

密级： 保密期限：

北京邮电大学

硕士学位论文



题目： 视频多跳传输网络中
纠删码的研究与应用

学 号： 2011140230

姓 名： 左佳

专 业： 计算机技术

导 师： 马华东

学 院： 计算机学院

2014 年 1 月 6 日

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

关于论文使用授权的说明

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在__年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

视频多跳传输网络中纠删码的研究与应用

摘 要

视频的实时可靠传输是多媒体传感网络的重要研究内容。数字喷泉码作为一种纠删码技术，因其无码率的特性，在数据传输领域有着重要的应用。本文的目的是研究无线多跳网络中对视频进行纠删编码的可行方法，并搭建系统实现。

本文详细分析了多媒体传感网络对视频传输的功能需求和性能需求；研究了常用数据喷泉码的编解码原理以及功能和性能优势，其中，重点研究了系统 Raptor 码的编解码过程和实现难点。本文设计了一套视频多跳传输系统，依据需求分析，提出了系统的功能模块以及工作流程，并给出了详细的实现过程，系统使用系统 Raptor 码，解决了数据包丢失的问题，最终实现了视频数据在多跳网络中的实时可靠传输。本文经过实验和测试，依据视频传输效果的对比和系统性能分析，证实了系统中的系统 Raptor 码有效避免了视频数据在多跳网络中传输时的丢包现象，提高了视频传输质量。

关键词：系统 Raptor 码 数字喷泉码 视频传输 纠删码 无线多跳网络

RESEARCH AND APPLICATION OF ERASURE CODE FOR VIDEO TRANSMISSION IN WIRELESS MULTI-HOP NETWORK

ABSTRACT

The real-time and reliable transmission of video is an important problem in multimedia sensor network. Digital fountain codes, as a kind of erasure code, because of its rateless features, has been becoming an important applications in data transmission. The purpose of this thesis is to study the method of erase code for video transmission in wireless multi-hop network and to implement it.

We analyze the functional and performance requirements of video transmission in multimedia sensor network. We also introduce the principle of digital fountain codes and its functional and performance advantages. Particularly, we focus on the coding process and implementation difficulties for the systematic raptor code. In this thesis, we design a video transmission system in wireless multi-hop network. According to the requirements analysis, we put forward the design of function module and workflow. Additionally, we present the detailed implementation process. The system uses the systematic Raptor code as the solution to solve the problem of packet loss in wireless multi-hop network. We finally realize the real-time and reliable transmission of the video data on multiple hops network. Through the experiment and test, as well as the contrast of video transmission result and the analysis of system performance, we verify that the systematic raptor code effectively avoids the packet loss in video data transmission in the multiple hops network and improves the quality of the video transmission.

KEY WORDS: systematic raptor code, digital fountain code, video transmission, erasure code, wireless multi-hop network

目录

第一章	绪论	1
1.1	研究背景	1
1.2	研究的意义和目标	2
1.3	纠删码的研究现状	2
1.4	论文主要内容及结构	3
第二章	视频多跳传输系统的概述	5
2.1	视频多跳传输系统的需求分析	5
2.2	系统框架	6
2.3	搭建系统开发环境	6
2.3.1	S3C6410 嵌入式开发平台	7
2.3.2	视频发送程序的开发环境	8
2.3.3	视频接收程序的开发环境	9
2.4	系统的总体设计	10
2.4.1	发送端总体设计	11
2.4.2	接收端总体设计	12
2.5	本章小结	13
第三章	数字喷泉码的研究	15
3.1	数字喷泉码与纠删码	15
3.2	数字喷泉码的基本原理	15
3.2.1	LT 码	16
3.2.2	Raptor 码	17
3.3	系统 Raptor 码的编码原理	19
3.3.1	编码过程中的相关概念和参数	19
3.3.2	编码参数的确定	22
3.3.3	数据处理	22
3.3.4	生成中间符号的过程	24
3.3.5	生成编码符号的过程	28
3.4	系统 Raptor 码的解码原理	29
3.4.1	解码出中间符号的过程	30
3.4.2	解码出源符号的过程	31
3.5	本章小结	31
第四章	视频多跳传输系统的设计与实现	32
4.1	发送端概述	32
4.2	数据采集模块的实现	33
4.3	数据压缩模块的实现	33
4.3.1	H.264 视频压缩格式	33
4.3.2	数据压缩的过程	34
4.4	数据编码模块的实现	35

4.4.1	数据分割.....	35
4.4.3	数据结构和函数.....	36
4.4.4	编码过程.....	39
4.4.5	编码符号的传输.....	41
4.5	接收端概述.....	41
4.6	数据解码模块的实现.....	42
4.6.1	数据结构和函数.....	42
4.6.2	解码过程.....	44
4.7	数据显示模块的实现.....	45
4.8	修复符号个数的确定方法.....	45
4.8.1	解码开销的确定.....	46
4.8.2	冗余符号个数 R 的确定.....	47
4.9	本章小结.....	47
第五章	系统测试及性能分析.....	48
5.1	系统功能模块运行结果.....	48
5.1.2	发送端的运行结果.....	48
5.2	Raptor 码的应用效果.....	49
5.2.1	视频质量的对比.....	50
5.2.2	系统的性能.....	52
第六章	总结与展望.....	54
6.1	工作总结.....	54
6.2	展望及改进建议.....	54
参考文献	56
致谢	58

第一章 绪论

1.1 研究背景

随着现代科学技术不断进步,计算机和网络的普及,人类社会信息传播的方式出现了惊人的变化。网络在各个领域的应用扩展了信息传播的范围,也提高了信息传播的速度,而无线网络技术的出现和快速发展更是使信息的传递进一步打破了时间和地点的限制。近年来,无线自组织网络的出现为信息传输提供了更加灵活和简便的方式,使得信息的传播得以延伸至各个角落。无线自组织网络一大特点是组网方式简单,它由多个具有路由功能的节点组成,每一个节点都能够自发的寻找路由、建立路由,即使有节点的位置发生了变化,或者部分节点无法正常工作,使得网络拓扑发生了变化,节点将自动的重新建立路由,恢复网络通信。无线自组织网络的节点通常比较小巧,布置较为容易,因此十分适合应用于自然环境恶劣的并且其它网络通信方式无法覆盖的地区^[1-2]。

随着社会的不断发展,人类在观察自然、探索自然、探索世界等领域的需求逐渐扩大。各种类型传感器作为一种观察、观测自然的工具,在很多行业中都有广泛的应用,通过这些传感器,人类可以获得反映自然环境的客观现状的各种数据。以水源监测为例,要客观评价水源的情况,我们需要运用相应的传感器设备获得水中不同元素含量、微生物、酸碱度等数据,通过对这些数据进行处理和分析,就能对水源的状态进行客观的评价。

网络技术和传感器技术的不断发展和日益成熟,加上现实应用的需要,随即出现了以微型传感器为主的传感器网络^[3]。传感器网络以无线自组织网络环境为基础,通过节点之间的合作,能够协调地感知、采集和处理覆盖区域内的各种环境或检测对象信息,并将这些信息及时地传输、发布给需要的用户,它广泛应用于军事、工农业控制、生物医疗、环境检测等诸多领域^[4]。

以微型传感器为主的传统传感器网络获取的主要是简单的监测数据,但是,随着人们对数据信息的要求越来越全面,简单数据已经不能满足实际需要,以对世界遗产的状态进行监测为例,显然视频图像数据能够更加直观的体现遗迹的状态。基于这种需求,各种信息量更加丰富的数据形式,例如图像、音频、视频等多媒体数据被引入到传统的传感器网络中,从而出现了多媒体传感网^[5]。多媒体传感网的应用使得人类能够获得更加全面、直观、精细、准确的数据。

1.2 研究的意义和目标

通过多媒体传感网可以更准确的监控采集现场的情况,极大地丰富了监控数据的内容,这使得其有着很大的应用前景和应用价值,因此对多媒体传感网络的研究引起了很多科研人员的关注,国内外的相关学者都十分重视这方面的研究,建立了专门的研究小组并启动了相应的研究计划^[4]。

多媒体传感网络的研究一直面临很多的挑战,其中,数据的可靠传输是一项重要的研究内容。多媒体传感网络应该具有更强的媒体传输能力,但是多媒体数据,尤其是视频数据的自身特征以及无线多跳网络的特点导致了视频传输很难保证高可靠性,然而媒体信息,尤其是音频、视频,不但对传输的实时性、同步性有很高的要求,保证良好的传输质量更加重要。目前,多媒体传感网络的带宽资源和数据处理能力还十分有限,在这种情况下,解决多媒体数据的实时可靠传输问题,是多媒体传感网络实用化的关键之一^[4]。

多媒体传感网技术在国内很多重大项目中都有应用。例如,南水北调中线工程水质传输网构建与物联网平台集成技术与示范项目以及世界文化遗产地风险预控关键技术与示范项目。项目运用多媒体传感网技术克服了自然环境恶劣,节点安置困难等问题,同时,项目对实时视频传输的质量都提出了很高的要求。因此,对提高多媒体传感网络中的实时视频传输质量方法进行研究是具有较高的实用意义。

本文的研究思路是,首先对应用场景的特点进行了分析,并以此为依据,针对无线多跳网络中视频可靠传输问题,通过研究现有的可行方法,最终提出具体的解决方案并实现。该解决方案要全面考虑实际的功能需求,并适应多媒体传感网带宽有限、数据处理能力较低的特点。

根据以上研究思路,在本文的研究过程中,设计并实现了一个无线多跳网络中的视频传输系统,并且在其中应用纠错码中的数字喷泉码技术,降低网络丢包率对视频传输质量的影响,提高视频传输的可靠性。本文最终实现了数字喷泉码在视频多跳传输网络中的应用,为多媒体传感网中视频实时传输提供了一种可靠性保障方案。

虽然本文研究内容针对的应用环境是视频多跳传输网络,但是文中提出并实现的系统的应用范围不只局限于视频传输。视频传输的要求十分苛刻,具有数据量大、实时性要求高等特点,但是普通的传感数据通常数据量小,对实时性的要求更宽松,可以考虑将本文中实现的方案应用于一般的无线多跳网络数据传输。

1.3 纠错码的研究现状

纠错码是一种前向纠错 (Forward Error Correcting, FEC) 技术, 它的原理是通过编码, 将 k 个原数据编码为 n 个的数据后再进行传输, 只要接收端接收到足够量的数据, 则运用适当的译码方法就可恢复 k 个源数据。通常采用的信道模型是删除信道^[6]。

常见的纠错码技术的码率是固定的, 无法适应任意的网络情况^[6-7]。

数字喷泉码是一种新兴的纠错码技术, 具备传统纠错码没有的性能优势, 在数据传输领域有着广泛的应用。

1998 年, Luby 等人首次提出了数字喷泉码技术, 用于分布数据, 但是当时并没有提出可实行的方案, 只是进行了概念和特征的介绍。2002 年, Luby 等人提出了 LT 码 (Luby Transform Codes), 这是第一个可实行的数字喷泉码设计方案。随后, A.Shokrollah 基于 LT 码提出了性能更加优越的 Raptor 码^[8-9]。LT 码和 Raptor 码都不是系统码, 为了使得 Raptor 码具有更好的实用性, A.Shokrollah 提出了系统 Raptor 码的编解码方案, 获得了更好的性能。

数字喷泉码具有无码率的特性, 并且编解码性能十分优越, 在网络通信传输领域有着广泛的应用可能。现阶段, 数字喷泉码在多源下载、移动广播、内容分发网络、数据传输等方面的研究受到了很多关注^[9]。

数字喷泉码的各种优势, 使得它受到了一些国际组织的关注, 成功被纳入很多相关标准之中, 例如, 3GPP 的文件传输标准中, 系统 Raptor 码被应用于 MBMS 服务^[10]。数字喷泉码良好的性能和可进行灵活的码率控制的特点, 十分适合在无线多跳网络上的进行应用。现阶段, 对它的研究还有面临着很多问题。例如, 虽然数字喷泉码有着无码率的特性, 因此可以无限进行编码直至解码端解码成功, 但是, 这对资源有着很高的要求, 然而在实际应用时资源通常是十分受限制的, 因此需要更加灵活的控制码率, 使它既能满足解码成功率的需求也能充分适应资源的限制。除此之外, 数字喷泉码的编解码性能的提高, 度分布的优化等研究领域也受到很多的关注。

1.4 论文主要内容及结构

本文主要包含五个章节的内容, 将对文的研究内容、研究过程和研究结果进行详细的介绍。论文的内容主要包括两个方面: 数字喷泉码的研究和文中实现系统的介绍。对于数字喷泉码, 本文将首先对研究过程中涉及到的各种编码方式进行概括, 介绍他们的特点和应用, 然后对系统中应用的编码方案做重点介绍。对于文中应用的实际系统, 本文将首先对系统的整体架构进行概述, 介绍系统的首先平台, 随后介绍各个模块的功能以及各模块之间的关系, 最后将对系统的实现过程做详细的阐述。

本文各章节的内容如下：

第一章绪论。绪论部分对研究背景进行了介绍，同时还指出了研究的应用背景，体现了实用性，并进一步讨论了研究的意义和最终的研究目标。绪论中还提及了文中将要应用到的关键技术，对它的研究和应用现状进行了简要的描述。

第二章系统概述。这一章将从整体的角度对系统进行描述。首先，指出了本文研究过程中需要考虑的一些问题，本文将围绕这些问题讨论系统的实现过程。接着介绍了系统的整体框架和开发环境。最后，给出了系统的总体设计思路，列出了各个模块的具体功能以及工作流程。

第三章数字喷泉码的研究。给出数字喷泉码的原理，详细分析和解释了系统 Raptor 码的编解码原理以及实现方案。

第四章视频多跳传输系统的设计与实现。按照数据传输的顺序介绍各个功能模块的详细实现过程。重点介绍系统 Raptor 的实现过程。

第五章系统测试。对各个模块的运行结果进行展示，重点对系统解决视频可靠传输问题的方案的效果进行展示，通过对比数据验证系统成果。

第六章总结与展望。对整个研究过程做出总结，提出系统优化的方向。

第二章 视频多跳传输系统的概述

2.1 视频多跳传输系统的需求分析

本文的研究的项目背景是多媒体传感网中的视频传输。多媒体传感网络是由大量部署在观测环境中的微型廉价低功耗的传感器节点通过多跳通信方式形成的网络系统^[4]，它的组网方式常常是无线多跳自组织。

多媒体传感网络具有如下特点^[1-4]：

(1) 硬件设备性能有限。传感器节点由于其体积、成本和功耗的限制，其内存空间和计算能力都比较弱，不能频繁进行大数据量的复杂计算。

(2) 无线信道传输带宽有限。无线信道本身的物理特性决定了移动自组织网络的带宽比有线信道要低很多，而竞争共享无线信道产生的碰撞、信号衰减、噪音干扰及信道干扰等因素使得无线网络的实际带宽远远小于理论值。

(3) 丢包率高。由于无线信道易受干扰、信道竞争、多跳导致丢包率叠加等原因，无线多跳自组织网络的丢包率较高。

多媒体传感网的硬件组成、组网方式和网络结构特性都十分不利于视频的传输。与其它传感器数据相比较，视频数据的特点使得其传输时对网络的性能要求比较高。同时，实际的应用需求也不能忽略。

在实际的应用时，对视频传输的性能有如下要求：

(1) 保证视频传输的可靠性。视频数据的可靠传输是多媒体传感网的重要研究方向，也是对视频传输的最基本要求，同时也是本文研究的最终目的。

(2) 保证传输的实时性能。实际应用中，多媒体传感网中的视频传输对实时性有较高的要求，因此，为解决视频可靠传输问题而采用的方法不能够破坏传输的实时性。

视频数据的特点是数据量大。视频数据的数据量比一般传感器获得的数据量大很多，因此，当网络中有多个视频数据流在传输时，对带宽有很高的要求。

综上所述，对于一个视频多跳传输系统提出以下需求：

(1) 减少传输数据量。选用恰当的方法对原始视频数据要进行压缩处理，以减小对无线带宽的压力。

(2) 选取运算复杂度低的抗丢包方案。复杂度低的算法能够满足节点低功耗的要求，同时，低复杂度的算法也意味着更快的运行速度，更小的时延，更符合系统对实时性的要求。

(3) 数据传输保证实时、快速。

(4) 流畅清晰的视频显示。体现了系统对视频传输质量要求。

本文的研究最终将实现一套视频多跳传输系统，使用纠删码技术降低网络中丢包率，提高视频数据的恢复概率，实现视频的可靠传输。

2.2 系统框架

本文系统网络环境总体框架如图 2-1 所示。

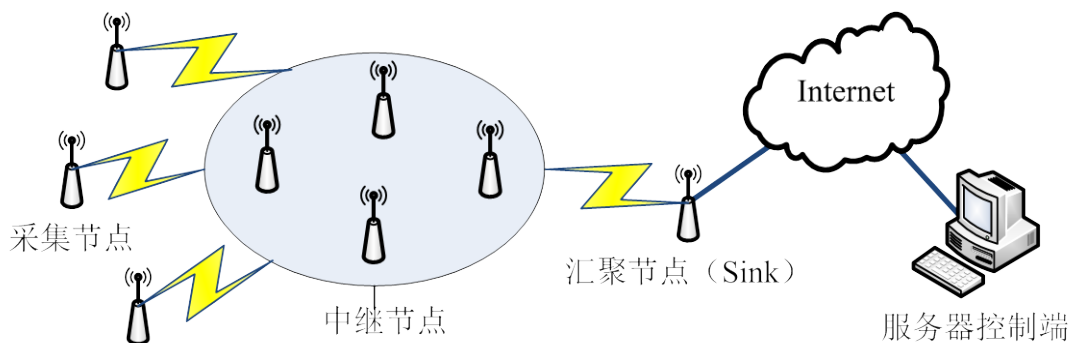


图 2-1 系统框架

可以看出，网络中共包含四个部分，分别是：采集节点、中继节点、汇聚节点（Sink），服务器控制端。

采集节点的主要功能是获取外界的数据并传输。在多媒体传感器网络中，采集节点上根据实际应用的需要会搭载不同类型的传感器，这些传感器获取各种数据交给采集节点，采集节点需要对这些数据进行相应的处理，例如将数据标准化、格式化，数据压缩等。采集节点具有寻路功能，数据处理完成后，它会将数据打包传输给最近的中继节点。

中继节点的主要功能是自动建立路由并转发数据包，中继节点之间的组网方式是无线自组织，具备灵活的路由拓扑。

汇聚节点（Sink）的主要功能是汇集最终收到的数据并转发，通过 Internet、3G 等其他网络形式传输给服务器控制端。

服务器端的主要工作是收集数据，并对数据进行必要的分析和处理，为用户提供交互的工具和界面。

本系统主要包括两个部分，一部分部署在采集节点上，称为发送端，包括对视频数据的采集和处理等功能；另一部分部署在服务器控制端，称为接收端，包括对视频数据的接收和展示等功能。

2.3 搭建系统开发环境

系统的开发工作主要有两个部分，一是运行于采集节点上的视频数据发送程

序，二是在服务器端运行的视频数据接收程序，因此需要两套开发环境。本节将对系统程序的运行和开发环境进行介绍。

采集节点采用的硬件基础是 S3C6410 嵌入式开发平台，在该开发平台上搭载 OV3640 数字摄像头作为视频传感器。S3C6410 嵌入式开发平台也可以搭载模拟摄像头进行视频画面的采集，模拟摄像头通常较容易调整画面角度，在实际应用时更为实用、灵活。无论平台上搭载哪种摄像头，系统的开发工作没有本质的区别。

在嵌入式平台上运行的程序是在 linux 系统下进行开发的。代码编写完成后，需要进行交叉编译，这样最终编译出来的应用文件才能在嵌入式平台上运行。

服务器端运行的视频数据接收程序的开发环境是 Windows 系统，采用的开发工具是 VS2010。接收端需要使用开源跨平台的视频、音频解码方案对 H264 格式的视频数据进行解码，得到 YUV 图像数据，为了将视频画面在 MFC（Microsoft Foundation Classes）的图片控件中进行展示，需要使用开源计算机视觉处理库 OpenCV 对 YUV 数据进行处理。

以上是系统搭建过程中使用到的开发平台、开发工具和开源函数库。

2.3.1 S3C6410 嵌入式开发平台

S3C6410 嵌入式开发平台在 2.2 节所描述的网络架构中作为数据采集节点和数据传输节点的硬件平台。

该平台由核心板和外设板组成。核心板上集成了 ARM11 处理器、128MDDR 内存和 1GB 的 NANDFLASH，另外还预留了 256K 的 NORFLASH。外设板上提供了多种外设接口，包括 RS-232 串口、USB 接口、10M/100M 自适应以太网接口、TFTLCD 接口、SD 卡接口等。同时，还配合提供了一系列必须的软件资源，包括引导程序、内核源码、文件系统和图形界面。按照平台的说明文档，用户可以自主定制内核，安装文件系统和图形界面，并在平台上进行自主的开发活动。

S3C6410 嵌入式平台可以搭载数字摄像头或者模拟摄像头，使用数字摄像头时，通过平台上的拨码开关进行设置即可。

S3C6410 嵌入式平台芯片中集成了 Multi_Format Codec 模块，简称 MFC。MFC 支持多种视频编解码格式，包括 MPEG4 SP（Simple Profile）的编解码、H.263 P3 的编解码、H.264 BL 的编解码等，并且，MFC 支持多数据流和多格式，也就是说，可以同时进行多个视频流以多种格式的编解码。在编解码前，开发人员可以通过调用相关的 API 编解码过程的属性进行设置，本系统中，只用到了编码相关的 API。

通过平台提供的串口，采用相应的软件工具就可在平台环境下执行各种操

作。本系统开发过程中私用了 Putty 软件。利用串口线连接平台与电脑，在电脑的设备管理器中查看该连接使用的串口号，然后对 Putty 进行相应的配置，如图 2-2 所示：

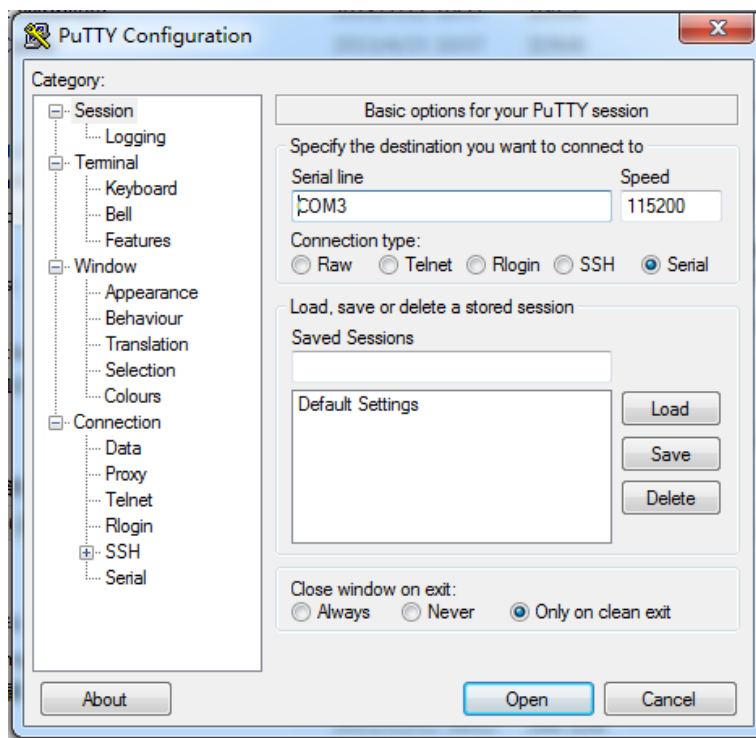


图 2-2 putty 软件配置

点击 open 按钮后，就会出现操作作用的窗口。由于平台提供的内核是 s3c_Linux 系列，因此对平台的操作使用的即是 Linux 的各种命令。此时，就可以在嵌入式平台中调试程序了。

2.3.2 视频发送程序的开发环境

发送端的开发方案是：在 PC 上进行代码的编写、交叉编译，接着将生成的程序通过 U 盘挂载的方式，拷贝到嵌入式开发平台中进行测试和运行。

首先，为了能偶同时进行采集节点程序和服务端程序的开发工作，需要在 PC 端安装虚拟机程序，并按照实际需要进行配置，本系统使用的是 VMware 软件创建虚拟机。接着在虚拟机中安装需要的系统，本系统选用 Ubuntu。

为了将代码编译为可以在嵌入式平台中运行的程序，需要安装交叉编译环境，安装步骤如下：

- (1) 下载交叉编译器压缩包 arm-linux-gcc-3.4.1.tar.bz2 。
- (2) 解压压缩包，解压命令如下：tar -jxvf arm-linux-gcc-3.4.1.tar.bz2，将解压后的/usr/local/arm 文件夹拷贝到系统/usr/local 文件夹下。
- (3) 设置环境变量，在/etc/profile 文件中加入如图 2-3 所示的一行内容：


```

28
29 PATH=/usr/local/arm/4.3.2/bin:$PATH

```

图 2-3 profile 文件内容

(4) 使得环境变量即时有效，命令如下：source /etc/profile

(5) 要检测交叉编译环境是否安装成功，需输入：arm-linux-gcc -v。出现如图 2-4 所示提示内容，则说明安装成功。

```

rita@rita-virtual-machine:~$ arm-linux-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/julian/lite-respin/linux/src/gcc-4.3/configure --build=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-linux-gnueabi --enable-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --with-gnu-as --with-gnu-ld --enable-languages=c,c++ --enable-shared --enable-symvers=gnu --enable-__cxa_atexit --with-pkgversion='Sourcery G++ Lite 2008q3-72' --with-bugurl=https://support.codesourcery.com/GNUToolchain/ --disable-nls --prefix=/opt/codesourcery --with-sysroot=/opt/codesourcery/arm-none-linux-gnueabi/libc --with-build-sysroot=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/libc --with-gmp=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-mpfr=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --disable-libgomp --enable-poison-system-directories --with-build-time-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin --with-build-time-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
rita@rita-virtual-machine:~$

```

图 2-4 交叉编译环境安装成功提示

交叉编译环境安装完成后，除了编译自己的工程，还可以很方便的将需要的开源函数库重新编译，移植到嵌入式平台下，如其他的视频编解码器、图像处理工具等等，这样就可以进一步扩展系统的功能。

2.3.3 视频接收程序的开发环境

接收端使用的开发工具是 VS2010，界面使用 MFC（Microsoft Foundation Classes）实现。

在接收端需要对接收到的视频帧进行 H.264 解码，因此需要在开发环境中配置 ffmpeg 开源音视频编解码库。

首先，将 ffmpeg 的链接库文件和头文件拷贝到工程文件夹下，在开发时将需要的头文件和库文件包含进来即可。由于 ffmpeg 的源码是 C，而接收端使用的是 C++ 语言，因此在包含头文件时，需要添加 extern "C" “{}”，如图 2-5 所示：

```

8 extern "C"
9 {
10 #include "libavformat/avformat.h"
11 #include "libswscale/swscale.h"
12 #include "libavcodec/avcodec.h"
13 #include "libavutil/mathematics.h"
14 };

```

图 2-5 在 C++ 工程中调用 C 头文件

H.264 解码结束所得到的是图像文件的 YUV 数据，将这些数据封装后才能在 MFC（Microsoft Foundation Classes）的控件中显示。因此，选用 OpenCV 对 YUV 数据进行处理。安装 OpenCV 的过程如下：

- (1) 下载 OpenCV_2.3.1 并解压。
- (2) 在接收端工程属性中添加 OpenCV 头文件路径，如图 2-6 所示：

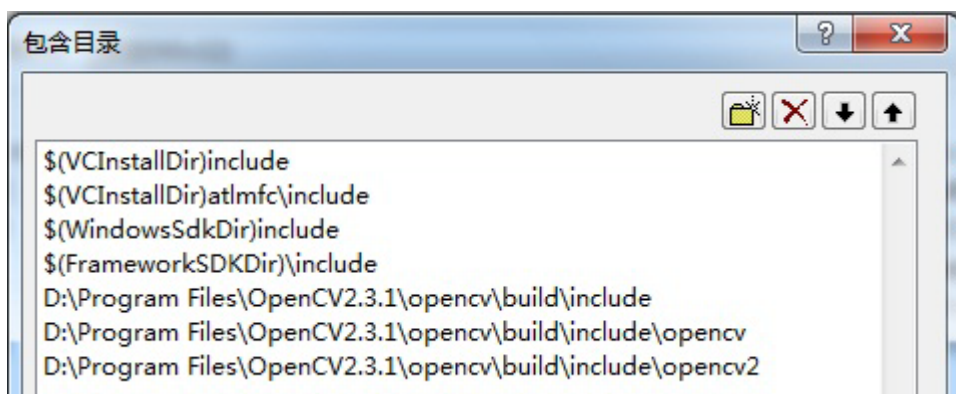


图 2-6 工程中配置 OpenCV 头文件

- (3) 在接收端工程属性中添加 OpenCV 动态链接库路径，如图 2-7 所示：

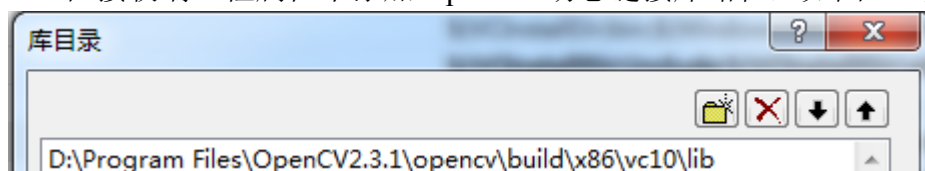


图 2-7 工程中配置 OpenCV 库文件

2.4 系统的总体设计

在本文 2.1 节中，对系统的应用环境进行了分析，提出了系统需要解决的主要问题，同时列出了系统设计过程中需要考虑的各种因素，根据以上内容，系统在发送端和接收端设计了若干功能模块，如图 2-8 所示。

本系统中，数据流方向是双向的，视频数据从发送端传输到接收端，而接收端将收集到的网络情况数据反馈给发送端，这些反馈数据将提供给数据编码模块参与编码过程中相应参数的确定。

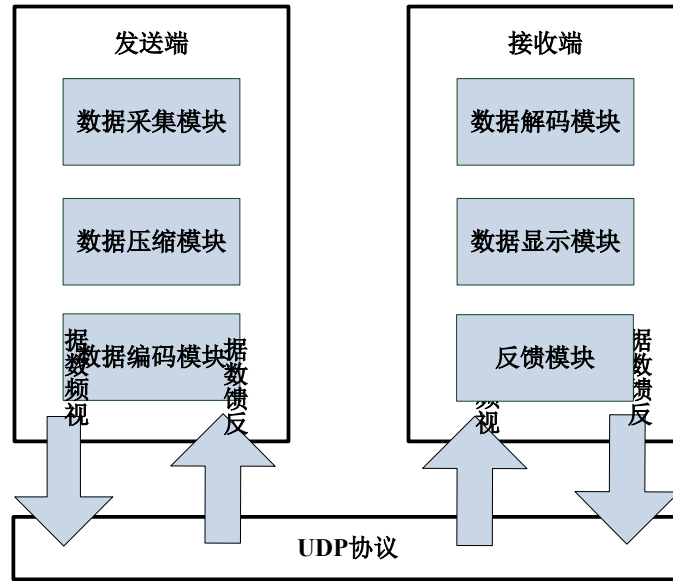


图 2-8 系统功能框架图

2.4.1 发送端总体设计

发送端包括视频数据采集、数据压缩和数据编码三个模块，如表 2-1 所示：

表 2-1 发送端功能模块列表

模块名称	模块主要功能和实现方法
数据采集模块	主要功能采集源数据。调用 S3C6410 的设备文件读取命令，通过读取 OV3640 摄像头的相应寄存器抓取视频数据。
数据压缩模块	调用 S3C6410 的 MFC (multi format codec) 模块的 API 进行 H.264 编码。通过编码获得视频的关键性数据，将关键性数据按一定频率加入视频帧序列中，以保证接收端正常解压缩。
数据编码模块	将视频数据分片后，读取反馈数据，根据反馈数据更新编码相关的参数，根据参数进行系统 Raptor 编码，最后将编码结果打包成编码包，使用 UDP 协议发送出去。

发送端的不断循环数据采集、压缩、编码、传输的过程，每循环一次，完成一帧视频图像的处理和传输工作。循环过程中不断查询是否收到反馈数据，根据反馈数据，修改数据编码过程中使用的参数。

系统发送端的工作流程如图 2-9 所示：

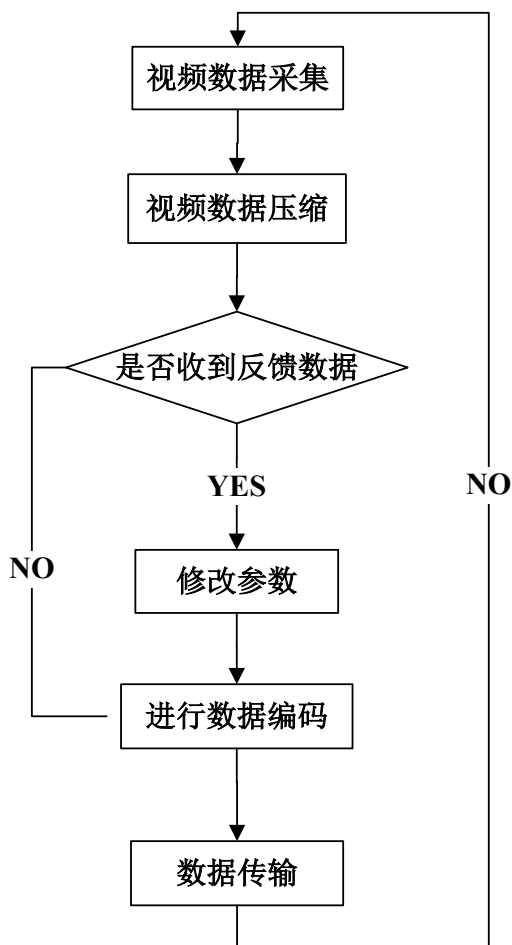


图 2-9 发送端工作流程

2.4.2 接收端总体设计

发送端相对应的，接收端包括数据解码和数据显示两个模块，如表 2-2 所示：

表 2-2 接收端功能模块列表

模块名称	模块主要功能和实现方法
数据解码模块	将接收到的数据包从新组合成视频帧，当发生丢包时，进行 Raptor 解码，恢复完整的视频数据。
数据显示模块	对视频数据进行 H.264 解码并存储进 OpenCV 的结构体中，最后利用 MFC (Microsoft Foundation Classes) 架构为用户提供良好的界面，将视频数据展示出来。
反馈模块 (AMQ)	统计编码包的丢失情况，计算丢包率，通过 UDP 传输回发送端。

系统接收端的工作流程如图 2-10 所示：

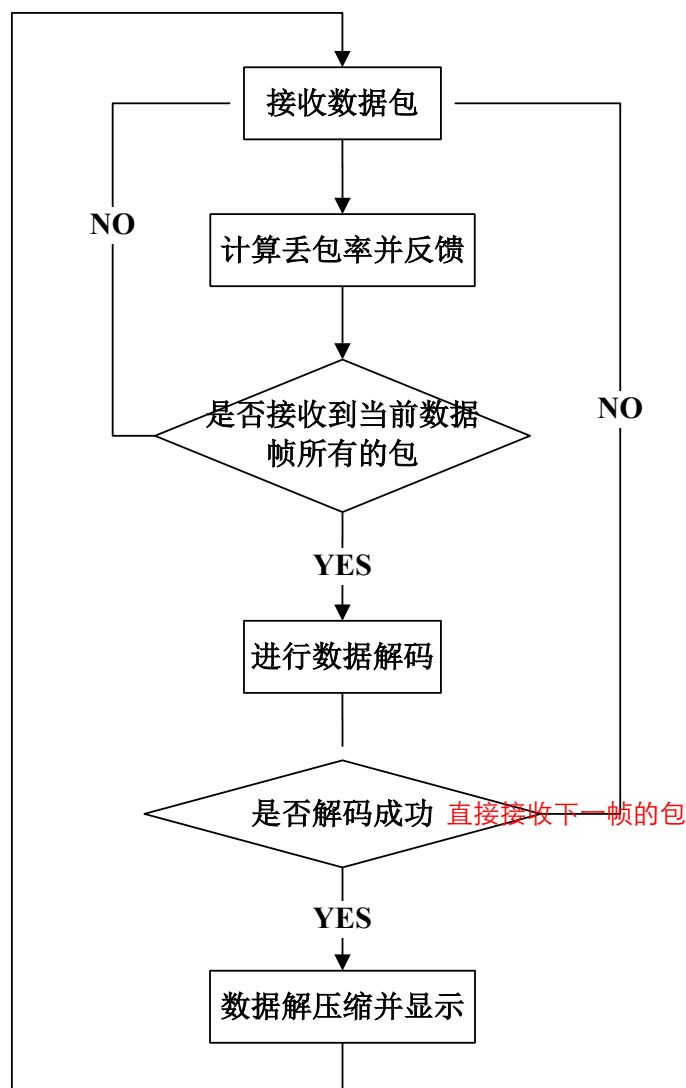


图 2-10 接收端工作流程

接收端不断循环数据接收、拼接、解码、解压缩并展示这一过程。当接收到某一帧视频画面所有相关的数据包后，开始对该视频帧进行解码，解码成功即显示后继续接收数据包，不成功则直接接受下一帧的数据。同时，还要统计网络的丢包情况，及时反馈给发送端。

2.5 本章小结

本章对系统的应用场景的特点进行了分析，并介绍了系统的应用框架。这些内容为系统的设计提供了依据。

根据本文 2.1 节的内容，系统设计的最终目的是实现视频在无线多跳网络中的实时可靠传输，数据编码模块和数据解码模块正是为了实现这项功能而设计。系统的编解码方案是系统 raptor 码，属于纠删码的一种，它具有良好的解码性

能、较低的编码运算复杂度，既有恢复数据包的功能，也符合系统对低功耗和实时性的要求。

进行数据传输时，使用了 UDP 协议，它是一种不可靠的传输协议，对报文的丢失没有相应的保证机制，但是，无线多跳网络中建立连接十分困难，而 UDP 协议是面向无连接的，它能够尽可能快的将报文放入网络，不用等待连接的建立，很好的保证了实时性。而 UDP 协议传输不可靠的问题通过数据编解码模块解决。

数据压缩模块通过信源编码，去除了视频数据中的冗余，有效降低需传输的数据量，减少了无线信道的带宽压力，同时也保证了编码模块的运行速度。

在所有的模块中，发送端的数据编码模块和接收端的数据解码模块是本文的介绍重点，在第三章和第四章中将重点介绍编解码的原理和实现过程。

第三章 数字喷泉码的研究

3.1 数字喷泉码与纠删码

数字喷泉码是纠删码的抽象概念。数字喷泉的特征与如下:喷泉源源不断的喷出水滴,这些水滴之间的关系都是对等并且无关的,当用杯子去接水时,接水的人只需关心杯子是否接满,而不必关心接到杯子里的水具体是哪一滴。类似的,数字喷泉码的基本原理是,编码器根据若干个源码数据,不断生成相互独立的编码数据,解码端只要接收到足够数量的编码数据,就可以恢复出完整的源码数据,不必关心具体接收到了哪个编码数据^[6]。

常见的纠删码,例如 RS 码、Tornado 码等,能够近似的实现数据喷泉,但是,它们的编码复杂度较高^[6-7],因此不能满足实时系统的应用要求。同时,传统纠删码的码率固定,所以它的纠删性能受到了限制,不能应用于网络环境多变的情况。

3.2 数字喷泉码的基本原理

数字喷泉码具有很多其它编码方式不具备的特点,使得它十分适合在多媒体传感网中应用。

理想的数字喷泉码具有以下特点^[6-9]:

(1) 根据有限个数的源码,编码器可以生成无限多个编码数据,也就是说,数字喷泉码具有无码率的性质。

(2) 对于被分割成 k 个源码数据的数据,只要解码端接收到编码数据中的 k 个,就能够恢复出全部的源码数据,且恢复过程很快。

在实际应用数字喷泉码时,可以适当放松要求。编码包的序列可以是有限的,并且接收端在恢复源码数据前接收到的编码数据数量 n 可以略大于 k ,恢复的速度可以稍慢,符合具体需要即可。

相对 RS 码、Tornado 码,数字喷泉码有明显的优势^{[8][11]}:

(1) 数字喷泉码是无码率的。在实际应用中,数字喷泉码的码率可以根据应用环境改变,具有更强的实用性。

(2) 在数据量较大的情况下,传统纠删码的编译码算法计算复杂,速度较慢,不适用与实时应用,而数字喷泉码的编译码算法计算简单,速度较快。

(3) 传统纠错码的译码成功率不是 100%，而只要条件适合，数字喷泉码的译码成功率是 100%。

相较 RS 码等纠错码，数字喷泉码具有更低的计算复杂度，更好的带宽利用率^[11]，因此在设备性能有限、信道状况不稳定、传输数据量大的多媒体传感网中，数字喷泉码十分适用。

下面将介绍两种数字喷泉码的实现方案，为方便表述，我们将编码时输入的数据分片成为输入符号，将编码完成后输出的数据分片称为输出符号。

3.2.1 LT 码

LT 码是数字喷泉码的第一个可行性方案，是之后各种实现方案的基础，它充分体现了数字喷泉码无码率的特征。

LT 码生成一个编码符号的流程如下^{[12][13]}：

- (1) 将需要传输的源数据分割成 k 个相同大小的输入符号。
- (2) 按照选定的度分布函数选择一个度 d 。
- (3) 在 k 个源码中选取 d 个进行异或运算，运算的结果就是一个输出符号。
- (4) 重复进行 (2) 和 (3) 步骤，即可生成任意多个输出符号。

传输输出符号时，需要将该输出符号与输入符号的关联信息包含在传输包中，这样解码才能有据可循。因此，为了使传输时需携带的关联信息尽量小，在生成一个输出符号时，需要给该输出符号分配一个关键字 ID，度分布函数根据这个关键字 ID 确定 d 的值以及于输出符号关联的输入符号。

通过以上描述可以看出，优秀的度分布函数对 LT 码十分重要。度分布函数必须保证两点：

- (1) 输入符号的信息完整的包含于任意固定数量的输出符号。
- (2) 输入符号信息在输出符号中的分布必须均匀。

这样解码的成功与否才能完全取决于接收到的输出符号的个数。

LT 码的解码流程如下^{[12][13]}：

图中输入符号未被恢复前是空心，恢复后为实心。

(1) 规定两个符号集：输入符号集和输出符号集。解码开始前，输入符号集中的符号个数是 0 图中输入符号未被恢复前为白色，恢复后为蓝色，输出符号集中的符号个数是 N 。

(2) 从接收到的输出符号中找出一个度值为 1 的，该输出符号只和一个输入符号有关联关系，将它从输出符号集中移出，加入输入符号集。如果找不到度为 1 的输出符号，解码过程结束。此时，如果输入符号没有被完全恢复，则说明解码失败。

- (3) 将该符号与所有与它相关联的输出符号进行异或。
- (4) 删除所有与该符号的关联关系。
- (5) 重复进行步骤 (2) 到 (4)。

根据以上流程，给出 LT 的一个解码示例如图 3-1 所示，图中输入符号为白色时表示未被恢复，为蓝色时表示已恢复，输入符号与输出符号之间的连线代表它们的关联关系：

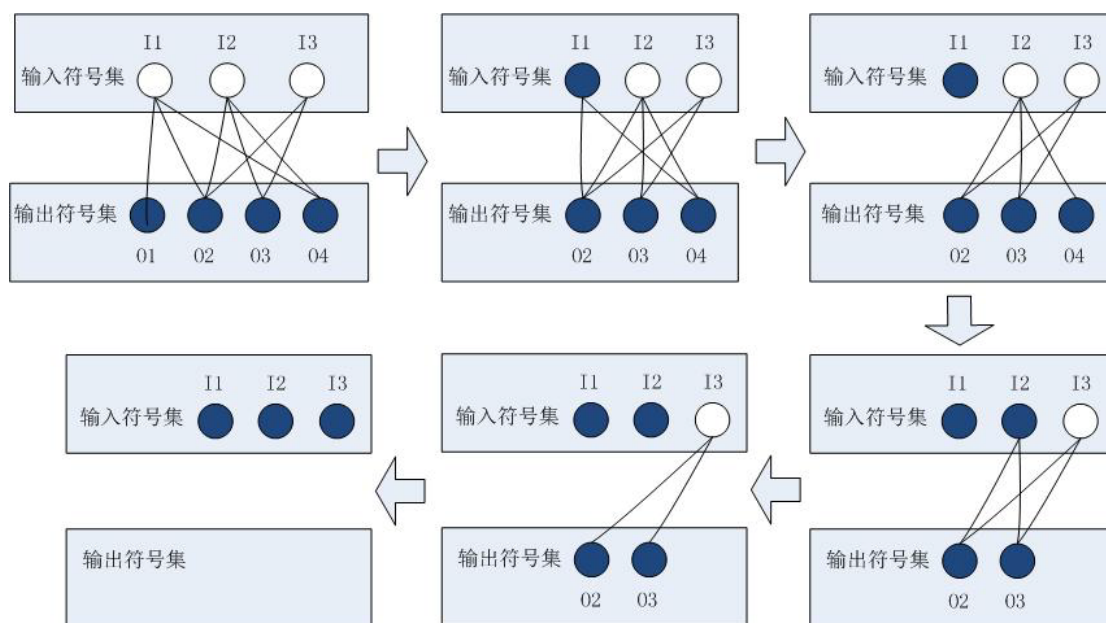


图 3-1 LT 的解码流程示例

从上述解码的过程可以看出，度为 1 输出符号十分重要，每次重复 (2) 到 (4) 步骤时，都需要首先找到它，因此，度函数的选取直接影响了 LT 码的性能。LT 码目前应用较多的度分布函数是 Robust Solution 度分布，该度分布能够在解码的迭代过程中保证较高的找到度为 1 的输出符号的概率，从而保证较高解码成功率。LT 码编解码 k 个输入符号平均需要进行 $k \ln k$ 次异或计算，所以 LT 码不具备线性的解码复杂度。

3.2.2 Raptor 码

普通的 Raptor 码是在 LT 码的基础上提出的数字喷泉码实现方案，它具有更好的编解码性能。Raptor 码是一种级联码，它的原理是，在进行 LT 编码之前，首先进行线性预编码，在输入符号中引入一定量的冗余信息生成中间符号，从而降低 LT 码解码时的复杂度。它的编码过程^[14]如图 3-2 所示：

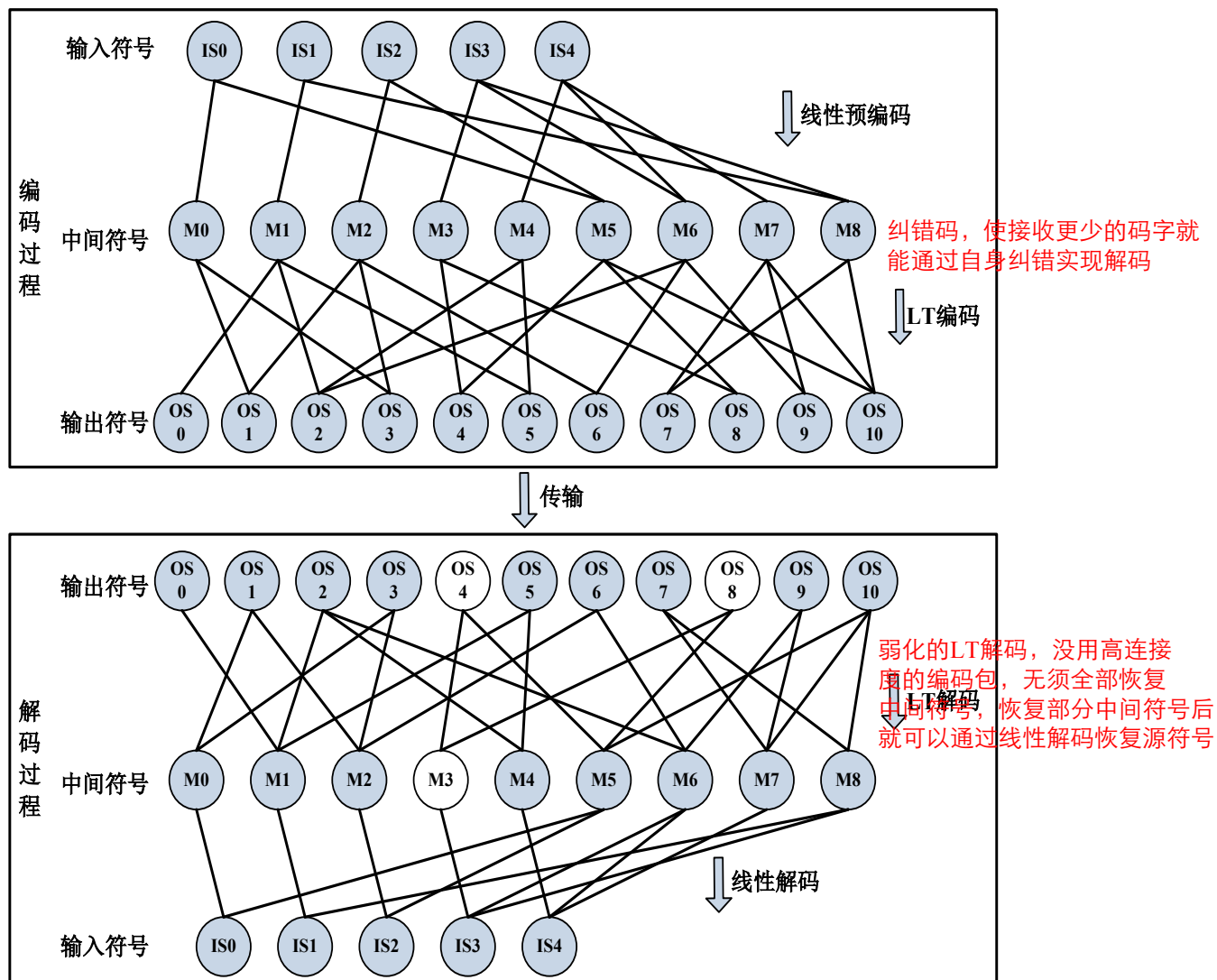


图 3-2 Raptor 的编解码流程

首先对输入符号进行线性预编码, 编码方式可能 Hamming 码、LDPC 码等等, 增加了适当的冗余符号, 获得了中间符号, 接下来对中间符号进行 LT 编码, 获得最终的输出符号进行传输。解码的过程是编码过程的逆过程, 先进行 LT 解码, 得到中间符号, 再通过线性解码, 恢复所有的输入符号。

输出符号传输到解码器时, 一部分符号可能丢失。如图 3-2 所示, OS4 和 OS8 丢失或未被接收的情况下, 由于与 OS4 和 OS8 相关联的多数中间符号被其它输出符号覆盖, 可以被恢复, 而 M3 则无法被恢复, 此时, 由于与 M3 关联的 IS3 被其它中间符号覆盖, 因此所有输入符号都能够恢复。

综上所述, 由于中间符号中包含了冗余符号, 所以解码端在进行 LT 解码时不必完全恢复中间符号, 即使有少部分中间符号没有恢复出来, 也可以通过线性解码完全恢复所有输入符号, 因此, LT 解码过程的复杂度明显下降, 而线性解码的复杂度很小, 整体上, 解码过程的算法复杂度降低了。

3.3 系统 Raptor 码的编码原理

上文提到的 LT 码和普通的 Raptor 码都不是系统码。系统码是指将输入符号作为输出符号的一部分传输给解码端的一种编码方式。由于输入符号是输出符号的组成部分，在编码时，这部分输出符号就不需要进行编码，解码时，由于接收到的一部分内容本身就是输入符号，无需进行解码，大大简化了编解码的过程。因此，在工程应用是，通常更希望使用系统码提高工程的效率。

A. Shokrollahi 在普通 Raptor 码的基础上提出了一种系统 Raptor 码的实现方案，该方案兼具系统码和普通 Raptor 码的优势，具有很高的实用价值。

系统 Raptor 码方案已经应用于视频数据的广播多播业务，已经证实了它对于大数据量传输的优秀性能。在本系统中，要将系统 Raptor 码应用于无线多跳自组织网络上的实时视频传输。与已有应用不同的是，网络状况不稳定同时还对实时性有较高要求。

本节的各小节将对系统 Raptor 码的编解码原理和方法进行详细的说明。

3.3.1 编码过程中的相关概念和参数

我们将需要进行编码的文件数据称之为源数据，编码过程要求源数据将被分割为更小的数据块，以数据块为单位进行编码活动，数据块又会被分割为更小的数据符号，以数据符号为单位进行运算操作。

为了表述方便，表 3-1 中列出了编码过程中涉及到的概念：

表 3-1 编码过程相关概念

概念名称	解释
源块 (Source block)	通过对源数据进行分割而来。每个源块作为一个整体参与 Raptor 编码过程，在编码时将会进一步分割为 K 个源符号。
符号 (Symbol)	符号是编码过程中的一个数据单元，单位是字节，编码中的所有运算都是在符号间进行的，因此，它是最小的操作单位。编码过程中涉及到不同种类的符号，包括源符号、中间符号、修复符号和编码符号。
源符号 (Source symbol)	通过对源块进行分割而来，是在编码过程中使用到的最初数据单位。同一个源块中的源符号大小一致。
编码符号 (Encoding symbol)	编码结束时得到的输出，也即用来传输的符号，由 <u>源符号和修复符号两部分组成</u> ，同一源块中的编码符号大小相同。

(续上表)

概念名称	解释
中间符号 (Intermediate symbols)	按照与源符号的 LT 编码约束关系以及自身符号间的预编码关系生成的符号，中间符号被用于直接生成编码符号。 <u>中间符号只是产生编码符号过程中的中间产物，不会被传输至解码端。</u>
修复符号 (Repair symbol)	修复符号是编码符号的一部分，由源符号与矩阵运算产生。生成一个修复符号的源符号来自同一个源块，修复符号的大小与生成它的源符号的大小相等。
编码符号组 (Encoding symbol group)	被放到同一个传输包内传输的一组编码符号。在传输包内，各个编码符号之间使用编码符号 ID 进行区分，通过编码符号 ID 也可以求出该编码符号与源符号之间的关联关系。
子块 (Sub-block)	一个源块有时会被继续分割为子块，每个子块都足够小，使得解码过程可以在规定大小的内存里进行。
子符号 (Sub-symbol)	由子块分割而成。每个源块包含 K 个源符号，每个子块包含 K 个子符号。从一个源块的所有子块中都抽出一个子符号，就可以组成一个源符号。每个源符号中包含的子符号个数与每个源块中的子块数相同。
源包 (Source packet)	包含源符号的传输数据包。
修复包 (Repair packet)	包含修复符号的传输数据包。
编码包 (Encoding package)	含有编码符号的传输数据包。

下面对一些概念的缩写进行说明：

表 3-2 缩写说明

缩写	含义
ESI	<u>全称 Encoding Symbol ID，编码符号 ID</u> ，用来区分同一编码包中的所以编码符号。
LDPC	全称 Low Density Parity Check，低密度奇偶校验码， <u>一类具有系数校验矩阵的线性分组码</u> ，具有良好的性能和较低的编解码复杂度。
SBN	全称 Source Block Number，用于区分源块的整数值。

以下是编码过程涉及的一些参数和基本函数的描述：

表 3-3 参数和基本函数说明

参数或函数名称	描述
A1	用来使符号大小对齐的参数，符号和子符号的大小都是 A1 的整数倍。
K	表示一个源块中的源符号个数
S	表示与源符号个数 K 对应的 LDPC 符号个数。
H	表示与源符号个数 K 对应的 Half 符号的个数。
L	表示与源符号个数 K 对应的中间符号的个数。
X	一个正整数值，用于表示某个符号的 ESI
G	一个编码符号组中包含的编码符号个数。
N	一个源块中子块的个数。
T	符号的大小，单位是字节。
T'	子符号的大小，单位是字节。如果源块被分割称子块，那么 $T = T' * N$ ；如果源块未被分割，则 T' 无意义。
F	源数据的大小，单位是字符。
I	子块的大小，单位是字节。
P	在数据传输中，每个传输包的净负载，单位是字节。
Q	$Q=65521$ ，小于 2^{16} 的最大素数。
Z	源块的个数。
J(K)	与 K 关联的系统索引。
$I_{a \times b}$	表示 $a \times b$ 的单位矩阵。
$0_{a \times b}$	表示 $a \times b$ 的 0 矩阵。
a^b	a 的 b 次方。
ceil(x)	表示大于或等于 x 的最小正整数。
choose(i,j)	表示在 i 个对象中无重复的抽取 j 个对象的方法的个数。
floor(x)	表示小于或等于 x 的最大正整数。
$i \% j$	i 模除 j。
X^Y	表示位数相同的两个字符串 X 和 Y 进行按位异或。
Transpose[A]	表示 A 矩阵的转置矩阵。
Rand[X, i, m]	一个伪随机码生成器。
Deg[v]	度生成器。
LTEnc[K, C, (d, a, b)]	LT 编码生成器。
Trip[K, X]	一个三元整数组生成器。

3.3.2 编码参数的确定

解码端需要一些参数作为解码的依据, 这些参数要和解码端保持一致, 包括: F、T、Z、N、Al。 每个编码符号的对应参数都不相同, 而这些参数在解码时必不可少, 因此需要跟随每个传输包一起传输, 包括: SBN、ESI。因此在传输时, 需要携带一个包头承载这些参数, 根据设计的包头所占位数的限制来设计每个参数所占用的位数。

设 W 是子块大小的目标值, Kmin 是一个源数据块包含编码符号个数的最小目标值, Gmax 是每个传输包包含符号个数的最大目标值, Kt 是所有源块中符号数的总和。AL, Kmin, Gmax 的参数值需最先确定, 其它参数的确定方法如下^[15]:

$$\begin{aligned} G &= \min\{\text{ceil}(P \cdot K_{\min}/F), P/Al, G_{\max}\} \\ T &= \text{floor}(P/(Al \cdot G)) \cdot Al \\ K_t &= \text{ceil}(F/T) \\ Z &= \text{ceil}(K_t/K_{\max}) \\ N &= \min\{\text{ceil}(\text{ceil}(K_t/Z) \cdot T/W), T/Al\} \end{aligned}$$

上述方法得出的 G 和 N 的值是下限, 可以在有利于编解码的情况下提高这两个值。上述参数不保证数据包的负荷大小 P 能整除 T, 此时, 不能够使用的实际的 P 值, 而应该采用能被 T 整除的 P 值。如果 G 的值可以整除与 P/Al 的值, 那么 P 可以使用实际值。

包头的格式或者参数的值并不是一定的, 不同系统 Raptor 码的实现方案的参数确定方法都不相同, 这些参数与最终的解码效率有密切的联系。在本文系统实现时, 会按照系统的实际情况和需求对参数进行确定。

3.3.3 数据处理

编码过程中操作的数据单元是符号, 因此需要对源数据进行分割。源数据首先被分割成 $Z \geq 1$ 个源块。源块之间使用 SBN 来识别, 第一个源块的 SBN 是 0, 第二个的 SBN 是 1, 以此类推。每个源块分割成 K 个大小为 T 的源符号。源符号之间使用 ESI 来识别, 第一个源符号的 ESI 是 0, 第二个的 ESI 是 1, 以此类推。每个源块作为一个整体送入编码器, 进行一次编码过程, 并且每个源块的编码过程是独立的, 彼此之间没有任何关联。

如果解码端的存储器大小导致需要子块的存在, 那么, 首先将每个源块分割为 $N \geq 1$ 个子块, 子块要足够小, 以便放入内存进行解码。然后将每个子块分割为 K 个大小为 T 的子符号。需要注意的是, 每个源块包含的符号个数 K 可以不同, 但是所有符号的大小必须相等, 都为 T; 在一个源块内部, 每个子块的包含的符号的大小 T 可以不同, 但是每个子块包含的子符号个数必须相等, 都为 K。

进行分割时，源块和子块的结构是由以下五个参数：F、Al、T、Z、N 和函数 Partition[]决定的。

五个参数的值必须符合条件： $\text{ceil}(\text{ceil}(F/T)/Z) \leq K_{\max}$ 。函数 Partition[]使用两个整数参数 (I, J) 作为输入，最终输出一个参数组 (IL, IS, JL, JS)。输出的参数计算方法是 $IL = \text{ceil}(I/J)$, $IS = \text{floor}(I/J)$, $JL = I - IS \times J$, $JS = J - JL$ 。Partition[]的功能是将一个大小为 I 的块分割成 J 个大小近似相等的块，包括 JL 个长度为 IL 的块和 JS 个长度为 IS 的块。//1个IL大小的大子块，J-1个JL大小的小子块

分割方法如下^[15]:

$$K_t = \text{ceil}(F/T)$$

$$(K_L, K_S, Z_L, Z_S) = \text{Partition}[K_t, Z]$$

$$(T_L, T_S, N_L, N_S) = \text{Partition}[T/Al, N]$$

首先，源数据被分割为 $Z = Z_L + Z_S$ 个连续的源块，共 K_t 个符号。前 Z_L 个源块大小为 $K_L \times T$ 字节，后 Z_S 个源块大小为 $K_S \times T$ 个字节。如果 $K_t \times T > F$ ，那么在编码过程中，最后一个符号的末尾必须要补足 $K_t \times T - F$ 个零字节。

接下来，每个源块又被分割为 $N = N_L + N_S$ 个连续的子块，前面的 N_L 个子块，每个都由 K 个连续的子符号组成，每个子符号的大小为 $T_L \times Al$ ；后面的 N_S 个子块，同样由 K 个连续的子符号组成，每个子符号大小为 $T_S \times Al$ 。每个子符号的大小都是 Al 的倍数。

最终，一个源块中的第 m 个符号是由每个子块中的第 m 个子符号拼接而成的。这说明，如果子块的个数 $N > 1$ ，那么，符号就不是文件中的一个连续的片段。

数据分割完毕即可进行编码，编码完成后，需要将编码符号打包为编码包进行传输。每个编码包或者是完全由源符号组成的源包，或者是完全由修复符号组成的修复包。一个包可能包含任意个符号，即一个编码符号组，但是不同源块编码生成的编码符号不可以放在同一个编码包中传输。

如 3.3.2 小结所述，为了使编解码端的参数保持一致，编码端需要将参数 F、T、Z、N、Al 的值传输给解码端，每个编码包还需要携带与自身相关的参数作为包头，以用于解码。包头中包括以下信息：SBN、ESI。SBN 是源块的标识，同一源块编码产生的编码包的 SBN 值相同。所有编码符号中，ESI 的值在 0 到 $K-1$ 之间的是源符号，ESI 的值与它们在源块中的摆列顺序保持一致即可。ESI 的值为 K 或大于 K 的是修复符号。包头中携带的 ESI 应该是编码包中的第一个源符号的 ESI 值，也就是所有编码符号中的 ESI 最小值。剩下的源符号的 ESI 的值是从 $X+1$ 到 $X+G-1$ ，在编码包中的排列顺序与 ESI 的大小顺序相同。

相似的，每一个修复包中携带的第一个修复符号的 ESI， X ，作为修复包的 ESI 放置在包头，剩下的修复符号的 ESI 的顺序是从 $X+1$ 到 $X+G-1$ 。

3.3.4 生成中间符号的过程

首先需要明确的是，在编解码过程中，符号之间所进行的所有计算都是异或。

为方便描述，使用 $C'[0], \dots, C'[K-1]$ 表示 K 个源符号。使用 $C[0], \dots, C[L-1]$ 表示 L 个中间符号。

生成中间符号的过程十分特殊，它事实上是一个逆编码的过程。首先，要确定源符号和中间符号以及中间符号之间的约束关系。中间符号与源符号之间通过 LT 码的编码矩阵产生关联关系，中间符号之间通过预编码矩阵产生关联关系，这两种关系是生成中间符号的重要依据。

中间符号之间的关联关系是， $L > K$ 个中间符号中，排在后面的 $L-K$ 个中间符号是由排在前面的 K 个中间符号根据预编码矩阵生成的。这 $L-K$ 个中间符号由两个部分组成，即 S 个依据 LDPC 编码矩阵生成的 LTPC 编码符号和 H 个依据 Half 编码矩阵生成的 Half 编码符号。 $S+H = L-K$

设 G_LDPC 为 $S \times K$ 的 LTPC 码生成矩阵， G_Half 为 $H \times (K+S)$ 的 Half 码生成矩阵，则，中间符号的关联关系可以描述为：

$$G_LDPC * \begin{pmatrix} C[0] \\ \vdots \\ C[K-1] \end{pmatrix} = \begin{pmatrix} C[K] \\ \vdots \\ C[K+S-1] \end{pmatrix} \quad (\text{式 3-1})$$

$$G_Half * \begin{pmatrix} C[0] \\ \vdots \\ C[S+K-1] \end{pmatrix} = \begin{pmatrix} C[K+S] \\ \vdots \\ C[K+S+H-1] \end{pmatrix} \quad (\text{式 3-2})$$

在此特别要注意的是，矩阵与中间符号向量之间的“*”所代表的运算并不是矩阵的乘法运算。

生成矩阵的每一行都与等式右边的符号向量中的元素一一对应，每一列都与等式左边参与计算的符号向量的元素一一对应。若某一行的某一列为 1，则表示该列对应的符号参与了该行对应的符号的运算，而运算的实际过程是将该行中所有值为 1 的列所对应的符号进行异或运算，最终得到的符号就是该行所对应的的中间符号。编解码过程中，所有的“*”符号所表示的运算都是如此。

根据上述两个等式，如果要求一个满足式 3-3 的矩阵：

$$G_LDPC_H * \begin{pmatrix} C[0] \\ \vdots \\ C[L-1] \end{pmatrix} = \mathbf{0}_{(S+H) \times L} \quad (\text{式 3-3})$$

则根据公式 3-3 与公式 3-1、3-2 之间的联系，可以反推出矩阵 G_LDPC_H 的结构，该结构如图 3-3 所示：

所有的生成矩阵都是元素为 0 或 1 的矩阵，所有的“*”运算都是异或运算

	K	S	H
S	G_LDPC	I_S	0_SxH
H	G_Half		I_H

 图 3-3 矩阵 G_{LDPC_H} 的结构

中间符号与源符号之间的关联关系是，源符号是中间符号通过 LT 编码得出的，即满足等式：

$$C'[i] = \text{LTEnc}[K, (C[0], \dots, C[L-1]), (d[i], a[i], b[i])] \quad (\text{式 3-4})$$

其中 i 的取值范围为 $0 \leq i \leq K$ 。

等式中的 $(d[i], a[i], b[i])$ 是 K 个源符号关联的三元整数组。从中间符号和源符号的关联关系描述中可以看出，源符号是中间符号与 LT 编码矩阵运算得出的结果，即：

$$G_{LT} * \begin{pmatrix} C[0] \\ \vdots \\ C[L-1] \end{pmatrix} = \begin{pmatrix} C'[0] \\ \vdots \\ C'[K-1] \end{pmatrix} \quad (\text{式 3-5})$$

其中 G_{LT} 是 $K \times L$ 的 LT 编码矩阵。

如果将 G_{LDPC_H} 和 G_{LT} 拼接为一个矩阵与中间符号向量进行运算，那么得到的是 0 向量和 C' 拼接而成的向量。即：

$$A * \begin{pmatrix} C[0] \\ \vdots \\ C[S+H-1] \\ C[S+H] \\ \vdots \\ C[L-1] \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ C'[0] \\ \vdots \\ C'[K-1] \end{pmatrix} \quad (\text{式 3-6})$$

将该式简化为：

$$A * C = D \quad (\text{式 3-7})$$

其中 D 的构成是：前 $S+H$ 个符号是零符号和后 K 个符号是源符号。矩阵 A 的结构如图 3-4 所示：

	K	S	H
S	G_LDPC	I_S	0_SxH
H	G_Half		I_H
K	G_LT		

 图 3-4 矩阵 A 的结构

因此，由公式可以进一步推出：

$$\mathbf{C} = \mathbf{A}^{-1} * \mathbf{D} \quad (\text{式 3-8})$$

至此，即可得出了利用源符号生成中间符号的步骤，如下所述：

- (1) 生成 LDPC 码编码矩阵。
- (2) 生成 Half 码编码矩阵。
- (3) 生成与源符号关联的三元整数组。
- (4) 生成 LT 码的编码矩阵。
- (5) 得出矩阵 \mathbf{A} ，并求 \mathbf{A}^{-1} 。
- (6) 根据 \mathbf{A}^{-1} ，利用源符号生成中间符号。

接下来要按步骤说明生成中间符号的具体方法。

第一步，在生成 LDPC 码编码矩阵之前，先要确定 S 的值以及所需要的一些参数的值，这些参数的生成方法如下^[15]：

X 是符合 $X * (X - 1) \geq 2 * K$ 的最小正整数。

S 是符合 $S \geq \text{ceil}(0.01 * K) + X$ 的最小素数。

$L = K + S + H$ 。

生成 S 个 LDPC 符号的伪码如下^[15]：

```
Function LDPC(C[], S, Triple(d, a, b)){
  For i := 0 to K-1 do{
    a := 1 + (floor( $\frac{i}{S}$ ) % (S-1));
    b := i % S;
    C[K + b] := C[K + b] ^ C[i];
    b := (b + a) % S;
    C[K + b] := C[K + b] ^ C[i];
    b := (b + a) % S;
    C[K + b] := C[K + b] ^ C[i];
  }
}
End LDPC
```

最终获得的 $C[K], \dots, C[K+S-1]$ 即是 LDPC 编码符号。

生成 H 个 Half 编码符号的所需参数如下：

H 是符合 $\text{choose}\left(H, \text{ceil}\left(\frac{H}{2}\right)\right) \geq K + S$ 的最小整数， $H' = \text{ceil}\left(\frac{H}{2}\right)$ 。

$g[i] = i^{\text{floor}(\frac{i}{2})}$ ，其中 i 是所有的正整数。

$m[k]$ 表示 $g[]$ 的子序列, 该子序列满足条件: 每个元素的二进制表示中都恰好含有 k 个~~非0~~~~0~~

$m[j,k]$ 表示 $m[k]$ 的第 j 个元素, j 为正整数。则伪码如下^[15]:

```
Function Half(C[], m[]){
    For h := 0 to H-1 do{
        For j := 0 to K+S-1 do
            if (  $m[j,H]$  的第  $h$  位等于 1 ) then
                 $C[h+K+S] := C[h+K+S] \wedge C[j]$ ;
        /}
    /}
End Half
```

第三步, 生成三元整数组步骤如下:

(1) 生成所需参数。计算过程中需要输入的参数包括源符号的个数 K 和源符号的 ESI 值 X 。 L' 是大于或等于 L 的最小素数, $J(K)$ 是与 K 相关的系统索引值, 它能够确保生成的三元整数组用于生成中间符号时, 矩阵 A 可逆。

(2) 生成伪随机数。伪随机数生成器的计算方法是^[15]:

$$\text{Rand}[X, i, m] = (V_0[(X + i) \% 256] \wedge V_1[(\text{floor}(\frac{X}{256} - \frac{X}{256}) + i) \% 256]) \% m$$

其中 V_0 和 V_1 是两个大小为 256 的给定数组, 随机数的大小范围是 0 到 m 。

(3) 生成度 $\text{Deg}[v]$ 。首先利用 (2) 中的伪随机数生成器得到一个随机数 v , 根据 v 的取值, 在如图 3-5 所示的表中查找满足 $f[j-1] \leq v \leq f[j]$ 的 j 的取值。

Index j	$f[j]$	$d[j]$
0	0	--
1	10241	1
2	491582	2
3	712794	3
4	831695	4
5	948446	10
6	1032189	11
7	1048576	40

图 3-5 度生成器^[15]

确定 j 的值后, 即可得出度的值 $\text{Deg}[v] = d[j]$ 。

(4) 生成三元整数组。三元整数组包括 d , a , b 三个值, 计算方法如下^[15]:

```
Function Triple (K, X, L'){
     $A := (53591 + J(K)*997) \% Q$ ;
     $B := 10267*(J(K)+1) \% Q$ ;
     $Y := (B + X*A) \% Q$ ;
```

```

v := Rand[Y, 0, 220];
d := Deg[v];
a := 1 + Rand[Y, 1, L'-1];
b := Rand[Y, 2, L'];
/}
End Triple

```

第四步，构建 LT 码的编码矩阵。生成一个 LT 编码的方法，即 LTEnc[] 的流程如下^[15]：

```

Function LTEnc (K, Triple(d, a, b), L){
  While (b ≥ L) do b = (b + a) % L';
  Let result := C[b];
  For j := 1 to min(d-1, L-1) do {
    b := (b + a) % L';
    While (b ≥ L) do {
      b := (b + a) % L';
    }
    result := result ^ C[b];
  }
/}
End LTEnc

```

第五步，构造矩阵 A 并求逆。按照图 3-4 将生成的 LDPC、Half、LT 编码矩阵与单位矩阵和零矩阵拼接起来得到矩阵 A，利用高斯消元求出逆矩阵 A^{-1} 即可。

第六步，求中间符号。

在源符号向量中添加进 0 元素，得到 D，然后带入式 3-8 进行运算，得出中间符号的值。

求中间符号的运算过程正如 3.3.4 中所提到“*”运算的过程，根据 A^{-1} ，将一行中值为 1 的列所对应的 D 中的元素进行异或运算，即可得出行所对应的 C 中元素的值。至此，中间符号生成完毕。

3.3.5 生成编码符号的过程

由于本文介绍的是系统 Raptor 码，系统 Raptor 码生成的编码符号中的前 K 个符号正是源符号，这其中的原理也十分容易理解，源符号与中间符号的约束关系就是 LT 编码的过程，而由中间符号生成编码符号的过程同样也是 LT 编码，由于，因此，求出的编码符号就是源符号。

编码符号中，除了前 K 个源符号，剩下的是修复符号，设修复符号的个数为 R ， R 的值可以根据实际需要进行调整。编码符号中的源符号的 ESI 的取值范围是 0 到 $K-1$ ，因此编码符号的 ESI 取值范围是 K 到 $K+R-1$ 。

在生成一个修复符号之前，首先使用伪随机数生成器和度生成器计算与之相对应的 $\text{Deg}[v]$ 的值，然后根据该修复符号的 ESI 和 K 的值计算与之相关联的三元整数组，接下来将就可以使用 LT 编码器生成该修复符号。至此，编码过程结束

3.4 系统 Raptor 码的解码原理

根据 3.3 节的内容，系统 Raptor 码的解码端接收到的每个编码符号，都是由中间符号经过异或计算得出的，根据它们与中间符号的关系可以联立出线性方程组，任何解出该线性方程组的算法都可以用来解出中间符号，随后即可恢复出源符号。因此，解线性方程组的算法决定了解码端的效率。

解码一个源块时，解码端需要的参数包括 F 、 T 、 K ，这些参数的值与编码端一致。根据 K 的值，解码端也可以计算出 $L = K+S+H$ 的值，也就可以得出中间符号之间的预编码约束关系，以及中间符号和源符号之间的 LT 码约束关系。同时，只要知道每个编码符号的 ESI，就可以求出该编码符号对应的三元整数组，那么每个编码符号与中间符号的关联关系也是可知的。

需要注意的是，在编码过程中，最后的一个源符号可能填充了用于对齐 0 字节，因此，如果编码端接收到了这个符号，填充的 0 字节不会被传输，那么在编码前需要重新填充。

设 N 是解码端接收到的编码符号的个数， $M = S+H+N$ 。需要注意的是，要正常解码必须满足 $M \geq L$ ，也就是 $N \geq K$ 。C 仍然代表中间符号向量。

使用 $E[0], \dots, E[M-1]$ 表示解码端接收到的 M 个编码符号。

设 D' 向量的结构是：前 $S+H$ 个是值为 0 的符号，后 N 个是接收到编码符号。

矩阵 A' 在解码端是一个 $M \times L$ 的矩阵，并且满足条件 $A' * C = D'$ 。

矩阵 A' 的元素 $A'[i, j]$ 的值的确定方法如下：如果 $A'[i, j]$ 中的 j 对应的中间符号如果参与了 LDPC 编码、Half 编码，或者参与了 i 对应的编码符号的 LT 编码过程，那么 $A'[i, j] = 1$ ；如果 i 和 j 对应了相同的 LDPC 编码或者 Half 编码，那么 $A'[i, j] = 1$ ，其他情况下， $A'[i, j] = 0$ 。

解码一个源包的过程，本质上就是要根据已知的 D' 和 A' 解出 C 。容易看出，只有矩阵 A' 的秩为 L 时， C 才能被解出，这也就是为什么接受到的编码符号个数 $N \geq K$ 。一旦 C 被解出，就可以丢失的源符号的 ESI 求出它对应的三元整数组，从而获得它与中间符号的约束关系，最终将其恢复出来。

3.4.1 解码出中间符号的过程

根据 A' 的元素取值方法，可以很容易的构建出矩阵 A' ，如图 3-6 所示。

可以看出构建解码端矩阵 A' 的方法与构建编码端矩阵 A 的方法是相似的，两个矩阵不同的地方只有 G_{LT_M} 的部分。

	K	S	H
S	G_{LDPC}	I_S	$0_{S \times H}$
H	G_{Half}		I_H
M	G_{LT_M}		

图 3-6 解码端的矩阵 A' 结构

编码端使用的是 K 个源符号的 ESI 构建 LT 编码矩阵，而解码端使用的是接收到的 N 个编码符号的 ESI 构建 LT 矩阵，但是构建流程是相同的。

最终， $A' * C = D'$ 的实质是：

$$A' * C = \begin{pmatrix} \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \begin{matrix} S \uparrow \\ \\ \end{matrix} \\ \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \begin{matrix} H \uparrow \\ \\ \end{matrix} \\ \begin{matrix} E[0] \\ \vdots \\ E[M-1] \end{matrix} \begin{matrix} M \uparrow \\ \\ \end{matrix} \end{pmatrix} \quad (\text{式 3-9})$$

可以看出，这是一个线性方程组的形式，求解 C 的过程也就是解线性方程组的过程。解方程的过程本质上是对 A' 矩阵进行高斯消元变为单位矩阵的，该过程可以采用各种不同的算法，文献[15]给出的算法效率低下^[10]，尤其是在 K 较小的情况下，因此，此处可以采用更简洁的算法进行高斯消元，但是，无论采用哪种方法，必须遵循以下要求：

首先设 $c[0] = 0, c[1] = 1, \dots, c[L-1] = L-1$ ，设 $d[0] = 0, d[1] = 1, \dots, d[M-1] = M-1$ ；运算过程与解码过程的关系如下所述：

- (1) 若 A' 矩阵中的 i 行要异或到 i' 行，那么符号 $D'[d[i]]$ 要异或到 $D'[d[i']]$ 。
- (2) 若 A' 矩阵的 i 行与 i' 行进行交换，那么符号 $D'[d[i]]$ 要与 $D'[d[i']]$ 进行交换。

(3) 若 A' 矩阵的 j 列与 j' 列进行交换, 则符号 $C[c[j]]$ 要与符号 $C[c[j']]$ 进行交换。

高斯消元的最后结果是矩阵 A' 的最后 $M-L$ 行被消除, 并变为单位矩阵, 而 D' 的前 L 个符号 $D'[d[0]]$, $D'[d[1]]$, ..., $D'[d[L-1]]$ 的值就是中间符号的值。

3.4.2 解码出源符号的过程

解出中间符号 C 之后, 就可以根据源符号与中间符号的关联关系, 恢复出丢失的源符号, 即:

$C'[i] = \text{LTEnc}[K, (C[0], \dots, C[L-1]), (d[i], a[i], b[i])]$, 其中 i 是需恢复的源符号的 ESI, $(d[i], a[i], b[i])$ 是与之相对应的三元整数数组。

至此, 解码过程结束。

3.5 本章小结

本章详细介绍了数字喷泉码的工作原理以及系统 Raptor 码的原理和实现方法, 包括了编码和解码两个部分。关于编码部分, 本章的第三节给出了从参数确定、编码原理、方法和最终数据传输时的细节; 解码部分, 本章的第四节给出了基本的原理和算法。

系统 Raptor 码的原理和方法被写入一些国际组织的传输标准被用于 3GPP 的多媒体广播或组播服务。本文将系统 Raptor 码应用与无线多跳自组织网络上的视频传输, 主要因为它的编解码计算复杂度低, 速度快, 且同时适用于源符号数较小的情况, 普通 Raptor 码的良好性能需要在源符号数较大的情况下才能显现。

第四章 视频多跳传输系统的设计与实现

本章将详细描述系统实现过程，包括程序的流程、实现的细节、遇到的问题等。内容整体分为发送端的详细实现和接收端的详细实现两个部分，每个部分的介绍按照功能模块的顺序依次进行。

4.1 发送端概述

发送端进程总共包含三个线程：

(1) 主线程。主线程的工作是初始化编码器参数、初始化程序参数、初始化并打开摄像头设备，建立 UDP 连接，建立子线程。

(2) 视频图像处理线程。该线程承担了所有与视频数据相关的操作，包括采集、压缩、编码、传输，在后面的几节中将分模块详细描述。该线程的工作流程如图 4-1 所示：

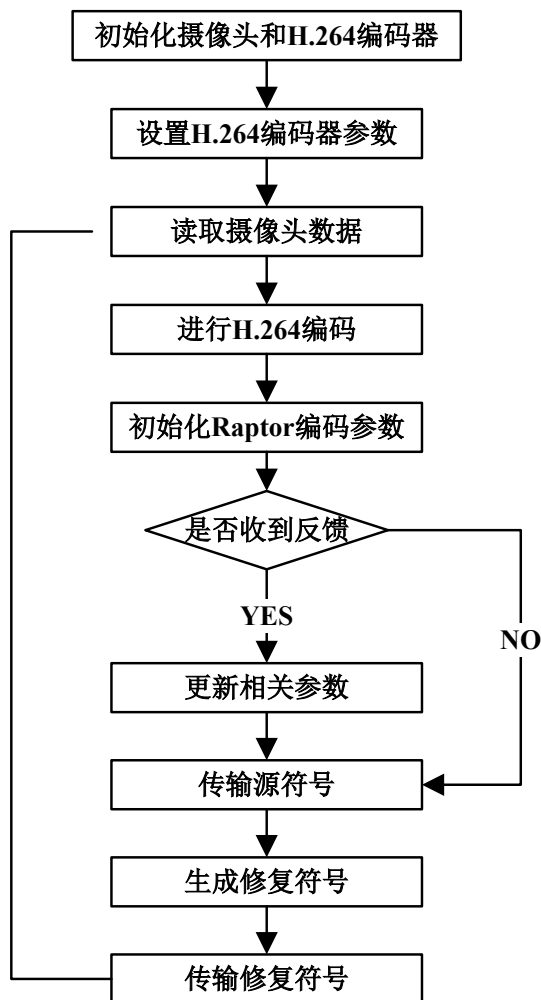


图 4-1 发送端视频处理工作流程

(3) 进程间通信线程。发送端进程是采集节点中多个传感器进程中的一个, 采集节点中会有一个主进程对所有的传感器进程进行管理。该线程的功能是与采集节点中的主进程保持通信, 及时上交发送端进程的状态。

在运行发送端程序时, 可以通过命令行输入参数, 改变这些参数的默认值, 方便程序的调试。

4.2 数据采集模块的实现

所有的设备挂载到嵌入式平台上后, 在/dev 文件夹下都会有一个文件名与这个设备相对应, 摄像头对应的是/dev/video0, 系统将设备的细节全部屏蔽, 对设备的操作就等同于对文件的操作。

在初始化时, 程序使用文件 I/O 中的 open 函数打开摄像头文件, open 函数会返回一个文件描述符, 通过该文件描述符即可对摄像头进行一系列的操作。数据采集模块使用 read 函数从摄像头文件中读取图像数据, 数据的储存格式是 YUV420。根据嵌入式平台中摄像头驱动的设置, 采集到的图像分辨率为 480*272。如果摄像头本身能够获得更好质量的图像, 可以修改摄像头的驱动中的相关参数, 这样就能采集到更高分辨率或更高帧速率的图像。

4.3 数据压缩模块的实现

数据压缩模块采用 H.264 编码标准对数据进行压缩。

4.3.1 H.264 视频压缩格式

H.264 是新一代的编码标准, 以高压压缩高质量和支持多种网络的流媒体传输著称。概括来说, H.264 编码的理论依据是视频图像之间的关联性。

如果观察一段时间很短视频, 可以很容易的发现, 视频相邻的若干帧画面中, 大部分的像素都是没有变化的, 尤其是对于拍摄角度固定的摄像头, 很多时候画面几乎是静止的。如果将这些相似的图像信息无差别全部存储, 很多数据都会是重复的。H.264 编码就是将这些重复的信息只存储一次, 从而减低了数据量。

视频画面在时间维度上的连续性, 使得相邻帧的画面之间的区别有限, 而在同一画面内, 各个像素之间存在空间上的连续性。根据视频图像的特点, H.264 标准在编码一帧画面时, 根据像素之间的相关性将图像分割成块, 以块为单位, 结合块之间的相关性进行存储, 既完整保留了图像信息, 又减少了数据量。对于多帧图像, H.264 标准采用的方法是, 将图像以 N 帧为单位分为多组, 每一组内首先编码第一帧图像, 完全保留该图像中的数据, 作为关键帧, 剩下的 N-1 帧图像只编码它们与关键帧的区别。在 H.264 编码标准中, 完整编码的关键帧称为 I

帧，参考 I 帧编码的帧称为 P 帧^[16]。

采集模块获得的 YUV420 格式图像的分辨率是 480*272，因此，一个格式的视频图像的大小是 195840 字节，经过 H.264 编码后，I 帧的大小平均为 15000 字节左右，而 P 帧的大小平均为 5000 字节左右，充分达到了减少数据量的目的。

H.264 编码后的数据包括两个重要的字符串，SPS 和 PPS。SPS 是 sequence_parameter_set，即序列参数集，PPS 是 picture_parameter_set，即图像参数集^[16]。这两个参数集包含着初始化解码器所需要的重要参数，解码端获得这两个参数集后才能正常进行解码。

H.264 还具有错误隐藏的能力，在数据传输出现错误的情况下，这种机制能够将错误进行掩盖，达到比较好的解码效果。但是这种机制只在个别数据出现错误的情况下有效，对于网络丢包这种大范围的错误没有效果。

4.3.2 数据压缩的过程

整个 H.264 编码过程通过调用嵌入式平台 MFC(Multi_Format Codec)模块的 API 函数完成。

H.264 编码前，可以对编码中涉及的参数进行设置，例如图像组大小 GOP，量化精度 QP，其中 GOP 决定了 I 帧个数与 P 帧个数的比例，QP 决定了画面质量，这两个值都能对需传输的数据量和图像质量产生影响。

编码的过程包含三个步骤：

(1) 编码器的初始化。实现函数是 `mfc_encoder_init()`，该函数的输入参数为图像的分辨率和基本参数，返回的是 MFC 的控制句柄，之后所有的 MFC 操作都需要使用该句柄完成。

(2) 编码。实现函数为 `mfc_encoder_exe()`，该函数的输入参数包括控制句柄、YUV 数据的起始地址和大小、I 帧标记、编码后数据大小的存储地址，函数返回的是编码后数据的首地址。

(3) 编码器释放。实现函数为 `mfc_encoder_free()`，该函数的功能是释放编码器，在进程被结束时被调用。

由于只有在编码第一帧时编码器才会将 SPS 和 PPS 添加到编码结果中，随后无论是 I 帧还是 P 帧的编码结果中都不在包含这些信息。如果第一帧没有在接收端恢复成功，或者解码端的运行时间是随机的，可能收不到第一帧，那么解码端的 H.264 解码将永远不成功。系统解决这个问题的方法是，将第一帧 H.264 编码后生成的 SPS、PPS 信息单独保存，并添加到所有 I 帧编码后的数据前面，这样，接收端只要接受并恢复任意一个 I 帧，就能够顺利开始解码。

综上所述，数据压缩的详细流程如图 4-2 所示：

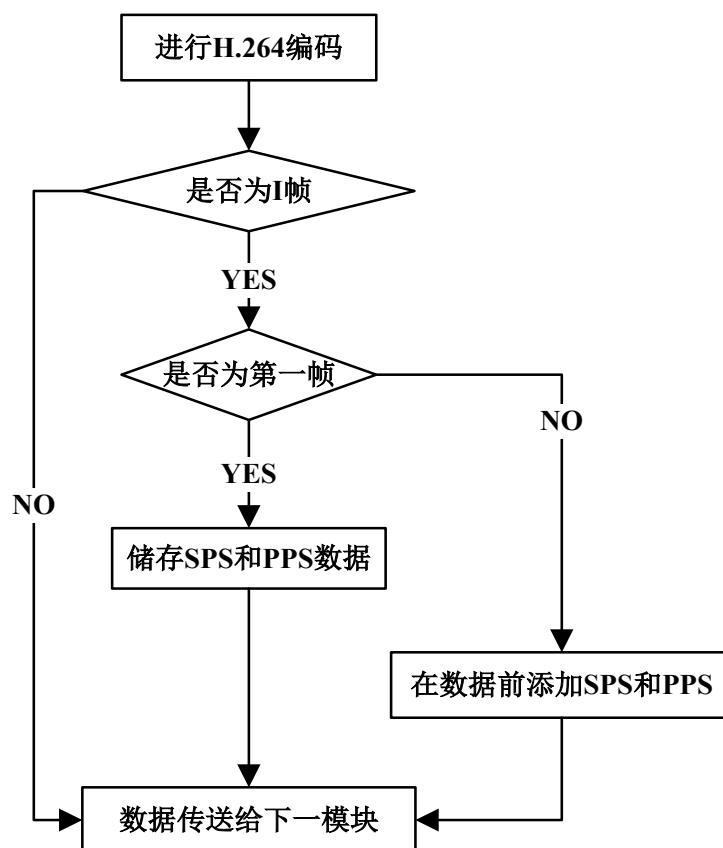


图 4-2 数据压缩模块工作流程

4.4 数据编码模块的实现

该章节将详细描述将系统 Raptor 码的编码方法应用于系统的过程。

根据本文 3.2 节对于系统 Raptor 编码原理的描述，可以发现，无论是编码过程或者解码过程，都包含两种操作：Raptor 编码操作和矩阵操作。因此，编码模块的代码设计也分为这两个部分。编码操作部分中会引用到矩阵操作部分中的数据结构和函数，而主函数进行编码时需调用编码操作部分中的数据结构和函数。

为了方便主函数中的操作以及避免频繁的申请和释放内存空间，源符号、中间符号和编码符号的存储空间在主函数中申请，并且要足够大，能够满足所有 K 取值的情况，编码时将这些存储空间的地址传输给编码器。

4.4.1 数据分割

系统中每次编码结束后传输的数据应该包含一整个视频帧，根据原理描述，参与一次编码的单位是一个源块。系统将一个视频帧视作源数据，如果源数据分为多个源块，这些源块轮流编码并传输，势必破坏了系统的实时性，因此，源数据包含的源块数 $Z=1$ ，为了尽可能简化编码过程，源块不含子块。


```
typedef struct {
    uint32 K; //源符号个数
    uint32 S; //LDPC 编码符号个数
    uint32 H; // Half 编码符号个数
    uint32 L; //中间编码符号个数
    triple trp; //三元整数组
    mymatrix Amat; //用于存储 L*L 矩阵 A
    mymatrix A_1mat; //用于存储矩阵 A 的逆矩阵
}RaptorParam,*RParam
```

其中，triple、mymatrix 为自定义的结构体，triple 储存三元证书组的值，mymatrix 是矩阵操作部分的重要数据结构，内容如下：

```
typedef struct {
    uint32 row; //该矩阵的行数
    uint32 colum; //该矩阵的列数
    uint8 ** rowpoint; //二维指针
}MyMatrix,*mymatrix
```

其中，二维指针 rowpoint 指向矩阵的首地址。

编码操作部分包含的函数如下：

表 4-1 编码操作部分包含的函数

函数名称	函数功能
trip()	三元整数组生成器。
deg()	度生成器
myrand()	随机数生成器
raptor_getLDPC()	LDPC 编码矩阵生成函数
gray_m()	计算 $m[j,k]$ (见 3.2.4)
raptor_getH()	Half 编码矩阵生成函数
raptor_getLT()	LT 编码矩阵生成函数
my_xor()	符号之间进行异或运算的函数
raptor_intermediate()	计算中间符号生成矩阵的函数
raptor_reset()	参数重置
raptor_init()	编码参数初始化
raptor_encode()	编码函数
raptor_parameterfree()	编码参数释放

矩阵操作部分包含的函数如下：

表 4-2 矩阵操作部分包含的函数

函数名称	函数功能
search_col_1()	从第 i 行开始，查找第 j 列中值为 1 的行的行号
row_exchange()	矩阵进行行互换操作
row_or()	矩阵的行之间进行异或操作
matrix_init()	初始化矩阵
matrix_inverse()	矩阵求逆
matrix_reset()	将矩阵重置为零
matrix_free()	释放矩阵存储空间

主函数和以上列出的函数之间的调用关系如图 4-5 所示：

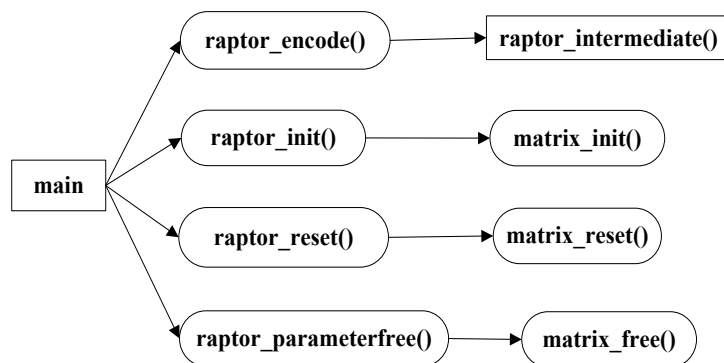


图 4-5 编码函数调用关系图 1

第一次进行编码时，主函数调用 raptor_init()对 Raptor 编码器的参数进行初始化，在每次编码生修复符号时调用 raptor_encoder()函数，进行下一次编码之前需要调用 raptor_reset()重新设置新的编码参数。raptor_encode()除了调用了 raptor_intermediate()生成编码矩阵 A，还实现了求出中间符号和修复符号的过程，这些过程中调用的运算操作函数在图中被省略。

raptor_intermediate()调用的函数如图 4-6 所示：

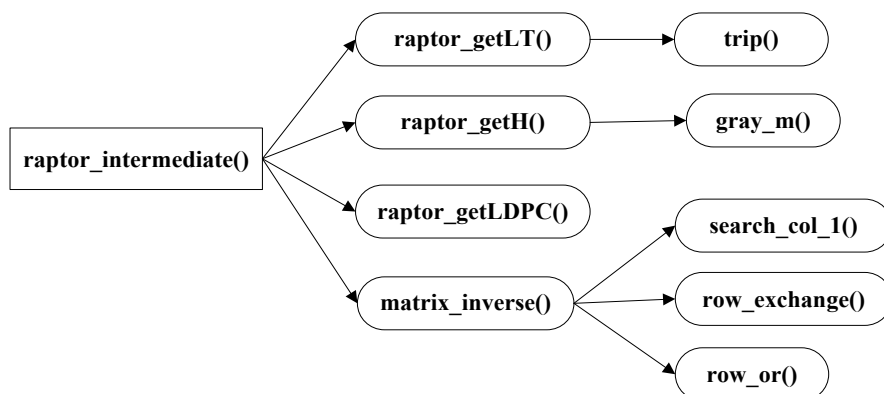


图 4-6 编码函数调用关系图 2

4.4.4 编码过程

解码工作主要通过函数 `raptor_encode()` 实现，该函数中包含了系统 Raptor 编码的主要两个步骤，生成中间符号和生成修复符号。主函数在调用该函数时已经在源符号前面填充了 0 符号，组成了 3.2.4 中提到的 D 向量。该函数的工作流程如图 4-7 所示：

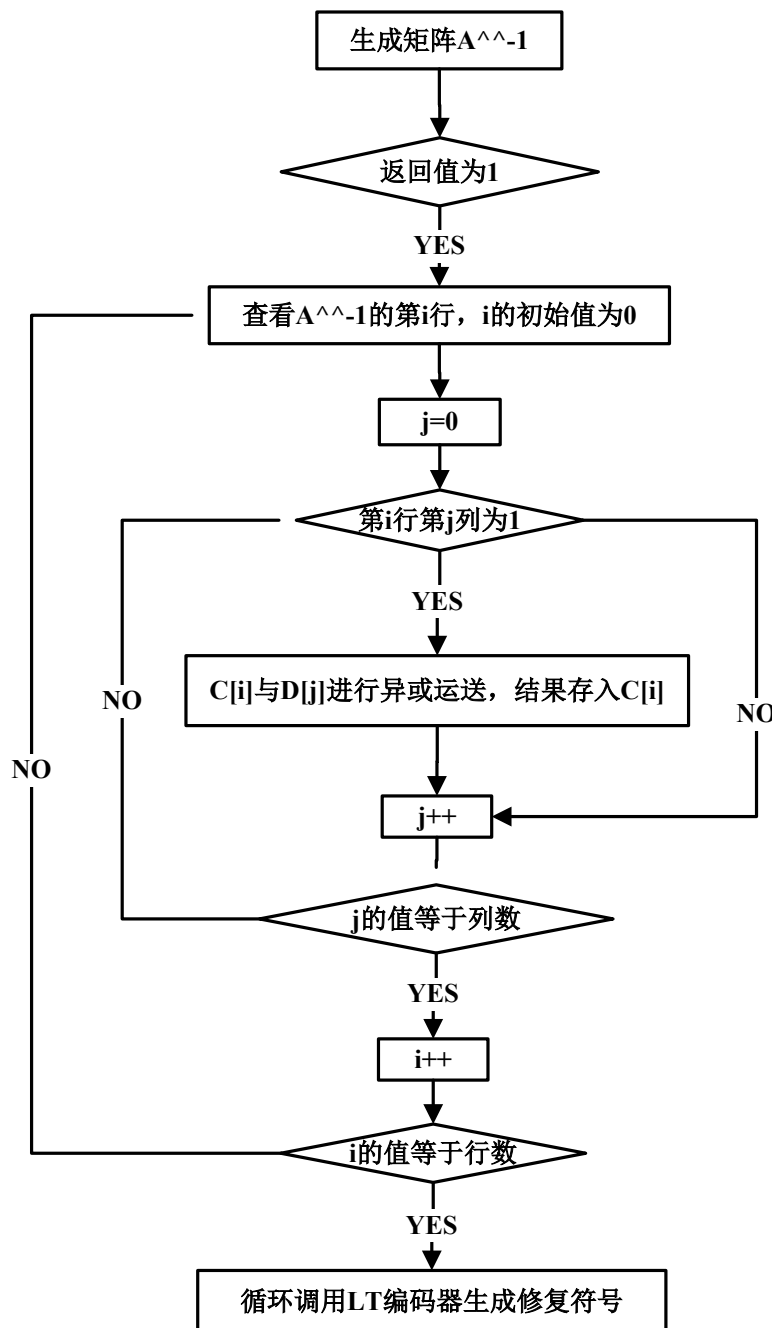


图 4-7 编码流程图

首先，将初始化中间符号和编码符号的值为 0，计算生成中间符号的矩阵 A^{-1} ，如果返回值为零，则说明计算矩阵的过程出错，编码过程终止，如果返回

值大于零，则说明矩阵 A^{-1} 生成成功。接下来，按行数遍历 A^{-1} ，将值为 1 的列所对应的 D 向量中的符号进行异或运算，得到中间符号，最后，进行 R 次的 LT 编码，最终得到 R 个修复符号并返回。

在整个流程中，最重要的部分是计算矩阵 A^{-1} 的过程。求 A^{-1} 之前，要先求矩阵 A，矩阵 A 由 LDPC 编码矩阵、Half 编码矩阵、LT 编码矩阵、 $S \times S$ 的单位矩阵、 $H \times H$ 的单位矩阵和 $H \times H$ 的零矩阵组成，具体组成方法见图 3-4，各个矩阵的生成方法见 3.3.4 中的伪码。

需要注意的是，伪码的结果并不是生成编码矩阵，而是获得编码符号，因此，在实现伪码时，要将生成编码符号的语句改为将矩阵中某元素的值设为 1 的语句。例如 LDPC 编码中的语句：

$$C[K+b] = C[K+b] \wedge C[i]$$

语句的含义是：第 $K+b$ 个中间符号的值等于自身的原始值与第 i 个中间符号进行异或后的值。

实现时进行的操作应该是：将矩阵 A 第 b 行第 i 列和第 b 行第 $K+b$ 列的值置 1，第 b 行第 $K+b$ 列置 1，就是矩阵 A 中 I_S 的一部分。

Half 编码和 LT 编码实现过程也参照此方法。

计算 A^{-1} 的过程中，最为关键的部分是矩阵 A 求逆的过程，系统采用的矩阵求逆算法是初等变换法，其原理是：

$$(A|E) \xrightarrow{\text{初等行变换}} (E|A^{-1})。$$

采用初等变换法的原因是，计算简单，实现容易。求逆过程中的矩阵消元的算法是高斯约当消元法。

高斯约当消元法的过程可以总结为以下三个步骤。算法需要遍历矩阵 A 的，对每一列都要进行以下操作：

(1) 遍历第 j 列，从该列的第 j 行开始寻找第一个值为 1 的行。

(2) 将该行与第 j 行的值进行交换。

(3) 现在第 j 行的值一定是 1，随后将第 j 行与所有第 j 列值为 1 的行进行异或操作，经过此过程，第 j 列中除了第 j 行的值，所有的值都为 0。

可以看出，与一般的初等变换不同的是，行之间所有的运算操作不是加减乘除，而是异或。

在初等变换中，矩阵的状态变化如图 4-8 所示：

$$(A) \rightarrow \begin{pmatrix} I & U \\ 0 & \end{pmatrix} \rightarrow (I_{M \times K})$$

图 4-8 高斯消元中矩阵的变化

可以看出，矩阵的初始状态是 A，通过初等行变换，矩阵逐渐变为单位矩阵、

0 矩阵和矩阵 U 的组合, 矩阵 U 是还未被上述操作处理过的列的集合。最终, 矩阵 A 被处理为单位矩阵。

实现时, 需要在求逆前初始化两个 $L \times L$ 的矩阵, 一个是需要求逆的矩阵 A , 另一个要初始化为单位矩阵 E , 需要注意的是, 求逆过程中对矩阵 A 进行的所有初等行变换, 矩阵 E 也要进行一遍, 这样, 当矩阵 A 变为单位矩阵时, 原先的矩阵 E 将会变为矩阵 A 的逆矩阵。

4.4.5 编码符号的传输

将编码符号进行传输时, 每个编码包中只包含一个编码符号, 每个编码符号都要携带一个包头, 该包头包含了接收端需要获得的信息, 包头格式如下:

```
typedef struct{
    int frame_no; //编码包所属的帧的编号
    long slice_no; //该编码包在传输的所有编码包中的序号
    long F; //表示该编码符号所属源数据的大小
    int T; //该编码符号的大小
    int K; //该编码符号所属源块中源符号的个数
    int R; //所有编码符号中包含的修复符号的个数
    int esi; //表示该编码符号的 ESI
} Frame_header;
```

包头中, `frame_no` 相当于本文第三章中提到的 SBN, 它标记了哪些编码符号应该被放在一起用于恢复一个视频帧的数据; `slice_no` 主要用与体现编码包的有序性, 方便接收端统计丢失的编码包数目; `F` 和 `T` 的单位是字节。

根据系统 Raptor 码的特性, 编码符号中 ESI 范围为 0 到 $K-1$ 的符号是源符号, 因此, 为了进一步压缩编码的时间, 系统不再编码这 K 个符号, 直接传输源符号, 然后再将生成的修复符号依次传输出去。

4.5 接收端概述

接收端进程包含三个线程:

(1) 主线程。主线程中的主要操作包括初始化需要的参数, 创建子线程, 建立 UDP 连接。

(2) 视频图像处理线程。该线程承担了接收端所有与恢复视频帧相关的工作, 包括接收编码包、统计编码包丢失情况、解码恢复视频帧数据、解压缩、显示, 在后面的内容中将进行详细的介绍。

(3) 丢包率反馈线程。该线程负责以固定的频率将网络丢包率通过 UDP 反

馈给发送端。

其中视频图像处理线程的工作流程如图 4-9 所示：

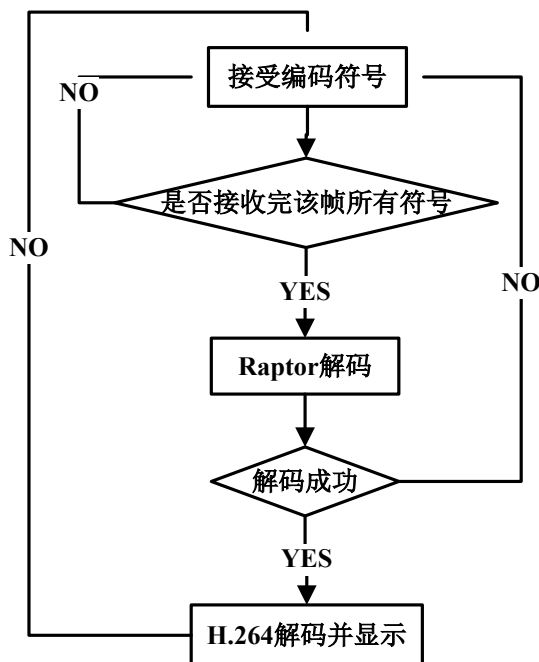


图 4-9 解码端视频处理流程图

4.6 数据解码模块的实现

本节将详细描述将系统 Raptor 码的解码方法应用于系统的过程。

根据本文 3.3 中介绍的系统 Raptor 解码原理，解码模块也分为 Raptor 解码操作和矩阵操作两个部分。因此，解码模块分为这两个类。解码操作类中会引用到矩阵操作类中的数据结构和函数，而主函数进行编码时需调用解码操作类中的数据结构和函数。

根据解码原理的描述，若所有的编码符号中的源符号都被接收到，则不需要进行解码，若源符号有丢失，并且接收到的编码符号的个数 N 大于源符号的个数 K ，则可进行 Raptor 解码。

4.6.1 数据结构和函数

系统 Raptor 解码过程中涉及到的主要结构体为，该结构体中包含了解码过程需要的参数：

```

typedef struct RaptorParam_dec{
    uint32 K; //源符号个数
    uint32 S; //LDPC 编码符号个数
    uint32 H; //Half 编码符号个数

```

```

uint32 L; //中间编码符号个数
uint32 N; //接收端接收到的所有编码符号的个数
mymatrix Amat_dec; //表示  $M \times L$  的解码
uint16* list; //ESI 的列表
}*RParam_dec

```

其中, N 和 M 满足 $N \geq K$, $M = S + H + M$; list 存储了接收端接收到的所有编码符号的 ESI, 按照解编码符号的存储顺序排列即可。mymatrix 结构体与本文 4.4.3 中给出的定义相同。

解码操作类包含的函数如下:

表 4-3 解码操作类的函数

函数名称	函数功能
trip()	三元整数组生成器。
deg()	度生成器
myrand()	随机数生成器
raptor_getLDPC()	LDPC 编码矩阵生成函数
gray_m()	计算 $m[j,k]$ (见 3.2.4)
raptor_getH()	Half 编码矩阵生成函数
raptor_getLT()	LT 编码矩阵生成函数
raptor_getLTrow()	计算 LT 编码矩阵中的一行
my_xor()	符号之间进行异或运算的函数
raptor_intermediate()	计算中间符号生成矩阵的函数
raptor_init()	解码参数初始化
raptor_decode()	解码函数
raptor_parameterfree()	解码参数释放

矩阵操作类包含的函数如下:

表 4-4 矩阵操作类的函数

函数名称	函数功能
search_col_1()	从第 i 行开始, 查找第 j 列中值为 1 的行的行号
row_exchange()	矩阵进行行互换操作
row_or()	矩阵的行之间进行异或操作
matrix_init()	初始化矩阵
matrix_equations	矩阵高斯消元操作
matrix_reset()	将矩阵重置为零
matrix_free()	释放矩阵存储空间

主函数、解码操作类和矩阵操作类中主要函数的调用关系如图 4-10 所示：

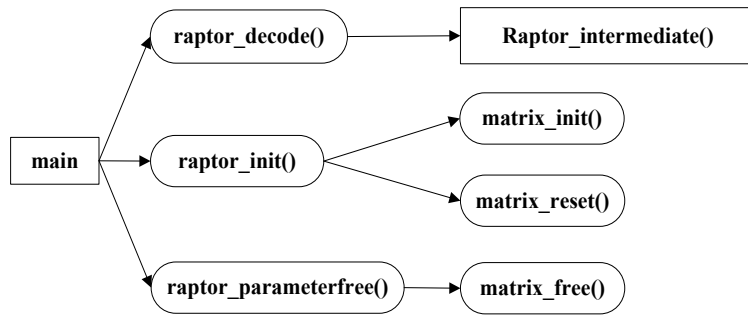


图 4-10 解码端函数调用关系图 1

raptor_intermediate()调用的主要函数如图 4-11 所示：

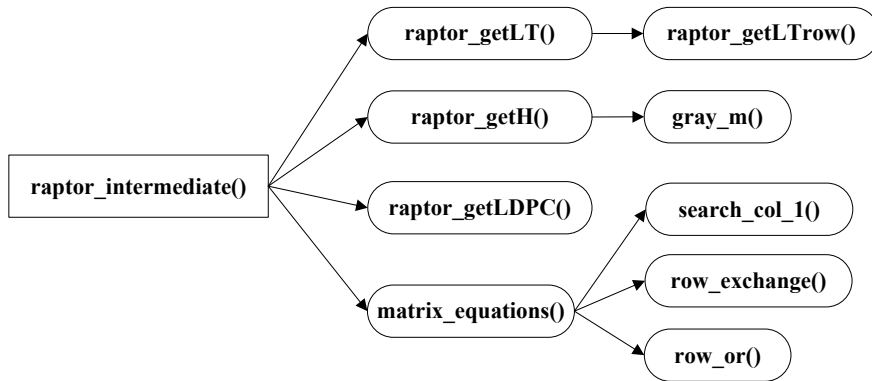


图 4-11 解码端函数调用关系图 2

4.6.2 解码过程

解码过程主要通过 raptor_decode()函数实现，该函数包含了解码中的两个主要操作：解线性方程组求中间符号和恢复源符号。

解码前，要在接收到的编码符号前添加 S+H 个 0 符号，拼接成本文 3.4 节中提到的向量 D'。

函数的工作流程与编码端 raptor_encode()十分相似。第一步是计算解码矩阵 A'的值，与计算编码端的矩阵 A 的步骤相同，需要分别计算 LDPC、Half、LT 编码矩阵。但是解码矩阵 A'是 M*L 的矩阵，它包含的 LT 编码矩阵是按照接收到的编码符号的 ESI 列表依次生成。

计算出解码矩阵 A'后，如式 4-1 所示，矩阵 A'作为线性方程组等式左边的参数矩阵，中间符号的向量是待求的未知数，向量 D'是等式右边的值。

$$\mathbf{A}' * \mathbf{C} = \mathbf{D}' \quad (\text{式 4-1})$$

线性方程组的算法同样也是高斯约当消元法，算法的工作流程与描述参考本文 4.4.4 小结的内容。与编码端不同的是，消元时只需要初始化一个矩阵 A'，而对矩阵 A'进行行变换时，这些行所对应的的向量 D'中的符号也要进行相应的操作。当矩阵 A'变成如图 4-12 所示形式：

$$\begin{pmatrix} I_{L \times L} \\ 0_{(M-L) \times L} \end{pmatrix}$$

图 4-12 矩阵 A' 的最终结果图

此时，向量 D' 中前 L 个符号即为中间符号。

计算出中间符号后，即可通过 LT 编码的过程计算出丢失的源符号。

4.7 数据显示模块的实现

视频帧的源符号被恢复后，需要将它们按 ESI 的顺序拼接起来，送入 ffmpeg 的 H.264 解码器，最终获得视频帧的 YUV 数据。

系统采用 MFC (Microsoft Foundation Classes) 构架作为视频的展示平台。为了使视频数据在 MFC 的控件中显示，同时，为了方便未来对系统进行计算机视觉方向的功能扩展，需要将 YUV 数据转为 IplImage 格式进行储存。

OpenCV (Open Source Computer Vision Library) 是一款跨平台的计算机视觉库，它实现了实现了图像处理和计算机视觉方面的很多通用算法。IplImage 是 OpenCV 的函数库中最为重要的结构体，几乎所有的图像处理的函数都要使用该结构体。

将 YUV 的 Y、U、V 通道数据使用 cvSetData 函数分别存入三个初始化好的单通道 IplImage 结构体中，然后使用 cvMerge 函数将三个 IplImage 合并为一个三通道的 IplImage，最后使用 cvCvtColor 将 YUV 格式转为 RGB 格式。接下来使用 CvvImage 结构体的 copy 函数将最终的 IplImage 拷贝过来，画在控件上即可。

4.8 修复符号个数的确定方法

在本文 4.5 节中可以看到，本系统的接收端有一个反馈丢包率的线程。其中，估计丢包率所使用的计算模型是 Gilbert 模型^[17]。在解码端统计丢包率的目的是反馈给编码端用于确定编码需要生成的修复符号个数 R。

在编码过程中，源符号个数 K 的值决定了编码过程的速度，K 越大，编码速度越慢。而修复符号的个数 R 决定了最终发送的编码包的个数，如果 R 的值过大，那么传输一个视频帧的时间会更长，不利于实时性。同时，R 的值也决定了解码端能够接收到的编码包的个数 N 的值，如果 R 的值过小，解码端将无法接收到足够数量的编码符号，从而无法正确恢复出所有的源符号。

在确定 R 的值时，需要考虑以下几点：

- (1) 解码端的解码成功率。解码端想要成功恢复所有的源符号，需要一定

的解码开销，解码开销即 $N-K$ 的值。当解码开销到达一定值时，Raptor 解码的成功率就能达到让人满意的程度，因此 R 的值一定要保证解码开销达到必须的值。

(2) 网络的丢包率。网络的丢包率直接了解码端平均能够接收到的编码符号的个数，因此在确定 R 的值时，需要考虑网络丢包率。

4.8.1 解码开销的确定

由刚才描述的内容可知，为了确定修复符号 R 的值，首先要确定的是本系统中解码端所需的解码开销。在一些文献资料^[10]中可以查找到 K 的值较大时，解码端的解码开销 $N-K$ 的值，但是，在本系统中， K 的值很小，范围在 5 到 50 之间，而文献中的仿真时的 K 值一般在 200 以上，因此文献中的仿真结果并不适用于本系统。因此，在系统的发送端和接收端都基本实现后，对系统的解码开销进行了计算。

在解码端首先要统计进行的 Raptor 解码的次数、解码失败的次数，在解码失败的情况下，需要统计 $N-K$ 各个取值的出现概率。

统计时，设修复符号与编码符号的比率为固定值 1:1。

如图 4-13 所示，图中 P_{lose} 为丢包率，展示了在不同丢包率的情况下，解码过程失败时， $N-K$ 的值分别为 1、2、3、4 的概率。

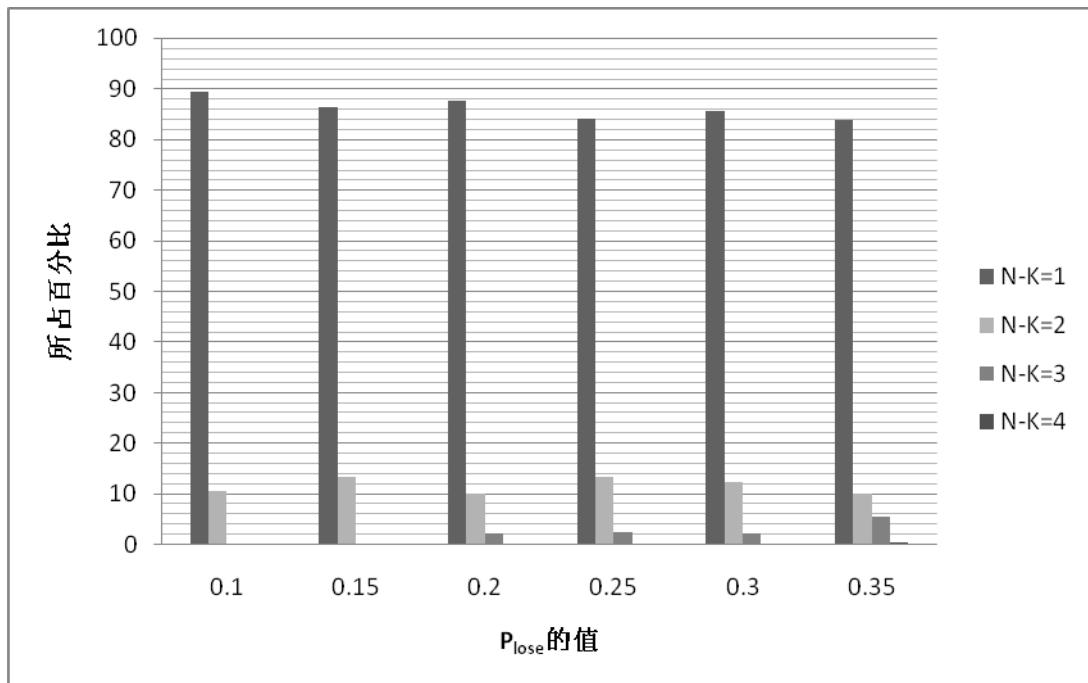


图 4-13 $N-K$ 取值概率

图中由左至右，丢包率分别是 0.1、0.15、0.2、0.25、0.3、0.35。从图中可以看出，虽然丢包率不同，但是，解码失败时， $N-K$ 的值为 1、2、3 的概率的和

几乎是 100%，而 $N-K \geq 3$ 的情况几乎没有。由此得出结论，当接收端接收到的编码符号值 N 符合 $N-K \geq 3$ 时，本系统的 Raptor 码能够保证解码成功。

根据以上统计数据，本系统确定的编码开销值为 3，即 $N-K = 3$ 。

4.8.2 冗余符号个数 R 的确定

确定了解码开销以后，解码端就可以根据丢包率进行 R 值的计算。计算公式如下：

$$R = \text{ceil}((K + 3)/(1 - P_{\text{lose}})) - K \quad (\text{式 4-2})$$

4.9 本章小结

本章介绍了视频多跳传输系统的详细设计和实现细节，分模块展示了各个功能的实现过程以及过程中遇到的问题和解决方法。

本章具体描述了系统 Raptor 码实现过程中的重要步骤，并且通过实验得出了在视频多跳传输系统中的冗余确定方法，完整实现了系统 Raptor 码在系统中的应用。

第五章 系统测试及性能分析

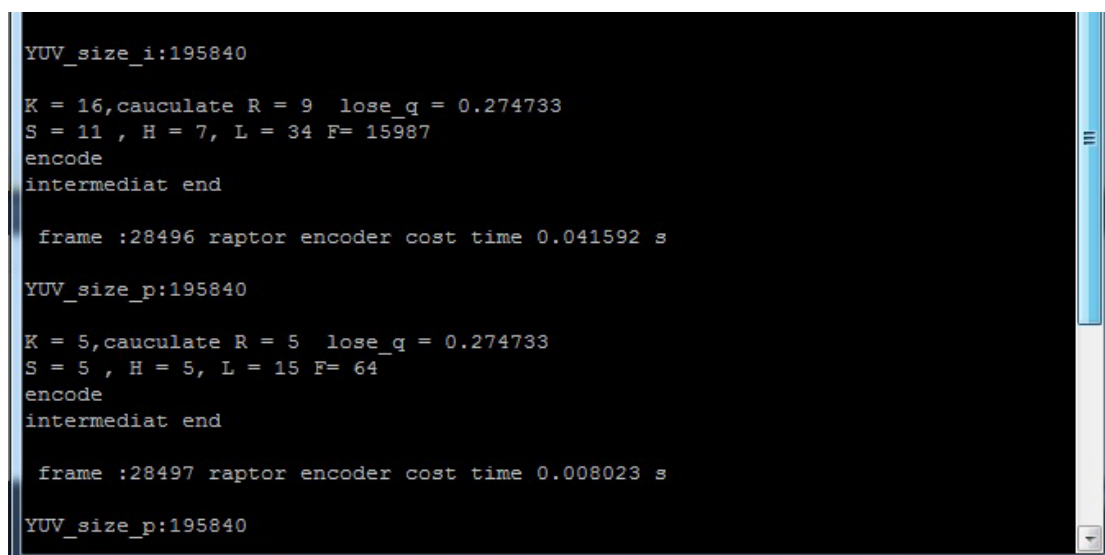
系统测试过程主要包括两个部分，一是展示系统中各模块的运行结果，二是对系统最终的运行效果进行展示。

5.1 系统功能模块运行结果

发送端的运行结果主要是通过将参数的值打印到终端上来进行展示，接收端的运行结果是视频的播放。

5.1.2 发送端的运行结果

发送端运行结果如图 5-1 所示：



```
YUV_size_i:195840
K = 16,cauculate R = 9  lose_q = 0.274733
S = 11 , H = 7, L = 34 F= 15987
encode
intermediat end

frame :28496 raptor encoder cost time 0.041592 s

YUV_size_p:195840
K = 5,cauculate R = 5  lose_q = 0.274733
S = 5 , H = 5, L = 15 F= 64
encode
intermediat end

frame :28497 raptor encoder cost time 0.008023 s

YUV_size_p:195840
```

图 5-1 发送端运行结果

根据图中所展示的参数可以看出以下几点：

(1) YUV_size_i 是采集到的 YUV 格式下 I 帧的数据大小，YUV_size_p 是采集到的 YUV 格式下 P 帧的大小，单位都为字节。在 YUV 格式时，由于图像的大小只与分辨率有关，因此 I 帧和 P 帧的大小是一样的。

(2) 图中 F 的值是 Raptor 编码时的源数据的大小，也就是 YUV 图像经过 H.264 编码后的大小。其中，I 帧编码后的大小为 15987，P 帧编码后的大小为 64。事实上，每个 I 帧或者每个 P 帧 H.264 编码后的大小不是相同的，但是范围波动不大，I 帧和 P 帧经过 H.264 编码后大小相差比较悬殊。

(3) 图中 K、R、S、H、L 的值是系统 Raptor 编码时用到的参数值。

(4) 图中显示了每一帧的编码时间。I 帧编码时间约为 40 毫秒，P 帧编码

时间约为 8 毫秒。

接收端的运行结果如图 5-2 所示：

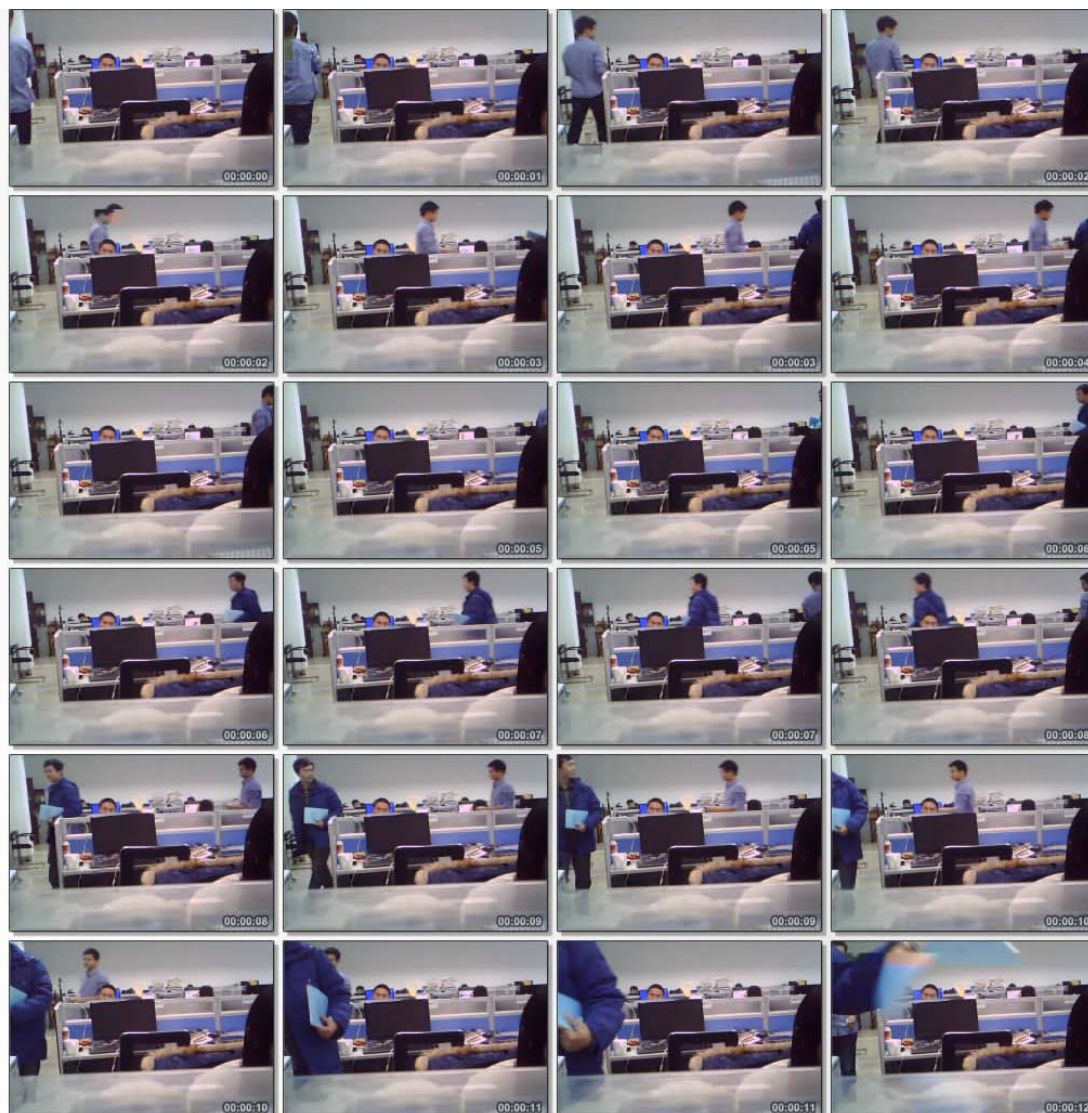


图 5-2 接收端运行结果

以上是视频的截图，展示了连续画面的效果。可以看到，视频画面中偶尔会产生少量的马赛克，对画面质量影响不大，没有图像模糊不清无法辨认的现象，视频比较流畅。

5.2 Raptor 码的应用效果

为了验证本系统对视频传输质量的提高效果，需要对使用系统 Raptor 编码前和使用后的视频传输效果进行对比。为了对比不同丢包率下的效果，本系统采用了文献^[17]中提出的模型，模拟不同的网络丢包情况，丢包率 P_{lose} 的取值范围为 $0.1 \leq P_{\text{lose}} \leq 0.35$ 。测试时，使用有线网络传输视频数据，确保数据包全部从发送端传输至接收端，接收端通过模型控制一定数量的数据包不参与解码，模拟网

络丢包现象。

5.2.1 视频质量的对比

为了更直观显示系统的运行效果，需要将两种情况下传输的视频进行对比。

情况一：发送端直接传输源符号，接收端不进行 Raptor 解码过程，直接将接收到的源符号进行 H.264 解码并展示。

情况二：发送端进行 Raptor 编码，发送源符号和修复符号，解码端根据接收到的符号进行 Raptor 解码、H.264 解码并展示。

为方便对比，界面的上半部分将展示在情况一下显示的视频图像，下半部分将展示在情况二下显示的视频图像。

实际测试时，使用了不同丢包率，分别是 $P_{\text{lose}} = 0.10$; $P_{\text{lose}} = 0.15$; $P_{\text{lose}} = 0.20$; $P_{\text{lose}} = 0.25$; $P_{\text{lose}} = 0.30$; $P_{\text{lose}} = 0.35$ 。测试结果表明，在丢包率不同的网络环境下，系统均能保证良好的视频质量；经过与未经 Raptor 编解码过程的传输情况对比，本系统显著提高了视频的传输质量。

图 5-3 和图 5-4 显示了在实验环境中视频的传输质量对比。

图 5-3 中展示的是摄像头前没有剧烈运动的物体时的情况，可以看出当画面中的物体运动情况不剧烈时，如果不经 Raptor 编解码过程，由于丢包，画面经常会出现一片模糊的情况，这是因为当画面基本静止时，P 帧将会很小，丢包后，这些关键信息大量丢失，通过 H.264 的错误掩盖过程，发生错误的地方被附近的信息掩盖掉。经过 Raptor 编解码的处理，画面基本能够保持清晰。

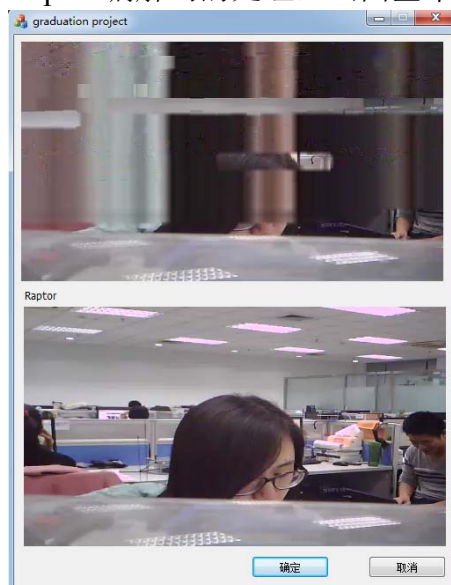


图 5-3 无剧烈运动物体时视频画面对比图

当画面中有剧烈运动的物体时，在情况一下，运动物体经过的部分会出现明显的马赛克，运动物体的轮廓被破坏；而经过 Raptor 编解码处理后，视频画面

能够保持清晰，运动物体的轮廓明显，偶尔才会出现较少的马赛克。如图 5-4 所示：



图 5-4 有剧烈运动物体时视频画面对比图

PSNR 值，即峰值信噪比（PSNR），是一种评价图像的客观标准。PSNR 的值能够反映两个图像之间的差别，PSNR 的值越大，表示两个图像的差别越小，反之，则说明两个图像的差别越大。

计算传输后恢复的视频帧与原视频帧的 PSNR 值，即可看出传输后视频帧与源视频帧的差别大小，差别越大，说明视频帧的质量越差，反之说明质量越好。

将传输后的视频恢复为 YUV 格式并存储，即可使用 YUVviewer 工具计算两个 YUV 视频流的 PSNR 值。图 5-5 给出了在情况一和情况二下所恢复的视频数据与源视频数据的 PSNR 值对比。

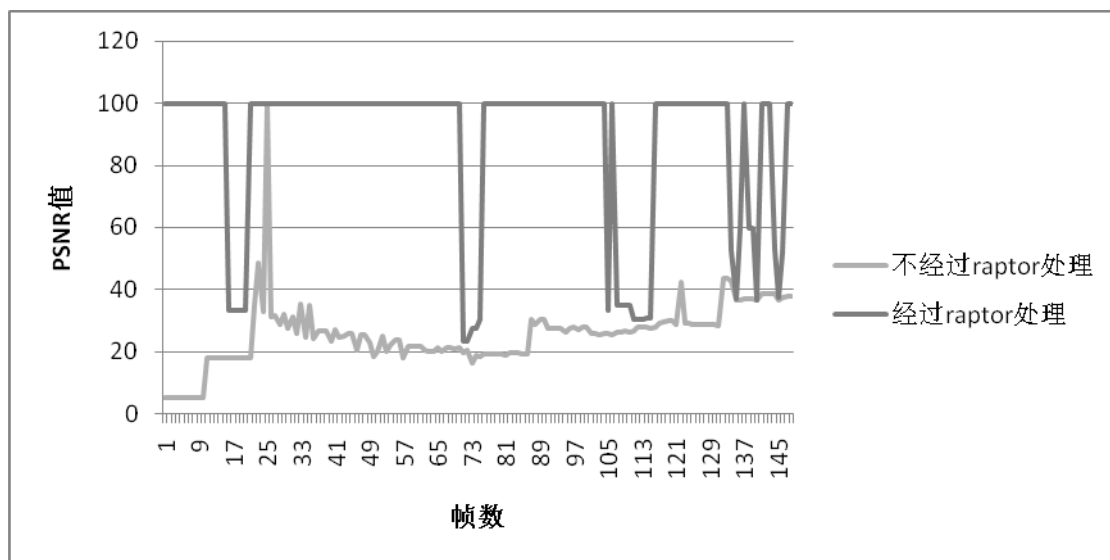


图 5-5 PSNR 值的对比

可以看出，在情况一的条件下，传输后恢复的视频与原视频的差别较大，而在情况二的条件下，传输后视频与原视频的差距明显减小，只在个别情况下 PSNR 值才衰减到与情况一相当，即经过 raptor 编解码后，传输后得到的视频质量较不经过 raptor 编解码的情况明显提高。

所述可以看出，无论是通过人眼的直观感受还是通过 PSNR 值进行衡量，经过系统 raptor 码处理，传输的视频质量得到了明显的提高。

5.2.2 系统的性能

为了进一步明确本系统的解码性能，需要对系统最终的抗丢包率效果进行验证。

在接收端对需要的参数进行统计，得到的源数据如图 5-6 所示：

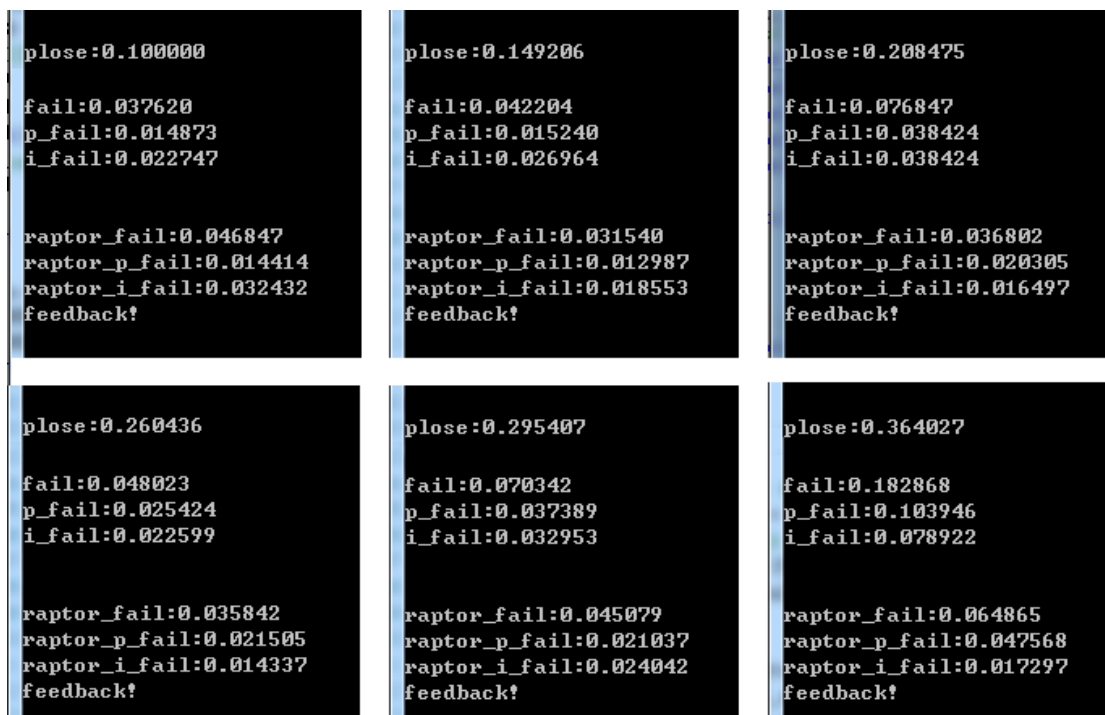


图 5-6 系统性能源数据

图中是六种丢包率情况下统计的丢包率和系统 Raptor 解码成功率。

下面以 Plose = 0.1 的情况为例，解释图中不同数值的意义。

fail 表示解码端没有完整恢复的视频帧个数占所有全部视频帧的比例，计算方法如下：

$$fail = \frac{\text{无法进行Raptor解码的帧数} + \text{Raptor解码失败的帧数}}{\text{总帧数}}$$

图中带有 raptor 前缀的参数体现了 Raptor 解码器的解码成功率，raptor_fail 表示 raptor 解码失败的概率，计算方法如下：

$$raptor_fail = \frac{\text{Raptor解码失败的帧数}}{\text{进行Raptor解码的帧数}}$$

可以看出，fail 值即为本系统视频画面恢复失败的概率，直接体现了视频传输的质量，raptor_fail 值即为本系统实现的系统 raptor 码的解码失败率，直接体现了本系统实现的 Raptor 码的性能。整理数据，结果如图 5-7 所示：

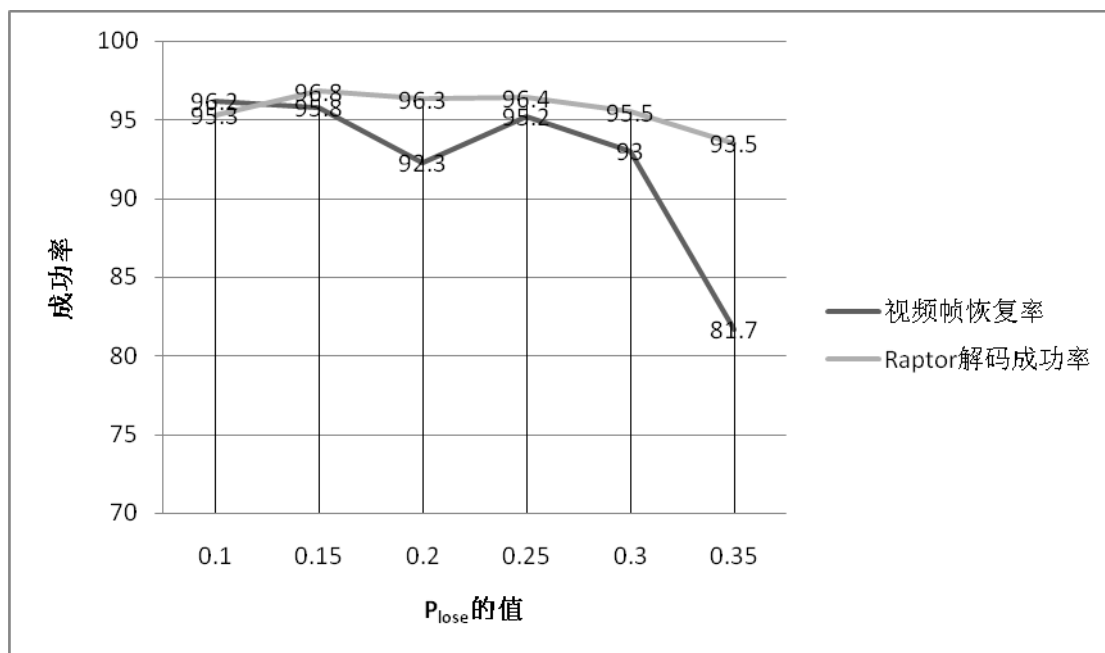


图 5-7 系统性能数据

从图中可以看出在 P_{lose} 的值未超过 0.3 时，系统的性能比较稳定，视频帧的恢复率和 Raptor 解码的成功率都在 90% 以上，当 P_{lose} 的值达到 0.35 时，系统视频帧的恢复率有明显的下降，但是依然保持在 80% 以上，Raptor 解码成功率下降不多，这说明，视频帧的恢复率下降的主要原因是接收到的编码符号个数不足以进行 Raptor 解码。

综上所述，系统的性能能够满足应用环境的要求。

第六章 总结与展望

6.1 工作总结

视频的实时可靠传输是多媒体传感网络的重要研究内容。提高视频多跳传输的质量，对于多媒体传感网络的应用具有重大意义。本文详细设计了一套视频多套传输系统，目的是在系统中应用系统 Raptor 码技术，解决无线多跳网络丢包频繁的问题，实现视频数据的实时可靠传输。

本文首先分析了多媒体传感网络对视频传输的功能需求和性能需求，根据需求分析，提出了系统需要提供的各项功能，主要包括视频数据采集、数据压缩、数据解码、数据传输、数据解码、视频展示。本文还对几种纠删码技术进行了系统的研究，重点分析了系统 Raptor 码的编解码原理、性能优势和应用难点。系统将系统 Raptor 码作为视频数据的编码方案，它能够有效的减小网络丢包对视频传输的影响，提高视频帧的恢复概率，从而提高视频的传输质量。

通过对系统的运行效果测试，证实系统能够稳定、正常的连续工作。使用 gilbert 模型丢包仿真，测试系统在不同丢包率情况下的性能数据，验证了系统能够适应丢包率在 0 到 0.35 范围内的网络状况，高概率的恢复视频帧数据，保证良好的视频传输效果，实现了视频的可靠实时传输。

6.2 展望及改进建议

本文的研究完成了最初的目标，实现了无线多跳自组织网络中的视频实时可靠传输。但是，针对系统 Raptor 码的应用，系统中仍有几项内容留待改进：

（1）对编码过程的继续优化改进。对于本文中的编码方案可以进一步的研究和优化，使之适合更加复杂的网络环境。

（2）冗余的确定。本文中确定 R 值的方法比较基础，使得编码产生的冗余比较大，这显然不利于节省无线多跳网络中的带宽。如果能够通过大量的实验或理论证明在不同网络情况下的最佳冗余值，从而使得系统能够根据网络信道的带宽、丢包率等因素反推出冗余值，则能够进一步节省系统的带宽，获得更好的编码效果。

（3）无反馈。本系统中设计了反馈系统用于确定 R 的值，但是数字喷泉码的一个特点就是码率灵活，充分满足解码的需求，从而无需解码端的反馈，本系

统的设计方法事实上牺牲了这一特点。未来对系统进行改进，最好设计出不需要反馈系统的方案，充分利用数字喷泉码的优势。

(4) 从本文 5.2.2 节图 5-5 中可以看到 I 帧和 P 帧的视频帧恢复概率，可以发现 I 帧的恢复率略低于 P 帧的恢复率，这对视频的传输不利。可以进一步研究影响 I 帧恢复率的因素，提高关键帧的质量。

本系统已经验证了系统 Raptor 码在无线多跳自组织网络中的性能。下一步，对系统 Raptor 码进行更深入的研究，希望能够将其更广泛和灵活的应用于无线多跳自组织网络的领域。

此外，对系统可以进行进一步的功能扩展和标准化，使得系统的应用更具有普遍性。

参考文献

- [1]陈荣梅, 胡诗玮.无线多跳自组网络视频传输研究的综述[J].计算机与现代化, 2009,161(01):97-100.
- [2]李少谦, 兰岚.无线 Ad hoc 网络技术[J].中兴通讯技术, 2002(01):9-12.
- [3] Ren FY, Huang HN, Lin C.Wireless sensor networks[J].Journal of Software , 2003(07)
- [4] 马华东, 陶丹. 多媒体传感器网络及其研究进展[J]. 软件学报, 2006,17(09): 2013-2018.
- [5]罗武胜, 翟永平, 鲁琴.无线多媒体传感器网络研究[J].电子与信息学报, 2008,30(06):1511-1516
- [6] Michael Mitzeumacher. Digital Fountains: A Survey and Look Forward[C].ITW 2004,San Antonio,Texas,October 24-29,2004.
- [7]慕建君, 路成业, 王新梅.关于纠删码的研究与进展[J].电子与信息学报, 2002,24(09):1276-1281.
- [8]郭春梅, 毕学尧.纠删码的分析与研究[J].信息安全与技术, 2010(09):38-42.
- [9]慕建君, 焦晓鹏, 曹训志.数字喷泉码及其应用的研究进展与展望[J].电子学报, 2009,37(07):1571-1577.
- [10]石东新, 杨占昕, 张铨.3GPP MBMS 中 Raptor 编解码研究[J].数据采集与处理, 2010,25(S):121-124.
- [11]Pei Wang, Li Song, Songyu Yu. Analysis and comparison of FEC and FEC-ARQ protection schemes based on RS and Raptor code. Wireless Communications and Signal Processing (WCSP), 2010 International Conference on:1-6.
- [12]Michael Luby.LT Codes[C].Proceedings of the 43rd Annual IEEE Symposium on Foundation of Computer Scienc(FOCS'02).Vancouver,BC,Canada,2002:271-282.
- [13]朱宏鹏, 张更新, 谢智东.喷泉码中 LT 码的次优度分布[J].应用科学学报, 2009,27(01):6-11.
- [14]A Shokrollahi. Raptor codes[J]. IEEE Transactions on Information Theory, 2006, 52(6): 2551—2567.
- [15]M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer. Raptor Forward Error Correction Scheme for Object Delivery[S].Network Wording Group,RFC:5053,2007.
- [16]毕厚杰.新一代视频压缩编码标准——H.264/AVC[M].人民邮电出版社, 2004.

- [17] 兰帆, 张尧弼. 基于 Gilbert 模型的网络丢包仿真[J]. 计算机工程, 2004, 30(S): 200-203.

致谢

转眼之间，两年的硕士研究生学习就要画上句点。在这两年时间里，我做了更多了思考，学了更多的知识，认识了更多的人。这两年虽然也有遗憾，但是更多的是真实的成长。在本文也将画上最终的句号之时，我要对学习、工作和生活帮助过我的人致以谢意。

首先要感谢我的导师，马华东老师。在论文完成的过程中，马老师给予了我严格的督促和悉心指导，他渊博的学识、严谨的治学态度和教导风格都让我难忘。马老师在这两年对我的指导和关心让我受益匪浅，在此对他致以最真诚的谢意。

我还要感谢段鹏瑞老师对我学习工作的关心。段老师对学生的态度平易近人，十分为学生考虑。在工作中，段老师经常与我们积极地讨论，对我的课题研究也给予了启发式的建议和积极的指导。十分感谢段老师对我学习上的帮助。

我也要感谢实验室小组的成员，他们是张莫、杨天昊和刘孟轩。与他们的相处十分愉快，他们让我感受到了团队合作的力量和乐趣。在一起挂板子、做实验的时光我永远不会忘记。

最后我还要感谢与我同住一间宿舍的姐妹，我们在学习和生活上的彼此帮助，彼此关心，她们给了我很大的支持。