



北京大学

博士研究生学位论文

题目： 自适应视频流媒体中的关键技术研究

姓 名： 孟胜彬

学 号： 1101111141

院 系： 信息科学技术学院

专 业： 计算机应用技术

研究方向： 数字视频信息处理

导 师： 郭宗明 研究员

二〇一六年五月

版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则一旦引起有碍作者著作权之问题，将可能承担法律责任。

摘要

近年来，计算机技术的发展使得移动设备逐渐普及，通信技术的发展使得互联网无处不在。在这样的条件下，通过网络随时随地观看视频成为可能，而且逐渐发展为视频内容消费的主要形式。这种无需先下载好全部视频数据而是一边下载一边播放的技术称为视频流媒体。视频流媒体的相关应用，如按需点播、实时直播、在线教育、视频会议等，已经成为人们生活中不可或缺的部分。在这前所未有的机遇同时，视频流媒体也面临着网络异构性和带宽波动的挑战。为应对这一挑战，视频流媒体系统需要能够根据不同的网络条件自动调整所发送的数据码率，以适应带宽的变化。本文^①从数据源和数据传输两方面入手，结合点播和直播两种应用模式，对自适应视频流媒体中的关键技术进行了较为全面深入的研究。首先，本文针对可伸缩视频数据源提出了新的失真模型和码流截取方案，在支持可变码率的同时提供尽可能高的视频质量；其次，本文为视频点播系统设计了新的码率自动调整策略，用控制论的方法来解决如何适应带宽变化的问题；最后，本文详细分析了现在非常流行的视频直播系统的传输过程，结合直播的特点提出了数据上传时的码率自适应算法。本文主要的创新性贡献可以归纳为如下三个部分：

1. 采用线性误差模型的可伸缩视频码流截取方案

对于可伸缩视频数据源，本文首先推导并验证了一个线性误差模型，用于准确估计丢弃任意数据包组合带来的失真变化；然后采用该模型设计了一个贪心型算法来根据每个数据包的码率和失真影响对其赋优先级，作为截取子流时丢弃数据包的顺序。相比于参考软件，这一码流截取方案能够在同样的复杂度和码率限制下取得更高的视频质量。

2. 基于 PID 控制思想的点播系统码率自适应算法

本文基于经典的比例-积分-微分（PID）控制思想，为视频点播系统的数据传输过程提出了一个综合考虑带宽的历史状况、当前状态和未来趋势的码率自适应算法，既能充分利用带宽，传输较高的视频质量，又能减小带宽波动的影响，保证视频质量的平滑性。该算法集成在了苹果公司 QuickTime 流媒体服务器的开源版本上，将发送的视频平均质量提高了 8.6%，质量波动降低了 24.8%。

3. 基于缓冲区分析的直播系统码率自适应算法

为给视频直播中的数据上传阶段增加码率自适应的特性，本文首先详细分析了系统整个传输过程中各个缓冲区的关系，建立了一个多缓冲区模型；然后把上述点

^①本研究得到国家自然科学基金（项目编号 61271020）和国家科技支撑计划（项目编号 2014BAK10B02）资助。

播系统中用到的 PID 方法与多缓冲区模型相结合，提出了一个有效的上传过程码率自适应算法。相比于没有自适应的上传过程，带宽的利用率得到了提升，视频播放的连续性也得到了改善。

关键词：视频流媒体，码流截取，码率自适应，视频直播

Research on Key Technologies of Adaptive Video Streaming

Shengbin Meng (Computer Science)

Directed by Prof. Zongming Guo

ABSTRACT

In recent years, the development of computer technology has made the mobile devices popular, and the communication technology has made the Internet accessible everywhere. Under such circumstances, watching videos at any time and any place becomes possible, and even an increasingly important way for people to consume video content. This is called video streaming, where the video can play as the data are being transmitted, before the entire file has been downloaded. Applications of video streaming, e.g., video on demand (VoD), live broadcasting, online education, and remote video conference, have become an indispensable part of people's life. Along with those opportunities, video streaming also has the big challenge brought about by the variety of networks. To cope with this challenge, the video streaming system should be able to adjust the video's bitrate or quality according to the network condition. In other words, the video streaming system needs to be adaptive. In this paper, we investigate and solve the key problems in adaptive video streaming. First, focusing on the adaptive video streaming system based on the Scalable Video Coding (SVC) extension of the H.264/AVC video coding standard, this paper proposes a novel bitstream extraction scheme to provide highest possible video quality while supporting bitrate adjustment at the same time. Second, this paper designs a new rate adaptation algorithm for the VoD system, adjusting the bitrate to fit the current bandwidth from the control perspective. And finally, this paper analyzes the transmission process of live video streaming in detail and proposes a rate adaptation algorithm for its uploading stage. The innovative contributions of this paper can be summarized as follows.

1. Bitstream extraction scheme utilizing a linear error model

For scalable video data source, a simple and effective linear error model is proposed and verified, which can be used to accurately estimate the distortion caused by discarding any combination of data packets from an SVC bitstream. Then utilizing this model, a

greedy-like algorithm is designed to assign priority for each data packet according to its Rate-Distortion (R-D) impact, thus enabling optimized bitstream extraction. Comparing with the reference software, this extraction scheme achieves higher video quality with the same computational complexity and bitrate constraint.

2. Rate adaptation algorithm for VoD systems based on the idea of PID control

Based on the classical Proportional-Integral-Derivative (PID) controller, a rate adaptation algorithm is proposed for the data transmission process of VoD systems. By monitoring and predicting past, current and future bandwidth information, the algorithm is able to reduce quality fluctuation while still preserving a high quality level. Integrated into the open source version of Apple's QuickTime streaming server, this algorithm increases the streamed video's quality by 8.6% and decreases the quality variance by 24.8% at the same time.

3. Rate adaptation algorithm for live streaming systems based on analysis of data buffers

This paper also proposes to add adaptation for the data uploading stage of live video streaming systems. A multi-buffer model is built based on analysis of the several data buffers during the transmission, and it is combined with the PID method to provide rate adaptation effectively. Compared with non-adaptive uploading, the bandwidth utilization is increased and the playback continuity is improved.

KEYWORDS: Video streaming, Bitstream extraction, Rate adaptation, Live streaming

目录

| | |
|----------------------------------|-----------|
| 第一章 绪论 | 1 |
| 1.1 研究背景和意义 | 1 |
| 1.2 研究框架和问题 | 2 |
| 1.3 本文研究内容和主要贡献 | 5 |
| 1.4 本文的结构安排 | 7 |
| 第二章 研究基础和相关工作 | 9 |
| 2.1 视频的压缩编码 | 9 |
| 2.1.1 H.264/AVC 视频编码标准 | 10 |
| 2.1.2 可伸缩视频编码 | 13 |
| 2.2 视频文件的流式传输 | 16 |
| 2.2.1 典型的传输协议 | 17 |
| 2.2.2 流媒体服务器 | 19 |
| 2.3 可伸缩视频码流截取 | 21 |
| 2.3.1 问题分析 | 21 |
| 2.3.2 相关工作简介 | 22 |
| 2.4 传输中的码率自适应 | 24 |
| 2.4.1 问题分析 | 24 |
| 2.4.2 相关工作简介 | 25 |
| 2.5 本章小结 | 26 |
| 第三章 采用线性误差模型的可伸缩视频码流截取方案 | 29 |
| 3.1 线性误差模型 | 29 |
| 3.1.1 模型推导 | 29 |
| 3.1.2 误差向量获取 | 31 |
| 3.1.3 失真估计与模型验证 | 32 |
| 3.2 码流截取方案 | 33 |
| 3.2.1 数据包优先级度量 | 33 |
| 3.2.2 优先级赋值的贪心算法 | 35 |
| 3.2.3 无参考源的情况 | 36 |
| 3.2.4 优先级值的实际使用 | 37 |

| | |
|------------------------------------|-----------|
| 3.3 实验结果 | 37 |
| 3.3.1 实验配置 | 37 |
| 3.3.2 结果分析 | 38 |
| 3.3.3 无参考截取实验 | 40 |
| 3.4 本章小结 | 42 |
| 第四章 基于 PID 控制思想的点播系统码率自适应算法 | 43 |
| 4.1 PID 控制器简介 | 43 |
| 4.2 基于 PID 的码率自适应 | 44 |
| 4.2.1 质量等级划分 | 44 |
| 4.2.2 过程变量和控制目标 | 44 |
| 4.2.3 控制模型和算法描述 | 45 |
| 4.2.4 参数选取与调优 | 47 |
| 4.3 实验结果 | 49 |
| 4.3.1 实验配置 | 49 |
| 4.3.2 结果分析 | 50 |
| 4.3.3 与码流截取相结合的效果 | 54 |
| 4.4 本章小结 | 54 |
| 第五章 基于缓冲区分析的直播系统码率自适应算法 | 57 |
| 5.1 视频直播系统概述 | 57 |
| 5.2 直播中的多缓冲区模型 | 58 |
| 5.2.1 TCP 缓冲区 | 58 |
| 5.2.2 应用层发送缓冲区 | 59 |
| 5.2.3 播放缓冲区 | 60 |
| 5.2.4 多缓冲区模型分析 | 61 |
| 5.3 码率自适应算法 | 62 |
| 5.3.1 最小可调度单元 | 62 |
| 5.3.2 过程变量选取 | 63 |
| 5.3.3 算法详情 | 64 |
| 5.4 实验结果 | 64 |
| 5.4.1 实验配置 | 64 |
| 5.4.2 结果分析 | 65 |
| 5.5 本章小结 | 68 |

目录

| | |
|-----------------------------|-----------|
| 第六章 总结和展望 | 69 |
| 6.1 本文工作总结 | 69 |
| 6.2 未来工作展望 | 70 |
| 参考文献 | 73 |
| 致谢 | 79 |
| 个人简历和在学期间研究成果 | 81 |
| 北京大学学位论文原创性声明和使用授权说明 | 83 |

插图

| | | |
|------|--|----|
| 1.1 | 视频流媒体研究框架 | 3 |
| 1.2 | DASH 系统的工作原理示意图 | 4 |
| 1.3 | YouTube 视频码率自适应功能图示 | 5 |
| 1.4 | 优酷网视频码率自适应功能图示 | 6 |
| 2.1 | H.264/AVC 中的视频编码层和网络抽象层 | 10 |
| 2.2 | H.264/AVC 采用的混合视频编码框架 | 11 |
| 2.3 | SVC 中通过层次化预测实现时间可伸缩 | 13 |
| 2.4 | SVC 中空间可伸缩的实现原理 | 15 |
| 2.5 | SVC 中的 NALU 头结构 | 16 |
| 2.6 | 典型的流媒体应用框架 | 17 |
| 2.7 | 流媒体服务器的组件及其与外界的交互 | 20 |
| 2.8 | 可伸缩视频码流截取示意图 | 21 |
| 2.9 | JSVM 基本截取器的算法图示 | 23 |
| 2.10 | 基于训练的模型所取得的失真估计准确度图示 | 24 |
| 2.11 | 基于不同优先级缓冲区的码率自适应算法 | 26 |
| 3.1 | 增强层数据包集合关系示例 | 30 |
| 3.2 | SVC 码流中的图像组 (GOP) 结构示例 | 31 |
| 3.3 | 不同序列估计失真与实际失真的比较 | 34 |
| 3.4 | 使用优先级值进行实际码流截取的程序流程图 | 38 |
| 3.5 | 三种码流截取方案的性能比较 | 39 |
| 3.6 | 无参考源时三种码流截取方案的性能比较 | 41 |
| 4.1 | PID 控制器原理示意图 | 44 |
| 4.2 | 达尔文流媒体服务器中的两个时间线 | 45 |
| 4.3 | PID 控制器中参数 K_i 、 K_d 对系统的影响 | 48 |
| 4.4 | 长/短期预测算法与达尔文服务器中的包延迟反馈 (PDF) 算法的比较 . | 51 |
| 4.5 | 本文提出的 PID 控制算法与达尔文服务器中的包延迟反馈 (PDF) 算法 的比较 | 52 |
| 4.6 | 不同质量控制算法下视频质量等级随时间的变化情况 | 53 |

| | |
|---|----|
| 4.7 不同流媒体系统所发送视频的 PSNR 随时间的变化情况 | 55 |
| 5.1 通用的手机直播系统模型 | 58 |
| 5.2 简化的直播系统传输架构 | 59 |
| 5.3 直播系统接收端数据关系 | 60 |
| 5.4 多缓冲区模型结构图 | 62 |
| 5.5 直播系统在不同测试条件下码率随时间变化的情况 | 66 |
| 5.6 直播系统在不同测试条件下播放时间占比的变化情况 | 67 |

表格

| | |
|--|----|
| 2.1 H.264/AVC 中的 NALU 结构 | 12 |
| 3.1 采用线性误差模型进行不同序列失真估计的估计误差 | 33 |
| 3.2 本文提出的截取方案相比于 JSVM 的 PSNR 提升 | 40 |
| 3.3 无参考源时本文提出的截取方案相比于 JSVM 的 PSNR 提升 | 42 |
| 3.4 不同的无参考截取方法对比：相对于 JSVM QL 的平均 PSNR 提升 | 42 |
| 4.1 质量等级定义中组 ID 与层 ID 的对应关系 | 50 |
| 4.2 不同质量控制算法与包延迟反馈算法的定量比较 | 53 |
| 4.3 采用本文算法的系统与原始系统 PSNR 均值与方差的比较 | 54 |
| 5.1 直播系统实验中的带宽利用率 | 68 |
| 5.2 直播系统实验中的播放时间占比 | 68 |

第一章 绪论

1.1 研究背景和意义

进入二十一世纪以来，人们表示和传递信息的媒介从文本、声音扩展到了图像、视频。如今视频已经成为人们生产和生活中不可缺少的部分。另一方面，在计算机技术和通信技术的推动下，人们对网络的依赖和要求也越来越高。智能手机、平板电脑等移动设备的普及，Wi-Fi、4G 等无线网络的覆盖，让人们能够随时随地访问互联网。这样的环境催生并促进了一项技术的蓬勃发展，这就是视频流媒体。

视频流媒体简单来说就是通过网络在线播放视频。所谓流媒体，是指音视频等多媒体数据通过网络以连续稳定的流的形式传输到客户端的一系列技术、协议和方法的总称。在视频流媒体中，视频数据从服务器连续不断地传输到客户端，客户端可以一边接收一边播放，无需等待整个文件发送完毕。与下载方式相比，这种采用流式传输的视频播放具有显著的优点^[1]，包括：1) 启动延迟大大缩短，用户可以在等待几秒或十几秒的缓冲后就立即开始观看；2) 视频数据不在客户端长时间驻留，不仅节省了用户存储空间，也一定程度上避免了内容版权保护问题；3) 支持数据的实时生成和获取，大大扩展了视频应用场景的范围。正是由于其优秀的特性，视频流媒体得到了广泛的应用，逐渐成为人们消费视频内容的主要形式^[2]。

视频流媒体应用按照其对实时性要求的不同可以分为点播和直播两大类。在点播应用中，内容提供商将预先制作好的视频放在服务器上，并发布内容的描述信息和链接，用户选择自己感兴趣的内容请求播放相应的视频。典型的例子是在线视频网站，如国外的 YouTube^①、Netflix^②，国内的优酷^③、乐视^④等。大多在线教育网站如 Coursera^⑤、网易公开课^⑥等也属于视频点播类应用。在直播应用中，视频数据则是通过现场录制实时生成的，上传到流媒体服务器之后再即时分发给观看者，具有很强的时效性。这类应用包括现在互联网上正兴起的秀场、游戏、生活类直播软件，以及已经很成熟的远程监控、视频会议等等。直播应用往往也会把实时事件的内容存储起来，后续以点播的形式继续提供。

网络条件的改善，采集与播放设备的普及，使得视频流媒体应用进入了一个高速

^①<https://www.youtube.com/>

^②<https://www.netflix.com/>

^③<http://www.youku.com/>

^④<http://www.le.com/>

^⑤<https://www.coursera.org/>

^⑥<http://open.163.com/>

增长的时期。根据 Cisco 的一项预测^①，从 2014 年到 2019 年，视频流量在所有互联网流量中的占比将从 64% 上升至 80%。路透社的一篇报道^②指出，在美国这一比例在 2018 年即将达到 83%。可见，视频流媒体正迎来一个高峰期。

在前所未有的机遇同时，视频流媒体也面临着网络异构和带宽波动所带来的严峻挑战。虽然有学者在探索下一代网络架构^[3-5]，但现在传统的互联网架构仍将长期占据绝对主流地位。从当前的网络构成来看，既存在几百 kb/s 速度的拨号或 3G 上网，又有几十 Mb/s 速度的局域网或光纤接入，而且随着移动通信网与 Internet 的融合，无线与有线网络的混联，网络异构性更加明显。视频流媒体的终端用户可能处于不同的网络中（例如有线网，Wi-Fi，蜂窝移动网络等），甚至有可能在观看视频的过程中从一种网络切换到另一种网络（典型的例子是手机用户在 Wi-Fi 和 4G 网络之间的切换）。即使是在同一网络环境中，视频传输可用的带宽也会因为网络拥堵情况的变化而不断波动。

在这样的实际条件下，如果视频流媒体系统以恒定的速率传输数据，将很难或者不可能会给用户带来最好的视频观看体验。举例来说，如果设置一个较低的码率，则对应的视频质量不高，达不到带宽充足用户的满意度；而如果设置较高的码率，则对于低带宽的用户或者在带宽波动较大的情况下，可能会导致视频无法流畅播放。因此，为应对这一挑战，视频流媒体系统需要能够根据不同的网络条件自动调整所发送视频的码率，以适应带宽的变化。换句话说，视频流媒体需要具备自适应性。

考虑到视频流媒体的广泛应用，通过加入自适应性来进一步改善视频流媒体服务质量，具有重要的现实意义。自适应视频流媒体的研究既受到了工业界的普遍关注，也成为了学术界的一大热点。

1.2 研究框架和问题

自适应视频流媒体的研究属于视频流媒体研究领域的一个子集。视频流媒体是视频与流媒体的结合，因此这个领域的研究也涉及包括编码和传输在内的多个方面。如图1.1所示，视频流媒体的研究可以从系统模块的角度分为以下几个部分：

- 数据源端的研究，一方面是进行高效的压缩编码，以尽可能少的数据表示尽可能高的视频质量；另一方面是提供具备灵活性的码流，支持码率的变化甚至考虑容错，等等。
- 传输过程的研究，即如何又快又好地将视频数据送给用户，包括改善网络状况以

^①http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

^②<http://www.reuters.com/article/us-internet-consumers-cisco-systems-idUSKBN0EL15E20140610>

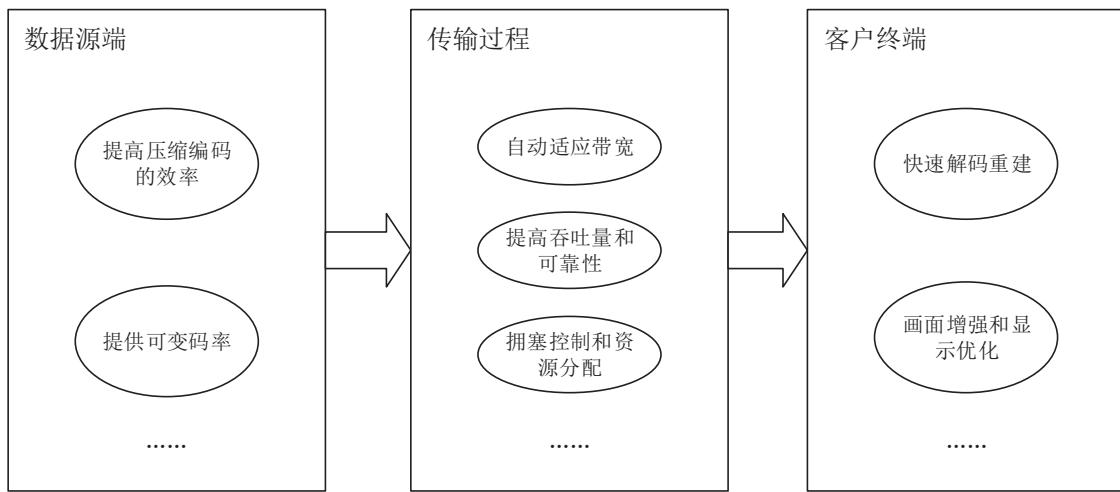


图 1.1 视频流媒体研究框架

提高传输的吞吐量和可靠性，合理公平地利用带宽资源，根据带宽情况自适应地调整传输速率等。

- 客户终端的研究，即优化用户设备收到视频码流之后的播放阶段，包括如何快速解码，进行一些画质增强等后处理过程从而更好地显示，等等。

在这一研究框架中，自适应视频流媒体主要涉及的是数据源端提供可变码率、传输过程中自动适应带宽这两个方面。虽然其他方面的研究工作，例如对编码效率的提升、对网络架构和协议的改进、在用户终端的解码优化等，也非常重要，但本文主要关注的是上述与自适应性相关的两个方面。

在视频流媒体系统中支持视频码率的可变性通常有两种选择。第一种选择是采取多码流的方法。多码流方法的原理很简单，就是预先编好不同码率的多个码流存放在服务器上，根据网络情况选取其中合适的一个进行传送。近年来兴起的 HTTP 动态自适应流媒体（Dynamic Adaptive Streaming over HTTP, DASH）^[6] 就属于这类方法。如图1.2所示，DASH 系统在服务器端提供不同码率的多个码流切片，客户端通过 HTTP 协议拉取数据，在某个时间段内可以选择性地接收这几个码流中任何一个的切片，通过在多个码流之间切换来更改码率以适应带宽波动。此外，有的系统会在服务器上将预先编好的一个高码率码流实时转码成不同低码率的码流，这本质上也属于多码流的方法。

第二种选择是采用可伸缩视频编码^[7] 技术。作为国际视频编码标准 H.264/AVC^[8] 的扩展之一，可伸缩视频编码的初衷就是在单个码流中加入伸缩性的支持，使其码率和视频质量能够根据需要改变。可伸缩视频码流中的数据被划分为基本层和增强层数据包，可以从中丢弃一部分增强层数据包而截取出一个子流，以牺牲视频质量的方式

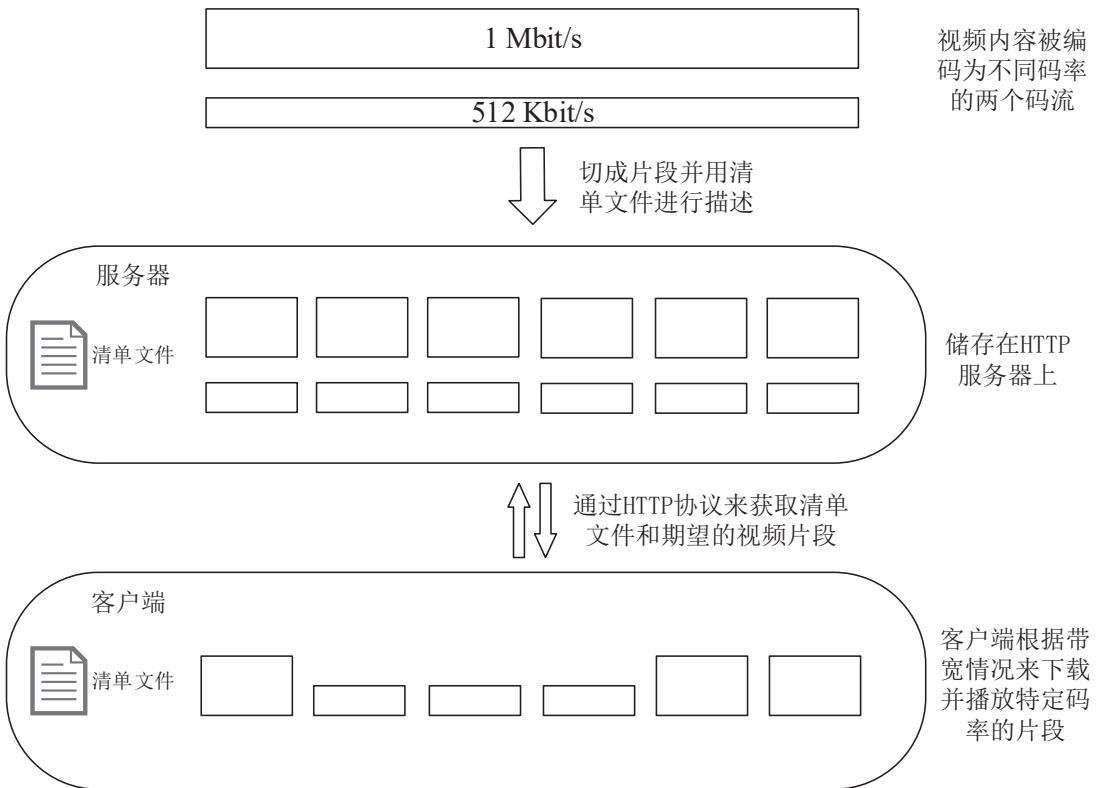


图 1.2 DASH 系统的工作原理示意图

来降低码率。因此，用它来实现码率可变性正符合其设计目的。

多码流方案的优点是无需对已有的程序和系统做较大的改动，因此比较易于部署^[9]。很多商业视频网站（如 YouTube、优酷等）已经用它实现了自适应功能。但是，多码流方案能提供的适应范围和粒度非常有限。例如，YouTube 网站上的视频最多只提供 144p、240p、360p、480p、720p 和 1080p 这六个等级（参见图1.3，其中右下角矩形框中显示了支持的等级），而优酷网上的视频大多只提供标清、高清、超清这三个画质（参见图1.4）。此外，多个码流的编码和存储需要大量的计算资源和磁盘空间，使得时间和空间开销成倍增加。要想以合理的代价提供精细无缝的自适应性，就需要采用可伸缩视频编码的方案。可伸缩视频码流能够以非常小的数据粒度进行码率调整，而且性能分析表明可伸缩视频编码的压缩效率远高于同一内容多次编码的多码流方法^[10]。正是因为这些优势，基于可伸缩视频编码来实现自适应视频流媒体从技术上来说是一个更好的方案，也吸引了学术界很多研究者的关注^[11-15]。

从图1.1所示的研究框架中可以看出，采用可伸缩视频编码或是采用 DASH 之类的多码流方案，其区别只在于数据源端提供可变码率的方式不同。当可伸缩视频作为数据源时，在给定码率下选择丢弃哪些数据包，即如何最优地从整个码流中截取出一个



图 1.3 YouTube 视频码率自适应功能图示

子流用于发送是需要研究的问题之一。多码流作为数据源时则不存在这个问题。而对于在传输过程中如何自动调整码率以适应带宽的变化，却是二者共有的问题。这些问题的解决，是提高视频流媒体服务质量和用户体验的关键。

1.3 本文研究内容和主要贡献

本文结合视频流媒体所面临的挑战，对自适应视频流媒体中的关键技术进行研究，以更好地解决上面所提到的问题。首先，本文针对可伸缩视频数据源提出了新的失真模型和码流截取方案，在支持可变码率的同时提供尽可能高的视频质量；其次，本文为基于可伸缩视频编码的视频点播系统设计了新的码率自动调整策略，用控制论的方法来解决如何适应带宽变化的问题；最后，本文详细分析了现在非常流行的视频直播系统的传输过程，结合直播的特点提出了数据上传时的码率自适应算法。本文主要的研究内容和创新性贡献可以归纳为如下三个部分：

1. 采用线性误差模型的可伸缩视频码流截取方案

作为码率适应带宽波动的前提条件，视频流媒体中的数据源需要能够灵活调整。可伸缩视频编码技术通过将数据划分为基本层和增强层并丢弃增强层的数据包来实现即时码率变化。从完整的可伸缩码流中丢弃部分数据得到一个子流的过程称为码流截取。本文以最小化特定截取码率限制下的视频失真为目标，首先提出了一个线性误差模型来估计丢弃任意数据包组合带来的失真变化，然后利用它设



图 1.4 优酷网视频码率自适应功能图示

计了一个贪心型算法来根据每个数据包的码率和失真影响对其赋优先级，作为截取过程中丢包的顺序。相比于参考软件，这一码流截取方案能够在同样的复杂度和码率限制下取得更高的视频质量。

2. 基于 PID 控制思想的点播系统码率自适应算法

自适应视频流媒体中的另一个关键问题是传输过程中的码率调整策略，即在可用带宽不断变化的情况下，决定何时调整码率并确定调整到多少。本文基于经典的比例-积分-微分（Proportional-Integral-Derivative, PID）控制思想，提出了一个综合考虑带宽的历史状况、当前状态和未来趋势的码率自适应算法，既能充分利用带宽，传输较高的视频质量，又能减小带宽波动的影响，保证视频质量的平滑性。该算法集成在了苹果公司 QuickTime 流媒体服务器的开源版本上，并在实际应用中取得了很好的效果。

3. 基于缓冲区分析的直播系统码率自适应算法

在视频直播中，由于数据是实时产生的，其传输过程与视频点播有所不同。视频数据需要先上传到服务器，然后由服务器分发到观看者进行播放。本文为这个过程中的上传阶段增加了码率自适应的特性：首先通过详细分析系统整个传输过程中各个缓冲区的关系，建立了一个多缓冲区模型；然后把上述点播系统中用到的 PID 方法与多缓冲区模型相结合，提出了一个有效的码率自适应算法。相比于没

有自适应的上传过程，带宽的利用率得到了提升，视频播放的连续性也得到了改善。

1.4 本文的结构安排

本文共分为六章，后续章节具体内容安排如下。

第二章概述自适应视频流媒体领域的研究基础和相关工作。首先对视频编码和流式传输的基础知识进行简要介绍，然后对本文所要研究的可伸缩视频码流截取、传输中的码率自适应这两个问题进行具体描述，并对已有的相关工作进行分析。

第三章讨论采用线性误差模型的码流截取方案。首先针对可伸缩视频推导并验证线性误差模型，然后介绍采用该模型的失真估计方法和以码率失真影响为度量标准的优先级赋值算法。最后展现并分析所提出的码流截取方案的实验结果。

第四章讨论基于 PID 控制思想的点播系统码率自适应算法。首先对 PID 控制器做简单的介绍，然后将 PID 模型运用到视频传输中的码率调整策略，提出了一个新颖的码率自适应算法。该算法被集成在了苹果公司 QuickTime 流媒体服务器的开源版本中，其有效性通过对比实验得到了验证。

第五章讨论针对直播系统的码率自适应算法。首先详细分析了直播系统的传输过程，提出了一个多缓冲区模型；然后将其与 PID 控制方法相结合，用缓冲区状态作为系统变量进行控制，实现了上传过程的码率自适应，改善了带宽利用率和视频播放的连续性。

第六章总结全文内容并对未来工作和应用前景进行展望。

第二章 研究基础和相关工作

本章首先对视频编码和传输领域的一些基础知识进行概述，为本文后续研究做准备；然后结合本文所涉及的研究内容，对要解决的问题进行详细分析，并对国内外研究现状和已有工作进行简要介绍。

2.1 视频的压缩编码

原始视频信号数据量过于庞大，必须经过压缩才能有效进行存储和传输。视频信号之所以能被压缩是因为其中存在大量冗余信息，包括空间冗余、时间冗余、统计冗余等^[16]。以仙农信息论^[17] 和率失真理论^[18] 为基础，通过预测（在空域或时域寻找相似信号，用与相似信号的残差来重建当前信号的一种压缩手段，如时间预测和空间预测）、变换（如离散余弦变换^[19]、哈达玛变换^[20]）、量化^[21]、熵编码（如变长编码^[22]、算术编码^[23]）等技术手段，可以去除这些冗余，以一定的失真为代价获得很高的数据压缩比。数字视频压缩编码技术经过了半个多世纪的发展，形成了一系列的国际标准。这些标准是包括视频流媒体在内的所有视频应用的基础。

最早的视频编码标准通常被认为是由国际电信联盟于 1990 年制定的 H.261^[24]。同一时期，由国际标准化组织和国际电工委联合成立的运动图像专家组（Moving Pictures Expert Group, MPEG）也开始了视频相关标准的制定工作，并于 1992 年推出了 MPEG-1^[25]。此后，以 MPEG-2^[26]、H.263^[27] 为代表的標準不断演进，大大推动了数字视频产业的发展。时至今日，国际电信联盟与国际标准化组织已经在标准制定方面互相合作，二者共同打造的 H.264/AVC 视频编码标准在蓝光光碟、高清数字视频光盘（Digital Video Disk, DVD）、数字电视广播、互联网视频共享和在线观看等各个领域都占据了主流地位，普及度极高。H.264/AVC 的后继者、新一代视频编码标准 HEVC（High Efficiency Video Coding）^[28] 也已于 2013 年发布，但因为其编解码复杂度高^[29] 且大多设备和系统尚未支持，所以还处于推广阶段^[30-32]。我国也发布了自主知识产权的视频编码标准 AVS (Audio Video coding Standard)^[33]，并进入了第二代 AVS 标准的制定阶段。考虑到应用的广泛程度，本文的研究是基于 H.264/AVC 及其可伸缩扩展的。但需要指出的是，随着视频编码技术的逐步成熟和定型，各种编码标准大同小异，因此本文的方法和结果可以很容易适配到其它标准上去。

作为本文的研究基础，下面依次对 H.264/AVC 视频编码标准和可伸缩视频编码进行介绍。

2.1.1 H.264/AVC 视频编码标准

H.264/AVC 是目前应用最广泛的国际视频编码标准。它的设计目标有两方面，一是显著提高压缩编码的效率，二是在网络化的趋势下满足对灵活性和可定制性的要求。为此，这一标准相对于之前的标准提出了两个新概念：视频编码层（Video Coding Layer, VCL）和网络抽象层（Network Abstract Layer, NAL）。视频编码层着重于高效地压缩和表示视频内容，而网络抽象层则用于封装视频编码层的数据并提供额外的头信息，使之适用于多样化的传输协议和存储介质。二者的关系如图2.1所示。

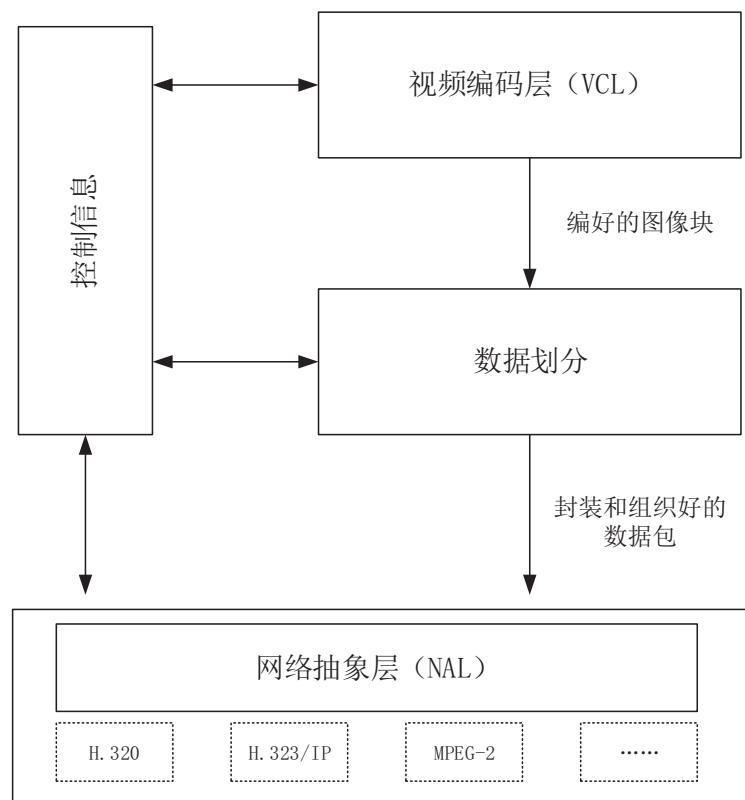


图 2.1 H.264/AVC 中的视频编码层和网络抽象层

在图2.1中，H.320 是国际电信联盟制定的关于在综合数字交换网上进行视频会议的标准，H.323 是 IP 电话数据传输方面的标准，MPEG-2 则规定了音视频文件存储格式，常用于广播电视网。可见，经过 NAL 层的抽象，H.264/AVC 编码得到的码流可以灵活适应各种应用。下面，分别对 VCL 和 NAL 进行介绍。

VCL 层的目标是高效率地表示视频内容。与其他视频编码标准一样，H.264/AVC 的 VCL 层采用了基于块的混合视频编码框架，如图2.2^[34] 所示。所谓“基于块”，是指将原始图像数据分成 16×16 大小的宏块（Macro Block, MB），接下来的编码过程将以宏

块为基本单位。对每一个宏块，先进行空间或时间预测；预测得到的残差数据进行类似于离散余弦变换（Discrete Cosine Transform, DCT）的整数变换，然后对变换系数进行量化、重排、熵编码，最终形成比特流。

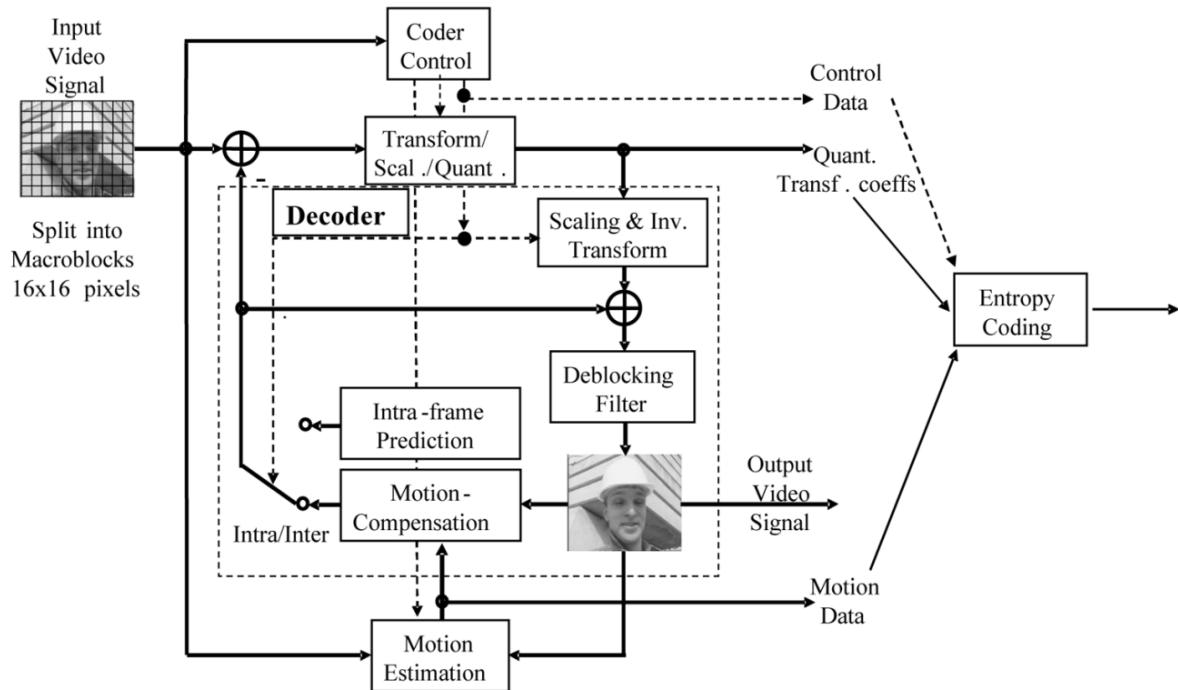


图 2.2 H.264/AVC 采用的混合视频编码框架

虽然这样的编码流程与之前的编码标准并无二致，但由于 H.264/AVC 对各个阶段进行了精心设计，并将灵活性和适应性贯穿于整个编码过程，使得其编码效率大大提高。下面简要介绍 H.264/AVC 的一些代表性技术。

视频压缩编码中首先要进行帧内预测或帧间预测，目的分别是去除空间和时间冗余。在 H.264/AVC 中，预测时可将一个宏块进一步分为小的子块进行，最小可以分成 4×4 的小块。标准还提供了多达 9 种的预测模式，从而能够更精细地利用局域特性。而对帧间预测，H.264/AVC 除支持单向和双向预测之外，每个方向还允许选用多个参考帧。在运动向量的准确度上，H.264/AVC 标准允许精确到 $1/4$ 像素，而且还允许跨越边界的值（采用外推插值技术）。

预测得到的残差接下来要经过变换阶段。考虑到精细的预测模式已经很好的去除了相关性，H.264/AVC 一改之前标准以宏块为单位的 DCT 变换，采用了更小的变换块 (4×4)。而且对于相对平滑的色度值 (YCbCr/YUV 中的 U、V)，还对每个宏块中的 4 个 4×4 子块变换后的直流系数又做了一次哈达玛变换。小尺寸块的变换减小了计算复杂度，而额外的哈达玛变换则进一步去除了信息冗余。

H.264/AVC 提供了两种熵编码方法 CAVLC (Context Adaptive Variable Length Coding) 和 CABAC (Context-based Adaptive Binary Arithmetic Coding) 来去除量化后的变换系数 (以及其他一些参数) 的统计冗余。它们都是上下文自适应的，前者为可变长编码，后者为二进制算术编码，后者计算较复杂但压缩率高，前者反之，可根据应用场景配置不同的算法。

以上只选取视频编码技术中的几个关键部分进行了简述。除了所提到的方面，H.264/AVC 还包含更多可由编码器选择的配置。这样不仅能够更有针对性地去除冗余、提高编码效率，而且也增加了该标准的普适性、灵活性和可定制性。正因为如此，能够在其基础上方便地进行扩展。

编码得到的比特流将被加上特定的头信息，在 NAL 层进行封装。NAL 的设计目标是提供网络友好性，使得 VCL 得到的数据在不同系统中的使用可以简单而高效地定制。我们后面将看到，在 H.264/AVC 基础上扩展得到的码流能与普通的 H.264/AVC 码流无缝兼容，很大程度上得益于这种网络抽象层机制。

网络抽象层的基本概念和语法结构是 NAL 单元 (NAL Unit, NALU)。这是一种由整数个字节组成的数据包。一个 NALU 以 1 字节的头信息起始，之后是不定长度的载荷。参见表2.1。

表 2.1 H.264/AVC 中的 NALU 结构

| NALU 头字节 | | | NALU 载荷 |
|----------|-----|------|---------|
| F | NRI | TYPE | |

NALU 字节头包含禁止位 (F, 1bit)、重要性指示位 (NRI, 2bit)、NALU 类型 (TYPE, 5bit) 三方面信息。其中禁止位规定为 0，重要性指示位在一定意义上表示该 NALU 的重要程度，而 NALU 类型则表明该 NALU 中携带的是什么样的载荷。

在 H.264/AVC 标准中，NALU 类型的 0 ~ 12 已被规定了用途，13 ~ 23 被保留用于以后可能的标准扩展，24 ~ 31 不做规定，可根据应用场景定制。按照 NALU 中载荷的不同，将其划分为两类：VCL 单元和非 VCL 单元。前者主要包含来自 VCL 的图像样本值编码数据，后者主要含有图像或序列参数集、补充增强信息 (Supplemental Enhancement Information, SEI) 等解码多个 VCL 单元所共用的一些重要数据。这种机制不仅能节省开销，也便于将重要数据通过更可靠的信道传输。对如何使用 SEI 来存储和传输额外的信息，后文在码流截取部分会有涉及。

2.1.2 可伸缩视频编码

可伸缩视频编码（Scalable Video Coding, SVC），简言之，就是在视频编码时生成一个包含不同层次信息的码流。从这个单一码流中，我们能够方便地丢弃一些数据，得到不同码率的多个码流，从而解码出在空间大小、帧率、画面质量等方面可伸缩的视频。虽然之前的標準也考虑了对可伸缩性的支持，但目前提到可伸缩视频编码，大多指的是 H.264/AVC 的可伸缩扩展。

SVC 在 H.264/AVC 的基础上引入了“分层编码”的概念。SVC 在对数据源一次编码得到的码流中，含有一个或多个的“层”，其中最低分辨率、最低帧率和最差图像质量的部分称为基本层，除此之外为增强层。增强层对应着更高的分辨率、帧率和图像质量。这样的码流能够提供在空间、时间、质量（或信噪比）等方面的伸缩性。当终端没有能力播放增强层或者网络无法承担高码率时，可伸缩码流的增强层数据可以从比特流中移除，通过牺牲一定的视频质量来保证解码和播放的顺利进行，因此提供了更好的适应性。

SVC 编码标准得益于 H.264/AVC 灵活的编码逻辑结构，也继承了 H.264/AVC 中许多良好设计的编码工具。SVC 不仅在编码效率和复杂性上与单层编码增加不多^[10]，并且完全兼容 H.264/AVC。以下结合传统编码框架，在 H.264/AVC 的基础上，对 SVC 中时间、空间和质量可伸缩的实现原理进行概述。

具备时间可伸缩的码流能够根据需要动态改变视频帧率。在 H.264/AVC 标准中，把用于解码一帧的多个 NALU 合称为一个 AU (Access Unit)。SVC 就是通过丢弃 AU 来降低帧率。时间可伸缩的实现不需要对 H.264/AVC 做任何增加，只需采用一种称为“层次化预测结构”的技术。

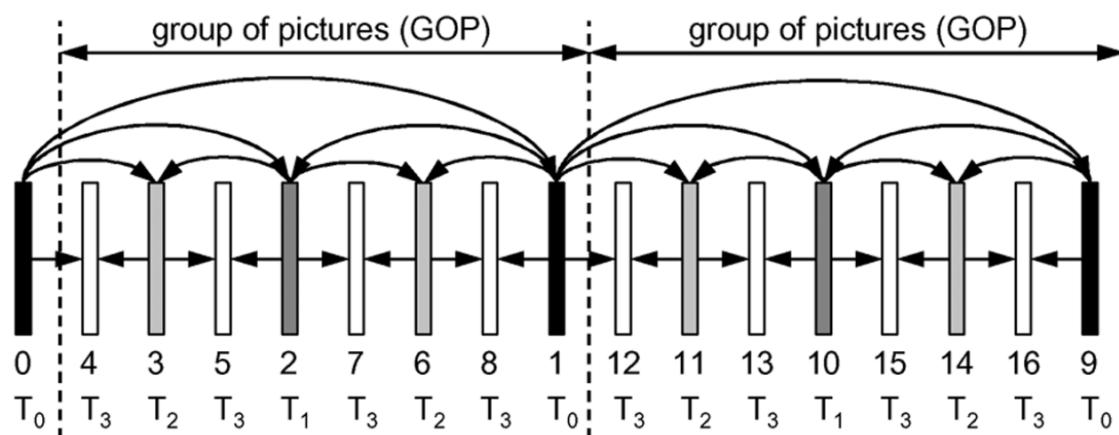


图 2.3 SVC 中通过层次化预测实现时间可伸缩

图 2.3^[7] 说明了通过层次化预测来实现时间可伸缩的原理。图中一个 AU 下面的数

字表示其编码顺序，数字下面的符号 T_i 表示其属于第 i 时间层。增强层 ($i > 0$) 的帧被编码为 B 帧（即参考其前后的帧进行双向预测），而且只参考较低层的帧。这样的结构下，低层的 AU 可以不依赖于高层，即任一较高层及其以上各层的 AU 可以被丢弃而不影响解码，以此实现帧率的伸缩性。需要说明的是，此例中这种“层次化 B 帧”的结构会带来编解码时延，且仅限于 1:2 伸缩比，这些不足均可通过仔细设计层次化预测结构而改善。

空间可伸缩是指视频分辨率大小的改变。空间可伸缩的实现比较复杂，其所支持的每个分辨率都对应一个编码层，称之为“dependency layer”，由 DID (Dependency ID) 标记。每个层都需要不同尺寸的输入图像，这一般由最高分辨率的图像下采样得到。每层的运动补偿和帧间预测都和单层 H.264/AVC 编码一样，然而由于表示的是同一图像内容，各层之间必然有很强的相关性。SVC 规定了各层之间的预测机制，称之为“层间预测”，以此来提高编码效率。空间可伸缩的实现原理参见图2.4^[35]。

质量可伸缩顾名思义就是视频各帧的图像质量可变。这是通过丢弃一个 AU 中的一些质量精细化 NALU 来实现的。具体操作时，给一个 AU 内的每个 NALU 打上质量层标记 QID (Quality ID)，可以将 QID 高于某个值的所有 NALU 丢弃，剩下的即可组成较低质量的码流。

以上分析的主要是在 VCL 层实现可伸缩编码的原理，可以看到多处涉及对特定 NALU 的取舍。这就需要对 H.264/AVC 的 NAL 层进行扩展从而能够在码流中标记和传送特定的可伸缩性信息与数据。

在前面介绍 H.264/AVC 时曾提到，NALU 头字节中的 NAL 类型 (nal_type) 在 13 ~ 23 的取值留作以后扩展使用；SVC 就增加了一些扩展的 NAL 类型。例如，nal_type=20 的 NALU 为新增的 VCL 单元，包含的是 SVC 中增强层的编码数据；nal_type=14 的 NALU 为新增的 SVC 前缀单元，它可能是 VCL 单元也可能是非 VCL 单元，取决于紧接着它到来的下一个 NALU。nal_type=15 的 NALU 用来传送 SVC 特有的图像参数集。对于这些新增类型的具体分析可以参见相关文档^[36]。

普通 H.264/AVC 解码器遇到 nal_type 大于 12 的单元会忽略之，但能成功解码基本层，因此 SVC 可与之兼容。而支持 SVC 的解码器将对类型为 14、20 的 NALU 进行利用，实现可伸缩性。与普通 H.264/AVC 的 NALU 不同，类型为 14、20 的 NALU 并非只有 1 个头字节，而是扩展为 4 字节的头。如图2.5^[36] 所示。可以看出，其中除了与普通 NALU 一致的第一字节，还包含了空间、时间和质量层的标志 (DID、TID、QID) 以及其他一些用于提供可伸缩性的信息。

综上，SVC 是在 H.264/AVC 标准的基础上扩展得到的。SVC 标准由国际电信联盟与国际标准化组织的专家组成的联合视频小组制定，该组织在标准形成过程中还开发

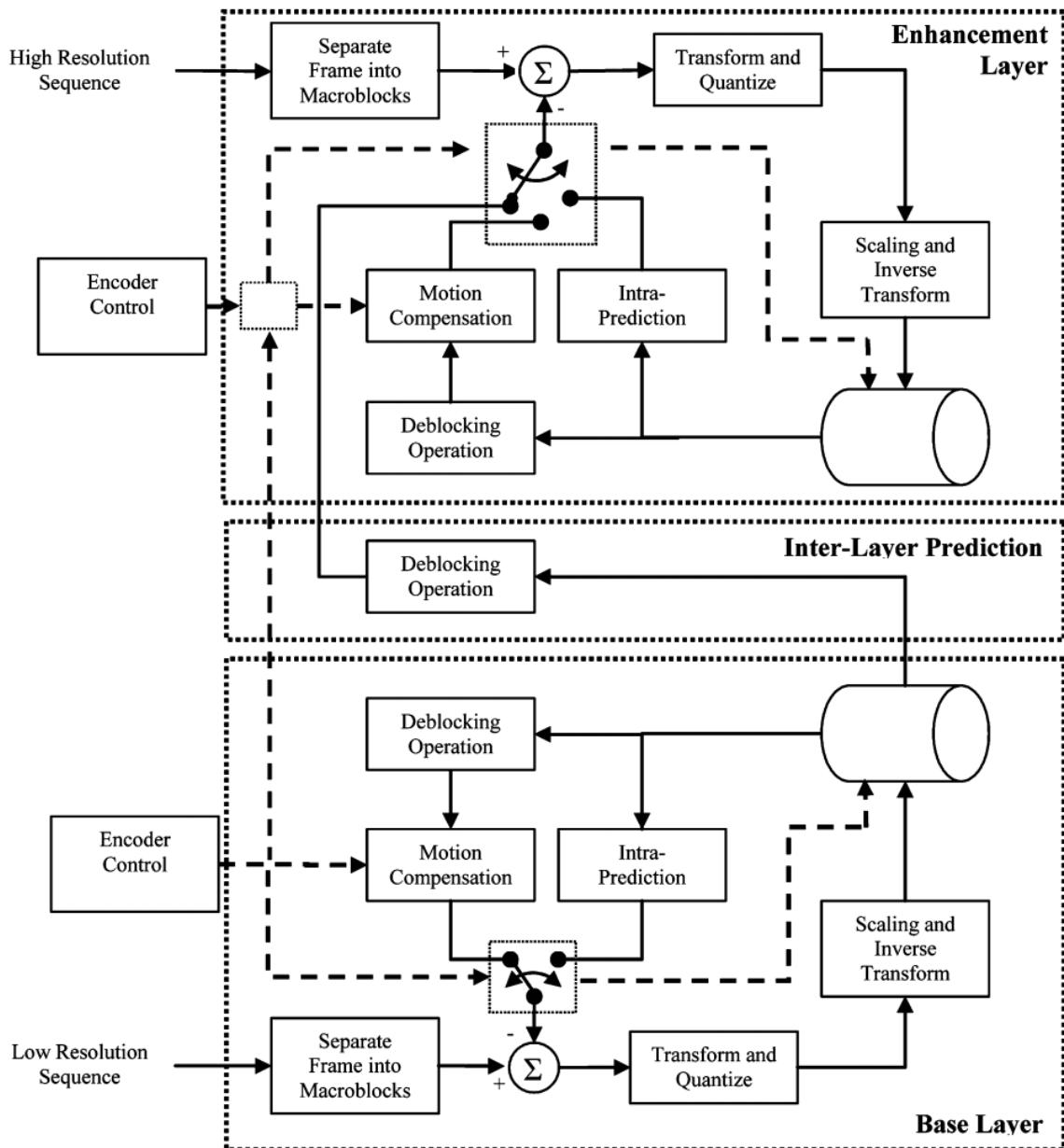


图 2.4 SVC 中空间可伸缩的实现原理

了一个官方的参考软件，称为 JSVM（Joint Scalable Video Model）^[37]。该软件完整实现了 SVC 的编解码过程。虽然该软件未做任何优化，很难在实际中应用，但由于其囊括了 SVC 标准中的几乎所有特性且清晰逻辑、配置简单，比较适合学习参考，以及作为学术研究的实验平台。因此，几乎所有的学术工作都是基于 JSVM 参考软件，在其上实现自己的算法并与内置的算法进行对比，体现其效率的提升。本文的研究中也采用了 JSVM 作为比较对象。

从最早的 H.261 到现在的 H.264/AVC、SVC，再到最新的 HEVC，一系列视频编码

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|-----|-----|---|---|---|---|-----|---|---|---|---|-----|---|-----|---|---|-----|---|---|---|---|----|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| F | NRI | NUT | | | R | I | PID | | | | N | DID | | QID | | | TID | | U | D | O | R2 | | | | | | | | | |

图 2.5 SVC 中的 NALU 头结构

标准的制定和推广过程不仅促进了产业发展，还在以下两个方面催生了大量的学术研究成果。一是对信源编码理论及应用的研究，包括对图像视频中不同信源的概率分布建模^[38-41]，对特定信源分布下率失真函数和量化器性能的分析^[42-44]，基于率失真分析和估计来改进编码中的率失真优化和码率控制过程^[45-48]，等等；这些研究旨在不断提高压缩效率和编码器性能，是视频产业链上后续相关环节的基础。二是视频编码与网络传输相结合的研究^[49]，包括在 Internet 上传输视频^[50,51]，采用可伸缩视频编码以应对网络异构和拥塞导致的带宽波动^[52,53]，等等。这方面的研究则直接推动了视频流媒体应用不断改善和蓬勃发展。本文的工作主要偏向于上述第二方面，因此接下来先对视频传输的相关知识进行介绍。

2.2 视频文件的流式传输

流媒体系统之所以能够在不必下载完整音视频文件的情况下进行播放，主要是因为采用了流式传输。流式传输的对象是存放在服务器上的流式文件。流式文件是流媒体系统特有的视频文件。流式文件经过了特殊处理，使其适合在网络上边下载边播放。对于编码压缩好的视频文件，需要将数据分成适当大小的分组并考虑差错恢复功能，此外一般还要加入特定的时间戳和索引（hint 或 index）信息，用于提供如快进、后退和随机访问等控制功能。上述过程称为媒体文件的流化。流化得到的流式文件存放在服务器上等待传输。

流式传输是流媒体系统的关键。通常流式传输可以分为两种：顺序流式传输和实时流式传输。顺序流式传输虽然也能边下边播，但用户只能观看已下载的部分，不能跳到未下载的前面部分。这种流式传输方式通过 HTTP 即可实现，不需其他特殊协议，但其功能和特性也不够灵活。与之相对应的是实时流式传输。这种方式允许用户对媒体流的发送进行更多的控制，可随机访问前后内容。相应地，这种传输方式也需要特定的服务器和特殊的协议。图??给出了典型的流媒体视频点播系统的基本框架。

在图2.6所示的系统中，通过流式传输播放在线视频的一般过程如下：

1. 用户根据观看意愿检索和选择视频；这个过程是在 Web 浏览器与 Web 服务器之间通过 HTTP/TCP 进行通信。
2. Web 浏览器利用从服务器得到的参数对客户端的流媒体播放进行初始化；这些参

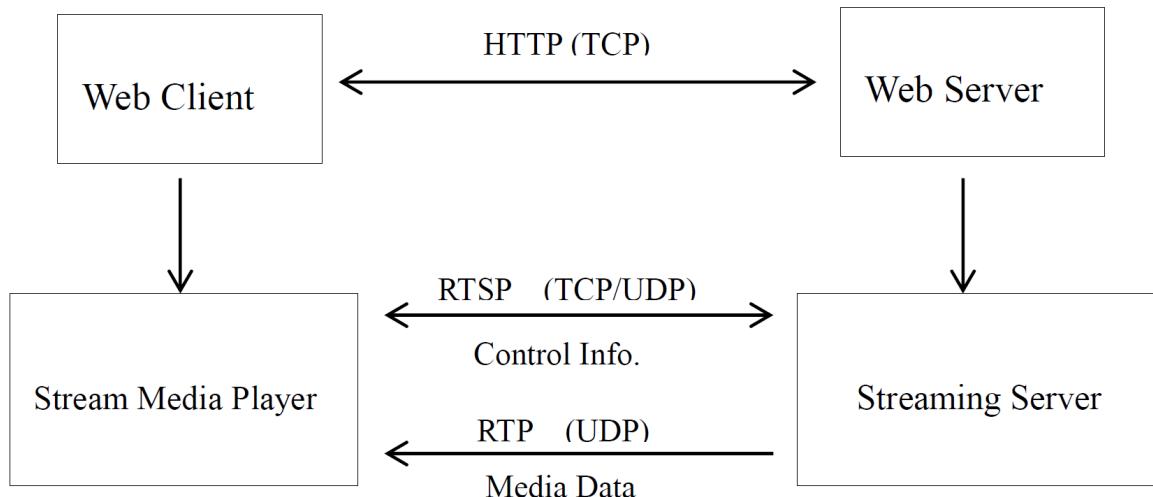


图 2.6 典型的流媒体应用框架

数包括流媒体服务器地址、目录和媒体文件信息等。

3. 播放器与流服务器之间用实时流协议（Real-Time Streaming Protocol, RTSP）^[54] 进行音视频数据传输的控制，如准备、开始、暂停、后退等。
4. 流服务器通过实时传输协议（Real-time Transport Protocol, RTP）^[55] 将媒体数据传送给客户端，数据抵达时即可解码播放。

需要指出的是，上面描述的系统只是一个代表性示例，随着技术的发展，其中很多特定的技术会发生变化。例如，用户检索和选择视频所用的可能不是 Web 浏览器，而是手机应用；传输数据可能用的是其他协议等，而且不限于客户端和服务器两点通信。但是，基本框架仍然没有太大的变化。

流媒体是面向网络的应用，既需要终端软件，也需要传输和控制协议。本节剩余部分中，首先介绍流媒体传输中常用到的几个典型协议，然后介绍代表性的流媒体服务器。

2.2.1 典型的传输协议

(一) RTSP

RTSP 是一种应用层协议，其目的是为了在 IP 网络中有效传输流媒体数据。RTSP 本身并不发送媒体数据内容，它主要起到建立流和控制流的作用。RTSP 类似于 HTTP，在客户端和服务器之间发送请求和回应。一般会有以下交互：

- **OPTIONS:** 客户端询问服务器有哪些操作，服务器进行回应，列出可用的方法。
- **DESCRIBE:** 客户端向服务器请求描述信息，服务器以会话描述协议（Session Description Protocol, SDP）的形式返回该信息，包括音视频的编码类型、参数，控制

通道等等。

- **SETUP**: 客户端在得到媒体信息后建立流；对音频、视频需要分别建立。
- **PLAY** (类似的有 PAUSE、SCALE 等): 开始播放媒体 (或进行控制)。服务器在收到 PLAY 后会开始以 RTP 协议发送媒体数据包。
- **TEARDOWN**: 客户端请求关闭流。

可见，RTSP 只在播放流媒体之前和结束时发挥作用。实际传输数据采用的是实时传输协议 RTP。

(二) RTP

RTP 负责传输媒体数据。它是单向的，将流媒体内容以包的形式从服务器发送至客户端。RTP 包由头信息和载荷组成^[55]。包头中提供了版本号、有效载荷类型、时间戳、序列号等信息，并且允许扩展。包的载荷则是实际的音视频编码数据。接收端解码时必须知道 RTP 载荷中数据的编码方式，包头中的有效载荷类型即标识了这一信息。有效载荷类型域共 7bit，可表示多达 128 种载荷。除了标准中指定的类型外，还可以进行扩展。

可以用指定有效载荷类型的 RTP 包来表示所传送数据的编码方式为 H.264。此时，RTP 载荷 (payload) 将呈现为 H.264 定制的结构^[56]。H.264 的一个 NALU 大小不定，有时会超过一个 RTP 包的容量 (受限于 RTP 下层协议的包大小，如 UDP 数据报长度)；有时又较小，可将多个 NALU 封装在一个 RTP 包中。为此，针对 H.264 的 RTP 载荷将第一个字节作为头字节来标记上述的分拆和组合。事实上，H.264 的 RTP 载荷的第一字节采用的结构与 NALU 的头字节相同，并且各个比特的意义也一致。其中后 5bit 表示该 RTP 包中的 NALU 的类型，若为 0 ~ 23 则表示 H.264 标准的一个完整 NALU，大于 24 的被用来表示拆分或组合 NALU。

RTP 为包括 H.264 在内的不同标准流媒体数据提供了有效的传输支持。但它却是不可靠的，它本身没有任何错误检测或拥塞控制机制，因此无法提供 QoS (服务质量) 保证。RTP 底层可以使用 UDP 也可以使用 TCP，但大多采用的是 UDP。而且 RTP 一般与下面介绍的实时传输控制协议 (Real-time Transport Control Protocol, RTCP) 配合使用。

(三) RTCP

实时传输控制协议 RTCP 被设计用来监测流媒体传输质量并在 RTP 会话参与者之间传递信息。RTCP 包可携带不同类型的信息，其中最典型的是接收者报告。这种 RTCP 包由接收者向服务器发送，报告其收到的 RTP 包数目、丢包数、包的抖动、延时情况等。这相当于提供一种反馈信息，发送端应用程序可以利用这些信息做出调整，如改变发送速率等。一般流媒体软件都会利用 RTCP 来控制和保证传输质量。

上面介绍的 RTSP、RTP、RTCP 是具有代表性的一个协议系列，它们在传统视频点播系统中得到了广泛的应用。现在随着技术的发展和直播应用的兴起，更多的视频流媒体协议被提出并逐渐占据了主流。典型的例子有实时消息传输协议（Real Time Messaging Protocol, RTMP）^[57] 和 HTTP 直播流协议（HTTP Live Streaming, HLS）^[58]。前者的特点是延迟较低，所以多被用在有实时性要求更强的场景，例如有交互的直播；后者的优势在于直接基于 HTTP 进行传输，便于部署，在允许一定延时的场合是不错的选择。

在本文的研究平台中，点播系统采用的是 RTSP+RTP 协议族，而直播系统采用的是 RTMP。

2.2.2 流媒体服务器

实用的流媒体系统离不开服务器端软件和客户端播放软件。客户端软件主要是支持流媒体的播放，为此除了普通媒体播放功能，还需要实现传输所用的特定网络协议，但总体来说，流媒体客户端要比流媒体服务器简单得多。下面着重对流媒体服务器端程序进行介绍。

在流媒体服务器上运行的程序负责响应连接请求，从存储系统中取出音视频数据并以流式传输的方式发送给客户端。如图2.7所示，完整的流媒体服务器平台需提供会话服务、内容服务、流服务、媒体数据存储管理等。市场上最知名的流媒体解决方案主要有三种：RealNetworks 公司的 RealSystem，微软公司的 Windows Media，以及苹果公司的 QuickTime。它们分别有各自的流媒体服务器和播放器。前两家的产品都只有闭源的商业版，而苹果公司的流媒体服务器除了商业版本外，还同时提供开源的版本，称为达尔文流媒体服务器（Darwin Streaming Server, DSS）^①。该服务器经常被用来做相关的研究测试^[59]，本文在视频点播方面的码率自适应研究就是基于它进行的。

原本的达尔文流媒体服务器是不支持 SVC 的，为了本文的研究和实际的应用，我们在其源代码上修改使其增加了对 SVC 的支持。达尔文流媒体服务器包含多个子工程，其中负责视频数据传输的主要集中在名为“StreamingServer”的工程中。在这里，由 RTSP 协议与客户端建立连接会话，并在客户端的 SETUP 命令下建立多个音频和视频的流，然后由 RTP 协议向客户端发送数据。每个会话对应一个 RTPSession 类，每个流对应一个 RTPStream 类， RTPStream 类的 Write 函数起到实际的发送数据功能。实现对 SVC 的支持主要是在此类中改变向流中写数据的逻辑。当然，前提是数据的来源必须是 SVC 的。

达尔文服务器处理的媒体数据来源于特定的 MP4 文件。正如前文提到的，与本地

^①<http://dss.macosforge.org/>

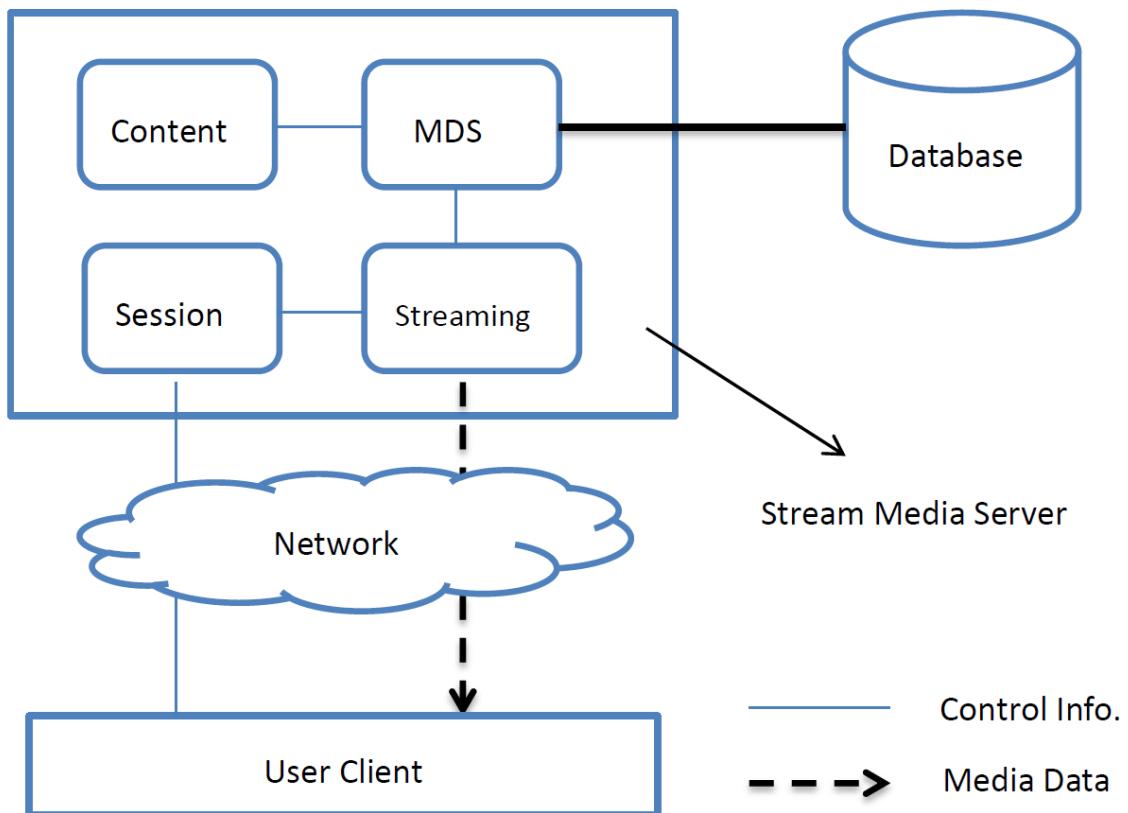


图 2.7 流媒体服务器的组件及其与外界的交互

播放的普通 MP4 文件不同，用于流式传输的 MP4 文件必须经过特殊处理（即流化）。流化后的 MP4 文件不仅包括对应音、视频的 audio track 和 video track，还包括一个 hint track，其中信息用于流式传输。SVC 的 MP4 文件区别于非 SVC 的地方就是在 hint track 中有 SVC 伸缩层的数目信息。我们采用特定的工具来处理 MP4 文件，将 SVC 视频流所支持的空间、时间、质量层数目写入 hint track。这样流化得到的 MP4 文件就是支持 SVC 的，可送给达尔文服务器使用。

达尔文服务器在接收到客户端按照 RTSP 协议发送的 DESCRIBE 命令后，会解析所提供的 MP4 文件中的相应媒体信息，以 SDP 的形式发给客户端。SDP 中包含了音视频的参数，其中可以任意添加参数项。我们在描述视频的参数中加入 svclayercount 这一项，把从 hint track 读到的空间、时间、质量层数目告诉客户端。这就使得客户端可以主动设置想要接收的层。这只是提供一种支持，客户端也可以完全不主动设置，服务器会根据网络状况自动调整发送的增强层数据。具体如何调整，就是本文所要研究的码率自适应问题。

上面介绍了视频编码和传输的基础知识。接下来，针对本文要研究的两方面问题进行具体分析，并给出国内外已有的相关工作。

2.3 可伸缩视频码流截取

2.3.1 问题分析

可伸缩视频码流截取使视频码率可以动态调整，是基于可伸缩视频编码进行自适应流媒体传输的前提条件。所谓码流截取，是指从一个完整的码流中抽取所有数据包的一个子集，得到一个更低码率的子流的过程。我们希望在给定的码率限制下，能够截取出一个质量尽可能好的子流。

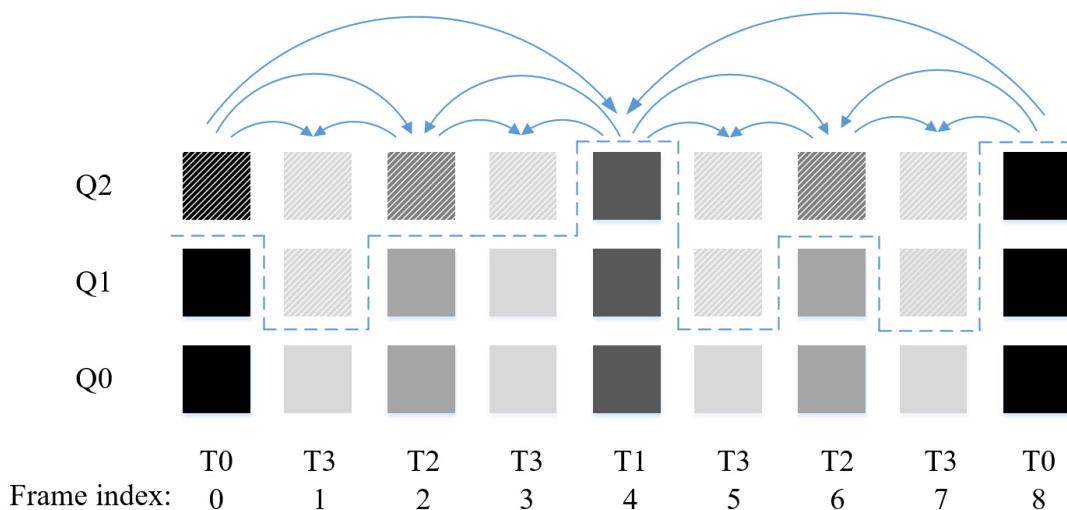


图 2.8 可伸缩视频码流截取示意图

图2.8展示了一个可伸缩视频码流的结构，以及从中截取子流的一种可能的方式。可以看到，该码流中的数据包（也就是前面介绍过的标准规定的组织数据的单元 NALU）被分为了不同的层。时间层（T0 ~ T3）可以提供时间可伸缩（也就是视频帧率的变化），主要反映了各帧之间的参考或依赖关系。图形上方的箭头从被参考帧指向参考它（也就是依赖于它）的帧。质量层（Q0 ~ Q2）反映了一帧之内的视频质量伸缩性。处于高层（也就是 Q1 和 Q2 层）的数据包可以被部分丢弃从而实现码流截取。图中的虚线表示了一个截取的例子。虚线上方的数据包全部被丢弃，只有虚线下方的数据包保留在截取出来的子流中。

容易发现，可伸缩视频码流截取从本质上来看其实是一个组合优化问题。在所有的数据包中，我们希望在给定的码率限制下，选出一个最优的数据包组合，使得它们构成的子流具有最大的视频质量。如果用 R_T 表示码率限制， θ 表示某种截取方案，那么码流截取问题可以用下面的式子表示：

$$\theta^* = \arg \min_{\theta \in \Theta} D(\theta), \quad \text{s.t.} \quad R(\theta) \leq R_T. \quad (2.1)$$

其中 $D(\theta)$ 和 $R(\theta)$ 分别表示这种截取方案下对应的失真和码率。

更准确地说，这个问题与大家所熟知的“0-1 背包问题”^①在形式上是一致的。每个数据包可以看作是一个具有特定重量和价值的物品。这里的重量就是数据包的大小，价值就是它对重建视频质量的贡献。码流截取的过程就是决定每个数据包是否包含在最终的子流里。用“0-1 背包问题”中的术语来说，就是选择将哪些物品放入背包中。

对于典型的“0-1 背包问题”而言，其最优解可以用动态规划^②的方法得到。然而对于码流截取问题来说，这个方法是不可行的。主要原因在于数据包之间的依赖关系，下面具体分析。

首先，截取的数据包子集不能任意挑选，因为如果一个包被包含进了子流中，那么这个包所依赖的数据包也必须被包含进去，否则将无法正确解码。例如，在图2.8中，我们不能只选择一帧中的 Q2 层数据包而丢弃同一帧的 Q1 层。

其次，不同于“0-1 背包问题”中定义良好的物品价值，码流截取问题中一个数据包对最终视频质量的贡献并不是明显且确定的。这是因为，数据包所在的帧在解码中是互相参考的，它们对视频质量的影响会互相干扰，对于不同的截取方案，每个数据包的实际贡献可能有所不同。事实上，除非实际进行一遍截取、解码、计算的操作，我们甚至无法准确地得到所截取出的子流的真实视频质量。这就使得可伸缩视频码流截取问题成为了一个复杂得多的问题。

2.3.2 相关工作简介

就我们所知，学术界没有人提出过一种确定能找到码流截取最优解的方法。当然可以用暴力穷举的方法来解决这个问题，但其复杂度显然是无法接受的。相关的工作都是尽可能用较低的计算量获得较好的次优解。下面对可伸缩视频码流截取的相关工作做简要的介绍。

最初的参考软件 JSVM 提供了一个非常简单的基本截取器。该截取器并没有考虑任何优化，只是实现了一个从完整码流中丢弃一些层的包来使码率低于给定值的程序。如图2.9^[60] 所示，该程序按照 SVC 中的层的概念将所有 NAL 数据包分成了几组，在截取丢包的时候，高层的包先丢，如果整个高层都丢弃之后仍然不能满足码率要求，再考虑丢低层的数据包。这种方法符合直观理解，因为一般来说低层的数据作为被参考对象，影响的范围更广，重要性更大一些，所以应该优先保留；反之，高层的数据可以先丢。但编码过程中形成的数据包之间的关系并没有这么简单，有可能某个高层的包比某个低层的包更重要，基本截取器并没有考虑这一点。因此，基本截取器的效果离

^①http://en.wikipedia.org/wiki/Knapsack_problem

^②http://en.wikipedia.org/wiki/Dynamic_programming



图 2.9 JSVM 基本截取器的算法图示

最优还有很大的差距。此外，对于同一个层的数据包，基本截取是按某个固定的顺序依次丢弃，也并没有对这些同一层数据包的重要性做区分。这也是基本截取器效率低下的另一个原因。接下来介绍对基本截取器的改进工作。

Amonou 等人^[60]提出了一个基于“Quality Layers”的码流截取方案。这一方案被 SVC 参考软件 JSVM 所采用，而且取得了比 JSVM 中的基本截取器更好的性能。在 Amonou 等人的方案中计算了一个“Quality Layers”(简称 QL)值，它反映了一个数据包(NALU)的码率失真影响，据此进行截取能得到率失真优化过的结果。对于计算出来的 QL 值，可以用两种方法将其存储在码流中：

- 占用 NALU 头信息中的 priority_id 字段来存储这个 NALU 的 QL 值。这个字段有 6 比特，只能存储 64 个不同值；
- 用专门的补充增强信息 (SEI) 数据包来存储所有 NALU 的 QL 值。

在计算 QL 值的时候，对数据包失真影响的估计是一个计算量非常大的过程，而且估计的准确性也有提高的空间。因此，后续的研究工作大多旨在探索新的失真模型，以求快速、准确地估计丢弃数据包所带来的失真。新提出的失真模型中一方面是降低复杂度，另一方面是进一步提高准确度。例如，Sun 等人^[61]和 Maani 等人^[62]所构造的模型都是基于帧间的误差漂移来计算失真。Sun 等人的模型在性能方面几乎与 JSVM 相当，但计算复杂度却大大减少。而 Maani 等人的模型取得了比 JSVM 更高的估计准确度（由图2.10^[62]可见，其估计的失真与实际失真基本相同），但是由于其是基于训练

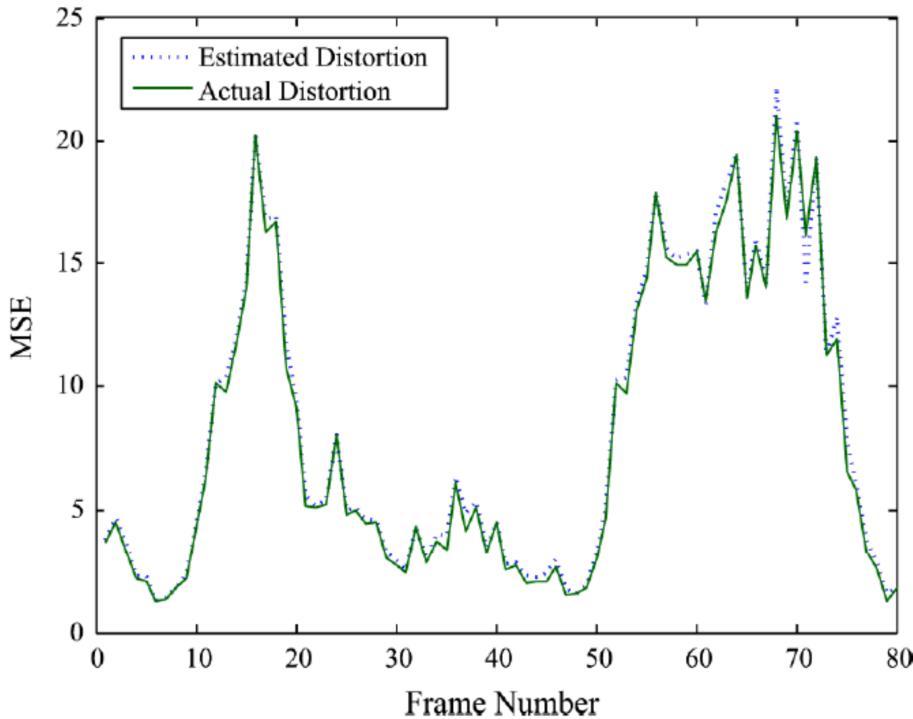


图 2.10 基于训练的模型所取得的失真估计准确度图示

的，为了得到更鲁棒的模型参数，需要更大的计算量。

此外，Ramanathan 等人^[63]提出了一个“Quality Metadata”的概念来用于码流截取，这与上面所说的“Quality Layers”是同样的思想，只是计算方法有所差别；Yang 等人^[64]提出了一个基于模拟退火的方法来解决码流截取问题，他们简单地用丢弃较低增强层数据带来的失真来估计丢弃较高增强层数据带来的失真。因为码流截取从另一个角度看是丢弃数据包的过程，丢弃的先后顺序可以认为是这些数据包的优先级。于是一些研究者从赋优先级的角度来提出相应的算法。例如，Lim 等人^[65]把可伸缩视频码流视为树形结构并据此来为每个数据包赋予优先级，而 Zhu 等人^[66]则采用拉格朗日乘子方法计算优先级值。在这些已有的工作中，失真估计的准确性都显著地影响最终结果。因此，提出简单有效的误差或失真模型对于码流截取的研究至关重要。本文的研究工作正是在此处着力并取得了创新性成果。

2.4 传输中的码率自适应

2.4.1 问题分析

在数据源具备码率可变的前提条件之后，实际传输时就可以根据带宽来调整发送的速率。改变码率去适应带宽的变化，这就是所谓的码率自适应问题。对于一个码率自适应算法，有三个最基本的目标。这三个目标可以用来衡量自适应算法的效果，也

是设计算法时需要考虑的方面。下面分别进行描述。

对于一个流媒体系统，最首要的目标应该是保证视频播放能够连续不断。一旦视频开始播放，每个数据包都有了自己的显示时间。这个时间也决定了传输的截止时间。如果这个数据包没有在截止时间前达到客户端，那么客户端播放器就会因缺少数据而暂停。这就是用户所遇到的卡顿现象，应该努力避免。避免卡顿一般要依赖于传输中的缓冲机制。如果检测到带宽下降、当前码率确实过高，应及时下调质量，以防止数据不能按时传送而导致播放卡顿。

其次是要保持视频质量的平滑性。网络条件变化可能比较剧烈，但视频质量应该尽可能维持在一个比较稳定的水平。因为频繁的质量调整会给用户带来反感。举例来说，当用户适应了较低的质量后，突然调高质量并在短时间内又调低，给用户的体验还不如不调整。因此，在保证播放连续的前提下，应尽可能减少质量调整次数，充分利用缓冲区的作用，避免因为临时或短暂的带宽变化就调整质量。

最后，还要提供给用户尽可能好的视频质量。假设我们一直发送较低码率的视频流，那么既能保证播放的连续性又不存在平滑性的问题，上述两个目标可以轻而易举达到。但是这样的话用户看到的视频质量将一直处于较低的水平，显然不是令人满意的选择，所以不可取。码率自适应算法应该能够充分利用可用带宽，在带宽足够的情况下提高发送质量，给用户最好的观看体验。

综上所述，在视频流媒体的码率自适应研究中，保证播放流畅是最基本的要求，同时视频的高质量和平滑性也是所追求的指标。

2.4.2 相关工作简介

保证播放流畅性的研究主要解决的是如何选择和调度视频数据包的问题。例如，Gao 等人^[67]提出的调度策略首先确保重要性较高的数据优先发送，而对于重要性相同的数据包，播放时间最早的首先发送。Schier 等人的工作^[68]也与此类似，把视频数据放在具有不同优先级的缓冲区中，通过调整优先级来确保数据及时发送。如图2.11^[68]所示，高优先级的缓冲区中的数据最多，它们对应较低的视频质量，优先保证视频连续播放；而较低优先级的缓冲区中数据较少，这些对应的是高码率和高质量的视频，它们只有在带宽足够高的情况下才会被发送。系统根据带宽情况决定填充哪个优先级，相当于是在改变发送视频的码率。

在达尔文流媒体服务器中内置有一个被称为包延迟反馈的码率自适应算法。当一个包要发送时，服务器会检测它的延时（即这个数据包应播放的时间和当前时间的差），并根据这个延时来判断当前发送顺畅还是阻塞，以此来推测带宽情况，并相应调整发送码率和质量。具体判断方法是将这个延时作为反馈，与预先设定的阈值进行比较。例

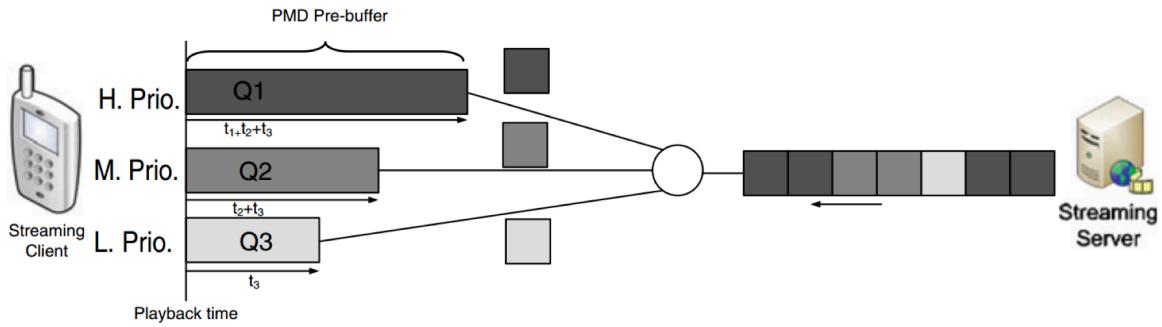


图 2.11 基于不同优先级缓冲区的码率自适应算法

如，预先决定当前带宽下发送某个数据包之后 T 时间内应该播放，那么理想延时应该保持为 T 。如果发现延时小于 T 了，说明发送受阻，可能是带宽减小了；反之，如果 T 变大了，说明发送顺畅，可以考虑增加质量。这种算法能够在一定程度上检测带宽的变化并作相应调整。但是它的调整策略是保守的，在它检测到延时变小时，很可能客户端已经因为缺少数据而停止了。而且该算法对带宽的变化趋势也不能提前预测，很可能导致质量波动，这也是需要改善的一个问题。

近些年来，由于 HTTP 动态自适应流媒体（DASH）^[69] 的兴起，大部分码率自适应的研究工作都围绕它进行。例如，Akhshabi 等人^[70] 分析了来自商业公司和开源社区的多个 DASH 客户端播放器的自适应行为，比较了他们之间性能的优缺点。张辉帅等人^[71] 提出了一种基于拉模式的码率自适应算法，利用滑动窗口分析最近若干分片的下载时间，基于此来选择最合适的码流。Huang 等人^[72] 结合了缓冲区的状态和对带宽容量的估计这两个判据来做如何调整码率的决策。Juluri^[73] 等人考虑到每个分片数据量大小不同可能导致下载时间的差异，因此提出了一种提前查看分片信息的码率自适应算法。

这些已有的研究工作在一定程度上提高了传输效果，但大都是针对点播模式，没有考虑到直播模式下的特殊问题。目前实时性直播的应用越来越流行，从某种程度上来说是一个更重要的模式。直播由于其数据是实时产生的，而且要先上传到服务器，因此与点播有所不同，需要进行针对性的研究。本文的工作不仅适用于点播系统，也对直播系统有所涉及。

2.5 本章小结

本章首先对视频的压缩编码和流式传输做了介绍，其中包含了本文研究工作的知识基础；然后结合本文的研究内容对码流截取和码率自适应这两个方面的问题和已有

工作进行了分析。后续章节将对这些问题依次进行研究，提出相应的新方案和算法，并展示实验结果。

第三章 采用线性误差模型的可伸缩视频码流截取方案

能够调整数据源的码率是流媒体系统码率自适应的前提条件。对于 SVC 视频而言，这意味着从整个码流中截取出一部分数据包来得到不同的低码率子流。如何在给定的码率限制下使得截取的失真最小，是码流截取问题的关键。在本章中，我们提出一个线性误差模型和一个基于贪心思想的优先级赋值算法来解决这个问题。

3.1 线性误差模型

3.1.1 模型推导

在 H.264/AVC 中，解码过程具有线性性质^[74]。在不考虑舍入、截断和去块滤波的情况下，解码的主要步骤，如预测、变换、量化等，都可以近似认为是线性操作。在这种近似下，一个图像组（Group Of Pictures，GOP）的重建像素可以看作如下这三部分的线性组合：该图像组的已重建像素、残差、静态预测子。假设这个图像组中有 K 帧，每一帧的宽度和高度分别为 W 和 H ，则这个图像组的解码重建过程可以用矩阵的形式描述如下：

$$s = \mathbf{M} \cdot s + \mathbf{T} \cdot c + p . \quad (3.1)$$

令 $N = K \times W \times H$ ，则在公式 (3.1) 中， s 是 $N \times 1$ 维列向量，指代重建像素； c 是 $N \times 1$ 维列向量，指代变换系数； p 是 $N \times 1$ 维列向量，指代静态预测值； \mathbf{M} 和 \mathbf{T} 都是 $N \times N$ 维的方阵，从而使得 $\mathbf{M} \cdot s$ 得到的结果是运动校正预测信号， $\mathbf{T} \cdot c$ 得到的结果是残差。 \mathbf{M} 的实际值由所选择 N 的宏块类型、参考帧索引和运动向量决定，而 \mathbf{T} 的实际值取决于所选的量化参数（Quantization Parameter，QP）。

将上述解码过程的线性模型运用到 SVC 的质量可伸缩中，可以得到一个线性误差模型（Linear Error Model，LEM）。根据公式 (3.1)，对于一个只包含基本层的码流，有：

$$s_B = \mathbf{M}_B \cdot s_B + \mathbf{T}_B \cdot c_B + p , \quad (3.2)$$

其中带下标 B 的表示是基本层变量。在质量可伸缩中，增强层主要是变换系数的精细化。因此，若一个增强层的数据包集合记为 I_1 ，则含有该增强层的码流解码时重建关

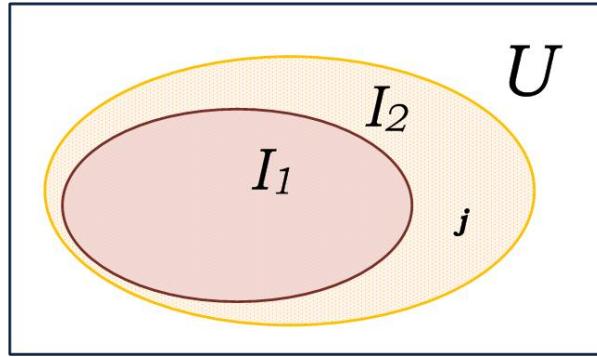


图 3.1 增强层数据包集合关系示例

系可表示为：

$$s_{I_1} = \mathbf{M} \cdot s_{I_1} + \mathbf{T}_B \cdot c_B + \left(\sum_{i \in I_1} \mathbf{T}_i \cdot c_i \right) + p, \quad (3.3)$$

其中 i 表示集合 I_1 中的一个增强数据包； c_i 是一个向量，含有增强层数据包 i 中的变换系数； \mathbf{T}_i 是由 i 的 QP 值所决定的变换矩阵，而 $\mathbf{T}_i \cdot c_i$ 得到的就是从 i 解码出来的残差值。

同样的，对于一个含有增强层数据包集合 I_2 的码流，有：

$$s_{I_2} = \mathbf{M} \cdot s_{I_2} + \mathbf{T}_B \cdot c_B + \left(\sum_{i \in I_2} \mathbf{T}_i \cdot c_i \right) + p. \quad (3.4)$$

如果 I_1 是 I_2 的子集，那么二者的重建误差就可以由公式 (3.4) 与 (3.3) 相减得到：

$$(s_{I_2} - s_{I_1}) = \mathbf{M} \cdot (s_{I_2} - s_{I_1}) + \left(\sum_{i \in I_2} \mathbf{T}_i \cdot c_i - \sum_{i \in I_1} \mathbf{T}_i \cdot c_i \right), \quad (3.5)$$

$$\Rightarrow e = s_{I_2} - s_{I_1} = (\mathbf{I} - \mathbf{M})^{-1} \cdot \left(\sum_{i \in I_2} \mathbf{T}_i \cdot c_i - \sum_{i \in I_1} \mathbf{T}_i \cdot c_i \right). \quad (3.6)$$

特别地，如果 I_2 和 I_1 的差别只有一个增强层数据包 j ，也就是说 $I_2 \setminus I_1 = \{j\}$ （如图3.1所示），那么根据公式 (3.6)，这两个重建序列的差为：

$$e_j = (\mathbf{I} - \mathbf{M})^{-1} \cdot \mathbf{T}_j \cdot c_j. \quad (3.7)$$

可以看到， e_j 只由数据包 j 决定，与其他的增强层数据包无关。这个值 e_j 被定义为数据包 j 的“误差向量”，表示丢弃数据包 j 后所带来的像素值误差。我们可以通过如下的方式得到任何一个数据包 j 的误差向量：选取两个只差 j 的数据包子集，将它们解码得到的视频序列相减即可。

一旦获取了所有增强层数据包对应的误差向量，那么丢弃任意一个增强层数据包子集所导致的失真就可以用这个子集中的数据包的误差向量之和来估计。举例来说，如果增强层数据包 x, y, z 从一个码流中被丢弃，那么丢弃前后的重建序列之差应该为：

$$e = e_x + e_y + e_z. \quad (3.8)$$

这个简单的相加之所以合理有两方面的原因。第一，不像 MSE 或 PSNR，误差向量仅仅是像素的差值，并不涉及二次方运算。第二，每个增强层数据包在重建中会各自对像素进行精细化，其效果是叠加的（参见前面的推导过程）。

应该指出，上述第二点原因的成立是有条件的。在公式 (3.2) 到 (3.4) 中， \mathbf{M} 和 p 的值都相同是因为，在质量可伸缩的情况下，增强层数据包仅仅是通过操作变换系数得到的，并不涉及运动校正和静态预测过程。然而这对于其他维度的可伸缩（例如视频分辨率改变的空间可伸缩）并不正确。因此，此处推导的线性误差模型并不适用于任意配置的可伸缩视频编码。在本文的研究中，只用到了质量可伸缩；其他伸缩维度将作为未来工作中的探索对象。

3.1.2 误差向量获取

如果按照上一小节中所说，每次计算一个数据包的误差向量，那么所需的解码次数将是所有数据包的个数。这么做的计算量太大。在本小节中，我们给出一个高效获取所有误差向量的方法。

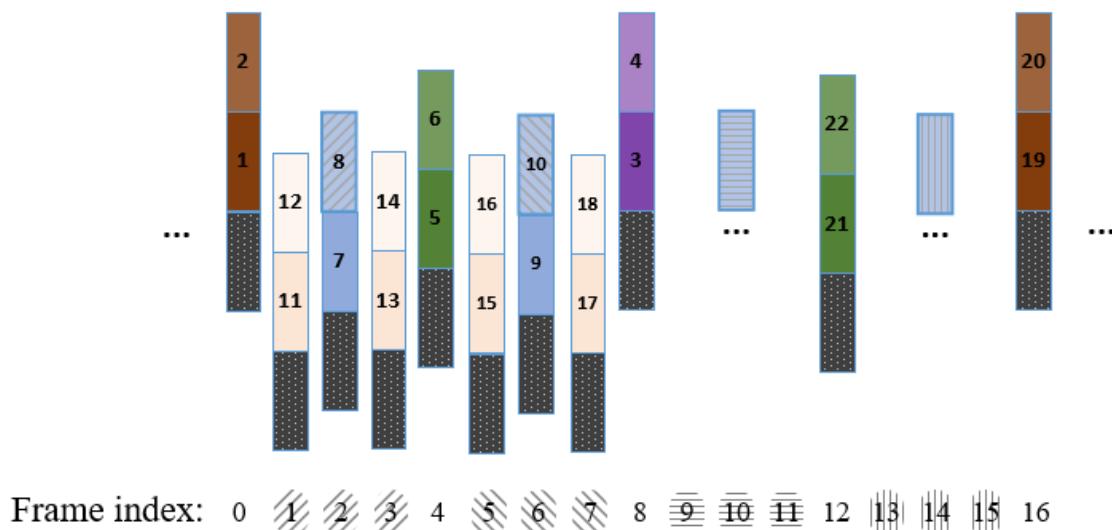


图 3.2 SVC 码流中的图像组 (GOP) 结构示例

为了减少解码次数，我们将所有的增强层数据包分成若干组，使得每一组中的多个误差向量能在一次解码中同时获取。考虑如图3.2所示具有8帧图像组结构的一个视频码流。在该图中，基本层数据包是黑色的，增强层数据包按照颜色分为了几个组（相同颜色的属于同一组）。在每一组中，增强层数据包的误差向量可以同时计算，因为这些数据包所影响的像素是互不干扰的。例如，丢弃8号数据包只会在帧1、2、3中带来像素误差，而丢弃10号数据包只会在帧5、6、7中带来像素误差（帧的编号从0开始）。因此，我们可以只通过一次相减来同时得到8号和10号数据包的误差向量 e_8 和 e_{10} ，以及所有其他GOP中的相同位置数据包（用相同颜色表示它们属于同一组）的误差向量。应该指出，这些误差向量是非常稀疏的，因为丢弃单个数据包带来的失真影响非常有限。

为了获取所有的误差向量，所需的解码次数大约等于增强层的层数($L_Q - 1$)乘以时间层的层数 L_T 。对于图3.2中的例子而言， $L_Q = 3$, $L_T = 4$ ，所以所需的解码次数照此计算将会是 $(3 - 1) \times 4 = 8$ 。然而，我们还需要额外的 $L_Q - 1 = 2$ 次解码，来分离“关键帧”^[34]（图3.2中的帧0、8、16）中数据包的失真影响。关键帧中增强层数据包的丢弃将会给它两边相邻的两个GOP都带来失真。例如，丢弃4号数据包带来的失真影响范围将会从帧1直到帧15，从而与帧0和帧16中的数据包的失真影响互相交叠。因此，关键帧同一位置的增强层数据包（2、4、20号数据包）应该被分为两个组：一组包含偶数GOP的关键帧数据包（2和20号包），另一组包含奇数GOP的（4号包）。于是，误差向量获取真正所需要的解码次数应为：

$$N = (L_Q - 1) \cdot (L_T + 1). \quad (3.9)$$

这与JSVM中Quality Layer信息的获取所需的解码次数大致相当。

3.1.3 失真估计与模型验证

基于线性误差模型，我们可以估计丢弃任意增强层数据包组合带来的失真，只需把所丢弃的每一个数据包对应的误差向量加起来即可。

例如，对于一个通过丢弃数据包集合 I_x 而得到的子流来说，它解码出来的视频序列与原始序列间的差值可以估计为：

$$e(I_x) = e_{full} + \sum_{i \in I_x} e_i. \quad (3.10)$$

这个公式中 e_{full} 表示全部增强层数据包都参与解码得到的视频序列与原始序列之间的

差值。这是编解码过程固有的失真，在一个数据包都不丢弃的时候也是存在的。

在采用基于线性误差模型的失真估计之前，我们先做实验验证一下其准确性。该实验中的码流都是用 JSVM 编码器压缩得到的，编码采用了如图3.2所示的 GOP 结构，熵编码过程配置的是 CABAC。每个码流基本层和增强层的 QP 分别为 33 和 27。增强层通过配置 Medium Grain Scalability (MGS) 向量进一步划分为了 2 个 MGS 层，分别包含 4 个和 12 个变换系数。我们从码流中随机丢弃一组数据包，将得到的子流解码，然后比较实际的失真与通过线性误差模型估计的失真。解码用的是 JSVM 解码器，其他计算都通过 MATLAB 进行。

对于序列 *Bus*、*Foreman*、*Soccer*，图3.3分别给出了其中一个比较结果（为简单起见，只采用了前 33 帧）。图中的横轴代表帧序号，纵轴表示以 MSE 度量的失真。实线代表真实计算的失真，虚线代表用所提出的线性误差模型估计的失真。可以看到虚线和实线非常接近，验证了用提出的模型做失真估计的准确性。我们对所有 8 个 SVC 标准序列都进行了实验，定量计算了估计误差，如表3.1所示。可以看到，平均估计误差只有 5%，进一步说明了模型的准确性。

表 3.1 采用线性误差模型进行不同序列失真估计的估计误差

| <i>Bus</i> | <i>City</i> | <i>Crew</i> | <i>Football</i> | <i>Foreman</i> | <i>Harbour</i> | <i>Mobile</i> | <i>Soccer</i> | Average |
|------------|-------------|-------------|-----------------|----------------|----------------|---------------|---------------|----------------|
| 2.83% | 2.39% | 10.36% | 7.08% | 4.95% | 2.63% | 5.77% | 5.26% | 5.16% |

3.2 码流截取方案

在这一节中，我们采用上面提出的线性误差模型来进行码流截取。因为找到码流截取问题的最优解不太现实，我们的方案采用贪心法^①来获得一个次优解。与“0-1 背包问题”的贪心解法类似，每次当一个数据包需要被丢弃时，我们选择具有最小的码率失真影响的包。一个包的码率失真影响可以被认为决定它是否重要的优先级度量标准。下面我们首先对所采用的这个度量标准做简单的介绍，然后具体给出优先级赋值的算法。

3.2.1 数据包优先级度量

对于一个数据包，我们采用如下的码率失真影响来度量它的优先级：

$$\Phi = \frac{\partial D}{\partial R}, \quad (3.11)$$

^①https://en.wikipedia.org/wiki/Greedy_algorithm

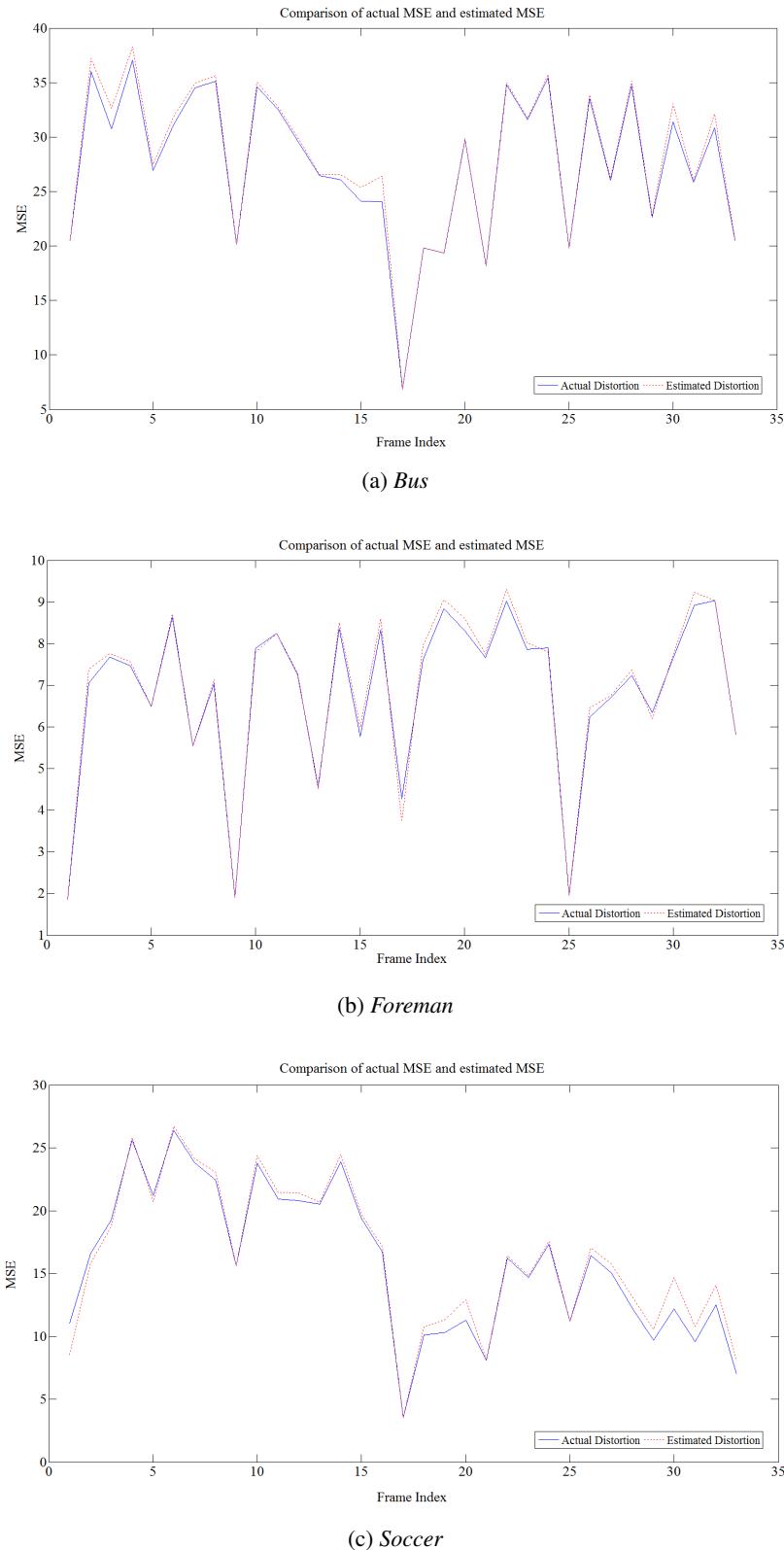


图 3.3 不同序列估计失真与实际失真的比较

其中 ∂D 表示丢弃这个数据包所带来的失真的变化，而 ∂R 是相应的码率的变化。 ∂R 显然只与该数据包本身的大小有关，很容易得到。但 ∂D 的真实值需要通过分别计算丢包前后的失真来求取，在码流截取中只能估计。

这一度量标准具有直观上的意义。如果丢弃了一个数据包之后所带来的码率变化很小 (∂R 小)，而丢掉它产生的失真变化却很大 (∂D 大)，那意味着这个数据包非常重要，其优先级高 (Φ 大)，在码流截取中应该尽量予以保留。这样的数据包本身没有很多数据，但它所处的帧被较多的其他帧所参考，丢弃之后具有较大的失真影响。采用公式 (3.11) 所示的度量标准能够全面考虑到这种情形。

3.2.2 优先级赋值的贪心算法

我们将模拟依次从一个码流中丢掉所有增强层数据包的过程，丢包顺序就按照上面所说的贪心策略。这个顺序就作为数据包的优先级值被记录下来。这些优先级值可以被存储在码流中 (NALU 头信息的优先级 ID 字段或者补充增强信息 SEI 中)，用于实际的码流截取。本节详细介绍优先级赋值的算法。

对于很长的视频序列，往往将其分为一个个优化窗口进行操作。每个窗口含有一定数量的帧，优先级赋值和码流截取都在每个窗口中独立地进行。假设优化窗口的长度为 K ，也就是说每次处理 K 帧；再假设每一帧包含的质量伸缩层个数为 L_Q ，也就是一个基本层加 $L_Q - 1$ 个增强层。这样的话所有能够被丢弃的增强层数据包的个数将是 $K \cdot (L_Q - 1)$ 。在每一次贪心选择的步骤中，当前可以被丢弃的数据包只能是每一帧最上层的数据包（因为下层的包被上层所依赖），也就是最多有 K 个。在这样的优化窗口中进行优先级赋值的过程具体描述如下：

- 1) 初始状态下，把整个序列的误差向量 e_{seq} 设为 e_{full} （所有增强层数据包都在时解码出的序列与原始序列的差值，参见公式 (3.10) 的说明）。
- 2) 在所有当前可以丢弃的数据包中，找到对序列具有最小码率失真影响的数据包 m ，即寻找 m ，使得：

$$\Phi_m = \min_{i \in I_{top}} \Phi_i, \quad (3.12)$$

其中 I_{top} 代表当前处于每一帧最上层的数据包，现在只有它们是可以丢弃的。一个包的失真影响基于公式 (3.11) 计算如下：

$$\Phi_i = \frac{\partial D}{\partial R} = \frac{PSNR(e_{seq}) - PSNR(e_{seq} + e_i)}{SIZE(i)}, \quad (3.13)$$

其中 $SIZE(i)$ 是数据包 i 的大小， $PSNR(*)$ 是由误差向量求取 PSNR 的函数。PSNR 是由 MSE 定义的，而 MSE 就是误差向量的元素平方后的均值。

3) 将数据包 m 从码流中移除，并把它的误差向量加到序列误差向量中：

$$e_{seq} = e_{seq} + e_m . \quad (3.14)$$

4) 重复 2) 到 3) 的步骤，直到所有的增强层数据包全部被移除，且每个包都根据它被移除的次序被赋予了优先级值。

上面描述的过程用算法伪码表示可参见 Algorithm1。

Algorithm 1 优先级赋值的贪心算法

```

初始化  $e_{seq}$  为  $e_{full}$ , 初始化  $q$  为 0;
while  $q < K \cdot (L_Q - 1)$  do
    for (最多)  $K$  个处于每帧最上层的数据包 do
        找到具有最小失真影响  $\Phi_m$  的数据包  $m$ ;
    end for
    设置  $m$  的优先级为  $q$ ;
    将  $m$  从码流中移除;
    将  $e_{seq}$  更新为  $e_{seq} + e_m$ , 并将  $q$  更新为  $q + 1$ ;
end while

```

从伪码很容易看出优先级赋值算法的时间复杂度为 $O(K^2)$ 。需要指出的是，在优先级赋值过程中的这些计算量相比于误差向量获取过程中解码的计算量来说是微不足道的。通过增加优化窗口大小 K ，可以每次比较更多的数据包来确定优先级，因此有望得到更优的截取结果。然而，这样做不仅增加计算量，更重要的是会使存储误差向量所需的内存有显著的增加。此外，如果优先级值采用 NALU 头信息中的字段存储的话，因为该字段只有 6 个比特，所支持的范围是 0 ~ 63，这也限制了 K 的大小。

3.2.3 无参考源的情况

在某些场合，优先级赋值和码流截取需要在没有原始视频序列的情况下进行。这将增加问题的难度，因为当原始序列未知的时候更难以获取实际失真。这种无参考的情形在实际中甚至比有参考的情形更常见。因为通常来说，流媒体服务提供商拿到的都是压缩后的码流，很少能拿到原始视频序列。在这一小节中，我们对所提出的算法稍作修改以适应这种没有参考源的特殊情况。

在缺少原始序列的情况下，我们无法计算上面提到的 e_{full} 。一个简单直接的想法是把 e_{full} 置为 0，也就是所有失真都是基于整个码流完全重建的像素来计算，而不是基于原始序列。但我们发现这样做效果并不好，算法过程中的 PSNR 值太大，无法很好的反映实际失真。这是因为丢弃一些数据包后，与完全重建像素比较起来并不能带来很大的误差，据此计算 PSNR 是不合适的。为了解决这个问题，我们改为直接用所

有像素的均方差 (MSE) 作为失真的度量标准，不再计算 PSNR。即，公式 (3.13) 应修改为：

$$\Phi_i = \frac{\partial D}{\partial R} = \frac{MSE(e_{seq}) - MSE(e_{seq} + e_i)}{SIZE(i)}, \quad (3.15)$$

其中， $MSE(*)$ 表示直接由误差向量得到 MSE。这样做使得无参考情况下的截取效果有所提升，被证明是一个合理有效的改进（参见下一节中的相应实验结果）。

3.2.4 优先级值的实际使用

上面所述的算法为可伸缩视频码流中的每个增强层数据包赋予了一个优先级值，这些优先级值将用于实际的码流截取。本小结具体介绍在实际系统中如何使用优先级值来进行码流截取。

首先，系统应该提前确定每个优先级值与码率的对应关系，这只需要进行一次性的统计操作。有了这个对应关系之后，在给定的截取码率下，就知道了应保留的数据包优先级下限。假设这个下限为 P ，则在传输时，所有增强层数据包优先级低于 P 的都应该丢弃。在发送一个数据包前，传输程序首先检查它是基本层数据包还是增强层数据包。如果是前者，则发送；如果是后者，则读取其优先级值与 P 比较。如果大于等于 P ，则发送；如果小于 P ，则将其丢弃。

在我们所实现的支持可伸缩视频的达尔文流媒体服务器中，传输时每次发送的是一个 RTP 包，而一个 RTP 包中可能包含多个或者不到一个 SVC 数据包，即 NALU（参见2.2.1小节）。考虑到这一点，服务器程序应该按照如图3.4所示的流程来实现发送前的码流截取。

3.3 实验结果

本节通过与参考软件 JSVM (9.16 版本) 的对比来展示所提出的码流截取方案的实验结果。

3.3.1 实验配置

所有的测试码流都是按照3.1.3中模型验证时所述的配置编码得到的。对于每个码流，我们选取十个截取点：一个是包含所有增强层数据包的完整码流，一个是只含有基本层数据包的最低码率码流，二者之间又等距离地选取了 8 个码率点。对每个码率点，我们用三种方法进行截取：JSVM 中的基本截取器（用 JSVM Basic 指代），JSVM 中采用了 Quality Layers 的截取器（用 JSVM QL 指代），以及本文提出的截取器。在本文提出的截取方案中，优先级赋值算法里的优化窗口大小是 32 帧，也就是 4 个 GOP

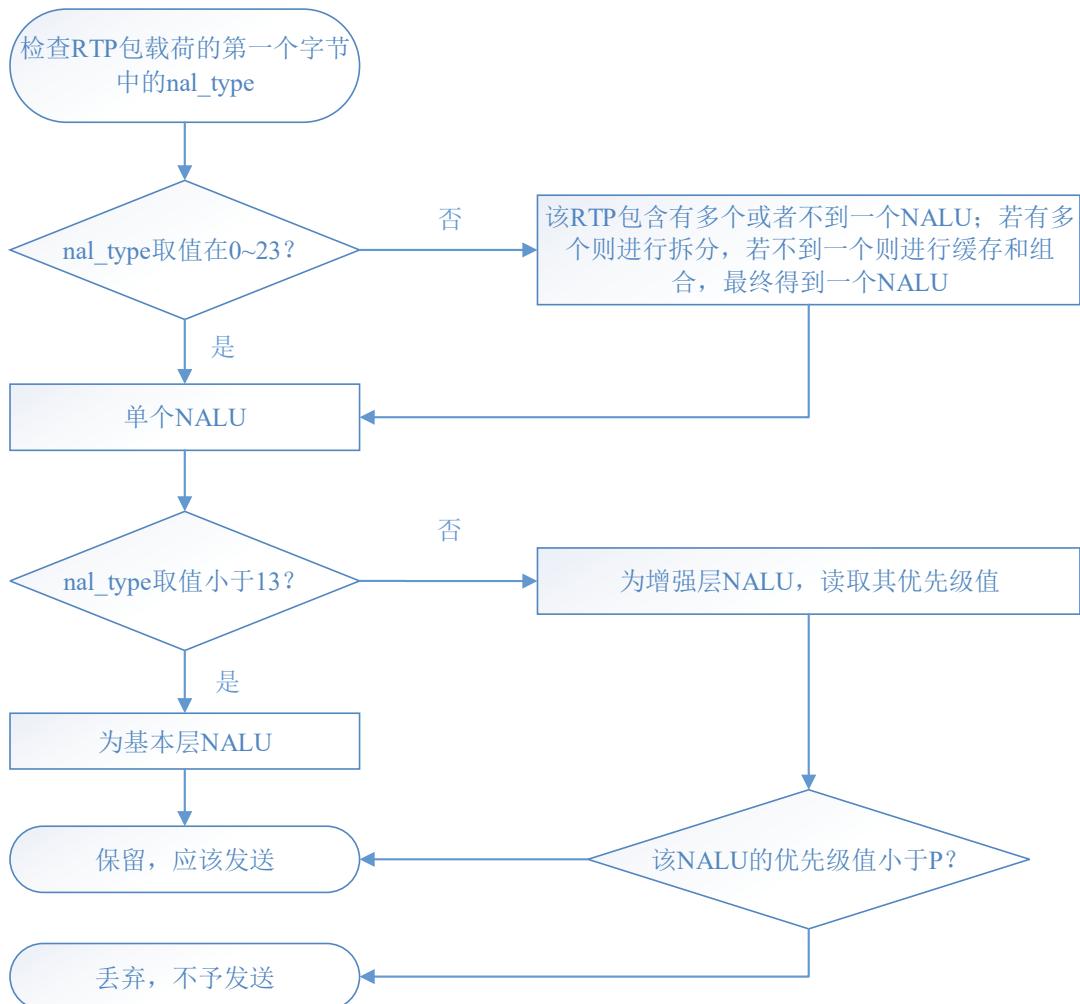


图 3.4 使用优先级值进行实际码流截取的程序流程图

(GOP 大小是 8)。所有 CIF 大小的 SVC 标准测试序列都进行了测试，码率计算时采用了 30FPS 的帧率。

3.3.2 结果分析

图3.5展示了全部八个测试序列的性能比较结果。可以看出，本文提出的方法对应的码率失真曲线几乎总是在最上方。这意味着在同样的码率限制下，本文提出的算法能取得更好的视频质量。注意到率失真曲线的第一个和最后一个点分别对应着最大程度截取和不进行截取的情况；在这两种情况下各种截取算法的结果是一样的，因此图中的三条曲线在这些点重合。

表3.2列出了本文提出的截取方案相对于比较对象的 PSNR 提升。对于每一个测试

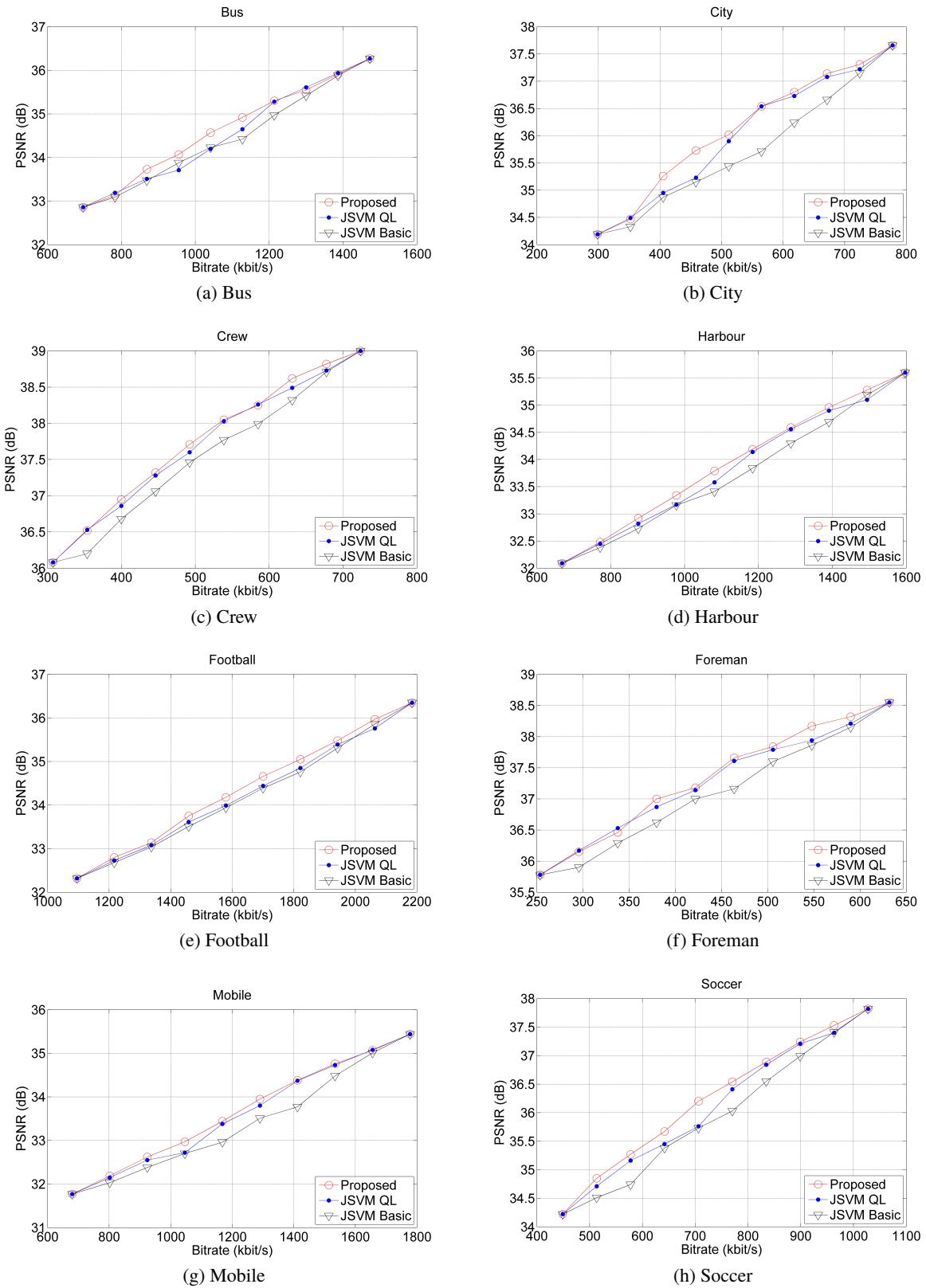


图 3.5 三种码流截取方案的性能比较

表 3.2 本文提出的截取方案相比于 JSVM 的 PSNR 提升

| 序列名 | | <i>Bus</i> | <i>City</i> | <i>Crew</i> | <i>Football</i> | <i>Foreman</i> | <i>Harbour</i> | <i>Mobile</i> | <i>Soccer</i> |
|----------------------|------------------|------------|-------------|-------------|-----------------|----------------|----------------|---------------|---------------|
| JSVM QL, $K = 32^a$ | Max ^b | 0.37 | 0.50 | 0.13 | 0.22 | 0.23 | 0.19 | 0.25 | 0.44 |
| | Ave ^c | 0.11 | 0.11 | 0.05 | 0.12 | 0.05 | 0.05 | 0.06 | 0.13 |
| JSVM Basic, $K = 32$ | Max | 0.50 | 0.83 | 0.32 | 0.29 | 0.50 | 0.34 | 0.61 | 0.53 |
| | Ave | 0.18 | 0.37 | 0.20 | 0.15 | 0.22 | 0.15 | 0.25 | 0.29 |
| JSVM QL, $K = 48$ | Max | 0.36 | 0.39 | 0.27 | 0.30 | 0.30 | 0.38 | 0.21 | 0.40 |
| | Ave | 0.12 | 0.23 | 0.16 | 0.13 | 0.15 | 0.13 | 0.08 | 0.21 |
| JSVM Basic, $K = 48$ | Max | 0.36 | 0.81 | 0.32 | 0.29 | 0.49 | 0.47 | 0.57 | 0.60 |
| | Ave | 0.19 | 0.40 | 0.19 | 0.17 | 0.28 | 0.22 | 0.28 | 0.42 |

^a 与 JSVM QL 对比；优化窗口大小为 32 帧

^b 所有码率限制下的最大提升

^c 所有码率限制下的平均提升

序列，10 个码率点的最大和平均 PSNR 提升都列了出来。可以看到，我们的方法相对于 JSVM 有明显的性能优势。该表中还列出了当优化窗口设置为 48 帧（也就是 6 个 GOP）时的结果，证明了通过增大优化窗口确实能提升性能。不过在实际系统中，窗口大小还需考虑其他因素，并非能任意增大（参见3.2.2小节末尾的分析）。

最后需要指出，在 JSVM 之外的其他码流截取方法中，Maani 等人的工作^[62]取得了比 JSVM QL 高得多的性能，甚至比本文提出的方法性能还好。但是，为了得到适用于不同序列的鲁棒的模型参数，该工作中的方法需要更多的计算量去训练。而本文提出的方法所需计算量总是与 JSVM 相当的。据我们所知，在不显著增加计算量的情况下，码流截取很难取得较大的 PSNR 提升。

3.3.3 无参考截取实验

对于无参考源的应用场景，我们也做了相应的实验。图3.6展示了全部八个测试序列在无参考截取实验中的性能比较结果，直观地表明了本文所提方案取得了较好的率失真性能。表3.3列出了对应的 PSNR 提升数据，同样可以看出本文所提出的方案的优越性。

为了进一步证明本文提出的码流截取方案在无参考情况下的有效性，我们除了与 JSVM 对比，还做了进一步的对比实验。比较对象一方面选了 Maani 等人的方法^[62]，另一方面选了不进行本文3.2.3小节中的修改而直接采用公式(3.13)的方法。由于 Maani 等人的方法所取得的结果取决于其训练出的模型参数，为了公平起见，我们控制了训练过程的计算量，使其与我们的方法计算复杂度相同。表3.4列出了三种不同的无参考截取方法相对于 JSVM QL 的平均 PSNR 提升。可以看出，Maani 等人的方法在无参考情况下跟本文的方法性能相差无几，而在进行本文3.2.3小节中的修改之前，直接采用公式(3.13)则性能差了许多，甚至在有些序列上取得了比 JSVM QL 还差的结果。这进

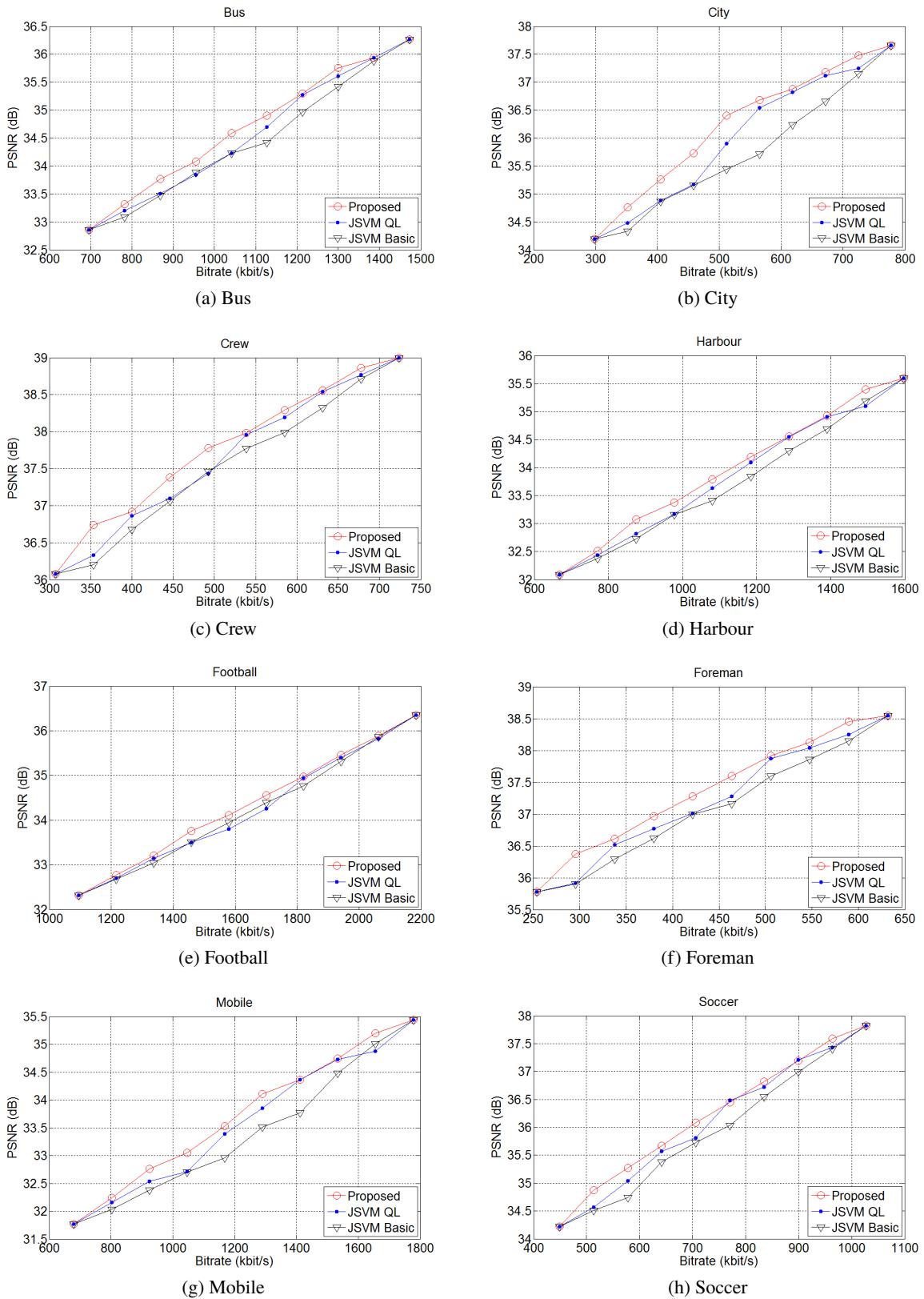


图 3.6 无参考源时三种码流截取方案的性能比较

表 3.3 无参考源时本文提出的截取方案相比于 JSVM 的 PSNR 提升

| 序列名 | | <i>Bus</i> | <i>City</i> | <i>Crew</i> | <i>Football</i> | <i>Foreman</i> | <i>Harbour</i> | <i>Mobile</i> | <i>Soccer</i> |
|----------------------|------------------|------------|-------------|-------------|-----------------|----------------|----------------|---------------|---------------|
| JSVM QL, $K = 32^a$ | Max ^b | 0.36 | 0.56 | 0.41 | 0.31 | 0.45 | 0.30 | 0.34 | 0.30 |
| | Ave ^c | 0.14 | 0.22 | 0.13 | 0.12 | 0.17 | 0.11 | 0.14 | 0.11 |
| JSVM Basic, $K = 32$ | Max | 0.48 | 0.97 | 0.54 | 0.25 | 0.47 | 0.38 | 0.60 | 0.53 |
| | Ave | 0.23 | 0.48 | 0.23 | 0.12 | 0.28 | 0.21 | 0.31 | 0.26 |

^a 与 JSVM QL 对比; 优化窗口大小为 32 帧

^b 所有码率限制下的最大提升

^c 所有码率限制下的平均提升

表 3.4 不同的无参考截取方法对比: 相对于 JSVM QL 的平均 PSNR 提升

| 序列名 | <i>Bus</i> | <i>City</i> | <i>Crew</i> | <i>Football</i> | <i>Foreman</i> | <i>Harbour</i> | <i>Mobile</i> | <i>Soccer</i> |
|------------------|------------|-------------|-------------|-----------------|----------------|----------------|---------------|---------------|
| 本文方法 | 0.14 | 0.22 | 0.13 | 0.12 | 0.17 | 0.11 | 0.14 | 0.11 |
| Maani 等人的方法 [62] | 0.13 | 0.22 | 0.13 | 0.11 | 0.16 | 0.02 | 0.08 | 0.12 |
| 采用公式 (3.13) 方法 | 0.06 | 0.07 | 0.01 | 0.10 | 0.05 | 0.04 | -0.03 | -0.01 |

一步说明了3.2.3小节中修改的重要性。

3.4 本章小结

本章介绍了一个采用线性误差模型的可伸缩视频码流截取方案。我们从 H.264/AVC 解码过程的线性性质推导出了质量可伸缩情况下的线性误差模型。该模型能使我们准确地估计丢弃任意数据包所带来的失真变化。基于此，我们设计了一个贪心算法来根据数据包的码率失真影响来为其赋优先级值，并按照这些优先级值来进行码流截取。实验表明，本文提出的线性误差模型对失真估计的误差率只有 5%，而采用该模型的码流截取相比于参考软件中的截取器有高达 0.5dB 的 PSNR 提升。本章的工作使得视频流媒体的数据源能够以很高的精度改变码率，为下一章要介绍的码率自适应提供了条件。

第四章 基于 PID 控制思想的点播系统码率自适应算法

在实际网络环境中，带宽波动是不可避免的。对于一个视频流媒体系统而言，只有根据带宽变化来自适应地调整发送码率才能为用户提供更好的服务。在上一章中，我们通过可伸缩视频的码流截取，使得码率能够调整；在本章中，我们主要关注如何调整。由于视频码率与视频质量一般来说是直接相关的（高码率对应高质量），调整码率也就是调整所传送视频的质量。因此，本章所解决的问题又被称为质量控制问题。我们将基于经典控制实践中的 PID 思想，为采用达尔文流媒体服务器的直播系统设计一个综合考虑带宽的历史状况、当前状态和未来趋势的码率自适应算法，提高所发送视频的总体质量和平滑度。

4.1 PID 控制器简介

PID 控制器是一个在工业控制系统中常用的闭环反馈控制器。PID 控制中的两个重要概念是过程变量和控制目标。过程变量代表着系统当前的状态，而控制目标是系统希望达到的理想状态。PID 控制的基本思想就是根据过程变量和控制目标之间的误差来确定一个控制输出，该输出反馈作用到系统以最小化上述误差。

PID 控制器的控制输出一般如下定义：

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) d\tau + K_d \cdot \frac{d}{dt} e(t), \quad (4.1)$$

其中 $e(t)$ 指代过程变量与控制目标在时刻 t 的误差， K_p 、 K_i 、 K_d 是三个可调参数，依次称为比例增益、积分增益和微分增益。公式 (4.1) 的结果通过某种方式来影响系统状态，使得过程变量朝着控制目标变化，如图4.1^①所示。在 PID 控制器中，比例部分直接对应着当前系统误差，积分部分反映了过去一段时间内误差的累积，微分部分则表示对误差变化情况的预测。将这三个部分结合起来作为控制器输出，能够使系统综合考虑过去、现在和未来的关系，更好地进行调整决策。

作为一个灵活有效的控制技术，PID 可以被用来解决各种各样的控制问题，例如视频编码过程中的码率控制^[75-79]。在本文的工作中，我们基于它设计了一个码率自适应算法，用来控制视频流媒体中的视频质量。

^①http://en.wikipedia.org/wiki/PID_controller

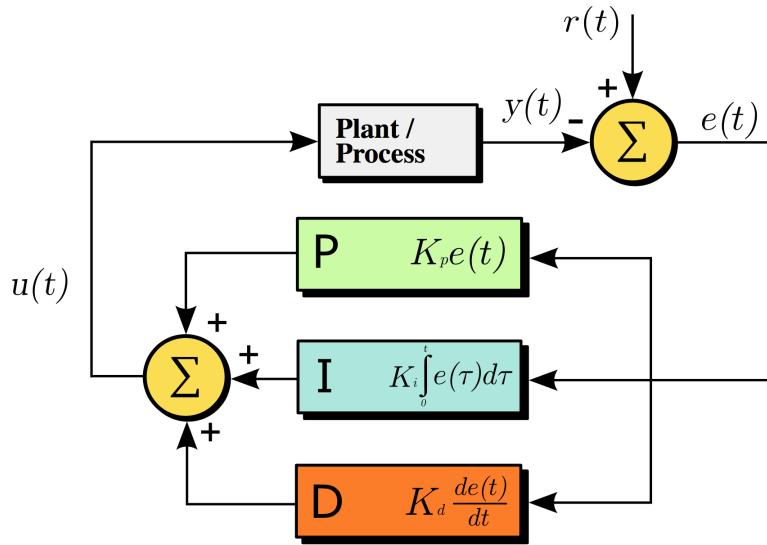


图 4.1 PID 控制器原理示意图

4.2 基于 PID 的码率自适应

本节首先介绍质量等级的概念，将视频流媒体中的视频质量和码率值离散化，接着定义在所提出的码率自适应算法中用到的过程变量及其控制目标。之后，我们详细解释控制模型和算法流程，并简要讨论 PID 参数的选取和调优。

4.2.1 质量等级划分

视频质量或码率是一个连续的值，以任意的精度来控制或调整它是不现实的。因此，我们首先提出一个质量层级的概念来离散化视频质量和码率。当需要调整时，我们只是增加或减少质量等级。每一个质量等级对应一个确定的码率。高等级对应高码率，反之亦然。有多少个质量等级以及它们跟具体码率值如何对应该可以根据实际需求来确定。质量等级越多，系统就能越精细地调整码率来适应带宽变化。如绪论中所说，用多码流实现的自适应流媒体系统只能提供有限的几个质量等级，而基于可伸缩视频编码的流媒体系统能够以更高的精度来定义质量等级。

4.2.2 过程变量和控制目标

本章所提出的质量控制算法主要是针对达尔文流媒体服务器的视频传输过程，所选取的过程变量称为“实际与检测间隔比”。下面介绍其具体定义。

在达尔文流媒体服务器中，有两个时间线非常重要。第一个是推送时间线（push time），也就是视频数据包被推送到缓冲区的时间；第二个是播放时间线（play time），也就是数据包应该在播放中被显示的时间（通常记作该数据包的 PTS）。这两个时间线

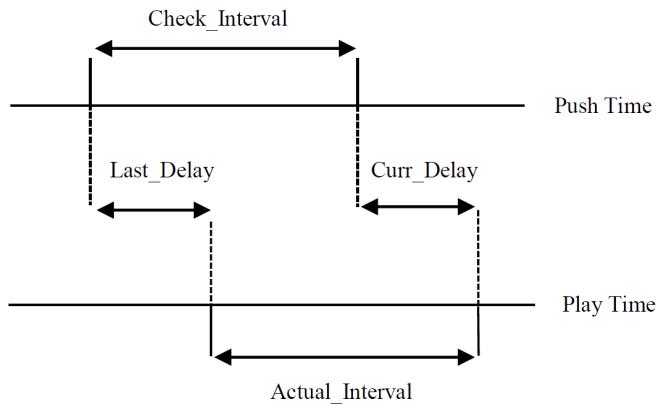


图 4.2 达尔文流媒体服务器中的两个时间线

的关系如图4.2所示。服务器程序会定时检测这两个时间线，并根据数据包的延迟来决定是否需要调整发送的视频质量。

图4.2中的两个间隔，即检测间隔（check interval）和实际间隔（actual interval），值得我们注意。在理想情况下，每次检测间隔中服务器推送的数据量应该恰好能实际播放等于检测间隔的一段时间，也就是 $actual_interval = check_interval$ 。如果实际间隔小于检测间隔，说明这段时间内推送了较少的数据，推送数据变慢很可能是带宽有所下降。反过来，如果实际间隔大于检测间隔，说明这段时间内推送了较多的数据，意味着网络条件良好且数据被快速推向客户端。这两个间隔的比值，也就是所谓的“实际与检测间隔比”，能够反映当前推送的速率是否与网络带宽相符。

“实际与检测间隔比”适合用作质量控制有三方面的原因：

- 第一，这个比例直接对应着适合带宽的码率与当前传输的码率的比值，只需据此进行调整就可以准确地使码率符合带宽。
- 第二，根据定义这个比例的理想值就是 1，这就是算法的控制目标，无需再通过别的方法确定。
- 第三，这个值在实际系统中便于计算，普适性较好。

4.2.3 控制模型和算法描述

在通常的 PID 控制器中，控制输出是用过程变量与控制目标之间的差值进行比例、积分、微分求取出来的。考虑到我们所选的过程变量是一个比例，且控制目标是 1，本文对通常的 PID 控制器做了修改，提出了一个基于比例的模型。在这个模型中，过程变量直接被用来计算控制器输出，而且所有求差的运算都被适当的改为了求比例的运

算, 以保持模型的合理正确性, 如下所示:

$$u_t = K_p \cdot E_p^t + K_i \cdot E_i^t + K_d \cdot E_d^t, \quad (4.2)$$

$$E_p^t = \frac{\text{actual_interval}_t}{\text{check_interval}_t}, \quad (4.3)$$

$$E_i^t = \frac{\text{long_actual_interval}_t}{\text{long_check_interval}_t}, \quad (4.4)$$

$$E_d^t = E_p^t / E_p^{t-1}, \quad (4.5)$$

$$\text{long_actual_interval}_t = \sum_{\tau=T_l}^t \text{actual_interval}_{\tau}, \quad (4.6)$$

$$\text{long_check_interval}_t = \sum_{\tau=T_l}^t \text{check_interval}_{\tau}. \quad (4.7)$$

在公式 (4.2) 中: K_p 、 K_i 、 K_d 分别表示比例部分、积分部分、微分部分的可调参数; E_p^t 、 E_i^t 、 E_d^t 则对应各部分在第 t 次检测时的值, 它们的定义如公式 (4.3) 至 (4.7) 所示; 控制器输出 u_t 将用来计算适合当前带宽的码率, 并据此选择要传送的质量等级。

在公式 (4.6) 和 (4.7) 中, T_l 代表上一次质量等级改变的时间。按照理论定义, 积分部分应该从系统启动开始记录, 但在实际环境中, 网络带宽有可能变动非常大, 如果积分部分记录累积太长时间将不适合反映当前的网络条件, 从而减缓调整质量的决策。因此这里采用了一个折衷, 选择从上一次质量等级变化的时间开始计算积分部分。

这个基于比例的模型对于所选的过程变量和控制目标比通用模型更加简单有效。而且, 在该模型中, 当三个参数进行归一化处理后, 控制器输出具备明显直观的意义。从公式 (4.3) 至 (4.7) 容易看出, E_p^t 、 E_i^t 、 E_d^t 这三个值都是在 1 附近波动的比值; 而根据公式 (4.2), 控制器输出 u_t 可以被视作这三个比值的加权平均 (三个参数作为权值)。因此, u_t 也是一个在 1 附近的比值, 它对应着适合当前带宽的码率与当前发送码率的关系。换句话说, 如果 u_t 大于 1, 那么控制器决定当前发送码率需要增加; 反之, 如果 u_t 小于 1, 意味着控制器认为当前发送码率需要减小。用 u_t 作为一个乘子来调整当前发送的码率, 我们就能控制流媒体系统符合带宽的变化。

本文提出的码率自适应算法描述如 Algorithm2 所示。相比于其他的调度策略, 这个基于 PID 的自适应算法具有以下几个优点:

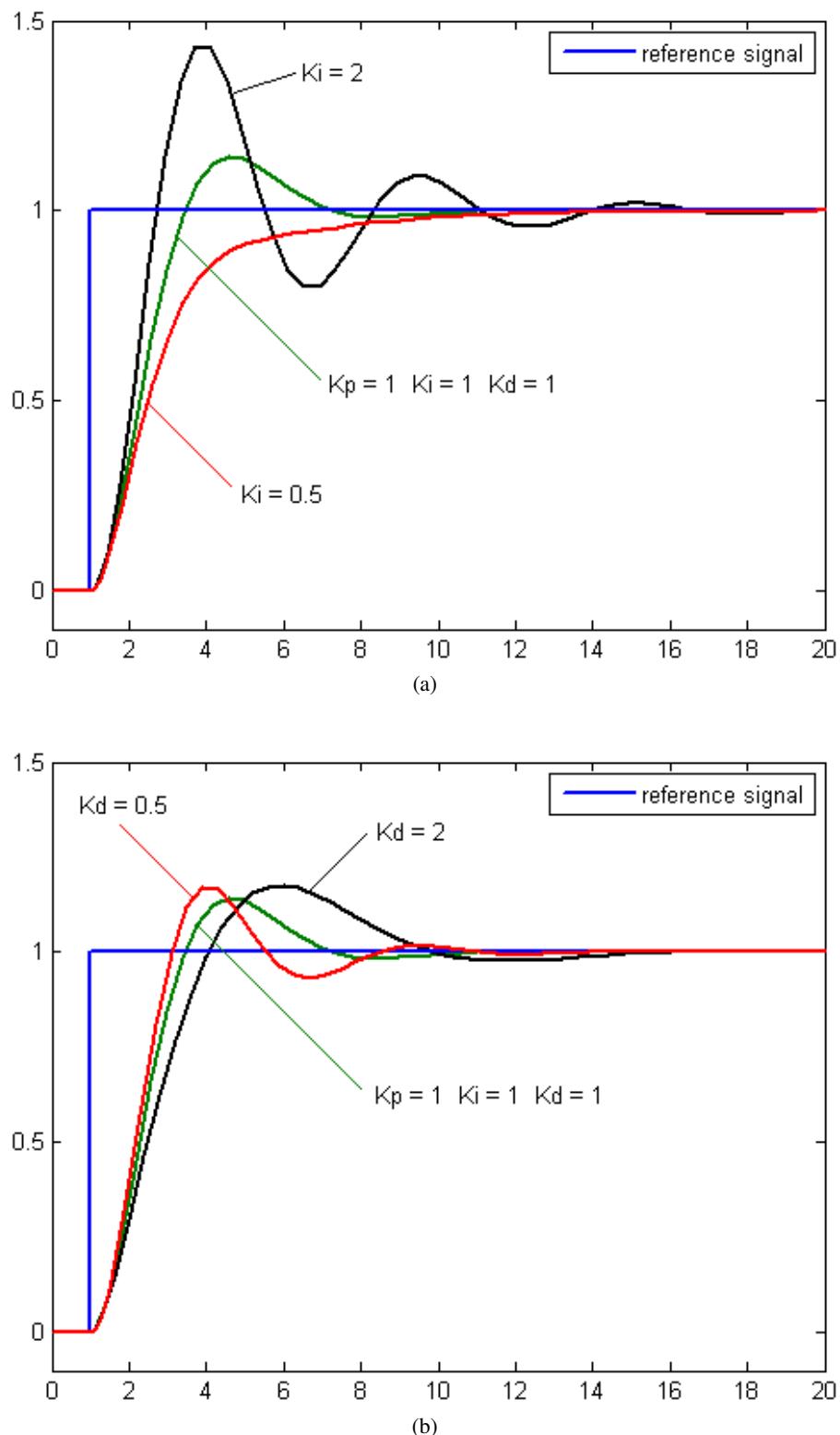
- 首先，该算法不仅仅依据当前状态或者固定时间点的信息来做决策，因此质量调整可以在任何合适的时间发生。
- 其次，因为 PID 模型（微分部分）在某种程度上考虑了对未来的预测，该算法能够判断带宽在未来一段时间内是否会变得足够大或足够小，因此能避免一些不必要的调整。
- 最后，该算法并不一定一次只调整一个质量等级，这就使得当带宽剧烈下降时能够快速下调质量以保证流畅播放，而当系统启动时能够快速上调质量以使用户尽早收到更清晰的视频。

Algorithm 2 基于 PID 的码率自适应算法

准备发送数据包
check_interval = curr_time - last_check_time
actual_interval = curr_media_time - last_check_media_time
long_check_interval += check_interval
long_actual_interval += actual_interval
根据公式 (4.3) - (4.5) 计算 E_p^t 、 E_i^t 、 E_d^t 的值
 $output = K_p \cdot E_p^t + K_i \cdot E_i^t + K_d \cdot E_d^t$
new_bitrate = output * curr_bitrate
new_level = get_level(new_bitrate, bitrate_of_level[])
if 质量等级变化 **then**
 long_check_interval = 0
 long_actual_interval = 0
 curr_bitrate = bitrate_of_level[new_level]
end if
根据新的质量等级发送数据包

4.2.4 参数选取与调优

与工业界的其他 PID 控制系统一样，本文提出的算法中三个参数的选取也是跟具体实现相关的。大多数情况下，为取得最优的效果，需要手动试错调节。每个参数对系统的不同方面产生影响，知道这些影响有助于进行参数选取。一般来说，PID 控制器中的比例部分会影响系统的稳定性。如果比例增益 K_p 过高，系统容易变得不稳定。但是，太低的 K_p 又会导致过长的上升时间，使系统响应迟缓。积分部分除了能消除稳态误差，还能加速过程变量向控制目标的变化。这意味着，增大 K_i 能减小上升时间，提高系统的响应速度。但是，积分部分会累积过去一段时间的误差，使过程变量容易超出目标值（即超调），因此过大的 K_i 也会导致系统不稳定。上述这些隐患都能通过微分部分来校正。微分部分通过预测系统行为，具有减小超调、提高稳定性的双重效果。但是微分部分的特点是对误差非常敏感，也正是因为此，工业实践中微分增益 K_d 通常

图 4.3 PID 控制器中参数 K_i 、 K_d 对系统的影响

都设置的比较低。参数 K_i 、 K_d 对系统的影响效果可以参见图4.3^①。

本文采用了经典的 Ziegler–Nichols 方法^[80] 来进行参数选取。该方法首先把积分和微分参数设为 0，调节比例参数 K_p 到系统开始震荡时的值 K_u ，假设震荡周期为 T_u ，则按照以下公式设置三个参数：

$$K_p = 0.6K_u, \quad (4.8)$$

$$K_i = 2K_u/T_u, \quad (4.9)$$

$$K_d = K_u \cdot T_u/8. \quad (4.10)$$

最后将 K_p 、 K_i 和 K_d 进行归一化，就得到最终的参数： $K_p = 0.22$ ， $K_i = 0.73$ ， $K_d = 0.05$ 。在这样的参数配置下，由本文提出的算法所控制的流媒体系统是稳定的，而且能够很快开始适应带宽的变化，具体参见下一节中的实验结果。

4.3 实验结果

本章提出的基于 PID 的码率自适应算法集成在了达尔文流媒体服务器中，本节通过对比实验来验证该算法的有效性。

4.3.1 实验配置

为了简单起见，我们先用可伸缩视频内置的伸缩层级组合成质量等级来用于质量控制。该定义的基本思想是将可伸缩视频码流中的数据包分成若干组。分组的依据是这个数据包所属的伸缩层，也就是储存在 NALU 头信息中的层 ID 字段。表4.1展示了组 ID 与层 ID 的对应关系。我们所用的测试码流含有 3 个质量层和 3 个时间层，因此其中的数据包被分为 9 个组。质量等级被定义为组 ID 不大于等级值的所有组的合集。例如，质量等级 2 含有组 0 到组 2 内的所有数据包，因此将包含所有的时间层，但只有质量基本层；质量等级 5 含有组 0 到组 5，于是将进一步包括第一个质量增强层的数据包。类似这样，随着质量等级的升高，越来越多的增强层数据包被累积进来，视频质量逐渐提升；与此同时，码率也越来越高，因为更好的质量显然需要更高的码率。我们测试用的完整码流码率是 1587 kbps，对应着最高的质量等级。这样用伸缩层来构造质量等级可以不用额外的数据，只需解析码流中本来已经包含的层 ID 即可，实现起来比较简单。我们实验中所用的可伸缩码流一共可定义 9 个质量等级。

^①http://en.wikipedia.org/wiki/PID_controller

表 4.1 质量等级定义中组 ID 与层 ID 的对应关系

| 组 ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|--------|---|---|---|---|---|---|---|---|---|
| 质量层 ID | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
| 时间层 ID | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |

在我们的实验中用限速软件 NetLimiter^①来模拟实际网络环境中的带宽情况。平均带宽的取值从 400kbps 到 1400kbps，以 100kbps 的间隔递增。对于每个平均带宽值，通过在平均值基础上加随机干扰来模拟带宽波动。在带宽波动的情况下，流媒体系统在每种质量控制算法下各运行 10 分钟，并记录所发送的质量等级变化。所有质量等级的平均值作为视频质量的度量，而所有质量等级的方差作为质量平滑性的度量。这两个指标被用于不同算法的比较。比较中的基准算法是达尔文流媒体服务器中内置的包延迟反馈（Packet Delay Feedback， PDF）算法。

接下来，我们展示各种质量控制算法的结果并进行分析比较。

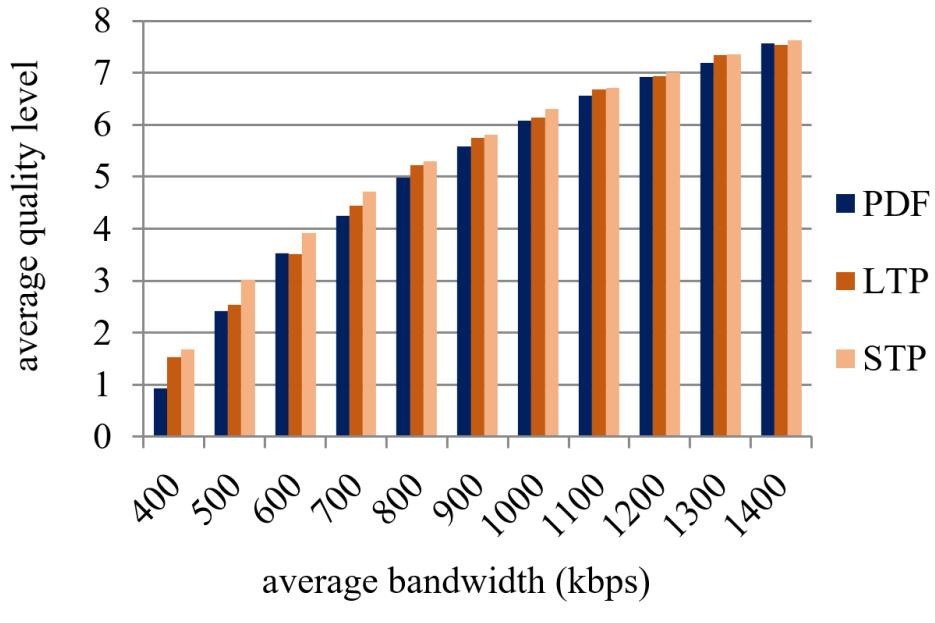
4.3.2 结果分析

为了分析 PID 控制模型各个部分的效果，我们首先只用比例部分和积分部分来进行了初步的测试实验。由于这两个部分各自对应当前的短期误差和积累的长期误差，我们分别称之为短期预测算法（Short-Term Prediction， STP）和长期预测算法（Long-Term Prediction， LTP）。图4.4展示了这两种控制算法和基准算法的实验结果。可以看出，短期预测算法能取得最高的平均质量等级，但它的质量等级方差也是最高的。这正符合短期预测总是倾向于紧紧跟随带宽变化的特点。长期预测算法恰恰与之相反，保持了一个较低的质量等级方差，但也牺牲了一部分平均质量。虽然它的平均质量不如短期预测算法，但也超出了基准算法。

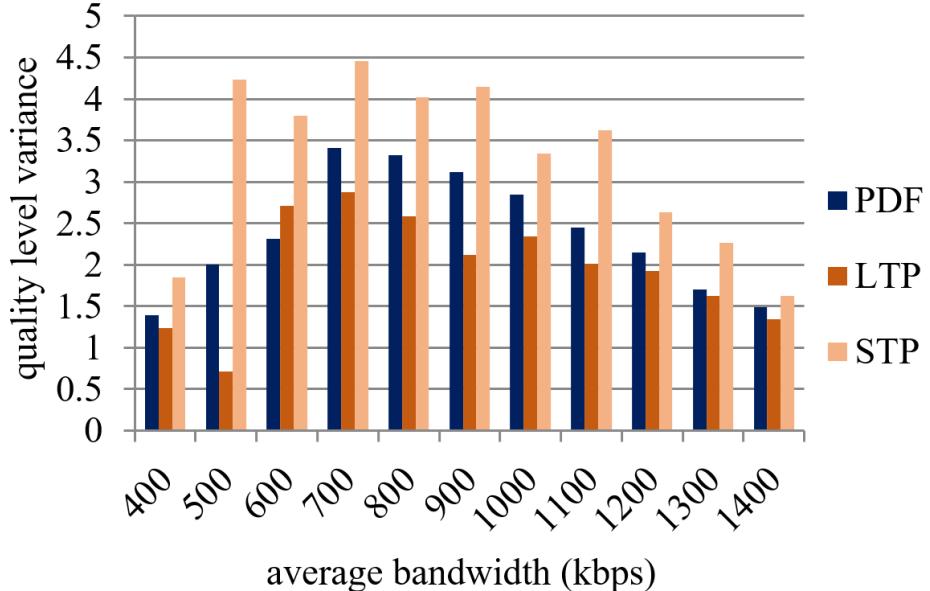
短期预测和长期预测算法有各自的局限性，完整的 PID 控制算法则既能降低质量波动又能提升平均质量。实验结果可参见图4.5。可以看到，当与积分和微分相结合后，比例部分在短期预测算法中那样的激进效果已经弱化了。如图4.5b所示，PID 算法取得了比 PDF 算法更低的质量波动。

表4.2列出了四种控制算法的定量比较数据，从中我们可以看出 PID 算法相比其他几种算法有明显的优势。与基准算法 PDF 相比，基于 PID 的质量控制算法取得了 8.6% 的视频质量提升，同时将质量波动也降低了 24.8%。当带宽在平均值 800kbps 附近波动的时候，四种控制算法对应的视频质量等级变化情况如图4.6所示。从中我们可以明显观察到 PID 控制算法得到的视频质量更加平稳。

^①<http://www.netlimiter.com/>

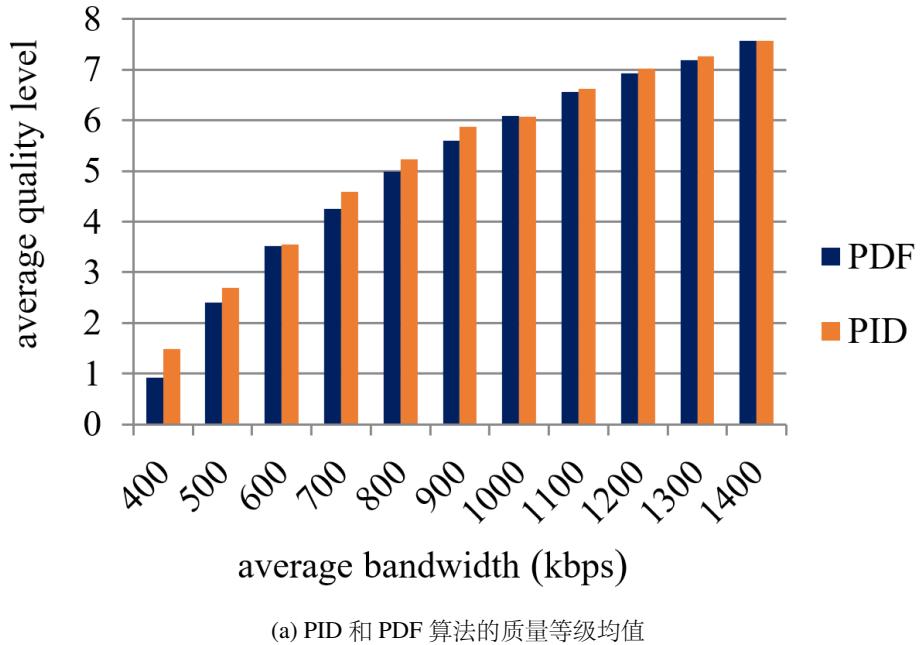


(a) 不同质量控制算法的质量等级均值

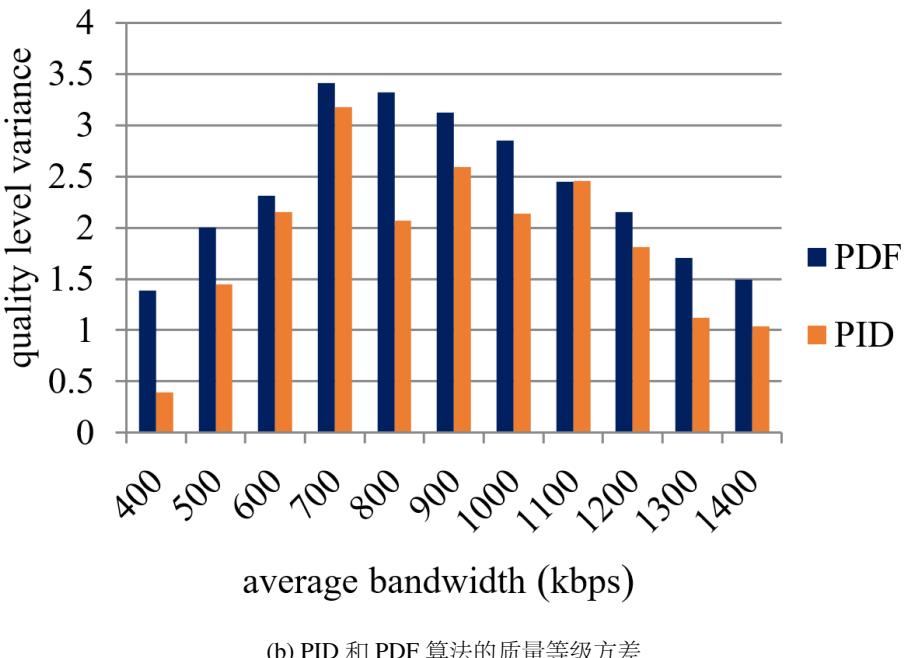


(b) 不同质量控制算法的质量等级方差

图 4.4 长/短期预测算法与达尔文服务器中的包延迟反馈 (PDF) 算法的比较



(a) PID 和 PDF 算法的质量等级均值



(b) PID 和 PDF 算法的质量等级方差

图 4.5 本文提出的 PID 控制算法与达尔文服务器中的包延迟反馈（PDF）算法的比较

表 4.2 不同质量控制算法与包延迟反馈算法的定量比较

| 控制算法 | 质量等级均值的增长 | 质量等级方差的增长 |
|--------------|-----------|-----------|
| 包延迟反馈 (PDF) | 0.0% | 0.0% |
| 短期预测 (STP) | +13.5% | +38.5% |
| 长期预测 (LTP) | +7.9% | -17.2% |
| PID 控制 (PID) | +8.6% | -24.8% |

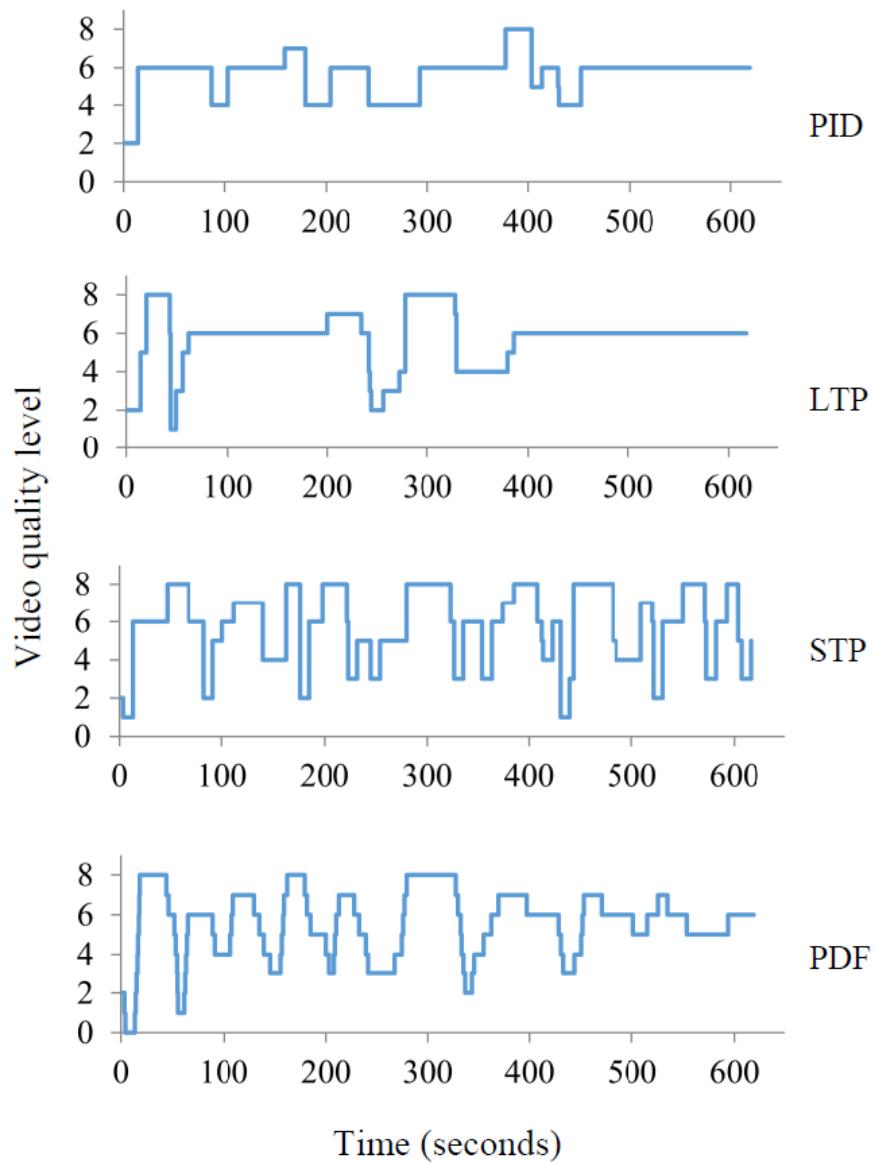


图 4.6 不同质量控制算法下视频质量等级随时间的变化情况

4.3.3 与码流截取相结合的效果

上面的实验直接用可伸缩视频中内置的伸缩层来构造质量等级，并没有结合本文前一章中提出的码流截取算法。实际的可伸缩视频流媒体系统中，当码率自适应算法给出一个适合传输的码率后，由码流截取方案根据这一码率限制截取出一个子流向客户端发送。本小节展示将本文提出的码流截取算法与码率自适应算法相结合的系统运行效果。

我们的比较对象是采用其内置的包延迟反馈算法和 JSVM 基本码流截取方案的达尔文流媒体服务器。比较结果如图4.7所示。其中展示了带宽在 400kbps、600kbps、800kbps 和 1000kbps 附近波动时，所发送视频流的 PSNR 变化情况。可以看到，采用本文算法的系统性能要好于比较对象，体现在以下两个方面：第一，成功避免了不必要的质量下降，例如图4.7c 的 350 秒处和图4.7d 的 150 秒处；第二，能够维持一个相对更高的 PSNR 值，例如图4.7a 的 450 秒附近和图4.7b 的 500 秒附近。

整个时间过程中的 PSNR 均值和方差在表4.3中列出，定量地证明了本文算法的优越性。例如，当带宽在 800kbps 附近波动时，采用本文算法的系统所传送视频的 PSNR 平均值为 38.75dB，比比较对象高了 0.83 个 dB，同时 PSNR 方差也由 1.24 降低到了 0.69，意味着视频质量平滑性也更好。

表 4.3 采用本文算法的系统与原始系统 PSNR 均值与方差的比较

| 带宽和比较项 | | 原始系统 | 本文系统 | 提升 |
|-----------------|----------|-------|-------|-------|
| 在 400kbps 附近波动 | PSNR 的均值 | 36.00 | 36.50 | 0.50 |
| | PSNR 的方差 | 0.30 | 0.23 | -0.07 |
| 在 600kbps 附近波动 | PSNR 的均值 | 37.06 | 37.32 | 0.26 |
| | PSNR 的方差 | 0.61 | 0.33 | -0.28 |
| 在 800kbps 附近波动 | PSNR 的均值 | 37.92 | 38.75 | 0.83 |
| | PSNR 的方差 | 1.24 | 0.69 | -0.55 |
| 在 1000kbps 附近波动 | PSNR 的均值 | 38.28 | 38.94 | 0.66 |
| | PSNR 的方差 | 1.30 | 0.71 | -0.59 |

4.4 本章小结

本章首先对 PID 控制器的基本思想做了简单介绍，然后将其用到了视频流媒体系统的码率自适应问题中。在选择了合适的过程变量与控制目标之后，对通用 PID 模型稍作修改，得到了更加直观有效的基于比例的模型，并据此给出了码率自适应算法（又称为质量控制算法）。所提出的算法相比于其他算法不仅提高了所发送视频的平均质量，还降低了质量波动，取得了更好的平滑性。本章提出的算法实现在了苹果公司

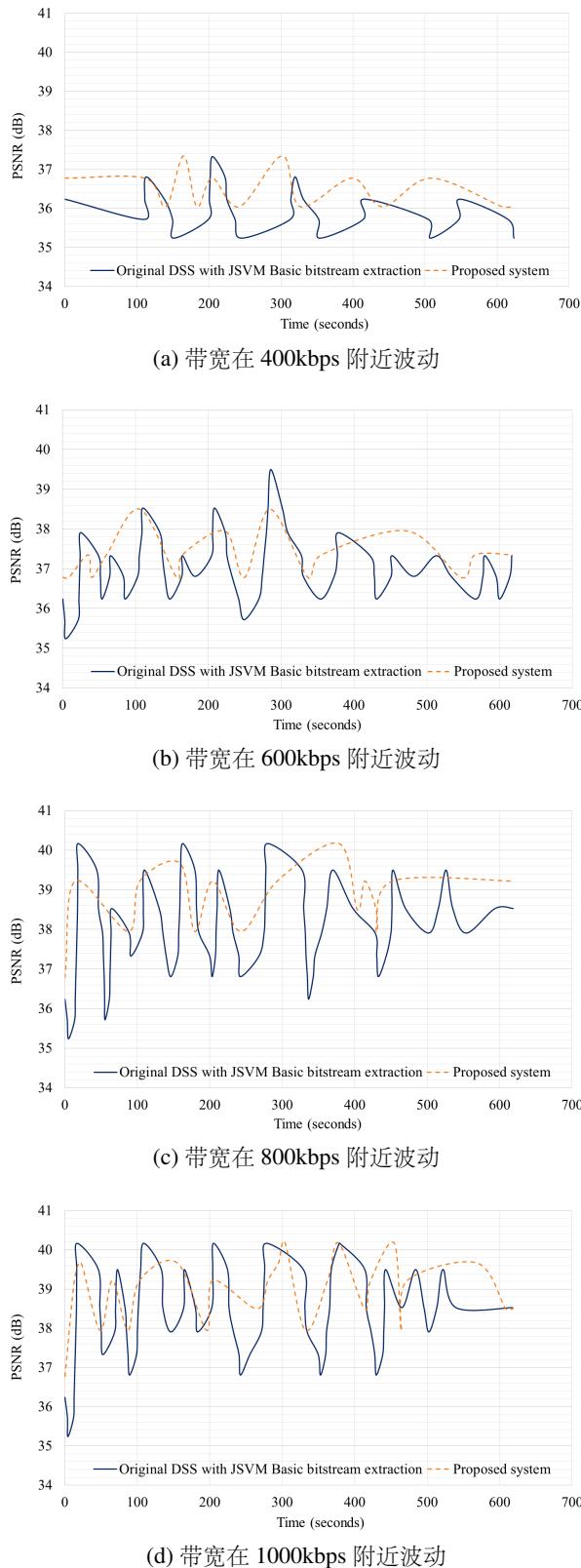


图 4.7 不同流媒体系统所发送视频的 PSNR 随时间的变化情况

QuickTime 流媒体服务器的开源版本（即达尔文流媒体服务器）上，在实际视频点播系统中表现出了良好的性能。

第五章 基于缓冲区分析的直播系统码率自适应算法

在上一章中，我们以达尔文流媒体服务器作为平台为视频点播系统的传输过程设计了一个码率自适应算法。本章针对如今越来越流行的用户生成内容（User Generated Content, UGC）直播应用进行研究，提出了一个基于传输中缓冲区分析的码率自适应算法，来改善从数据产生设备向服务器上传数据的效率。

5.1 视频直播系统概述

在有些视频直播系统中，视频内容由服务器或者与服务器通过局域网紧密连接的设备生成。在这种情形下，视频直播和视频点播系统有很大的相似性。这时视频数据的传输主要是从服务器到观看者客户端。直播服务器将动态生成的视频数据发送给客户端，和点播最主要的区别在于传输速率不能超过视频数据生成的速率，因为当所有生成的数据都传输之后，客户端需要等待下一个数据片段就绪^[81]。

但在 UGC 直播模式下，服务器本身不充当初始的内容产生方，而是在其他设备上持续生成的视频内容经过互联网（具有不确定的带宽和不可忽略的延迟）传输到中转服务器，然后再由服务器分发给观看者。目前这种模式的直播越来越流行，因为已经非常普及的智能手机成为了理想的内容产生方和观看接收方。图5.1展示了通用的手机视频直播系统的主要模块。集成了高效视频编码器的智能手机作为内容提供者，将实时视频数据传输到服务器。服务器所在的本地网络内设有转码器和提供播放服务的内容服务器。转码服务器会将上传的码流实时转码成多路具有不同码率的视频流，交由内容服务器供不同的观看者播放。播放客户端可以根据自己的网络环境选择最合适的码率，从服务器获取视频数据。

容易看出，在这样的直播系统中，依次有两个传输阶段。第一个是上传阶段，即内容产生方将录制的视频传给服务器；第二个是分发阶段，即视频数据发送给观看者。第二个阶段与点播系统类似，而且服务器端转码成多码流正是为了提供这一阶段传输的码率自适应性。本文主要关注的是第一个阶段，即上传阶段的码率自适应问题。

为了提供高质量的直播视频流，内容产生方理论上应该上传尽可能高码率的视频。然而高码率可能导致数据传输时间过长，造成接收端的播放中断。因此，在带宽变化的情况下，而内容产生方一方面需要尽可能利用网络吞吐量传输高质量的视频，另一方面还要保证接收端的流畅播放。与此同时，由于 UGC 直播应用大多具备交互功能，所以延迟需要控制在一个可以接受的范围。这就是我们在内容产生方设计的上传码率

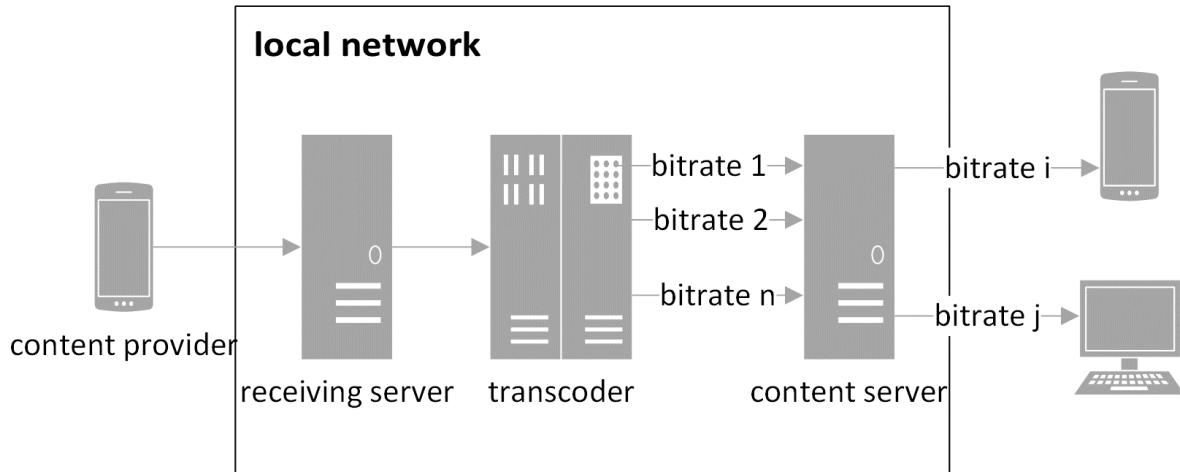


图 5.1 通用的手机直播系统模型

自适应模块所要达到的目标。

为简单起见，我们假设服务器具有快速的内部操作，同时接收端通过稳定高效地网络与服务器进行连接，这意味着服务器和接收端的视频流与内容产生方和服务器的视频流保持同步。这个假设对我们的研究重点没有影响。于是我们将传输架构简化成如图5.2所示。内容产生方的编码器将输入视频压缩之后传递给基于 TCP 的应用层协议。经过各层协议的封装，数据被传递到网络进行实际传输。在接收端经过一个逆向过程之后，数据输入到解码器进行解码，然后在播放器中进行渲染。与此同时，在发送方的自适应控制模块每隔特定的周期对传输信息进行检查，并采用码率自适应算法所给出的结果对编码器参数进行调整。

现在几乎所有的 UGC 直播应用采用的都是 RTMP 协议，而 RTMP 协议是基于 TCP 的，所以在我们的研究中，主要考虑的是传输层协议为 TCP 的情况。接下来首先详细分析直播中的传输过程，建立一个多缓冲区模型。然后根据缓冲区模型提供的系统状态信息来设计码率自适应算法。

5.2 直播中的多缓冲区模型

本节介绍所提出的多缓冲区模型。我们把直播中的传输过程视为几个不同的数据缓冲区的串联。这些缓冲区的状态可以被用来评估当前的网络状况，从而为码率自适应提供信息。

5.2.1 TCP 缓冲区

TCP 是一种网络传输层协议，属于互联网协议栈中的重要组成部分。在传输过程当中，TCP 协议采用了一种滑动窗口机制来保证可靠性，同时控制流量。为了实现滑

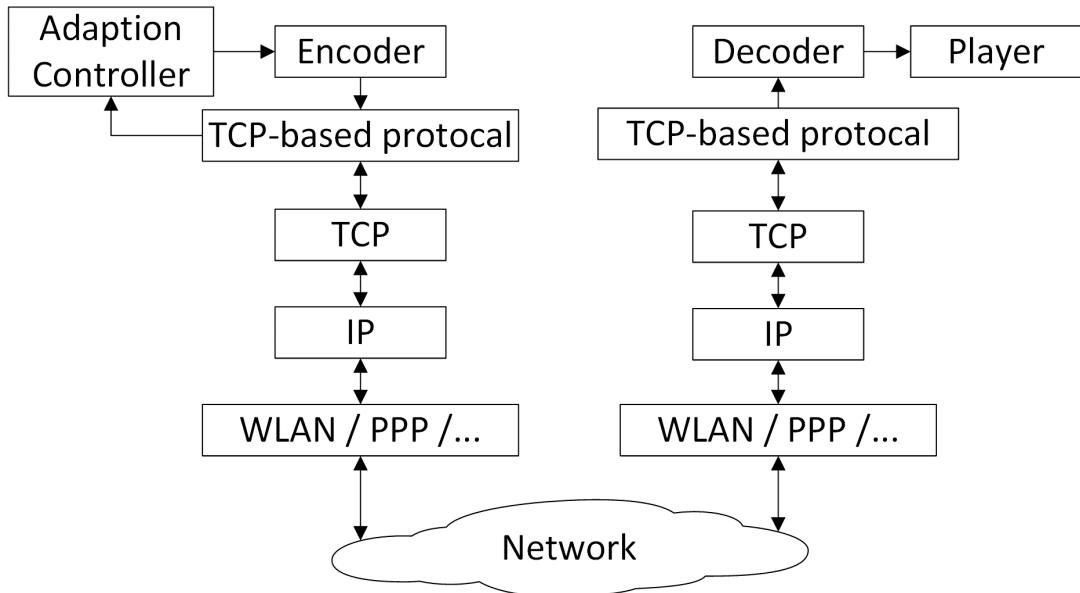


图 5.2 简化的直播系统传输架构

动窗口机制，存在一个 TCP 发送缓冲区（TCP Sending Buffer, TSB）来缓存将要发送的数据同时维护滑动窗口；对应的有一个 TCP 接收缓冲区（TCP Receiving Buffer, TRB）来缓存接收成功但未被上层协议获取的数据。TSB 中的数据收到到接收确认之后才会被移除，而 TRB 中的数据在被上层协议获取之后才会被移除。这些缓冲区一般是由协议内部实现的，应用程序很少干涉。

5.2.2 应用层发送缓冲区

在 TCP 协议的数据接收端，只有当 TRB 中存在足够空间的时候，数据才能被接收，这时传输才会真正发生。反之，在数据发送端，只有当 TSB 中存在足够空间的时候，其上的应用层协议才能够将数据放入 TSB。否则，上层协议的进程将会被阻塞，等待 TSB 中缓存的数据被移除。我们将应用层每次向 TSB 写入数据的单位定义为“数据段”(data segment)，以方便后续的讨论。

我们将第 i 次阻塞时间表示为 $\Delta t(i)$ ，将直播系统中视频的帧率表示成 f ，并假设每一个数据段内的视频帧数为 n 。如果对于所有的 i ，都满足 $\Delta t(i) \leq n/f$ ，那么这是较为理想的情况，因为不存在发送数据的积累。否则，在产生阻塞的时候需要采用应用层发送缓冲区（Application-layer Sending Buffer, ASB）来缓存持续产生的视频数据。当 TSB 中产生足够的空间的时候，ASB 队列头部的数据将会被移出进行发送。我们定义 ASB 的长度为其中数据段的数量，在大多数情况下我们期望它是一个较小的值。在通常情况下，存在 ASB 的最大长度。如果 ASB 达到了最大长度，后面到来的数据段将会被丢失。ASB 长度上限的设置是一种通用的限制延迟的方法。

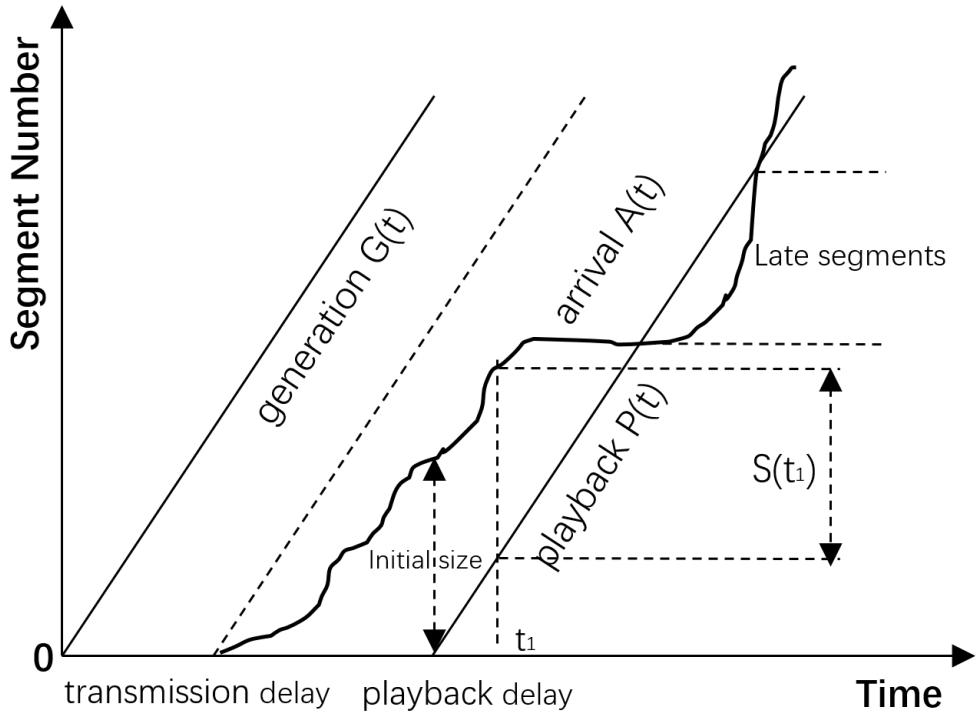


图 5.3 直播系统接收端数据关系

5.2.3 播放缓冲区

由于网络带宽和延迟的抖动，接收方收到数据并不是完全匀速的。在这种情况下，为了保持连续的播放，几乎所有播放器都会设置一个播放缓冲区（PB），它从 TRB 中获取数据，并按照固定的速率将数据解码并显示。当 PB 的长度为 0 时，视频停止播放直到 PB 的长度增长到一个固定值。我们称这个值为初始值，并表示为 S_0 。在很多系统中，播放缓冲区都非常有用，它弥补了短时间视频码率和网络带宽的不匹配，有效减少了视频中断的现象。通常情况下， S_0 就是 PB 的最大长度。当 PB 长度达到 S_0 之后，再收到的数据段在接收端将被丢弃。

图 5.3 展示了接收端的数据曲线。 $G(t)$ 表示在时间 t 之前在内容产生方生成的数据段的数量，因此 $G(t) = \mu t$ ，其中 μ 是一个常量，表示数据段生成的速率。 $A(t)$ 和 $P(t)$ 表示在接收端接收到和播放的数据段的数量，图中有 $S(t) = A(t) - P(t)$ 。第一个数据段在经过传输延迟 d_t 之后到达接收端，PB 开始接收到数据。我们假设在时间 t_0 ，第一次使得 $S(t_0) = S_0$ ，开始进行播放。 t_0 可以看做播放延迟，我们表示成 d_p 。因此，存在 $S(t) \leq \mu(d_p - d_t)$ 。当 $S(t)$ 减小到 0，播放终止，直到 $S(t)$ 重新增长到 S_0 。

5.2.4 多缓冲区模型分析

如上文所讨论的，在直播传输过程中有四个缓冲区：TCP 发送缓冲区（TSB）、TCP 接收缓冲区（TRB）、应用层发送缓冲区（ASB）、播放缓冲区（PB），他们组成了多缓冲系统。图5.4显示了多缓冲区模型的结构图，其中 $G(t)$ 、 $P(t)$ 以及 $A(t)$ 的定义与上一小节讨论的相同， $D(t)$ 表示在时间 t 之前进入 TSB 的数据段数量。我们使用 $S_b(t)$ 表示在时间 t 存在于缓冲区 b 的数据段的数量，则存在：

$$S_{ASB}(t) = G(t) - D(t), \quad (5.1)$$

$$S_{TSB}(t) = D(t) - A(t), \quad (5.2)$$

$$S_{TRB}(t) = A(t) - D(t) = 0, \quad (5.3)$$

$$S_{PB}(t) = A(t) - P(t), \quad (5.4)$$

$$S_{ASB}(t) + S_{TSB}(t) + S_{TRB}(t) + S_{PB}(t) = G(t) - P(t). \quad (5.5)$$

公式5.3表示 TRB 的大小为 0，这是因为在 RTMP 协议实现中 TRB 内一旦累积到一个完整的数据段，该数据段都会迅速被应用层获取。当接收端连续正常播放视频的时候，存在

$$G(t) - P(t) = \phi, \quad (5.6)$$

其中 ϕ 是一个常量，反映了播放延迟。将公式 (5.3)、(5.6) 代入公式 (5.5)，我们可以得到：

$$S_{ASB}(t) + S_{PB}(t) = \phi - S_{TSB}(t). \quad (5.7)$$

当 $S_{ASB}(t) = 0$ 时，由公式 (5.1) 和 (5.2) 可知， $S_{TSB}(t)$ 的长度由 $A(t)$ 决定（因为 $G(t)$ 以匀速增长），它反映了网络状况。当 $S_{ASB}(t) > 0$ 时，意味着在发送数据到 TSB 时发生了阻塞，TSB 一直处于充满状态， $S_{TSB}(t)$ 达到了上限，我们表示为 α 。在这种情况下，公式 (5.7) 成为：

$$S_{ASB}(t) + S_{PB}(t) = \phi - \alpha = \delta. \quad (5.8)$$

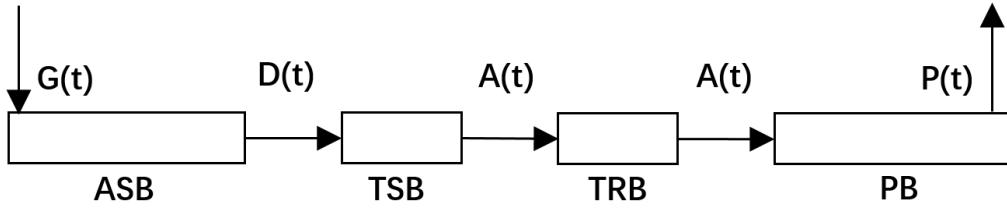


图 5.4 多缓冲区模型结构图

这意味着可以通过 ASB 的变化来控制和估计 PB 的变化。当 $0 < S_{PB}(t) \leq S_0$ 的时候，播放是连续的；这可以通过保持 $\delta - S_0 \leq S_{ASB}(t) < \delta$ 来实现，其中 $\delta \geq S_0$ 。为了实现最小的延迟， δ 期望被设置为 S_0 。

$S_{ASB}(t)$ 作为应用层的信息很容易获取，我们下面基于它在内容产生方设计一个码率自适应算法来实现高效的上传过程。

5.3 码率自适应算法

在这一节中，我们详细介绍所提出的直播系统中上传过程的码率自适应算法。具体来讲，我们希望自适应过程实现以下几个目标：（1）接收端能够连续播放；（2）传输和当前吞吐量匹配的最高视频质量；（3）尽可能低的延迟；（4）在快速波动的网络状况下灵敏地自适应；（5）方法具有通用性，能够容易地实现。

为实现目标（5），我们的方法建立在通用的基于 TCP 的模型上，并采用了方便获取的应用层信息。对于目标（3）和（4），我们考虑了最小可调度单元（Minimum Schedulable Data Unit, MSDU）的大小。对于目标（1）和（2），我们利用了 PID 控制思想的优势，通过选取系统变量和控制目标，能够动态地将系统保持在理想状态。

5.3.1 最小可调度单元

最小可调度单元（MSDU）指的是为系统能够完整传输或丢弃的最小的单位。在我们所讨论的系统中，MSDU 就是上面提到的数据段。数据段的大小可以在两个方面对系统产生影响：（1）传输延迟，数据段中的数据越多，第一个数据段完整到达接收端需要的时间就越长；（2）自适应的灵敏度，一次自适应调整只有在当前正进行的数据段传输过程结束之后才会起作用。因此我们期望数据段的大小能够最小。

在基于 DASH 的方案中，数据段通常是一个图像组（GOP）^[82]。一个 GOP 可以单独播放，当一个 GOP 被丢弃时，对其他的 GOP 没有影响。通常一个 GOP 的持续时间是 1 到 5 秒。而在我们的方法中，我们将数据段（也就是 MSDU）设置为视频帧，这

样可以将传输延迟最小化，以及将自适应的灵敏度最大化。如果在缓冲区极端情况下当一帧被丢弃时，我们需要丢弃其同一个 GOP 中后续的帧，这样才不会带来解码的错误。

5.3.2 过程变量选取

利用 PID 方法进行码率自适应的关键在于选取一个能够反映传输状况的过程变量。根据上面对多缓冲区模型的分析，基于 TCP 的直播系统在传输过程中对内容产生方最重要的信息是 ASB 内的数据长度，即 $S_{ASB}(t)$ （在时刻 t 的值），它能够直接反映视频码率和吞吐量之间的匹配情况。当视频码率高于当前的吞吐量时，该缓冲区内的数据长度趋向于增大，否则保持相对稳定或者减小为 0。

根据自适应的目标，首先需要确保流畅的播放，如上面所讨论过的，这可以通过保持 $\delta - S_0 \leq S_{ASB}(t) < \delta$ 来实现。为了实现最低延迟，我们将 δ 设定为 S_0 。因此，ASB 的大小应当被控制在 S_0 以内，否则将会出现播放中断。

与此同时，我们系统网络带宽被充分利用，这可以通过使 TSB 充满来实现。如上面所讨论过的，当 $S_{ASB}(t) > 0$ 时，TSB 为充满状态， $S_{TSB}(t)$ 达到了上限。

综合以上两点， $S_{ASB}(t)$ 的理想状态应为：

$$\varepsilon \leq S_{ASB}(t) \leq \varepsilon + \theta, \quad (5.9)$$

其中 ε 以及 θ 是很小的常量，并且 $\varepsilon + \theta < S_0$ 。

根据上述分析，我们定义系统的过程变量为 $S_{ASB}(t)$ ，它代表在时间 t 时 ASB 的大小，其目标值为 ε 。程序将周期性地自动检查 ASB，获取当前的过程变量值，并采用基于 PID 的自适应算法来调整视频码率。

然而，如果网络吞吐量在一个小的范围频繁波动，将使得该过程变量值不稳定而且相对随机，导致不必要的频繁调整。为了消除这样的随机性，我们将过程变量的定义修改为 $\overline{S_{ASB}(t)}$ ，它表示在时刻 t 之前的一个检查间隔时间段 τ 内 $S_{ASB}(t)$ 的平均值。它可以表示为：

$$\overline{S_{ASB}(t)} = \frac{1}{\tau} \int_0^\tau S_{ASB}(t) . \quad (5.10)$$

另一方面，为了减小不稳定性，我们对通用 PID 模型中的误差也进行量化，将其表示为：

$$e(t) = step \times \left\lfloor \frac{\varepsilon - \overline{S_{ASB}(t)}}{step} \right\rfloor, \quad (5.11)$$

其中 $step$ 是量化步长，并且 $step < \varepsilon$ 。

5.3.3 算法详情

为了计算 $\overline{S_{ASB}(t)}$, 我们会维护一个发送状态列表。每一个数据段(视频帧)从 ASB 发送出去的时候, 我们都会记录下此时的 ASB 长度, 放入列表中。当每一个检查间隔时间点到来时, 通过这个列表来计算 ASB 长度的平均值。

对于直播应用来讲, 由于发送的码流是实时生成的, 所以调整码率时直接改变编码器参数以改变生成的码率即可, 不需要涉及码流截取等数据源端的操作。但是, 质量等级的概念仍然需要, 它在这里变成了改变码率的单位。控制模型直接采用原始 PID 中的做法, 用过程变量 $\overline{S_{ASB}(t)}$ 和控制目标 ε 的差值来计算比例、积分、微分这几个部分。这个差值从物理意义上来看单位是数据段, 当用它计算得到控制器输出时, 该输出需要根据实际意义转换成码率的调整量。

具体的算法流程如 Algorithm3 所示。PID 中三个增益参数的选取和调优可以参见上一章中的相关内容, 这里不再详述。

Algorithm 3 基于 PID 的直播码率自适应算法

```

while is_started do
    wait for check period
    average_s = get_average_ASB_size()
    error = get_error(average_s, set_point)
    if error == 0 then
        last_error = error
        summation = 0
    else
        summation = summation + error
        difference = error -last_error
        output =  $K_p * \text{error} + K_i * \text{summation} + K_d * \text{difference}$ 
        change_bitrate(output)
    end if
end while

```

5.4 实验结果

5.4.1 实验配置

为了对基于 PID 的直播系统内容发送码率自适应方法进行测试, 我们实现了一个实际的手机视频直播系统。根据图5.1, 这个系统包含三个部分: 内容产生方、中转服务器、播放终端。

我们选择业界已经广泛使用的 RTMP 协议作为应用层流媒体协议, 具体实现采用

了第三方开源的代码库 `librtmp`^①。我们在 Android 平台上实现了内容产生和上传的程序，其中采用了开源软件 `x264`^②作为视频编码器，并集成了本文提出的码率自适应算法。我们用 `nginx-rtmp-module`^③帮助完成了中转服务器的搭建。播放终端采用了开源播放器 `ffplay`^④，它可以播放 RTMP 协议的流媒体并对播放缓冲区长度进行设置。

在我们的实验中，我们通过动态改变内容产生方所连接的路由器所允许的最大带宽来模拟出波动的网络状况。视频的采集帧率设置为 15 FPS；ASB 的最大长度为 150，PB 最大长度 S_0 为 60，过程变量 $\overline{S_{ASB}(t)}$ 的控制目标值 ϵ 为 15，误差量化步长 $step$ 为 5（这些值的单位是数据段的个数，每个数据段为一个视频帧）；改变码率的单位为 20kbps，初始码率为 500kbps；对过程变量的检查间隔为 2 秒。我们调节得到的 PID 控制参数 K_p 、 K_i 和 K_d 分别为 0.8、0.13 以及 0.07。

我们在 3 种条件下进行了测试：(1) 固定带宽 (Constant Bandwidth, CB)，带宽固定为 1Mbps；(2) 长周期波动带宽 (Long-Term Bandwidth Variation, LTBV)，带宽每隔 40 秒变化一次，在 200kbps 到 1.4Mbps 之间波动；(3) 短周期波动带宽 (Short-Term Bandwidth Variation, STBV)，在 200kbps 到 1.4Mbps 之间随机波动。

5.4.2 结果分析

测试结果参见图5.5、5.6和表5.1、5.2。由于移动设备上的 UGC 视频直播应用刚兴起不久，我们暂未发现有针对这一传输模式的研究工作可供对比。因此，在实验中我们只与不进行码率调整的系统进行了比较。

从图5.5a中可以看出，我们提出的自适应方法使得码率在 35 秒以内保持上升，直到趋近带宽。从图5.5b和5.5c可以看出，通过 PID 自适应算法的调整，在一个合理的延迟之内，码率和波动的带宽基本保持同步，而且系统对带宽的频繁变化进行了一定的平滑。如表5.1所示，码率不进行调整时带宽占用率较低，而我们提出的算法对于条件 (1)、(2) 和 (3) 分别将带宽占用率提高到 92.1%、88.9% 以及 87.1%。

从图5.6a可以看出，在带宽固定的情况下，对于进行自适应码率调整和未进行码率调整的视频流，播放基本都是连续的。但从图5.6b和5.6c可以看出，未进行码率调整的视频流的播放过程被频繁地中断，而进行码率调整的视频流的播放过程在大部分时间是连续的。如表5.2所示，我们所提出的方法对于条件 (2) 和 (3) 分别将播放时间占比提高到 97.3% 和 96.1%，因此显著提高了播放连续性。

总之，加入 PID 码率自适应算法比不进行码率调整的直播效果有所改善。

^①<http://rtmpdump.mplayerhq.hu/librtmp.3.html>

^②<http://www.videolan.org/developers/x264.html>

^③<https://github.com/arut/nginx-rtmp-module>

^④<https://ffmpeg.org/>

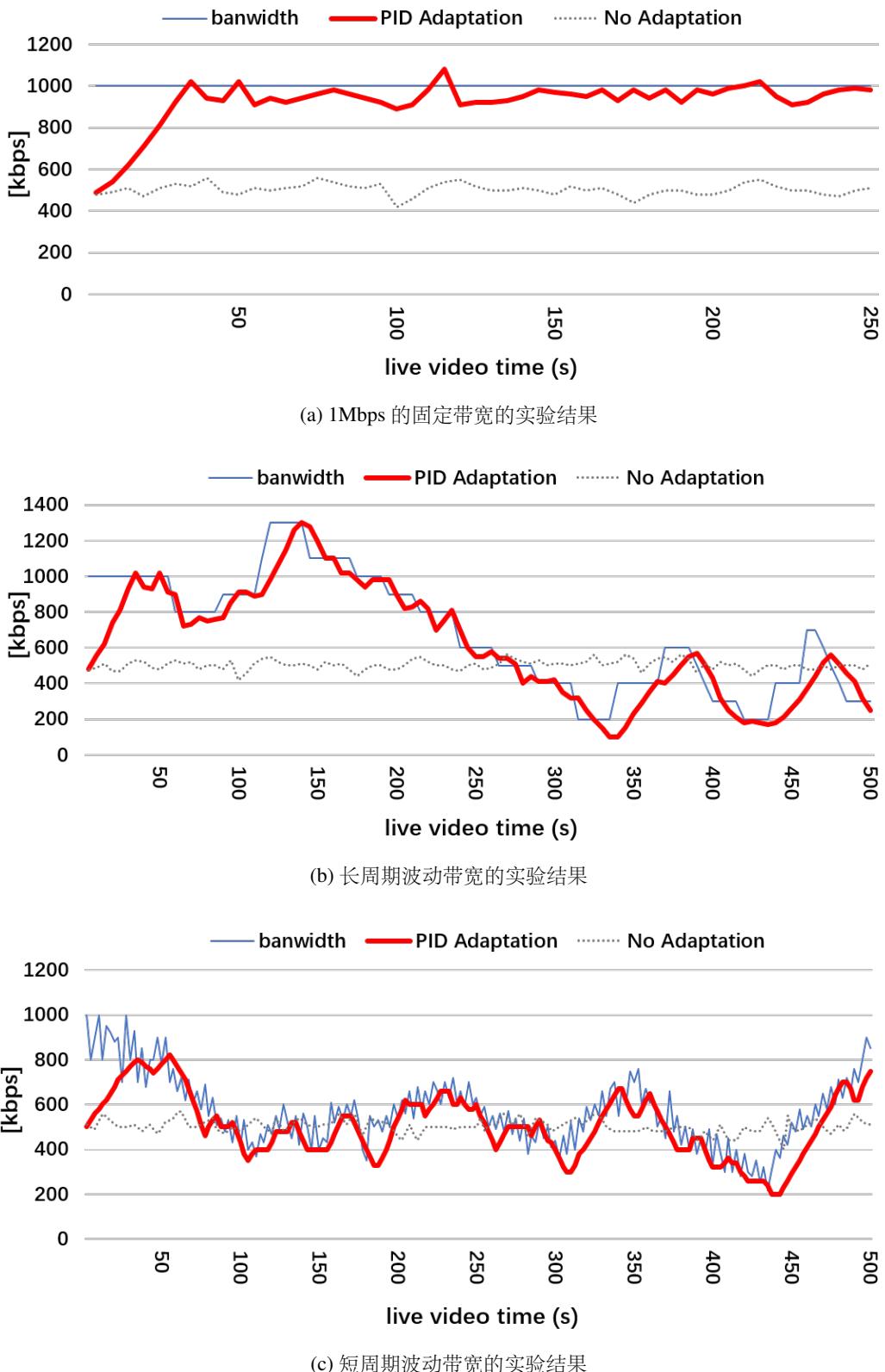


图 5.5 直播系统在不同测试条件下码率随时间变化的情况

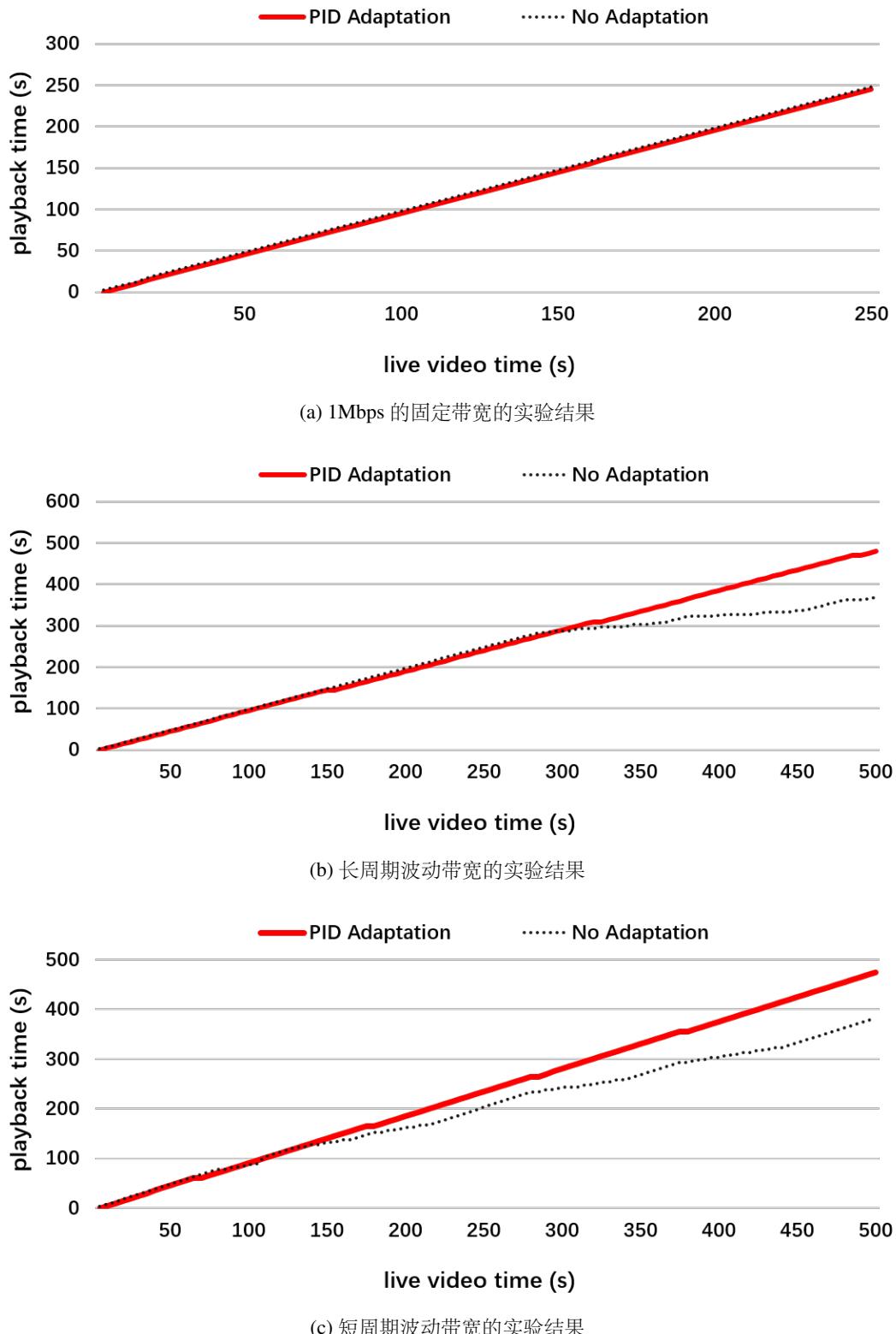


图 5.6 直播系统在不同测试条件下播放时间占比的变化情况

表 5.1 直播系统实验中的带宽利用率

| | CB | LTBV | STBV |
|---------|-------|-------|-------|
| 自适应调整码率 | 92.1% | 88.9% | 87.1% |
| 不进行调整 | 50.4% | 64.7% | 80.3% |

表 5.2 直播系统实验中的播放时间占比

| | CB | LTBV | STBV |
|---------|------|-------|-------|
| 自适应调整码率 | 100% | 97.3% | 96.1% |
| 不进行调整 | 100% | 73.4% | 71.5% |

5.5 本章小结

本章提出了一种针对视频直播系统中数据上传阶段的码率自适应算法。首先采用多缓冲模型来详细分析了整个传输过程，并根据缓冲区的状态来评估网络状况。在此基础上设计了一种基于 PID 的码率调整策略，能够灵敏地调节视频码率来匹配当前的网络带宽，从而在保证流畅播放的情况下提供最高能达到的视频质量。在我们实现的手机视频直播系统中的测试结果表明，所提出的算法提高了带宽的利用率，并改善了带宽波动情况下播放的流畅性。

第六章 总结和展望

6.1 本文工作总结

本文围绕自适应视频流媒体这一热点领域展开了较为系统和深入的研究。首先，本文针对可伸缩视频数据源提出了新的失真模型和码流截取方案，在支持可变码率的同时提供尽可能高的视频质量；其次，本文为基于可伸缩视频编码的视频点播系统设计了新的码率自动调整策略，用控制论的方法来解决如何适应带宽变化的问题；最后，本文详细分析了现在非常流行的视频直播系统的传输过程，结合直播的特点提出了数据上传时的码率自适应算法。总结来说，本文取得的主要工作成果包括以下三个部分。

1. 采用线性误差模型的可伸缩视频码流截取方案

作为码率适应带宽波动的前提条件，视频流媒体中的数据源需要能够灵活调整。可伸缩视频编码将数据划分为基本层和增强层，通过丢弃增强层的数据包来实现即时码率变化。从完整的可伸缩码流中丢弃部分数据得到一个子流的过程称为码流截取。本文以最小化特定截取码率限制下的视频失真为目标，首先提出了一个线性误差模型来估计丢弃任意数据包组合带来的失真变化，然后利用它设计了一个贪心型算法来根据每个数据包的码率和失真影响对其赋优先级，作为截取过程中丢包的顺序。实验表明，本文提出的线性误差模型对失真估计的误差率只有 5%，而采用该模型的码流截取相比于参考软件 JSVM 中的截取器有高达 0.5dB 的 PSNR 提升。

2. 基于 PID 控制思想的点播系统码率自适应算法

自适应流媒体的另一个关键问题是传输过程中的码率调整策略，即在可用带宽不断变化的情况下，决定何时调整码率并确定调整到多少。本文基于经典的比例-积分-微分（PID）控制思想，提出了一个综合考虑带宽的历史状况、当前状态和未来趋势的码率自适应算法，既能充分利用带宽，传输较高的视频质量，又能减小带宽波动的影响，保证视频质量的平滑性。该算法在达尔文流媒体服务器中实现后与原有系统相比，平均质量等级提高了 8.6%，而质量波动降低了 24.8%，即不仅发送视频的质量更高，还取得了更好的平滑性。

3. 基于缓冲区分析的直播系统码率自适应算法

在视频直播中，由于数据是实时产生的，其传输过程与视频点播有所不同。视频数据需要先上传到服务器，然后由服务器分发到观看者进行播放。本文为这个过程中的上传阶段增加了码率自适应的特性。首先通过详细分析系统整个传输过程

中各个缓冲区的关系，建立了一个多缓冲区模型；然后把上述点播系统中用到的 **PID** 方法与多缓冲区模型相结合，提出了一个有效的码率自适应算法。相比于没有自适应的上传过程，带宽的利用率得到了提升，视频播放的连续性也得到了改善。

本文的创新性算法和方案在实际的视频流媒体系统中得到了实现和一定的应用。

6.2 未来工作展望

以本文的成果作为基础，后续的研究可以扩展到其他类型的流媒体系统、立体视频和虚拟现实等更为复杂的应用场景，以及新一代视频编码标准。下面分三个方面对未来工作进行展望。

1. 更广泛的数据源优先级划分

本文的码流截取工作只是针对可伸缩视频编码的码流，但事实上，流媒体系统中的源端数据多种多样。从目前的趋势来看，视频正朝着立体和全景等更加生动也更为复杂的方向发展，这其中除了传统的图像数据，还有深度数据、空间几何数据等，它们对视频质量的贡献方式与大小都差异很大。为了高效自适应地传输这些数据，弄清楚各个数据包对最终用户观感质量的重要性意义重大。这就是广义上的优先级划分问题。根据不同数据的性质考察其对信息重建的影响，为其建立失真模型并确定传输时的优先级，将是未来研究的内容之一。

2. PID 控制思想的扩展应用

PID 作为能有效控制和调节各种系统行为的基本思想，具有非常广阔的应用潜力。在本文中，**PID** 用在了采用达尔文流媒体服务器的视频点播系统和用户生成内容的直播系统；在后续研究中，还可以将其应用到 **DASH** 等其他流媒体系统。此外，**PID** 的关键在于反馈，即通过从系统获取信息来反作用于系统的输入，这些信息除了读取系统状态外，还可以从客户端或者观看者收集。尤其是对于虚拟现实应用来说，用户与系统更为紧密，可以成为这一研究方向的用武之地。进一步地，以本文对 **PID** 的应用作为启发，在后续工作中可以继续探索如何将非线性控制、最优控制等现代控制论中的原理和方法用于视频和多媒体相关领域。

3. 基于新一代视频编码标准的研究

本文的研究主要是在上一代视频编码标准 **H.264/AVC** 及其可伸缩扩展 **SVC** 的基础上进行的。近两年间，最新一代的国际视频编码标准 **HEVC** 以及我国自主知识产权的新一代视频编码标准 **AVS2** 都已经发布，并且在不断的推广应用中。新的标准不仅进一步提高了压缩编码效率，而且仍然保持对可伸缩性的支持。下一步的研究可以基于新一代视频编码标准及其可伸缩扩展来进行，针对新标准的码流

截取、编解码优化、以及在视频流媒体系统中的集成应用等方面均可以作为未来工作的一部分。

目前移动直播和虚拟现实的热潮方兴未艾，视频流媒体作为其重要的技术支撑在很长一段时间内都将会有大量的需求和新的问题。服务器上的数据源处理、网络上的数据传输等本文所涉及的系统模块不会有大的改变，而且本文提出的思想方法很容易在新的应用场景或数据形式上适配。现有的这些工作和未来的研究将为人们突破时间和空间的限制更好感受这个世界带来更多可能。

参考文献

- [1] 李向阳, 卞德森. 流媒体及其应用技术 [J]. 现代电视技术. 2002-04:18–27
- [2] Y. Chen, B. Zhang, Y. Liu, W. Zhu. Measurement and Modeling of Video Watching Time in a Large-Scale Internet Video-on-Demand System[J]. IEEE Transactions on Multimedia. 2013, **15**(8):2087–2098
- [3] H. E. Egilmez, A. M. Tekalp. Distributed QoS architectures for multimedia streaming over software defined networks[J]. IEEE Transactions on Multimedia. 2014, **16**(6):1597–1609
- [4] S. Li et al. Flexible Traffic Engineering (F-TE): When OpenFlow Meets Multi-Protocol IP-Forwarding[J]. IEEE Communication Letters. 2014, **18**(10):1699–1702
- [5] N. Xue et al. Demonstration of OpenFlow-Controlled Network Orchestration for Adaptive SVC Video Manycast[J]. IEEE Transactions on Multimedia. 2015, **17**(9):1617–1629
- [6] I. Sodagar. The MPEG-DASH standard for multimedia streaming over the internet[J]. IEEE MultiMedia. 2011, **18**(4):62–67
- [7] H. Schwarz, D. Marpe, T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard[J]. IEEE Transactions on Circuits and Systems for Video Technology. Dec. 2007, **17**(9):103–1120
- [8] ITU-T. Advanced video coding for generic audio visual services[C], ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), ITU-T and ISO/IEC JTC 1, Version 1 , May 2003
- [9] N. Bouten, S. Latre, J. Famaey, W. V. Leekwijck, F. D. Turck. In-Network Quality Optimization for Adaptive Video Streaming Services[J]. IEEE Transactions on Multimedia. 2014, **16**(8):2281–2293
- [10] M. Wien, H. Schwarz, T. Oelbaum. Performance analysis of SVC[J]. IEEE Transactions on Circuits and Systems for Video Technology. 2007, **17**(9):1194–1203
- [11] S. Chuah, Z. Chen, Y. Tan. Energy-Efficient Resource Allocation and Scheduling for Multicast of Scalable Video Over Wireless Networks[J]. IEEE Transactions on Multimedia. 2012, **14**(4):1324–1336
- [12] Z. Zhu, S. Li, X. Chen. Design QoS-aware multi-path provisioning strategies for efficient cloud-assisted SVC video streaming to heterogeneous clients[J]. IEEE Transactions on Multimedia. 2013, **15**(4):758–768
- [13] G. Dan, O. Hadar, R. Ohayon, N. Amram. Live video streaming with adaptive pre-processing by using scalable video coding[C]. Proceedings of IEEE International Conference on Consumer Electronics. 2013, 588–589
- [14] E. Yang, Y. Ran, S. Chen, J. Yang. A multicast architecture of SVC streaming over OpenFlow networks[C]. Proceedings of IEEE Global Communications Conference. 2014, 1323–1328

- [15] S. Cicalo, V. Tralli. Distortion-Fair Cross-Layer Resource Allocation for Scalable Video Transmission in OFDMA Wireless Networks[J]. IEEE Transactions on Multimedia. 2014, **16**(3):848–863
- [16] 高文, 赵德斌, 马思伟. 数字视频编码技术原理 [M]. 北京: 科学出版社, 2010
- [17] C. E. Shannon. A mathematical theory of communication[J]. Bell System Technical Journal. Jul. 1948:379–423
- [18] T. Berger. Rate distortion theory[M]. Englewood Cliffs, NJ: Prentice Hall, 1984
- [19] K. R. Rao, P. Yip. Discrete Cosine Transform: algorithms, advantages, applications[C]. CA: Academic. 1990, 215–224
- [20] W. K. Pratt, J. Kane, H. C. Andrews. Hadamard transform image coding[J]. Proceedings of the IEEE. Jan. 1969, **57**(1):58–68
- [21] R. M. Gray, D. L. Neuhoff. Quantization[J]. IEEE Transactions on Information Theory. May 1979, **44**:2353–2383
- [22] D. A. Huffman. A method for the construction of minimum redundancy codes[C]. Proceedings of IRE. 1952, vol. 40, 1098–1101
- [23] J. Rissanen, G. G. Langdon. Arithmetic coding[J]. IBM Journal of Research and Development. 1979, **23**:149–162
- [24] ITU-T. Video codec for audio visual services at px64 kbit/s[C], ITU-T Recommendation H.261 , 1993
- [25] MPEG. Coding of moving pictures and associated audio - for digital storage media at up to about 1.5 Mbit/s - Part 2: Video[C], ISO/IEC 11172-2 (MPEG-1) , 1993
- [26] ITU-T. Information technology - generic coding of moving pictures and associated audio information: video[C], ITU-T Recommendation H.262 | ISO/IEC 13818-2 (MPEG-2) , 1995
- [27] ITU-T. Video coding for low bitrate communication[C], ITU-T Recommendation H.263 , 1998
- [28] G. J. Sullivan, J.-R. Ohm, W.-J. Han, T. Wiegand. Overview of the High Efficiency Video Coding (HEVC) standard[J]. IEEE Transactions on Circuits and Systems for Video Technology. Dec. 2012, **22**(12):1649–1668
- [29] F. Bossen, B. Bross, K. Suhring, D. Flynn. HEVC complexity and implementation analysis[J]. IEEE Transactions on Circuits and Systems for Video Technology. Dec. 2012, **22**(12):1685–1696
- [30] K. Veera, R. Ganguly, B. Zhou, N. Kamath, S. Chowdary, J. Du, I. S. Chong, M. Coban. A real-time ARM HEVC decoder implementation[C]. JCT-VC, document JCTVC-H0693 , Feb. 2012
- [31] M. Chavarrias, F. Pescador, M. Garrido, E. Juarez, M. Raulet. A DSP-Based HEVC decoder implementation using an actor language dataflow model[J]. IEEE Transactions on Consumer Electronics. Nov. 2013, **59**(4):839–847
- [32] Y. Duan, J. Sun, L. Yan, K. Chen, Zongming Guo. Novel Efficient HEVC Decoding Solution on General-purpose Processors[J]. IEEE Transactions on Multimedia. 2014, **16**(7):1915–1928

- [33] AVS Workgroup. Information technology - advanced coding of audio and video - Part 2: video[C], AVS1-P2 , 2005
- [34] T. Wiegand, G. J. Sullivan, G. Bjntegaard, A. Luthra. Overview of the H.264/AVC Video Coding Standard[J]. IEEE Transactions on Circuits and Systems for Video Technology. 2003, **13**(7):560–576
- [35] C.A. Segall, G.J. Sullivan. Spatial scalability within the H. 264/AVC scalable video coding extension[J]. IEEE Transactions on Circuits and Systems for Video Technology. 2007, **17**(9):1121–1135
- [36] Y.-K. Wang, M. M. Hannuksela, S. Pateux, A. Eleftheriadis, S.Wenger. System and transport interface of SVC[J]. IEEE Transactions on Circuits and Systems for Video Technology. 2007, **17**(9):1149–1163
- [37] Text of ISO/IEC 14496-4:2001/PDAM 19 Reference Software for SVC, Joint Video Team (JVT) of ISO-IEC MPEG & ITU-T VCEG, N9195[Z], Sep. 2007
- [38] K. A. Birney, T. R. Fischer. On the modeling of DCT and subband image data for compression[J]. IEEE Transactions on Image Processing. Feb. 1995, **4**(2):186–193
- [39] E. Y. Lam, J. W. Goodman. A mathematical analysis of the DCT coefficient distributions for images[J]. IEEE Transactions on Image Processing. Oct. 2000, **9**(10):1661–1666
- [40] K. Sharifi, A. L. Garcia. Estimation of shape parameter for generalized Gaussian distribution in subband decomposition of video[J]. IEEE Transactions on Circuits and Systems for Video Technology. Mar. 1995, **5**(1):52–56
- [41] N. Kamaci, Y. Altunbasak, R. M. Mersereau. Frame bit allocation for the H.264/AVC video coder via Cauchy-density-based rate and distortion models[J]. IEEE Transactions on Circuits and Systems for Video Technology. Aug. 2005, **15**(8):994–1006
- [42] Z. He, S. K. Ma. A unified rate-distortion analysis framework for transform coding[J]. IEEE Transactions on Circuits and Systems for Video Technology. Dec. 2001:970–982
- [43] G. J. Sullivan. Efficient scalar quantization of Exponential and Laplacian random variables[J]. IEEE Transactions on Information Theory. Sep. 1996, **42**(5):1365–1374
- [44] G. J. Sullivan, S. Sun. On dead-zone plus uniform threshold scalar quantization[C]. Proceedings of SPIE Visual Communication and Image Processing. 2005, 1041–1052
- [45] G. J. Sullivan, T. Wiegand. Rate-distortion optimization for video compression[J]. IEEE Signal Processing Magazine. Nov. 1998, **15**(6):74–90
- [46] L. Lin, A. Ortega. Bit-rate control using piecewise approximated rate-distortion characteristics[J]. IEEE Transactions on Circuits and Systems for Video Technology. Aug. 1998, **8**(4):446–459
- [47] J. Sun, W. Gao, D. Zhao, Q. Huang. Statistical model, analysis and approximation of rate-distortion function in MPEG-4 FGS videos[J]. IEEE Transactions on Circuits and Systems for Video Technology. Apr. 2006, **16**(4):535–539
- [48] B. Lee, M. Kim, T. Q. Nguyen. A frame-Level rate control scheme based on texture and nontexture rate models for High Efficiency Video Coding[J]. IEEE Transactions on Circuits and Systems for Video Technology. Mar. 2014, **24**(3):465–479

- [49] M. T. Sun, A. Reibman. Compressed video over networks[M]. Marcel Dekker, New York, 2001
- [50] D. Wu, Y. Hou, W. Zhu, Y. Q. Zhang, J. Peha. Streaming video over the Internet: approaches and directions[J]. IEEE Transactions on Circuits and Systems for Video Technology. Mar. 2001:282–300
- [51] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, Y. Reznik. Video coding for streaming media delivery on the Internet[J]. IEEE Transactions on Circuits and Systems for Video Technology. Mar. 2001:269–281
- [52] D. Wu, Y. W. Thomas and Y. Q. Zhang. Scalable video coding and transport over broadband wireless networks[J]. Proceedings of the IEEE. Mar. 2001, **89**(1):6–20
- [53] J. Ohm. Advances in scalable video coding[J]. Proceedings of IEEE. Jan. 2005, **93**(1):42–56
- [54] H. Schulzrinne, A. Rao, , R. Lanphier. Real Time Streaming Protocol (RTSP)[C]. IETF, RFC2326 , 1998
- [55] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. RTP: A transport protocol for real-time applications[C]. IETF, RFC3550 , 2003
- [56] S. Wenger, M. Hannuksela, T. Stockhammer, M. Westerlund, D. Singer. RTP payload format for H.264 video[C]. IETF, RFC3984 , 2005
- [57] Adobe Systems Inc. Real-Time Messaging Protocol (RTMP) Specification[C], 2009
- [58] R. Pantos, E.W. May. HTTP Live Streaming[C]. IETF, internet draft, work in progress , 2011
- [59] 黄拔峰, 钟明, 杨传钧, 张家钰. Darwin Streaming Server 的研究与应用 [J]. 计算机工程: 多媒体技术与应用. 2004, **30**(19):134–135
- [60] I. Amonou, N. Cammas, S. Kervadec, S. Pateux. Optimized Rate-distortion Extraction with Quality Layers in the Scalable Extension of H.264/AVC[J]. IEEE Transactions on Circuits and Systems for Video Technology. 2007, **17**(9):1186–1193
- [61] J. Sun, W. Gao, D. Zhao, W. Li. On Rate-distortion Modeling and Extraction of H.264/SVC Fine-Granular Scalable Video[J]. IEEE Transactions on Circuits and Systems for Video Technology. 2009, **19**(3):323–336
- [62] E. Maani, A. Katsaggelos. Optimized Bit Extraction Using Distortion Modeling in the Scalable Extension of H.264/AVC[J]. IEEE Transactions on Image Processing. 2009, **18**(9):2022–2029
- [63] P. Ramanathan, N. Jayant. RD optimized bitstream extraction for H. 264/SVC based video streaming[C]. Proceedings of IEEE International Conference on Image Processing. 2012, 2241–2244
- [64] K. Yang, S. Wan, Y. Gong, Y. Feng. A fast algorithm of bitstream extraction using distortion prediction based on simulated annealing[J]. Journal of Visual Communication and Image Representation. 2013, **24**(7):752–759
- [65] J. Lim, M. Kin, S. Hahm. An optimization-theoretic approach to optimal extraction of SVC Bitstreams, ITU-T VCEG JVT-U081[Z], Oct. 2006

- [66] T. Zhu, X. Zhang. Frame distortion estimation and its application to extraction priority assignment for MGS of H. 264/SVC[J]. Przegld Elektrotechniczny. 2011, **87**(1):284–289
- [67] K. Gao, J. Zhai, J. Li, C. Wang. Real-Time scheduling for scalable video coding streaming system[C]. Proceedings of IEEE Sarnoff Symposium. 2006, 1–4
- [68] T. Schierl, Y.S. De La Fuente, R. Globisch, C. Hellge, T. Wiegand. Priority-based media delivery using SVC with RTP and HTTP streaming[J]. Multimedia Tools And Applications. 2010, **55**(2):227–246
- [69] T. Stockhammer. Dynamic adaptive streaming over HTTP: standards and design principles[C]. Proceedings of the second annual ACM conference on Multimedia systems. 2011, 133–144
- [70] S. Akhshabi, S. Narayanaswamy, A.C. Begen, C. Dovrolis. An experimental evaluation of rate-adaptive video players over HTTP[J]. Signal Processing: Image Communication. 2012, **27**(4):271–287
- [71] 张辉帅, 王劲林, 朱小勇. 一种实时自适应 HTTP 流化码流切换算法 [J]. 计算机工程. 2013, **39**(2):1–5
- [72] T. Huang, R. Johari, N. McKeown, M. Trunnell, M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service[J]. ACM SIGCOMM Computer Communication Review. 2015, **44**(4):187–198
- [73] P. Juluri, V. Tamarapalli, D. Medhi. Look-ahead rate adaptation algorithm for DASH under varying network environments[C]. Proceedings of 11th International Conference on the Design of Reliable Communication Networks. 2015, 89–90
- [74] M. Winken, H. Schwarz, T. Wiegand. Joint Rate-distortion Optimization of Transform Coefficients for Spatial Scalable Video Coding Using SVC[C]. Proceedings of IEEE International Conference on Image Processing. 2008, 1220–1223
- [75] B. Li, K. Nahrstedt. A control-based middleware framework for Quality-of-Service adaptations[J]. IEEE Journal on Selected Areas in Communications. 1999, **17**(9):1632–1650
- [76] C.W. Wong, O.C. Au, H.K. Lam. PID-based real-time rate control[C]. Proceedings of IEEE International Conference on Multimedia and Expo. 2004, 221–224
- [77] L. Shen, Z. Liu, Z. Zhang, X. Shi. Frame-level bit allocation based on incremental PID algorithm and frame complexity estimation[J]. Journal of Visual Communication and Image Representation. 2009, **20**(1):28–34
- [78] J. Yang, Y. Sun, Y. Zhou, S. Sun. Incremental rate control for H.264 scalable video coding[C]. Proceedings of IEEE Global Telecommunications Conference. 2010, 1–5
- [79] Y. Zhou, Y. Sun, Z. Feng, S. Sun. PID-based bit allocation strategy for H. 264/AVC rate control[J]. IEEE Transactions on Circuits and Systems II: Express Briefs. 2011, **58**(3):184–188
- [80] J. B. Ziegler, N. B. Nichols. Optimum settings for automatic controllers[J]. ASME Transactions. 1942:759–768
- [81] T. C. Thang, H. Le, A. Pham, Y. M. Ro. An evaluation of bitrate adaptation methods for HTTP live streaming[J]. IEEE Journal on Selected Areas in Communications. 2014:693–705

- [82] L. De Cicco, S. Mascolo, V. Palmisano. Feedback control for adaptive live video streaming[C]. Proceedings of the second annual ACM conference on Multimedia systems. 2011, 145–156

致谢

这篇论文的完成，也意味着我五年的博士研究生之路即将走到终点。在这期间，我经历了很多，也收获了很多。我的工作成果和个人成长离不开许多人的帮助和支持，在此我想向他们表达谢意。

首先衷心感谢我的导师郭宗明老师。感谢郭老师这几年来对我无私的培养和关怀。郭老师在我学术研究方面高屋建瓴的指导，在我为人处事方面微言大义的教诲，都让我受益匪浅。郭老师生活中随和豁达、工作上认真严谨的风格，对科研和教育事业的热情，都深深地影响了我。郭老师百忙之中仍重视和支持研究室的团体活动和体育锻炼，让我们得以健康成长。郭老师不仅为我传道、授业、解惑，还给予我信任、期望和鼓励，在人生重要时期遇到这样一位导师真是我的幸运。

其次要特别感谢我的指导老师孙俊老师。感谢孙老师让我接受工程的训练，提高了我的动手能力；领我进入学术的大门，开启了我的科研道路。从选取研究题目，到探索新的方法，再到实验验证和论文写作，我的研究工作无不得益于和孙老师的交流讨论。我所取得的科研成果也都是在孙老师的指导和帮助下完成。孙老师精益求精的态度让我深受感染，严格细致的要求督促我不断进步，科研上的悉心指导，生活中的言传身教，这些对我而言都是一笔巨大的财富。

感谢数字视频研究室的刘家瑛、张行功、李晓龙等各位老师，我从他们那里得到过不少有用的建议和过来人的经验。感谢北京大学计算机科学技术研究所的彭宇新、肖建国、汤帜、赵东岩、陈晓鸥等各位老师，清华大学自动化系的季向阳老师，北京大学信息科学技术学院的马思伟、熊瑞勤两位老师，感谢他们在我博士不同阶段的指导和帮助。感谢计算机所综合办公室的戴永宁、张猛、韩玉晶、姚春霞等各位老师，感谢这些老师为我学业和科研的顺利进行提供保障。

感谢陈科吉、舒爽、范英明、张奇、王杰西、林镇安、钟鸣、李马丁、耿玉峰等数字视频研究室的同学们，以及曾在研究室待过如今已经离开这里的邓戬峰、段一舟、颜乐驹、刘森、周燕萍、任杰、周超、汤凯、白蔚、王一磊、张文尧等人，还有与我同班或同宿舍的刘丙双、姚金戈、李也等人。感谢他们给我带来工作上的协助和生活中的友谊。

感谢我的家人，尤其是我的父母和妻子。感谢他们一直以来的牵挂和支持，他们的爱是我力量的源泉。

最后，感谢各位评审老师拨冗评阅此论文并提供宝贵意见。

个人简历和在学期间研究成果

个人简历

1988 年 12 月 8 日出生于河南省郑州市。

2007 年 9 月考入清华大学自动化系，2011 年 7 月本科毕业并获得工学学士学位。

2011 年 9 月保送进入北京大学计算机科学技术研究所攻读理学博士学位至今。

发表或被接收的论文

- [1] **Shengbin Meng**, Jun Sun, Yizhou Duan, Zongming Guo, “Adaptive Video Streaming with Optimized Bitstream Extraction and PID-based Quality Control”, IEEE Transactions on Multimedia (TMM), vol. 18, no. 6, pp. 1124-1137, June, 2016.
- [2] **Shengbin Meng**, Jun Sun, Zongming Guo, “Software Solution for HEVC Encoding and Decoding”, Proceedings of International Conference on Multimedia Modeling (MMM), Sydney, Australia, January 5-7, 2015.
- [3] **Shengbin Meng**, Yizhou Duan, Jun Sun, Zongming Guo, “Highly Optimized Implementation of HEVC Decoder for General Processors”, Proceedings of IEEE International Workshop on Multimedia Signal Processing (MMSP), Jakarta, Indonesia, September 22-24, 2014.
- [4] **Shengbin Meng**, Jun Sun, Yilei Wang, Zongming Guo, “A PID-based Quality Control Algorithm for SVC Video Streaming”, Proceedings of IEEE International Conference on Communications (ICC), Sydney, Australia, June 10-14, 2014.
- [5] **Shengbin Meng**, Jun Sun, Yizhou Duan, Zongming Guo, “An Efficient Method For No-Reference H.264/SVC Bitstream Extraction”, Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Florence, Italy, May 4-9, 2014.
- [6] Jiexi Wang, **Shengbin Meng**, Jun Sun, Zongming Guo, “A General PID-based Rate Adaptation Approach for TCP-based Live Streaming Over Mobile Networks”, accepted by IEEE International Conference on Multimedia & Expo (ICME), Seattle, USA, July 11-15, 2016.

申请的专利

- [1] 孟胜彬, 孙俊, 段一舟, 郭宗明; “视频质量控制方法和装置”; 申请日期: 2014 年 08 月 29 日; 申请号: 201410428958.0。
- [2] 王杰西, 孟胜彬, 孙俊, 郭宗明; “基于 PID 控制的视频直播传输控制方法及系统”; 申请日期: 2016 年 03 月 03 日; 申请号: 201610121496.7。
- [3] 林镇安, 孟胜彬, 孙俊, 郭宗明; “流媒体音频同步方法及装置”; 申请日期: 2016 年 01 月 18 日; 申请号: 201610031208.9。
- [4] 刘森, 孟胜彬, 孙俊, 郭宗明; “视频质量控制方法”; 申请日期: 2015 年 05 月 08 日; 申请号: 201510231060.9。
- [5] 王杰西, 孟胜彬, 孙俊, 郭宗明; “基于 WebRTC 的交互式直播方法及装置”; 申请日期: 2015 年 05 月 07 日; 申请号: 201510230757.4。

参与的科研项目

- [1] 国家自然科学基金面上项目“高效可伸缩视频的编码、分析和调度研究”, 项目号: 61271020。
- [2] 国家科技支撑计划项目“智能视听服务平台研发与应用示范”, 项目号: 2014BAK10B02。
- [3] 国家高技术研究发展计划(863 计划)项目“真三维视频紧凑表示与高效压缩”, 项目号: 2015AA011605。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名： 日期： 年 月 日

学位论文使用授权说明

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校在一年/两年/三年以后在校园网上全文发布。

(保密论文在解密后遵守此规定)

论文作者签名： 导师签名： 日期： 年 月 日