基于网络处理器的DDoS入侵防御控制系统

专业：　　无线电物理

硕士生：　　杨柳清

指导老师：　余顺争教授

摘要

随着网络应用的普及和发展，网络系统面临越来越大的挑战。层出不穷的各种攻击方式成为网络安全的巨大隐患。DDoS（Distributed Denial of Service）是其中一种威力极大的攻击，并由于其攻击源隐蔽性强等特点，尚不能得到有效防御。而现有的入侵检测系统对可疑流量的处理一般都限于简单的丢弃或过滤，易于发生误判漏判。

针对目前的网络安全状况和发展需求，本文实现了一个基于网络处理器Intel IXP2400的用于服务器前端的入侵控制系统。该系统对不同用户的流量进行分类，为正常程度较高的流量提供较高优先级的服务，为正常程度比较低的流量提供低优先级的服务，而低正常程度的流并非完全地被丢弃。本系统降低了对检测模块准确性的要求，减少了因误判漏判带来的影响，为大型网站防御DDoS攻击提供了实用性方案。

关键词：DDoS攻击 队列调度 拥塞控制 网络处理器

A Network Processor Based Control System for DDoS Intrusion Prevention

Major: Radio Physics

Name: Liuqing YANG

Supervisor: Professor Shunzheng YU

Abstract

As the Internet becomes more and more easy to access, Distributed Denial of Service attack (DDoS) pose an immense thread to the Internet, because of the difficulty to identify the attack patterns and locate the real attack source. While many existing schemes focus on detecting attack traffic, few are done to control it after detection. Most schemes simply drop the suspicious packets, whose performance rely on the precision of the detection heavily. In this paper, a network processor based control system for DDoS intrusion prevention is implemented. This system assigns priorities to different network traffic according to the normality. The traffic of higher normality will be served with higher bandwidth, and the traffic of lower normality will be served with smaller bandwidth instead of dropping immediately. This system can reduce the influence of misdsection, and network processor platform will enable the system to meet the needs of the fast developing Internet.

Key words: DDoS Attack, Queue Scheduling, Congestion Control, Network Processor

第1章 绪论

1.1 选题背景

网络时代的来临，使人们的工作和生活越来越方便，也使人们的工作和生活越来越依赖于网络。而随着网络应用的普及和网络速度的大幅度提升，网络系统由于其脆弱性越来越容易受到攻击。近几年来新出现的 DDoS（Distributed Denial of Service 分布拒绝服务）攻击更是严重地威胁了网络系统的安全。DDoS 攻击曾经令 Yahoo!、eBay、Amazon、CNN 等众多知名站点系统瘫痪达几个小时甚至几十个小时之久，而发生在 2003 年 1 月 15 日的 DDoS 攻击甚至造成北美[1]，欧洲和亚洲的互联网交通发生大面积堵塞，至少 2.2 万台网络服务器和 25 万台计算机遭到攻击，受灾最严重的韩国甚至举国网络瘫痪 24 小时之久。DDoS 攻击可以说是目前对互连网的安全和稳定破坏力最强的攻击。在典型的 DDoS 攻击中，攻击者控制大量的傀儡主机向被攻击的目标服务器发送大量的无用分组，从而大量地消耗目标服务器的资源和网络带宽，以致正常用户的正常请求得不到响应，造成“拒绝服务”的现象。由于 DDoS 攻击的分布式特性，攻击工具提供了伪造源地址， 随机变化分组内容，任意指定的通信端口和通信协议等多种手段，使得 DDoS 攻击的特征越来越难以提取，给 DDoS 攻击的检测防御和跟踪带来极大困难。

近年来 DDoS 攻击已经引起学术界和商业界的广泛重视，不少学者和研究人员纷纷进行了 DDoS 攻击的检测和防御方案的研究。目前提出的方案大致可分成三个类别：IP Trace back（攻击源追踪）、攻击检测与控制、Rate Limiting（速率限制）。其中[2]IP Trace back 技术，主要原理是根据攻击分组定位攻击的源头，从而定位出攻击流的真正源头，为基于源头的防御和追踪黑客提供支持；攻击检测与控制方式，主要通过对分组进行分析和识别，与正常的或合法的模式进行对比，识别出可疑的分组并采取相应的措施进行控制；Rate Limiting 则力图通过对识别系统识别出的恶意数据流，根据一定策略对该数据流进行过滤从而达到限

速的目的。而目前商用中比较广泛使用的主要是数据流指纹检测技术。数据流指纹检测技术实质上就是匹配分析法，通过匹配到达数据流与 IDS 中的正常数据流的特征值来判定数据流的合法性，检测出异常流量。

现有的实用型系统中，对 DDoS 的入侵解决方案往往将重点放在对 DDoS 攻击的检测和追踪上，而在检测和追踪模块之后的控制模块功能一般都很简单。在这些系统中，往往只区分出正常流量和异常流量，对异常流量进行简单的丢弃。这种粗粒度的控制方式，对判决门限有着非常严格的要求，因为一旦判决门限稍有偏差，就会出现误报而大大影响正常用户。而一旦存在误判，正常用户的流量将不能得到任何保证，攻击流量就可能造成很大危害。可见，现有的入侵检测和控制系统不足以有效地抵御大规模的 DDoS 攻击。

另一方面，近年来的网络应用的发展对网络设备提出了支持高速分组处理的优异性能和支持不断变换高层网络服务的高灵活性的要求。按照摩尔定律[3]，电处理器处理速度每 18 个月增加一倍；而随着 DWDM 等光纤技术在主干网络的广泛应用，每 12 个月光纤链路容量就增加一倍。因此以电处理器为核心的路由器仍然是网络发展瓶颈。而 Internet 的爆炸性增长，网络应用范围不断扩大、新型业务不断涌现，导致新协议不断出现，对服务质量和安全性能的要求越来越高。网络的发展要求网络设备能够在网络 2～7 层上对高速数据流进行细化分组分类处理，而不仅是在网络 2～3 层上进行数据流的简单存储转发处理。而数据分组处理涉及层次越多，系统资源负荷开销就越大。传统的基于 GPP(General Purpose Processor)的网络设备只满足灵活性要求，而基于 ASIC(Application Specific Integrated Circuit)的网络设备只满足高性能要求，两者都不能满足网络高速发展的需求。

1.2 论文的主要成就

本文是国家自然科学基金项目——网络时空行为与 2008 奥运会网络安全关键技术研究的一个子课题，旨在用基于 Intel IXP2400 的网络处理器实现一个用于服务器前端的流分类和入侵检测控制系统。本系统位于受保护服务器的前端，从用户分类出发，有效地对不同级别的用户进行区分和提供优先级服务，并通过拥塞控制和队列调度机制控制服务器所接收到的流量，能有效地保证正常流量获

得应得的服务，限制攻击流量。同时，由于采用网络处理器开发，得益于网络处理器的高速处理能力和灵活性，使得本系统面对泛洪攻击具有更强的抵御能力。本系统利用了网络处理器本身对队列的底层支持，体现了网络处理器为核心的下一代网络设备所必须具备的高性能和灵活性特点，能较好满足了未来网络和市场对网络设备的技术需求。本系统的实现为大型网站的 DDoS 防御提供了实用性方案。

1.3 章节安排

本文共分为七章。第一章为绪论，对论文的选题背景，意义以及主要工作进行介绍。第二章主要介绍了 DDoS 攻击的原理和现有的一些检测防御技术。第三章是对本文所实现的系统中用到的流量控制技术进行介绍。第四章介绍了网络处理器技术。第五章是本论文的重点章节，详细介绍了基于网络处理器的 DDoS 入侵控制系统的设计与实现。第六章则给出了本系统的部分测试结果及其分析。最后一章是对本文的总结和后续工作介绍。

第2章 DDoS 攻击及检测防御技术

2.1 DDoS 的概念

分布式拒绝服务攻击（DDOS）其实是拒绝服务攻击（DOS）的一种。DOS 是现今 Internet 上主要的攻击方法之一，其目的在于通过大量地消耗服务器的资源或者网络带宽，或者利用服务器软件上的漏洞，使网络或者服务器瘫痪，致使 Internet 上的服务器和主机不能正常地提供和获得服务。

拒绝服务攻击大体上可以分成两大类。一类是利用大量的数据包来充斥带宽和消耗服务器资源的攻击。一类是服务器系统，软件，或者协议本身的漏洞和配置上的缺陷被攻击者利用而产生的攻击。

分布式拒绝服务攻击（DDOS）是在 DOS 基础上发展而来的。但与普通的 DOS 不同的是，DDOS 集合大量性能较弱的计算机同时对性能优越的服务器发起攻击，通过这些主机同时对服务器发出海量的数据流消耗服务器的系统资源以及带宽资源。因此 DDOS 不需要依赖系统或者协议的漏洞，只要攻击者所控制的主机资源足够多，就可以令目标服务器拒绝服务甚至崩溃。

2.2 DDoS 的攻击原理

一个攻击者要进行一次 DDOS 攻击，必须拥有以下三种主机：

(1) ATTACKER：攻击者的主机，攻击者通过它来入侵其他主机，植入 DDOS 工具，以及发号施令，控制整个攻击流程。

(2) 主控端：直接受攻击者控制的主机，但未必为攻击者所拥有，攻击者在这些计算机上安装上特定的主控制软件（主控端）.这类主机类似于攻击者和真正进行攻击的主机之间的传令官，负责把攻击者的指令传达给受控的攻击主机。

(3) 攻击端：真正负责攻击的主机。这样的计算机数以百计，攻击者在这些计算机上安装了守护程序(称为攻击端)，运行并产生 DDoS 攻击代码。这

些主机将在收到主控端的命令后发送大量的攻击数据包或者采取其他行动。[4]

DDOS 的攻击模型如下图：

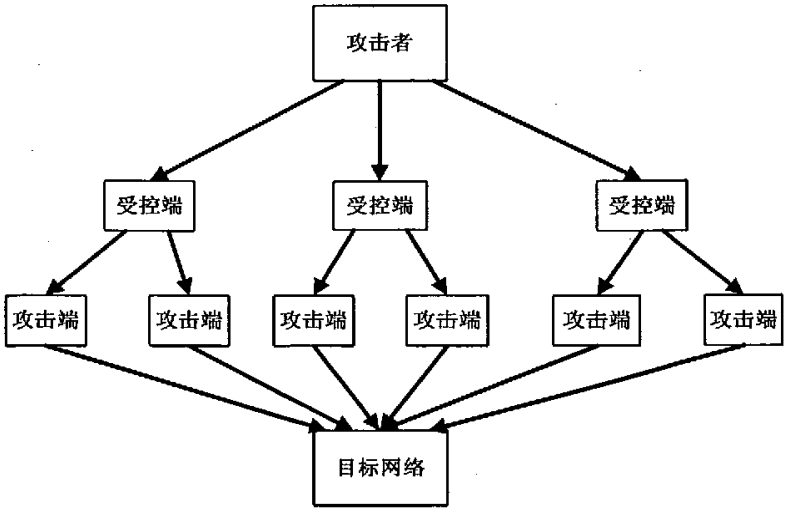

图 2-1 DDoS 攻击模型

要进行一次的 DDOS 攻击，攻击者必须控制大量的主机，因此进行攻击是从寻找有漏洞的主机开始的。一次完整的 DDOS 攻击一般有以下几步：

(1) 扫描主机。借助一系列安全漏洞扫描工具 ，在 Internet 上寻找某些安全性脆弱计算机，以便用这些主机充当主控端（受控服务器）和攻击端（攻击服务器） 。

(2) 入侵主机。对扫描所获得的可入侵主机，攻击者通过各种方法试图获得其控制权。一旦获得控制权，攻击者就在这些主机上安装 DDOS 攻击工具，使其成为主控端或者攻击端。攻击端主机上的守护程序在指定端口上监听来自主控端主机发送的攻击命令，而主控端主机接受从攻击者的 ATTACKER 计算机发送的指令。为了安全起见，攻击者会在主控端 和攻击端计算机上安装 ROOT KIT 程序，使主控制软件和守护程序本身不被主控端和攻击端计算机的管理员发现。

(3) 进行攻击或者继续入侵。攻击者可以通过 TELNET 到主控端主机上面控制攻击端进行 DDOS 攻击或者继续扫描和入侵更多的主机。

2.3 DDoS 的种类

大体上说，DDOS 攻击可以分成两类：直接攻击和反弹攻击（reflector attack)。

(一) 直接攻击

直接攻击的模型如图 2-2 所示：

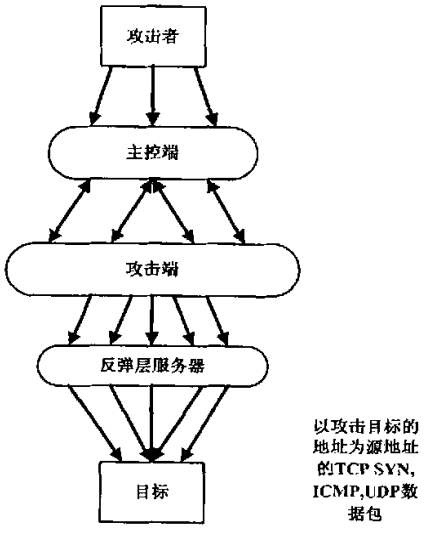

图 2-2 DDOS 直接攻击

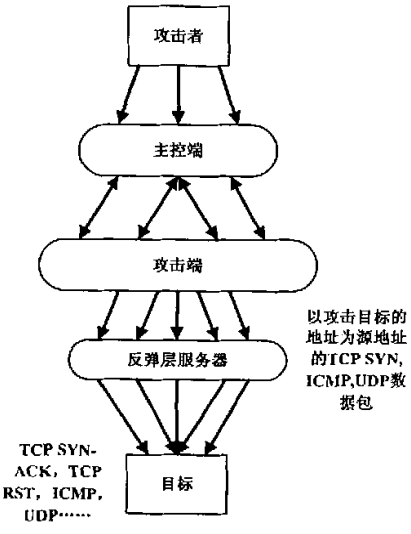

图 2-3 DDOS 反弹攻击

直接攻击往往是由攻击者指挥大量的受控端主机直接向攻击目标发送大量的攻击数据包。

(二) 反弹攻击

反弹攻击的模型如图 2-3 所示

反弹攻击跟直接攻击的区别在于反弹攻击多了一个反弹服务器层。所谓反弹服务器（reflector）是指当收到一个请求数据包后就会产生一个回应数据包的主机。例如，WEB 服务器，DNS 服务器和路由器等，这些服务器会对 SYN 报文或其他 TCP 报文回应 SYN |ACKs 或 RST 报文，以及对一些 IP 报文回应 ICMP 数据包。而攻击者可以利用这些回应的数据包对目标服务器发动 DDOS 攻击。发动攻击时，攻击者不是指挥受控端直接向目标发送数据包，而是指挥受控端向大量的反弹服务器发送伪造的源地址为攻击目标的请求分组。当反弹服务器收到这些分组之后，就会向攻击目标发送大量的响应分组，从而达到 DDOS 的攻击目标。

2.4 DDoS 的具体攻击方法

DDOS 的攻击方法可以分成两类：泛洪攻击（Flood Attack）和畸形包攻击(Malformed Packet Attack)。这其实也就是上面提到的两大类 DOS 攻击，只不过在 DDOS 中也继承了这两类攻击，而且威力更为强大。泛洪攻击就是纯粹的资源比拼，以大量无用的请求或者无用的数据包堵塞带宽，消耗服务器的系统资源。而畸形包攻击就是针对漏洞发送畸形包，使得服务器崩溃。而在 DDOS 攻击中，泛洪攻击的优势和威力尤其明显，所以这是目前 DDOS 攻击的主流。

(一) 泛洪攻击

泛洪攻击细分有 Smurf 泛洪，TCP SYN 泛洪，UDP 泛洪，ICMP 泛洪以及应用层攻击几种。以下就这几种逐一进行介绍。

(1) Smurf 泛洪

攻击者以攻击目标的地址为源地址，以某一网段的广播地址为目标地址，发送一些伪造的 ICMP ECHO REQUEST 数据包。如果这个网段不禁止这种形式的广播包的话，这个网段上的所有主机收到这个 ICMP ECHO REQUEST 包的时候就都会对攻击目标发送 ICMP ECHO REPLY 的数据包，从而快速地耗尽攻击目标的带宽，使其他合法用户无法正常访问该目标。

(2) TCP SYN 泛洪

SYN 泛洪实际上是利用了 TCP 三次握手中的漏洞进行的攻击。攻击者向攻击目标的服务器发送大量伪造源地址的 SYN 请求包。服务器收到这些连接请求之后，会向这些伪造的源地址的主机发送 SYN|ACK 包。由于这些主机并没有真的向服务器发送请求，因此不会响应服务器的 SYN|ACK 包。于是服务器就不得不保存越来越多的这种半打开状态的连接，等待一些永远不会到来的连接确认。最后服务器的内存和 CPU 时间就会被这种等待的队列占满，于是合法用户就无法与服务器建立连接，从而达到拒绝服务的攻击目的。

由于这种伪造的 SYN 包与普通的连接请求看上去没有什么两样，而且伪造的源地址为这种攻击提供了更好的隐蔽性。因此服务器极难把这种攻击流量与正常的请求流量区分开来。

(3) UDP 泛洪

与 TCP 不同，UDP 是一种无连接的传输协议，因此也不需要三次握手建立连

接等过程。当提供基于 UDP 的服务的服务器收到一个 UDP 数据包的时候，会将这个数据包的目的端口跟自己提供服务的端口比较，如果这个 UDP 包不是这个端口的，服务器会向客户端返回一个 DESTINATION PORT UNREACHABLE 的 ICMP 包。攻击者正是利用这一点，向服务器的 UDP 端口随机地大量地发送 UDP 数据包，致使服务器处理不过来而 down 机。

（4） ICMP 泛洪

攻击者向攻击目标发送大量的 ICMP 包，使得服务器忙于处理和响应这些 ICMP 包而出现对正常服务响应速度变慢甚至死机的现象。

（5） 应用层攻击

随着 Internet 上的新技术新应用的不断出现，攻击者可以获得的攻击资源和攻击方法也越来越多。而一种新的 DDoS 攻击类型也渐渐崛起，这种新的攻击类型就是应用层的 DDoS 攻击。与前面的泛洪攻击不同，应用层 DDoS 攻击合法地使用 TCP/IP 协议（protocol-compliant）[5]，不需要利用系统漏洞入侵系统（non-intrusive），而是利用合法的应用层请求来攻击服务器。随着 Internet 应用复杂度的不断增加和带宽的不断增大，服务器资源如 CPU 时间和 I/O 带宽等逐渐代替网络带宽成为网络性能的新瓶颈。应用层攻击正是利用了这一事实，通过向服务器发送与合法的用户请求一样的请求来消耗服务器资源。应用层攻击可以分为三类：

a. 请求泛洪攻击。这种攻击以高于正常会话的请求速度发送应用层请求。
b. 非对称攻击（asymmetric attack）。这种攻击向服务器发送大运算量的高负荷请求。
c. Reapeted one-short attack。在这种攻击中，攻击者把多个请求分散在不同的会话中，以高于正常会话的请求速度发送应用层请求。

（二） 畸形包攻击

畸形包攻击是利用系统漏洞进行的攻击，但这在资源对抗的 DDOS 攻击中并不是起主要作用的因素，因此畸形包攻击并不是流行的 DDOS 攻击方式。

（1） Ping of death

早期路由器和许多操作系统对 TCP/IP 栈的实现在 ICMP 包上都是规定 64kB，并且在对包的标题头进行读取之后，要根据该标题头里包含的信息来为有效载

荷生成缓冲区，当产生畸形的，声称自己的尺寸超过 ICMP 上限的包也就是加载的尺寸超过 64k 上限时，就会出现内存分配错误，导致 TCP/IP 堆栈崩溃，致使接收方宕机。[6]

（2） Tear Drop

攻击者把同一个 IP 分组分成几个碎片，碎片中的偏移量是错误的，使到服务器不能正确重组，这会导致服务器重启或者系统崩溃。

（3） Land

在 LAND 攻击中，攻击包是一个特别打造的 SYN 包，它的原地址和目标地址都被设置成某一个服务器地址，此举将导致接受服务器向它自己的地址发送 SYN - ACK 消息，结果这个地址又发回 ACK 消息并创建一个空连接，每一个这样的连接都将保留直到超时。不同的操作系统对 LAND 攻击反应不同，许多 UNIX 最终将崩溃，NT 变得极其缓慢(大约持续五分钟) 。[7]

2.5 DDoS 检测防御技术

如上文所述，DDoS 的攻击由寻找傀儡主机到开始攻击先后经历三个步骤，而第一，第二步实际上是攻击前的预备阶段，第三步是攻击的进行阶段。针对攻击的这两个阶段，对 DDoS 攻击的防御可相应分为两个层次：预防、攻击检测和控制。

2.5.1 DDoS 攻击的预防机制

DDoS 攻击的预防机制主要是在攻击未发生之时减低 DDoS 攻击发生的可能性[2]。预防措施包括个人系统端杜绝非法接入，定期升级系统，打补丁，安装防毒软件和防火墙，使用安全性较高的通信协议。而用户接入网络的预防机制主要是检测网络中具备某些特征的不正常流量，如端口扫描的流量，基于某些可疑端口的流量，数据包报文中含有某些特殊信息的流量等。另外服务器端的预防机制包括，提高资源审计和用户认证机制的健壮性，通过负载分担等增大服务器系统的处理能力等。

2.5.2 DDoS 攻击的检测与控制机制

在 DDoS 攻击进行阶段，检测和控制可以分为两个阶段，一个阶段是 DDoS 攻击分组的检测，另一个阶段是对检测出的可疑分组的控制（目前一般为对可疑分组的过滤或速率限制）。攻击检测的准确率由假阳性比例和假阴性比例来衡量。假阳性比例是指被错误判断为攻击包的正常分组占全部正常分组的比例，假阴性比例则指的是未被检测出的攻击包占全部攻击包的比例。分组过滤的效率常常用正常分组通过比例来计算。显然，一个好的攻击检测和控制系统应尽量降低检测的假阳性、假阴性比例和提高分组过滤的效率。

现有的 DDoS 检测防御系统实施的位置主要有两个，一个是在攻击主机所在的网络，另一个是被攻击主机所在的网络。越靠近被攻击主机所在的网络，攻击分组就越容易被检测出来；而越靠近攻击主机所在的网络，攻击包被检测出来的难度就越大。与此相对应的，被攻击主机所在的网络，分组过滤的效果越好；而靠近攻击主机所在的网络，越多的正常分组可能被过滤掉，因而效果变差。

位于攻击主机所在的网络的检测防御系统使用的机制主要有包过滤机制和数据流速率控制机制两种。包过滤机制主要通过输入过滤（Ingress Filtering）[8]对在本网段内发出的而源地址又不属于本网段的数据分组进行过滤，这样就有效地阻止了于这一网段发起的伪造源地址的泛洪。而另一种同样应用于攻击源端网络的基于包过滤的分布式包过滤系统（Distributed Packet Filtering）[9]除了能过滤源地址为非本网段地址的分组以外，还可以有效地追踪没有被过滤掉的伪造分组。而采用速率限制的典型系统 D-WARD[10]则将系统安装在攻击主机所在网络的边界路由器上，对高速率的输出流进行监控，在让正常流顺利通过的同时，限制攻击包的速率，并随着攻击速率的增加指数降低攻击包的速率。

靠近攻击主机网络进行检测和控制，固然可以及早过滤攻击流，但其实现目前为止其实施都是没有法律上的保证的。要在自己的网络边缘增加专用的网络安全系统需要网络管理者增加投资，在没有法律强制的前提下很难保证每个网络都实施这种安全措施。因此，对于大型网站来说在网站服务器前端实施 DDoS 检测控制系统就变得十分有必要。

David K.Y.Yau 等人提出的 Router Throttles[11]是一个由近服务器端的路

由器组成的系统。系统中的路由器均安装 Router Throttle，每个路由器对流向服务器的数据流量进行控制，从而限制流向服务器的最大流量，保护服务器。另一种可以用于近服务器端的检测控制系统是 Transport-Aware IP Router[12]。这种系统认为 TCP 的控制流量是 Internet 上最重要的流量，应该与 TCP 数据流量区别对待，而其他 UDP 流量和 ICMP 流量在 Internet 上应该受到限制。这种系统通过区别以上几种流量，提供不同 QoS 控制，实现对网络资源的保护，防止流量对网络资源的滥用。而 Yoohwan Kim[13]等人提出的系统针对每个 TCP 会话的变化情况对每个会话的速率进行动态调节，以达到阻止 TCP 泛洪攻击的目标。

另外一些学者提出的近服务器端的 DDoS 检测防御系统则侧重于对攻击源的追踪。精确的 IP Traceback 是指当攻击发生时或攻击结束后确定每个攻击者的攻击路径和攻击发起点。但由于目前的 IP Trace back 技术还难以准确找到攻击分组的真正来源，而且并不能阻止正在发生的 DDoS 攻击，对于 DDoS 攻击的实时防御未能起到实质性的作用。目前提出的 IP Trace back 机制主要有概率分组标记法[14]、基于哈希的 IP Trace back[15]、ICMP Trace back[16] 这几类。

在攻击主机所在的网络和被攻击主机所在的网络实施 DDoS 检测防御各有利弊：在攻击源端进行防御可及早地滤除攻击流，减少网络资源的浪费，并且有利于定位出攻击源；而在受害主机处的攻击防御则有益于攻击的检测。为了达到更好的防范效果，一些研究人员提出了在 Internet 中实施多点检测控制的防御架构[12] [17]，其核心思想是在 Internet 中部署一组分布式节点执行攻击的检测和控制，通过多节点联合探测和协调来保护 Internet 免受 DDoS 攻击。

2.5.3 现有的检测防御系统的不足

在现有 DDoS 入侵防御系统的研究中，研究人员一般都将工作的重点放在攻击的检测上面，而攻击防御系统的控制模块在结构和功能上往往过于简单，大都只是将检测出的可疑分组进行简单的丢弃。而一些基于速率限制的系统，如 Transport-Aware IP Router 等，主要着眼于对网络资源进行管理，防止有流量滥用带宽，而没有针对 DDoS 攻击的特点来分配和管理带宽。另外这些速率限制系统都是基于传输层以下进行管理，对每个会话进行跟踪管理，粒度过小，会消

耗大量的处理器资源，而且不能对用户的访问行为进行控制，不能防御应用层的攻击。而多点防御系统虽然能起到良好的控制和追踪作用，但由于施用范围广，投入大，目前来说可行性并不高。因而现有的系统都不能有效地保护 Internet 上的大型服务器。

第3章 流量控制技术

本文所实现的系统，主要是针对目前对很多大型网站造成危害的应用层DDoS 攻击，基于流量控制机制实现的。以下简要介绍本系统所采用的相关的流量控制技术。

3.1 拥塞控制

随着近年来计算机网络的发展，特别是高速网络的大量涌现，拥塞控制成为了网络技术领域的重要研究课题之一。拥塞是指在要求网络传输的分组数量开始接近网络的分组处理能力时，通信网络不能很好地提供网络通信服务以满足用户要求的情况[18]。网络产生拥塞的根本原因在于，用户输出给网络的负载大于网络的容量和处理能力。

控制网络拥塞，一方面要通过源端拥塞控制机制，使源端可以根据网络拥塞的状况调整数据的发送速率，缓解网络拥塞；另一方面网络的中间设备应该参与到资源管理和控制中，这就是要在路由器等中间设备中采用队列管理技术：当路由器处理能力不足时，队列管理算法对来不及处理的分组进行丢弃或者标记，源端在检测到分组的丢失时做出响应。

3.1.1 TCP 的拥塞控制机制[19]

TCP 协议的拥塞机制属于源端拥塞控制机制。TCP 协议主要通过发送方主机发现丢包和拥塞，然后主动地起用拥塞避免和控制。

TCP 拥塞控制有两个版本：Tahoe 和 Reno[20]。

(一) TCP Tahoe

早期的 TCP 实现算法是基于积极响应并通过超时重传来重发丢失的数据，当丢包时 TCP 减小拥塞窗口，并重传被丢失的分组，然而在使网络拥塞达到最小方面这些算法的性能很差。TCP Tahoe 参考了早期的实现方法并增加一些机制，包括慢启动（Slow-Start）：窗口大小以指数速度增加；拥塞避免（Congestion

Avoidance)：窗口的大小以一定的线性速度增加；快速重传（Fast Retransmit)：从一个丢包状态恢复而不需要等待重传定时器超时。在快速重传机制下，当接收方收到几个对同一 tcp 报文的应答时，发送方就推断已经发生了丢包并进行重传，而不用等到重传定时器超时才进行判断。随后发送方将自己的 ssthresh（slow start threshold，慢启动阈值）设为当前 cwnd（current congestion window）的一半，并且进入拥塞窗口为一的慢启动模式。快速重传提高了信道利用率，但也可能对已经正确接收的包进行重传。TCP Tahoe 拥塞控制是通过控制一些重要参数的改变实现的，这些参数主要有[21]：拥塞窗口，公告窗口，发送窗口，慢启动阈值，回路响应时间，超时重传计数器，快速重传阈值等。

（二） TCP Reno

TCP Reno 同样具有慢启动、拥塞避免、快速重传等机制，并在此基础上作了一些改进。TCP Reno 将 TCP Tahoe 的快速重传改进为快速恢复：当接受方收到几个对同一 tcp 报文的相同应答时，接收方判断丢包并重传，与 TCP Tahoe 不同的是 TCP Reno 仅使拥塞窗口减半而不是降为一，并将 cwnd 设为 ssthresh 与三倍丢失包大小的积。快速恢复机制防止通信管道在快速重传之后变空，因而避免了慢启动在单包丢失之后重填。Reno 的理想情况是在一个窗口中单包丢弃时 Reno 发送方在每个回路响应时间（RTT）中最多重传一个包，但是它在同一窗口中出现多包丢弃时可能出现问题。

3.1.2 中间设备的拥塞控制机制

TCP 拥塞控制在保证互联网的稳定性方面起着至关重要的作用，但随着互联网技术的发展和网络情况的变化，仅依靠 TCP 拥塞控制从端用户来管理网络业务已经显得越来越力不从心了。为了更有效地管理网络拥塞部分，中间网络设备如路由器也必须参与进行拥塞控制。而中间设备的拥塞控制主要是通过队列管理机制实现。

目前的队列管理机制可以分为两大类：被动式队列管理（Passive Queue Management，PQM）和主动式队列管理（Active Queue Management，AQM)。

（一） 被动式队列管理

传统的队列管理一般采用先来先服务（FCFS）策略，对每个队列的缓冲区设置一个最大值，分组按到达顺序在队列中排队，如果到分组到达时缓冲区已满，

则丢弃该分组，因此 FCFS 又称为“丢尾”(DropTail)。由于 FCFS 算法简单，容易实现，所以仍是目前最常用的一种队列管理策略。但 FCFS 被动式队列管理策略本身存在着很多局限性：

（1） 中间设备不能起到调控网络状况的作用，而将拥塞控制的所有责任都推给端用户，不考虑丢包优先级，也不能处罚恶意流。

（2） 在某些情况下，FCFS 机制可能会导致某个流或者少数几个流独占队列空间，而其他流的包不能进入队列的情况。

（3） 由于 FCFS 机制只有在队列满时才会发出拥塞信号，因此会使得队列在相当长时间内处于充满（或几乎充满）的状态。而队列在满状态下的“丢尾”会导致“TCP 全局同步”(TCP global synchronization) 现象，即所有的连接同时被告知拥塞，因而同时降速。

(二) 主动式队列管理

主动式队列管理机制本质上就是要求网络本身参与资源的管理和控制，与 FCFS 机制相类似，AQM 也是利用了 TCP 对发送速率的自适应调节功能，当路由器的处理能力不足的时候，就对来不及处理的分组进行丢弃或者标记，以此来通知源端降低发送速率。主动式队列管理与传统 FCFS 机制的不同之处在于：传统的 FCFS 机制对到达的分组全部进行接收，直到缓冲区全部被填满，而随后到达的分组全部被丢弃。而主动式队列管理机制通过采用特定的分组丢弃技术，维护队列的长度，也就是在网络轻度拥塞的时候就按照预定的规则来丢弃少量的分组，从而使得某些源端降低发送速率；这样使得缓冲区队列的长度保持在一定水平，也就将平均排队时延控制在一定的范围，减少了分组因为在缓冲区中的时间过长而导致的超时重传，并且防止了拥塞的进一步恶化，降低了丢包率；同时，由于缓冲区不会处于满状态，所以它还具有容纳突发流量的能力。通常对主动式队列管理算法的性能评价主要包括：队列的稳定性，公平性[22]和适应性等。

目前应用最广最为典型的是随机早期检测算法（RED）[23]。RED 有两个和队列长度相关的阈值：minth 和 maxth。当有分组到达路由器时，RED 计算出平均队长 avg，当平均队列长度 avg 小于 minth 时，接收到达的分组；当 avg 大于 minth 并小于 maxth 时按照一定的概率 P 丢弃到达的分组；当 avg 大于 maxth 时 RED 退化为 FCFS 算法，到达的分组全部被丢弃。

RED 在计算平均队长 avg 时，采用了滑动平均的方法，见公式（3-1）

$$avg = (1 - w) \times avg + q \times w \quad (3\text{-}1)$$

其中，w 为权值，q 为采样测量时实际队列长度。从而“过滤”掉由于 Internet 数据的突发性导致的短期队长变化，尽量反映长期的拥塞变化；权值 w 决定了路由器对输入流量变化的反应程度。计算平均队长的目的就是为了反映拥塞程度，并据此来计算丢包概率。

计算丢包概率 P 的方法见公式（3-2）、（3-3）

$$Pb = maxp \times (avg - minth) / (maxth - minth) \quad (3\text{-}2)$$

$$P = Pb/ (1 - count \times Pb) \quad (3\text{-}3)$$

概率 P 不仅和 avg 有关，而且还和从上一次丢包开始到现在连续进入队列的包的数量 count 有关。随着 count 的增加，下一个包被丢弃的可能性也在缓慢增加。这主要是为了在到来的包之间均匀间隔地丢包，避免连续丢包，以消除对突发流的偏见和产生全局同步现象。

RED 的优点有：

（1） 在队列满之前丢包，使某些信源降低发送速率——有效防止拥塞

（2） 避免所有的连接被同时告知拥塞——避免同时降速

（3） 各用户丢包的概率近似比例于其所占带宽——较好的公平

3.2 队列调度

队列调度的主要任务，是实现把有限的出口带宽，公平合理地分配给不同的数据流量。通过实施队列调度，Internet 的管理者们希望达到以下目的：

(1) 实现带宽在不同业务类型的流量中的公平分配。这种公平，并非简单地指让每种流量平均地瓜分带宽，而是指某些业务类型的流量按照管理者的意图获得比其他业务类型更多的带宽。只要业务类型获得的带宽和管理者给予它的权重相吻合，就称为“公平”。

(2) 保护正常的流量免受劣质（poorly behaved）流量的影响。也就是说，某些业务类型中的流量的行为不合理或者不正常的时候，不能影响其他业务类型的流量的调度和输出。

(3) 充分合理地利用带宽资源。当一个业务类型没有用完分配给它的带宽的时候，队列调度策略要使其他业务类型可享有剩余的带宽，以免带宽的浪费。

目前常用的队列调度算法有[24]：FIFO（先进先出）、PQ [25]（Priority Queuing，优先级队列）、FQ [26](Fair Queuing，公平队列)、WRR [27][28]（Weighted Round Robin，加权循环）、WFQ [29]（Weighted Fair Queuing，加权公平队列）、DRR[30][31]（Deficit Round Robin，赤字循环）。

(1) FIFO（先进先出）

FIFO 又称为 FCFO（First-come，First-out），是一种最基本也是目前使用最多的队列调度策略。FIFO 按照分组的到达顺序入队，又按照这个顺序对每个包进行无区别的服务。FIFO 的算法的复杂度非常低，这是 FIFO 与其他常用的队列调度策略相比最大优点。但在 FIFO 中，每个队列都被无差别的对待，因此不支持区分服务，难以保证一些特殊业务类型的服务要求。

(2) PQ（优先级队列）

在 PQ 算法中，分组首先被分为不同的类型，每个类型对应于一个优先级队列。在调度的过程中高优先级的队列总是在低优先级队列之前被服务，而同一优先级队列中的分组则按照 FIFO 服务。PQ 具有算法简单，容易实现的优点，关键的业务可以得到保证。PQ 一般有两种应用：绝对优先级队列（Strict priority queuing）[31]和速率限制优先级队列（Rate-controlled priority queuing)。在绝对优先级队列中某一队列只有在比其优先级高的队列为空的情况下才被调度，可用于保证一些十分重要的数据或对时延要求很高的流（例如 VoIP）的服务，但在高优先级队列流量较大的情况下，较低优先级的队列中的数据包可能会长时间得不到发送服务。为了克服绝对优先级队列的缺点，速率限制优先级队列则给高优先级队列加上了带宽限制，在高优先级队列消耗的带宽低于此限制时，按照绝对优先级队列方式调度，一旦高优先级队列消耗的带宽超过此限制，则低优先级队列可先调度。

(3) FQ（公平队列）/ WFQ（加权公平队列）

FQ 的基本思想是防止突发流对网络资源的强占，使各个队列享有完全相同

的输出带宽。调度器对各个队列循环服务，每一队列在一次循环中只传送一个分组。在使用 FQ 的队列中，如果一个数据流的速率超过了分配给它的带宽，受影响的只是同一队列的数据流，而其他队列中的数据流不会受到影响。FQ 的缺点在于，对每个队列间提供无差别的服务，因此不能提供区分服务，不能保障实时性要求高的数据流。另外，当一个队列中的分组明显大于其他队列中的分组时，这个队列就会占有比其他队列更多的带宽，那么 FQ 也就不能保证各个队列享有的带宽相同。在实际的应用当中，分组的大小不可能相同，因此 FQ 并不能真正实现带宽的公平分配。

WFQ 是对 FQ 的一种改进，它通过给各个队列分配不同的权重（weight）来满足不同业务对于带宽的不同要求。WFQ 也可在一定程度上解决 FQ 要求分组大小一致的问题，如果一个队列的分组大于其他队列，则可将这个队列的权重调小。WFQ 的计算复杂度高，因而其可扩展性差。

(4) WRR（加权循环）

WRR 是一种基于类的队列调度策略，一个类可以是单个流，也可以是流的汇聚，它可以表示不同的应用、用户或服务器的数据流。分类器首先将用户业务分成不同的类，每个类对应一个队列，然后为每个队列分配资源和优先级，采用轮询的方式服务这些队列。WRR 克服了 PQ 和 FQ 的缺点：采用轮询的机制，在高优先级队列速率高的情况下低优先级队列也可以得到服务，一次循环当中，每个队列至少可以发送一个分组；各个队列被分配一个权重，根据权重的不同获得不同的带宽。

(5) DRR（赤字循环）

DRR 是一种公平队列调度策略，其算法可简述如下：

首先各队列的赤字计数器（Deficitcounter）初始为 0，通过为各队列分配不同的配额（quantum）决定该队列所占的带宽。

调度器轮询每个非空的队列；当访问到一个队列时，先将赤字计数器的值加上该队列的配额，得到新的值。

若此队列队首分组的大小大于赤字计数器的值，则调度下一非空队列；

若此队列队首分组的大小不大于赤字计数器的值，则对其进行服务，同时将计数器的值减去队首分组的大小，得到新的值；如此对该队列服务，直到队列为

空或队首分组的大小大于计数器的值，则调度下一非空队列。

DRR 克服了 FQ、WFQ 和 WRR 的缺点：DRR 算法简单，计算复杂度低，容易在硬件设备上实现；调度时考虑到分组的大小问题，各个队列可真正得到和权重相当的带宽。DRR 具有算法简单、提供良好的公平性、各队列间互不影响、可按需要给各个队列分配不同带宽等优点，得到了广泛的应用。其改进调度策略 MDRR（Modified Deficit Round Robin）在 Cisco 12000 系列的路由器中被采用。[30]

第4章 网络处理器技术

传统的网络设备体系结构发展经历了两个阶段[3]：

(1) 以 GPP 为核心的网络设备体系结构

在网络发展早期，网络传输速率低，服务少，研究集中在服务框架构建和网络协议的实现上。设备以 GPP（General Purpose Processor）为核心，在通用操作系统基础上，以软件方式实现各种网络服务。目前许多边缘设备：如防火墙、VPN 设备、VOIP 设备，还在采用这种通用处理器+通用操作系统+专用网络服务软件的体系结构。其优点是灵活性好，缺点是处理性能差。这种结构为支持各种复杂运算，采用通用体系结构和指令集，其通用性导致网络处理性能较差。

(2) 以 ASIC/RISC 为核心的网络设备体系结构

随着网络带宽的增长速度远大于通用计算机处理速度的增长，网络瓶颈变成基于 GPP 的节点设备。采用基于 ASIC 和 RISC(reduced instruct set compute) 为核心的体系结构成为主流，尤其是骨干设备的设计。为获取高性能，通常由 RISC 负责非实时管理，ASIC 负责高速数据处理。这种结构缺点是开发周期长，缺乏灵活性。ASIC 不具备可编程性，一旦将计算逻辑固化到硬件，很难修改。设计制造复杂 ASIC 需要花费 18 个月到两年时间，设备制造商必须准确预测未来的市场需求和技术趋势。

在过去的十几年里[33][34]，网络设备的研发和 Internet 的发展相互促进。而当今网络规模和性能迅速增长，新业务不断涌现，这就要求网络设备具有优异的性能，支持高速分组处理；具有高度灵活性，支持不断变换高层网络服务。传统的基于 GPP 的网络设备只满足灵活性要求；而基于 ASIC 的网络设备只满足高性能要求；相比之下，网络处理器是更适应这种需要的一种新型、有效的统一解决方案。网络处理器（NP，Network Processor）是新一代用来执行数据处理和转发的高速可编程处理器。与传统的处理器不同，它的设计采用了全新的理念，使其既有 ASIC 的高速处理能力又有完全的可编程特性，因此具有强大的生命力。从 1999 年到现在，已有包括 AMCC，Intel，IBM，Vitesse，Hifn，Motorola 等

多家大公司相继推出自己的网络处理器产品。其中 Intel 公司在网络处理器的开发方面走在了同行的前列。因而本文的入侵防御控制系统将基于 Intel IXP2400 架构实现。

4.1 网络处理器的基本概念

网络处理器是一种基于可编程 ASIC 结构的新一代 SoC 芯片，专门用于网络通信设备的开发。它综合精简指令集计算机 RISC 的可编程性和 ASIC 成本低速度快的优点，使研发人员可以用相同的芯片实现不同功能和特色的网络设备[35]。网络处理器可以广泛地应用在 Internet 的核心层，分布层和接入层中。在核心网中，网络处理器可以用于实现核心路由器；在分布层中，它可以应用于 ISP 接入设备和数据中心，提供对新型业务和网络汇接的支持；在接入层中，则可以用于用户管理和接入控制，实施安全与网络监控，以及 Intranet 防火墙等。

网络处理器的基本组成分为硬件和软件两个方面。

网络处理器的硬件结构采取了与传统处理器不同的策略，因而使网络处理器兼备了 GPP 和 ASIC 的优点。为了更好地理解网络处理器的硬件结构，我们要了解一下网络处理器上可能运行的应用以及它们各自的特点。网络设备上的应用可以分成以下三类[36]：

(1) 数据面（Data Plane）应用。属于这一类应用的网络设备从端口接收输入的数据包，对数据包进行处理，然后在相应的端口把数据包转发出去。属于此类的应用包括路由数据包，数据包分类，协议转换等。

(2) 控制面（Control Plane）应用。此类应用主要处理辅助数据面的各种控制数据包。路由表更新，建立 ATM 虚电路和 IPsec 的密钥交换等属于此类应用。

(3) 管理面（Manage Plane）应用。这一类的应用程序是负责对网络管理数据包进行处理的。一个非常典型的例子就是 SNMP 进程。

以上三类应用中，对网络设备要求最高的就是数据面应用。此类应用需要对数据包进行线速处理，以防止数据包的丢失。针对这几类应用的不同需求，网络处理器在硬件架构上提供了快速和慢速两个通道，方便处理不同类型的数据包。

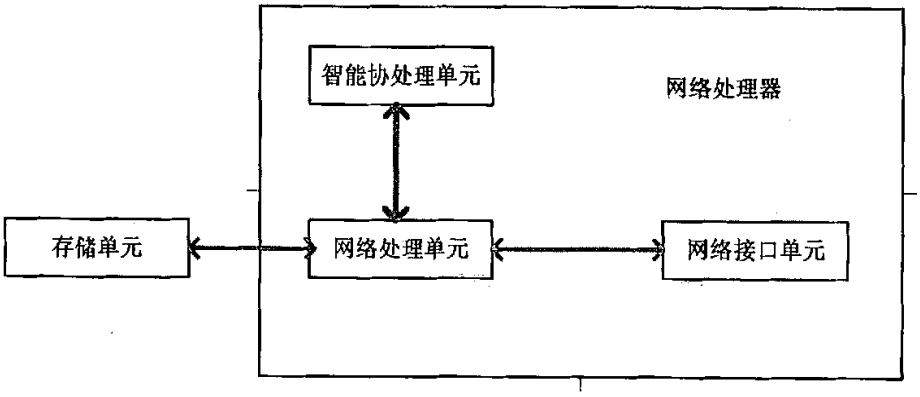

图 4-1 网络处理器的简要硬件结构

图 4-1 给出了网络处理器的简要硬件结构。如图所示，网络处理器主要包括两个功能模块，网络处理单元和专用的智能协处理单元。其中，智能协处理单元是网络处理器的核心，它一般需要嵌入式操作系统支持，用于对网络处理器单元和其他硬件单元进行控制；主要完成控制面和管理面的数据包处理，通过运行操作系统之上的路由协议软件包，完成路由信息的接收，处理和发送，生成并维护路由表等。网络处理单元（一般有多个）通常采用多线程结构，可以完成高速、大容量的智能数据面处理功能，如数据包的收发、包头处理和路由查询等。网络处理器的这种硬件结构，大大提高了其处理速度：多个处理器并行工作的结构可以提高网络处理器的处理能力；处理器内部的硬件多线程结构可以降低数据处理时访问存储器所需要的平均时间，提高芯片的效率；实时性要求不同的任务交由不同的处理器来完成，从一定程度上可以减少数据包的处理延时。从系统灵活性的角度看，智能协处理单元是可编程的，网络处理单元一般由多个 RISC 芯核组成，而 RISC 芯核也是可编程的。这就充分保证了网络处理器的灵活性。不难看出，网络处理器的高速处理能力主要是通过一个芯片内集成多个微处理器以及一个微处理器包含多个硬件线程实现的，当然也采用了其他的一些硬件加速技术，而网络处理器的灵活性则主要是通过智能协处理单元与网络处理单元的完全可编程能力实现的。

网络处理器的软件构成包括：板级支持包(BSP，Board Support Packet)、嵌入式操作系统，路由协议软件包和微代码（或标准语言代码）等四部分。其中前三者运行在智能协处理单元上。板级支持包记录着智能协处理单元需要管理的硬件信息以及它们的主要配置信息；嵌入式操作系统是路由协议或其他应用程序

运行的基础；智能协处理单元通过运行路由协议软件包可以生成并维护路由表；微代码运行在网络处理单元上，用于对数据进行处理和转发。

4.2 基于 Intel IXP2400 的网络处理器硬件平台

随着网络的发展和业务需求的增长，开发者对新一代的网络处理器提出了包括灵活性，高性能，可扩展性和可移植性四方面的新要求。为了满足上述要求，Intel 公司推出了 Intel IXA（Intel Internet Exchange Architeture）它包括两个方面：在硬件平台方面，Intel 公司开发了 IXP2XXX 第二代网络处理器；在软件平台方面，Intel 开发了 IXA Software Portability Framework。上述平台使得 Intel 公司的包括 IXP2400 在内的第二代网络处理器能很好地满足新发展的需求。

4.2.1 IXP2400 硬件体系结构

IXP2400 使 Intel 第二代网络处理器的其中一款产品。与第一代网络处理器相比，IXP2400 改良了硬件体系结构，增加了一些重要的硬件加速设备，使得 IXP2400 能够更好地满足网络处理的高速化和复杂化需求。

IXP2400 的硬件结构如图 4-2 所示[37]

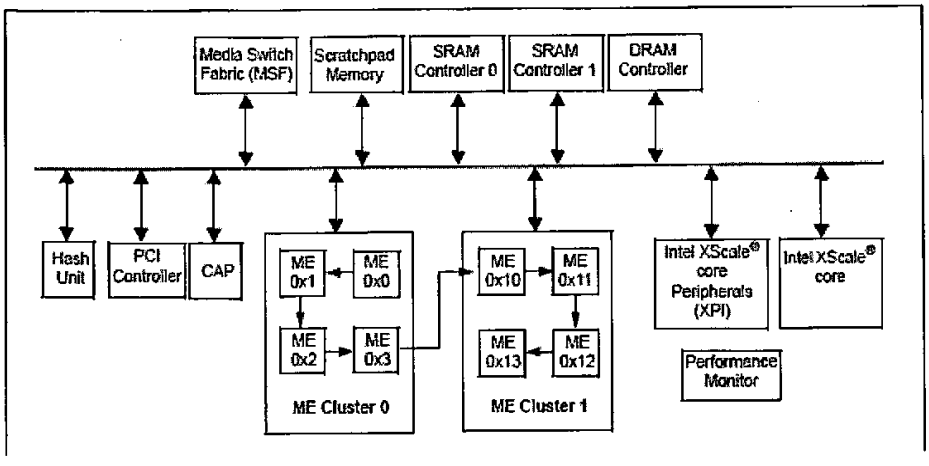

图 4-2 IXP2400 的硬件结构

IXP2400 主要由以下几部分组成：

(1) Intel Xscale Core。Intel Xscale 核心处理器是一个时钟频率为 600MHz

的 32 位 RISC 处理器，遵循 ARM V5TE 的体系结构，包括介质访问控制（MAC）单元、内存管理单元（MMU）、指令和数据缓存等。与其他嵌入式通用处理器一样，它可以编程（用微代码 Microcode）还可以植入 Linux 和 Vxworks 等操作系统。Intel Xscale 主要用于网络处理器的初始化配置和系统管理，还可用于运行路由协议和更新路由表等操作，以及慢通道处理和其他一些控制层任务。[35]

(2) 微引擎（Micro Engine）。IXP2400 中绝大部分数据面的工作由 ME 完成，8 个 ME 分成 2 个簇（cluster），每簇有 4 个 ME。所有 ME 都可以访问共享资源，每个 ME 还可以直接对它邻近的 ME 进行访问。ME 的工作频率与 Xscal core 的频率相同。每条指令执行时间为 1 个时钟周期，它支持软件控制的多线程操作（每个 ME 有 8 个线程），当一个线程在等待耗时较长的存取操作时，另一个线程就可以进行其他操作，从而大大提高了 ME 的资源利用率。每一个 ME 内部都拥有大量的存储器和寄存器，包括可编程的 4k 指令存储区、256 个 32bits 的通用寄存器，512 个 32bits 的传输寄存器(DRAM 和 SRAM 各 256)各 128 个邻居寄存器(NNR,next neighbor registers) 和 640 个 32bits 的本地寄存器（LM，local memory)。LM 对于 ME 中的线程来说，它的速度最快容量最小，只能被 ME 中的线程访问。其它类型的存储器则可被微引擎和 XScale core 共享。在共享存储器中，scratchpad 拥有最快速度和最小容量，DRAM 容量最大但速度最慢，它主要用来存放数据信息。SRAM 容量和速度介于前二者之间，它主要用于存放控制信息。开发者可根据对时间和空间的不同需求来选择适用的类型，这一特点兼顾了网络数据处理的灵活性和高性能。

(3) SRAM Controller，SRAM 控制器。用于接口 SRAM 存储设备，控制、管理 IXP2400 中其他功能单元对 SRAM 存储设备的访问、操作。IXP2400 中有两个 SRAM Controller，SRAM Controllor0 和 SRAM Controllor1，相互独立，可并行工作。

(4) DRAM Controller，DRAM 控制器。用于接口 DRAM 存储设备，控制、管理 IXP2400 中其他功能单元对 DRAM 存储设备的访问、操作。DRAM 存储空间可达 2GB，主要用于存储数据包，路由表等大型的数据结构。

(5) Media and Switch Fabric Interface (MSF)，介质和交换结构接口。它是 IXP2400 与外部物理层设备、交换结构的接口单元，是网络处理器接收和发送数据包的接口。MSF 支持 UTOPIA、SPI 和 CSIX 等标准化协议，方便与其他厂家的产品接口。

(6) PCI Controller，PCI 控制器。64bit 66MHz 的 PCI 总线，用于连接控制面处理器、系统管理处理器、其他网络处理器和 PCI 以太网网卡等符合 PCI 规范的设备。

(7) ShaC 单元：包括一个中间结果暂存器 (Scratchpad memory)、控制状态寄存器访问代理 (Control Status Register Access Proxy) 和一个哈希单元 (Hash unit)。其中 Scratchpad 容量为 16kB，用于微引擎之间的通信以及重要数据的内部缓存；Hash Unit 支持 48bit，64bit，128bit 的哈希运算；CAP 用于对 IXP2400 中的控制状态寄存器 (CSR) 进行访问、操作。

4.2.2 ENP2611 结构概述

ENP2611 是一款由 RadiSys 公司出品的基于网络处理器 Intel IXP2400 的开发板。IXP2400 本身只是一个单纯的处理器，要实现完整的网络设备的功能，单单具备处理器是不足够的，还必须把网络处理器的各种接口作适当的扩展，连接到相应的外设和存储设备上，才能连接到网络上成为一个完整的网络设备。ENP2611 开发板就完成了对 IXP2400 的接口扩展和外围电路的设计连接，开发人员只要按需要把开发板连接到网络上，把正确的可执行文件装载到 IXP2400 上，ENP2611 就可以作为一个完整的网络设备运转。

ENP2611 对 IXP2400 所作的主要扩展包括：从 MSF 扩展出三个千兆以太网口，把 IXP2400 的内部 PCI 接口扩展成可以和 PC 主机的 PCI 槽连接的接口，从 IXP2400 的相应引脚扩展出 DDR SDRAM SODIMM 内存卡接口等。

4.3 基于 Intel IXP2400 的典型软件结构和包处理流程

4.3.1 基于 Intel IXP2400 的典型软件结构

基于 IXP2400 系统的典型软件结构与其他软件一样，都采用了层次结构来进行组织。如图 4-3 所示。

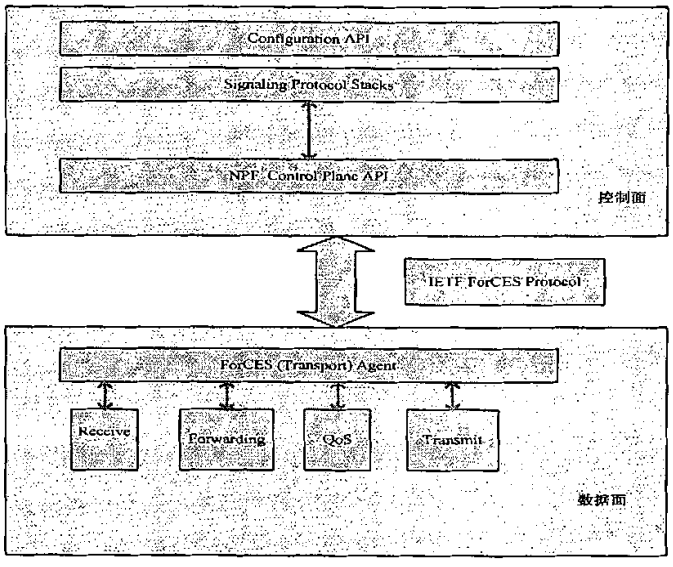

图 4-3 IXP2XXX 的典型软件软件结构

IXP2400 的软件系统分为两大部分，控制面和数据面。其中控制面主要由系统配置 API，系统通信协议栈，控制面 API 和 IETF 消息协议组成，由于在 IETF For CES Protocol 中采用了 Transport Agent 通信机制，使得当系统底层的硬件平台发生变化时，上层的软件结构仍能保持不变。而数据面的几个主要模块，都是以线速对数据包进行处理，数据包经过接收模块进入系统，然后经过一系列的中间模块的处理，最后由发送模块发出。而在这些环节中，数据面的每个模块都与控制面中相应的核心组件（Core Component）对应，这些核心组件负责控制、管理与之对应的数据面模块，接受来自控制处理器的控制请求，更新相关的表项和数据。

4.3.2 基于 Intel IXP2400 的典型包处理流程

一般而言，网络处理器中的数据包处理流程包括以下几个阶段：

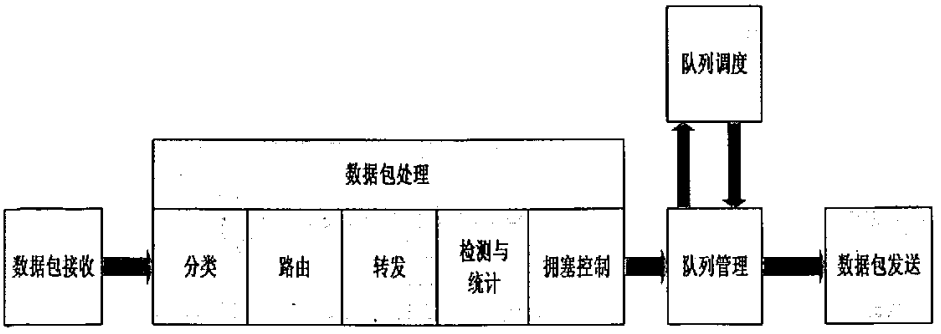

图 4-4 数据包处理流程

(1) 数据包接收阶段

MSF 从 PHY 接收数据包分片（mpacket）并将其缓存在 RBUF 中，并向微引擎发出信号，表示数据已经收到，微引擎将接收缓存中数据传输到 DRAM，并将数据包分片重组为完整的 Packet。然后同一线程为数据包创建数据包描述符（Packet Descriptor）并写入 SRAM 中，同时写入 Scratch Ring，传递给下一级的数据包处理模块。

(2) 数据包处理阶段

数据包处理阶段从第一个处理模块于 Scratch Ring 中取出数据包描述符开始。对数据包的处理一般包含数据包分类，路由，转发，检测和统计，拥塞控制等过程，还可能包括数据包的封装和解封装以及其他更高级的应用处理。

(3) 队列操作阶段

数据包完成了前面的数据包处理阶段之后，就准备进入发送阶段。而队列操作模块是位于处理模块和发送模块之间的模块。队列操作阶段包括入列、出列等队列操作和发送调度。入列，出列等队列操作由队列管理（Queue Management）模块执行。该模块接受上一模块发来的入列请求，执行入列操作；接受发送调度模块发来的出列请求，执行出列操作。而发送调度模块对各个队列进行调度管理，尽可能使得带宽资源可以在多个队列之间得到有效而公平的利用。调度算法的相关介绍见前面“队列调度”一节。

(4) 数据包发送阶段

当调度模块成功调度一个数据包之后，它会向队列管理模块发送出列请求。队列管理模块就会从相应队列中取出一个数据包，并把数据包相应的数据包描述符交给发送模块。发送模块把 DRAM 中的数据包读入到 TBUF 中，然后发送出去。

4.3.3 Intel IXP2400 中的数据结构的管理和操作。

IXP2400 是采用存储、处理、转发的方式来进行数据包处理的。在整个处理的过程中，IXP2400 首先将数据包接收进来，存储在适当的存储器中，然后对存储器中的数据包进行处理，最后将存储器中的数据包发送出去。由此可见，存储器是 IXP2400 的一项极为重要的硬件资源，对存储器的使用和操作是数据包处理中必不可少的一环，而对存储器中数据结构的操作方式和操作速度则对网络处理器的处理性能有着直接的影响。

(1) 数据包处理流程中存储器的使用

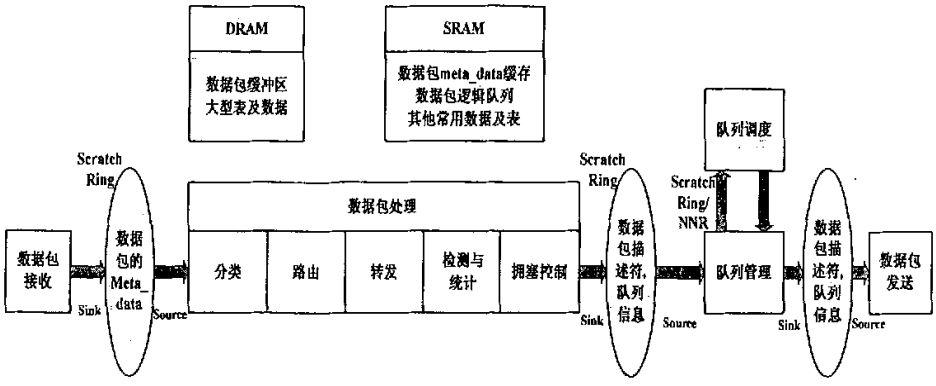

图 4-5 数据包处理过程存储器的使用

如图 4.5，MSF 接收数据包后把数据包缓存在 RBUF 中。然后接收模块从 Free Buffer Pool（空闲缓冲区池，是 Link List 格式的一种数据结构）取出一个 Free Buffer（位于 DRAM 中）。然后，将 RBUF 中缓存的数据包读入 Free Buffer，并创建数据包描述符以及数据包的 meta_data。Meta_data 是存储数据包相关信息的数据结构。该数据结构的内容如表 4-1 所示。

表 4-1 meta_data 存储格式

<table>
<tr><td colspan="5">buffer_next 32 bits</td></tr>
<tr><td colspan="3">offset 16 bits</td><td colspan="2">BufferSize 16 bits</td></tr>
<tr><td colspan="2">header_type 8 bits</td><td>rx_stat 4 bits</td><td>free_list 4 bits</td><td>packet_size 16 bits</td></tr>
<tr><td colspan="3">output_port 16 bits</td><td colspan="2">input_port 16 bits</td></tr>
<tr><td>nhid_type 4 bits</td><td>Reserved 4 bits</td><td colspan="2">fabric_port 8 bits</td><td>nexthop_id 16 bits</td></tr>
<tr><td colspan="2">Color 8 bits</td><td colspan="3">flow_id 24 bits</td></tr>
<tr><td colspan="3">reserved 16 bits</td><td colspan="2">class_id 16 bits</td></tr>
<tr><td colspan="5">packet_next 32 bits</td></tr>
</table>

各个字段的含义简介如表 4-2 所示。

表格 4-2 meta_data 各字段含义

字段	含义
buffer_next	指向存储数据包的下一个 buffer。当数据包比较大的时候，可能会用多于一段的 buffer 来存储。
offset	数据包在 buffer 中的偏移量，单位：字节。
BufferSize	当前 buffer 的长度，单位：字节。
header_type	Buffer 中的数据包包头类型。
rx_stat	接收状态字，是 MSF 用来高速 Receive_Thread_FreeList 中的 Thread 要处理的数据包的某些重要信息。
free_list	Freelist ID。Receive_Free_List 是用来存储记录当前空闲线程的。当数据包接收下来，缓存在 RBUF 中的时候，MSF 会通知某一特定的线程来对 RBUF 中的数据包进行处理。线程完成指定的处理任务后，会把自己重新加入 Free_List。MSF 根据 Receive_Free_List 中的信息决定哪个 Thread 对数据包进行处理。
packet_size	数据包在其所有 buffer 中的总长度。
output_port	数据包的发送端口。
input_port	数据包的接收端口。
nhid_type	Next_hop_id 的类型
Reserved	保留字段
fabric_port	当系统需要提供多种业务支持的时候，可将多个 IXP2400 网络处理器通过交换接口芯片（FIC）与交换结构（SF）接口。处理时 Ingress IXP2400 从外部网络接收数据包，对数据包进行处理，然后发送到交换结构。Egress IXP2400 则从交换结构接收数据包，进行处理，再把数据包发送出去外部网络。而 fabric_port 则是用于标识与 SF 的接口的字段。
nexthop_id	下一跳 ID。这个字段的值由路由转发进程决定。
Color	QoS 中用于标识不同流量的字段。
flow_id	用于 MPLS 协议。
class_id	标识一个端口中的不同队列。
packet_next	指向下一个数据包。

当整个数据包读入 Free Buffer 后，接收模块将数据包描述符和一部分的 meta_data 写入 Scratch 中的 Ring Buffer。然后属于数据包处理阶段的模块会首先从 Scratch Ring 中取出数据包描述符和 meta_data，然后对数据包进行相应的处理。处理完毕以后，数据包描述符或者一部分的 meta_data 会被写入 Scratch Ring 中，然后会被队列管理模块取出，对数据包进行入列操作。队列管理模块还会从 Scratch Ring 中取出队列调度模块的出列请求，然后进行相应的出列操作，把数据包描述符写入相应端口使用的 Scratch Ring。最后发送模块从 Scratch Ring 中取出数据包描述符，把相应的数据包发送出去。在上述处理流程中，某些环节还可以利用 NNR 进行微引擎之间的快速参数传递。由上述处理流程可知，IXP2400 使用 Link List 来存储数据包（实际上是数据包描述符），使用 Ring Buffer 来传递不同数据包处理模块之间的参数，使用 NNR 来加速模块之间的参数传递，在 SRAM 中存储数据包的基本信息，在 DRAM 中缓存数据包。

(2) Intel IXP2400 的队列架构

在网络处理器中，数据包的逻辑队列是以 Link List 形式来实现的。事实上，Link List、Ring Buffer 等数据结构在网络数据处理中起着非常重要的作用，在将 mpacket（数据包分片）重组成 packet 时需要 Buffer List（Link List 的一种形式），在线程之间通信时需要用到 Ring Buffer。但如果用软件实现对这些数据结构的操作将会相当复杂，费时费力。而在 IXP2400 中，采用一种被称为 Q-Array 的硬件结构来实现对 Link List 的自动，快速操作[33]，因此网络处理器实际上物理集成了对队列操作的支持。

网络处理器中的队列架构包含以下几部分[38]：

a. 缓存在 DRAM 中的数据包：DRAM 中划分成固定大小的缓存（pages），每一段缓存存贮一个数据包，其数据包描述符指向这段缓存。

b. SRAM 中的数据包描述符：包括数据包的基本信息（如包长等）和包调度的参数（如时间戳等）。数据包描述符以 Link List 的形式组成逻辑队列。

c. SRAM 中的队列描述符（Queue_Descriptor）和链表环节（Q_link）：在 IXP2400 中，一条队列的基本组成包括队列描述符和一系列的链表环节。队列描述符包含逻辑队列的基本信息，由 Head，Tail，Q_Count，Cell_Count，EOP 等字段组成。而 Q_Link 中包含了存放数据包的 Free

Buffer 的地址，以及指向下一个 Q_Link 的指针。队列的最后一个 Q_Link 的值为零，用于标识队列的结束同时也是为下一个加入队列的数据包预留一个 Q_Link。

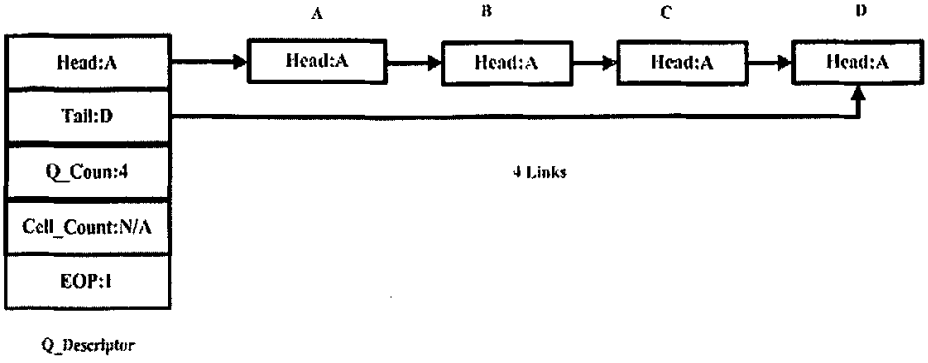

图 4-6 队列示意图

在图 4-6 中，队列中共有 4 个 Q_link。Q_Descriptor 的 Head 指向队列中的第一个 Q_Link：A；Tail 指向队列中的最后一个 Q_Link：D；Q_Count 的值为 4，表示队列中有 4 个数据包；Cell_Count 字段没有使用，因为 Q_Descriptor 中 Head 指向的数据包由一个 Cell 组成（EOP=1 表明 Head 指向的 Buffer 中包含数据包的最后一个 Cell，可以推断该数据包只包含一个 Cell）。

一般而言，SRAM 控制器执行入列（Enqueue）操作包含以下 4 个步骤：

1、 从 SRAM 存储器中读取 Q_Descriptor。获得其中的 Tail、Q_Count 字段的值。由于 Enqueue 操作不需要使用 Head 字段，因此可以不读取而节省存储器带宽。

2、 将新的数据包描述符写入 Tail 指针指向的队列中的最末一个 Q_Link。

3、 将 Q_Count 字段的值加 1，创建一个空的 Q_Link 作为发送队列的最末一个 Q_Link，并将 Q_Descriptor 中的 Tail 指针指向该 Q_Link。

4、 将 Q_Descriptor 写回存储器。

与入列（Enqueue）操作相反，出列（Dequeue）操作是从队列中取出一个数据包（实际上是数据包描述符）。Dequeue 操作同样包含四

个步骤：

1、从外部存储器（一般是 SRAM）中读取 Queue_Descriptor，获得其中的 Head、Q_Count、EOP 等字段。由于 Dequeue 操作不需要使用 Tail 字段，因此可以不读取而节省存储器带宽。

2、将 Head 指针指向的 Q_Link 中的数据包描述符取出。

3、更新 Head 指针，将其指向队列中的下一个 Q_Link，将 Q_Count 值减 1。

4、将更改后的 Q_Descriptor 写回外部存储器。

入列操作与出列操作对于微引擎来说都是原子操作，无需数据结构的锁定，解锁等操作，简化了操作步骤，加快了操作速度。

d. SRAM 中的 Q-Array：可自动，快速地对队列执行 enqueue，dqueue 操作，缓存常用的 Link list。前述的入列和出列操作，是基于 Q_Descriptor 在外部存储器（SRAM）中的假设进行的。当然，利用 SRAM Controller 对于 Q_Descriptor 的自动操作以及操作的原子特性，在低速应用场合，这种入列出列的操作速度也能满足 IXP2400 的线速处理的性能要求。但是，当 IXP2400 以较高的线速进行处理时，如果频繁地进行外部存储器读写，就会带来极大的存储器访问时延，IXP2400 的处理性能就会受到严重的影响。IXP2400 中有两个 SRAM Controller，每个 SRAM Controller 中有一个 Q_Array，每个 Q_Array 中有 64 个存储单元。这些存储单元是 IXP2400 的内部存储资源，可用于缓存 Q_Descriptor，Ring Buffer Descriptor 等复杂数据结构。在这种专门的硬件和相应的软件支持下，IXP2400 就能快速执行对队列结构的入列和出列操作。

e. Scratchpad 中的队列状态矢量：保存各个逻辑队列的状态（一位（bit）代表一个队列，表明队列是否为空），使用时通过对矢量的查询代替对每个 Q_Descriptor 逐个查询，可以优化队列操作的性能。

4.4 Intel IXP2400 的开发工具——Workbench 简介

基于网络处理器的开发过程，不仅包括模型算法的构建，数据流程的设计，软件源文件的编写等步骤，还包括程序的汇编（或编译），连接和调试等环节。

Workbench 是 Intel 开发的基于 IXP2XXX 的软件开发平台，用于程序的开发，编译，链接和调试。Workbench 是基于 IXP2XXX 的集微代码开发，编译，链接和调试于一体的集成环境，运行于 Windows 2000 或以上的操作系统之上。Workbench 的特征主要包括：支持源码级的调试，支持网络设备仿真和网络流量仿真，支持网络仿真器的命令行接口，支持定制用户图形界面等。

第5章 入侵防御控制系统的实现

本论文研究的课题属于国家自然科学基金项目——网络时空行为与2008奥运会网络安全关键技术研究的一个子课题。本课题组的其中一个研究任务就是实现一个基于网络处理器的服务器前端DDoS入侵防御系统。该系统可以实现对Web服务器用户进行标识，采用HSMM对网站用户流量进行统计得出其正常程度，根据用户的不同正常程度进行流量分类，进行相应的防御控制，以及对流量进行多维统计等多项功能。而本论文所实现的入侵防御控制系统主要是实现该入侵防御系统中的流量分类以及控制功能。图5-1给出服务器前端的DDoS入侵防御系统的总体结构：

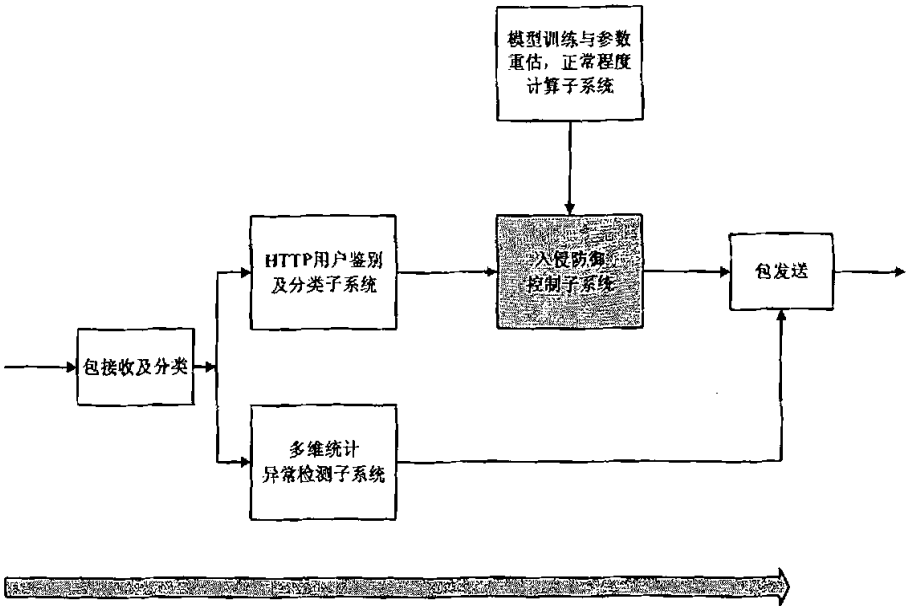

图 5-1 服务器前端的DDOS入侵防御系统总体结构图

本论文研究的内容即该系统中的入侵防御控制子系统，主要功能是对前一模块鉴别出来用户进行分类，同一类的用户分到同一队列中，不同队列享受不同的带宽和优先级。

5.1 系统架构

5.1.1 系统设计总体思想

本系统处理的目标是对从用户端到服务器端的Web数据流分类，并提供不同级别的服务，达到控制和限制DDoS攻击流量的目的。要达到这个目的，要考虑以下几个方面：

(1) 对用户的分类标准必须体现该用户的合法性。也就是对于一些非法可能性比较小的用户，应该给予高的优先级和高带宽；反之，对于非法可能性很大的用户流量，应该给予较低的优先级，使得其流量受到比较严格的限制。本系统采用用户的正常程度作为分类的标准。

(2) 按照正常程度来对用户进行分类，并不能排除正常程度小的用户也是合法用户的可能性，虽然这种可能性比较小；而对于正常程度比较高的用户，也存在是攻击流量的可能。虽然这些情况的可能性相对较小，但也要尽可能保证合法用户的权益以及限制非法流量。因此，本系统采取一种公平的队列调度算法，保证各个队列都能使用属于自己的那部分带宽。另外，每个队列都采用拥塞控制进行管理。

(3) 队列调度算法和所起的作用是尽量公平有效地分配有限的带宽，使得各个队列都能够获得一定的服务质量保证。当存在某个时间段，只有一个队列有数据的时候，则这个队列在这个时间段内总能不断获得调度。而网络处理器本身的快通道是线速处理的，所以服务速度非常快，可能会出现其服务速度超过服务器的处理速度的情况。为了避免这种情况，必须在队列调度后端加上速率限制，以根据服务器的处理速率限制网络处理器的发包速率。

基于以上考虑，在结合现有队列调度原则和拥塞控制方法的基础上，本文控制系统的实现方案如图5-2所示。

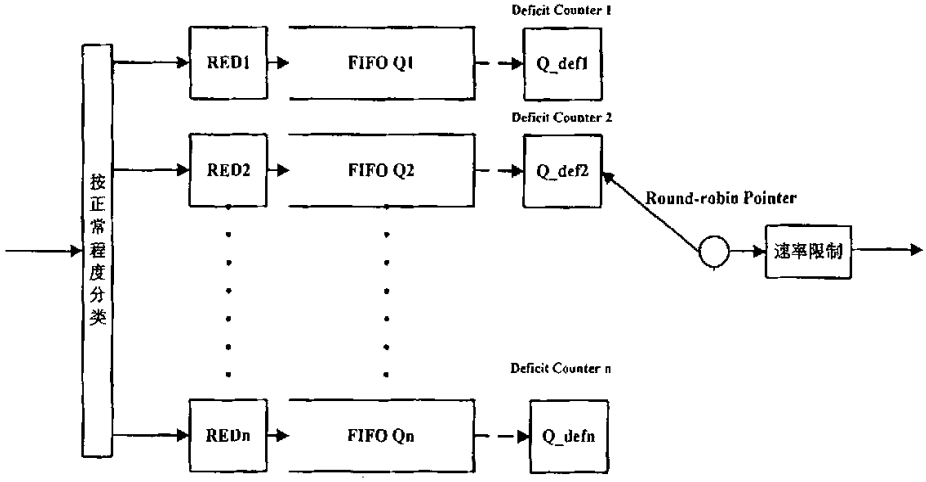

图 5-2 入侵控制系统的实现方案

旧用户的数据经过前面的用户鉴别之后都会得到一个标识这个用户的 ID。本控制系统首先根据用户的 ID，通过查表得出其正常程度，然后根据正常程度得出数据包所应该归入的队列。得出数据包所属的队列后，按照该队列的 RED 算法进行运算，以判定这个数据包是否可以入列。如果可以入列，则把数据包加到相应的队列中。这些队列是按照 FIFO 对本队列的数据包进行管理的。数据包按照入列的先后顺序进行出列操作。出列时，各个队列间的队列调度由 DRR 算法控制。每个队列都有一个特定的配额值。DRR 轮询各个队列，根据各个队列的配额值计算队列的当前配额值，并根据当前配额值决定这个队列中的当前的数据包能否发送。若不能，则轮询下一队列，若能，则发送数据包并从当前配额值中减去包大小。在决定要发送数据包时还必须受到速率限制模块的控制，以限定的速率发送。

5.1.2 系统的功能架构

围绕流量分类和流量控制这个功能核心，本系统包含以下几个功能模块：

(1) 分类模块。分类模块的功能主要确定数据包所属的队列。该模块的分类过程是根据 HTTP 用户鉴别及分类子系统提供的用户 ID，得出该用户行为正常程度的或然概率所在的范围，进而得出其所在的队列（由正常程度计算系统所提供）。研究表明，正常流的或然概率出现的频率类似于正

态分布[39]，如图 5-3 所示，98 世界杯的正常流的平均对数或然概率都集中在-3.18 附近，因此并非或然概率越大其正常度就越高。我们用正常度来表示用户数据的对数或然概率靠近-3.18 的情况：对数或然概率靠近-3.18 表明此用户数据的正常程度较高，放入高优先级队列；对数或然概率远离-3.18 则说明此用户数据的正常程度较低，放入较低优先级队列；远离到一定的程度，表明极有可能是攻击，对这些数据直接作过滤处理。这样的分类机制克服了以前的入侵防御控制系统只区分正常用户和非正常用户的缺点，不用确定一个区分正常和攻击的门限，而是提供多个具有不同优先级的队列。即使某些正常的但行为较为特殊的用户被前面的 DDoS 入侵防御检测系统给予远离-3.18 的对数或然概率，只要不低于最小的门限（该门限可以很低，例如图 5-3 中的-8；也可以不设），都可以获得一定的服务。这样就消除了唯一一个区分门限对攻击防御效果的影响，具有较大的灵活性。

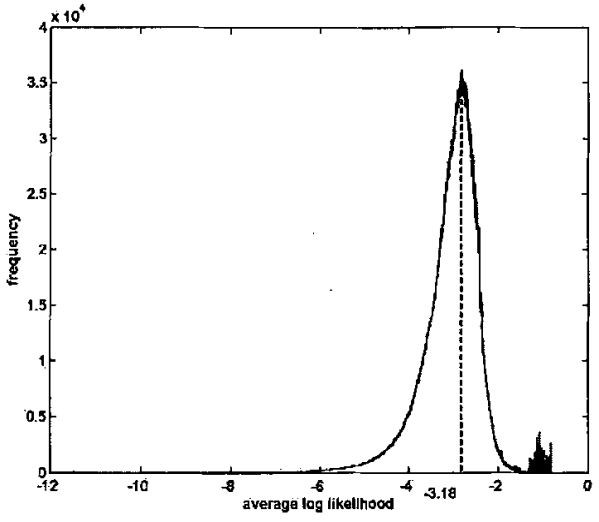

图 5-3 98 世界杯正常流或然概率统计结果

(2) RED 模块。RED 模块是实施拥塞控制的模块。当用户访问频繁数据流量比较大的时候，各个队列中都有可能出现拥塞，因此要对每个队列都进行拥塞控制。在队列调度时不同队列间要体现优先级，而在拥塞控制时又要考虑同一队列各用户间的公平性，即，不能让速率大的用户过多抢占

速率小的用户的带宽，因为根据 DDoS 攻击的特点，攻击流一般都具有很大的速率，所以 RED（随机早期检测算法）是一个值得借鉴的方案。在 RED 中使用平均队列长度 avg 来反应队列的拥塞情况，而平均队列长度 avg 常常与实际队列长度相差较远[40]，不适合本控制系统的情况，本控制系统需要及时了解队长的变化，这样才能在有攻击发生的时候及时作出反应；另外， RED 有许多参数需要设置，如最小队长 minth、最大队长 maxth、权值 w 等等，各个参数的设置直接影响到 RED 的效果，而其最优设置尚有待进一步的研究和探讨[23]；最后，网络处理器是不支持浮点运算的，因此给实现 RED 算法带来了很大的困难。

为了简化各个影响因素，同时考虑到网络处理器的实际情况，在实现时对 RED 采用了简化算法。用即时队列长度 len 来反映队列的拥塞情况，maxth 等于队列的缓冲区大小，minth 表示开始按概率丢弃的最小门限，p 代表各个队列的丢弃概率。当即时队列长度 len 小于该队列的 minth 的时候，数据包直接入列；当即时队列长度 len 大于该队列的 maxth 的时候，数据包直接丢弃。当队列长度 len 大于该队列的 minth 而小于该队列的 maxth 的时候，由网络处理器产生一个随机数，比较这个随机数与该队列的丢弃概率 p 的大小。如果随机数大于 p，则不丢弃该数据包，否则就丢弃该数据包。

该简化算法的优点包括：

a. 只用设置最小队长 minth、最大队列长度 maxth，丢包概率 p 三个参数，减小了参数设置的复杂度。

b. 用即时队列长度 len 来反映队列的拥塞情况，便于更及时的控制拥塞。

c. 简化了队列长度的计算，避免了使用浮点数表示权值，队列长度和丢弃概率，易于在网络处理器上实现。

(3) 队列管理（Queue Manage）模块。队列管理模块是对队列进行实际操作的模块。对队列进行的实际操作包括数据包的入列和出列操作。

(4) 队列调度与速率限制模块。队列调度与速率限制模块主要是实现对各个队列的数据包进行发送管理。由于 DRR（赤字循环）调度算法具有算法

简单、提供良好的公平性、各队列间互不影响、可按需要分配给各个队列不同的带宽等优点，因此本控制系统在队列调度策略上采用了 DRR。通过分配给不同队列不同的 quantum（配额）来体现队列的优先级，分配 quantum 的同时也达到了分配和限制各个队列带宽的目的。具体实现时，本系统采用了 Intel 公司提出的实现方法。结合网络处理器的具体情况，Intel 公司对 DRR 算法进行了一些修正。以下是在 IXP2400 上实现 DRR 调度算法要考虑的问题：

a. 在进行队列调度的时候，队列中第一个数据包的包长是不可知的。由前面对网络处理器的介绍可知，在每个队列的描述符中，并没有包含第一个数据包的长度的信息，因必须要在完成队列调度，出列了该数据包的描述符，才能够得出数据包的长度。

b. 当数据包的包长较大时，调度模块的速度可能会大于发送模块发送数据包的速度。这种情况下可能发生的情况是发送模块来不及处理出列的数据包，造成数据包的丢弃。

c. 在执行出列操作时，可以根据出列的数据包描述符得出包的长度。但此处得出的包长度并不是一个以字节为单位的包长度，而是以 Chunk 为单位的包长度。而数据包的 Chunk Size 和数据帧帧长的关系，通过在 work bentch 上仿真得出为 ：

Chunk_Size=Frame_length(bytes)/128

考虑以上情况，在网络处理器上做具体实现的时候，对 DRR 算法作了以下的调整：

a. 引入“负配额”机制。一个队列要有权发送数据，必须满足以下条件：队列中有数据，队列的当前配额值为正。当队列的当前配额值变成负值的时候，这个队列在这一轮的调度中就再无权发送数据包。当一个端口中的其他所有队列都没权发送数据（队列为空或者当前配额值为负值）时，这一轮的调度就结束，进入下一轮调度。在每一轮调度开始的时候，各个队列会重新加上各自的配额值。而空队列则保持上一轮的当前配额值。

b. 当队列调度模块调度了一个数据包时，该端口已调度数据包的值就

加一。当发送模块发送完一个数据包时，该端口的已发送数据包的值就加一。队列调度模块每次调度前都会检查这两个值的差值，发现这个差值超过一定的门限就停止调度，直到该差值下降到门限值以下。

c. 配额值和当前配额值以 Chunk 为单位计算。

d. 另外，本系统中的速率限制模块是通过增加调度时延，限制队列调度的服务速度，减少每秒队列调度的次数来实现的。

5.2 系统的网络处理器实现

5.2.1 系统的总体框架

本系统的网络处理器实现基本上按照图 5－2 实施。由于本系统目前尚处于独立测试阶段，所以在实现的时候加入了前端的数据包接收和流量分类模块，并且假定各个用户和各个队列的正常程度，而不另外按照正常程度计算子系统的计算结果进行分类。系统的模块框图如图 5－4 所示。

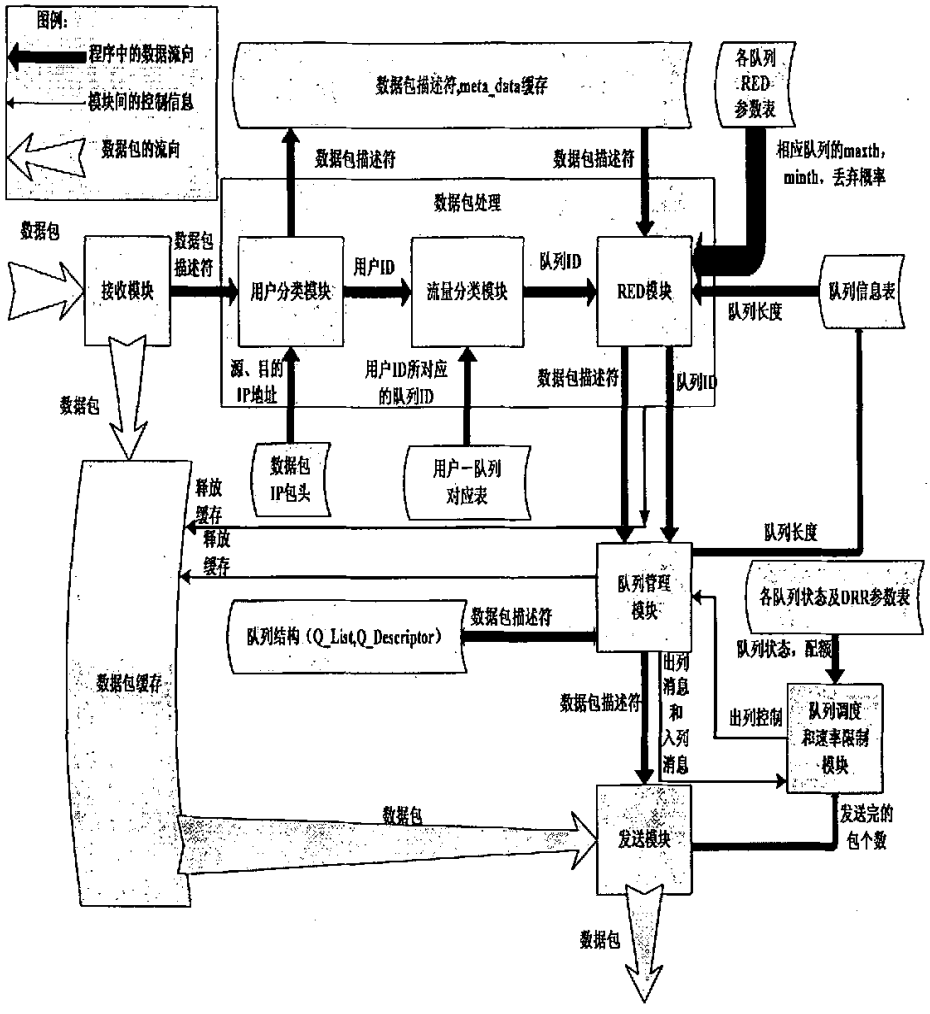

图 5-4 系统总体框图

系统的工作流程如下：

(1) 接收模块从 MSF 接收数据包，把数据包（包含二层帧的帧头和帧尾）缓存到 DRAM 的数据包缓存中，并根据数据包的相关信息产生数据包的 meta_data，并传递到用户分类模块。

(2) 用户分类模块接收到有效的 meta_data 之后，把数据包描述符保存到本 ME 的寄存器中，然后根据其中的信息从数据包缓存中读入数据包的 IP 包头，并按照其源 IP 地址和目的 IP 地址对数据包进行用户分类。一个用户用一个 IP 地址对标识。分类完成后该模块把用户 ID 传递给流量分类模块。

(3) 流量分类模块收到用户 ID 后，根据用户－队列对应表，查找到用户所属队列的 ID，然后把该 ID 传递给 RED 模块。

(4) RED 模块收到队列 ID 后，从队列 RED 参数表中查找出该队列的各个 RED 参数，并从队列信息表中读入该队列的当前长度。如果该队列的当前长度小于 minth，则取出该数据包的数据包描述符，和队列 ID 一并传递给队列管理模块；如果该队列的当前长度大于 maxth，则丢弃该数据包，释放该数据包的缓存。如果队列的当前长度在 minth 和 maxth 之间，则按照该队列的丢弃概率决定该数据包是否丢弃。

(5) 队列管理模块接收到要求入列的数据包描述符和队列 ID 之后（包含在 enqueue 数据中），对数据包进行入列操作，并且更新队列信息表中的信息。接着队列管理模块会生成一个包含队列 ID 的 enqueue 消息。如果队列管理模块收到 dequeue 控制消息，则根据 dequeue 控制消息给出的队列号对相应队列执行出列操作，并根据出列操作的执行结果和队列号生成一个 dequeue 消息。最后队列管理模块把 enqueue 消息和 dequeue 消息一并传递给队列调度和速率限制模块。

(6) 队列调度和速率限制模块收到队列管理模块发过来的 enqueue 消息之后，对相应的队列在队列状态和 DRR 参数表中作出标记，表明该队列非空。队列调度模块在速率限制模块的控制下每过一段时间就进行一次队列调度。在进行队列调度之前，队列调度模块会先判别该端口中已经被调度出列而尚未被发送模块发送出去的数据包个数是否达到了门限值，如果是则不进行调度，直到该值回落到门限值以下为止。进行队列调度的时候，要首先判别出当前有哪些队列是非空并且队列的当前配额值是非负的。只有同时符合以上两个条件的队列才能进行调度。当一个队列被调度以后，队列调度模块把包含队列号的 dequeue 控制消息发给队列管理模块，然后继续调度下一个队列。

(7) 队列管理模块收到队列调度模块的 dequeue 控制信息后，对相应的队列执行出列操作，取出数据包描述符，发给发送模块，更新队列信息表。然后根据出列操作的信息，生成一个包含数据包 Chunk Size 和队列号的 dequeue 消息，发给队列调度模块。

(8) 队列调度模块收到 dequeue 消息之后，从 dequeue 消息中取出数据包的 Chunk Size，然后在队列状态和 DRR 参数表中，找到该队列对应的配额值，减去数据包的长度。如果此时当前配额值为负值的话，就把该队列标识出来，该队列在这一轮调度中就再无权发送数据包。然后队列调度模块重新加上该队列的配额值。

(9) 发送模块根据接收到的数据包描述符和接收来源，判定该数据包要发出的端口。然后把数据包从 DRAM 中取出来发送出端口。发送完毕之后，发送模块把已经发送完的数据包个数发送给队列调度于速率限制模块。队列调度与速率限制模块通过这个个数就可以判断已经出列而尚未发送的数据包个数。

5.2.2 系统中的数据结构

系统中用到的数据结构主要有队列结构和各个模块要用到的表格。关于队列结构在前面“Intel IXP2400 中的数据结构的管理和操作”一小节已经详细介绍过了，此处不再详述。下面介绍一下各个模块所用到的表格的具体结构。

(1) 用户－队列对应表。如表 5-1 所示。

表格 5-1 用户－队列对应表

User_ID (32bits)	Queue_ID (32bits)
User1	Queue1
.....	.....

该表中每个表项包含两个单元，每个表单元的长度是 32bits。查表时只要根据用户 ID，计算出该用户 ID 所在表项的偏移量，再加上该表表头的偏移量，就可以找到该用户对应的表项，然后读出该表项的第二个单元，就能得到该用户对应的队列 ID。该表存储于 SRAM 中，表中的内容由流量分类模块的初始化程序于系统开始运行的时候进行初始化。

(2) RED 参数表。表所示。

表格 5-2 RED 参数表

Maxth(16bits)	Minth(16bits)	Probability(32bits)
Maxth1	Minth1	Probability1
.....	.....	.....

该表的每个表项都包含三个单元，分别对应一个队列的三个 RED 参数。其中 maxth 单元和 minth 单元的长度分别为 16bits，丢弃概率（Probability）的单元长度为 32bits。如前文所述，网络处理器是不支持浮点运算的，所以在网络处理器上实现 RED 算法，是通过产生一个随机数，然后跟丢弃概率比较，以决定数据包是否丢弃。由于网络处理器产生的随机数是长度为 32bits 的二进制数，所以丢弃概率的长度也必须为 32bits。该表存储于 SRAM 中，表中的内容由 RED 模块的初始化程序于系统开始运行的时候进行初始化。当程序运行到 RED 模块的时候，RED 模块会根据数据包的队列 ID，把相应队列的表项读入 ME 的 Local Memory 中，以便对该表项进行快速访问。

(3) 队列信息表。如表 5-3 所示。

表格 5-3 队列信息表

Q_time_H(32bits)	Q_time_L(32bits)	Queue_length(32bits)
Q_time_H1	Q_time_L1	Queue_length1
……	……	……

队列信息表的每一个表项包含三个单元，分别对应每个队列的三个参数。其中 Q_time_H 和 Q_time_L 分别是 timestamp 的高 32 位和低 32 位。这个 timestamp 的值是上一次队列为空的时候网络处理器生成的 timestamp。这个值可以在 RED 算法中用来计算平均队列长度。但由于本系统采用的是简化的 RED，所以暂不使用该值。而 Queue_length 是当前的队列长度，单位是包。这些表项的内容是队列管理模块进行队列操作时生成的。这个表格存放在 SRAM 中，必须对其进行初始化。初始化时把表中的每个单元的值都置为 0。RED 模块会在运行时把相应表项读入 ME 的 Local Memory 中。

(4) 队列状态及 DRR 参数表。

队列状态及 DRR 参数表其实分为两个部分。第一部分的表项是以端口为单位，标识一个端口的所有队列的各项状态的，而第二部分 DRR 参数表的表项则是以队列为单位，包含各个队列的具体信息。队列状态及 DRR 参数表 5-4 如表所示。

表格 5-4 队列状态及 DRR 参数表

Schedule vector(32bits)		High queue mask (32bits)
Low queue mask(32bits)		Queue empty vector(32bits)
Packet scheduled(32bits)		Current port weight(32bits)
Port weight quantum(32bits)		
.....		.....
Queue credit(16bits)	Queue credit increment (16bits)	Packet count(32bits)
.....	.....	.....

其中，schedule vector 是标识各个队列的配额的状况的。每个队列对应该矢量中的一位。当队列的当前配额值为负的时候，schedule vector 中对应的位置 0，否则置为 1。High queue mask 和 low queue mask 是用于为端口的队列提供不同优先级服务的，当队列在 High queue mask 中对应的位置为 1 的时候，该队列就属于高优先级队列，可以优先于低优先级队列组中的队列获得调度。本系统实现的时候，把所使用的队列都放到高优先级队列组中进行调度。Queue empty vector 是标识队列空/非空状态的矢量。当队列为空的时候，队列在该矢量中对应的一位就置为 0，否则就置为 1。Packet scheduled 是记录这个端口中被调度出列的包个数，当队列调度模块向队列管理模块发出一个出列控制消息的时候，Packet scheduled 就会相应地加一。Current port weight 和 Port weight quantum 两个变量是用于端口间的调度的。在实际环境中，肯定存在多个端口都有数据发送的情况，这个时候就要进行端口间的调度，保证各个端口能公平地获得发送数据的机会。本系统中的端口调度采用 WRR 调度算法，Current port weight 记录的是该端口当前的权值，而 Port weight quantum 则是端口的配额。由于本系统的侧重点不在于端口间的调度，此处不再详述。

再来看 DRR 参数表部分。每个队列对应一个表项，每个表项包含三个单元，分别是队列的当前配额值（Queue credit），队列配额值（Queue credit increment）和队列中的包个数（Packet count）。这里的 Packet count 变量是由队列调度与速率限制模块管理的，不同于队列信息表中的 Queue_length。对 Packet count 的操作主要是，当队列调度模块收到队列管理模块发过来的入列

消息后，对相应队列的 Packet count 加一，当队列调度模块收到队列管理模块发过来的出列消息后，对相应队列的 Packet count 减一。这个变量的作用主要是便于调试。

队列状态及 DRR 参数表是存放在运行队列调度与速率限制模块的 ME 的 Local Memory 中。初始化程序在该模块运行之初对该表进行初始化。

5.2.3 系统模块详细设计

本系统实现时，按照各模块间的联系，结合网络处理器的典型软件结构，把上述的各个模块分成几个程序块（block），每一个程序块使用一个微引擎。每个程序块中，除了包含相应的模块以外，还包含各个初始化函数。初始化函数的主要作用是初始化各个模块使用到的表，信号，和 Link List 的指针和 CSR。不同微引擎之间通过 Scratch Ring，NNR 以及 reflector 机制通信。Reflector 是一种由网络处理器提供的，在不相邻的微引擎之间进行数据传输的机制。各个程序块、模块，微引擎间的对应关系如表 5-5 所示。

表格 5-5 程序块，模块和微引擎对应表

程序块	包含的模块和初始化函数	微引擎
数据包接收	接收模块，系统初始化函数，接收模块初始化函数。	ME0:0
数据包处理	用户分类模块，流分类模块，RED 模块，流分类初始化函数，RED 初始化函数，IP 包头缓存初始化函数。	ME0:1
队列管理	队列管理模块，队列管理模块初始化函数。	ME0:3
队列调度	队列调度与速率限制模块，队列调度初始化函数。	ME1:0
数据包发送	发送模块	ME0:2

不同微引擎上的程序块之间的通信如表 5-6 所示

表格 5-6 程序块间通信

程序块	程序块间通信
数据包接收	利用 Scratch Ring4 传递数据包描述符给数据包处理程序块。
数据包处理	1，从 Scratch Ring4 取得数据包接收程序块传递过来的数据包描述符。 2，利用 Scratch Ring5 传递 enqueue 数据给队列管理程序块。Enqueue 数据包括：数据包描述符(dl_buf_handle)，dl_eop_handle，队列号。
队列管理	1，从 Scratch Ring5 取得 enqueue 数据。 2，利用 Scratch Ring6 取得从队列调度程序块传递过来的 dequeue 控制消息。 3，利用 Scratch Ring7 到 Scratch Ring10 四个 Scratch Ring 把要从不同端口发出的数据包的描述符传递给数据包发送程序块。 4，利用 NNR 把 enqueue 消息和 dequeue 消息传递给队列调度程序块。
队列调度	1，从 NNR 中取得 enqueue 消息和 dequeue 消息。 2，利用 Scratch Ring6 把 dequeue 控制消息传递给队列管理程序块。 3，利用 reflector 机制取得数据包发送程序块已经发送的数据包个数。
数据包发送	1，利用 Scratch Ring7 到 Scratch Ring10 四个 Scratch Ring 取得要从不同端口发出的数据包的描述符。 2，利用 reflector 机制把不同端口的已经发送的数据包个数传递给队列调度程序块。

其中数据包接收和数据包发送涉及大量网络处理器架构相关的信号和控制过程，与本系统最终实现的功能关系不大，所以本文中不做详细介绍。而数据包处理，队列管理和队列调度几个程序块实现了本系统的核心功能，以下是对这几个程序块工作流程的简要描述。

(1) 数据包处理

图 5-5 是数据包处理的总体流程。由于这个程序块是同时使用 ME 上的八个线程执行的，因此在每次开始执行都要等候上一个线程发过来的信号以及上一次 Scratch Ring5 写操作的完成信号。获得这两个信号之后，线程从 Scratch Ring4 中读取数据。如果返回的第一个长字 (Long Word) 的值为 0，表示 Scratch Ring4 为空，则把各个信号发给下一线程，并重新等待上一线程的信号。若 Scratch Ring4 非空，程序就把从 Sratch Ring 读入的数据包描述符，meta_data 等缓存起来，并根据数据包描述符 dl_buf_handle 把数据包 IP 包头读入缓存。然后用

户分类模块会根据其源，目的 IP 地址对数据包进行用户分类。用户分类模块的流程图如图 5-6 所示。由于本系统实现该模块只为测试用，所以为了简单起见，将不能匹配的 IP 对都划分为最后一个用户。

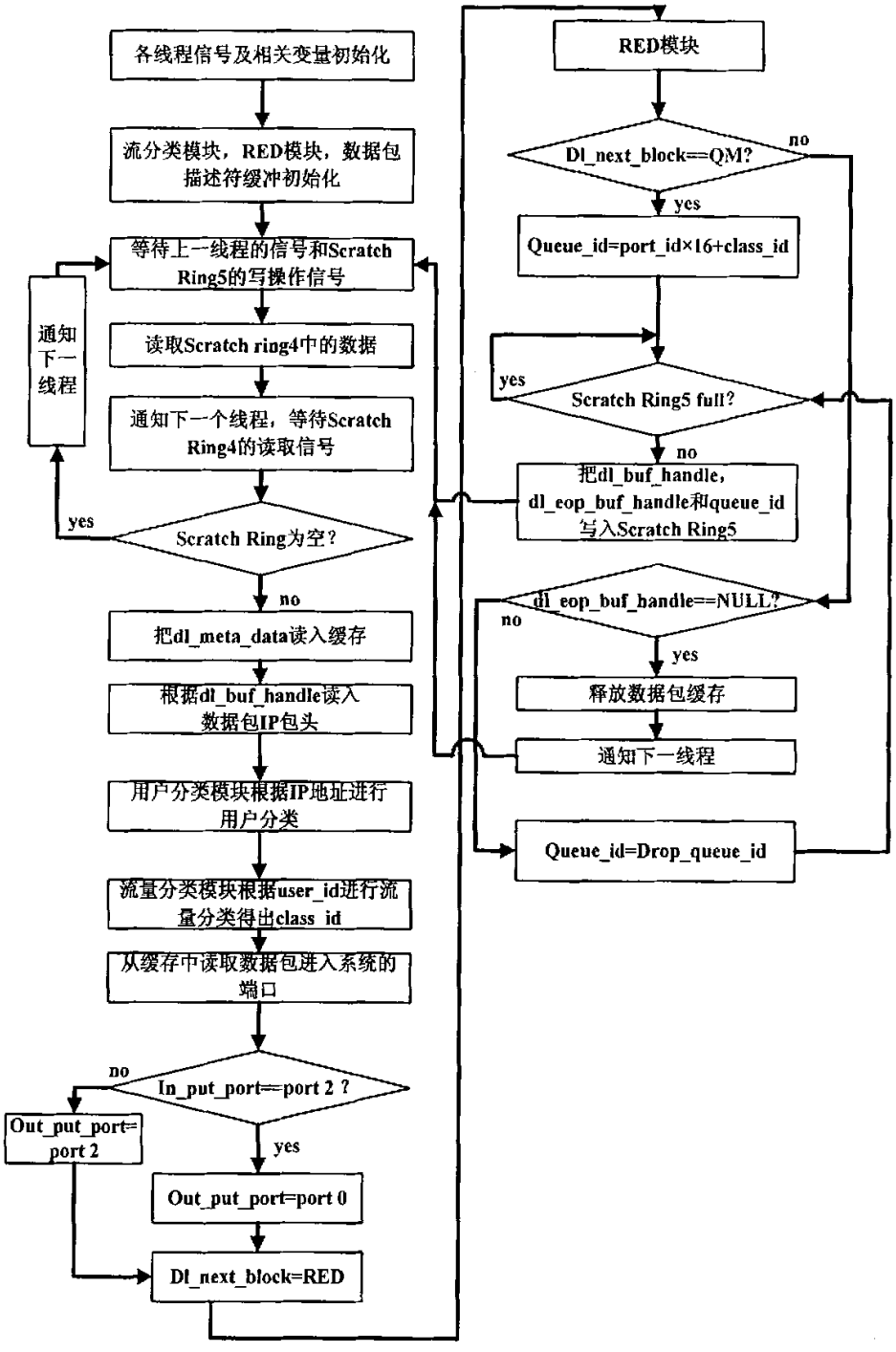

图 5-5 数据处理程序块流程图

用户分类模块得出数据包的 user_id 后，把 user_id 传递给流分类模块。该模块由用户的 user_id 得出其 class_id。该模块流程图如图 5-7 所示。由于队列管理程序块所处理的队列号是从 port 0 的队列 0 开始编号的，所以必须给出发送端口的端口号。由于本系统是用于服务器前端的，因此考虑只采用两个收发端口，即采用 port 0 和 port2 作为接收和发送端口。所以由 port0 接收到的数据就由 port2 发送出去，由 port2 接收到的数据就由 port0 发送出去。然后程序把 dl_next_block 变量设为 RED 模块的 ID，当 RED 模块开始的时候会对该变量的值进行检查，如果不是自己的 ID，则不做处理。

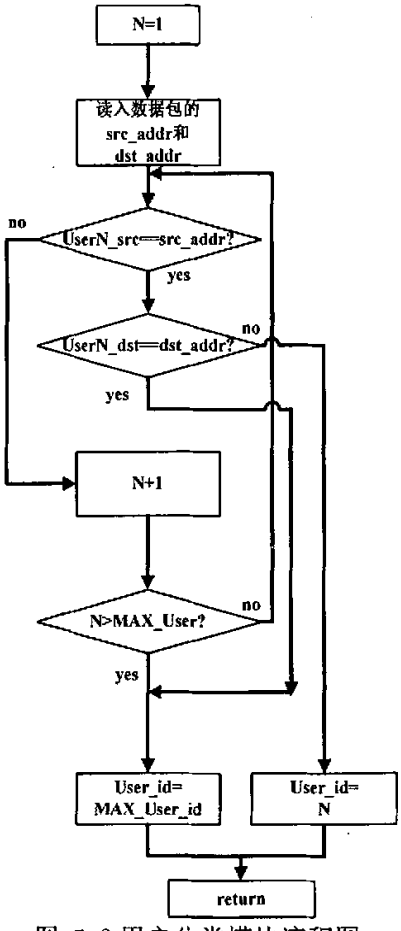

图 5-6 用户分类模块流程图

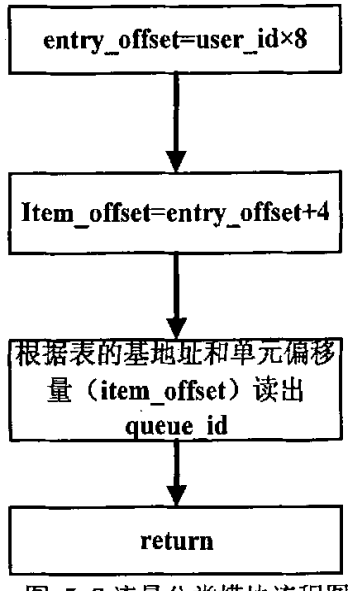

图 5-7 流量分类模块流程图

RED 模块的流程图如图 5—8 所示。RED 模块首先根据 port_id 和 class_id 计算出 queue_id，然后进入一个线程同步阶段。RED 模块有自己专用的线程同步信号，在模块开始执行的时候，都要等待上一个线程的信号，然后把信号传递给

下一个线程，保证线程的顺序执行。如果当前线程是 thread0，该线程还会执行清空 CAM 的操作。CAM 是微引擎中的内容访问寄存器，一般用于快速查找，与微引擎中的 LM，SRAM Controller 中的 Q-Qrray 配合使用，实现对数据结构的快速操作。CAM 中有 16 个缓存单元，查找时通过比较要查找的数值和 CAM 缓存单元中的值，找到内容与要查找的数值一样的缓存单元的编号，而这个编号就对应着这个值所代表的数据结构在 LM 和 Q-Array 中的存放地址。而在 RED 模块中，每一轮线程执行开始的时候，也就是每次 thread0 开始运行 RED 模块的时候，都要把原来 CAM 中的内容清空。这是由于队列信息表中的信息是由队列管理程序块更新的，所有线程轮过一次之后，之前读入的队列信息很可能已经过时了，必须读入新的信息。线程同步之后，RED 模块会检查 dl_next_block 的值。如果这个值不是自己的 ID 号，则通知下一线程，然后结束。如果这个值是自己的 ID 号，则根据 queue_id 计算 RED 参数表和队列信息表中该队列的相应表项的偏移量。然后对该偏移量进行 CAM 查找。如果这个偏移量在 CAM 中，则可以直接找到这个队列对应的信息在 LM 中的位置，并根据 LM 中的值运行 RED 算法。如果该值不在 CAM 中，则表明这一轮的运行中，在当前线程之前没有其他线程把这个队列的信息读入 LM，就要根据该偏移量和表的基地址从 SRAM 中读入相应信息，并更新 CAM 的内容。完成这部分操作之后，RED 模块利用网络处理器的 CSR 生成一个随机数备用。然后，判断数据包能否入列。首先判断当前队列长度 cur_len 是否大于 maxth，如果是则置 dl_next_block 为 DROP，数据包直接丢弃。再判断 cur_len 是否小于 minth，如果是则数据包可以入列，dl_next_block 置为 QM。如果 cur_len 是在 minth 和 maxth 之间，则比较刚获得的 random 值和队列的丢弃概率 probability 之间的大小。如果 random 值比较大，则数据包可以入列，否则数据包就要被丢弃。RED 模块只判定数据包能否入列，而不直接进行入列或者丢弃的操作。这些操作由后续的程序根据 dl_next_block 的值进行。

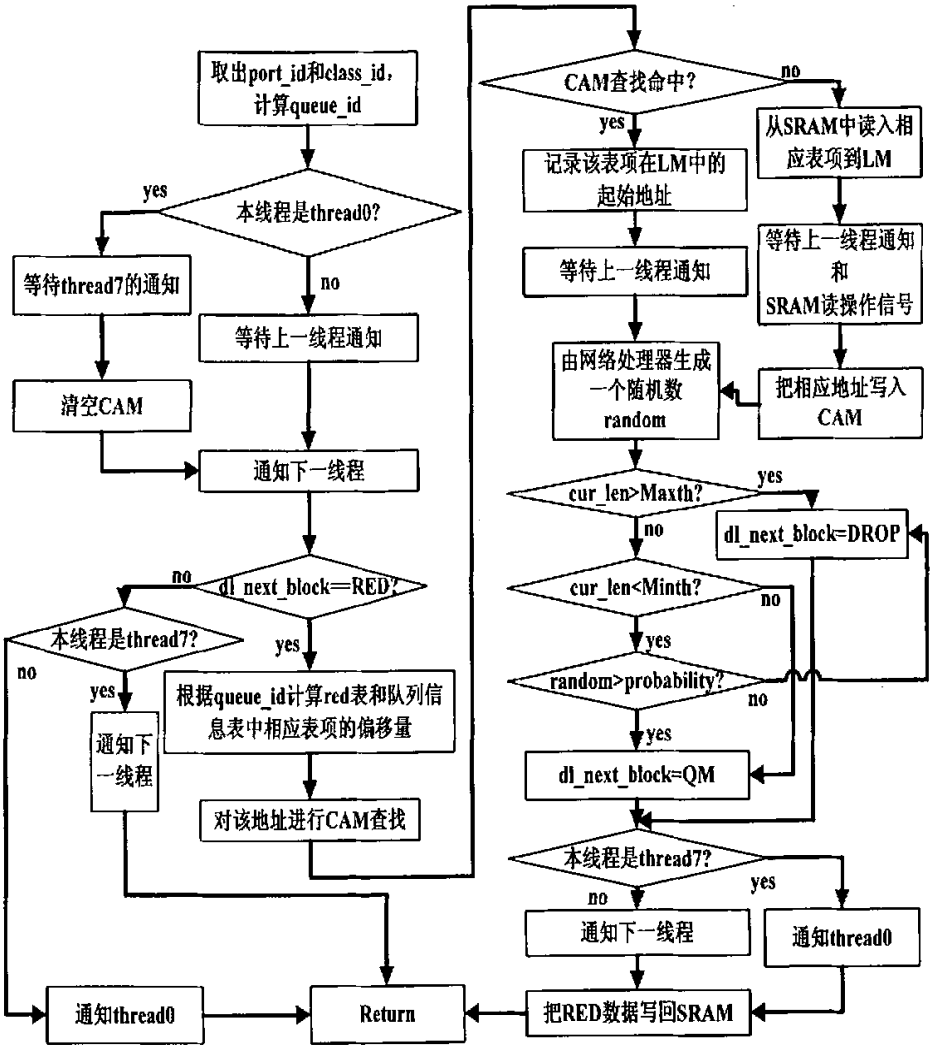

图 5-8 RED 模块流程图

程序从 RED 模块返回之后，检查 dl_next_block 的值。如果该值为 QM，则计算该数据包的 queue_id，把 queue_id 和 dl_bufer_handle 和 dl_eop_buf_handle 一起写入 Scratch Ring5 中。在每次对 Scratch Ring 进行写操作的时候，都要先检查 Scratch Ring 是否被写满了，如果是则一直查询直到 Scratch Ring 的状态不再为 full。程序发出了 Scratch Ring5 的写操作之后，就会回到“等待上一线程信号和 Scratch Ring5 写操作信号”这一步，完成一次循环。如果 dl_next_block 的值不是 QM，则该数据包是要被丢弃的。此时程序会检查 dl_eop_buf_handle 的值。如果这个变量的值为 NULL，则表明该数据包只存储在 DRAM 的一段 buffer 中，而非存储在多段 buffer 中。如果是这样，程

序就根据 dl_bufer_handle 找到这一段 buffer，并释放它。如果数据包是存储在多段 buffer 中，则给这个数据包的 queue_id 赋一个丢弃队列的 id。在本系统中，丢弃队列的 id 值是 1024。队列管理程序块收到入列到这个队列的数据包，就会在稍后的操作中释放这个数据包的缓存。

(2) 队列管理

队列管理程序块的处理流程如图 5—9 所示。当模块中的线程收到上一个线程发来的开始信号之后，线程就开始运行。线程把开始信号发给下一线程，并且从 Scratch Ring5 中读取数据处理程序块发过来的 enqueue 数据。当 Scratch Ring5 的读信号返回而前一个线程也发来信号时，当前线程就通知下一线程，并从 Scratch Ring6 中读取队列调度程序块发过来的 dequeue 控制消息。如果 enqueue 数据的第一个长字的值不为 0，则判定为有 enqueue 数据。然后程序会判断 enqueue 数据中提供的队列 ID 是否为 drop queue 的 ID。如果是 drop queue 的 ID，则这个数据包是要被丢弃的，程序就把数据包入列到 drop queue 中，并且更新丢弃数据包个数的计数值，并把变量 sop_handle 置为 0。这个变量是用于标识 enqueue 消息的有效性的。如果该变量为 0，则表明相应的 enqueue 信息无效，无需发往队列调度程序块，因此当没有 enqueue 数据时，sop_handle 也会置为 0。如果 enqueue 数据中的队列号并非 drop queue 的队列号，则对该 enqueue 数据中的数据包描述符执行入列操作。入列操作包括两部分。第一部分操作是在 CAM 中查找该队列的 Q_Descriptor，如果找到则进行第二部分操作，利用 Q_Descriptor 进行入列操作；否则就读入该队列描述符到 Q_Array 中并更新 CAM，然后再进行入列操作。发出入列指令后，线程会等待入列操作完成的信号和上一个线程发来的信号。当两个信号都到来时，就生成一个 enqueue 消息，并把 enqueue 数据中 dl_buf_handle 的值赋给 sop_handle，然后通知下一线程，并发出从 Scratch Ring5 中读取 enqueue 数据的命令，以备下一次循环时使用。然后线程就开始对 dequeue 控制消息进行处理。其处理过程与 enqueue 数据的处理过程类似，也是先根据从 Scratch Ring6 中读出的数据第一个长字的值是否为 0，不为 0 则开始进行 dequeue 操作，为 0 则认为没有 dequeue 控制信息。Dequeue 操作与 enqueue 操作类似，而 dequeue 操作除了把数据包描述符出列以外，还要把数据包描述符发往相应端口的 Scratch Ring。在本系统

中，port0 对应 Scratch Ring7，port1 对应 Scratch Ring8，port2 对应 Scratch Ring9，port3 对应 Scratch Ring10。当 dequeue 操作完成后，如果队列长度为 0，则更新队列信息表中的 timestamp 值。而队列信息表中的队列长度的值则在 dequeue 操作和 enqueue 操作时进行更新。Dequeue 操作完毕后，会生成一个 dequeue 消息。该消息会连同之前的 enqueue 消息一起，通过 NNR 发给队列调度程序块。然后线程就完成本次循环的操作，重新从“通知下一线程并从 Scratch Ring6 读取 dequeue 控制消息”开始。如果这一轮没有收到 dequeue 控制消息，则线程要先判别 sop_handle 的值是否为 0，才决定是否发送消息给队列调度程序块。如果 sop_handle 为 0，则不发送消息给队列调度程序块，并且检查 drop queue 是不是为空。如果 drop queue 中有数据，则对 drop queue 中的数据包进行清空，并释放数据包缓存，然后完成这一轮循环，回到循环开始处——“通知下一线程并从 Scratch Ring6 读取 dequeue 控制消息”。

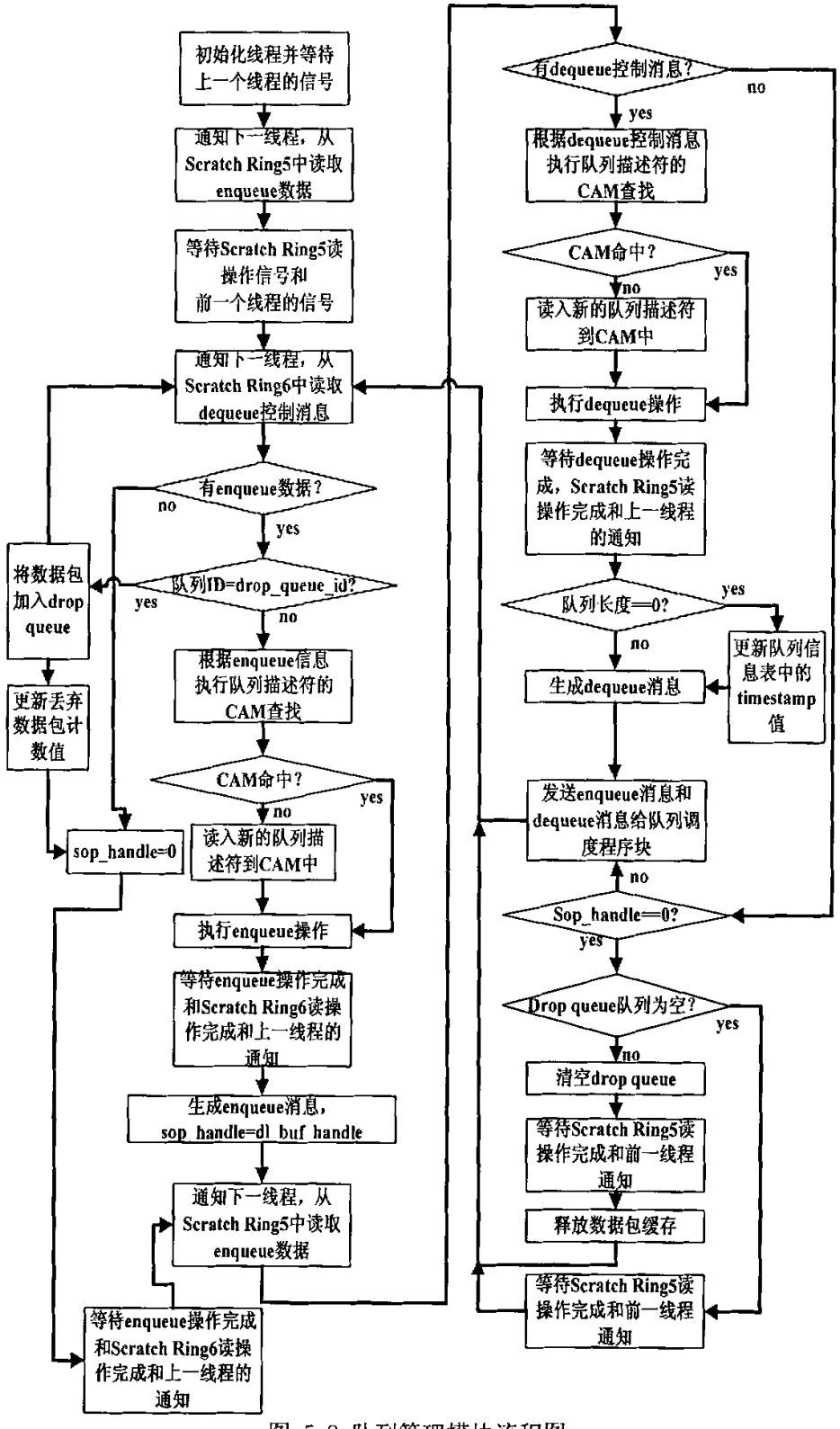

图 5-9 队列管理模块流程图

(3) 队列调度

本系统实现的队列调度是基于包的调度，也就是每次调度一个数据包。而队列调度程序块要完成数据包调度的任务，除了要运行调度算法，还需要与队列管理程序块通信，对队列管理程序块发送控制消息，处理队列管理程序块发过来的消息。所以这个程序块运行时分为两个线程来分别实现这两个功能。一个线程是负责运行调度算法的，称为调度线程；另一个线程是负责消息处理工作的，称为QM消息处理线程。以下分别对这两个线程的工作流程作简要介绍。

队列调度程序块除了要进行端口中队列的调度，还必须进行不同端口之间的调度。在实现的时候是先对系统的四个端口进行调度，确定一个要发送的端口，然后再对该端口中的队列进行调度。因此该程序实际上分为两个大的部分，第一部分是对端口的调度，实行的是WRR调度算法。第二部分是对端口中队列的调度，实行的是DRR算法。

调度线程的流程如图 5-10 所示。线程开始调度之前都会运行一个延时程序。这个延时程序实际上起到速率限制的作用，通过控制调度算法运行的速度，限制数据包出列发送的速度。延时完毕之后，程序首先对端口进行调度。端口调度主要用到的状态矢量有 port_empty_vector，credit_vector，port_round_robin_vector。其中 port_empty_vector 是标识端口是否为空的矢量，当端口为空时，该端口在矢量中的对应位置为 0，否则置为 1。credit_vector 是标识端口当前的权重值的矢量，当端口的权重值不为 0 的时候，该矢量中对应的位就置为 1，否则置为 0。port_round_robin_vector 矢量是用于循环调度的。在一次端口轮询中，已经被查询过的端口在这个矢量中对应的一位就置为 0，则在这次轮询中就不再查询这个端口，要下一次轮询开始的时候才查询这个端口。在开始端口调度的时候，把这三个向量相与，只有在三个向量的相应一位都为 1 的端口才参加调度。然后用 ffs 指令取出第一个可以调度的端口的 port_id。ffs 指令的作用是按照从左到右的顺序，确定源寄存器中第一个值为 1 的比特的位置。在本程序中，对端口的轮询是由 port0 开始到 port4 结束。所以利用 ffs 指令就可以得出四个端口中，从低到高顺序下第一个可以调度的端口。如果 ffs 返回的结果表明没有一个端口可以调度的话，则调度线程就返回到循环的开头，交出 ME 的控制权让 QM 消息处理线程运行。直到延时结束，

QM 消息处理线程把 ME 控制权交回来，线程才开始重新进行调度。如果 ffs 返回了调度得出的端口 ID，调度线程就把该端口的权值 port_credit 减去一。如果此时的 port_credit 值不为 0，则程序可以继续进行对当前端口的调度。如果此时的 port_credit 值为 0，则该端口在本轮端口调度中的配额已经使用完了，port_empty_vector 的对应位置为 0，在本论调度的剩余时间不能再被调度。为了保证该端口在下一轮调度开始能重新接受调度，所以把该端口的权值增量 port_credit_increment 加到 port_credit 上。此时程序查询是否有其他端口可以接受调度，如果没有的话，程序就重置所有端口的状态标志，使下一次程序循环开始时可以开始新一轮的调度。如果有的话，则不重置标志位。程序的各个分支运行到此时，都要开始对当前端口进行剩余的操作。首先要计算当前端口正在发送的数据包数量。该数量值的大小等于本端口已经调度的数据包数目，减去数据包发送程序块已经发送的数据包个数。如果正在发送数据包的数量大于门限值，则更新 port_round_robin_vector，重新回到循环的开头，等待新一轮的调度。否则就更新 port_round_robin_vector，开始对这个端口中的各个队列进行调度。

端口的队列调度中，同样有三个主要的队列状态矢量，sche_vector，queue_empty_vector 和 high_queue_mask。这三个状态矢量都存储于队列状态表中。各个矢量的含义如前所述。其中，high_queue_mask 在本程序中的作用与端口调度中的 port_round_robin_vector 矢量类似，因此每查询完一个队列后都要把该队列在 high_queue_mask 矢量中对应的一位置 0，在本次轮询中不再对该队列进行查询。在开始进行队列调度的时候，所作的操作与端口调度是类似的，也是先确定可以调度的队列（把几个状态矢量相与），再用 ffs 取得队列号。如果 ffs 失败的话，则不考虑 high_queue_mask，重新进行 ffs 操作，因为这个端口中是必定有队列有数据能进行调度的，否则端口调度就不会选中该端口。得出要调度的队列 ID 之后，程序按照队列 ID 相应地更新 high_queue_mask，并根据队列 ID 找到 DRR 参数表中该队列的 packet_count 一项，把这个值减一。如果 packet_count 为 0，则 queue_empty_vector 的对应位置 0，表示该队列为空。然后程序生成一个 dequeue 控制信息，并判断是否有其他队列可以进行调度。如果没有的话，则把 sche_vector 设为初始值，

以便下一轮循环开始的时候可以开始新一轮的调度。然后程序把前面生成的 dequeue 控制信息发送到 Scratch Ring6 中，传递给队列管理程序块。然后程序将该端口已经被调度的数据包计数值加一，并等待 Scratch Ring6 写操作的完成。当完成了等待之后，调度线程就执行线程交换，回到循环之初重新等待。

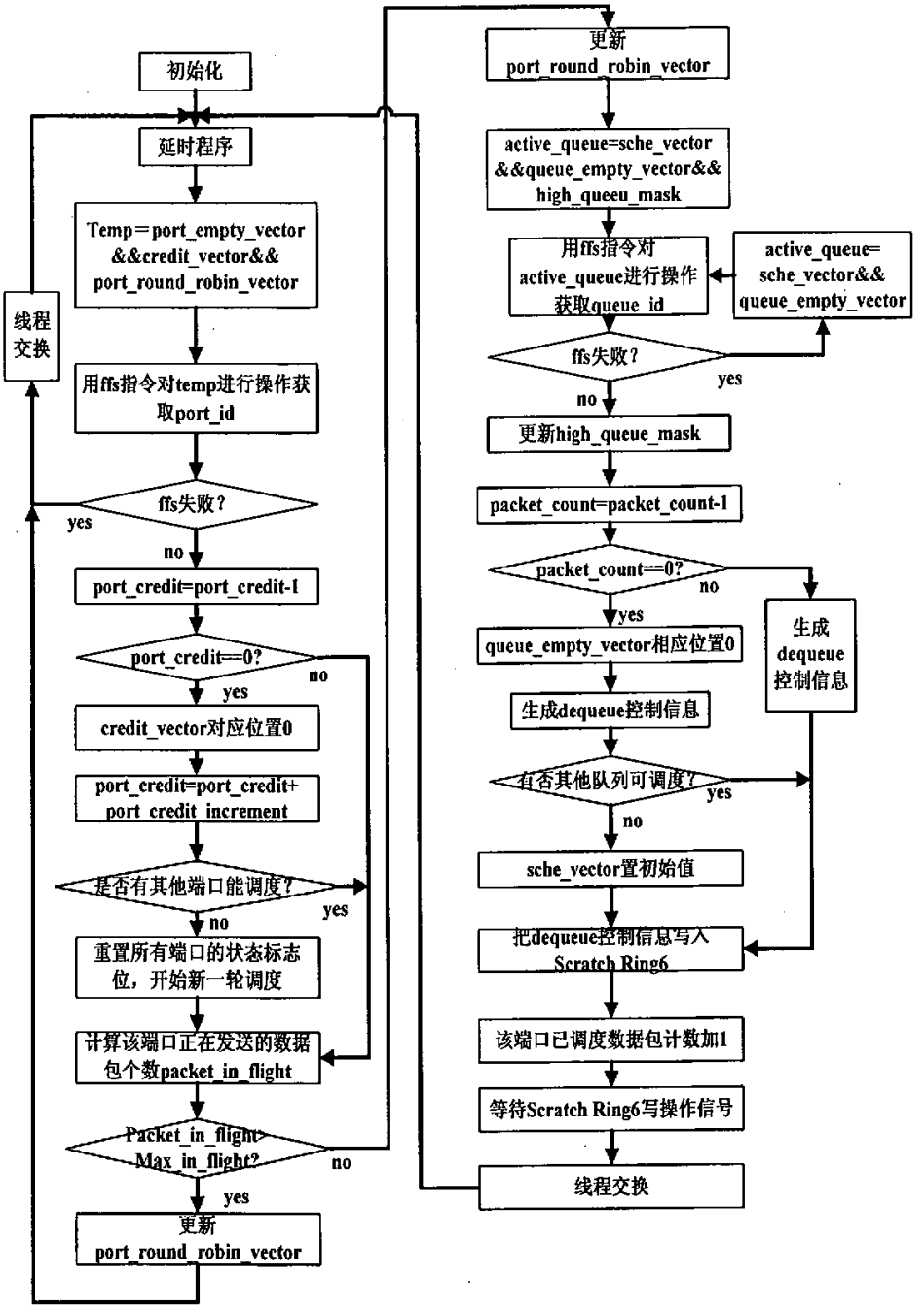

图 5-10 队列调度程序块-调度线程

另外一个线程是专门负责消息处理的QM消息处理线程，如图5-11所示。这个线程是专门负责从NNR接收队列管理程序块传送过来的消息，并根据这些消息的内容更新队列和端口的状态矢量。如图所示，QM 消息处理线程从调度线程处接管ME之后，就开始运行。本线程先判断NNR中是否有消息，如果没有则重新交出ME的控制权。如果有，则开始进行消息处理。首先判断是否收到enqueue消息，如果有则从enqueue消息中取出有数据包入列的队列号和端口号，并在相应端口的port_empty_vector和相应队列的queue_empty_vector中的对应位置1，表示该端口和该队列都非空。然后线程把DRR参数表中该队列的对应表项中的packet_count的值加一，这样就完成了对enqueue消息的操作，接着开始对dequeue消息的操作。如果没有enqueue消息，则直接跳到对dequeue消息操作。如果没有dequeue消息，则线程交换出ME，重新进入等待。否则就开始对dequeue消息进行处理。线程在dequeue消息中取出有数据包出列的队列号和出列的数据包长度packet_length（单位是chunk）。然后在DRR参数表中相应队列的表项中的queue_credit减去packet_length。如果此时queue_credit的值为负，则把该队列在队列状态表中的sche_vector的对应位置0，该队列在这一轮调度中的剩余时间不再有权发送数据包。程序再把queue_credit_increment加到当前的queue_credit上，使得队列在下一轮调度中有配额进行发送。接下来程序会检查是否有其他队列可以参加调度。如果没有，则表明此轮调度已经完成，可以进入下一轮调度，于是程序把sche_vector矢量置回初始值，以便开始下一轮调度。

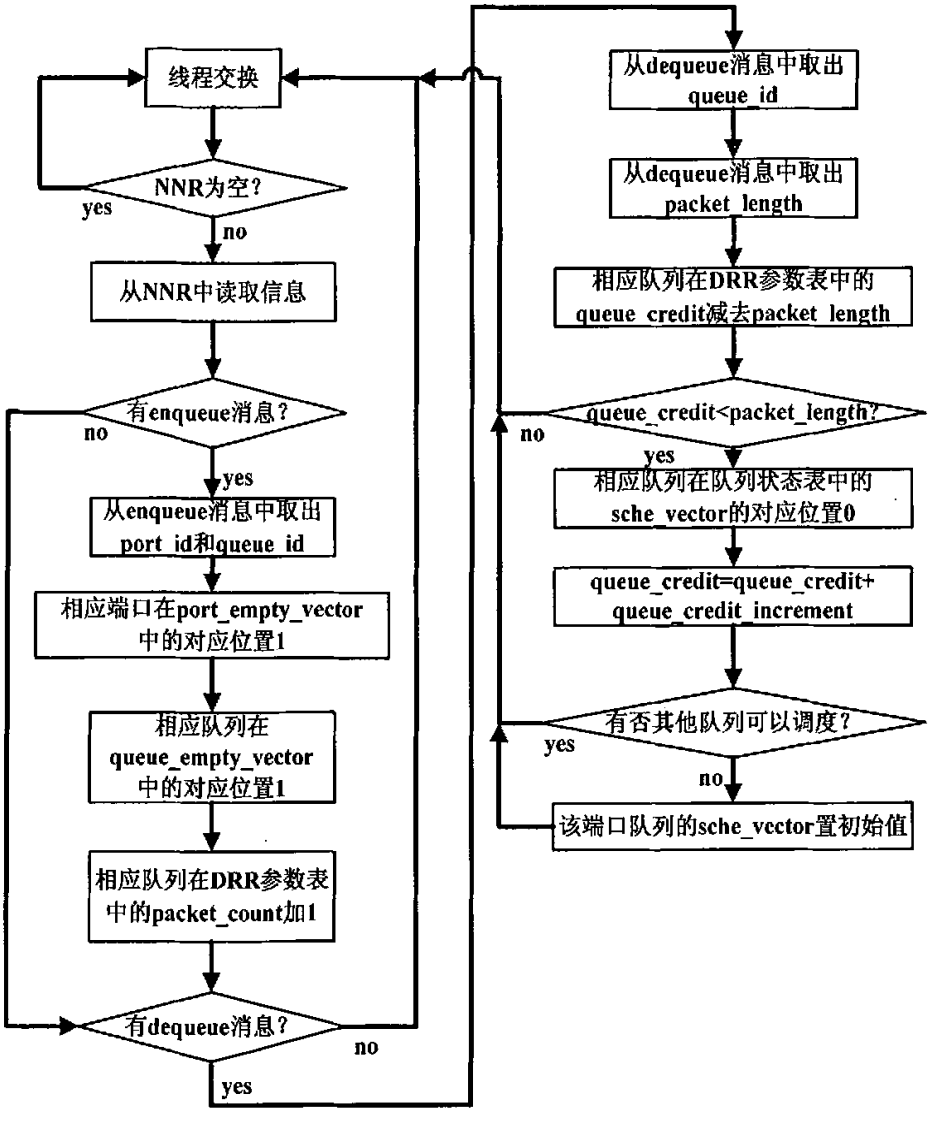

图 5-11 队列调度程序块-QM 消息处理线程

5.2.4 系统的参数设定

在本系统的实现中，涉及很多参数的设置，包括队列个数，各个队列的 maxth，minth 和丢弃概率，各个队列的配额值等。下面是本系统中对各个参数值的设置。

(1) 速率限制

前文提到，为了保护服务器，使到达服务器的请求速率不超过服务器的处理能力，本系统对网络处理器的服务速率进行限制，实现手段为对队列调度服务

的服务速率进行限制，从而限制服务器端的服务请求的到达速率，达到保护服务器的目的。为了提高系统的适应性，系统的队列调度服务延时模块的延时量是可调的，因此可以根据实际需要来设定相应的延时值。在系统测试时，考虑到测试使用的相关设备的性能，延时模块的延时量设为 10000。延时模块的延时值通过下面的式子计算：

延时值=延时量×16×(1/600 000 000)　　(5－1)

其中，600 000 000 是网络处理器的频率，16 是每次延时的时钟周期个数。当延时量为 10000 的时候，延时值约为 267 微秒，每秒调度次数为 3750。也就是说网络处理器每秒的发包个数不超过 3750 个。

(2) 队列的 RED 参数

在本系统中，考虑比较简单的情况，设置三个优先级不同的队列。按照每秒最高发包速率为 3750 个，考虑到计算方便，把三个队列的总长度设成 4000 个，然后把这个总长度按比例分配给三个队列。分配比例由这三个队列对应的正常流的流量决定。如果将正常流的平均对数或然概率出现的最大频率设为 FRQ，我们将三个队列与或然概率作如下对应：

若数据包的对数或然概率对应的纵坐标（即出现频率）在（1/2 FRQ，FRQ] 之间，那么该数据包属于第一队列；

若数据包的对数或然概率对应的纵坐标（即出现频率）在（1/4 FRQ，1/2 FRQ] 之间，那么该数据包属于第二队列；

若数据包的对数或然概率对应的纵坐标（即出现频率）在（1/8 FRQ，1/4 FRQ] 之间，那么该数据包属于第三队列；

若数据包的对数或然概率对应的纵坐标（即出现频率）在（0，1/8 FRQ] 之间，则认为这个数据包的行为偏离正常过远，可以认为这种流量会被本系统前端的用户识别系统所过滤，本系统中暂时不考虑为这种流量提供服务。

根据以上的对应关系，本文取“98 世界杯正常流或然概率统计结果”为样本，可以计算出三个队列的队列长度比例为 0.6568 ： 0.1685 ： 0.086，相应地三个队列的最大长度比值为 0.6568 ： 0.1685 ： 0.086。因而有

maxth1 =4000 * 0.6568/(0.6568+0.1685+0.086) = 4000 * 72.07% =2883

maxth2 =4000 * 0.1685/(0.6568+0.1685+0.086) = 4000 * 18.49% = 740

maxth3 =4000* 0.086/(0.6568+0.1685+0.086) = 4000 * 9.44% = 383

RED算法的提出者建议，maxth 与minth 的比值应该大于或等于2[23]，在实际应用中，常常取maxth 与minth 的比值为3[40]。考虑到DDoS攻击的特点，为了防止高速率的攻击流拥塞队列，队列的拥塞控制应该较早进行，因而本系统选取minth=0.3maxth，即

minth1 =2883 * 0.3= 865

minth2 =740 * 0.3 = 222

minth3 =383* 0.3 = 113

如前文所属，本系统实现时，以一个 32bit 的长字储存队列的丢弃概率（probability）。当队列长度 len 大于该队列的 minth 而小于该队列的 maxth 的时候，由网络处理器产生一个随机数，比较这个随机数与该队列的丢弃概率p的大小。如果随机数大于p，则不丢弃该数据包，否则就丢弃该数据包。因此丢弃概率的值越大，则数据包被丢弃的机会就约大。为了体现三个队列的优先级，三个队列的丢弃概率值分别设为：

p1=0x00008000，p2=0x00040000，p3=0x00c00000

则三个队列的RED参数如下表

表格 5-7 RED参数设置

队列ID	Maxth（单位：包）	Minth（单位：包）	Probability
队列1	2883	865	0x00008000
队列2	740	222	0x00040000
队列3	383	113	0x00c00000

(3) 队列的DRR参数

DRR算法的主要参数就是各个队列的配额值。队列的配额值实际上是反映各个队列的带宽分配情况的，而各个队列所获得的带宽应该跟其正常程度成正比，因为在正常情况下，队列中的用户正常程度越高，这个队列的流量就越大，所占的带宽也就越多。本系统为三个队列分配的带宽，同样依照0.6568 ：0.1685 ：0.086的比例，三个队列的配额比例近似为8：2：1，所以三个队列的配额相应设置为：

quantum1=48，quantum2=12，quantum3=6

第6章 系统测试结果及分析

6.1 系统的测试环境

本系统主要是通过实验网中的实验来进行测试。实验网的拓扑图如下。

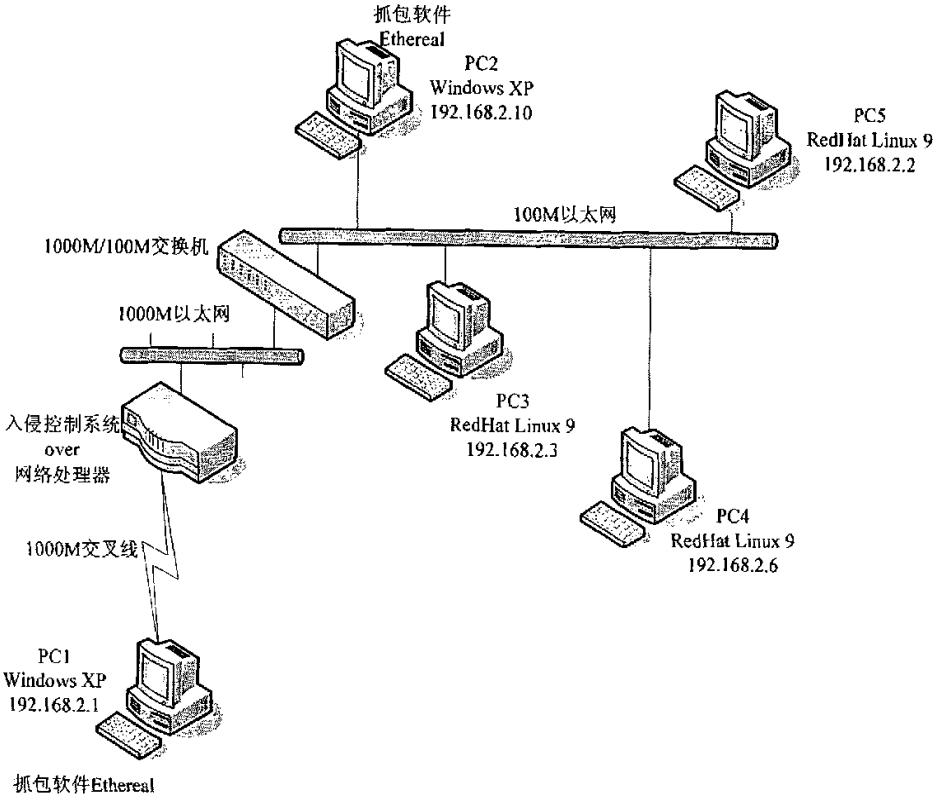

图 6-1 测试拓扑

实验网中的所有PC机的IP编址都在192.168.2.0这个网段内。其中PC2，PC3，PC4，和PC5都连接在交换机的100M端口上。把PC机直接连接在交换机上，是为了分割冲突域，增大吞吐量。在以太网中，由于其共享介质的本质，所以同一时间只有一台主机可以发送或者接收数据。而交换机的作用，就是能够分割冲突域。交换机的不同端口属于不同的冲突域，因此连接在不同端口上的主机可以同时发送数据，生成比较大的汇聚流量。交换机的千兆up_link口与网络处理器的port0相连，而网络处理器的port2则与PC1的千兆网口相连。测试的方法，就是要利用port0一侧的PC3，PC4，PC5生成不同的流量，让流量从port0进入

网络处理器，经过本控制系统的控制之后，再从 port2 流出。通过记录和对比通过控制系统之前的流量和通过控制系统之后的流量，测试本系统的功能。

6.2 系统的测试工具

系统测试中用到的工具主要包括流量生成工具和流量检测工具。下面对这两类工具进行简单的介绍。

(1) 流量生成工具。

流量生成工具的种类非常繁多，基本上所有可以发包的工具都可以充当流量生成工具。而在本系统的测试中，需要产生数据包大小可调，速率可控，数据包包头字段可按需求改写的流量，同时该工具使用应该尽量简单方便。根据这个要求，本系统选用了 hping3 和 tcpreplay 这两种工具作为发包工具。

a. Hping3 是一个运行在 Linux 上的免费的数据包产生和分析软件，一般用于防火墙和网络的安全测试和审计。不过它并非仅仅是一个 ICMP 请求/响应工具，它还支持 TCP、UDP、ICMP、RAW-IP 协议。如果 hping3 是以本机的 IP 为源地址发送数据包，则 hping3 还可以像 ping 程序一样，把目的主机的返回数据包显示出来。而通过这些回显信息，可以推断目标主机的状态和网络的状况。如下图所示。

```
[root@localhost ylq]# hping3  202.116.78.147
HPING 202.116.78.147 (eth0 202.116.78.147): NO FLAGS are set, 40 headers + 0 data bytes
len=46 ip=202.116.78.147 ttl=128 id=910 sport=0 flags=RA seq=0 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=911 sport=0 flags=RA seq=1 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=912 sport=0 flags=RA seq=2 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=913 sport=0 flags=RA seq=3 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=914 sport=0 flags=RA seq=4 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=916 sport=0 flags=RA seq=5 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=917 sport=0 flags=RA seq=6 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=918 sport=0 flags=RA seq=7 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=919 sport=0 flags=RA seq=8 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=920 sport=0 flags=RA seq=9 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=921 sport=0 flags=RA seq=10 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=923 sport=0 flags=RA seq=11 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=924 sport=0 flags=RA seq=12 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=925 sport=0 flags=RA seq=13 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=926 sport=0 flags=RA seq=14 win=0 rtt=0.1 ms
len=46 ip=202.116.78.147 ttl=128 id=927 sport=0 flags=RA seq=15 win=0 rtt=0.1 ms

--- 202.116.78.147 hping statistic ---
16 packets tramitted, 16 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.1 ms
```

图 6-2 hping3 的回显输出

当不加任何参数，直接使用 hping3 的时候，hping3 会向目标主机的较低的端口发送载荷长度为 0，各个标志位（FLAG）均不置为的 TCP 包，并且该 TCP

包头的 TCP Checksum 是错误的。当目标主机收到这种 TCP 包，如果该主机是没有做任何防护措施（如运行防火墙等），则该主机会向运行 hping3 的主机返回一个 TCP Checksum Error 的 TCP 包。Hing3 收到这个 TCP 包之后，就会计算 RTT（Round Trip Time）并输出相关信息，而网管人员可以根据这些信息分析网络的安全状况。Hping3 提供了丰富的参数和选项，使用者可以通过“-d”选项指定载荷的长度，通过“-a”选项指定假冒的源地址，通过“--flood”选项命令 hping3 发起泛洪。此时 hping3 就成为一个发出洪水攻击的攻击流的工具了。本系统测试实验中，利用 hping3 的这种灵活性和多样性，产生了包括正常流量和攻击流量在内的流量。

b. 与其他发包工具不同，tcpreplay 并不是一个自己产生数据包并发送的工具，而是一个把 tcpdump 记录的数据包重放到网络中，还原网络中的流量状态的工具。Tcpreplay 在不加其他参数的情况下，按照 dump 文件中的时间戳来发包，并且只发送 dump 文件中记录的 TCP 包头，而不发送其载荷。对于本系统的测试来说，tcpreplay 跟 hping3 相比，其优点为：可以指定发送数据包的目的 MAC，可以在命令控制下，在不超过 PC 机处理能力范围内以稳定的速率发送数据包。由于 tcpreplay 的这些优点，它可以与 hping3 良好地互补，满足本系统的测试需求。在使用时，先用 hping3 按测试需要生成一定大小的数据流，用 tcpdump 记录在 dump 文件中，然后在测试时以 tcpreplay 按照指定的速率发送数据包。

(2) 流量检测工具。

本系统测试时采用 Ethereal v0.10.15 作为流量检测工具。Ethereal 是一个开放源代码的网络分析系统，支持 Linux 和 Windows 平台，其强大的协议分析能力几乎媲美商业的网络分析系统。利用 Ethereal，可以对网络中的流量进行记录，协议分析和协议还原，流量统计等多种功能，并且能够以曲线图直观地图示出各项统计数据。Ethereal 的捕获数据可以以多种格式（文本文件，dump 文件等）存放，便于分析和流量重放。

6.3 系统的测试方法

进行系统测试时，网络中的流量由 PC3，PC4，PC5 利用流量产生工具生成，

然后流量通过网络处理器之后到达 PC1 端，测试结果通过对比通过网络处理器前的流量和通过网络处理器后的流量得出。在实际测试时，要考虑以下几点：

(1) 通过交换机相连的主机，由于相互之间处于不同的冲突域，发往某台特定主机的数据流是不会被转发到其他端口的，则连接在其他端口上的主机不能截获这些数据流的。同理，在本系统的测试环境中，PC3，PC4 和 PC5 发往网络处理器的数据，是不会被交换机转发出其他端口的。这样一来就无法通过连接在交换机上的主机记录 PC3，PC4 和 PC5 发出的流量了。如果把交换机换成集线器，则连在集线器上的机器都处于同一个冲突域，这样就可以解决这个问题。但是使用集线器会降低吞吐量，可能使各 PC 机的产生的流量不稳定。

(2) 当测试中存在流量比较大的非正常流的时候，交换机可能会出现丢包状况，则此时通过控制系统之前的流量就不等价于各个源所发出的流量之和。如果直接以源发送的数据流总和作为通过控制系统前的流量，与通过控制系统后的流量相比，则会造成较大的误差。

(3) 如果 PC3，PC4 和 PC5 所产生的流量以 PC1 的地址作为目标地址的话，当流量到达 PC1 的时候，PC1 会返回大量的 port unreachable 或 TCP bad checksum 信息。这些信息产生的流量会影响源的发包速率，同时增加了流量分析的复杂度。

基于以上几点考虑，系统进行测试时采用了如下方法：

以拓扑图所示进行连接，由其中的 PC3，PC4，PC5 产生网络中的流量，定义这些主机为流量的源，产生的流量为通过交换机前的流量。由 PC2 记录流过交换机之后通过控制系统之前的流量，并定义该流量为通过控制系统前的流量。以 PC1 记录流过网络处理器后的流量，并定义该流量为通过控制系统后的流量。为了让交换机同时向 PC2 和网络处理器转发数据，在源端（PC3，PC4，PC5）发送数据时，指定一个在本测试网络中不存在的目的 IP 和目的 MAC。根据交换机的工作原理，交换机收到数据帧之后，如果该数据帧的目的 MAC 不在交换机的端口—MAC 对应表中，则交换机会把该数据帧向除了收到该帧的端口外的其他所有端口转发。当指定的目的 MAC 在本网络中不存在时，交换机的端口—MAC 对应表中自然也没有该表项，因此交换机会同时把流量转发给网络处理器和 PC2，PC2 和

PC1 就能对流量进行记录。同时，由于该目的 IP 在本网络中也是不存在的，所以不会有返回的数据流生成。在测试中，把流量的目标 IP 设为 192.168.2.5，目的 MAC 设为 00-EE-AA-BB-CC-DD。流量的三个源（PC3，PC4，PC5）分别向这个目的发送数据流，形成三中不同的流量，分别对应本系统所设定的三个队列。实验网中各台 PC 的具体配置和作用如下表所示。

表格 6-1 PC 的配置和作用

PC	IP 地址	操作系统	运行软件	目标地址	作用
PC1	192.168.2.1	Windows XP	Ethereal	--	记录通过控制系统后的流量
PC2	192.168.2.10	Windows XP	Ethereal	--	记录通过控制系统前的流量
PC3	192.168.2.3	Red Hat 9	Tcpreplay	192.168.2.5	产生队列 1 的流量
PC4	192.168.2.6	Red Hat 9	Tcpreplay	192.168.2.5	产生队列 2 的流量
PC5	192.168.2.2	Red Hat 9	Hping3	192.168.2.5	产生队列 3 的流量

6.4 系统测试中涉及的定义

流量的源：指 PC3，PC4，PC5。

源平均速率：指 PC3，PC4，和 PC5 上发包软件所统计的平均速率或根据发包软件的发包统计所计算出的平均速率。其中，对于 PC3 和 PC4，其上运行的 tcpreplay 软件在发送结束时会显示其平均发包速率。而 PC5 上的 hping3 在发包结束时会显示其发包的总个数，则对于 PC5 来说，其源平均速率的值就是发包总数除以在 PC2 上截获的流量所持续的时间。

通过交换机前的流量：指 PC3，PC4，PC5 所发出的流量总和。

通过控制系统前的流量：指 PC2 所截获的流量。

通过控制系统后的流量：指 PC1 所截获的流量。

丢包率：指同一队列通过控制系统前的流量减去通过控制系统后的流量再除以通过控制系统后的流量所得到的百分比。

6.5 系统的测试结果

(1) 测试一：各个队列的带宽比例。

前面提到，本系统为三个队列所分配的配额分别为：48，12，6，即比例为 8：2：1。本系统测试的第一项，就是验证该带宽分配比例是否能起到作用。首先通过上面的比例计算出各个队列理论上的带宽分别为：

3750*8/(8+2+1)=2727（包/秒）

3750*2/(8+2+1)=682（包/秒）

3750*1/(8+2+1)=341（包/秒）

以上带宽分配产生的条件，是每一轮调度时各个队列中都有数据，即每个队列都能全额使用分配给自己的带宽。因为如果有队列中的数据流没有全额使用该队列的带宽的话，也就是存在当其他队列中有数据包的时候，这个队列中没有数据包，则其他队列就多了被调度到的机会，也就是其他队列会分享这个队列没有使用到的带宽，因此控制系统所发送的流量就不能体现带宽分配比例了。为此，在进行测试一的时候，各个流量源都以高于本队列理论带宽的速率发包，以达到各个队列都满载的效果，得出各个队列的带宽比例。测试一的设置如下表。

表格 6-2 测试一的设置

队列	流量源地址	流量目的地址	源主机	源平均速率（包/秒）	包长度（bytes）
队列 1	192.168.2.3	192.168.2.5	PC2	3503	140
队列 2	192.168.2.6	192.168.2.5	PC3	1000	140
队列 3	192.168.2.9	192.168.2.5	PC4	9769	140

三个队列通过控制系统后的流量如图 6-3 所示。时间轴时刻是按照 Ethereal 抓包软件记录的时间，是以 Ethereal 开始运行为时间 0 点。由于三个流量在开始的第一秒和最后一秒均没达到稳定状态，而这些不稳定状态对本系统的测试意义不大而且容易引起误解，所以下面的图示都去掉了各个队列的流量的第一秒和最后一秒，仅画出流量处于稳定时的状态。从图中的曲线可以看出，三个队列的

流量通过控制系统之后，基本上符合 8：2：1 的比例。对比图 6-4，可以看出，队列 3 的流量在通过控制系统前是所有队列的流量中最大的，所占的带宽也最多。但由于队列 3 在控制系统中是属于正常程度最低的，其带宽配额也是最小的，所以通过控制系统后，队列 3 的流量所占的带宽比例就变成最小了。可见，在各个队列都满载的情况之下，通过控制系统的各个队列的流量只与其自身分配的带宽大小有关，而与该流量通过控制系统前所占的带宽没有直接联系。

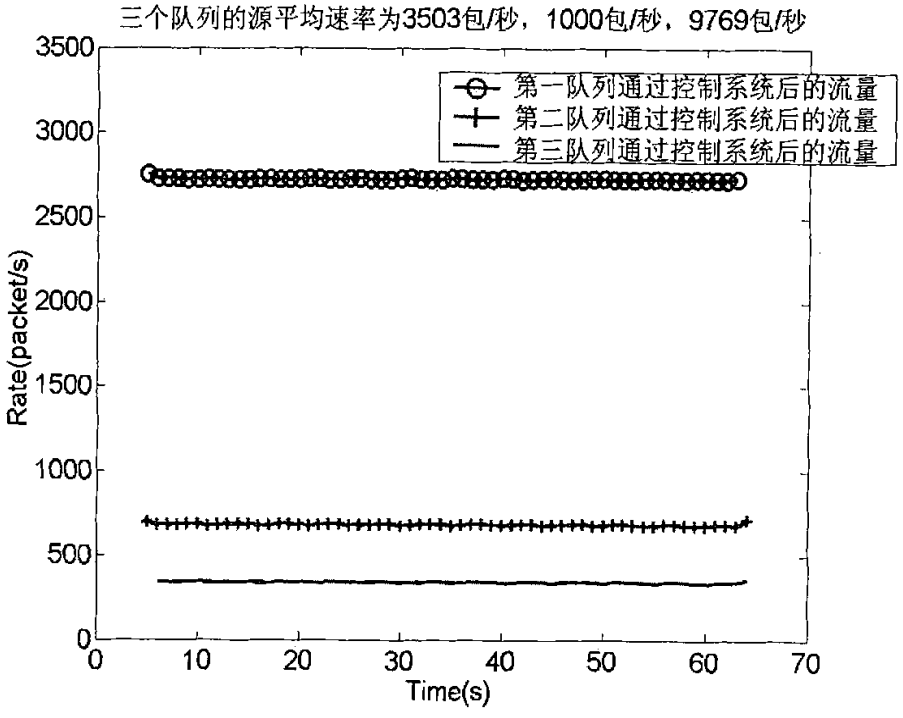

图 6-3 三个队列通过控制系统后的流量

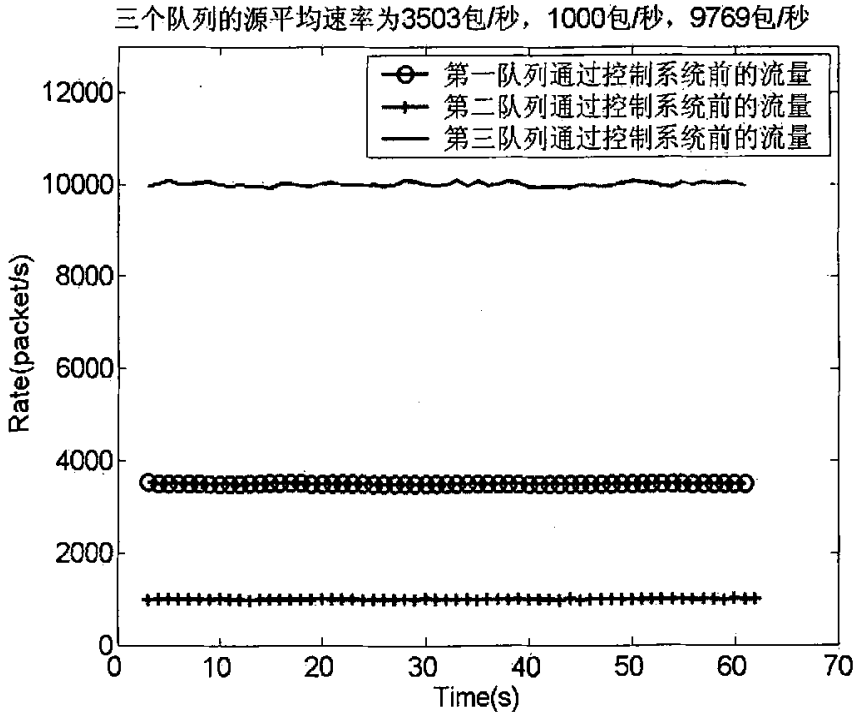

图 6-4 三个队列通过控制系统前的流量

三个队列通过控制系统前的流量与通过控制系统后的流量对比如下表。在表中计算通过系统前和通过系统后的平均流量时，去掉该流量中第一秒的统计值和最后一秒的统计值，然后对图示的稳定状态的流量的统计值求和再计算其平均值，从而获得流量在稳定状态下的平均速率。

表格 6-3 测试一的统计结果

队列	通过控制系统前的流量		通过控制系统后的流量		丢包率
	总流量(包)	平均速率（包/秒）	总流量（包）	平均速率（包/秒）	
队列 1	210911	3509	164828	2726	21.85%
队列 2	61450	1001	42923	682	30.15%
队列 3	595788	10006	20669	341	95.93%

由表中的数据可知，各个队列流量通过控制系统后的速率与前面根据带宽分配所计算的值相等或非常接近，因此可以认为本系统能够有效地控制各个队列的流量，达到按要求分配各个队列带宽的目的。

(2) 测试二：各个队列的流量均在正常范围以内。

确定了三个队列的带宽比例以后，就可以根据带宽比例，控制流量源产生不同大小的流量来代表不同的网络状况。测试二的目的是测试各个队列的流量大小都在其额定带宽以内，也就是各个队列的流量都属正常时，控制系统对流量的影响。测试二中的设置如下表。

表格 6-4 测试二的设置

队列	流量源地址	流量目的地址	源主机	源平均速率（包/秒）	包长度（bytes）
队列 1	192.168.2.3	192.168.2.5	PC2	2702	140
队列 2	192.168.2.6	192.168.2.5	PC3	680	140
队列 3	192.168.2.9	192.168.2.5	PC4	99	140

三个队列中，第二队列的流量大小是最接近该队列分配的带宽的，而第三队列的流量大小与分配给它的带宽相差是最大的。图 6-5 是第一队列的流量示意图。

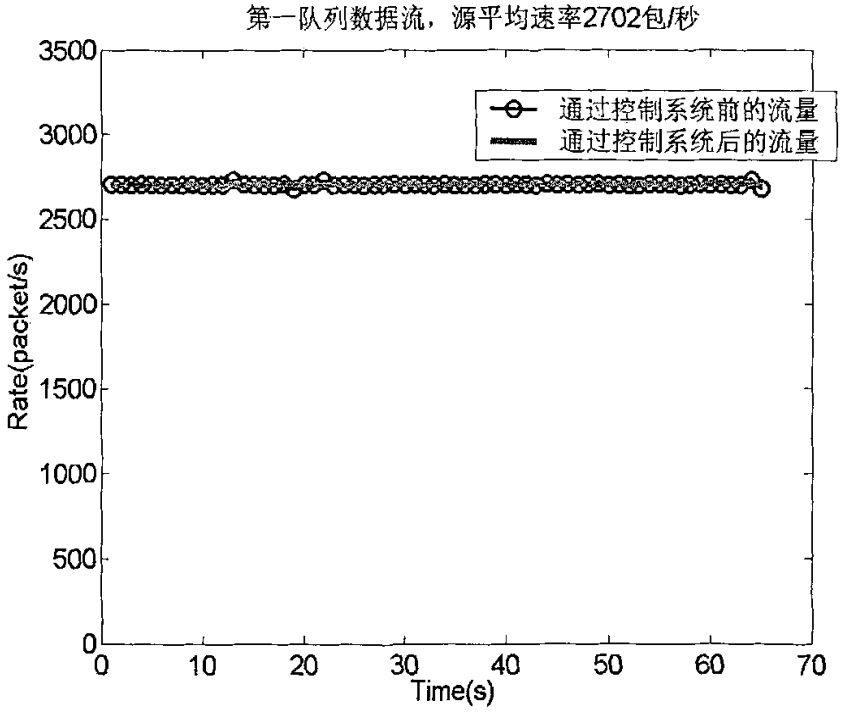

图 6-5 第一队列流量示意图

第二队列的流量如图 6－6 所示。

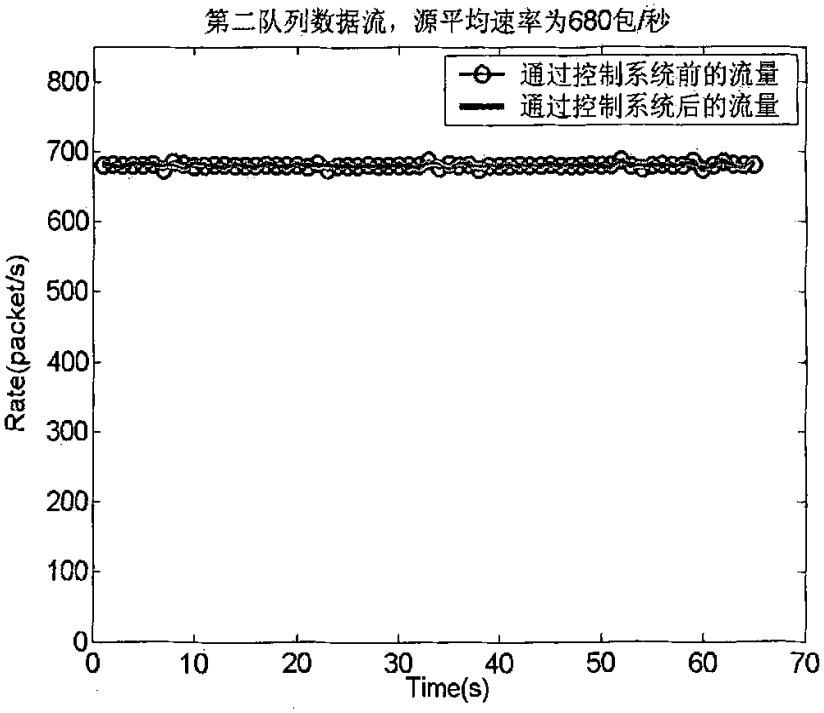

图 6-6 第二队列流量示意图

第三队列的流量如图 6－7 所示。

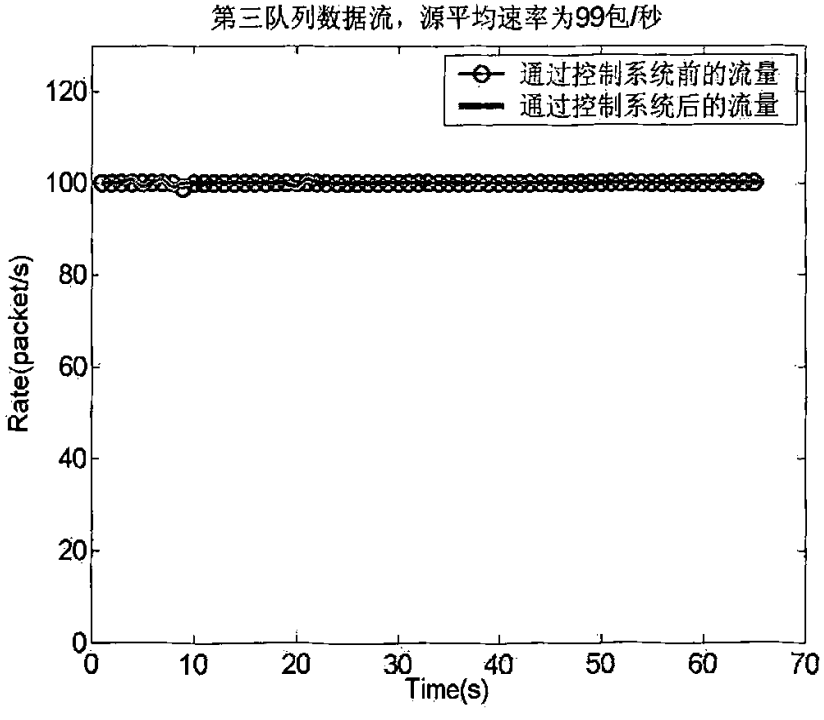

图 6-7 第三队列流量示意图

由以上三幅流量图，可以看出，各个队列的通过控制系统前的流量曲线和通过控制系统后的流量曲线是基本上重合的。也就是说，在网络中各个队列的流量大小都在正常范围以内时，本控制系统对各个队列的流量的速率基本上没有影响，各个队列的流量都能够在通过控制系统之后，保持其原本的速率。以上三幅图片均忽略了网络处理器的处理时延。

三个队列的流量统计如下表所示。在计算平均速率的时候，只考虑队列流量平稳的时间段，去掉流量开始的第一秒和最后一秒。

表格 6-5 测试二的统计结果

队列号	通过控制系统前的流量		通过控制系统后的流量		丢包率
	总流量(包)	平均速率（包/秒）	总流量（包）	平均速率（包/秒）	
队列 1	177968	2703	177968	2703	0.00%
队列 2	44839	680	44839	680	0.00%
队列 3	6653	100	6653	100	0.00%

三个队列的流量通过控制系统前和通过控制系统后的包间隔统计如下表所示。两个数据包之间的包间隔由抓包软件所记录的两个数据包的时间相减所得。

表格 6-6 测试二中三个队列的包间隔统计

队列号	通过控制系统前的流量		通过控制系统后的流量	
	包间隔均值(微秒)	包间隔方差	包间隔均值(微秒)	包间隔方差
队列 1	369.6	2530.6	369.6	485.8
队列 2	1469.9	4640.6	1469.8	3375.8
队列 3	10001	854.8	10000	755.2

如上表所示，各个队列数据流在通过控制系统前和通过控制系统后，平均包间隔基本上没有变化，而通过控制系统后的包间隔的方差比通过控制系统前的包间隔的方差小，也就是说控制系统对流量起到缓冲平滑的作用，能减少流量的时延抖动。

(3) 测试三：第一队列中的数据流量大于其额定带宽

第一队列，是系统在设置时认为正常程度最高，给与的优先权最高，带宽配额最大的队列。第一队列出现攻击数据流的概率比其他队列出现攻击数据流的概

率要小，而且就算真的有攻击数据流能成功假冒正常的流量，其所花的代价必然比较大，因而其速率必定不会太高。在测试三中，假设第一队列的数据流中混入了少量的攻击数据流，使得第一队列的流量大小大于其额定的带宽值，观测这种情况下其他队列的流量受到的影响。测试三的设置如下表。

表格 6-7 测试三的设置

队列	流量源地址	流量目的地址	源主机	源平均速率（包/秒）	包长度（bytes）
队列 1	192.168.2.3	192.168.2.5	PC2	3504	140
队列 2	192.168.2.6	192.168.2.5	PC3	680	140
队列 3	192.168.2.9	192.168.2.5	PC4	98	140

图 6-8，6-9，6-10 分别为三个队列的流量在通过控制系统之前和通过控制系统以后的对比示意图。

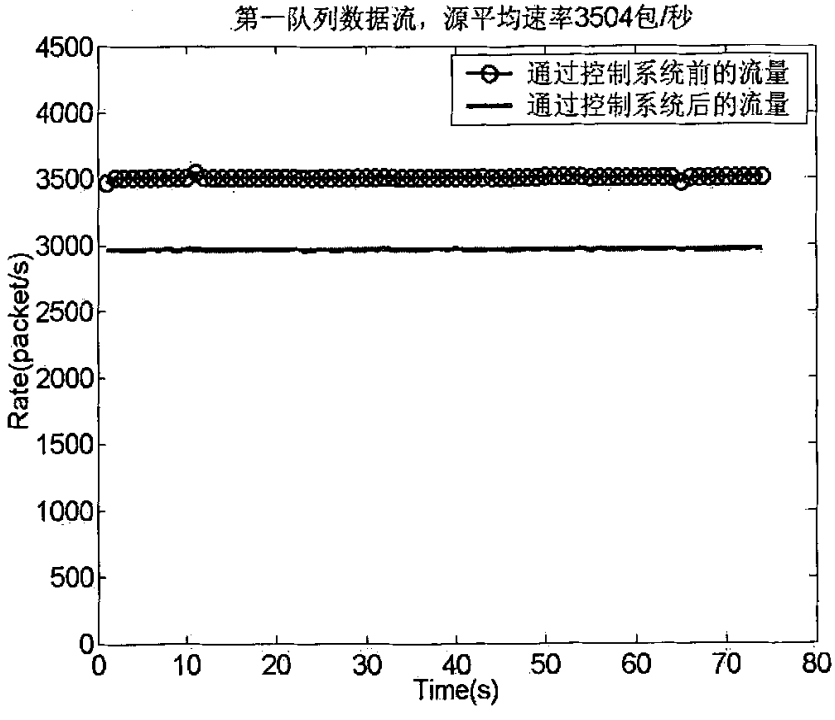

图 6-8 第一队列流量示意图

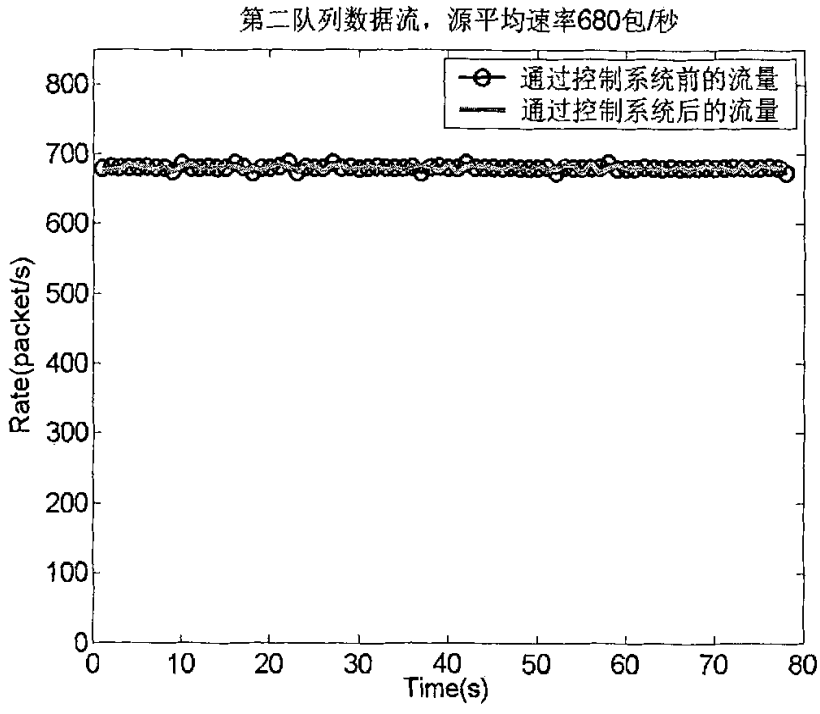

图 6-9 第二队列流量示意图

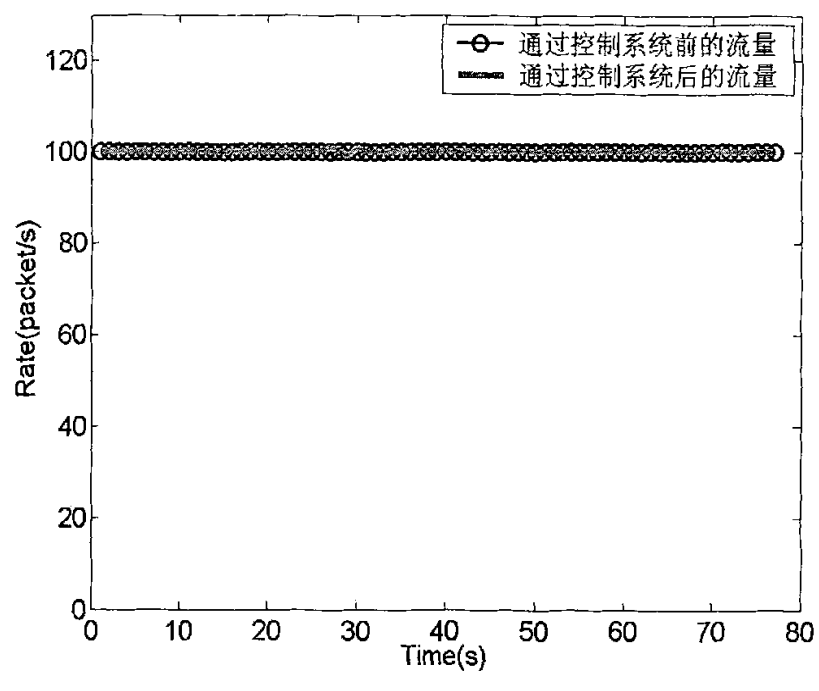

图 6-10 第三队列流量示意图

由以上三个图可以看出，当第一队列的流量超出其额定带宽的时候，其通过控制系统后的流量与通过控制系统前的流量相比，速率明显减少。而另外两个队列的通过控制系统前的流量曲线和通过控制系统后的流量曲线是基本上重合的，即这两个队列通过控制系统后其速率没有受到第一队列的流量的影响。也就是说，虽然第一队列的流量拥有的额定带宽比较大，其正常程度被认为是所有队列的流量中最大的，但当其流量出现不正常时——在本次测试中表现为其流量大于其额定带宽——该队列的流量并不能掠夺其他正常使用带宽的队列要使用的带宽。也就是说，其他队列仍然能够正常获得他们额定带宽以内的带宽。

另外在图 6-8 中可以看到，第一队列的流量通过控制系统后其速率大概为3000 包/秒。这个速率仍然是大于第一队列的额定带宽值。产生这个现象的原因是，其他两个队列并没有把它们的额定带宽用完。也就是说，出现了某几轮调度执行的时候，其他两个队列中都没有数据包，则第一队列就多了发送数据包的机会，也就是说，第一队列可以使用其他队列没有使用的带宽，因此第一队列能够发送超过其额定带宽的数据包。这是本系统的队列调度算法能够充分合理利用带宽的一种表现（参见“队列调度”一节）。而这种情况并不会对其他队列的用户造成拒绝服务。首先是因为本控制系统是可以根据服务器的处理能力限制到达服务器的流量速率的，这些攻击流的速率是不能超过控制系统限定的速率的，所以不会影响服务器；其次是因为，虽然该队列能够发送超过其额定带宽的数据流，但是这是在其他队列没有满额使用自己的额定带宽的前提下的，当其他队列的流量满额使用自己队列的带宽时（如测试一中的情况），该队列是只能使用其额定带宽发送数据包的。也就是说，该队列的流量是无法干预其他队列中的流量使用其额定带宽以内的带宽的，每一个队列的流量使用其额定带宽以内的带宽，都是受到保护的。

三个队列的流量统计如下表所示。在计算平均速率的时候，只考虑队列流量平稳的时间段，去掉流量开始的第一秒和结束的最后一秒。

表格 6-8 测试三的统计结果

队列号	通过控制系统前的流量		通过控制系统后的流量		丢包率
	总流量(包)	平均速率（包/秒）	总流量（包）	平均速率（包/秒）	
队列 1	261272	3508	221734	2967	15.13%
队列 2	53154	680	53154	680	0.00%
队列 3	7733	100	7733	100	0.00%

(4) 测试四：第三队列的流量为泛洪攻击流量。

测试四要测试的情况就是，当有流量极大的泛洪攻击流量出现的时候，控制系统在流量识别系统能正确地识别出攻击流量的前提下，对各个流量的控制和影响的情况。本次测试中考虑比较简单的情况，即队列三中只有攻击流量，没有正常的访问流量。测试四的设置如下表所示。

表格 6-9 测试四的设置

队列	流量源地址	流量目的地址	源主机	源平均速率（包/秒）	包长度（bytes）
队列 1	192.168.2.3	192.168.2.5	PC2	2603	140
队列 2	192.168.2.6	192.168.2.5	PC3	670	140
队列 3	192.168.2.9	192.168.2.5	PC4	10208	140

三个队列流量在通过控制系统之前和通过控制系统以后的对比如下面三幅图所示。

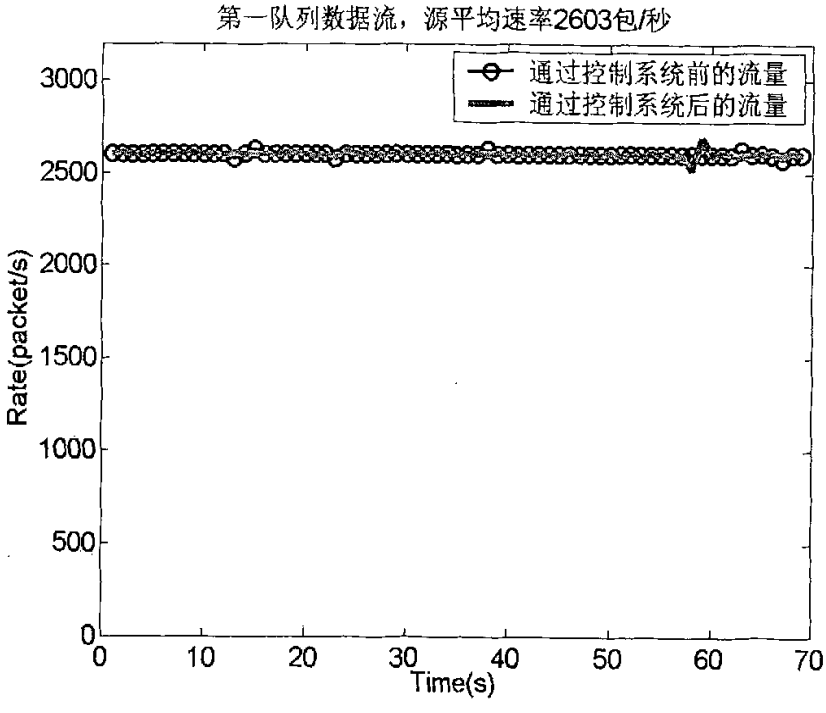

图 6-11 第一队列流量示意图

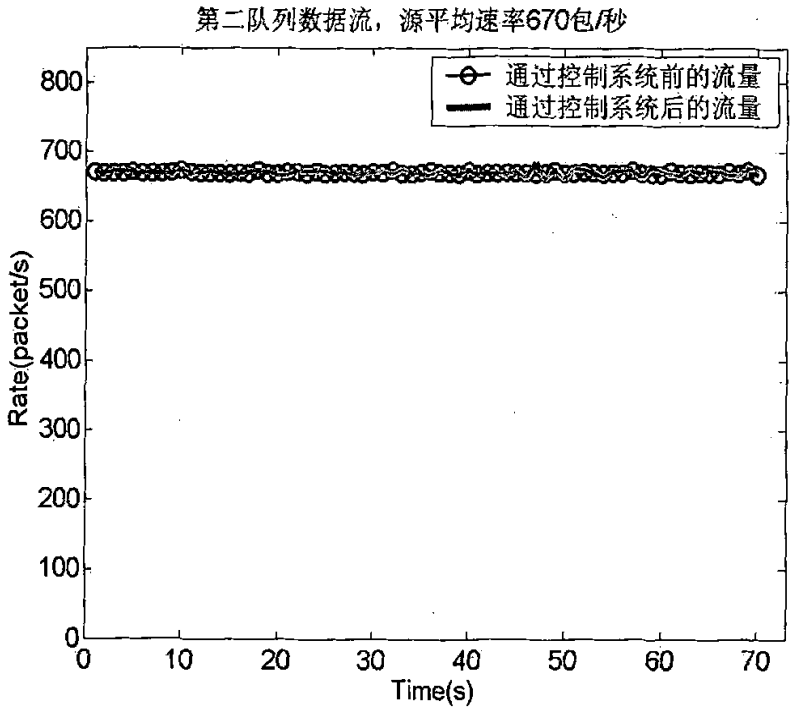

图 6-12 第二队列流量示意图

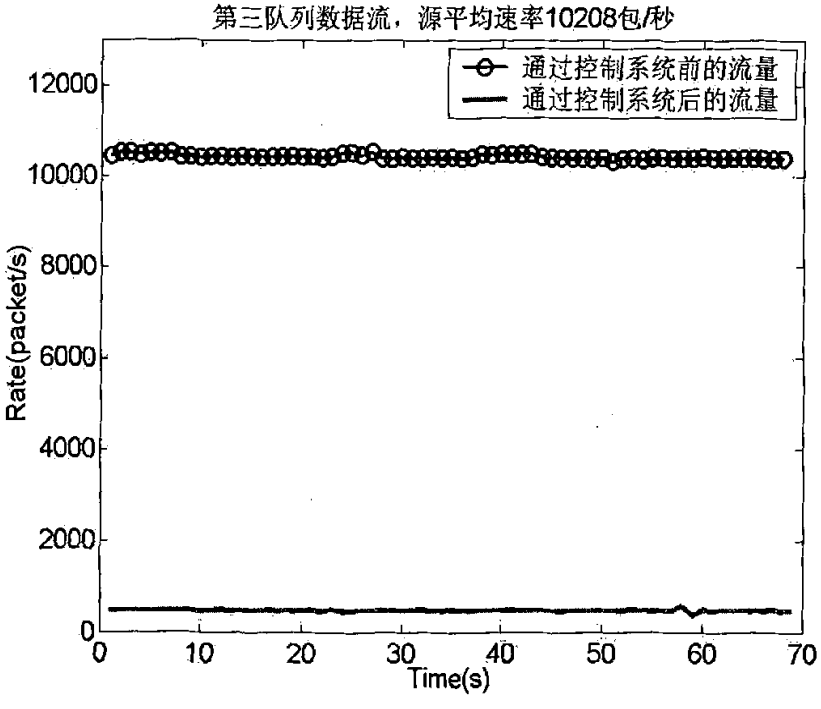

图 6-13 第三队列流量示意图

三个队列的流量统计如下表所示。在计算平均速率的时候，只考虑队列流量平稳的时间段，去掉流量开始的第一秒和最后一秒。

表格 6-10 测试四的统计结果

队列号	通过控制系统前的流量		通过控制系统后的流量		丢包率
	总流量(包)	平均速率（包/秒）	总流量（包）	平均速率（包/秒）	
队列 1	183513	2604	183513	2604	0.00%
队列 2	47994	670	47994	670	0.00%
队列 3	714558	10433	32809	473	95.41%

以上图片和统计数据表明，在泛洪攻击出现时，本控制系统可以把识别出来的攻击流量控制在一个非常小的水平，而对正常流量则影响极小。证明本控制系统能够有效地保障不同队列的带宽分配和使用，在按照流量的正常程度进行流量分类的基础上，本控制系统可以为正常程度高的流量提供较大的带宽，而对正常程度低的流量则不是一律丢弃，而是提供较低的带宽，允许一部分流量通过，以此减少对检测系统的依赖，提高容错的能力。

第7章 总结与展望

7.1 小结

在当今的网络时代，网络与人们的工作生活结合得日益紧密，而大型网站的安全问题也成为备受关注的问题。DDoS 攻击由于其良好的隐蔽性和强大的破坏力，给大型网站带来了严重的威胁。控制和防御 DDoS 攻击的实用系统亟待提出。

入侵检测控制系统一般由两个部分组成，第一部分是对分组进行检测，第二部分是针对可疑分组进行控制。在目前的实用系统中，一般把重点放在前一部分，而第二部分则只进行简单的丢弃或者过滤。这种单一门限的控制方式过分依赖检测系统，不能有效地保证正常用户的流量。

本文从用户分类出发，基于为不同正常程度的用户提供不同的服务这一思想，利用流量控制技术和网络处理器技术，实现了一个基于网络处理器的入侵控制系统。本系统根据用户的不同正常程度，把用户分类到拥有不同带宽的队列当中，在保证各个队列正常使用其额定带宽的权利的同时，有效地限制任何一个流量掠夺其他队列的应得带宽。本系统的实现方法既避免了单纯的过滤和丢弃，减低了误判漏判的影响，也有效地起到保护正常流量和限制攻击流量的作用。同时，基于网络处理器的实现使本系统具有良好的稳定性，可扩展性和灵活性，可以满足未来网络发展的要求。

7.2 未来的工作

本系统目前还处于独立测试阶段，但最终要作为课题组的服务器前端 DDoS 入侵防御系统中的一部分与系统的其他组成部分进行联合调试，组成一个完备的 DDoS 入侵防御系统。而本系统的功能也需要进一步的完善，以便更全面地起到控制和防御 DDoS 攻击的作用。后续的研究工作可以包括：

(1) 联合用户分类与识别系统和正常程度计算系统进行调试，讨论资源分配，参数设置和模块间的同步等问题。

(2) 加入专门针对没有 cookie 的数据包进行处理的速率控制模块。目前本系

统的实现是基于前端的用户鉴别及分类系统的能根据 cookie 区分用户、然后提供用户 ID 给本系统的功能。但实际上用户发送包含 cookie 的 HTTP 数据包之前，会先进行 TCP 三次握手。而这些数据包是不包含 cookie 的，而且这些数据包同样可能造成 DDoS 攻击。要保证正常用户的三次握手顺利完成，又要防范 DDoS 攻击，就要为本系统添加相应的处理模块，对这种流量进行速率控制。

(3) 由于网络中的情况是会随时间的推移而变化的，因此增加系统对网络流量的适应性非常重要。可以考虑增加一个离线的参数重估和半静态调整模块，通过对网络中的流量的一段时间内的学习，对系统的队列参数如 RED 参数和 DRR 参数等进行调整，以适应网络情况的变化。

参考文献

[1] 王学飞“DDoS的新发展和对策”，计算机应用与软件，2004年5月，第21卷第5期，99页－101页。

[2] J. Mirkovic, J. Martin, and P. Reiher, “A taxonomy of DDoS attacks and DDoS defense mechanisms”, Technical report 020018.Computer Science Dept., University of California, Los Angeles.

[3] 蔡一兵　石晶林“下一代网络设备核心单元－网络处理器应用研究”http://www.ednchina.com/Article/html/2006-02/2006226113030.htm

[4] DeokJo Jeon “Understanding DDOS Attack, Tools and Free Anti-tools with Recommendation”April 7, 2001,in SANS Institute,Information Security Reading Room. http://www.sans.org/infosecFAQ/threats/understanding_ddos.htm

[5] Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysal, Edward Knightly, “DDoS-Resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection”, in Proceedings of IEEE INFOCOM 2006, Barcelona, Spain, April 2006. http://www.ece.rice.edu/networks/papers/dos-sched.pdf

[6] 吕慧勤，张宏，罗守山，杨义先，“无缓冲服务的 DOS 攻击”，北京邮电大学学报，Sep,2003,Vol 26,No.3，61页－65页。

[7] 罗新密，“DDOS攻击防范策略研究”，湖南商学院学报，may 2003,vol.10，No.3。135页－136页。

[8] P. Ferguson and D. Senie, “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing”, RFC 2827, 2000.

[9] K. Park and H. Lee, “On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets”，http://www.cs.cornell.edu/People/egs/syslunch-spring02/syslunchsp02/park-lee.pdf

[10] J. Mirkovic, G. Prier, P. Reiher, “Source-end DDoS defense”, NCA 2003. Second IEEE International Symposium, pp. 171 – 178, April 2003.

[11] D.K.Y. Yau, J.C.S. Lui, Liang Feng, Yam Yeung, “Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles”, IEEE/ACM Trans. Networking, pp. 29 – 42, Feb. 2005.

[12] Haining Wang; Shin, K.G. “Transport-aware IP routers: a built-in protection mechanism to counter DDoS attacks”; Parallel and Distributed Systems, IEEE Transactions on Volume 14, Issue 9, Sept. 2003 Page(s):873 – 884

[13] Yoohwan Kim; Ju-Yeon Jo; Chao, H.J.; Merat, F.; “High-speed router filter for blocking TCP flooding under DDoS attack”; Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International 9-11 April 2003 Page(s):183 - 190

[14] Stefan Savage, David Wetherall, Anna Karlin and Tom Anderson, “Network support for IP traceback”, IEEE/ACM Trans. Networking, pp. 226-237, June 2001.

[15] Alex C. Snoeren, Carig Partridge and Luis A. Sanchez et al, “Hash-based IP traceback”, http://www.bbn.com/docs/whitepapers/spie-sigcomm01.pdf

[16] S. M. Bellovin, “ICMP traceback messages”, Internet Draft, 2000.

[17] Kalman K.K.Wan and Rocky K.C.Chang, “Engineering of a Global Defense Infrastructure for DDoS Attacks”, Networks, 2002. ICON 2002. 10th IEEE International Conference on

27-30 Aug. 2002 Page(s):419 – 427.

[18]任立勇，卢显良，“Internet 拥塞控制研究”，电子科技大学学报，2002 年 2 月，第 31 卷，第 1 期，48 页－52 页。

[19]J. Nagle, “Congestion Control in IP/TCP Internetworks”, RFC 896, Jan. 1984.

[20]B. Sikdar, S. Kalyanaraman, K.S. Vastola, “Analytic models for the latency and steady-state throughput of TCP Tahoe, Reno, and SACK”, IEEE/ACM Transactions on Networking, Vol. 11, Issue 6, pp.959 – 971, Dec. 2003.

[21]M. Allman, V. Paxson and W. Stevens, “TCP Congestion Control”, RFC 2581, April 1999.

[22]B. Braden, D. Clark, J. Crowcroft et al, “Recommendations on Queue Management and Congestion Avoidance in the Internet”, RFC2309, April 1998.

[23]S. Floyd, V. Jacobson, “Random early detection gateways for congestion avoidance”, IEEE/ACM Trans. Networking, Vol. 1 , Issue 4, pp. 397-413, Aug. 1993.

[24]Chuck Semeria, “Supporting Differentiated Service Classes: Queue Scheduling Disciplines”, http://www.juniper.net/solutions/literature/white_papers/200020.pdf

[25]D.I. Choi, Y.Lee, “Performance analysis of a dynamic priority queue for traffic control of bursty traffic in ATM networks”, Communications, IEE Proceedings- , Vol. 148, Issue 3, pp. 181 – 187, June 2001.

[26]Demers, A., Keshav, S., and S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm”, Proceedings of ACM SIGCOMM '89, pp. 3-12.

[27]J. Wang, Y. Levy, “Managing performance using weighted round-robin”, Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on, 3-6, pp. 785 – 792, July 2000.

[28]S. Nananukul, “Latency of weighted round-robin scheduler”, Electronics Letters, Vol. 39, Issue 2 , 23, pp. 256 – 257, Jan. 2003

[29]G. Quadros, A. Alves, E. Monteiro, F. Boavida, “How unfair can weighted fair queuing be?”, Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on, 3-6, pp. 779 – 784, July 2000.

[30]M. Bartoli, P. Castelli and F. Saluta., “Statistical Performance analysis of a drr scheduler”, http://exp.telecomitalialab.com/upload/articoli/V02N03Art287.pdf

[31]M. Shreedhar, G. Varghese, “Efficient fair queuing using deficit round-robin”, IEEE/ACM Trans. Networking, Vol. 4, Issue 3, pp. 375 – 385, June 1996.

[32]Chuck Semeria, “Supporting Differentiated Service Classes: Queue Scheduling Disciplines”, http://www.juniper.net/solutions/literature/white_papers/200020.pdf

[33]张宏科，苏伟，武勇《网络处理器原理与技术》。

[34]Wenjiang Zhou; Chuang Lin; Yin Li; Zhangxi Tan, “Queue management for QoS provision build on network processor”, Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of 28-30 May 2003 Page(s):219 – 224

[35]罗华，张思东，张宏科“基于 IXP2400 网络处理器的网络防护系统设计和实现” 电信快报 2004 年 No.11，38 页－41 页。

[36]Esam Khan, M.Watheq El-Kharashi, A.N.M. Ehtesham Rafiq, Fayez Gebali, and Mostafa Adb-El-Barr “Network Processors for Communication Security : A Review”

[37]Intel IXP2400 Network Processor Hardware Reference Manual

[38]Wenjiang Zhou; Chuang Lin; Yin Li; Zhangxi Tan, “Queue management for QoS provision

build on network processor", Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of 28-30 May 2003 Page(s):219 – 224

[39]余顺争，“大型网站业务流的统计异常检测”，待发表

[40]H. Wang, K.G. Shin, "Refined design of random early detection gateways", Global Telecommunications Conference, Vol. 1b, pp.769 - 775, 1999.

致谢语

随着论文的完成，我的大学生活也将接近尾声了。在这两年的学习和生活中，实验室的各位老师以及同学都给予我非常多的指导和帮助，在此对他们表示衷心的感谢！

首先感谢我的导师余顺争教授。自大四下学期在余顺争老师指导下完成本科毕业论文的撰写以来，余老师对我释心指导，多次在我感到困惑时为我指点迷津。余老师渊博的学识，严谨的治学态度，让我深为折服，更为我树立了一个科学工作者的榜样。一日为师，终身为师，在日后的工作中，我必定牢记余老师的教导，在工作中秉承实事求是，不断探讨的科学精神。

另外要感谢我的父母和家人对我无微不至的照顾和关怀，感谢他们对我的理解和支持。他们不但助我完成学业，更释心关怀和照料我的身体，使我能够保持健康的体魄，顺利完成硕士学位的攻读。

最后要感谢我的师兄陆伟宙，感谢他在数据处理方面提供的大量的帮助；感谢我的同学王颖熙，赖阳涌，顾俊佳，刘云龙和金志平，感谢他们在实验环境和实验方法上提供的宝贵意见和帮助。

原创性声明

本人郑重声明：

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

学位论文作者签名：杨柳清

日期：2006 年 5 月 30 日