广东工业大学硕士学位论文

（工学硕士）

基于RTSP的音视频传输系统研究与实现

张 宇

二〇一六年五月

分类号：_____ 学校代号：11845

UDC：_____ 密级：_____ 学号：2111305015

广东工业大学硕士学位论文

(工学硕士)

基于 RTSP 的音视频传输系统研究与实现

指导教师姓名、职称：曾碧 教授

学科(专业)或领域名称：计算机科学与技术

学 生 所 属 学 院：计算机学院

论 文 答 辩 日 期：二〇一六年五月

A Dissertation Submitted to Guangdong University of Technology

for the Degree of Master

(Master of Engineering Science)

Research and Implementation of Audio and Video Transmission System based on RTSP

Candidate: Zhang Yu

Supervisor: Prof. Zeng Bi

May 2016

School of Computer Science and Technology

Guangdong University of Technology

Guangzhou, Guangdong, P. R. China, 510006

摘 要

家庭服务机器人逐渐从提供单一的家庭劳动服务发展为面向家庭用户提供多层次的信息化服务，其中包括视频监控，多传感器的环境监测等。家庭服务机器人中实现的音视频传输系统需要考虑音视频同步问题，以及面向家庭用户的多客户端等问题。本文研究的目的是以家庭服务机器人为应用背景，设计并实现其中的音视频传输子系统，并重点研究系统中的音视频同步技术。主要工作包括如下几个方面的内容：

(1) 介绍了家庭服务机器人的发展趋势，对音视频同步技术的研究现状进行深入分析，对 RTSP(Real Time Streaming Protocol)协议以及 Android 系统下的音视频解码技术的现状进行研究。接着对文中所涉及的流媒体技术进行阐述，对音视频数据的渐进式下载和实时流式传输方式进行分析，并重点研究了用于实时流式传输方式的流媒体传输协议。

(2) 针对音视频传输系统中存在的音视频失同步问题，进行重点研究。在大量查阅文献和分析得出引起音视频失同步现象的原因的基础上，提出了一种基于网络延时检测的自适应音视频同步方案。该方案通过将网络时延波动的方差和 RTCP(Realtime Transport Control Protocol)协议的网络状况的拥塞控制机制结合起来，完成网络状况的判定，并基于此选择不同网络状况下的同步方案。在高延迟网络下采用基于同步数据节点和时间戳的音视频同步技术，而在低延迟网络下采用接收端基于时间戳的同步技术。并搭建实验平台对该方案进行仿真实验，实验结果表明能很好的实现音视频同步。

(3)音视频流媒体传输系统设计与实现。

在服务器端主要深入分析 Live555 的源码，重点研究流媒体服务器的工作原理。在此基础上，针对 Live555 存在的一些性能问题和功能缺陷，进行二次开发，很好地改善了存在的性能问题，扩展了其作为流媒体服务器的功能。

在客户端主要设计并实现了基于 Android 平台的多媒体应用软件，详细介绍了在 Android 平台上完成的音视频播放过程。通过移植 FFMPEG 和 SDL 到 Android 平台，开发了 Android 平台下的全格式支持的多媒体播放器，很好地解决了 Android

平台对于多媒体文件格式和音视频编码格式支持有限的问题。并基于此提出了Android 平台下的音视频同步算法，实现了音视频的同步播放，而且提出的同步算法具有非常好的跨平台特性，可以方便地移植到其他客户端平台。

(4) 介绍用于运行整个音视频传输系统的软硬件环境，在系统部署完成后进行相关的测试工作，测试结果表明，系统能很好地满足音视频传输要求。

关键词：RTSP； Live555； 音视频同步； FFMPEG； Android

ABSTRACT

The home service robot has gradually developed from providing single family labor service to provide multi-level information services for domestic users, including video surveillance, multi sensor environment monitoring and so on. The audio and video transmission system implemented in the home service robot needs to consider the problem of audio and video synchronization, as well as multi client for home users. The purpose of this paper is to design and implement the audio and video transmission subsystem based on the home service robot, and the audio and video synchronization technology in the system. The main work includes the following aspects:

(1) Firstly, this paper introduced the development trend of home service robot, the research status of audio and video synchronization technology and RTSP (real time streaming protocol) protocol, the audio and video decoding technology in Android system. The media technology involved in the paper, such as progressive download and streaming real-time transmission mode, are analyzed. Then we focus on the real-time streaming transmission protocol.

(2) The problem of audio and video synchronization in the audio and video transmission system is studied. On the basis of a large number of literature and analysis of the reasons for the loss of synchronization of audio and video, an adaptive audio and video synchronization scheme based on network delay detection is proposed. The scheme combined with the fluctuation of network delay and the network congestion control mechanism of RTCP (realtime transport control protocol) protocol to complete the judge of the network status, and choose synchronization scheme in different status of network. The solution based on synchronous data node and time stamp is used in the high latency network. Then we set up the experimental platform to carry out the simulation experiment. The experimental results show that it can achieve a good effect of audio and video synchronization.

(3) The Design and implementation of audio and video streaming media

transmission system. In the server side, we have analyzed the source code of Live555 project, and focused on the working principle of streaming media server. We have updated this project for some performance problems and functional defects, and improved these performance issues and expanded its functionality as a streaming media server.

In the client side, we used the Android platform as the receiver of the audio and video transmission system, and then deeply introduced the play of audio and video on the Android platform. Then we transplanted FFMPEG and SDL to Android platform, and developed a multimedia player under the Android platform, and solved the existing problem of supporting a limited format of multimedia file format and audio and video coding format in Android platform. According to these, we proposed an algorithm of audio and video synchronization based on Android platform and realized the synchronous playback of audio and video. Moreover, the proposed synchronization algorithm in this paper, has a very good cross platform characteristics and can be easily ported to other platforms.

(4) Finally, we introduced the software and hardware environment for running the whole audio and video transmission system and did some tests after the completion of the system deployment. Test results show that the system can satisfy the requirement of audio and video transmission, and has better effect in synchronization of audio and video.

Keywords: RTSP, Live555, Synchronization of audio and video, FFMPEG, Android

目 录

CONTENTS

第一章 绪论

1.1课题背景与研究意义

移动互联网和嵌入式技术的高速发展，从各个方面深刻地改变了人们的生活和生产方式。基于这两个技术的机器人不再是传统的玩具或者是电影中的英雄形象，而逐渐变成人们日常生活中不可缺少的家庭服务提供者，家庭服务型机器人市场逐渐成为机器人产业中的很大一个组成部分[1]。家庭服务机器人在代替人工完成家庭日常清洁等家庭劳动中具有无可替代的优势，而且随着我国的人口老龄化等等问题逐渐加剧，对服务机器人的数量有了更大的需求，同时对机器人的智能程度以及信息化技术方面提出了更高的要求。这已经被列入了国家的发展规划中[2]。家庭服务机器人逐渐从提供单一的家庭劳动服务发展为向家庭用户提供多层次的信息化服务，其中包括视频监控，多传感器的环境监测等[3]。

不少高端家庭服务型机器人的面世，进一步推动了服务机器人技术的发展。近年来集成电路技术的发展成熟，使得处理器芯片价格逐渐回落，部分高端机器人中已经搭载了比较强大的处理器芯片和先进的操作系统，使得服务机器人的计算能力得到了很大的提高，这为家庭服务机器人进一步进行高复杂度、高精度以及高智能运算奠定了硬件和软件基础。如有学者在机器人上实现通过人机交互进行控制的半自动化SLAM(Simultaneous Localization And Mapping)算法[4]。而在有些应用场景中，具有较强计算能力的服务型机器人却并不需要完成复杂的计算，因此，在进行低计算量任务的情形下存在处理器资源等的浪费。针对这一部分闲置计算资源的利用与研究对于机器人技术的发展也是有着很重要的意义。有学者提出基于马尔科夫链和 BP 神经网络来实现家庭服务机器人上的语音识别[5]，还有学者利用家庭服务机器人平台的 GPU 进行硬件加速，来实现仿射过滤实时图像处理算法[6]。服务型机器人以其超强的多媒体处理能力逐渐成为安防领域的研究热点。正是在这样一种背景下，家庭安保机器人成为数字化家庭系统中的一个重要研究部分。

数字化安防系统是通过在家庭中部署安防设备以及智能软件系统，对家庭中环境和状况进行实时监控，以通过网络等通信方式实现紧急情况的报警处理，来确保家居环境的安全。其中安防视频监控是一个重要的组成部分。受摄像头位置固定的特点限

制，传统视频监控具有留有死角，工程安装布线麻烦等缺点。因此，将机器人引入家庭安保系统逐渐成为了一种趋势。利用机器人的自主运动，结合移动互联网以及无线网络等通信技术，来实现家庭安保机器人成为一个热点研究的问题[7]。

伴随着技术和市场需求的演变，视频监控技术主要经过了三个阶段的发展[8]。第一代主要为 20 世纪 70 年代，模拟电子电路技术为主要的电子技术，在那个时期，主要使用的是模拟摄像机进行模拟信号采集，通过模拟显示器来向用户呈现视频内容，视频存储以最原始的磁带作为数据载体，这种技术做成的视频监控系统叫做 CCTV(Closed Circuit Television)。第二代主要为 20 世纪 90 年代中期，这个阶段以数字化为鲜明特点，过渡期使用模拟摄像机做为视频数据采集设备，使用 DVR(Digital Video Recorder)作为视频数据存储和处理的设备，随着数字技术和网络技术的发展，逐渐演变成为了 NVR(Network Video Recorder)，使用数字化的设备形成的视频监控系统很好的满足了大规模的安防系统的需求。第三代主要为 2000 年左右，主要以智能为特点。随着视频监控的广泛运用，视频监控摄像头呈现出高密度的发展趋势，中国用于城市安防等的摄像头已经达到了千万量级[9]，由此引起了海量监控数据的问题。使用人来对海量的视频监控数据进行处理是一个非常棘手的问题，有学者研究过人在盯着监控画面 22 分钟以后，对视频内容的忽视程度高达 95%。于是，为了解决人眼对于海量视频监控数据处理存在重大遗漏的问题，提出了研发智能视频监控系统。

3G 技术的成熟和 4G 技术的日益普及，使移动互联网更进一步渗透进人们的生活中，已经在全世界范围内引起了深刻的变革[10]。作为移动互联网中的重要的一环，移动终端技术目前正处于高速发展阶段，其中的终端硬件技术和终端软件技术更迭速度非常快。Google 公司研发的 Android 系统作为一个成功的移动操作系统逐渐占据了很大的市场份额，基于此形成了庞大的移动互联网生态体系。

基于 RTSP(Real Time Streaming Protocol)的流媒体技术在工程应用中非常普遍，在各大类的流媒体应用领域中都有出现，扮演着非常重要的角色。正是在这样的一个大背景下，本文针对家庭服务机器人的音视频传输子模块进行研究。对移动家庭服务机器人的特定应用场景下，音视频的实时流式传输和 Android 平台下的流媒体接收、音视频解码、音视频同步播放，以及服务机器人端的音视频采集、流媒体实时转发等关键技术展开研究。

1.2 国内外研究现状

1.2.1 音视频同步技术研究现状

在视频点播（Video On Demand）、安防监控等多媒体应用场景中，音视频是否同步对于服务质量（Quality of Service）非常关键，是直接影响用户体验的关键点[11-12]。国内外不少学者针对音视频同步问题进行了研究[13-24]。目前已有若干种方法，其中有Escobar 等使用的流同步方式是有一个非常重要的前提，整个算法的同步过程都必须在一个同步时钟下进行[13]。在视频会议的应用场景中，为了保持音视频同步，Aggarwal等人提出在视频会议这种对方可视性场景中，可以利用人的嘴唇形状来完成音视频同步[11]。也有学者提出将音频数据和视频数据在单独采集的过程中，在数据帧上保存时间戳信息，在通过传输系统以后，使用时间戳完成播放过程中的同步控制[15-16]。基于这种思路，也可以基于音视频数据的内容等生成一个具有标志意义的数值，使用这个值使得接收端根据这个标志的值来确认具体的数据同步过程[17]。随着视频水印技术的发展，不少学者研究将水印技术嵌入 H.264 的视频编码过程，主要是利用 H.264 的帧内预测部分[18-20]。携带有水印的数据载体能通过编码隐藏进视频编码过程，同样，携带有音频编码的数据载体也能以同样的方式隐藏进去。刘添等提出了将音频编码数据嵌入进视频编码的帧间预测过程，从而将音视频传输信道由两个复用为一个，在接收端在解码视频过程中提取出音频编码数据，使得在时间上本就具有相关性的音视频数据之间在传输过程中也是以一个整体进行传输，保证接收端对于音视频数据处理的完全同步[21]。而时美强等人则是利用 DCT 系数的奇偶不同，而将音频编码数据隐藏到视频中[22]。在视频传输过程中的音视频同步控制也是一种重要的同步控制方法，利用流媒体数据传输协议 RTP/RTCP 中的时间戳信息，在接收端以音频数据流来同步视频数据流的方式来实现音视频的同步控制[23-24]。

不同的音视频同步技术是针对特定的应用场景的特点提出的，因此同步方法本身存在一些固有的特点。使得必须对于各种音视频同步方法进行深入分析，进而设计适合本文的家庭服务机器人的特定应用场景的方法。在发送端和接收端需要统一的同步时钟的方法中，双方的同步时钟的计算过程容易受网络状况的影响，在网络出现拥塞等情况下，对于音视频数据帧的同步的计算过程比较困难。在视频会议等双方通信用户均可视的情况下的基于嘴唇形状的同步方法中，其应用场景的局限性较大，必须是在通信双方可见这个大前提下工作，不适合用于通常的视频监控场合，而且，在同步过程中对于嘴唇的定位算法本身也是一个研究的热点问题，算法的实现过程比较复杂。

在 H.264 的帧间预测编码过程中，通过一定的映射规则，把音频的编码数据映射到视频的编码过程中去，这种方法确实可以实现视频数据传输的信道的复用，减少传输过程中占用的带宽，但是，在应用过程中需要针对 H.264 的编码器进行修改，经过编码后的数据必须使用专用的解码算法才能实现视频解码以及音频数据的提取，传输的流不再是标准的音视频媒体流，对于以后系统集成等过程中的兼容性有影响。而且，在网络出现拥塞的情况下，如果丢帧，那么音频和视频数据将同时出现丢帧的现象，这对接收端的正常解码流程的影响非常明显。通过 DCT 的奇偶特性来将音频编码数据映射进去的同步方法，其传输占用的带宽较大，在网络状况出现拥堵时，出现延迟较大的现象。

1.2.2 RTSP 协议与 Android 系统音视频解码技术研究现状

多媒体数据的实时流式传输需要使用专用的流媒体传输协议，基于 RTSP 协议的流媒体传输技术在视频监控和流媒体服务器中应用十分广泛。茅炎菲等基于 Live555 实现了多媒体服务端，并通过与基于 HTTP 协议的监控系统进行比较，验证了 RTSP 在流媒体传输方面具有更好的性能[25-26]。刘大红等基于 VLC 开源项目研究流媒体服务器的设计与实现，并将其移植到嵌入式 Linux 系统中[27]。Android 系统以其开放源码和基于 Linux 等特点吸引了不少学者对其各个方面进行研究。有学者对 Android 系统的音视频编解码技术进行了深入的研究，指出 Android 系统提供的多媒体处理应用程序接口存在音视频编解码格式和多媒体文件格式支持有限的缺点，通过将开源项目 FFMPEG 移植到 Android 平台作为一种插件来扩展其多媒体处理能力[28-29]，这个研究中仅仅实现了解码过程，并没有指出如何将 FFMPEG 与 Android 平台的图像渲染结合起来。还有学者基于 Android 进行音视频数据采集，并通过 RTSP 协议发送出去[30]，杨飞等实现在嵌入式开发板上移植 Android 系统，并以此系统作为服务器，通过调用 Android 的 API 进行音视频数据采集与编码，移植 Jrtplib 实现 RTP 打包发送[31]。

在本文的家庭服务机器人的视频监控应用场景下基于 RTSP 协议进行音视频数据传输，需要考虑多元化的客户端的音视频处理跨平台问题。在上述研究中在 Android 平台进行解码大多采用的是 FFMPEG 移植到 Android 平台的方案，而 FFMPEG 对于主流媒体编码标准 H.264 格式的视频数据进行解码后生成的是 YUV 格式的图像数据，Android 平台的 ImageView 等接口中对于图像数据的支持格式为 RGB 格式，还需要使

用格式转换算法进行原始图像数据的格式转换，增加了系统的计算量。同时，基于 Android 系统提供的应用程序接口函数完成的音频和视频播放过程与平台具有紧耦合性，在这种应用场景中进行音视频同步更加复杂。对于多种客户端平台必须开发多套程序以实现对应功能。在软件设计过程中能够解决平台问题的方法是分层，即在软件体系结构中插入一层中间层，实现程序库的平台无关性，使得程序库的用户在任一支持平台下可以通过相同的接口来完成相同的功能。而 SDL(Simple DirectMedia Layer)开源项目通过在底层为不同的平台实现不同的代码，屏蔽了各种平台的底层技术细节，为多媒体数据的输出提供了非常好的跨平台特性，因此，本文中将研究基于 FFMPEG 和 SDL 实现一套在不同平台下可以完成音视频解码、同步控制、以及输出给用户的程序库。

1.3课题来源与本文的研究内容

本课题来源于家庭智能服务机器人上的家庭安防视频监控项目。家庭智能服务机器人逐渐从单一的提供简单的家庭智能服务，演变为利用人工智能向广大用户提供多元化功能丰富的智能服务。实验室科研团队基于嵌入式技术背景，结合工程实践，针对市面上已有的家庭服务机器人不够智能化、功能单一化等缺点提出了基于智能服务机器人进行安防视频监控的扩展。而随着移动互联网技术的逐渐兴起，越来越多的用户习惯于使用移动智能终端来完成日常的信息事务处理，这引起了大量应用从传统的个人计算机客户端走向了移动智能终端。而 Google 公司研发的 Android 平台由于其基于强大的 Linux 内核，而且开放源代码，使不少学者将目光投向了这一平台。正是基于这样一个学术研究和工程应用背景，本课题针对项目中的音视频传输这一个子模块进行深入研究，对其中运用到的各项开源技术进行优化并对其中遇到的音视频失同步现象提出解决方案，解决存在的音视频失同步问题，优化服务端的性能，改善客户端对于音视频解码格式支持不足的缺点，为用户提供更好的移动应用体验和更加优质的流媒体服务。

1.4本文的结构与章节安排

第一章 绪论。通过分析国内外的研究现状，总结前人的研究成果，综合分析当前研究中存在的不足与诸多问题，提出本文拟要解决的家庭服务机器人的音视频传输系

统中的音视频同步等问题。

第二章 流媒体技术概述。对本文所涉及的流媒体基础概念以及流媒体技术等做一个简要的概述与归纳。

第三章 音视频同步技术研究。为了能够保证多媒体通信的服务质量，提出针对音视频采集的特点来实现音视频的全局同步控制，实验表明该算法能够很好地解决音视频同步问题。

第四章 流媒体传输系统设计与实现。通过绪论中分析得出在 Live555 的工程实践中存在很多不足，本章通过在深入分析 Live555 的源代码的基础上，针对项目本身的这些不足与缺陷提出了优化以及解决方法。通过分析 Android 平台对于音视频解码以及多媒体格式支持的不足等存在的问题，提出了将 FFMPEG 移植到 Android 平台，并基于此来做多媒体解码，开发基于 FFMPEG 的多媒体客户端，实现 RTSP 音视频流的同步播放。

第五章 系统测试。这章主要针对第四章中设计的流媒体传输系统的服务端和客户端进行功能测试，并分析和总结测试结果。

第二章 流媒体技术概述

流媒体是一种可以将媒体内容直接通过网络进行实时传输的传输方式。本章先针对流媒体传输技术的两种传输方式等基本概念进行介绍，接着阐述流媒体传输过程中涉及到的相关的 RTSP 协议等，接着介绍在实时的网络传输中对于不同网络状况下的拥塞控制策略等，最后针对流媒体传输中常用的开源流媒体项目进行简要的分析。

2.1 音视频的流式传输概述

2.1.1 两种传输方式

在整个流媒体技术框架中最为重要的就是流媒体本身，流媒体是由音频和视频数据的流式传输所形成的。流媒体技术是音视频传输中非常重要的技术[32]，它综合运用多个学科的技术，如网络传输，多媒体处理，通信等等。在工程实践中音视频主要通过渐进流(Progressive streaming)和实时流(Realtime streaming)这两种流式方法来完成传输过程。

(1) 渐进流式下载

渐进流式下载方式是通过使用超文本传输协议 HTTP 对多媒体数据进行渐进式的顺序下载，在媒体数据传输到本地的同时，已经缓存到本地的媒体数据可以进行播放。在用户看来实现的是下载的同时进行播放。这种流式传输技术通过标准的 HTTP 服务器就可以实现。基于这种技术特点，使得下载过程中的文件管理十分简单，但是不具备现场直播能力，属于点播的范畴。这种传输方式的特点，使得其非常适合于较小文件的传输，对于网络状况的适应性较好，即使在低带宽的网络连接下也可以工作。传输过程与传输内容具有独立的特点，因此，所有的文件类型都可以采用这种方式来实现传输过程。HTTP 协议基于 TCP 的特性使得这种传输方式具有非常稳定可靠的特性，当网络出现拥塞等现象时，丢失的数据包会自动超时重新传输等，保证了媒体数据的完整性。基于 HTTP 的传输方式具有最大的优点是可以穿透防火墙，缺点是 HTTP 协议本身不支持广播或者多播。

渐进流式下载方式具有这些优缺点使得其适合对于视频质量要求较高的小文件等，如节目的片头和插播在节目中的广告部分等。而不适合于文件时长较大的片段以

及需要对视频节目的内容有随机访问需求的应用场景。

(2) 实时流式传输

实时流式传输指的是通过将媒体数据的传输速率与网络的带宽相匹配的流式传输技术，使得媒体数据可以通过信道实时发送出去。这种传输方式较渐进流式下载复杂，必须有专用的服务器和传输协议作为支持。在客户端接收到的图像的质量与发送端的网络连接带宽有关系，也就是说，网络带宽降低时，图像质量会因此而受到影响。这种流式传输方式的显著特点就是可以进行媒体流的广播和多播，不同于渐进式的顺序下载方式，实时流式传输支持节目内容的随机访问，这样可以节约客户端的存储空间。基于复杂的实时流式传输协议，可以实现文件中的独立流的传输，还可以为每个流分配带宽。同时，其带宽匹配特性会导致在媒体文件的码率超过网络传输速度时，视频观看过程将出现卡顿、断断续续等现象，极大地破坏用户体验，并且这种传输方式不支持所有的文件格式，特别对于码率不固定的媒体文件。

实时流式传输方式通常采用 RTP/UDP、RTSP/TCP 协议与流媒体服务器进行通信，流媒体服务器再将音视频流发往运行音视频播放器程序所在客户机。这些专用协议的具体细节将在下文中进行阐述。

2.1.2 流媒体技术原理

基于 IP 的网络中传输音视频数据的过程中，媒体数据会被拆分成一个个的数据包，在此基础上进行流媒体数据的封装打包。经过 IP 网络进行传输，在传输的过程中受网络状况的影响较大，其中网络可能存在拥塞问题，传输中路由选择不同等，使得各个数据包的到达时间不同。

实时流式传输使用专用的协议，在流媒体服务系统中通常使用 HTTP 协议完成应用数据和控制信息等的传输，而运用 RTP、RTSP 等协议完成音视频数据的流式传输。其传输过程如下图 2-1 所示，用户从浏览器中浏览页面内容，通过 HTTP 协议与 Web 服务器进行应用类数据和控制信息的交互；Web 服务器从用户的请求报文中提取出用户对于流媒体数据的请求。同时，浏览器中会通过运行音视频助手类软件，通过 HTTP 协议获取初始化数据，用于对自身进行初始化。在初始化以后，助手类软件将与流媒体服务器通过专用的流媒体传输协议进行通信，通过 RTSP 协议实现通信双方控制信息的交互，接着流媒体服务器将被访问的媒体数据按照 RTP 协议的格式发送到客户端，

客户端按照 RTP 协议格式从接收到的数据包中提取出媒体数据，完成音视频数据的解码，最终呈现给用户。

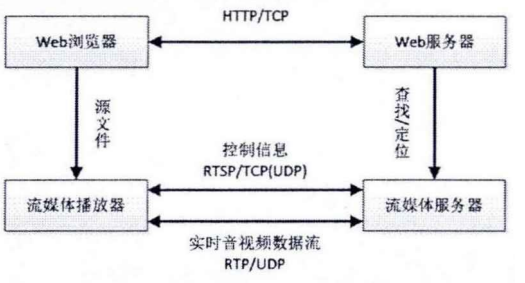

图 2-1 流媒体数据传输原理图

Figure 2-1 Diagram of streaming media data transmission

2.2 流媒体传输协议

上一节介绍了音视频数据的实时流式传输，在流式传输过程中非常关键的技术是流媒体传输协议。基于 IP 的网络本身存在一些复杂的网络状况，网络状况较差时会出现网络拥塞、数据包发送的网络延时较大、以及抖动等现象，这些问题需要从协议层次进行分析与考虑才能便于处理这类问题。流媒体传输协议就是针对在 IP 网络中可能出现的复杂网络情况而设计的媒体数据传输协议，可以帮助用户解决传输过程中存在的诸多问题。

2.2.1 RTP/RTCP 协议

RTP(Real-Time Transport Protocol)实时传输协议是由 IETF(The Internet Engineering Task Force)提出的，其相关的 RFC 文档为 RFC1889 和 RFC3350，前者是实时传输协议的主体部分，而后者则是针对 RFC1889 文档的修订以及对内容的补充。RTP 协议的设计是为实时媒体数据提供传输功能，其可以实现一对一以及一对多的网络传输，主要给用户提供时间信息并且实现媒体流的同步等功能。其主要完成媒体数据传输功能，因此对于传输层的可靠性要求较控制类协议低，直接基于 UDP 工作。

RTP 协议数据格式主要分为头部和正文两块，其中头部如图 2-2 所示。各个字段均非常重要，限于篇幅，本文针对其中与本文中的同步算法相关的部分进行介绍。

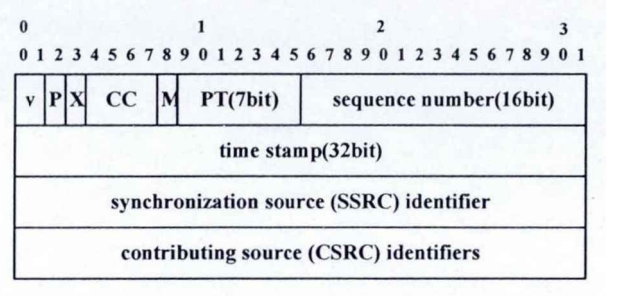

图 2-2 RTP 头部

Figure 2-2 RTP header

PT：负载数据类型字段，在头部中占 7bit 长度数据，用来表示正文部分媒体数据使用的编码算法类型，RFC 中对于早期就存在的编码算法都作出相应的标识，而由于这些 RFC 文档出现的时间较早，使得在其后出现的编码算法在文档中没有对应的标识值，如最常用的 H.264 算法使用的是 96。在具体应用中要通过 SDP 文件等来进行协商。在传输过程中，接收端可以通过这个字段确定正文部分的有效载荷是何种类型的编码数据，以便据此采用相对应的的解码算法进行视频解码。

Sequence number：序列号字段，在头部中占 16 位长度数据，接收端根据这个字段进行判断是否存在数据包的损失现象。出于安全性的考虑，该字段的初始值是通过随机的方式进行选择的。在传输的过程中，这个字段在成功发送出 RTP 数据包后依次递增。

Time stamp：时间戳字段，在头部中占 32 位长度数据，该字段表示的是正文部分中第一个八进制数据的采集时间戳，RFC 文档中规定，这个字段要用呈线性增加的时钟来生成。这个字段完全可以用来表示整个数据单元各自的先后顺序，方便用户在运用的过程中实现同步和抖动的分析，同时其作为第一个八进制数据的时间戳，使得其可以用于表示正文中数据的采样时间，便于利用其来实现媒体同步等算法。

RTP 协议仅仅实现了多媒体数据的封装传输，其在复杂网络环境中传输必须要使用相关的协议进行传输控制来保证 QoS。RTP 协议配套使用 RTCP(Real-time Transport Control Protocol)协议来实现这个功能。RTCP 为 RTP 协议添加流量控制以及拥塞控制机制以保证传输服务质量。

RTCP 协议主要通过在 RTP 会话中以一定频率发送 RTCP 的数据包到对端，通过与对端交互实现对于 QoS 的监听并交换参与会话的双方各自的基本信息。RTCP 协议规定有五种基本报文，各种报文功能不同，其主要的报文类型如下表 2-1 所示。

表 2-1 五种基本的 RTCP 报文

Table 2-1 Basic RTCP messages

报文类型	报文名称	报文作用
Sender Report	发送者报告	在通信的两方中，发送方向其他参与者报告发送方的信息，包括 NTP 和 RTP 时间戳等
Receiver Report	接收者报告	在通信的两方中，接收方向发送方通告接收方的统计数据，包括丢包和抖动数据等
SDES	源描述报文	实现将自身信息通知给其他同步源
BYE	结束报文	供正常或者出错情况下结束会话
APP	应用定义报文	供应用程序自行定义用可实现用户自定义功能

RTCP 协议可以针对 RTP 协议的传输质量提供一个评价指标，这个评价指标对于各种根据网络环境进行调整的自适应编码系统非常重要，同时也可以为用户对于网络是否出现故障以及网络状况的好坏做出评价提供客观依据。对于所有的传输参与者发送这种类型的报文可以对网络故障的范围进行评判，是整个网络都处于一个故障状态，还是局部存在故障。这种评价功能主要是通过 RTCP 的 SR 和 RR 报文实现的。

2.2.2 RTSP 协议

实时流传输协议是一个对媒体数据传输过程的控制协议，其自身是不对流媒体数据进行传输的，其对参与数据传输的一个或多个数据发送方进行控制[33]。RTSP 对于媒体数据在服务器端和客户端的传输过程提供的控制主要有开始播放，播放暂停，以及快速前进后退等。不同于超文本传输协议，RTSP 是一个有状态的协议。RTSP 在传输层默认使用 TCP 协议进行传输，占用端口号为 554。类似于 HTTP 协议，RTSP 主要有两类报文，分别为请求报文和响应报文，报文数据格式也是依据 RFC822 中的消息格式。其请求和响应报文的消息格式如图 2-3 所示。

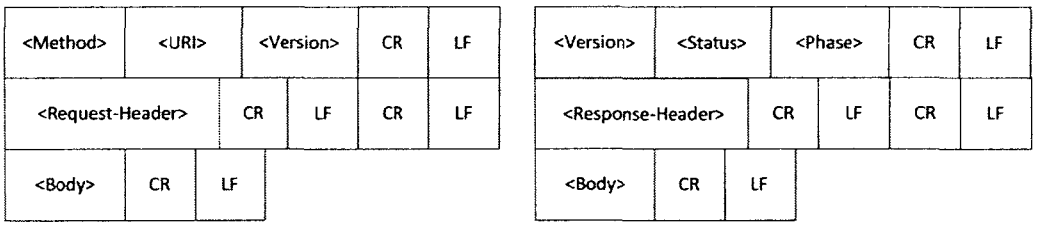

图 2-3 RTSP 的两种报文消息格式

Figure 2-3 Message formats for RTSP

在请求报文中的 Method 字段表示的是 RTSP 方法，RTSP 协议中可选用的方法字段的值如表 2-2 所示，在表中流程列表示是从服务端向客户端的请求报文，还是服务端向客户端发送的响应报文。请求的内容列表示该方法是对流数据还是对表示数据进行请求，备注一栏表示的是该字段对于 RTSP 会话过程是否为必须要的还是可选的。

表 2-2 RTSP 协议请求报文中的 Method 字段详解

Table 2-2 Method field detailed for RTSP protocol request message

方法名	流程	请求的内容	备注
OPTIONS	客户端发到服务端 服务端发到客户端	表示、流数据	必须
DESCRIBE	客户端发到服务端	表示、流数据	推荐
ANNOUNCE	客户端发到服务端 服务端发到客户端	表示、流数据	可选
SETUP	客户端发到服务端	表示数据	必要
PLAY	客户端发到服务端	表示、流数据	必要
PAUSE	客户端发到服务端	表示、流数据	推荐
RECORD	客户端发到服务端	表示、流数据	可选
REDIRECT	客户端发到服务端	表示、流数据	可选
SET_PARAMETER	客户端发到服务端服务端 发到客户端	表示、流数据	可选

GET_PARAMETER	客户端发到服务端服务端发到客户端	表示、流数据	可选
TEARDOWN	客户端发到服务端	表示、流数据	必要

2.2.3 SDP

会话描述协议 SDP(Session Description Protocol)是一种基于文本的，用于描述会话的数据格式，具有很强的可扩展性，应用范围比较广。其主要为会话过程中的各个环节，如通知，邀请，以及各种多媒体会话的初始化等提供描述功能。SDP 可以使用如表 2-3 中所示的各种传输协议。

表 2-3 供 SDP 使用的传输协议

Table 2-3 Transport protocol for SDP use

协议简称	协议全称	协议简介
SAP	Session Announcement Protocol	会话通知协议，协助视频会议广告及其他组播会话过程
SIP	Session Initiation Protocol	会话初始化协议，由 IETF 制定的多媒体通信协议，用于创建修改释放会话
RTSP	Real Time Streaming Protocol	实时流传输协议，基于 IP 网络定义一对多发送多媒体数据
MIME	Multipurpose Internet Mail Extensions	多用途互联网邮件扩展类型，为不同后缀名的文件指定特定应用程序打开
HTTP	HyperText Transfer Protocol	超文本传输协议

SDP 协议发出多媒体会议通告，并通知参与者该会议地址以及参与此次会议需要的工具等内容，其对于大范围的网络环境等具有较强的适应性和通用性，但其不参与协商媒体编码，而是通过 RTSP 来实现。其本质上就是一个关于格式的描述，其中包括会话参与者支持的多媒体传输协议，音视频编解码算法等。在会话中包含多个媒体数据流的场景中，SDP 协议中必须包含每一个媒体数据各自的参数。SDP 协议中用于描述的数据的格式如下图 2-3 所示。

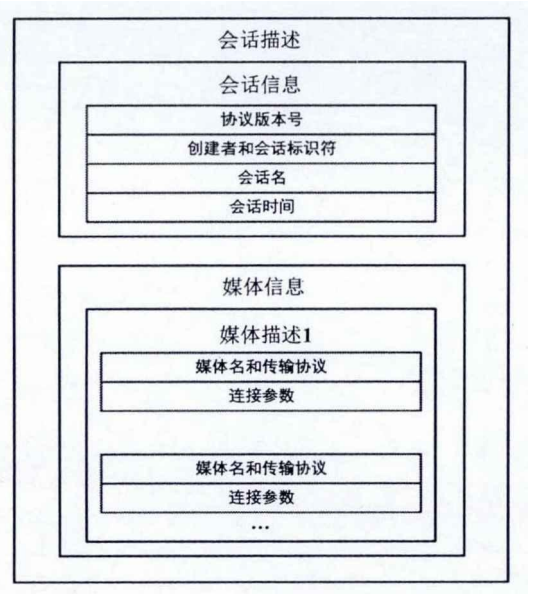

图 2-3 SDP 协议中的描述数据格式

Figure 2-3 Description data format in protocol

从图 2-3 中可以看出，描述数据格式中总共分为两部分信息：关于会话本身的信息和关于媒体数据的信息。本协议使用若干行数据表示会话本身的信息，一行对应一个属性，使用的是键值的形式。会话本身的描述信息先于媒体的描述信息。由含有 *m*= 的一行来引起一个媒体数据的描述信息，跟着的是关于媒体的具体的描述信息，而多个关于媒体数据的信息则使用相同的方式进行描述，一个 *m*=引起的行对应着一个媒体数据描述信息。

下面针对会话描述数据格式中的一些非常重要的字段进行阐述：

1 .协议版本字段，由 *v*=引起一行，是一个会话描述结构的开始标志。

2 .会话创建者以及会话的标识，由 *o*=引起一行，是对会话的标识信息的描述。

3. 会话的名称，由 *s*=引起一行，是在被通知的一方进行显示的字符串。

4 .会话的持续时间，由 *t* 引起一行，是用来描述会话的开始和结束的这一段时间的字段。

5. 媒体名称和传输地址，由 *m*=引起一行，用来描述音视频等数据流具体是哪一种类型，以及传输的音视频数据的网络方面的、使用的实时流式协议的类型以及音视频数据具体采用的编码器类型。其标志着媒体数据信息的开始，接下来的是具体的对于媒体信息的描述数据。

以上的这些格式单元是一定要填充的，此外，还有若干格式单元可供使用，如下的表 2-4 所示，其中以 *a*=引起的一行描述数据表示一个属性，在媒体数据的描述中较

为常见。

表 2-4 SDP 中的可供选用的字段

Table 2-4 Field available for selection in SDP

行首	字段对应的值
i=	会话信息
u=	URI
e=	Email
p=	联系号码
c=	连接信息
b=	带宽信息
r=	重复时间
k=	加密密钥
z=	时域调整
a=	属性

2.3 实时流式传输中的拥塞控制

音视频编码技术已经采用压缩技术对原始数据进行压缩，以方便在基于 IP 的网络中进行传输。实时流式传输协议在设计中也针对媒体数据的实时传输提供诸多保障。但是，网络具有不确定性，存在发生拥塞等的可能，为了保证即使在网络状况较差的情况下依然保证流媒体传输的服务质量，需要在流媒体数据的传输中结合拥塞控制策略来进行控制，减少拥塞。

针对流媒体数据在网络传输过程中出现的拥塞问题等的研究主要面向参与传输的双方。通过对接收方、发送方以及同时对两者进行拥塞控制，形成了三种常见的拥塞控制策略[34-35]。

对于接收方进行拥塞控制，就是在数据的接收过程中，由接收的一方控制速率。控制速率的方式是通过对通道的数量进行增减来实现的。这种控制策略的缺点很明显，

就是发送端没有参与控制的过程，所以，发送端对于网络的状况没有进行监视，也不会在网络状况较差时进行速率控制，此时，会导致网络拥塞问题更严重。

对于发送方进行拥塞控制，就是通过接收的一方对于发送的一方的数据进行统计，同时把统计的结果回传给发送的一方，供发送方用作进行发送速率调节的依据。这种方法可以灵活地根据网络状况进行速率控制，更好地适应带宽以及实时网络状况变化的应用场景。

同时对于发送方和接收方进行拥塞控制，就是将上述的两种方式结合起来。这种方式可以使得收发双方都能够及时有效地参与拥塞控制，对于实现网络状况的有效调控非常有用，但是该算法的实现复杂度相对高。

2.4 流媒体开源项目简介

2.4.1 Live555 流媒体解决方案

实时流式传输需要专用的流媒体服务器，其作为整个系统的核心，是流媒体传输技术中最为关键的一环。其主要通过 RTSP 等流媒体协议与客户端程序进行多媒体数据交互。客户端以特定的流媒体协议向流媒体服务器发出流媒体数据请求之后，服务器对请求作出响应，根据客户端请求参数查找到对应的资源等，开始将多媒体数据以实时流的形式向客户端发送。Live555 是一个基于事件驱动的流媒体传输解决方案，该项目使用 C++语言开发，支持实时流式传输协议。其具有精简的架构和良好的可移植性，易于通过交叉编译而用于多个平台，特别是嵌入式系统中。这个开源项目在开发之初未将高并发等大规模数据和大访问量的应用场景考虑进去，现阶段在工程应用中，仍存在一些不足和缺点，不少学者基于此对其进行二次开发来改善，使其更好地应用于流媒体系统中。

2.4.2 FFMPEG 开源项目

FFMPEG 是一个非常强大的，遵循 LGPL/GPL 协议的开源音视频编解码方案，基于 C 语言编写。作为一个优秀的开源编解码解决方案，这个开源项目提供了一个可以扩展的开源框架，其中包含几乎所有编码格式的编解码器。其中用于音视频编码数据的编解码等的程序库 avcodec 中有着十分成熟和高效的编解码器。其在工程中的运用大

大简化了用户的编解码过程，只需通过调用这个程序库提供的应用程序接口就能完成复杂的音视频编解码过程。大量的开源项目都是基于这个程序库的，如著名的 VLC 播放器，MPlayer 播放器等。整个 FFMPEG 开源库的各部分的框图如下图 2-4 所示，该项目提供了三个可以运行的功能强大的可运行命令行工具可满足用户对于音视频数据处理的需求，其中包括 ffmpeg 主要用于根据用户输入的命令行参数对音视频文件进行编码格式和封装格式等的转换；ffprobe 主要用于对于音视频文件进行格式分析；ffplay 是一个基于跨平台的 Simple DirectMedia Layer 库进行音视频数据播放的播放器工具。

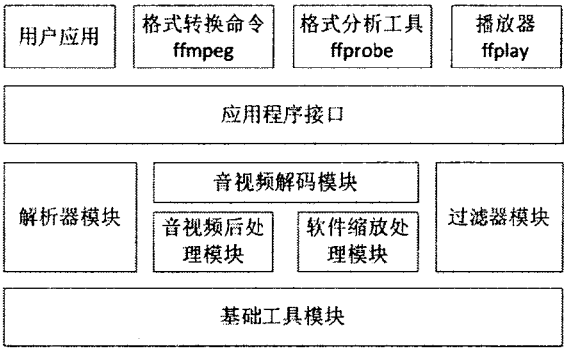

图 2-4 FFMPEG 整个开源库的组成框图

Figure 2-4 Composition of the entire open source library for FFMPEG

在 FFMPEG 的源码实现中，有一些非常重要的数据结构贯穿整个应用程序的编写过程，在上图中的应用程序接口中会经常用到。AVFormatContext 是对于媒体文件或者是来自网络的多媒体流的抽象，是在基于 FFMPEG 的编程中用得最多的数据结构。AVCodecContext 是对于编码器的相关信息的描述的结构体，其中包括了编解码器在编解码中会用到的很多相关的参数。AVStream 作为一个媒体数据流的抽象。AVCodec 是一个编解码器的抽象，在程序中通过调用特定的查找编码器的函数来找到编解码器，用这个结构体类型的指针指向一个具体的编解码器，用于编解码过程。AVPacket 和 AVFrame 都是用来存储音视频数据，而前者用于存储的是经过编码的数据，其中有很多非常有用的参数信息等，包括表示解码时间的解码时间戳，以及用于播放的显示时间戳等；后者存储的是经过解码的数据或者是原始数据。

多媒体文件中对于音视频数据的保存是通过先将原始的音视频数据编码后，按照一定的容器格式装进文件中。如图 2-5 所示，在 FFMPEG 对于多媒体文件的处理过程中，是先要通过对封装格式文件进行解封装，分别得到独立的音频和视频编码数据，在此基础上分别对编码的音频和视频数据进行解码，得到原始的音视频数据用于后续进行播放。

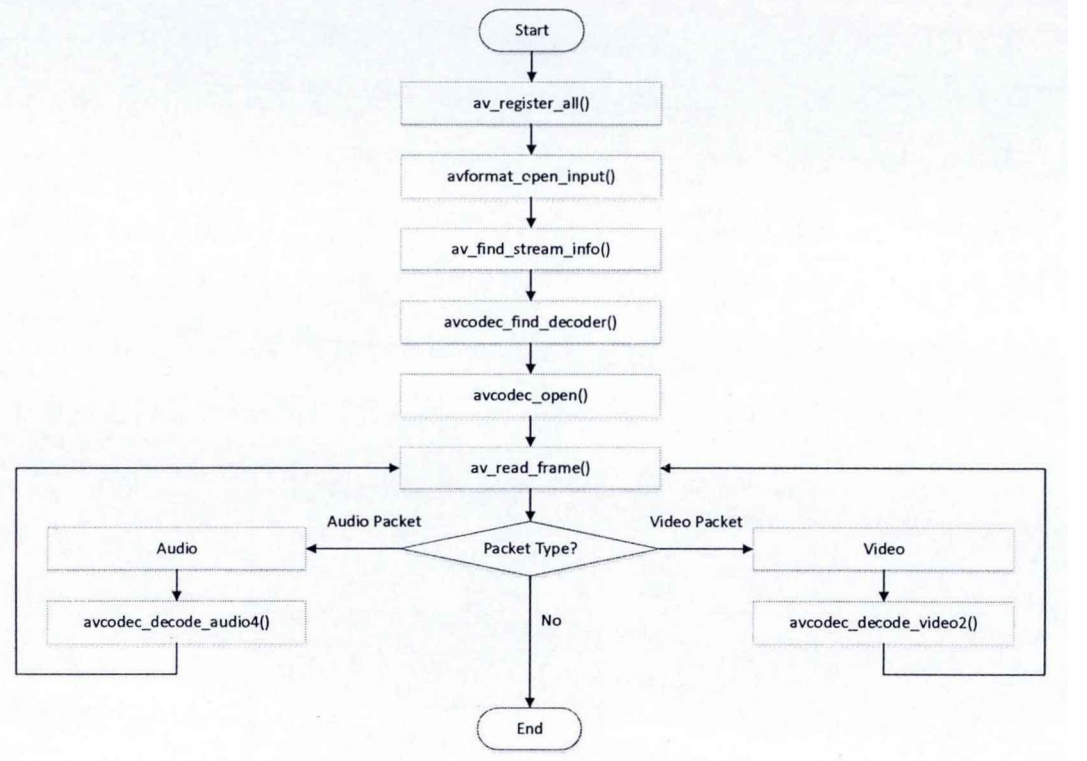

图 2-5 多媒体文件的播放过程图

Figure 2-5 Play process of multimedia file

2.5 本章小结

本章主要是对本文中所有涉及的相关技术内容概要描述。首先对于当前的流媒体技术现状进行概述，其中包括流媒体技术中常用的流媒体传输方式及各自的原理和特点以及流媒体技术的实现原理的介绍。对于工程实践中运用最多的流媒体传输控制协议进行简要地阐述。针对流媒体传输过程中存在的网络状况差等情况，引入了网络拥塞控制算法。最后，对于文中用到的两个著名的开源流媒体项目，分别做了介绍。

第三章 音视频同步技术研究

音视频同步对于服务质量非常关键[36]。在本文设计的家庭智能服务机器人应用场景下的流媒体传输系统中，为了保证音频和视频的同步效果，需要在整个过程中分别进行同步控制。本章在针对多媒体同步的概念进行简单地介绍的基础上，分析音视频失同步现象引起的原因。针对多媒体数据从采集到发送的过程进行改进，设计了一种贯穿于系统各个环节的音频和视频同步方案。

3.1 音视频同步的主客观评价标准

根据 ITU-R BT.1359-1[37]的内容,人类生物学视听系统对于声音和视频的不同步的察觉程度是在一个具体的范围内的。这是人类的视听系统的固有特性，在进行音视频同步的过程中，要参考这一个特性。公式（3.1）中是声音的呈现时间 T_{sound} 与视像的呈现时间 T_{vision} 的差值，根据文献中的内容，当这个差值为正值时，表示音频比视频要超前；当这个差值为负值时，表示音频后于视频。根据文献中的内容可知，人类的视听系统对于音频超前于视频的感知能力要强于音频落后视频的感知能力。

当 ΔT 属于[-90ms,20ms]的区间内时，这个区间内是音视频同步效果最好的，人类能感受到的是完全的音视频同步。

当 ΔT 属于[-125ms,-90ms]以及[20ms,45ms]的区间内时，人类能够轻微地察觉到音视频之间存在不明显的失同步现象。

当 ΔT 属于[-185ms,-125ms]以及[45ms,90ms]的区间内时，人类已经完全能够感受到音视频失同步，但依然处在能够接受的范围，不至于影响用户体验.

在以上的区间以外就完全是不可以接受的失同步区间，如果 ΔT 落在这些失同步区间中，用户将感受到严重的音视频失同步现象，在对多媒体系统进行开发和设计的过程中必须考虑这个指标。

$$\Delta T = T_{sound} - T_{vision} \tag{3.1}$$

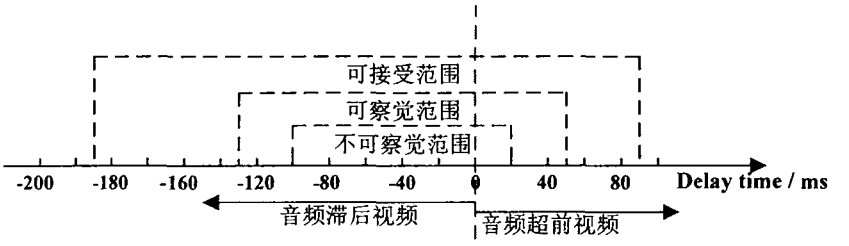

图 3-1 人类生物学声音视像系统对音视频同步的感知

Figure 3-1 Human biology voice video system for perception of audio and video synchronization

音视频同步性能的客观评价准则通常是使用同步相位失真 SPD（Synchronization Phase Distortion）这个数学量来进行[38]。其数学表达式如下：

$$D_{av} = (P_a(i) - P_v(j)) - (G_a(i) - G_v(j)) \tag{3.2}$$

公式中的 D_{av} 就是音频和视频流中的两个在时间上具有强相关性，也就是最邻近的帧 i 和 j 之间的同步相位失真 SPD，$P_a(i)$和 $G_a(i)$分别是音频流中第 i 帧数据单元的播放时刻和产生时刻，相应地，$P_v(j)$和 $G_v(j)$分别是视频流中第 j 帧数据单元的播放时刻和产生时刻。SPD 从客观的角度反映了在时间上具有相关性的音频和视频数据之间的失同步程度。在本文的实验中将以这个数学量来作为客观的音视频同步评价标准。

3.2 音视频失同步原因分析

视频监控系统从前端摄像机等采集到原始音视频数据，音频数据为 PCM 格式，视频数据为 YUV 数据格式，通过编码器进行编码之后，使用网络协议进行数据封装打包以方便在 IP 网络上实现传输，经过网络传输而被客户端接收，客户端接收完成后通过对数据包进行一系列的逆操作，如解封装、解码等将媒体数据还原成为原始数据格式，最终呈现给用户。在整个过程中，每一个环节都会对音视频的同步产生影响。

在音视频数据采集过程中，由于程序是顺序执行的，而且操作系统对于进程和线程是异步调度执行，因此，不可能做到音频和视频的采集时间完全相同。使得音频数据和视频数据的对应关系在前端采集过程中存在固有的偏差；而在接下来的音视频编码过程中，由于一帧视频数据相对于一帧音频数据要大很多，因此在相同的计算机主频下，对视频进行编码的时间相对于一个音频采样的编码时间要长很多，如图 3-2 所示。

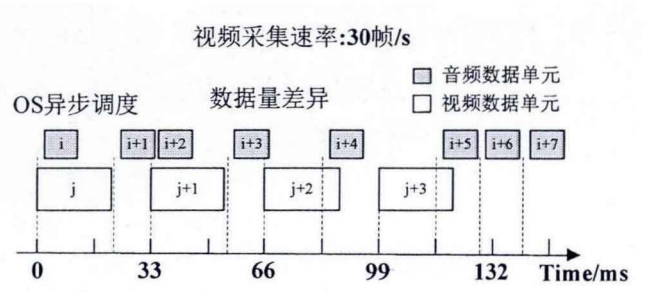

图 3-2 音视频数据采集过程的不同步

Figure 3-2 Non Synchronization in capturing raw data

编码数据在发送过程中要按照 RTP 协议的格式来进行打包，在打包的过程中，由于编码后的一帧视频数据长度仍然远大于一帧编码后的音频采样数据。因此，在将编码数据封装进 RTP 包的过程中，这个长度差异同样地会引起音频和视频数据的失同步。

在音视频数据进行传输的过程中，由于 RTP 协议是一种尽最大努力交付的协议，不具备可靠传输的特点，受网络状况影响较大，当网络状况比较拥堵时，数据包存在延时、丢失、乱序到达等可能性。而对于音视频数据，一旦出现丢包的问题，将可能影响到编码数据的完整性，特别是当 H.264 的数据单元超过了 MTU 的大小时，要采用分包发送，此时将会引起视频编码数据的不完整，在解码过程中将出错，这也会引起失同步问题。如图 3-3 所示，为不同网络状况下，网络传输时延的变化，引起了数据包接收到的间隔时间不一致，而这会引起不同步的问题。

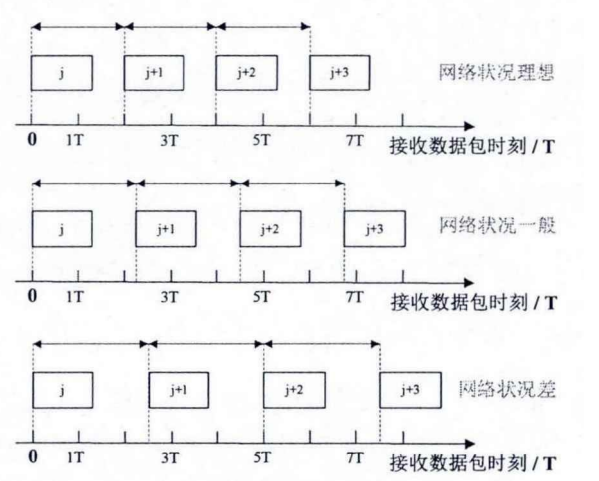

图 3-3 不同网络状况下的时延

Figure3-3 Network delay in different condition

在客户端接收流媒体数据后，进行网络传输格式的解封装，在对协议进行解封装之后，提取出音视频的编码数据。通过调用解码器，按照发送端对应的编码算法进行相应的解码，解码完成即可得出原始音视频数据。在这个过程中，同样由于视频数据

量较大，而使得解封装和解码的时间与音频的不相同。应用程序最后进行音视频的播放过程中，如果不协调控制好音频数据和视频数据的时间戳而直接予以播放，那么也将引起严重的失同步问题。

3.3 基于网络延时检测的自适应音视频同步方案

分析上一节的音视频失同步原因，可以总结为两大类：音视频数据处理引起的时延和网络时延。音视频数据处理引起的时延是由音视频的数据量不等以及对应编码算法引起的时间差，来自于音视频系统本身固有的属性，是固定的可计算的，可以通过估算以及修正等方法解决；网络时延则由网络环境等复杂因素影响，来自于音视频系统以外的随机性因素的影响，具有不可预测性。常见的做法是基于实时网络状况，调节多媒体数据的发送速率。

不同的音视频同步算法有不同的特点，在不同的网络状况下选择更适合当前状况的音视频同步算法，可以获得更好的服务质量。

为了解决网络延时对于传输的影响，需要使用网络延时检测算法进行网络状况的实时监测，并对网络状况进行评级，为传输系统音视频同步算法的选择提供理论依据。

3.3.1 实时网络状况检测

网络时延具有偶然性和突发性，将延时检测点的网络状况作为当前的网络状况的做法带来的误差较大，不能正确地反映实时的网络状况，同时这种情况下作出的同步方案决策也是不准确的，需要对于网络时延进行总体把握。

选定一个时间间隔 T，将其分为 n 等分，每一个时间间隔为 $1/n$。在每一个等分都进行一次网络时延的检测，检测得到的当前传输时延记为 $d_i(1 \leq i \leq n)$，通过计算当前传输时延的方差(公式 3.3 和 3.4)，可以得出当前时间间隔 T 内的时延变化率。

$$s^2 = (1/n)\sum_{i=1}^{n}(di - \bar{d})^2 \tag{3.3}$$

$$\bar{d} = (1/n)\sum_{i=1}^{n} di \tag{3.4}$$

根据方差的数学意义可知 s^2 仅表示网络状况的时延波动大小，仅仅是对其变化率的反映。因此，结合传输过程中用到的 RTCP 协议中 Receiver Report 报文中的两个值：

控制信息的丢包率 *fraction_lost* 和包间抖动 *interarrival_jitter* 的值。RTCP 协议自动每隔一段时间发一次，通过结合 RTCP 的 RR 报文，对网络状况进行如下的分类：

首先，定义 S_{min}^2，S_{max}^2 为网络时延波动判断的下阈值和上阈值：

当网络延时的方差 $S^2>=S_{max}^2$ 的时候，网络延时波动较大，此时，网络状况直接判定为很不稳定，即网络状况较差；

当网络延时的方差 $S^2<=S_{min}^2$ 的时候，网络延时波动相对小，此时，需要结合 RR 报文进行进一步判断；

当网络延时的方差 $S_{max}^2<=S^2<=S_{max}^2$ 处于两者之间的时候，网络时延波动相对稳定，此时，可以直接判定为一般。

接着，对 $S^2<=S_{min}^2$ 进行进一步细分：

（1）$S^2<=S_{min}^2$，且 *fraction_lost* < *threashhold_fraction_lost*，*interarrival_jitter* < *threashhold _interarrival_jitter*，判定当前网络状况为良好；

（2）$S^2<=S_{min}^2$，且 *fraction_lost* >= *threashhold_fraction_lost*，*interarrival_jitter* < *threashhold _interarrival_jitter*，判定当前网络状况为一般；

（3）$S^2<=S_{min}^2$，且 *fraction_lost* >= *threashhold_fraction_lost*，*interarrival_jitter* >= *threashhold _interarrival_jitter*，判定当前网络状况为较差。

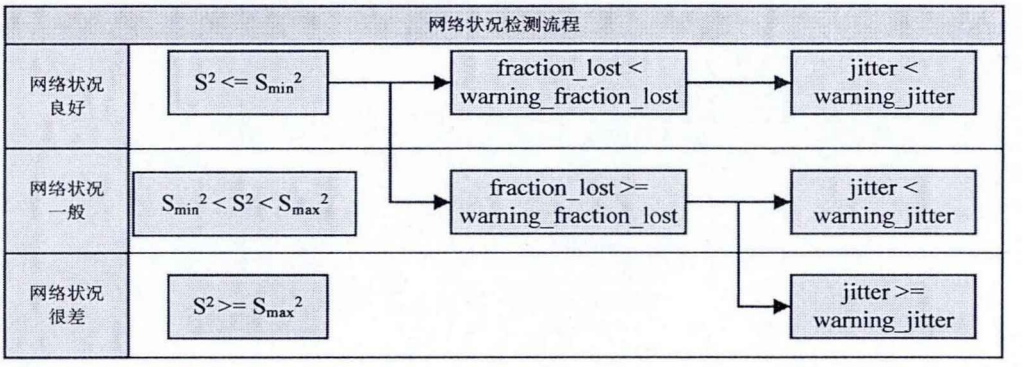

图 3-4 网络状况检测流程图

Fig. 3-4 Network status testing

3.3.2 高延迟网络下的音视频同步技术研究

在高延迟网络下进行音视频间的同步，需要针对音视频数据处理的每一个环节进行同步控制。在整个传输与转发的过程中，对同步有影响的部分包括前端摄像机的采集模块和编码打包模块，客户端的解码和播放模块，以及在基于 IP 网络中实现音视频数据的流式传输的收发模块。因此，抽取出其中最关键部分并分别予以同步控制，如

图 3-5 所示。

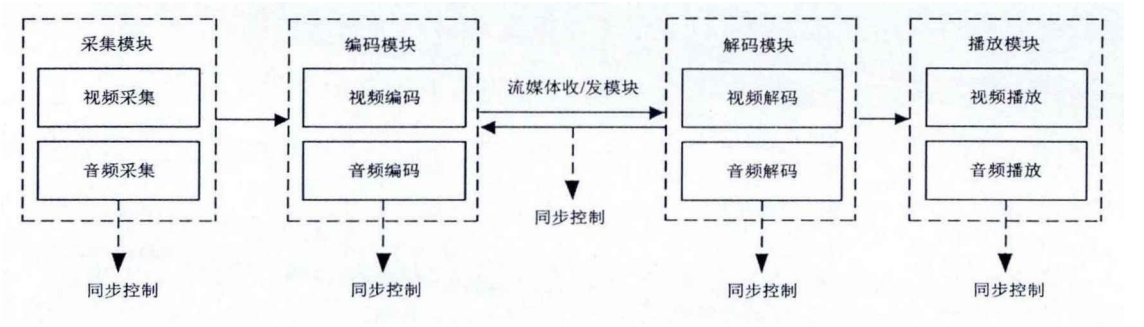

图 3-5 音视频同步传输系统框图

Figure 3-5 Block diagram of synchronous transmission of audio and video

1 采集模块

在嵌入式 Linux 开发板上进行音视频数据的采集过程中，由操作系统的原理可以知道，内核调度执行线程运行具有异步性，音频和视频数据的采集过程不可能真正做到严格意义上的同步。另一方面，摄像头对于视频数据的采集是一次采集一帧，对音频数据的采样则是通过使用 ioctl 编程设置采样的间隔时间来进行度量的。以采集一帧分辨率为 320X240 的 YUV420 格式的视频数据和设置每秒采样 0.1s 的音频数据为例，其持续时间很难完全一致，这就使得两个采集过程不会有严格意义上的相同的开始和结束时间。

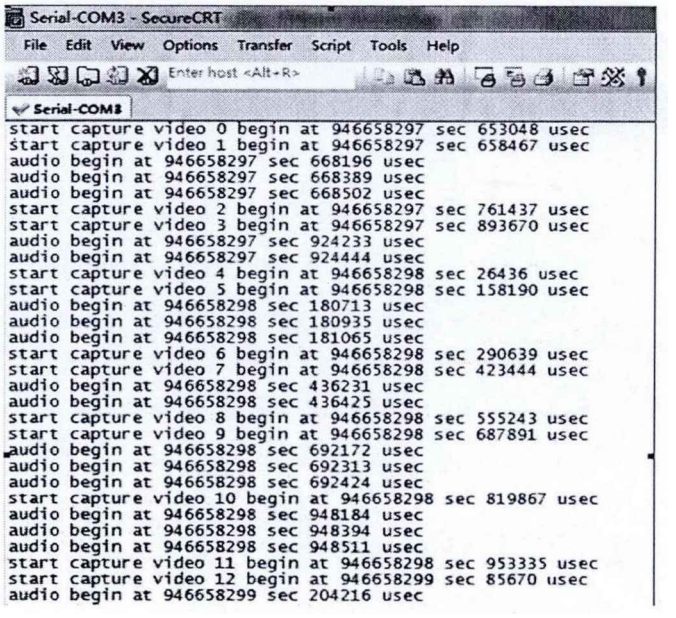

图 3-6 使用 gettimeofday 获取的时间戳

Figure 3-6 Time stamp gotten by using gettimeofday

为了对采集时间进行分析，在程序中使用 gettimeofday 函数获取当前开始采集一个音视频数据单元的时刻，通过计算出当前的微秒数之后，以这个微秒数作为音视频数

据采集的时间戳。为了方便观察，将一次音视频数据的采集时间中的秒和微妙均打印出来如图 3-6 所示。从图中可以看出，音频和视频数据的采集过程是完全异步的，各自在时间上不具备特定的规律，而且在音频和视频的采集时间上几乎没有开始时间严格相同的。

而从图 3-1 得出的结论是，如果想要不破坏用户对于音视频同步的主观感受，必须保证 ΔT 不会落在失同步区间内，那么必须要保证其始终落在一个固定的区间内[-125ms,45ms]。这个区间可以为音视频同步控制提供理论基础。对图 3-6 的捕捉时间进行分析，如视频帧 video0 先于第一个音频采样数据 15ms，按照同步区间的定义可知，当音频超前视频 15ms 的时候，如果在同一时刻播放这两个数据，显然是不会感觉到任何失同步。这样可以保证同步数据节点内的音视频数据本身具有很强的时间相关性。基于此将 video0 的采集时间戳作为当前同步数据节点的时间戳，并将系统中采集的时间戳处在同步时间区间内的还没有存入任何同步节点中的音频和视频数据存入到当前同步数据节点中。

将采集到的符合上述规则的音视频数据存入如图 3-7 所示的同步数据节点中，以节点内第一帧视频的采集时间作为该同步数据节点的时间戳。再将同步数据节点插入到原始采集数据的同步节点队列中暂存起来，等待后续的软件模块对采集的音视频数据进行提取和音视频编码。

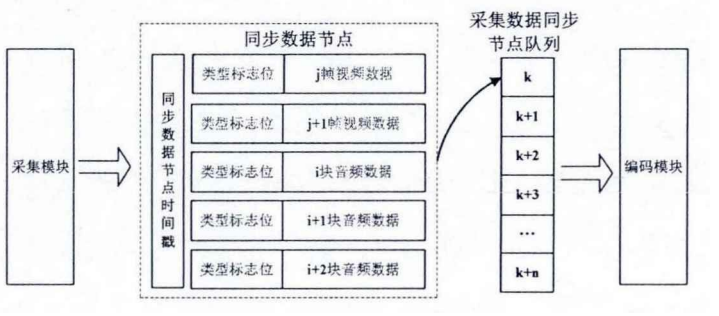

图 3-7 原始音视频数据的同步数据拼装过程

Figure 3-7 Synchronous data assembling process of the original audio and video data

2 编码模块

在上一节中，实时采集的音频和视频原始数据已经按照特定的同步数据节点的格式进行封装，同时插入队列中进行缓存。对于音频数据和视频数据的编码算法是不相同的，因此，需要从采集数据同步节点队列中提取出缓存的音频和视频原始数据，分别进行编码。在对一个同步节点进行拆分之前，保存下这个节点的时间戳，在拆分的过程中主要判断音视频数据类型标志字段，根据其原始数据类型为其进行对应的音视

频编码：当数据是 PCM 格式的音频数据时，进行 G.729 的音频编码；当数据是 YUV 格式的视频数据时，进行 H.264 标准的视频编码。

此时，按照对拆分同步数据节点之前保存的时间戳进行编码数据的封装。如图 3-8 中所示，将音视频数据再次以同步数据节点的形式填充好后，插入到编码数据同步节点队列中，进行缓存，以供下一个模块进行提取和访问。

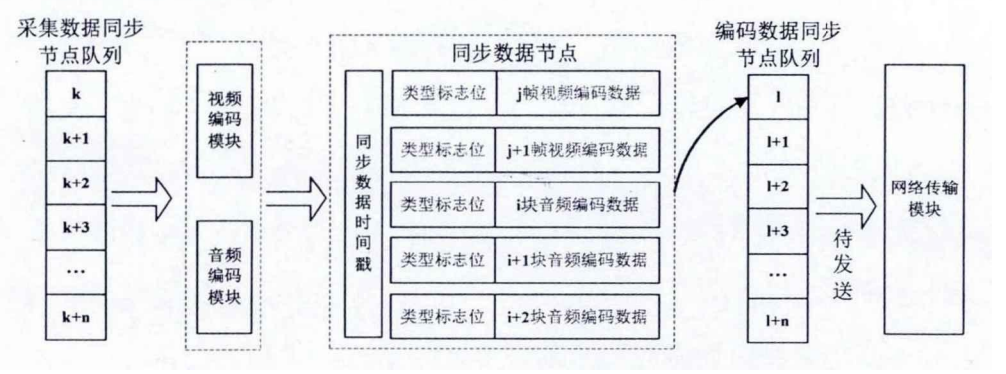

图 3-8 对同步数据节点进行音视频编码的过程

Figure 3-8 Process of audio and video coding for synchronous data nodes

3 传输模块

基于 IP 网络的实时流式传输模块的实时传输模块主要分为发送方和接收方两部分。发送方主要完成从编码数据同步节点队列中取得同步数据节点，再从节点中提取音视频编码数据，分别对音视频数据进行 RTP 打包。

由于在以太网进行数据传输存在 MTU 限制，而一帧视频数据可能超过这个限制，因此，需要对于这种情况进行处理。如果 RTP 包超过了 MTU 的限制，那么，将这个 RTP 包分为若干个子包，同时，在这些子包中均存有同步数据结构中的时间戳字段 $T_{syncblock}$，以及 RTP 时间戳字段 T_{tsrtp}。在后续接收端将通过 $T_{syncblock}$ 字段来判断子包属于哪个同步数据节点。而一个同步数据节点中存在多个视频帧，这里要用到 RTP 包头的时标 *Sequence Number* 字段。*Sequence Number* 字段指明该包中第一个三位二进制数据的采样时间。

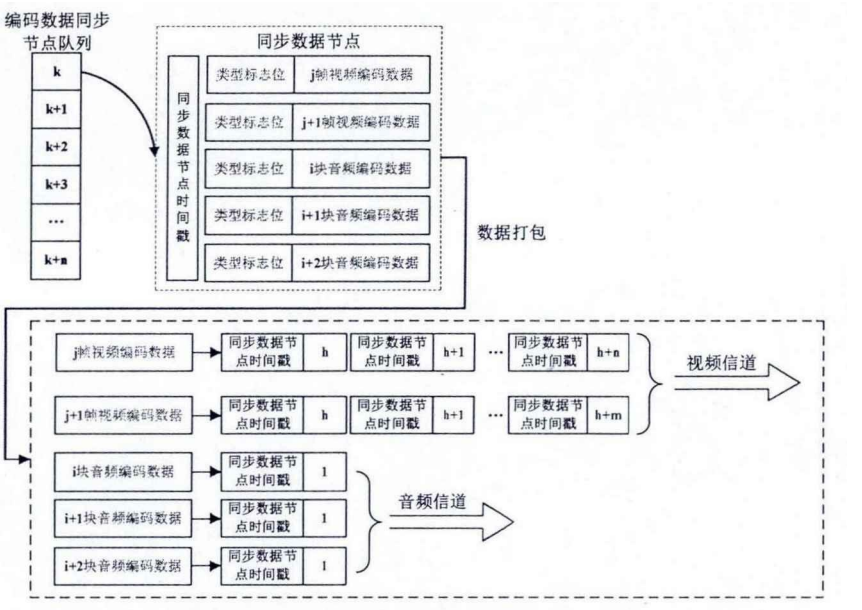

图 3-9 对编码数据进行 RTP 打包过程

Figure 3-9 RTP packaging process for coding data

传输模块中的接收部分主要完成的是将接收到的 RTP 数据包在接收传输模块中重新还原为同步数据节点的组织形式，并最终插入到位于接收方的传输模块中的编码数据同步节点队列中。

当 RTP 包到达接收方，解码模块从接收缓冲区中取出同步结构体。从得到的数据中提取出三元组<$T_{syncblock}$,T_{tsrtp},$N_{SequnceNumber}$>，$T_{syncblock}$ 表明该 RTP 包从属的同步结构体，T_{tsrtp} 表明这个包的帧归属，也就是说如果是一个子包的话，子包将通过这个时间戳能定位到自己属于哪一帧，$N_{SequnceNumber}$ 确定属于同一帧内的不同的子包之间的顺序。此时，判定该包是否为子包或者为一个完整包。如果是一个完整的包，那么直接将其中的数据插入到如图 3 所示的同步数据节点中；如果是一个子包，那么根据 $N_{SequnceNumber}$ 依次将子包中的数据插入到链表中，其拼接成同步数据节点的过程如下图 3-10 所示。

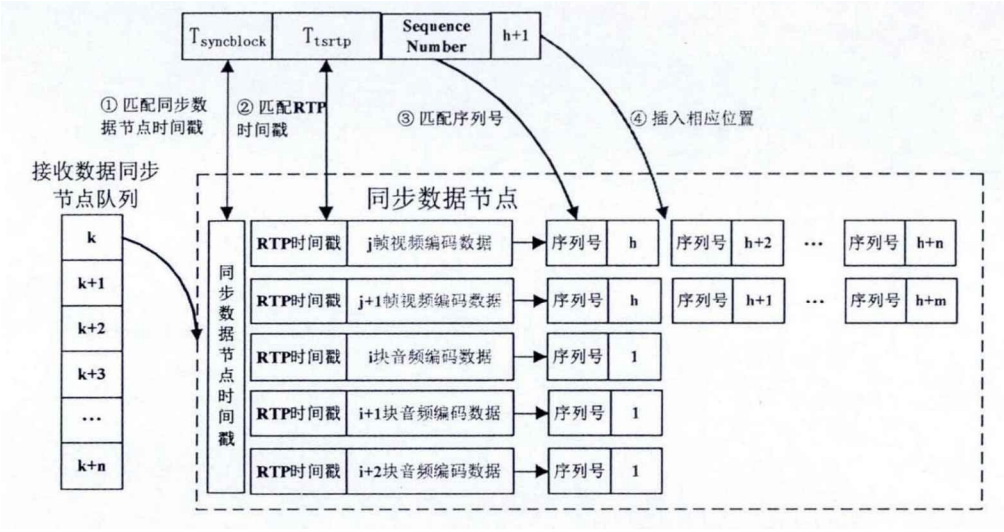

图 3-10 填充同步数据节点的过程

Figure 3-10 Process of filling the synchronous data nodes

在图 3-10 中，显然在接收数据同步节点队列中的数据中都存在传输过程中的控制信息，如 RTP 时间戳 $T_{syncblock}$ 和 *Sequence Number* 字段，而这些信息并非有效的编码数据信息，而且在后续的处理环节中也不再需要，因此在这一个环节中就可以直接将这些传输控制信息去掉，方便后续操作，清理的过程如图 3-11 所示。

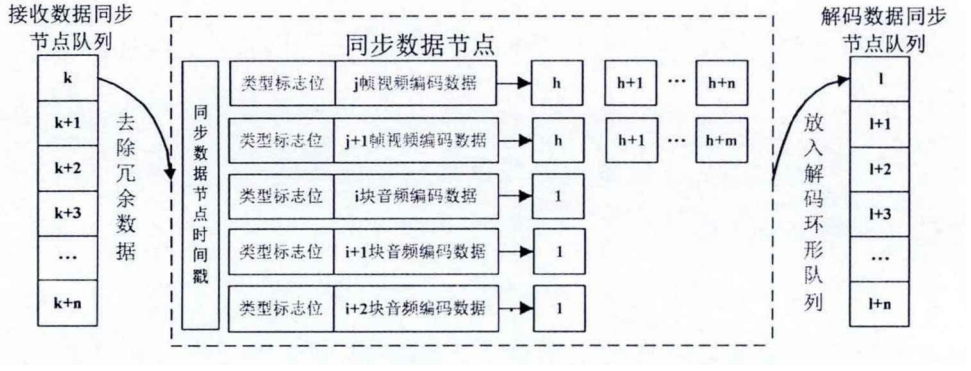

图 3-11 对于同步数据节点中传输控制信息的清理

Figure 3-11 Cleaning of the transmission control information in the synchronous data nodes

4 解码模块

接收端在收到由服务端经 IP 网络传输过来的数据，并经接收模块对数据完成了预处理之后，数据已经缓存在本地的编码数据队列中。该模块互斥访问编码数据同步节点队列中的节点，同样将音视频数据区分开后，分别进行对应的音视频解码。解码出来的数据已经是设备可以直接播放的 PCM 和 YUV 数据。如图 3-12 所示将解码后的音视频原始数据重新按照播放数据同步数据节点的格式进行组装，并插入播放数据同步节点队列中，等待后续的模块进行访问和提取。

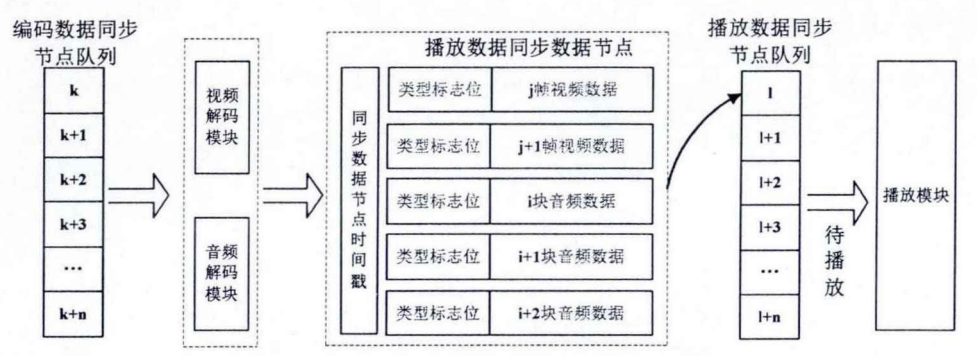

图 3-12 对同步数据节点进行解码的过程

Figure 3-12 Process for decoding synchronous data nodes

5 播放模块

收到的网络数据经过前面的两个阶段已经完成解协议和解码的过程，可以播放的音视频数据缓存在播放数据同步节点队列中。播放模块需要从播放数据节点队列中取出节点并且提取出音视频数据，采用同步视频到音频的方式，按时间顺序分别对数据进行播放，其播放过程如图 3-13 所示。

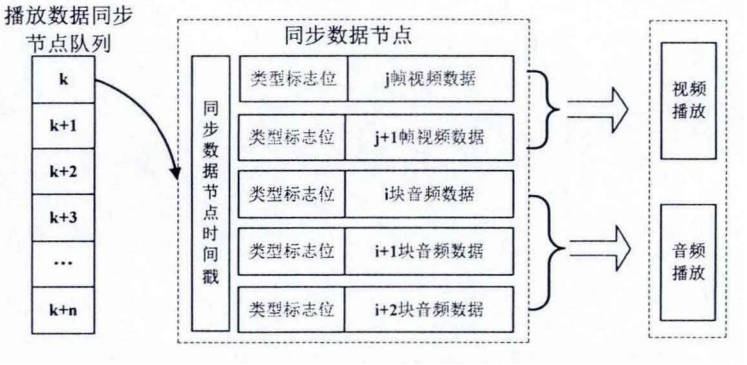

图 3-13 对解码后的同步数据节点进行实时播放的过程

Figure 3-13 Process of real time play of synchronous data nodes after decoding

3.3.3 低延迟网络下的音视频同步技术研究

在低延迟网络下，音视频流是在各自的信道上，采用 RTP 协议按照一定的速率传输。低延迟网络中对于音视频同步主要考虑的问题不再是网络延迟的影响，而主要考虑在接收端基于音视频数据在采集编码过程中的时间戳进行同步。

RTP 的头部中有一个 32 位的时间戳字段，其表示当前数据报有效载荷中的第一个字节的采样时间，且为线性单调递增。音视频数据 RTP 时间戳以及世界时间的关联关系如图 3-11 所示。不同 RTP 会话中的 RTP 包的时间戳不能直接比较，而通过 RTP 包中的时间戳和 RTCP 的 SR 报文中的 NTP 时间戳进行关联则可以将媒体流中的 RTP 时

间戳字段对应到 NTP 时钟上。

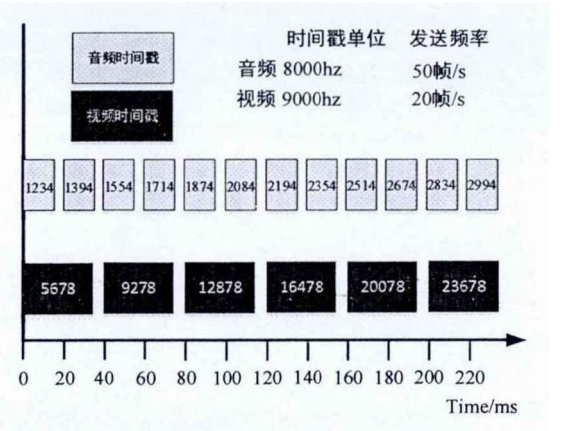

图 3-11 音视频 RTP 时间戳关联

Figure3-11 Maping of timestamp of audio and video

由 RTP 协议可知第 i 帧多媒体数据的时间戳为第 i-1 帧多媒体数据的时间戳与一帧多媒体数据的采样点数量，如式 3.5 所示。

$$\mathrm{Tstamp}(i) = Tstamp(i-1) + \mathrm{Sample}N \tag{3.5}$$

其中一帧多媒体数据的采样点数量计算公式如 3.6 和 3.7 所示。

$$\mathrm{Sample}N = f \times T \tag{3.6}$$

$$T = \frac{1}{\mathrm{FrameRate}} \tag{3.7}$$

其中的 f 是多媒体数据的采集系统的采集时钟频率，T 为帧率的倒数，就是每一帧的播放延时，对应为当前播放的多媒体数据的帧率。综合 3.5~3.7 式可得：

$$Tstamp(i) = Tstamp(i-1) + \frac{1}{\mathrm{FrameRate}} \tag{3.8}$$

$$\Delta TS\mathrm{inc} = \frac{f}{\mathrm{FrameRate}} = T \times f \tag{3.9}$$

由公式 3.9 结合文献中的时间戳映射模型[39],可以得到公式 3.10 和 3.11。公式 3.11 中的$\Delta TS\mathrm{inc}$表示两个时间上靠近的 RTP 包中的时间戳的增量。

$$NTPts_i(\mathrm{j}) = NTPts_i(1) + \sum_{k=2}^{j} 2\frac{\Delta ts\mathrm{Inc}_i(k)}{f} \tag{3.10}$$

$$\Delta TS\mathrm{inc}_i(k) = \mathrm{T}s_i(k) - Ts_i(k-1) \tag{3.11}$$

其中的$\mathrm{T}s_i(k)$表示第 i 个 RTCP 的 SR 报文发出后，第 k 个媒体数据包的 RTP 时间戳。$NTP_Ts_i(\mathrm{j})$为与之相关联的 NTP 绝对时钟时间。其中的f是发送端的时钟频率。

再由公式 3.10 和公式 3.11 可以得出：

$$NTP\_Ts_i(1) = NTP\_Ts_i(0) + \frac{Ts_i(1)-Ts_i(0)}{f} \tag{3.12}$$

将公式 3.11 和 3.12 代入 3.10 中可以得到如下的公式 3.13：

$$NTP\_Ts_i(j) = NTP\_Ts_i(0) + \frac{Ts_i(1)-Ts_1(0)}{f} \tag{3.13}$$

当发送端编码速率是个常量，此时，编码后的时间戳呈单调线性增长，有如下的公式 3.14:

$$f = \frac{Ts_i(0)-Ts_1(0)}{NTP\_Ts_i(0)-NTP\_Ts_1(0)} \tag{3.14}$$

则由 3.13 和 3.14 可以得出如下的公式 3.15，即为其 NTP 绝对时间戳计算公式：

$$NTP\_Ts_i(j) = NTP\_Ts_i(0) + \frac{Ts_i(j)-Ts_1(0)}{f} \tag{3.15}$$

再由公式 3.15，即可实现将音视频包的 RTP 时间戳重新映射到发送端的系统时钟。因此，其可以直接利用发送端系统时钟来代替全网同步时钟。

接收端通过将视频流同步到音频流的方式来实现音视频同步。视频流的播放过程中将当前音频播放时钟作为参考时钟，通过本节中的时间戳映射公式进行时间戳映射，使用如下的音视频同步算法进行同步播放过程，其中定义 T_v 为视频当前的播放时间戳，定义 T_{ref} 为当前的参考时间戳，定义 $Diff=T_v-T_{ref}$，定义 *ThreshHold* 为一个正值且为同步区间的上限，则-*ThreshHold* 为同步区间的下限。

当-*ThreshHold*<=*Diff*<=*ThreshHold* 时，视频同步到音频的播放过程，视频进行正常播放即可。

当 *ThreshHold*<*Diff* 时，视频超前于音频的播放过程，暂时不播放该早到的视频帧。

当 *Diff*<-*ThreshHold* 时，视频落后于音频的播放过程，直接丢弃当前视频帧以使后续视频跟上当前播放进度。

3.4 算法实验与分析

要对文中提出的基于网络延时检测的自适应音视频同步方案进行验证，首先要对其中的网络延时检测算法进行实验分析，得出其对于网络状况的判定是否准确有效。其次，对于文中提出的高延迟网络下的音视频同步算法的同步效果进行分析。而低延迟网络下的音视频同步算法则在大量文献中经过了实践的检验，本文不再单独对其进

行仿真实验。

3.4.1 网络延时检测算法实验

网络延时的检测实验需要在网络状况可控制的条件下进行，本文通过在计算机上同时进行网络数据下载来模拟网络状况变化，进而控制网络延时。在未进行数据下载的情形下设置 100 次测试点，结果如图 3-12 所示。经计算其方差为 S^2=101.3。

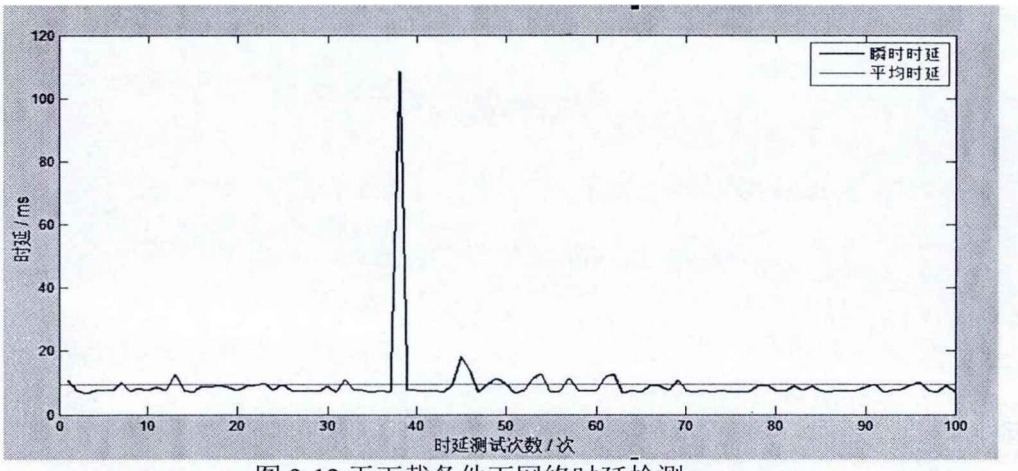

图 3-12 无下载条件下网络时延检测

Figure 3-12 Time delay detection in no congestion condition

通过控制下载来模拟一般情形下的拥塞程度，同样设置 100 次测试点，结果如图 3-13 所示。经计算其方差为 S^2=1663.3，平均时延为 33.1ms。由 RR 报文得到的丢包率为 19.23%，到达间隔抖动保持在[1,9ms]的区间内。

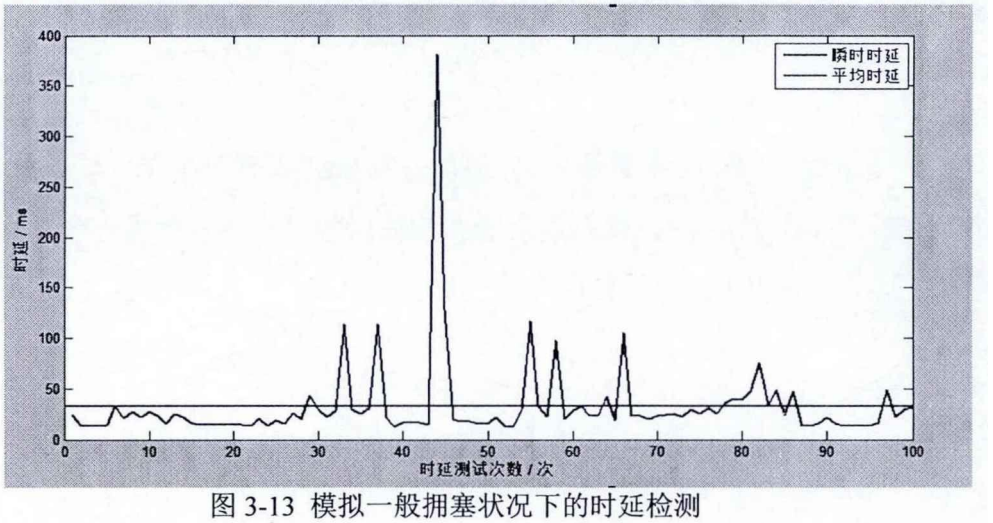

图 3-13 模拟一般拥塞状况下的时延检测

Figure 3-13 Time delay detection in a normal congestion condition

通过控制下载来模拟较差状况的网络，同样设置 100 次测试点，结果如图 3-14 所示。经计算其方差为 S^2=1642.1。由 RR 报文得到的丢包率为 42.19%，到达间隔抖动保

持在[1,8us]的区间内。

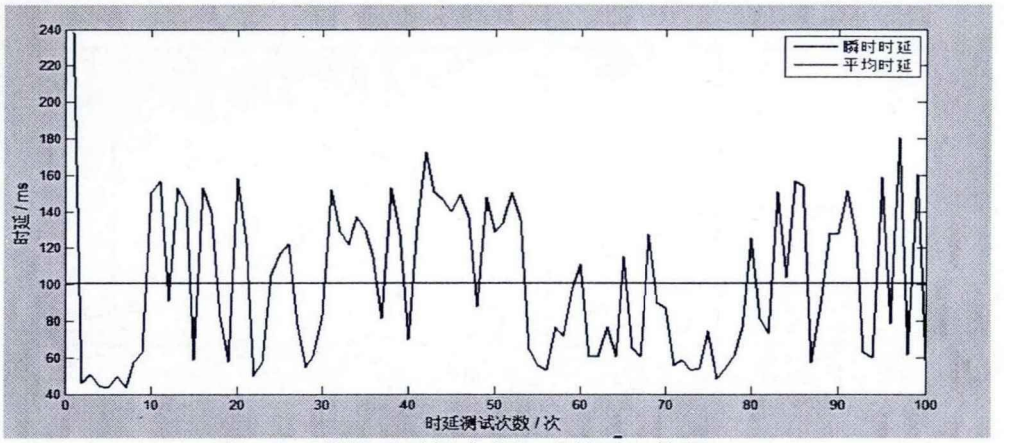

图 3-14 模拟严重拥塞状况下的时延检测

Figure 3-14 Time delay detection in a severe congestion condition

通过多次测试可以得出算法中的四个阈值参数大致的值为 S^2 的两个上下限为[1000,2500]，*threashhold_fraction_lost*=15%，*threashhold _interarrival_jitter*=40ms。但这四个参数仅对于文中的软硬件系统适用。当前通过实验得出的数据，仍然不能用于更多情形的网络条件，今后的研究中将会针对系统的自适应阈值计算进行深入。

3.4.2 高延迟网络下音视频同步算法实验

为了对文中提出的高延迟网络下的音视频同步控制算法进行仿真实验，搭建了如表 3-1 所示的实验环境，并通过在实验环境中进行网络数据传输来模拟高延迟网络状况。

表 3-1 实验环境

Table.3-1 Experimental environment

软件	版本号	系统主频
FFMPEG	2.8.1	
Android 终端	5.1	四核 1.3GHz
UBUNTU 桌面系统	14.04	Intel I7 四核 2.4GHz

在实验过程中进行两组实验，在 PC 端通过 Linux 下的 Video For Linux 框架采集视频，设置视频的分辨率，经 FFMPEG 指定进行 H.264 编码并设置视频的码率。并进行音频采样，采样后同样经 FFMPEG 用 G.729 编码器编码并指定码率。其中两组实验参数如表 3-2 所示。

表 3-2 两组实验参数

Table.3-2 Experimental environment

编号	视频码率	视频帧率	视频分辨率	音频码率	音频帧率	音频分辨率
1	500kb/s	25fps	320 X 240	16kbps	100fps	8KHz-16bit
2	400kb/s	30fps	640 X 480	16kbps	100fps	8KHz-16bit

按照表 3-2 中的参数对两组实验中采集的 120 帧视频和 80 帧音频数据用文中方法进行传输。同时，为了验证文中方法对于整个过程的同步控制效果，使用一组仅在播放过程中进行音视频同步控制的实验数据作为参考。

如图 3-15 所示，受文中分析的多个导致失同步的因素的影响，在仅对播放过程采用时间戳进行简单的同步控制的这一组中，SPD 单帧存在失同步情况明显，有不少次甚至接近 200ms，失同步幅度较大；图 3-16 中是用 L Bertoglio 的方法[39]得到的实验结果，其中存在 SPD 超过 100ms 的情况，而文中提出的方法的实验结果如图 3-17 和图 3-18 所示，能很好地控制在[-40ms,+40ms]区间内，而且较为稳定，经计算分析得出 SPD 平均值相比于仅在播放时使用简单的同步控制的方式下的平均值要减小 42.5731ms，薛彬等[40]提出的改进方法的实验结果为 SPD 平均减小 30.1972ms，如图 3-19 所示，相比之下本文的同步方法更优，同时，本文的 SPD 的整体值的波动范围也要比其更小。

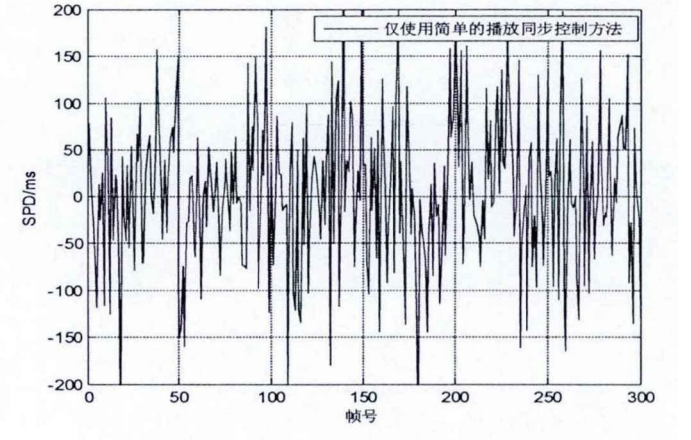

图 3-15 仅使用播放同步控制方法的 SPD

Figure 3-15 The SPD Of Simple Synchronization Method

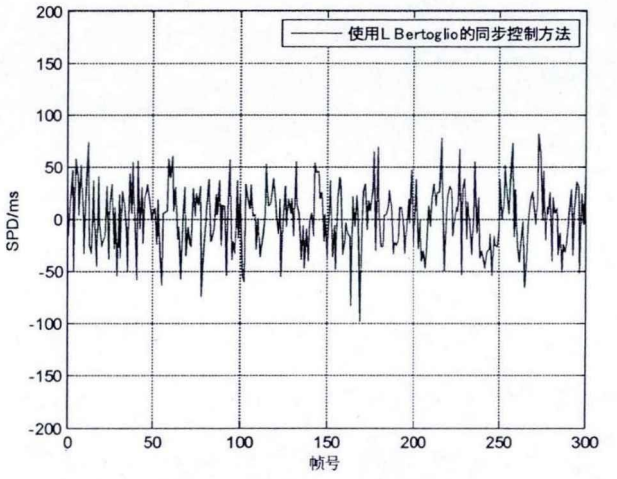

图 3-16 L Bertoglio 方法的 SPD

Fig. 3-16 The SPD of L Bertoglio Method

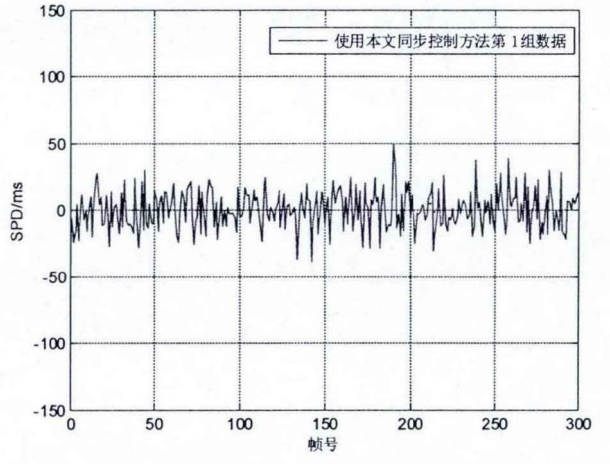

图 3-17 使用本文同步控制方法的第一组 SPD

Fig. 3-17 The SPD Of The Method Proposed By This Paper

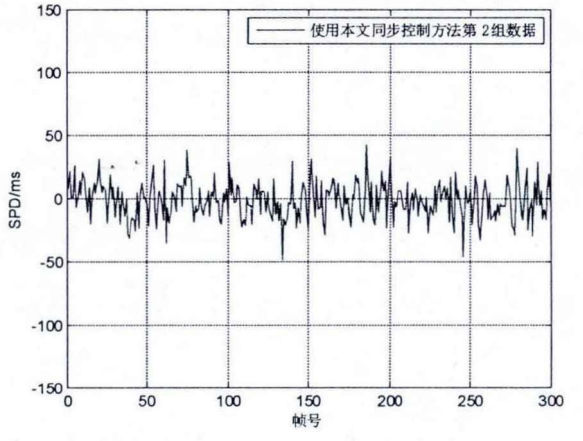

图 3-18 使用本文同步控制方法的第二组 SPD

Fig. 3-18 The SPD Of The Method Proposed By This Paper

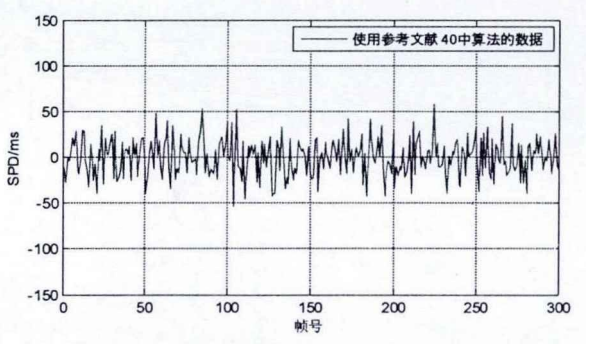

图 3-19 参考文献 40 中的方法的实验结果 SPD

Fig. 3-19 The SPD Of The Method Proposed In40th Reference Documentaion

3.5 本章小结

本章先引入音视频同步的主客观评价标准，在进行简要的介绍的基础上，结合实践中遇到的失同步问题，通过查阅文献和实验分析得出能够引起失同步现象的一系列原因。接着对音视频传输系统中的音视频同步问题进行分析，结合网络传输延时的判断，针对网络状况较差和一般的情形，提出了一种贯穿于整个音视频传输各个环节的同步控制方法，针对网络状况良好的情形，进行音视频同步算法分析。最后，通过搭建实验环境进行延迟检测仿真实验，以及高延迟下的音视频同步实验，经实验数据计算 SPD 参数，表明本章的自适应算法能很好地满足不同网络条件下音视频同步要求。

第四章 流媒体传输系统设计与实现

4.1 流媒体传输系统总体方案设计

在家庭智能服务机器人的音视频传输系统中，需要在基于 Android 操作系统的智能设备上开发应用程序实现监控画面的实时显示功能，语音数据的实时播放功能等。摄像头等音视频采集设备直接接在机器人的嵌入式开发板上，基于 Live555 的流媒体服务器也运行于此硬件平台上，来实现音视频数据的采集编码，以及使用流媒体协议进行实时转发。如图 4-1 中所示为整个视频监控系统的总体方案图，下面将进一步对每一个组成部分进行概要性的介绍。

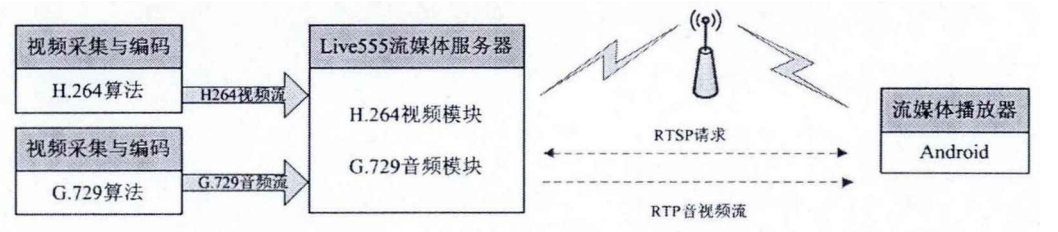

图 4-1 流媒体传输系统总体方案图

Figure 4-1 General plan of the streaming media transmission system

流媒体服务器是基于 Live555 开源项目而开发的，其中 Live555 中对于网络套接字的连接的处理是基于 select 进行多路复用的，而这种传统的 IO 模型存在很多严重的性能问题。我们需要在熟悉其整个源码的基础上，针对这个 IO 模型及相关部分进行重新开发，在 Linux 系统平台上使用基于事件的 epoll 模型来替换掉这部分源码，以提高系统性能，提高并发量，为系统规模的进一步扩大和高并发量等进行优化。Live555 在工程实践中饱受诟病来自于其对于多媒体格式的支持不够完善，同时也不支持对主流视频编码标准如 H.264 的视频直播。在接下来的 4.2 小节中，针对这些问题进行优化和扩展。

按照音视频传输系统中的同步解码过程，位于客户端中的传输模块中的接收部分先将流媒体数据。客户端主要是基于 Android 平台开发流媒体播放器，通过调用 FFMPEG 完成多媒体处理的关键环节，其中包括进行 RTSP 协议的信令交互，通过 RTP 包传输音视频数据，以及解码，最后结合 SDL 多媒体框架实现音视频的同步播放。

4.2 基于改进的 Live555 的流媒体服务器

在实时流媒体传输系统中，采用 Live555 开源项目作为基础。前文已经简要地介绍了 Live555 开源项目的基本情况等。本节将先提出本文设计的流媒体服务器的总体框架，通过深入分析 Live555 项目在工程中的应用现状以及实践中存在的功能以及性能方面的问题，对 Live555 开源项目的代码进行深入的分析，提出能够解决和改善这些问题的技术和方法，进一步补充和完善 Live555 开源项目。

4.2.1 流媒体服务端总体架构

Live555 是一个非常优秀的精简的开源的多媒体传输解决方案，可以用在客户端进行流媒体接收也可以用在服务端进行流媒体的发送。服务端的框图如图 4-2 所示，流媒体服务器接收到来自网络的 RTSP 请求后，遵循 RSTP 协议的标准，处理建立 RTSP 会话的信令流程，并分配相应的资源，建立对应的媒体通道。开始播放后，从摄像头和麦克风采集音视频数据，视频数据在 FFMPEG 下进行 H.264 编码，并将编码后的视频流交由 Live555 流媒体服务器处理；音频数据直接由 Live555 流媒体服务器进行 G.729 编码。Live555 流媒体服务器根据 RTSP 协商的结果，将编码后的音视频数据封装为 RTP 数据，发送到请求的客户端。其中，Live555 需要进行二次开发以增加流媒体的实时转发功能，并且需要对网络连接的 IO 模型进行改进以提高性能，扩展对于本地视频文件格式的支持如 MP4 格式。在本章的改进版服务器中针对这些不足进行二次开发以扩展和升级。

图 4-2 基于 Live555 的服务器框图

Figure 4-2 Block diagram of server based on Live555

4.2.2 Live555 流媒体服务器工作原理

Live555 开源项目主要由四个程序库和一个简单服务器程序和一些测试程序组成，其中 LiveMedia 程序库中的类主要用于处理 Live555 所支持的音视频编码格式和封装格

式，其均继承自 Medium 这个基类。UsageEnvironment 库是用于表示上下文环境的，可以通过其实现各种信息的打印等。BasicUsageEnvironment 库主要包含环境类的子类实现和任务调度类的子类实现。其中主要有两个特别重要的子类实现。就是 BasicUsageEnvironment 和 BasicTaskScheduler 类，这两个类分别基于 UsageEnvironment 和 TaskScheduler 类，实现了各自的功能。前者主要用于向命令行打印错误和打印用户想要输出的信息；后者主要实现了对于任务的调度处理，在 Live555 中主要有三类任务，分别为定时任务，socket 任务和事件任务。GroupSock 库是 Live555 中用于对底层 socket 操作进行封装供上层进行调用实现数据收发的网络库。一般以组播的形式工作，同时也能用单播的方式实现网络数据交互，但这种方式下传输层仅仅能使用 UDP 协议。MediaServer 是一个精简的基于 Live555 中的程序库的流媒体服务器程序。其使用了 BasicUsageEnvironment 类库中实现的 BasicUsageEnvironment 和 BasicTaskScheduler，而 BasicTaskScheduler 这个子类的实现中采用的是比较传统的基于 select 系统调用的 I/O 模型。这种方式在性能方面存在问题，因此，本文中将针对这个问题使用 Linux 系统下基于 epoll 的 I/O 模型对源码进行进行扩展。测试程序均是使用了上面介绍的程序库编写而成的客户端程序，用于为用户示范正确的使用方法。

通过分析项目中的 MediaServer 中的服务器程序的代码，得出 Live555 开发流媒体服务器的主要有四个核心步骤。

1. 为流媒体服务器创建任务调度类和环境类的子类实例

每一个基于 Live555 的流媒体服务器都必须有任务调度类（TaskScheduler）和环境类(UsageEnvironment)的子类实例。调用 BasicTaskScheduler 类和 BasicUsageEnvironment 类的静态工厂方法 createNew()来生成实例，代码如图 4-3 所示。其中 BasicTaskScheduler 类主要实现流媒体服务器中的任务调度，完成服务器中的传输等操作。在创建 BasicUsageEnvironment 类的实例时已经将任务调度类的实例作为引用参数传递给其实例，故可以通过其完成任务的调度等功能。

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

图 4-3 创建任务调度子类实例和任务调度子类实例

Figure 4-3 Creating instance of task scheduler subclass and task scheduler subclass

2. 创建 RTSP 服务器类的实例

RTSP 服务器类使用默认的 554 端口，在该端口被占用时选择 8554 端口，通过

RTSPServer 类的子类实现在该端口上创建服务器对象的代码如图 4-4 所示。该类主要在使用的 554 或者 8554 端口上建立基于 TCP 协议的监听套接字，将这个创建的监听套接字和对于监听套接字的请求处理函数一起注册给任务调度器，由任务调度器来响应监听套接字的连接请求并执行后续的连接套接字的任务处理过程。

```
RTSPServer* rtspServer;
portNumBits rtspServerPortNum = 554;
rtspServer = DynamicRTSPServer::createNew(*env, rtspServerPortNum, authDB);
if (rtspServer == NULL) {
  rtspServerPortNum = 8554;
  rtspServer = DynamicRTSPServer::createNew(*env, rtspServerPortNum, authDB);
}
```

图 4-4 创建 RTSP 服务器对象

Figure 4-4 Create a RTSP server object

3. 通过任务调度类执行三类基本任务的循环调度

BasicTaskScheduler 任务调度器类中是把监听套接字的文件描述符使用基于 select 的 I/O 模型进行多路复用。首先将对应的连接处理函数和监听套接字的文件描述符一起存入保存这个映射关系的 HashSet 数据结构中，接着把监听套接字的文件描述符加入到 select 的套接字描述符集合中。

在 RTSP 服务器的代码中将通过环境类获取到其中的 TaskScheduler 成员，再调用这个任务调度类执行事件循环，事件循环中执行子类 BasicTaskScheduler 类的成员函数 SingleStep()，如图 4-5 所示。而在 BasicTaskScheduler 的 SingleStep 函数中主要依次完成三类事件的调度：使用 select 多路复用执行套接字任务并执行，找到第一个响应的事件并执行，最后找到第一个应执行的延迟任务并执行。而其中的对于套接字任务进行处理的 select 模型就是后文中需要改进优化的。

```
void BasicTaskScheduler0::doEventLoop(char* watchVariable) {
  // Repeatedly loop, handling readble sockets and timed events:
  while (1) {
    if (watchVariable != NULL && *watchVariable != 0) break;
    SingleStep();
  }
}
```

图 4-5 执行三类基本任务的循环调度

Figure 4-5 Cyclic scheduling of three basic tasks

4. 在服务器端为客户端创建 RTSP 客户会话

在客户端请求连接时，在任务调度类 BasicTaskScheduler 的 SingleStep 函数中的 select 调用会因此而返回，返回值为之前向其可读文件描述符集添加的监听套接字的文件描述符，通过这个监听套接字的文件描述符在 HashSet 中可以找到对这个连接进行

处理的 incomingConnectionHandler 函数对象，接着由这个函数对象来完成 RTSP 协议的请求以及后续过程。

在 incomingConnectionHandler 处理过程中，首先通过 accept 函数来建立与用户的连接，接着为这个新建立的套接字连接创建一个 RTSPClientConnection 对象，这个会话对象实现对用户的请求进行处理的过程，并使用其中的 handleRequestBytes 函数处理来自客户端的 RTSP 协议中定义的命令，在处理 SETUP 命令时，会创建一个 RTSPClientSession 对象，并将对于命令的处理分为两部分。对于 SETUP、PLAY、PAUSE、TEARDOWN 等命令传递到 RTSPClientSession 对象中处理，而 OPTIONS、DESCRIBE 及其他命令存在异常的情况的响应直接在 RTSPClientConnectio 对象中处理。对于各个命令的具体交互过程如下图 4-6 所示。

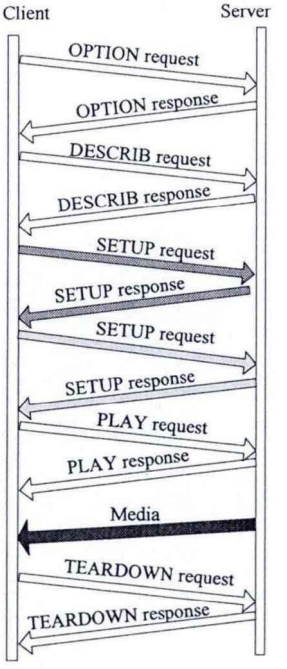

图 4-6 RTSP 命令交互过程

Figure 4-6 Process of RTSP command interaction

4.2.3 音视频数据采集以及编码传输

在 Linux 平台下通过 Video For Linux 的第二个版本 V4L2 框架对视频数据进行采集。在 Linux 系统下，所有的设备都通过系统抽象成为了文件的概念。因此，对于设备的操作都是通过类似于对文件的访问的方式来进行的。对于 V4L2 框架的编程过程主要遵循如下图 4-7 中左图所示的六大步骤 。Linux 下音频数据的采集可以选择 ALSA(Adanvanced Linux Sound Architecture)架构的声卡驱动程序，通过对音频设备编

程完成音频数据的采集过程，编程过程如图 4-7 中的右图所示。

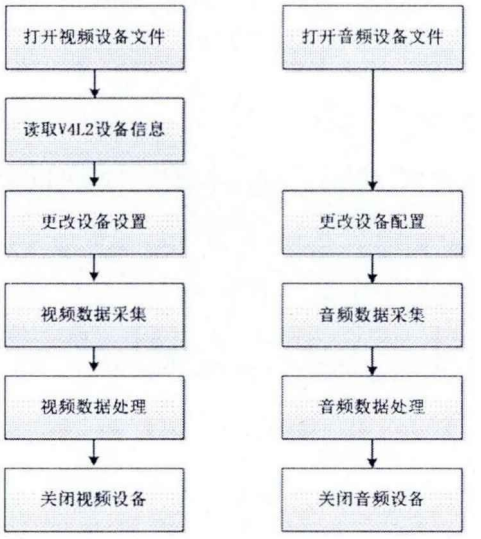

图 4-7 对音频和视频设备编程的步骤

Figure 4-7 Procedures for audio and video equipment programming

1. 打开设备文件和关闭设备文件

使用 open 和 close 系统调用来对设备进行打开和关闭操作，Linux 系统下对于设备的处理与对于文件的编程概念上相同。Linux 会将连接在系统上的摄像头和声卡设备分别虚拟成为一个设备文件，位于根文件系统的/dev 目录下。通过插拔设备可以找到这个设备文件的文件名，在本系统中，视频设备文件名为/dev/video2，音频采集设备的文件名为/dev/dsp。使用如下的关键代码进行：

```
#define V4L2_UVC_CAMERA  "/dev/video2"
#define AUDIO_DEVICE "/dev/dsp"
int camera_fd = open(V4L2_UVC_CAMERA, O_RDWR);
int audio_fd = open(AUDIO_DEVICE, O_RDONLY);
```

在两个设备各自的数据采集任务结束以后，最后需要对设备进行关闭操作，以将设备归还给系统，并且释放掉资源等，需要调用 close 系统调用来完成。

2. 读取 V4L2 设备信息

V4L2 设备是通过 ioctl 系统调用对设备进行编程控制的。读取设备的信息，主要有如下的两个关键信息：查询设备的功能，获取设备支持的视频格式。查询设备的功能是通过在 ioctl 中使用 VIDIOC_QUERYCAP 命令，其使用 struct v4l2_capability 来返回所有需要查询的设备信息。将这个结构体和对应的宏定义的掩码按位与就可以得出

是否支持相对应的功能。

```
if (!(m_v4l2_capability ->capabilities & V4L2_CAP_VIDEO_CAPTURE))
 {  // 表示是一个视频捕捉设备
    exit(-1);
  }
if (!(m_v4l2_capability->capabilities & V4L2_CAP_STREAMING))
 {  // 表示具有数据流控制模式
  exit(-1);
 }
```

3. 更改音视频设备设置

摄像头对于支持的视频数据采集格式以及分辨率等都是有限的，在进行视频数据采集之前必须针对这些参数等进行编程设置。在设置之前，要了解整个系统对于视频设备的格式支持情况，系统提供了 VIDIOC_ENUM_FMT 命令来对视频采集设备支持的格式进行枚举，返回的视频支持格式存在结构体 struct v4l2_fmtdesc 中。从中选择一个视频格式，填充好 struct v4l2_format 结构体，通过 VIDIOC_S_FMT 可以将其写入到设备中完成配置。

设置完摄像头采集的视频数据格式以及分辨率等之后，需要协定视频数据的读写方式。V4L2 一共支持三种数据读取方式，分别为直接使用 read、write 系统调用来进行原始数据的读写，通过共享内存的方式进行内存映射，以及使用用户指针。操作系统把虚拟内存分为用户内存和内核内存，在 V4L2 框架采集到了视频数据时，数据在设备驱动程序中也就是处于内核内存中，需要通过以上的三种方式中的一种来完成内存的读取。其中，以 read、write 系统调用进行的 IO 操作，每一次系统调用都会让操作系统完成从用户态到内核态的切换，以及频繁地拷贝数据，其时间开销为最大；共享内存方式使得内核在采集到了视频数据之后，通过映射内存到用户空间，减少了数据在内存中的拷贝次数与系统内核态与用户态的频繁切换；用户指针方式需要用户提供内存给驱动，需要用户来管理和分配内存，相对共享内存方式稍复杂和易出错。因此，文中采用共享内存方式来采集视频数据。使用 VIDIOC_REQBUFS 命令通过 v4l2_requestbuffers 结构体请求驱动申请内核空间的内存用于视频数据的缓存。此时，将申请到的内存通过 mmap 系统调用映射到用户空间中，如下为关键代码：

```
mmap(NULL , buff.length, PROT_READ | PROT_WRITE , MAP_SHARED, camera
```

_fd, buff.m.offset);

返回的指针指向了映射到的用户空间的起始地址，随后去读取采集到的视频数据就是通过这个地址来进行的。

音频设备需要对于采集的音频数据的参数进行设置，这主要包含声卡量化的位数，以及声卡的声道数目，声卡的采样频率等。其中都是通过直接使用 ioctl 系统调用，按照特定的定义在 linux/soundcard.h 头文件中的命令进行配置的，其具体命令以及作用如下表 4-1 所示。关键代码如下：

```
#define SAMPLE_FREQUENCE   8000   //声卡的采样频率，设置为 8KHz
#define SAMPLE_SIZE   16        // 声卡的采样位数，设置为 16 位
#define SAMPLE_CHANNELS   1   // 声卡的采样声道，设置为单声道
int sndcard_param = SAMPLE_SIZE;
status = ioctl(audio_fd , SOUND_PCM_WRITE_BITS, & sndcard_param);
sndcard_param = SAMPLE_CHANNELS;
status = ioctl(audio_fd, SOUND_PCM_WRITE_CHANNELS, & sndcard_param);
sndcard_param = SAMPLE_RATE;
status = ioctl(audio_fd, SOUND_PCM_WRITE_RATE, & sndcard_param);
```

表 4-1 声卡编程相关的配置命令

Table4-1 Command for sound adapter

声卡命令名	命令作用
SOUND_PCM_WRITE_BITS	配置声卡的量化位数，一般设置为 8 或 16
SOUND_PCM_READ_BITS	读取声卡的量化位数配置信息；
SOUND_PCM_WRITE_CHANNELS	配置声卡声道类型：1 为单声道，2 为立体声
SOUND_PCM_READ_CHANNELS	读取声卡声道的声道类型配置信息；
SOUND_PCM_WRITE_RATE	配置声卡的采样频率，为 8K 或 16K
SOUND_PCM_READ_RATE	读取声卡的采样频率配置信息
SOUND_PCM_SYNC	配置改变播放文件参数时播放完缓冲区中内容

4. 音视频数据采集

音频数据的采集使用 read 系统调用直接对音频设备文件进读取，类似于文件操作可以将一帧音频数据从设备中采集出来。其中的第三个参数使用的是 SAMPLE_LENGH 秒数采集的音频数据的大小，是通过计算出来的。

```
#define FRAME_SIZE ( SAMPLE_LENGTH * SAMPLE_FREQUENCE * SAMPLE_SIZE * SAMPLE_CHANNELS/8.0)
```

read(audio_fd, audio_buff, FRAME_SIZE);

视频设备在初始化时已经设置成共享内存方式读取，故只需要像访问普通内存一样去访问用户空间就可以。V4L2 提供了两个命令 VIDIOC_DQBUF 和 VIDIOC_QBUF 用于实现此种方式下的与设备驱动交互。通过 VIDIOC_DQBUF 命令从驱动中获得已经捕获了视频数据的缓存块，在 ioctl 系统调用返回时，视频数据就已经准备就绪；通过 VIDIOC_QBUF 则再将指定的内存再次加入设备驱动的空闲缓存队列中去，供驱动使用。其关键代码如下：

ioctl(camera_fd, VIDIOC_DQBUF, &buf);

// 进行视频数据的具体处理过程，如进行视频数据编码等

ioctl(cam->fd, VIDIOC_QBUF, &buf));

5. 音视频数据编码

通过移植 FFMPEG 到系统上，调用其中的编码器进行音视频编码。使用 FFMPEG 进行音视频编码的流程如图 4-8 所示，主要包括注册编码器，在 FFMPEG 中找到 G.729 算法和 H.264 标准对应的编码器，接着打开编码器对音视频频数据进行编码，当编码结束的时候需要释放掉所有相关的资源。

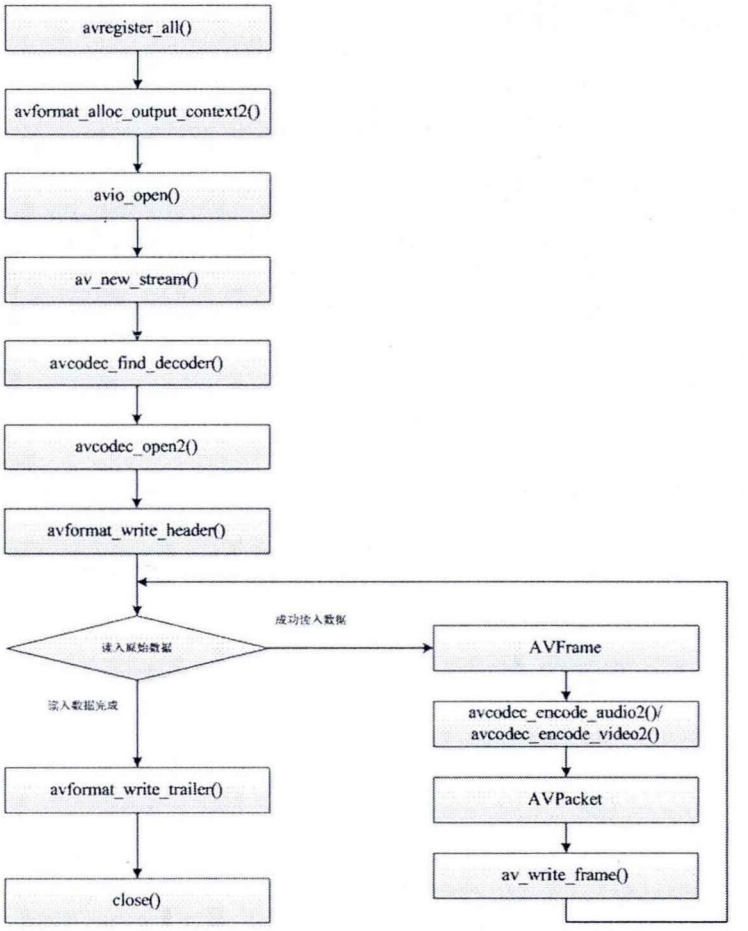

图 4-8 使用 FFMPEG 对音视频数据进行编码的流程

Figure 4-8 Process of encoding audio/video data using FFMPEG

A) 注册并找到音频编码器

FFMPEG 编码的流程都是从 avcodec_register_all()开始，向系统注册所有的编解码器，进而通过 avcodec_find_encoder 函数找到对应 G.729 音频编码算法的 Codec。关键代码如下：

```
av_register_all();
pCodec = avcodec_find_encoder(CODEC_ID_G729);
```

B) 打开编码器、设置音频编码参数并完成音频编码

在获取到了音频编码器之后，打开音频编码器进行音频编码，调用 avcodec_open2()函数来打开编码器，此时，在 AVCodecContext 结构体中就已经保存了指向打开的音频编码器的句柄，调用 avcodec_encode_audio2()函数就可以完成音频的编码，关键代码如下所示：

```
avcodec_open2(pCodecCtx, pCodec,NULL);
avcodec_encode_audio2(pCodecCtx, &pkt,pFrame, &got_frame);
```

4.2.4 优化 Live555 的 IO 模型

1. Live555 中基于 select 系统调用的 IO 模型存在的问题

通过分析 Live555 的源码可以发现，其中的基本的任务调度类 BasicTaskScheduler 类在对于客户端的连接到来时，采用的是单线程的 select 复用模型。这种 I/O 多路复用模型在高并发访问时存在严重的性能问题。主要表现为 select 在默认不修改内核参数的情况下，其最大支持的文件描述符数为 1024；在每一次调用的过程中，被监视的文件描述符集合都从用户空间拷贝到内核空间，当服务端进行高并发访问时，在这部分的开销将显著增加，进而影响系统吞吐率；select 在对于文件描述符进行监视是在内核中通过依次遍历所有的被监视文件描述符完成的，同上在高并发访问场景中，系统开销显著增加。下面通过对流媒体服务器中与网络连接相关的部分进行分析，找出其中对于服务器性能存在问题的部分。

在 BasicTaskScheduler 中的 SingleStep 函数中对套接字任务、事件任务、延时任务的处理过程如下图 4-9 所示。任务调度器在创建的时候就将服务器创建的监听套接字存入对应的 HandlerSet 数据结构中，接着使用一定的超时时间调用 select 函数，在 select

非超时和出错返回之后，表示监听的文件描述符集已经准备就绪。BasicTaskScheduler 中的 fHandlers 成员指向的 HandlerSet 中保存了套接字描述符和对应的 Handler 的映射关系。在 select 返回的就绪的文件描述符集中如果有对应的套接字描述符集，那么就通过 fHandlers 找到对应的 HandlerDescriptor，进而执行其中的处理函数。

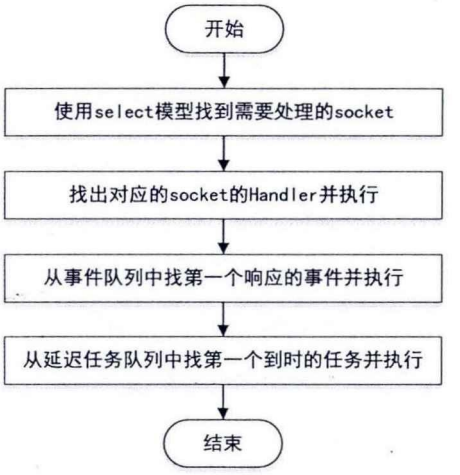

图 4- 9 BasicTaskScheduler 类中对于任务的一次调度

Figure 4-9 A scheduling for tasks in the BasicTaskScheduler class

2. 使用 epoll 模型改进的套接字任务调度

在深入分析了 BasicTaskScheduler 类的代码后，针对使用 select 的部分进行二次开发。如上一小节中图 4-9 所示，其中主要针对其中的用于处理任务调度的 SingleStep 函数以及用于将套接字和对应的向任务调度类添加套接字任务的 setBackgroundHandling 等函数使用 epoll 模型重新实现。

Linux 平台下使用 epoll 实现 IO 多路复用主要有三个步骤：使用 epoll_create()创建 epoll 句柄，使用 epoll_ctl 以及对应的命令对事件进行管理，使用 epoll_wait 等待事件的产生。设计 BasicTaskSchedulerEpoll 类继承自 BasicTaskScheduler 类，重新实现其中与套接字任务有关的函数。在 BasicTaskSchedulerEpoll 的构造函数中调用 epoll_create 创建 epoll，接着在 SingleStep 函数中使用 epoll_wait 函数来替换掉 select 的调用，实现对于网络套接字的任务的事件处理。重新定义 setBackgroundHandling()，这个函数向任务调度对象添加套接字任务改为使用 EPOLL_CTL_ADD 命令的 epoll_ctl()函数实现。

4.2.5 扩展 Live555 对 MP4 等多媒体格式的解析

为了以实时流的形式访问存储在服务端使用 MP4 格式保存的音视频文件，需要对 Live555 添加格式支持。在基于 Live555 的流媒体服务器开发中，采用源端(Source)和

接收端(Sink)的概念来抽象描述流媒体数据的传输过程。MP4 是一种容器格式，需要通过对其进行基于 FFMPEG 的解封装和解复用，从其中得到音频数据流和视频数据流，通过 RTPSink 将音视频数据流以 RTP 数据包的形式经过网络发送到客户端。对于其他多媒体格式的支持也是使用相同的原理，都是基于 FFMPEG 来扩展 Live555 支持的多媒体文件格式的种类。对于其他多媒体文件格式的支持的扩展与 MP4 格式的在整体流程上是相同的，本小节针对 MP4 格式的支持进行详细阐述，其中主要是参考了 MPG 文件格式的代码。

在 Live555 的编程框架中，ServerMediaSession 是与服务端的资源文件相对应，也就是一个存储在流媒体服务端的多媒体文件抽象为一个 ServerMediaSession 类的实例。通过分析发现 ServerMediaSession 类实例是在 DynamicRTSPServer 类中创建的，因此，首先创建一个 FFmpegRTSPServer 类，这个类继承自 DynamicRTSPServer 类，并且重新定义 lookupServerMediaSession 这个函数，在其中扩展 MP4 文件格式的支持，并对于 MP4 文件中的 AAC 音频数据流和 H.264 视频数据流进行创建与添加。

为了对于其中的音视频流进行处理，需要通过创建 SeverMediaSubsession 实例来抽象表示音视频流，并通过调用 ServerMediaSession 类中的函数把这两个子会话实例添加到其中去。而创建这两个 ServerMediaSubsession 实例的任务就在于 FFmpegServerDemux 类中，该类通过使用 FFMPEG 中的对于文件操作的 API 函数，av_open_input_file()和 av_find_stream_info()对 MP4 文件中的视频和音频数据流分别进行探测，并负责创建对应的 H.264 和 AAC 的 ServerMediaSubsession 实例。此外，通过对 Live555 中现有 MPG 格式的支持代码的分析可知，FFmpegServerDemux 还需要完成创建 FramedSource 子类的实例，故需要创建 FFmpegDemuxedElementaryStream 实例。

FFmpegDemux 类是 Medium 类的子类，用 FFMPEG 扩展视频格式的代码中最核心的步骤就是使用 FFMPEG 提供的 API 去对 MP4 文件进行解复用操作，把文件中的音频和视频流分离，在分离以后创建对应的数据源。客户端与 FFmpegDemux 是一对一的映射关系。

FFmpegDemuxedElementaryStream 类是 FramedSource 的子类，该类从 FFmpegDemux 中获取已经分离的音视频数据，并向 Sink 发送，而且是在与之相关联的 Sink 的调用下工作的。

在 FFmpegServerDemux 类中在探测了 MP4 格式文件中的音视频数据流之后，创建对应数据流的 ServerMediaSubsession 子类的实例。在每一个编码格式对应的

ServerMediaSubsession 子类中，需要重新定义 createNewStreamSoruce 函数，来创建数据源。具体分析 H.264 编码格式对应的 FFmpegH264ServerMediaSubsession，它是 H264VideoFileServerMediaSubsession 的子类，其通过 FFmpegServerDemux 来获得 ElementaryStream 流，并作为参数传入 H264VideoStreamFramer 中。

4.2.6 基于 Live555 的实时视频转播的实现

Live555 不支持从本地采集的实时视频直接转发给客户端，而这个实时视频转播的功能非常关键，因此，需要开发服务端的实时视频转发功能。通过分析 Live555 的源码可知，要进行本地视频采集的转发功能的开发，最重要的是要根据自己需要转发的流的特点来开发 OnDemandServerMediaSubsession 类的子类，并重新实现其中的函数。对于视频和音频流的处理过程方法上是相同的，这里针对 H.264 标准的视频流的本地采集转发功能的实现进行分析。

开发 H264ServerMediaSubsession 类作为 H.264 视频流对应的 OnDemandServerMediaSubsession 的子类。其中 CreateNewRTPSink 函数重新定义用于创建 H264VideoRTPSink。createNewStreamSource 函数中生成自定义的数据源子类 V4L2FramedSource 的实例。在 V4L2FramedSource 中重新定义了函数 doGetNextFrame()，这个函数中通过 Video For Linux2 框架进行原始视频数据采集，并对采集的视频数据进行 H.264 编码。在 H264VideoRTPSink 类中会调用其父类 MultiFramedRTPSink 的 packFrame()，而这个函数中会调用与之相关联的数据源类的 getNextFrame()方法。从数据源 Source 到 Sink 的连接就已经建立起来了。为视频转播功能开发的 UML 类图如图 4-10 所示。

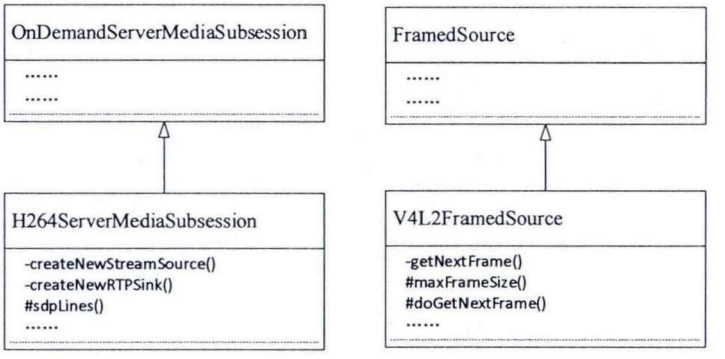

图 4-10 视频转播功能开发的类图

Figure 4-10 Class diagram of repost video function

4.3 流媒体系统客户端研究

嵌入式技术以及移动互联网的高速发展，使得基于 Android 平台的软件研发已经成为一个非常重要的热点。本节将针对 Android 平台，研究流媒体客户端的实现方案。在家庭智能安防机器人的应用场景中，需要通过 Android 设备作为流媒体接收端，同时，为了对音视频进行播放，需要进行音视频数据的解码，在解码过程中需要完成同步控制，最终完成多媒体内容的播放。本节先从整体上介绍客户端的设计与架构，再针对 Android 平台自身的音视频解码存在的问题等，提出移植 FFMPEG 到 Android 平台，再基于 FFMPEG 进行多媒体处理以及音视频同步。

4.3.1 流媒体客户端总体分析与设计

在客户端接收到音视频数据之后，需要对接收的音视频数据流进行提取和解码，最终在客户端通过调用本地的应用程序接口，通过声卡和显卡播放给用户。如图 4-11 所示，是 Android 平台流媒体客户端的总体框架图。在客户端由 FFMPEG 库中的函数完成 RTSP 协议的信令交互过程，并完成 RTP 数据包的接收缓存与重排。经过 FFMPEG 的读取和处理之后，音视频数据已经可以供程序读取。在 Android 平台上通过 JNI 技术封装解码的过程，同时移植 SDL(Simple Direct Layer)多媒体开源项目到 Android 平台，在此多媒体框架上利用多线程和时间戳技术实现音视频原始数据的播放过程的媒体间同步。

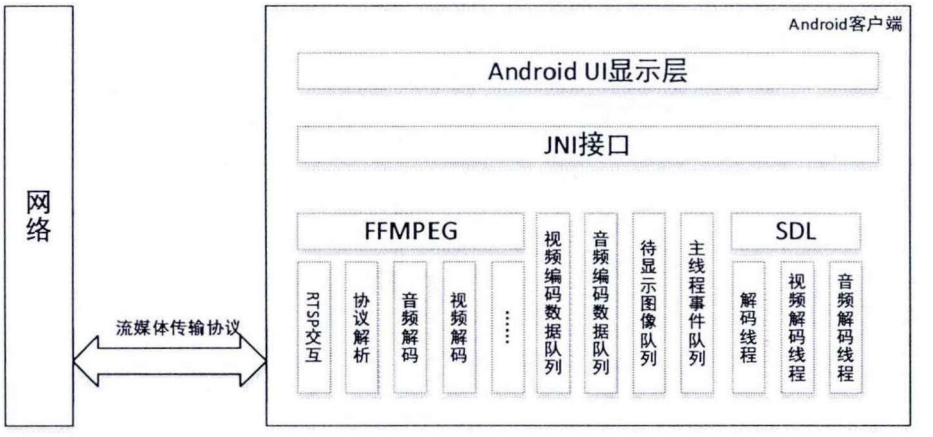

图 4-11 基于 FFMPEG 的 Android 流媒体客户端框图

Figure 4-11 Block diagram of Android streaming media based on FFMPEG

4.3.2 移植 FFMPEG 到 Android 平台

Android 系统使用 StageFright 作为多媒体框架。但是 Android 系统对于音视频格式以及封装格式的支持不够丰富。而开源项目 FFMPEG 能够支持几乎所有的格式。FFMPEG 项目是用 C 语言开发的，因此需要通过对其进行封装并编译生成动态链接库文件，用 JNI 来调用链接库中的接口，以使用 FFMPEG 强大的多媒体处理功能。SDL(Simple DirectMedia Layer)是跨平台的开源多媒体支持库，其也是基于 C 语言写成。它通过封装多媒体处理的技术细节，如内部向底层调用 OpenGL 接口来完成图像渲染等，向上层应用软件提供强大的跨平台（Linux、Android、Windows 等）的多媒体类应用功能。本节通过交叉编译将 FFMPEG 移植到 Android 上，为下文在 Android 平台基于这个强大的多媒体处理库和 SDL 库，进行基于时间戳的音视频同步技术研究做准备。

将 FFMPEG 应用在 Android 平台主要有两种方案。第一种是把 FFMPEG 中的解码器加入到 Stagefright 框架中。这种方式的软解码效率较 Android 自带的软解码效率高，而且在应用层编程接口可以统一使用，但必须要修改系统源码，不适合作为应用开发者使用。第二种是使用 Native Development Kit 交叉编译 FFMPEG，同时使能 FFMPEG 自带的 Android 硬解码支持库 libstagefright，再在应用开发过程中通过编写 JNI 接口代码实现调用。这种方法不用修改系统源代码，更适合于应用开发者，本文也采用这种方式，交叉开发环境如下表 4-2 所示：

表 4-2 交叉开发环境

Table 4-2 Cross development environment

交叉编译主机	UBUNTU14.04LTS 64 位
Android NDK	android-ndk-r9d-linux-x86_64.tar.bz2
Android SDK	adt-bundle-linux-x86_64-20140702.zip
FFMPEG	FFMPEG-2.8.1
SDL	SDL2-2.0.4

首先将 FFMPEG 的最新版源码下载到本地，在 Linux 平台下为嵌入式软件进行交叉编译和移植主要分为配置，构建和安装三步。配置的过程使用 FFMPEG 的 configure 文件进行，由于在需要配置的选项非常多，可以使用如下图 4-12 所示的 shell 脚本进行配置。

```
#!/bin/bash
NDK=/home/abc/tools/android_ndk/android-ndk-r9d/
SYSROOT=$NDK/platforms/android-16/arch-arm/
TOOLCHAIN=$NDK/toolchains/arm-linux-androideabi-4.8/prebuilt/linux-x86_64

function build_one
{
  ./configure \
    --prefix=$PREFIX \
    --enable-shared \
    --disable-static \
    --disable-doc \
    --disable-ffserver \
    --enable-cross-compile \
    --cross-prefix=$TOOLCHAIN/bin/arm-linux-androideabi- \
    --target-os=linux \
    --arch=arm \
    --sysroot=$SYSROOT \
    --extra-cflags="-Os -fpic $ADDI_CFLAGS" \
    --extra-ldflags="$ADDI_LDFLAGS" \
    $ADDITIONAL_CONFIGURE_FLAG
}
CPU=arm
PREFIX=$(pwd)/android/$CPU
ADDI_CFLAGS="-marm"
build_one
```

图 4-12 对 FFMPEG 进行配置的 SHELL 脚本

Figure 4-12 SHELL script for FFMPEG configuring

通过配置脚本进行构建参数的配置后，使用 make 工具进行构建，交叉编译生成需要的动态链接库，主要有如下表 4-3 所示的 so 文件，抽取这些库和相关的头文件用于 Android 项目中，供 Android 部分的代码使用静态代码块来加载，并通过编写 JNI 层的本地代码调用其中的 API。

表 4-3 移植的 FFMPEG 的动态链接库列表

Table. 4- 3 The List Of FFMPEG Libs Cross-Compiled For Android

动态链接库名称	功能描述
libavutil	辅助工具库
libswresample	音频数据重采样以及采样格式转换等
libavcodec	提供一个通用的编解码框架和大量的多媒体数据编解码器
libavformat	提供一个通用的复用和解复用框架和大量的复用和解复用器
libswscale	图像缩放、颜色空间转换像素格式转换等
libavfilter	提供一个通用的过滤器框架以及大量的过滤器等
libavdevice	提供通用框架从输入设备捕获数据渲染数据到输出设备

4.3.3 基于 FFMPEG 和 SDL 的音视频解码与同步研究

对一个多媒体流进行处理，从中提取出需要的多媒体数据，并且完成本地的播放，需要进行如图 4-13 所示的关键步骤。第一步是流媒体数据经过流媒体协议解析成为带

有封装格式的数据。协议解析的过程，主要是去除掉收到的数据中用于网络传输的那部分信令数据。第二步是对封装格式解析，将封装格式的数据分离成为音频编码数据和视频编码数据，常见的封装格式有 MP4、MKV、TS 等。第三步是分别对音频和视频压缩编码数据进行解码，得到原始的非压缩的音频和视频数据，此时的数据是可以直接通过硬件设备驱动直接播放的，解码的过程是最复杂的。第四步是音视频的同步播放。在解码的过程中，音视频数据会存在时间戳等信息，通过相互协调可以实现两种数据的媒体间同步。而这个同步过程主要有三种方式：同步音频的播放过程到视频时间戳，同步视频的播放过程到音频时间戳，以及利用外部系统时钟同步音频和视频。而在移动视频监控中，语音数据的实时连续性非常重要，本节中实现的就是同步视频的播放过程到音频时钟上。在整个四个步骤中，由 FFMPEG 完成前三步的处理，而最后显示与同步则需要使用 SDL 以及同步算法进行。

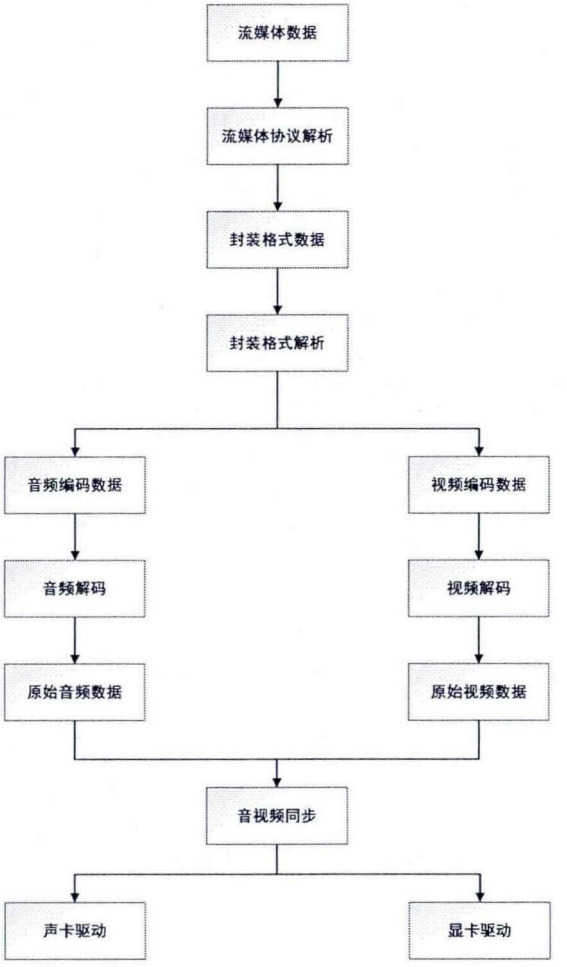

图 4-13 对流媒体进行播放的过程

Figure 4-13 Process of streaming media play

1. JNI 部分架构设计

在 Android 系统上进行 SDL 和 FFMPEG 的编程出于其两个开源库本身的跨平台特性，可以做到与 Android 平台本身编程无关。主要在 JNI 层通过 C 语言实现底层的解码和音视频同步并且基于 SDL 媒体框架渲染视频和播放声音，而在 Android 的应用层则通过 SDL_main 函数进入底层功能模块。整个客户端的多媒体处理的核心部分就在于 JNI 层，这部分采用多线程模型实现，主要分为四个线程，主线程以及解码子线程，音频解码线程和视频解码线程。四个线程的从属关系为主线程创建解码子线程，解码子线程创建和启动音频和视频解码子线程。线程与线程间使用 SDL 的互斥锁和条件变量等机制互斥访问音视频数据队列来完成线程间通信，其整体架构设计如图 4-14 所示。

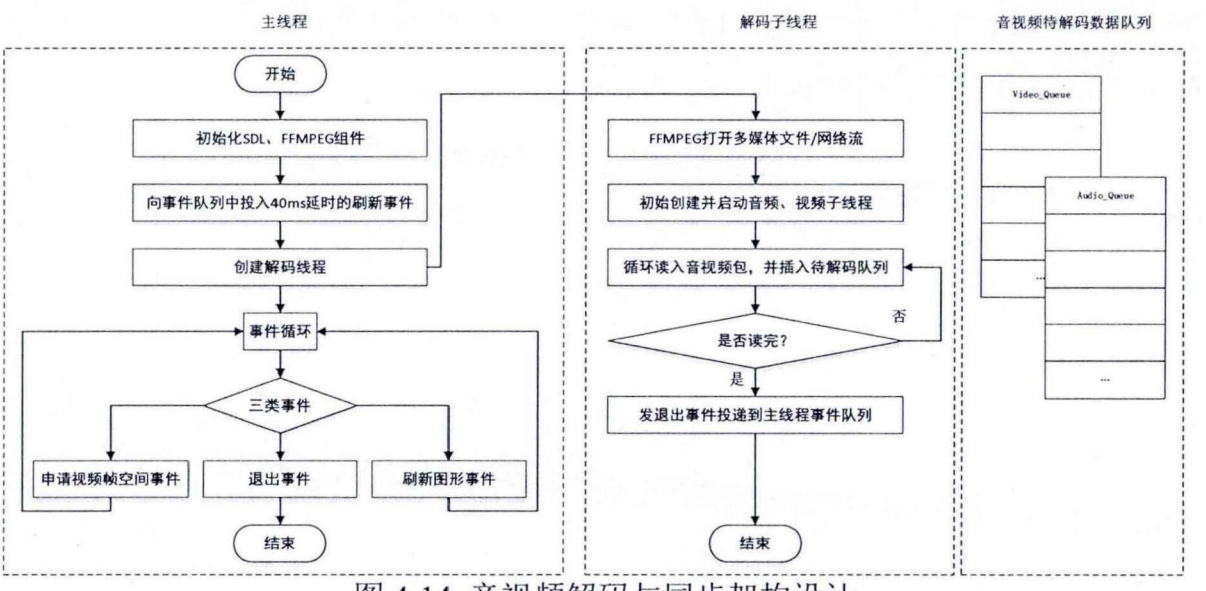

图 4-14 音视频解码与同步架构设计

Figure 4- 14 Audio and video decoding and synchronization architecture design

在主线程中主要完成对 FFMPEG 和 SDL 框架的初始化配置，创建解码子线程，并且利用 SDL 事件循环机制进入事件循环。事件循环中主要处理三类事件，分别为程序退出事件、申请视频帧空间事件、以及刷新播放画面事件。其中整个系统的待显示图像的播放就是在刷新播放画面事件中完成的，利用 SDL 的定时器机制，在视频解码子线程中将解码完成后的视频帧插入到解码数据循环队列中。主线程中在接收到这个刷新事件时，通过视频帧的显示时间戳 PTS(Presentation TimeStamp)参考音频的当前播放时钟实现同步视频到音频。

解码子线程中主要完成 FFMPEG 对指定 URL 的读取的前期配置工作，接着创建视频子线程，并通过 SDL 创建并启动音频子线程，最后进入读取网络数据流的循环中，将读入的音视频数据包分别投入到对应的编码数据队列，供音频和视频解码子线程各

自进行读取。

音频解码子线程是由 SDL 框架管理的，由其进行数据回调，回调函数中主要完成的是音频解码以及拷贝到声卡实现音频的播放的过程。

视频解码子线程中从视频编码数据队列中读取数据包，在解码之后，对当前解码数据帧的显示时间戳 PTS 进行修正，将视频数据以及对应当前解码的视频帧的时间戳封装到 VideoPicture 数据结构中，并插入到解码数据队列中。

在整个过程中对于音视频同步最关键的是在视频解码子线程中对每一个解码帧的显示时间戳的修正和在主线程中的基于 PTS 同步视频到音频的过程，下面将深入分析。

2. 修正 PTS

每一帧视频帧的显示时间戳需要考虑到视频的编码中，有完整的 I 帧和需要依赖其他参考帧的 P 帧和 B 帧。P 帧需要参考前向的 I 帧，但 B 帧需要参考前向和后向的帧。因此由于 B 帧的存在，视频帧的显示顺序和存放顺序可能不同。在通过 FFMPEG 对视频进行解码后，解码函数将对这个乱序进行重排，使得解码后得到的帧一定是按照显示顺序进行排序的。解码后，视频数据存储在 AVFrame 数据结构中，其中不再存储当前解码帧的显示时间戳 PTS。

为了解决这个问题，需要对 FFMPEG 源码进行分析，在 avcodec_decode_video2() 函数中，在当为一帧申请内存的时候是要调用 AVCodecContext 结构体中的 get_buffer2() 这个函数的，通过重新实现这个函数，并向 AVCodecContext 中注册可以让 FFMPEG 调用自定义函数，实现内存分配，以及在此同时加入对解码后的视频帧的 PTS 的保存，利用 AVFrame 结构的 opaque 成员变量来存储这个 AVFrame 的第一个包在申请帧内存时的 PTS，修正算法如图 4-15 所示。当通过 avcodec_decode_video2()解码得到一个能够显示的帧时，由于这个帧的 PTS，仍然可能存在为 0 的情况，故需要进一步进行修正，此时使用系统维护的当前视频播放时钟 video_clock 来继续对 PTS 为 0 的情况进行修正，将当前帧的 PTS 设置为 video_clock。

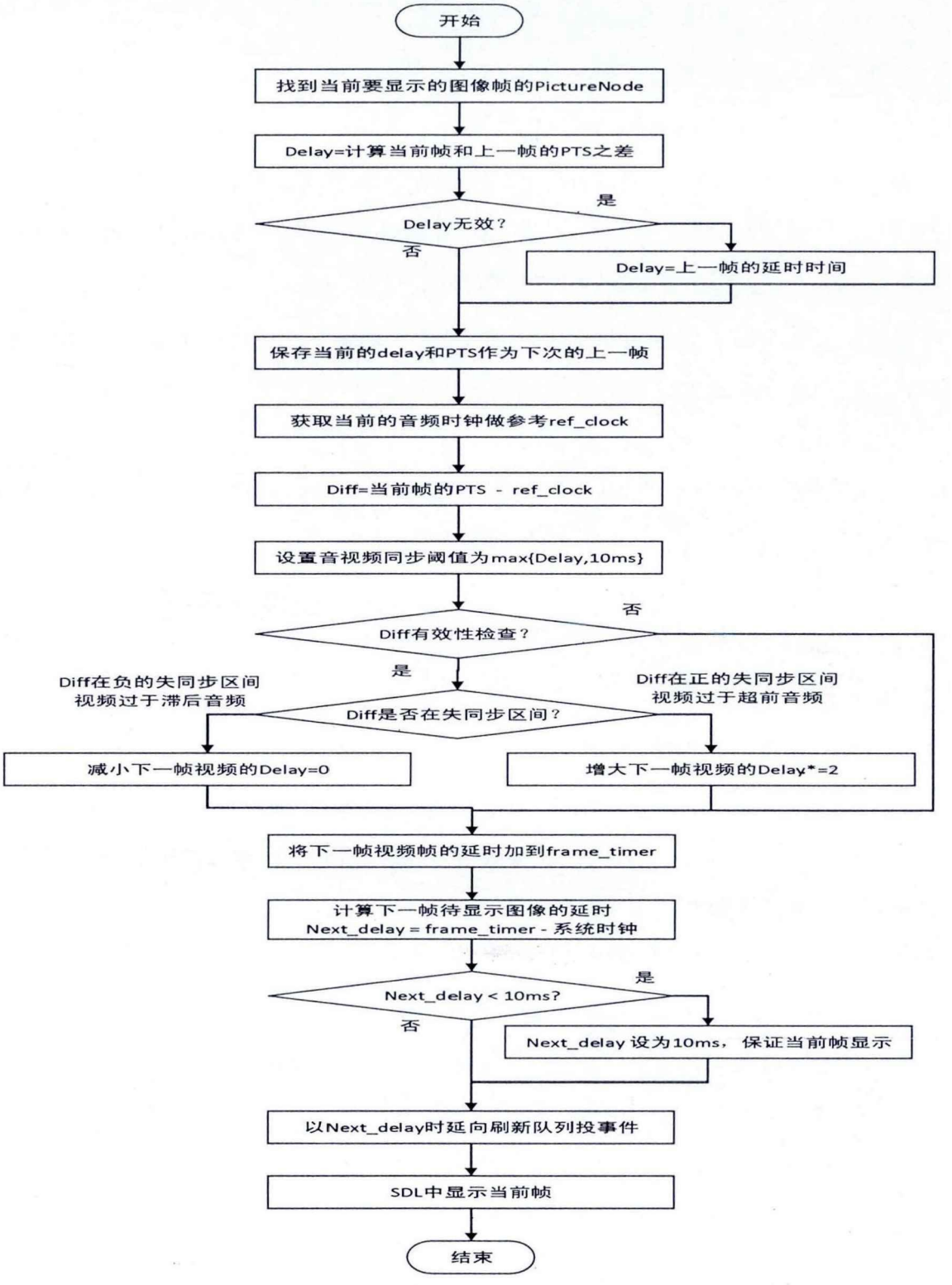

图 4-15 当前帧的 PTS 的修正算法

Figure 4-15 Correction algorithm for PTS of current frame

通过上述修正算法之后，得到的待显示图像的的PTS已经是当前帧的正确的PTS，故可以把在 SDL 中要渲染的数据和其对应的 PTS 等信息封装为如下的 PictureNode 结构，并插入到待显示图像队列中。

```
typedef struct PictureNode {
    SDL_Window *m_screen;
    SDL_Renderer *m_renderer;
    SDL_Texture *m_bmp;
    AVFrame* m_rawdata;
    int m_width, m_height; /*source height & width*/
    int m_allocated;
    double pts;
} PictureNode;
```

3. 基于 PTS 同步视频到音频

在上文对每一个视频帧的显示时间戳进行了修正之后，每一帧都有正确的 PTS 值。采用同步当前视频帧的显示时间戳到音频时钟上的方式来实现同步。从上文可知，画面的实时更新是通过向主线程中的事件循环中投入刷新图像事件来完成的，在对应的事件处理函数中，首先从待显示图像队列中取出一个图像数据节点 PictureNode，从中取出该图像的 PTS，通过图 4-16 中所示的同步视频到音频时钟的算法预测下一帧图像的播放延时，继续向事件循环中按照这个延时投入一个刷新图像事件，用于显示下一帧图像。接着将本次待显示的图像从节点中提取出来，并用 SDL 的渲染机制完成当前帧的同步播放。在同步算法中，Delay 用于表示对于下一帧待显示图像与当前显示的图像帧之间的延迟时间，frame_timer 表示下一帧的显示时刻，用于和操作系统当前时钟进行比较完成下一帧待显示图像的延时的计算。

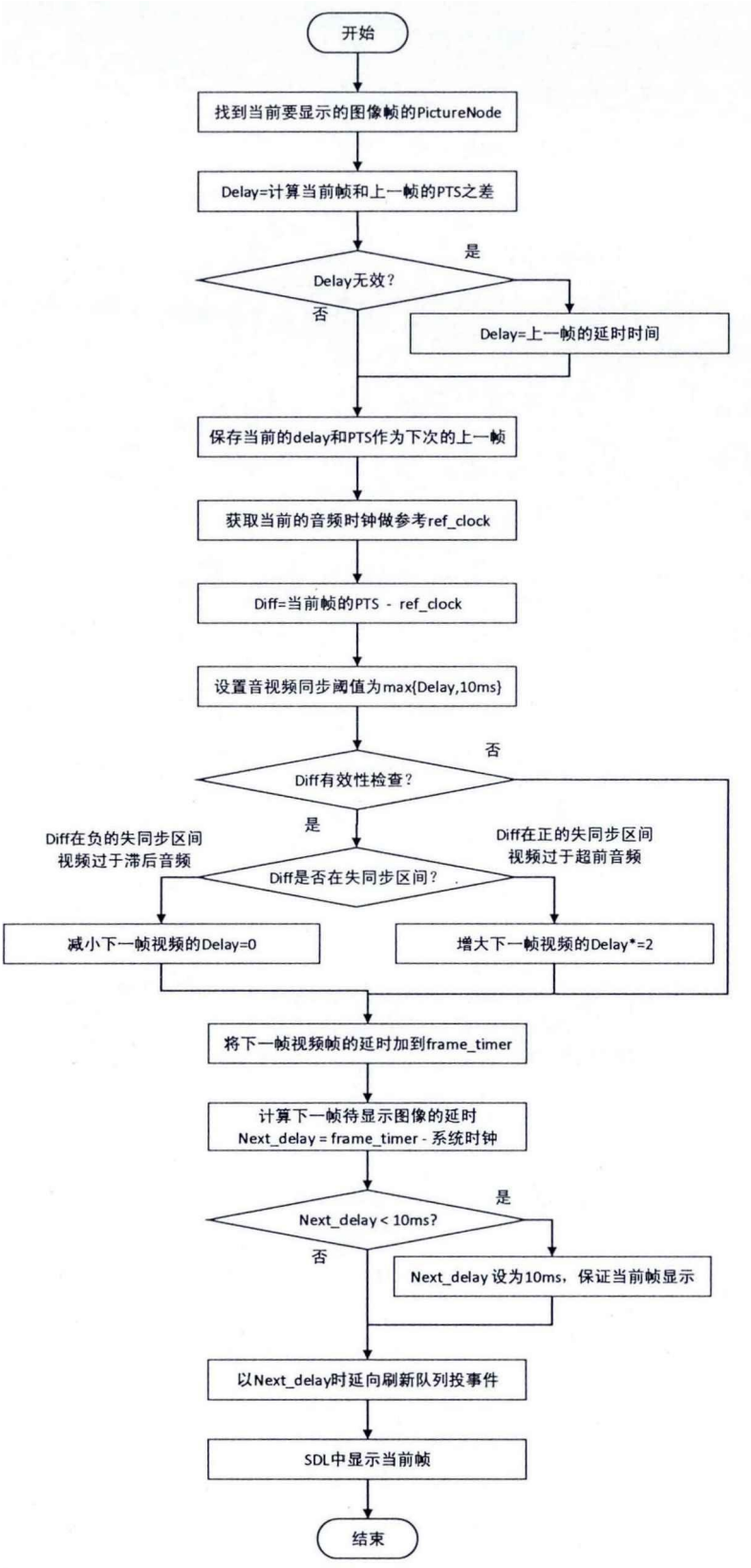

图 4- 16 同步视频到音频时钟的算法

Figure 4- 16 Algorithm for synchronous video to audio clock

4.4 本章小结

本章通过对流媒体传输系统进行总体上的设计，引入了基于改进的 Live555 的流媒体服务端和基于 FFMPEG 解码和 SDL 播放的可跨平台的流媒体音视频同步技术。通过对 Live555 进行功能扩展与性能优化，保证了其在流媒体传输系统中的功能完整性，分析多线程模型下基于时间戳进行媒体间同步的过程，并且结合 Android 平台，详细阐述了使用 NDK 工具进行相关开源库的移植，最后对客户端的音视频同步控制与播放过程进行设计与实现。

第五章 系统测试

在上一章中已经介绍了整个流媒体传输系统的详细的开发过程，本章针对上一章中开发的流媒体传输系统的运行平台进行介绍，并完成功能测试，最后针对功能测试结果进行分析与总结。

5.1 系统软硬件环境及平台搭建

在系统的实际开发过程中，首先在 ARM 嵌入式开发板上移植嵌入式 Linux 操作系统，并添加 Video For Linux 等框架，使得其支持网络摄像头，以及移植无线网卡的设备驱动，使其通过无线网卡接入到无线局域网中，并将所有相应的用于视频编码以及传输过程中的开源库通过交叉编译移植到嵌入式开发板上。其中使用到的开发板 Tiny6410 中的处理器是三星公司的基于 ARM1176JZF-S 处理芯片设计的 S3C6410，其内部自带非常强劲的多媒体处理单元，从硬件上原生支持 H.264 等格式的硬件编码，同时带有 3D 图形硬件加速器，可以支持 OpenGL ES 1.1 & 2.0 的加速渲染，以及 2D 图形图像的平滑缩放，翻转等操作。

表 5-1 Tiny6410 硬件参数

Table 5-1 hardware parameters of Tiny6410

性能标称	性能参数
处理器型号、主频	Samsung S3C6410A，ARM1176JZF-S 核 533MHz
内存 RAM	256MB DDR RAM
FLASH 存储	1GB SLC NandFlash
操作系统	基于 Linux2.6.38 内核

客户端使用的是魅族公司的 MX5 型号的手机，在其上如前文所述使用 NDK 等工具交叉编译并移植了 FFMPEG 库，用于扩展多媒体处理功能。该型号智能手机的系统性能参数如下表 5-2 所示。

表 5-2 MX5 手机的系统参数

Table 5-2 System parameters of MX5 mobile phone

性能标称	性能参数
处理器型号、主频	联发科 Helio X10 2.2GHz 八核 Cortex-A53
内存 RAM	3GB DDR RAM

主摄像头	2070 万像素
操作系统	基于 Android5.0

5.2 系统测试

5.2.1 Live555 的实时转发功能测试

通过在 Android 客户端输入 RTSP 协议播放地址，rtsp://192.168.1.43/live.sdp，向服务端来请求实时音视频流数据，服务端则从前端摄像头取数据，并进行实时视频监控画面的播放。功能测试结果如图 5-1 所示。

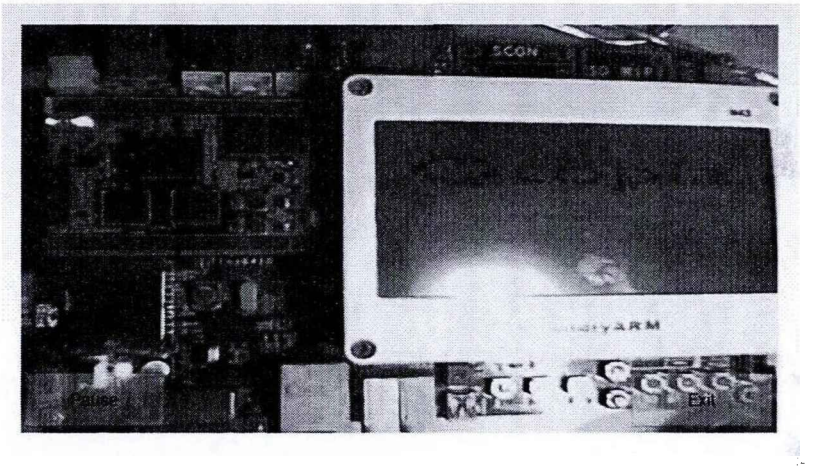

图 5-1 Live555 实时转发视频流功能测试

Figure 5-1 Real time video stream function test for Live555

5.2.2 Live555 对于 MP4 格式的支持

为了测试开发的流媒体服务器对于 MP4 格式的流式播放的功能，在 Android 客户端通过访问 rtsp 协议地址 rtsp://192.168.1.43/live.mp4，经网络连接后开始出现视频画面，画面效果如图 5-2 所示。

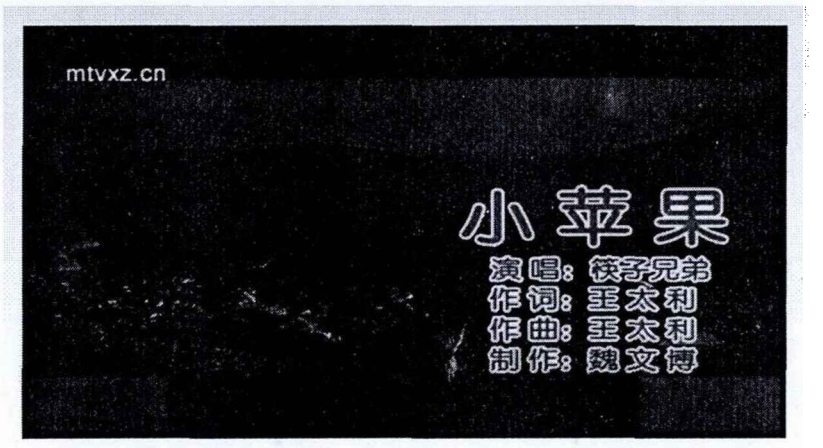

图 5-2 Live555 对于 MP4 格式文件的支持

Figure 5-2 Live555 support for MP4 format files

5.2.3 客户端对本地特殊视频编码格式支持视频功能测试

基于 FFMPEG 开发的 Android 客户端可以支持几乎所有的多媒体编码格式和封装格式。而 Android 本地对于音视频文件的支持是有限的。根据 Android 官方文档，使用 Android 系统本身自带的多媒体处理 API 仅仅可以对 H.263、H.264、MPEG-4 SP 以及 VP8 这几种视频编码格式进行解码，根据官方文档以及实验结果可知，对于视频编码的新标准如 H.265 都不支持。而本文开发的基于 FFMPEG 的多媒体播放器则可以实现对本地视频格式的全支持，如图 5-3 所示是测试对于采用 H.265 标准进行视频编码的多媒体文件的解码支持的结果。

图 5-3 使用 Android 客户端播放本地 H.265 编码格式视频文件

Figure 5-3 Local H.265 encoding format video file played by client using Android

5.3 测试结果分析

在对摄像头的实时数据流的点播功能测试中，服务端能够完成实时转发从摄像头和麦克风上采集的经过压缩的音视频数据流的功能。在基于 Live555 的流媒体服务端的 MP4 文件格式支持的功能测试中，在嵌入式开发板上拷贝 MP4 文件，并通过客户端使用 RTSP 协议进行点播，能够正常的播放。在客户端应用对本地视频文件全格式支持的功能测试中，测试的全部视频格式都能够正常播放。在客户端播放本地音视频同步格式的功能测试中，使用 Android 原生不支持的视频编码格式的文件进行测试，能正常播放，实验结果表明，基于 FFMPEG 编解码的 Android 客户端具有 FFMPEG 库的全格式支持的特点，极大地扩展了 Android 的多媒体格式支持。通过对音视频编码数据中具有完整 PTS 时间戳的 H.264 和 AAC 音视频文件进行本地视频文件同步测试，结果为主观上感受不到音视频不同步，表明在客户端进行的基于 PTS 的音视频同步算法能够满足系统中的音视频同步需求，具有很好的同步效果。

5.4 本章小结

本章先对本文开发的音视频传输系统运行的软件和硬件环境进行了介绍，接着在嵌入式开发板上对基于改进的 Live555 项目的流媒体服务器进行功能测试，对于 Android 平台上开发的流媒体客户端的各项功能进行测试。经测试，各项功能均达到了预期的需求，该流媒体传输系统具有很好的可行性。

总结与展望

本文对家庭服务机器人中的音视频传输系统进行了研究，主要针对音视频传输过程中存在的音视频失同步问题，通过查阅文献和分析得出引起音视频间失同步现象的原因，提出先通过对网络状况进行检测，在判定网络当前延迟状况的基础上，进行音视频同步方案的选择。并针对高延迟网络提出了一种贯穿整个音视频传输过程的利用音视频同步数据节点和同步时间戳进行同步控制的方案，并基于此搭建仿真实验环境，进行传输实验，实验结果表明提出的这种针对高延迟的音视频同步方案具有非常好的音视频同步效果。针对家庭服务机器人中音视频传输应用场景，实现了一个基于改进的 Live555 开源项目和 Android 平台的流媒体传输解决方案，该系统中包含了流媒体服务器以及基于 FFMPEG 的 Android 客户端。其中服务器端主要通过针对 Live555 在工程实践中存在的诸多问题，在深入分析该项目的代码的基础上，针对基于 select 模型作为输入输出方案存在的性能问题等进行优化，使用 Linux 下的 epoll 模型进行重新实现，针对支持的视频格式过少进行扩展，使用 FFMPEG 实现了 Live555 对于 MP4 文件格式的支持，针对实时流的转发进行实现。在客户端针对 Android 平台对于音视频编解码格式和多媒体文件格式支持有限的问题，提出将多媒体开源项目 FFMPEG 移植到 Android 平台上进行多媒体文件格式和编码格式的扩展，并通过开发多媒体应用程序实现实时视频的软解码，同时基于此库和开源的跨平台多媒体框架 SDL 设计并实现了客户端的音视频同步算法，实现了 Android 平台上的音视频同步播放。最后针对设计的音视频传输系统进行功能测试，测试结果表明本文设计的音视频传输系统能够很好地工作，同时客户端的音视频同步算法具有非常好的跨平台特性，可以基于此扩展到更多的客户端平台，具有很好的工程参考意义。

在撰写论文的过程中主要做了以下的工作：

1、通过大量阅读流媒体技术文献和网络技术博客，对文中使用到的技术进行了一个整体的概述与归纳。其中包括流媒体技术的各个部分，流媒体传输协议的种类，常见的音视频编解码格式，多媒体文件格式，以及常用的流媒体开源项目。熟悉这些开源的流媒体项目的使用和编程接口，并且针对 Live555 的源码进行了深入分析。

2 在大量阅读相关的文献的基础上，分析现有的音视频同步方案，针对引起音视频失同步的原因进行分析。针对网络状况情况复杂的问题，提出了一种基于网络延时

检测的自适应音视频同步方案。该方案通过将网络时延波动的方差和 RTCP(Realtime Transport Control Protocol)的对网络状况的拥塞控制等结合起来，完成网络状况的判定，并基于此选择不同网络状况下的同步方案。在高延迟网络下采用基于同步数据节点和时间戳的音视频同步技术，而在低延迟网络下采用接收端基于时间戳的同步技术。并搭建实验平台对该方案进行仿真实验，实验结果表明能很好的实现音视频同步。

3、基于 Live555 开源项目在具体应用场景中存在的不足与缺陷，在深入分析源代码之后，提出解决办法，对源代码进行了大幅度的改进与扩展，将基于 select 的 IO 模型替换成 Linux 系统下独有的 epoll 模型，将任务调度器的执行性能大大地提高了，改进了多线程，扩展了视频支持的格式，实现了实时媒体流的转发。

4 通过移植 FFMPEG 和 SDL 到 Android 系统，基于此开发应用程序，其中主要完成了 Android 平台对第三方解码库调用的 JNI 代码的实现，实现了 Android 平台下基于时间戳的音视频同步算法，而且该音视频同步方法具有很强的平台独立性，可以轻易地移植到其他平台使用。并最终借助 SDL 可跨平台的强大的多媒体功能，完成了解码后的多媒体数据的播放。

本文提出的流媒体传输方案，从前端摄像头到服务端采用 Live555 实现实时媒体流直播，到在 Android 客户端实现解码播放的整个过程中都进行了深入的研究，对于 Live555 的性能优化起到了很好的作用。

但是，对于用户量呈指数增长，数据规模变得越来越大的移动互联网时代，需要从后台服务器端整体架构上以及分布式计算技术上对服务端的整体架构以及设计进行更为大规模的优化与扩展。由于本论文还只是处于初步研究阶段，后续还将继续对于大数据量高并发高性能服务器进行深入研究与完善。

参考文献

[1] 赵辛丽. 新科技生活的“侵略者”--家用智能机器人[J]. 机器人技术与应用, 2012(2):12-13.

[2] 李骁宇. 智能家居服务机器人记忆库的设计与实现[D]. 哈尔滨工业大学, 2010.

[3] 冀龙涛. 基于 Android 手机的家用机器人控制技术研究[D]. 哈尔滨工业大学, 2013.

[4] Lee C L, Chen H T, Lin Y B, et al. Semi-automatically simultaneous localization and mapping for home service robots[C]// International Conference on Advanced Robotics and Intelligent Systems. IEEE, 2015.

[5] Liu C Y, Hung T H, Cheng K C, et al. HMM and BPNN based speech recognition system for home service robot[C]// International Conference on Advanced Robotics and Intelligent Systems. 2013:38-43.

[6] Wang Y H，Cheng H E，Lin C J，et al.Realization of affine SIFT real-time image processing for home service robot[C]// System Science and Engineering (ICSSE), 2013 International Conference.

[7] 徐凯. 基于网络的家庭安保机器人远程控制系统的分析与实现[D]. 复旦大学, 2010.

[8] 黄凯奇, 陈晓棠, 康运锋,等. 智能视频监控技术综述[J]. 计算机学报, 2015(6):1093-1118.

[9] 2013-2018 Chinese intelligent video surveillance market forecast report of supply and demand situation and development prospect(in Chinese)（2013-2018 年中国智能视频监控市场供需现状及发展前景预测报告）

[10] 文军, 张思峰, 李涛柱. 移动互联网技术发展现状及趋势综述[J]. 通信技术, 2014(9):977-984.

[11] 柴若楠等.音视频同步技术综述[J].计算机系统应用,2011,20(11):223-226

[12] Radhakrishnan R, Terry K, Bauer C. Audio and video signatures for synchronization[C]// Multimedia and Expo, 2008 IEEE International Conference on. IEEE, 2008:1549-1552.

[13] Escobar J, Partridge C, Deutsch D. Flow synchronization protocol[J]. Networking IEEE/ACM Transactions on, 1994, 2(2):111-121.

[14] Aggarwal S, Jindal A. Comprehensive overview of various lip synchronization techniques[C]// Biometrics and Security Technologies, 2008. ISBAST 2008. International Symposium on. IEEE, 2008:1-6.

[15] ZHANG JingFeng，LI Ying，WEI YanNa.Using timestamp to realize audio-video synchronization in real-time streaming media transmission[C] .Proceedings of the 2008 IEEE International Conference on Audio, Language and Image Processing.Piscataway：IEEE Press，2008：1073-1076.

[16] El-Helaly M, Amer A. Synchronization of Processed Audio-Video Signals using Time-Stamps[C]// Proceedings / ICIP ... International Conference on Image Processing. 2007:VI-193 - VI - 196.

[17] Radhakrishnan R, Terry K, Bauer C. Audio and video signatures for synchronization[C]// Multimedia and Expo, 2008 IEEE International Conference on. 2008:1549-1552.

[18] 尹秋来, 王宏霞, 赵杨. 一种基于预测模式的 H.264 视频信息隐藏算法[J]. 光电子:激光, 2012(11):2194-2199.

[19] 徐达文, 王让定. 一种基于预测模式的 H.264/AVC 视频信息隐藏改进算法[J]. 光电工程, 2011, 38(11):93-99.

[20] 郑建峰, 唐建, 郭立. 一种基于快速重编码的H.264/AVC压缩域信息隐藏方案[J]. 中国科学技术大学学报, 2013, 43(1):35-41.

[21] 刘添. 基于 H.264 可变尺寸帧间预测的音视频同步算法研究[D]. 吉林大学, 2011.

[22] 时美强, 李冰, 熊军,等. 基于 H.264/AVC 的音视频同步压缩方法[J]. 电视技术, 2009, 33(10):15-17.

[23] Bertoglio L, Leonardi R, Migliorati P. Intermedia synchronization for videoconference over IP[J]. Signal Processing Image Communication, 1999, 15(s 1–2):149-164.

[24] 陈运德, 张灿, 陈德元. 空间多媒体通信中音视频同步技术研究[J]. 计算机仿真, 2010, 27(6):130-134.

[25] 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, 32(7):2523-2526.

[26] 茅炎菲. 基于 RTSP 协议和 H.264 编码的网络视频监控系统的设计与实现[D]. 浙江大学, 2011.

[27] 刘大红. 基于 RTSP 流媒体服务器的设计与实现[D]. 西安电子科技大学, 2013.

[28] He J, He J. The research of plug-in extension technology based on Android multimedia player platform[C]// Computer Science and Service System (CSSS), 2011 International Conference on. IEEE, 2011:874-877.

[29] Fu X, Wu X, Song M, et al. Research on Audio/Video Codec Based on Android[C]// Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on. IEEE, 2010:1-4.

[30] Fan B, Sun G, Liu M, et al. Image acquisition and transmission in the video telephone based on android[C]// International Conference on Fuzzy Systems and Knowledge Discovery. IEEE, 2015.

[31] 杨飞. 基于 Android 的家用移动视频监控系统的研究与实现[D]. 广东工业大学, 2013.

[32] 詹雪峰. 流媒体系统同步机制和缓冲机制的研究与应用[D]. 电子科技大学, 2006.

[33] Surhone L M, Tennoe M T, Henssonow S F, et al. Real Time Streaming Protocol[J]. Betascript Publishing, 2011.

[34] Oh B H, Han J, Kim K, et al. A New Receiver-Based Retransmission Scheme with TFRC[J]. IEEE Communications Letters, 2012, 16(12):2091-2094.

[35] Handley M, Floyd S, Padhye J, et al. TCP Friendly Rate Control (TFRC): Protocol Specification[J]. Sonstiges, 2003(1):146–159.

[36] 柴若楠, 曾文献, 张鹏云. 音视频同步技术综述[J]. 计算机系统应用, 2011, 20(11):223-226

[37] ITU-R BT.1359-1 RELATIVE TIMING OF SOUND AND VISION FOR BROADCASTING [S]. 1998.

[38] 张昕, 吕凝, 王春雷,等. 多媒体同步传输方法研究[J]. 吉林大学学报:信息科学版, 2009, 27(6):573-578.

[39] L Bertoglio,R Leonardi,P Migliorati Intermedia synchronization for videoconference over IP[J] IEEE Transaction on signal Processing : Image Communieation,1999,15(1):149-164

[40] 薛彬,徐京等.一种改进的基于时间戳的空间音视频同步方法[J].电子设计工程.2013, 21(11):88-93

攻读学位期间发表的论文

[1] 张宇，曾碧.一种基于 FFMPEG 的音视频同步算法[J].广东工业大学学报,已录用待发表

学位论文独创性声明

本人郑重声明：所呈交的学位论文是我个人在导师的指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明，并表示了谢意。本人依法享有和承担由此论文所产生的权利和责任。

论文作者签名：张宇　　日期：2016.5.28

学位论文版权使用授权声明

本学位论文作者完全了解学校有关保存、使用学位论文的规定，同意授权广东工业大学保留并向国家有关部门或机构送交该论文的印刷本和电子版本，允许该论文被查阅和借阅。同意授权广东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、扫描或数字化等其他复制手段保存和汇编本学位论文。保密论文在解密后遵守此规定。

论文作者签名：张宇　　日期：2016.5.28

指导教师签名：　　日期：2016.5.28

致 谢

在计算机学院老师的指导下，本研究得以顺利开展，感谢学院在硬件、软件上的大力支持。向所有在论文选题、研究、实验过程中给予帮助的老师、同学表示衷心的感谢。

在即将毕业之际，首先要向我的导师曾碧教授表示最衷心的感谢，要感谢她对于我的悉心关怀以及学习上的帮助。她严谨、敬业的态度使我受益匪浅。感谢她在这 3 年期间的帮助和关心，在她的支持下，硕士期间提高了科研能力和专业知识。

其次，我要感谢实验室的小伙伴们，感谢他们对于我工作和学习上的帮助，还要感谢我的师弟师妹，感谢他们彼此间的关心和帮助，让我感受到了集体的温暖。

在今后的工作、生活中，我将牢记各位老师的教导，不断提升自己，同时热心帮助师弟师妹以及工作的同事。

最后，感谢各位专家和教授对本文的评阅！

张宇

2016 年 4 月中国广州