浙江工业大学

硕士学位论文

论文题目：基于流媒体技术的移动视频直播系统的设计与实现

作者姓名 吴海巍

指导教师 彭 宏

学科专业 信息与通信工程

所在学院 信息工程学院

提交日期 2015 年 4 月 18 日

浙江工业大学硕士学位论文

基于流媒体技术的移动视频直播系统的设计与实现

作者姓名：吴海巍

指导教师：彭　宏

浙江工业大学信息工程学院

2015 年 4 月

Dissertation Submitted to Zhejiang University of Technology
for the Degree of Master

Design and Implementation of Mobile Video Live System Based Streaming Media Technology

Candidate: Wu Haiwei

Advisor: Peng Hong

College of Information Engineering
Zhejiang University of Technology
April 2015

浙江工业大学

学位论文原创性声明

本人郑重声明：所提交的学位论文是本人在导师的指导下，独立进行研究工作所取得的研究成果。除文中已经加以标注引用的内容外，本论文不包含其他个人或集体已经发表或撰写过的研究成果，也不含为获得浙江工业大学或其它教育机构的学位证书而使用过的材料。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人承担本声明的法律责任。

作者签名：吴海巍　　日期：2015年 5月 29日

学位论文版权使用授权书

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，同意学校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，允许论文被查阅和借阅。本人授权浙江工业大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密☑

（请在以上相应方框内打"√"）

作者签名：吴海巍　　日期：2015年 5 月 29 日

导师签名：　　日期：2015年 5月 29 日

基于流媒体技术的移动视频直播系统的设计与实现

摘　　要

随着互联网技术的不断发展及网络带宽的不断改善,流媒体实时传输技术已成为近年来研究的一个热点，而 3G/WiFi 技术的普及、4G 技术的兴起及移动手持设备性能的飞速提升，更加促进了流媒体实时传输技术的移动化，以往基于传统 PC 端架设的有线互联网应用也将逐渐移动化。本文提出的基于流媒体技术的移动视频直播系统是借助手机设备在无线环境下，随时随地通过手机摄像头采集视频，并在远程进行实时观看的通信系统，很好地发挥了移动智能设备的便捷性和无线性，比较适用于移动视频会议、实时新闻发布，自媒体、交通事件监控，灾难现场救援等领域。

本文的主要工作如下 :移动视频直播系统主要分为移动端视频采集软件和流媒体服务器两部分。移动端视频采集软件部署在手机设备端，为了能让手机设备在性能和资源有限的条件下获得更快速的 H.264 编码效率，对 x264 编码库的运动估计算法进行了分析与改进，并实现 H.264 编码 ;而为了能更加适应复杂的无线网络环境，调用 FFmpeg 库将 H.264 视频帧转换成 TS 流并经 RTP 打包进行传输，并且保存 TS 录像 ;同时，利用 HLS 技术实现本地 TS 录像回放，节约了视频解码成本，也提高了视频观看质量。而流媒体服务端是基于 Live555 开源协议栈进行二次开发的，选用 RTSP 协议作为本系统的交互协议，实现服务端和各个客户端的信令通信 ;根据直播需求，流媒体服务器从网络上接收来自手机端视频采集软件的 RTP 包并转发给请求直播的播放器，实现视频流的转发功能。通过这套系统用户可以使用任意支持 RTSP 串流播放的播放器观看来自移动视频采集端软件的直播视频，移动设备端也可以随时随地通过无线网络接入到系统中实现视频直播。本系统中采用 TS 流传输技术既能适应比较复杂、容易发生丢包的无线网络环境，又能方便音频或字幕等业务扩展而无需增加很大成本，具有很好的扩展性。在本文最后对本系统进行了测试，测试结果表明移动视频直播画面清晰，实时性强，性能良好，具有很好的实用价值。

关键字：移动视频直播，流媒体技术，视频编码，TS 流， RTSP 协议

DESIGN AND IMPLEMENTATION OF MOBILE VIDEO LIVE SYSTEM BASED STREAMING MEDIA TECHMOLOGY

ABSTRACT

With the development of internet technology and the improvement of network bandwidth, the real-time streaming media transmission technology has become a hot topic in recent years. The popularity of 3G/Wifi technology, the rise of 4G technology and the rapid performance increase of mobile handheld device promote the mobility of the real-time streaming media transmission technology. The wired internet applications based on the traditional personal computer in the past will be mobility increasingly. In this thesis, the mobile video live system based on streaming media technology is a real-time communication system. It captures real-time video with the help of camera which is built in mobile phone in the wireless network environment, anytime, anywhere, and allows remote users to view video in real time. The system plays the convenience of wireless handheld device, and it is suitable for many fields, such as mobile video conferencing, real-time news releasing, media self-making, traffic event monitoring and disaster site assisting.

The main work is as follows: the mobile video live system is consisted mainly of two parts corresponding respectively to capturing video software built in mobile phone and streaming media server. Capturing video software is deployed to mobile phone, and in order to allow mobile devices to get high-efficiency H.264 encoding performance under the limited processing capacity and resource, the software analyses and improves the motion estimation algorithm included in the x.264 encoding library, and applys it to implement H.264 video encoding. And in order to adapt to more complex wireless network environment, the software builds FFmpeg library to convert H.264 video frame into TS stream transmitted by RTP packages, and saves TS stream in video. At the same time, the software achieves video playback with the usage of local

HLS technology saving the cost of video decoding and improving the quality of video reviewing. However, the secondary development of the streaming media server is based on open source stack named Live555, and the server chooses the RTSP protocol as the interacting protocol for communicating between server and client. According to the live requirement, server needs to receive RTP packets sended by the capturing video software in the network and resends the RTP packets to the player that requests the live video to be broadcasted. Users can use any player supporting RTSP stream to watch live video which is sended by capturing video software through this system. Mobile phone can also access to this system to implement live video broadcasting over the wireless network, anytime and anywhere. The system with the technology of TS stream can adapt to the wireless network environment that is more complex and prone to losing packets, but also can facilitate expansion of business without more cost, such as audio or subtitles. The system has a good scalability. At the end of this thesis,the test results on this syetem show that the picture is clear, live video is real-time. This system has a practical value.

Key Words: mobile live video broadcast, stream media, video coding, TS stream, RTSP

英文名词缩写对照表

HTTP	Hypertext Transport Protocol
HLS	HTTP Live Streaming
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JVT	Join Video Team
PES	Packetized Elementary Stream
TS	Transport Stream
PS	Program Stream
PSI	Program Specific Information
MMSP	Microsoft Media Server Protocol
MTU	Maximum Transmission Unit
MVC	Model View Controller
NAL	Network Abstraction Layer
VCL	Video Coding Layer
NALUs	Network Abstraction Layer Units
CAVLC	Context-adaptive variable-length coding
CABAC	Context-adaptive binary arithmetic coding
RTP	Real Time Transport Protocol
RTCP	Real Time Transport Control Protocol
RTSP	Real Time Streaming Protocol
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VCL,	Video Coding Layer
MPEG	Motion Picture Expert Group
VCEG	Video Coding Experts Group

目　　录

第 1 章　绪　　论

1.1　课题研究背景及意义

当今，我们正处在移动互联快速发展的时代，不但面对巨大的信息量，信息的表现形式也是越来越丰富，越来越多的的信息以音频或视频等多媒体的形式进行发布和传播。流媒体技术是一种顺应时代的发展的在线媒体播放技术，也对互联网发展产生了深远的影响，它的主要特点是把经过压缩编码后的图像和声音信息传输到网络服务器上，以满足用户能够在多种网络带宽环境下欣赏到连续的高品质多媒体节目的需求[1]。从最早的在线音乐播放器，到现在的视频点播、视频监控、视频会议、电视直播、远程医疗、远程教育等，都应用到了流媒体技术。

自从 2007 年苹果公司发布 iPhone 智能手机以来，移动智能设备发展迅速，移动智能设备正在成为每家每户必备的数码产品[2]。以往基于传统 PC 端架设的互联网应用伴随着移动设备性能的飞速发展也逐渐移动化，移动互联网应用也越来越多，人们对它的要求也越来越高，随时随地观看或直播视频便是其中一项，这给移动视频应用带来了新的商机，同时也给移动流媒体技术迎来了新的挑战。

3G/4G 技术的发展，更加促进了无线移动互联网的发展，移动视频直播系统就是借助移动智能终端在无线环境下开展视频直播业务的通信系统，系统进行传输的视频具有无线性和实时性两个重要特征[3]。在移动互联网快速发展的大环境下，现在视频监控、视频会议及视频调度等领域已经开始慢慢引入了移动实时性的特征了，并且已经发挥了明显的优势。例如，D-LINK 公司推出的无线网络摄像头产品（如 DCS-930L）支持即便人们出行在外也能方便地通过手机在 3G 网络和 WiFi 网络进行远程监控，对家中的小孩、老人进行实时看护[4]。未来，移动视频直播将会在突发事件服务、医疗救助和灾难救援等领域发挥巨大作用。

高清化、移动化、多样化必将是视频通信未来发展的方向。随着 CPU 处理技术和视频压缩编码技术的快速发展，视频通信将会深入到教育产业、医疗救助、商务活动及多媒体等方方面面，包括视频监控，视频会议、视频调度等多样化的应用。移动视频直播业务伴随移动互联网的快速发展必将成为一项不可或缺的重要应用，还会在丰富人们业务生活、

促进人与人之间感情交流等应用中发挥显著作用。例如，著名互联网电话公司 Skype 高价收购的 Qik 公司推出的移动视频应用，用户可以通过它自制视频，并上传至云端，任何设备都可以请求观看，也可以将视频分享到 Facebook、Twitter 等社交应用中[5]。

鉴于在无线互联网时代，语音和视频业务将会有长足的发展，本课题针对无线网络环境下，如何开展高质量的移动视频直播通信显得尤为重要，无疑具有重要的实践意义。

1.2 国内外研究现状及发展趋势

1.2.1 移动流媒体的现状及发展趋势

流媒体是指视频、音频、文本字幕等不同格式的多媒体数据，通过实时网络传输协议以连续媒体流的形式从源端传送到目的端，并在目的端接收、缓存后，按照网络传输时间先后顺序进行实时播放连续音视频流的过程[6]。而流媒体技术是一种应用于流媒体的综合技术，它涉及多媒体采集、编码、传输、解码和存储等方面。

目前，随着多媒体及网络领域的相应国际标准与协议的制定，极大地促进和推动了流媒体技术的应用与发展。国际标准化组织(ISO)和国际电工委员会(IEC)下的运动图像专家组 MPEG 制定的 MPEG-1、MPEG-2、MPEG-4 和 MPEG-7 等标准已经广泛应用在广播、电缆网络、有线电视网(CATV)以及卫星直播等领域了[7]；而国际电信联盟(ITU-T)下的视频编码专家组 VCEG 制定的 H.26x 系列标准则在视频点播(VOD)、数字电视、实时视频监控、视频会议和多媒体服务(MS)等领域应用广泛，如 H.261、H.263、H.264 等[8]。

移动互联网以“无处不在的网络，无所不能的业务”为发展方向，随时随地地通过便携式移动终端设备访问互联网的各种应用，已经极大改变了人们的生活方式。随着宽带无线接入技术及移动终端技术的快速发展，移动用户数量迅猛增长。近几年，通过移动互联网接入网络的用户数已经远超通过桌面互联网访问网络的用户数[9]，如 Facebook 的月活跃 8.45 亿用户中有 4.25 亿来自移动端，占 50%[10]；Twitter 用户中移动用户比例为 55%[11]；早在 2011 年 12 月为止, 中国网民突破 5 亿, 移动互联网用户达 3.5 亿, 约占总网民的 70%[12]。

移动流媒体是传统流媒体向移动互联网的延伸，通过向连接上移动网络（如蜂窝移动网络、无线宽带通信网络等）的移动终端设备提供流式多媒体服务，结合移动终端的便利性，使用户能够随时随地地享受“ 实时观看 ”的新型通信服务。并且移动流媒体类型众多，除了典型的移动流媒体点播服务、在线教育及远程监控，还包括移动流媒体消息、流媒体邮件、移动流媒体游戏、移动流媒体电话/会议、移动流媒体购物等新型的移动流媒体应用。

目前，移动流媒体业务已经在很多国家和地区推出，并在某些国家和地区（日本、韩国）得到了良好的发展。3G 技术的普及和 4G 技术的兴起开创了无线通信与 Internet、流媒体融合的新时代，移动流媒体业务也将迎来下一个增长点。最新的移动互联网分析预测报告，到 2016 年，移动流媒体流量的年复合增长率将达到 90%，移动流媒体将占据整个移动互联网中超过 70%的流量[13]。由此可见，移动流媒体业务的发展必将对人们的生产生活带来深远影响。

1.2.2 移动视频直播的现状及发展趋势

视频直播主要分为两类：电视直播和网络直播。其中电视视频直播可以根据内容是否事先采集分为电视栏目直播和电视现场直播，然后运用电视媒体使栏目或现场与异地的观众同时互动的一种信息传播方式。它实现了信息生成、采集、传播、及几乎完全同步的接收，具有异地同步、声音和画面兼备、现场感强、受众面广等独有的特点，这使电视直播一直处于直播行业中最为强势的媒体传播媒介。但其传播方式可以主要包含普通的有线电视直播和卫星电视直播两种，有线电视直播要求用户必须是处于有线电视网并订购了相关电视服务才可以进行观看，而卫星电视直播则是通过地球的同步卫星转发器向服务区发送电视广播信号，供地面用户进行个体或集体接收的业务，这两种传播方式都需要巨大的架设成本，而且业务变动不够灵活，受众的个体或集体接收和观看都要有相应的特殊设备支持。这种传统的电视直播由于存在先天的这些弊端，加上网络直播的兴起和冲击，传统的电视直播近年来也有一些缩水现象[14]。

而网络视频直播是指不同的终端设备，通过有线或无线的方式接入互联网，把互联网当成信息的输入或输出平台，将活动现场信息以多媒体的形式进行传递的一种传播方式，本文所提出的移动视频直播系统就是属于网络直播的一种表现形式。相比传统的电视直播，网络直播采用网络作为大众接入平台，网络用户只要登录相关的网络平台就能参与直播互动，摆脱了传统单项传播的弊端，直接提升了受众的体验感。网络直播的廉价性、便利性及更强互动性直接带动了一些视频网站的兴起，如国外的 YouTube、国内的 Youku 等。

尽管网络直播已经在国内外都得到了比较好的发展，但这些商业化的视频网站并没有发挥手机终端高性能处理能力和便捷性的优势，而是依然需要高价的特殊的拍摄设备进行拍摄后直播给观众，这使得这项直播服务不能很好普及到人们随时随地的生活当中，基于智能终端的移动视频直播仍然处于刚起步的阶段。而本文提出的移动视频直播是网络视频直播的轻应用化的一种，利用随身携带的手机终端通过无线接入网络，对任何现场进行视

频采集直播，充分利用了手机终端高性能处理能力和便捷性，任何人都可以成为一个媒体的来源，使视频直播更为多样化和个性化，同时也特别适用于移动视频会议、实时新闻发布，自媒体、交通事件监控，灾难现场救援等领域。并且在不远将来，4G 和 5G 技术的快速发展，将会给移动视频直播带来更大的发展空间和市场价值。

1.3 本文主要研究内容

在本文设计的基于流媒体技术的移动视频直播系统中，移动视频采集软件是基于 iPhone 手机进行开发的，主要利用了手机的便携性，随时随地通过手机摄像头采集视频流数据，并上传至流媒体服务器，供远程播放器请求实时播放，以达到一些需要即时视频直播或监控的场景需求。实现本移动视频直播系统涉及的关键技术主要包含视频编码技术及改进、信令消息交互、流媒体转发及流媒体传输技术，主要工作内容如下：

1) 为了提高视频数据传输效率和节省用户流量及带宽，调用 x264 开源编码库对视频帧进行 H.264 视频编码处理。

2) 结合移动智能手机设备的性能和资源有限的情况，对 x264 编码库中最耗时的 UMHexagonS 运动估计算法进行分析与代码改进，实现手机端更快速地 H.264 编码效率。

3) 为了能更好的兼容移动无线网络复杂多变的特性，调用 FFmpeg 开源库对编码好的 H.264 帧数据做进一步的 TS 流封装，同时满足了未来语音或字幕等业务的扩展需求，并且 TS 流可以很好的处理音视频同步问题。

4) 基于 Live555 开源协议栈实现流媒体服务器的开发，选用 RTSP 协议作为服务器与各个客户端的信令交互协议，处理 RTSP 信令播放请求，实现流媒体 RTP 数据包的接收与转发。

5) 基于 iOS 系统框架完成移动视频采集软件的设计，用苹果特有的 OC 语言进行开发，实现一个有较好用户体验的 APP 应用。

6) 本系统兼容大部分的支持 RTSP 串流播放的播放器，具有较好的扩展性和兼容性，最后利用具有代表性的 VLC 播放器对本系统进行功能测试。

1.4 本文结构安排

本文设计并实现了基于流媒体技术的移动视频直播系统，整个系统主要包括移动视频采集软件和流媒体服务器。重点介绍移动视频采集编码并打包发送过程、x264 开源库中运

动估计 UMHexagonS 代码改进、以及流媒体服务器的搭建过程，包括 RTSP 服务调度流媒体转发过程。最后，对整个系统进行功能测试和移动视频采集软件进行性能评估，及对改进运动估计 UMHexagonS 代码后的 x264 进行测试比较。

第一章：绪论。论述了课题的研究背景及意义，阐述了移动流媒体技术及移动视频直播的现状及发展趋势，总结了流媒体技术移动化的趋势及移动视频直播的前景，最后给出了本文的研究内容和组织结构。

第二章：流媒体协议及关键技术分析。重点介绍了移动直播系统开发过程中涉及的相关网络协议及编码技术，协议包括 RTSP 协议、RTP/RTCP 协议、HLS 协议；编码技术包括 MPEG-2 标准、H.264 视频编码及 x264 和 FFmpeg 开源库。

第三章：总体架构设计。首先介绍了移动视频直播系统拓扑架构设计，然后分别介绍了移动视频采集软件和流媒体服务器框架设计，及其开发平台的选择。

第四章：移动视频采集软件的实现。重点介绍了移动端进行 H.264 视频编码、封装、打包发送的详细过程，包括针对移动端设备对 x264 编码库中运动估计 UMHexagonS 模块的改进、FFmpeg 实现 TS 流封装、RTP 打包发送过程，及 iOS 软件的设计实现。

第五章：流媒体服务器的实现。详细介绍了如何基于 Live555 开源协议栈实现 RTSP 服务和流媒体转发服务之间的调度，包括 RTSP 服务接收和处理 RTSP 请求过程，流媒体会话建立过程和调度流媒体实现转发过程。

第六章：测试、总结与展望。对整体系统系统的功能测试、移动端软件性能测试及 UMHexagonS 模块的改进测试。总结整个系统设计和实现中存在的不足，对今后的优化工作提出进一步的展望。

第 2 章　移动流媒体的关键技术

2.1　流媒体网络协议

本课题选择 RTSP 协议作为流媒体服务器与流媒体客户端之间的信息交互协议，实现资源描述信息的获取、建立连接、播放实时流等功能，而 RTSP 控制的实时流则是通过 RTP/UDP 协议进行传输的。

2.1.1　RTSP 协议

RTSP 协议作为实时流媒体协议族的成员，它控制实时数据的传送。RTSP 协议的目的是为了选择并控制多个数据源的传送通道。该协议充当多媒体服务器的网络远程控制协议，用于建立和协商在客户端和服务器端实时流会话，并且可以在同一时间内建立并控制一个或几个连续性的媒体流，它本身并不发送实时流数据，但为基于 RTP 上的实时流发送机制提供了对流媒体播放、暂停、快进等操作，开发者可以使用此协议来控制客户端与服务器间的多媒体串流，实现控制视频和音频数据的传输功能[15]。并且 RTSP 提供了一个可扩展框架，使实时数据视频或音频可以以点播方式实现。

RTSP 消息主要分为两种：客户端对服务器的请求消息、服务器对客户端的回应消息。在 RFC8224 中规定了这两种消息的消息格式，包含起始行、标题域(header)、标题结束的空行(CRLF)及消息主体(message-body)。RTSP 请求消息格式如图 2-1 所示，RTSP 回应消息格式如图 2-2 所示。RTSP 常用的消息类型包含 OPTION、DESCRIBE、SETUP、PLAY、TEARDOWN 等。

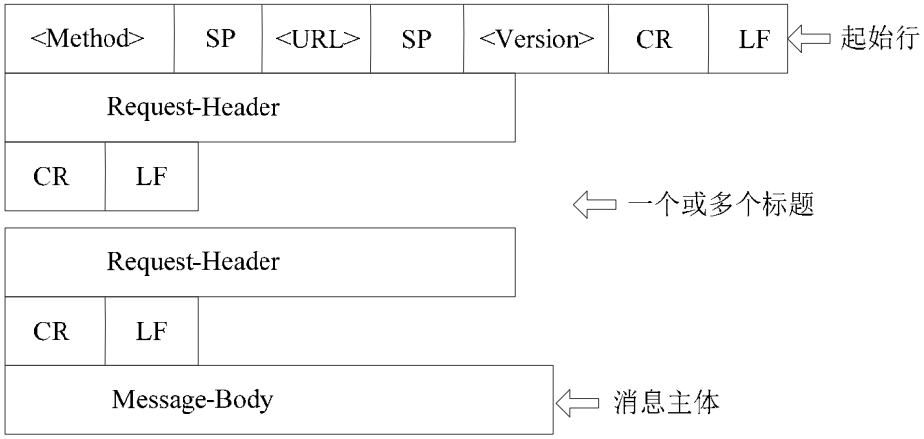


图 2-1　RTSP 请求消息格式

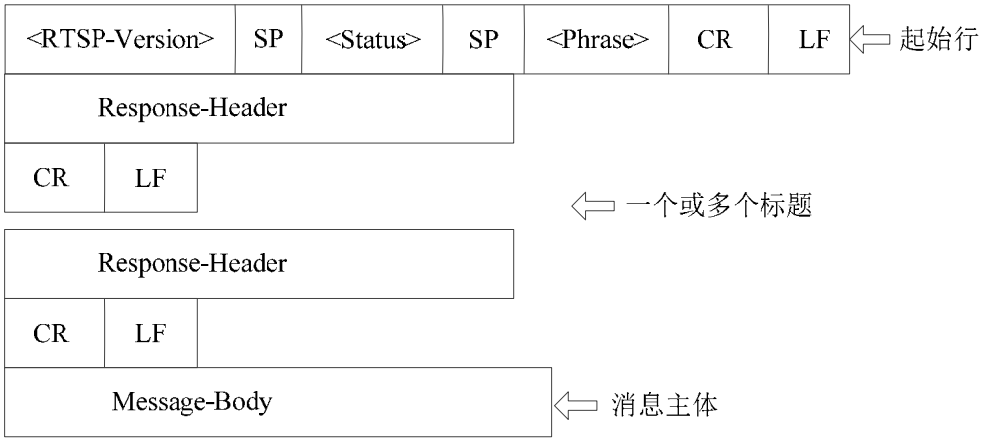


图 2-2　RTSP 回应消息格式

2.1.2　RTP /RTCP 协议

RTP 协议是用于 Intemet 网上的主要针对流媒体数据流的一种传输协议，RTP 被定义为在一对一或一对多的传输情况下工作，其目的是提供时间信号和实时视频流同步[16]。在实际工作中，RTP 数据包一般在 UDP 协议上传送数据，但在特殊的业务需求下 RTP 也可以在 TCP 协议上工作[17]。当应用程序开始一个 RTP 会话时将同时使用两个端口：一个用于实时传输数据包的 RTP 端口，但这个 RTP 端口只用于不断的向接收端实时地传送 RTP 数据包，并不能为按照顺序传送的数据包提供稳定和可靠的传送机制，更不能为 RTP 链路提供必要的网络拥塞控制和流量控制；所以还需要使用另外一个用于为 RTP 这些保障服务的 RTCP 端口, RTCP 和 RTP 一起提供用户所在网络下的最好的流量控制和拥塞控制服务。在 RTP 会话传输数据期间，各参与者会周期性地传送 RTCP 包。RTCP 包主要包含已经发送的数据包的数量、丢失的数据包的数量、间隔抖动等数据，发送端可以根据已接收到的

RTCP 包中所包含的信息动态地调整传输 RTP 包的速率，或相关媒体信息的码率，甚至改变有效载荷类型，以得到最适合该网络环境下的数据流。RTCP 和 RTP 的一起使用，它们能以最小的开销和有效的反馈使传输效率最佳化，因而特别适合用于传输实时视频数据流。

1) 实时传输协议 RTP 技术

RTP 协议主要为实时性数据提供端到端的传输服务。一个 RTP 数据包[17]主要包括 RTP 头、实时视频数据的 RTP 负载、时间戳 TS 和序列号 Seq No 等参数，UDP 协议是实际中应用的负责传输 RTP 包的传输层协议，通过 RTP/UDP/IP 方式传输。由于 UDP 协议对数据包大小的限制，所以每次发生的 RTP 数据包大小不得超过 1460 个字节。RTP 包中包含一个用来标识唯一 RTP 会话的 SSRC 标识符，所以 RTP 还可以利用 UDP 的组播来支持多点传送, 使得 RTP 可以应用于多站点的应用中, 同时也满足了复杂网络多媒体应用的需求。具体的 RTP 数据包格式如图 2-3 所示。

<table>
<tr><td>V</td><td>P</td><td>X</td><td>CC</td><td>M</td><td>PT</td><td>序列号</td></tr>
<tr><td colspan="7">时间戳</td></tr>
<tr><td colspan="7">同步源标示符（SSRC）</td></tr>
<tr><td colspan="7">提供源标识符列表（CSRC）</td></tr>
<tr><td colspan="7">载荷（Payload）</td></tr>
</table>

图 2-3　RTP 数据包格式

V：Version 版本号，默认值为 2。

P：Padding 填充位，置 1 表示对净荷进行填充，用于加密。

X：Extension 扩展位，置 1 表示在数据包头后面跟随扩展报头。

CC：CSRC Count，表示跟随的贡献源标识数量。

M：Marker 标志位，常用于标识帧边界。

PT：Payload Type 载荷类型，标识媒体类型[18]。

序列号：标识 RTP 数据包生成顺序，接收端可用来对包进行排序以及统计丢包率。

时间戳：标识 RTP 包第一个比特的采样时刻，后面包的时间戳按线性增长。接收端可

以根据时间戳来进行媒体内同步和媒体间同步。

SSRC：为随机值且同一 RTP 会话中不同的两个同步源不会产生相同的 SSRC 值。任一同步源都有独立的时间戳和序列号空间，因此不同的发送端和接收端可以通过 RTP 中的 SSRC 来标识。

CSRC：表示载荷来源，CC 字段指定其个数且最多 15 个。

载荷：RTP 数据包中包含的媒体数据。

2) 实时传输控制协议 RTCP

在 RTP 会话过程中，周期性地向所有与会话成员发送控制信息包，通过分析控制信息包得到当前的网络状况。RTCP 的消息格式与数据分组比较相似，一个 RTCP 消息可以由许多可堆积式分组构成，其中每个分组均有自己的类型码和长度指示[19]。RTCP 包的类型主要包含：

接收报告包 RR（Receiver Report）：接收端周期性地向所有的点用多播的方式进行报告。内容主要包括所接收到的 RTP 流的 SSRC、该 RTP 流分组丢包率、接收到的 RTP 流中的最后一个 RTP 分组的序号、RTP 分组到达时间间隔的抖动等信息。发送 RR 分组包主要为了两个目的，第一，可以让所有的接收端和发送端了解当前所处的网络状况；第二，可以让所有发送 RTCP 分组的参与者自适应地调整自己发送 RTCP 分组的速率，根据 RFC 的建议，RTCP 分组的通信量不应该超过网络中的分组数据的通信量的 5%，而接收端分组报告的分组通信量又不应小于所有的 RTCP 分组的通信量的 75%。

发送报告包 SR（Sender Report）：发送端用多播的方式周期性向所有接收端发送的报告。内容主要包括 RTP 流包含的分组数、RTP 流包含的字节数、RTP 流的 SSRC、RTP 流中最新产生的 RTP 分组的时间戳和绝对时钟时间。绝对时钟时间是用于 RTP 流媒体中的图像和声音同步使用的。

数据源描述包 SDES（Source Description Items）：给出会话中参加者的描述，根据参与者的描述信息不同主要可分为：CNAME、NAME、EMAIL、PHONE、LOC、TOOL、NOTE、PRIV 几种类型。

数据源退出包 BYE（Indicates End Of Participation）：关闭一个数据流。

特定应用包 APP（Application Specific Functions）：应用程序能够自定义的分组类型。

同时每个要发送的 RTCP 包也可以由多个 RTCP 包组合而成，但是该组合的 RTCP 包必须是包含一个 SR 或 RR、一个 SDES，且该 SDES 中必须有 CNAME 项的描述信息，该项描述是为了将数据源与该数据源的发送者对应起来。

2.1.3 HLS 协议

HLS 协议是 Apple 公司提出并在 QuickTime 以及 IPhone 上实现的一种基于 HTTP 协议的流媒体交互协议[20]。该协议是基于 HTTP 协议将视频和音频从服务器发送到 iOS 设备上，目前 Apple 公司已有的 iOS 产品主要有 iPhone、iPad、iPod touch 和 Apple TV，当然也可以是装有 OS x 的苹果计算机。HLS 协议支持实时视频广播和视频内容回放，同时为了适应客户端软件不同网络带宽环境（如 3G/4G、WiFi），还支持了为同一节目编码为不同的码率的多个替换流，以便客户端软件可以在不同的比特率的流媒体中进行却换，得到当前网络环境下最适合播放的视频流。为了方便保护媒体发布者对其工作内容成果的版权，HLS 协议提供了通过 HTTPS 协议对媒体内容加密和用户认证的支持。并且 Apple 公司在 iOS3.0 之后的所有设备都内建了支持 HLS 的客户端播放软件，调用系统库解码视频流，以达到性能和效果最优。

HLS 协议规范现在是一项 IETF 因特网草案。HLS 是基于标准 HTTP 之上的协议，在分发流媒体内容上，它避免了传统流媒体协议一些部署成本和维护成本上的麻烦，并且能与大规模的流媒体分发网络无缝接合。

2.2 流媒体视频编码技术

2.2.1 MPEG-2 标准概述

IUT-T 和 ISO 从 1991 年开始联合制定新一代活动图像编码国际标准，于 1993 年底通过了“通用活动图像编码”标准草案，即 ISO/IEC13818（MPEG-2）。MPEG-2 系统层主要有三部分组成:系统层、视频层、音频层[21]。

MPEG-2 系统层原理如图 2-4 所示，系统层协议标准 ISO/IEC13818-1 规定了有关多路 AUDIO、VIDEO、数据的复用和同步标准，也是电视分配、图像通信、多媒体应用中媒体信息传输的基础[21]。而 ISO/IEC13818-2[22]和 ISO/IEC13818-3 标准[23]则规定了视频和音频信号是如何进行压缩编码组合成不定长的分组基本码流 PES 后，再对 PES 分组进行编码，合成单个或多个数据流的。MPEG-2 系统层定义了两种数据传输的编码方式是节目流 PS 和传送流 TS[21]。节目流 PS 是针对错误相对较少的环境而设计的，适用于像交互式多媒体、储存媒介这样一些涉及软件处理系统信息的应用，比如 DVD、VCD，PS 流分组是可变的，一般以基本流的一帧为一个分组。而 MPEG-2 传送流 TS 则是针对那些很容易发生错误的

环境而设计的，它的用途主要是在可能发生严重差错（诸如比特值错误、包丢失等）的环境中进行一路或者多路节目的传输，比如 DVT 等数字视频，TS 流的分组是定长的，为 188 字节。

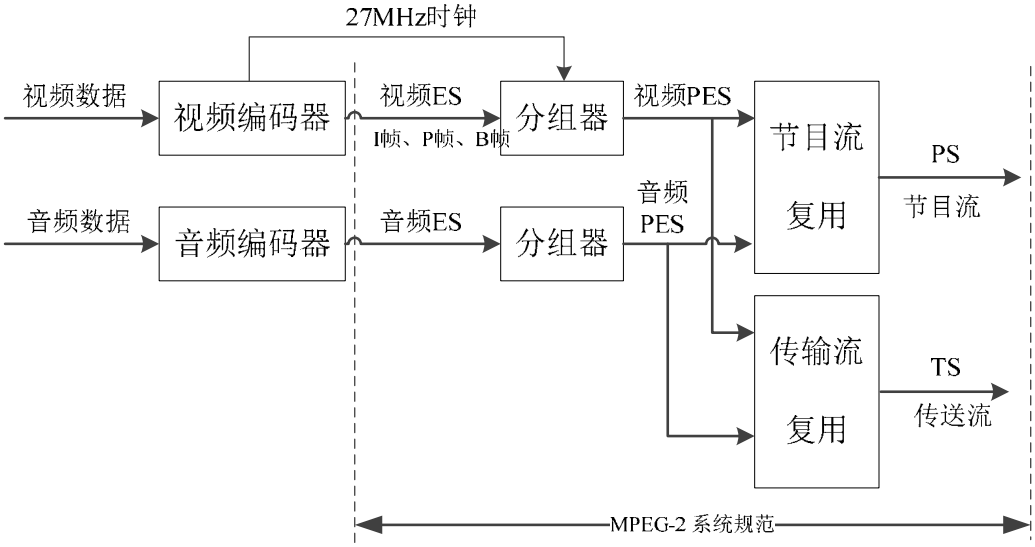


图 2-4　MPEG-2 系统框图

TS 码流广泛应用在数字电视领域，多路节目的复用通常是指综合多路的节目到单路节目中去，在复用过程中也非常灵活，多个节目可以具有相同的时间基础，也可以具有不同的时间基础。在 TS 发送程序中，使用 0x47 作为标识 TS 包的开头，TS 包的大小又是固定的 188 字节，即可以通过检查 0x47 来发现 TS 包头，因此可以比较方便地查找到帧的起始位置，所以即使当传输误码破坏了某一 TS 包的同步信息时，接收解码方可以比较容易从后面某一固定的位置检测出它后面包中的同步信息，及时地恢复同步，避免信息丢失，这也就使得 TS 码流比较适合于恶劣、传输误码比较高的环境。而 PS 包由于长度是变化的，一旦丢失某一 PS 包的同步信息，接收解码端就会导致严重的失去同步状态，从而引起严重的信息丢失事件，所以 PS 码流一般只适应于信道环境比较好、传输误码较低的环境。TS 的包长度为 188 字节，采用该包结构还有个好处就是在未来新增业务时可以在无需新增比特流的情况下，就将新业务在传送层中进行复用传输，获得了很大的灵活性。

2.2.2　H.264 视频编码技术概述

自视频图像出现以来，为了能在尽可能低的码率下获得尽可能高的图像质量，在 2003

年的 3 月，国际电信联盟（ITU）的视频编码专家组发布 H.264 编码标准。H.264 视频编码技术不但在压缩效率上相比于 H.263+或 MPEG-4 提高了近 50%，而且对网络也有比较好的亲和力，保证了网络传输的可靠性，在网络流媒体的处理方面也表现的非常突出[24]。H.264 在高压缩效率和高可靠性传输上的突出表现，使得 H.264 被广泛应用到众多领域，其中包括卫星、有线调制解调器、DSL 上的广播等；通过无线移动网、以太网、ISDN 等网络会话服务，图像传输，或者多媒体流服务和多媒体消息服务；在生活中随处可见的有线电视、数字电视移动接收、远程图像监视、手机电视、视频会议等。

H.264 和 H.263 在基本编码框架是非常相似，但 H.264 在各个部分的技术方案上引入了当前视频编码的许多新技术。可以说，H.264 之所以能获得如此高的性能是以增加存储和计算复杂度为代价的。同时这种高复杂度的计算量也成为 H.264 实现和应用的一个瓶颈。其中运动估计和运动补偿技术是 H.264 编码中非常关键的两个技术，H.264 编码标准中并没有指定运动估计的算法，但运动估计计算的速度和准确性对整个编码性能影响非常大，同时运动估计几乎占据了整个 H.264 编码过程的 60%~80%的时间[25]。因此，H.264 一直在寻找一种能快速提高性能的运动估计算法。目前，“非对称十字型多层次六边形格点搜索”UMHexagonS 算法以其相比快速全搜索算法节约 90%的运算量和较好的率失真性能被 H.264 正式采纳[26]，这也是本文中要介绍的运动估计算法。

2.2.3 x264 开源库介绍

H.264 是目前主流的视频编码标准，JM 是官方给出的标准 H.264 编码器模型标准，其最大的特点是实现了 H.264 标准规定的所有功能，但代码执行效率并不理想。x264 是基于 H.264 视频压缩标准的一个工程实现，是一种免费的、具有更优秀算法的编码库。它同 xvid 一样都是开源项目，前者采用的是 H.264 标准，而后者采用的是 MPEG-4 早期标准。因此，质量相同的视频压缩文件，采用 x264 压缩出的文件要比采用 xvid 小。

x264 之所以如此优秀是因为它为 H.264 的实用性做了部分功能简化，舍去了一些对编码性能贡献小，但计算复杂度高的特性，比如，多参考帧、帧间预测中不必要的块模式等技术。从而极大地降低了编码计算复杂度，但编码性能却没有明显降低。x264 优秀的编码表现还源自于它所支持的特性。首先，x264 采用了 CAVLC/CABAC 多种算法编码[27]；其次，x264 中内置了所有 macroblock 宏块格式，同时 Inter P 对所有的分割块都是支持的。在码率控制方面，x264 采用恒定的分层编制。x264 具有场景剪切侦测功能，支持无损模式，还可以在多个 CPU 中平行编码，这为我们并行处理数据提供了有效支持。

2.2.4 FFmpeg 开源库介绍

FFmpeg 是一个在 Linux 平台上开发的开源项目，具有夸平台性，比如支持 Windows、Mac OS X、Linux 等。它主要为音视频的录制、格式转换、编解码提供一个完整的解决方案。同时，它可以在其他大多数操作系统中编译和使用。并且它目前已经支持 AC3、DV、MPEG、FLV 等 40 多种编码，MPEG、ASF、AVI 等 90 多种解码[28]。如今，已经被许多开源项目所使用，比如 Google Chrome、MPlayer、VLC、QuickTime 的 Perian、Directshow 的 ffdshow 等。FFmpeg 项目是由 Fabrice Bellard 发起，现在由 Michael Niedermayer 维护，项目名字的来自 MPEG 视频编码标准, 前缀 FF 代表 Fast Forward, 表示更快的意思。FFmpeg 是在 LGPL/GPL 协议下开源发布的，任何人都可以自由使用。

FFmpeg 包含了很多库文件，最常用的包括以下四个库文件：libavformat、libavutil、libswscale 和 libavcodec，其中 libavcodec 是目前领先的音视频编解码库。表 2-1 描述了 FFmpeg 的主要组件及其功能。

表 2-1 FFmpeg 的主要组件及其功能

名称	功能
libavformat	用于各种音视频封装格式的生成和解析，主要包括视频格式的解析器 Demuxers 和产生器 Muxer。
libavcodec	用于各种类型音视频编解码。
libavutil	包含了 FFmpeg 函数库所需的公共工具函数，如 Base64 编解码器、DES 加密器、解密器等。
libswscale	用于视频场景比例缩放，色彩映射转换，如 RGB 转换成 YUV。
libpostproc	用于编解码后期效果处理。常常利用它来增加后期图像或音频的效果。
libavfilter	时能够在音视频解码器和编码器之间进行修改或审查的 vhook 替代。
libswresample	是一个包含音频重采样程序的库。
ffmpeg	项目中提供的一个工具，用于格式转换、解码等。
ffsever	一个 HTTP 多媒体即时广播串流服务器。
ffplay	是一个简单的播放器，使用 FFmpeg 库解码，通过 SDL 显示。

2.3 本章小结

本章主要介绍了实现基于流媒体技术的移动直播系统需要用的一些流媒体协议及视

频图像编码技术。流媒体协议包括本系统需要使用的交互协议 RTSP、传输协议 RTP/RTCP、及苹果特有的 HLS 技术 ;视频图像编码技术包括移动视频采集端需要用到的 H.264 编码技术及引出本文后面要进行改进的 UMHexagonS 运动估计算法, 及用于封装 H.264 视频帧成 TS 流的 MPEG-2 标准。最后介绍了两个开源库, x264 负责将采集的原始视频帧编码成 H.264 数据, 而 FFmpeg 用于将 H.264 视频帧封装成 TS 流。

第 3 章　移动视频直播系统的架构设计

3.1　系统的总体设计

本系统是基于 C/S 模式进行架构的，主要由移动视频采集端软件、实时视频流媒体转发服务器、RTSP 信令控制平台及播放器组成。由于本系统要求灵活便捷，适应各种拍摄场合，选择当前使用最主流的手机作为移动视频采集端，手机行业的飞速发展，手机的摄像功能、图像处理功能、手机上网互联等功能将可以远远满足视频采集处理的需求，所以移动视频采集端软件可以部署在每个人随身携带的手机上，具有了方便灵活的特点。移动视频采集端软件作为整个移动视频直播系统的前端软件，结合一些必要的通信协议及视频处理协议栈，最终完成前端与服务端之间的信令交互、现场视频的拍摄、压缩编码、TS 流封装、RTP 打包和发送，及本地录像 TS 文件信息的保存。实时视频流媒体转发服务器配合 RTSP 信令控制平台完成端到端的链路建立、维护链路信息，使视频数据流能在端到端之间高效转发。而 RTSP 信令控制平台作为系统通信协议实现的关键部分，管理系统中各个终端，主要负责处理前端设备上下线和播放器的视频播放请求等会话功能，根据不同信令请求来控制实时视频流媒体的转发工作。播放器在本系统中选择支持 RTSP 协议的开源播放器为主，如 VLC（VideoLan）播放器，降低了开发成本。这种平台化的设计既方便了对移动视频采集端和播放器端的统一管理，又能增强系统整体的负载能力。移动视频直播系统设计架构的拓扑图如图 3-1 所示。

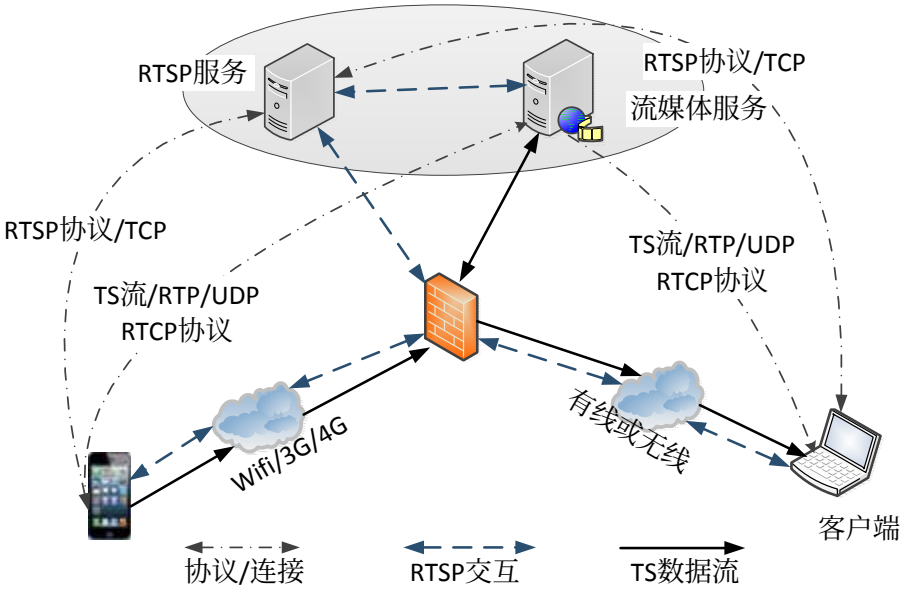


图 3-1　移动视频直播系统拓扑图

在本系统中，移动视频采集端并非只是将采集的原始数据压缩编码成 H.264 视频帧，而是对编码出来的 H.264 数据做了再一次的封装。根据 MPEG-2 标准，将编码出来的 H.264 视频帧作为整个流媒体传输中的 ES 流，再将 ES 通过 PES 分组封装成 PES 包，本系统选择用 TS 流作为最终流媒体传输视频流格式，参照 MPEG-2 标准（ISO13818-1）的视频压缩编码，将 PES 流再按 188B 的固定长度加上必要的节目专用信息 PSI 组成传送 TS 码流。TS 码流还有另外两个个优点，一是能将音视频封装到同一路节目的传输流中，这样避免了传统流媒体中的音视频分开传输带来的一些由于网络环境引起的丢包同步问题；二是 TS 流能配合 HLS 协议在 iOS 设备上调用内置系统播放器解码，大大节省了解码开销，提高了解码性能。但本系统并未涉及到音频相关的业务研究，音频在今后作为扩展业务，也可以方便的插入到本系统中而不需要太多额外的开销。而系统中的交互协议选用 RTSP 协议，具有很强的网络穿透性，并且能很好的与已经存在的一些主流播放器相结合。

3.2　移动视频采集端软件设计

3.2.1　移动视频采集软件开发平台选择

基于上述移动视频直播系统框架，本系统中移动视频采集端软件选择部署手机移动终端上的，当今较主流的手机平台分两种：iOS 平台（苹果手机专有）、Android 平台。本系统移动视频采集软件选择基于 iOS 平台进行开发，出于以下几点原因：

1) 软件与硬件整合度高：iOS 系统的软件与硬件的整合度相当高，使其分化大大降低，在这方面要远胜于碎片化严重的 Android。这样也增加了整个系统的稳定性，iPhone 手机很少出现死机、无响应的情况。

2) 安全性强：对于用户来说，保障移动设备的信息安全具有十分重要的意义，不管这些信息是企业和客户信息、或者是个人照片、银行信息或者地址等，都必须保证其安全。苹果对 iOS 生态采取了封闭的措施，并建立了完整的开发者认证和应用审核机制，因而恶意程序基本上没有登台亮相的机会。iOS 设备使用严格的安全技术和功能，并且使用起来十分方便。iOS 设备上的许多安全功能都是默认的，无需对其进行大量的设置，而且某些关键性功能，比如设备加密，则是不允许配置的，这样用户就不会意外关闭这项功能。

3) 软件平台统一性：iOS 系统由苹果一家公司独有，系统 API 不会像 Android 一样受到手机生产商不同而影响，在开发时不需要刻意去兼容不同品牌的手机，节约了开发成本。

4) 体验和高效性：iOS 手机在界面的体验上比 Android 手机已经存在比较明显的优势，具有简洁、美观、有气质又易用等特点。而在软件高效性方面，iOS 系统更是能在使用较低硬件配置的情况下，达到 Android 手机较高配置的效果。

虽然 iOS 设备有如此多的优点，但 iOS 手机价格昂贵，iOS 系统不开源，比较封闭，无法做一些高权限的操作，这也是 iOS 手机的市场份额一直比 Android 手机低的原因。但出于 iOS 性能优越，iOS 依然受到用户追捧，这也直接导致 iOS 应用市场的收入占整个手机应用市场的 80%以上。综合上述，本文选择在 iOS 设备上开发移动视频采集软件不仅具有更加好的用户体验，而且更加具有市场价值。

3.2.2 iOS 系统架构简介

本系统移动视频采集端软件选择基于 iOS 平台进行开发，了解 iOS 系统架构对开发是必不可少的。iOS 衍生自 Mac OS X 的成熟内核，但 iOS 操作系统更加紧凑和高效，支持 iPhone、iPod Touch 和 iPad 的硬件。iOS 继承了 Mac OS X 的风格，包括：统一的 OS X 内核，针对网络的 BSD 套接字，以及 Objective-C 和 C/C++编译器。iOS 系统框架包含四个层次，分别为 Cocoa Touch（可轻触层）、Media（媒体层）、Core Service（核心服务层）、Core OS（核心系统层）[29]，iOS 系统框架如图 3-2 所示，iOS 系统框架中各层主要负责功能如表 3-1 所示。

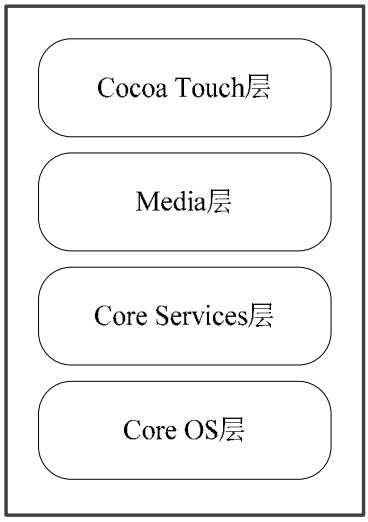


图 3-2　iOS 系统框架

表 3-1　iOS 框架和功能

名称	功能
Cocoa Touch 层	iOS 上关于用户交互的可编程框架，它主要由 UIKit 框架、Foundation 框架和附属框架组成。附属框架包括 Event Kit UI 框架、Game Kit 框架、iAdd 框架、Map Kit 框架、Messsage UI 框架和 Address Book UI 框架。iPhone 应用上的按钮、表格表单、页面过渡以及触摸手势都是通过 Cocoa Touch 框架实现的。
Media 层	对 iPhone 音频和视频协议的封装，主要提供图像渲染、音频播放和视频播放的功能。例如，OpenGL ES、Quartz、EAGL、Core Animation、Core Audio、Open Audio Library、Core Video 和 Media Player。采用这些技术框架可以在苹果设备上创建十分丰富的多媒体体验。他们可以让开发者们从繁重的底层编码工作中脱离出来，专心于创意上的设计，更快捷地设计出外观音效俱佳的多媒体应用程序。
Core Service 层	Core Service 层提供了一些核心框架，由核心服务库和基于核心服务库的高级功能组成，如 Address Book 框架、 Core Foundation 框架、Core Data 框架、Event Kit 框架、CFNetwork 框架等一系列框架。
Core OS 层	Core OS 层位于 iOS 系统构架最底层，它包括内存管理、电源管理、文件系统以及一些其他的操作系统任务，如线程、I/O、和网络。它提供了处理安全问题和应用程序与硬件沟通服务。Core OS 层包括了 OS X Kernel、BSD、Sockets、Keychain、Bonjour 等一系列组件。

本系统的移动视频采集端软件在进行网络通信和发送流媒体数据时主要需要用到 Core Service 层的 CFNetwork 框架；在软件的 UI 交互主要基于 Cocoa Touch 层的 UIKit 框架采用目前比较流行的 MVC（Model 模型、View 视图、Controller 控制器）模式进行设计；Media 层的 AV Foundation 框架用于图像渲染和视频播放功能。

3.2.3 iOS 移动视频采集软件的框架设计

iOS 移动视频采集端软件在设计上遵循 MVC 模式，MVC 模式鼓励弱耦合和组件的重用，它实现了软件业务逻辑、核心控制和视图界面的分离，这样做不仅可以减少代码的耦合和逻辑上的复杂度，还可以大大提高软件的执行效率和可维护性，完成一些界面设计和手势操作功能。这里我们按照自顶向下的原则，将 iOS 移动视频采集软件的设计模块化分为三个层次：视图层、核心控制层、业务层。各层次之间的关系如图 3-3 所示。

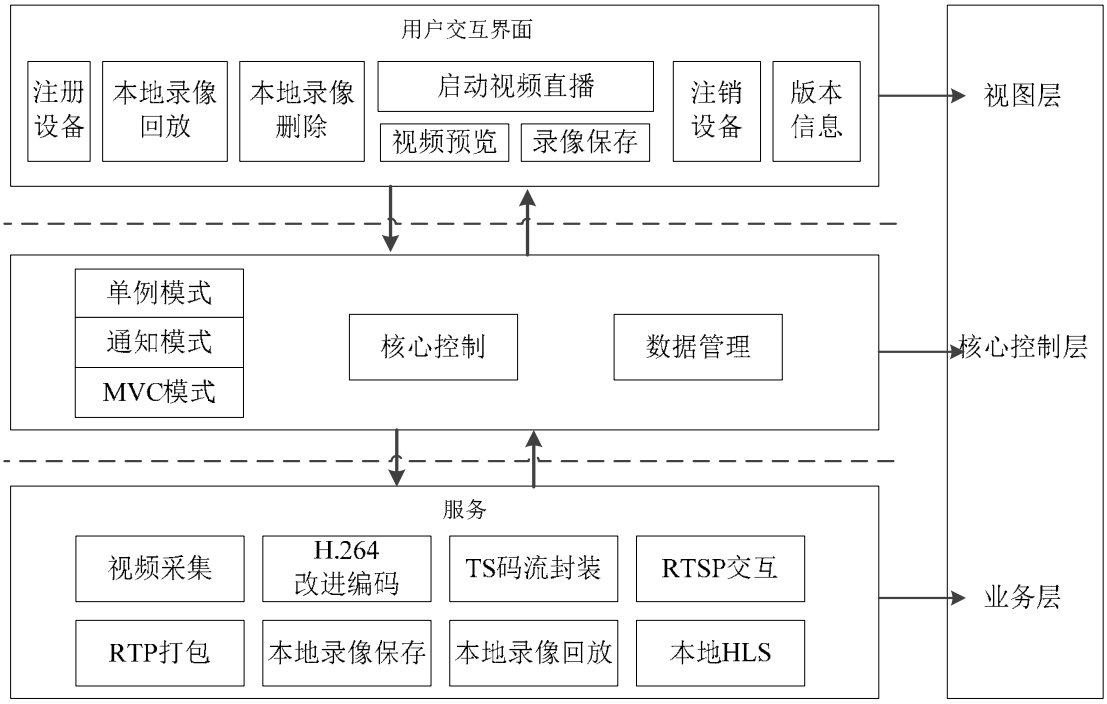


图 3-3 软件总体设计框图

视图层：该层主要负责人机交互、前端用户界面的展示，以及一些业务逻辑的跳转等。其中包括注册设备、本地录像回放、本地录像删除、启动视频直播、视频预览、TS 录像保存、注销设备、软件版本信息等基本业务界面。同时可以根据用户的操作，将相应的操作事件传递给核心控制层进行处理，也可以响应核心控制层传来的信息来更新当前层的显示界面，给用户最直观的体验。

核心控制层：核心控制层主要包括模式设计，核心控制和数据管理三个部分，每个部分说明如下：

1) 模式设计：在本软件设计是为了尽量减小代码之间的耦合、增加代码的复用及降低业务逻辑的复杂度，采用了几个比较典型的设计模式，很好的实现了软件业务逻辑、核

心控制和视图界面的分离，如单例模式、通知模式、MVC 模式。单例模式主要是为了保存应用一些统一的配置信息，作为管理者一直存在在整个应用的生命周期，在本软件中负责管理设备注册信息，录像保存，设备注销，软件版本信息等功能；通知模式主要利用 iOS 系统的通知中心，采用的一对多的方式，需要监听通知的观察者向通知中心注册自己，而通知发出者则向通知中心推送通知，观察者就可以接收到所观察对象发出来的通知，进而响应相应的动作处理该通知，在本软件中启动视频直播和保存录像信息模块中；MVC 模式则是将业务模型和视图模型解耦的主要模式，让业务逻辑可以被抽象出来单独处理，同时也达到了业务逻辑模块的复用，也是在 iOS 软件开发中使用最常用的模式之一，在本软件中主要应用在代码设计上。

2) 核心控制：实时监听用户的点击和手势事件，根据当前软件所处的状态，实现用户界面层到业务层的交互，为各个业务模块提供管理机制。实现的功能包括用户启动视频直播到预览视频、录像本地回放到本地 HLS 服务等功能。

3) 数据管理：主要利用 iOS 系统接口提供的存档功能，负责对录像信息的存档，录像信息包括录像 TS 文件、M3U8 文件、及每个录像所拍摄的时间和录像名称、录像的地理位置等信息。

业务层：用模块化的思想，根据不同的业务需求分成不同的业务模块，为各个模块之间的调用，模块内部功能的封装提供底层实现，各个模块说明如下：

1) 视频采集：主要利用手机的摄像功能，对原始图像数据进行采集，图像的格式、尺寸、及采集图像的速率将直接影响后面视频压缩编码和封装的速率。

2) H.264 改进编码：H.264 视频编码是整个业务层比较关键的业务之一，由于 iOS 系统自身没有配备 H.264 的硬编码的接口，本软件的视频编码是基于软编码的，视频编码的效率会受 iOS 手机不同配置所影响。针对 x264 编码库改进运动估计 UMHexagonS 算法，并实现高效率的 H.264 视频编码。

3) TS 码流封装：将 TS 流技术应用到移动视频直播上，参照 MPEG-2 标准，将 H.264 视频帧封装成负荷数据为 H.264 帧数据的 TS 流，这样以后可以非常方便地扩展封装音频数据到一个传输流中，而无需增加太多额外的开发和维护成本，并且 TS 流在比较复杂、容易产生丢包的网络环境中传输比较有优势。

4) RTSP 交互：主要负责设备的上下线工作，及直播视频邀请和开始直播信令交互。当移动视频直播软件处于在线而不处于直播状态时，服务器端可以利用 RTSP 协议主动邀请移动直播软件开启视频直播；移动直播软件也可以主动直播视频，播放器端可以直接请

求观看。

5) RTP 打包：将已封装好的 TS 码流打包成 RTP 包，然后将打包好的 RTP 包发送给流媒体服务器，并配合 RTCP 协议分析当前网络状况。

6) 本地录像保存：负责 TS 录像文件的保存，及生产对应的 M3U8 文件和录像的附加信息，保存至 iOS 应用程序沙盒 NSDocument 目录下，以便用户对录像进行本地回放。

7) 本地录像回放：调用 iOS 系统视频播放器，访问本地 HLS 服务，播放 NSDocument-Directory 目录下的 M3U8 索引文件，对 TS 录像进行解码播放。

8) 本地 HLS：为本地录像回放搭建一个临时的轻量级 web 框架，这样可以利用苹果推出的 HLS 技术播放 TS 文件回放录像，最大的发挥 iOS 播放性能。

3.3 流媒体服务器端设计

3.3.1 开源框架选择

典型的流媒体服务器有微软的 Windows Media Service，它采用 MMS 协议接收、传输视频，采用 Windows Media Player 作为前端播放器；RealNetworks 公司的 Helix Server，采用 RTP／RTSP 协议接收、传输视频，采用 Real Player 作为播放前端；Adobe 公司的 Flash Media Server，采用 RTMP 协议接收、传输视频，采用 Flash Player 作为播放前端。这些流媒体服务器没有开源，不能根据自身需求经行二次开发不适合本课题开发与研究。

Live555 是一个为 RTP／RTCP、RTSP、SIP 等高效流媒体传输协议提供解决方案开源项目，使用 C++语言编写[30]。它能对多种格式的视频编码数据进行转发，这使得 Live555 成为大多数流媒体服务器首选开发平台，受到广泛应用。同时它要求的前端播放器软件也比较宽泛，只需要播放器支持 RTP/RTSP 协议都能比较无缝的接入这套平台，如现在比较流行的 VLC、MPlayer。由于 Live555 支持多种流媒体标准传输协议，使该开源项目中类众多，类与类之间的关系比较复杂，对阅读源代码比较困难，各个对象生命周期比较难以理清，但作为流媒体技术处理来说，Live555 架构已是相当合理精简了，能同时为不同的流传输方式提供解决方案。

同时，本系统选用 Live555 作为流媒体服务器开发平台的原因还有三个：第一，强大的开源团队经过多年的开发和改进，使 Live555 已经发展成一个性能稳定、容易扩展的开发平台了；第二，它允许无线客户端接入，能方便的利用手机接入直播；第三，Live555 Media Server 对视频数据流的转发延时短，有利于实时视频接通及观看。

本系统的架构设计之初，考虑播放器只需采用支持 RTP/RTSP 协议的播放器就行，如 VLC、MPlayer，这也直接的节约了开发成本，在本章后面的测试均以 VLC 开源播放器为主，甚至在业务需要的情况下，可以在 VLC 开源播放器上做二次开发，但本系统只应用 VLC 播放器播放网络串流的功能，并不做其他功能的开发，所以播放器的开发和设计并不在本文的讨论范围。

3.3.2 Liv555 流媒体服务器架构简介

Live555 开源协议栈中主要包括四个基本组件，分别为 UsageEnvironment、BasicUsageEnvrionment、groupsock、LiveMedia[30]，Live555 的基本框架如图 3-4 所示。以下分别对 Live555 的四个基本库进行简单说明：

① UsageEnvironment：构建程序运行环境的抽象描述，其中包含两个抽象类 UsageEnvironment 和 TaskScheduler。前者主要用于消息的输入输出、管理日志及交互功能；后者作为整个程序的引擎，包含处理 RTSPServer 端的网络功能、异步事件处理等。

② BasicUsageEnvrionment：针对运行环境 UsageEnvironment 组件进行逐步实现，初始化一些成员对象和数据，为控制台应用配置基本环境。

③ GroupSock：管理网络层接口，对套接字和网络接口进行封装，支持广播、组播和单播等通信方式，实现视音频数据包的发送和接收功能。

④ LiveMedia：是 Live555 的核心组件。其中定义了一个最基本的抽象基类 Medium，可以通过继承这个基类实现不同业务模块。MediaSession、MediaSource、MediaSink、RTSPClient 和 RTCPInstance 等都是派生自该类。MediaSession 代表着一个媒体名字，负责建立媒体相关的 RTP 会话，同时管理一个或多个子媒体，如视频、音频或字幕媒体信息；MediaSource 作为媒体数据源的抽象，负责从网络或文件获取数据流；而 MediaSink 则是 Source 的消费者，将 Source 获取的数据发送到网络或存储到文件；RTSPClient 主要负责处理 RTSP 交互过程中的客户端请求；RTCPInstance 实现 RTCP 协议控制消息，监控网络。因此，LiveMedia 在 Live555 整个系统框架中起着至关重要的作用。

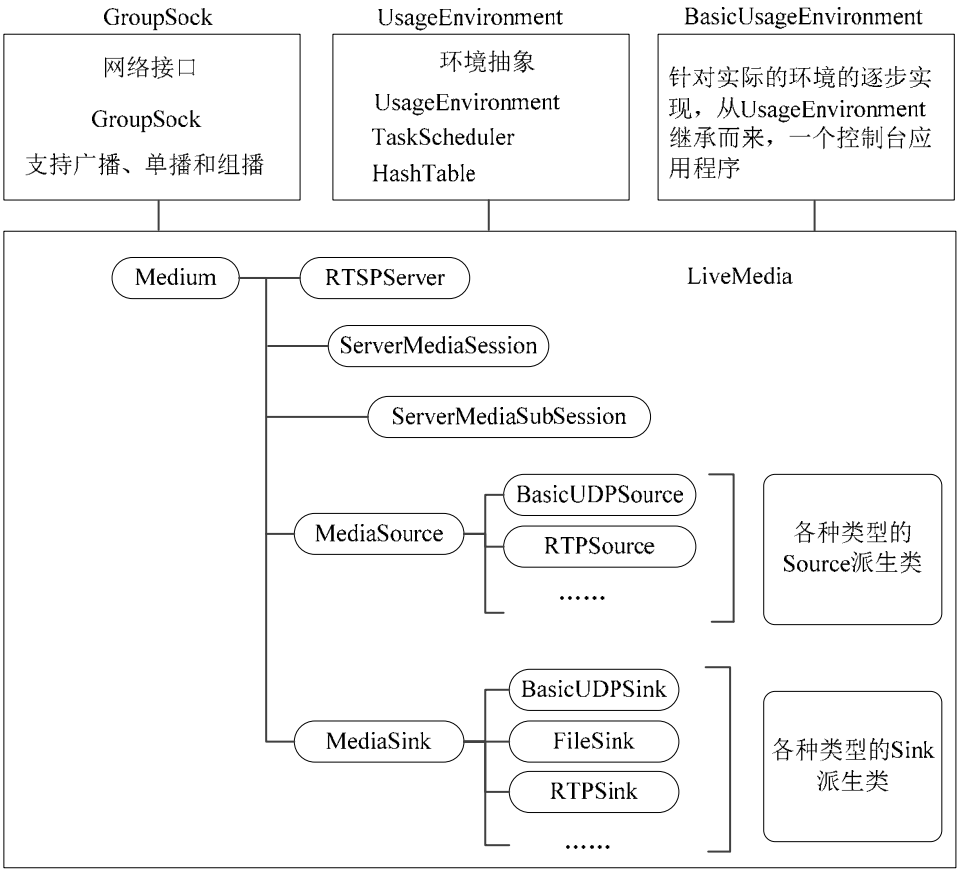


图 3-4　Live555 基本框架图

3.3.3　流媒体服务器端框架设计

流媒体服务器主要包含两项服务：RTSP 服务和流媒体转发服务。RTSP 服务是整个系统通信中的关键部分，主要负责系统中信令消息的交互处理，及视频流链路的会话管理，如播放器的用户的摘要认证、请求直播、暂停、结束等操作，为移动手机端到播放器端提供平台化的统一管理，同时又可以增强整个系统的负载能力；流媒体转发服务采用的是模块化设计，各个模块之间采用高内聚低耦合、按功能划分的原则，主要分成了 4 个模块：流媒体接收模块、流媒体处理或过滤、流媒体发送模块、及 RTCP 控制模块。各模块主要负责的功能如下：

1) 流媒体接收模块：流媒体接收模块可以按照不同媒体格式分成若干子模块，分别处理来自不同媒体的数据，它只负责媒体数据的来源，而不关心媒体数据的去向，也不关心媒体数据是否被转发、或中间是否被处理、过滤等，数据的来源可以是来自本地文件，也可以是网络串流等。本系统中的媒体数据是来自网络的 RTP 包直播串流，那么它就只需要在允许的缓存空间中尽量接收数据，提供给其他模块使用。

2) 流媒体处理、过滤：在整个系统中，流媒体处理、过滤不是一定需要的，针对不

同的业务，可以灵活地加入到系统中，主要实现对来自流媒体接收模块的媒体数据进行相应的处理或过滤掉一些不需要给流媒体发送模块的数据。本系统中的流媒体数据是 TS 流，利用本模块可以很好的过滤或添加一些节目信息到 TS 流中，或做些参数校验工作。

3) 流媒体发送模块：当流媒体数据已经就绪的情况下，流媒体发送模块只需要负责将流媒体数据发送到相应的播放器端，而不关心发送媒体数据的类型、格式及来源。本系统传输流媒体数据的方式是 TS/RTP/UDP 的方式，所以本模块只负责将读入的 TS 流打包成 RTP 包发送到相应的播放器端就行。

4) RTCP 模块：监控播放器端到服务端和移动手机端到服务端的网络状况，对流媒体传输的信道信息进行反馈，从而让服务端和移动手机端做出相应的调整。

媒体管理为 RTSP 服务和流媒体转发服务提供桥梁，每个通道管理着一种类型流媒体的传输工作。媒体管理可以同时管理多个媒体通道，每个媒体通道管理着各自 RTSP 请求的流媒体转发相关工作，它并不关心流媒体是如何传输处理的，这样的设计能让 RTSP 服务和流媒体转发服务完美结合，让各个功能模块充分解耦，使整个系统也更加具有扩展性，更能适应今后业务发展的需求。流媒体服务端框架设计如图 3-5 所示。

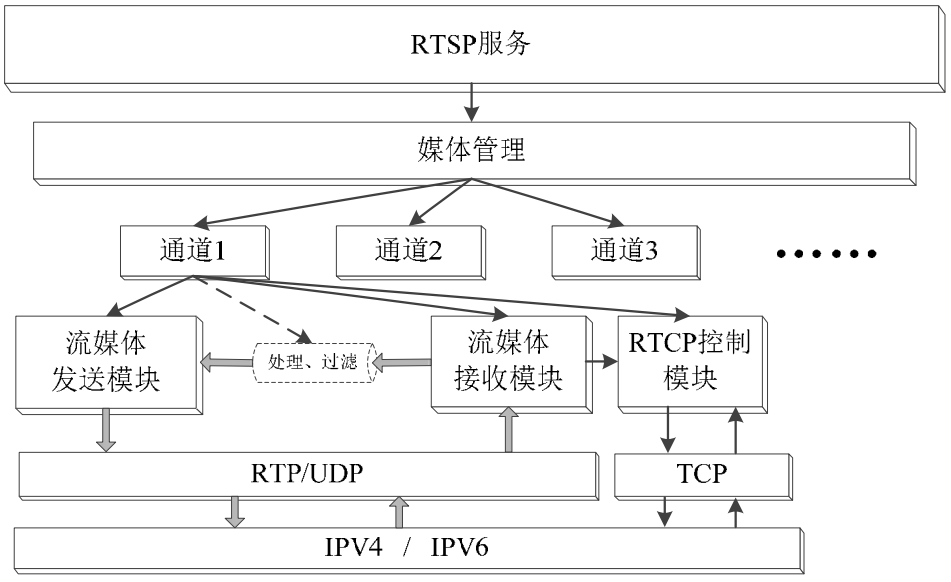


图 3-5 流媒体服务器框架

3.4 本章小结

本章首先介绍了移动视频直播系统的架构设计，并且根据当今手机移动设备发展和流媒体开源框架现状，选择 iOS 平台作为移动视频采集软件的开发平台，及基于 Live555 开

源协议栈开发流媒体服务器。然后介绍了本直播系统中的移动视频采集端和流媒体服务端两大部分的框架设计，及对各个模块的功能进行了说明。

第 4 章　移动视频采集软件的实现

4.1　H.264 视频编码技术改进与实现

4.1.1　H.264 视频编码算法原理概述

H.264 是新一代视频编解码标准，提供了之前视频算法所没有的新特性。它之所以具有高压缩编码效率和高可靠的网络适应性，是因为 H.264 将整个编码算法标准分为网络抽象层 NAL 与视频编码层 VCL 两层[31]。其中视频编码层 VCL 通过时域、空域预测和变换编码来实现对视频的压缩，完成对视频内容的有效描述；而网络抽象层 NAL 则将与网络相关的信息从视频压缩系统中抽象出来，从而使视频编码对于网络来说是透明的，它主要完成对编码后的视频数据进行封装，以提高其鲁棒性、降低误码率。这种分层结构可以实现压缩编码与网络传输分离，有效的解决了视频服务质量和网络服务质量的问题，使视频编码层可以移植到不同结构的网络中，适应不同的网络传输。H.264 的分层编码结构如图 4-1 所示。

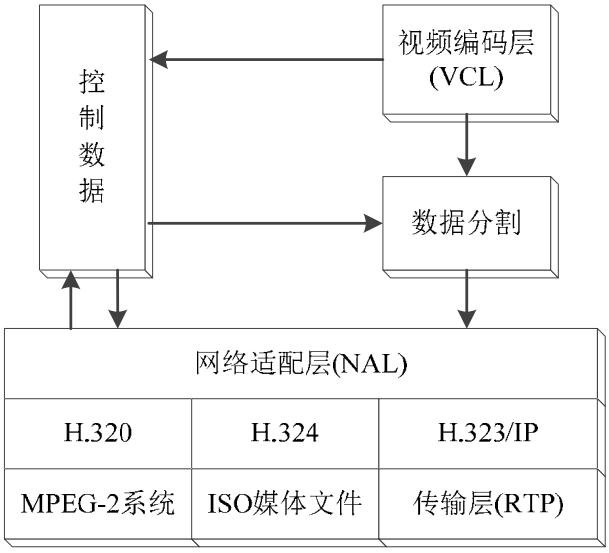


图 4-1　H.264/AVC 分层编码结构

视频编码技术的共同目标就是在尽量低的带宽下提供尽量高的图像质量。H.264 编码技术是在已有的技术 H.263、MPEG4 等编码技术上提出并改进的，其最大的优势体现在以下四方面[32]：

① 视频帧分割成由像素组成的宏块为单位，编码时可以达到块级别。

② 对视频帧采用空间冗余的方法，进行空间预测、转换、优化及熵编码处理。

③ 根据视频中连续图像在时域上存在的强相关性，利用运动估值和运动补偿在时域上进行压缩，临时存放连续帧的不同块，只对连续帧中有改变的部分进行编码，并在已经编码的帧中搜索特定的块的运动矢量，用于编解码中预测主块。

④ 对视频帧里的残差块采用剩余空间冗余技术进行再次转换、优化和熵编码。

H.264 的核心技术为视频编码层 VCL，包括变换、量化以及编码三个模块。每个模块都加入了新的技术，从而使系统的整体性能有了很大的提升。变换的数据根据 I 帧编码、P/B 帧编码，采用帧内预测和帧间预测（运动估计 ME）。宏块类型除了传统的 16×16 的方式外，还增加了 16×8、8×16、8×8 等类型，这样更加接近事物运动模型。变换采用可逆的整数变换，这种方法有效消除了解码过程中的误差积累。熵编码采用上下文自适应的可变字长编码 CAVCL[33]、二进制算术编码 CABAC[34]，得到更低的码流。由于 DCT 块处理会使图像产生块效应，为解决这一问题，H.264 引入了内部环路滤波器[35]。H.264 编码器的功能框图如图 4-2 所示。

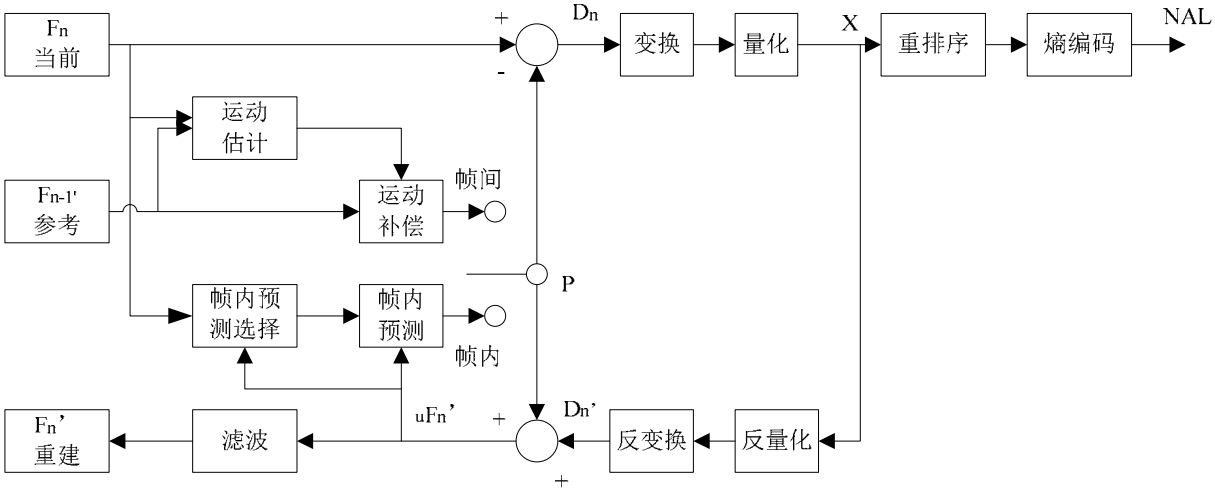


图 4-2　H.264 编码器功能框图

从图 4-2 可以看出，H.264 编码器是采用变换和预测相混合的编码方法。当前帧 F_n 以帧或场的形式输入，并以宏块为单位被编码器处理。若采用帧间预测，可以根据当前片中已编码的参考图像经运动补偿之后得到其预测值 P，其中 F_{n-1}'为参考图像。预测值 P 和当前块相减后可以得到残差块 D_n，再经过变换和量化得到一组量化后的变化系数 X，再对 X 进行熵编码，结合解码所需的头信息，如运动矢量等，最后，经过 NAL 层封装后即为 H.264

编码数据。另外，编码器必须要有重建图像的功能，为进一步提供预测用的参考图像。因此需要将 D_n 经反量化、反变换得到 D_n' 与 P 相加，得到 uF_n'。由于编解码环路中噪声的存在，uF_n'还需要经过滤波处理得到 F_n'，即重建图像，才能用于参考图像。

4.1.2 基于 x264 的快速运动估计 UMHexagonS 算法

H.264 的关键编码算法技术是基于块的混合编码，运动估计和运动补偿是两种比较有代表性的技术。运动估计[36]是根据活动图像的邻近帧中的物体必定存在相关性的理论，将图像分成若干宏块或块，并搜索每个宏块在邻近帧中空间位置上的相对偏移量，而得到整个相对偏移量的过程称为运动估计，得到的相对偏移量就是运动矢量，一般应用于在帧间预测编码中。运动估计示意图如图 4-3 所示。

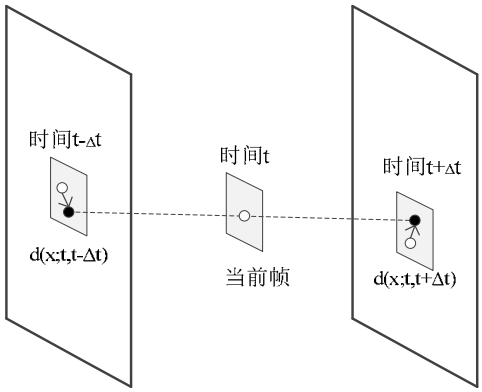


图 4-3　前向运动估计和后向运动估计

运动估计预算法是 H.264 标准中最重要的环节，也是 H.264 优越于 MPEG4 和 H.263 的重要方面。然而其运算的复杂度还是过于高，难以实现高帧率的编码要求，仍有很大的改进空间，运动估算的速度和准确性将直接影响编码性能和效率。目前常用的运动估计方法为块匹配法，最简单的块匹配算法是全搜索法（FS）[37]，其精度极高，性能最好，但计算量太大，不适宜实时压缩；比较有代表性的早期三步搜索法（TSS）[38]和二维对数搜索法（LOGS）[39]，是靠限制搜索点数而减少计算量的，比较容易陷入局部最优；改进型的三步法（NTSS）[38]、菱形算法[40]、六边形算法（HEXS）[41]及被 H.264 采纳的“非对称十字行多层次六边形格点”算法（UMHexagonS）等利用中心偏置特性，提高了搜索精度。其中，又以 UMHexagonS 算法最优秀，该算法使用混合扩展的运动搜索方法，能降低 90%的整像素运动估计计算复杂度，而 PSNR 下降幅度在 0.1dB 以内，是目前为止效果最好的快速运动估计算法之一[42]。但是由于其搜索模块过多以及模块的过于复杂，针对移动手机终

端 CPU 处理性能有限，不同的设备处理能力的参差不齐的问题，编码速度仍不能满足实时直播的需求。所以，在视频质量下降很小但运动估计时间大大减小的前提下，为 x264 编码库选择并优化 UMHexagonS 算法是有必要的。以下先介绍 UMHexagonS 算法的搜索步骤，然后将 UMHexagonS 算法中的比较耗时的搜索模块进行简化，更改相应的 x264 源码，实现更快的 H.264 编码。

H.264 中定义的块匹配误差函数[43]（率失真开销）：

$$J\left(MV,\lambda_{MOTION}\right)=SAD\left(s,c\left(MV\right)\right)+\lambda_{MOTION}\times R\left(MV-PMV\right) \tag{4-1}$$

$$\lambda_{MOTION}=0.85\times 2^{(QP-12)/3} \tag{4-2}$$

公式（4-2）为拉格朗日算子

其中 *SAD*（绝对差值和）定义如下：

$$SAD\left(s,c\left(MV\right)\right)=\sum_{x=1,y=1}^{B_x,B_y}\left|s\left[x,y\right]-c\left[x-MV_x,y-MV_y\right]\right| \quad B_x,B_y=16,8or4 \tag{4-3}$$

s：当前要进行编码的宏块的像素值

c：是已编码重建的用于进行运动补偿宏块的像素值

MV：为候选的运动矢量

PMV ：为预测运动矢量

$R\left(MV-PMV\right)$：代表运动矢量差分编码所需的比特数

QP ：为宏块量化参数

$J\left(MV,\lambda_{MOTION}\right)$最小的点记为最小误差点 MBD（Minimum Block Distortion）

UMHexagonS 算法具体的搜索步骤[43]如下：

步骤 1：起始搜索点的搜索

通过费用计算函数从起始搜索矢量集合 S（主要包括空间中值预测矢量、空间上层预测矢量和时间域邻近参考帧预测矢量）中计算出费用函数值最小的候选运动矢量作为起始搜索点，即最佳匹配点，并作为下一步的搜索的起始点。利用提前终止条件判断是否提前终止，若提前终止则跳转至步骤 4 进行精细搜索，否则跳转到步骤 2。费用计算函数表达式如公式(4-4)所示。

$$MCOST_{\min}=\arg\left[\min_{mi}J\left(mi,\lambda_{MOTION}\right)\right],s.t.mi\in S \tag{4-4}$$

步骤 2：非对称十字交叉搜索

以$cm = m_{\min}$的 MBD 点为搜索中心点，根据自然界中物体运动水平方向要比垂直方向剧烈规律，采用非对称的十字型搜索策略，矢量集合为Ω_1，搜索模板如图 4-4 中(1)示。

$$\Omega_1 = \{m = (m_x, m_y)^T \mid m = (cm_x \pm 2i, cm_y)^T, i = 0,1,2,\ldots,\frac{W}{2}; m = (cm_x, cm_y \pm 2j)^T, j = 0,1,2,\ldots\frac{W}{4}\} \quad (4\text{-}5)$$

W 为搜索区域大小

搜索步骤如下：

1) 查找 MBD：$m_{\min 2} = \arg\left[\min_{m_i} J(m_i, \lambda_{MOTION})\right], s.t. m_i \in \Omega_1$

2) 计算新 MBD：$m_{\min} = \arg\left[\min\left(J(m_{\min}, \lambda_{MOTION}), J(m_{\min 2}, \lambda_{MOTION})\right)\right]$

3) Early-Termination（提前结束）

步骤 3：非均匀多层次六边形格点搜索，该步分成两个子步骤：

步骤 3-1：小矩形窗全搜索

候选运动矢量集合为Ω_2，搜索模板如图 4-4 中(2)所示。

$$\Omega_2 = \{m = (m_x, m_y)^T \mid\mid m_x - cm_x \mid \le 2, \mid m_y - cm_y \mid \le 2\}, cm = m_{\min} \quad (4\text{-}6)$$

搜索步骤：

1) 查找 MBD：$m_{\min 3} = \arg\left[\min_{m_i} J(m_i, \lambda_{MOTION})\right], s.t. m_i \in \Omega_2$

2) 计算新 MBD：$m_{\min} = \arg\left[\min\left(J(m_{\min}, \lambda_{MOTION}), J(m_{\min 3}, \lambda_{MOTION})\right)\right]$

3) Early-Termination（提前结束）

步骤 3-2：扩展的多层次六边形格点搜索

六边形的 16 个点的矢量集合为Ω_{16-HP}，搜索模型如图 4-4 中(3)所示。

$$\Omega_{16-HP} = \{m = (x, y)^T \mid m = (\pm 4, \pm 2)^T, (\pm 4, \pm 1)^T, (\pm 4, 0)^T, (\pm 2, \pm 3)^T, (0, \pm 4)^T\} \quad (4\text{-}7)$$

扩展搜索区域：

for(k=1;k<W/4;k++) {

$\Pi_k = \{m = (m_x, m_y) \mid m_x = cm_x + kgx', m_y = cm_y + kgy', (x', y') \in \Omega_{16-HP}\}$

搜索：

查找 MBD：$m_{\min k} = \arg\left[\min_{m_i} J\left(m_i, \lambda_{MOTION}\right)\right], s.t. m_i \in \Pi_k$

计算新的 MBD：$m_{\min} = \arg\left[\min\left(J(m_{\min}, \lambda_{MOTION}), J(m_{\min k}, \lambda_{MOTION})\right)\right]$

Early-Termination（提前结束）

}

步骤 4：扩展的六边形搜索，该步分成两个子步骤：

步骤 4-1：六边形矢量集合为Ω_3，搜索模型如图 4-4 中(4)所示。

$$\Omega_3 = \{m = (m_x, m_y)^T \mid m = (cm_x \pm 2, cm_y)^T, (cm_x \pm 1, cm_y \pm 2)^T\}, cm = m_{\min} \tag{4-8}$$

搜索：

1) 查找 MBD：$m_{\min 4} = \arg\left[\min_{m_i} J\left(m_i, \lambda_{MOTION}\right)\right], s.t. m_i \in \Omega_3$

2) 计算新的 MBD：$m_{\min} = \arg\left[\min\left(J(m_{\min}, \lambda_{MOTION}), J(m_{\min 4}, \lambda_{MOTION})\right)\right]$

若 MBD 不变$m_{\min} = cm_2$跳转至步骤 4-2，否则跳转至步骤 4-1。

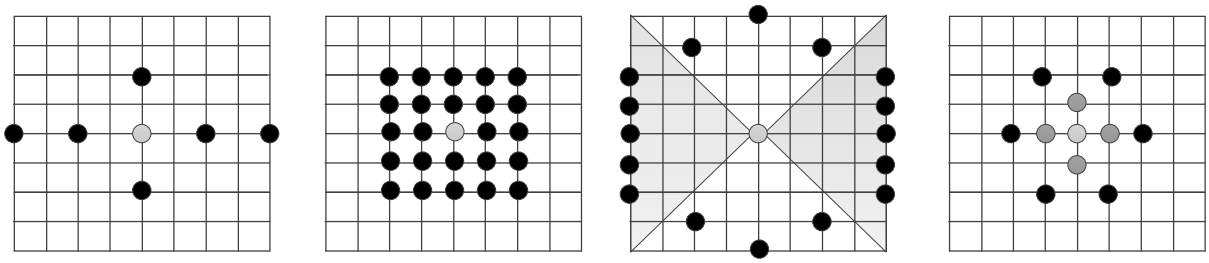
步骤 4-2：该步骤的基本搜索模块为小菱形搜索模块，如图 4-4 中(4)所示，搜索矢量集合为Ω_4。

$$\Omega_4 = \{m = (m_x, m_y)^T \mid m = (cm_x \pm 1, cm_y)^T, (cm_x, cm_y \pm 1)^T\}, cm = m_{\min} \tag{4-9}$$

计算匹配误差：

$$m = \arg\left[\min Cost\left(m, \lambda_{MOTION}\right)\right], m \in \Omega_4 \tag{4-10}$$

若 MBD 不变$m_{\min} = cm$搜索截止，否则跳转至步骤 4-2。



(1) 非对称十字型　(2) 5*5正方形　(3) 非均匀六边形　(4) 六边形和小菱形

图 4-4　UMHexagonS 算法中的搜索模板

UMHexagonS 算法的搜索过程如图 4-5 所示。

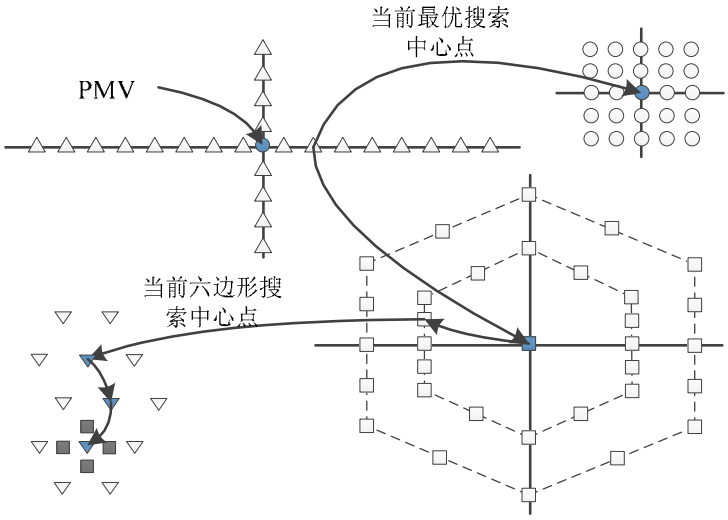


图 4-5　UMHexagonS 搜索过程

搜索起始点的过程主要在 x264 库函数 x264_mb_predict_mv_16x16()中，根据空间相关性，通过使用左、上、左上编码宏块的运动矢量（MV），以获得当前宏块（MB）的运动矢量预测值（MVP），预测矢量的终点作为起始搜索点。代码实现过程如图 4-6 所示。

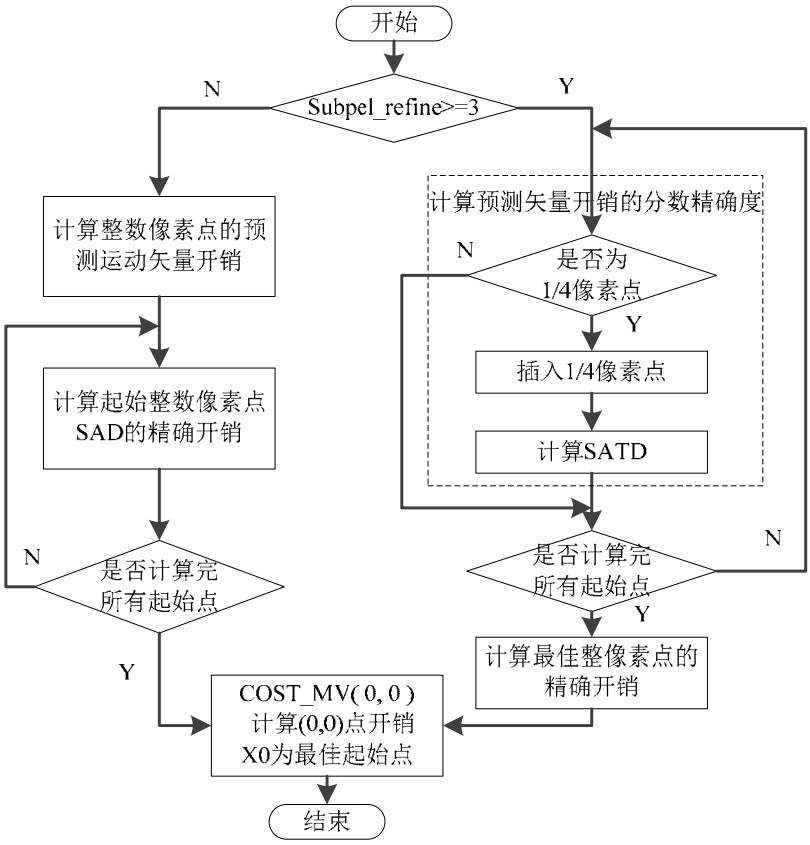


图 4-6　起始点搜索过程

在 x264 中通过 i_me_method 参数设置为 X264_ME_UMH 初始化编码器，运动估计方式为 X264_ME_UMH，主要的运动估计计算过程在 x264_me_search_ref()中，采用的是 UMHexagonS 算法进行搜索。但 x264 开源库源于实际经验，实现过程与官方的 JM 模型[44]实现有比较大区别，它更加注重灵活性、实用性和高效性，更加符合实时的视频编码系统，代码的实现过程如图 4-7 所示。

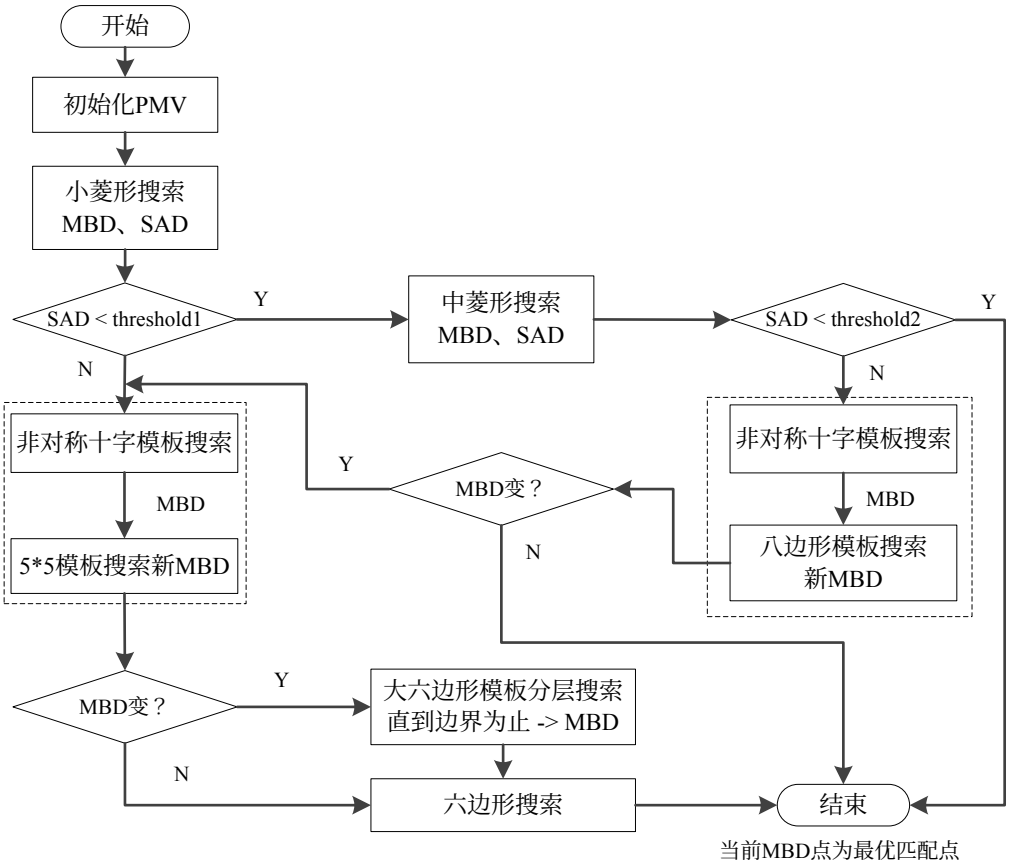


图 4-7　X264_ME_UMH 搜索的实现过程

4.1.3　基于 x264 实现 UMHexagonS 改进

通过对 UMHexagonS 快速估计算法的研究可知，UMHexagonS 结合多种简单模型的特性，高效的预测起点，由中心点分级地向外扩展搜索，可以最大限度地避免搜索点的重复计算，使得 UMHexagonS 算法具有优秀的编码性能。但 UMHexagonS 算法还是存在许多不足，主要表现在以下几方面：

① UMHexagonS 算法的阈值在同一宏块中是固定，对于运动缓慢的视频序列不能有效提前终止，而运动剧烈的视频又过早终止。

② 非对称十字型模板和大六边形模板搜索都更利于水平方向运动矢量搜索，只适用

大部分情况，而没有兼顾在垂直方向上运动剧烈的情况。

③ 5*5 模板和大六边形模板搜索的点数过多（25-1+16*4=88），尽管提高了精确度，但过大的计算量难以满足运动剧烈视频序列的实时要求。

④ 在 x264 中的 COST_MV()函数计算 SAD 时，需要 16x16 次的减法运算，但其实大多数情况不需要计算这么多次就可以判断 SAD 的值已经大于最优的 SAD 值。

所以本文结合 x264 的源码及手机终端普遍偏低的计算处理能力现状，修改局部搜索模板来优化 UMHexagonS 的搜索速度，使拍摄更能适应手持抖动及运动剧烈的实时要求。针对上述的不足点，结合文献[45]、[46]和[47]，对 x264 库中的 UMHexagonS 运动估计算法进行以下几点改进：

1) 加入非对称小钻石型搜索模板

上述应用的非对称十字型和大六边形及其扩展非对称六边形搜索模板都对水平方向比垂直方向上运动剧烈的情况是有利的，为了兼顾垂直方向比水平方向上的运动更加剧烈的情况，通过阅读 x264 源码发现，在非对称十字型搜索得到 MBD 时，如果 MBD 不变，可以进行进一步根据当前 MBD 点的$|omy| \geq 2|omx|$是否成立来判断水平或垂直方向运动更剧烈，然后可以选择水平或垂直小钻石型搜索模板就行进一步搜索新的 MBD；如果 MBD 变，则再做下一步的搜索。垂直非对称小钻石型搜索矢量集合Ω_{ver}如公式(4-11)所示，水平非对称小钻石型搜索矢量集合Ω_{ver}如公式(4-12)所示，搜索模板分别如图 4-8 中(1)和(2)所示。

$$\Omega_{ver} = \{m = (x, y)^T \mid m = (cm_x \pm 1, cm_y)^T, (cm_x, cm_y \pm 2)^T\} \tag{4-11}$$

$$\Omega_{hor} = \{m = (x, y)^T \mid m = (cm_x \pm 2, cm_y)^T, (cm_x, cm_y \pm 1)^T\} \tag{4-12}$$

2) 用新 5*5 正方形搜索模板代替 5*5 全矩形搜索模板

运动矢量具有中心偏置的特性，概率分布是以（0,0）点为中心向四周递减的。通过对标准视频序列运动矢量的统计研究发现，80%的运动矢量集中在搜索中心的 5*5 像素范围内，而中间的 3*3 像素区域的运动矢量就高达 70%，所以中间 3*3 像素区域是必须进行搜索的，而 5*5 全矩形模板的最外层 16 个点运动矢量的概率却只占 10%，但这 16 个点的搜索点数却占总搜索点数的一般以上，显然可以在信噪比要求不高的场合下直接舍弃，或减半搜索。本文选用减半搜索，搜索点数变成 16 点（中心点除外），是原来全搜索 24 点的 2/3，但精确度几乎不变，并经过测试显示，满足视频质量基本不下降或下降很小的前提下，

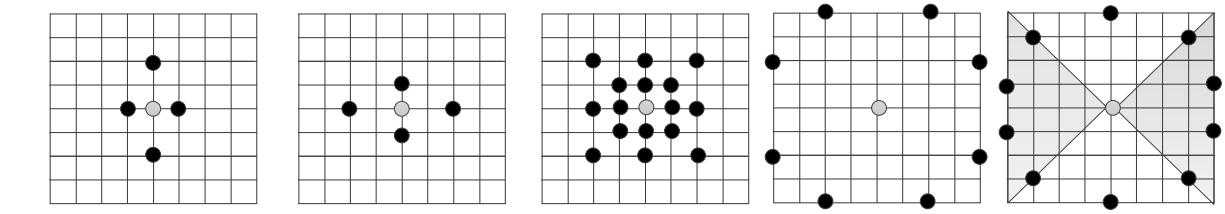
编码效率明显提高。新 5*5 正方形搜索矢量集合Ω_{new}如公式(4-13)所示，新搜索模板如图4-8 中(3)所示。

$$\Omega_{new}=\{m=(m_x,m_y)^T \mid\mid m_x-cm_x|\le 1,|m_y-cm_y|\le 1, m=(\pm 2,\pm 2)^T,(\pm 2,0)^T,(0,\pm 2)^T\} \quad (4\text{-}13)$$

3) 多层级十边形代替多层级非对称六边形

一个非对称六边形有 16 个点，如果限定区域内有 4 层，那么就有 16*4 个点。为了能进一步提高搜索速度，本文用多层级十边形（共 10*4 个点）代替多层级非对称六边形，搜索点数减少近 37.5%，同时十边形更加接近园，同时在水平方向上检测的概率更大，又兼顾了垂直方向的。十边形的搜索矢量集合Ω_{10-HP}如公式(4-14)所示，十边形搜索模板如图4-8 中(5)所示。

$$\Omega_{10-HP}=\{m=(x,y)^T \mid m=(cm_x\pm 4,cm_y\pm 1)^T,(cm_x\pm 3,cm_y\pm 3)^T,(cm_x,cm_y\pm 4)^T\} \quad (4\text{-}14)$$



(1) 垂直非对称小钻石型 (2) 水平非对称小钻石型 (3) 新5*5正方形 (4) 八边形 (5) 十边形

图 4-8 UMHexagonS 算法优化模板

4) 加入自适应阈值技术

自适应阈值是根据运动物体的整体性和视频运动的连续性，对比相邻块的运动矢量和 SAD 值及前后帧对应位置上运动矢量和 SAD 值在空间和时间上的关联性，实时参考前块的左、上、右上及参考帧对应位置的 SAD 值去调整阈值 threshold1 和 threshold2，相比固定的固定的门限值，能更快地检测出最优点，具体设置 threshold1 和 threshold2 如下：

```
threshold2 = MIN(SAD0,SAD1,SAD2,SAD3); //获取相邻块中最小SAD
threshold1 = threshold2 + 1000；
if(threshold2 < 500) threshold2 = 500;
if(threshold2 > 1000) threshold2 = 1000; //限幅threshold2[500,1000]
if(threshold1 > 2000) threshold1 = 2000; //限幅threshold1[1500,2000]
```

5) 修改 COST_MV()宏计算次数

在 x264 库中的 me.c 文件中定义了 COST_MV(mx, my)宏块，用于计算对应点的匹配误差 SAD 值，不同的宏块调整到相应函数执行，其中两个嵌套的 for 循环用于计算 SAD

值的主要修改代码如下：

```
unsigned int calnum;  // 计数器 <= 16*16次
for(y=0; y<ly; y++){        /* lx=ly=16为宏块长宽 */
   for(x=0; x<lx; x++){
        i_sum += abs(pix1[x] - pix2[x]);
        if(i_sum>=bcost) break; // 已经比最优SAD大，提前结束
        calnum++; // 计算一次加1
   }
   if(i_sum >= bcost) break; // 已经比最优SAD大，提前结束
   pix1 += i_stride_pix1;
   pix2 += i_stride_pix2;
}
return i_sum; // 返回SAD
```

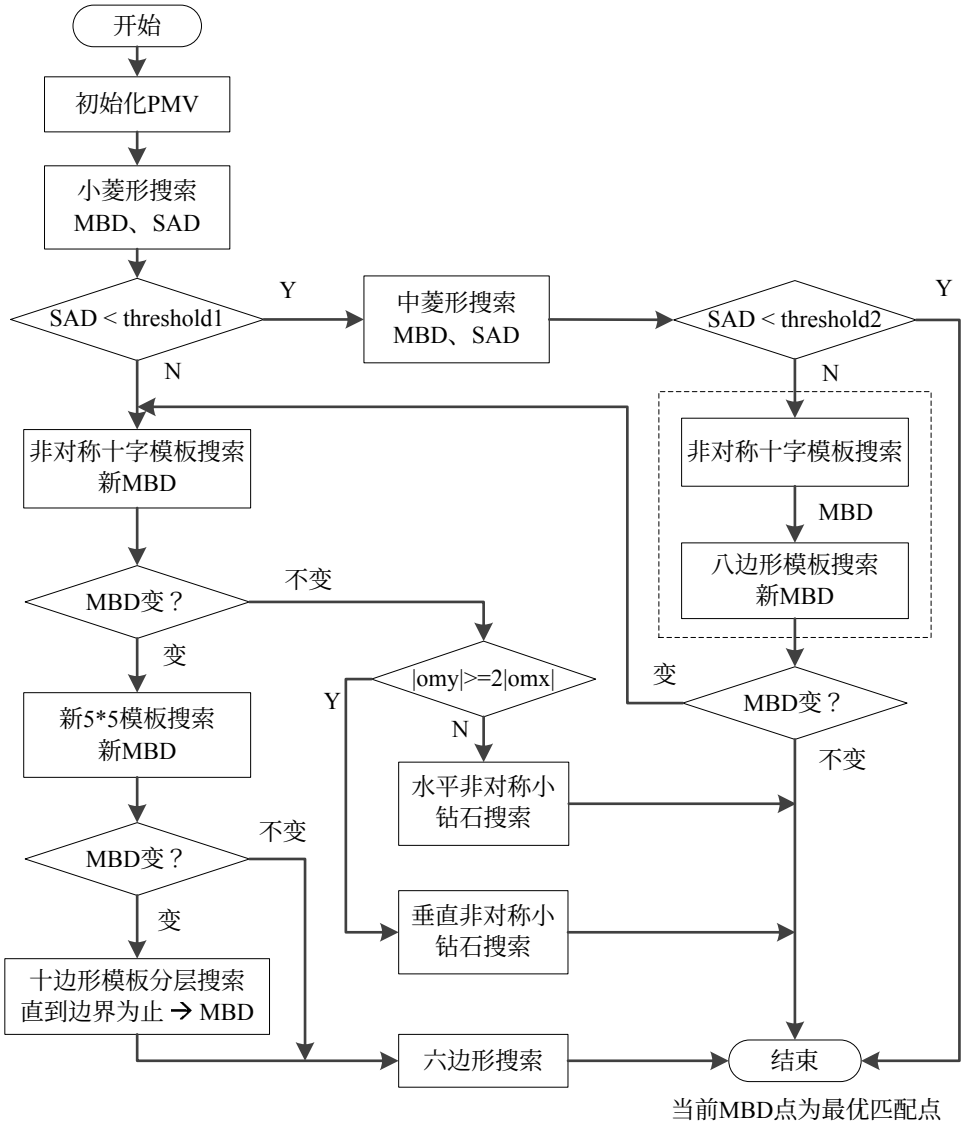


图 4-9　改进型 UMHexagonS 搜索实现过程

综合上述改进方案，对改进型 UMHexagonS 算法的 x264 编码库进行测试发现（测试结果在 6.1 节详细说明），改进型 UMHexagonS 算法相对于原 UMHexagonS 算法的搜索速率明显快了很多，编码帧率约提高 14%。同时改进型 UMHexagonS 算法信噪比（PSNR）比原 UMHexagonS 算法下降不到 0.15db，结构相似性（SSIM）基本没有下降，基本保持原有算法的视频质量。优化后的 x264 编码更适合实时性高的场合。改进型 UMHexagonS 算法搜索的实现如图 4-9 所示。

4.1.4 x264 实现 H.264 视频编码

x264 是一个用 C 语言编写的开源项目，包含 libx264 编码库和 x264 控制台程序[48]。本文按照上一节中所介绍的改进对 x264 开源库进行了相应的改进，但需要在 iOS 平台使用 x264 来实现 H.264 视频编码，在 iOS 平台编译 x264 开源库需要完成以下步骤：

① 先下载 x264 的工程代码，git clone git://git.videolan.org/x264.git。

② 通过 MAC 终端命令行进入 x264 目录，执行./configure –enable -shared –enable -static。

③ 执行 make && sudo make install 就可以生成 libx264.so 和 libx264.a 文件。

④ 进入 ffmpeg 目录，执行./configure --enable-gpl --enable-libx264 将 x264 添加到 FFmpeg 开源库中，如此，便可以通过 FFmpeg 调用 x264 中的功能函数实现编码。

⑤ 将 libx264.a 静态库添加到项目工程中。

如上述 H.264 编码器原理的层级关系可知，要实现 H.264 视频编码主要要实现原始视频帧的 VCL 编码和 NAL 打包。下面本章节将介绍如何调用 x264 开源库实现 H.264 视频编码。

1) VCL 编码视频帧

首先在编码之前创建并初始化编码器，x264 工程库中提供了一个统一设置 x264 编码器所有参数的数据结构 x264_param_t，包括设置 profile、编码复杂度、图像质量、码率、SPS/PPS、I 帧间隔等参数，并可以通过 x264_param_default 函数设置默认参数，在实际编码过程中只需去调整必要的参数来适应不同的编码需求。本视频采集端软件会根据当前的网络环境尽量给客户端最好的视频观看质量和体验，可以对编码器的以下字段参数进行调整，如表 4-1 所示。

表 4-1　x264 编码器参数描述

字段	描述
x264_profile_names[]	x264_profile_names[]数组中包含了 baseline、main、high 等档级，通过函数 x264_param_apply_profile 可以设置编码器的档级。
i_level_idc	设置比特流的 level，取值范围 10~51。默认为 40，即 4.0。用来告诉编码器需要支持什么级别的兼容性，根据应用需求设置。
f_rf_constant	控制图像实际质量，值越大图像越花，越小越清晰。
f_rf_constant_max	设置图像质量的最大值。
i_rc_method	码率控制参数，包括 CQP（恒定质量）、CRF（恒定码率）、ABR（平均码率）。
i_bitrate	编码输出比特率，并启用 ABR（平均码率）模式。
b_repeat_headers	值为 1 时，使每个 I 帧都附带 SPS/PPS；为 0 时只在流开始处产生一次。
i_keyint_max	I 帧的最大间隔帧数。
i_keyint_min	I 帧的最小间隔帧数。
i_bframe	设置最大 B 帧数。
i_me_method	设置运动侦测方式，可选项包括 ME_EPZS、ME_HEX、ME_UMH、ME_FULL、ME_ESA 等。

完成编码器初始化只是实现 H.264 视频编码的第一步, 接下来需要为编码器读入 YUV 格式图像帧数据分配必要的图像空间，YUV 格式是按亮度和色差来描述颜色空间的，而本视频采集软件是基于 iOS 平台进行开发的，原始视频帧都是通过 iOS 设备的前置或后置摄像头采集得到的，采集的原始视频图像帧数据是 RGB 格式的，RGB 格式是按照三基色加光系统的原理来描述颜色空间的, 所以在 x264 编码器读入视频图像帧之前需要做一个 RGB 到 YUV 格式的色彩映射转换，先将其转换成 YUV420 格式的，此功能是用 FFmpeg 开源库 libswscale 组件来完成完成，在下一小节中讲到 FFmpeg 开源库时再做介绍。

正如上述的 H.264 视频编码原理所介绍的一样，在 x264 编码器中，提供了一个数据结构体 x264_t 用于保存编码器的初始化信息和编码中的一些中间产物，其中子结构体字段 frames 分配了四个临时视频帧序列空间，分别为 current、next、reference 和 unused，它们分别用于保存当前编码帧、下一个编码帧序列、参考帧序列和未处理视频帧序列，编码器还申请了 fenc 和 fdec 空间用于存放已编码帧和重建帧，各参数的主要用途参照图 4-2。

VCL 层编码过程主要在 encoder.c 文件中的 x264_encoder_encode()函数中完成, 分成四步：第一步相关参数的初始化，包括帧类型获取、写码流结构的初始化、码流控制获取当

前帧编码量化步长、写片头序列参数集（SPS）、图像参数集（PPS）；第二步进行以宏块为单位对图像帧数据进行编码；第三步为保存编码过程的中间产物信息，待编码完一帧图像数据时由函数 x264_macroblock_cache_save()完成保存，以保存编码器的状态信息；第四步对已经编码完成的一帧图像进行去块效应、环路滤波等处理，并从 unused 空间中取出新的视频帧放入 fanc 空间以备下次编码使用。编码过程中使用到的一些主要函数及功能如表 4-2 所示。

表 4-2　x264 VCL 编码中的主要函数及功能描述

函数名	功能描述
x264_encoder_open()	用于创建并初始化编码器。
x264_macroblock_analyse()	分析参数，确定最佳编码模式。
x264_macroblock_probe_pskip()	功能是判断宏块是否为跳过编码模式。
x264_mb_analyse_inter_p16×16()	实现帧间预测。
x264_me_search_ref()	实现运动估计。
x264_me_refine_qpel()	实现亚像素精细搜索。
x264_mb_analyse_intra()	实现 Intra 宏块的模式抉择，依次检测 Intra_16×16、Intra_4×4、Intra_8×8 的最佳模式.
x264_mb_analyse_transform()	函数实现对残差进行 8×8 的 SATD 变换。
x264_macroblock_encode()	完成宏块编码。
x264_macroblock_write_cavlc()	采用 CAVLC（上下文自适应变长编码）方式进行熵编码。
x264_macroblock_write_cabac()	采用 CABAC（上下文自适应的二进制算法编码）方式进行熵编码。
x264_macroblock_cache_save()	保存编码信息。

在 x264 编码库中本身提供了两种编码方式：上下文自适应变长编码（CAVCL）和上下文自适应二进制算法编码（CABAC）。CAVCL 适用于亮度和色度残差数据的编码，与 CABAC 编码方式相比，它具有很高的压缩比，能够大大减少处理要求，也是一种无损的压缩技术。VCL 编码最核心的部分就是以宏块为单位对图像进行压缩编码，需要大量的矩阵运算，是整个编码过程中最耗时的部分。本移动视频采集软件是基于 iOS 设备进行开发的，处于对编码效率、设备性能消耗及移动设备的电能消耗等方面考虑，选择 CAVCL 的方式进行编码，这也是 CAVCL 在实际应用中得到广泛应用的原因。

2) NAL 打包

NAL 主要是将已经完成的 H.264 视频帧进行打包封装，让 VCL 视频编码不与网络相关的信息相关联，提高码流的健壮性以及更能适应网络传输，降低误码率。在 x264 库中，可以通过库函数 x264_nal_encode()和 p_write_nalu()来实现 NAL 打包。当然实现 H.264 视频编码，得到 NAL 视频数据之后，编码结束时不要忘记释放编码所占用的内存空间并关闭编码器，节省手机设备的资源。x264 编码大致流程如图 4-10 所示。

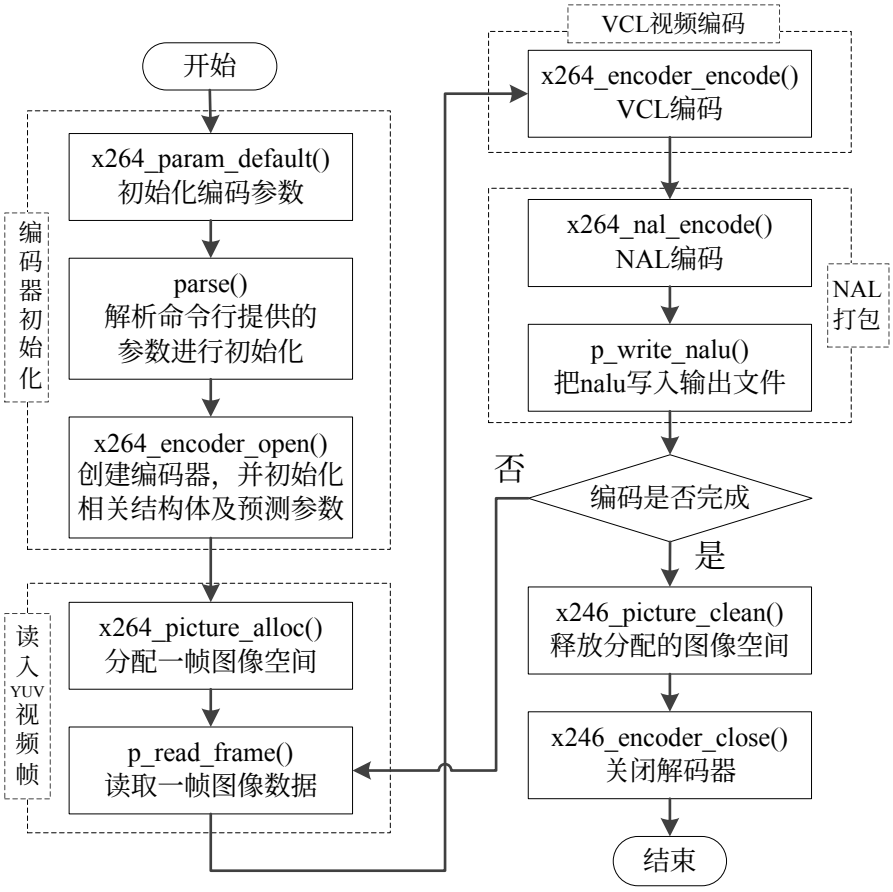


图 4-10　x264 编码流程

4.2　FFmpeg 实现 TS 打包

4.2.1　MPEG-2 TS 包结构分析

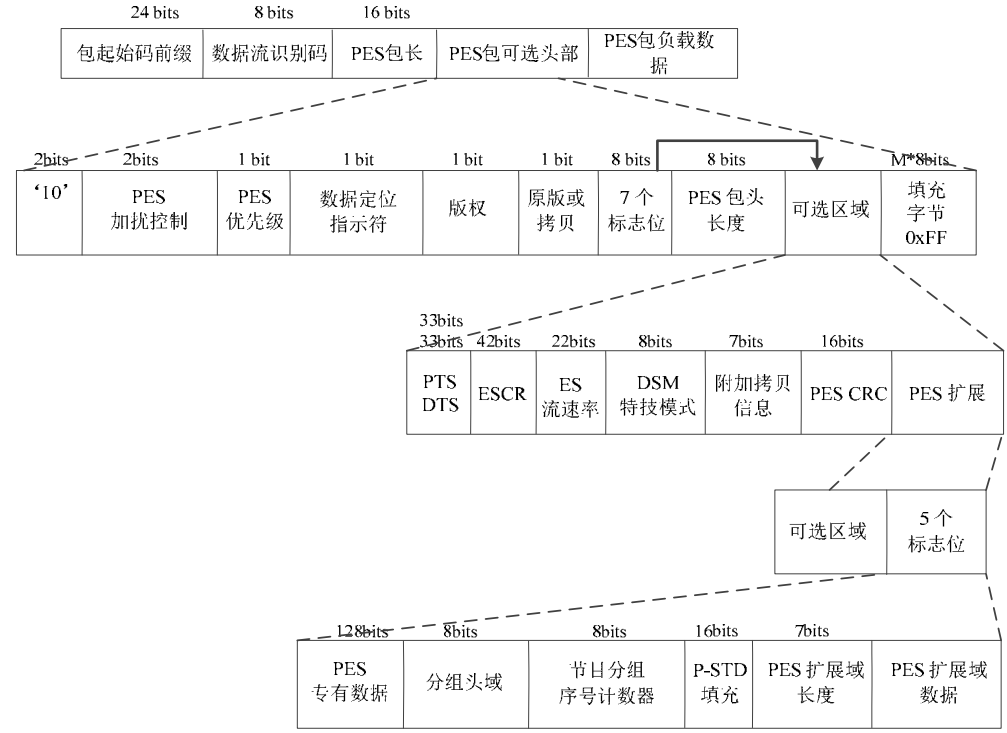
MPEG-2 系统层的主要作用就是将一个或多个音频、视频或其他的基本数据流合成适于存储和传送的单个或多个数据流。两种不同的系统编码方式 :传输流(TS)和节目流(PS)，分别适应于不同的场合。结合本系统的设计思想及移动视频采集端的网络环境情况，本前端软件决定采用 TS 流来传输 H.264 视频帧数据，第一是更能适应移动互联网复杂的网络

特性，第二能更好的适应未来业务发展的需求，具有更好的扩展性，如添加音频业务时可以无缝的接入这套系统，第三视频接收端非常简单，普通能播放网络串流的播放器都可以胜任，大大降低开发成本。

前面章节我们已经介绍了如何利用 x264 视频编码库得到 H.264 视频帧数据，也就是 MPEG-2 系统层中的 ES 流。ES 流还需要经过 PES 打包器打包成 PES 流，最后 PES 流经过 TS 复用器将视频数据封装到节目通道中传输到网络，接收端进行解复用解码播放视频。所以在进行 TS 流封装之前，先了解下 MPEG-2 TS 包结构是有必要的。

1) PES 包结构

PES 包主要由 PES 包起始码、PES 头标志、PES 包头域及 PES 包负载数据部分组成，图 4-11 给出了完整的 PES 包结构图[21]。



- PTS - Presentation Time Stamp 显示时间标记
- DTS - Decoding Time Stamp 解码时间标记
- ESCR - Elementary Stream Clock Reference 基本流参考时钟
- DSM - Digital Storage Media 数字存储媒体
- CRC - Cyclic Redundancy Check 循环冗余校验
- P-STD - Program System Target Decoder 节目系统目标解码器

图 4-11 PES 包结构图

2) TS 流

原始的视频流先经过编码为 ES 流，ES 流再打包成 PES 流，最后 PES 流作为数据负

载传送到接收端。传输流将属于同一节目的各个原始数据流的 PES 数据包设置一样的时间基，可以将时间基相互独立的多路节目合成一个单独的数据流，并成一路传输，提高传输效率，同时在信道环境较差的情况下也适应传输，具有较强的网络适应能力。

传输流 TS 的速率可以是固定码率也可以是变码率，对于组成 TS 流的基本流，同样也可以是固定码率或变码率的。TS 流是由一个个 TS 数据包组成的，TS 数据包长度为固定的 188 字节，每个 TS 数据包都由包头、自适应区和有效负载三部分组成，而 PES 包则是作为 TS 包的有效负载加以传输，每个 PES 包的包头的第一个字节是传输流数据包有效负载的第一个字节，TS 包结构如图 4-12 所示[21]。

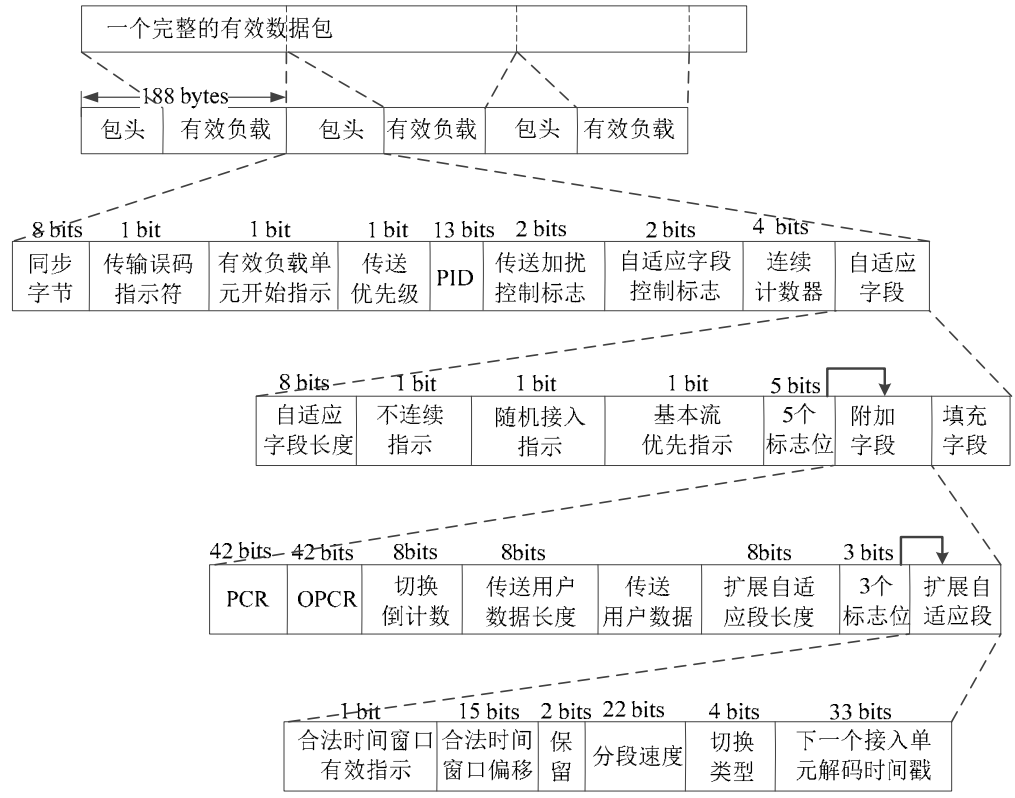


图 4-12　TS 包结构图

4.2.2　FFmpeg 实现 ES 流打包成 TS 流

1）FFmpeg 在 iOS 平台的移植

FFmpeg 是基于 Linux 平台开发的开源项目[49]，底层采用的是 C 和 C++代码，并且它支持大多数操作系统，但由于不同硬件设备使用的处理器指令不同，因此需要区别对待。当前的 iOS 设备硬件处理器主要包含 arm6、arm7、arm7s 及 arm64 处理器，FFmpeg 移植

到 iOS 平台时，针对不同的 iOS 处理器，需要修改不同的交叉编译参数，但编译过程基本相似，本文以比较主流的 iOS SDK7.1 和 arm7s 编译 FFmpeg 库进行说明。编译脚本如下：

```
--sysroot=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS7.1.sdk \
--extra-ldflags='-arch armv7s -miphoneos-version-min=7.0 -isysroot/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS7.1.sdk' \
--prefix=compiled/armv7s \
--disable-ffplay \ --disable-ffserver \
--cc=/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/usr/bin/gcc \
--as='gas-preprocessor.pl /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/usr/bin/gcc' \
```

采用 gcc 交叉编译器，指定开发和编译运行环境，用 shell 命令行进行编译，编译完后生成四个静态库，分别为 libavcodec.a、libavformat.a、libavutil.a 和 libswscale.a，最后只需要将要用到的静态库导入到项目工程，并引入必要的头文件就使用 FFmpeg 库进行开发，在本软件中主要用到 libavcodec.a 与 libswscale.a 实现 RGBA 视频帧到 YUV 视频帧转换、及利用 libavformat.a 来实现 ES 流到 TS 流的封装。

2) 实现 RGBA 帧转换成 YUV 视频帧

转换过程主要要使用到 libswscale.a 中的三个函数，sws_getContext()函数用于初始化原始图像数据和输出图像数据的高和宽、输入和输出图像类型、及 scale 算法类型，本软件采用的是 CIF 格式的图像，高宽为 352x288，输入格式为 PIX_FMT_BGRA，输出格式为 PIX_FMT_YUV420P，为了获得最快速地转换采用 SWS_FAST_BILINEAR 类型算法，最高转换帧率可以到达 95（在高配置的硬件设备下），足以满足本软件的转换效率要求；sws_scale()函数是图像转换或缩放的执行函数，主要是对原始图像进行扫描运算出另一格式或尺寸的图像；sws_freeContext()函数为结束函数，需要释放相应资源。转换过程如图 4-13 所示。

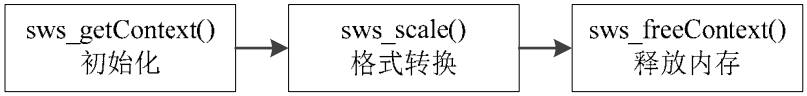


图 4-13 图像格式转换

3) FFmpeg 实现 ES 流封装成 TS 流及本地保存 TS 录像

在本软件实现时 x264 编码出来的 H.264 视频帧并没有直接将其打包成 RTP 包发送出去，而是先将其转成 TS 流，便于网络传输和业务扩展。优秀的 FFmpeg 库给出了几种文件格式之间的转换，但本软件的需要转换的是实时的输入 H.264 视频帧数据，所以需要对读入文件时的操作更改成读取 H.264 视频帧，此功能由 init_packet()函数完成；封装完之后写入到文件操作时，还需要将 TS 数据包发送到网络服务器，send_RTP_WithTSPacket()完成发送，下一小节中详细介绍；最后直播结束时，要对本次实时播放数据进行保存录像信息，save_M3U8()完成。转换流程如图 4-14 所示。

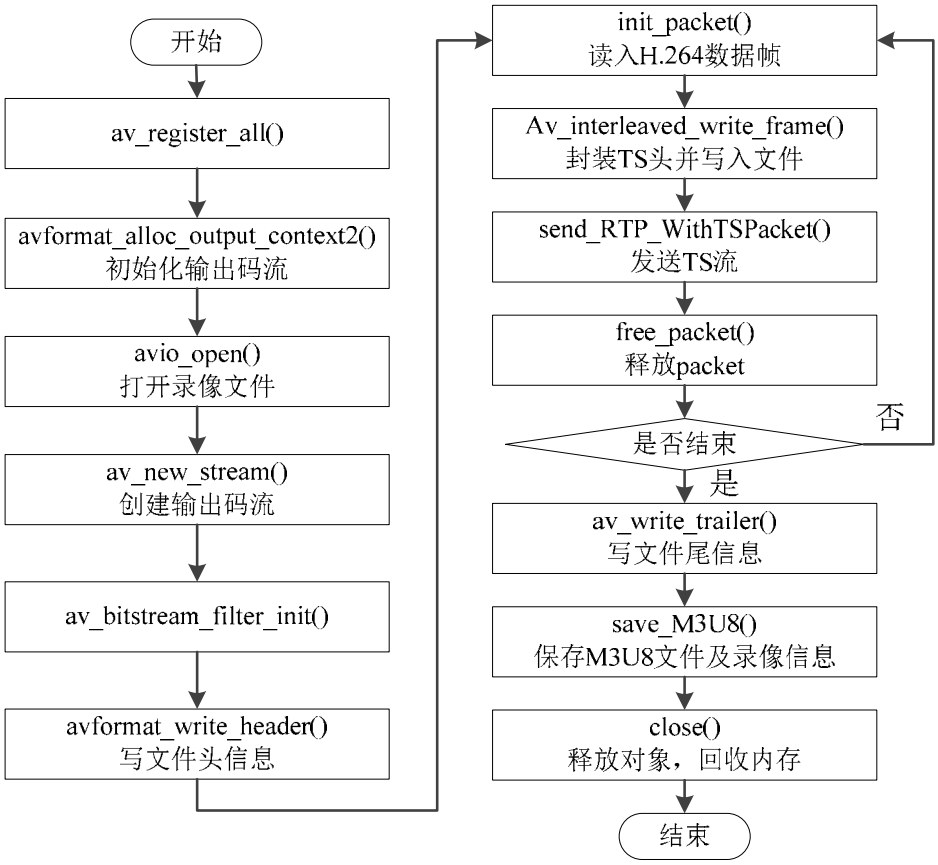


图 4-14　ES 流封装成 TS 流

4.3　基于 RTP 实现视频传输

jrtplib 开源库是广泛被应用于发送 RTP 包的工具库[50]，本项目工程中需要引入该库。创建 static library 工程，将 jrtplib 库的源文件导入工程修改编译配置，编译得到 jrtplib 库的静态库文件，导入项目工程即可。

利用 jrtplib 实现发送 RTP 包数据分为以下几个步骤：

① 初始化。

② RTPSession 对象设置：包括指定发送和接收端口端口，设置负载类型为 TS 数据流，通过 setSsrc(ssrc)函数设置同步源，设置时间戳单元。

③ 数据发送：需指定接收方的 IP 地址和端口号，在 jrtplib 中我们可以通过成员函数 AddDestination()、DeleteDestination()和 ClearDestinations()实现。目标指定后，即可调用 SendPacket()函数实现数据的发送。

RFC3984 协议定义了三种 RTP 封包模式，分别为：单一包模式、分片包模式和聚包模式。

1) 单一包模式

单一包封装主要针对长度不超过 1460 字节，当一帧的 H.264 数据打包完后的 TS 流数据小于或等于 6*188 字节时，采用单一包封装 RTP 包，节省带宽，直接装到 RTP 的载荷部分，如图 4-15 所示。

图 4-15 单一包模式

2) 分片包模式

在实际编码中，对于运动图像进行 H.264 编码，输出的 NAL 单元再进行 TS 封装后的 TS 流字节长度往往会超过 MTU（最大传输单元）值的限制。一般需要将 TS 码流进行分片发送，参照 RCF3984 协议规定，分片模式分为 FU_A 和 FU_B 两种，本软件采用 FU_A 方式，分片头结构如图 4-16 所示。

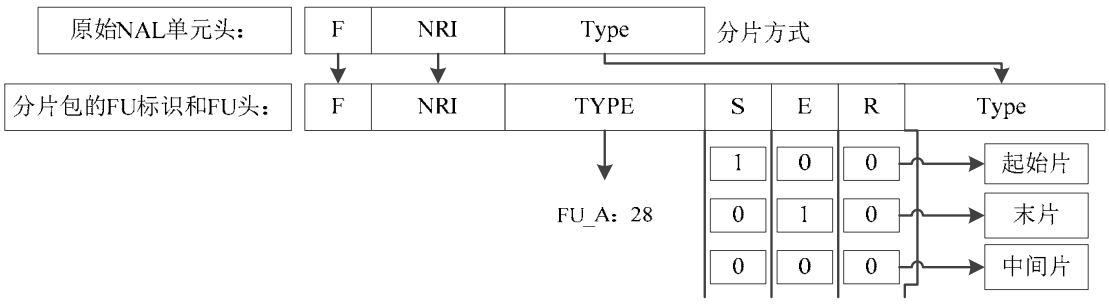


图 4-16 FU_A 分片头结构

3) 聚包模式

聚包模式是指当 NAL 单元长度特别小时，把多个 NAL 单元封装在一个 RTP 包中，可以提高带宽利用率。但在图像编码应用更需要实时性，聚包模式会增加网络延时和丢包现象。

因此，本软件选择了单一包与分片包两种模式封装并发送 TS 流数据，RTP 封装过程如图 4-17 所示。

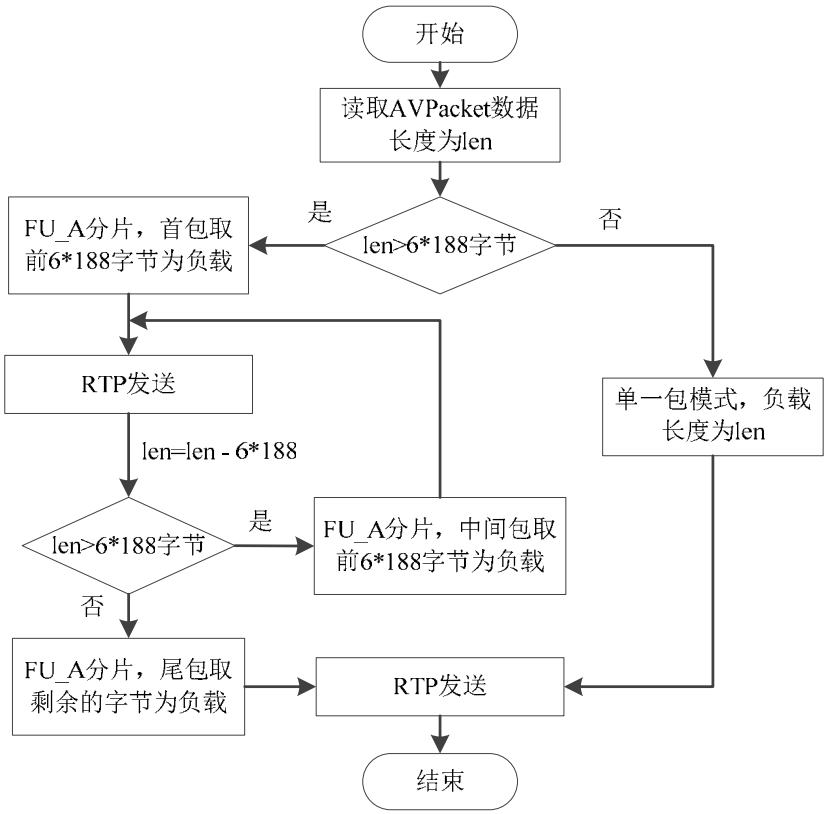


图 4-17　RTP 封装 TS 流数据流程

在本软件中的 RTSP 消息处理模块是将流媒体部分 Source 和 Sink 裁剪后只留下 RTSP 部分的 Live555 库移植到 iOS 工程项目中进行二次开发的，这部分的工作原理和流媒体的 RTSP 服务模块基本一样，所以 RTSP 消息处理模块将在下章内容中进行详细介绍。

4.4　视频采集过程的实现

针对 iOS 系统架构，通过调用 iOS 提供的视频 API 获取来自摄像头的视频流，并调用上述介绍技术，对视频流进行 H.264 编码压缩、TS 流封装及 RTP 打包传输，实现基于 iOS

手机的视频采集软件，实现过程如图 4-18 所示。

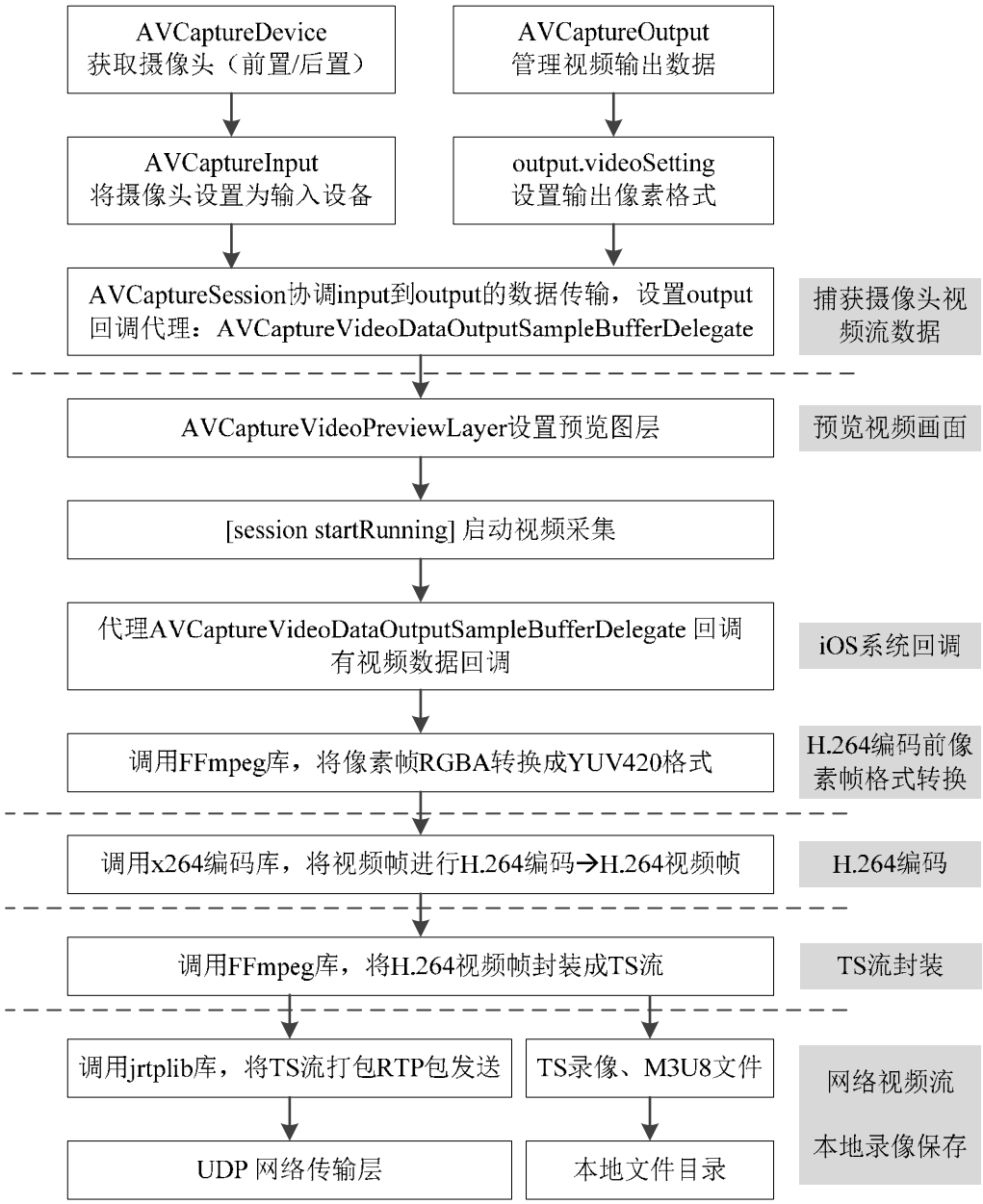


图 4-18　iOS 软件视频采集实现过程

软件实现时需要先将 AVFoundation.framework 系统库包含到工程项目中，其中 AVCaptureDevice 对象是对一个物理摄像头的抽象，它的所有属性就是摄像头的属性，需要用它来获取并设置摄像头 ;AVCaptureSession 对象是 AVCaptureInput 和 AVCaptureOutput 之间的桥梁，可以控制图像帧分辨率；AVCaptureVideoPreviewLayer 可以让用户清楚对视频图像进行预览；根据系统回调函数捕获 RGBA 格式的图像帧数据，并调用 FFmpeg 库将其格式转为 YUV420 格式，然后调用改进了 UMHexagonS 运动估计算法的 x264 库对

YUV420 格式图像帧进行 H.264 编码；最后将 H.264 视频帧进行 TS 流封装并 RTP 打包发送和录像保存。

在实际应用中，用户往往需要对直播后的视频进行回放或上传第三方的视频网站供点播观看的，而录像本地保存就很好的满足了这个需求。本地播放录像采用的是 HLS 技术，HLS 技术是由苹果公司提出的基于 HTTP 的流媒体传输协议，只需在手机本地搭建一个轻量级的 web 服务 HTTPServer（搭建过程很简单，不再阐述），支持静态文件访问即可，然后调用系统内置播放器控制类 MPMoviePlayerController 播放本地的 M3U8 文件即可，内置播放器会根据 M3U8 文件的索引文件，下载缓存 TS 视频流进行播放，虽然不是严格意义上的边下边播的方式播放流文件，但达到了基本实时播放的效果，同时大大提高解码播放效率，节约 CPU 资源。这种方式也可以扩展应用到后续的服务器视频点播上，选用 TS 格式进行录像保存适应了业务的发展需求。

4.5 iOS 软件界面

iOS 视频采集软件的主要功能是完成用户交互工作，将用户的操作事件传递给逻辑业务层，让每个底层的业务模块能受用户的指示调度，并更新相应的显示界面，给用户直观的体验。

视频采集软件是 MAC 下的 Xcode 集成开发工具下进行编译的，工程主要由三个相互独立的视图控制器（UIViewController）和一个标签控制器（UITabBarController）组成。首页控制器主要负责管理录像文件，从 iOS 沙盒机制中的 NSDocumentDirectory 目录下获取录像列表，通过 UITableView 组件以列表的形式展示到主界面，用户可以通过左滑某一录像列显示删除按钮，点击删除，则将相应的 TS 录像和 M3U8 文件从 NSDocumentDirectory 目录中删除；点击录像列表时，会先去启动 HTTPserver 本地服务，然后新分配一个 UIWindow，利用 MPMoviePlayerController 进行播放本地 HTTPserver 服务的 M3U8 索引文件，使录像回放和 UIApplication 的 UIWindow 相互独立。直播控制器负责视频直播，主要包括直播的开启、关闭及保存录像，开启直播时控制器会去捕获摄像头、初始化 x264 编码器、准备 FFmpeg 库封装 TS 流及 RTP 打包发送所需内存空间；而关闭直播时需要回收编码过程中所占用的内存及关闭编码器；并且将相应的 TS 录像文件保存至 NSDocumentDirectory 目录中。配置控制器负责配置注册密码、流媒体服务器 IP 和端口、及展示软件本身的一些信息。软件 UI 原型设计如图 4-18 所示，各个视图控制器之间的结

构关系如图 4-19 所示。

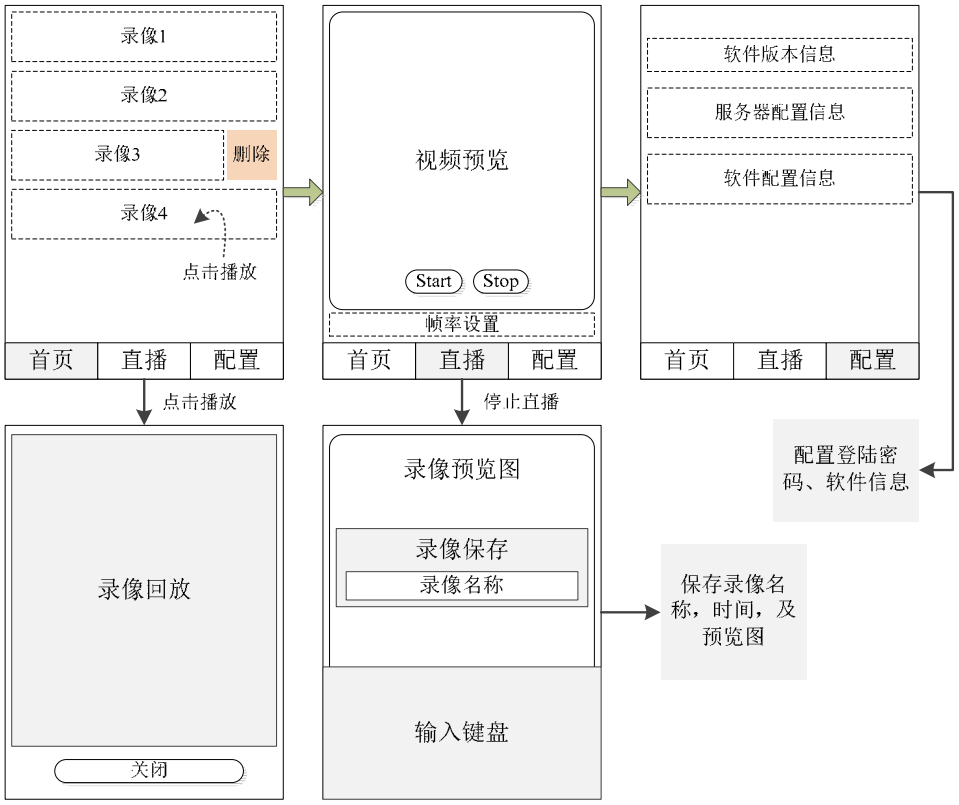


图 4-18　UI 原型设计

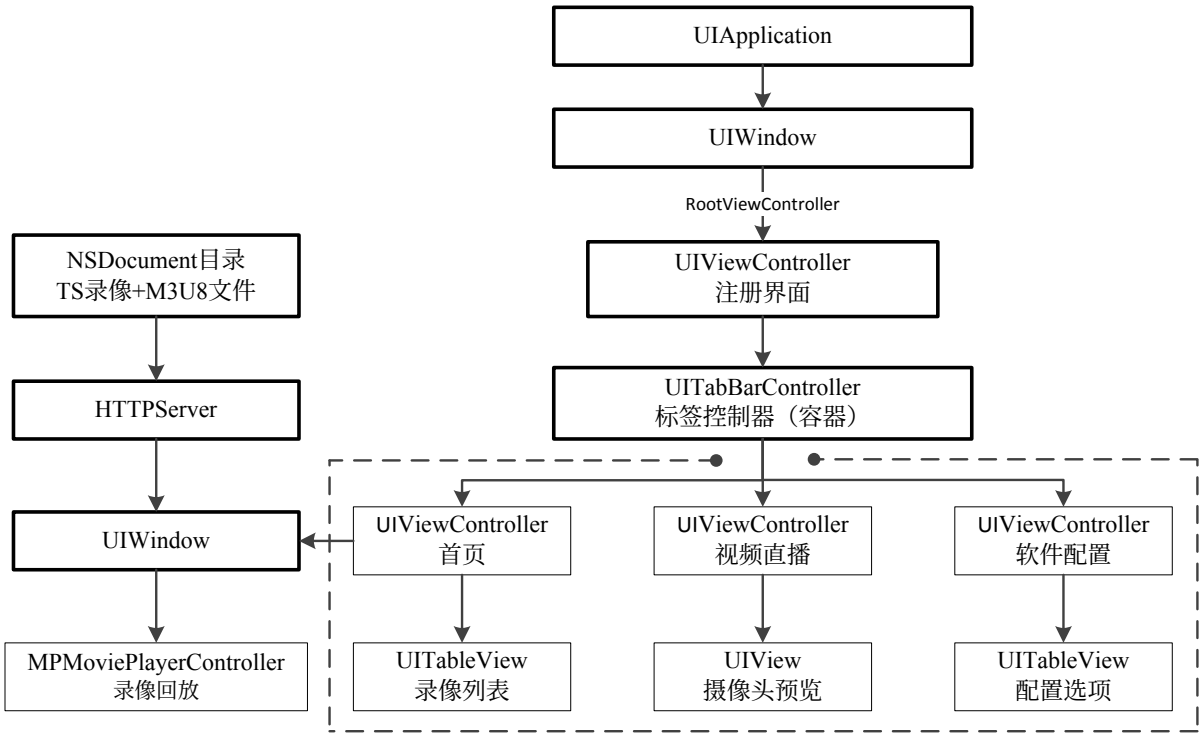


图 4-19　视图控制器结构图

4.6 本章小结

本章主要针对移动视频采集端软件的实现进行了详细的介绍。首先对 H.264 的编码原理进行了简要概述；并分析了运动估计 UMHexagonS 算法对 H.264 编码的影响，改进 UMHexagonS 算法以优化 x264 编码库的编码效率；再介绍了如何调用改进后的 x264 编码库进行 H.264 编码和利用 FFmpeg 库将 H.264 数据封装成 TS 流；再是将 TS 流打包到 RTP 包进行发送；然后调用 iOS 系统摄像头采集视频流，利用 x264 视频编码库和 FFmpeg 库封装技术，实现视频流数据的采集和传输；最后介绍了 iOS 采集端软件的原型设计及实现。

第 5 章　流媒体服务器的实现

5.1　引言

流媒体服务端是基于 Live555 开源协议栈进行二次开发的，实现满足本系统业务需求的流媒体服务器是本章节的重点内容。如图 3-5 流媒体服务器框架图所示，流媒体服务器根据层级划分主要分为三大块：RTSP 服务、媒体管理、及流媒体转发服务。以下将对这三大功能的实现进行详细的介绍。

首先从 Live555 开源项目官方网站 http://www.live555.com 下载源码，它本身是用 C++语言进行开发编译的，具有很好的跨平台性，能完美兼容 Windows、Unix、Mac OS 等平台，本软件选择在 Windows 平台下进行开发，使用 VS2010 作为开发编译环境，分别编译生成静态库文件 BasicUsageEnvironment.lib、groupsock.lib、UsageEnvironment.lib、liveMedia.lib，然后添加依赖项到自己新建的工程项目中，即可以包含引用流媒体栈中的相关库文件了。

5.2　RTSP 服务的设计与实现

5.2.1　RTSP 信令交互设计

本系统采用 RTSP 协议作为控制消息传输协议，实现 RTSP 控制消息传输是 RTSP 信令控制平台的关键。采用 C/S 模式，服务端不仅时刻监听着播放器端发送过来的 RTSP 消息，也可以作为 RTSP 客户端向移动视频采集端发送 RTSP 控制消息，移动视频采集端也可以作为服务端接收来自服务器发送而来的 RTSP 消息。在这里我们先来了解下 RTSP 几个常见必须实现的消息类型及其作用，以下为每个方法的名称、交互过程、作用及实现要求定义如下表 5-1（C 表示 RTSP 客户端，S 表示 RTSP 服务端）：

RTSP 协议负责完成流媒体视频的请求和应答过程，定义特定的控制消息为流媒体的发送和接收做好准备。根据上述的定义控制消息及自定义控制消息，本系统的 RTSP[51]交互会话过程主要如下：

① 移动视频采集端软件向 RTSP 服务端发送 REGISTER+摘要认证请求，成为在线的视频直播采集前端设备。

② RTSP 播放器端可以在任意时刻向 RTSP 服务端发起 OPTION 请求，目的是为了得到 RTSP 服务端能提供的方法，RTSP 服务端会向 RTSP 播放器端回应有哪些方法，如 OPTION、DESCRIBE、SETUP、PLAY、PAUSE、TEARDOWN 等方法。

③ RTSP 播放器端向 RTSP 服务端发起 DESCRIBE 请求，RTSP 服务端将会对播放器端进行必要的用户摘要认证，如果播放器端通过了必要的用户认证，服务端将返回给播放器端 SDP（会话描述信息），包括所请求的媒体名（Video 或 Audio）、媒体协议（RTP）、媒体封装格式等；如果用户认证失败，则返回失败信息并结束请求。

④ RTSP 播放器端向 RTSP 服务端发送 SETUP 请求时，表明播放器端希望和服务端建立会话，要求服务端为会话设置相应的属性，不仅包含播放器端到服务端的会话，还包含服务端到视频采集端的会话。会话内容主要包括协商后媒体传输的方式（单播或组播）、流媒体发送端的 ip 地址和端口号、流媒体接收端的 ip 地址和端口号、及会话的唯一标识号 Session 等。

⑤ RTSP 服务端回应的 100 Try 表示告诉播放器端正在请求视频采集前端设备，请耐心等待。

⑥ PLAY 请求为视频播放请求，RTSP 服务端通知流媒体转发服务将来自视频采集前端设备的视频数据转发到相应的 Session 对应的播放端去进行直播，并返回流媒体传输信息，如 RTP 包信息，序列号以及时间戳等。

⑦ 播放器端向 RTSP 服务端发送 TEARDOWN 请求，请求结束直播，服务端将会结束对应的 Session 会话，并释放会话中占用的所有资源，为了能处理更多其他请求。

表 5-1 RTSP 交互过程方法说明

方法名称	交互过程	作用	实现要求
OPTION	C->S:OPTION Request	询问 S 有哪些方法可用	必须实现
	S->C:OPTION Response	S 回应消息中包括提供的所有可用的方法	
DESCRIBE	C->S:DESCRIBE Request	要求得到 S 提供的流媒体初始化信息	必须实现
	S->C:DESCRIBE Response	S 回应流媒体初始化信息，主要是 sdp	
SETUP	C->S:SETUP Request	设置会话的属性，以及传输模式，提醒 S 建立会话	必须实现
	S->C:SETUP Response	S 建立会话，返回会话标识符，以及会话相关信息	
100 TRY	S->C:100 Try	告诉请求客户端，正在尝试建立链路	选择实现

方法名称	交互过程	作用	实现要求
PLAY	C->S:PLAY Request	C 请求播放	必须实现
	S->C:PLAY Response	S 回应该请求的信息	
S->C：发送流媒体数据			
TEARDOWN	C->S:TEARDOWN Request	C 请求关闭会话	必须实现
	S->C:TEARDOWN Response	S 回应该请求	

5.2.2 RTSP 服务的实现

本系统流媒体服务器是基于 Live555 协议栈进行二次开发的，根据 3.3 节中介绍的流媒体服务端框架设计和 Live555 本身的设计框架，RTSPServer 类将承担完成 RTSP 服务的主要角色，以下将详细介绍如何 RTSPServer 类完成 RTSP 服务，实现处理来自播放器端和移动视频采集前端的 RTSP 请求的会话。

1) 任务调度系统

首先在搭建 RTSP 服务之前，要先创建两个基础对象：TaskScheduler 和 UsageEnvironment 的派生类对象，TaskScheduler 类和 UsageEnvironment 类都是虚基类。TaskScheduler 类定义了一个任务调度器，它是整个程序运行的发动机，也是 Live555 的任务调度中心，主要负责任务的调度和执行；而 UsageEnvironment 类定义了整个系统的运行环境，主要负责控制台错误的输入和输出。他们共同构建 Live555 流媒体架构的灵魂。通过调用如下方法进行创建：

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

在 TaskScheduler 任务调度系统中，任务按应用场景的不同被分为三种不同的任务：Socket Handler、Event Handler 和 Delay Task。TaskScheduler 任务调度系统维护着三个任务执行队列，分别是用于存放网络 Socket Handler 处理事件的 fHandlers 单链表队列、由于存放 Event Handler 处理事件的 fTriggeredEventHandlers 数组、及用于存放延时任务 Delay Task 处理事件的 fDelayQueue 双向链表队列。Socket Handler 和 Event Handler 一直存在于队列中，而 Delay Task 延时任务只会在当前延时任务事件的延时时间为零时才会被执行一次，执行完毕则从队列中删除。Socket Handler 主要用于处理来自网络的读写事件，当监听到某个网络 Socket 对象有可读、可写或出错的事件发生时，则会被调用处理网络事件；Event

Handler 可以被用于处理一些特殊的功能，被添加到 TaskScheduler 任务调度系统的 fTriggeredEventHandlers 数组（专门用于存储 Event Handler）中，采用 32 位的位运算进行管理；Delay Task 则是向 TaskScheduler 任务调度系统中添加的延时处理事件，让整个任务调度系统更加灵活。三种任务的调用接口定义如表 5-2 所示。

表 5-2　TaskScheduler 三种任务函数

任务类型	接口定义
Socket Handler	typedef void BackgroundHandlerProc(void* clientData, int mask)
Event Handler	typedef void TaskFunc(void* clientData)
Delay Task	typedef void TaskFunc(void* clientData)

向 TaskScheduler 任务调度系统中添加上述三种任务事件, 分别要调用三种不同的添加接口 API，各个接口的说明如下。

① 添加 Socket Handler 处理事件的添加接口原型：

void setBackgroundHandling(int socketNum, int conditionSet ,

BackgroundHandlerProc* handlerProc, void* clientData)

参数说明：

SocketNum：建立 TCP 连接时 socket()返回的 socket 对象，所有的 RTSP 协议底层数据传输都是通过这个 socket 对象进行通信的。

conditionSet：标识 socket 对象的处理状态，select 前 socket 时，判断 socket 是否满足 conditionSet 条件，如可读、可写还是出错，如满足则回调处理函数。

handlerProc：检测到 socket 对象符合 conditionSet 状态时的回调处理函数。

ClientData：void 指针类型，指向 handlerProc 的回调对象。

② 添加 Event Handler 处理事件的添加接口原型：

EventTriggerId createEventTrigger(TaskFunc* eventHandlerProc)；

参数说明：

eventHandlerProc：事件的回调处理函数。

③ 添加 Delay Task 处理事件的添加接口原型：

TaskToken scheduleDelayedTask(int64_t microseconds, TaskFunc* proc,void* clientData)；

参数说明：

microseconds：表示 Task 延迟等待的微秒（百万分之一秒）数。

proc：当在延时队列中检测到延时等待时间为 0 时，执行延时事件处理函数，并且处理完毕之后从延时队列中删除。

clientData：处理延时事件时传入的对象。

TaskScheduler 任务调度系统的工作流程大致如图 5-2 所示。

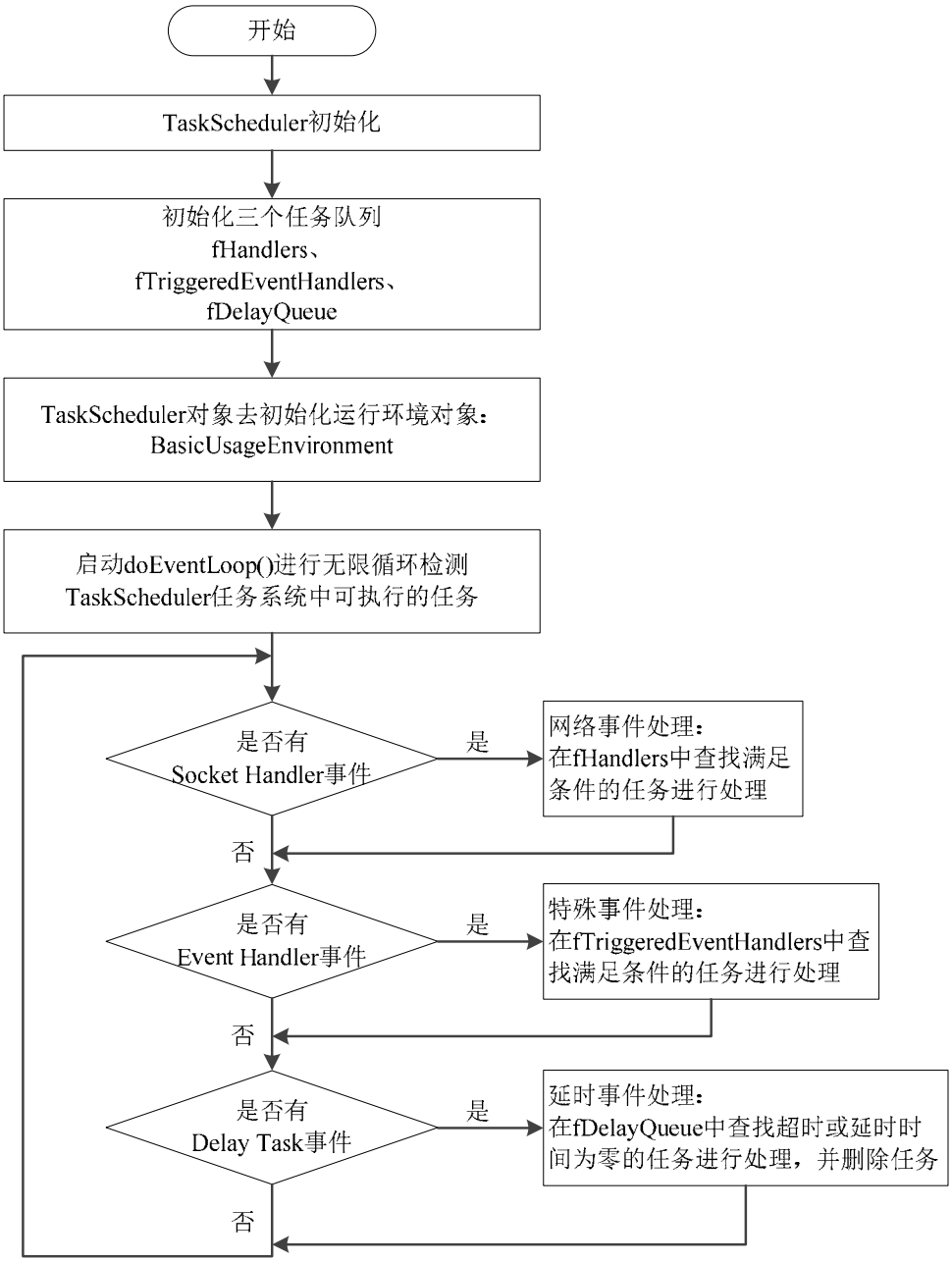


图 5-2　TaskScheduler 任务调度系统调度流程

2) RTSP 服务的实现

TaskScheduler 任务调度系统任务调度流程非常简单清晰，这种设计思想将不同的业务逻辑最直接地进行解耦，只需要将需要处理的业务逻辑加入到这个调度引擎中，就能高效的运行。构建 RTSP 服务只需要利用好 Socket Handler 事件队列就可以很好的处理来自网络的 socket 请求, 实现本系统要求的播放器端到服务端及视频数据采集端到服务端的 RTSP 交互。下面将介绍如何在 TaskScheduler 任务调度系统之上构建 RTSP 服务。

由于 RTSP 协议传输的都是文本型数据量较小的字符串格式，所以选择 RTSP 协议建立在 TCP/IP 之上进行传输，保障 RTSP 消息的可靠性和安全性。RTSP 服务需要实现监听播放器端和视频数据采集端的 RTSP 连接，利用 TaskScheduler 任务调度机制，需要先创建一个 Socket Handler 添加到任务调度系统中来处理 RTSP 请求，但是本系统还要对来自网络的 RTSP 请求进行必要的用户认证，所以还需要一个用户数据认证管理中心来初始化这个 Socket Handler，以下是创建这个 Socket Handler 的过程。

初始化用户数据认证管理中心：

```
UserAuthenticationDatabase* authDB
                                    = new UserAuthenticationDatabase("Database Name");
authDB->addUserRecord("username", "passwd123");
```

本系统并没有采用传统的大型或轻量级数据库（如 oracle 或 MySQL 等）进行用户管理，数据库的管理和设计并不是本系统的研究重点，在本章中不做介绍。用户数据认证管理中心中采用本地的一个哈希表 fTable 来存储用户数据，哈希表的先天优势就是查找速度快，缺点则是会浪费一些内存空间，但为了性能上的提升适当牺牲一些内存也是可以接受的。

创建 RTSPServer 对象：

```
RTSPServer::createNew(*env, 8554, authDB): fAuthDB(authDB){
    int ourSocket = setUpOurSocket(env, ourPort);
    env.taskScheduler().turnOnBackgroundReadHandling( fRTSPServerSocket,
    (TaskScheduler::BackgroundHandlerProc*)&incomingConnectionHandlerRTSP, this);
                                                        //加入 Socket Handler
}
```

创建 RTSPServer 对象时，主要工作：通过 setUpOurSocket()调用 setupStreamSocket()函数来创建一个 Socket 连接；为 Socket 连接设置发送缓存区大小（默认为 50Kb）；调用系

统函数 listen()函数开始监听这个端口；为这个 RTSPServer 对象设定最大的并发连接数 LISTEN_MAX_SIZE，默认为 20，如果服务设备配置比较好，性能高的话可以设置更大的并发数；把连接处理回调句柄 incomingConnectionHandler 和 Socket 句柄 serverSocket 传给 taskScheduler 任务调度系统进行关联，并将这个 Socket Handler 处理事件添加到 fHandlers 中。这样就可以灵活的将 RTSP 服务添加主程序的任务调度系统中，等待 RTSP 客户端的连接。

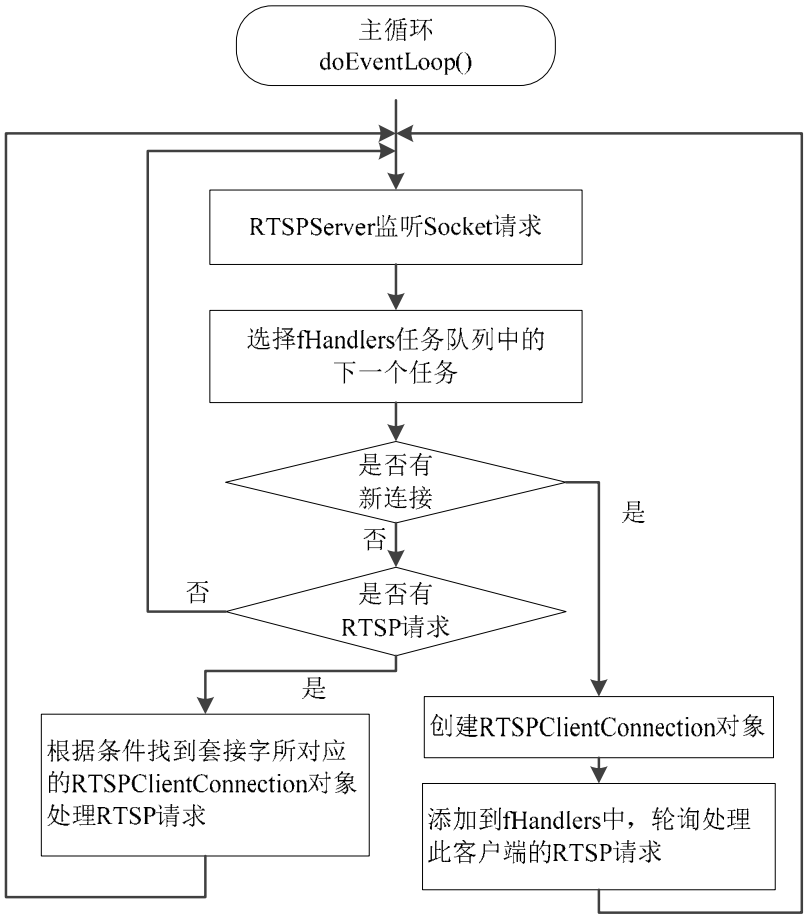


图 5-3　RTSP 请求处理流程图

主循环 doEventLoop()在不断的轮询任务任务队列中是否有事件发生，当 RTSP 客户端向 RTSP 服务端发起 Socket 连接时，主循环就会检测到来自客户端的连接请求，RTSPServer 对象会为这个新套接字创建一个新的 RTSPClientConnection 对象来保存这个套接字的相关属性，以便可以用这个套接字与这个 RTSP 客户端通讯。这个 RTSPClientConnection 对象管理着一个 RTSP 客户端所有的 RTSP 会话信息，包括所请求的流媒体 RTP 信息，也就是 RTSPClientConnection 需要监听客户端发送的请求信息，必须把它的 Socket Handler 加入到

TaskScheduler 任务调度系统的任务调度队列中。RTSPServer 在主循环 doEventLoop()中的工作流程如图 5-3 所示。

为了保证 RTSP 服务端的并发性和公平性，在从 fHandlers 任务队列中选择下一个有可读、可写或出错的任务进行执行时，由于 fHandlers 是一个单链表，所以要做一些特殊的处理来保证每个 RTSP 客户端都能公平的享受服务器资源，对选择下一个任务的方式进行简单介绍：有一个 fLastHandledSocketNum 变量记录着上次访问的 Handler，下一轮询时不是从头开始顺序检测 Socket 事件，而是从上次访问的 Handler 的下一个开始检测，这样保证了各个 RTSP 客户端的请求被处理的机会是平等的，处理过程如图 5-4 所示。

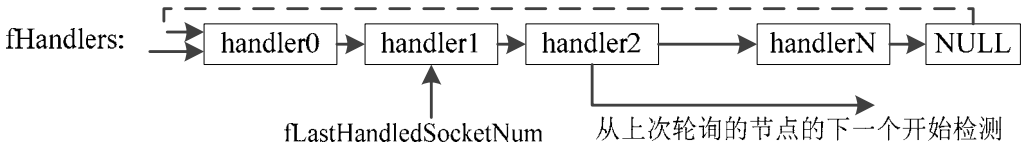


图 5-4　任务队列中选择任务

RTSPClientConnection 对象收到 RTSP 客户端的不同的消息要进行不同处理，它先会将收到的 RTSP 消息进行解析，得到每条消息的方法名称、请求头域、及请求体。当收到 DESCRIBE 消息时，需要响应客户端 RTSP 地址、验证用户名和密码、及根据请求的 streamName 查找出 ServerMediaSession 等；当收到 SETUP 消息时，需要为这个客户端建立流媒体的会话信息；当收到 PLAY 消息时，将启动流媒体转发，Sink 会从 Source 源中取数据开始转发；而收到 TEARDOWN 消息时，需要关闭流媒体会话信息，停止转发等工作，在后面将会进行详细介绍流媒体转发部分。

5.3　流媒体转发服务模块的实现

从图 3-5 流媒体服务架构可知，RTSP 服务调度流媒体转发模块，将客户端请求的媒体数据流从相应的流媒体 Source 源转发到流媒体 Sink 消费，实现流媒体转发服务。下面本章节将详细介绍流媒体转发服务模块是如何响应 RTSP 服务平台调度实现流媒体流的转发及流处理、过滤工作的。

5.3.1　流媒体调度

流媒体管理模块主要负责对不同类型的流媒体进行统一管理，包括来自网络的不同格

式流媒体或本地的媒体文件，按照不同的流媒体类型选择不一样的 Source 源，转发给相应的 Sink 去消费，形成一条通道，即在这个流媒体管理模块内可以将一条流媒体转发链路看成是一条通道，下面来看看 RTSP 服务平台是如何根据不一样的通道名称来调度不一样的流媒体转发工作。

Live555 开源协议栈中提供了一个可扩展的框架结构，ServerMediaSession 对象就代表着一个实际的媒体，一个实际的媒体可以同时包含视频、音频、字幕、甚至广告信息等视频附加信息, ServerMediaSession 根据实际需求设计成一个可以任意扩展媒体信息的管理容器，这也是本章节要讲的如何利用 ServerMediaSession 来管理视频转发通道。

创建 ServerMediaSession 对象：

ServerMediaSession::createNew(*env, streamName, streamName, descriptionString);

需要传入的参数为：运行环境对象 env，媒体名字，及媒体描述，创建时会对 ServerMediaSession 对象进行必要的初始化工作，分配一些内存空间来管理媒体通道。媒体通道主要是要独立负责实际视频流、音频流、字幕或其他媒体数据流的产生和消费的，Live555 中提供的一个 ServerMediaSubsession 类代表媒体中一个数据流，可以继承 ServerMediaSubsession 类来自定义不一样的数据流，完美的添加到 ServerMediaSession 对象中进行统一管理，这种解耦的设计方式可以让不同的业务数据流可以独立开发而不会相互干扰，增加了管理系统的扩展性。本系统中的媒体通道的设计思想是一个媒体通道对应媒体中一个数据流，并负责这个数据流的所有工作，包括流媒体数据流的产生（如来自文件读取、从网络上获取 RTP 数据包等）、流媒体数据流的过滤处理（如删除 TS 流中某一节目的数据流、数据流格式转换、校正系统时钟等）、流媒体数据流的消费（如打包 RTP 包转发、将数据流写入本地文件进行保存等）及相应的 RTCP 网络拥塞控制。

RTSPServer 管理着 ServerMediaSession，ServerMediaSession 管理着 ServerMediaSubsession，而 ServerMediaSubsession 管理着单个数据流，在 RTSPServer 收到来自客户端的 SETUP 消息时，它会和客户端协商数据流通信的细节，如通信协议、地址等。数据流建立流程图如 5-5 所示。

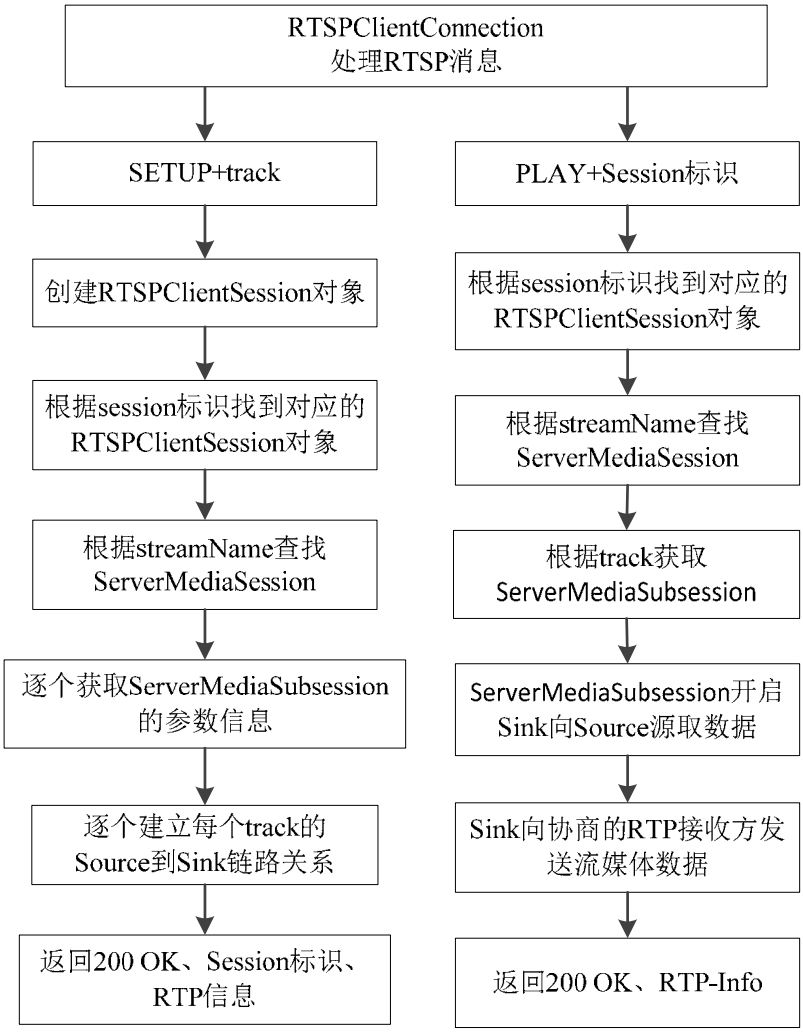


图 5-5　数据流建立过程

1）RTSPServer 接收到 SETUP 消息之后，它会回调服务端为这个客户端分配的 RTSPClientSession 去处理这个 SETUP 请求，回调处理方法 RTSPServer::RTSPCIientSession::handleCmd_SETUP()中会为每个 ServerMediaSubsession 分配一个 struct streamstate，用于控制当前数据流及记录相应的状态；然后分析 RTSP 请求字符串中的传输要求，获取客户端所要求的传输协议、客户端 RTP 端口号、客户端地址等；根据这些传输信息，在流媒体转发端建立起 RTPsocket，进行 RTP 连接，最后为这个 streamstate 返回一个 streamToken，表示数据流已经建立起来。streamToken 的建立过程存在于函数 ServerMediaSubsession 声明的虚函数 getStreamParameters()中，在这个函数中需要为为数据流创建 Source、创建 Groupsock、创建 Sink，最后回应字符串。

2）RTSPServer 接收到 PLAY 消息之后，RTSPCIientSession 会根据通道号 track 查找到相应的 ServerMediaSubsession 对象，开始启动 RTP 数据流传输，启动代码如下：

```
fStreamStates[i].subsession->startStream(fOurSessionId,
                                         fStreamStates[i].streamToken,
                                         (TaskFunc*)noteClientLiveness, this,
                                         rtpSeqNum, rtpTimestamp,
                    RTSPServer::RTSPClientConnection::handleAlternativeRequestByte,
                                         ourClientConnection);
```

其中其实是通过 StreamState::startPlaying()函数将客户端的 IP 地址和 RTP 端口传给 rtpGroupsock，开始向客户端发送 RTP 数据流，并返回 RTP 相关信息字符串给客户端。

5.3.2 建立流媒体转发

1) 流媒体发送模块

在整个流媒体转发过程中真正控制着流媒体的转发或停止的 RTSP 消息主要就是 SETUP、PLAY、TEARDOWN 消息。SETUP 消息负责告诉流媒体服务端创建好流媒体转发所需的数据流链路，包括 RTP 信息的协商、Groupsock 的初始化、Source 和 Sink 的初始化等工作；PLAY 消息负责通知流媒体服务端一切就绪了，可以开始转发流媒体数据；TEARDOWN 消息则是告诉流媒体服务端停止转发数据流，并释放数据流链路所占用的对象，回收内存。

流媒体转发数据流开始于函数 MediaSink::startPlaying()，在这个函数最后调用 continuePlaying()，然后直接调用 buildAndSendPacket()去打包 RTP 包。在 buildAndSendPacket()中主要是为 RTP 准备包头，为一些需要实际数据改变的字段预留位置，代码如下：

```
unsigned rtpHdr = 0x80000000; //预留字段位置
rtpHdr |= (fRTPPayloadType<<16);
rtpHdr |= fSeqNo; //设置 RTP 包序号，客户端根据序号排序重组 RTP 包
fOutBuf->enqueueWord(rtpHdr);
```

然后通过 MultiFramedRTPSink::packFrame()向 RTP 包中添加负载流媒体数据。packFrame()函数中首先检测上次打包是否剩下的数据，如果有数据，则先将上次打包剩下的数据放入包中；如果没有，则通过 fSource->getNextFrame()向 Source 中获取数据，并将 fSource 指向新数据所存放的位置，获取完一帧数据之后，调用 afterGettingFrame()这个回调函数返回给 Sink，其实主要是 afterGettingFrame1()在完成这个工作。之后 Sink 会去判断包是否已经

装满数据（RTP 数据包长度不能超过 IP 网络的 MTU-最大传输单元 1460 字节），如果没有装满，则又调用 packFrame()继续向 RTP 数据包中加入数据；如果 RTP 数据包已经注满数据，即打包完毕，则通过在函数 MultiFramedRTPSink::sendPacketIfNecessary()中调用 fRTPInterface.sendPacket()发送 RTP 包，并计算发送 RTP 包所需要的发送时间，把 MultiFramedRTPSink::sendNext()作为一个 Delay Task 添加到 TaskScheduler 任务调度器中，作为一个延时事件调度，等发送完当前 RTP 时才去发送下一个 RTP 包，以减小网络拥塞，保证发送速率的平滑性。在主循环中，当 sendNext()事件被调度时，又开始调度 buildAndSendPacket()去发送新的数据过程，形成一个循环，通过这样不断的循环，客户端可以源源不断的收到流媒体服务端发送来的 RTP 包。整个流媒体发送流程如下图 5-6 所示。

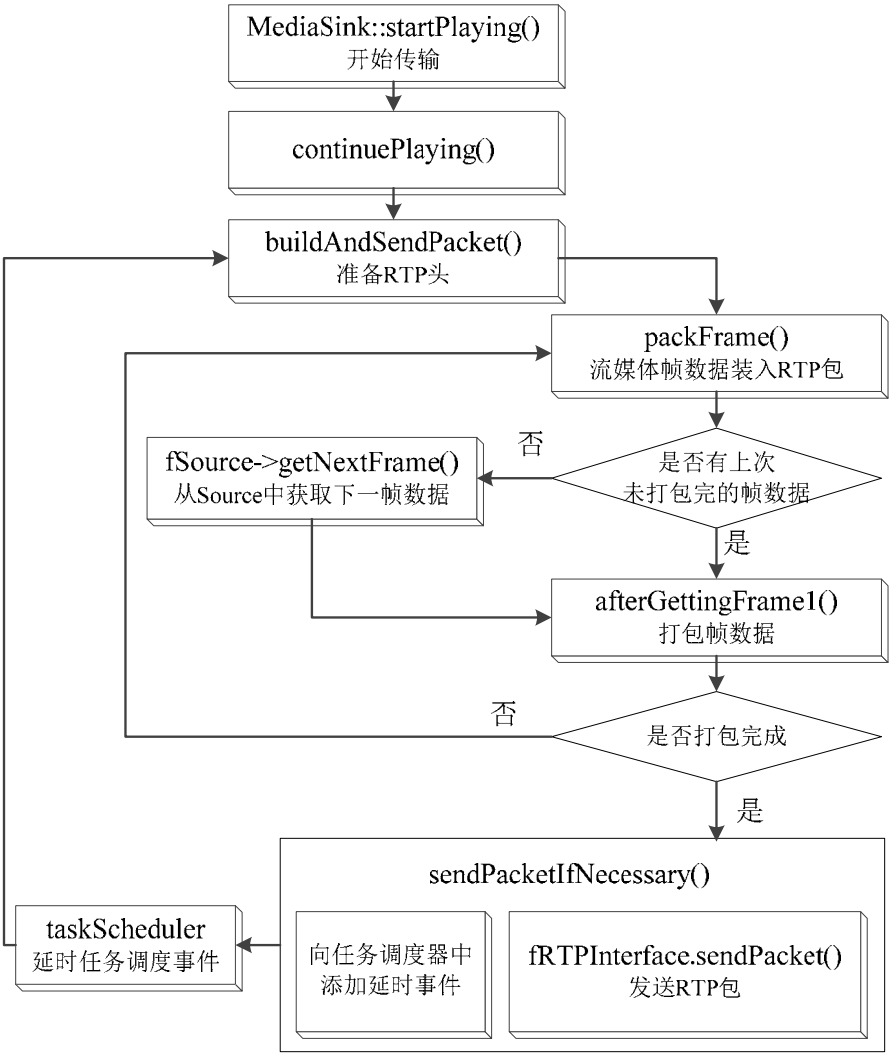


图 5-6　流媒体发送数据流程

2) 流媒体接收模块

流媒体接收模块需要为整个流媒体服务提供数据来源，也就是上述介绍的 Source 数据源，根据 Live555 开源协议栈结构，所有的 Source 需要从 FramedSource 基类继承而来。本系统中，流媒体数据是源自移动视频数据采集端向服务端发送的 RTP 数据流的，本系统选用 Live555 的 SimpleRTPSource 作为流媒体接收模块进行搭建。

在开始传输数据之后，Sink 会去 Source 中不断读取数据进行转发，Source 提供了一个统一的接口 getNextFrame()给 Sink 调用，用于读取数据流。本系统中 Source 的数据来源主要来自 RTP 网络包，在 getNextFrame()中 Source 会向和前端建立起的 RTPInterface 会话中读取完了 RTP 包，然后对 RTP 包进行解析，并可以对 RTP 负载 TS 数据流进行处理，如加入或过滤一些其他媒体信息数据，但在本系统中暂时没有对 TS 流进行处理，为业务扩展做准备。再交给 Sink 消费，也就是上面所说的流媒体发送模块，这样就实现了流媒体的调度转发。RTCPInstance 类负责收集 RTP 链路的传输情况信息，但本文并没有将收集的网络拥塞控制信息反馈给移动视频采集源端，没有直接根据根据拥塞信息去控制发送端的采集速率或码流，因为移动视频采集端的拥塞控制不应该受播放端网络情况的影响，而是由移动端自身所处的网络环境决定，所以，本文只是将拥塞控制信息收集而并未处理，将 RTCP 拥塞控制信息处理作为今后的扩展优化功能。

5.4 本章小结

本章主要针对流媒体服务器的实现进行了详细的介绍。首先根据系统架构对 RTSP 服务进行了设计；分析 Live555 开源框架的运行环境及运行原理，并利用 RTSPServer 实现 RTSP 服务；再根据 RTSP 请求，RTSP 服务平台调度相应流媒体 Sink 从 Source 源中获取流媒体数据进行发送，从而实现流媒体的转发。

第 6 章　测试、总结与展望

6.1　x264 开源库 UMHexagonS 运动估计优化测试

本实验使用的 x264 开源库版本为 x264-r2538-121396c，测试环境 visual studio 2010，测试机器为 Lenovo 笔记本，CPU 为 AMD A4 处理器，主频 1.9GHz，内存为 3G，操作系统 Windows7。根据改进型 UMHexagonS 运动估计方案，参照 C 语言源码，根据 4.1.3 节中修改方案对 x264 源码进行修改，最终编译为 x264_encoder.exe 文件。参照文献[52]的测试方法，选用 7 个具有代表性的不同运动等级的测试视频帧 CIF 序列进行测试，运动剧烈程度从慢动快，分别为 akiyo_cif、news_cif、flower_cif、foreman_cif、mobile_cif、bus_cif、football_cif，可以通过[53]提供的网址进行下载，格式为.y4m 文件，对优化前 UMHexagonS 算法（记作 x264_ME_UMH）和优化后 UMHexagonS 算法（记作 X264_ME_UMH_O）的编码帧进行如表 6-1 所示的参数指标测试，结果如表 6-2 所示。在 Windows 7 操作系统的命令行模式下，切换到 x264_encoder.exe 目录下，测试命令如下：

x264_encoder.exe --profile baseline --preset fast --merange 32 --me umh/umho --crf 26 --ssim --psnr -o <output file> <input file>

表 6-1　编码器测试指标

参数	指标说明
FPS	压缩时的帧率，用于比较压缩速度，越高越好。
Kb/s	压缩时的码率，单位为 Kb/s，越低越好（压缩率高）。
SSIM	结构相似性指标，用于比较画质，越高越好，上限为 1.0。
PNSR / △PNSR	编码后信噪比，越大越好；信噪比损失率，‘ - ’为损失，‘ + ’为改善。
crf	调节压缩画质，crf 越大画质越差，压缩比越大，取值范围（0，51）.

在无线环境下，手机端要追求画质，但也要兼顾压缩比，crf 越大画质越差，压缩比越大，本文选择 crf 为 26 进行测试；同时为了在手机端性能和资源有限的情况下能获得更快的编码效率, preset 设置编码速率为 fast 型的, 但对 PSNR 的稳定性会有一定影响；--profile

设置编码器档次 baseline；--merange 设置搜索窗口为 32*32，--me 设置运动估计搜索算法为 umh/umho；--ssim 设置打印 ssim 值；--psnr 设置打印编码之后信噪比，其他编码器参数采用默认的编码参数。测试结果如表 6-2 和 6-3 所示。

表 6-2　编码器测试结果数据

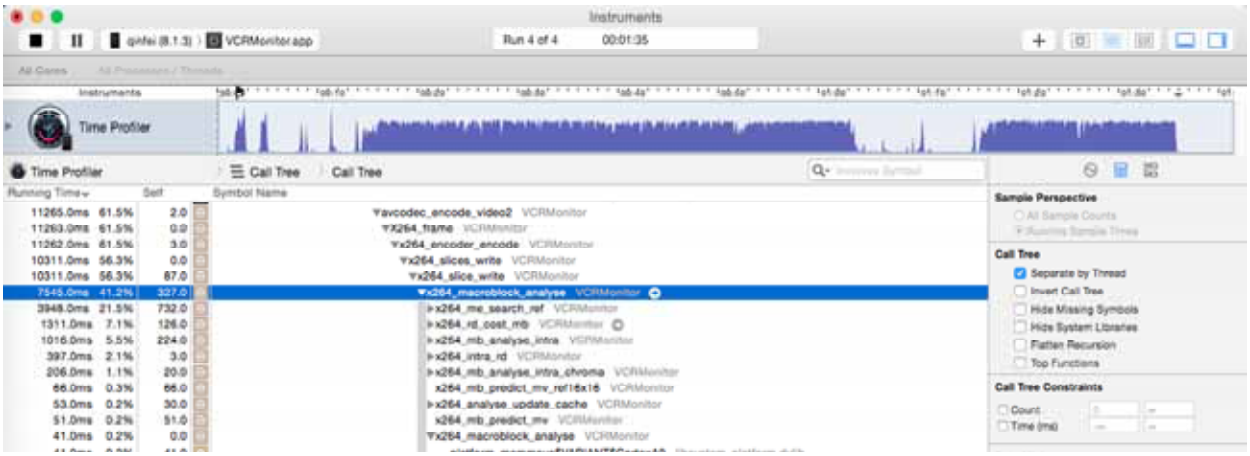
视频序列	运动估计算法	FPS	Kb/s	SSIM	PSNR-Y (db)
Akiyo	x264_ME_UMH	156.74	82.37	0.9731785	41.035
	x264_ME_UMH_O	170.43	83.98	0.9722323	40.947
News	x264_ME_UMH	104.44	170.38	0.9653111	38.383
	x264_ME_UMH_O	112.39	176.03	0.9661173	38.291
Flower	x264_ME_UMH	43.43	970.35	0.9633944	32.232
	x264_ME_UMH_O	49.74	981.71	0.9617750	32.054
Foreman	x264_ME_UMH	59.89	341.35	0.9301628	35.978
	x264_ME_UMH_O	65.52	343.90	0.9290824	35.872
Mobile	x264_ME_UMH	35.19	889.50	0.9546810	31.593
	x264_ME_UMH_O	43.87	901.81	0.9518327	31.409
Bus	x264_ME_UMH	44.35	626.30	0.9334666	33.061
	x264_ME_UMH_O	51.99	633.65	0.9305993	32.929
Football	x264_ME_UMH	39.31	850.43	0.9046360	34.407
	x264_ME_UMH_O	47.29	861.36	0.9009768	34.240

表 6-3　UMHexagonS 算法改进前后的参数变化值

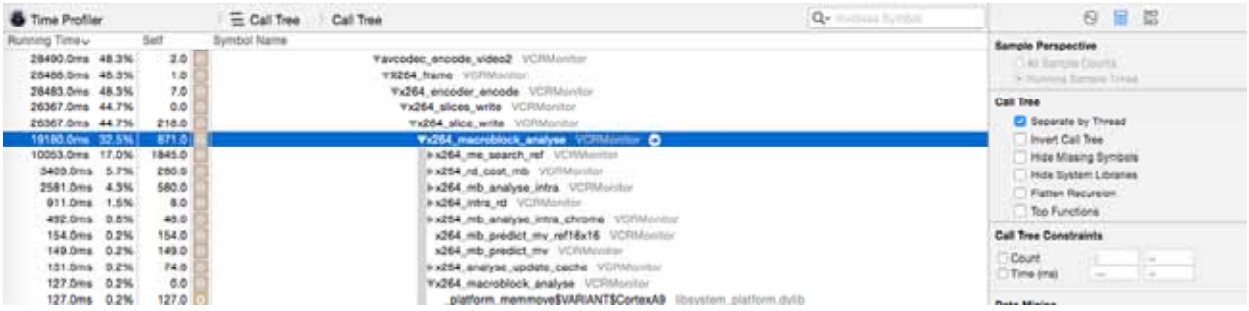
视频序列	△FPS	△Kb/s	△SSIM	△PSNR-Y(db)
Akiyo	+8.73%	+1.95%	-0.0009462	-0.088
News	+7.61%	+3.32%	+0.0000862	-0.092
Flower	+14.53%	+1.17%	-0.0016194	-0.178
Foreman	+9.40%	+0.75%	-0.0010804	-0.106
Mobile	+24.67%	+1.38%	-0.0028483	-0.184
Bus	+17.23%	+1.17%	-0.0028673	-0.132
Football	+20.30%	+1.29%	-0.0038592	-0.167
AVG (平均值)	+14.64%	+1.58%	-0.00176554	-0.135

从上述测试数据可以看出优化 UMHexagonS 算法后的 x264 编码库在压缩画质 crf=26 的情况下编码帧率大约提升了 14.64%，但压缩码率 Kb/s 和结构相似性指标几乎没有损失，压缩码率只增加了 1.58%，而结构相似性指标只降低了 0.00176554，编码后的信噪比下降只有 0.135db。但从测试数据大致可以看出，随着运动剧烈程度的加深，编码帧率越来越小，改善帧率更加明显，而结构相似性指标和信噪比下降更大，说明运动剧烈程度越高，视频编码质量下降越大，但并不影响观看质量。处理性能和资源有限的手机端牺牲了小部分图像质量而获取更高的编码效率也是值得的。

将改进型 UMHexagonS 算法的 x264 编码库通过静态编译移植到 iOS 平台，实现 iOS 平台移动视频采集软件的开发，利用 mac 下的 Instrument 工具对移动视频采集软件进行优化前后的 Time Profile 指标检测，Time Profile 指标可以清楚的看出每个函数调用时所占的时间比，测试结果如图 6-1 所示。



(1) 未改进的x264_ME_UMH编码



(2) 改进的x264_ME_UMH_O编码

图 6-1 改进前后的 x264 编码时间对比

根据改进方案，改进的地方主要包含在 x264_macroblock_analyse()中，其中 x264_me_search_ref()为运动估计的主函数，是本文改进重点。从图 6-1 测试结果可以看出

改进前的x264_macroblock_analyse()函数所占时间比为41.2%,改进后所占时间比为32.5%,运动估计的时间节约将近 10%，编码速率明显提升。

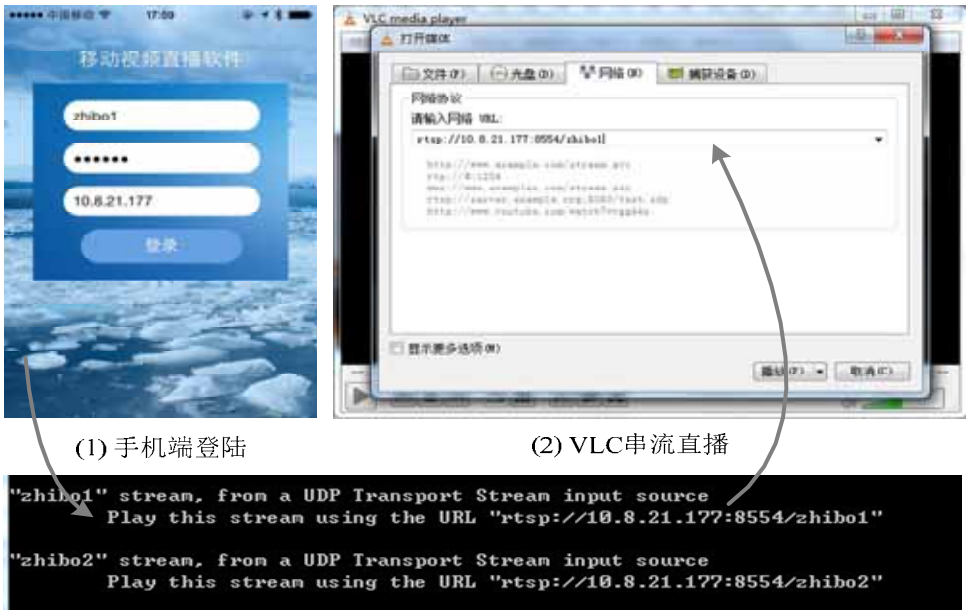
综合上述测试数据发现，改进型的 UMHexagonS 运动估计算法 x264 编码库相比于原 x264 编码库搜索速率明显变快，编码帧率约提高 14%，信噪比（PSNR）下降不到 0.15db，结构相似性（SSIM）基本没有下降，基本保持原有算法的视频质量。iOS 手机端移动视频采集软件的采集每帧视频帧的时间明显加快，改进后的 x264 编码库更适合实时性高的场合。

6.2 移动直播系统功能测试和性能分析

视频直播系统的测试方案：视频直播系统主要移动视频采集端和服务端，对于移动端软件，除了重点测试软件本身的功能模块外，软件的性能测试也是不容忽略的；而服务端处理请求的成功率、流媒体数据转发时间、及并发处理能力是衡量服务器性能的一个重要标准。具体的功能测试包括移动视频采集软件注册测试、视频直播测试、视频录像的播放和删除测试；性能分析包括移动视频采集软件内存占用量、CPU 使用率及流媒体服务系统对视频转发的延时测试。以下是对移动直播系统的各个组成部分的详细测试。

硬件配置：移动设备采用的是 iPhone 4s（iOS8.1），主频 800MHz，内存 500M，后置摄像头 800 万像素，前置摄像头 400 万像素；采样的视频格式为 CIF，帧率设置为 10 帧每秒；网络环境为电信 WIFI 网络，带宽为 4M。服务器端使用的硬件设备为联想笔记本，双核，AMD A4 处理器，主频 1.9GHz，内存 3G，操作系统为 Windows 7。以 VLC 开源播放器作为 RTSP 网络串流播放器代表，测试过程如下。

先通过真机调试，将视频采集软件安装到手机上，由于 iOS 系统不支持一般应用后台运行，所以，先要将软件启动并注册到服务器上，输入媒体名称、密码及流媒体服务器 ip，点击登录，向服务器发送 REGISTER 消息并对媒体名称和密码进行摘要验证，完成设备注册过程，如图 6-2 中(1)所示；注册成功之后，流媒体服务器根据用户的媒体名称，为用户添加媒体直播请求地址，如 rtsp://10.8.21.177:8554/zhibo1，如图 6-2 中(3)所示；通过 VLC 播放器通过打开 RTSP 网络串流的方式，请求媒体流进行实时播放，如图 6-2 中(2)所示。

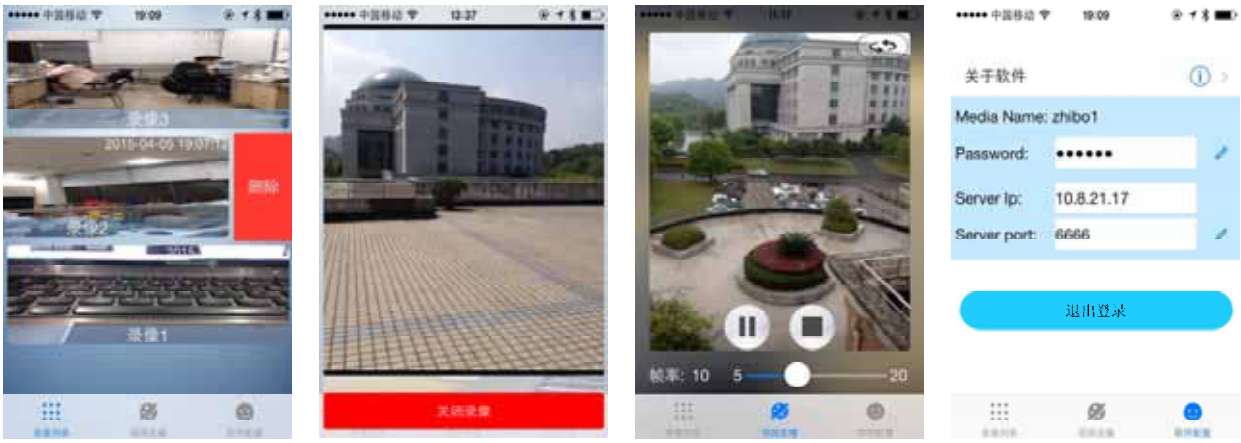


(1) 手机端登陆　　(2) VLC串流直播

(3) 流媒体服务器

图 6-2　移动视频采集软件注册

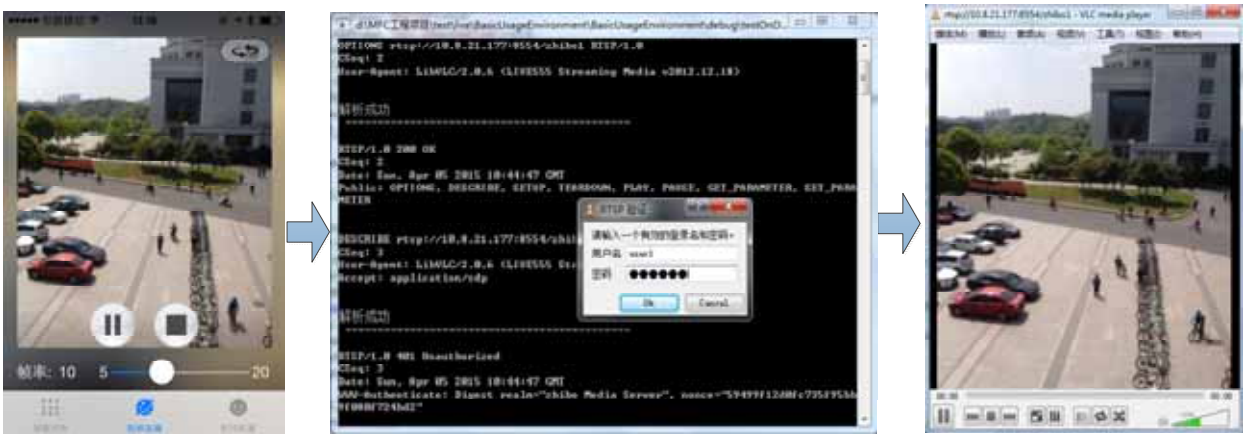
移动视频采集软件向服务器登录成功之后，进入含有三个标签页的主界面，第一个为录像列表页面，可以左滑删除录像或点击播放，如图 6-3 中(1)和(2)所示，由于调用的是系统播放器进行播放，播放的效果非常好，速率流畅，画面清晰；第二个为视频直播时的预览、控制和保存录像页面，可以随时随地开启实时视频直播，并且在直播过程中可以主动停止，暂停和切换前置或后置摄像头等操作，直播过程中同时对视频就行录像，停止直播时对录像进行保存，录像可以支持本地回放，也可以连接电脑通过 iTunes 提取录像，实时直播预览界面如图 6-3 中(3)所示；第三个为软件配置页面，可以配置软件的版本信息、服务器信息、直播信息，具有配置登录密码、服务器 ip 及端口等功能，如图 6-3 中(4)所示。



(1)　录像列表　　(2)　录像回放　　(3)　视频直播界面　　(4)　软件配置项

图 6-3　移动视频采集软件

VLC 播放器进行请求视频直播时，流媒体服务器需要先对播放器进行必要的用户验证，防止不明用户进行请求播放，如图 6-4 中(2)所示；若用户验证成功，服务器通过邀请移动视频采集端开始视频直播，直播效果如图 6-4 中(1)所示；然后将视频流转发到播放器端进行解码播放，实现移动端到播放器的实时直播，VLC 播放器直播效果如图 6-4 中(3)所示。从图中可以看出，视频清晰可辨，具有较好的视频效果。



(1) 手机视频直播预览　　(2) 播放器验证　　(3) VLC实时视频直播

图 6-4　VLC 播放器请求视频直播

利用 Instrument 工具对移动视频采集软件进行性能检测, VCRMonitor 为本移动视频采集软件的工程名，其他为系统自带软件或第三方软件，检测结果如图 6-5 所示，CPU 使用率可以从‘ %CPU ’指标看出约为 39%，CPU 占用时间比从‘ CPU Time ’指标看出约为 55：56，分配物理内存从‘ Real Memory Usage ’看出为 64.07M 等，通过上述参数可以发现 H.264 视频编码对硬件性能要求比较高，对配置较低的手机实现起来比较占用资源，但对 iPhone 手机来说还是可以很好的实现视频采集编码工作。

作为移动视频直播系统，视频流转发时延是衡量整个系统不可忽视的一个参数，测试方法如下：利用一个计时器，移动视频采集软件对计时器进行视频采集，并在 VLC 播放器端进行直播观看；由于整个测试过程有两人同时操作完成的，操作上存在着不完全同步的时延误差，主观判断操作时延抖动为±0.4s 或±0.3s 左右（‘ + ’表示手机端先截图，‘ - ’表示 VLC 播放端先截图），连续进行了 5 组测试，测试时延结果如表 6-4 所示，测试示意图如图 6-6 所示。结合测试结果显示，视频测试平均时延为 2.16s。在比较宽裕的带宽但硬件配置不高的个人电脑的测试环境下，2s 左右的时延对于一个直播系统还是可以接受的，达到了视频直播的要求。

图 6-5　移动视频采集软件性能检测

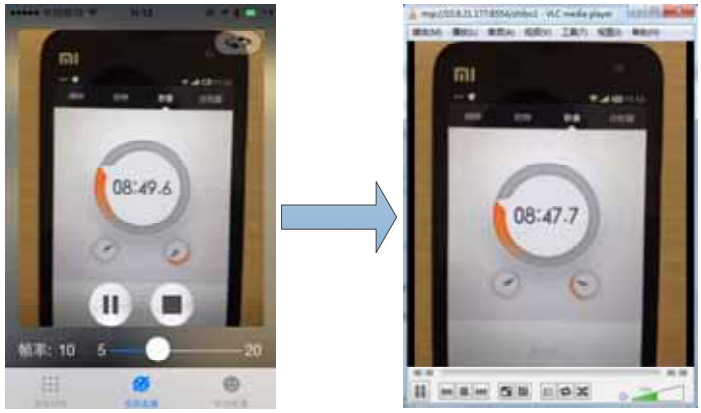


图 6-6　视频直播延时测试

表 6-4　时延测试

采集端的时间	VLC 播放的时间	操作抖动(s)	时延(s)
00:35.4	00:33.5	+0.3	2.2
01:45.1	01.43.7	+0.4	1.8
03.39.6	03.36.8	-0.3	2.5
05.55.1	05.52.8	-0.3	2.0
08.49.6	08.47.7	+0.4	2.3
平均时延(s)			2.16

综合上述测试结果表明，整个移动直播系统功能运行正常，从直播效果可以看出，流媒体服务器对视频数据转发流畅性较好，画面质量直观感觉很好，具有较好的实用性。

6.3 总结

本文主要介绍了基于 iOS 平台开发移动视频采集软件的设计实现过程及利用 Live555 开源协议栈实现流媒体服务器，移动视频采集软件是通过无线接入网络，将摄像头采集到的视频流进行编码、打包、发送到流媒体服务器，流媒体监听来自移动视频采集端和播放器的 RTSP 信令消息，调度流媒体的接收和转发工作。在实现本移动视频直播系统的过程中主要完成了以下几方面的工作：

1) 介绍了移动视频直播系统的设计架构，根据系统架构对各个组成部分进行了解耦开发, 并且选用了 RTSP 协议作为它们之间的通信方式, 基于 Live555 开源协议栈实现 RTSP 信令的交互，完成了手机端移动视频采集软件的设备注册及远程视频播放器的直播请求功能；并根据播放器端的 RTSP 请求，调度媒体流的转发工作，包括流媒体服务器接收来自手机端移动视频采集软件发送而来的 RTP 包, 并对 RTP 包的负载数据 TS 流进行复制打包 RTP 包再发送给播放器端进行播放，实现流媒体的转发。

2) 设计并实现移动视频采集软件。首先，利用 iPhone 手机摄像头实现视频流的采集及预览；根据手机无线接入带宽和用户流量资费情况，并将 x264 编码库移植至 iOS 平台，实现了对手机摄像头采集的视频流进行 H.264 视频编码处理。

3) 为了能在手机资源和处理能力有限的情况下也能获得较快的编码效率，对 x264 编码库中最耗时的 UMHexagonS 运动估计算法代码部分进行了改进, 实现了手机端更快速的 H.264 编码 ;同时经过测试, 改进后的 x264 编码库对编码之后的视频帧信噪比影响并不大, 下降不到 0.15db，码率变化不大，平均增加了 1.58%，改进了编码速率的同时又保证了视频图像的质量。

4) 移植 FFmpeg 库至 iOS 平台，实现了将 H.264 视频帧封装成 TS 流，更好的适应了移动无线网络复杂多变的特性；同时实现将 TS 流保存至本地录像，利用 HLS 技术实现录像的本地回放；并且 TS 流满足了未来语音或字幕等业务的扩展需求，让系统更具有扩展性。

5) 基于 iOS 系统框架完成了移动视频采集软件的开发，最后对移动视频采集软件进行了功能和性能测试。测试结果表明，移动视频采集软件功能运行正常，在 iPhone4s 机器

上直播效果较好，软件资源占用不算太高，很少发现应用程序崩溃现象。

6.4 展望

本文设计的基于流媒体技术的移动视频直播系统具有一定的实用性，但是仍然有很多需要改进的地方。

1) 由于移动视频采集软件是部署在手机移动端的，但开启视频采集时手机的 CPU 使用率较高，耗电量较大，对于手机续航和寿命会有所影响，进一步的改进编码器，降低 CPU 使用率和耗电量是以后工作中改进的方向。

2) 移动视频采集软件虽然已经经过一些优化，但手持和人的步行都会造成较大抖动，从测试的结果可以看出，视频编码效果对运动较为剧烈的编码并不很好，还有很大的改善空间。

3) 本系统仅为实验环境下开发，并没有线上版，流媒体服务器端负载承受能力没有经过实际业务的考验，可能还存在着很多隐藏的问题需要改进。

4) 由于开发条件有限，本测试环境主要以 iPhone4s 作为视频采集前端，并没有对 iOS 所有的机型设备进行系统的测试，各种机型之间差异性测试并没有体现出来。

未来，移动视频直播在视频直播行业将有很大的应用发展前景，因此，实时反馈用户的使用和体验情况是非常有必要的，为用户提供最优质量的视频直播将是今后努力改善的目标与方向。

参考文献

[1] 陈锋锋. 基于 RTSP 的流媒体传输系统的应用开发[D]. 南京邮电大学, 2013.

[2] 刘国成，基于 iOS 的多功能流媒体播放系统设计[D]. 电子科技大学, 2014

[3] 霍龙社, 甘震. 移动流媒体协议综述[J]. 信息通信技术, 2010, (4): 6-13.

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

[5] 刘然. 移动互联网环境下高效流媒体分发关键技术研究[D]. 华中科技大学, 2013.

[6] 章民融, 徐亚锋. 基于 RTSP 的流媒体视频服务器的设计与实现[J]. 计算机应用与软件, 2006, 23(7): 93-95.

[7] Wenger S. H.264/AVC Over IP[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2003, 13(7): 645-656.

[8] Wiegand T, Sullivan G J, Bjontegaard G. Overview of the H.264/AVC Video Coding Standard[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2003, 13(7): 560-576.

[9] Morgan Stanley. Mobile Internet Research Report[EB/OL]. http://www.morganstanley.com/institutional/techresearch/mobile_internet_report122009.html, 2009

[10] InsideFacebook. Facebook Says It Has 845M Users With 425M on Mobiledevices[EB/OL]. http://www.insidefacebook.com/ 2012/02/01/facebook-claims-845m-users-425m-on-mobile/, 2012

[11] The Next Web. Twitter: 100M active users per month, 50% log on every day, 55% on mobile, 1B Tweets every 5 days[EB/OL]. http://thenextweb.com/twitter/2011/09/08/twitter-100m-users-per-month-50-log-on-every-day-55-on-mobile/, 2011

[12] Net2Gather. 2011 Annual Report[EB/OL]. http://www.hkexnews.hk/listedco/listconews/sehk/2012/0417/LTN20120417329.pdf, 2012

[13] Cisco Systems, Inc. Cisco Visual Networking Index: Global MobileData Traffic Forecast Update, 2011–2016[EB/OL]. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf, 2012

[14] 曹达海. 新浪 NBA 文字直播发展现状及对策研究[D]. 上海体育学院, 2011.

[15] 李校林, 刘海波, 张杰等. RTP/RTCP, RTSP 在无线视频监控系统的设计与实现[J]. 电视技术, 2011, 35(19): 89-92.

[16] 王彦丽, 陈明, 陈华等. 基于RTP/RTCP的数字视频监控系统的设计与实现[J]. 安防科技, 2009, (7):

33-36.

[17] RFC 3550, RTP: A Transport Protocol for Real-time Applications[S], 2003.

[18] RFC 2250, RTP Payload Format for MPEG-1/MPEG-2 Video[S], 1998.

[19] 任延珍, 喻占武. 基于 RTP/RTCP 协议的实时数据传输与同步控制策略[J]. 计算机工程与应用, 2003, 39(10): 144-147.

[20] Pantos R, May W. HTTP Live Streaming Draft-pantos-http-live-streaming-05[R]. 2011.

[21] ISO/IEC 13818-1, Information Technology-generic Coding of Moving Pictures and Associated Audio Information:System[S], 1996

[22] ISO/IEC 13818-2, Information Technology-generic Coding of Moving Pictures and Associated Audio Information:Video[S], 1996

[23] ISO/IEC 13818-3, Information Technology-generic Coding of Moving Pictures and Associated Audio Information:Audio[S], 1996

[24] 周敬利, 金毅, 余胜生. 基于 H.264 视频编码技术的研究[J]. 华中科技大学学报: 自然科学版, 2003, 31(8): 32-34.

[25] 段青青, 宋学瑞. 一种 H.264/AVC 中的快速运动估计算法[J]. 计算机工程, 2008, 34(16): 244-246.

[26] 汪世瑜. 基于 H.264 的 UMHexagonS 算法的研究与优化[D]. 杭州电子科技大学, 2011.

[27] 韩峥, 夏志进, 唐昆. x264 解码器的设计与实现[J]. 微计算机信息, 2007, (18): 85-86.

[28] 覃艳. 基于 FFmpeg 的视频格式转换技术研究[J]. 电脑知识与技术, 2011, 12(7): 2912-2913.

[29] 杨宏焱. 企业级 iOS 应用开发实践[M]. 北京: 机械工业出版社, 2012.

[30] Live555 开源库网址[EB/OL]. http://www.live555.com/.

[31] 汪霞. AVI 文件视频流到 H.264 编码转换研究与优化设计[D]. 武汉: 武汉理工大学, 2008.

[32] Richardson I. White Paper: An Overview of H.264 Advanced Video Coding[R]. 2011.

[33] Su H, Zhang C, Chai J, Wen M, Wu N, Ren J. A High-efficient Software Parallel CAVCL Encoder Based on GPU[A]. 34th International Conference on Telecommunications and Signal Processing(TSP)[C], Budapest: IEEE, 2011, 534-540.

[34] Heo J, Ho Y. Improved CABAC Design in H.264/AVC for Lossless Depth Map Coding [A]. 2011 IEEE International Conference on Multimedia and Expo (ICME)[C], Barcelona: IEEE, 2011, 1-4.

[35] 路锦正. MPEG-4/H.264 视频编解码工程实践[M]. 北京: 电子工业出版社, 2011.

[36] 吴晓军, 白世军, 卢文涛. 基于 H.264 视频编码的运动估计算法优化[J]. 电子学报, 2009, 37(11): 2541-2545.

[37] Dufaux F, Moscheni F. Motion Estimation Techniques for Digital TV: a Review and a New Contribution[J]. Proceedings of the IEEE, 1995, 83(6): 858-876.

[38] Li R, Zeng B, Liou M L. A New Three-step Search Algorithm for Block Motion Estimation[J]. IEEE Transactions on Circuits and Systems for Video Technology, 1994, 4(4): 438-442.

[39] Jain J, Jain A. Displacement Measurement and Its Application in Interframe Iimage Coding[J]. IEEE Transactions on Communications, 1981, 29(12): 1799-1808.

[40] Zhu S, Ma K-K. A New Diamond Search Algorithm for Fast Block-matching Motion Estimation[J]. IEEE Transactions on Image Processing, 2000, 9(2): 287-290.

[41] Zhu C, Lin X, Chau L P. Hexagon-based Search Pattern for Fast Block Motion Estimation[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2002, 12(5): 349-355.

[42] Lifen X, Chunqing H, Bihui C. UMHexagonS Search Algorithm for Fast Motion Estimation[A]. 2011 3rd International Conference on Computer Research and Development (ICCRD)[C], Shanghai: IEEE, 2011, 1: 483-487.

[43] 汪世瑜. 基于 H.264 的 UMHexagonS 算法的研究与优化[D]. 杭州电子科技大学, 2011.

[44] JVT S202, Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG[S], 2003.

[45] Qiu X, Huang C. An Improved Algorithm of Fast Motion Estimation Based on H.264[A]. 2010 5th International Conference on Computer Science and Education (ICCSE)[C], Hefei: IEEE, 2010, 1717-1721.

[46] Li Y, Xiao J, Wu W. Motion Estimation Based on H.264 Video Coding[A]. 2012 5th International Congress on Image and Signal Processing (CISP)[C], Chongqing: IEEE, 2012, 104-108.

[47] Cheng Y, Guo S X, Xiao M L, et al. An Optimized Motion Estimation Algorithm Based on UMHexagonS[A]. IET International Conference on Information Science and Control Engineering 2012 (ICISCE 2012)[C], Shenzhen: IET, 2012, 1-5.

[48] 王磊. 基于 x264 的智能手机监控系统的设计与研究[D]. 昆明理工大学, 2012.

[49] 吴张顺, 张珣. 基于 FFmpeg 的视频编码存储研究与实现[J]. 杭州电子科技大学学报, 2006, 26(3): 30-34.

[50] JRTPLIB 开源库网址[EB/OL]. http://research.edu.uhasslte.be/jori/page/index.php?n=CS.Jrtplib, 2004-11-18/2010-12-20.

[51] 章民融, 徐亚峰. 基于 RTSP 的流媒体视频服务器的设计与实现[J]. 计算机应用与软件. 2006, 23(7): 93-95.

[52] Ismail Y, McNeely J B, Shaaban M, et al. Fast Motion Estimation System Using Dynamic Models for H. 264/AVC Video Coding[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2012, 22(1): 28-42.

[53] Xiph.org 测试视频序列[EB/OL]. http://media.xiph.org/video/derf/.

致谢

岁月如歌，光阴似箭，三年的研究生生活即将结束。回首这两年多的求学历程，我收获颇多，不仅学到了许多有用的知识，同时理论研究水平和实践能力也得到了提升，对那些引导我、帮助我、激励我的人，我心中充满了感激。

首先要感谢导师彭宏副教授，论文定题到写作定稿，都离不开彭老师大量的细心指导。在我攻读硕士研究生期间，深深受益于彭老师的关心、爱护和谆谆教导。她作为老师，以严谨治学的态度令我敬仰，帮我点拨迷津；作为长辈，关怀备至，教会我做人的道理，让人感念至深。能师从彭老师，我为自己感到庆幸。在此谨向彭老师表示我最诚挚的敬意和感谢!

感谢孟利民老师给我提供良好的科研环境，让我有机会参与实验室项目。在做项目期间，孟老师带领我们一起参与项目开发，耐心指导在项目过程中遇到的问题，培养我的分析问题和解决问题的能力。

感谢浙江省通信网技术应用研究重点实验室的各位老师，他们在我的学习和生活上给予了我极大的帮助，帮我克服了许多困难。感想一直关心与支持我的同学和朋友们，陈娴，潘力策，王慧州，朱彭，郑骁能等，感想你们的鼓励和帮助，同窗之谊，我将终生难忘！

感谢我的家人一直在背后默默地支持我，是家人的关爱和信任，让我更加充满前进的动力。

最后，衷心地感谢百忙之中抽出时间来评阅论文和参加答辩的各位专家、教授！

攻读学位期间参加的科研项目和成果

参加的科研项目

[1] 企业委托的开发项目：浙江省通信网技术应用研究重点实验室-视频监控 iOS 客户端软件

录用和发表的论文

[1] 彭宏, 吴海巍, 叶敏展, 陈娴. 基于流媒体的移动视频直播系统的设计与实现[J]. 电子技术应用, 2014, 40(9): 111-113.