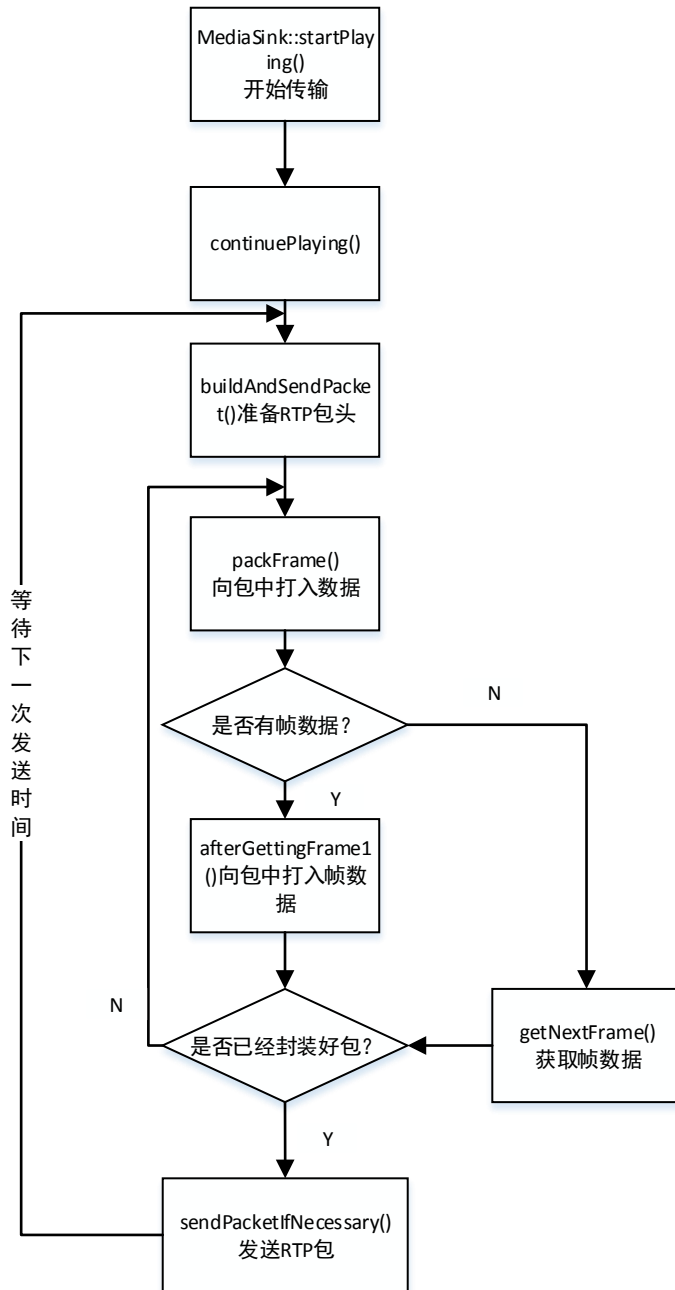


图 4-2 RTP 打包与发送流程

4.4 扩展 RTSPServer 支持处理客户端的上传

Live555 中的 RTSPServer 类，并没有完全实现 RTSP 协议，参考 playCommon 的实现方法，并在其基础上扩展了 Announce 方法、Setup 方法和 Record 方法。在 Announce 方法中根据客户端传过来的 SDP 信息创建相应的 MediaSession 对象。

```

char* sdpDescription = NULL;
parseSdpDescription(fullRequestStr, sdpStrLen, sdpDescription);
fMediaSession = MediaSession::createNew(envir(), sdpDescription);
    
```

在 Setup 方法中设置了两种模式，一种是 play 模式，另一种是 receive 模式。Play 模式是用于处理播放服务器中的媒体资源的情况，而 receive 模式则是用于处理客户端媒体资源的上传。

并在 Setup 方法的 receive 模式中对 MediaSession 中的每一个 MediaSubSession 创建相应的 RTP、RTCP 套接字和 RTPSource、ReadSource。RTPSource 指媒体资源来自于网络，需要经过网络传输给服务器。最后创建 RTCPInstance。此外如果是 RTSP over TCP 的话还需要为每一个会话的 RTPSource 和 RTPInstance 设置对应的套接字和通道号以及回调函数。如果是 RTSP over UDP 的话还需要为每一个会话的 RTPSource 和 RTPInstance 设置对应的套接字地址和端口号以及回调函数。

然后在 Record 方法中选择合适的宽度和高度以及视频并创建 FileSink 来封装 MP4 或 AVI 文件。交互如图 4-3 所示

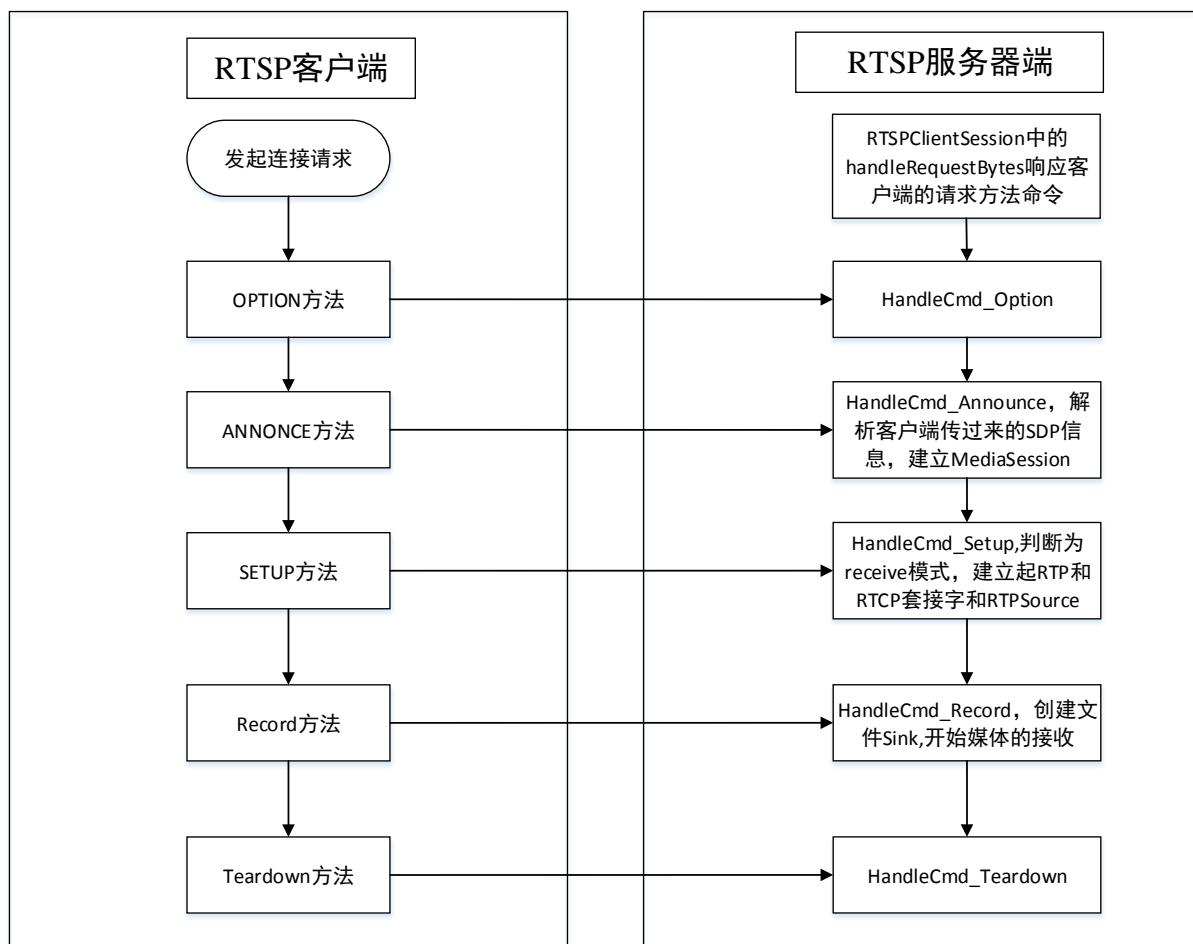


图 4-3 客户端上传服务器接收流程

4.5 扩展 RTSPServer 支持 MP4 视频格式

开源项目 Live555 的 Media 服务器端提供了一个可扩展的多媒体框架，把数据从源

(source) 发送到一个接受端 (Sink)。在点播模式中, 程序从本地磁盘读取 MP4 视频文件, 利用 ffmpeg 将 MP4 文件进行解析, 按照 MP4 格式提取其中一帧数据, 然后把数据发送 RTPSink, RTPSink 根据 RTP 传输协议将一帧数据通过网络发送到播放器^[18]。在 Media 服务器里, Medium 是所有类的基类, 根据不同功能而派生出不同的子类, 我们又根据需要的功能而继承相应的子类进行扩展。为了支持 MP4 格式, 所用到的类如图 3 所示, 添加了 7 个类, MyRTSPServer, H264VideoMP4FileServerMediaSubsession, MP4DemuxedElementaryStream, MP4Demux, MP4FileServerDemux 和 AACAudioMP4FileServerMediaSubsession。

首先明确, ServerMediaSession 是与资源对应的, 即是说每一个媒体资源对应只有一个 ServerMediaSession 类对象, 而不是每一个连接上的用户对应一个 ServerMediaSession 类对象。由于创建 ServerMediaSession 对象的过程在 DynamicRTSPServer 类当中, 因此要想扩展 live555 以支持 mp4 媒体格式, 那么必需要继承 DynamicRTSPServer 类, 在子类中去创建 mp4 的 ServerMediaSession。MyRTSPServer 的父类是 DynamicRTSPServer, 在 MyRTSPServer 类中重新实现了 lookupServerMediaSession 这个虚函数方法, 添加了 MP4 的支持代码, 创建并添加视频 H.264 和音频 AAC 的 ServerMediaSubSession。

创建完 ServerMediaSession 类对象后, 紧接着就要创建 ServerMediaSubsession, 并 add 进 ServerMediaSession 对象中。但并不知道要创建是的什么编码格式的 ServerMediaSubsession, 比如是创建 h264 的 ServerMediaSubsession, 还是 aac 的 ServerMediaSubsession? 为此, 需要增加 MP4FileServerDemux 类, 该类会对 mp4 文件进行分析, 然后根据分析结果再创建相应的 ServerMediaSubsession。

另外, 分析 MPEG1or2FileServerDemux 类可知, MPEG1or2DemuxedElementaryStream 也是在 MPEG1or2FileServerDemux 类中创建, 这个 MPEG1or2DemuxedElementaryStream 就是 FramedSource 的子类。因此类似地, 应该在 MP4FileServerDemux 类中实现 FramedSource 子类的创建, 这里创建出 MP4DemuxedElementaryStream 类。

MyDemux 继承 Medium, 它的主要功能是调用 ffmpeg 中的接口函数完成 MP4 文件的解复用, 将 H.264 视频流和 AAC 音频流分离出来以创建相应的 Source。每一个 MyDemux 实例对应每一个客户端。

MP4DemuxedElementaryStream 继承自 FramedSource, 类似于 MPEG1or2DemuxedElementaryStream 类, MP4DemuxedElementaryStream 类主要是作为一个中间类, 通过它向 MP4Demux 类获取具体的媒体数据。该类主要是由与 sink 直接

关联的 source 来调用。

Subsession, 代表每一个子会话, 我们需要将媒体文件中的各个流进行分离, 针对每一种流实现一个 subsession, 重新实现里面的 createNewStreamSource 虚函数方法, 以提供 source。对于 h264, 处理 h264 的 H264VideoMP4FileServerMediaSubsession 从 H264VideoFileServerMediaSubsession 继承, 从 MP4FileServerDemux 类中获取 es 流并传给 H264VideoStreamFramer 处理。在我们从 mp4 文件中提取 h264 数据时, 数据包中并不会包含 SPS 序列参数集和 PPS 图像参数集信息, 我们需要从 FFMPEG 中的 AVCodecContext 结构中的 extradata 中进行提取, 以保证 h264 能够正常地进行解码播放。对于 aac, 处理 aac 的 AACAudioMP4FileServerMediaSubsession 从 FileServerMediaSubsession 继承, 从 MP4FileServerDemux 类中调用 newElementaryStream 方法得到音频流的 Source。类的关系图如图 4-4 所示:

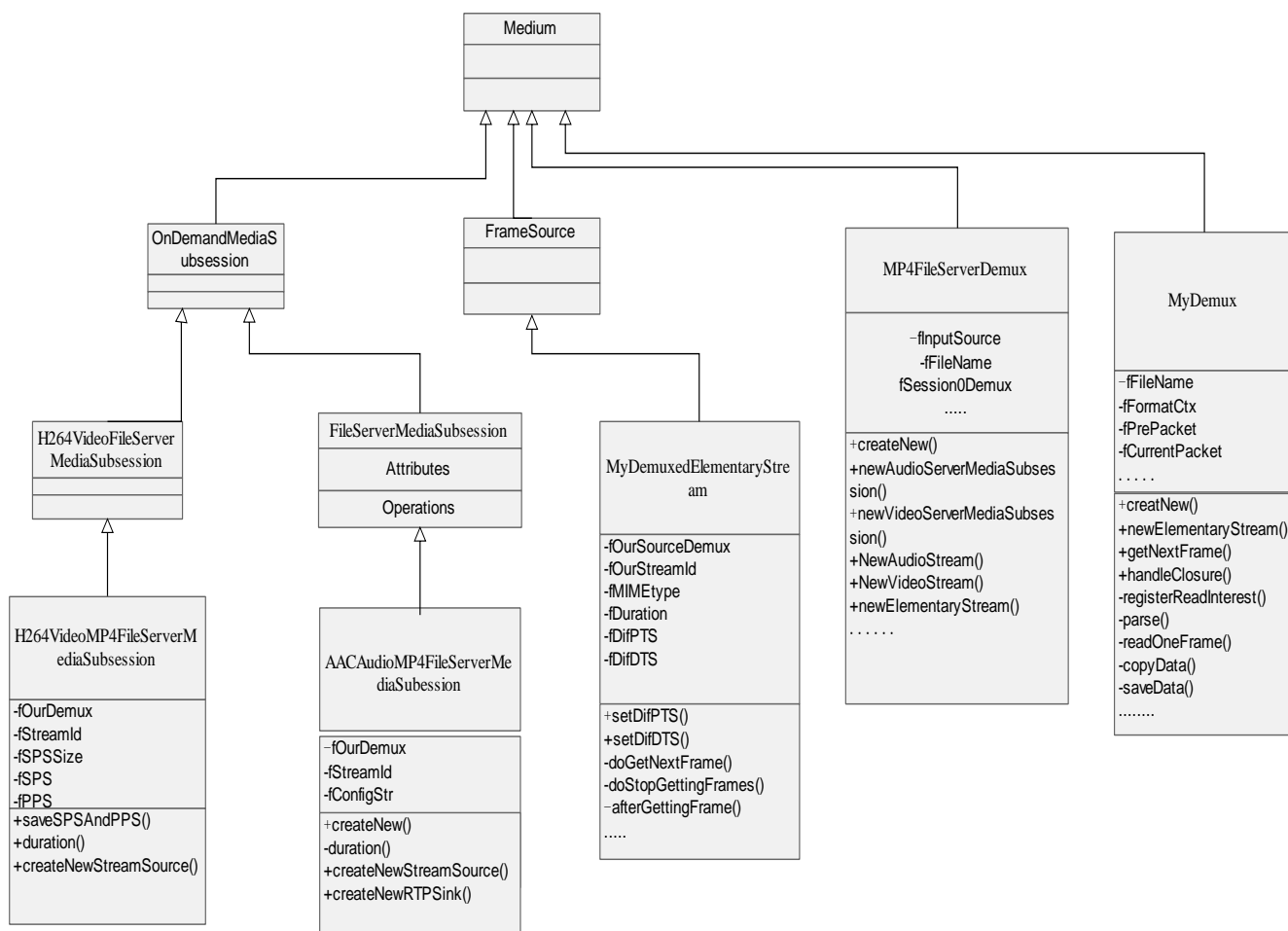


图 4-4 Media 服务器扩展 MP4 主要类图

4.6 扩展 RTSPServer 支持实时转发

此项功能可以实现当客户端上传视频资源给服务器的同时, 另外一个客户端可以实

时的播放，服务器实现实时转发的功能。

ANNOUNCE 方法处理：

首先，服务器根据客户端传过来的 SDP 会话描述信息创建 MediaSession，接着再通过 ProxyServerMediaSession 创建 ServerMediaSession 对象，之后依次迭代遍历 MediaSession 对象，并利用 ProxyServerMediaSubsession 来创建 ServerMediaSubsession 对象，同时将创建的 ServerMediaSubsession 加入到 ServerMediaSession 对象的成员中，ServerMediaSubsession 里面实现了 RTPSink 和 RTPSource 的创建，以实现将音视频 RTP 包传输到网络或者从网络中接收 RTP 流，最后将 ServerMediaSession 对象添加到 RTSPServer 类中。

SETUP 方法处理：

在 SETUP 方法当中实现了两种模式：一种是 play 模式，另一种是 receive 模式。在实时转发中这两种模式都会用到，receive 模式用于客户端的音视频上传，而 play 模式则用于另一个客户端的实时播放。在实时转发过程中，需要创建两套传输机制，一套用于接收客户端的音视频流，创建好接收流的套接字，并调用 ProxyServerMediaSubsession 中的 createNewStreamSource 函数来创建 RTPSource。另一套则用于服务器传输音视频流给播放客户端，并调用 ProxyServerMediaSubsession 中的 createNewRTPSink 来创建 RTPSink，将音视频流传给客户端。在扩展中需要注意 mediaSubSession 中的 id 为 Streamid=%d,而 ServerMediaSubSession 中的 id 为 track%d。因此需要将 serverMediaSubSession 中的 trackID() 改为虚函数，然后在 ProxyServerMediaSession 中重载 trackID 函数，否则会出现找不到视频资源具体如“RTSP/1.0 404 Stream Not Found”的错误提示。在 SETUP 方法中，为了实现正确的传输，还必须处理好 RTCP 中 SR 包和 RR 包的发送和接收。

DESCRIBE 方法处理：

在 DESCRIBE 方法中，会根据流的名字查找 ServerMediaSession，如果找不到的话，会创建一个。在实时转发的过程中，会发现在之前的 ANNOUNCE 方法中已经创建了相应的 ServerMediaSession，因此会直接利用进行后续处理，同时创建对应的 SDP 会话描述信息返回给播放客户端。

RTP 转发细节：

RTP 转发的重点在于建立从客户端传过来的流 RTPSource 到另一个客户端进行播放从服务器拉流 RTPSink 的循环，理想的转发在于流媒体服务器收到 RTP 包不进行任何

的重组，直接转发给其他客户。这里用到的是从 `MediaSubsession` 中的 `readsource` 中获取一帧数据，然后再传给 `RTPSink` 进行重新分片打包。本次转发中，用到的是 `H264VideoStreamDiscreteFramer`，它是一个继承自 `H264VideoStreamFramer` 的类，重写了 `doGetNextFrame`，从 `MediaSubsession` 中的 `source` 中获取数据，并复制到 `fTo` 中，并在 `afterGettingFrame1` 设置好 `SPS` 和 `PPS` 参数信息，然后再交由 `H264VideoRTPSink` 进行重新切片组装。

实时转发的时序图如图 4-5 所示：

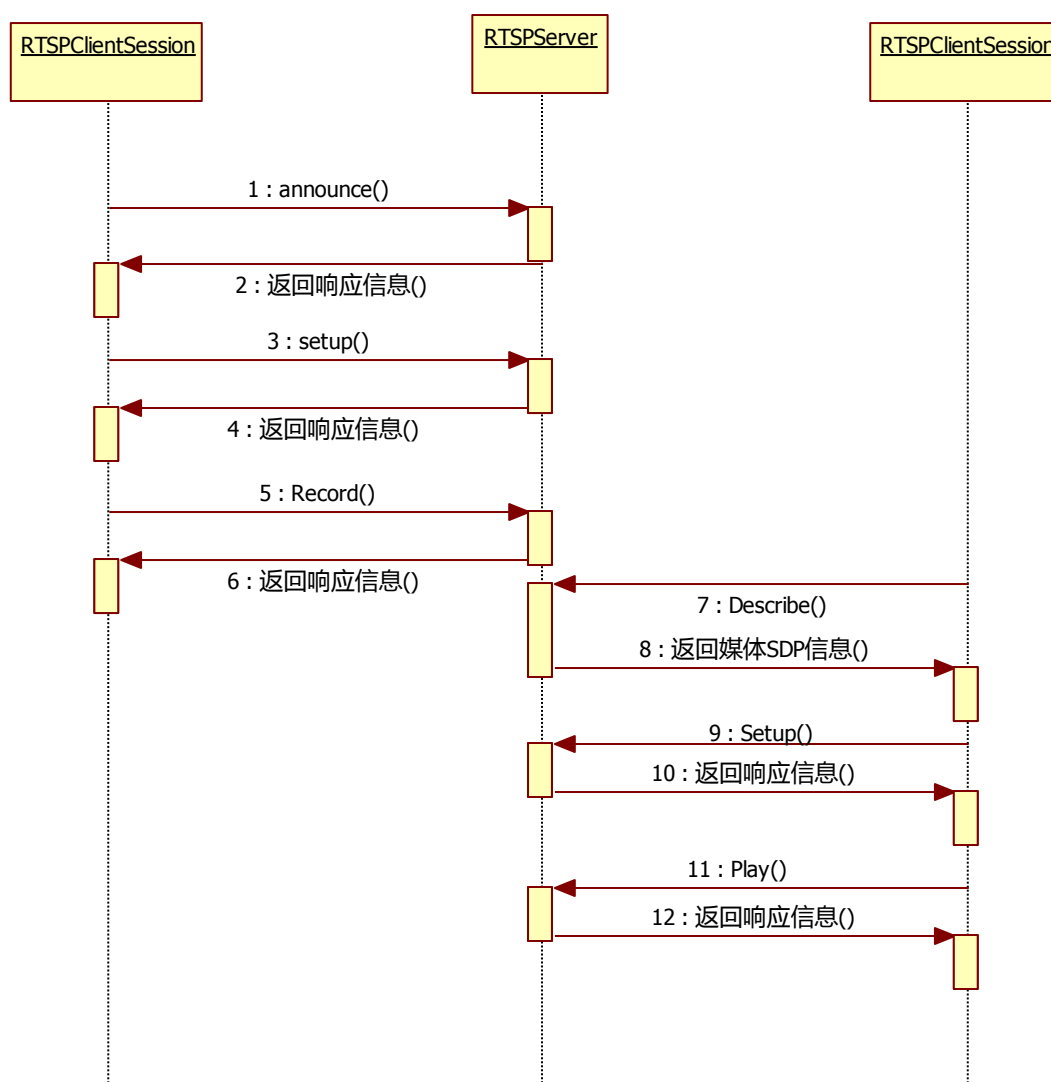


图 4-5 RTSP 实时转发时序图

4.7 使 RTSPServer 支持多进程

在主进程中先获取机器的 CPU 核数从而确定进程的个数，先创建一个套接字，设

置地址端口利用，绑定并监控套接字，设置发送缓存区大小，之后利用 fork 创建几个子进程，在子进程中先创建 TaskScheduler 和 UsageEnvironment

```
TaskScheduler* scheduler = BasicTaskScheduler::createNew();  
UsageEnvironment* env = BasicUsageEnvironment::createNew(*scheduler);
```

之后创建 RTSPServer

```
RTSPServer* rtspServer;  
rtspServer = MyRTSPServer::createNew(*env, rtspServerPortNum, socket, authDB);  
并开始大循环
```

```
env->taskScheduler().doEventLoop();
```

4.8 本章小结

本章首先介绍了流媒体服务器的整体方案，对 live555 中 RTSP 服务器的主要逻辑进行剖析，接着分析了 live555 中 RTP 的打包与发送，进而介绍了在 live555 的二次开发中实现的支持客户端的上传，MP4 格式的下发，实时流媒体的转发以及多进程的扩展。

第五章 实时流媒体客户端方案实现

流媒体客户端既要实现音视频采集编码，又要实现音视频解码播放，播放过程中还要确保播放质量和音视频同步，同时还要实现 RTSP 的上传和播放，所以对客户端功能的完整性要求比较高，本章将对这些方面进行简述：

5.1 流媒体客户端整体方案

通过在 Android 系统上移植 FFMPEG，调用 Android 的 API 开发库，客户端既实现了音视频采集编码上传的功能，又实现了解码播放的功能，同时还实现了流媒体的 RTP 传输。流媒体客户端为了支持边拍边传功能采用的是 FFMPEG 的软编码技术，通过调用 Android 的 API 实现音视频的采集，然后通过 MP4 文件预解析模块得到 H264 的 PPS 和 SPS 参数信息，再进行音视频编码封装成本地 MP4 文件或者通过 RTP 传输到服务器进行接收。在播放这一端，客户端既可以播放本地的 MP4 文件也可以播放远程的 RTSP 流，通过 FFMPEG 的 API 进行音视频解码，进而应用 Android 的 API，结合音视频同步进行 MP4 文件的播放。FFMPEG 里面实现的 RTCP 协议可以动态计算数据接收和发送的速率，从而比较好地解决了拥塞控制问题。

整体方案结构如下图 5-1 所示：

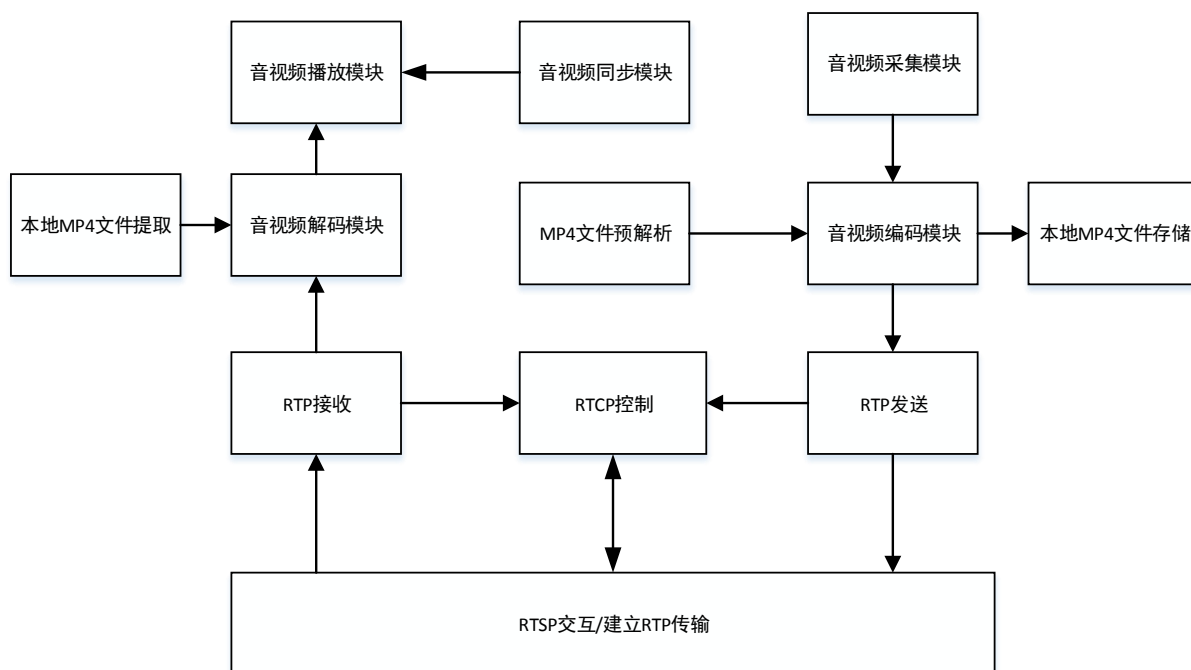


图 5-1 流媒体客户端整体方案

5.2 FFMPEG 在 Android 上的移植

开发环境如下：

- 交叉编译环境 Cygwin
- Android NDK: android-ndk-r4b-windows 注意应为 windows 下的版本。
- FFmpeg: 0.10.3

首先在 Android NDK 上建立一个 project。将 FFmpeg 的 source code 整包 copy 到 \$(PROJECT)/jni 底下。

然后在 \$(PROJECT)/jni/ffmpeg 底下建立一个 config.sh:
#!/bin/bash

```
PREBUILT=/home/lltlfw/android-ndk-r4b-windows/build/prebuilt/arm-eabi-4.4.0
PLATFORM=/home/lltlfw/projects/android-ndk-r4b-windows/build/platforms/arch-arm
```

```
./configure --prefix=/opt/toolchains --disable-doc --enable-ffmpeg --disable-ffplay --enable-ffserver --disable-avfilter --disable-devices --disable-debug --enable-encoders --enable-cross-compile --enable-decoders --enable-demuxer=mov --enable-demuxer=h264 --enable-protocol=file --enable-avformat --enable-avcodec --enable-decoder=rawvideo --enable-decoder=h264 --enable-decoder=mjpeg --enable-decoder=h263 --enable-decoder=mpeg4 --enable-parser=h264 --enable-zlib --enable-shared --enable-static --enable-pic --nm=arm-linux-androideabi-nm --sysroot=/opt/toolchains/sysroot --cc=arm-linux-androideabi-gcc --arch=arm --target-os=linux --cross-prefix=arm-linux-androideabi- --disable-asm
```

其中 PREBUILT 和 PLATFORM 需要符合自己机器的配置。

然后在 Cygwin 中，\$(PROJECT)/jni/ffmpeg 底下执行 config.sh:

```
$ ./config.sh
```

Configure 会依据 config.sh 内的设定将 config.mak 及 config.h 建立出来。

在根据上面步骤生成的 config.h 中找到: #define restrict restrict, 在 Android 中, gcc 并不认识 restrict 这个关键字, 所以我们要把它改写成: #define restrict

将/libavutil/libm.h 中所有的 static 函式, 全都注释掉, 依次修改 libavcodec、libavformat、libpostproc、libavfilter、libavutil 以及 libswscale 下的 makefile, 将下面两条语句注释起来:

```
include $(SUBDIR)../config.mak
include $(SUBDIR)../subdir.mak
```

接着需要创建一系列的 mk 文件，首先在\$(PROJECT)/jni/ffmpeg 下建立一个 av.mk 文件，内容如下图 5-2:

```
# LOCAL_PATH is one of libavutil, libavcodec, libavformat, or libswscale
include $(LOCAL_PATH)/../config.mk
OBJS :=
OBJS-yes :=
MMX-OBJS-yes :=
include $(LOCAL_PATH)/Makefile
# collect objects
OBJS-$(HAVE_MMX) += $(MMX-OBJS-yes)
OBJS += $(OBJS-yes)
FFNAME := lib$(NAME)
FFLIBS := $(foreach,NAME,$(FFLIBS),lib$(NAME))
FFCFLAGS = -DHAVE_AV_CONFIG_H -Wno-sign-compare -Wno-switch
-Wno-pointer-sign
FFCFLAGS += -DTARGET_CONFIG=\"config-$(TARGET_ARCH).h\"
ALL_S_FILES := $(wildcard $(LOCAL_PATH)/$(TARGET_ARCH)/*.S)
ALL_S_FILES := $(addprefix $(TARGET_ARCH)/, $(notdir $(ALL_S_FILES)))
ifndef $(ALL_S_FILES)
ALL_S_OBJS := $(patsubst %.S,%o,$(ALL_S_FILES))
C_OBJS := $(filter-out $(ALL_S_OBJS),$(OBJS))
S_OBJS := $(filter $(ALL_S_OBJS),$(OBJS))
else
C_OBJS := $(OBJS)
S_OBJS :=
endif
C_FILES := $(patsubst %o,%c,$(C_OBJS))
S_FILES := $(patsubst %o,%S,$(S_OBJS))
FFFILES := $(sort $(S_FILES)) $(sort $(C_FILES))
```

图 5-2 av.mk

在\$(PROJECT)/jni 底下修改 Android.mk，内容如下:

```
include $(all-subdir-makefiles)
```

在 /ffmpeg 底下新增一个 Android.mk 如下图 5-3 所示:

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_STATIC_LIBRARIES := libavformat libavcodec libavutil libpostproc libswscale
LOCAL_MODULE := ffmpeg
include $(BUILD_SHARED_LIBRARY)
include $(call all-makefiles-under,$(LOCAL_PATH))
```

图 5-3 Android.mk

在/ffmpeg/libavformat 底下新增一个 Android.mk，如下图 5-4 所示:

```

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
include $(LOCAL_PATH)/../av.mk
LOCAL_SRC_FILES := $(FFFILES)
LOCAL_C_INCLUDES :=      \
    $(LOCAL_PATH)      \
    $(LOCAL_PATH)/..
LOCAL_CFLAGS += $(FFCFLAGS)
LOCAL_CFLAGS += -include "string.h" -Dipv6mr_interface=ipv6mr_ifindex
LOCAL_LDLIBS := -lz
LOCAL_STATIC_LIBRARIES := $(FFLIBS)
LOCAL_MODULE := $(FFNAME)
include $(BUILD_STATIC_LIBRARY)

```

图 5-4 Android.mk

在ffmpeg/libavcodec 底下新增一个 Android.mk，如下图 5-5 所示：

```

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
include $(LOCAL_PATH)/../av.mk
LOCAL_SRC_FILES := $(FFFILES)
LOCAL_C_INCLUDES :=      \
    $(LOCAL_PATH)      \
    $(LOCAL_PATH)/..
LOCAL_CFLAGS += $(FFCFLAGS)
LOCAL_LDLIBS := -lz
LOCAL_STATIC_LIBRARIES := $(FFLIBS)
LOCAL_MODULE := $(FFNAME)
include $(BUILD_STATIC_LIBRARY)

```

图 5-5 Android.mk

在 libavfilter、libavutil、libpostproc 和 libswscale 底下都分别新增一个 Android.mk，如下图 5-6 所示：

```

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
include $(LOCAL_PATH)/../av.mk
LOCAL_SRC_FILES := $(FFFILES)
LOCAL_C_INCLUDES :=      \
    $(LOCAL_PATH)      \
    $(LOCAL_PATH)/..
LOCAL_CFLAGS += $(FFCFLAGS)
LOCAL_STATIC_LIBRARIES := $(FFLIBS)
LOCAL_MODULE := $(FFNAME)
include $(BUILD_STATIC_LIBRARY)

```

图 5-6 Android.mk

最后在\$(PROJECT)底下执行 ndk-build:

\$ /android-ndk-r4b/ndk-build 完成后, 会产生出 libavcodec.a、libavformat.a、libavutil.a、libpostproc.a、 libswscale.a 和 libffmpeg.so。

5.3 客户端音视频采集

5.3.1 音频采集实现

手机客户端的音视频采集采用的是继承 android 自带的 MediaRecord 类来实现。

其中音频采集使用到了 AACMP4Record 类。该类是继承自 MediaRecord, 实现的音频的采集, 封装了设置音频源, 编码格式, 设置输出格式, 音频通道数, 采样率, 最长录制时间等功能。

具体类模型如图5-7所示:

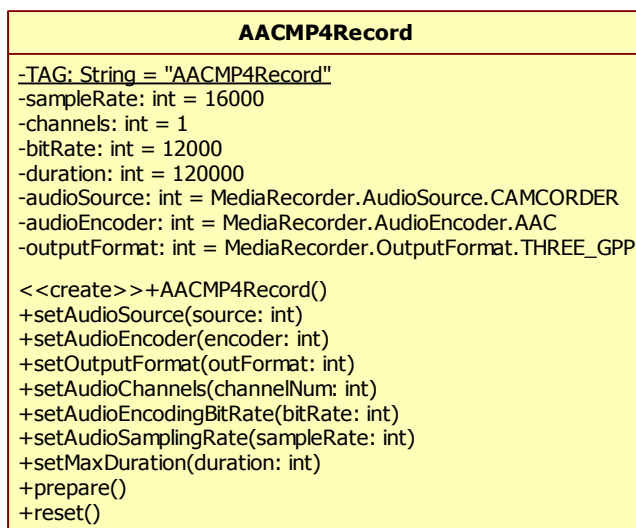


图 5-7 AACMP4Record 类图

i. 成员变量简介:

TAG 为日志标签, sampleRate、channels、bitRate、duration 为音频的采样率, 音频录制的声道, 编码比特率, 录制时间, audioSource 为音频获取源, audioEncoder 为音频编码方式, outputFormat 为输出文件的格式。

ii. 成员方法简介:

AACMP4Record 为构造类, setAudioSource(source:int)用于设置音频获取源, setAudioEncoder(encoder:int)用于设置音频编码方式, setOutputFormat(outFormat:int)用于设置文件输出格式,

setAudioChannels(chaneINum:int)用于设置音频声道类型，
 setAudioEncodingBitRate(bitRate:int)用于设置编码比特率，
 setMaxDuration(duration:int)用于设置录制时间最大限制。

5.3.2 视频的采集实现

视频的采集也是继承了 MediaRecord 方法。下图 5-8 是类的关系图：

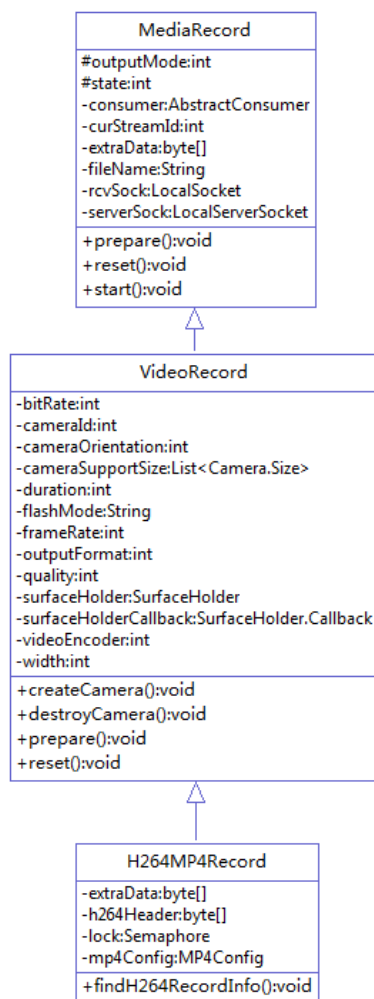


图 5-8 视频编码类关系图

其中 videoRecord 继承 MediaRecord，而 H264MP4Record 继承 VideoRecord。

i. 成员变量简介：

bitRate 为视频的比特率，cameraId 为使用的摄像头 ID，cameraOrientatio 为当前摄像头的旋转角度，flashMode 为闪光灯模式，camerasTotalNum 为手机的摄像头总数，cameraSupportSize 为手机当前摄像头支持的分辨率，frameRate 为编码的帧率，outputFormat 为输出的文件格式，quality 为录制质量，

surfaceHolder 为显示一个 surface 的抽象接口, surfaceHolderCallback 用户可以实
现此接口接收 surface 变化的消息, videoEncoder 为视频编码方式, duration 为录
制最大时长, width 和 height 为视频的尺寸。

ii. 成员方法简介:

setVideoQuality(int quality)设置录制质量。默认为

CamcorderProfile.QUALITY_LOW (no use)

setFlashMode(String mode)设置闪光灯模式, 包括

Camera.Parameters.FLASH_MODE_OFF,

Camera.Parameters.FLASH_MODE_TORCH 等

setVideoSize(int width, int height) 设置视频尺寸。默认为 352*288

setVideoFrameRate(int frameRate) 设置视频帧率。默认为 15

setVideoBitRate(int bitRate) 设置视频比特率。默认为 200kb/s

setVideoEncoder(int encoder) 设置视频编码。默认为

MediaRecorder.VideoEncoder.H264。

setOutputFormat(int outputFormat) 设置视频输出格式。默认为

MediaRecorder.OutputFormat.THREE_GPP。

setMaxDuration(int duration) 设置视频录制时间。默认为 120s。

setPreviewDisplay(SurfaceHolder sh) 设置预览的 SurfaceHolder。createCamera 和
prepare 前必须先设置。

getBestVideoSize(Camera cam, int width, int height) 获取与要求最适合的分辨率。

createCamera()打开摄像头并预览。调用前先设置参数。

prepare()设置 camera 和录制参数, 准备录制。所有的参数必须在调用该函数前
设置, 设置顺序可随意。

reset()重置该类, 释放 camera, 所有参数必须重新设置。

而 H264MP4Record 继承自 videoRecord, 为了实现 H264 的上传, 我们首先需要得
到 H264 的编码信息 SPS (序列参数集) 和 PPS (图像参数集)。为了获取它, 首先需要
录制一小段如 1S 的视频数据, 然后对 MP4 的源文件进行解析, 获取 SPS 和 PPS 参数,
并填充 extradata。

i. 成员变量简介:

mp4Config 用于存储录制 1S 视频数据后的 mp4 信息, extraData 是用于存储 h264

的各种参数包括 h264 头, SPS 和 PPS 信息, h264Header 为 h264 头编码。

ii. 成员方法简介:

findH264RecordInfo()用于录制 1000 毫秒数据, 并分析获取编码信息(pps, sps)。

5.4 客户端音视频编码

编码的实现如下图 5-9 所示:

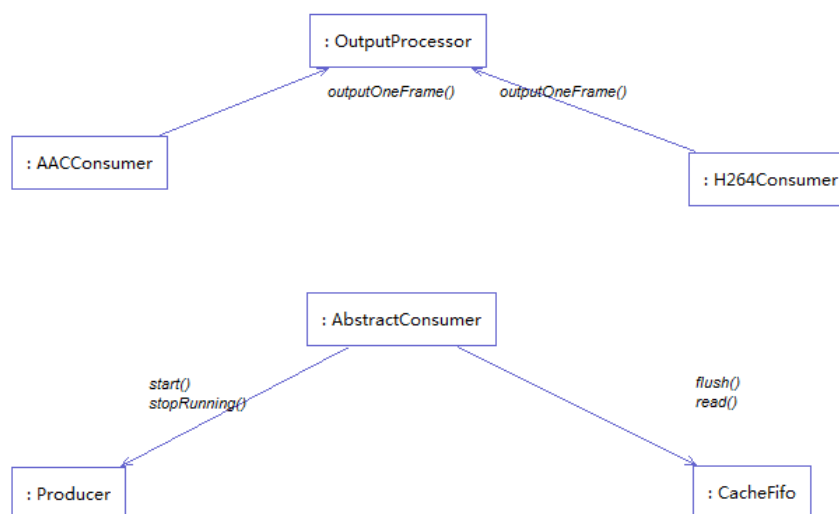


图 5-9 视频编码各类关系图

系统采用了多线程的方式, 分别对音视频和视频进行编码, 底层通过 JNI 的方式调用 ffmpeg 的 C++库来进行软编码。其中 AACConsumer 用于从 CacheFifo 提取音频的编码数据, 而 H264Consumer 用于从 CacheFifo 中提取视频的编码数据, AbstractConsumer 是这个类的基类, 里面包含了使用了一个缓冲类来保留编码的数据流和一个 Producer 类来对缓冲区进行填充。对编码后的音视频帧, 再调用 ffmpeg 的 rtsp 上传接口来实现 rtsp 的传输。

5.5 客户端 MP4 文件的解析

MP4 文件解析类的关系图如图 5-10 所示

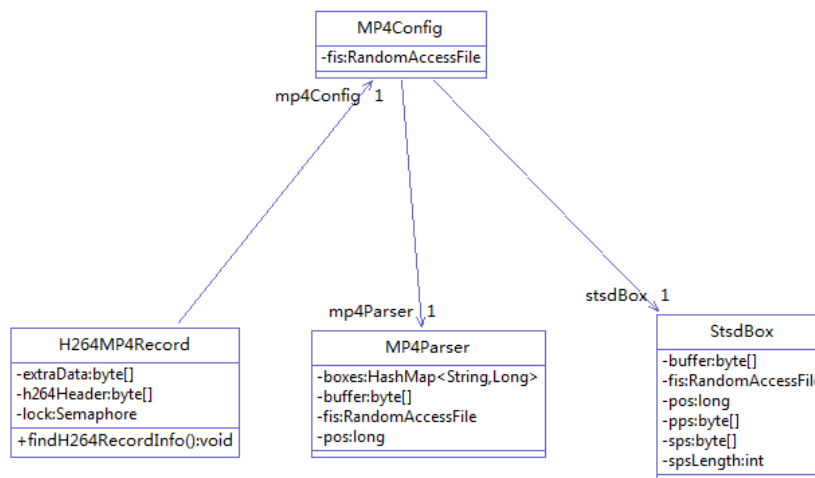


图 5-10 MP4 文件解析类关系图

MP4 文件有着巧妙的封装格式，它所有的数据都封装到 box 中，box 有很多种类型，包括 flyp、moov、mdat 等。Box 又可以包含另外的 box，这种 box 叫做 container box，每个 box 都包含头部 header 和数据 data。在头部中包含长度和类型等信息。

首先每一个 MP4 文件有且仅有一个“ftyp”类型的 box，这个 box 表明此文件是 mp4 文件，并且包含有文件的长度、帧率、版本以及兼容协议等信息。

其次，像“ftyp”一样，MP4 文件也只能含有一个“moov”类型的 box，它是一样 container box，它不包含具体的音视频等多媒体数据，而只包括了媒体的 metadata 信息。在 moov box 中包含了 mvhd 和 trak box，mvhd 中记录了 mp4 文件的记录时间、修改时间以及播放时长等信息，而 trak 的一系列的子 box 中包含了每个媒体轨道的媒体信息。

而“mdat”类型的 box 中则包含了所有需要解码播放的媒体数据信息，此类型的 box 也是一个内含 box，可以有多个或者没有，媒体信息的数据结构在 moov 中的 metadata 有详细描述。

H264MP4Record 类使用到了 MP4Config 类，里面录制了一段 MP4 视频并用 MP4Parser 对其进行了解析，其中 StsdBox 位于/moov/trak/mdia/minf/stbl/stsd 之下，然后通过获取 StsdBox 中的数据提取出了 SPS 和 PPS 信息。SPS 即序列参数集信息，PPS 即图像参数集信息，在视频进行解码播放的时候非常重要，所以我们在上传音视频数据前需要获取这两个重要的参数信息。

5.6 客户端音视频解码

音视频的解码模块的类之间的关系图如下图 5-11 所示：

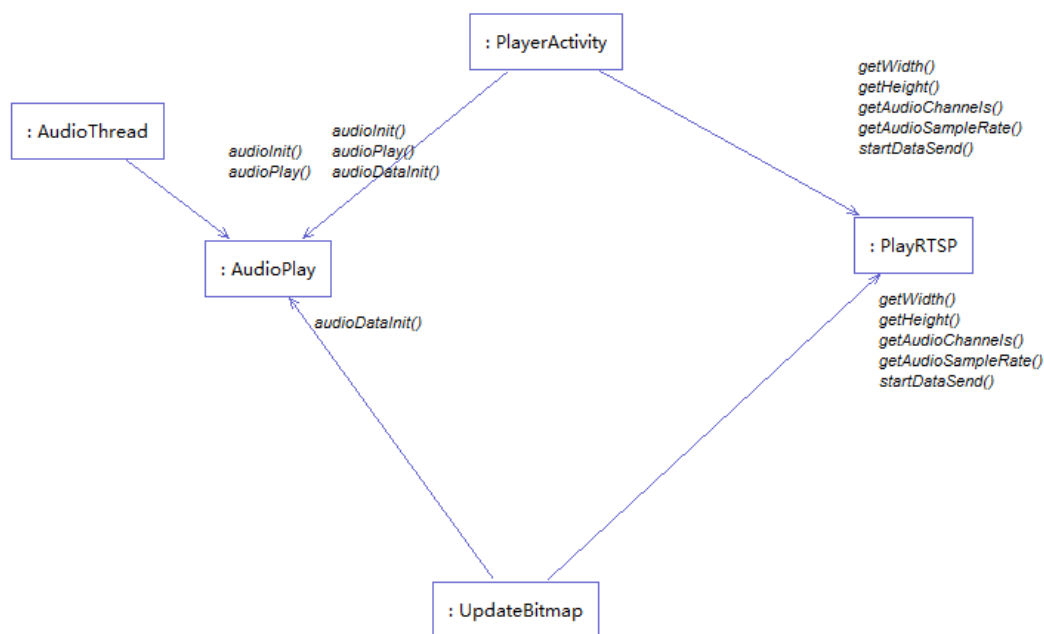


图 5-11 音视频解码类关系图

其中 AudioPlay 类为 android 端播放音频的类，而 UpdateBitmap 用于渲染视频帧，playRTSP 为 java 的本地方法，其中通过 JNI 调用 C++的方法，底层调用 FFMPEG 的方法来进行视频文件的解码。

其中 RTSPDecoder 的类结构如下图 5-12 所示：

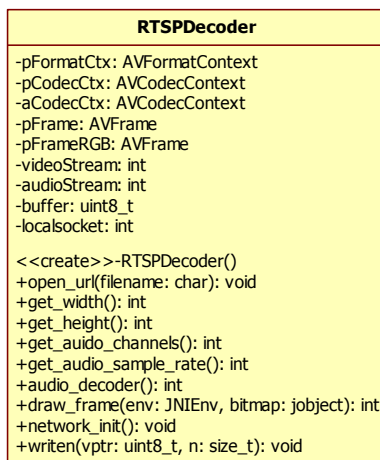


图 5-12 RTSP 解码类结构

pFormatCtx 主要存储视音频封装格式中包含的信息、pCodecCtx 用于描述视频编解码器上下文、aCodecCtx 描述音频编解码器上下文、pFrame 描述一个媒体流、pFrameRGB 描述 RGB 媒体流。

5.7 客户端音视频播放

客户端的解码播放^[19]流程如图 5-13 所示。客户端的解码也是用到 FFmpeg 库中的解码器，先将视频文件进行解析，包含音频流时就实例化 AudioPlay 类，包含视频流时就实例化 BitmapUpdate 类。然后进行解码，当为音频流时就从 C 层用 UNIX LocalSocket 传至 JAVA 层的 AudioPlay 播放，当为视频流时就调用 JNI 的 Bitmap 接口进行 ImageView 来显示图片。其中重点在于音视频的缓冲和同步处理，引入了队列和时间戳来解决。

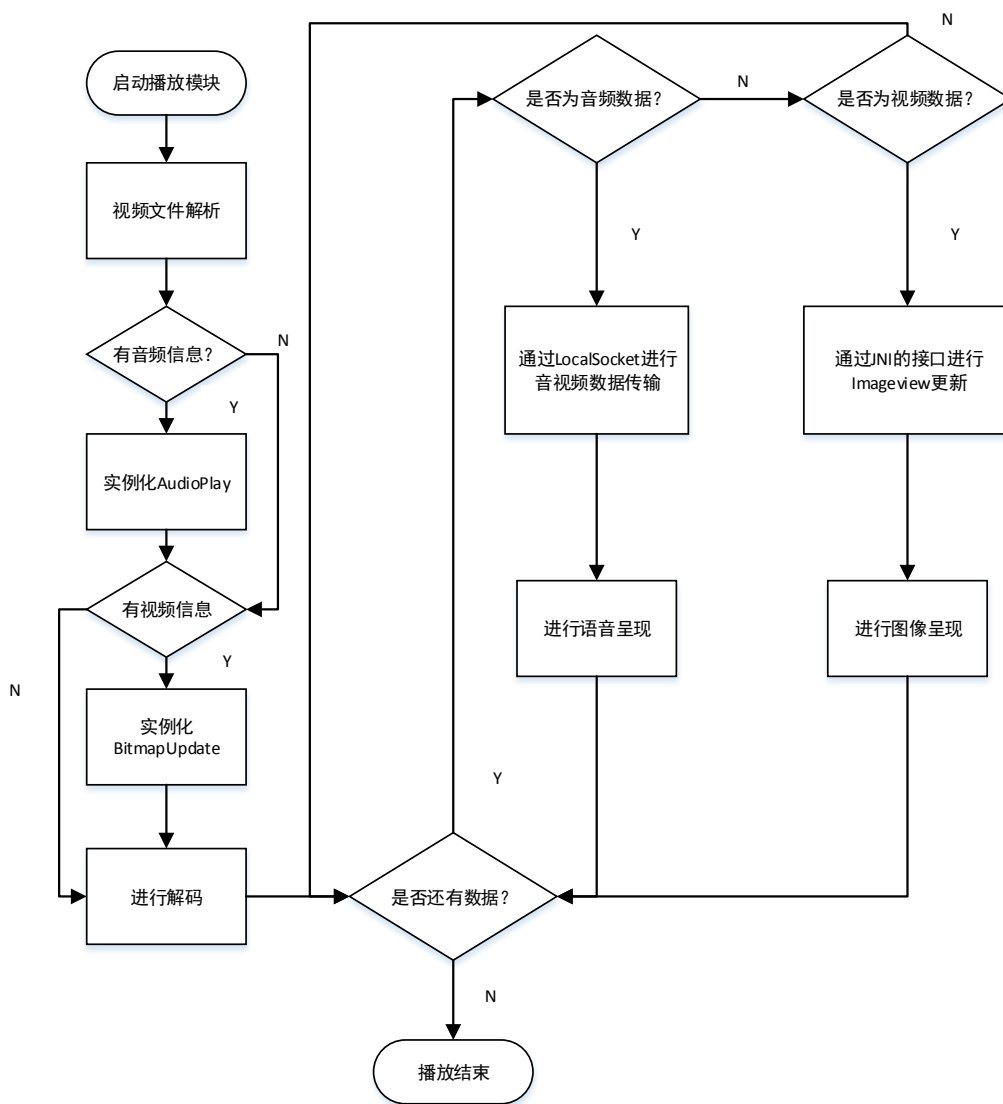


图 5-13 客户端解码播放主要流程图

5.8 客户端音视频同步

本系统是在 Android 终端^[20]上实现的流媒体传输方案，实时性是系统考虑的最重要的指标之一，也是系统的一大亮点。在客户端播放模块，采用流媒体传输，可以使音视频数据边下载边播放，而不用等全部下载完或者下载切割好的视频块再播放，减少了时延，提高了实时性。而在客户端上传时，在采用了两种技术，以应对 Android 终端在不同网络情况下的上传。

以往许多视频传输解决方案，大多都是拍摄一段视频，然后对所得到的视频数据进行传输。而本系统，则分别对终端所处网络环境不同，采取了更加细腻的传输控制，从而可以得到更好的传输性能和更少的时延，进一步提高系统实时性。当终端处于网络环境良好，信号较强、带宽足够的情况下时，系统采用边拍摄边上传的策略。系统选用 MP4 作为音视频数据的编码容器。用户在拍摄视频时，将拍摄得到的音视频数据写入缓冲区，然后对音视频数据进行分析定位，找到一系列的文件描述信息，如名为 `ftyp` 的 `box`，即找到 MP4 文件的文件类型描述以及完整性信息；名为 `mdat` 的 `box`，即找到文件的音视频数据起始位置；找到 `stco` 的 `box`，即文件数据的偏移量信息等。然后连接服务器，将所定位找到的信息分块发送给流媒体服务器以完成数据上传。当终端所处网络环境较差，信号较弱、带宽不足的情况下时，则采用传统数据传送策略，先对所拍摄的音视频数据进行存储，然后等回到网络环境较好的情况下，采用多线程并发对音视频数据进行上传。

音频与视频的同步问题，是许多与视频相关的系统都会碰到的问题，也是处理音视频的难点之一。音频与视频同步，一般来说有三种方法：同步视频到音频，同步音频到视频，同步音频和视频到同一个外部时钟。

本系统采用同步视频到音频，即音频解码之后则自顾自的播放，而视频则需要随着音频播放的时间而不断调整其播放的时间。客户端接收 Live555 发过来的音频流与视频流，用 FFmpeg 对音频流和视频流进行解析处理，读取音频流和视频流中的数据包，分别存入音频队列中与视频队列中以进行下一步处理。每个数据包都包含有 PTS（显示时间戳）和 DTS（解码时间戳），分别表示数据的显示时间和解码时间。通过一些方法，我们可以得到每帧图像的 PTS。我们定义视频连续两帧之间的时间差 `delay`，音频播放时间 `audio_time`。在某个时间点，我们取出此时视频帧的 PTS，与此时音频的播放时间做差，得到的差值 `diff`，然后将 `diff` 与设置好的阈值 `THRESHOLD` 比较。若 `diff` 小于阈

值 THRESHOLD, 则认为此时音频与视频是同步的, delay 则不便, 保持上两帧的时间差; 若 diff 大于阈值 THRESHOLD, 则认为此时音频与视频失去同步, delay 需要进行调整, 从而调整下一帧视频的显示时间, 以使音频与视频恢复同步。

5.9 本章小结

本章介绍了流媒体客户端的整体方案, FFMPEG 在 android 上的移植过程, 接着介绍了音视频的上传包括采集、编码、以及 MP4 文件的解析等模块, 然后以及对客户端媒体文件的播放包括音视频的解码、播放、同步模块的实现进行详细分析。

第六章 系统测试

6.1 硬件环境

本文实现了一套基于 RTSP 协议的 H.264 的实时流媒体视频传输方案，在实际测试中，客户端采用的是 HTC G11 手机作为通信终端，该手机配置如表 6-1 所示：

表 6-1 手机客户端的硬件配置

硬件	参数
CPU/主频	ARM Cortex-A8 1GHZ
内存	1024M RAM + 16GB ROM
摄像头	800 万像素

服务器端采用四台 PC 组成 LVS 集群，每一台的硬件配置都一样，一台 PC 作为 Master，另外三台是 RealServer，部署在带宽为 1000Mbps 的局域网中，通过以太网交换机进行相连。配置如下表 6-2 所示：

表 6-2 LVS 集群结点的硬件配置

硬件	参数
CPU/主频	Intel(R)Core(TM) i3 CPU 2.66GHz
内存	4G
网卡	1000Mbps
硬盘	500GB

6.2 软件环境

手机端软件环境:手机端自带 Android OS v2.3 操作系统，安装有视信客户端 APP，在 APP 中移植了 FFMPEG 编解码库。

服务器端软件环境：服务器端安装了 Ubuntu11.04 操作系统，并在 RealServer1、RealServer2、RealServer3 中搭建了本系统的流媒体服务器。

6.2.1 LVS 环境搭建

表 6-3 为 LVS 集群^[21 22]系统中调度服务器 master 和三个 RealServer 的 IP 地址。

表 6-3 LVS 集群结点的 IP 地址

主机	IP 地址
Master	218.192.170.144
RealServer1	218.192.170.158
RealServer2	218.192.170.200
RealServer3	218.192.170.19

接下来介绍一个如何搭建一个 LVS 集群：

1、maser 调试服务器上环境准备

先安装 ipvsadm:

```
http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.24.tar.gz
```

```
#tar -zxvf ipvsadm-1.24.tar.gz
```

```
#cd ipvsadm-1.24
```

```
#make && make install
```

再进行如下设置：

```
#!/bin/bash
```

```
echo "1" >/proc/sys/net/ipv4/ip_forward
```

```
ifconfig eth5:100 218.192.170.144 broadcast 218.192.170.144 netmask 255.255.255.255
```

```
route add -host 218.192.170.144 dev eth5:100
```

```
ipvsadm -C
```

```
ipvsadm -A -t 218.192.170.144:554 -s rr -p 600
```

```
ipvsadm -a -t 218.192.170.144:554 -r 218.192.170.158 -g
```

```
ipvsadm -a -t 218.192.170.144:554 -r 218.192.170.200 -g
```

```
ipvsadm -a -t 218.192.170.144:554 -r 218.192.170.19 -g
```

```
ipvsadm
```

2、在 3 台 rtsp 服务器上添加如下脚本：

```
ifconfig lo:0 218.192.170.144 netmask 255.255.255.255 broadcast 218.192.170.144
```

```
/sbin/route add -host $SNS_VIP dev lo:0
```

```
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
```

```
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
```

```
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
```

```
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
```

```
sysctl -p >/dev/null 2>&1
```

```
echo "RTSP RealServer Start OK"
```

这样由一台调试服务器，三台 RTSP 服务器的 LVS 集群系统就运行起来了。

6.3 系统测试

边拍边传效果如图 6.1 所示，播放效果如图 6.2 所示，本实验结果跟以往的方案相比在于不是上传完再播放，或者边下载边播放，而是实现了实时播放，播放的延时在 1S 之内，较好地实现了实时转发即视频通话的效果。

6.3.1 边拍边传功能测试

测试效果如下图 6-1 所示，客户端进行边拍边传，服务器接收视频数据并将视频数据以 mp4 格式保存在服务器中。



图 6-1 客户端边拍边传

6.3.2 播放 rtsp 视频功能测试

客户端可以点播服务器上的 mp4 资源，播放延时大概在 1S 左右，实现了实时观看视频资源。在客户端上输入 rtsp 地址：`rtsp://218.192.170.144/userId/1.mp4`，测试效果如图 6-2 所示：



图 6-2 客户端播放 rtsp 视频

6.3.3 播放本机视频功能测试

客户端还可以通过读取本地的视频文件进行播放。测试结果如图 6-3 所示：

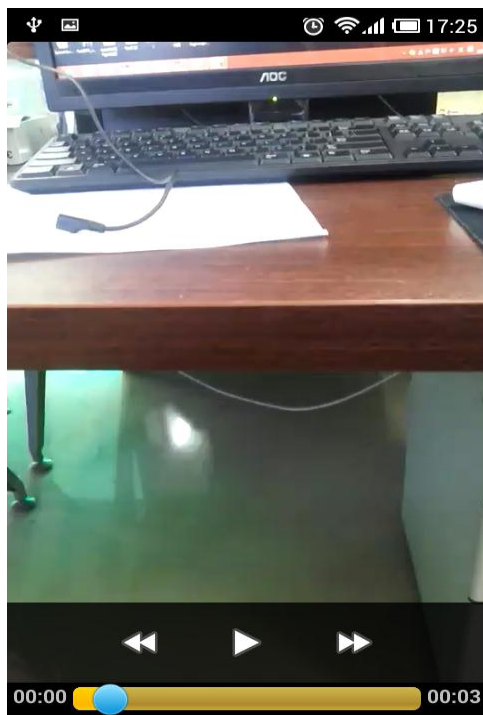


图 6-3 客户端播放本机视频

6.3.4 实时转发功能测试

手机客户端边播边上传视频，另外一个 VLC 客户端通过输入实时 rtsp 地址：
rtsp://218.192.170.144/12345/live.sdp 来进行实时播放。效果如图 6-4 与图 6-5 所示：



图 6-4 客户端边播边上传视频



图 6-5 PC 端通过 VLC 实时播放客户端上传至服务器的视频

为了对实时转发的延时进行估算，通过进行 10 次测试得出实时转发的延时如下表 6-4 所示，对一个视频边上传边播放 10 次，依次间隔 10 秒进行一次拍照，记录当前时刻上传客户端和播放客户端的视频播放时间，从测试结果可以得出时延在 500 毫秒左右，较好地满足了实时性的要求。

表 6-4 实时转发时延测试

测试次数	播放客户端时间(ms:ms)	上传客户端时间(ms:ms)	时延(S)
1	0:10:312	0:10:722	0.410
2	0:20:343	0:20:820	0.477
3	0:30:454	0:31:012	0.558
4	0:40:262	0:40:802	0.540
5	0:50:286	0:50:709	0.423
6	1:00:163	1:00:680	0.446
7	1:10:187	1:10:630	0.517
8	1:20:472	1:21:030	0.558
9	1:30:287	1:30:832	0.545
10	1:40:582	1:41:012	0.430

6.3.5 服务器性能测试

目前国际上对流媒体系统的测试工具相分有限，相应的测试工具也很少，负载测试工具 HP LoadRunner 也只是支持 Real 协议和 MMS 协议，而对支持应用非常广泛的 RTSP 协议，我们国内市面上也没有类似开发工具^[23]。所以为了测试服务器的性能，又因为实验室资源的有限，不能购买大量手机，只能采用在 PC 上开启多个客户端来代替手机客户端，下图 6-6 是用 QT 写的一个测试程序，里面采用多进程的方式开启了 ClientNum 个客户端，ClientPath 为 VLC 的安装地址，rtspAddr 为 RTSP 资源地址。由于客户端需要进行解码所以单台 PC 的客户端不宜开太多。这里选择 10 台 PC 机来进行测试。测试在 50 路，75 路，100 路，150 路视频服务器的各种参数性能。用于测试的 mp4 文件大小为 8MB，时间为 127 秒，码率约为 516kbps，播放界面如图 6-7 所示。

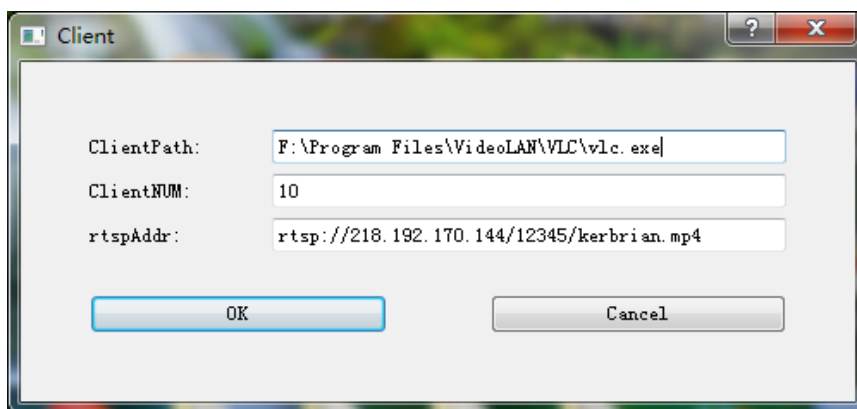


图 6-6 开启 N 个 VLC 客户端的测试程序界面



图 6-7 VLC 客户端播放界面表

服务器的性能测试结果如表 6-5 所示：服务器的并发路数为 50、70、100 时，CPU、内存、带宽都呈线性增长，视频播放也比较流畅，基本没有什么丢包现象。当视频达到 150 路时，CPU 增长缓慢，播放也比较流畅，当视频达到在 200 路以上时，CPU 几乎没有变化，视频也有了轻微的卡片现象，所以视频服务器的峰值在 200 路左右。

表 6-5 服务器性能测试结果

视频路数	CPU 占用率	内存占用率	视频是否流畅	输出带宽 (Mb/s)
50 路	20%	30%	流畅	36
70 路	43%	44%	流畅	47
100 路	55%	54%	流畅	70
150 路	80%	66%	流畅	95
200 路	90%	78%	非常较轻卡片	146
210 路	92%	80%	有轻微卡片	150
230 路	95%	85%	有中度卡片	176
250 路	95%	90%	有较严重卡片	192

6.4 测试结果分析

- a) 在边拍边传测试中，客户端和服务端较好地实现了客户端采集编码上传，服务器接收用户上传的媒体文件功能。
- b) 在客户端的 rtsp 播放和本机播放测试中，在局域网中，rtsp 播放时延在 1S 以内，较好地满足了实时性的要求。
- c) 在实时转发的测试过程中，从两个客户的播放时延差来看，经统计得出时延在 500 毫秒左右，基本实现了实时转发功能，播放视频质量相对来说也比较流畅。
- d) 从服务器的性能测试中可以看出，用实验室的 PC 作为流媒体服务器，极限在 200 路左右，但通过集群和增加服务器的硬件性能，相信并发量还有很大程度的提高。

6.5 本章小结

本章首先介绍了流媒体服务器和客户端的硬件环境和软件环境，然后介绍了 LVS 集群系统的搭建步骤，接着对客户端和服务端进行了相关测试，并对测试结果进行了分析，表明系统方案的可行性。

第七章 总结与展望

7.1 总结

本文的实现了一个基于 RTSP 的 H.264 实时流媒体传输方案，该系统包括了手机客户端和流媒体服务器两大块。手机客户端实现了音视频的采集编码；视频文件的边拍边传；音视频的解码播放；以及音视频的同步处理。流媒体服务器实现了支持客户端的文件上传，MP4 文件的点播支持，以及实时视频的转发。系统采用了基于 RTP 和 RTCP 协议来进行流媒体的传输控制，从而保证了流媒体传输的实时性和质量。该系统功能丰富，较好地解决了一些关键问题，如支持上传、音视频的同步以及良好的服务质量等，具体较好的扩展性。

论文主要完成以下几方面的工作：

1、首先概述了流媒体的相关技术，包括介绍各种流媒体的协议包括 RTSP、RTP、RTCP 以及 SDP 协议，H264 的编码技术包括网络提取层（NAL）、RTP 包头、H264 的 RTP 传输以及流媒体的主要播放方式。

2、在对流媒体方案进行比较选型之后，对 live555 库和 ffmpeg 库以及 android 的进行分析的基础之上，给出了系统的总体架构图，简单分析了流媒体服务器端和流媒体客户端的功能模块。

3、在 live555 的基础之上对其进行了二次开发，实现了支持客户端的视频上传，MP4 视频格式的文件点播下发，以及流媒体文件的实时视频转发，利用 RTCP 技术解决了拥塞控制等问题。

4、客户端在移植了 ffmpeg 和 x264 的基础之上，利用缓冲技术和时间戳解决了音视频同步播放问题，结合 android 的开发库实现了音视频的采集编码、视频文件分析上传、解码播放以及音视频文件的同步功能。

5、最后对系统进行了测试，包括边拍边传和 MP4 文件的点播、实时转发的延时性能以及服务器的性能分析。

7.2 展望

本系统还有一些不足的地方，包括当视频文件过多时，如何处理文件的存储问题，后期可以通过磁盘阵列和云存储的方式来进行研究；系统的并发量相对来说还不高；由于本论文还只是初期阶段，后续还要再进行相应的完善。

参考文献

- [1] 曲丽君. 基于 H.264 的视频流式传输技术研究[D]. 哈尔滨: 哈尔滨工程大学, 2007.
- [2] 宋宇. 视频监控流媒体服务器的设计与实现 [D]. 广州: 华南理工大学, 2010.
- [3] 高雪峰. 流媒体技术的应用和安全[J]. 信息安全与技术 2010-10-10.
- [4] Xecclipse, RTSP_百度百科, <http://baike.baidu.com/view/70534.htm>.
- [5] IETF.RFC2326.RTSP:Real-Time Streaming Protocol.
- [6] 马海滨. 基于对等网络的流媒体版权保护机制研究[D]. 南京: 南京邮电大学, 2011.
- [7] IETF.RFC3550.RTP: A Transport protocol for Real-Time Applications.Jul, 2003.
- [8] 李季. VoIP 视频服务器的设计与实现[D]. 北京: 北京交通大学, 2012.
- [9] IETF.RFC 2327.SDP: Session Description Protocol.
- [10] 孙倩.基于 RTSP 的媒体服务器流媒体功能研究与实现[D]. 北京: 北京邮电大学 2010.
- [11] 陈锋锋. 基于 RTSP 的流媒体传输系统的应用开发[D]. 南京: 南京邮电大学, 2013.
- [12] 刘洁彬. 面向实时监控的流媒体播放器的设计与实现 [D]. 北京: 北京邮电大学 2010.
- [13] 郭超等. IP 组播技术在视频监控系统中的应用[J]. 北方交通, 2010 年 2 期.
- [14] holyvampire, live555_百度百科, <http://baike.baidu.com/view/3495912.htm>.
- [15] <http://www.FFmpeg.org>.
- [16] leo_ss_pku, FFMPEG_百度百科, <http://baike.baidu.com/view/856526.htm>.
- [17] 杨佳胜. 基于 Android 终端的视频通话系统设计与实现 [D] 大连: 大连理工大学, 2011-10-08.
- [18] 茅炎菲, 黄忠东. 基于 RTSP 协议网络监控系统的研究与实现[J]. 计算机工程与设计, 2011-07-16.
- [19] MAO S, JIE S, XIANG L F. Design and Implementation of Media Player Based on Android[C].2010 6th International Conference on Wireless Communications, NetworkingandMobileComPuting, Chengdu, 2010:4pp.
- [20] JIN H, JIA M H. The Research of Plug-in Extension Technology Based on Android

Multimedia Player Platform. 2011 International Conference on Computer Science and Service System, Nanjing, 2011:874-7.

[21] Wensong Zhang, Creating Linux Virtual Servers, National Laboratory for Parallel & Distributed Processing, LinuxExpo Conference, 1999.

[22] 王晋鹏, 潘龙法, 李降龙. LVS 集群中的动态反馈调度算法. 计算机工程, 2005, 31(19): 40-42.

[23] 杜彬, 王淑玲, 杨海波. RTSP 流媒体服务器性能测试工具. 计算机系统应用, 2011, 20(3).

攻读硕士学位期间取得的研究成果

一、已发表（包括已接受待发表）的论文，以及已投稿、或已成文打算投稿、或拟成文投稿的论文情况（只填写与学位论文内容相关的部分）：

序号	作者（全体作者，按顺序排列）	题目	发表或投稿刊物名称、级别	发表的卷期、年月、页码	相当于学位论文的哪一部分（章、节）	被索引收录情况

注：在“发表的卷期、年月、页码”栏：

- 1 如果论文已发表，请填写发表的卷期、年月、页码；
 - 2 如果论文已被接受，填写将要发表的卷期、年月；
 - 3 以上都不是，请据实填写“已投稿”，“拟投稿”。
- 不够请另加页。

二、与学位内容相关的其它成果（包括专利、著作、获奖项目等）

致 谢

三年的研究生生活马上就要结束了，我也将马上踏入社会，走向工作岗位。回首这三年的时光，碰到过各种困难和挑战，但在老师、同学和自己的不懈努力下，也都一一克服了。在这里我要感谢所有帮助过我的人，是他们让我的研究生生涯更加充实快乐。

首先，我要感谢我的导师吴宗泽老师。吴老师待人平易近人，就像兄长和朋友一样和我们相处。实验室的氛围也很融洽，让我们可以充分地发挥所长。在学术科研上，吴老师也经常指出我们科研和项目方案上的不足，告诉我们研究问题要抓住关键点，要深入到技术细节本身。在平时生活中，吴老师对我也是关怀备至，教导我做人做事的道理，非常庆幸我能得遇恩师。

其次，要感谢胜利老师、傅予力老师、周智恒老师、李波老师、向友君老师以及实验室其他的老师给我学习和科研上的指导与帮助。

感谢实验室的兄弟姐妹们，排名不分先后，吴容、林耀城、华耀波、李石清、曾星、段伊竹、沈东凯、谭国坚、梁啟成、蒋业文、李爽、赖文华、吴东承、陈钦波、张洁柯、陈妍蓉以及其他小伙伴们，你们的相伴，让我的研究生生涯除了学到更多的知识之外，也收获了生活的欢乐与幸福。

感谢我的家人，是你们的支持和理解才让我有勇气辞职后再全职读研，我一直心存感激，希望我能早日尽一点孝心。

感谢我的老婆，工作一年后辞职读研的我一直有你的陪伴和默默地支持，你的付出让我感动，我一定会好好珍惜你。

李罗涛

二零一四年五月

IV - 2 答辩委员会对论文的评定意见

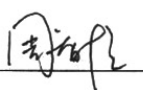
李罗涛同学的硕士学位论文《基于 RTSP 的 H.264 实时流媒体传输方案的研究与实现》，在深入研究 RTSP、RTP、RTCP 协议的基础上，通过对 Live555、FFMPEG、Android 的架构进行详细的分析，提出了一套基于 RTSP 的 H.264 的实时流媒体传输方案。同时详细介绍了实时流媒体传输方案的具体实现，采用 C/S 的开发模式实现了一个实时流媒体传输系统，包括流媒体服务器和客户端。选题具有较好的工程实践意义和研究价值。

论文的主要工作包括：

- 1、流媒体的相关技术进行了介绍，对网络传输协议进行了深入的研究，选择了 RTSP、RTP、RTCP 协议作为网络传输和控制协议，H.264 和 AAC 作为主要的视频、音频编解码标准。
- 2、对流媒体方案进行了比较，选择了 live555 和 FFMPEG 作为主要的系统框架。
- 3、介绍了实时流媒体传输系统的服务器方案，利用 RTCP 技术解决了实时传输的拥塞控制问题，针对 Live555 不支持客户端上传、MP4 文件下发、实时转发等问题，对 Live555 进行了二次开发，增加了上述功能模块。
- 4、通过在 Android 平台上移植 FFMPEG，结合 Anroid API 开发了一个既支持 RTSP 上传又支持播放的流媒体客户端。

论文结构合理，论述清楚，有较好的实践价值，表明作者具有较强的独立科研工作能力，论文达到了硕士学位论文的水平。答辩过程中表述清楚，回答问题基本正确，答辩委员会无记名投票表决，一致同意通过李罗涛同学的硕士论文答辩，并建议授予其工学硕士学位。

论文答辩日期：2014 年 6 月 7 日
 答辩委员会委员共 5 人，到会委员 5 人
 表决票数：优秀 (0) 票；良好 (4) 票；及格 (1) 票；不及格 (0) 票
 表决结果 (打“√”)：优秀 (0)；良好 (√)；及格 (0)；不及格 (0)
 决议：同意授予硕士学位 (√) 不同意授予硕士学位 ()

答辩委员会成员签名	 (主席)	