Y1802504

独创性声明

本人声明所呈交的学位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。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它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谢意。

签名：张璐　　日期：2010年5月25日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，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，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：张璐　　导师签名：郝玉洁

日期：2010 年 5 月 25日

摘 要

近十五年里因特网普及之迅速，据报道中国的网络用户由 96 年的 10 万增长到现在的 3.38 亿，可见人们已经非常依赖网络这个载体来交换信息和处理事务。同时基于网络的应用也日渐多元化，如：网上购物、聊天、电子邮件、博客等。但是在含有众多应用的网络环境中也存在着各种隐患，网站遭受黑客攻击，用户数据的泄露等事件频繁发生。为了有效地应对复杂的网络环境，各种防范软件相应浮出水面，如：网络监控系统，杀毒软件，防火墙软件等。

无论上述的网络应用还是网络安全领域的各种防范软件都依靠于底层的数据包捕获，可见数据包捕获在高速发展的网络领域中占据的重要地位。传统的数据包捕获过程中的中断处理，协议栈的处理和数据从内核空间到用户空间的拷贝等过程都需要处理器资源。在网络传输速度有很大提升的今天，对数据的实时采集往往由中断频率过高和数据处理速度过慢造成丢包现象，直接影响到基于其工作的系统的性能。如何高效地数据包捕获成为很多学者研究的课题。

本文所研究的数据包捕获方案是以某企业网络监控系统为背景，依靠以太网的广播特性实现的。重点研究的是在高流量的网络环境下对数据包捕获性能的改善。我们首先通过对数据包捕获流程的分析寻找出数据捕获过程中影响其性能的各种因素，然后根据这些因素将目前在数据包捕获领域出现的改进方案进行归类，最后选择零拷贝的思想实现网络监控系统中的底层数据捕获部分，并在文章的最后一部分对此捕获系统进行了性能上的测试。

高效捕获系统的具体实现是基于内核提供给用户空间的内存映射接口去完成的。我们的主要工作有三点：第一，需要修改网卡驱动源码，将数据包放入指定的内核空间中；第二，设计内核共享缓冲区的数据结构，以便实现数据包的填充和释放的同步；第三，提供给后期编程者数据包获取接口。在以后的工作中，我们还要设计一个精简的基于应用层的“网络协议栈”，来完善数据包的处理功能。

关键词：数据包捕获，内存映射，设备驱动，网络监控

ABSTRACT

The spreading speed of internet was fast in the past 15 years, and it was reported that the number of netizen increased 33 million, from 100 thousand in 1996 to 33.8 million in 2008. Now people have been accustomed to exchanging message or dealing with affairs by the carrier of internet. The applications based on internet are various, such as e-commence, chatting, e-mail, blog and so on. However, there are many security risks in the complex network environment. In order to management network environment effectively, some software such as network monitoring system, intrusion detection system, and firewall software appear.

Whether various applications based on internet or network security software mentioned above, they always relay on bottom packet capturing. Network packet capturing plays an important role in the rapid development of network field. Especially, the faster speed of network leads to high frequency of NIC interrupt, low speed of packet processing and packet loss. These factors above can impact on the performance of upper system. Now many scholars research on how to capture packet efficiently.

On the background of Enterprise network monitoring system, we relay on the broadcasting feature of Ethernet to capture packets. How to improving the performance of capture in high-traffic network environment is our focus. First, we introduce normal packet capturing method in Linux, and then model the process of packet capture to find out the factors which impact performance of system. According to these factors, we introduce some improved technology and choose zero-copy to implement our system. At last we build testing environment, and test the performance of the system by different length packets and different transmit speed.

The implementation of efficient packet capturing system is based on a memory mapping interface, which is provided by kernel. Our main work emphases on three important parts. The first one is to modify the source code of NIC driver and put packets into pre-allocated kernel space; the second one is to design data structures which make the communication and synchronization between NIC and user processes; the last part is

to provide a programming interface for application programmers. In the future, we will design a simplified protocol stack working in user space to improve the function of packet processing.

Keywords: Packet Capturing, Memory Mapping, Device Driver, Network Monitoring

目 录

第一章 引 言 1
1.1. 课题背景 1
1.2 .研究现状 2
1.3. 主要工作 3
1.4. 论文特色 4
1.5. 组织结构 4
第二章 提升数据包捕获性能的方法及相关知识的介绍 5
2.1. 数据包捕获流程的简介 5
2.2. 类 UNIX 系统中的数据包捕获机制介绍 6
2.2.1 Linux 中的网络套接字接口 7
2.2.2 BSD 中的 BPF 过滤机制 7
2.2.3 SVR4 中的数据链路提供者接口 8
2.3. 影响数据包捕获性能的因素 9
2.4. 提高数据包捕获性能的方法 11
2.4.1 中断结合 11
2.4.2 中断轮询结合机制 12
2.4.3 零拷贝思想 14
2.4.4 硬件卸载引擎 15
2.4.5 提高 Linux 系统的实时性的方法 16
2.4.6 对改进方法的总结与选择 17
2.5. 实现改进方法所需的基础知识 17
2.5.1 内存映射 17
2.5.2 设备驱动及设备类型 21
2.5.3 进程通信机制 22
2.5.4 模块化的内核编程 24
2.6. 本章小结 25
第三章 基于千兆网卡的数据包捕获系统的概要设计 26

3.1. 企业网络行为监控系统的简介 26
3.1.1 企业网络行为监控系统的功能简介 26
3.1.2 企业网络行为监控系统的架构 28
3.2. 数据包捕获系统的性能需求 30
3.3. 数据包捕获子系统的设计方案 32
3.3.1 数据包捕获子系统的功能模块划分 33
3.3.2 功能模块间的同步 33
3.3.3 用户接口的设计 35
3.4. 本章小结 35
第四章 基于千兆网卡的数据包捕获系统的详细设计与实现 36
4.1. 千兆网卡驱动模块的设计 37
4.1.1 关键数据结构的分析 37
4.1.2 网卡驱动接收流程的分析 39
4.1.3 网络驱动程序的修改 42
4.2. 内核共享缓冲区模块 42
4.2.1 数据结构的建立 43
4.2.2 共享缓冲区中内存的分配 45
4.2.3 网卡与应用程序之间的同步 48
4.3. 接口函数模块 50
4.4. 系统整体工作流程 52
4.4.1 系统初始化阶段流程 52
4.4.2 系统运行阶段流程 53
4.5. 本章小结 55
第五章 基于千兆网卡的数据包捕获系统的测试与分析 56
5.1. 测试环境 56
5.2. 测试数据的分析 57
5.3. 本章小结 59
第六章 总结与展望 60
6.1. 全文总结 60
6.2. 工作展望 60
致 谢 62
参考文献 63

攻硕期间取得的研究成果..... 66

第一章 引 言

1.1. 课题背景

因特网技术的发展可追根溯源到上个世纪六十年代的线路交换技术，后来随着分组交换技术的发展及相应的网络协议和光纤等硬件设施的改善，到上世纪八九十年代因特网网络数目激增并逐步商业化。

根据 CNNIC 第 23 次中国互联网发展报告[1]：中国网民已经接近三亿，普及率超过全球平均水平；网民通过宽带接入互联网的比例已经占到总体网民的 90%。人们已经非常依赖网络这个载体来交换信息和处理事务。基于网络的应用也日渐多元化，如：网页冲浪、网上购物、聊天、电子邮件、博客等等。除此之外，大多数企业也开始进入信息化规划方案，架构自己内部的局域网，并且通过 SOA[2] 理念等，开发一系列的 ERP[3]软件来有效的管理企业。

当网络技术带给我们便利的同时它还带来了很多安全隐患，如：网站遭受黑客攻击，用户数据的泄露等等。据美国《金融时报》报道，在计算机网络环境下入侵事件的发生概率为平均 20 秒/次，超过 1/3 的互联网被攻破，全球范围内每年由于信息系统的脆弱性而导致的经济损失达数亿美元并且呈逐年上升趋势。面对这么多网络安全问题，许多网络入侵检测系统，网络安全监控系统浮出水面。不管是基于网络的各种应用还是网络安全领域的各种防范软件，它们的实现都离不开数据捕获（我们这里介绍的数据捕获都是在以太网下的），可见数据捕获的研究价值。

我国从拨号上网时代从 03 年发展为顶峰，当时用户的有效速率远远低于 56KBPS，到 08 年宽带用户已经占据绝对比例，2008 年中国网络国际出口带宽达到 640,286.67 Mbps。可见网络的发展是伴随着上网配套的硬件设施也不断的改进，最直接表现的就是带宽的上升。随着这样的发展趋势，千兆网络时代的到来并不遥远。

带宽的扩充及网络数据包传输流量变大将会对数据包捕获技术提出一次新的挑战。在百兆网络环境下，linux 常见的数据包捕获接口是 libpcap[4] (windows 下存在相应的 winpcap)，它的工作原理是旁路网卡通过中断传送给系统协议栈的数据

包，将它们存放在两个大小 32kb 的公共缓冲区，应用程序取出缓冲区中的数据进行分析。在此过程数据还需要经历一次用户空间到内核空间的拷贝。但是在千兆网络下，它的捕获性能明显下降，对于小包的成功接收率不到百分之十。近些年千兆网络适配器的出现已解决了硬件上的丢包问题，数据包捕获的瓶颈已不在于网络适配器，而在于操作系统协议栈对包的处理，比如系统接收数据包的中断方式，用户空间与内核空间数据拷贝方式等都是十分耗费系统资源[5]。据相关材料介绍，千兆网络环境下包长为 64 字节的数据包到达速度最大可达到 1230000 个/秒，每个数据包到达网卡产生一次中断，中央处理器必须达到 0.82us 的处理速度才能保证收包率。但是一个 PIII 450MH 装有 linux 2.2.10 的操作系统处理一个数据包需要 13.23us[6]。近些年出现了许多高频率的处理器，但是处理器的频率的线性增加并不会使接收数据包产生的中断消耗线性减小，主要原因还在于像主存、I/O 设备并不具备高速缓存的读写速度，大大限制了中断处理时间的减少。一个主频为 2.4GHZ 的处理器相对于 800MHZ 的处理器性能上只提高了 60%[7]。可见单纯的通过提升处理器的性能并不能根本解决数据包捕获的接收活锁问题。

1.2. 研究现状

这些年，在高性能通信领域出现了很多研究课题，大致可以分为软件实现和硬件实现两部分。在软件领域对数据包捕获性能研究做出最大贡献的还属于 Luca Deri，他开发的 ntop 软件中使用了一种类型为 pf_ring 套接字[8-9]，这种套接字相对于 libpcap 而言，能够使用户进程直接访问到内核空间的数据包缓冲区，避免了数据从内核空间到用户空间的第二次拷贝。这种方式类似于零拷贝[10]但又有不同，它将网卡中断接收的数据先复制到内核缓冲区空间，再将这个区间直接与用户空间进行映射。在后期 Luca Deri 又提出一种 ncap[11]的捕获方案，这种方案让用户进程直接跟网卡进行交互，网卡硬件寄存器和内部缓冲区都直接映射到用户空间。据关于资料报道，这种捕获方式在处理器频率足够高的时候甚至可以达到千兆网卡转发速度的极限值 1.48Mps，但是 ncap 的内部实现机制并未公开。

在硬件领域的改善数据包捕获方案的基本思想是用硬件分担操作系统协议栈对数据包处理的压力，让数据包的处理放到在网卡内部完成，包括我们在第二章将介绍到的协议栈卸载引擎[12]。硬件方式捕获数据包能力很强大，比较有名的 DAG[13]卡和 COMBO 卡，特别是 DAG4.3GE 在发送速率达到千兆每秒时的丢包率为零（包长 1500 字节和 64 字节），但它的价格颇高，达到了几万甚至十万人民币。

由于价格因素，一般的网络管理系统，远程监控系统，入侵检测系统等还是基于千兆网卡的普通硬件。后来生产 DAG 卡的公司使用 ncap 技术开发出了一款新的捕包软件 encap，这款收费软件对硬件的要求不高，但可以很大程度对数据包的捕获能力，而且提供给用户便捷的二次开发环境。

在国内，许多研究者把高性能的数据包捕获研究重点放零拷贝的实现和中断轮询机制的转换两个方面。在零拷贝的实现是直接将数据 DMA 到用户空间或者在内核空间开辟一片缓冲区采用地址映射机制让用户进程访问内核空间。在中断轮询的切换方面，有根据即时网络流量灵活控制的，也有在轮询失败后自动转化的方式。虽然实现的方法不一样，但都是为了减少网络适配器带给中央处理器的中断消耗而建立的。

1.3. 主要工作

本文所研究的数据包捕获是以某企业网络行为监控系统为背景，依靠以太网的广播特性实现的。重点研究的是在高流量的网络环境下对数据包捕获性能的改善。在对 linux 环境下各种网络数据包捕获技术的了解后，通过改变默认情况下数据包的流向和利用内核映像等方法来提高数据包的捕获效率。

本人完成的主要工作如下：

(1)详细地介绍了 linux 系统中数据包捕获机制。

(2)根据 linux 系统下数据包捕获流程建立模型，推导出影响数据包捕获性能两个重要因素（硬中断频率，软中断处理速度）及其关系。并根据这些因素将最近提出的各种改进思想归类分析，选择零拷贝思想实现数据包的捕获。

(3)分析了千兆网卡驱动程序 Broadcom Netxtreme II Gigabit Ethernet Driver 的数据包接收流程和中断机制。

(4)深入了解 linux 虚拟内存的实现机制（分页管理），编写 mmap 系统调用实现用户空间到内存空间的缓冲区映射。

(5)对企业网络监控系统的数据包捕获功能进行了改进，通过在原有系统中添加虚拟设备模块的形式实现了零拷贝思想。

(6)搭建测试平台以完成数据包捕获系统性能上的测试，并对测试结果进行了分析。

1.4. 论文特色

根据企业网络负载情况采用零拷贝的数据捕获方案改善系统的性能。对零拷贝实现主要基于以下几点：

选择在内核态开辟空间，让用户进程通过内存映射对数据包进行读取。

引入虚拟字符设备概念管理内核态的共享内存，建立循环管理队列解决了网卡传入数据与用户进程取出数据的同步，采用互斥信号量方式控制多用户进程对虚拟字符设备的访问。并为后期的数据包处理提供了数据捕获接口。

1.5. 组织结构

本文在分析 linux 内存管理及报文捕获机制的基础上提出了一种新的报文捕获方法，并以内核模块的形式实现底层数据的捕获功能。下面我们对文章的各个章节做一次概括以方便读者阅读。

第一章：简明介绍了本课题的背景，研究现状和主要内容。

第二章：介绍了传统的基于软件的数据包捕获方式，分析了产生数据包捕获瓶颈的关键参数。在此基础上引出了各种的改善的方法，最终选择零拷贝思想实现数据包捕获。最后介绍了实现零拷贝所需要的基础知识。

第三章：介绍了数据捕获系统的大项目背景及数据包捕获系统的高性能要求，最后将数据包捕获系统进行模块划分，讨论了各种模块的交互及实现中需要考虑的问题。

第四章：介绍数据捕获系统的详细设计与实现。

第五章：搭建测试环境，对数据包捕获系统性能进行了一系列测试。

第六章：对整个课题进行总结，提出后期工作的方向和目标。

第二章 提升数据包捕获性能的方法及相关知识的介绍

2.1. 数据包捕获流程的简介

数据包捕获是指从网络设备中取出数据包提交给系统协议栈或者其他相应的模块接口以供其分析及其他用途。从底层网络接入的环境来看，有两种数据包捕获方法。一种是通过路由器对端口进行监听来捕获全部网络的数据包，这种方式适合于很多网络中，但是以太网本身的广播特性给数据包捕获提供了另一个途径。本文讨论的数据包捕获都属于以太网的数据包捕获[14]。

以太网是一种高速的局域网，由于其采用的多路访问协议（CSMA/CD）[15]设计简单，高速率低成本等特性，所以它被普遍使用。以太网的广播性表现为每个主机发送的数据都在信道中传输，处于传输信道中的机器可以获取到信道中的所有数据。这种广播特性可以为 PC 机提供网络数据包捕获的基础条件。网络中的主机系统，数据的收发是由网卡来完成的。以太网卡工作在 TCP/IP 协议模型的数据链路层，通过网卡的 MAC 地址进行定位。

由于以太网的广播特性，以太网卡（在以太网下工作的网卡）的工作模式可以分为两种：混杂模式和非混杂模式。混杂模式就如上一段提到联网的主机可以通过以太网卡接收到流经网络的所有数据帧，并不需要检查这些数据帧是否是发送给本网卡的。非混杂模式下以太网卡只能接收目标地址与自己 MAC 地址一致的数据帧进行处理，其余的数据帧例如广播帧等全部被丢弃。所以在数据捕获过程中我们必须将以太网卡设置为混杂模式。

混杂模式下的以太网卡进行数据捕获时只需要将接收到数据帧通过中断的方式送往内核协议栈，然后我们可以根据上层应用程序的需求对数据包进行物理地址、MAC 地址、端口，或者内容的分析来决定它们是否丢弃还是保留以备后期操作。这个过程其实就是一次数据包的捕获与处理。下面我们具体地了解数据包经过操作系统这一过程。

整个网络体系架构分为五个层次，分别为应用层，传输层，网络层，数据链路层和物理层，这种划分与 OSI 七层体系结构非常类似，根据不同的功能将网络划分层次的方式被广泛的接受。各种操作系统根据这种划分方式设计了类似的协

议栈去处理网络数据。这里我们简单介绍下 linux 网络协议栈[16]的架构和数据包在各层次中会经历哪些处理。Linux 协议栈的框架如下图 2-1 所示。

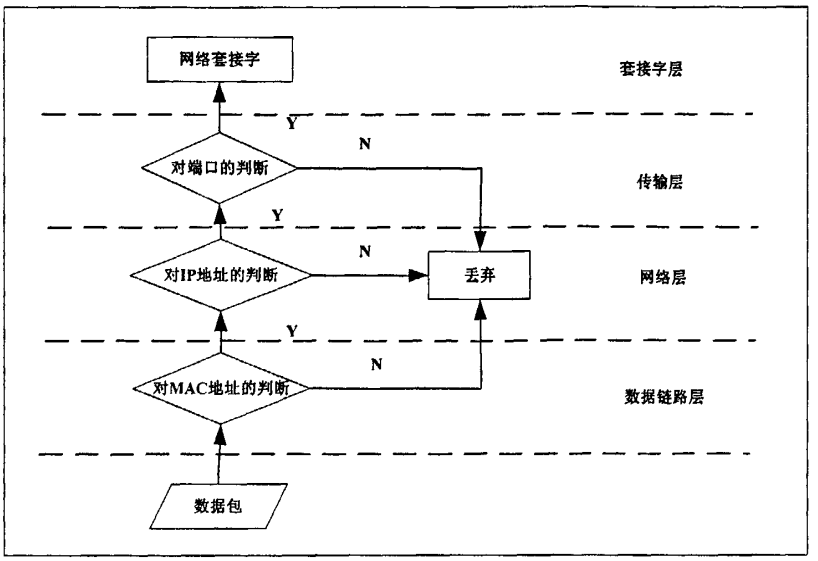

图 2-1 Linux 网络协议栈

数据包通过网卡中断被送入操作系统内核空间，并且在协议栈的等待队列中排队等待软中断处理。软中断的处理就是对接收到的数据包进行逐层解析，将数据帧一步步转化为用户可以读取的消息。数据包在各个层次充当着不同的角色，从物理层的 P-PDU 到链路层的帧，再到网络层的分组和传输层的数据段，最后作为消息到达应用层。协议栈会对链路层的数据帧进行一系列的校验工作，根据帧头的 MAC 地址对数据包进行丢弃、转发或者上传的操作，被上传到网络层的数据包要根据 IP 地址进行路由协议，仍然过传输层要对其端口检查，端口的检查可以看做是对应用进程的匹配，明确数据包发往哪个进程。

准确的说，网络适配器负责完成对链路层帧的操作。在上文中我们了解到网卡可以通过不同的模式决定是否接收所有经过的帧还是只接收与本机有关的数据帧。操作系统的网络协议栈负责完成数据包从网络层到应用层的解析，在解析过程中我们还可以在每层间对数据包进行各种操作，不一定必须要丢弃数据包。此外，网络协议栈对数据包的处理需要一定的时间。

2.2. 类 UNIX 系统中的数据包捕获机制介绍

在本小节中，我们将介绍多种由 UNIX 衍生的各种系统下的捕获机制，所有机制对数据包只是旁路监听，而启不到拦截作用。

2.2.1 Linux 中的网络套接字接口

Linux 系统中提供了多种类型的套接字，比如：链路层套接字 SOCK_PACKET，流式套接字 SOCK_STREAM，数据报套接字 SOCK_DGRAM 和原始套接字 SOCK_RAM。我们常用的套接字是链路层套接字，它是操作系统提供给用户程序访问网络链路层数据包的接口。用户只需设置以太网卡模式为混杂模式，在用户态使用链路层套接字，就可以获得所有由网卡提交给操作系统的数据包。对 SOCK_PACKET 的调用不影响数据包在内核中的流向，但是由于此套接字在内核中没有缓冲区，用户通过它捕获局域网中数据包效率低下在高速网络中无法使用。

2.2.2 BSD 中的 BPF 过滤机制

BPF（BSD Packet Filter）[17]的过滤机制被广泛用于 BSD 系统及起源于 BSD 框架的其他系统中。与 SOCKET_PACKET 一样，BPF 对数据包的获取也在链路层，而且它还能实现数据包过滤的功能。为了理解 BPF 的工作流程，我们需要对网络数据包的默认流向做个介绍：当有数据包到达网卡，网卡会产生中断，操作系统响应中断，将数据包从硬件缓存区复制到内核缓冲区中保存，随后提交到操作系统默认的协议栈进行相应的操作。BPF 在数据包提交到默认的协议栈之前，在网卡中断过程中对数据包进行过滤。它首先会过滤掉目的地址不是本主机的数据包，然后根据过滤规则（规则由调用 BPF 的进程制定）再对数据包进行过滤，经过两次过滤后将最终数据包提交给上层协议栈。具体的过滤机制如下图 2-2 所示。

当网络数据量过大时，BPF 的第一次过滤需要一定的处理时间，占用处理器的资源，可能造成数据包的流入速度大于数据包第一次过滤速度的情况。这种情况最典型的解决办法就是建立缓冲区。其实，在实际应用中，BPF 取消了我们所说的第一次过滤，而是先将所有数据包全部放到内核缓冲区中，然后再根据过滤原则进行过滤，满足条件的数据包再由系统调用传送到用户空间。我们可以看到，起初的数据包没有经过处理全部复制到内核缓冲区中，但用户空间的接收到得数据包数量绝对要小于或者等于内核空间缓冲区数据包的数量，说明通过对内核缓冲区中数据包的过滤还可以减少所需申请的用户空间的大小和数据包从内核空间到用户空间的复制数量。BPF 支持多用户调用的方式是为每个用户进程在内核中开辟一个缓冲区，一个 BPF 对应于一个缓冲区，各自不受影响。当数据过滤完成后，在将此段缓冲区的数据一次性复制到对应的用户空间。根据以上分析，可以

明显得出 BPF 优于 SOCK_PACKET。

系统地总结，BPF 由两部分组成：Network tap 和 Packet filter。Network tap 用于对链路层数据包的捕获，Packet filter 用于对捕获的数据包进行过滤。应用程序通过调用 BPF 可以实现数据包的捕获和过滤。它优于 SOCK_PACKET 的原因有一下几点：

1. 过滤机制的实现在内核空间中，相对于用户空间的过滤，速度快，开销小。

2. Packet filter 使得传递给用户空间的数据包数量减少，减少了内核空间到用户空间传递的资源消耗。

3. 数据包不是一个一个传送给用户空间，而是一次性传递一定数量的数据包，减少了系统调用产生的开销。

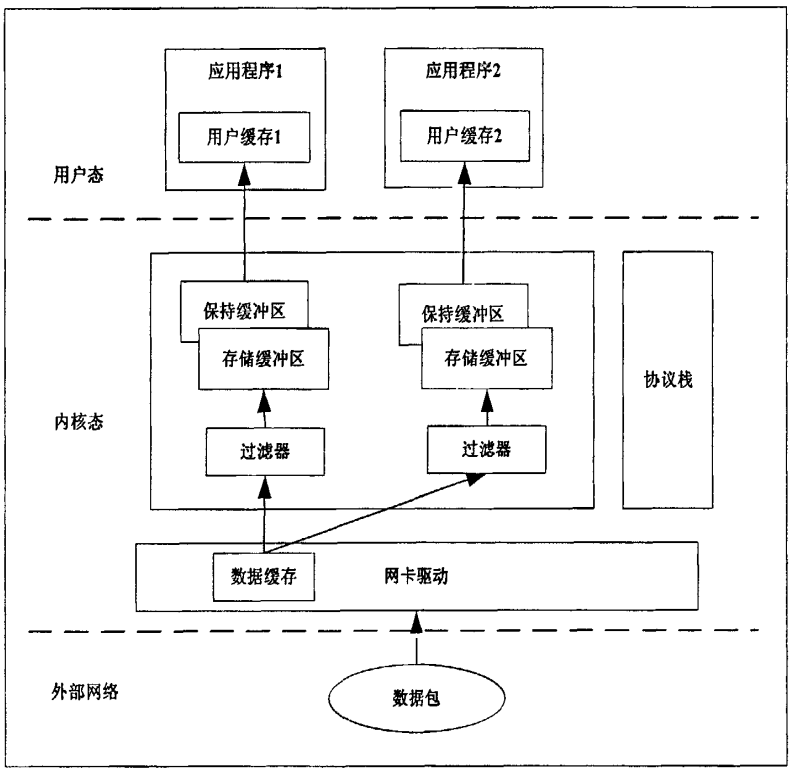

图 2-2 BPF 过滤机制

2.2.3 SVR4 中的数据链路提供者接口

数据链路提供者接口 DLPI[18]是由 AT&T 设计出来的，它有良好的独立于协议的特性，不论数据链路层是基于以太网还是基于令牌环网的协议。此接口的实现是基于 UNIX 的流机制。接口的使用者（上层协议或者用户进程）在不知道链路层协议状态的情况下可以通过此接口直接访问数据包/帧。使用者和服务提供者可

以通过定义的原语集合和规则进行交互。

由于流消息的形式影响系统效率，设计者采取类似于 BPF 的数据捕获和过滤机制，引入 pfmod 和 bufmod 两个模块[19]。 前者对数据包分组过滤，后者将过滤好的数据包复制到用户空间。与 BPF 不同的是，pfmod 模块的实现是通过虚拟机完成，这样避免了数据包拷贝的开销；在 bufmod 模块中引入了超时机制和捕获长度机制等来减少系统调用次数和内存拷贝数据量。但是基于底层协议为 IEEE802 的通信服务，还是要比 BPF 慢。DLPI 具体的模块关系如下图 2-3 所示。

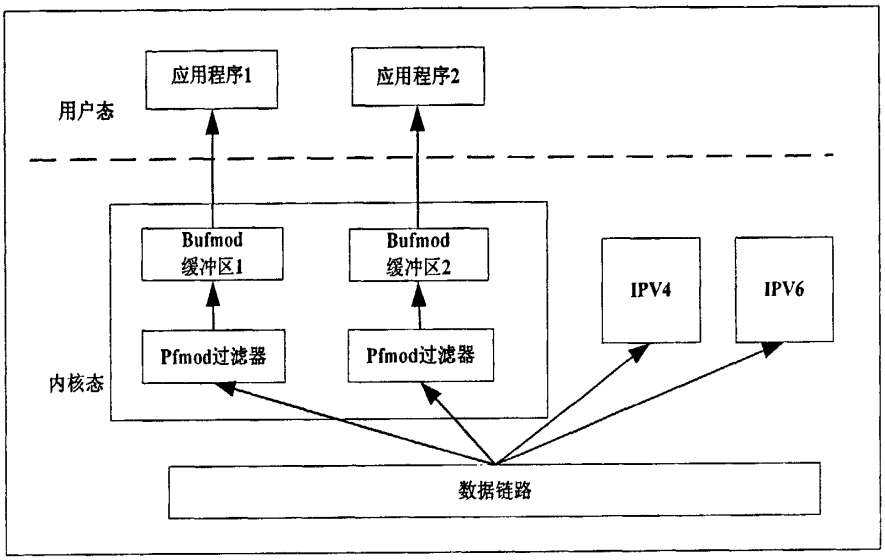

图 2-3 DLPI 接口

2.3. 影响数据包捕获性能的因素

传统的 Linux 报文捕获[20]可以分为两部分：硬中断和软中断。当报文到达网络适配器，适配器将利用中断的方式通知操作系统，系统调用已经被注册的中断处理函数进行接收，把数据包处理成一定格式放到一个由 backlog 指针指向的链表中等待上层传输。在此期间，系统处于关中断阶段，我们将这个阶段称为硬中断。捕获流程的第二部分就是将 backlog 队列中的数据包通过软中断的方式向上层提交。软中断过程中，系统处于开中断状态，当有新的数据包到来时候，系统则会停止软中断过程，回到第一部分的硬中断阶段，将链路层数据包接收到 backlog 队列。具体的流程如下图 2-4 所示。

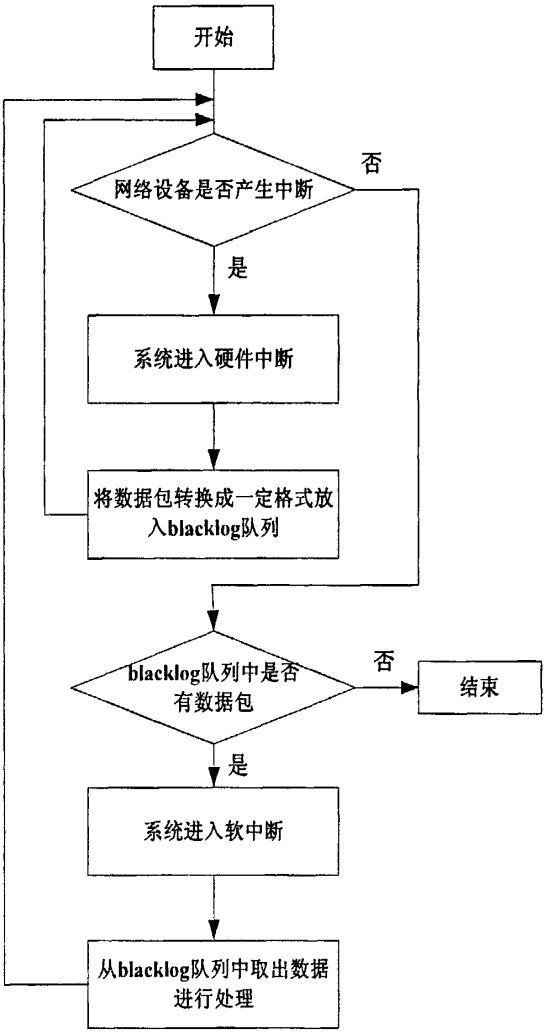

图 2-4　Linux 数据捕获流程图

下面我们从两个方面来分析报文捕获的瓶颈，第一个方面是数据包接收存在不同步性。假设软每次中断处理时间为t_{rp}，软中断产生的时间间隔为t_{rw}，硬中断产生的时间间隔t_{yw}，理想情况下：

$$t_{yw} \geq t_{rp} + t_{rw} \tag{2-1}$$

在低速网络下，t_{yw}较大，所以容易满足不等式（2-1）。在高速网络下，每秒钟到达网络设备的数据包增多，故每秒产生中断的次数增多，t_{yw}变小，此时t_{rw}足够小才能满足公式（2-1），否则 blacklog 队列的输入速率将大于输出速率，造成其拥塞即甚至丢包。

第二个方面是软中断的处理速率对整个捕获机制性能的影响。根据排队论的观点，我们假设每秒钟到达的数据包服从参数为λ的泊松分布，轮询设备处理速度服从参数为u_2的指数分布，每秒钟 blacklog 队列中排队的平均数据包量为L，具体模型如图 2-5 所示。

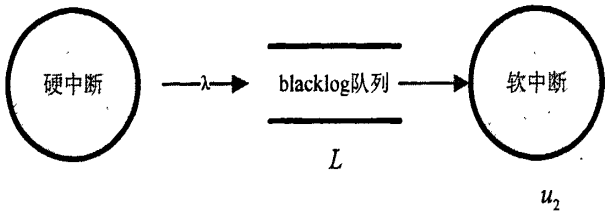

图 2-5 中断机制模型分析

在软中断阶段，为假设λ为常量，我们研究下以u_2为变量的函数的单调性。

$$L(u_2)=\frac{\lambda^2}{u_2(u_2-\lambda)} \tag{2-2}$$

我们将（2-2）以u_2为变量求导，得出：

$$L(u_2)=-\frac{2}{u_2} \tag{2-3}$$

从等式（2-3）可以看出：由于u_2大于 0，函数单调递减，故只有增大u_2才能减小 blacklog 队列中数据包的数量。

2.4. 提高数据包捕获性能的方法

通过 2.3 小节的分析可以知道，我们可以从提升软中断处理速度或者减少中断次数两个方面来提高数据包的捕获性能。下面我们将对从这两方面对现有的一些改进性能方法进行介绍，并引出本文采用的零拷贝思想。

2.4.1 中断结合

中断结合（interrupt-coalesing）[21]是千兆网络下减少中断的开销的常见方法。近些年，许多网络适配器包括以太网卡都实现了中断结合，而且许多操作系统，比如 Linux, windows，也支持中断结合技术。与传统的中断模式相比较，中断结合将一个个到来的数据包归组然后产生一次中断，这种中断模式适合接收包长较小

的数据包。当以太网卡在中断结合模式下工作时，可以根据接收到数据包的数量或者一定时间来产生中断，数据包的个数和时间间隔都可以修改。

虽然中断结合技术可以明显地减少中断开销，但是它会增加数据包的平均延迟时间。对有效的中断结合参数的设定需要权衡数据包的延迟和处理器的处理性能两个方面。当网络流量较低时，通过延时中断并不能改善性能，因为数据包的传输或接收速率不高，不会频繁产生中断，这种情况下采用减少时间间隔指标或者数据包个数指标可以缓解数据包的延时。从另一方面考虑，选择缩短时间间隔的方式有可能导致在数据包还没有到来时就产生中断，这样又浪费了处理器资源。当网络流量过大的情况下，增大时间间隔指标或数据包个数指标可以有效的减小中断开销和提升处理性能。

2.4.2 中断轮询结合机制

通过 2.3 小节的分析可以看到只有当数据包处理时间小于中断时间的时候，才能达到理想的捕获性能。在千兆网络下中断的频繁产生，t_{yw} 变小，给后期的处理带来巨大压力。为了改善数据包的处理性能，研究者们在对中断的开启或关闭机制上做了一系列探索，一种基于半轮询机制（NAPI）[22]的数据包接收方式随后被提出。

在单纯的中断关闭和开启方案中，中断关闭发生在内核协议栈队列不为空并且处理器需要处理协议栈中的数据包的时候，中断开启发生在协议栈中的处理队列为空时。由于处理器对协议栈的数据包进行处理的时候是网卡中断关闭的时候，所以处理器不会受到新流入的数据包的产生的中断干扰，避免了接收活锁现象的发生。

单纯的轮询机制是通过关闭数据接收中断来彻底减少中断开销。在此种机制下，操作系统每间隔一段时间对内存缓冲区中的数据包进行处理，但是这种彻底的关闭中断方式难以实现。当有大量数据包需要处理的时候，我们常用规定一次轮询过程处理数据包的个数，使其有时间预留给用户态应用程序。单纯轮询机制有两大缺陷：第一，我们不能确定每时每刻内存中都有数据包，当没有数据包流入的时候造成了处理器资源的浪费；第二，不能保证对最近流入的数据包进行及时的处理。轮询时间间隔的选择是非常关键的，当间隔过小，造成频繁的轮询，会影响系统的性能造成额外的开销，而当间隔过大，数据包又不能得到及时的处理，造成延时。

NAPI 已经被用于一些操作系统，比如 linux，FreeBSD 等。图 2-6 为 linux 系统中半轮询机制的实现流程。它主要用来减轻接收网络数据产生的中断开销。

在 FreeBSD4.6 和 Linux2.6 下都可以配置轮询模式。他们都是引入配额概念来实现轮询，配额代表每个设备每次可被处理数据包的最大个数。但是在某些方面还是有不同之处。前者在轮询模式下，接收数据引起的中断全部关闭，在轮询期间，一定配额的数据包会被送到协议栈进行处理。在这种情况下，当被轮询的设备数据包被逐个送入循环队列进行处理完成后，设备内部已经没有排队数据包，过一个时间间隔又产生一次轮询，这样将浪费处理器的资源。

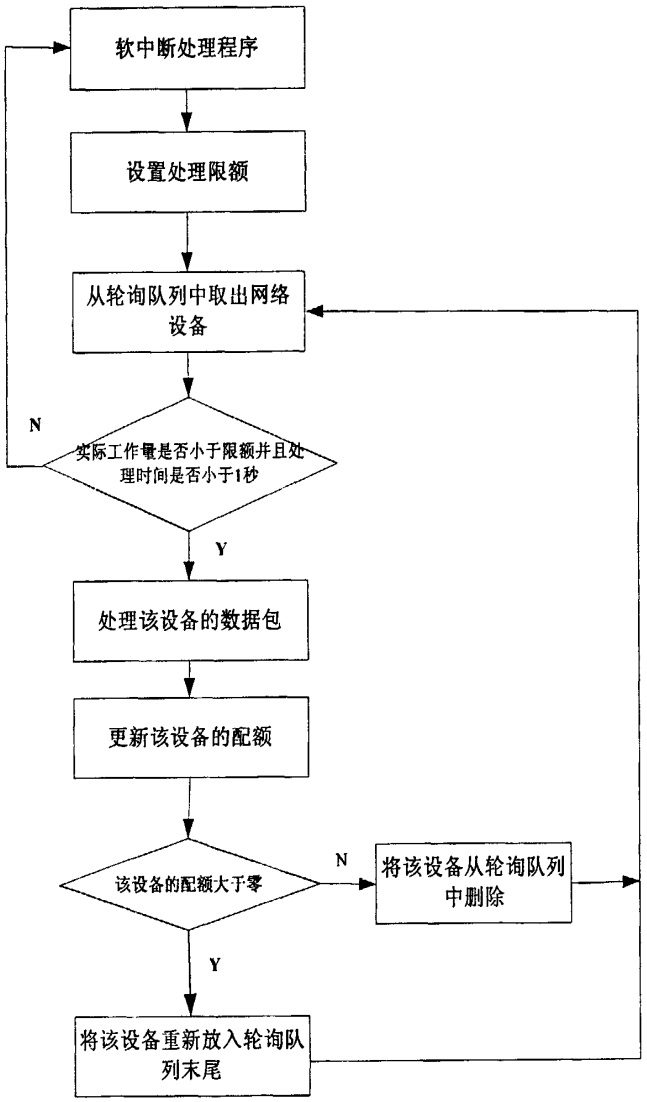

图 2-6 NAPI 轮询机制流程

我们常提到的 NAPI 轮询属于 linux 下的实现方式，为了保证操作系统内部每

个设备能够公平的享用处理器资源，除了引入配额概念，还引入了限额概念。限额代表每次处理数据包的最大个数。假设一批数据包中的第一包到达，网络设备采用中断方式通知系统，系统将进入硬中断阶段，把该设备注册到一个轮询队列。此后，操作系统进入软中断阶段，根据轮询队列对设备进行轮询。在总限额大于零的情况下，当设备的配额大于零且协议栈的等待队列还中仍然有数据包未被处理完，轮询机制将在下一个阶段被再次触发。当某个设备端口发送的数据包被处理完成后，此设备上的配额会实时的减少。上图 2-6 中进行了详细的描述。

Linux 下的 NAPI 轮询机制在网络负荷较大的情况下，趋近于轮询机制，网络负荷轻松的情况下趋于中断机制。所以它利于减少包长度较大数据包下的中断开销，但对长度较小的数据包处理不够及时。

2.4.3 零拷贝思想

在 2.3.小节我们讨论过：提高u_2能促进 Linux 的报文捕获能力。在此小节中，我们将介绍零拷贝机制，它避免了多次拷贝造成的资源消耗，提高了报文的处理速度。传统的 Linux 拷贝是指网络数据从设备缓存区拷贝到内核空间，再从内核空间拷贝到用户空间。为了消除这些不必要的数据拷贝和系统调用，缩短网络报文的传输路径，出现了零拷贝机制，它的基本思想是：利用 DMA 技术将网络数据包直接存储到内核空间，这样可以减少 CPU 的参与，然后利用区域映射技术将内核空间的数据映射到用户空间[23-25]。Linux 零拷贝机制如下图 2-7 所示。

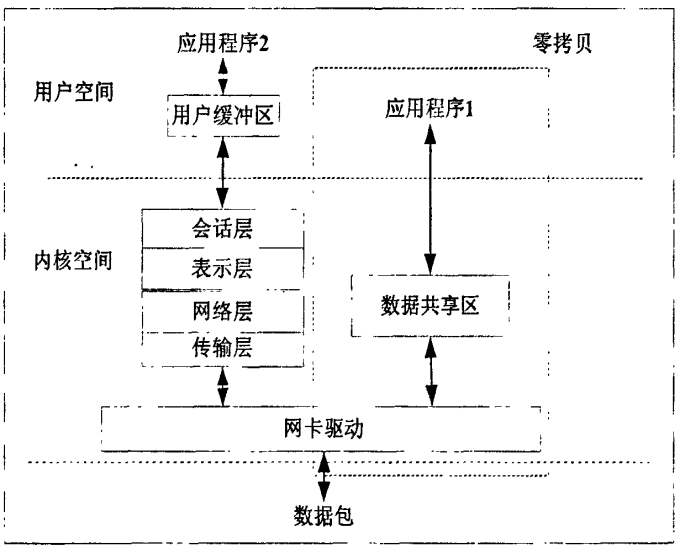

图 2-7　零拷贝流程与传统流程的对比

显然零拷贝的实现需要用到内存映射，还需要考虑到用户进程与设备驱动的同步及多个用户进程间对内存区域的互斥访问。它突破了一贯的内核空间和用户空间的界限，使得用户进程能够访问内核空间。内核空间有自己的一套管理体制，包括内存管理，进程同步，中断处理等。所谓的多进程多任务的实现都依靠着操作系统的管理，在有保护机制的内核区域来实现。当内核区域被我们划分出去，我们应当维护好其同步性。避免多个应用程序对此共享区域进行无序的操作。这些内容我们将在第三章节及第四章节的实现部分来详细分析。

2.4.4 硬件卸载引擎

上述小节介绍的方法都是软件方法，对报文的捕获我们可以使用专门的硬件来完成，相当于软件方法，硬件的可移植性和价格高昂是它的一个劣势，下面我们引入 TOE(TCP/IP Offload Engine)，它被称为 TCP 卸载引擎，这种技术属于用硬件的方式来提升数据包的捕获能力。正如我们曾经提到的，操作系统对网络数据包的处理速度跟不上数据包的流入速度主要是因为操作系统更善于处理高速计算等事务，而不是去频繁的处理中断，系统调用，数据复制等等慢速的事务。在网络数据包接收过程中，存在着大量的中断，数据拷贝，例如：

当网络数据包的流入速度变大，网卡产生的硬件中断次数也随之上升，这样造成中央处理器处理忙于处理硬件中断，而忽略了将接收到的数据包逐个处理的软中断。

在软中断处理过程中，网络数据包需要从协议栈的最底层逐层往最高层送入处理，处理时间过长也将会引起内核缓冲区满溢造成丢包；有数据表明，处理一个比特的数据将会耗费 1HZ 的处理器资源，设想在千兆网络下，处理器的负荷将会达到无法接受的地步。

当数据经过协议栈到用户态，还需要经历数据拷贝过程，数据的拷贝对操作系统占用率很大。传统的协议栈是个庞大的网络数据包处理体系，TOE 技术的核心是通过硬件网卡减轻了操作系统的负荷，将协议栈处理部分直接交给网卡内部芯片实现。TOE 网卡与传统网卡的区别在图 2-8 中可以形象的体现出来。

TCP 卸载引擎可以选择对整个协议栈功能进行卸载，也可以对部分协议栈功能进行卸载，其最终目的都是为了分担系统独占资源的压力[26]。我们知道在位于传输层的 TCP 协议提供了可靠数据传输，然而可靠的传输的实现必然会牺牲掉一大部分系统独占资源去维持，其提供的拥塞控制，流量控制等最为影响数据包的

在协议栈中的处理时间。TCP 卸载引擎可以将这些操作放到特有的网络芯片中实现，这就是对部分协议栈功能进行卸载。在部分卸载中，网卡只需要与操作系统进行交互，对于协议的建立和维护，内核空间数据的分配都由操作系统完成，这种实现方式较简单。

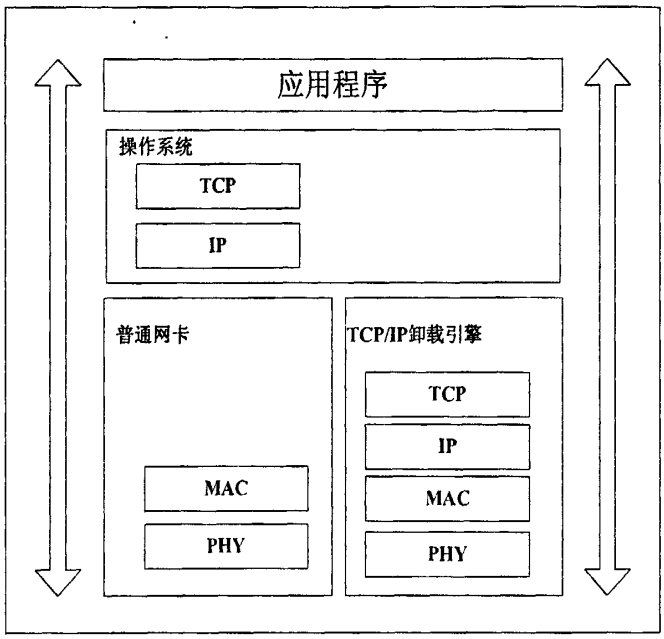

图 2-8 TOE 网卡与传统网卡的对比

对网络协议栈的功能全部卸载就更容易理解，TOE 网卡要完成操作系统协议栈完成的所有工作。全部卸载实现起来比较困难，网卡直接与应用程序进行通讯，自己维护各种连接，还要实现内核空间与地址空间的映射。应用程序一般通过套接字缓存接口获取网络数据包，为了使应用程序能与 TOE 网卡相互通信，两者中间需要再添加一个层次，将套接字缓存接口转化为 TOE 网卡能够识别的数据结构。

目前根据这种技术设计成型的芯片有很多种类，一般这些芯片安装在高流量的服务器上，对于专用的网络负荷卸载卡比上述提到的各种软件方法的效果要显著，同时他的价格也不菲。

2.4.5 提高 Linux 系统的实时性的方法

Linux 是一个内核源代码开放、提供较完善的开发环境和强大的网络支持的低成本操作系统,但是在实时性能方面有先天的缺陷[27-29]。若系统能够更好的支持软实时，比如最小化关中断时间、优化调度策略（设备轮询）[30]等，对网络报文捕

获效率会有所改善。我们曾发现一个问题：CPU、内存有很大空闲的时候，系统仍然产生大量的丢包现象。原因在于当应用程序要处理数据包的时候，NAPI 机制下的 CPU 要定时地轮询网络设备或者其他任务，所以寻找一个适当的轮询频率，及优化任务调度策略是一个不仅涉及报文捕获，甚至整个系统工作性能的关键。研究者们已尝试很多方法来改进 Linux，比如 Gregory Haskin 提出的优化调度器、Paul Mckenney 对 RCU 的改进和自愿抢占内核算法等[31]。目前，最易于开发者的方式还是使用现成的 RT 补丁[32]。在内核版本 2.4X 以上的 Linux 系统下，存在 RTIRQ、RTAI 等补丁，这些补丁可以使内核具备硬件级实时中断。

2.4.6 对改进方法的总结与选择

在 2.4 小节中我们介绍了各种改进方法，主要是三个方面：减少硬件中断的次数，提高软中断的处理速度和通过有效调度提高处理器的使用率。

中断结合和 NAPI 机制的本质都是减少硬件中断次数，中断结合是 NAPI 机制实现的一个基础，目前的千兆网卡在硬件上都支持它们，我们不打算在其实现上进行过多的研究。

零拷贝思想和硬件卸载引擎都是提高软中断处理速度的方法，硬件卸载引擎主要依靠硬件提升处理速度，而硬件加速的价位较高，并不是我们研究的初衷。所以零拷贝思想是我们研究的一个重点。本文的主要工作就是在企业网络行为监控系统的底层捕获中实现零拷贝的思想。

2.5. 实现改进方法所需的基础知识

在上一小节中我们选择了零拷贝思想来提高数据捕获部分的性能，故本节将首先介绍 linux 下虚拟内存的实现机制，以便在后续章节中理解 mmap 的具体实现。其次讨论 linux 下的设备驱动及各类设备的特点，有助于理解后续章节中的虚拟设备模块建立。

2.5.1 内存映射

理论上在一个程序的进程执行时，它需要将整个程序全部装入内存中，直到进程运行完毕再全部换出。其实一次性装入的方法并不一定可取，我们发现在程序运行阶段一些数据并不是频繁地被用到，有的甚至只用到一次。它们对内存的

占据有可能导致其他需要运行的进程因为内存满溢而无法运行，这样照成了大量内存空间的浪费。

在上述情况下诞生了一种虚拟内存思想，将存放一个进程所需要的代码和数据的全部空间以页面或者段落形式划分，运行时候仅把当前需要使用的页面或者段装入内存中，若进程在后续阶段请求的段或者页面不在内存中，则采取换入换出的机制将它们重新换入内存中。虚拟内存的实现是建立在分页或分段的基础上的，对以往的内存线性编制的机制做了一次突破。Linux 下采用分页管理实现虚拟内存，下面我们将详细的介绍这种方式管理下的内存地址的变化。

在虚拟内存管理系统中，进程运行时候有虚拟地址空间和物理地址空间两个概念，它们之间的对应关系放在页表中。用户程序的虚拟内存空间和物理内存空间被我们划分成若干个大小一致的物理块，虚拟内存空间中我们将物理块称为页，在物理内存空间中我们将块称为页框。虚拟内存的实现就是将指定页面装入相应的页框中，在必要时候又可以将相应的页面从页框中换出。

为了让进程在运行时能找到需要的页面的物理地址，系统为每个进程建立一张页表。它是虚拟地址空间的某一页到物理地址空间的某一个页框的映射关系。虚拟地址的被划分为两个部分：页面首地址和页内偏移。我们通过页表查找到页内首地址然后与页内地址相加即为物理地址。在 Intel X86 平台上 Linux 系统定义每一页的大小为 4KB，采用的是三级页式管理，如图 2-9 所示。

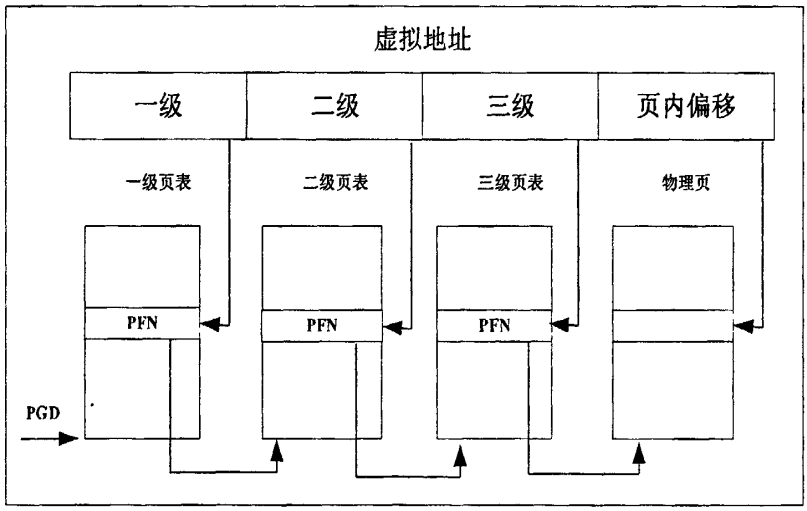

图 2-9　Linux 下的页表结构

处理器根据一级页表的入口地址(PGD)和虚拟地址中一级区域的偏移量(PFN)读取二级页表的入口地址，再根据虚拟地址中二级页表的偏移量读取页表的入口

地址，最后根据页内偏移量读取其中的数据。

Linux 把物理内存分为两个区域[33]：内核空间和用户空间，这两个空间的虚拟地址与物理地址的实现不一样。由于内核空间运行着系统的核心进程，这部分进程永驻与内核中并且所占据的页面永远不会被换入换出。用户空间中运行的进程存在着页面的换入换出操作，所以涉及到其地址空间的分配将比较繁琐。传统的 linux 虚拟内存空间为 4GB，其中 1GB 的高端空间供内核进程使用，其他 3GB 为用户进程空间所使用。

内核空间中虚拟地址到物理地址的转换是通过简单的线性映射来实现的，他们之间的位移量为 0XC0000000。Linux 内核定义的宏__pa()和__va()就是用于虚拟地址与物理地址的转换。内核空间的地址分配都是基于页面为单位的，动态分配的方式有两种，一种使用 kmalloc()进行线性分配，另一种使用 vmalloc()进行非线性分配。两种分配内核空间的函数都是基于 get_free_page()的，此函数能以页为单位来分配物理内存。两者分配机制对比如下图 2-10 所示：

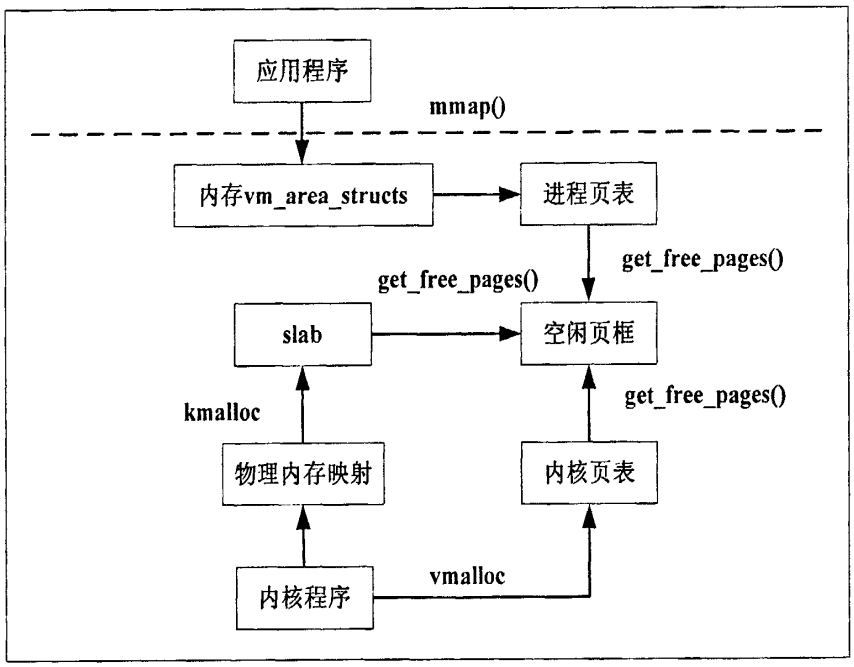

图 2-10 内核空间的分配内存的两种方式

当需要分配的内核空间过大时，选择 kmalloc()实现有可能不成功，因为它的分配的是一片线性连续的空间，内核空间有可能没有那么大的连续空间而只存在小块的非连续空间，这个时候更适合用 vmalloc()实现大空间的分配，它通过内核页表机制将不连续的各个物理页面组织在一起形成逻辑上连续的虚拟空间。相对

于 vmalloc()，kmalloc()的优势是分配的页面管理起来简单化。对内核空间的释放我们使用相应的 vfree()和 kfree()实现。

在图 2-10 中我们还可以看到一个数据结构 vm_area_struct，它是实现内存映射的关键。Virtual memory area（VMA）[34]管理内核空间中的虚拟地址区域，这片区域存放着进程运行时的多个数据区和可执行代码等。vm_area_struct 中包含了 VMA 区域所覆盖的虚拟地址范围和被映射的用户空间的文件区域的物理地址和一系列关于此区域的操作函数。为了让用户空间的进程访问到内核空间中的共享区域，系统可以创建一个新的 VMA 区域并初始化。VMA 与虚拟地址和物理地址之间的关系如下图 2-11 所示。

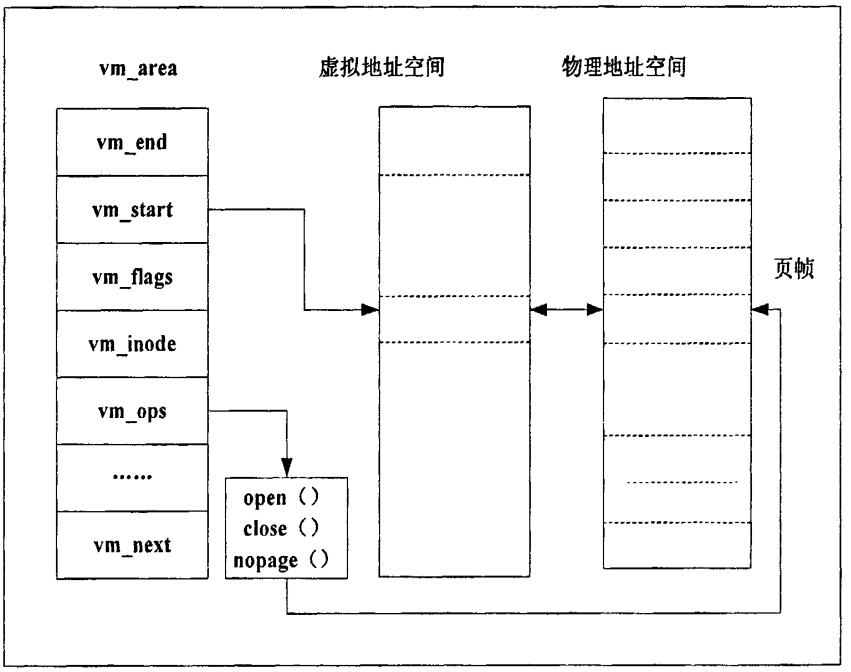

图 2-11　联系虚拟地址与物理地址的 VMA 结构

Linux 操作系统中的内存映射机制是其一大特色，它主要用于用户进程和各种设备的直接交互。在下一小节我们介绍了设备及设备驱动程序的概念，可以看到每个设备可以通过驱动程序注册一定的方法，然后用户进程通过操作系统提供的接口，以系统调用的方式使用驱动程序注册的方法与设备交互。内存映射机制 mmap 方法[35-36]就是设备驱动程序提供的方法而且它是基于 VMA 来实现的。当用户进程将自身空间通过 mmap 方法映射内核空间后，对用户空间的操作实际是对内核空间的直接操作。mmap 方法的实质就是为用户空间创建一个页表。

内核提供了两种建立页表的操作，我们可以选择一次性建立页表也可以动态

地建立页表。remap_pfn_range()操作是一次性将一段长度固定的内核区间到用户区间的映射建立好。nopage()操作用于用户访问 VMA 中页面产生缺页中断的时候，它为产生缺页的虚拟地址重新查找正确的 page 结构，更新映射关系。

2.5.2 设备驱动及设备类型

操作系统提供了一系列的接口实现用户对设备的操作，至于接口内部的实现就由设备驱动程序[37]来完成。故设备驱动程序是内核的一部分，它的主要职责就是管理设备：对设备的初始化及读写释放等操作，保障操作系统与设备之间的数据传输，对异常情况及时的检测。

设备驱动例程不是每时每刻都被调用的，用户进程通过系统调用对设备进行读写或其他操作时，操作系统才会执行设备驱动例程让设备驱动完成了对设备的各种操作的实现。硬件设备也可以通过中断的方式通知操作系统，使操作系统在中断中调用设备驱动例程。

Linux 系统中的设备可以分字符设备，块设备，网络设备这三类。对于字符设备和块设备我们可以将它们作为普通的文件操作，唯一的不同之处在于不同类型的设备对应的设备文件类型不一样。系统在完成对设备的初始化后应用进程通过 open 操作可以获得该设备的设备描述符（相当于 windows 下的句柄），获得设备文件的描述符后可以对这个设备进行一系列的操作。网络设备是一类特殊的设备，应用进程通过套接字接口获取网络设备传输的数据包。关于驱动程序如何在操作系统中被调用如下图 2-12 所示。

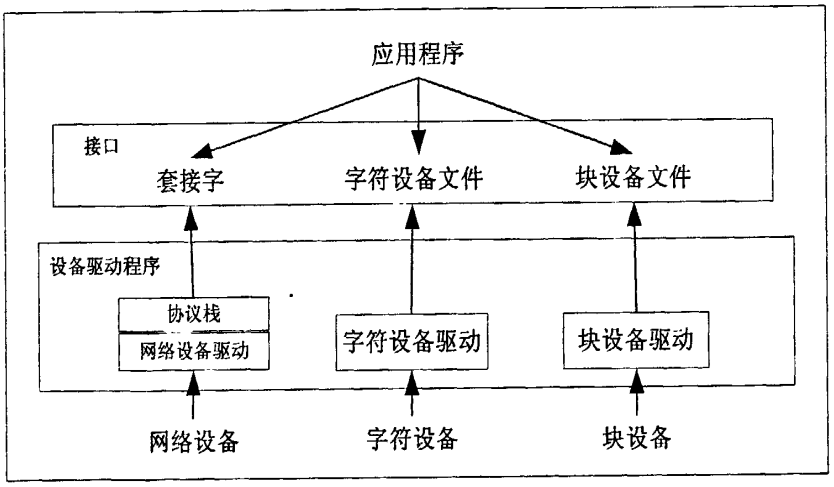

图 2-12 Linux 下设备驱动程序与接口

一般 Linux 设备驱动程序需要完成的工作有对设备的注册，打开，读写，释放

等。对设备的注册时在最设备驱动程序初始化阶段必须完成的一项任务，因为操作系统并不知道有什么设备可用，即使设备是准备好的。我们需要通过注册的方式通知操作系统该设备可用。操作系统中维护着一个用于管理设备的队列，每次设备驱动程序初始化后将被加入这个设备管理队列中。

对于字符设备和块设备而言，它们都将被分配到设备号。设备号可以看作是设备的标识，分为主设备号和从设备号两种，前者主要用于对应每个设备的驱动程序，后者表示与文件相关的设备入口。在实现过程中，驱动程序和设备号的并不是一一对映的，许多驱动程序使用一个设备号的现象常常出现，所以我们也可以把它理解为同类型的设备共享一个主设备号，主设备号只是用来表示设备的类型。块设备和字符设备驱动的唯一标识符被放到次设备号中。

设备驱动完成的注册设备工作应放到其入口函数__init()中完成，对设备进行注销应在其出口函数__exit()中完成。

字符设备和块设备在内核中都存在设备文件的定义，上图 2-12 中可以看用户进程访问设备的形式就如同访问文件，其实设备并不在文件系统中占用空间，只是在设备定义时同时定义一系列文件结构及操作，构造虚拟的文件结构的方法只是为了让用户进程能够获得一个指向该设备的描述符，对设备进行操作。

除了注册设备，驱动程序还需要提供对设备的各种操作让用户能够操作设备。这些操作的实现都是基于设备类型，比如是独占还是共享设备。有些设备甚至能够提供除了打开读写等基本操作以外的方法给应用程序。

Linux 下的设备驱动程序还有一个很好的特性，就是可以作为模块动态加载到运行的内核中。这种加载方式不仅可以使系统资源的利用率提高，还可以方便编程人员对内核模块的编译调试。灵活的加载方式甚至可以给用户带来功能选择上的方便，当涉及到多个模块的加载与卸载，编程人员可以将一系列的操作编写成 shell，方便用户的调用。

2.5.3 进程通信机制

Linux 支持多任务，使多个进程可以并发地执行。于是对于独占资源就存在着多个进程对其的互斥操作问题，对于需要协作工作的进程还存在着同步问题。一般，通过内存共享、文件共享、设备共享等方式，我们可以让资源被多个进程访问，在 Linux 下实现上述资源的共享依靠缓冲区或者规定的数据结构，每个进程只需根据规定的数据结构访问方法既可以享用资源。如何根据各种资源的特性（是

否可以同时读取，是否可以同时访问）去管理需要进程对它们访问的先后顺序则是进程通信的一个重要部分。Linux 提供了信号量，原子操作，自旋锁等方式，当进程需要访问共享资源时首先对查看资源的使用情况，若是信号量没有达到默认要求或者资源已经被锁定，那就进入排队等待状态。其实等待队列，进程的阻塞、挂起、休眠都是实现同步的机制，只不过我们习惯从 cup 资源的调度角度使用它们。

下面介绍在编写设备驱动程序常用的几种同步机制。信号量是实现进程同步与互斥的基础方法，它其实是一个整型数值和一对 PV 操作，即可以用于资源的互斥访问也可以用于进程的同步。在对资源互斥访问时，共享独占资源的进程在使用资源前都必须使用 P 操作对整型数值减 1 进行判断，若其大于零则使用资源，否则处于等待状态。进程使用完资源后还需使用 V 操作将整型数值加 1。互斥信号量在 Linux 下的实现是以 semaphore 结构体的形式。读写信号量可以维持生产者和消费者的同步，可以将两进程共同访问的一片空间在 Linux 下定义成 rw_semaphore 类型，然后使用相应的读写函数对其访问。

除了信号量机制，还有自旋锁也是一种实现进程同步和互斥的方案。它弥补了进程在不能休眠的情况下对独占资源的访问。独占资源的两个状态分别为锁定和非锁定，进程根据资源的状态对它进行访问。资源被锁定时，进程不会像信号量机制一样被放入等待队列中，而是主动地去检查需要访问的资源状态，直到状态位为可用。

其实除了上面提到的两种方式，当只有一个生产者的时候，我们还可以用到循环队列来实现进程的同步与互斥。这种方式也将会在我们的捕获系统实现中应用到。循环队列有两个头尾指针，头指针由消费者控制，生产者只控制尾指针，两者互不影响。队列为满时头指针和尾指针相等，队列为空时两指针相差 1。

以上介绍的都属于进程之间主动对资源的访问，我们只是从如何保证互斥资源互斥访问的角度去分析，没有从进程的角度去考虑若获取不到资源，他们的状态应该如何转变。比如：用户进程调用度方法去读写设备，但是设备驱动程序在使用中，我们应当怎样让用户进程阻塞；当设备可以使用的读写的时候，我们又应当如何将其唤醒。在 linux 中有一个等待队列来存放等待指定设备的进程，并且通过 wait_queue_head_t 结构体来管理这些被阻塞进程，当某个条件达到的时候这些被阻塞的进程将会全部通过 wake_up()或者 wake_up_interruptible()来唤醒。操作系统为了避免各种意外情况而造成阻塞的进程不能被正常唤醒或者该进程本身属于不能被阻塞的进程，提出了非阻塞 I/O 的概念：当获取不到资源时直接放弃该资

源的时候返回错误或者使用 poll 函数轮询资源的状态。但是轮询资源的状态会消耗系统资源，操作系统又提出了异步 I/O 的概念[38]，每个进程都拥有一个信号，此信号会通知资源的状态发生了改变。

2.5.4 模块化的内核编程

模块化编程思想在 linux 内核中体现的淋漓尽致，在编译内核的时候除了一些必要功能源码外，还可以灵活的选择各种附件功能源码进行编译；在运行的时候，也可以编译模块并加载。对于驱动程序，我们一般都采用动态加载的方式执行。如：将调试好的驱动源代码进行 make 编译生成.ko 文件后通过 insmod 命令动态加载，需要卸载模块则使用 rmmod 命令即可。linux 的模块化编程思想也影响了我们对数据包捕获子系统的设计。

内核模块的编写不同于应用程序的编写[39]：应用程序的入口是 main 函数，模块内核编程模块的入口为一个指定的 xxx_init()，在模块退出时候还有一个指定的出口函数 xxx_exit()；应用程序可以包含一系列的库函数作为头文件，内核编程的头文件都是由 linux 内核提供的并存放在源码树指定目录下；相对于应用程序编程，内核编程参数的传入需要使用 module_param()声明；应用程序可使用 gcc 编译，内核程序则是由 make；此外内核程序的编写模板中还存在着一系列的规定，在此不一一叙述。

同于应用程序编程，由于一个内核模块运行在内核态，所以一个模块的运行出错可能导致整个系统的瘫痪，所以对内核编程的调试技术了解很重要，常见的调试方法是在合适位置插入 printk 打印出所需信息，编程人员通过控制台查看这些信息。但是大量的 printk 在内核中运行会降低系统的性能，linux 提供了一种文件系统/proc。这是种特殊的文件系统里面可以绑定每个模块的及时信息，用户只需要通过创建 proc 文件的方式查看编程模块的运行状态。

Proc 文件是个虚拟文件系统，其根目录下并没有实质的文件。只有当用户进程调用它的时候，proc 系统才根据用户进程请求的内核结构生成相应的文件，并把信息通过终端返回给用户。因为 proc 系统可以实时地访问到内核模块的基本信息，所以经常被编程人员用于内核调试。在 3.3.2 小节提到的设备驱动也可以看做是一个模块，当设备驱动模块被加载后，我们可以再 proc 系统中查找到它的相关运行信息。

2.6. 本章小结

本章首先介绍了数据捕获的基本知识，然后对 unix 系列操作系统中的数据包侦听机制做了详细的说明。在对数据捕获有一定了解后，本人建立排队论模型，采用数学公式推导，说明了影响数据包捕获性能的三个主要因素：接收数据包的方式（轮询/中断）、数据包的软中断处理速度和系统的实时性。在数据包捕获的每个环节都需要处理器的参与，对于这种资源的合理利用是个关键。最后根据这三个因素从软件和硬件的角度分别介绍了各种改进的数据捕获技术。本人将采用零拷贝技术来改善网络监控系统的数据包捕获性能，并具体介绍了在 linux 中实现零拷贝所需要的基础知识。

第三章 基于千兆网卡的数据包捕获系统的概要设计

本课题的研究衍生于校企合作项目某企业网络行为监控系统。数据捕获子系统是该系统中的一部分，故在介绍数据包捕获子系统的的概要设计之前，我们需要了解某企业网络行为监控系统这个大背景。

3.1. 企业网络行为监控系统的简介

某某企业是一个具有 500 多号员工的国有企业，其 Internet 节点规模为 1000 个左右，重点节点 120 个，关键节点 20 个。为了规范企业信息管理并提高企业信息通信效率和信息资源利用效率，该企业决定开发一款企业级的网络行为监控系统，此网络行为监控系统能够远程监控企业内部网络的主机行为，确保企业内部网络的安全及企业内部信息不被外泄。该网络安全系统具有反应迅速、平台无关、安全可靠、实时性强和使用维护方便的特点。

3.1.1 企业网络行为监控系统的功能简介

企业网络行为监控系统能够监控存在于内部网络的主机的上网行为，具体表现为对互联网应用服务的分类管理，即对每一类应用服务按照不同用户组的规则做出网络管理，以便及时的评估当前网络环境。

网络行为监控系统的监控内容包括：管理监控端主机信息，包括主机名、IP 地址、分组信息等；监控主流即时通讯软件，监控主机上网记录、监控邮件收发，并及时统计网络流量。为方便网络管理员对各个员工上网行为的了解，所有监控内容将以日志的形式保存在数据库中并及时更新备份。图 3-1 为系统功能模块图。

下面我们详细介绍七大功能需求：

第一，IP/Mac 管理功能，绑定局域网内所有主机的 IP 和 Mac 地址，使两者一一对应，用户不能随意修改其 IP 或 Mac 信息。其主要包括以下三个子功能。

对 IP 地址的分配和查询：网路管理员可以根据相应情况来为企业内部各个部门分配 IP 地址，并且将每个 IP 地址与用户的详细信息（例如姓名、工号、部门等）绑定。管理员只需通过 IP 地址对每个的用户进行管理。为解决外来人员临时网络

接入问题，管理员可以方便快捷地为某个部门追加分配临时 IP 地址。同时管理员掌握着 IP 地址的修改权限。

对 IP 地址分组：在此将每个分组定义成不同的角色，如管理员，领导，普通员工等。这些角色对应着不同的控制策略。管理员可以将员工放入相应的角色分组中，对他们进行管理，分析不同角色的网络行为。

防止篡改 IP 地址：系统不允许用户擅自修改已绑定的 IP 地址。为避免 IP 资源的浪费及混乱分配而影响到用户的业务，针对擅自修改 IP 地址的用户给以系统警告，并自动切断其网络接入。

第二，恶意网络行为识别功能。此项功能主要针对 ARP 攻击等一系列的局域网攻击行为。系统监控局域网内的所有主机的网络行为，一旦发现恶意行为则立即切断该主机的网络连接，并向管理员报警示意。这样避免产生广播风暴等影响网络安全和效率情况的现象发生。

第三，邮件监控功能。本系统可以监控到内部网络中的主机收发邮件情况，并将这些信息以邮件日志的形式存放到后台数据库。日志记录了发件人，收件人，主题，大小，发送时间等等。网络管理员通过邮件日志来查看并管理所有用户的邮件日志。系统还提供了日志单条件检索功能，例如网络管理员可以根据时间关键字查找某天的邮件日志。

第四，网页监控功能。管理员可以对内部网络中浏览网页的主机进行监控，并设置分组策略，以便对用户的网络浏览范围进行限制，例如限制员工分组的用户在工作时间浏览娱乐、体育新闻等。与邮件监控类似，每台主机浏览过的网页信息都会以日志的形式被存储，主要包括标题、URL、被监控主机的 IP 地址、时间等。网络管理员通过时间关键字查找某天的网页日志。

第五，股票软件和网站监控功能。管理员可以监控员工利用工作时间进行炒股等与工作无关的网络行为。若发现员工利用上班时间超过，系统将及时通知管理员。通过后台采用的过滤技术，系统可以有效地屏蔽各大证券公司开辟的“绿色通道”，禁止员工在主机上运行各种炒股软件。以此保证企业网络资源的有效利用。

第六，关键字屏蔽功能。系统自动对管理员事先设置好的关键字进行匹配过滤，如果用户试图浏览包含有敏感关键字的网页或者在各大论坛上发表包含有敏感关键字的有关言论时，系统将自动对其进行拦截和记录，方便候管理员查看。

第七，对网络游戏、P2P 下载和即时通讯软件监控功能。本系统对被监控网络中的主机的 P2P 流量进行监控，具体包括各种 P2P 下载软件（如电驴，BT、迅雷等）、即时通讯软件（如 MSN、QQ、UC、网易 POPO 等）。管理员可以根据组

别采取不同的控制策略。

对网络游戏的监控主要包括：使用者 IP 地址、游戏时间、以及游戏的种类等信息方便管理员进行阻断和记录等。

对 P2P 软件的监控主要包括：监控局域网内的用户通过 P2P 软件下载的文件类名称、下载时间和文件大小和使用者的 IP 地址，并根据管理员配置的策略进行拦截或记录。

对即时通讯软件的监控包括：聊天者的 IP 地址、消息发送的时间以及消息的内容。

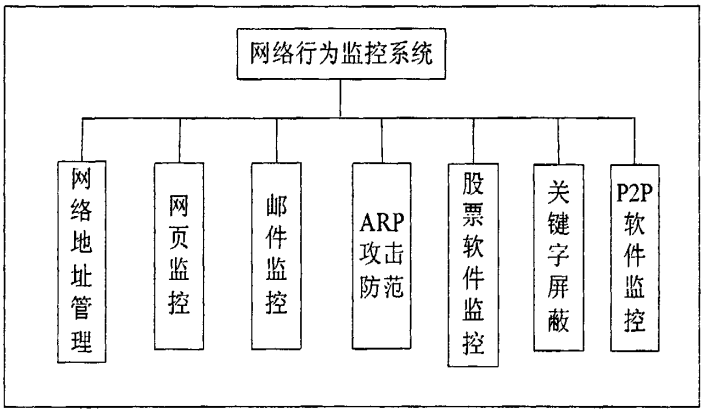

图 3-1 网络行为监控系统的功能模块

3.1.2 企业网络行为监控系统的架构

该网络行为监控系统可以通过网关或者端口镜像的方式接入到企业内部网络中。前者串接在企业局域网的防火墙和交换机之间，对内网的数据包进行全面的捕获和分析，最终决定是否屏蔽或者放行。此部署方案可以控制内网中数据流向，及时了解内部网络环境。后者接入到交换机的镜像端口，虽然对流过的数据数据包可以进行捕获，但并不能对内网数据包流向进行控制。

在该网络系统的初始阶段，我们完成工作是实现监控系统的审计功能，对流入交换机的数据包逐一地进行分析，将有效信息存入到后端数据库中待以后处理。第二阶段中我们才实现数据包的转发功能，即系统网关的接入方式。转发功能实现后就可以对流入数据包执行规则匹配，用户只需要设定规则即可以员工的上网行为，比如屏蔽指定网页，游戏软件等等。

一般，网络行为监控系统的架构存在两种选择：C/S 和 B/S 架构。在前期的需求分析工作中我们了解到企业希望管理员在内部网络中的任一结点都能及时了解到企业员工的上网情况并尽可能少地对内部网络结点的进行配置和维护，我们确

定了该监控系统采用 B/S 架构。图 3-2 为监控系统审计部分框架。

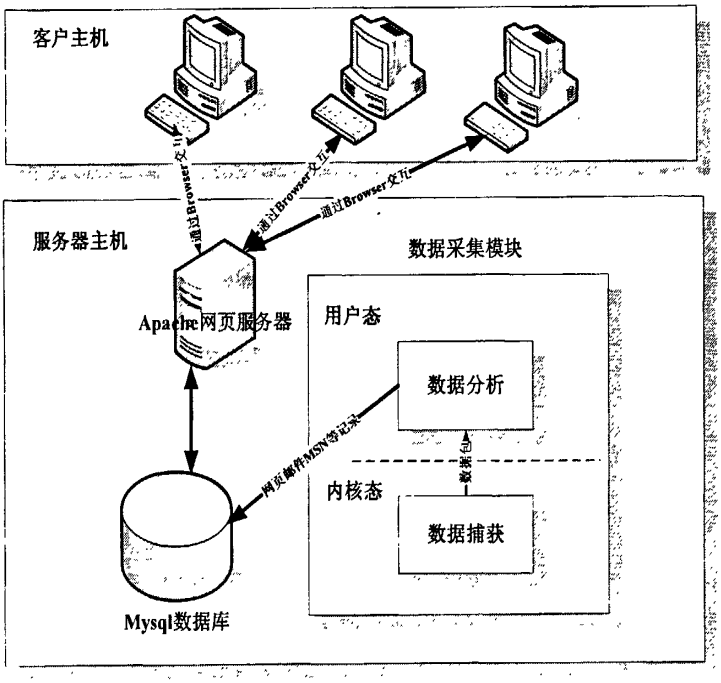

图 3-2 内容审计部分框架图

我们把整个系统分为前台和后台两大部分。前台部分完成与管理员的交互，通过 LAMP（Linux+apache+mysql+php）实现。其主要模块划分如下：

系统资源监视模块包括对流量、cpu、内存、磁盘、协议流量的监视。

日志管理模块包括对记载员工上班期间发生的各种网络行为的日志进行多条件查询。

ARP 预警模块帮助管理员有效检测 ARP 欺骗，并进行预警和记录。

控制设置模块包括软件监控、内容监控、网站屏蔽、关键字屏蔽、策略模板。

基本设置模块包括员工管理、分组管理、部门管理。

数据管理模块包括各种数据表（员工，IP/MAC 地址）导入、导出、保存。

用户管理模块包括对用户的添加删除和密码修改。

系统维护模块包括对数据库的备份、还原、初始化，服务器重启，部署模式的设置，软件特征更新。

后台我们将系统分为内容审计，流量控制和对应用的过滤三大部分。此篇文章主要研究对网络数据包的捕获功能，与数据分析模块一起实现了网络监控系统的审计功能，如上图 3-2 所示。

3.2. 数据包捕获系统的性能需求

在上小节中我们介绍到数据包捕获系统的大背景，根据图 3-2 了解到从网络中流入的数据包经过捕获后用于分析处理，并将处理过后的信息例如：每个员工曾经浏览的网页，MSN 记录，邮件记录等存入数据库。网络管理员通过浏览器对数据库进行访问来获得各员工的网络行为记录。可见无论是网络地址的管理，网页信息的监控，邮件信息的监控，关键字的屏蔽，都需要通过底层数据包捕获系统将采集的数据包进行还原处理或者转发。数据包捕获是网络监控系统的实现的基础。为了保证数据包捕获系统的工作性能，我们首先对企业内部流量做一个分析。

通过对企业企业局域网网关网卡接口进行流量监视并记录流量数据，最后得到该企业一天内网路流量负载变化情况和通过关口设备的网络数据包流量。具体信息如图 3-3，3-4 所示。

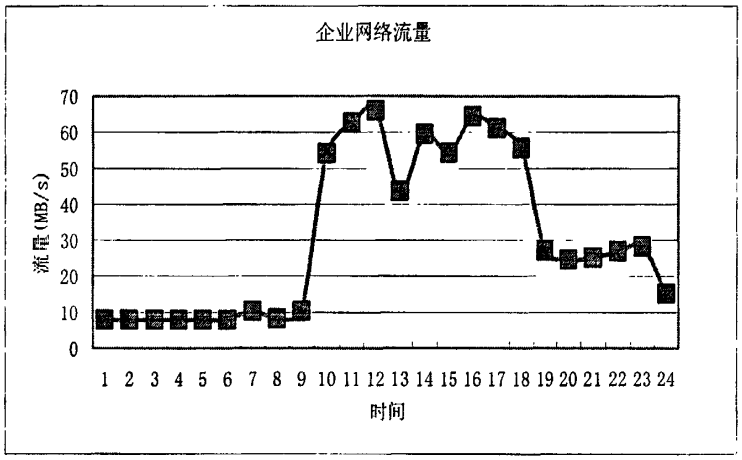

图 3-3 网络关口流量负载

从图 3-3 和图 3-4 中可以看出一天的正常流量中，在上午 10 点及下午 4 点间流量较大，峰值流量约 67MB/S。同时在这个时间段中通过网关设备的数据包的个数也在增加，因此给监控系统对数据包分析和后期处理带来了较大的负荷。故我们需要从数据包捕获系统开始对系统性能做一次优化设计。

我们捕获网关数据包后并对其类型进行了抽样分析，捕获的数据包大多为非业务流数据包。数据包的类型可以分为聊天工具类，如 QQ，MSN；网络视频类，如土豆，PPLIVE，PPSTREAM 等；下载类，如 BT、电驴、迅雷；炒股软件类，如大智慧以及常见娱乐网站类，如网易，新浪等。得出确定非业务流流量如图 3-5

所示的。

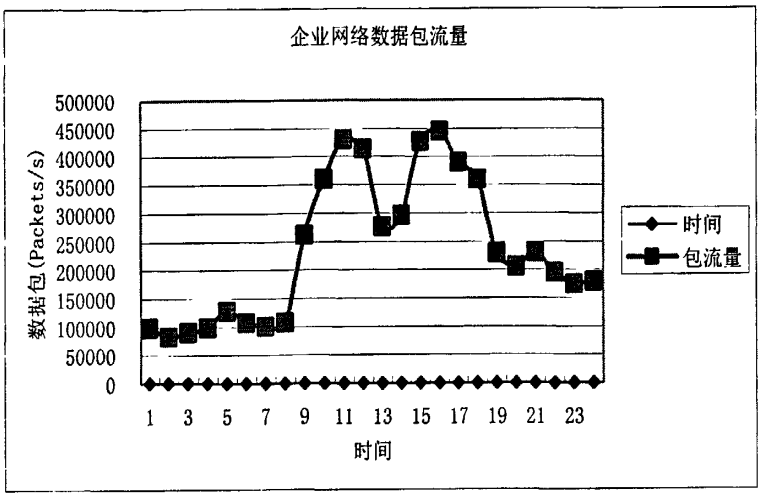

图 3-4 网络关口数据包负载

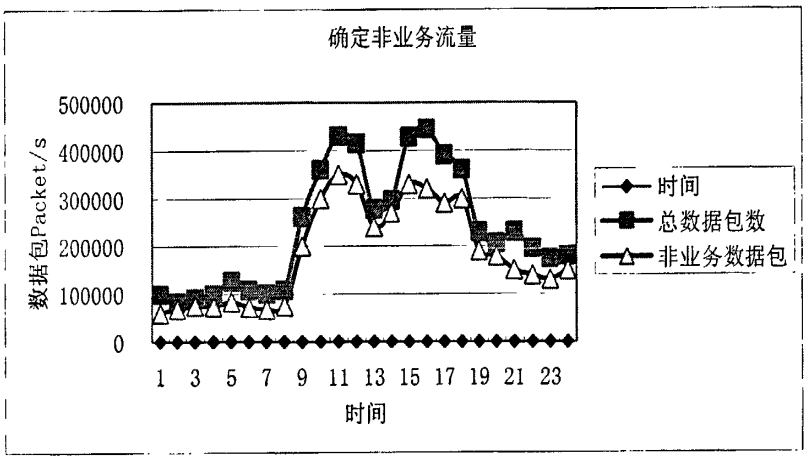

图 3-5 网络关口非业务流量

从图 3-5 中可以了解到:在正常上班时间,确定非业务流占总流量的 60%左右,在其它时间，非业务数据流所占比重更大。说明非业务数据流占据了企业网络资源的大部分带宽。这与网络监控系统的初衷一致，通过上班时间控制非业务性流量来提高企业内部员工的工作效率。

通过对企业流量负载的分析我们可以了解到企业内部为千兆网络环境，在此高速网络环境中，底层数据包的捕获性能影响着整个系统的稳定性。在项目初期开发阶段，我们选择现有的 netfiler/iptables 工具来实现网络监控系统。其中，我们通过对抓取的数据包的关键字进行分析，寻找到特定规律，使用 iptables 设置制定规则来完成对某组员工的网络行为控制。系统后台获得页面传来的参数（对哪些

员工实现哪些控制）后将其传入 shell 中，shell 的功能是用来设置 iptables 过滤规则来实现过滤控制。

netfiler/iptables 在高速网络中的性能不够理想，比如数据包的捕获速度，协议栈对数据包转发的处理速度等跟不上数据包的流入速度，造成丢包现象发生。产生这种现象的主要原因应该在于 netfiler 框架依靠于 linux 的协议栈，在每个网络层次提供全面的数据包过滤功能。他的通用性和全面性不适用于专用网络监控系统。在双处理器机器中，当不添加规则时，数据包通过 netfiler 模块所用时间占整个处理时间的百分之三十左右[40]。对于专用的网络监控系统，我们需要对协议栈进行精简以减少数据包通过协议栈的处理时间，同时借鉴 netflier 数据拦截方式去控制员工的上网行为。

故在此项目后期我们改变方案，首先实现系统的旁路审计功能，在此种状态下，数据包通过零拷贝绕过系统协议栈直接被用户程序使用。对数据过滤功能预计在使用数据捕获功能的前提下交给应用程序模块去实现。根据第一小节的功能介绍我们可以知道，对员工的控制功有 IP/MAC 的控制，还有各种下载软件及游戏软件的控制等，要实现这些功能，我们需要屏蔽报文的端口号，目的地址，源地址等信息，若将数据包放入标准协议栈处理的话，需要依次处理，我们直接通过零拷贝获得数据包，根据需要建立新的协议栈做相应处理即可。

3.3. 数据包捕获子系统的设计方案

在 3.2 小节中我们了解到企业内部一天内网络流量基本维持在 60MB/s，峰值流量小于 70MB/s。本系统所处在高速网络环境下，我们不能单纯基于 libpcap 或 netfilters 去捕获。Libpcap 在百兆网络环境下，不适用于千兆网络环境，而且只对数据包起到侦听作用，无法拦截数据包。Netfilters 可以对数据包启到拦截作用，但在不设置任何捕获策略规则前提下使用 Netfilters，其处理器占用率较高，不够理想。为了在性能上有所改善，我们将借鉴第二章的零拷贝思想，将网卡中的数据包直接送入到内核中，然后通过用户进程通过内存映射直接读取内核中的数据包来减少因 libpcap 数据包的两次复制或 netfilters 协议栈的处理而造成的接收活锁现象。

我们会在 linux 内核本身提供的机制基础上稍加去实现零拷贝思想。例如：分析了网络驱动设备的工作流程，并对网卡驱动做了更改；采用了其内核提供的运行时可扩展性建立了一个虚拟设备来管理公共内存；对内存的映射的实现是基于

linux 的分页机制和虚拟内存概念；后期应用程序对数据包分解可以借鉴 linux 网络协议栈的套接字。

数据包捕获依附的软件平台是 redhat linux AS4.0。在 linux 系统中通过网卡中断产生的数据包会被放入协议栈的等待队列中进行处理，直到数据包被成功处理完，其在内核中所属空间才能得到释放。改进的数据包捕获系统使数据包由网卡 DMA 到内核环形缓冲区中排队，等待用户将数据包提取处理，处理成功后释放掉。

3.3.1 数据包捕获子系统的功能模块划分

数据包捕获子系统完成的工作为：将网络数据包传送到指定的一片内核缓冲区中，并将这片缓冲区映射到用户空间，为用户空间提供直接获取网络数据包的借口。下面我们将此系统分为三个模块来实现。

一般网卡驱动分配内核空间并充当从硬件采集数据包的角色，在此为了实现良好的模块性我们将内存分配从网卡驱动中脱离出来，自编写一个共享缓冲区模块管理共享缓冲区。网卡驱动的工作简化为将数据包通过 NAPI 方式 DMA 到共享缓冲区中，后续操作全部交给共享缓冲区模块实现。

共享缓冲区模块的消费者是用户进程，用户进程读取数据包并处理完毕后共享缓冲区再将该数据包释放掉。用户接口模块负责完成内核空间到用户空间的映射，并将数据包直接提供给用户处理。

共享缓冲区模块是此工作中最关键的模块，它需要管理缓冲区中数据包的分配和释放，还需要维持生产者网卡和消费者用户之间的同步。从另一个角度思考此模块类似于基于内存的字符设备驱动。

3.3.2 功能模块间的同步

将数据包捕获系统分为了三个模块以后，如何完成这些模块初始化及如何处理模块运行过程中与应用程序的协同是我们的需要考虑的问题。

首先，网卡驱动模块必须要在缓冲区管理模块分配好空间的后才能工作，我们需要通知网卡驱动模块缓冲区管理模块已经准备好。在 linux 下模块与模块之间的通信一般基于函数或者共享变量。对于驱动模块之间的调用其实可以用函数的方式来实现。两个模块完成加载后，缓冲区管理模块调用网卡驱动模块的 open 函数让其运行。

其次，用户进程与网卡之间的工作方式也需要考虑：由用户主动发起接收数

据还是选择由网卡主动发起接收数据。

方案一，在共享缓冲区模块和网卡驱动模块初始化完成后网卡中有数据流入，网卡驱动程序通知用户进程可以接收数据，用户进程通过内存映射机制接收数据。两进程（驱动和用户）在协同工作的时候：当共享缓冲区中没有数据包，用户进程访问失败并放入等待队列中；当共享缓冲区数据满溢，则丢弃最早时间流入的没有经过处理的数据包。

方案二，用户需要获得网络数据的时候通过系统调用完成共享缓冲区的映射并向共享缓冲区模块发送信号并进入等待状态，缓冲区模块收到信号后打开网络设备接收数据包，若有数据包到来，网卡驱动模块将数据包放入共享缓冲区中，唤醒用户进程接收数据包。在开始正常工作后类似方案一处理。

上述两个方案中都涉及到了模块间的通信，一般模块间的通信都是通过函数或者公共变量来实现的。更具体的说，操作系统内核提供了一系列的机制如信号量，自旋锁等来同步。但是用户进程通过 mmap 后不需要转化为内核态就可以访问内存，所以初始化工作和运行时候的同步工作都必须做好。

对于初始化工作，在方案一中网卡驱动通知用户进程接收数据的实现办法是建立一个守护进程及时地启动应用程序进行数据包捕获。方案二采用的是用户主动地捕获，通过系统调用虚拟设备相应的 mmap 方法完成映射后，若发现没有其他用户进程使用此虚拟设备则通过系统调用进入睡眠状态，直到网卡数据流入的时候才将其唤醒。若虚拟设备资源被别的用户进程占用，此用户进程进入阻塞状态等待资源释放。我们选择方案二，由用户去实现数据包的主动捕获功能。主要原因在于用户有主动权较灵活，对系统的设置易控制，比如：在模块加载中通过传入参数设定大小，或者用户可以通过 ioctl 方式[41]设定共享缓冲区的大小。

对于运行时的同步工作，我们不提倡使用内核提供的信号量，自旋锁机制。首先用户态的进程只需通过一次 mmap 映射后直接读取内核空间的数据，并不是每次都通过系统调用转换到内核态进程（毕竟过多的系统调用会给整个系统的性能造成影响），我们无法直接使用内核提供的同步机制。其次，自旋锁和原子操作等对系统开销很大。我们可以使用另一种免锁机制——循环队列来实现两种状态的同步。循环队列是一种在用户空间和内核空间都适用的基本结构，使系统的实现更加简单。

最后，对多个用户进程同时读取数据的时候，怎样维持其互斥。初步想法是将共享缓冲区模块虚拟成一个字符设备，每次只有一个用户进程对该设备进行操作，其余的预读取数据的用户进程被拒绝。但是这种方式的效率较慢，有待改进。

3.3.3 用户接口的设计

一般当网卡接收到的以太网帧后，网卡驱动程序为其分配一个套接字缓冲结构的空间放入数据包，然后通过协议栈的函数对套接字缓冲结构的控制部分进行填充。在我们的捕获系统中将数据包直接取出放入共享缓冲区中，然后用户进程根据需要取出分析，用户进程取出的数据包是链路层帧，对于链路层帧的处理我们交给具体的应用程序，但是我们将提供跟这些应用程序一个接口，我们以回调函数的形式调用编程者自编的数据包处理函数，具体的内容在 4.3 小节中介绍。

3.4. 本章小结

本章首先介绍了数据包捕获系统所依附的项目背景，包括基于数据包捕获子系统的各种应用的介绍，由此可见数据包捕获系统在整个网络监控系统中的基础作用；然后根据企业的内部网络流量分析图放弃了传统的数据包捕获方案，提出了一种基于零拷贝思想的数据包捕获系统的设计方案来提高数据包捕获性能。最后在系统地介绍了与系统的实现相关的基础知识前提下，将数据包捕获子系统进行模块化划分，并讨论了各种模块的交互及实现中需要考虑的问题，以便读者能够更好地理解第四章节的实现部分。

第四章 基于千兆网卡的数据包捕获系统的详细设计与实现

在上一章节我们介绍了数据包捕获系统的设计方案，将其分为三个功能模块实现：网卡驱动模块，共享缓冲区管理模块，用户接口模块。网卡驱动模块将网络数据包直接传送到共享缓冲区中，应用程序通过内存映射直接使用我们提供的接口函数从内核共享缓冲区中取出数据后期处理。具体流程如图 4-1 所示。

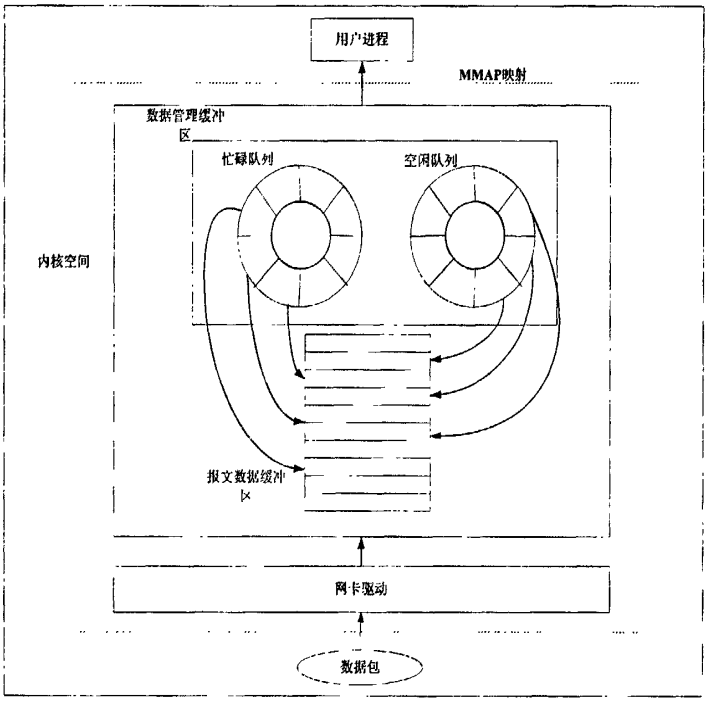

图 4-1 零拷贝模块结构图

网卡驱动模块：正常情况下，网络数据包经网卡驱动程序发送到内核缓冲区中，然后逐个传送到 Linux 协议栈进行处理。由于网络硬件设施发展飞速，千兆网络的普及对数据包处理的速度要求越高，传统的网卡通过中断方式获取数据包已经不能满足需求，一种通过轮询方式获取数据包的方法相继被提出。目前的网卡驱动中，对两种获取数据包的方式都支持。我们的目标是在这两种方式下实现将网卡获取的数据包绕过默认协议栈直接通过接口函数发送给上层应用程序处理。

共享缓冲区管理模块：实现数据包捕获的关键在于内核缓冲区的管理，通过网卡驱动截取的数据包将被发送到内核缓冲区中供上层应用程序读取。我们采用

Linux 提供的内核映射机制完成用户进程和内核进程对内核态缓冲区的共享，避免数据包从内核态到用户态的一次拷贝。

用户接口模块：作为应用层的接口，需要完成的工作是将内存映射到数据包的提取封装好，最后循环读取共享缓冲区，将数据包一个个送给数据包处理程序进行处理。

以上三点中，最难实现的内核缓冲区的管理，其中包括内核缓冲区的分配，与用户进程的共享映射，内核进程与用户进程对数据的存取的同步操作。在 linux 模块编程中，模块与模块之间的交互通常使用共享变量或者函数接口。我们设计函数接口 ykvm_enqueue()与网卡驱动模块通信,提供已分配好的空闲区间供其存放。我们还提供 ykvm_open()和 get_packet()接口供应用程序编程者使用。

4.1. 千兆网卡驱动模块的设计

网卡驱动模块将接收到的数据包传递给共享缓冲区管理模块，缓冲区管理模块负责将这些数据包逐个传送到用户进程中去。由于千兆网卡都采用 DMA 方式传输数据包，所以我们必须对其数据结构和数据包接收流程进行了解，以便后期实现数据包到共享缓冲区的一次拷贝。

4.1.1 关键数据结构的分析

实验中我们涉及到的网卡驱动为 IBM X6350 的集成网卡 Broadcom Netxtreme II Gigabit Ethernet Driver，故将其网卡驱动模块简称为 bnx2。在 bnx2 驱动中网络数据包通过 DMA 方式被直接存放到内核态内存中，避免了 CPU 参与，节省了处理器资源。首先我们需要了解 bnx2 如何实现网络数据包的传输及网卡与驱动的同步。

在 bnx2 中，网络数据包都是离散地以套接字缓存结构的形式存储的。若我们将接收缓冲区看做由多个固定长度的物理块组成，每个块中存放着一个数据包。在此存在一个 sw_bd 结构与每个数据包对应，sw_bd 结构是其对应的数据包的描述符，包含着该数据包的物理地址 mapping 及数据包的虚拟地址 skb。此结构在数据包被提交到网络协议栈中被使用到。虽然数据包是离散存储的，但是数据包的描述符以数组的形式线性存储。我们对数据包的管理简化为对描述符数组的管理。具体的结构如下图 4-2 所示。

Bnx2 中与网卡 DMA 直接交互的数据结构是 rx_bd。此结构包含着内核中可覆

盖空间的首地址（DMA 可识别的物理地址）rx_bd_haddr，该片连续空间的长度 len 及相关硬件信息。根据源码分析可以知道，每次的数据缓冲区都是在设备打开以后通过 bnx2_alloc_rx_skb() 调用 netdev_alloc_skb() 分配一批空间。bnx2_alloc_rx_skb()除了简单的分配空间，还需要完成分配的空间到网卡 DMA 的地址映射，让 DMA 知道每次的数据往哪块缓冲区中送。其中 pci_map_single()函数完成此工作，把分配空间的逻辑地址转化为物理地址。在 X86 平台下，内核物理地址与 DMA 可识别的物理地址是相同的，得到的物理地址后需要将相应的 sw_bd，rx_bd 和 bnx2_rx_ring_info 结构变量中的物理地址域赋值。

sw_bd 与 rx_bd 是一一对应的，它们都为单个数据包的描述符，只不过使用者不一样。前者是针对驱动的，后者是针对网卡 DMA 寄存器的。

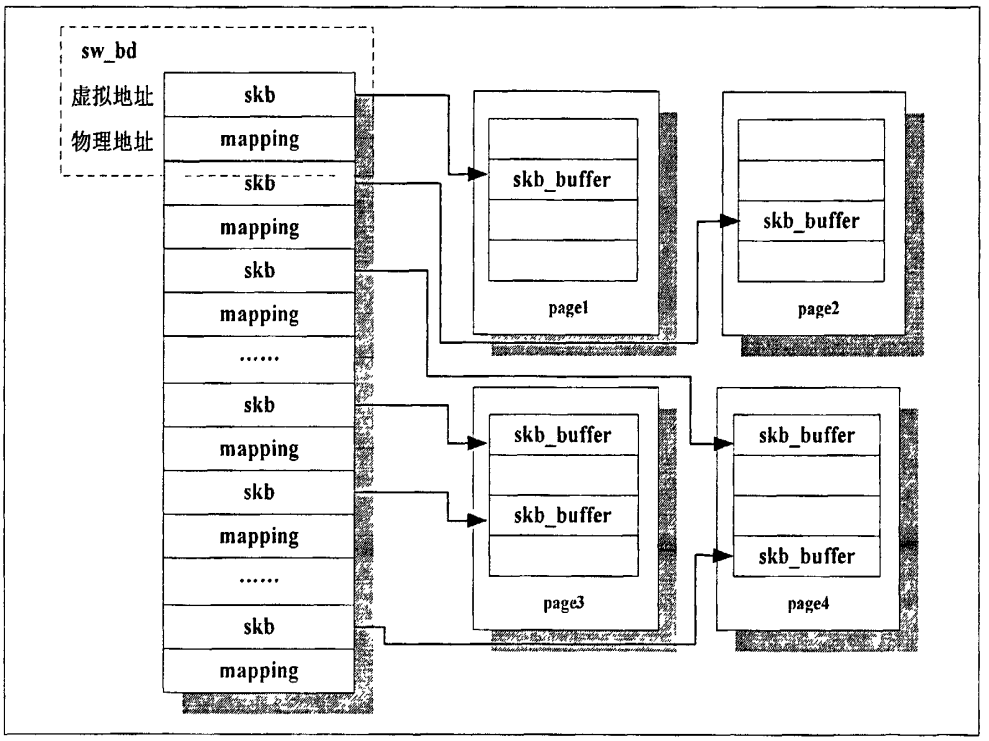

图 4-2　描述符数组结构

下面将介绍接收缓冲队列的索引信息 bnx2_rx_info 结构，其中包含的消费者索引 rx_cons 和生产者索引 rx_prod 是维持缓冲区的同步的关键。bnx2_rx_info 中还包含着一个 DMA 类型地址构成的数组 rx_desc_mapping，它记载着的每个描述符的 DMA 地址。

bnx2 中针对每个 bnx2_napi 结构（实现 napi 传输的关键结构，下小节将提及）定义了 4 个 DMA 描述符队列，为了保持页对齐，每个描述符队列存放着

RX_DESC_CNT 个 DMA 描述符，其中 RX_DESC_CNT=BCM_PAGE_SIZE / sizeof(struct rx_bd)，即一页大小空间可以存放 rx_bd 结构的个数 。这些描述符在内核中以二维数组的形式存放，bnx2_rx_info 中采用一个 rx_desc_ring[4]一维数组来存放四组 DMA 描述符队列的首地址，rx_desc_mapping[4]存放四组 DMA 描述符队列的物理首地址。以后对每个 rx_bd 元素的访问可以通过首地址和索引 rx_prod 来完成。在上一段落中介绍了 sw_bd 结构变量的存储方式为一维数组连续存放，故 bnx2_rx_info 中采用 rx_buf_ring 来记录该一维数组的首地址。下图 4-3 将说明其关系。

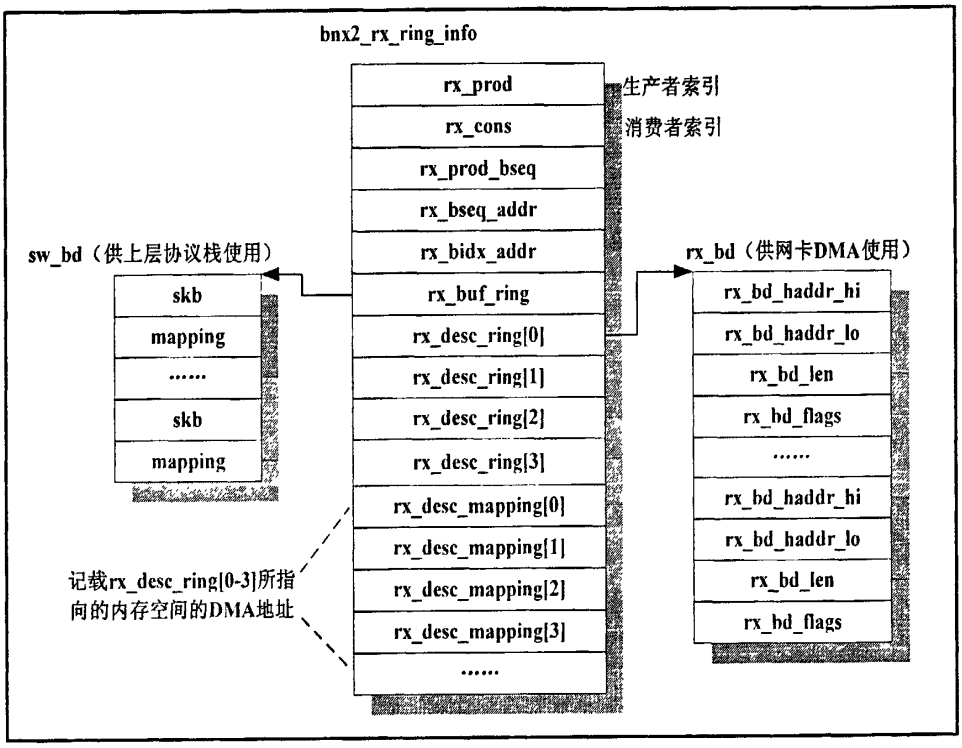

图 4-3 内核接收缓冲区的数据结构

4.1.2 网卡驱动接收流程的分析

Broadcom Netxtreme II Gigabit Ethernet 网卡支持 NAPI 方式接收数据包。一般情况下系统进入中断处理后调用 process_backlog()函数处理数据包，若网卡驱动中实现了具体的轮询函数，系统则调用网卡驱动提供的轮询函数。Bnx2 在初始化阶段注册轮询函数 poll()并通过 NAPI_STATE_SCHED 标志位来设定 NAPI 数据接收方式。图 4-4 为网卡驱动实现的 NAPI 轮询函数的流程，图 4-5 为接收数据包处理流程，此函数为我们实现零拷贝的关键函数。

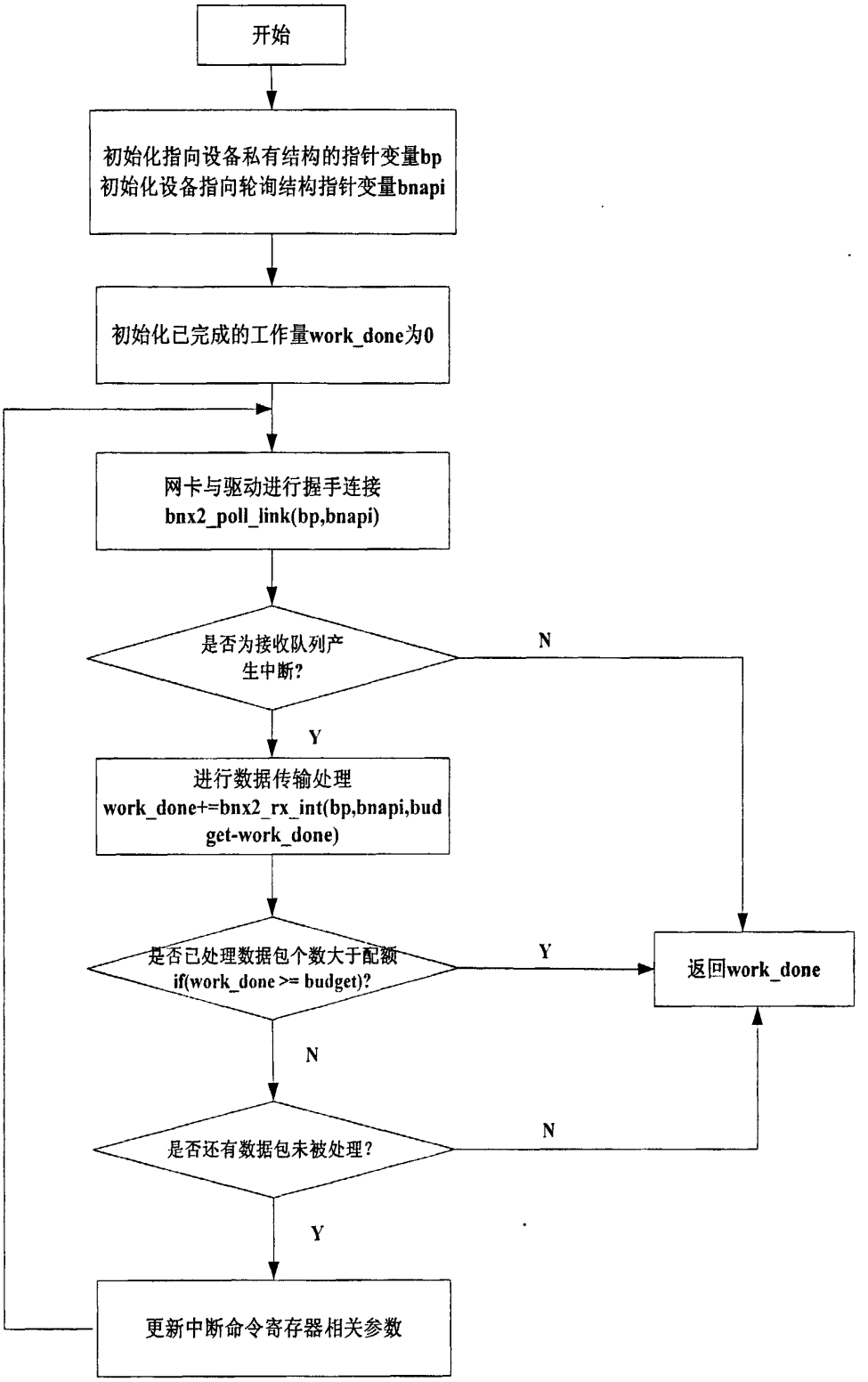

图 4-4　bnx2 驱动轮询函数流程

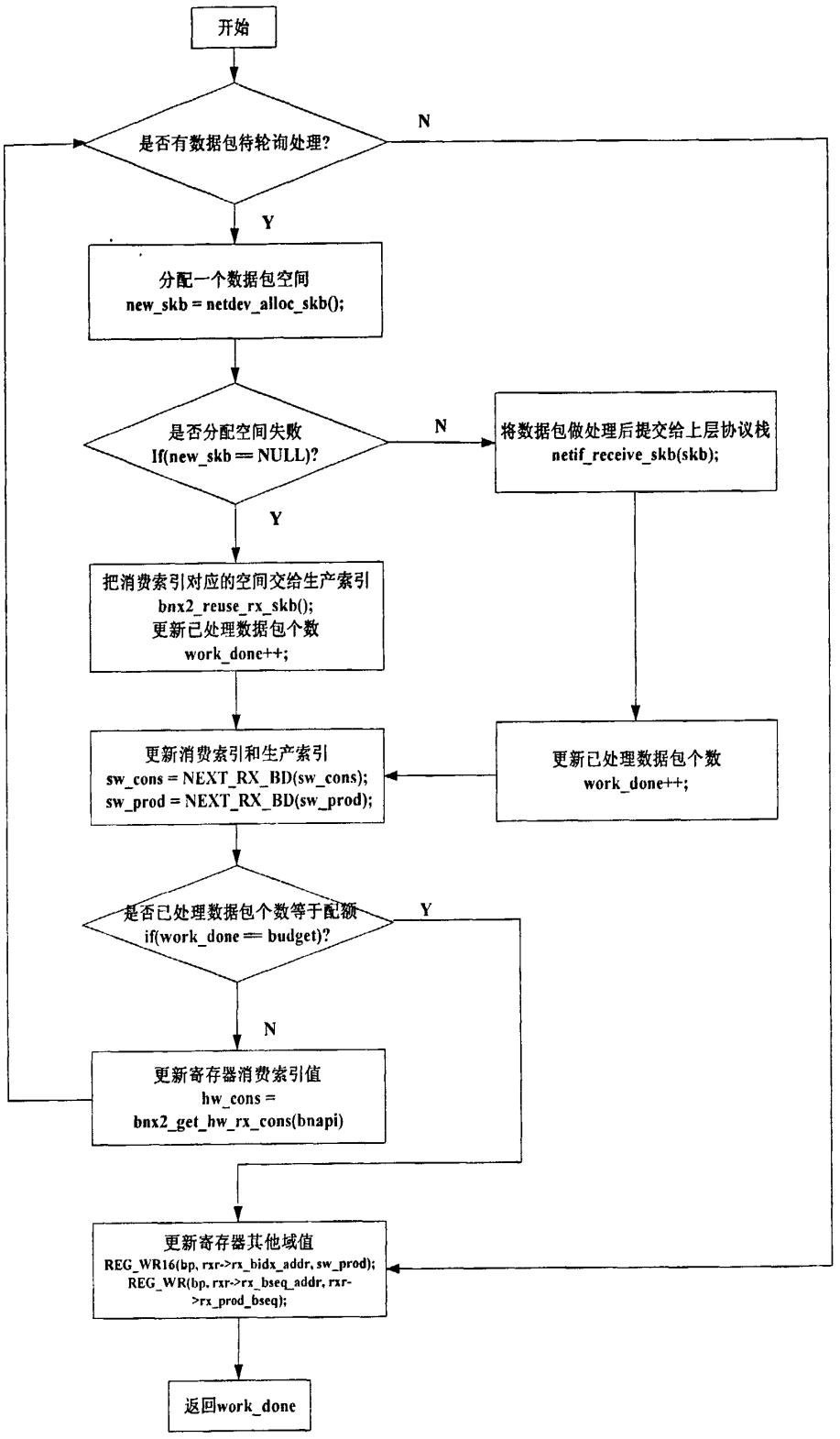

图 4-5 bnx2 驱动处理数据包流程

网卡私有结构 bnx2 中存在一个 bnx2_napi 结构数组，每个数组元素包含着轮

询队列，并有一个 bnx2_rx_info 结构变量与其对应，以此实现 NAPI 方式的数据接收功能，以下的讨论全部针对单个 bnx2_napi 结构的实现，其实我们可以将此结构看成一个设备。

在 bnx2 中以网卡每次以单个 bnx2_napi 结构变量进行配额轮询，通过在 bnx2_poll(struct napi_struct *napi, int budget)中调用 bnx2_poll_work(bp, bnapi, work_done, budget)函数来完成，如上图 4-4。其中 budget 为一次轮询中传输的最大数据包个数即我们在 2.4.2 小节提及的配额概念，在该驱动中设为 64。work_done 为轮询中已完成轮询的数据包个数。数据包最终需要通过 bnx2_rx_int()函数进行相应的解包处理后，通过 netif_receive_skb()提交给上层协议栈，如上图 4-5。

在 4.1.2 小节我们介绍过 bnx2 通过描述符循环数组来控制网卡与驱动之间的同步，实际上是对其消费者索引 sw_prod 和生产者索引 sw_cons 的更新。默认情况，向协议栈提交完数据包后 sw_prod 和 sw_cons 索引都需要加 1，但索引值超过数组长度的时候，需要从 0 开始。图 4-5 中数据包更新两索引使用到宏 NEXT_RX_BD 即完成此功能。对于内核空间紧张，协议栈处理的数据包空间不能及时分配的情况，bnx2 直接将缓冲区中未被送入协议栈的数据包空间覆盖，此功能由 bnx2_reuse_skb()来完成。

4.1.3 网络驱动程序的修改

我们需要做的工作是将通过网卡 DMA 存放到内核的数据包送入指定的共享缓冲区中，并不提交给上层协议栈，故需要修改 bnx2_rx_int 函数中分配空间的函数 netdev_alloc_skb()，协议栈提交函数 netif_receive_skb()。

我们将空间分配放入内核共享缓冲区模块中，ykvm_init()函数中采用一次性线性分配的方式 vmalloc()分配接收缓冲区空间，具体见 4.2.2 小节

对于消费者和生产者索引的更新也需要做修改，在下一章节，我们采用两个接收队列来维持同步，存在着两对生产者和消费者索引，具体见 4.2.3 小节。

对于提交给协议栈空间的修改，我们在用户接口模块中提供相应的接口函数 get_packet()让用户对数据包进行更直接的操作，具体见 4.3 小节。

4.2. 内核共享缓冲区模块

此模块的工作主要有三点，第一是对指定共享缓冲区的分配和初始化，让网卡驱动程序能够获取此缓冲区，往里面填充数据包。第二是对用户进程缓冲区的

映射，让用户进程能够直接读取此缓冲区中的数据。第三是同步用户进程和内核进程对缓冲区的读写和多个用户进程对缓冲区的读写。

网卡驱动模块中没有合适的 mmap 方法，此模块功能类似于字符设备功能，可以通过虚拟字符设备的功能来完成，而且对多个用户的互斥访问的控制使用虚拟设备结构易实现，我们将此模块看做虚拟设备模块实现，模块的名字为 YKVM。

4.2.1 数据结构的建立

共享缓冲区内存的数据结构建立需要考虑很多问题。由于用户空间和内核空间同过 mmap 共享缓冲区，用户进程不需要通过系统调用转换到内核态，所以对缓冲区的同步机制的实现的最简单的办法使用免锁算法——循环队列。内存的数据结构中必须分为两部分设计：真实的数据包存放区间和管理进程同步的管理空间。在第一小节我们了解到 bnx2 中 DMA 缓冲区的数据结构，其中的 bnx2_rx_ring_info 就是一种管理结构，我们可以借鉴。

我们建立一个新的 ykvm_rx_desc_ring 结构供 ykvm 使用。在写入数据的时候，网卡驱动通过 bnx2_rx_desc_ring 提供的索引信息 rx_cons 去寻找描述符 rx_bd，rx_bd 中记载着空闲数据块的物理地址，再根据物理地址将数据填入其中，然后通知 ykvm 分配相应空间去取数据。

其次我们从用户进程的角度来考虑，通过 mmap 访问共享缓冲区的时候我们必须知道管理空间的用户地址和数据空间的用户地址，管理空间放在内存结构的初始位置，ykvm 模块被当做虚拟字符设备打开的时候，用户可以获得其文件描述符，既可以对管理空间进行访问。数据空间的用户地址可以放入描述符 ykvm_rx_desc 中。ykvm_rx_desc 为虚拟设备接收描述符的结构，此结构类似于 sw_bd 结构。其中的 dma_addr 为对应数据块的内核物理地址，buf_addr 为对应数据块的内核虚拟地址。

ykvm_rx_desc_ring 结构为虚拟设备接收描述符队列，队列中存储着被网卡填满的数据块待用户进程读取。这里的数据块指的是大小一定的连续空间，每个数据缓冲区块可包含一个数据包，每个数据缓冲块都对应一个描述符。在 ykvm_rx_desc_ring 中的 rx_cons 和 rx_prod 分别为其队头和队尾索引，rx_prod_bseq 为设备已处理的字节总数。desc_buffer[YK_NUM]存放着 ykvm_rx_desc 结构类型的描述符，其中 YK_NUM 可以通过用户的设定，默认情况下为 1024（根据 bnx2 源码中的默认描述符个数得出）。

我们将数据包放入一个大小为 1520 的数组中保存。以太网数据帧最大为 1518 字节：数据 1500+目的地址 6+源地址 6+类型 2+FCS 4，其中 FCS 在物理层被网卡略去，故传入内核协议栈的数据包最大为 1514 字节，在此分配 1520 个字节考虑到结构体对齐的方式。缓冲块 ykvm_buffer 中不仅包含了网络数据，还有数据包的实际长度 length。为了实现页面对齐，还需要定义 buf_in[520]进行填充。

ykvm_rx_desc_ring，ykvm_rx_desc 与 ykvm_buffer 具体关系如图 4-6 所示。

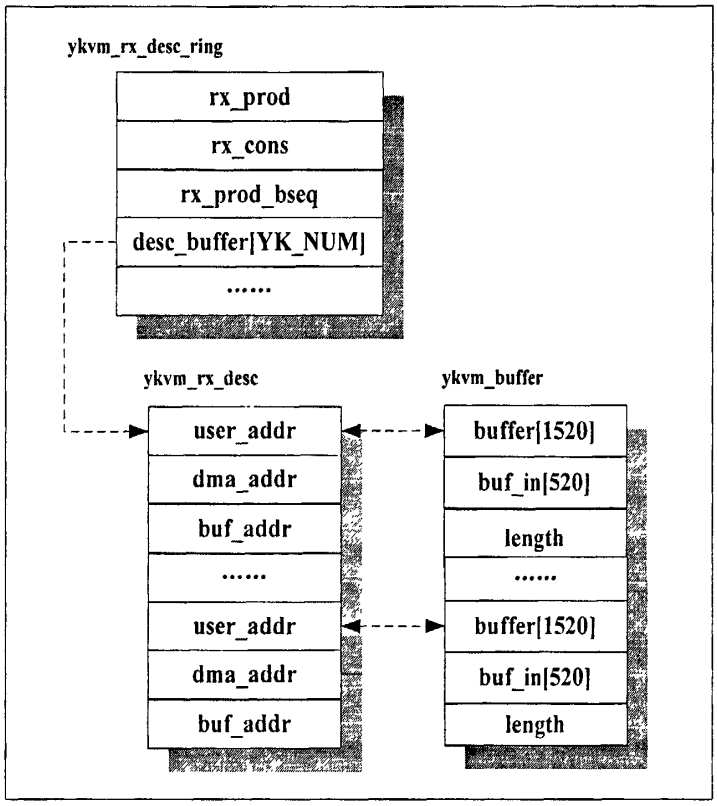

图 4-6　YKVM 关键数据结构

传统的网卡驱动都是在设备打开时申请描述符空间和一部分数据包存储空间，待数据接收阶段再将剩余的数据包空间动态分配完。由于涉及到地址映射问题，ykvm 模块被启动的时候就要进行内核空间的分配，我们在模块初始化阶段将描述符和对应的缓冲区队列在)中通过 vmalloc 动态分配。

Ykvm 模块启动时的内核分配结构如图 4-7，由管理空间和数据空间组成，其中管理空间中包含两个队列：ykvm_rx_busy 忙碌队列和 ykvm_rx_free 空闲队列。数据空间线性存储着数据包。用户空间的数据结构与图 4-6 中的共享数据结构一致。用户进程对数据包的读取是通过打开在虚拟设备，将其文件描述符传递给在

虚拟设备中定义 mmap 方法，mmap 建立新的 vma 结构，完成虚拟地址页表的创建，返回给用户进程一个用户空间的虚拟地址（ykvm_rx_busy 队列的首地址），用户通过这个地址就可以访问内核。在初始化的时候，用户程序根据首地址和约定的数据结构（ykvm_rx_desc，ykvm_buffer 等）计算出 ykvm_rx_free 队列及 data_buffer 的首地址，用户进程根据地址对取出数据包。

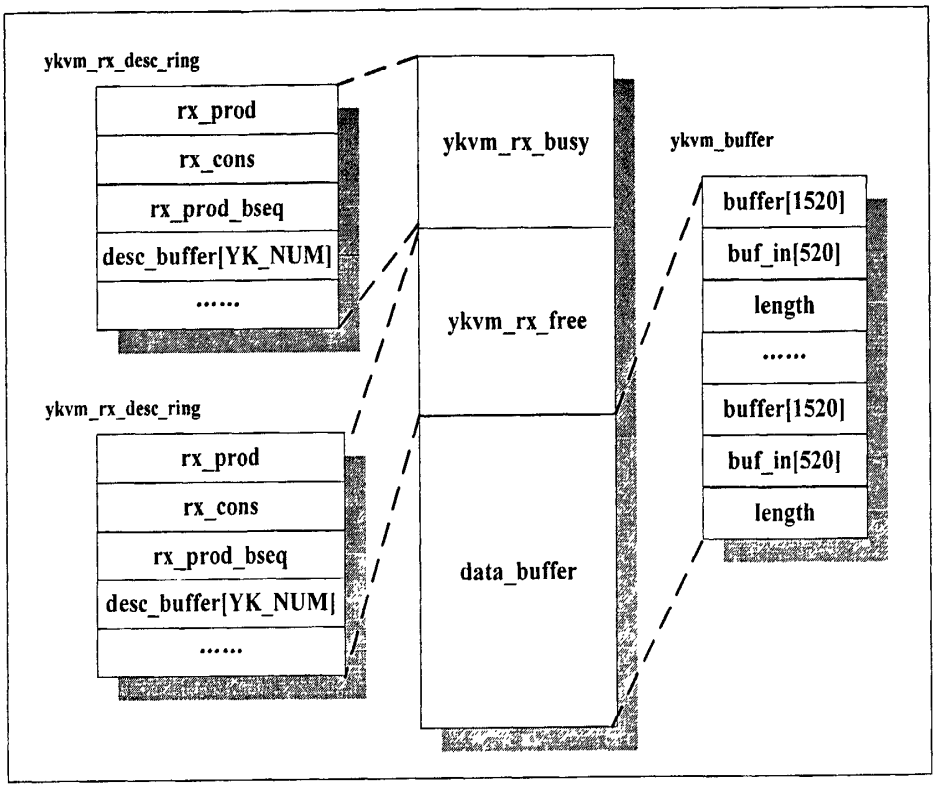

图 4-7 共享缓冲区公共部分结构

4.2.2 共享缓冲区中内存的分配

为了实现用户态进程与内核态进程的内存共享，我们采用 linux 内核提供的 mmap 映射机制将内核态内存空间映射到用户态内存空间，地址空间的映射有两种方法。

一种是在用户空间划分缓冲区进行分配[42]，还有一种是在内核空间划分缓冲区进行分配。在用户空间分配共享缓冲区的优点是可以一次性成功分配大量的缓冲区，但是由于用户空间的内存管理是虚拟地址管理机制，存在着页面的换入换出，DMA 无法访问，所以我们还需要在内核空间建立一张地址映射表，将所有被分配的页面锁住，然后计算出每页对应的物理地址放入表中。每次 DMA 存放数据

包需要内核态的地址映射表去指定的地址访问。

另一种方式是在内核空间分配共享缓冲区，我们可以利用 linux 默认提供的 mmap 函数直接将缓冲区与用户空间映射。本篇文章介绍的是第二种方法，它的优点是映射机制简单。下面我们讨论共享缓冲区的分配上的两种选择。

第一种方法是对由网卡 DMA 过来的数据，我们可以采用引导分配获得一大片连续空间，将数据放入其中，然后通过调用 ioremap()实现内核空间与物理空间的映射，用户空间与物理空间的映射则使用 mmap()。但是这种方法对可加载模块来说无法实现，只使用与直接编译到内核的模块，所以并不适合我们。

我们选用是动态地在内核空间分配内存，get_free_pages 函数可以满足要求，但我们还需考虑到内存资源的紧张，get_free_pages 存在着分配失败的可能性，故当其分配失败的话，或者分配的空间大于 128K，我们只能采取 vmalloc()分配缓冲区，此函数分配的缓冲区的物理地址是不连续的，但是内核虚拟地址是连续的。所以对于此种情况，内核共享缓冲区到用户空间的映射我们不能靠调用系统提供的 remap_pfn_range()来完成，此函数适用于一次性为一段物理地址连续的空间建立新的页表，所以我们还需要通过自己定义的 ykvm_mmap(struct file *file, struct vm_area_struct *vma)来完成映射工作。

ykvm_mmap 函数具体的工作流程是：找到 vmalloc 实现连续虚拟空间分配所用到的页表，根据虚拟地址项找到页表项所对应的页面的内核逻辑地址，然后通过这个真实的逻辑地址线性完成与物理地址的映射。

除此之外，由于 vmalloc 分配空间时的页表建立并不是像 remap_pfn_range 一样，一次性建立好，而是在需要访问的物理页面不存在内存中的时候建立页表，所以我们还需要实现一个 nopage 方法，处理缺页情况。当我们使用 vmalloc 分配空间的时候真正完成内存映射功能的是 nopage 方法，以页为单位返回用户虚拟地址对应的物理地址的 page 结构指针。缺页函数的流程如下图 4-8 所示。

struct page*ykvm_nopage(struct vm_area_struct *vma, unsigned long_address, int *type)为此函数的原型。

vma 是指向 vm_area_struct 结构的指针，该结构联系用户虚拟地址与内核物理地址，是我们的缺页转换需要用到的一个关键结构。

long_address 是产生缺页的用户虚拟地址。

type 指向一个 int 型整数，此整数主要用来存放函数的错误信息，不同的整数值代表着不同的错误类型。

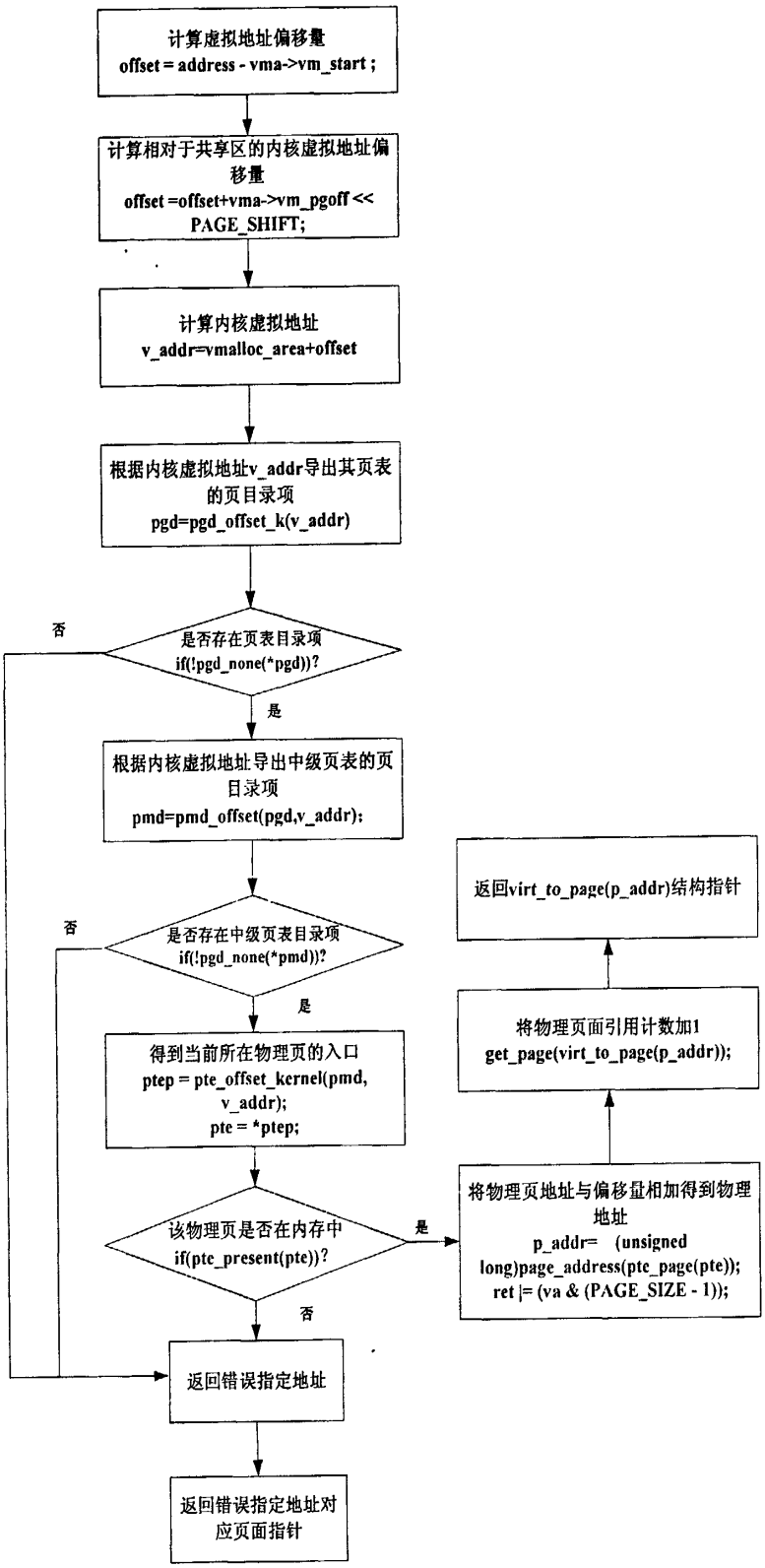

图 4-8 ykvm_nopage 函数流程图

在 2.5.1 小节中介绍了 vm_area_structs 结构，mmap 的实现需要使用虚拟地址结构体。每次对 mmap 的方法调用是生成用户空间到内核空间的一个 vma，它包含了被映射的虚拟空间的起始地址 vm_start，尾端地址 vm_end 和被映射的第一页相对于映射内存的偏移量 vm_pgoff（以页为单位）。从虚拟地址 v_addr 到物理地址 p_addr 的转换只需要计算出该虚拟地址的偏移量 offset=v_addr-vm_start，然后利用 vm_pgoff 将其转换（start_buff+offset+vm_pgoff<<PAGE_SHIFT）后加被映射文件内核起始地址。注意，此刻得到的地址只是内核的虚拟地址，我们还需要根据内核页表找到相应的物理地址。

在内核中的空间分配都是以页为单位的，为了提高系统的运行速度，在定义数据结构时有必要考虑页对齐。以太网网卡接收到的数据包是以帧为单位的。一帧数据包的最大长度为 1500 字节，最小长度为 46 字节。Linux 下一个页面的长度为 4KB，所以一个页面下最多可以存放两个以太网数据包，一个数据包可以占用 2KB 的空间，起始地址为二分之一页的整数倍。为了实现页对齐我们采用数组填充，从 ykvm_buffer 中可以看到 buf_in[520]为我们硬性填充的数组。

内存中的页面不是所有的都被用来进行换入换出操作，我们在内核空间分配的页面应该常驻内存中，但我们使用 mmap 使其与用户空间发生联系，故存在着被置换出去的可能性，若不对缓存区所使用的页面进行锁定会导致系统运行出错，我们采用将其 SetPageReserved 将其锁住。

4.2.3 网卡与应用程序之间的同步

网卡与用户进程的同步，我们常用数组和索引来实现。在此只介绍接收数据包的工作流程。

我们将接收描述符数组分为两个队列，空闲描述符队列和忙碌描述符队列。空闲描述符队列管理着空闲的，可以被网卡驱动程序写入缓冲块对应的描述符；忙碌描述符队列管理着网卡已经写入完毕，能被用户进程读取的缓冲块对应的描述符。我们以 ykvm_rx_busy 和 ykvm_rx_free 表示。

无论是网卡向 ykvm 模块提供的共享缓冲区写入数据还是用户进程读出数据，它们需要得到共享缓冲区中的一块相应数据块的物理地址或用户地址。在共享缓冲区中，物理地址保存在 rx_bd 的 rx_bd_haddr 中，数据块的内核虚拟地址在 ykvm_rx_desc 的描述符字段 buf_addr 中，用户地址在其 user_addr 中，对描述符的查找可以根据描述符数组中的 rx_cons 和 rx_prod 索引来实现，如图 4-9 所示。

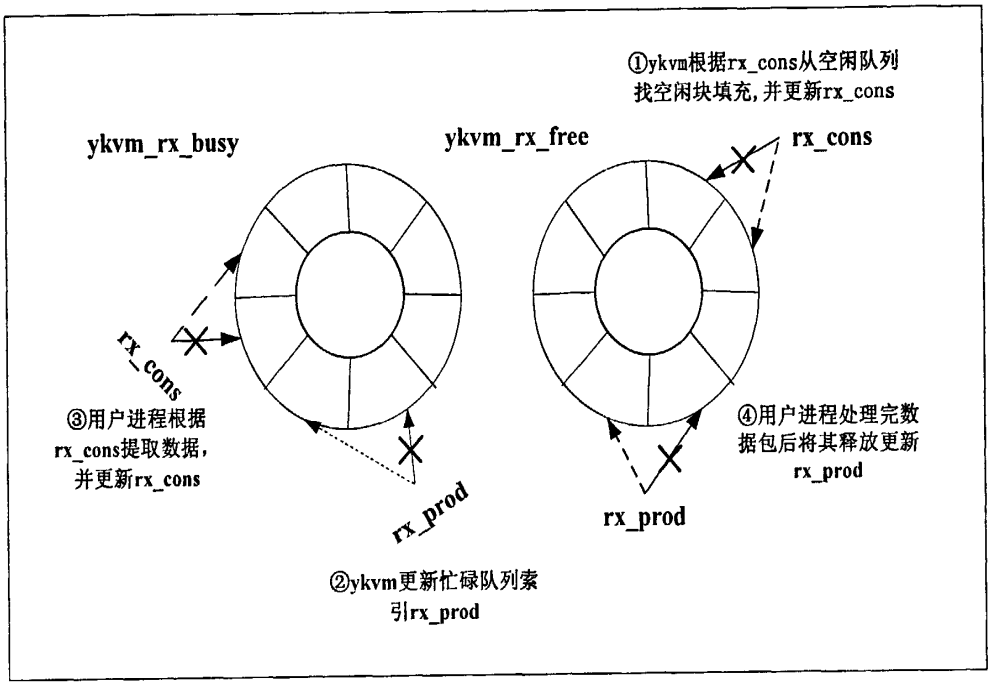

图 4-9 ykvm 模块同步机制

循环数组在 ykvm 及用户进程中的实现（下文中用 index++替代）：

if (unlikely(++index == MAX_TOTAL_RX_DESC_CNT-1)) index = 0;

其中 MAX_TOTAL_RX_DESC_CNT 为最大描述符的个数，x 和 index 分别为索引值。队列为空的时候：rx_prod == rx_cons。

当有数据到达时候，网卡根据保存在寄存器中的空闲队列索引 rx_cons 找到描述符，根据其提供的物理地址向空闲块中添加数据，然后将 rx_cons 其更新。

网卡寄存器索引被更新后会产生中断。此时调用 ykvm_enqueue()，根据 ykvm_rx_free 中的 rx_cons 索引值将数据包送入其对应的空间中并置 rx_cons++，然后更新 ykvm_rx_busy 队列描述符 rx_prod，使其指向更新空间，并置 rx_prod++；最后驱动及时更新其保存 ykvm_rx_free 的 rx_cons 索引和 rx_prod 索引值：NEXT_RX_BD(rx_cons) ，NEXT_RX_BD(rx_cons)。

当用户进程接收数据的时候，首先根据 ykvm_rx_busy 中的 rx_prod 和 rx_cons 索引值判断是否有数据，若数组不为空，根据 rx_cons 索引值找到数据包的用户地址取出操作，并置 rx_cons++。操作成功后，释放该数据包所在的内存地址，更新 ykvm_rx_free 中的 rx_prod 对应的空间，并置 rx_prod++。

在运行的过程中有可能遇到空闲描述符被使用完或者忙碌的描述符没有的情况。前者 ykvm 覆盖最早放入的数据包空间，即 ykvm_rx_busy 中的 rx_cons 空间；

后者将阻塞用户进程，等待 ykvm 写入数据后将其唤醒。具体的阻塞操作在 4.4 小节中介绍。

4.3. 接口函数模块

我们选择可编译的模块化方式的主要目的是为了让用户能够更灵活的选择其操作，但每个设备还需要提供给用户进程各种操作方法，比如打开，读取操作等等。因此，我们还需要在以上基础上建立一个用户接口模块。其功能在于创建一系列类似 libpcap 包提供给用户程序的接口函数。用户程序通过这些接口函数就可以完成相应的命令。注意，这里的接口函数模块其实是应用层完成的，而不是内核中。

我们对网络进行监控其实就是需要根据完整的数据帧中的各种信息：mac 地址，ip 地址，端口号和各种协议类型。比如：在 3.1.1 小节项目需求分析中需要记录每个员工的上网记录，我们首先需要通过 ip 包头判断目前抓取的数据包是否为 tcp 类型的数据包，然后根据 tcp 报文头判断目前抓取的数据包是否为 http 类型，然后对数据包的内容进行分析，将相应的 url 和 ip 包头中的数据信息填入数据库中。所以我们选择从底层捕获原始数据包，进行需要的操作。这些操作全部放到应用层中执行。应用进程从共享缓冲区中获的数据包为以太网帧，我们必须了解此帧的结构对其包头信息和数据信息进行分析。假定这个分析是通过一个用户函数来完成的，我们需要做的事情就是向这个函数可被分析的数据包。

为了避免应用层编程的繁琐性，我们把用户空间的映射和数据的提取及处理后的释放全部封装到此接口中。由于每次只处理一个数据包，为了循环处理数据包，我们借鉴 libpcap 提出的回调函数思想，定义两个函数接口来实现数据包的捕获。

我们定义 void yk_handle _xxx (const u_char * buf_data)为后期数据包处理的原型，此函数被 get_packet()函数调用。若数据包处理完成，需要在此函数中做相应的空间释放。

get_packet()的原型为：int get_packet(struct user_buff* yk_data, int length, yk_handle callback, char *err)。

用户只需要完成后期处理函数的实现，然后调用 get_packet 函数处理数据包。get_packet 调用 mmap 函数完成共享缓冲区的初始化，然后根据用户传入的字节数 length 循环获取数据。图 4-10 即为其流程图。

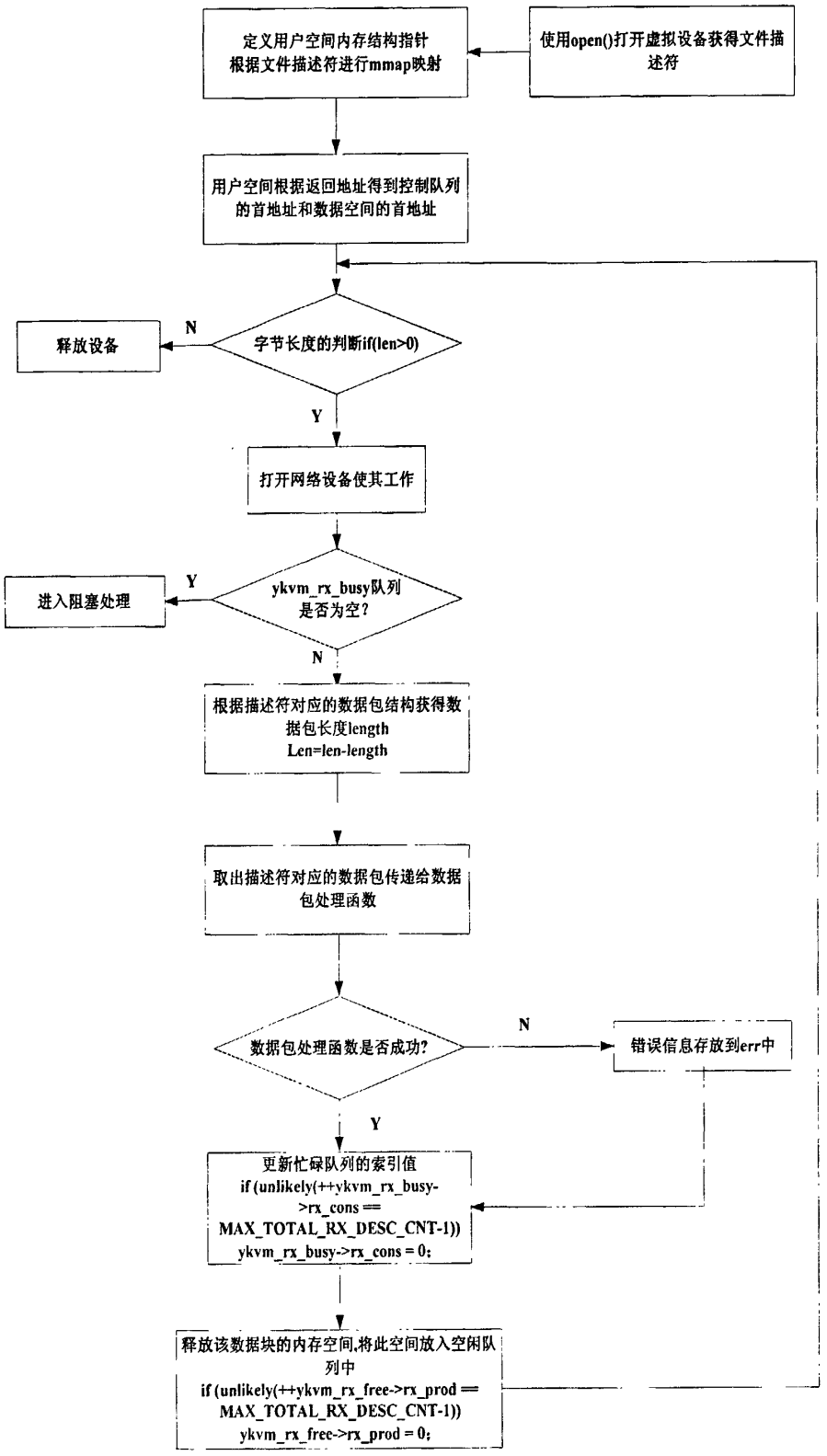

图 4-10 get_packet 函数的工作流程

yk_data 为指向用户空间的共享内存结构，通过用户空间提供的地址，我们与内核共享缓冲区进行映射。

length 为取出数据的长度，以字节为单位。当为-1 的时候默认为循环取出直至缓冲区为空。

callback 为数据包处理函数，此函数为用户自己实现的，其参数 buf_data 由 get_packet 函数提供。

err 为存储出错信息的数组，函数调用失败后的出错信息存放在此数组中。

4.4. 系统整体工作流程

本章节将介绍整个数据捕获系统的工作流程，其中包括初始阶段和运行阶段各个模块需要完成的工作以及它们之间的同步。

4.4.1 系统初始化阶段流程

第一，加载内核缓冲区模块 ykvm 和底层驱动模块 bnx2。两模块被成功加载后根据用户提供的共享缓冲区的大小 YK_NUM 分配一大片连续的空间并初始化。分配一大段连续空间中包括 ykvm_rx_desc 和 ykvm_buffer 各 YK_NUM 个。ykvm_busy_ring 和 ykvm_free_ring 各一个。初始化过程需要完成的操作如下：

对 ykvm_rx_busy 和 ykvm_rx_free 中的 rx_prod 和 rx_cons 进行初始化。初始状态下缓冲区为空，所以 ykvm_rx_busy 的 rx_prod 与 rx_cons 相等。将 ykvm_rx_free 中的结构体 ykvm_rx_desc[YK_NUM]中的所有描述符的 dma_addr 域和 buf_addr 域赋值。对于 dma_addr 域的赋值时候首先需要得到其逻辑地址 buf_addr。假设 i 为当前描述符的索引号，在 0 至 YK_NUM-1 之间，逻辑地址的获取办法是用数据块缓冲区的首地址 yk_buf+i*2048，一个数据包的存放空间为 2048 字节。然后通过虚拟页面到内核页面的转换（vmalloc 分配的内核空间地址转换见 4.2.2）得到物理地址 dma_addr。

第二，ykvm 模块将调用网络驱动程序的 open 函数使其工作。内核模块之间可以使用函数来通信，网卡驱动程序模块只能在 ykvm 模块完成空间的初始化之后才能使用，故我们可以通过调用网卡驱动程序的 open 函数来使网络设备工作。

第三，用户空间初始化相应的数据结构，通过接口函数 get_packet()完成内核空间到用户空间的映射。实际是对 mmap 函数的系统调用及描述符空间的赋值。

我们可以通过 mmap 调用获得的内存映射区域初始地址，对于用户进程而言

mmap 映射的空间是一片连续的空间，故根据 4.2.1 中的内核空间的分配和关键数据结构来计算数据空间中每个数据块的用户空间地址，然后填充到对应的用户描述符结构的 user_addr 中，以便以后用户进程根据此地址读取数据包。

4.4.2 系统运行阶段流程

网卡不断地往内核中送入数据，用户进程不断的读出数据。对于用户程序，当出现 ykvm 共享缓冲区的缓冲对列为空的时候，用户进程会被阻塞。我们在 ykvm 结构体中定义了读等待队列 r_wait，若共享缓冲区中没有数据包，使用 __set_current_state(TASK_INTERRUPTIBLE)将进程阻塞。等待忙碌队列中有数据流入时将其唤醒。若共享缓冲区中有数据包则循环取出数据包进行分析。

此外我们将设备设置为独占设备，每次只能有一个进程对其访问，其他进程的访问都会被拒绝。具体的实现是通过虚拟设备结构中设置一个互斥信号量 sem。我们在 ykvm 虚拟设备的加载函数中将其初化：init_MUTEX(&ykvm_dev->sem)。当设备被打开，down_trylock(&ykvm_dev->sem)，其他进程打开设备时由于此变量已经改变而返回错误。当一次访问结束后还需要释放该信号量 up(&ykvm_dev->sem)。用户接口工作流程图如图 4-11 所示。

在用户进程通过调用 get_packet()函数无限循环地读取数据包的情况下（length==-1），虚拟设备被一直占有，其他 无法访问。为了避免此种现象的发生或其他应急情况比如数据包处理时间过长而可能产生处理错误时，可以发送关闭信号，将进程关闭，释放相应设备。

网卡驱动及 ykvm 的工作流程如下图 4-12 所示。首先加载 ykvm 模块，在其 init 入口函数中静态分配一片连续空间；然后加载 bnx2 模块，通过 open 函数工作当数据包流入的时候，若有用户进程读入数据，则会唤醒该进程让其做相应的数据包处理工作。

若用户进程不能及时读取数据或处理速度赶不上数据包的流入速度，造成缓冲区为满的时候，ykvm 将忙碌队列 ykvm_rx_busy 中 rx_cons 索引对应的空间交给 ykvm_rx_free 中相应的 rx_prod 索引，并将自身的空间赋值成 NULL 并将 rx_cons 进行加 1 操作，然后 ykvm 根据 rx_cons 索引将此空间直接覆盖（队列为空状态下 rx_cons 与 rx_prod 相等，当数据空间被赋给空闲队列的 rx_prod 索引时其实就赋给了 rx_cons 索引），覆盖成功后两索引值同时加 1 操作。若空闲队列为空，则循环进行上述操作。

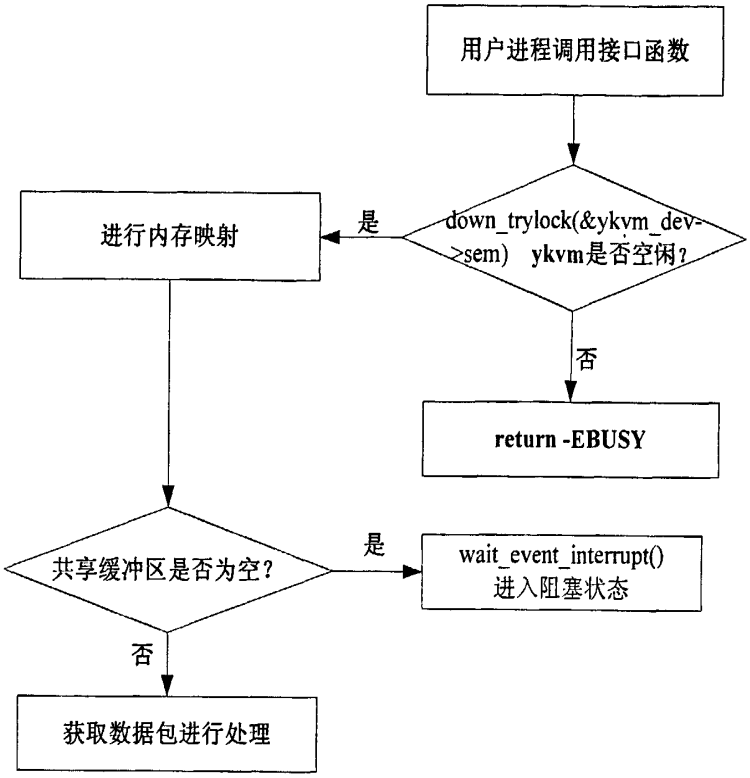

图 4-11 用户接口的工作流程

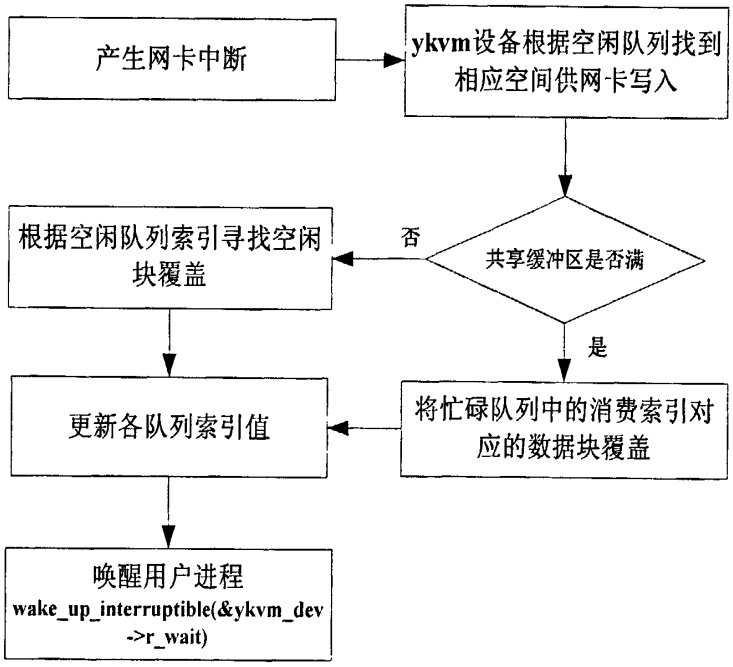

图 4-12 内核模块的工作流程

4.5. 本章小结

本章首先介绍了数据包捕获系统中三大模块：网卡驱动模块，ykvm 模块，用户接口模块的功能及实现中的难点。然后分别对这三个模块的实现做了详解。主要包括网卡驱动模块需要修改的地方，在 ykvm 模块中对实现零拷贝的内存映射 mmap 机制的具体实现和实现中需要注意的问题做了介绍，在用户接口模块中利用回调函数的形式提供给用户进程一个接口函数。最后介绍了三大模块在初始化及运行阶段的同步。

第五章 基于千兆网卡的数据包捕获系统的测试与分析

5.1. 测试环境

此章节我们搭建测试环境对数据包捕获系统进行测试，并将原始的 libpcap 捕获方式与我们的捕获子系统进行性能上的比较。在测试系统时，我们在用户端调用 mmap 进行映射，循环获得数据包后计数，然后将其对应空间及时释放不做其他处理。下面介绍测试环境。

我们将系统与千兆交换机直接连接，通过发包机发送数据模拟高流量网络环境。由于实验环境有限，使用的发包机并非硬件发包机器。我们在每台机器上安装 PKTGEN 内核模块进行发包。我在发包过程中调整发包时间间隔和报文长度来记录相应的数据包接收速率，每秒接收流量等。具体的拓扑结构如图 5-1 所示：

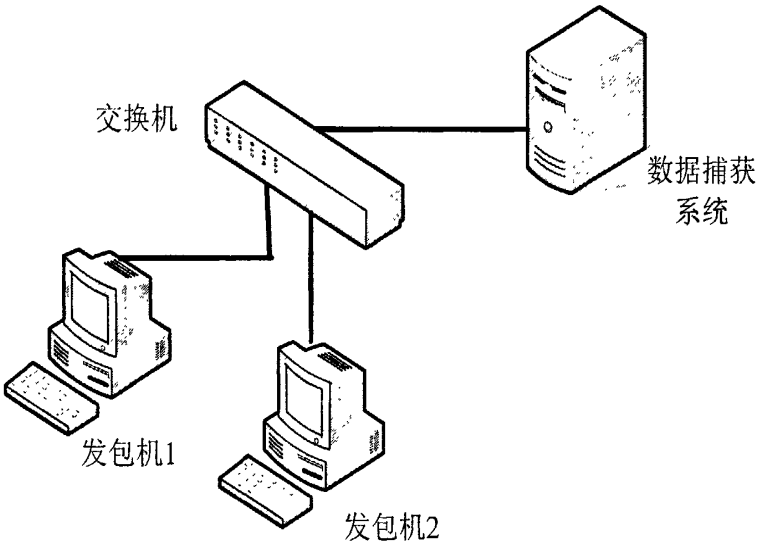

图 5-1 数据包捕获系统的测试部署

数据包捕获系统的硬件配置为双 Intel xeon5404 3.0G，内存 4G。根据企业内部流量的分析报告，下班时间（0：00-9：00，19：00-24：00）平均流量维持在 20MB/s；上班时间（9：00-19：00）流量的峰值为 67MB/S，平均值为 60MB/S。我们根据这些数据来确定测试过程中的发包速率，尽可能的模拟真实环境。

实验中发包机对长度为 64 字节，512 字节，1500 字节的数据包进行发包处理，在发送 64 字节数据包时候，发包机器的最高发送速度达到为 311,427,072bps，在发送 512 和 1500 字节数据包的时候，发包机器的最高发送速度分别为：883,550,208bps 和 991,952,896bps。我们可以调整发包时间间隔来控制发包速度，通过两台发包机同时往测试系统发包来虚拟高速网络流量环境。

发包/收包速度 pps 为每秒钟发送或接收数据包的个数；发包流量 bps 为每秒钟发送的字节数。

5.2. 测试数据的分析

数据捕获系统接收不同字节长度数据包的测试结果如下表 5-1，5-2，5-3 所示：

表 5-1 数据捕获系统接收 64 字节长的数据包

发包机 1 速度（pps）	发包机 1 流量（bps）	发包机 2 速度（pps）	发包机 2 流量（bps）	收包速度（pps）	收包流量（bps）
325,632	166,723,584	331,776	169,869,312	655,449	335,589,888
489,472	250,609,664	493,568	252,706,816	979,976	501,747,712
569,344	291,504,128	581,632	297,795,584	1,134,592	580,911,104

表 5-2 数据捕获系统接收 512 字节长的数据包

发包机 1 速度（pps))	发包机 1 流量（bps）	发包机 2 速度（pps）	发包机 2 流量（bps）	收包速度（pps）	收包流量（bps）
40,448	165,675,008	40,960	137,772,160	81,355	333,230,080
61,440	251,658,240	61,952	253,755,392	123,403	505,458,688
71,680	293,601,280	72,192	295,698,432	143,784	588,939,264

表 5-3 数据捕获系统接收 1500 字节长的数据包

发包机 1 速度（pps）	发包机 1 流量（bps）	发包机 2 速度（pps）	发包机 2 流量（bps）	收包速度（pps）	收包流量（bps）
13,894	166,728,016	13,481	161,772,000	27,319	327,828,000
21,059	252,707,984	20,972	251,664,032	41,969	503,628,000
24,816	297,791,978	24,467	293,604,074	49,245	590,940,000

我们调整两台发包机器的发包流量分别为 20MB/S，30MB/S，35MB/S 三个阶段。表 5-1 中，当两台发包机各自流量在 20MB/S 左右时，捕获系统接收流量约

40MB/S；随着发包机流量的增大到 30MB/S 时，捕获系统的接收流量为 60MB/S 左右；待发包流量上升到 35MB/S，捕获系统的接收流量大概为 69.25MB/S。可见当发包机速度上升到 35 MB/S，捕获系统的收包速度会受到一定影响，虽然不能满收包，但是丢包并不明显。

表 5-2 和 5-3 中，发包机器发送较大字节的数据包时候，捕获系统的收包速度都很稳定。可以看出，在网络流量较大的情况下，对捕获系统性能的评估的关键在于系统对短包的接收率。

使用 libpcap 数据不同字节长度数据包的测试结果如下表 5-4，5-5，5-6 所示：

表 5-4 libpcap 接收 64 字节长的数据包

发包机 1 速度（pps）	发包机 1 流量（bps）	发包机 2 速度（pps）	发包机 2 流量（bps）	收包速度（pps）	收包流量（bps）
325,592	166,703,104	331,865	169,914,880	178,894	91,593,728
489,450	250,598,400	493,589	252,717,568	182,272	93,323,264
569,401	291,533,312	581,592	297,775,104	180,329	92,328,448

表 5-5 libpcap 接收 512 字节长的数据包

发包机 1 速度（pps）	发包机 1 流量（bps）	发包机 2 速度（pps）	发包机 2 流量（bps）	收包速度（pps）	收包流量（bps）
40,468	165,756,928	40,876	167,428,096	81,305	333,025.280
61,425	251,596,800	61,952	253,755,392	123,359	505,278,464
71,680	293,601,280	72,194	295,706,642	143,786	588,947,456

表 5-6 libpcap 接收 1500 字节长的数据包

发包机 1 速度（pps）	发包机 1 流量（bps）	发包机 2 速度（pps）	发包机 2 流量（bps）	收包速度（pps）	收包流量（bps）
13,904	166,848,000	13,479	161,748,000	27,378	328,536,000
21,059	252,708,000	20,966	251,592,000	41,985	503,820,000
24,823	297,876,000	24,452	293,424,000	49,267	591,204,000

通过表 5-4 我们可以看出，随着两台发包机的发包速度的提升，目标机的接收流量并没有相应的增大，在发包机的发包流量达到 20MB/S 时候，目标机的接收流量在 10MB/S 左右；其后发包机的流量上升到 30MB/S 和 35MB/S 的时候，目标机器的接收流量并没有很大的提升，维持在 11MB/S 左右，可见丢包率很严重。

从表 5-5,5-6 中我们可以看出，目标机的收包流量都很稳定。故 libpcap 对中长

包的捕获比短包的捕获效果要好很多。

对比数据捕获系统与 libpcap 的测试数据，我们可以归纳出以下三点：

第一，捕获系统和 libpcap 处理短包的能力不及处理中长包的能力。

第二，libpcap 在处理短包时的极限速度为 93,323,264bps，约 11.13MB/S，远远达不到企业的网络环境要求。

第三，ykvm 在处理短包的能力上相比 libpcap 有优势，当发送速率增长的时候，优势更加明显，能够达到 580,911,104bps，约 69.25MB/S 超过企业流量峰值 67MB/S，满足企业内部网络的要求。

5.3. 本章小结

本章介绍了数据包捕获系统的测试网络环境测试环境的搭建，对测试数据进行了相应的分析，验证了千兆网络环境下的数据包捕获系统对短包捕获的优越性。

第六章 总结与展望

6.1. 全文总结

网络的高速发展伴随着带宽的增加，信息传输速率的提高。当上网的硬件设施得到提高时，网络用户的增加和网络环境的复杂却侧面给我们带来了各种安全隐患。无论是个人用户使用的杀毒软件，还是企业用户使用的网络监控系统或者入侵检测系统，都是目前最常见的保障个人用户或企业内部网络安全的一种方式。而上述系统全部基于网络数据的捕获。数据捕获是各种网络安全软件实现的基础，特别是在网络传输速度急速提高的今天，如何在高速网络环境中提高数据捕获的性能成为了研究的一个重点。本篇文章主要介绍了数据捕获领域的发展现状，分析了各种捕获技术，并将一种提高数据捕获性能的零拷贝思想运用于实际的网络行为监控系统的开发中。以下为本人所做的工作：

研究了目前网络环境的现状，明确了数据捕获在网络各种上层应用，网络安全等领域的基础性和重要性。

研究了传统的基于软件架构的数据捕获方法并分析了其各种弊端，了解了现有的各种改进的数据捕获方式，为改善网络行为监控系统的数据捕获性能打下了基础。

根据零拷贝思想将数据捕获系统划分为三个功能模块实现，对系统进行了详细设计。最后以可加载模块的形式实现了数据包的抓取，并为上层应用程序提供了数据捕获的接口。

6.2. 工作展望

在本论文的研究过程中，由于作者水平有限及硬件设施的不齐备，尚有些功能可以在未来的工作中完成。

第一，本系统是针对高速网络环境下的数据包捕获。共享缓冲区的大小对数据包捕获性能也有一定影响，我们可以尝试将共享缓冲区建立到用户空间，然后灵活的控制其大小，进行针对性测试，并比较其与建立在内核空间的方案的区别。

第二，我们还可以采用流量分析，使系统管理员能够灵活的切换网卡的中断模式，以提升数据包的捕获性能。

第三，对捕获到的数据包处理不够理想。其实可以根据系统需求分析，定义数据包的封装形式，借鉴原有的网络协议栈框架，设计一个精简的专用协议栈。

致 谢

首先感谢我的导师郝玉洁教授从论文开题到完成过程中给予我全面，具体的指导和帮助。郝老师为人热情善良，治学严谨，诲人不倦，使我在学业上能够不断进步，而且还为我提供了大量的实践机会。在此，谨向恩师致以崇高的敬意和深深的谢意！

还要感谢胡德昆师兄在项目开发过程中凭借自己丰富的实践经验和高超的工作能力提供给我的宝贵建议和指导。为此，我对胡德昆师兄深表谢意！

还要感谢韩巧玲，刘亦男，郝杰，向波，芦志朋，阳丽。与你们共处的三年让我难以忘怀，有了你们的陪伴和帮助，我这三年的学业生活充满了乐趣。

还要感谢养育了我 25 年的父母，在你们的支持下我能够顺利地完成这二十年的学生生涯。

还要感谢我的男友一直以来对我的鼓励和支持。

最后向在百忙之中为我评审论文及参加答辩的专家们表示感谢！

参考文献

[1] CNNIC. 第 23 次中国互联网发展报告. http://it.sohu.com/s2009/cnnic23/, 2009

[2] 左治勤, 张良. 对 SOA 业务敏捷与敏捷企业的探讨. 软件导刊，2008，7(1):19-21

[3] 杨久龙. ERP 系统在国有企业的运用. 上海微型计算机，1999，52(12)：28

[4] V.JACOBSON, C. LERES, S. MCCANNE. Libpcap. http: //www. tcpdump.org, 2008

[5] Mogul JC, Ramakrishnan KK. Eliminating receive livelock in an interrupt-driven kernel. ACM Transactions on Computer Systems, 1999, 15(3):217-252

[6] A. Foong, T. Huff, H. Hum. TCP Performance Re-Visited. IEEE Symposium on Performance of Systems and Software, 2003, 70-79

[7] R. morris, E. Kohler, J. Jannotti. The Click Modular Router. ACM Transactions on Computer Systems, 1997, 15(3):217-252

[8] Luca Deri. Improving passive package capture: beyond device polling, http://luca.ntop.org/R_ing.pdf, 2004.

[9] 顾瑞春, 谭跃生, 贾元春等. Linux 系统中基于 PF_RING 套接字的高性能包捕获技术.内蒙古大学学报, 2007, 26(2):152-156

[10] Burd, Dana. zero-copy interfacing to TCP/IP. Dr. Dobb's Journal, 1995, 20(9): 6

[11] Luca Deri. nCap: Wire-speed packet Capture and Transmission. Third IEEE Internationals Workshop on End to End Monitoring. 2005.

[12] Wind River System Inc. TCP/IP Offload Introduction Datasheet. http://www.win driver.c_om/products/tina/tcpip_offload.pdf

[13] DAG cards.Endaeeeorporation. http://www.endace.com.

[14] 杜英明. 高速网络环境下基于零拷贝的捕获机制研究与实现[硕士学位论文]. 长沙：国防科学技术大学, 2007, 7

[15] James F. Kuros, Keith W. Ross.计算机网络——自顶向下方法与 Internet 特色. 北京：清华大学出版社, 2003, 367-369

[16] Guo Chuan-xiong，Zheng Shao-ren. Analysis and Evaluation of the TCP/IP Protocol Stack of Linux. Communication Technology Proceedings, 2000, 444-453

[17] 陈建明. 高速网络数据的存储及内容监控:[硕士学位论文]. 福州：福州大学, 2005, 18

[18] 杨洪. 数据链路提供者接口 DLPI 及应用. 计算机工程与应用——网络与系统集成, 1997, (10): 10-12

[19] 胡文伯. 网络流量分析研究中的应用层协议分析的设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2008, 18-21

[20] 刘峰. Linux 环境下基于 Intel 千兆网卡的高速数据包捕获平台的研究[硕士学位论文]. 厦门: 厦门大学, 2008, 10-11

[21] K. Salah, K. EL-Badawi. Analysis and Simulation of interrupt Overhead impact on OS Throughput in High-Speed Networks, International Journal of Communication Systems,2005, 18(5):501-526

[22] K. Salah, A.Qahtan. Implementation and experimental performance evaluation of a hybrid interrupt-handling scheme. Computer Communications, 2009, 32(1):179-188

[23] 乔思远. 基于 DMA_ring 的高速网络报文捕获机制的实现及应用[硕士学位论文]. 青岛: 山东大学, 2007, 23

[24] Tian-hua, Liu, Hong-feng Zhu, Gui-ran Chang. The design and implementation of zero-copy for Linux Proceedings. 8th International Conference on Intelligent Systems Design and Applications,2008, 121-126

[25] Tian-hua Liu, Hong-feng Zhu, Gui-ran Chang. Research and implementation of zero-copy technology in Linux.2006 IEEE Sarnoff Symposium, 2006

[26] 张曦煌, 柴志雷. Linux 中 TCP/IP 协议实现及嵌入式应用. 北京航空航天大学出版社, 2008, 162-210

[27] Michael Barabanov. A Linux-based Real-Time Operating System. Http://www.fsmlabs.c_ om/devel-opers/white-papers,1997.

[28] Finite State Machine Labs, Inc. RealTime Linux (RT Linux). http://www.fsmlabs.com/.

[29] E. Gregory, L. Brian Evans. Real-time sonar beam forming on workstations using process networks and POSIX threads. IEEE Transactions on Signal Processing, 2000,921-926

[30] 涂刚. 软实时系统任务调度算法研究[博士学位论文]. 武汉: 华中科技大学, 2004, 77-78

[31] Mckenney Paul, E. Brown, Angela Demke, Walpole Jonathan. Performance of memory reclamation for lockless synchronization. Journal of Parallel and Distributed Computing, 2007, 67(12):1270-1285

[32] A. Biswas, P. Sinha. High Performance Packet Capturing Support for Alarm Management Systems. Parallel and Distributed Computing and Systems Proceedings of the 17th IASTED International Conference. 2005

[33] Jonathan Corbet, Alessandro Rubini, Greg Kroah-hartman. LINUX 设备驱动程序. 北京：中国电力出版社, 2006, 213-231

[34] Michael Beck, Hrald Bohme, Mirko Dziadzka 等.Linux 内核编程指南. 北京：清华大学出版社, 2004, 56

[35] Liu Wei, Zheng Wei—Min, Shen Jun, Ju Da-Peng. Design and implementation of memory map mechanism in underlying communication. Journal of Software, 1999, 10(1):24-28

[36] M. Welsh, A. Basu, T. von Eicken. Incorporating memory management into user-level network interfaces. Cornell University Ithaca, NY, USA:Technical Report 1997,97-1620

[37] 毛德操, 胡希明. Linux 内核源代码情景分析(下册). 杭州:浙江大学出版社, 2001.119-600

[38] 杨树青, 王欢. Linux 环境下 c 编程指南. 北京：清华大学出版社, 2007, 225-311

[39] Jonathan Corbet, Alessandro Rubini, Greg Kroah-hartman. LINUX 设备驱动程序. 北京：中国电力出版社, 2006, 137-173

[40] 郑卫兵, 丁会宁, 李庆海. LINUX 的网络转发性能研究. 西安交通大学学报, 2004, 38（2）：124-127

[41] W.Richard Stevens, Stephen A.Rago. UNIX 环境高级编程(第 2 版). 北京：人民邮电出版, 2007, 66

[42] 王佰玲, 方滨兴, 零拷贝报文捕获平台的研究与实现. 计算机学报, 2005, 28(1)：48-51

攻硕期间取得的研究成果

[1] Zhang Rui, Hao Yu-Jie, Zhang Wei-Hua, Han Qiao-Ling, Lu Zhi-Peng. Muti-Biometric Feature Identification Technologies based on Visual Perception and Cognitive Mechanism. International Conference on Apperceiving Computing and Intelligence Analysis 2008(ICACIA'08):232-235, Accession_ Number: 20092412118474

[2] 山东兖矿国泰集团公司项目“信息安全创新项目”，2008 横 344 号，2008.8-2009.5

[3] 成都市科技一条街 IT 促进会项目“成都市科技一条街数字化建设”，SCSZDJS-2, 2007.8-2008.5