## 内 容 简 介

计算机网络的服务质量(QoS)是当今国际上网络研究领域最重要、最富有魅力的研究方向之一,是计算机网络研究与开发的热点,被称为新一代计算机网络的核心问题之一。本书分为 4 个部分,共 15 章。第一部分是 QoS 的体系结构,包括 QoS 的定义及概述,IntServ 和 DiffServ 两种 Internet QoS 体系结构,及其二者的结合。第二部分是 QoS 的实现机制,包括 ATM 网络的传输管理与 QoS 控制,IP 网络的拥塞控制、报文分类、流量整形与监测、队列管理、分组调度、QoS 路由等控制问题。第三部分是 QoS 的性能评价与应用扩展,包括 QoS 控制的综合性能评价标准,以及应用层的 Web QoS 控制。第四部分是 QoS 的仿真与实现,包括网络仿真软件 NS2 的介绍和基于 NS2 的网络仿真实现方法,以及基于网络处理器平台的 QoS 实现。

本书细致而全面地展示了计算机网络 QoS 领域的研究进展和最新成果,具有完整性、新颖性和学术性。非常适合我国计算机网络与通信领域的教学、科研工作和工程应用参考。既可以供计算机、通信、电子、信息、自动化等相关专业的科研人员、研究生和大学高年级学生作为教学参考书,也可以供计算机网络研究开发人员、网络运营商等网络工程技术人员参考。



图书在版编目(CIP)数据

计算机网络的服务质量(QoS)/林闯,单志广,任丰原著. —北京:清华大学出版社,2004
ISBN 7-302-08076-3

Ⅰ. 计… Ⅱ. ①林… ②单… ③任… Ⅲ. 计算机网络 - 服务质量 - 质量管理 Ⅳ. TP393. 07

中国版本图书馆 CIP 数据核字(2004)第 008216 号

出 版 者:清华大学出版社　　地　　址:北京清华大学学研大厦
　　http://www. tup. com. cn　　邮　　编:100084
　　社 总 机:010-62770175　　客户服务:010-62776969
责任编辑:薛　慧
版式设计:肖　米
印 刷 者:北京市世界知识印刷厂
装 订 者:北京市密云县京文制本装订厂
发 行 者:新华书店总店北京发行所
开　　本:185 ×260　印张:23. 5　字数:529 千字
版　　次:2004 年 4 月第 1 版　2004 年 4 月第 1 次印刷
书　　号:ISBN 7-302-08076-3/TP・5843
印　　数:1 ~4000
定　　价:36. 00 元

# 前　言

QUALITY OF SERVICE OF COMPUTER NETWORK

**谁掌握了信息、谁控制了网络，谁就将拥有整个世界。**

Alvin Toffler

**（阿尔文·托夫勒，美国著名未来学家）**

计算机网络是20世纪60年代人类的伟大创造。从1969年Internet的前身——美国国防部高级研究计划署为冷战目的而研制的ARPANET网络开始投入运行，迄今为止的短短三十几年时间，计算机网络已经对人类社会的进步与发展产生了巨大的推动作用和深远的影响。然而，如何透彻地认识和理解计算机网络这个人工非线性复杂巨系统，如何有效地管理和控制以Internet为代表的大尺度现代计算机网络，如何高效、高质量地传输多媒体业务和满足各种现代网络应用的多元化需求，在理论上和技术上至今依然存在着许多重大的科学问题和挑战。计算机网络的服务质量（quality of service，QoS）问题就是其中之一。

顾名思义，服务质量体现的是消费者对服务者所提供服务的满意程度，是对服务者服务水平的一种度量和评价。同样，现代计算机网络作为计算和信息等服务的提供者，同样面临着提供QoS的问题。事实上，从计算机网络系统诞生伊始，人们就一直孜孜不倦地致力于提高系统的服务性能和服务质量，因此，QoS问题实际上由来已久。目前，在高速网络中按照用户的要求提供QoS控制是一个普遍的要求，也是Internet发展的重要挑战。计算机网络的QoS问题已经成为当今国际网络研究领域最重要、最富有魅力的核心研究领域之一，是目前计算机网络中研究与开发的热点问题，并且和网络安全等问题一起被称为新一代计算机网络最重要的核心研究领域，因此其对未来网络技术的研究、应用和发展具有举足轻重的意义。

然而，由于网络基础设施的庞杂性，实现端到端的QoS并非易事。影响网络QoS的因素有许多，网络QoS控制涉及到构成网络的每一个节点和元素，从网络链路与协议，到路由器、交换机、服务器的硬件、软件体系结构。计算机互联网络发展至今，已成为一个庞大的非线性复杂巨系统：系统的规模和用户数量巨大且仍在不断增长，异质异构的网络不断融合发展；网络协

议体系庞杂,垂直方向上呈现出多样化的层次结构,而水平方向上又以地域和功能为标准进一步形成分布且多级的架构;在业务性质上表现为多种业务的集成与综合,业务量突发性日渐明显,且不同业务要求不同的 QoS 保证;网络节点间、节点与数据分组间由于协议而产生的非线性作用以及用户之间的合作与竞争,使网络行为呈现出相当的复杂性并且难以预测。因此,要真正实现针对各类网络用户提供端到端的 QoS 保证,仍有很长的路要走。

针对网络的 QoS 控制问题,世界各国的大学和研究机构、标准化组织(IETF、ITU-T、ANSI 等)、计算机网络产品的开发商和网络运营商都纷纷投入到这一领域的研究和开发。目前,QoS 控制技术的研究和开发进展得都非常迅速,并且已经取得了许多重要的基本研究成果。国内近些年也开始了有关 QoS 控制方面的研究。

清华大学计算机科学与技术系计算机网络技术研究所林闯教授领导的"计算机网络传输控制与性能评价"研究小组是国内 QoS 研究领域非常活跃的一支研究队伍,近年来在 QoS 控制领域开展了一系列卓有成效的研究工作,本书就是本课题组多年研究工作的全面总结,书中绝大部分内容取材于我们近期已在国际、国内一流学术期刊发表的论文,细致而全面地展示了我们很多最新的研究成果和进展。

本书从组织结构上分为 4 个部分 15 章。

第一部分是 QoS 的体系结构。其中第 1 章的内容主要是 QoS 的定义及概述,给出了 QoS 的一般性描述、各种定义和标准,并对 QoS 的控制和管理机制进行了概述。第 2 章和第 3 章重点介绍了 IETF 提出的两种不同的 Internet QoS 体系结构:综合服务(IntServ)和区分服务(DiffServ),对其各自的原理机制、服务类型、研究热点等进行了详细的描述,并进行了对比性分析。第 4 章讨论了 DiffServ 与 IntServ 相结合的端到端 QoS 提供机制。

第二部分是 QoS 的实现机制。其中第 5 章介绍了 ATM 网络的传输管理与 QoS 控制的技术与策略。第 6 章介绍了 IP 网络的拥塞控制机制以及主动队列管理的策略与算法。第 7 章详细描述了报文分类问题,包括报文分类基础知识、报文分类算法以及报文分类器的设计与应用。第 8 章介绍了流量整形与监测。第 9 章对队列管理进行了详细的讨论,包括缓冲管理的意义、目标、控制策略、典型算法、研究方向等。第 10 章重点介绍了分组调度,包括分组调度算法的本质分析、性能指标以及各种调度算法的比较和分析。第 11 章讨论了 QoS 路由问题,包括 QoS 路由的实现机制、各种路由协议与算法等。

第三部分是 QoS 的性能评价与应用扩展。其中第 12 章讨论了 QoS 控制的综合性能评价标准问题。第 13 章全面描述了应用层的 QoS 问题——Web QoS,介绍了 Web 请求的分类机制、Web 服务器应用软件的 QoS 控制机制、操作系统的 Web QoS 控制机制、中间件的 Web QoS 控制机制、Web 服务器集群的 QoS 控制,以及 Web 服务器集群 QoS-aware 负载均衡的策略、模型与性能分析。

第四部分是 QoS 的仿真与实现。其中第 14 章介绍了网络仿真软件 NS2 以及基于 NS2 的网络仿真实现方法。第 15 章描述了基于网络处理器平台的 QoS 实现,首先对网络处理器进行了综述,然后介绍了基于 Intel 网络处理器的路由器队列管理的 QoS 实现。

清华大学计算机科学与技术系计算机网络技术研究所"计算机网络传输控制与性能评价"研究小组的科研人员和研究生对本书的编写提供了大力的协助。在此特别感谢田

立勤对本书第 7、8 章所做的工作，李寅对第 9 章所做的工作，周文江对第 10 章所做的工作，崔逊学对第 11 章所做的工作，以及谭章熹、郑波等人的工作。

本书具有以下鲜明的特色：

（1）完整性：本书内容全面丰富，体系完整，结构合理，层次分明，细致而全面地描述了计算机网络 QoS 的体系结构、实现机制、性能评价与应用扩展，以及 QoS 的仿真与实现，是全面、深入了解网络 QoS 技术的难得的参考书。

（2）新颖性：本书全面反映了当今计算机网络 QoS 领域的最新研究进展，论述的各项 QoS 技术正是目前网络研究与应用的热点或是将要引起人们关注的技术，内容新颖，别具一格。

（3）学术性：本书具有一定的理论高度和学术价值，书中绝大部分内容取材于作者近期已在国际、国内一流学术期刊发表的论文，细致而全面地展示了大量最新的科学研究成果和发展动向，具有一定的前瞻性和很高的学术参考价值。

本书特别适合我国计算机网络与通信领域的教学、科研工作和工程应用参考。既可以供计算机、通信、电子、信息、自动化等相关专业的科研人员、研究生和大学高年级学生作为教学参考书，也可以供计算机网络研究开发人员、网络运营商等网络工程技术人员参考。

本书作者的研究工作得到了国家重点基础研究发展计划（973 计划）项目（No. G1999032707，2003CB314804）、国家自然科学基金项目（No. 60373013，60218003，60273009 和 90104002）、国家高技术研究发展计划（863 计划）项目（No. 2001AA112080）和高等学校博士学科点专项科研基金项目（No. 20020003027）等的连续资助，在此表示深深的谢意！

由于作者水平所限，加之计算机网络 QoS 问题的研究仍处于不断的发展和变化之中，书中错误和不足之处在所难免，恳请专家、读者指正。

作 者

2003 年 10 月

北京清华园

# 目　录

QUALITY OF SERVICE OF COMPUTER NETWORKS

## 第一部分　QoS 的体系结构

## 第二部分 QoS 的实现机制

## 第三部分 QoS 的性能评价与应用扩展

## 第四部分 QoS 的仿真与实现

# 第一部分

## QoS 的体系结构

QUALITY OF SERVICE OF COMPUTER NETWORKS

# 第1章

# QoS的定义及概述

服务质量是日常生活中人们再熟悉不过的字眼。顾名思义,服务质量往往体现了消费者对服务者所提供服务的满意程度,是对服务者服务水平的一种度量和评价。计算机系统,特别是计算机网络系统,作为计算和信息等服务的提供者,同样存在服务质量(quality of service, QoS)优劣的问题。事实上,从计算机系统诞生伊始,人们就一直孜孜不倦地致力于提高系统的服务性能和服务质量,因此,QoS问题实际上由来已久。目前,计算机网络的QoS问题已经成为国际网络研究领域最重要、最富有魅力的研究领域之一,并且和网络安全等问题一道被称为新一代计算机网络最重要的研究领域之一,对未来网络技术的研究、应用和发展具有举足轻重的意义。

## 1.1 QoS的一般性描述

### 1.1.1 QoS的应用需求

随着高速网络技术和多媒体技术的飞速发展,人们越来越多地提出了包括多媒体通信在内的综合服务要求。传统的分组交换网络,如Internet,是面向非实时的数据通信(如FTP和E-mail的传输)而设计的,采用的TCP/IP协议主要是为了优化整个网络的数据吞吐量并保证数据通信的可靠性。而当今分布式多媒体应用(如视频会议、视频点播、IP可视电话、远程教育)不仅包括文本数据信息,还包括语音、图形、图像、视频、动画这些类型的多媒体信息。分布式多媒体应用不但对网络有很高的带宽要求,而且要求信息传输的低延迟和低抖动等,同时,这些应用大都能够容忍一定程度的信息丢失和错误。

表1.1.1给出了一些典型应用的QoS需求。

由此可见,当今高速网络中的多媒体应用对网络提出了不同于数据应用的服务质量要求,需要提供端到端的QoS控制和保证。

目前,在高速网络中按照用户的要求提供QoS控制是一个普遍的要求,也是Internet发展的重要挑战。多媒体信息传输与管理的QoS控制技术作为下一代网络的核心技术之一,是当前计算机网络中研究与开发的热点问题[1]。

表 1.1.1 一些应用的 QoS 需求

| 应用类型 | QoS 要求参数 | 范 围 |
|---|---|---|
| FTP | 带宽 | 0.2 ~ 10Mbps |
| Telnet | 相应延迟 | ≤800ms |
| 电话 | 带宽 | 16kbps |
| | 端到端延迟 | 0 ~ 150ms |
| | 端到端抖动 | 1ms |
| | 分组丢失率 | $\leq 10^{-2}$ |
| MPEG-1 | 带宽 | ≤1.86Mbps |
| | 端到端延迟 | 250ms |
| | 端到端抖动 | 1ms |
| | 分组丢失率 | $\leq 10^{-2}$(未压缩的视频) |
| | | $\leq 10^{-11}$(压缩视频) |
| HDTV | 带宽 | ≥1Gbps(未压缩) |
| | | ≈500Mbps(无损压缩) |
| | | 20Mbps(有损压缩) |
| | 端到端延迟 | 250ms |
| | 端到端抖动 | 1ms |
| | 分组丢失率 | $\leq 10^{-2}$(未压缩的视频) |
| | | $\leq 10^{-11}$(压缩视频) |

## 1.1.2 QoS 的概念描述

QoS(quality of service),即服务质量。它有多种等价或互补的定义形式。

RFC2386[2] 中描述为:QoS 是网络在传输数据流时要求满足的一系列服务请求,具体可以量化为带宽、延迟、延迟抖动、丢失率、吞吐量等性能指标。此处的服务具体是指数据包(流)经过若干网络节点所接受的传输服务,强调端到端(end-to-end)或网络边界到边界的整体性。QoS 反映了网络元素(例如,应用程序、主机或路由器)在保证信息传输和满足服务要求方面的能力。

另一种描述[3] 为:QoS 是指发送和接收信息的用户之间以及用户与传输信息的综合服务网络之间关于信息传输的质量约定。该约定可以被理解为服务提供者与用户之间的一份服务契约,即服务提供者承担支持给定的服务质量,当且仅当用户按照约定的信息流特征产生数据。换句话说,服务质量包括用户的要求和网络服务提供者的行为两个方面,是用户与服务提供者两方面主客观标准的统一。用户的要求是指用户在 Internet 上进行多媒体通信时所要求的服务类型以及相应的传输性能和质量等;网络服务提供者的行为

则指 Internet 针对某一类服务所能提供和达到的性能与质量。

QoS 控制的目标是为 Internet 应用提供服务区分和性能保证：服务区分是指根据不同应用的需求为其提供不同的服务；性能保证则要解决诸如带宽、丢失、延迟、延迟抖动等性能指标的保证问题。然而，在网络中，特别是在 Internet 这样大规模的全球网络中提供 QoS 绝非易事，它需要自顶向下所有网络层（即 ISO-OSI 模型中的第 1 层 ~ 第 7 层）以及端到端（即从信息的发送者到接收者）的所有网络元素的整体协作。

QoS 与用户以及网络系统的关系如图 1.1.1 所示。

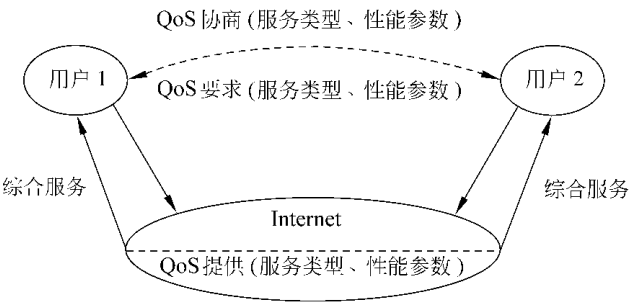



图 1.1.1　QoS 与用户以及网络系统的关系

从图 1.1.1 中可以看出，QoS 不是网络中某个个体或元素的行为描述，它涉及到用户与用户、用户与网络以及网络内部节点（或元素）的整体行为。例如，图 1.1.1 中当用户 1 与用户 2 之间要相互通信时，事先必须相互协商通信时的服务类型以及相应的性能参数。如果不事先进行协商，若用户 2 接收来自于用户 1 的实时图像信息时，用户 1 每秒按 30 帧发送，而用户 2 只有每秒接收 20 帧的能力的话，则尽管 Internet 提供了每秒 30 帧的传输服务，信息仍会由于用户 2 的接收能力不够而丢失，从而也无法进行满意的实时通信。相反地，如果事先经过协商，用户 1 放慢速度，使其满足用户 2 的接收能力，则用户 2 会获得较好的 QoS，而且网络的负载也会相应减轻。

除了用户与用户之间的协商之外，用户与网络、网络中的各个元素之间也存在着 QoS 协商和管理的问题。当用户的 QoS 要求太高、网络无法提供相应的综合服务时，将要求用户降低其 QoS 要求，甚至为了保证其他用户的 QoS 而拒绝某用户的 QoS 要求。用户与网络系统之间的 QoS 协商通常称为接纳控制（admission control）。

在网络内部的路由器、交换机的端口以及端主机系统中，为了保证用户要求的 QoS，必须进行资源预留，并采用相应的资源调度算法，这就需要相应的资源预留协议和资源调度算法。除了上述机制以外，将 QoS 控制过程引入 Internet、参数化的 QoS 描述以及 QoS 维护、QoS 降级等控制管理机制也是必不可少的。

### 1.1.3　QoS 的发展概述

自从计算机系统诞生开始，就一直存在提高系统的服务性能和服务质量的问题，因此，可以说计算机系统的 QoS 问题由来已久。

对计算机网络 QoS 的研究可以追溯到 20 世纪 80 年代初期。那时，尽管网络的性能

还比较低,提供的服务种类也比较少,但一些有远见的研究者已经认识到服务质量的重要性。Seitz 和 Wortendyke 等人在研究 ARPANET 中的 X.25 通信时已提出基于用户的性能评价问题,这也许是关于计算机网络 QoS 研究的最早文献[4]。在早期的 OSI 协议制定中,也为服务质量的一些参数留有相应的表示手段,但一直空缺未用[5]。很长的一段时间,由于计算机网络的性能所限,人们对 QoS 的关注只停留在数据流传输中的正确率、吞吐量和延迟等单一服务质量的评价与控制上[6]。直到 20 世纪 80 年代末期,随着 B-ISDN 技术以及 ATM 交换网的出现和分布式多媒体应用的急剧增加,人们才开始系统地对 QoS 管理和控制进行较为深入的研究。一些实验性系统也应运而生,代表性的有英国兰开斯特大学的 QoS-A 工程[7]、美国哥伦比亚大学的扩展的集成化参考模型(XRM)系统[8]、国际合作项目 TINA-C 工程[9]、美国加州伯克利大学的 TENET 工程[10]、IBM 公司黑森伯格欧洲网络中心的 HeiProject 工程[11],等等。

随着 Internet 商业化的巨大成功,网上传输的多媒体信息迅速增多,网络拥塞现象日益严重,Internet 的 QoS 问题研究也随之开始深入。IETF 于 1997 年 9 月开始制定了有关 QoS 定义与服务的一系列 RFC 标准,典型的工作是提出了两种不同的 Internet QoS 体系结构:综合服务(integrated services, IntServ)[12] 和区分服务(differentiated services, DiffServ)[13]。截至目前,QoS 控制技术的研究和开发都进展得非常迅速,并且已经取得了许多基本的成果。国内也于近些年开始了有关 QoS 控制方面的研究。

目前,计算机网络的 QoS 问题已经成为国际网络研究领域公认的最重要、最富有魅力的研究领域之一,并且被称为下一代计算机网络为数不多的最重要的研究领域之一。

## 1.2 QoS 的定义和标准

### 1.2.1 OSI 参考模型中的 QoS 定义

ISO 最早开始计算机网络 QoS 问题的研究。针对 OSI 参考模型的七层协议,ISO 组织要求每层协议都在向高层提供相应服务的同时,提供如表 1.2.1 所示的服务质量[5]。

表 1.2.1 OSI 参考模型中的 QoS 定义

| 参 数 | 含 义 |
|---|---|
| 吞吐量 | 单位时间内在一个连接上传递的最大字节数 |
| 传输延迟 | 从数据传输请求开始到数据传输完成确认为止的时间间隔 |
| 出错率 | 数据单元错传、丢失或重传的概率 |
| 建立连接延迟 | 从请求建立连接开始到建立连接确认为止的时间间隔 |
| 连接失败率 | 建立连接失败的概率 |
| 传输失败率 | 传输失败的概率 |
| 重置率 | 在给定的时间内服务提供者释放连接或重置连接的概率 |
| 释放延迟 | 从释放请求开始到释放确认为止的时间延迟 |
| 释放失败概率 | 释放连接时失败的概率 |

如表 1.2.1 所示,OSI 参考模型中的服务质量(QoS)用参数方式进行定义。即当高层

协议要求低层协议提供相应的服务质量时,高层协议向低层协议发送包含QoS参数值的服务数据单元(分组),低层协议按高层协议的QoS要求进行操作,如图1.2.1所示。

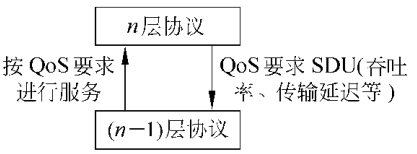



图1.2.1 OSI参考模型的参数传递

另外,ISO还定义了与协议功能本身无关的安全性、成本与传输优先级等QoS参数,见表1.2.2。

表1.2.2 OSI中与协议功能无关的QoS参数

| 参 数 | 含 义 |
|---|---|
| 访问权限 | 防止非法用户访问 |
| 优先级 | 包括传输优先级和使用优先级 |
| 成本 | 信息传输时所消耗的资源或资金 |

## 1.2.2 CCITT(ITU)的QoS定义

OSI参考模型中的QoS参数定义主要是针对数据传输的。这些定义既未考虑服务类型,也未考虑出错的概率分布以及传输峰值的变化等因素。而且,OSI参考模型的QoS定义未给出如何实行QoS要求的方法与框架。1990年,当时的国际电话电报咨询委员会CCITT(Consultative Committee on International Telephone and Telegraph)又针对QoS制订了CCITT-I系列建议。

该建议从呼叫控制、连接以及数据单元的传输控制等三个不同的层次定义了宽带ISDN的QoS[14]。

呼叫控制级的QoS包括呼叫次数、失败率等。

连接级的QoS包括连接延迟、连接失败率、释放的延迟和释放失败率等参数的定义。

数据单元传输控制级的QoS定义包括分组的峰值到达率、峰值持续时间、分组平均到达率、分组丢失率、分组插入率以及比特出错率等。

不过,这些QoS定义仍然未提供实现QoS控制和用户QoS要求的机制,也未对用户要求的服务进行分类。这些QoS定义并没有站在用户的立场,而是站在服务提供者,即网络的立场上来定义的。

## 1.2.3 ATM的QoS定义

ATM论坛不仅把服务质量QoS的概念引入到了ATM交换机中,而且对用户的应用和ATM所对应的服务进行了分类。

ATM精确地定义了QoS的概念,但是方式十分复杂。ATM论坛把ATM网络的服务定义为如下5类:

(1) CBR(constant bit rate),恒定位速率服务;

(2) rt-VBR(real-time variable bit rate),实时可变位速率服务;

(3) nrt-VBR(nonreal-time variable bit rate),非实时可变位速率服务;

(4) ABR(available bit rate),可用位速率服务;

(5) UBR(unspecified bit rate),未指定位速率服务。

表 1.2.3 是 ITU 制订的一些 ATM 网络的 QoS 参数。

表 1.2.3 ATM 网络的 QoS 参数定义

| 参 数 | 含 义 |
|---|---|
| 峰值信元速率(PCR) | 用户发送信元的最大瞬间速率 |
| 持续信元速率(SCR) | 经过一个长时间测量得到的平均速率 |
| 最小信元速率(MCR) | 用户希望至少达到的最小速率 |
| 信元丢失率(CLR) | 因为错误和拥塞信元不能到达目的地而导致在网络中丢失的信元所占的百分率 |
| 信元传输延迟(CTD) | 一个信元从进入网络到离去所经历的延迟 |
| 信元延迟方差(CDV) | CTD 的方差 |
| 突发容许(BT) | 决定可以按照峰值速率发出的最大突发长度 |

根据 ATM 论坛的 4.0 版传输规范,表 1.2.4 归纳了上述 ATM 层服务类的属性。

表 1.2.4 ATM 论坛服务类的属性、QoS 保证及反馈的使用

| 服务类 | 传输描述符 | 是否提供以下 QoS 保证 | | | |
|---|---|---|---|---|---|
| | | 丢失(CLR) | 延迟方差(CDV) | 预留资源 | 反馈控制 |
| CBR | 峰值信元速率 PCR | 是 | 是 | 是 | 否 |
| rt-VBR | 峰值信元速率 PCR,持续信元速率 SCR,最大突发尺寸 MBS | 是 | 是 | 是 | 否 |
| nrt-VBR | 峰值信元速率 PCR,持续信元速率 SCR,最大突发尺寸 MBS | 是 | 否 | 是 | 否 |
| ABR | 峰值信元速率 PCR,最小信元速率 MCR | 是 | 否 | 是 | 是 |
| UBR | 峰值信元速率 PCR | 否 | 否 | 否 | 否 |

表 1.2.5 归纳了上述 ATM 层服务类及其对各种应用的适用情况。

这些服务类型的具体传输特点将在 5.1 节中给出。

ATM 论坛也定义了相应的呼叫接纳控制 CAC(call admission control),用来检查用户连接请求的服务类型,并根据 ATM 交换机中的资源空闲状况来决定接受或拒绝用户的连接请求。

由于 ATM 用面向连接的方式交换信元,因此,一旦一个虚连接 VC(virtual connection)被建立起来以后,在数据传输完成之前将不会被切断。从而,ATM用户网络

表1.2.5 ATM论坛服务类对各种应用的适用情况

| 应 用 | CBR | rt-VBR | nrt-VBR | ABR | UBR |
|---|---|---|---|---|---|
| 关键数据 | 较好 | 一般 | 最好 | 一般 | 不可 |
| LAN互联 | 一般 | 一般 | 较好 | 最好 | 较好 |
| WAN数据传输 | 一般 | 一般 | 较好 | 最好 | 较好 |
| 电路仿真 | 最好 | 较好 | 不可 | 不可 | 不可 |
| 电话 | 最好 | 较好 | 不可 | 不可 | 不可 |
| 视频会议 | 最好 | 较好 | 一般 | 一般 | 较差 |
| 压缩的音频 | 一般 | 最好 | 较好 | 较好 | 较差 |
| 视频发布 | 最好 | 较好 | 一般 | 不可 | 不可 |
| 交互式多媒体 | 最好 | 最好 | 较好 | 较好 | 较差 |

接口UNI(user network interface)方面的QoS应用主要以服务为主。ATM交换机则把UNI提交过来的服务类型自动匹配到相应的QoS参数上去。当然,这些参数包括优先级、交换速率、延迟时间等。ATM交换机把不同服务类型的信元放入不同优先级的队列,并为它们分配不同数量的缓冲区。

### 1.2.4 IETF的QoS定义

如何在Internet上提供综合服务的关键是QoS控制。早期在ATM交换机及其协议中开展的QoS支持工作为后来IP网络中的QoS问题研究奠定了基础。进一步地,对于Internet,目前人们正在研究新的方法来实现QoS控制和传输管理,以迎接由于Internet规模日益扩展和业务流爆炸式增长而带来的挑战。

根据Internet提供综合服务的要求,Internet工程特别工作组IETF(Internet Engineering Task Force)把QoS控制问题划分为两大部分,即综合服务模型与QoS实现框架。自RFC1633[15]提出了Internet综合服务的概念以后,RFC2211[16]定义了Internet的可控负载型服务(controlled-load service),RFC2212[17]定义了Internet的质量保证型服务(guaranteed service),RFC2215[18]则定义了Internet的综合服务(integrated services, IntServ),也就是QoS控制用的通用特性参数,RFC2216[19]定义了QoS服务规范,而RFC2205[20]则给出了实现QoS的最关键部分——资源预留协议RSVP(resource reservation protocol)。

为了解决IntServ的可扩展性不好、实现难度大等问题,IETF又提出了一种可扩展性好的区分服务(differentiated services, DiffServ)体系结构[21, 22]。定义了每个IP包头的区分服务标记域[23],称为DS标记(differentiated services codepoint, DSCP);定义了奖赏服务(premium service,PS)[21]和确保服务(assured service,AS)[24]两种典型的区分服务技术。

IntServ和DiffServ的详细讨论将在第2章和第3章中分别给出。

下面仅以RFC2216中的QoS定义为例,对Internet的QoS定义进行介绍。

在RFC2216中,QoS定义为:用带宽、分组延迟和分组丢失率等参数描述的关于分组传输的质量。传统的Internet只提供单一的服务质量,即"尽力而为"(best-effort)服务。在该服务中,可利用的带宽以及相应的延迟特性取决于网络中的负载状况。

为了进一步描述 QoS 控制过程和服务模型与实现框架,RFC2216 还定义了网络元素(network element)、流(flow)、服务(service)、行为(behavior)、特性化(characterization)以及相应的流量规范(traffic specification, TSpec)、服务要求规范(request specification, RSpec)以及 QoS 控制有关的其他词语定义。

IETF 规定,“网络元素”是指任何一个可在 Internet 网络中处理数据分组的构件,它具有在数据通过时进行 QoS 控制的能力。网络元素包括路由器、子网、端主机系统的操作系统等。“流”则指具有相同 QoS 要求和服从同一 QoS 控制方法的通过某个网络元素的分组集合。在一个给定的网络元素中,一个流的分组可以来自于某个单一的应用,也可以来自于不同的应用。

“服务”与 QoS 控制服务具有相同的意义。它描述网络元素的 QoS 控制能力。服务包括规范和功能两大部分。“行为”是指与 QoS 相关的端到端性能。行为是应用直接可见的由服务提供的最终结果。

TSpec 是要求服务提供的流量描述。TSpec 实际上是一份数据流和网络元素提供的服务之间的合同。RSpec 则是用户对网络元素的 QoS 要求。TSpec 和 RSpec 都被资源预留协议 RSVP 规定了相应的格式和定义[25]。

基于上述定义,IETF 把 QoS 定义为一个两维空间:

〈服务类型〉、〈参数类型〉

服务类型与参数类型两者都用整数表示。

〈服务类型〉的取值范围为[1, 254]。该取值范围被进一步划分为三个区。即:1,[2, 127],[128, 254]。

服务类型 1 被保留下来用于指定通用参数。即当服务类型值为 1 时,参数类型中给出的任何参数都可以被所有服务所使用。

服务类型[2, 127]表示 IETF 定义的各种服务。IETF 的 IntServ 工作小组负责定义各种服务的编号。例如,保证型服务的编号为 2,可控负载型服务的编号为 5。目前,IETF 还未定义更多的服务类型。如果研究人员开发和定义了新的服务类型,并准备提交给公众使用的话,应从 IETF 的 IntServ 工作小组获得相应的服务编号。

服务类型[128, 254]是专为服务与实验开发保留的。研究人员在实验阶段可任意从该区间选取服务号在本地使用。服务号中未使用编号 0,这是因为这些编号将被用做直接访问 MIB 对象库的指针。

与〈服务类型〉相同,〈参数类型〉的取值范围也是[1, 254]。

区间[1, 127]是保留区间,专门用于指定那些供所有服务公用和共享的参数,例如,当前可利用的带宽等。该区间的参数值与服务类型值 1 一起组成公用共享参数,例如[1, 5]表示一个可供各种服务共享的 QoS 参数。

区间[128, 254]由服务规范的设计人员给定,它们不是共享的,只针对相应的服务类型。

### 1.2.5 QoS 定义的分层、分类及分维

由于综合服务网络系统是一个层次化的体系结构,因此每层所考虑的 QoS 的含义是不同的。对于每层来说,QoS 的含义通常包括该层次的 QoS 表示、特定的服务承诺和成本

以及该层对等实体间的协商协议。图 1.2.2 显示了一个典型系统的 QoS 分层情况[26]。

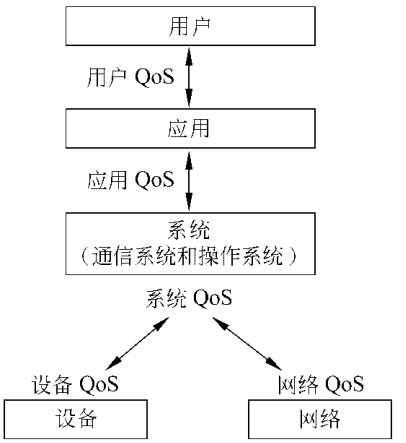



图 1.2.2　QoS 的分层

在图 1.2.2 中，用户关心的是可感知的 QoS，即对服务效果的主观感觉的好坏程度；应用 QoS 反映的是应用所期望的底层支持系统的性能；而更底层的 QoS 则反映了特定的多媒体设备和网络的性能。

前面描述的几种 QoS 的定义由于没有很好地考虑应用和网络的分层以及多媒体 QoS 要求的多样性，因此不能很好地描述多媒体应用的 QoS 请求。例如，OSI 虽然按照层次描述 QoS，但由于制定各层标准的工作小组是单独工作的，致使各层的 QoS 之间的关系没有明确的定义，因而是不一致的[5]；CCITT 的 QoS 局限在传输层服务上，所有的用户服务请求都只在传输层进行映射，但是传输层以下的各个层次无法提供相应的支持[14]；IETF 的 QoS 定义针对的都是网络传输延迟，未考虑抖动、同步、声音/图像的失真率等因素，同时，尽管定义了 flowspec 等数据流的 QoS 参数，但用户如何表示自己的 QoS 需求，仍是一个十分困难的问题。特别是很难量化用户对画面和音质等感觉上的质量要求，而且，IETF 的 QoS 定义也没有对综合服务以及 QoS 控制的层次进行适当的定义和划分。

针对上述问题，国际上许多研究小组（包括 IETF 的工作小组）都进行了对应的研究。英国 Lancaster 大学的 QoS-A 工程就提出了一个按系统处理方式对 QoS 进行分层，按用户需求和应用进行分维，并将不同维中具有某些共同属性的 QoS 组成类的新方法[27]。QoS-A 的层次结构如图 1.2.3 所示。

在图 1.2.3 中，QoS 被划分为 4 个不同层次，即应用平台和操作系统层、协商调整层、传输系统层与 ATM 网络层。与各层相对应，用户要求的 QoS 被设置成了不同的维，例如

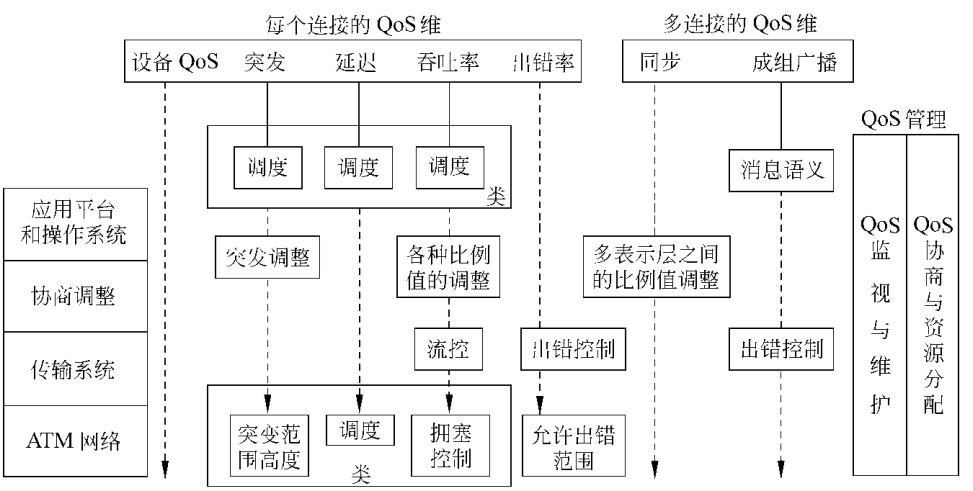



图 1.2.3　QoS-A 中定义的 QoS 及层次

突发、延迟、吞吐量、出错率等。图 1.2.3 只给出了较少的的用户 QoS 要求。这些 QoS 要求在不同的层中映射成不同的操作。例如,为了获得用户要求的吞吐量,在操作系统和应用平台层应选择合适的调度策略和方法,以保证用户应用的 QoS 要求。而在操作系统与传输系统之间,系统还必须调整传输速率等比例值,以便在用户应用得到 CPU 等资源处理之后,按所需要求的速率传输转发。流量控制则是缓冲区资源和传输速率等之间的平衡。

## 1.3 QoS 控制和管理概述

20 世纪 80 年代末至今的 QoS 研究工作焦点之一是 QoS 的控制和管理问题。所谓 QoS 控制和管理,是指计算机网络系统采用一定的方法接受用户应用的服务请求,并保证其 QoS 的过程。在这一过程中,网络系统将用户应用的服务请求映射成一些预先定义的 QoS 参数,进而与系统的有效资源对应起来,通过资源分配和调度,完成用户的应用。这一过程也称为 QoS 协商。QoS 协商有时可能不成功,其原因是系统无法完全满足应用的网络带宽、延迟和信息正确率等要求,这时通常就要进行 QoS 重协商,以确定应用是否容忍降级的服务。

为了实现 QoS 的控制和管理,首先必须明确在设计 QoS 控制和管理机制时应遵循的基本原则,其次要对 QoS 作准确而细致的描述,接着针对端到端 QoS 的控制过程制定出相应的 QoS 控制和管理机制[28],最后还要了解综合服务网络系统不同层次的 QoS 实现是不同的。

### 1.3.1 QoS 设计的基本原则

将 QoS 控制过程引入网络系统,必须考虑以下 5 个基本原则:

(1) 集成的原则

该原则阐述了为了满足用户的端到端 QoS 控制要求,在所有的层次系统上 QoS 都必须是可配置、可预测和可维护的[29]。当多媒体信息流从源媒体设备产生,向下经过源协议栈,穿过网络,再向上经过接收端协议栈进入媒体播放设备时,所穿越的系统各个层次的资源模块(如 CPU、内存、设备和网络等)都必须提供 QoS 的可配置性(基于 QoS 描述)、资源担保(由 QoS 控制机制提供)以及流的 QoS 维护(由 QoS 的管理机制实现)。

(2) 分离原则

该原则阐述了媒体的传输、控制和管理是系统的不同功能行为。这些任务的分离一方面是要区分控制信令和媒体数据的传输,另一方面是由于两者所要求的服务不同。媒体数据流通常要求高带宽、低延迟、低抖动并容许一定丢失率的服务,控制信令的传输通常要求低带宽、无抖动限制和高可靠性的服务。

(3) 透明原则

该原则阐述了应用必须被屏蔽在底层复杂的 QoS 控制和管理机制之外。透明性的一个重要方面是用户期望得到的 QoS 能够通过一个基于 QoS 的 API 来描述。透明性的好处表现在三个方面:可以将 QoS 功能与多媒体应用设计相分离,从而简化应用设计并

增强可移植性；可以向应用屏蔽底层网络的服务细节；可以将复杂的 QoS 控制和管理任务交给底层的网络实现。

（4）异步资源管理原则

由于在分布式通信环境中不同行为（如调度、流控、路由以及 QoS 管理等）的产生在时间上是不同的，因此对综合服务网络系统资源的管理是异步的。为了协调这些被异步管理的资源，从而为用户应用提供一致的端到端 QoS 控制，必须在这些资源之间周期性地交换控制信息。

（5）性能原则

该原则阐述了 QoS 控制和管理机制的设计必须是以提高网络系统的性能为目的。换句话说，综合服务网络既要能担保多媒体应用的 QoS，又能灵活地支持不同的用户应用，具有非常高的网络利用率。

### 1.3.2　QoS 的描述

QoS 描述是为了精确地表达用户应用的 QoS 要求和管理策略。系统不同层次的 QoS 描述通常是不同的，分别用于配置和维护该层次的 QoS 控制和管理机制。例如，应用层的 QoS 描述主要是面向用户而不是面向系统的，而较低层次的相关考虑，如线程调度的细节，应该被应用层的 QoS 描述隐藏。从用户应用的角度来看，QoS 描述应该包含以下几个方面：

（1）信息流特征

它描述了用户应用的信息流量特征[30]，如信息流产生的峰值速率和平均速率、信息流的突发长度以及平均周期等。信息流特征描述是综合服务网络系统为用户应用分配和管理资源的依据，也是用户应用向网络系统所作的承诺。

（2）信息流性能要求

它描述了多媒体应用对网络性能的要求[30]，如网络系统的吞吐量、信息的延迟、抖动以及丢失率等。不同多媒体应用对网络性能的要求是不同的，为了进行相应的端系统和网络资源分配，综合服务网络必须提前了解应用对网络性能的要求。

（3）信息流同步要求

它描述了应用多个相关信息流之间的同步程度[31]。例如，同时记录的视频图像必须被精确地一帧帧同步播放，以便有关的特征能够被同时观察到。当然，在播放多媒体视频时，如果主要关注声音信息，而视频图像只是为了增强表示效果，那么声音与图像流之间的唇同步也就不需要绝对精确了。

（4）服务层次（level of service，LoS）

它描述了端到端 QoS 担保的程度，如可控负载型服务[16]、保证型服务[17]以及尽力而为型服务。尽管信息流性能要求的描述允许用户以量化的方式表达所要求的性能尺度，然而服务层次描述能够对这些要求做质的提炼，以便区分出哪些要求更强烈，哪些要求不那么强烈，从而使系统能够更细致地分配和调度资源，使综合服务网络的总体用户满意度最佳。服务层次描述也有助于系统在资源紧张或不足时采用更灵活而有效的 QoS 控制策略。

(5) QoS 管理策略

它描述了当系统承诺的应用 QoS 不能达到时,应用能够忍受的 QoS 降级的程度以及可被采用的管理行为[32]。例如,当网络系统因为某些原因无法满足多媒体信息的延迟要求时,通过对可得到的带宽做时间和空间质量的折中,或者操纵连续媒体的播放时间,最大可能地减少音频和视频流在播放设备上产生的失真,使得感觉仍可接受。QoS 管理策略也包括用户层的 QoS 降级指示以及周期性的带宽、延迟、抖动以及丢失率的通知。

(6) 服务成本(cost of service,CoS)

它描述了用户对所要求的服务愿意付出的成本。服务成本是 QoS 描述中非常重要的一个方面,设想一下,如果在 QoS 描述中不包含服务成本的描述,那么用户将没有任何理由不去选择最高层次的服务。

## 1.3.3 QoS 的控制和管理机制

QoS 的控制和管理机制(也称为算法),是由用户的 QoS 描述、资源能力以及资源管理策略所驱动的。在对资源的管理上,QoS 机制可以分为静态和动态两种 静态的资源管理是处理信息流的建立和端到端 QoS 的重协商阶段(也可称为 QoS 的提供),动态的资源管理是处理媒体传输阶段(也可称为 QoS 的控制和管理)。前者通过对系统资源的协商和预留,使系统接受用户应用的 QoS 请求,而后者是在系统接受应用的 QoS 请求之后,通过资源调度和流控等机制来保证和维护用户应用所获得的 QoS。

### 1.3.3.1 QoS 的提供机制

为了使用户应用与综合服务网络系统之间达成 QoS 约定,必须提供以下几种机制。

1. QoS 协商和资源准许测试机制

QoS 协商是指用户与系统以及用户与用户(端到端)之间就所传输信息的服务质量进行交互,最后根据应用和系统资源确定系统和用户的 QoS 的过程。

用户和系统之间的协商过程如下:首先,协商程序分析用户的行为,将其分解为一些元素,使得每一个元素都可由一个或多个独立的 QoS 参数表示。用户行为分解的基本准则是这些参数的组合应该保证用户行为的 QoS。在对用户行为分析结束之后,系统把所得到的参数组映射到相邻的低层,并确定相邻的低层是否能够支持这些 QoS 要求。在此基础上,系统将为用户的行为元素定出各自的 QoS 级别,或者向用户报告拒绝服务的情况和原因[29]。

端到端的 QoS 协商分为前向和后向两个阶段。如图 1.3.1 所示,前向阶段由一端的用户发起,将该端用户应用所需要的 QoS 逐级向下层映射并进行比较,如果得到允许,就通过网络将 QoS 要求发送给网络另一端的用户。在网络另一端,用户的 QoS 需求逆向映射到用户应用层并显示给另一端用户。在网络该端的用户得到另一端用户的 QoS 需求之后,将根据自己的要求和处理能力修改得到的 QoS 参数,并逐层向下协商后通过网络传回发送端。发送端将根据所得到的 QoS 参数和协商情况决定 QoS 参数的级别[3]。

在 QoS 协商的过程中,对每一个用户应用的每一个新的行为,系统都要把其所对应

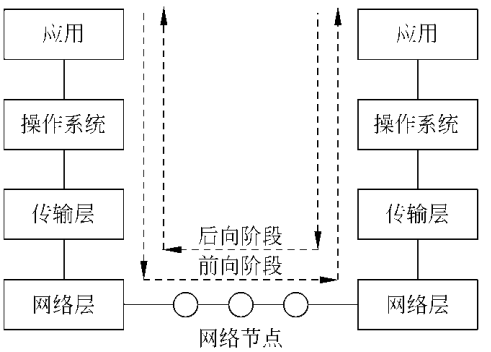



图 1.3.1　端到端 QoS 协商

的 QoS 请求与系统当前可用的资源进行比较，并从所用资源最少和获得用户满意的 QoS 的角度出发，确定该行为是否能够被实施。这种过程和判断策略就是 QoS 接纳控制的主要内容[29]。

当前，有关 QoS 协商协议和接纳控制的研究主要集中在 QoS 协商协议、协商模型以及 QoS 比较和判断策略等方面。

2．QoS 映射机制

由于综合服务网络系统的不同层次所要求的 QoS 参数不同，为了完成用户与系统以及用户与用户之间的 QoS 协商，必须在不同系统层次的 QoS 表示之间进行功能上的翻译。例如，多媒体信息在网络传输中期望的丢失率是非常重要的一项传输层 QoS 参数，然而对应于视频捕捉设备来说，它却是毫无意义的；同样，以单位面积内像素的个数来表示的视频帧质量是用于设定帧捕捉缓存大小的非常重要的一个 QoS 指标，但是它对于底层的网络传输来说也是毫无用处的。

QoS 映射是指系统自动将用户的高层次 QoS 请求解释成低层次的 QoS 参数，以使其不必关心该 QoS 请求在较低层次是如何通过各种复杂的方式来表示的[3]。以图 1.2.2 为例，需要完成的 QoS 映射主要有以下几种：

（1）用户 QoS 与应用 QoS 之间的映射：多媒体应用应根据综合服务网络系统所提供的 QoS 编程接口和 QoS 参数模板库，设计和实现与特定应用 QoS 相关的图形用户界面，该界面应该是用户友好和易于用户理解的。

（2）应用 QoS 与系统 QoS 之间的映射：它将把特定应用的 QoS 翻译为通用操作系统所提供的 QoS，如线程调度与同步、内存管理等。

（3）系统 QoS 与网络/设备 QoS 之间的映射：它将把操作系统的 QoS 翻译为底层传输网络或多媒体设备的 QoS，如把传输层分组的端到端延迟映射为 ATM 网络的信元传输延迟。

应当注意的是，系统不同层次间的映射需要双向进行，但并不对称。如视频帧率和帧大小一起决定了网络的吞吐量大小，但一个新的吞吐量的值却有可能翻译为或者降低图像质量或者减少视频帧率。为了实现低层到高层的映射，必须增加辅助的规则，这使得

QoS 控制策略具有极大的灵活性。

3. 资源的预留及分配机制

为了担保多媒体应用的 QoS,综合服务网络必须根据应用的 QoS 请求预留和分配资源。资源分配的方式依系统对应用资源使用情况的估计可以分为悲观方式和乐观方式两种。前者为了避免资源使用冲突,根据应用最大的资源要求预留资源,例如网络系统根据应用的峰值速率来预留带宽,这种方式尽管能够最大程度地保证应用的 QoS,但却大大降低了网络的利用率。后者根据应用的平均负载预留资源,例如网络系统根据应用的平均速率预留带宽,这种方式可能会使应用的 QoS 由于短时间内的资源使用过载而无法保证,但却具有很高的网络利用率。最佳的资源分配方式介于两者之间,在保证多媒体应用 QoS 的前提下,最大可能地提高网络的利用率。

综合服务网络系统的资源预留和分配是由端主机和路由器中的资源管理器协同完成的。资源管理器主要由以下三部分组成:

(1) 资源表:它包含被管理资源的信息,包括静态资源信息和动态资源信息。静态资源信息有总的资源能力、最大允许的消息长度和所使用的调度算法等 动态资源信息有当前正使用资源的连接指针以及当前预留的资源总量等。

(2) 预留表:它提供预留资源的应用连接的信息,包括这些连接的 QoS 要求以及系统为之分配的资源量等。

(3) 预留函数:它为应用的 QoS 请求计算并预留所需要的资源。

由于多媒体应用要求端到端的 QoS 保证,因此不同网络节点中的资源管理器的资源预留和分配行为并不是孤立的,必须有一种通信机制将它们联系在一起。资源预留协议的设计就是为了满足这种要求。它并不预留或分配网络节点上的资源,只是作为资源管理器之间交换资源信息和协商 QoS 的通信工具,完成相应控制分组的传递。IETF 的 RSVP 协议[20] 以及 TELNET 的 RCAP 协议都属于此类协议。

4. QoS 的重协商机制

QoS 重协商是对已确定的 QoS 级别进行再调整的响应过程。当系统在最上层的用户应用接口处检测到下层已无法保证用户的 QoS 要求时,向用户报告需要进行 QoS 降级的消息。用户将根据系统报告决定是进行 QoS 重协商还是终止用户行为。

一般来说,QoS 重协商不是对某一个 QoS 元素的局部调整,而是指整个行为所有 QoS 元素由用户应用级开始往下全部重新进行 QoS 协商的过程[3]。

### 1.3.3.2 QoS 的控制机制

当用户应用与网络系统达成 QoS 约定之后,网络系统就必须提供基于 QoS 的信息流实时拥塞控制。这种控制是通过 QoS 控制机制来实现的。基本的 QoS 控制机制如下所述。

1. 信息流整型机制

它是根据用户所提供的信息流特征描述来调节信息流量。信息流整型可基于一个简

单的固定分组速率(如分组峰值速率)或某种形式的统计分组速率(如可承受分组速率和分组突发)来进行。信息流整型的好处是使网络系统可以为该应用分配足够的端对端资源并且能够适当地配置流调度程序。在数学上已经证明,如果在网络边缘进行信息流整型,同时在网络中采取适当的调度策略,可以为应用信息流提供非常高的网络性能保证[33]。

2. 信息流调度机制

为了保证应用的端到端QoS,网络系统必须对每个网络元素(包括端系统和网络节点)中等待处理的数据分组进行排队,根据相应的QoS要求和级别赋予相应的优先级而后调度网络资源(如带宽、缓冲、CPU、线程等)执行。而且,在等待队列超过一定的长度时,还需要按照一定的策略放弃一部分等待的分组。网络系统的这种控制行为就是信息流调度机制。当前,几种比较常用的分组排队和调度算法有先入先出排序法、优先级排序法、基于等级的排序法以及加权公平排序法等。另外,还有避免拥塞发生的随机早期探测法等。

3. 信息流监控机制

信息流监控机制可以被认为是下面1.3.3.3节所说的QoS监控机制的对称行为:后者通常与QoS管理有关,它监视网络系统是否提供了所承诺的QoS;而前者监视用户的行为是否符合QoS要求。通常该行为只发生在管理与收费范围出现交界的地方,例如,在一个用户与网络的接口处。一个好的信息流整型策略允许监控机制轻易地发现不规则的信息流。当发现有不规范的信息流时,监控机制可以简单地接受它,同时提醒用户,也可以将信息流整型到QoS可接受的级别。

4. 信息流控制机制

信息流控制机制包括开环和闭环两种模式:开环式流控广泛应用于电话业务,只要资源已被分配,就允许发送者向网络中按一致认可的速率发送数据;而闭环流控需要发送者根据接收者或网络的反馈信息调整发送速率。使用闭环流控机制的应用必须能够适应网络资源的波动。目前的许多多媒体应用具有这种适应能力并且可以在这种环境中工作。而不具有这种使用能力的其他多媒体应用更适合采用开环流控策略,即提前预留必要的资源,使得它们的带宽、延迟和丢失率的要求在会话存在期间都能得到保证。

5. 信息流同步机制

该机制用于控制事件顺序和多媒体交互行为的精确定时。唇同步是最常见的多媒体同步形式(多媒体播放设备上视频和音频的同步);其他同步形式包括:有用户参与交互和没有用户参与交互的事件同步,有别于唇同步的连续同步以及不同的源和接收器之间的连续同步。所有这些都对流同步协议提出了基本的QoS需求。与流同步有关的动态QoS管理主要考虑流之间同步的紧密程度。

#### 1.3.3.3 QoS 的管理机制

为了保证应用的端到端 QoS,网络系统只分配资源通常是不够的,还需要经常维护已担保的应用 QoS。这是由 QoS 管理机制来完成的。QoS 管理机制包括以下几个主要方面。

1. QoS 监控机制

QoS 监控机制允许系统的每一层跟踪在低层所获得的 QoS 级。它在维护应用 QoS 的管理行为中起核心作用。监控算法既可以作为调度程序的一部分(例如一种 QoS 控制机制)来测量正被处理的单个信息流的性能以便更有效地控制分组调度和接纳控制,也可以作为传输层反馈机制的一部分。

2. QoS 维护机制

QoS 维护机制将被监控的 QoS 与期望的性能作比较,然后调整资源的使用策略以便维护应用的 QoS。如果当前的网络资源确实无法满足应用的 QoS 要求,那么 QoS 的维护将引发 QoS 的降级。

3. QoS 降级机制

当网络系统低层无法保证应用信息流的 QoS 并且 QoS 维护机制也无能为力时,高层的 QoS 降级机制将向用户发出一个 QoS 指示。用户可以选择或者适应可获得的 QoS 级或者降低服务级(如端到端重协商)来响应该指示。

4. QoS 扩展机制

QoS 扩展机制包括 QoS 过滤机制(当信息流通过通信系统时处理流)以及 QoS 适应机制(只在端系统上处理流)。许多传输连续媒体流的应用具有适应端到端 QoS 变化的能力。根据用户接受的 QoS 管理策略,端系统中的 QoS 适应机制可以采取一定的措施来处理不满足要求的信息流。在多媒体组播应用中,不同接收者的设备处理能力经常是不同的,QoS 过滤机制将弥合这种差异,使信息流能满足接收者的不同处理要求。

## 参考文献

1 林闯. 多媒体信息网络 QoS 的控制. 软件学报, 1999,10(10):1016~1024

2 Crawley E, Nair R, Rajagopalan B, Sandick H. A framework for QoS-based routing in the internet. IETF RFC 2386, August 1998

3 Hutchison D, Coulson G, Campbell A, et al. Quality of service management in distributed systems. In: Sloman M, ed. Network and Distributed Systems Management. Chapter 11. Addison Wesley, 1994

4 Seitz N B, Wortendyke D R, Spies K P. User-Oriented performance measurements on the ARPANET. IEEE Communication Magazine, 1983, 1(5): 28~44

5 Henshall J, Shaw S. OSI Explained: End-to-End Computer Communication Standards. ISBN 07458-0253-

2，Ellis Horwood，1988
6 Anderson D P，Herrtwich R G，Schaefer C. SRP：A resource reservation protocol for guaranteed performance communication in the Internet. Internet Report，University of California at Berkeley. 1991
7 Campbell A，Coulson G，Garc A F，et al. Integrated quality of service for multimedia communications. In：IEEE INFOCOM'93. San Francisco，USA，April 1993. 732～739
8 Lazar A A. Challenges in multimedia networking. In：International Hi-Tech Forum. Osaka，Japan. February 1994
9 Nilison G，Dupuy F，Chapman. An overview of the telecommunications information networking architecture. In：Tina'95. Melhourne，1995
10 Ferrari D. The tenet experience and the design of protocols for integrated services internetworks. Multimedia Systems Journal，November 1995
11 Volg C，Wolf L，Herrtwich R，et al. HeiRAT-Quality of service management for distributed multimedia systems. Multimedia Systems Journal，November 1995
12 IETF Working Group on Integrated Services. http：//www.ietf.org/html.charters/intserv- charter.html
13 IETF Working Group on Differentiated Services. http：//www.ietf.org/html.charters/diffserv-charter.html
14 CCITT. Draft Recommendations I.＊. Geneva，Switzerland，1990
15 Braden R，Clark D，Shenker S. Integrated services in the Internet architecture：An overview. IETF RFC 1633，June 1994
16 Wroclawski J. Specification of the controlled-load network element service. IETF RFC 2211，September 1997
17 Shenker S，Partridge C，Guerin R. Specification of guaranteed quality of service. IETF RFC 2212，September 1997
18 Shenker S，Wroclawski J. General characterization parameters for integrated service network elements. IETF RFC 2215，September 1997
19 Shenker S，Wroclawski J. Network element service specification template. IETF RFC 2216，September 1997
20 Braden R（Ed.），Zhang L，Berson S，et al. Resource ReSerVation Protocol（RSVP）—Version 1，Function Specification. IETF RFC 2205，September 1997
21 Nichols K，Jacobson V，Zhang L，et al. A two-bit differentiated services architecture for the Internet. IETF RFC 2638，July 1999
22 Carlson M，Weiss W，Blake S，et al. An architecture for differentiated services. IETF Internet RFC 2475，December 1998
23 Nichols K，Blake S，Baker F，et al. Definition of the differentiated services field（DS Field）in the IPv4 and IPv6 headers. IETF Internet RFC 2474，December 1998
24 Clark D D，Fang W J. Explicit allocation of best-effort packet delivery service. IEEE/ACM Trans on Networking，1998，6（4）
25 Wroclawski J. The use of RSVP with IETF integrated services. IEFT RFC 2210，September 1997
26 Nahrstedt K，Steinmetz R. Resource management in networked multimedia systems. IEEE Computer Magazine，May 1995
27 Davies N A，Nicol JR. A technological perspective on multimedia computing. Computer Communications，May 1991
28 **陈桦. 高速信息网络端到端服务质量控制的研究[博士学位论文]. 北京：清华大学，**1999 **年** 3 **月**

29 Campbell A, Coulson G, Garca F, et al. Integrated quality of service for multimedia communications. In: IEEE INFOCOM '93. San Fancisco, USA, April 1993. 732 ~ 739

30 Partridge C. A proposed flow specification. IETF RFC 1363, September 1992

31 Little T D C, Ghafoor A. Synchronization properties and storage models for multimedia objects. IEEE Journal of Selected Areas on Communications, 1990, 8(3): 229 ~ 238

32 Campbell A, Coulson G, Hutchison D. Supporting adaptive flows in a quality of service architecture. Multimedia Systems Journal, November 1995

33 Parekh A, Gallager R G. A generalized processor sharing approach to flow control in integrated service networks—the multiple node case. In: IEEE INFOCOM '93. San Francisco, USA, April 1993. 521 ~ 530

# 第2章

# 综合服务体系结构 IntServ

Internet 自 20 世纪 60 年代末出现以来，一直在以惊人的速度增长。同时，伴随多媒体技术的飞速发展，Internet 已逐步由单一的数据传送网向数据、语音、图像、视频等多媒体信息的综合传输网演化。视频点播、视频会议、IP 电话、远程教育、远程医疗等都是当今 Internet 上典型的分布式多媒体应用。通常，多媒体应用不但对网络有很高的带宽要求，而且要求信息传输的低延迟和低抖动等，另外，这些应用大多能够容忍一定程度的信息丢失和错误。可见，多媒体应用对网络提出了不同于数据应用的服务质量控制要求。但是，目前 Internet 中现有的传输模式仍为单一的尽力而为（best-effort）型服务，无法满足多媒体应用和用户对网络传输质量的不同要求。在这种情况下，以提高网络资源的利用率、为用户提供更高服务质量为目标的 QoS 控制技术应运而生，并且已经成为下一代网络的核心技术之一。

QoS 研究的目标是提供有效的端到端的服务质量控制或保证。近些年来 IETF 在此方面作了很多工作，先后提出了两种不同的 Internet QoS 体系结构：综合服务（integrated services，IntServ）[1] 和区分服务（differentiated services，DiffServ）[2]。本章将首先对 IntServ 予以介绍，DiffServ 将在第 3 章中进行讨论。

## 2.1 IntServ 概述

现有的 Internet 最初是面向非实时的、单一的数据类型通信而设计的。IP 协议提供的是一种无连接的网络层传输服务，必须辅以其他的高层协议（如 TCP 协议）才能更好地实现端到端的可靠传输，这种服务易受分组丢失、分组重复、路由器缓冲区队列延迟等的影响。由于缺少必要的 QoS 控制或保证，这种传统的 IP 传输服务称为“尽力而为”型服务（best-effort services）。尽力而为型服务的传输流不需要 QoS 控制，信息传输控制一般不依赖于网络状态，带宽分配可以动态地改变，但需要流控制，典型应用包括文件传输（FTP）和 E-mail 的传输。这种服务已经不能满足当今各种网络应用的需要。

尽力而为型服务无法为多媒体通信提供 QoS 保证。为了满足 Internet 多媒体应用实时传输的需求，IETF 定义了综合服务模型 IntServ[3]。RFC 1633 给出了 Internet IntServ 的框架，并将其划分为综合服务模型（integrated services model）与参考实现框架（reference

implementation framework)两大部分。

在服务的层次上,除了原来的尽力而为型服务以外,IntServ 还以每个流(单独的或聚集的)为基础,提供了两种端到端的面向实时传输的服务:质量保证型服务(guaranteed service)[4]和可控负载型服务(controlled-load service)[5],典型的应用如远程教学、视频点播等交互式音视频应用。实时性服务的本质在于要求服务提供保证。为了满足实时应用的需求,必须能够有效地管理资源(如带宽)。

在实现的层次上,IntServ 方案需要所有的路由器在控制路径上处理每个流的信令消息并维护每个流的路径状态和资源预留状态,并且在数据路径上执行基于流的分类、调度和缓冲区管理。

在技术层次上,IntServ 依靠资源预留协议 RSVP[6]提供 QoS 协商机制,逐节点(hop-by-hop)地建立或拆除每个数据流的路径状态和资源预留软状态(soft state);依靠接纳控制(admission control)决定链路或网络节点是否有足够的资源满足用户的资源预留请求;依靠传输控制(traffic control)将 IP 分组分类成不同的传输流,并根据每个流的状态对分组的传输实施 QoS 路由(routing)[7]、传输调度(scheduling)等控制。事实上,资源预留、准许控制和传输控制是实现 IntServ 的重要基石。

IP 网络 QoS 问题的研究促进了 IntServ 体系结构和 RSVP 信令协议的发展。RSVP 协议是为支持集成服务模型而设计的,它解决了应用的 QoS 需求问题,承诺对每个流的服务,使应用能够将每个流的请求信号发送给网络。在进行接纳控制时,使用集成服务参数(如流参数 TSpec 和 RSpec)来量化这些服务请求。实际上,RSVP 信令协议依靠一种动态的虚电路连接机制而改变了整个网络的体系结构。IntServ 是基于每个流的(per-flow)、状态相关(stateful)的体系结构。与状态无关的体系结构(如传统的 IP 网络和最新的 DiffServ 的体系结构)相比,IntServ 的优势在于它提供的服务具有更高的灵活性和更好的 QoS 保证。

## 2.2 IntServ 模型

IntServ 模型实质上是一个从端到端行为开始,到网络中各元素如何控制和实现这些行为,为用户提供满意的 QoS 的总称。

IETF 考虑综合服务模型时的主要出发点是网络的延迟因素,即在 Internet 中,被转发的分组实际上究竟具有多大延迟和只能允许它具有多大延迟。事实上,分组传输延迟在当前是衡量 QoS 好坏的主要标准。

在 Internet 的综合服务模型中,将延迟分为两大部分:传输延迟和等待延迟。

传输延迟是指传输介质以及路由和转发处理等带来的延迟。综合服务模型把这一部分延迟称为固定延迟。QoS 控制机制无法控制这一部分延迟,它由传输和交换设备、计算机的 CPU 处理能力等决定。

等待延迟是指分组组成的流在交换设备、路由器以及终端主机系统等网络元素的缓冲区中组成的等待队列,因为得不到处理或因网络拥塞所带来的延迟。IETF 所讨论的 QoS 控制主要集中在如何处理等待延迟上。

根据应用对延迟的要求,综合服务模型把 Internet 提供的服务分为与传输时间无关的尽力而为型服务、质量保证型服务以及可控负载型服务等。然后又进一步定义了这些服务的资源共享方式,以及为了保证 QoS 而应该采取的相应策略。

## 2.3　IntServ 的服务类型

如 1.2.4 节所述,用于定义 IETF 服务的服务类型空间为[2, 127],即允许 126 种服务存在。不过,当前,IETF 只定义了尽力而为型服务、质量保证型服务、可控负载型服务以及区分服务。

传统的 IP 网络协议只能提供简单的点到点的尽力而为型服务,而且服务类型惟一。所有应用不加区分地得到系统的尽力而为型服务,这显然不利于对 QoS 需求敏感的实时媒体等业务。为了提供多媒体通信的多点功能和 QoS 支持,就必须对传统的网络结构和服务模型进行扩展,从而提出了质量保证型服务和可控负载型服务。

### 2.3.1　可控负载型服务

IETF RFC2211[5] 中把可控负载型服务定义为一种端到端的行为。可控负载型服务使用户感到网络是在一种很轻的负载或具有很大的容量条件下运行,用户感觉不到不可忍耐的延迟。可控负载型服务是一种‘软实时’(soft real-time)服务,这种服务模式用于模拟用户在没有重负载和拥塞时的服务模式,它在本质上是一种定性的服务。具体而言:

(1) 很高百分比的分组被成功地转发给接收端,损失或丢失的分组百分比必须低于允许的范围;

(2) 绝大部分转发成功的分组的延迟率要小于一个可以接受的延迟范围。

实际上,上述定义是模糊的。很高的百分比、绝大部分以及允许范围都不是定量的描述。可控负载型服务把这些定量描述交给用户,以便该服务能够支持大范围的应用。

可控负载型服务需要指定用户服务指标作为接纳控制处理的关键输入,以限制流的数目,从而保证网络处于非重载的网络模式。为了保证可控负载型服务的条件得到满足,用户将首先给提供可控负载型服务的网络元素一个所需要的数据流量估值。这个估值用 TSpec 描述。然后,网络元素将验证自己是否具有足够的资源提供用户的 TSpec 所要求的流量。如果有的网络元素不具备相应的能力,系统就反馈给用户相应的过载信息,包括可能造成的大量丢失分组或延迟。

可控负载型服务模式提供的是一个得到保证的持续传输速率以及偶然出现的突发峰值速率,这两个速率可以沿用令牌桶方法加以描述,它适合于那些能够容忍一定限度丢失和延迟的应用类型。虽然它没有明确的延迟和丢失保证,但为服务描述的具体实现机制带来了一定的灵活性,它可以利用简单的 FCFS 调度机制与基于有效带宽的接纳控制或基于测量的接纳控制技术来配合实现。

### 2.3.2　质量保证型服务

IETF RFC2212[4] 中定义了质量保证型服务。质量保证型服务要求网络中各元素保

证用户所要求的最小延迟时间,从而保证会话(session)过程中每个分组确定的延迟界限(bound),即保证在规定的传送时间内到达,只要数据流的传输保持在特定的传输参数范围内,就不会因为队列的溢出而被丢弃。这一点不同于可控负载型服务。在可控负载型服务中,网络元素最大限度地接近用户的延迟要求,但不能给予保证。质量保证型服务是一种‘硬实时”(hard real-time)服务。

为此,提供质量保证型服务的各个网络元素必须先对各个服务参数所要求的资源进行计算,并给出这些元素可能带来的最大延迟。具体而言,质量保证型服务具有以下三个特点:

(1)质量保证型服务要求用户描述清楚应用的需求,而不是实现这些需求的机制或指明流的方法。

(2)为了获得一个有限的延迟,质量保证型服务的数据传输路径上的每个网络元素都必须支持质量保证型服务。也就是说,每个网络元素都必须提供有限的网络延迟。

(3)尽管质量保证型服务要求网络系统提供在要求延迟内的服务,但不能100%地保证每一个所要求的时间延迟都能得到满足。

质量保证型服务的目的在于模拟某种特定速率电路所提供的服务,对于实时性要求较高的多媒体应用较为合适。为了实现它所提供的定量带宽和延迟保证,网络节点必须采用更加复杂的调度机制来保证服务的要求。

## 2.4 资源共享要求与服务范围

无论是质量保证型服务还是可控负载型服务,它们都需要网络元素为其分配相应的资源。这些资源包括网络带宽、CPU和缓冲区等。为了分配和管理这些资源以获得所要求的服务,首先必须区分清楚是由哪些资源共享而带来了延迟。

网络元素的资源共享可以分为三个层次:链路的共享、协议的共享以及服务的共享。其层次关系如图2.4.1所示。

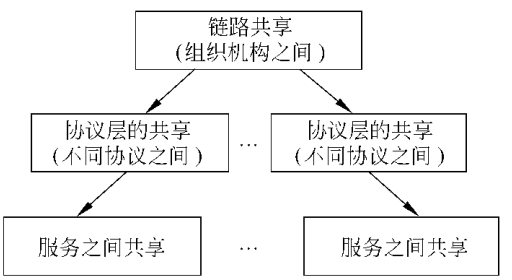



图2.4.1 Internet资源共享的层次结构

链路的共享一般由不同的组织机构和部门进行,它们共用相同的链路传输信息,例如以太网和公用数据网DDN等。

协议层的共享是指在Internet上除了IP协议之外,还运行着其他协议,例如IPX。这些协议经过IP进行头部封装之后或以其本身的状态共享路由器(很多路由器都是多协议

的）等软硬件资源。这造成了不同协议之间的带宽、CPU 等资源的竞争。

服务的共享则在一个协议家族，即 IP 协议内进行。不同用户的不同应用同时经过某一个网络元素，造成该网络元素的处理能力下降。

为了处理这三个层次的资源共享与竞争，网络系统必须在用户提出 QoS 要求之后进行资源检查，然后根据资源多少和用户的要求，接受或拒绝用户的请求。对进入 Internet 准备获得相应服务的用户应用进行控制的过程即为接纳控制。

在用户提出 QoS 要求之后，网络系统要检查各网络元素的资源状况，在网络资源充足的情况下，为了防止其他服务、协议或链路用户占有网络资源，系统要对网络各元素中的空闲资源按照相应的传输路径进行资源预留。RSVP 就是专门用于资源预留的协议。

对于那些已经在网络运行的应用，尽管在接纳控制和资源预留时已对它们所要求的资源进行验证和预留，但仍有可能因为其他因素或更高优先级应用的存在而导致资源得不到保障。这时，网络元素必须放弃部分所传输和转发的分组，或者是发出无法保证所要求的服务质量的信息。对于质量保证型服务而言，放弃或者延迟在某个范围内的音频与视频分组，有时对整个应用的影响不大，但有时将给应用带来很大影响。

## 2.5　QoS 控制的实现框架

QoS 控制的实现是一个端到端的过程，实现 QoS 控制的目的是为用户提供多种类型的综合服务。

具体而言，QoS 控制的实现框架包括以下几个部分：

（1）用户与用户、用户与网络系统的 QoS 协商方法与界面；

（2）用户 QoS 要求的接纳控制；

（3）QoS 参数与服务类型的控制分组的定义与实现；

（4）资源预留协议；

（5）分组调度与队列管理方法；

（6）QoS 控制管理与评价标准。

QoS 控制的实现框架必须是端到端的，即 QoS 的控制程序既要在端主机系统实现，也必须在提供用户服务的综合服务路径的各网络元素，包括路由器和交换机等中实现。

如图 2.5.1 所示的路径不能为用户提供所要求的 QoS。在图 2.5.1 中的 A，B 两点，

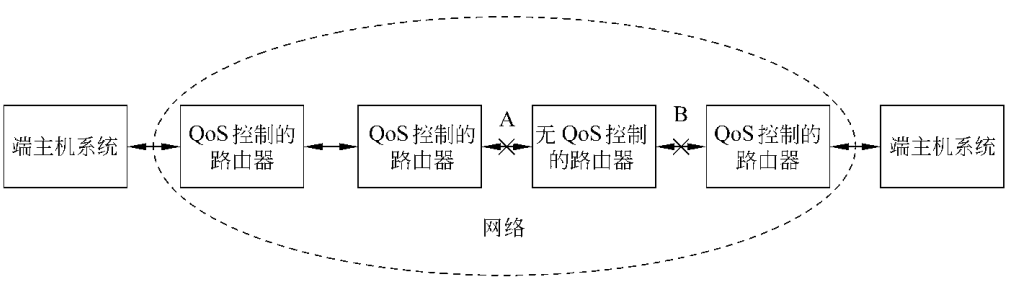



图 2.5.1　无法提供 QoS 服务的路径

由于相应的路由器设备不提供 QoS 控制功能,所以图中所示的 Internet 分组传输路径无法为用户提供端到端的 QoS 控制功能。

## 2.6 QoS 控制参数

QoS 控制参数的定义与 RSVP 协议密切相关。这些参数按照 RSVP 规定的瞬间进行操作和处理。QoS 控制参数相当于一种控制信令,它是实现用户要求的各类综合服务的基本依据和标准,集中反映了用户 QoS 要求中的服务类型、相关参数以及处理方式等。QoS 控制参数不同于数据分组,它是为了使数据分组得到良好的服务而设置的。

支持质量保证型服务与可控负载型服务的主要 QoS 控制参数有三类:FLOWSPEC,ADSPED 和 SENDER-TSPEC。这些参数由 RSVP 协议用来传送认证 QoS 要求以及调度策略等信息。RSVP 协议中把这些参数统称为对象。

对象 FLOWSPEC 传输信息接收端(例如访问服务器的用户)所要求的流量(TSpec)和调用这些服务的参数(RSpec)。

对象 SENDER-TSPEC 描述信息发送端的数据流量。需要注意的是,由于信息发送端的数据流量不受 RSVP 或其他 QoS 控制协议与机制的控制,因此网络中的其他元素只能接收该流量信息而不能对其做出任何修改。

对象 ADSPEC 则用于由网络中的各个网络元素向接收端发送关于延迟、估计带宽、QoS 控制服务操作的参数以及支持的 QoS 控制服务等信息。一般来说,ADSPEC 传送给接收端的信息是计算和综合了它所通过的各个节点的有关参数之后形成的。

综上所述,对象 SENDER-TSPEC 参数是由数据源端产生和发送给网络元素与数据接收者的,它不能在传输过程中被网络元素修改。

对象 ADSPEC 是由数据源端或其他网络元素产生并发送给数据接收者的。它传送两类信息,即数据路径信息和所要求的 QoS 服务的参数信息,是由用户在同一网络元素上的不同服务的相关参数构成的。这些信息是每个所通过的节点计算和综合后的结果。

对象 FLOWSPEC 是由数据接收者产生,并发送给数据传输路径沿途的网络元素与数据源端。为了获得相应的服务,FLOWSPEC 参数必须指明用户所要求服务的类型以及与该服务类型相关的参数。因此,FLOWSPEC 主要包括了资源预留所需要的信息。

上述控制参数的具体格式参见 RFC2210[8] 中的定义。

## 2.7 资源预留协议 RSVP

### 2.7.1 RSVP 简介

RSVP 协议[6] 最早于 1993 年提出,用于点到点通信(unicast)和点到多点通信(multicast)的 Internet 网络环境中多媒体用户对网络资源的预留。

RSVP 的资源预留必须是由接收端到发送端的端到端过程。它具有如下特点:

(1) 可对点到点通信、点到多点通信方式进行资源预留。

（2）RSVP 采用单方向预留方式，即由数据流的接收端向数据源沿路径进行预留。这里，数据流是指从数据传输源到所有目的地的有方向的树状路线，其中，接收数据流的接收端称为数据流的下游，发送数据流的源端称为上游。

（3）RSVP 在路由器等网络元素上设置和维护记录路由和资源预留信息的软状态表，并能根据路由和预留信息的变化进行自动更新和调整。

（4）RSVP 能够根据用户对数据源的访问需要提供不同的预留方式。

（5）RSVP 提供流量控制和传输策略控制。

（6）RSVP 既支持 IPv4 协议，也支持 IPv6 协议等。

RSVP 的主要构成元素是控制分组、控制分组参数及其有关格式（在 RSVP 中称为对象）、控制状态和预留风格。

RSVP 包括两类最基本的控制分组：PATH（控制）类分组和 RESV（预留）类分组。PATH 类分组由数据源端发出，RESV 则由数据接收端作为对 PATH 路径中各网络元素的资源要求沿 PATH 分组设置的路径返回发出。如果接收端不需要资源预留，则不返回 RESV 分组，而直接沿相应的路径接收来自源端的信息。RSVP 的分组之间的关系如图 2.7.1 所示。

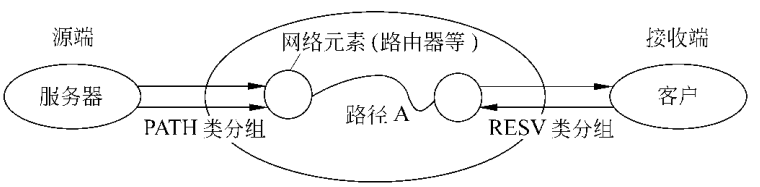



图 2.7.1　RSVP 协议中的路径与消息

发送 PATH 类分组或 RESV 分组的初始过程由 RSVP 会话决定。一个 RSVP 会话定义一个预留需要的点到点传输或点到多点传输的数据流。RSVP 的会话由相应的目的地址、传输层协议和会话端口来识别。

RSVP 协议可在 IP 协议（IPv4 或 IPv6）上实现，也可在 UDP 协议上实现。RSVP 协议在服务分类思想的具体实现时选择了无连接的方式。这种实现方式使 RSVP 协议在预留资源的共享方面更具灵活性。因此，RSVP 协议与 IP 的基本思想具有高度一致性。RSVP 协议的实现环境如图 2.7.2 所示。

RSVP 协议通过相应的 PATH 类分组和 RESV 分组，沿着数据流路径的所有节点传递 QoS 控制参数和要求，并在路由器等网络元素上建立和维护 QoS 控制服务用的路径状态和预留状态，这些状态称为“软状态”。软状态实际上是一种无连接的方式，它定期刷新存储在网络元素相应表格中的路径状态和预留状态信息。而且，预留分组和路径分组设置的软状态还可以被“拆除”分组来删除。

RSVP 协议从接收端开始向发送端要求预留资源。通常把发送端称做数据流的上游，接收端称做数据流的下游。所以，RSVP 协议的资源预留是一个由下游向上游方向进行预留的单向预留过程。

由于 RSVP 协议所对应的资源预留环境为多数据源和多接收端的点到点通信与点到

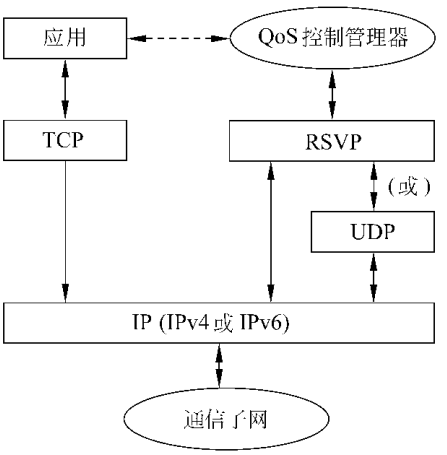



图2.7.2 RSVP协议的实现环境

多点通信的网络环境,当由接收端向源端进行资源预留时,就会产生多个接收端指定同一发送源或不同的接收端指定不同的发送源等问题。因此,针对不同的资源预留需要,RSVP定义了不同的资源预留模式,即固定模式(fixed filter style)、通配符模式(wildcard style)和显式共享模式(shared explicit style)。其中,固定模式是固定的接收端对固定的源端的预留,一旦用户的应用程序通过了接纳控制并进行了固定式预留,则在该预留被拆除之前,所确定的源不能被修改。通配符模式是指所有预留同一源端的不同接收端用户可以共享同一路径中的带宽和缓冲区等资源。使用显式共享模式的不同接收端用户在共享路径中的带宽与缓冲区等资源的同时,还可以选择指定不同的信息源。

## 2.7.2 RSVP的工作原理

### 2.7.2.1 RSVP实现资源预留的过程

基于上述对RSVP的介绍,图2.7.3具体描述了RSVP协议的路径分组'PATH'和资源预留分组'RESV'在数据流的发送者和接收者之间实现端到端资源预留的过程,具体的实现步骤如下:

(1)发送数据的源端确定发送数据流所需的带宽、延迟和延迟抖动等指标(即TSpec参数),并将其包含在PATH分组中发给接收端。

(2)当网络中的某一路由器接收到PATH分组时,它将PATH分组中的路径状态信息存储起来,该路径状态信息描述了PATH分组的上一级源地址(即发来该分组的上一跳路由器地址)。

(3)当接收端收到PATH分组之后,它沿着与PATH分组中获取的源路径相反的方向发送一个RESV分组。该RESV分组包含为数据流进行资源预留所需要描述的流量和性能期望等QoS信息。

(4)当某一路由器接收到一个RESV分组时,它通过接纳控制来决定是否有足够的

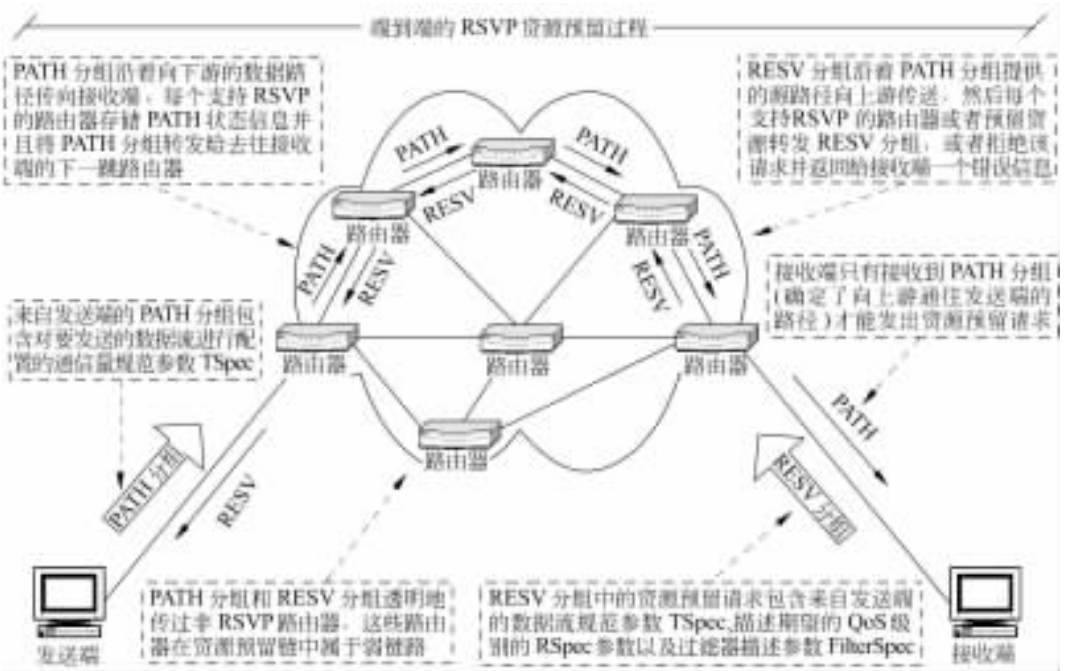



**图 2.7.3　RSVP 的 PATH 分组和 RESV 分组在发送端和接收端之间实现资源预留的示意图**

资源满足 QoS 请求。如果有，就进行带宽和缓冲区空间的预留，并且存储一些与数据流相关的特定信息，然后将 RESV 分组转发给下一个路由器 如果路由器必须拒绝该请求，则它返回给接收端一个错误信息。

（5）如果源端接到 RESV 分组，则表明数据流的资源预留已经成功，可以开始向接收端发送数据。

（6）当数据流发送完毕，路由器可以释放先前设置的预留资源。

### 2.7.2.2　RSVP 与其他 QoS 控制模块的关系

RSVP 的基本功能是通过预约和保留传输路径中的资源而改善或保证用户应用的 QoS。RSVP 对资源的预留是端到端的，它涉及到主机和路由器等网络元素。图 2.7.4 给出了在主机和路由器中使用 RSVP 实现 IntServ 的工作原理。

由图 2.7.4 可知，每一个具有 QoS 控制能力的节点都让应用的数据分组通过一个分组分类器。该分类器决定分组的路由和所要求的 QoS 的级别，然后，把具有相应路由和 QoS 级别的分组转交给分组调度器，并按照所要求的 QoS 进行调度转发。

在每个节点中，一个 RSVP 的 QoS 控制要求分组（路径分组与预留分组）都要交给两个决策模块处理，这两个决策模块是接纳控制模块和策略控制模块。如前所述，接纳控制模块原来决定预留分组是否可以进入该节点进行资源预留，也就是检查用户的预留分组中的 QoS 要求是否大于该节点中可供使用的网络资源。策略控制模块主要用于检查用户指定预留及其预留模式的权限。如果用户的预留分组通过了上述两个控制模块的检

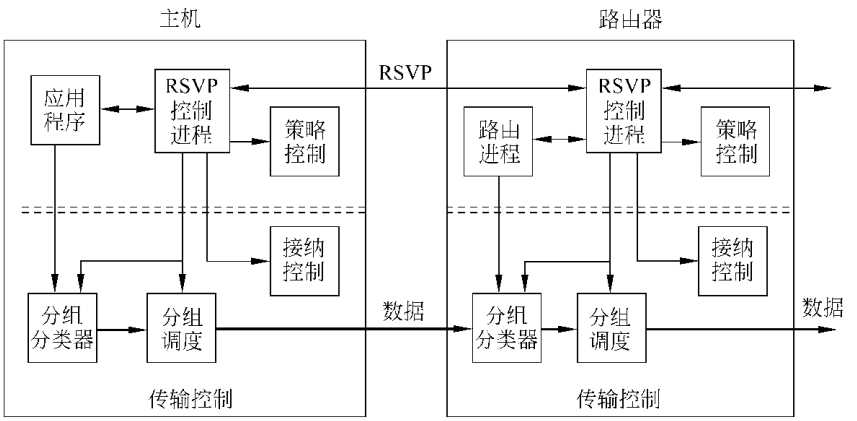



**图 2.7.4　在主机和路由器中通过 RSVP 实现 IntServ 的原理图**

查,则可以进入分组分类服务器与分组调度器设置相应的参数和状态,并控制数据分组的转发。

RSVP 控制进程则除了控制策略模块、接纳控制模块等之外,还负责转发路径分组和预留分组,并维护和刷新路由器中的软状态表。

### 2.7.2.3　RSVP 的控制分组

如前所述,RSVP 的基本分组分为两类,即 PATH(路径)类分组和 RESV(预留)类分组。PATH 类分组主要用来支持 OPWA(one pass with advertising)服务,即发送者把 PATH 类分组发送给下游,并沿着数据被传送的路径收集各网络元素的资源信息,同时将这些信息传送给数据接收端用户,以便其做出是否进行资源预留的选择。

PATH 类分组中包含前一节点的地址,另外还包括描述发送者数据分组的格式、定义发送者数据流的流量特性的 SENDER-TSPEC,以及各种服务类型参数的 ADSPEC。

RESV 类分组由数据分组的接收端发送到源端,以预留相应的资源。一个基本的 RESV 类分组包括参数 FLOWSPEC 和参数 FILTERSPEC。FLOWSPEC 由 RSpec 和 TSpec 组成。RSpec 和 TSpec 分别描述 QoS 所要求的流量和性能期望。FILTERSPEC 选择数据流的预留方式和相应的数据源会话层端口。

由于 RSVP 适用于点到点以及点到多点的通信环境,因此,在向上游转发预留分组时,将会碰到预留格式与合并以及状态更新等问题,其中软状态更新问题主要是协议在路径变化或超时时自动进行。

预留分组有三种预留模式,即固定模式、通配符模式和显式共享模式。这三种模式的含义已经在前面进行过简单描述,即采用固定模式的接收端不和其他数据流共享同一信息资源和相应的网络资源,而且一旦预留确定之后,只有等待相应应用结束之后或拆除分组到来之后才能释放。通配符模式的资源共享与数据流的源端无关,因此所有经过同一路径的数据流可共享相同的资源。显式共享方式则可以选择不同目的地址的数据源进行共享。这三种预留方式的预留分组都可以进行合并。

## 2.8　IntServ 的 QoS 研究

目前 IntServ 的 QoS 研究和开发在很多方面仍然是开放的，其主要问题是[9]：网络系统状态和链路带宽容量变化的不确定性，使传输通道端到端带宽预留缺乏有效的保证；QoS 路由、资源预留和传输调度算法的复杂性，还不能适应信息高速传输处理时间的要求；QoS 要求所导致的资源利用的无效性，不能充分利用网络资源提高网络的吞吐量；网络多址和多路径传输的 QoS 控制策略和算法的复杂性，仍然是所面临的巨大挑战；一些基本研究成果主要还存在于理论中，还没有形成专利或技术产品；现存的网络交换机或路由器还不能保证用户服务质量，缺乏简单、有效的控制方案和算法的实现，传输管理与控制亟待改进。

QoS 路由和基于 QoS 的传输调度是当前 IntServ 领域 QoS 研究的热点问题。

QoS 路由[7]需要在信息传输之前在源节点和目的节点之间建立某种连接[10]，为这个连接预留有效的带宽等资源。带宽预留的原则是确保 QoS 传输的端到端延时和分组丢失率，但目前的研究偏重端到端延时保证。QoS 路由主要包括管理路由信息（例如，链路状态或距离向量）和实现路由算法。QoS 路由机制要具备可行性（实现端到端延时等保证）和有效性（符合网络资源管理要求），其算法应达到两个重要目标：减少计算的复杂性和优化网络资源的使用[11]。其中，不确定信息环境中的路由[12]、具有多重度量（metric）的 QoS 路由[11, 15]、QoS 组播路由[13~15]、QoS 路由实现及其性能评测[16, 17]等问题难度较大，仍需更深入的研究。

调度算法的基本功能是从节点的每一个输出链路中挑选出在下一个有效周期发送的分组。QoS 传输调度控制要基于如下原则：带宽的保证、流的隔离、延时的保证和公平选择等[18]。协议和算法的复杂性要适应网络高速传输并便于实现，使其具有可扩展性和鲁棒性。调度算法可以分为两类：基于速率的调度算法和基于时间的调度算法。目前最主要的调度策略都是近似广义处理器共享 GPS（generalized processor sharing）的调度策略。简化近似 GPS 策略调度算法实现的复杂性是 QoS 调度策略主要的研究方向[19~22]。

## 2.9　IntServ 的局限性

IntServ 的产生是为了使 IP 成为新的信息基础设施，一旦实现，必然对人类的通信方式又是一次巨大变革。虽然 IntServ 在经过多年的研究和发展后已初具规模，但其中的问题也逐步显现出来。

IntServ 是基于流的（单独的或聚集的）、状态相关的体系结构，依赖于每个流（per-flow）的状态和对每个流的管理。这种实现机制一方面使 IntServ 能够提供较状态无关的体系结构具有更高的灵活性和更好的服务级别保证的服务，但同时也导致了 IntServ 的可扩展性问题和鲁棒性问题，后果是实现复杂，难于应用。

在 IntServ 体系结构中，网络中每个节点都要维护各类数据库（例如，对于使用链路状态协议的 QoS 路由，路由器必须同时维护链路状态数据库和 QoS 路由表；网络中的资源

管理器必须维护资源数据库等)并实现复杂的功能模块(如资源预留、路由、接纳控制等)。RSVP 信令协议提供 QoS 协商机制;各网络节点建立和维护预留信息,并根据自身资源状况对用户的预留请求进行接纳控制;数据传输时各网络节点监控传输流,并提供相应服务。这种完全分布式的控制带来了极大的复杂性。

最初 IntServ 只面向单个微流(micro-flow)(由源地址、目的地址、源端口、目的端口、IP 协议号五元组描述的应用流),在路由器配置和使用多域 MF(multifield)分类准则,这就给路由器尤其是主干网络核心路由器带来了巨大负荷。近期 RSVP 已开始尝试支持流聚集(aggregate)[23~26],一些提议试图通过将沿着同一路径的微流聚集成宏流(macro-flow)[23,27]来减少网络内部传输流的数量。这些提议对于问题是有效的,却不能根本解决问题——IntServ 体系结构本身已决定了其高度复杂性,而且宏流的数量在具有许多边界路由器的网络中仍然会很大,因为路径的数量是边界节点数量的平方函数。

IntServ 这种基于单个流的、状态相关的机制高度复杂,导致的直接后果是可扩展性差、鲁棒性差。而且,在纯粹数据网络的分层结构上实现 IntServ 和 RSVP 信令机制及其相关的功能需要大量的投资,需要在遍布世界范围的网络中引入繁重的软件和硬件修改,实现难度大。这些都严重妨碍了 IntServ 在大型网络,特别是重负载网络中的应用。

总而言之,RSVP 和 IntServ 在整个 Internet 网络应用,存在如下根本的局限[28]:

(1)基于流的 RSVP 资源预留、调度处理以及缓冲区管理,有利于提供 QoS 保证,但使系统开销过高,对于大型网络存在可扩展性的问题;

(2)目前,只有少量的主机产生 RSVP 信令,虽然其数量预计会大幅度增长,但许多应用却从不产生 RSVP 信令,因而实现上修改应用程序的阻力较大;

(3)许多应用需要某种形式的 QoS,但却无法使用 IntServ 模型来表达 QoS 请求;

(4)必要的策略控制(policy control)和价格(pricing)机制,如访问控制(access control)、鉴别(authentication)、记账(accounting)等,目前尚处于发展阶段,无法付诸应用[29]。

因而,单纯的 IntServ/RSVP 结构实际上无法被业界接受,在商业上不可能有大的作为。研究者们认为,IntServ/ RSVP 以其现有的形式将不会在 Internet 中得到广泛应用[30]。

现阶段,IntServ 极有可能会应用在企业网边缘、校园网以及小型的公司网络中,这些网络中的用户数据流可以在桌面用户一级进行管理。推动 IntServ 在桌面附近应用的一个重要因素是微软公司在 Windows 98,NT5.0 等操作系统中提供了 RSVP 和 QoS 功能。

## 参考文献

1 IETF Working Group on Integrated Services. http://www.ietf.org/html.charters/intserv-charter.html

2 IETF Working Group on Differentiated Services. http://www.ietf.org/html.charters/diffserv-charter.html

3 Braden R, Clark D, Shenker S. Integrated services in the Internet architecture: An overview. IETF RFC 1633, June 1994

4 Shenker S, Partridge C, Guerin R. Specification of guaranteed quality of service. IETF RFC 2212,

September 1997

5 Wroclawski J. Specification of the controlled-load network element service. IETF RFC 2211, September 1997

6 Braden R, ed. Zhang L, Berson S, et al. Resource ReSerVation Protocol (RSVP)—Version 1, Function Specification. IETF RFC 2205, September 1997

7 Crawley E, Nair R, Rajagopalan B, et al. A framework for QoS-based routing in the Internet. IETF RFC 2386, August 1998

8 Wroclawski J. The use of RSVP with IETF integrated services. IETF RFC 2210, September 1997

9 **林闯. 多媒体信息网络 QoS 的控制. 软件学报**, 1999,10(10):1016 ~ 1024
(Lin Chuang. On QoS control of multimedia information networks. Journal of Software, 1999,10(10): 1016 ~ 1024)

10 Cidon I, Rom R. Multi-path routing combined with resource reservation, In:IEEE INFOCOM '97. Kobe, Japan, April 1997. 92 ~ 100

11 Orda A. Routing with end to end QoS guarantees in broadband networks. In:IEEE INFOCOM '98. San Francisco, California, March 1998. 27 ~ 34

12 Lorenz D H, Orda A. QoS routing in networks with uncertain parameters. In:IEEE INFOCOM '98. San Francisco, California, March 1998. 3 ~ 10

13 Ye Ge, Jennifer C. Hou, Hung-ying Tyan. A packet eligible time calculation mechanism for providing temporal QoS for multimedia routing. In:Proc of the IEEE International Conference on Communications. Vancouver, Canada, June 1999. 721 ~ 726

14 Lim YuJin, Choe JongWon. A layered QoS multicast protocol. In:Proc of the IEEE International Conference on Communications. Vancouver, Canada, June 1999. 687 ~ 691

15 Melody Moh W, Nguyen Bang. An optimal QoS-guaranteed multicast routing algorithm with dynamic membership support. In:Proc of the IEEE International Conference on Communications. Vancouver, Canada, June 1999. 727 ~ 732

16 Apostolopoulos G, Guerin R, Kamat S. Implementation and performance measurements of QoS routing extensions to OSPF. In:IEEE INFOCOM '99. New York City, March 1999. 680 ~ 688

17 Apostolopoulos G, Guerin R, Kamat S, et al, QoS routing mechemisms and OSPF Extensions. IETF RFC 2676, December 1998

18 Lin Chuang, Lam E C M. Dynamic queue length thresholds for scheduling real-time traffic in ATM networks. In:Proc of 1999 International Conference on Communications. IEEE Computer Society, the Vancouver, BC Canada, 1999. 869 ~ 874

19 Golestani S J. A self-clocked fair queueing scheme for broadband applications. In:IEEE INFOCOM '94, Toronto, Canada, June 1994. 636 ~ 646

20 Zhang L. Virtual clock:A new traffic control algorithm for packet switching. ACM Trans on Comp Syst, 1991. 101 ~ 124

21 Bennett J C R, Zhang H. $WF^2Q$:Worst-case fair weighted fair queueing. In:IEEE INFOCOM '96. San Francisco, California, USA, March 1996. 120 ~ 128

22 Chiussi F M, Francini A. Implementing fair queueing in ATM switches:The discrete-rate approach. In: IEEE INFOCOM '98. San Francisco, California, March 1998. 272 ~ 281

23 Guerin R, Blake S, Herzog S. Aggregating RSVP-based QoS requests. IETF Internet Draft 〈draft-guerin-aggreg-rsvp-00. txt〉, November 21, 1997

24 Berson S, Vincent R. Aggregation of Internet integrated services state. IETF Internet Draft 〈draft-berson-rsvp-aggregation-00. txt〉, August 1998

25 Baker F, Iturralde C, le Faucheur F, et al. RSVP reservation aggregation. IETF Internet Draft 〈draft-ietf-issll-aggregation-00. txt〉, September 1999

26 Terzis A, Krawczyk J, Wroclawski J, et al. RSVP operation over IP tunnels. IETF Internet Draft 〈draft-ietf-rsvp-tunnel-04. txt〉, May 1999

27 Fred Baker, Carol Iturralde, Francois Le Faucheur, Bruce Davie. Aggregation of RSVP for IPv4 and IPv6 reservations. IETF Internet Draft 〈draft-baker-rsvp-aggregation-01. txt〉, June 1999

28 Bernet Y, Yavatkar R, Ford P, Baker F, Zhang L, Nichols K, Speer M. A framework for use of RSVP with diff-serv networks. IETF Internet Draft 〈draft-ietf-diffserv-rsvp-01〉, November 1998

29 Boyle J, Cohen R, Durham D, et al. COPS usage for RSVP. IETF Internet Draft 〈draft-ietf-rap-cops-rsvp-05. txt〉, June 1999

30 **林闯, 单志广, 盛立杰, 吴建平. 区分服务及其几个热点问题的研究. 计算机学报,** 2000,23(4): 419～433

# 第3章

# 区分服务体系结构 DiffServ

## 3.1 DiffServ 概述

在 IntServ 体系的发展遭遇巨大障碍的时候，DiffServ 应运而生[1]。事实上，也正是 IntServ 的推动者缔造了 DiffServ[2]——从这个意义上讲，两者是一脉相承的，因而 DiffServ 与 IntServ 结合的问题自然也就贯穿了整个 DiffServ 的发展过程。

DiffServ 的目标在于简单有效，以满足实际应用对可扩展性的要求，其实现途径是：

（1）简化网络内部节点的服务机制。在内部节点只进行简单的调度转发，而流状态信息的保存与流监控机制的实现等只在边界节点进行，内部节点是状态无关的。

（2）简化网络内部节点的服务对象。采用聚集传输控制，服务对象是流聚集（stream aggregate）而非单流，单流信息只在网络边界保存和处理。

具体而言，边界节点根据用户的流规定（profile）和资源预留信息将进入网络的单流分类、整形、聚合为不同的流聚集，这种聚集信息存储在每个 IP 包头的 DS（differentiated services）标记域（field）[3,4] 中，称为 DS 标记（differentiated services codepoint，DSCP）；内部节点在调度转发 IP 包时根据包头的 DSCP 选择提供特定质量的调度转发服务，其外特性称为逐点行为（per-hop-behavior，PHB）。网络边界对单流做分类聚合与网络内部对聚集流提供特定质量的调度转发服务，这两个过程通过 IP 包头内的 DSCP 协同起来。

除实现简单以外，区分服务体系还具有以下特点：

（1）层次化结构。分为 DS 区域（DS domain）与 DS 区（DS region）两级。在 DS 区域内，服务提供策略与 PHB 的语义和实现要一致；但 DS 区内的各 DS 区域可以支持不同的 PHB、有不同的服务提供策略，它们之间通过服务层协议（service level agreement，SLA）与传输调节协议（traffic conditioning agreement，TCA）协调以提供跨区域服务。这种结构适应了 Internet 中由各 ISP 提供接入服务的商业模式。

（2）总体集中控制策略（与 IntServ 分布式控制相对照）。网络资源的分配由总体服务提供策略（service provisioning policies）决定，包括在边界如何分类聚合流，在内部如何调度转发流聚集。

（3）利用面向对象的模块化思想与封装思想，增强了灵活性与通用性。各逻辑模块相对独立，并有多种组合。少量模块可组合实现多种服务，并在发展过程中保持模块的可

重用性。例如,服务类型与边界调节器(conditioner)和内部 PHB 相对独立,使得较少种类的边界调节器和内部 PHB 可进行各种不同的组合以实现多种服务类型;而且随着进一步研究发展可能有更多服务类型出现但仍可以重用已有模块构造。再如,PHB 与 PHB 的具体实现机制相分离,使 PHB 可以在发展中保持相对的稳定,这给商家留下了施展的天地。

(4) 不影响路由。与一些以虚电路方式实现 QoS 的方案(ATM, MPLS)以及服务类型标记方案不同,区分服务节点处提供服务的手段仅限于队列调度与缓冲管理,不涉及路由选择机制。

## 3.2 DiffServ 的体系结构

目前 DiffServ 仍在不断发展,其相关概念、模型的定义仍处于讨论阶段。到现在为止,DiffServ 的体系结构已比较明确[3,5],在此基础上,有关服务提供的相关问题——包括服务定义、设置、管理等细节也在逐步清晰化[6]。DiffServ 模型[7]从软件工程中概念模型的角度讨论了在路由器中实现区分服务所需各种模块的组织结构。Linux 核心从 2.0 版开始提供可支持 DiffServ 分类、调度等的 QoS 模块[8],因而 Linux 可以作为 DiffServ 的测试平台[9]。

区分服务体系机构的框架如图 3.2.1 所示。

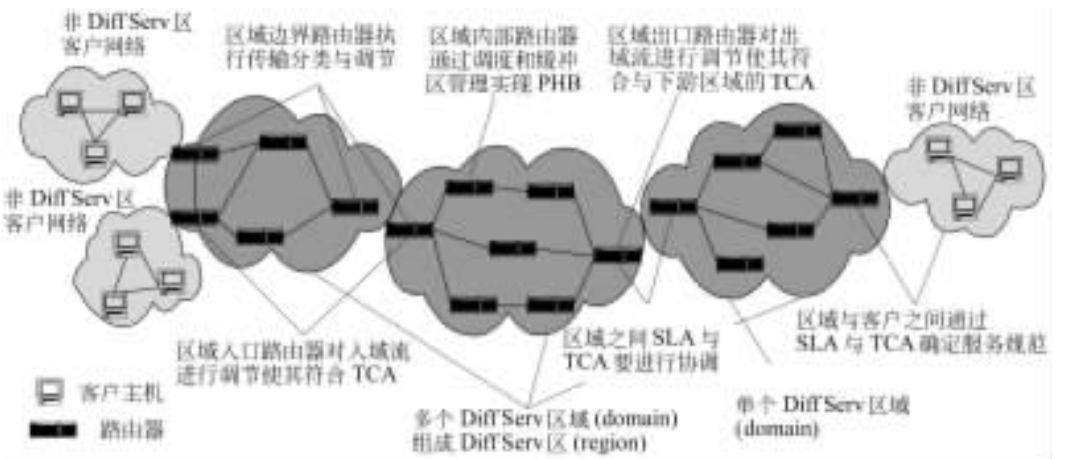



图 3.2.1 区分服务体系结构的框架示意图

下面详细介绍区分服务体系结构的各个组成部分。

### 3.2.1 DS 区域与 DS 区

DS 区域是由一些相连的 DS 节点构成的集合,它们遵循统一的服务提供策略并实现一致的 PHB 组。DS 区域有明确定义的边界,边界由边界节点(boundary node)构成。边界节点连通 DS 区域和非 DS 区域(或其他 DS 区域),其主要功能为:实现传输的分类(classify)和调节(condition)机制(逻辑上分为分类器与调节器),保存流(单流或聚集流)的状态信息,根据预定的流规格对进入(或离开)区域的流进行调节,包括计量

(metering)、标记(marking)、整形(shaping)、丢弃(dropping)几个动作,使输入流(或输出流)符合预先制定的 TCA,并在包头标记 DSCP 值,分类归入行为聚集。需要注意的是,边界节点上也要实现 PHB。

针对特定的流,边界节点又分为入口节点和出口节点。入口节点必须对入域流进行调节,确保其符合 TCA 规范;出口节点可能对出域流进行调节,保证其符合与下游 DS 区域签定的 TCA。

内部节点上实现一组或若干组 PHB。处理 IP 包时根据包头的 DSCP 值选择特定的调度转发行为(外特性即 PHB)。这一过程是多对一的映射(函数),即每个 DSCP 值只能对应一个 PHB,多个 DSCP 可能对应同一 PHB。这种映射关系在一个 DS 区域内应保持一致。内部节点的处理对象是流聚集,数量有限,因而处理的时间与空间复杂度低。内部节点也可能部分或全部实现分类和调节功能以增强网络的鲁棒性。

一般 DS 区域由毗邻的属于同一网络管理机构的网络构成,如某个 ISP 的网络或者内部网。

连续的 DS 区域构成 DS 区,区内支持跨越若干区域的区分服务。区内的各个区域可能支持不同的 PHB 组,并且各自区域的 DSCP 到 PHB 的映射函数也可能不同;如果有不同的 DS 区域,则区域之间必须有 SLA 与 TCA 定义域间的调节规则,协调彼此的服务语义。域间边界节点分别对出域与入域流进行调节以保证其符合 SLA 与 TCA 的规定。

## 3.2.2　区分服务标记域与区分服务标记 DSCP

IP 包头的区分服务标记域(DS field)是 DS 区域的边界节点与内部节点间传递流聚集信息的媒介,是连接边界的传输分类和调节机制与内部 PHB 的桥梁。DS 标记域定义为原 IPv4 包头的 TOS 字节或 IPv6 包头的流类型字节(traffic class octet)的前六位[4],如图 3.2.2 所示。CU 未在区分服务体系中定义,具有其他用途,如 ECN[10]。


IPv4 TOS 字节或 IPv6 流类型字节

图 3.2.2　IP 包头的区分服务标记域

DSCP 是区分服务标记域中的具体值,用来标识数据包所属的流聚集,供数据包经过 DS 节点时选择特定的 PHB。DS 节点上 DSCP 到 PHB 的映射在具体实现中必须是可配置的。在定义 PHB 时,应同时指定对应 DSCP 的推荐值。

## 3.2.3　边界节点的传输分类与调节机制

边界节点要根据 TCA 对入域(或出域)流进行分类和调节,以保证输入(或输出)流满足 TCA 中规定的规格,并将其归入某个行为聚集、标记相应的 DSCP 值。逻辑上分为分类器(classifier)与调节器(conditioner)两个模块。如图 3.2.3 所示。

(1) 分类器遵照 TCA 中的特定规则,根据包头的某些域(如 DSCP 值或 MF 五元组)

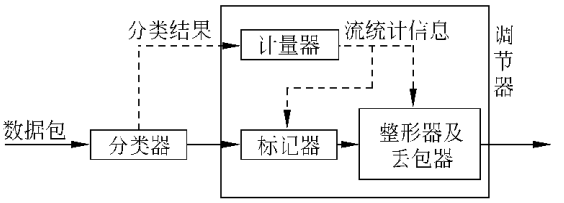



图 3.2.3 边界节点进行传输分类和调节的逻辑框图

将包划归到某一类别,然后交由相应的调节模块进一步处理。

(2) 调节器在逻辑上又分为计量器(meter)、标记器(marker)、整形器(shaper)和丢包器(dropper)。

- 计量器依据 TCA 中的流规格计量流的某些实时属性,如速率等,并将统计信息传给标记器、整形器和丢包器。
- 标记器在包头的 DS 标记域中标记适当的 DSCP,即将分组划入某个行为聚集。标记器可以将经过分类器分类后交给它处理的所有分组标记同一 DSCP 值,也可以根据计量器的统计信息将其标为同一 PHB 组内不同 PHB 对应的 DSCP 值(例如,确保服务。见下面介绍)。
- 整形器、丢包器通过延迟、丢弃等手段强制入流(或出流)符合 TCA 的流规范。

调节器的实现技术比较成熟,只要用令牌桶(token bucket)、漏斗桶(leaky bucket)等算法适当组合即可。例如,文献[11]中的一种通用调节器,不但包括了调节器的各逻辑模块,而且各模块及模块间的各种属性关系都有参数可以设定,通用性很强。通过合理设置参数,通用调节器可以实现奖赏服务(premium service,PS)、确保服务(assured service,AS)等服务需要的调节器。当然,某种服务所需要的特定调节器也可以单独实现,其优点在于简单、有效。如 PS 用令牌桶做整形、丢弃,AS 也可以用层次化的计量器与标记器来实现。总之,调节器的实现形式是多种多样的。

### 3.2.4 逐点行为 PHB、PHB 组与 PHB 组族

逐点行为 PHB 是一个 DS 节点调度转发特定流聚集这一行为的外特性描述。PHB 针对具体的流聚集,流聚集用 IP 包头的 DSCP 标识,因而实际上 PHB 是一个 DS 节点调度转发处理包头标有特定 DSCP 的 IP 包流的外部行为描述。PHB 可以用调度转发流聚集时的一些流特性参数(如延迟、丢失率)来描述。当某个 PHB 可能与其他 PHB 共存于一个节点时,还必须指出在分配资源(如缓冲区、带宽)时与其他 PHB 的相对优先级。事实上,也只有在多个流聚集竞争资源时,PHB 甚至整个 DiffServ 体系才有意义。PHB 本质上描述的就是单个节点为特定流聚集分配资源的方式;DiffServ 体系的整体资源分配策略也就通过这一个个单节点资源分配实现的。

应该注意的是,PHB 仅是外特性描述,而不涉及具体的实现机制。这类似于对象封装后的外部接口描述。PHB 的实现可以用队列调度与缓冲管理等各种算法,如优先级队列、分类队列[12]等。

多个 PHB 由于彼此关系密切(如具有按顺序排列的相对丢弃优先级)而必须同时定

义,则在实现时就构成一个 PHB 组。PHB 组是区分服务体系中的基本定义或实现模块,单个 PHB 是特殊的 PHB 组。若干 PHB 组有相似构造(即各组内的 PHB 间有相似关系),因而这些 PHB 组可以同时定义,则称其属于同一 PHB 组族。组族与组的关系类似于面向对象中类与类实例(对象)的关系,一个是抽象定义,一个是具体实例。典型的例子如 AF 组族中的 4 个独立的 AF 组。

定义具体的 PHB 是 IETF 区分服务工作组的重要工作内容。目前已标准化的 PHB 有缺省型 BE(best effort)、加速型 EF(expedited forwarding)[13]、确保型 AF(assured forwarding)[14]以及兼容 IP 优先级的类选择型 CS(class selector)4 种。此外,准尽量做好型 LBE(lower than best effort)[22]、允许丢失的加速型 EFD(expedited forwarding with dropping)[15]以及协同 PHB 组 PHB-I(interoperability PHB group)[23]也正在讨论发展中。EF 与 AF 将在 3.3 节加以讨论。下面简单介绍其他的 PHB。

(1) 缺省 PHB

BE 是相当于传统尽量做好调度转发行为的 PHB。显然,任何一个 DS 节点都应支持 BE。它的推荐 DSCP 是‘000000’。此外,在 DSCP 到 PHB 的映射表中,任何没有明确指出映射关系的 DSCP 也都映到 BE。换言之,包头 DSCP 无明确意义的 IP 包都属于缺省 PHB 对应的行为聚集,将接受通常的尽量做好服务。通常情况下,缺省 PHB 具有最低的优先级。BE 可以用如下方式实现:属于 BE 的 IP 包仅在带宽空闲未被其他流聚集使用时发送。但有一个例外是,当存在 LBE 时,因为 LBE 是比 BE 优先级更低的 PHB,为保证未支持区分服务的用户可以像往常一样使用网络,可以采用为 BE 保留少量资源的策略,以防止在高优先级聚集存在时 BE 流彻底‘饥饿’。

(2) 准尽量做好型 PHB[22]

LBE 是比缺省 PHB 优先级还要低的 PHB,其作用是在拥塞时能更有选择地丢包——LBE 比 BE 有更高的丢弃优先级——这将使普通 BE 的性能提高。LBE 可用于某种类似备份的相对不太重要的后台数据的传送;也可以用于避免降级的分组过分影响普通 BE 流的情况,这在 DiffServ 支持异质组播时很重要(见 3.4 节)。但目前 LBE 存在的必要性仍存在较大争议。

(3) 类选择 PHB

CS 是 DiffServ 向后兼容 RFC1812 中的 IP 优先级队列的产物。历史上,IPv4 TOS 字节的前三位曾作为优先级队列调度的选择标志,已被广泛实现。为保持兼容性,型如“xxx000”的 DSCP 都被保留为 CS PHB 的标记值。CS PHB 组的任何实现要保证:对应 DSCP 较大的流聚集处理优先级要高于 DSCP 较小的流聚集,即 DSCP 值较大的流在延迟、丢失率等参数上都要优于 DSCP 较小的流。

(4) 允许丢失的加速型 PHB

EFD[15]与 EF 属于同一 PHB 组。除 EF 几乎没有丢失而 EFD 允许一定限度的丢失以外,EF 与 EFD 的外特性完全相同。换句话说,EFD 的外特性是低延迟的、有一定限度的丢失。在实现上,EFD 没有自己的预留资源,仅使用 EF 的资源,因而 EFD 必须与 EF 同时实现。EFD 提出的背景是在无线移动网中提供类似 PS 的服务,但由于无线移动通信中高误码率及可能存在交接中断(hand-off)的特点[16],基本的 PS 与 EF 组合将消耗大量

资源从而不能大规模使用。EFD 及其对应的低延迟尽量做好服务 BELD(best-effort low-delay)提供了更合理的选择。EFD 还适用于对延迟敏感却允许一定丢失的多媒体实时应用。

(5) 协同 PHB 组

PHB-I[23] 的提出既有创意又可有广泛应用。其思路根据:

- PHB 描述的实质只有两点——延迟如何、丢失率如何;
- 与丢失、延迟相对应的,所有应用需求都可以归结为一定级别的重要性(importance)与紧迫性(urgency),简记为 I 级与 U 级。

因而,只要统一定义一组 PHB,包括不同 I 级与 U 级的组合,就可以满足所有的应用需求。其他的 PHB 如 EF、AF 等都可以根据其延迟与丢失特性映射到特定的 PHB-I。这样统一定义的好处在于,能够避免因分别定义不同 PHB 组可能导致的冲突。PHB-I 组包括 2 ~ 3 个 U 级、每级内 6 ~ 10 个 I 级。它的基本功能是可以成为域间协商服务提供策略的统一语言,因为各域可以很灵活地制定本域内各种 PHB 到 PHB-I 的特定映射规则。另一方面,PHB-I 的强大功能也赋予其广泛性——可以取代所有其他 PHB,成为区分服务区域内惟一的 PHB 组。

## 3.3 DiffServ 的典型服务与技术

自区分服务概念出现以来,奖赏服务 PS[2] 与确保服务 AS[17] 是讨论最为集中的两种典型服务,最初分别由 Van Jacobson 和 David D. Clark 提出,文献[2]首次将两者统一在一个框架中。正是对这两种服务的深入讨论导致了 EF 与 AF 两种 PHB 的产生。下面分别介绍 PS 和 AS。

### 3.3.1 奖赏服务 PS

奖赏服务为用户提供低延迟、低抖动、低丢失率、保证带宽的端到端或网络边界到边界的传输服务,是目前所定义的服务级别最高的区分服务种类。“三低一保证”的服务承诺使得用户可以享受类似专线的服务质量,因而奖赏服务也称为“虚拟专线”服务。由于 PS 的服务承诺针对用户流的最高速率,资源预留量也根据最高速率计算,因而 PS 也最昂贵。但 PS 并非要取代传统的 BE 服务,而是与之共存以提高网络资源的利用率——因为 PS 没有用尽的带宽可以分配给其他的流,如 BE 使用。实际上,PS 流只会占据很小一部分资源。最终结果是,ISP 的收入提高了,资源也不会闲置。

由于延迟、抖动、丢失主要由于分组在传送路途中排队所致,因而“三低一保证”实际上意味着传输流在传送路途中几乎不排队。而在路由器处出现排队的原因是在某些较短时间段内分组的入速率超过出速率(即请求速率超过处理速率)。则上述推导的最终结论是:任何时刻,在 PS 流传送道路上的任何节点处都要保证“PS 分组的入速率小于出速率”,或更进一步地,总体上的最大入速率要小于最小出速率。因此,提供这种服务要确保两点:

(1) 在传送节点处保证 PS 流有“良好定义”的最小出速率。“良好定义”意为最小出

速率不依赖于节点状态的动态变化,具体而言,即不依赖于此节点处其他流的强度。

(2) 调节 PS 流(通过整形或丢弃),以保证它在任何节点处的入速率都小于此处的最小出速率。

EF PHB 保证前者,后者由边界调节机制实现。

EF PHB 定义为一种逐点行为,它保证任何时候接受此服务的流的离开速率大于等于设定速率,而且这种保证不受其他传输流的影响。因而在与其他 PHB 共存时,EF 总是优先级最高的。若 EF 的实现机制是任意抢占式的,则必须设置 EF 流特性的上限(如最高速率、最大突发量),超过上限的分组一律丢弃,以防止恶意的 EF 流肆意侵害其他流。理论上似乎 EF 的实现不需要缓冲区,因为任何时候 EF 流的入速率总是低于出速率。但考虑到以下情况,一定的缓冲区还是必需的:

(1) 输出链路当前被其他流的分组占用;

(2) 在多个输入链路上同时到达多个 EF 包。

因而节点处缓冲区大小至少是各输入链路 MTU 的和,再考虑到 EF 低延迟的要求,缓冲区就可以取为此值。

在网络边界处,必须对 PS 流进行调节以保证其符合约定的流规格,不超过额定最高速率。这可以用令牌桶做整形与丢弃实现。

EF 初步模拟测试结果表明,EF PHB 与边界调节器的适当实现可以得到预期的"虚拟专线"服务[9, 18]。

### 3.3.2　确保服务 AS

与 PS 的相对成熟与稳定相比较,AS 目前仍处于不断改进和发展的阶段。AS 的初衷是:在网络拥塞的情况下仍能保证用户拥有一定量的预约带宽,使用户摆脱在单一尽量做好时无法把握自己实际占有带宽量的无奈窘况;着眼点是带宽与丢失率,不涉及延迟、抖动。服务原则是:无论是否拥塞,保证用户占有预约的最低限量的带宽;当网络负载较轻而有空闲资源时,用户也可以使用更多的带宽。则用户最终实际得到的带宽分为两部分,预定最小保证值以及与其他 AS 流或 BE 流竞争剩余资源获得的额外带宽。但与 PS 对带宽的严格承诺不同,AS 定位于统计性保证,这样可以提高资源利用率并降低价格,但也弱化了 AS 的质量保证。大量对 AS 的测试模拟[9, 19~21]表明,AS 的实际服务质量与诸多因素相关,较难达到量化标准,而更多的是一种优化服务(better service)。后面 3.5 节的讨论大多是针对 AS 的。

AS 实现的基本思路是:

(1) 分组进入网络时在边界节点给包作标记,预约带宽以内的流量标为 IN(in profile),超出预约带宽的流量标为 OUT(out of profile);

(2) 拥塞时包头标记决定分组的丢弃概率,OUT 的丢弃概率大于 IN,从而在一定程度上保护 IN 流;中间节点调度转发时保证源头相同的流不乱序,无论其中分组是 IN 还是 OUT。

这种思路与帧中继中的 IN/OUT 标记以及 ATM 网络中的 CLP(cell loss preference)位标记类似。在 DiffServ 体系中,前一包头标记动作由边界分类和调节器实现,后面的优先

级丢弃由 AF PHB 组(族)完成。

AF PHB 组族包含 N 个相互独立的 AF PHB 组,每组中 M 个 PHB 分属 M 个相对丢弃优先级。目前的定义为 N = 4,M = 3。根据资源预留规格,各 DS 节点为每个 AF PHB 组预留一定量资源,以保证 AF 组对应的流在任何时候都能获得预约最小带宽。DS 节点还应保证,在同一 AF 组内,低丢弃优先级流聚集的丢失率应小于高丢弃优先级流;即拥塞时,DS 节点应尽量避免低丢弃优先级流中的分组被丢弃,而更多地丢掉高丢弃优先级流中的分组。但无论属于哪个优先级,同一 AF 组内不能改变流内分组的顺序。

目前关于 AF 讨论的焦点集中在如何确定 AF 组内的相对优先级数目上。判断的依据是,多少个优先级才能有效保证各微流间的公平性,尤其是 TCP 与 UDP 流间的公平性,这将在 3.5 节专门进行讨论。此外,AF 组族定义中没有刻画各 AF 组间的关系,只说相互独立,却没有规范各组在分配资源上如何协调。

在网络边界节点,需要对 AS 流进行调节,根据某种 IN/OUT 标准将流划入若干丢弃优先级。这可以通过合理配置通用调节器[11]来实现,也可以用三色标记器[24~26]来实现。单速率三色标记器根据分组的突发程度将分组分别标为绿色、黄色与红色。非突发分组标为绿色,丢弃优先级最低;一般突发分组为黄色;超级突发分组为红色,丢弃优先级最高。双速率三色标记器(TRTCM)与时间滑动窗口三色标记器(TSWTCM)则根据实时速率标色:速率低的为绿色,高一点的为黄色,再高为红色。两者的区别在于:TRTCM 用令牌桶作计量器,TSWTCM 则用速率估测器(rate estimator)测量时间滑动窗口内的平均速率作为标记依据,并且标记策略随机化。多种三色标记器为不同配置策略提供了选择余地。

### 3.3.3 其他服务类型

在 DiffServ 体系中,服务类型与实现它的 PHB 在定义上是相互分离的。这种处理主要是基于灵活性的考虑:服务类型可能因 ISP 而异,而且发展变化较快,但实现模块 PHB 却相对保持稳定。因此 IETF 的标准化工作仅仅针对 PHB,而服务类型则是完全开放的,由各 ISP 自行确定。

同一 PHB 通过与不同的边界分类调节机制相结合,可以实现不同的服务。如 AF 也可以用来实现优先尽量做好服务(better than best-effort service,BBE)[6]、定量确保的多媒体播放服务(quantitative assured media playback service)[6]以及"奥林匹克"服务(Olympic service)[14]。

但如果新的服务类型无法用已有 PHB 实现,就需要定义新的 PHB。比如,为适应无线移动网络中误码率高并可能出现移动交接中断的特点,两种新的服务种类——移动奖赏服务(mobile premium service)与低延迟尽量做好服务 BELD[15]已被提出。前者仍可以用 EF 实现,后者却不能。因此专门定义了 EFD PHB 来实现 BELD。

## 3.4 DiffServ 网络中的组播问题

传统的 IP 组播模型主要包括两部分:主机组模型(host group model)和组播路由协议。在主机组模型中,一组主机仅由一个组播组地址决定,通过组地址来进行服务的订阅和调度转发。组播路由算法用来维持状态的数量和维护组播树。但是,传统 IP 组播存在

的地址分配问题、对于目的节点的无知性以及可扩展性问题都严重阻碍了组播在广域范围的推广应用。这促使人们对其他组播机制进行研究。一方面，关于主机组模型，最近有人提出对其进行改进以解决这些问题[27，28]；另一方面，近些年来人们在组播机制的算法和协议方面做了许多努力。设计和实现提供 QoS 保证的组播路由算法一直是人们追求的目标，也是目前研究的热点。目前已经提出了许多非常好的算法[29～32]，其中一些已经在网络中得到广泛的应用，如最著名的 Bellman-Ford 算法和 Dijkstra 算法。

随着 DiffServ 网络研究的开展，DiffServ 区域的组播问题也亟待解决。实际的组播应用需要 DiffServ 支持组播以使其同样享受 DiffServ 带来的好处。但 DiffServ 已有的框架、结构都是基于单一传播（unicast）的，所以加入组播必然会出现一些问题[28]。而且，对于含有 DiffServ 区的端到端 IntServ 网络来说，DiffServ 网络中的组播问题较 IntServ 网络要复杂得多。因为在 DiffServ 区的外部沿着组播树 IntServ 路由器都为每个流存储了状态。

下面讨论 DiffServ 网络区可能出现的组播问题及其解决方案，但只是一点到多点（point-to-multipoint），而非更一般的多点到多点（multipoint-to-multipoint）。

### 3.4.1　DiffServ 网络支持组播存在的问题

1. 被忽视的预留子树 NRS（neglected reservation subtree）问题

通常，DiffServ 的资源必须先预留再使用。在 DiffServ 网络区，当一个新的接收者加入 IP 组播组时，相应的组播路由协议，如，PIM-SM[33]，PIM-DM[34] 或 DVMRP[35]，负责将该组的组播树扩展出新的子树一直延伸到新成员。IP 组播的分组复制通常由新子树与原树接合点的路由进程完成，它将 DS 域的内容拷贝到每个复制的 IP 分组的报头。这样，复制的分组会获得与原来分组完全相同的 DSCP 值，并且接受与该组播组的输入分组同样的调度转发处理（如，分类、传输调节、排队等）及区分服务（如 PS，AS）。而这并没有经过接纳控制、资源预留的必要过程。由于额外的资源被绕过区域管理（如 BB）的新增接收节点所消耗，其他已正确预留资源的接收者的服务质量将受到负面影响甚至遭到破坏。

问题的关键在于：负责复制的接合点无法检测输出链路（接口）上的复制的分组流是否已经进行了资源预留，因为在内部节点和边界节点（除了第 1 跳节点以外）无法得到特定流的传输规格（profile）。

接合点的位置通常有两种情况：

（1）接合点正好是 DiffServ 区域的出口（egress）边界路由器，则拷贝后的新增节点的分组会与享受同级服务的合法分组争夺资源。由于新增组播流使所需的资源超过了原来预定的数量，所以出口边界路由器的管制（policing）单元将对分组进行丢弃直到传输聚集符合传输协议为止。但是由于缺乏分类功能，路由器无法识别分组是否经过预留，使丢包具有任意性，导致对预定的流失去了任何服务保证。

（2）接合点在 DiffServ 区域内部，由于 DiffServ 区域内部的路由器没有计量（metering）和管制（policing）功能，所以无法识别超额的传输而导致对新增的组播流一味调度转发。这使得新增加的未经预留的高级服务流能够与预留的聚集享受同样的服务，结果抢占了原来享受低一级服务的合法传输的资源。

这些问题的根源在于新成员的传输是绕过了接纳控制和资源预留的管理框架而直接享用了服务,或者具体地说,是 DSCP 的拷贝造成了非法享用服务的后果。

2. 异构组播组问题

同一组内的不同接收节点可能希望获得不同的服务质量。例如,有些节点仅仅想要基本的 BE 服务,而其他一些节点则希望获得更高质量的服务。对服务质量参数要求的不同更增加了问题的复杂性。更严重的是,不同的服务种类可能没有可比性而无法取其较好者,使得可能在传送树(delivery tree)的某些链路上需要同时支持这些不同的服务。

另外,因为任意一个新增接收节点都可以被嫁接到当前组播传送树的任何节点,包括内部路由器节点,那么在接合点复制分组时,必须根据接收端的 QoS 要求设置 DSCP,这与 DiffServ 区域内部节点不保留传输规格是矛盾的。

3. 发送方任意动态改变

对于组播传输,接收组经常是动态改变的。组播允许组内任意成员作为发送者(甚至不在组内的任意主机也可以作为发送者),而 DiffServ 是单向结构,这意味着如果有多个发送者同时发包的话,则其资源必须分别预留。

### 3.4.2 DiffServ 网络中支持组播的方案

1. NRS 的解决方案

要解决 NRS 问题,就要防止任何未经资源预留就享用较高级服务的情况。

当一个新的接收者要求加入组播组时,如果这两个树的接合点属于 DiffServ 区域,则该接合点把发向新成员的 DS 编码进行转换,使之对应的 PHB 比默认的 PHB 提供的服务级别还低,即比 BE 优先级还低的 LBE。这样可以制约新的传输流即使对 BE 服务的资源也不可以抢占,维护了公平性。只有当新扩展的子树上资源预留的请求被管理实体(如 BB)认可之后,新成员才可以享受较高级别的服务。

上述方案需要子树生成点在调度转发复制分组时知晓每个出口的传输所属的聚集,也即其 DSCP。解决的方法是在每个组播的路由表条目中增加一个包含 DS 域的字节,为每个输出链路增加一个 DSCP 值。这几乎没有额外的代价,复制时从组播路由表读取 DSCP 即可。

另外,还必须辅以一定的管理机制。对级别高于 BE 的服务(如 PS),必须在 DiffServ 网络区进行接纳控制和资源预留。这需要建立和更新区域边界节点上的传输规格,实现动态分配资源所需的信令机制和自动的接纳控制过程。

总之,只有在一定的管理框架下,在组播子树上预留过资源的接收者才能获得较好的区分服务,否则只能得到比 BE 还差的 LBE 服务。

2. 异构组播组的解决方案

这里仅针对不同的服务种类、不同参数的服务来讨论。解决方案与上面 NSR 问题的

情况相同，依靠在组播路由表中存储 DSCP，并辅以管理措施。在没有预约的情况下，接收者仅允许获得 LBE 服务，所以在一个组播组中至少可能有两种服务存在。这样，在不要求 QoS 的情况下，任何接收者都可以加入到组播会话，这在有些情况下也是有意义的[28]。当两种服务类型可以比较优劣排序时，在上游链路中只需要传送较好的即可。

3. 任意发送方改变的对策

当每个发送者要享用某种好的 DiffServ 服务质量时必须事先预留资源，而无论组内是否有其他发送者已经预留了同种服务的资源。对于只需要 BE 服务的参加者可以在任何时间向组发送分组而无须任何其他机制。

总之，上述问题主要是由于 DiffServ 体系结构的简化造成的。可以通过为单一传播和组播使用不同的 DS 编码值，即在组播路由表的每个输出链路条目中加入一项 DSCP，并辅以一定管理机制加以解决，这种方案实现简单，同时又保持了 DiffServ 良好的可扩展性。解决 DiffServ 网络区支持组播存在的固有缺陷，并最终实现在 DiffServ 网络区支持具有端到端 QoS 保证的组播传输，是目前研究的难点问题。

## 3.5　DiffServ 中带宽分配的公平性问题

在资源共享环境中，一定会有各共享者之间的公平性问题。具体到 DiffServ，在区域边界微流将聚合为流聚集，之后在区域内 PHB 的处理对象是流聚集而非微流，因而同一流聚集内的各微流实质上在共享预留资源。DiffServ 中的公平性指的就是属于同一流聚集的各微流能享受同等待遇，包括：

（1）资源总量充足时各微流能充分享用其预约资源，达到预期性能；

（2）有额外资源并允许竞争时各微流能平均分配或按比例分配额外资源；

（3）资源总量不足时，各微流能按预约资源比例获得相应的降级服务。

影响公平性的因素有如下两方面：

（1）各微流特性不同，包括突发程度、是否有末端拥塞控制机制、流量大小、回路响应时间（round trip time）、连接时间长短等[36]。一般情况下，突发程度较大、末端有拥塞控制机制、流量大、回路响应时间长、连接时间短的流在带宽竞争中处于弱势[9，19～21]。

（2）服务实现机制如何，包括传输过程中的各环节——边界分类调节、内部 PHB 以及是否有反馈控制等。

公平性问题的研究目标是，改进服务实现机制，消除各种流特性差异对公平性的影响。下面针对具体问题分别讨论。

### 3.5.1　适应流与非适应流共享 AF 时的公平性

适应流的含义为，末端系统实现了拥塞控制机制，能根据网络的拥塞情况自动调节发送速率，如 TCP 流 相反地，没有任何末端拥塞控制机制的流称为非适应流，如简单 UDP 流。当适应流与非适应流共处同一 AF 组时，由于拥塞时适应流会降低发送速率而非适应流却保持原速率发送，因此将导致非适应流压制适应流并占有较多带宽，而适应流却无

法获得公平待遇[19]。此外,当具有不同拥塞控制算法的 TCP 流以及具有特定拥塞控制算法的 RTP 流共处同一 AF 组时,也将面临同样问题。

一种解决思路是将适应流与非适应流分配于 AF 组内的不同丢弃优先级。AF 组目前的定义是每组内有三个丢弃优先级,因而可以令适应流的优先级相对高于非适应流。例如,适应流 IN 部分属于丢弃优先级 1,OUT 部分属于 2;而非适应流 IN 部分属于丢弃优先级 2,OUT 部分属于 3。这样相当于对适应流有一种政策倾斜,以抵消非适应流的优势。这种方法面临的主要问题是,多少个优先级是必要的。各种模拟分析的结果不同:有的认为两个优先级就够了,只要合理配置系统参数即可[37];有的认为两个优先级不够,三个是必要的[38];还有的认为三个优先级也不够[39]。导致结果千差万别的原因是模拟条件、环境不尽相同。

这一方法也可能带来新的问题——非适应流的公平性会受到损害,所谓矫枉过正。而多媒体应用的发展已使得以 UDP 包为发送单元的 RTP 流在 Internet 中的流量渐增,RTP 流的公平性无疑也应得到保证。最终方法可能是将适应流与非适应流直接划归不同的 AF 组,两厢相安。

### 3.5.2 Web 流的公平待遇

目前 Web 流在 Internet 总流量中所占的比例已超过 50%[36],DiffServ 是否能为 Web 流提供有效服务将影响其商业前途。Web 流的特点是连接时间短促,突发性强。连接时间短,则拥塞控制窗口在大部分时间内处于慢启动阶段,平均窗口小。小窗口连接对连续丢包更敏感,甚至连续 1~2 个丢包也会大大抑制发送速率。高突发性更增加了 Web 流竞争带宽的劣势。不但因为边界调节机制一般会限制突发程度,而且内部实现 PHB 的缓冲管理也可能会丢弃更多突发流中的分组,导致突发流的流量无法达到预约带宽[19]。

提高 Web 流的竞争实力有若干途径,其中之一是改变末端拥塞控制技术,如 TCP 的 NewReno。再有就是在传输路径中尽量避免个别流的连续丢包,并改善突发流的传送质量。在 DiffServ 体系中,主要从边界调节与 PHB 实现两方面入手。

(1)在边界节点,调节模块应充分考虑 TCP 拥塞控制机制的特点,拥塞时尽量在引起拥塞的各连接之间均匀分布丢弃概率,避免个别连接出现连续丢包,特别是保护小窗口连接。比如可以将不同微流的分组交叉排列[40];或者在边界节点适当统计微流的实时状态,并根据某种算法做随机化的公平标记[41,42]。对于突发流,可以在做其他调节之前先用速率自适应的整形器减小输入流的突发程度[43],使流中更多的分组可以通过调节进入优先级较高的流聚集,从而提高整个流的吞吐量。

(2)PHB 的实现一般要消除长期拥塞,但要允许短时间内的突发。前者通过丢包抑制,后者可以用队列缓冲。为保证突发流获得公平待遇,以消除长期拥塞为目的的丢包处理不应对突发流中的分组有歧视;或者说,分组的突发程度不应影响其丢弃概率。实现方法是,使用某种长期指标而非瞬时指标作为判断拥塞的依据,并且丢弃动作应随机化;例如随机尽早丢弃(RED)算法[46]。这样随机丢弃的好处还可以保持同一聚集内的不同微流间的公平性,避免个别流的连续丢弃。模拟结果显示,合理配置的 RED 实现可以很好地提高 Web 流的性能[20,21]。

以上处理手段的效果都很不错,但随机化、均匀化是更好的选择。因为事先整形的处理改变了微流的自然特性,可能在延迟、抖动等方面造成新的影响。

### 3.5.3　通用的解决办法

以上讨论的解决方案大部分有明确的针对性,一般只能避免某些流特性差异对公平性的影响。实际上,流特性差异的存在是必然的,而且各种差异的结果也是相同的——各微流竞争带宽的能力有强有弱。另一方面,目前 DiffServ 体系结构中边界调节机制与内部 PHB 实现是相对分离的,微流通过边界调节进入区域后是作为流聚集中的一员存在的,而 PHB 的服务对象是流聚集,不会对微流做任何单独的处理。因而在区域内部节点处,聚集内各微流实际上在共享预留资源,拥塞时必然发生自由竞争,争抢带宽能力弱的微流将遭受更多冲击。公平性遭到破坏的两个起因是:各微流的带宽竞争能力不同 ,拥塞导致了自由竞争的发生。因而,只要保证区域内部不发生拥塞,就能保证公平性。因为没有拥塞就意味着没有丢包 没有丢包的威胁,流聚集内的各微流就能和平相处,不会发生自由竞争。

避免内部不发生拥塞的方法是在边界节点严格控制进入区域的总流量。这就存在一个问题——边界节点如何判断进入区域的总流量是否合适?总流量的上限是多少才能避免内部拥塞?一种办法就是加入动态反馈机制。比如增加内部节点到边界的反馈渠道[44]:内部节点收集周边情况,并通过某种通信方式将此情况通报边界节点,边界节点根据反馈来的内部负载信息公平计算各微流的适宜速率,并据此动态调整各微流的流规格,从而调控进入区域的总流量,从根本上避免区域内发生拥塞。另外也有一种沿流聚集传送路径的边界到边界的动态反馈机制[45],针对具体流聚集,将流聚集出口节点的速率反馈给入口节点,入口节点根据此速率计算各微流的适宜速率并做调控,最终系统到达平衡状态时将有流聚集的出口速率等于入口速率,也就是说内部没有拥塞。

反馈机制的好处在于,它不但可以提高微流间的公平性,而且也能提高资源的利用率。因为它将拥塞排除在 DiffServ 区域以外,可以使区域内的资源利用最大化,避免因局部拥塞导致的全局性浪费。但它也可能存在可扩展性问题,毕竟动态反馈需要增加统计与相互通信,这种代价必须足够小才能保持 DiffServ 体系结构的可扩展优势。

## 参考文献


1　林闯,单志广,盛立杰,吴建平. 区分服务及其几个热点问题的研究,计算机学报,2000,23(4):419~433

2　Nichols K, Jacobson V, Zhang L, et al. A two-bit differentiated services architecture for the Internet. IETF RFC 2638, July 1999

3　Nichols K, Blake S, Baker F, et al. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. IETF Internet RFC 2474, December 1998

4　Grossman D. New Terminology for DiffServ. IETF Internet Draft〈draft-ietf-diffserv-new-terms-01.txt〉, October 1999

5　Carlson M, Weiss W, Blake S, et al. An architecture for differentiated services. IETF Internet RFC 2475,

December 1998

6 Bernet Y, Binder J, Blake S, et al. A framework for differentiated services. IETF Internet Draft 〈draft-ietf-diffserv-framework-02. txt〉, February 1999

7 Bernet Y, Smith A, Blake S. A conceptual model for diffServ routers. IETF Internet Draft 〈draft-ietf-diffserv-model-01. txt〉, October 1999.

8 Almesberger W, Salim J H, Kuznetsov A. Differentiated services on Linux. IETF Internet Draft 〈draft-almesberger-wajhak-diffserv-linux-01. txt〉, June 1999

9 Bless R, Wehrle K. Evaluation of differentiated services using an implementation under Linux. In: Proc of the 7th IFIP Int'l Workshop on Quality of Service (IWQOS '99). London, June 1999, available at: http://www. telematik. informatik. uni-karlsruhe. de/forschung/diffserv/KIDS/

10 Ramakrishan K, Floyd S. A proposal to add explicit congestion notification (ECN) to IP. IETF Internet RFC 2481, January 1999

11 Lin L, Lo J, Ou F. A generic traffic conditioner. IETF Internet Draft 〈draft-lin-diffserv-gtc-01. txt〉, August 1999

12 Floyd S, Van Jacobson. Link-sharing and resource management models for packet networks. IEEE/ACM Trans on Networking, 1995, 3(4): 365 ~ 386

13 Jacobson V, Nichols K, Poduri K. An expedited forwarding PHB. IETF Internet RFC 2598, June 1999

14 Heinanen J, Baker F, Weiss W, et al. Assured forwarding PHB group. IETF Internet RFC 2597, June 1999

15 Diederich J, Zitterbart M. An expedited forwarding with dropping PHB. IETF Internet Draft 〈draft-dieder-diffserv-phb-efd-00. txt〉, October 15, 1999

16 Li B, Lin Chuang, Chanson S T. Analysis of a hybrid cutoff priority scheme for multiple classes of traffic in multimedia wireless networks. ACM Journal of Wireless Networks, 1998, 4(4): 279 ~ 290

17 Clark D D, Fang W J. Explicit allocation of best-effort packet delivery service. IEEE/ACM Trans on Networking, 1998, 6(4)

18 Hou M, Hussein, Mouftah T. Performance evaluation of premium service. IETF Internet Draft 〈draft-hou-diffserv-premium-eval-00. txt〉, June 1999

19 Ibanez J, Nichols K. Preliminary simulation evaluation of an assured service. IETF Internet Draft 〈draft-ibanez-diffserv-assured-eval-00. txt〉, August 1998

20 Jose Ferreira de Rezende. Assured service evaluation. available at http://www. gta. ufrj. br/ftp/gta/TechReports/Reze99b. ps. gz

21 Kim H, Leland W, Thomson S. Evaluation of bandwidth assurance service using RED for Internet service differentiation. Preprint, available at ftp://ftp. bellcore. com/pub/world/hkim/assured. ps. Z

22 Bless R, Wehrle K. A lower than best-effort per-hop behavior. IETF Internet Draft 〈 draft-bless-diffserv-lbe-phb-00. txt〉, September 1999

23 Kilkki K, Ruutu J. Interoperability PHB group. IETF Internet Draft 〈draft-kilkki-diffserv-interoperability-00. txt〉, October 1999

24 Heinanen J, Guerin R. A single rate three color marker. IETF RFC 2697, September 1999

25 Heinanen J, Guerin R. A two rate three color marker. IETF RFC 2698, September 1999

26 Fang W, Seddigh N, Nandy B. A time sliding window three colour marker (TSWTCM). IETF Internet Draft 〈draft-fang-diffserv-tc-tswtcm-00. txt〉, October 1999

27 Perlman R. Simple multicast: A design for simple, low-overhead multicast. IETF Internet Draft 〈draft-

perlman-simple-multicast-02. txt〉, February 1999

28 Bless R, Wehrle K. IP multicast in differentiated services networks. IETF Internet Draft 〈draft-bless-diffserv-multicast-00. txt〉, September 1999

29 Hong S-P, Lee H, Park B H. An efficient multicast routing algorithm for delay-sensitive applications with dynamic membership. In: IEEE INFOCOM '98. San Francisco, California, March 1998. 1433 ~ 1439

30 Lin H-C, Lai S-C. VTDM—A dynamic multicast routing algorithm. In: IEEE INFOCOM '98. San Francisco, California, March 1998. 1426 ~ 1432

31 Mithal S. Bounds on end-to-end performance via greedy, multi-path routing in integrated service networks. In: IEEE INFOCOM '98. San Francisco, California, March 1998. 19 ~ 26

32 Rouskas G N, Baldine I. Multicast routing with end-to-end delay and delay variation constraints. IEEE JSAS, 1997, 15(3): 346 ~ 356

33 Cain B. Connecting multicast domains. IETF Internet Draft 〈draft-cain-mcast-connect-00. txt〉, October 28, 1999

34 Deering S, et al. Protocol independent multicast Version 2 dense mode specification. IETF Internet Draft 〈draft-ietf-pim-v2-dm-01. txt〉, November 1998

35 Pusateri T. Distance vector multicast routing protocol. IETF Internet Draft 〈draft-ietf-idmr-dvmrp-v3-09. txt〉, September 1999

36 Thompson K, Miller G J, Wilder R. Wide-area Internet traffic patterns and characteristics. IEEE Network, 1997, 10 ~ 23

37 Goyal M, Durresi A, Jain R. Effect of number of drop precedences in assured forwarding. IETF Internet Draft 〈draft-goyal-dpstdy-diffserv-02. txt〉, June 1999

38 Elloumi O, De Cnodder S, Pauwels K. Usefulness of three drop precedences in assured forwarding service. IETF Internet Draft 〈draft-elloumi-diffserv-threevstwo-00. txt〉, July 1999

39 Seddigh N, Nandy B, Pieda P. Study of TCP and UDP interaction for the AF PHB. IETF Internet Draft 〈draft-nsbnpp-diffserv-udptcpaf-01. txt〉, September 1999

40 Azeem F, Rao A, Lu X, et al. TCP-friendly traffic conditioners for differentiated services. IETF Internet Draft 〈draft-azeem-tcpfriendly-diffserv-00. txt〉, March 1999

41 Yeom Ikjun, Narasimha RAL. Impact of marking strategy on aggregated flows in a differentiated services network. In: Proc of the 7th IFIP Int 'l Workshop on Quality of Service (IWQOS '99), London, June 1999, published by IEEE, available at http://ee. tamu. edu/ ~ reddy/papers/index. html

42 Kim Hyogon. A fair marker. IETF Internet Draft 〈draft-kim-fairmarker-diffserv-00. txt〉, April 1999

43 Bonaventure O, Cnodder De S. A rate adaptive shaper for differentiated services. IETF Internet Draft 〈draft-bonaventure-diffserv-rashaper-01. txt〉, October 1999

44 Chow H, Leon-Garcia A. A feedback control extension to differentiated services. IETF Internet Draft 〈draft-chow-diffserv-fbctrl-00. txt〉, March 1999

45 Kalyanaraman S, Harrison D, Arora S, et al. A one-bit feedback enhanced differentiated services architecture. IETF Internet Draft 〈draft-shivkuma-ecn-diffserv-00. txt〉, March 1998

46 Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. IEEE/ACM Trans on Networking, 1993, 1(4): 397 ~ 413

# 第4章

# DiffServ 与 IntServ 相结合的端到端 QoS 提供机制

IP 网络的 QoS 研究导致了 IntServ 和 DiffServ 两种不同的 Internet QoS 体系结构，它们的设计与实现有着完全不同的目标和原则。表 4.1 对此进行了总结。

表 4.1　IntServ 和 DiffServ 体系结构的比较

| 对　比　项 | IntServ | DiffServ |
|---|---|---|
| 服务区分的粒度 | 单个流 | 流聚集 |
| 路由器中的状态维护 | 基于每个流 | 基于每个流聚集 |
| 分组的分类依据 | 多个分组头部域（即五元组） | IP 头部的 DS 字节 |
| 服务区分的类型 | 确定性的或统计上的保证 | 绝对的或相对的保证 |
| 接纳控制 | 需要 | 仅是绝对的区分需要 |
| 信令协议 | 需要（RSVP） | 相对保证的方案不需要；绝对保证的方案需要半静态的预留或中介代理 |
| 协调 | 端到端 | 局部的（逐跳） |
| 可扩展性 | 受流的数量限制 | 受服务的种类限制 |
| 网络计费 | 基于流的特性和 QoS 需求 | 基于服务类的使用 |
| 网络管理 | 近似于电路交换网络 | 近似于现有的 IP 网络 |
| 域间配置 | 需要多边协议 | 需要双边协议 |

但这两种 IP 网络的 QoS 标准都不能完全满足需要，如前所述，IntServ 和 DiffServ 这两种 Internet QoS 标准各有自己的长处和局限[1]，但都不能彻底实现整个网络的端到端 QoS。为此，可考虑将 IntServ，RSVP 和 DiffServ 看作互相补充的技术，将其结合，互相协同，共同实现端到端的 QoS 提供机制。最终达到既能提供类似状态相关网络的强有力的服务，又能实现与状态无关网络近似的可扩展性和鲁棒性。目前，这两种技术的结合仍然是一个开放的研究课题。从应用的观点，这种结合将极大地促进诸如 IP 电话、视频点播等多媒体应用以及各类非多媒体任务紧要应用的发展。

IntServ 体系结构提供了一种在异类网络元素之上为应用提供端到端 QoS 的方法。一般来讲，网络元素可以是单独的节点（如路由器）或链路，更复杂的实体（比如 ATM 云

或 802.3 网络)也可以从功能上视为网络元素。在这种意义下,DiffServ 网络(或“网络云”)也可以视为更大的 IntServ 网络中的一种网络元素。文献[2]描述了一种在 DiffServ 网络之上实施 IntServ 的框架,描述了在 IntServ 体系结构下 DiffServ 网络支持端到端 QoS 的方法。

在该框架中,端到端的、定量的 QoS 是通过在含有一个或多个 DiffServ 区的端到端网络中应用 IntServ 模型来提供的。为了优化资源的分配和支持接纳控制,DiffServ 区可以(但并不绝对要求)参加端到端的 RSVP 信令过程。从 IntServ 的角度来看,网络中的 DiffServ 区被视为连接 IntServ 路由器和主机的虚链路。该框架的目标是实现 IntServ 区与 DiffServ 区无缝的相互操作,网络管理员可以自由选择网络中的哪个区作为 DiffServ 区。

## 4.1　DiffServ 网络区支持 IntServ/RSVP 的意义

与 IntServ/RSVP 协同工作对于 DiffServ 网络区的意义主要表现在以下三个方面。

1. 在 DiffServ 区实现基于资源的接纳控制

在 IntServ 网络中,量化的 QoS 应用使用显式信令 RSVP 从网络请求资源,网络做出接受或拒绝的响应,这称为“显式接纳控制”。显式接纳控制有利于网络资源的优化使用。而对于只提供聚集传输控制而无信令机制的 DiffServ 网络区,其接纳控制是以隐式的方式通过网络元素上的管制参数实现的。例如,DiffServ 区入口的某个网络元素对 EF DSCP 只允许接受 50Kbps 的传输。隐式接纳控制能够在某种程度上对网络起到保护作用,但效率较低。而且隐式接纳控制会破坏端到端显式接纳控制的有效性[3]。如果在 DiffServ 网络中采用显式接纳控制(如利用 RSVP),则能够保证对接纳的传输流实现资源预留(虽然以损害未被接纳的流为代价)。因此,为 DiffServ 网络区指定一个支持 IntServ 的接纳控制代理可以优化资源的使用,提高 DiffServ 区对于定量 QoS 应用的服务质量。

2. 在 DiffServ 区实现基于策略的接纳控制

在使用 RSVP 的网络区,资源请求可以被识别 RSVP 的网络元素截取,并按照策略数据库的策略进行检查。因为资源请求标识了其所代表的用户和应用,所以在进行接纳控制时,网络元素可以考虑基于每个用户或每个应用的策略。因此在 DiffServ 网络区采用 RSVP 接纳控制代理,可以在决定资源分配时采用针对特定客户的策略,为特定的用户和应用有效地分配资源。否则,在没有 RSVP 信令的 DiffServ 网络区,策略典型地只能作用于发起传输的 DiffServ 客户网络,而不是客户网络中的某个传输发起用户或应用。

3. 传输识别及分类中的辅助作用

在 DiffServ 网络区内部,传输的资源分配基于每个 IP 分组头部标识的 DSCP 值,为此必须正确标记 DSCP。这里介绍两种实现机制:主机标记和路由器标记。主机标记要求主机知道网络如何翻译 DSCP。这类信息可以配置于每台主机,但却加重了管理负担。一种较好的方案是:主机使用显式信令协议(如 RSVP)通过询问网络来获取。文献[4]对

这种方案进行了描述。路由器标记要求必须在路由器配置 MF 分类准则。这可以由主机操作系统通过请求动态完成,或由手工配置或自动脚本静态完成。然而静态配置难度很大,一种更好的选择是允许主机操作系统代表用户和应用通过信令将分类准则发送给路由器,而 RSVP 正是理想的信令选择。

## 4.2 DiffServ 网络区支持端到端 IntServ 的实现框架

在此框架中,Internet 使用 IntServ 体系结构来为应用提供端到端的 QoS。整个网络是 IntServ 节点(采用基于 MF 的分类和基于流的传输控制)和 DiffServ 区(采用聚集传输控制)的结合体。

该框架的参考网络如图 4.2.1 所示。


图 4.2.1 DiffServ 网络区支持端到端 IntServ 服务的参考网络框架

如图 4.2.1 所示,在支持 IntServ 的端到端网络中央含有一个 DiffServ 区,它包含许多连接的路由器,至少其中的一部分提供聚集传输控制。DiffServ 区之外的区域(非区分服务区)也包含许多路由器和与之相连的主机,至少其中的一部分支持 IntServ 体系结构。为了简化起见,该参考网络只考虑一个 QoS 发送者 Tx 与一个 QoS 接收者 Rx 通过网络进行通信。邻近 DiffServ 区的边缘路由器(ER1,ER2)与 DiffServ 区内部的边界路由器(BR1, BR2)通过接口直接相连。

发送方和接收方主机都使用 RSVP 来传达主机应用的定量 QoS 请求。主机操作系统的 QoS 进程代表应用生成 RSVP 信令。RSVP 消息在主机 Tx 和 Rx 之间端到端地传播以支持 DiffServ 区外部的 RSVP 预留,端到端的 RSVP 信令至少应被透明地传过 DiffServ 区。依赖于特定的实现,这些消息可能不会被 DiffServ 区的路由器处理,也可能被 DiffServ 区内的一些路由器甚至所有的路由器处理。

边界路由器 ER1,ER2 和 BR1,BR2 的功能都依赖于该框架的特定实现。在 DiffServ 网络区不识别 RSVP 的情况下,边界路由器 ER1 和 ER2 作为 DiffServ 区的接纳控制代理。它们处理来自 Tx 和 Rx 的信令消息,根据 DiffServ 网络区内部的资源信息和客户定义的策略来实施接纳控制。而 DiffServ 区内的边界路由器 BR1 和 BR2 只作为纯粹的 DiffServ 路由器,惟一的任务就是基于 DSCP 描述的服务级别和与客户协商的协议对传输实施聚集传输控制。在 DiffServ 网络区能够识别 RSVP 的情况下,边界路由器 ER1 和 ER2 根据当地的资源情况和客户定义的策略实施接纳控制,而边界路由器 BR1 和 BR2 参加 RSVP 信令过程并作为 DiffServ 网络区的接纳控制代理。

DiffServ 网络区支持聚集传输控制，而不实现 MF 分类。依赖于该框架的特定实现，可能 DiffServ 区内部的一些路由器支持 RSVP，则能够实现基于流的信令和接纳控制。如果 DiffServ 区的设备不支持 RSVP，它们将透明地传递 RSVP 消息而不会对传输性能产生什么影响[5]。

DiffServ 区外部的网络包括 IntServ 主机和其他网络元素，如路由器和各种不同类型的网络（如 802 或 ATM 网络等）。这些网络元素可能支持 IntServ，如果不支持，它们也应将 RSVP 消息不受妨碍地进行传送。另外，也不排除 DiffServ 网络区外边的路由器可以为通过它的传输子集提供聚集传输控制。

## 4.3　支持端到端 IntServ 的 DiffServ 网络区资源管理方案

在 DiffServ 网络上实施 IntServ 框架大体上有两种基本的实现形式：

（1）网络中的 DiffServ 区以静态方式提供内部的资源管理，区内不含有能够识别 RSVP 的设备；

（2）网络中的 DiffServ 区以动态方式提供内部的资源管理，DiffServ 区内部某些选定的设备参加 RSVP 信令过程。

具体而言，根据端到端 IntServ 传输流的需要，DiffServ 网络区内的资源管理可以有多种实现方案。

### 4.3.1　静态资源管理方案

在此类方案中，DiffServ 网络区的客户和网络所有者之间协商建立一个静态的契约——服务层描述 SLS（service layer specification），保证在每个标准的 DiffServ 服务级别上向客户提供应有的传输能力。DiffServ 区及其外部的网络元素之间没有信令，它们之间的 SLS 协商是惟一的关于资源可利用性信息的交换形式。边界路由器 ER1 作为 DiffServ 网络区的接纳控制代理，配置了 SLS 所表示的信息。这种方案的缺点首先是灵活性差，不容易支持 SLS 的动态改变，因为 SLS 每次改变都需要重新配置 ER1。另外，DiffServ 网络区的资源也难于有效地利用，因为接纳控制无法充分反映 DiffServ 区中常受冲击的路径上的资源可利用性。

### 4.3.2　使用 RSVP 的动态资源管理方案

在此方案中，DiffServ 网络区的边界路由器 BR1 支持 RSVP，此外，在 DiffServ 网络区内的其他路由器也可能支持 RSVP。但值得指出的是：虽然这些路由器参加某种形式的 RSVP 信令，但它们仍使用 IP 分组头部的 DSCP 值，对聚集传输流进行识别、分类和调度（聚集传输控制），而不像标准的 IntServ/RSVP 路由器使用基于流的 MF 分类准则。当一个新流要加入到行为聚集中时，就使用动态提供机制和显式信令进行接纳控制[3]。此时，可以使用 RSVP 信令将流的描述和期望的 DSCP 传给 DiffServ 区的路由器。可以说，DiffServ 区路由器的控制平面是 RSVP 而数据平面仍是 DiffServ。这种方案既充分利用了 RSVP 信令的优越性，又保持了 DiffServ 的可扩展性。

在 DiffServ 网络区支持 RSVP,接纳控制代理就是 DiffServ 网络中的一部分,可以通过 RSVP 将 DiffServ 网络区可用资源的改变通知给 DiffServ 网络外部的 IntServ 节点。这样不但可以提高 DiffServ 区内部资源使用的效率,而且提高了接纳控制的可信度。后者是由于接纳控制能够与常受冲击路径的资源可利用性建立联系,因而称为拓扑性(topology aware)的接纳控制。此方案的另一个好处就是可以根据 DiffServ 区外部的资源请求实现 DiffServ 网络区内部资源提供上的改变(如,给路由器中的 EF 队列分配更多或更少的带宽)。

在具体实现上,DiffServ 网络区内部可以使用多种不同的机制来支持动态方案和拓扑性接纳控制,包括聚集的 RSVP 方案、面向单个流的 RSVP 方案和使用带宽中介服务器(bandwidth broker,BB)等。

1. 聚集 RSVP 方案

许多草案都提出使用扩展 RSVP 的机制,即聚集 RSVP[5~9]。在 DiffServ 网络区的边界之间为聚集流预留资源,而不对来自 DiffServ 区以外节点的、面向单个流的 RSVP 请求进行接纳控制。聚集预留的量可以(或不必)动态地调整。这种方案的优点是:它为 DiffServ 网络区提供了动态的、拓扑性的接纳控制,而无须支持面向单个流的 RSVP 信令处理。由于每个区域都有可能选择其独特的管理资源机制,所以使用聚集 RSVP 进行资源管理的 DiffServ 网络区最适合于单一的行政管理区域。

2. 面向单个流的 RSVP 方案

在此方案中,DiffServ 网络区内的路由器对发起于 DiffServ 区之外 IntServ 节点的、面向单个流的标准 RSVP 信令请求进行响应。这种方案同样具有上述聚集 RSVP 方案的优点,但不支持聚集 RSVP。由于使用面向单个流的接纳控制,资源的使用具有更高的效率。然而,这种方案对于 DiffServ 网络区内的 RSVP 信令资源的需求量也比聚集 RSVP 方案大得多。

值得指出的是,在一个单独的 DiffServ 区内,面向单个流的 RSVP 与聚集 RSVP 并不相互排斥。可以在 DiffServ 区的边界使用面向单个流的 RSVP,而只在 DiffServ 区内部的一些核心区使用聚集 RSVP。

在以上方案的具体实现中,网络管理员必须决定在 DiffServ 网络中参加 RSVP 信令的路由器数量。一种极端的情况是只有边界路由器参加 RSVP 信令,那么必须为 DiffServ 网络区提供过度的资源,或者必须仔细、静态地为 DiffServ 网络区有限的传输模式进行资源分配。另一种极端情况是 DiffServ 网络区的所有路由器都参加 RSVP 信令,则资源会被最优化使用,但信令处理的需求和相关的系统开销也会增加。前面讨论过的聚集 RSVP 就是一种以部分损害资源利用率来限制信令过程开销的方案。网络管理员需对此进行折中考虑。

综上所述,在 DiffServ 网络区使用 RSVP 动态地提供资源的方案既充分利用了 RSVP 信令的优越性,又保持了 DiffServ 的可扩展性。因为 DiffServ 网络区内的路由器其控制平面是 RSVP 而数据平面仍是 DiffServ。但事实上,既然采用了 RSVP,就不可避免地引入了

IntServ 接纳控制的可扩展性和鲁棒性问题，目前这仍然是一个开放的研究课题。好的方案应该针对实际应用需求在资源利用的有效性与实现机制的复杂性之间进行折中。

### 4.3.3　使用其他方式的动态资源管理方案

DiffServ 网络区内部的边界路由器可以使用任何形式的 RSVP 信令、Boomerang 协议[10]或其他定制协议与各 DiffServ 区域的专用的集中管理实体——带宽中介服务器 BB[11]相互作用。BB 含有资源可利用性和网络拓扑的充分的信息，负责记录本区域资源的占用情况，并据此对客户或相邻区域 BB 的服务请求实施接纳控制。与传统的基于状态预留的网络所采用的分布式逐节点的接纳控制不同，BB 以集中的方式在每个 DiffServ 区域或所有 DiffServ 区域进行接纳控制[12]。

一种实现方案可以只使用一个集中的 BB 来维护整个网络的拓扑和所有节点的状态，并基于 BB 实现接纳控制，删除了维护分布式预留状态的必要。这种完全集中式方案适合大多数传输流长期存在而建立和拆除事件不经常发生的情况。但是，如果需要支持精细粒度和动态流，就需要一种分布式的 BB 体系结构，对 BB 数据库进行复制或分割。目前分布式 BB 体系结构仍然是一个活跃的研究领域。我们可以设想这样一个体系结构：当一个 BB 接收到一个请求时，它基于自己的数据库，做出接收或拒绝的决定，而无须请求其他 BB。这避免了信令协议，但需要其他协议来维护不同 BB 数据库之间的一致性。但实际上不可能获得非常好的一致性，这将导致竞争或资源破碎。至此，问题转化为网络可扩展性和资源分割之间的一种基本的权衡：增加 BB 的数量可以使方案具有更好的可扩展性，但同时增加了资源的破碎程度。目前，DiffServ 控制路径上的接纳控制问题仍是一个开放问题，需要进一步研究。

## 4.4　DiffServ 网络区支持端到端 IntServ 的研究展望

为了支持上述的集成框架，DiffServ 网络区必须满足以下需求：

（1）必须能够在 DiffServ 网络区的边界路由器之间为标准的 IntServ QoS 服务提供支持，在 DiffServ 区内部使用标准的 PHB 来调用这些服务；

（2）必须在 DiffServ 区的边界节点执行适当的管理、控制（如包括整形或重标记）；

（3）DiffServ 网络区必须基于资源可利用性，为其客户（非区分服务）网络提供接纳控制机制。

（4）DiffServ 网络区必须至少能够透明地传递 RSVP 消息，以便在 DiffServ 区出口可以重新获取这些消息。DiffServ 网络区可以（但并不绝对要求）对 RSVP 消息进行处理。

要满足上述需求，以下问题有待进一步研究：

（1）IntServ 类型服务描述到 DiffServ 网络区提供的服务之间的映射[13]；

（2）定义 DiffServ 网络区内使用聚集传输控制的网络元素在支持 RSVP 信令时所需的功能；

（3）定义在 DiffServ 网络区内有效的、动态的资源提供机制（如聚集 RSVP，隧道，MPLS 等）以及 BB 如何将 DiffServ 区内的资源可利用性信息传递到边界路由器的定制协

议等。

目前,实现 IntServ 与 DiffServ 的结合以提供端到端 QoS 仍然是一个开放的研究课题。在此方面已经提出了若干不同的研究方案和实现机制,典型的有:面向 SCORE(scalable core)的‘动态分组状态’DPS(dynamic packet state)方案[12,14],在分组头部对传输流的状态进行编码,但存在实现上的困难;一种基于三种优先级别的带宽确保型服务(BGS)机制来提供端到端的 QoS[15]。文献[16]提出了另外一种在 DiffServ 的网络云中结合 RSVP 协议提供资源预留和 QoS 保证的框架,其中 DiffServ 网络云的边界设备 ED 允许与 RSVP 协同工作,而其内部的路由器不识别 RSVP 消息,DiffServ 网络的接纳控制在接纳控制服务器 ACS 的协助下完成,ED 与 ACS 之间通过一个叫作‘简单接纳控制协议’的 client/server 协议进行通信。

综上所述,DiffServ 适合在网络主干实现 QoS,端到端的 IP QoS 则需要实现 IntServ 与 DiffServ 的有机结合。而这必须以上述的基本框架为指导,综合考虑有效性与可扩展性,研究更好的实现方案及其算法和协议。目前,虽然基本框架和试验方案已经为研究提供了一定的基础,但要实现真正的端到端的 QoS,还有很长的路要走。

## 参考文献


1 林闯,单志广,盛立杰,吴建平. 区分服务及其几个热点问题的研究. 计算机学报,2000,23(4):419~433

2 Bernet Y, Yavatkar R, Ford P, et al. A framework for integrated services operation over DiffServ networks. IETF Internet Draft〈draft-ietf-issll-diffserv-rsvp-03.txt〉, September 1999

3 Bernet Y, Yavatkar R, Ford P, Baker F, Zhang L, Nichols K, Speer M. A framework for use of RSVP with diff-serv networks. IETF Internet Draft〈draft-ietf-diffserv-rsvp-01〉, November 1998

4 Bernet, Y. Usage and format of the DCLASS object with RSVP signaling. IETF Internet Draft〈draft-issll-dclass-00.txt〉,August 1999

5 Guerin R, Blake S, Herzog S. Aggregating RSVP-based QoS requests. IETF Internet Draft〈draft-guerin-aggreg-rsvp-00.txt〉, November21, 1997

6 Berson S, Vincent R. Aggregation of Internet integrated services state. IETF Internet Draft〈draft-berson-rsvp-aggregation-00.txt〉, August 1998

7 Baker F, Iturralde C, le Faucheur F, et al. RSVP reservation aggregation. IETF Internet Draft〈draft-ietf-issll-aggregation-00.txt〉, September 1999

8 Terzis A, Krawczyk J, Wroclawski J, et al. RSVP operation over IP tunnels. IETF Internet Draft〈draft-ietf-rsvp-tunnel-04.txt〉, May 1999

9 Fred Baker, Carol Iturralde, Francois Le Faucheur, Bruce Davie. Aggregation of RSVP for IPv4 and IPv6 reservations. IETF Internet Draft〈draft-baker-rsvp-aggregation-01.txt〉, June 1999

10 Ahlard D, Bergkvist J, et al. Boomerang—A simple resource reservation framework for IP. IETF Internet Draft〈draft-ahlard-boomerang-framework-00.txt〉, February 1999

11 Nichols K, Jacobson V, Zhang L, et al. A two-bit differentiated services architecture for the Internet. IETF RFC 2638, July 1999

12 Stoica I, Zhang H, et al. Per hop behaviors based on dynamic packet states. IETF Internet Draft〈draft-

stoica-diffserv-dps-00. txt〉, February, 1999

13　Ford Peter S, Bernet Yoram. Integrated services over differentiated services. IETF Internet Draft〈draft-ford-issll-diff-svc-00. txt〉, March 1998

14　Stoica Ion, Zhang Hui. Providing guaranteed services without per flow management. In: ACM SIGCOMM '99. http://www. cs. cmu. edu/ ~ hzhang/

15　Borgonovo F, Fratta L, Petrioli C. end-to-end QoS provisioning mechanism for differentiated services. IETF Internet Draft〈draft-borgonovo-qos-ds-00〉, July 29, 1998

16　Detti A, Listanti M, Salsano S, Veltri L. Supporting RSVP in a differentiated service domain: An architectural framework and a scalability analysis. In: IEEE International Conference on Communications (ICC '99). Vancouver, British Columbia, Canada, 1999. 204 ~ 210

# 第二部分

# QoS 的实现机制

QUALITY OF SERVICE OF COMPUTER NETWORKS

# 第5章

# ATM 网络的传输管理与 QoS 控制

异步传输模式(asynchronous transfer mode，ATM)是针对多媒体信息传输要求而提出的、当今通信网络领域中一种具有优势的技术[1]，它是一种基于信元的、能够提供服务质量(quality of service，QoS)控制和宽带保证的多路复用及交换技术。ATM 传送信息的基本载体是 ATM 信元。ATM 使用 53 个字节(包括 5 个字节的首部和 48 个字节的信息字段)的固定大小的信元。ATM 是针对广域载波的策略性建网技术。国际电信联盟(ITU，以前名为 CCITT)于 1988 年把 ATM 定义成：未来公用 B-ISDN 网络的基础技术。

ATM 网络虽有很大发展，但还没有广泛普及。究其原因，除了价格、标准等因素以外，其传输控制质量也存在不足。现有的 ATM 交换机还不能完全保证 QoS，缺乏简单、有效的控制方案和算法的实现，传输管理与控制亟待改进。而且，ATM 网络面向连接的特性也使其应用和发展受到一定局限。但是，在 QoS 研究转向 Internet 的过程中，ATM 网络的一些技术思想，尤其是其传输延迟控制、缓冲管理与调度的策略和算法[2~5]等对 IP QoS 等的研究仍具有重要的借鉴意义。

ATM 网络的研究主要包括协议、管理与控制等问题。协议的标准和定义是 ATM 论坛[6]的主要工作。传输管理与控制是 ATM 网络的核心技术，也是研究的重点，因为网络的每一个操作都涉及到管理与控制的问题，而且管理与控制问题大多数是非标准问题，是实现的问题。

ATM 的技术特点之一就是要提供服务质量(QoS)的保证。在 ATM 网络中，网络的各类资源，如网络带宽、节点中的缓冲器容量等是有限的。当网络的资源分配不平衡或调度不当时，网络的服务质量就会降低。因此，为了保证网络的服务质量始终处于较高状态，就必须对网络中的传输进行控制和管理[7]。

与传统网络不同，ATM 网络有潜在的能力支持所有类型信息(例如，声音、电视图像和数据等)在一条线路中传送。它具有更高的传输速率和传输容量，能够同时满足多种要求，提供异种传输服务。ATM 网络提供了综合的服务能力，能够按需分配带宽，可以更灵活地对网络进行存取，更有效、更经济地提供传输服务。ATM 网络的这些固有特征决定了它必须具有与传统网络不同的传输控制和管理的一系列策略和方法。

## 5.1 ATM 网络的传输特点

在 ATM 网络中,可以有 4 种不同类型的传输方式:

(1) 恒定位速率(constant bit rate, CBR):该类型用于仿真位速率等于常量的电路交换。

(2) 可变位速率(variable bit rate, VBR):该类型允许用户发送可变速率的信息,采用统计复用的方法减少非零随机丢失率。根据应用对信元延迟敏感的不同,又可分为实时 VBR(如视频影像信息)和非实时 VBR(如多媒体 E-mail)两种类型。

CBR 和 VBR 服务要求用户在传输前预报并声明所需的带宽,要求网络在传输过程中保留固定的带宽以达到需求的服务质量。

(3) 可用位速率(available bit rate, ABR):该类型规定了峰值信元速率和最小信元速率,用于传输文件、E-mail 等普通数据信息。ABR 业务用反馈控制使信息流量与网络实时可用带宽相适配,有了这种能力,用户可以在一定的时间段内可靠地传送突发数据,而使丢失的信元最少,可用的吞吐量最大。

(4) 未指定位速率(unspecified bit rate, UBR):该类型适用于对信元的丢失和延迟都不敏感而且又希望使用网络剩余资源的应用。对于这类应用,带宽的短缺并不影响连接的建立。当网络拥塞时,信元虽会丢失,但信源(信元发送节点)具有重传输机制,因而不必降低信元的发送速率。UBR 用于传输文件和 E-mail。不过,由于这些应用也适合 ABR 类型,因此通常 UBR 与 ABR 类型合并使用。

不同的传输类型由 ATM 网络在适配层(ATM adaptation layer)上提供不同的适配服务。ATM 适配是指 ATM 网络获取原始数据流,并将它们封装在 ATM 信元内,将不同种类的原始数据转换成 ATM 网络所要求的数据信元流的形式,实现在 ATM 网络上传输的过程。由于 ATM 网络可传输数据、语音以及视频等多种信息,而每一种都需要不同的适配操作,因此,ATM 定义了 AAL1,AAL2,AAL3/4 和 AAL5 共 4 类不同的适配层类型,根据适配层负载数据传输类型的不同和是否面向连接,可将适配层提供的服务分为 4 类,不同的适配层类型对应着不同的服务类型:

A 类——固定位速率(CBR)服务:对应于 AAL1,它支持面向连接的服务,用于传输率固定、对信元延迟和丢失都敏感的应用,例如具有固定传输速率的声音和图像信号。

B 类——可变位速率(VBR)服务:对应于 AAL2,它支持面向连接的服务,用于传输率变化、同时对延迟敏感的应用,经压缩分组的声音和图像信号的传输属于此类服务范畴。

C 类——面向连接的数据服务:对应于 AAL3/4,AAL5 也支持这类服务,它支持面向连接的服务,用于传输率可变、但对延迟不敏感的突发性业务和普通数据传输业务的应用,它采用 ABR 传输方式。面向连接的文件传输和 E-mail 都属于此类服务范畴。

D 类——无连接数据服务:AAL3/4 和 AAL5 都可以支持这类服务,通常,它采用 UBR 传输方式,那些在数据传输前不用建立连接的数据网络业务属于此类应用。

## 5.2　ATM 网络的传输管理与 QoS 控制技术

基于网络中事件受控时间量的考虑，ATM 网络的控制可以分为 4 个层次。每一层次都要确保它下面层次的所有服务满足性能要求。4 个层次包括：

（1）网络层控制：它在第 4 层，在层次控制的顶部。它的受控时间数量为几小时或一天。这个控制包括对上述 4 类传输的网络资源的分配和决定呼叫层控制所采用的接纳控制（admission control）策略的必要参数。

（2）呼叫或会话层控制：它在第 3 层，在网络控制层的底部。它的受控时间数量为几分钟。这个控制主要是接纳控制，决定新的呼叫或会话是否接受。呼叫接受策略要考虑突发层（第 2 层）的拥塞和延时的影响和信元层（第 1 层）的信元延时和信元丢失率的影响。

（3）突发或分组层控制：它在第 2 层，在呼叫或会话控制层的底部。它的受控时间数量为几百毫秒到几秒。主要目的是优化网络资源的使用。网络资源的使用可以采用两种方法：为保证会话期间的信源突发不造成拥塞，就要为会话保留充分的资源，但这要造成资源利用的低下。在统计多路复用（statistical multiplexing）使资源得到充分使用的同时，也会导致拥塞。突发层控制的目标就是取得新突发被阻挡的概率和突发被阻挡时间之间的折中。

（4）信元层控制：它在第 1 层，在突发或分组控制层的底部。它的受控时间数量为几十微秒到几毫秒。这层控制主要包括信元丢失控制和信元实时调度传输。

### 5.2.1　接纳控制

ATM 网络是面向连接的，它在各个用户入网前首先需要进行接纳控制。接纳控制是避免网络拥塞发生的一种技术。ATM 网在用户入网时要求用户首先把自己的传输特性和参数以及它所要求的服务质量告知网络，网络再基于用户的传输性能要求和网络现存的资源情况，来决定是否允许建立一个新的连接。

在接纳控制中，有三个主要问题需要研究：哪些传输参数可以确切地描述一个连接的传输？网络使用哪些判据来决定是否接受一个新的连接？网络性能与传输参数之间的关系如何？下面分别进行描述。

#### 1. 传输描述参数

为了保证网络的服务质量（QoS），当一个连接要求建立时，网络必须知道新连接的传输特性以便确切地预测它所能维护网络性能的水准。在目前的 ATM 网络研究中，要求用户提供的有关 QoS 属性的参数主要包括：

（1）峰值信元速率（peak cell rate）：指传输中的最大瞬间速率，可用产生两个相邻信元的最短时间间隔的倒数来表示；

（2）持续信元速率（sustained cell rate）：指一段时间内信元传输的平均速率；

（3）信元丢失率（cell loss rate）：由于出错或拥塞，网络上丢失的信元数与用户发出

的信元总数之比；

(4) 信元传送延时(cell transfer delay)：指信元从信源发出至到达目的节点之间的那段时间,由传播延时、排队延时、交换延时等组成；

(5) 信元延时方差(cell delay variation)：是信元传送延时的变化度量,当该变量取值高时,意味着要给延迟敏感数据(如声音和图像)传输提供较大的缓冲；

(6) 突发容忍量(burst tolerance)：这个量决定在峰值速率下可以发送的最大突发长度,它是传输控制‘漏斗’算法中表示‘斗’大小的参数；

(7) 最小信元速率(minimum cell rate)：这是由用户决定的信元最小传输速率。

在这些参数的规定中,突发的量度是最重要的参数,如何最好地判定突发的程度仍旧是一个困难的工作。

2. 允许判据

允许判据是指网络在判断是否接受一个新的连接时进行决定的依据。信元传送延时和信元丢失率是两个最常用的允许判据。这两个判据是网络拥塞程度的度量。

在 ATM 网络中,信元传输采用统计多路复用的方法共享资源,各个连接没有固定速率的专用信道,当某一连接传输量增加时,会占用其他连接的资源,从而会影响其他连接的服务质量,特别是由于 ATM 网络传输具有高突发性、高速率,其传输速率变化很快,因此 ATM 网络拥塞状态变化也非常迅速,增加了接纳控制过程的复杂性。仅使用长时间项的平均信元传送延时和信元丢失率作为允许判据,不能充分表示 ATM 网络快速、动态变化的拥塞程度。因此必须考虑能反映网络瞬间行为的判据。瞬间网络允许判据是 ATM 网络所特有的允许判据,一些短时间项的瞬间行为的判据,例如传送延时的变化、最大传送延时、信元丢失变化率等,已在 ATM 网络中采用。

3. 网络性能与传输参数之间的关系

在 ATM 网络的接纳控制中,一个重要的研究问题是各种传输参数与网络性能之间的关系。一些定性的关系和影响已经给出,但是如何给出定量的数学描述来表明它们之间的关系仍然是当前面临的挑战,尤其是在多个异种传输流被多路复用的情况下更是一个困难问题。

## 5.2.2 拥塞控制

拥塞控制(congestion control)是 ATM 网络传输控制与管理的核心问题。一般来说,当一个链路的输入信息量大于输出信息量时,就会发生拥塞。拥塞控制的主要功能是在维护对用户的网络资源公平分配的同时,确保好的吞吐和延时性能,尤其是在 ATM 网络中,存在大量高突发和不可预测的传输服务,这对拥塞控制技术是一项重大的挑战。

ATM 网络经接纳控制建立连接后,基于连接的特性和 QoS 要求,必须给这个连接分配一定的带宽。由于 ATM 网络允许所有的连接共享带宽资源,加上各类传输的速率变化很大,因此,实际入网的传输流量很有可能超过分配给它的带宽,造成拥塞。这时就需要对传输流量进行监控,以保证传输流传输过程中的特性与它申请入网时要求的传输特性

以及网络分配给它的带宽相符合。

在拥塞控制中，包括“开环预防控制”和“闭环反馈控制”两个方法。开环预防控制采用普通信元速率算法对信元的流速进行传输整形（traffic shaping）。闭环反馈控制也叫反馈流控制。

### 5.2.2.1　开环预防控制

1.“漏斗”算法

目前最常用的带宽监控技术是普通信元速率算法，也叫“漏斗”（leaky bucket）算法，这个算法可将突发传输流转化为平缓传输流。漏斗算法用于确保用户的传输流遵守用户在建立连接时的规定。

漏斗算法的基本思想是：任何一个信元要进入网络，一定要从令牌池（漏斗）中取得一个令牌 如果此时令牌池为空，则该信元被丢失。令牌以网络平均允许速率 R 产生，令牌池最多可存放 M 个令牌（M 即漏斗的大小，由传输参数的突发容忍量表示）；当令牌池满时，新产生的令牌被丢弃。图 5.2.1 是这种方法的示意图。

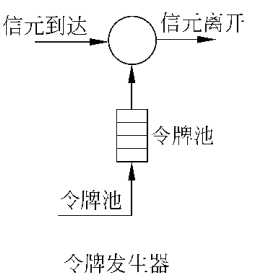



图 5.2.1　漏斗算法示意图

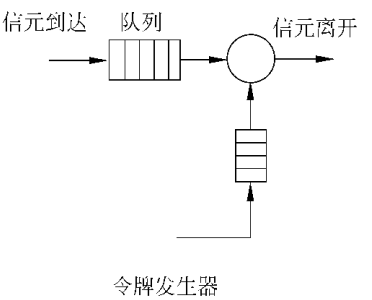



图 5.2.2　带缓冲器的漏斗算法

一个改进方法是在信元到达漏斗前增加一个缓冲区，这样，当令牌池为空时，只要缓冲器没有满，信元就可以缓存在缓冲器中而不被丢失，如图 5.2.2 所示。当然，这种改进以增加信元的等待时间来换取低的信元丢失率，因此，如何选择一个合适的缓冲区大小以达到两者的最优控制是该方法的关键。

这些漏斗算法还有一些缺点。例如，即使在网络负载很低时，漏斗算法对违约信元仍然采用丢弃或放入缓冲区的方法，由于算法的限制，减缓了传输流速，造成网络资源的浪费。采用标志法可以改善这一缺陷：当信元到达但令牌池为空或缓冲器已满时，就将该信元打上一个标志，说明是违约信元，然后允许它进入网络 如果在网络某处遇到拥塞，则丢弃，若一直没有遇到拥塞，则可到达目的节点。

2. 传输整形

在 ATM 网络中，传输流是高度突发的，其传输速率变化很大。如果根据排队理论，适当地改善传输流进入网络的统计特性，使得信元到达分布这一随机过程的统计特性越平滑，网络服务质量（时延、丢失率）就越好。传输整形就是要避免信元在网络中的突发性

传输,达到改善网络 QoS 性能的目的。漏斗算法注重于带宽的增强,传输整形技术则注重于降低突发度和改进传输流进入网络的分布。传输整形技术成功的关键在于减少了传输延时和信元的丢失。

传输整形的基本原理是采用缓冲来完成传输流的平滑整形:在缓冲区中建立两个阈值 A 和 T,其中 $A < T$;当缓冲区中信元数超过 T 时,网络控制器就将信元到达速率由峰值速率 $P_k$ 降到规定速率 $G_k$,从而降低缓冲区被占有的程度;当缓冲区被占水平低于 A 时,网络控制器就将信元到达速率恢复为峰值速率 $P_k$。

#### 5.2.2.2 反馈流控

开环流控制是静态的,即使在网络带宽空闲的情况下,信元的传输速率也不能超过信元进入时允许的连接速率。因此,开环流控制更适合于 CBR 和 VBR 服务。对于 ABR 服务来说,由于它的高突发性和不可预测性,仅有开环静态的控制是不充分的,因此闭环、动态的流控是不可缺少的。这种拥塞控制需要反馈机制以使信源知道拥塞点的信息状态。ATM 网络中主要有如下两类反馈流控(feedback flow control)方案:基于信用的流控(credit-based flow control)和基于速率的流控(rate-based flow control)。

1. 基于信用的流控

由于传输突发造成网络超载的最显著标志是缓冲的溢出,信用流控就是直接控制缓冲的分配。在沿着链路向前传送任何信元之前,发送节点首先需要接收来自接收节点的信用信息,以确定接收点有可接收信息的缓冲空间。发送节点接收到信用信息后,在接收节点可接收范围内传送一定数量的信元数,从而保证不会发生拥塞。信用流控是虚连接(virtual connections)链路到链路窗口流控的有效实现方法。它的特点是提高了链路的利用率、控制质量和公平性。最明显的一个缺点是具有一定的复杂性和缺少灵活性,以至于这种控制方案没有被 ATM 论坛选用。

2. 基于速率的流控

基于速率的流控直接控制连接的带宽(速率)。该控制方法在 1994 年被 ATM 论坛选用为 ABR 传输服务的最佳控制方法。基于速率方案采用端到端控制,使用网络反馈信息来规定每个虚连接上每个信源能发送的最大速率。基于速率的流控方案包括两个阶段:速率的设置和速率的控制。在这种方案中,分配给一个连接的带宽与两点间的延时无关,因此基于速率的流控在结构上是灵活的。这种方案支持公平的带宽分配。更重要的是这种方案无须复杂的队列管理。但是,它也面临着众多有待改进和研究的问题。

反向显式拥塞通告(backward explicit congestion notification, BECN)方案直接从拥塞点向每一个虚通道信源发送拥塞信息,因此,它的显著优点是对拥塞反应快速。它对速率的控制也简单而经济,因为它仅仅依赖于 BECN 信元来调整速率的变化。它的缺点是每一个中继节点都参与了拥塞控制,增加了系统开销。

前向显式拥塞通告(forward explicit congestion notification, FECN)采用前向显式拥塞指示(forward explicit congestion indication, FECI)机制。多个连接共享一个中间节点的链

路队列,拥塞状态由中间节点链路队列的平均队列长度决定。当平均队列长度达到某一给定阈值时,通过队列的信元就打上一个 FECI 标志,继续传输给目的节点,一旦目的节点接收到带 FECI 标志的信元,就向信源发送拥塞信息,信源据此按递增和倍减原则调整信元发送速率,其公式如下:

$$ACR = \min(ACR + b, PCR)$$

$$ACR = \max(ACR \cdot d, MCR) \qquad 0 < d < 1$$

其中,b 是增量,d 是减少系数,ACR 是信元的传输速率,PCR 是最大允许速率,MCR 是最小信元速率。这些系数都是在连接建立时由网络规定的。由于速率的调整由网络拥塞状态和信元瞬时的传输速率共同决定,因此,该方案允许各个连接公平地分享带宽。FECN 方案已被局域网和广域网所采用。

FECN 方案与 BECN 方案相比,是由终端节点直接向信源发送拥塞信息,各中继节点不直接参与拥塞控制,系统开销相对较小。其缺点是,拥塞信息传输路径过长,不利于信源对拥塞做出快速反应。FECN 方案与 BECN 方案各有优缺点,关于反馈流控具体实施方案的研究还在不断讨论之中。

### 5.2.3　信元丢弃控制

信元丢弃控制是与拥塞控制密切相关的。当拥塞发生时,ATM 网络为缓解拥塞程度,会丢弃部分信元。在可视图像和声音信息的传输中可以容忍一些信息的丢失。但另一方面,在数据文件的传输中对信息丢失的容忍度却非常低。所以传输控制必须根据应用的要求对不同类型的信息给予不同的丢失优先级。基于优先级的信元丢失控制方案可以改善系统的性能。目前有三种信元丢失控制方案:分别选道(separate route)、推出(push-out)和部分缓冲共享(partial buffer sharing)。

1. 分别选道方案

分别选道方案为每种级别的信元分别使用一个不同的缓冲器。这种方案实现简单,但在目的节点信元需要重新排序,它破坏了 ATM 面向连接的特性,因此很少采用。

2. 推出方案

推出方案使用同一个缓冲器存储所有级别的信元。当缓冲器已满时,如果有低优先级的信元存在于缓冲器中,则高优先级信元的到来将“推出”低优先级信元,并将之丢弃,而存储高优先级的信元。相反地,低优先级的新到信元则不能抢占高优先级信元的存储空间。既然信元的次序必须保留,则这种方案需要复杂的缓冲管理,因而未被广泛采用。

3. 部分缓冲共享方案

部分缓冲共享方案虽然也使用一个缓冲器来存储所有级别的信元,但它的控制是基于缓冲的阈值(threshold)。当缓冲的容量超过阈值时,仅高优先级的信元可进入缓冲器,而新来的低优先级的信元则被丢弃。当缓冲已满,即使缓冲中存有低优先级的信元,高优先级的信元也要丢弃。这种方案实现简单,已被广泛采用。

当前的信元丢失控制研究集中在：①如何增强部分缓冲共享方案的效率,改善网络的动态性能；②多优先级、多阈值缓冲共享方案的形式描述和阈值的确定；③在突发传输下网络性能的瞬间分析。

### 5.2.4 信元传输实时调度

前面已经提到,不同的信息类型有着不同的传输性能要求,例如,实时的声音和图像信息是延迟敏感的,数据信息则是丢失敏感的,即使在延迟敏感传输中,不同的传输流也存在着不同的延迟要求。由于 ATM 网络支持多种传输类型,为保证每一种传输类型都达到 QoS 要求,就必须在交换节点使用传输调度算法。调度算法的基本功能是从交换节点的每一个输出链路中挑选在下一个有效周期发送的信元。这种挑选要基于几个服务原则,例如带宽的保证、延时的保证和公平选择等。一般可采用优先级方案。

静态优先法是一种简单的优先级方案,该方案对延时敏感的实时传输进行优先调度传输,这样延时敏感信息的传输总是在丢失敏感信息的传输之前传输。静态优先法虽然降低了延迟敏感信息传输的延迟时间,却使得丢失敏感信息的丢失率较高,因此存在明显的缺陷。传输调度方案的一般性讨论见 5.3 节。

## 5.3 ATM 网络的传输管理与 QoS 控制策略

在 ATM 网络中,传输的管理是指资源的管理,也就是网络带宽和节点中缓冲器的管理 控制是指信元的储存和调度,也就是信元的实时、优先、公平等储存和调度的策略。对于这些问题的研究,一般包括给出策略方案(算法)、模型描述、求解证明三个方面。

### 5.3.1 资源管理策略

资源管理又可分为资源的分配和资源不足时的阻挡或丢弃策略。在资源分配策略中基本上包括三种：完全共享、完全分配和部分共享。

#### 1. 完全共享

总体资源没有限制地在所有的竞争用户(呼叫或会话)之间共享,只要还有充分的资源,任何一个新用户的资源要求都会接受。当没有充分的资源时,一个新用户的资源要求被阻挡。这种方案不要求控制,它的一个明显的缺点是资源的不公平分配。一个负载沉重的用户可能会强占其他用户使用资源的权利。

#### 2. 完全分配

在这个方案中,资源是在所有竞争用户(呼叫或会话)之间永久性地进行分配,亦即,给每一个用户分配固定数量的资源。这种方案的一个明显的缺点是资源不能充分利用。有的繁忙用户可能由于没有资源而被阻挡,相反地,懒惰用户的资源则闲置不用。

若要改变这一缺点,资源的动态分配是一个有效策略。资源的动态分配是以用户的需求和系统环境变化的预测为前提的。需求和环境变化的预测是一个困难的问题。为了

解决这一问题，可以采用概率预测和基于系统变化测量的预测方法。

在概率预测中，采用数学模型对系统运行进行描述，以求出各需求和变化的概率。

在基于系统变化测量的预测中，可以采用周期测量，预测思路是使用“最近的将来类似于最近的过去（已发生的是将要发生的）”。

3. 部分共享

为了克服上述两种方案的不足，可以采用将一部分资源分配给用户，剩余的资源供用户竞争共享的部分共享方案。上述两种方案是这种方案的特例。当部分资源扩大成全部资源，而另一部分资源简缩为零时，这种方案就变成了上述两种方案之一。部分共享方案可以有多种策略变化。

（1）截止（cut-off）优先方案

给用户分配不同的资源使用优先级，确定部分共享资源的阈值（threshold）。当已被使用资源和用户申请资源之和不超过阈值时，所有用户申请资源的要求都可以被接受。当已被使用资源和用户申请资源之和超过阈值时，则只有高优先级用户的资源使用申请才可以被接受，而低优先级用户的资源使用申请则被阻挡。进一步地，也可以考虑阈值随着系统运行而动态变化，以提高资源的利用率。

（2）概率保留策略

在这种方案中，一个用户的资源使用申请被接受与否取决于接受概率和阈值。当已被使用资源和用户申请资源的总和不超过阈值时，低优先级用户申请资源的请求被接受与否取决于接受概率。当已被使用资源和用户申请资源的总和超过阈值时，只有高优先级用户的资源使用申请可以被接受，而低优先级用户的资源使用申请则被阻挡。接受概率可以是资源占用的函数，此函数可以通过实际系统的测试获得。

（3）具有最小分配的共享

给每一个用户分配的资源能保证它最低的性能要求，剩余的资源供所有用户竞争共享。这种方案充分兼顾了资源分配的公平性和利用率。

4. 阻挡或丢弃策略

在用户的资源使用申请遭到阻挡后，可以将申请储存起来，等到下次有机会再申请，但一般情况下要将用户的申请（或信元）丢弃。丢弃可有多种策略：

（1）最简单的方法是将新用户的申请（或信元）丢弃；

（2）从已占有资源的用户申请（或信元）队列中挑选出具有最小优先级的用户申请（或信元）进行丢弃，而将资源分配给新的用户申请（或信元）；

（3）为了保证资源分配的公平性，从已占有资源最多的用户申请（或信元）队列中挑选出一个用户申请（或信元）进行丢弃，而将资源分配给新的用户申请（或信元）；

（4）为了保证系统的性能要求，从最能满足丢弃要求的已占有资源的某用户申请（或信元）队列中挑选出一个用户申请（或信元）进行丢弃，而将资源分配给新的用户申请（或信元）。

### 5.3.2 信元的存储和调度策略

信元的储存和调度主要考虑信元实时传输和公平性的要求,满足这些要求的不同方法表现为不同的策略。

1. 先入先出的调度策略

先入先出(FIFO)的调度策略是一种最自然、最简单的调度策略。它没有考虑信元传输的不同性能要求,使用相同的服务规则对待所有的信元,仅按信元到达的顺序进行服务和输出。

2. 静态优先级的调度策略

对于静态优先级调度策略,在信元进入网络之前,按它们的实时传输要求分配优先级,信元到达时按优先级排队,优先级最高的信元总是优先接受服务和输出。这个方案考虑了信元所相关联应用的性能目标,但是它没有考虑信元传输的紧迫性会随着时间变化而变化,有些信元是有时间期限的,超过了时间期限的信元是没有意义的,应予以丢弃。对这种情况的一种改进,就是保留充足的带宽,使信元的传输不超过时间期限,但会使网络的利用率降低。

3. 动态优先级的调度策略

动态优先级策略对静态优先级策略进行了改进。在动态优先级策略中,优先级是随时间而动态变化的。可以给定一个最小松弛阈值(minimum laxity threshold, MLT)或队列长度阈值(queue length threshold, QLT),用阈值控制来改善低优先级信元传输的性能。

在采用最小松弛阈值时,等待队列中信元的松弛度定义为缓冲区中空槽的个数,队列中的信元或者被传送出去,或者其松弛度变为零。若松弛度为零,则该信元被丢弃。如果队列中有松弛度小于最小松弛阈值的延迟敏感信元,则将优先级赋予延迟敏感信元的传输;否则,将优先级赋予丢失敏感信元的传输。在采用队列长度阈值时,若队列中丢失敏感信元个数超过阈值,则将优先级赋予丢失敏感信元的传输;否则,将优先级赋予延迟敏感信元的传输。

与 QLT 方案相比,MLT 方案在每一个时间区间内,都必须重新计算每个实时信元的松弛度,然后再从队列中找出具有最小松弛度的信元,这一过程非常复杂,当到达转换节点的信元数很多时,往往使该节点成为传输瓶颈。因此,相对而言,QLT 方案由于其算法简便,比 MLT 方案更具有实用性。

4. 最早截止优先的调度策略

最早截止优先(earliest-deadline-first)的调度方案要求在信元进入网络前给信元分配传输时间期限,具有最早期限但又不超过时间期限的信元被首先传输。在不考虑信元丢失率有不同优先级的条件下,这个方案是最优方案。

5. 信元丢失控制和实时调度的综合策略

这个方案同样要求在信元进入网络前给信元分配传输时间期限，但为了提高网络吞吐量，在信元进入等待队列时，将信元尽量放在靠近时间期限位置。当从等待队列头部到时间期限位置已满时，从这些位置中挑选出一个具有最小丢失优先级的信元，将其丢弃，而将新的具有较高丢失优先级的信元放入等待队列。这个方案采用工作保留规则，只要等待队列不为空，就对队列中排位最前的信元进行传输服务。这个方案具有信元加权丢失率最优的性质。

6. 公平调度策略

在信息传输的调度中，不仅要考虑传输的实时性，而且要考虑传输的公平性，亦即，使每个会话都同等地得到所要求的带宽。典型的调度方案如下。

（1）广义处理器共享方案

在处理器共享方案中，对于每个会话都有一个先进先出（FIFO）队列，它们共享着相同的链路。在任何时间间隔，都有正好 N 个非空队列，服务器以链路速率的 N 分之一同时传送在队列头部的 N 个信元。处理器共享方案以相同速率服务所有的非空队列。广义处理器共享（generalized processor sharing，GPS）方案则是处理器共享方案的扩充，允许不同的会话有不同的服务速率。GPS 方案有两个特性：可以保证端到端有界延时服务和确保带宽的公平分配。GPS 方案只是一个理想的流体模型，不能在实际中实现。

（2）加权公平排队方案

加权公平排队（weighted fair queueing，WFQ）是一个接近 GPS 的实际信元网络方案。流体系统与信元网络之间的区别是：在任何时间，在流体系统中有多个信元同时接受服务，而在信元网络中仅能有一个信元接受服务。在 WFQ 方案中，在长期运行的情况下，一个会话在流体系统和信元网络中所接受的服务是相同的。在传输延时方面，WFQ 比 GPS 至多慢一个信元传输时间。

## 5.3.3　模型描述和求解证明

各种方案和策略一般都需要形式化的模型描述或明确的算法，以便方案和策略的实现、比较、求解和验证。

1. 方案和策略的模型

方案和策略要尽量给出形式化的数学模型，有了数学模型才能对方案和策略的执行进行有效的分析和预测。模型的分析和求解结果是方案和策略取舍的基础。当前方案和策略常用的模型工具是排队论和随机 Petri 网等，它们的数学基础是马尔可夫随机过程。常用的模型方法是层次模型、面向对象和客户机-服务器等模型方法。在模型中，要做到准确地抽象出方案和策略的本质，尽量简化子模型之间的关联，精化模型的设计。

2. 模型分解、压缩的化简求解

方案和策略的模型求解的复杂性会随着系统的增大而呈指数性地增长,许多实际系统的方案和策略模型理论上可分析求解,但实际上由于受到计算机的储存等限制,这些模型是不可求解的。解决系统模型的状态空间爆炸问题的有效方法,就是采用‘分而治之”的策略。分是指模型的分解,从模型的结构上进行分割,将一个模型分割成多个子模型;治是指模型的压缩,将子模型压缩成更简单的模型、模型元素或模型参数,将它们按原子模型之间的关系连接起来。最后求解简化模型的性能参数。在排队论中,分解和压缩的技术思路主要是将具有非乘积解的模型转化为具有乘积解的模型。在随机 Petri 网模型中,主要的分解和压缩技术有:接近无关的分解、时间数量级分解、响应时间保留压缩替代、流等价压缩替代和层次模型分层分析等。

3. 测试方案的优化验证

基于环境和系统运行情况测试的方案和策略一般不能给出性能数学模型,但可以给出明确的算法。这个算法需要在一种性能度量的数学定义下进行证明,以表明方案和策略的最优性。方案和策略的优劣程度也可以通过模拟进行比较。

## 参考文献


1 Bae J J, Suda T. Survey of traffic control schemes and protocols in ATM networks. In:Proceedings of the IEEE, 1991, 7(2):170~189

2 林闯, 杨士强. ATM 网络传输实时调度的最少缓冲优先方案和性能评价. 计算机学报,1999,22(11):1189~1195

3 林闯. ATM 网络基于队列长度阈值的传输调度. 软件学报, 1998, 9(4):316~320

4 林闯. ATM 网络一种实时传输调度和信元丢失控制的综合方案. 计算机学报, 1998, 21(4):33~340

5 林闯, 张元生. 基于随机高级 Petri 网的 ATM 网络接纳控制过程模型. 通信学报, 1998, 19(12):1~7

6 The ATM Forum. http://www.atmforum.com

7 The Proceedings of the Fifteenth Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE INFOCOM'96, San Francisco, California, 1996

# 第6章

# 拥塞控制

近些年来,有相当多的研究都试图扩展 Internet 的体系结构,为即将大量出现的实时多媒体应用提供服务质量(QoS)保证。有关 QoS 的研究引起了不少争议,一些基于网络中间节点上单个流状态的业务模型通常具有较复杂的实现机制,可扩展性是该类业务模型存在的严重问题;另外一些研究建议提倡在具有充足资源的尽力传输(best-effort)网络中,所有的问题都会迎刃而解,这种观点令人难以置信。但是大多数人认为:更多、更合理的控制机制对已有网络的稳定运行无疑将是至关重要的。其中一个最基本和最重要的要求就是防止网络出现拥塞崩溃,使网络运行在轻度拥塞的最佳状态,同时保证一定的公平性;在现有的网络体系结构中采用恰当的控制机制也有可能引入一定的区分业务等级,这种思路强调已取得较大成功的 Internet 固有的本质属性和最初的设计原则,而不是一味地放大现有体系结构中存在的不足与缺陷,这应该是一种较为合理的工程技术途径。

最初设计的 Internet 是非面向连接的分组交换网络,所有的业务分组被不加区分地在网络中传输,网络能给出的惟一承诺就是尽自己最大的努力传输进入网络的每一个分组,但它无法给出一个定量的性能指标,比如吞吐量、端到端时延和分组丢失率等参量的界。相应地,用户也无须进行业务许可请求,因此网络的性能不仅仅是其本身可以确定的,还受用户施加负载的影响。很显然,这种网络体系结构缺乏一定的隔离和保护机制,但是建立在这种体系结构之上的传统网络应用与网络协议具有较强的灵活性和适应性。

随着网络的发展,其应用领域不断拓展,应用模式不断丰富,加之商业化进程的推动,越来越需要对网络所传输的业务类型有一个较为具体和明确的定义,即所谓的网络业务模型。从早期的 ISDN,到 IntServ,再到后来的 DiffServ,这些都是结合应用的需要和技术的发展提出来的。无论最终采用哪种业务体系结构,其技术的核心都需要在恰当的层次和粒度上对流量进行必要的管理,其中包括接纳控制、流量成形、队列管理、调度和拥塞控制等诸多方面,但最基本和最核心的应该依旧是拥塞控制,因为很难想象一个时常有可能出现严重拥塞且无法及时加以恢复的网络能够实现良好的 QoS 保证。实施拥塞控制应该是其他 QoS 机制正常工作的必要前提[50]。

当然,拥塞控制的研究并非由 QoS 保证而起,它一直是分组交换网络中备受关注的一个技术热点,对于它的研究,除了具有延续性的技术意义之外,在强调业务模型的新网

络体系结构中,如何通过增强的拥塞控制为 QoS 的实现提供一定的便利,也是拥塞控制研究的目标之一。

拥塞控制本质上是一个如何共享资源的问题。在包交换网络中,所有的激活终端共享网络资源。这些资源包括节点处理能力、缓存空间和通信链路带宽。这三者中的任何一个都可能成为潜在的瓶颈,从而导致网络拥塞。从用户需求的角度来说,网络必须为所有用户的请求提供服务,然而用户的需求在传输起始时间、需求速率、持续时间上变化很大,在很多情况下还是突发的。从网络提供资源的角度来说,任何网络物理资源都有固定的上限能力。因此,用有限的资源去适应波动很大的用户需求,一定会出现网络资源不能满足用户需求的时候,此时就必须使用拥塞控制来管理用户流量对瓶颈资源的共享。

## 6.1 拥塞的定义

拥塞是一种持续过载的网络状态,此时用户对网络资源(包括链路带宽、存储空间和处理器处理能力等)的需求超过了其固有的容量。就 Internet 的体系结构而言,拥塞的发生是其固有的属性。因为在事先没有任何协商和请求许可机制的资源共享网络中,几个 IP 分组同时到达路由器,并期望经过同一个输出端口转发的可能性是存在的,显然,不是所有的分组都可以同时接受处理,必须有一个服务顺序,中间节点上的缓存为等候服务的分组提供一定保护。然而,如果此状况具有一定的持续性,当缓存空间被耗尽时,路由器只有丢弃分组。表面上,增大缓存总可以防止由于拥塞引起的分组丢弃,但随着缓存的增加,端到端的时延也相应增大,因为分组的持续时间(lifetime)是有限的,超时的分组同样需要重传。因此,过大的缓存空间倒有可能妨碍拥塞的恢复,因为有些分组白白浪费了网络的可用带宽。

拥塞导致的直接结果是分组丢失率提高,端到端时延加大,甚至有可能使整个系统发生崩溃。当网络处于拥塞崩溃状态时,微小的负载增量都将使网络的有效吞吐量(goodput)急剧下降。拥塞崩溃对 Internet 的威胁可以追溯到其早期的发展中,1984 年,Nagle[1] 报告了由于 TCP 连接中不必要的重传所诱发的拥塞崩溃,1986—1987 年间这种现象曾经多次发生,严重时一度使 LBL 到 UC Berkeley 之间的数据吞吐量从 32 Kbps 跌落到了 40bps[2]。除此之外,还有其他一些诱发拥塞崩溃的原因,例如,不可达分组(undelivered packets)导致的网络崩溃,它与前一种有所不同,不是一种稳定状态,当负载减小时,拥塞可以自动恢复。Floyd[3] 也报告了一种形式的拥塞崩溃现象,即分片拥塞崩溃,网络传输了大量的分组分片,但因为无法在接收端重装成有效的分组而只好将它们丢弃。网络传输大量用户不再需要的陈旧分组(stable packets)会导致另一种形式的拥塞崩溃现象。图 6.1.1 刻画了负载与吞吐量之间的关系:当负载较小时,吞吐量与负载之间呈线性关系;到达膝点(knee)之后,随着负载的增加,吞吐量的增量逐渐变小;当负载越过崖点(cliff)之后,吞吐量却急剧下降。通常将 knee 点附近称为拥塞避免区间,knee 和 cliff 之间是拥塞恢复区间,而 cliff 之外是拥塞崩溃区间。

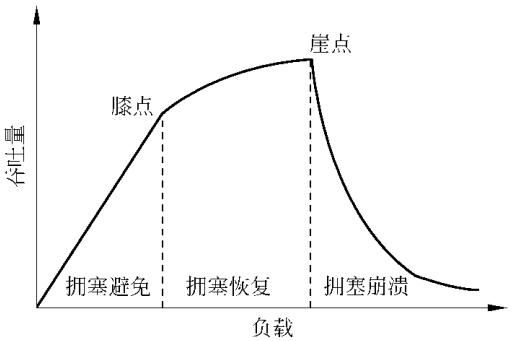



图 6.1.1　负载与吞吐量之间的关系曲线图

## 6.2　拥塞控制概述

为了最大限度地利用资源，网络工作在轻度拥塞状态时应该是较为理想的，但这也增加了滑向拥塞崩溃的可能性，因此需要一定的拥塞控制机制来加以约束和限制。

可以从两个方面考虑如何解决拥塞问题：一是增加网络资源；二是降低用户需求。前者一般通过动态配置网络资源来提高系统容量。如在高峰期增加接入线路，在卫星链路上增加发射功率来提高可用带宽，通过路径分裂（path splitting），用多条不同的物理路径来满足用户需求。在这些方法中，资源的使用者对这些拥塞机制都是不可见的，所有问题都由网络独立完成，而无须用户配合。降低用户需求主要表现在三个方面：拒绝服务、降低服务质量和调度。

（1）拒绝服务：在拥塞发生时，拒绝接纳新的用户请求。例如电话网中的忙音。此方法多用于面向连接的网络。

（2）降低服务质量：所有用户（包括新用户）在拥塞时降低其发送速率。例如分组交换网络中的滑窗算法等。

（3）调度：合理安排用户对网络资源的使用，保证总需求永远小于网络可用资源。例如轮循、加入优先级、资源预留等。

降低用户需求的具体机制与策略是多种多样的，选择何种策略依据拥塞的严重程度和持续时间而定。图 6.2.1 描述了拥塞持续时间和选择拥塞控制方法之间的关系。对频繁拥塞的网络，最好的方法是重新规划和设计以匹配需求的模式；对偶发的适度拥塞，连接接纳控制（CAC）是一种有效的措施。对拥塞持续时间少于连接持续时间的情形，端-端策略是可行的。例如，在连接建立时，通过协商，确定网络可以支持的业务量参数，借助漏桶算法实施监督，丢弃或标记过量分组，以防止用户对网络资源的无约束争用而可能导致的拥塞。这种监测控制措施是开环的，也有闭环的流量控制策略。网络中间节点以显式或隐式的方式向业务源反馈网络拥塞状态，业务源依此信息增加或减少它们注入网络的业务量，该机制既可以在数据链路层上以一跳接一跳（hop-to-hop）的方式实现，也可以在传输层以端到端（end-to-end）的形式完成。对于短暂的过载，hop-to-hop 比 end-to-end 更

有效 对于突发性的流量尖峰可能造成的拥塞,中间节点上适当的缓存是最好的解决办法。

| 拥塞持续时间 | 拥塞控制机制 |
| --- | --- |
| 长 | 链路容量规划、网络设计 |
| | 连接接纳控制 |
| | 基于负载的动态路由 |
| | 内容动态压缩 |
| 短 | 端-端反馈 |
| | 链路-链路反馈 |
| | 缓存 |

图 6.2.1 拥塞持续时间与拥塞控制机制的关系

有一点需要注意,解决短暂拥塞的有效方案并不适合于网络负载长期过载的情形,反之亦然。所以需要综合应用各种技术措施,从各个层次解决网络可能出现的拥塞。只不过从技术的角度出发,流量控制显得更有难度,受到的关注更多。

## 6.3 流量控制与拥塞控制的关系

很少有人否认 TCP 的流量控制是 Internet 正常运行的基础。据统计,Internet 上 95% 的数据流使用的是 TCP 协议[4],非弹性的 UDP 业务只占据很小的份额,围绕着 TCP 流量控制的拥塞控制一直是 Internet 研究的一个热点。

在 Internet 设计的初期,对于拥塞的控制是通过传输控制协议(transmission control protocol,TCP)中端到端基于滑窗的流量控制完成的。1988 年,Van Jacobson 在他那篇著名的论文[2]中指出了 TCP 在控制网络拥塞方面的不足,并提出了“ 慢启动 ”(slow start)和“ 拥塞避免 ”(congestion avoidance)算法,后来,它们被所有的 Internet 主机支持。在很长一段时间内,接收端驱动的 TCP 流量控制是惟一可行的拥塞控制方法,这也就是为什么术语‘ 流量控制 ’和‘ 拥塞控制 ’经常混淆的原因。实际上,前者只是实现后者的一种技术实现途径而已。在 ATM 网络中,两个术语之间的内涵和外延就相对比较清晰,当然,随着 IP 网络中拥塞控制机制的完善与进步,这两个概念之间的界限也越来越明显。

## 6.4 TCP 流量控制

### 6.4.1 TCP 流量控制的工作原理

要了解 TCP 流量控制的工作原理,得从分析 TCP 报文头的格式开始。随后简要介绍 TCP 的运行机制,并重点讨论 TCP 的流量控制和差错控制。

#### 6.4.1.1 TCP 报文头

TCP 报文头格式如图 6.4.1 所示。

<table>
<tr><td colspan="8">业务源端口</td><td colspan="2">目的地端口</td></tr>
<tr><td colspan="10">序列号</td></tr>
<tr><td colspan="10">确认序列号</td></tr>
<tr><td>数据偏移</td><td colspan="3">保留</td><td>GRU</td><td>KCA</td><td>HSP</td><td>TSR</td><td>NYS</td><td>NIF</td><td>窗口</td></tr>
<tr><td colspan="8">检验和</td><td colspan="2">紧急指针</td></tr>
<tr><td colspan="10">选项 + 填充</td></tr>
</table>

图 6.4.1　TCP 报文头格式

因为需要定义复杂的协议机制，TCP 报文头显得较大，其最小长度是 20 字节，各字段的意义如下：

- 源端口（16bit）：业务源端口号。
- 目的端口（16bit）：目的地端口号。
- 序列号（32bit）：当 SYN 标志没有置位时，本字段填入的是本报文段第 1 个数据字节的序号。在 SYN 标志置位的情况下，这个字段填入的是初始序号 ISN（initial sequence number），而第 1 个数据字段的序号变为 ISN + 1。
- 确认序列号（32bit）：如果控制位中的 ACK 被置位，该字段的值表示接收方期望收到的下一个数据字节的序号。
- 数据偏移（4bit）：报文头中 32 位字的个数。它表明数据从何处开始。
- 保留字段（6bit）：保留给将来使用，通常全置为零。
- 控制位（6bit）：
  URG：紧急（URGENT）指针字段有效。
  ACK：确认字段有效。
  PSH：急迫交付（PUSH）功能。
  RST：重建连接。
  SYN：序号同步。
  FIN：发送端不再发送数据。
- 窗口（16bit）：接收方期望接收的字节数。
- 检验和（16bit）：差错检测编码。
- 紧急指针（16bit）：以相对于序列号的正偏移确定紧急数据当前的位置，当然，必须是在相应的控制位置位时才有效。
- 选项（长度可变）：指定可选的特征。

### 6.4.1.2　TCP 的滑窗机制

与大多数提供流量控制的协议一样，TCP 也使用滑动窗口机制，但与其他协议，如 LLC，HDLC 和 X.25 等稍有不同，它将数据的确认过程与允许发送数据的通告分开处理。被传输数据的每个字节都被赋予一个序列号，当发端传输报文段时，为数据字段的每一个

字节编排序列号。用(A=i,W=j)的报文形式确认数据的传输,具体含义如下:

- 序号为 i-1 以及以下各字节均已得到确认;下一个期望收到的字节的序号是 i。
- 允许对方再发送一个窗口(W)共 j 字节的数据,这 j 个字节的序号为 i 到 i+j-1。

图 6.4.2 描述了上述 TCP 的滑窗机制。

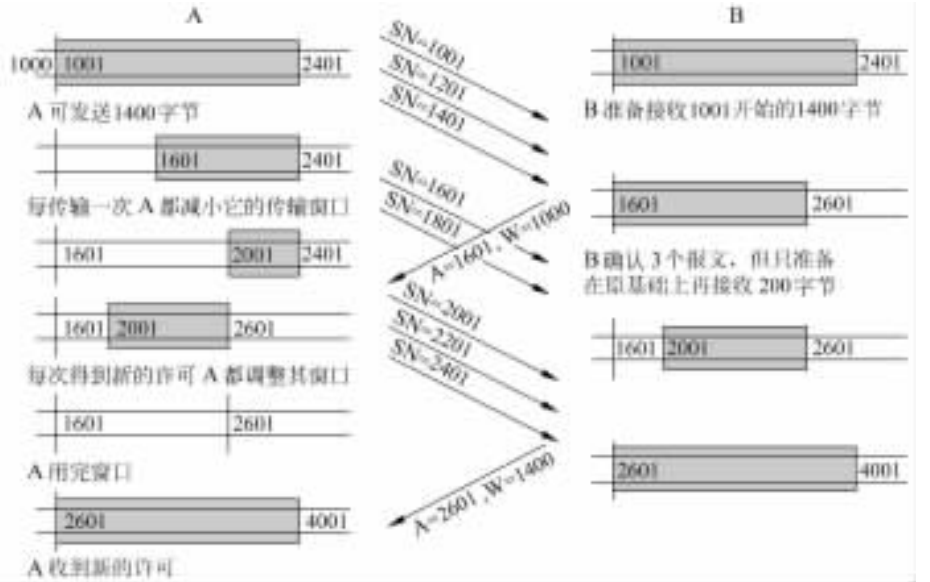



图 6.4.2 TCP 的滑窗机制

为简单起见,这里,图 6.4.2 中只画出了一个方向的数据流,并假设每个报文段中传送 200 字节的数据。初始化时,通过连接建立过程,发送及接收序号得到同步,A 被给予了 1400 字节的初始传输量,序号从 1001 开始。在发送了 3 个报文段共 600 字节数据之后,A 将其窗口缩小到 800 字节(序号从 1601 ~ 2400)。在收到这些报文段之后,B 确认已收到字节序号截止于 1601,并重新为 A 分配 1000 字节的许可,这意味着 A 可以发送从 1601 ~ 2600 的字节(共 5 个报文段)的数据。然而,在 B 的确认报文到达 A 时,A 又发送了两个报文段,包含字节 1601 ~ 2000(依据初始分配这是允许的)。因此,A 在这时允许发送的数据仅剩 400 字节。随着交互的继续,A 在每次发送后都将它的窗口后沿向前推,而在得到新的允许后,则将其窗口前沿向前推,周而复始,直到整个会话业务结束,连接被拆除。

接收方需要采用某种策略来决定允许发送方传送多少数据。保守的办法是仅允许对方传送缓存空间可以容纳的报文段。如图 6.4.2 所示,第一个确认报文意味着 B 有 1000 字节的缓存容量,而第二个报文意味着 B 有 1400 字节的可用缓存容量。在时延带宽的乘积较大的情况下,接收方通过预支缓存空间可以提高吞吐量。例如,如果接收方的缓存已满,但是它预期在一个往返时间内就可以释放掉 1000 字节的缓存空间,就可以立即发送 1000 字节的许可量。

#### 6.4.1.3　重传策略

与链路控制协议一样，TCP 不仅包括一个流量控制方案，而且还包括一个差错控制方案。在 TCP 中，没有诸如在链路控制协议中可以找到的 REJ 和 SREJ 等否认应答。TCP 完全依靠确认应答，当确认在给定的一个超时时段中报文没有到达时就进行重传。在以下两种情况下，报文段应该重传。第一种情况是报文段可能在传送中被损坏，但依旧到达了目的地，报文段中携带的“校验和”可以用来检测差错并丢弃该报文。另一种更常见的意外情况是报文段根本就未能到达目的地。在上述任何一种情况下，发送端都不可能了解到报文段是否成功传输。如果一个报文没有成功到达，那么就不会有 ACK 确认信息，于是发送方就必须重传。为了处理这种情况，就要为发送出去的每个报文设置一个定时器。如果在收到对该报文的确认之前定时器超时，则发送端必须重传。

TCP 中一个关键的设计问题就是确定重传定时器的数值。如果该值太小，会造成许多不必要的重传，从而浪费网络带宽。如果太大，协议对于报文段丢失的反应就会很迟缓。定时器应该设定为一个比往返时延稍大的值。当然，往返时延即使在固定的网络负载情况下也可能是变化的。更糟糕的是，时延的统计特性会随互连网状况的变化而变化。解决这一问题的方案是采用自适应策略。下面介绍 RFC 793 中规定的时间计算方法。

对观察到的一些报文段的往返时间简单地取平均值，如果这个平均值精确地预测了将来的往返时延，那么由此得到的重发定时器就会产生良好的性能。简单平均方法可以表达如下：

$$\mathrm{artt}(k+1) = \frac{1}{k+1}\sum_{i=1}^{k+1}\mathrm{rtt}(i) \tag{6.4.1}$$

其中，rtt(i) 是对第 i 个传输报文段所观察到的往返时间，而 artt(k) 是对头 K 个报文段所取的平均往返时间。

这个表达式可以重写为

$$\mathrm{artt}(k+1) = \frac{k}{k+1}\mathrm{artt}(k) + \frac{1}{k+1}\sum_{i=1}^{k+1}\mathrm{rtt}(i) \tag{6.4.2}$$

这一迭代公式省去了每次求和式的麻烦。注意，式(6.4.1)中的每一项都被赋予了相等的权值：也就是说，每一项是被乘以相同的常数 1/(k+1)。通常我们希望对更近期的采样值赋予更大的权值，因为它们更可能反映将来的行为。基于一个时间序列的过去值预测其下一个值有一种常用的技术，即指数加权移动平均(EWMA)，RFC 793 中就采用了这一技术。

$$\mathrm{srtt}(k+1) = \alpha \times \mathrm{srtt}(k) + (1-\alpha) \times \mathrm{rtt}(k+1) \tag{6.4.3}$$

其中，srtt(k) 称为往返时间的平滑估计，$\alpha$ 为常数（$0 < \alpha < 1$）。

TCP 维护了一个队列，以缓存已经发送但尚未得到确认的报文段。如果在给定时间内没有收到确认，那么它就重传相应的数据报文段。具体策略可以采取以下三种之一：

(1) 重传队头报文段(first only)：维持一个重传定时器，如果收到确认，就将相应的一个或多个报文段从队列中去除并重置定时器。如果定时器超时，重传队列首部的报文段并重新设置定时器。

(2) 成批重传(batch):维持一个重传定时器。如果收到确认,就将相应的一个或多个报文段从队列中去除并重新设置定时器。如果定时器超时,则重传队列中的所有报文段并重新设置定时器。

(3) 单个重传(individual):为对队列中的每个报文段单独设置一个定时器。如果收到一个确认,则将相应的一个或多个报文段从队列中去除并取消相应的一个或多个定时器。如果任一个定时器出现超时,则单独重传相应的报文段并重新设置其定时器。

#### 6.4.1.4 确认策略

当一个数据报文段按顺序到达时,接收端有两种可能的确认方式:

(1) 立即确认(immediate):当接收到数据时,立即传输一个包含相应确认号的空(无数据)报文段。

(2) 积累确认(cumulative):当接收到数据时,记录需要的确认,等待适当的数据报文携带该确认信息。为避免拖延太久,设置一个定时器;如果在发送确认之前定时器超时,则立刻传输一个包含适当确认号的空报文段。

立即确认策略比较简单,可以使发送端及时了解报文传输的状态,进而限制不必要的重传,然而,它却产生了额外的报文传输,增加了网络负载。因为立即确认策略的潜在开销,人们通常使用积累策略。当然,这需要接收端进行更多的处理,同时,发送端也更难估计往返时延。

### 6.4.2 自同步机制

自同步机制是 TCP 流量控制中的一个重要机制,它和"加性增加倍乘减小"一起构成了 TCP 拥塞控制的基础。V. Jacobson 在他那篇著名的论文中,用图 6.4.3 阐述了这一机制的工作原理[2]。前面的论述告诉我们:TCP 源端的发送速率是由 ACK 的速率确定的。但 ACK 的到达速率由源端与目的端之间往返路径上的瓶颈确定,该瓶颈可能是网络,也可能是目的地自身。

图 6.4.3(a)表示瓶颈出现在网络中。网络被抽象为一个连接源端和目的端的管道。管道的粗细与数据速率成比例。连接源端和目的端的链路都是高速链路,中间较细一段管道代表形成瓶颈的低速链路。每个报文段都用一个矩形表示,其中矩形的面积与报文段中所含比特数成比例。这样,当一个报文段被压进一条较细狭的管道时,它就会在时间上扩展开来。时间 Pb 表示最慢链路上的最小报文段的间隔。当报文段到达目的端时,即使数据速率增加,此间隔也将保持不变,这是因为到达间隔时间没有变化的缘故。因此,接收方的报文段间隔 Pr 就等于 Pb。如果目的端在报文段到达时就做出确认,那么 ACK 报文段离开接收方的间隔就由报文段到达间隔决定,所以有 Ar = Pr。因为时隙 Pb 对 ACK 报文段肯定足够大,所以 Ab = Ar,这样返回的 ACK 就起到同步信号的作用,使得在稳定状态时,发送方的报文速率和 ACK 的到达速率相匹配,即发送方的报文段速率等于其传送路径上最慢链路的速率。这样一来,TCP 就能自动感知网络瓶颈,并对其流量做出调整,这一过程称为 TCP 的自同步机制。

这种自同步机制在瓶颈处于接收方时也同样有效。接收方由于其本身的处理能力或

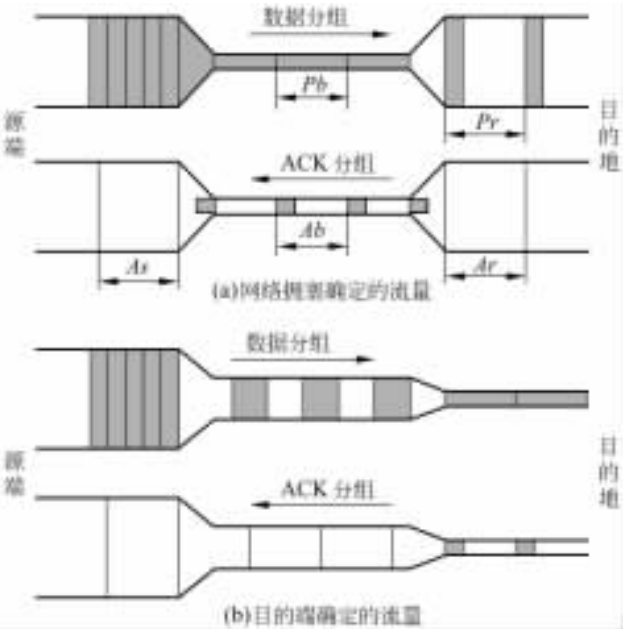



图 6.4.3　TCP 自同步机制

者其他连接上到来的报文段造成的压力不能及时接收给定连接上的报文段。如图 6.4.3(b)所示,假定网络中的最慢链路的带宽相对较高,大约是源端数据速率的一半,但目的端的管道较窄。此时,ACK 的发送速率就等于目的端的接收能力,通过 ACK 流速控制源端的速率,保证报文段只能以目的端可以处理的速度到达。

图 6.4.3 同时表明一个重要的事实:源端无法知道 ACK 的同步速率反映的是网络负载状况,还是目的端的处理状态。如果因为网络拥塞而导致 ACK 的到达变慢,那么源端应该以比 ACK 速率更慢的速率传输报文段,以有助于缓解网络的拥塞状态;但如果是由于目的端处理能力的限制导致 ACK 速率变小,那么发端以目前的速率传输报文段是恰当的。

## 6.4.3　加性增加倍乘减小

TCP 流量控制中使用的"加性增加倍乘减小"(additive increase and multiplicative decrease, AIMD)窗口调节策略是网络流量控制的一个基本机制,得到了较为广泛的应用。本节将对它的有效性、公平性和收敛性加以分析和讨论。

设业务源 i 在时刻 k 的发送速率为 $x_i(k)$,得到网络拥塞状态的反馈信息为

$$y(k) = \begin{cases} 0 \Rightarrow \text{增加发送速率} \\ 1 \Rightarrow \text{降低发送速率} \end{cases}$$

如果业务源依照线性规则调节发送速率,我们有:

$$x_i(k+1) = \begin{cases} a_I + b_I x_i(k), & y(k) = 0 \\ a_D + b_D x_i(k), & y(k) = 1 \end{cases} \tag{6.4.4}$$

其中,$a_I$, $a_D$, $b_I$, $b_D$为常数,$x_i(k)$表示业务源 i 在时刻 k 的发送速率。根据 $a_I$, $a_D$, $b_I$, $b_D$ 的不同,可以定义 4 类不同的控制算法,见表 6.4.1,其中,I 表示增长,D 表示减小;A 表示加性,M 表示乘性。

表 6.4.1 4 种控制算法及其参数范围

| 算法名称 | $a_I$ | $a_D$ | $b_I$ | $b_D$ |
|---|---|---|---|---|
| MIMD | 0 | 0 | $b_I > 1$ | $0 < b_D < 1$ |
| AIAD | $a_I > 0$ | $a_D < 0$ | 1 | $b_D = 1$ |
| AIMD | $a_I > 0$ | 0 | 1 | $0 < b_D < 1$ |
| MIAD | 0 | $a_D < 0$ | $b_I > 1$ | $b_D = 1$ |

在以上组合中,哪一种更优越,需要从有效性、公平性和收敛性等几个方面进行比较。

- 有效性

由 6.1 节我们知道,网络在膝点处性能达到最佳值。设达到此状态的最佳发送速率为 $X_{目标}$。从图 6.1.1 可知,此发送速率仅仅与总速率(网络负载)有关,而与流间速率的具体分配无关。因此只要能保持总速率接近 $X_{目标}$,则不同的分配算法的有效性是一致的。即有效性的目标是:

$$\text{当 } k \to \quad \text{时,} \sum_i x_i(k) = X_{目标}$$

- 公平性

当多个不同的流共享多个瓶颈资源,而每个流的价值取向不同时,公平性是一个非常复杂的问题。然而,当所有流共享惟一的瓶颈资源,而且网络对所有的流都等同视之时,公平性可以很简单地退化为在所有流间平均分配的问题。当流量分配偏离此值时,必须用一个量化的标准来度量其公平的程度。如下定义公平指数 F(X):

$$F(X) = \frac{\left(\sum_i x_i\right)^2}{n\sum_i x_i^2} \tag{6.4.5}$$

其中,n 为流的总数。

此公平性定义满足以下性质:

(1) F(X)的值域为[0, 1]。当所有流平均分配时($x_1 = \cdots = x_n$),F(X)为最大值 1;在最不公平的时候($x_i = A$;$x_j = 0$,当 $j \neq i$ 时),F(X)为最小值 1/n。当 n 趋近于无穷大时,F(X)趋近于 0。

(2) 此值无量纲。x 可以取任何单位。

(3) 当所有流的速率按比例扩大(或缩小)k 倍时,公平性不变。

(4) F(X)为连续函数。分配时公平性的任何微小变化,都可以在 F(X)上体现出来。

(5) 当 n 个用户中的 k 个平均分配资源,而 n - k 个用户不占用任何资源时,公平性的值为 k/n。

- 收敛性

对任何一种算法，我们都希望它能较快地收敛到目标状态。一般地，定义收敛速度为从任何初始状态到达目标状态所花的时间。我们期望能尽量减少这个时间，以提高系统响应速度。然而，由于二进制反馈的特性，系统并不能停留在惟一的状态，而是在稳定时振荡于最优状态附近。振荡的幅度表明控制的平滑程度。因此，在这个特定情况下，讨论收敛性有这样一层含义：系统能否稳定（即在微小范围内振荡，而不会逃逸出此范围）；从初始态到达稳态的时间长短；振荡范围的大小。

为简单起见，考虑仅有两个用户和一个瓶颈资源的系统。系统状态可如图 6.4.4 所示。图中任何一点表示分配给用户 1 和用户 2 的资源（即用户可用的发送速率）。当 $x_1 + x_2 = X_{目标}$ 时，系统效率最高，对应于图中的一条线段，称为效率线。当 $x_1 = x_2$ 时，系统公平性最高，对应于图中的公平线。公平线和效率线的交点为系统的控制目标，称为最优点。与效率线平行的直线，效率为一定值，称为等效率线。过原点的任何一条直线，其公平性 F（X）相同，称为等公平线。效率线的上方为过载区，下方为轻载区。

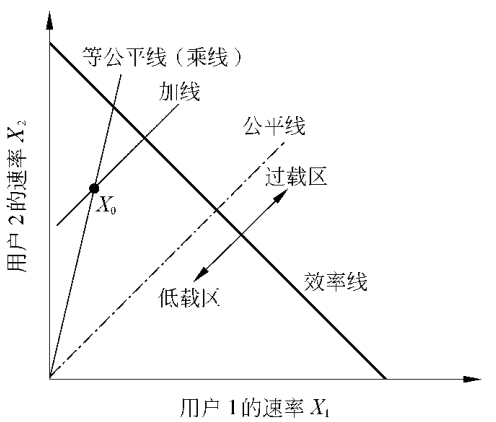



图 6.4.4　两用户的状态向量表示

我们的目标是找到一种控制算法，使系统能从任何初始点都能收敛到最优点附近。从图 6.4.4 可以发现以下特点：

- AI 和 AD 的轨迹为 45°的直线：AI 使公平性增加，同时向过载区移动；AD 使公平性降低，同时向轻载区移动。
- MI 和 MD 的轨迹为过原点的直线：MI 和 MD 在等公平线上移动，故保持公平性不变；MI 使轨迹向过载区移动，而 MD 使轨迹向轻载区移动。

因此，单纯的 AIAD 算法和 MIMD 算法的轨迹为一条直线，其能达到的最佳平衡点与初值有关，而并不能保证一定到达最优点。

剩下的算法还有 MIAD 和 AIMD 两种。图 6.4.5 和图 6.4.6 是其从初始点 $X_0$ 开始运行的轨迹图。

从图中可以发现，MIAD 算法是背离最优点而去的。因此，惟一可用的就只有 AIMD 算法。图 6.4.4 中起始点在公平线上方，用同样的方法也可以画出起始点在公平线下方的收敛图。因此，矢量图轨迹证明了 AIMD 算法一定能够收敛到最优点附近，而与初始值无关。这就是选取 AIMD 算法的原因。

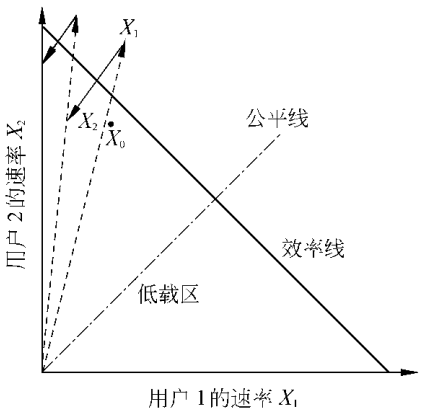



图 6.4.5 MIAD 算法轨迹图

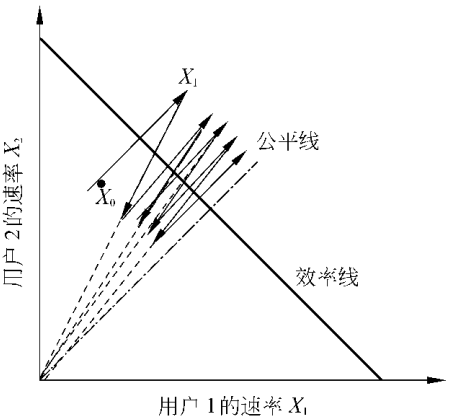



图 6.4.6 AIMD 算法轨迹图

定性分析了 AIMD 算法的公平性和有效性之后,需要对工作参数 $a_I$, $a_D$, $b_I$, $b_D$的取值范围加以界定。因为理想情形下,AIMD 算法的每一步响应都是向最佳的方向迈进,下面依据这一点来确定参数的工作范围。有效性控制所有流的总速率,因此应满足

$$\begin{aligned} y(k) = 0 &\Rightarrow \sum_i x_i(k+1) > \sum_i x_i(k) \\ y(k) = 1 &\Rightarrow \sum_i x_i(k+1) < \sum_i x_i(k) \end{aligned} \tag{6.4.6}$$

把 $x_i$代入式(6.4.4)可得

$$\begin{aligned} na_I + (b_I - 1)\sum_i x_i(k) > 0 &\Rightarrow b_I > 1 - \frac{na_I}{\sum_i x_i(k)} \\ na_D + (b_D - 1)\sum_i x_i(k) < 0 &\Rightarrow b_D < 1 - \frac{na_D}{\sum_i x_i(k)} \end{aligned} \tag{6.4.7}$$

除此之外,公平性要求 $F(X(k+1)) > F(X(k))$,代入 $F(X)$的定义可得

$$\begin{aligned} F[X(k+1)] &= \frac{[\sum_i x_i(k+1)]^2}{n[\sum_i x_i^2(k+1)]} = \frac{[\sum_i a + bx_i(k)]^2}{n[\sum_i (a + bx_i(k))^2]} = \frac{[\sum_i c + x_i(k)]^2}{n[\sum_i (c + x_i(k))^2]} \\ &= F(X(k)) + (1 - F(X(k))) \times \left[1 - \frac{\sum_i x_i^2(k)}{\sum_i (c + x_i(k))^2}\right] \end{aligned} \tag{6.4.8}$$

其中 $c = a/b$。因为 $F(X)$和 $1 - F(X)$恒大于等于 0,因此要公平性目标成立,只需 $1 - \dfrac{\sum_i x_i^2(k)}{\sum_i (c + x_i(k))^2} > 0$即可。由于 $x_i(k) \geqslant 0$,因此等价于要求 $c > 0$。可得

$$\frac{a_I}{b_I} \geqslant 0 \quad \text{且} \quad \frac{a_D}{b_D} > 0$$
$$\text{或} \quad \frac{a_I}{b_I} > 0 \quad \text{且} \quad \frac{a_D}{b_D} \geqslant 0 \tag{6.4.9}$$

由 $x_i(k) \geqslant 0$ 可知，a，b 非负。由式(6.4.9)、式(6.4.7)和非负性可得，

$$a_I \geqslant 0 \qquad b_I \geqslant 0$$
$$a_D \geqslant 0 \quad 0 \leqslant b_D < 1 \tag{6.4.10}$$

依据速率调节规则，我们有：

$$y(k) = 0 \Rightarrow x_i(k+1) > x_i(k), \quad \forall i,$$
$$y(k) = 1 \Rightarrow x_i(k+1) < x_i(k), \quad \forall i,$$

代入式(6.4.4)

$$a_I + (b_I - 1)x_i(k) > 0, \quad \forall x_i(k),$$
$$a_D + (b_D - 1)x_i(k) < 0, \quad \forall x_i(k)。$$

给定 $x_i(k) = 0$ 可得 $a_I > 0$，$a_D < 0$。假定 $x_i(k) \to \infty$，可得 $b_I > 1$。结合式(6.4.10)，我们有

$$a_I > 0, \qquad b_I \geqslant 1,$$
$$a_D = 0, \qquad 0 \leqslant b_D < 1。$$

这就是 AIMD 算法 4 个参数的取值范围。在 TCP 中，$a_I = 1$，$a_D = 0$，$b_I = 1$，$b_D = 1/2$。

## 6.4.4　重发超时管理

重发超时 RTO(retransmission time out)的管理对 TCP 拥塞控制至关重要，必须尽可能准确地估计往返时间的大小，为此，除使用前面讨论的 RTT 估计技术之外，还发展了三种技术来增加估计的精度，它们是：RTT 方差估计、指数 RTO 退避和 Karn 算法。

### 6.4.4.1　RTT 方差估计(Jacobson 算法)

指数加权滑动平均虽能适应 RTT 的变化，但当 RTT 的方差较高时便无法很好地处理，这种 RTT 方差变化来自于以下三个方面：

(1) 如果 TCP 连接上的数据速率较低，传输时间与传播时间相比，前者相对较大，此时，数据报大小的变化是导致 RTT 方差增大的一个不可忽略的因素。

(2) 大量突发性业务涌入网络也会造成 RTT 的突变。

(3) 积累确认机制的使用也会增加 RTT 的方差。

起初，TCP 协议是通过对 RTT 估计值乘以一个常数因子来消除 RTT 可能出现突变的影响：

$$\text{RTO}(k+1) = \beta \times \text{srtt}(k+1) \tag{6.4.11}$$

其中，$\beta$ 通常取 2。在稳定环境中 RTT 的方差很小，而在一个不稳定的环境中，$\beta$ 取 2 可能并不足以防止不必要的重传。

更有效的办法是估计 RTT 的变化，并记入 RTO 的计算。可以考虑计算 RTT 的标准差，但要涉及平方和平方根运算。为避免这一点，可以用如下简单的平均方法来估计标准差：

$$aerr(k+1) = rtt(k+1) - artt(k) \tag{6.4.12}$$

$$adev(k+1) = \frac{1}{k+1}\sum_{i=1}^{k+1} |aerr(i)|$$

$$= \frac{k}{k+1}adev(k) + \frac{1}{k+1}|aerr(k+1)| \tag{6.4.13}$$

其中 artt(k) 是式(6.4.1)中定义的简单平均,而 aerr(k) 是时刻 k 的采样平均偏差。Jacobson 建议用与 srtt 估计中完全相同的指数加权滑动平均技术来计算 RTT 的平均偏差,并提出了一套完整的算法:

$$\left.\begin{aligned} srtt(k+1) &= (1-g)\times srtt(k) + g\times rtt(k+1)\\ serr(k+1) &= rtt(k+1) - srtt(k)\\ sdev(k+1) &= (1-h)\times sdev(k) + h\times|serr(k+1)|\\ RTO(k+1) &= srtt(k+1) + f\times sdev(k+1) \end{aligned}\right\} \tag{6.4.14}$$

与 RFC 793 的定义一样(见 6.3 节),srtt 是 RTT 的指数加权滑动平均值,其中 1 - g 等效于 $\alpha$。但这里不用常数乘以 srtt,而是采用估计的平均偏差的倍数加到 srtt 上来形成重发定时器。基于实验,Jacobson 在他的原始论文中提出对各常数使用下列数值[2]:

$$g = 1/8;\ h = 1/4;\ f = 2。$$

在进行深入研究之后,Jacobson 又建议将 f 的值改为 4。这个值是目前 TCP 实现中所用的标准数值。经验证明:Jacobson 提出的定时器管理算法可以显著提高 TCP 的性能。

#### 6.4.4.2 指数 RTO 退避

如果 TCP 源端察觉一个报文段发生超时,就必须重传该报文段。RFC 793 规定,该重传报文段使用与原始报文段相同的 RTO 值。这有点不合乎情理,因为超时可能是由于网络拥塞而引起的(拥塞常常表现为分组的丢失或往返时间加长),保持相同的 RTO 值并不明智。举例来说,假定来自不同业务源 TCP 连接同时向网络注入流量,网络某局部发生了拥塞以致许多连接上的报文段被丢失了或延误时间超过了连接的 RTO 定时。这样,大约在几乎相同的时间内,许多报文段会被重传进入网络,使得拥塞无法解除,甚至有可能进一步恶化。所有源端然后再等待一个本地的 RTO 时间,并再次重传。这种操作模式可能造成网络拥塞状况延续下去。明智的策略是,TCP 源端在重传同一报文段时增加其 RTO,这一过程称为退避(backoff),即对一个报文段进行了第一次重传之后,TCP 源端在进行第二次重传之前要等待更长的时间,以使网络有机会消除拥塞。如果还要进行重传,在第三次超时重传之前将要等待更长的时间,使网络的拥塞状态得以充分的恢复。实现 RTO 退避的一种简单方法是,对一个报文段的每次重传都将 RTO 乘以一个常数,即

$$RTO \leftarrow RTO \times q \tag{6.4.15}$$

式(6.4.15)使 RTO 随重传次数而呈指数增长。q 通常为 2,取该值时,这种方法称为二进制指数退避(binary exponential backoff),这与以太网 CSMA/CD 协议中所用的是同一种方法。

#### 6.4.4.3 Karn 算法

如果不发生报文段的重传,依照 Jacobson 提出的算法计算 RTO 是很简单的。但如果

发生超时重传，随后收到了一个确认消息，情况就变得相对复杂，会有两种可能：

（1）ACK 是对报文段第一次传输的确认。此时，RTT 仅比期望的长一些，但却是网络状况的精确反映。

（2）ACK 确认的是重传报文段。

TCP 端系统无法区分这两种情况。如果发生的是第二种情况，而只简单地将 RTT 计算为第一次传输到收到 ACK 之间的时间，就会比实际的 RTT 要大许多。用这个错误的 RTT 代入式（6.4.14），会得到偏大的 srtt 和 RTO。此外，这个效应还会通过迭代算法向后传播许多次。

更糟糕的是，如果将 RTT 视为第二次传输到收到 ACK 之间的时间，但实际上，这个 ACK 是对第一次传输的确认，那么得到的 RTT 值就比实际值小许多，进而导致 srtt 和 RTO 值过小，并有可能引发正反馈效应，导致更多的重传和更多的错误测量。

Karn 算法采用以下规则解决了上述问题：

（1）不使用重传报文段的 RTT 更新 srtt 和 sdev（见式（6.4.14））。

（2）当发生重传时，采用式（6.4.15）计算退避 RTO。

（3）对后续各报文段使用退避 RTO 值，直到收到了一个未被重传报文段的确认消息为止。

当收到一个未被重传报文段的确认消息之后，重新激活 Jacobson 算法计算之后的 RTO。

## 6.4.5　窗口管理

窗口管理是 TCP 流量控制的关键。TCP 流量控制多个改进和增强版本的焦点都集中在窗口管理上。起初，它只包含了慢启动和拥塞避免两种机制，后来为了完善性能，又不断增加了快速重传和快速恢复机制，形成了称为 TCP Tahoe，TCP Reno 和 TCP SACK 等多种版本。

### 6.4.5.1　慢启动

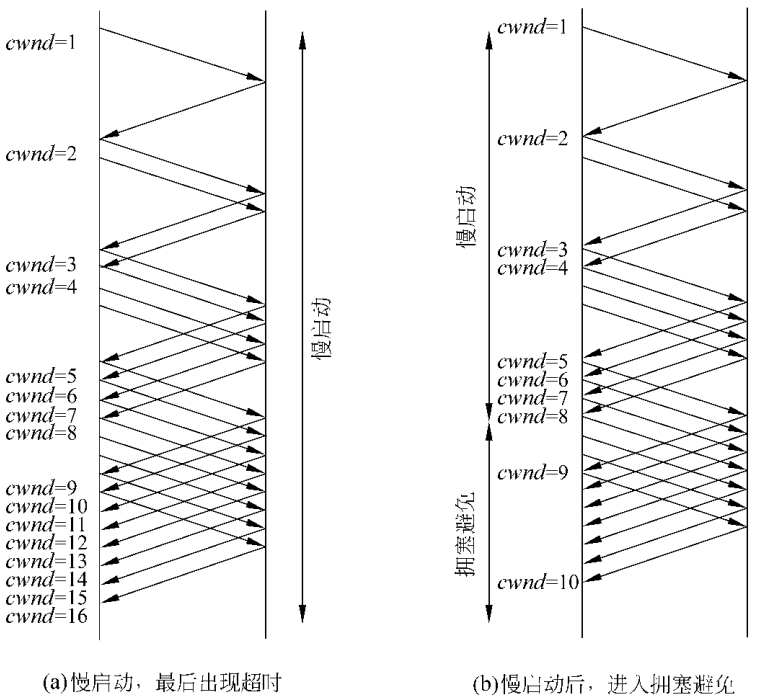
TCP 使用的发送窗口越大，TCP 源端在必须等待一个确认之前可以发送的报文段就越多。在正常情况下，TCP 的自同步机制会为 TCP 确定适当的速率。然而，当一个连接刚刚初始化时，不存在同步机制使其稳定工作在适当的速率。一种理想的策略是让 TCP 源端从某个相对较大的初始窗口开始，并希望能快速逼近网络所能支持的窗口大小，但这比较危险，因为业务源在通过超时感知到网络拥塞之前，可能已经向网络注入了相当过量的流量。因此，保守的策略是逐渐增加窗口，直到同步机制起作用为止。基于此，Jacobson 设计了一种称为慢启动窗口调节策略。定义拥塞窗口（cwnd）作为辅助变量，该窗口以报文段而非字节来计算大小。任意时刻，TCP 源端发送报文的速率受限于如下关系式[5]：

$$\min\{\text{cwnd},\text{awnd}\} \tag{6.4.16}$$

其中，cwnd 是拥塞窗口，由源端确定它的大小；awnd 为接收端宣告允许的窗口大小。一条新建连接 cwnd 被初始化为 1，也就是说 TCP 源只被允许发送 1 个报文段，等待确认后，再传输第二个报文段。每次收到一个确认，cwnd 的值就被加 1，一直加到某个最大值为

止。借助慢启动机制,TCP 源端可以有效地探测网络负载状态,确保不会盲目地向已经出现拥塞的网络注入过量业务量。

“慢启动”这一术语似乎有点名不符实,因为 cwnd 实际上是以指数规律增长的。当第一个 ACK 到达时,TCP 将 cwnd 增加到 2,允许发送两个报文段。当这两个报文段被确认之后,TCP 自然可以发送 4 个报文段,依此类推。图 6.4.7(a)示意了慢启动过程。




(a)慢启动,最后出现超时　　(b)慢启动后,进入拥塞避免

图 6.4.7　慢启动与拥塞避免

### 6.4.5.2　拥塞避免

慢启动算法在初始化连接时工作得很有效。它使得 TCP 发送方可以快速地为连接确定合理的窗口大小。但出现拥塞后是否有效则值得研究。我们设想一个新建连接经过慢启动过程,在某个时刻 cwnd 达到 awnd 之前或之后,一个报文段发生丢失,出现超时,产生拥塞信号,但拥塞的程度并不清楚。因此,保守的处理方法是复位 cwnd = 1,并重新开始慢启动过程。这似乎是一种合理的保守方案,但实际上它还不够保守。Jacobson 指出,“让网络进入饱和状态很容易,但让网络从中恢复却相当难”。换句话说,一旦发生拥塞,要得以恢复是需要很长时间的。因此慢启动中 cwnd 的指数增加就显得有点太激进,它可能会加重拥塞程度。Jacobson 又设计了拥塞避免机制,在探测到网络发生拥塞之后,改变 cwnd 的调节策略,由指数增加变为线性增长。具体操作如下。

一旦探测到拥塞超时:

(1) 将慢启动阈值(ssthresh)设为当前拥塞窗口的一半,即 ssthresh = cwnd/2。

(2) 置 cwnd = 1,并启动慢启动过程,直到 cwnd = ssthresh。在该阶段,cwnd 每收到一

个 ACK 就加 1。

（3）当 cwnd > ssthresh 时，则每经过一个往返时间对 cwnd 加 1。

图 6.4.7(b)示意了拥塞避免机制。拥塞超时发生，ssthresh 的值被设置为 8，在达到该值之前，TCP 使用指数慢启动过程来扩展拥塞窗口；在此之后，cwnd 则按线性规律增加。相比图 6.4.7(a)，将 cwnd 恢复到最初，需花 4 个 RTT，而图 6.4.7(b)中要花 11 个 RTT。

### 6.4.6　TCP Tahoe

为了防止对 RTO 估计过小可能出现的正反馈，通常将重传定时器设得比 RTT 大许多，但这也有一个不利因素：如果一个报文段发生丢失，TCP 便不能及时重传。如果目的端采用按序接受策略，那么就会导致许多报文段的丢失。即便使用按窗口接受的策略，不及时重传也会引起问题。设想 A 传输了一系列的报文段，而头一个报文段丢失了。只要其发送窗口不空，而且 RTO 没发生超时，A 就可以继续传输而不用等待收到确认。B 除第一个报文段以外收到了所有其他报文段，但 B 必须将所有到达的报文缓存起来，因为它无法正常地将数据交给应用程序而清除其缓存空间。如果对丢失报文段的重传被延迟太久，B 将不得不丢弃新到来的报文段。

可以采用快速重传克服以上缺陷，用 Tahoe 命名引入该机制的 TCP 流量控制算法。如果 TCP 目的端收到一个失序报文段，它立即发出一个对最后一个收到的按序报文段的 ACK 消息。对于接着到来的报文段也重复上述的发送过程，直到丢失的报文段到达，填补了缓存中的空隙。一旦空隙得以填充，目的端将对所有迄今为止按序接收的报文段发送一个积累 ACK 消息。当源端 TCP 收到一个重复的 ACK 时，意味着发生了下列两种事件之一：(1)被确认的报文段后面的报文段被延迟，以至于它最终失序到达。(2)该报文段丢失。在情形(1)中，延迟的报文段最后确实到达目的地，因此不需要重传；但情形(2)却不同，一个重复的 ACK 消息可以被视为是一个预警信号，它告诉源端一个报文段已经丢失了，需要重传。为确信发生的是情形(2)而不是情形(1)，TCP 发端要等待收到同一报文段的 3 个重复 ACK(即总共收到同一报文段的 4 个 ACK)之后，才认定随后的报文段已被丢失的可能性很大，因此应该立即启动重传，而不是等着超时才重传。

图 6.4.8 示例了快速重传过程。A 发送一系列报文段，每个报文段都包含 200 个字节的数据。报文段 1201 出现丢失，正常情况下，A 并不对此做出响应，直到经过了一个 RTO 时间。它会接着发送报文段，直到其窗口关闭为止。B 在收到报文段 1001 后，用 ACK1201 做出确认；接着，B 收到报文段 1401。因为它是失序到达的，B 会重复发一个 ACK1201 消息，并在收到报文段 1201 之前每收到一个到来的报文段就重复发一次 ACK1201。在 A 收到对报文段 1001 的 4 个 ACK 时，它已经发送了报文段 1201 之后的 7 个报文段。A 立即重传报文段 1201，然后继续进行正常的发送过程。

### 6.4.7　TCP Reno 和 TCP NewReno

Reno 在 Tahoe 的基础上增加了“快速恢复”(fast recovery)算法来提高拥塞恢复的效率。如果将探测到分组丢失至接收到重传分组之间的间隔定义为快速重传/快速恢复

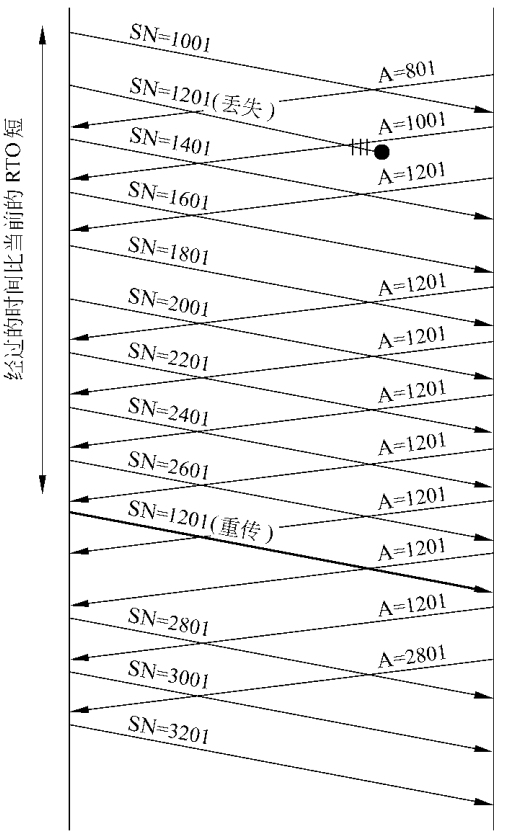



图 6.4.8 TCP Tahoe 中的快速重传机制

(FR/FR)间隙,那么在 Tahoe 中,流控窗口是冻结的,也就是意味着只有经过一个 RTT 时间之后,才有可能传输新的分组。这种机制造成的直接后果是,当重传分组到达接收端时,从源端到目的地的 TCP“管道”(pipe)是完全清空了的,当然,途经的路由器出现空闲也就完全有可能。为了防止这一现象的发生,当发送端收到一定数量的重复 ACK 之后进入“快速恢复”阶段,这个重复 ACK 的数量用 tcprexmtthresh 表示,通常设定为 3。与 Tahoe 将拥塞窗口减半、进入“慢启动”过程不同,Reno 用随后收到的重复 ACK 同步即将发出的分组。进入“快速恢复”阶段之后,源端可用窗口(usable window)的控制规则变成为 Min(awin,cwnd + ndup),其中 awin 表示收端的通报窗口大小,cwnd 是指发端拥塞窗口,进入“快速恢复”阶段时减半,ndup 在收到的重复 ACK 数小于 tcprexmtthresh 之前为 0、之后即为再收到的重复 ACK 个数。很显然,每收到一个重复 ACK,表明至少有一个分组已经离开网络。源端在接收到足够多的重复 ACK 之后,用接着到来的重复 ACK 触发新数据分组的发送。只有在接收到新分组的 ACK 之后,源端才退出“快速恢复”阶段,将 ndup 置为 0。

Reno 的“快速恢复”优化了单个分组从数据窗口丢失时的情况,比 Tahoe 的性能大有改进,但有多个分组从同一数据窗口丢失时,依旧存在性能问题。

New-Reno 主要是对 Reno 算法作了一些小改进,以消除有多个分组从同一数据窗口

丢失时对重传定时器的等待。改进考虑到发送端在“快速恢复”阶段收到的“恢复 ACK”是确认部分而不是全部出现在“快速恢复”阶段的分组。在 Reno 中,“恢复 ACK”使“可用窗口”降回到拥塞窗口大小,终结“快速恢复”阶段,而在 New-Reno 中,“恢复 ACK”并不结束“快速恢复”,相反,却假定那些在序列空间中紧跟在被确认分组之后的分组已经丢失,需要重发。因此,当在一个数据窗口中有多个分组丢失时,New-Reno 无须等待超时重发过程,一个 RTT 时间重传一个丢失分组,直到所有从该窗口中丢失的分组全部被重发。New-Reno 直到所有在“快速恢复”阶段开始时出现的分组都被确认,才会退出“快速恢复”。

### 6.4.8　TCP SACK

SACK(选择性重传)是有效恢复同一窗口中多个分组丢弃的另一种技术途径[6]。ACK 中的 SACK 域包含一定数量的 SACK 块,每一个 SACK 块都记录了信宿端接收或缓存的非连续分组。SACK 块的多少因应用和需要的不同而有所不同。与 Reno 相似,当发送端收到 tcprexmtthresh 个重复的 ACK 时,重发丢失分组,并将拥塞窗口减半,进入“快速恢复”过程。其间,SACK 维护了一个称为“pipe”的变量,用来估计出现在网络中的分组数。当 pipe 小于拥塞窗口大小时,发端发送新的或需重发的分组,并将变量 pipe 加 1。当发送端接收了一个带 SACK 选项的重复 ACK,表明新分组已被接收端接收,pipe 变量减 1。pipe 变量的使用将何时发送与发送哪一个分组有效解耦。发送端维护了一个特殊的数据结构(scoreboard),记忆前面 SACK 的确认信息。当发送端被许可发送分组时,依次发送丢失列表中记录的分组。如果没有这样的分组,而接收端的通报窗口又足够大,则发送端将发出新的数据分组。

当重传分组本身被丢弃后,SACK 用重传超时探测丢失,再次重传后进入“慢启动”过程。在确认了所有出现在进入“快速恢复”阶段的分组之后,发送端将从“快速恢复”阶段退出。

对于“部分 ACK”(partial ACK),即“恢复 ACK”,SACK 有特殊的处理。每收到一个“部分 ACK”,发端将变量 pipe 减 2,而非减 1。在启动“快速重传”时,当认定一个分组已经被丢失时,将变量 pipe 减 1;重传一个分组时,pipe 加 1。因此,对于第一个“部分 ACK”,将 pipe 减 2 似乎有些不合理,因为一个“部分 ACK”仅仅意味着一个分组离开了网络。对于后续的“部分 ACK”而言,重传分组进入网络时,pipe 增加了 1,但丢失在网络中的分组永远触发不了 pipe 的减小过程,因此,对于每一个持续到来的“部分 ACK”,实际上意味着有两个分组已经离开了网络:即最初认定丢弃的分组和重传分组。

### 6.4.9　TCP Vegas

Vegas[7] 对 Reno 进行了三项重要的技术改进。一是采用了新的重传触发机制,即用一个重复 ACK(而非 Reno 中的三个)来启动超时判定规程,这样可以更及时地检测到拥塞的发生。第二,在慢启动阶段,采用了更加谨慎的方式来增加窗口的大小,减少了不必要的分组丢失。第三个改进是“拥塞避免”阶段的窗口调整算法,Vegas 通过观测 TCP 连接中 RTT 时间的变化来调节拥塞窗口。如果发现 RTT 变大,则认为网络发生拥塞,相应

地减小 cwnd 如果 RTT 变小,则认为拥塞已经解除,并增加 cwnd 如果 RTT 保持不变,则不改变拥塞窗口的大小。具体的调节算法为:

$$\mathrm{cwnd}(t+\Delta t)=\begin{cases}\mathrm{cwnd}(t)+1, & \mathrm{diff}<\dfrac{\alpha}{\mathrm{base\_rtt}}\\ \mathrm{cwnd}(t), & \dfrac{\alpha}{\mathrm{base\_rtt}}\leqslant \mathrm{diff}\leqslant\dfrac{\beta}{\mathrm{base\_rtt}}\\ \mathrm{cwnd}(t)-1, & \dfrac{\beta}{\mathrm{base\_rtt}}<\mathrm{diff}\end{cases}.$$

其中,diff = cwnd(t) /base_rtt - cwnd(t) /rtt,rtt 是观测到的往返时间,base_rtt 是最小的往返时间,常数 $\alpha$ 和 $\beta$ 一般分别给定为 1 和 3。通过仿真实验分析 Vegas 的性能,已经得出结论:因为它没有采用分组的丢失来估计网络的可用带宽,而是通过改用 RTT 的变化进行判断,所以能够较好地预测网络带宽的使用情况,对小缓存的适应性较强,效率也较为理想。只是使用 Vegas 和未使用 Vegas 的 TCP 连接在竞争带宽方面存在严重的不公平现象。

## 6.5 端到端拥塞控制机制

TCP 流量控制遵守了两个主要原理:①实现流控自同步(self-clocking)的分组守恒定理;②加性增加倍乘减小(additive increase and multiplicative decrease, AIMD)[8] 的窗口管理算法。AIMD 依赖简洁的实现机制,在多个相互冲突的目标之间实现了较为理想的平衡与协调。虽然无法保证具有不同属性的终端系统(比如不同 RTT 时间、不同分组大小、经历不同跳数的拥塞链路等)能够分配到等量的带宽,但却能确保大致相似的用户得到基本相等的网络资源。以上两点是 TCP 拥塞控制最本质,也是最重要的特性。只是在探测空闲带宽和响应拥塞的过程中,TCP 流量控制算法造成了信源速率的大幅变化。虽然有些‘尽力而为型’的 Internet 业务能够很好地适应这种变化,但有些应用则显得极不适应,比如流媒体业务更希望拥塞控制机制对于拥塞的响应能够缓慢一些,从而有一个较为平滑的带宽占用以更好地匹配实时生成的业务码流。在组播业务中,TCP 流量控制更显得蹩脚,因为它需要组内所有的接收者都对分组的传输加以确认,由此产生了许多难以克服的技术问题。一方面,需要一致的 TCP 流量控制来达到公平的带宽分配;另一方面,为了适应业务的特点又期望放弃 TCP 流量控制机制,这种尴尬的两难局面促使研究者提出新的流量控制机制。于是,有了‘TCP 兼容性’(TCP-compatibility)这一概念,等价的说法是 TCP Friendliness[9]。这种新机制的基础是一个重要的研究结论:TCP 流占用的带宽正比于分组丢失率平方根的倒数[10]。如果一种拥塞控制机制使连接的吞吐量与分组的稳态丢失率之间的函数关系与 TCP 保持相似,那么就称这种拥塞控制机制是‘TCP 兼容的’。‘TCP 兼容性’概念的引入给拥塞控制的研究注入了活力,打破了拥塞控制由 TCP 流量控制垄断的格局,在理论上提供了为不同应用设计不同拥塞控制算法的可能。在短短两三年的时间里,已经产生了不少新的机制和算法,其中包括 TCP-Friendly 的速率控制(TFRC)[9] 和其他基于方程的拥塞控制[11]、二项式拥塞控制[12]、具有不同 AIMD 参数的

拥塞控制算法[13]以及接收端 TCP 仿真机制(TCP emulation at receiver, TEAR)[14]等。上述端到端拥塞控制机制在响应分组丢失时,不像 TCP 流量控制那样,将拥塞窗口(或传输速率)减半,而是采用较为缓慢的速率调节算法。

为了便于分析和研究,我们可以将已有的端到端拥塞控制机制与算法进行分类。基于稳态行为的特性,可以分为 TCP 等价机制、TCP 兼容机制和非 TCP 兼容机制三种。基于对拥塞的瞬态响应,可分为 TCP 等价机制、慢变响应机制和快变响应机制三种。

如果使用 AIMD 控制发送窗口或发送速率的大小,并且增加/减小常数与 TCP 保持一致,这样的拥塞控制算法称为 TCP 等价算法,典型的有各种 TCP 的变种算法和 Rejaie 等人提出的基于速率的 RAP(rate adaptation protocol)[15]等。因为失去了自同步机制,像 RAP 这样的 TCP 等价机制可能表现出与 TCP 完全不同的瞬态响应。

如果拥塞控制机制在大时间尺度(几个 RTT 时间)上的平均吞吐量与 TCP 连接在可用带宽不变条件下保持基本相等,就认为该机制是‘TCP 兼容’的。所有 TCP 等价的策略都是‘TCP 兼容’的,但反之则不成立。在探测和响应拥塞时,并非所有‘TCP 兼容’的算法都采用 TCP 的方式。在响应分组丢失或显式拥塞通告时,窗口(或速率)的减小如果小于 TCP,则称这类拥塞控制机制是慢响应的。典型算法有,TFRC、二项式拥塞控制和具有不同 AIMD 参数的拥塞控制算法等。慢响应算法可能是 TCP 兼容的,也可能不是,反之亦然。

一个基于 AIMD 的拥塞算法可用 a,b 两个参数来刻画。它们分别对应于窗口(或速率)调节中的增量常数和倍减参数[16]。当有分组丢失时,拥塞窗口从 W 减至(1 - b)W;如果没有分组丢失,在每一个 RTT 时间内,拥塞窗口从 W 增加为 W + a。对于标准的 TCP,a = 1,而 b = 0.5。如果一种 AIMD 策略是 TCP 兼容的,那么参数 a 和 b 之间将存在如下关系:

$$a = \frac{4(2b - b^2)}{3}$$

如果参数 $b < 0.5$,对应的拥塞控制算法将是慢响应的。

Bansal 和 Balakrishnan 提出了二项式拥塞控制算法[12],它是 AIMD 策略的非线性一般化推广,可以用 k,l,a 和 b 这 4 个参数来刻画此类算法。一旦探测到有拥塞发生,将窗口(或速率)从 W 减至 $W - bW^l$;没有拥塞发生时,每一个 RTT 时间内,拥塞窗口从 W 增加为 $W + a/W^k$。当且仅当 $k + l = 1$,同时 $l \leqslant 1$ 时,对于合适的 a,b 值,二项式拥塞控制算法是 TCP 兼容的。当 $l < 1$ 时,对于合适的 a,b 值,二项式拥塞控制算法具有慢响应特性。文献[12]详细研究了二项式拥塞控制算法的两个特例,IIAD($k = 1, l = 0$)和 SQRT($k = l = 0.5$),它们都是 TCP 兼容的,且具有慢响应特性。对于二项式拥塞控制算法,l 值越小,响应越缓慢。

Floyd 等人提出的 TFRC[9]与一般的拥塞控制机制有所不同,它不是对每一个分组丢失事件都产生响应,而是响应固定间隔时间上测得的分组丢失率。为了保持 TCP 兼容性,TFRC 将速率的控制律定义为丢失率和往返时间的函数。通常 TFRC(k)表示平均分组丢失率是在最近 k 个时间间隔上测量到的,k 的缺省值为 6。

TEAR 可以说是 TCP 的一个变种算法,但它是基于接收者的。接收者通过指数加权

滑动平均算法计算 TCP 拥塞窗口,并与估计得到的 RTT 时间相除得到具有 TCP 兼容性的发送速率。因此,TEAR 并没有改变 TCP 的拥塞窗口计算方法,只是在接收端对拥塞窗口进行了必要的修正,使其同时具备 TCP 兼容性和慢响应特性。

图 6.5.1 总结了各种端到端拥塞控制机制与算法之间的相互关系。

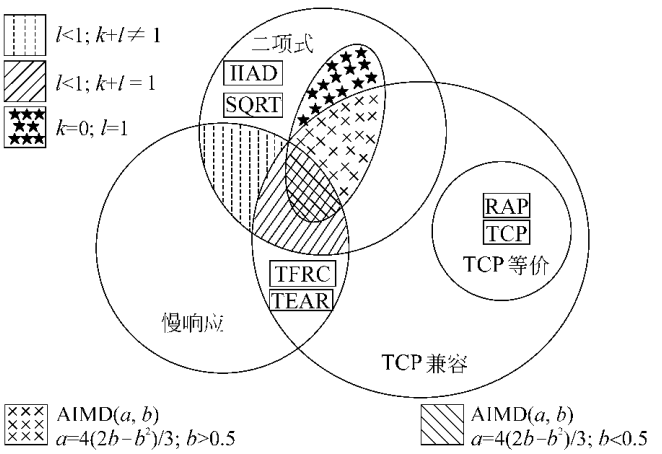



图 6.5.1 端到端拥塞控制算法的分类及其相互关系

## 6.6 中间节点上的增强机制

Internet 由许许多多异构的自治域通过松耦合方式组成,具有与生俱来的分散特性。对于其资源的管理和利用,考虑到业务的需要,应该从端到端原则出发。当然,本地决策的重要性也是不容忽视的。在 Internet 技术发展的初期,设计者遵从了一个技术理念:所有与流相关的状态都应该尽量在终端系统上实现与维护。这一指导原则的直接结果就是拥塞控制的绝大多数功能都是在主机端实现的,对网络中间节点所能发挥的作用考虑较少。毋庸置疑,这一原则提供的可扩展性在客观上有利于 Internet 的快速发展。但随着应用要求的日益丰富和技术的不断发展,研究者开始认识到,要想完全依赖实现在终端系统上的策略与算法是很难满足诸如 QoS 这样复杂的应用需求的。于是,开始将部分研究注意力转向网络中的路由器等中间节点设备,期望通过增强它们的功能来实现主机端无法达到的技术目标。就拥塞控制而言,网络中间节点有可能更及时,甚至提前准确了解网络的拥塞状态,并依此实施有效的资源管理策略,使网络能有效地避免拥塞或尽早从严重的拥塞状态中恢复过来。显式拥塞通告(explicit congestion notification,ECN)就是一个典型的应用,借助路由器的标记功能提高了有效吞吐量(goodput)[17]。当然,对路由器功能的扩展要受继承性和延续性的限制,否则将影响技术的实用性。事实上,现有的路由器扩展功能,主要包括调度和队列/缓存管理,并没有与 Internet 将流状态信息保存在主机端的早期设计理念相冲突。调度(scheduling)直接管理输出链路的带宽资源,而队列/缓存管理通过控制缓存与队列的占用间接影响带宽的分配。从拥塞控制的角度分析,加上传统的端到端流量控制,共有三种不同类型的机制可以协助解决网络拥塞问题。在分别阐

述每一种机制之前,我们首先讨论拥塞控制中的两个基本概念。

### 6.6.1　调度

调度规则决定数据分组接受服务的次序。最简单的调度算法是 FIFO 队列实现的 FCFS,因为它按照占用队列的多少分配带宽,无法实现公平性,于是有了 GPS(generalized processor sharing)[18]。数据分组位于不同的逻辑队列,在有限的时间间隔,每一个非空队列都有机会接受服务,至少一次。如果为队列赋予一定的权值,接受的服务与权值成比例,那么 GPS 便可以实现最大-最小比例公平性。GPS 仅仅是一种理论模型,最简洁的实现是轮循队列(round-robin),它用每次服务一个数据分组代替了 GPS 中的无穷小业务量。当为队列赋予权值后,相应的策略称为加权轮循队列(weighted round-robin, WRR),为了实现公平的带宽分配,WRR 需要了解各队列中分组的平均长度,这是不现实的,因为它与业务源的特性有关。GPS 的另一个近似实现是加权公平队列(weighted fair queue, WFQ)[19],它无须知道队列分组的平均大小,而是通过一种新机制仿真 GPS,即给分组一个与服务时间相联系的完成号(finish number),服务者依 finish number 的顺序服务分组。当分组较小时,在 WFQ 调度的网络中,端到端的时延存在确定的上界。最初的 WFQ 计算开销较大,随后又提出了一些优化和改进算法,比如 W2FQ(worst-cast WFQ)[20]、自同步公平队列(self-clocked fair queuing)[21]等。网络中间节点上的调度功能对于保证拥塞控制机制的公平性将起到不小的作用。因为目前有关拥塞控制的研究还没有过分地强调公平性,而是将注意力放在端到端的性能上,但随着应用需求的加强,为保证一定的公平性,调度在拥塞控制中的作用将会慢慢地显现出来,基于中间节点上的调度机制强化拥塞控制的功能也将会是一种有效的技术途径。

### 6.6.2　队列管理

队列管理通过选择何时丢弃何种业务流分组来控制队列长度。因为分组的丢弃通常被 TCP 的流量控制作为拥塞发生的信号,因此,队列管理和拥塞控制存在着密切的联系。路由器中最常用的队列管理策略是“尾丢弃”(tail drop),从拥塞控制的角度分析,它是一种拥塞恢复机制,已经在 Internet 上成功工作了许多年,但它有三个严重的缺陷:①持续的满队列状态;②业务流对缓存的死锁;③业务流的全局同步。“首丢弃”(drop from front)是另一种由队列溢出触发的管理策略,当有新的分组到达时,选择位于队头的分组丢弃。Lakshman 等人[22]指出该策略优化了 TCP 的性能,如改善了公平性、防止了死锁,因为“首丢弃”使得分组的丢失分布服从连接对缓存的占用。为了进一步解决全局同步问题,在分组丢弃策略中引入了随机概念,使得那些导致拥塞的主要连接在拥塞的恢复中承担了主要的责任。文献[23]指出:“随机丢弃”网关得到了增强的公平性,提高了大时延连接的吞吐量,随之而来的计算开销却是不可避免的。

“首丢弃”和“随机丢弃”对死锁和全局同步是有效的,但没有解决持续的满队列问题。如果路由器在拥塞发生前采取一些预防措施,那么满队列问题是可以得到解决的。主动的而非响应性的分组丢弃就是一种有效的手段,相应的队列管理策略被称为“主动队列管理”(active queue management, AQM),它是近来端到端拥塞控制研究中的一个热

点。在 AQM 中,通告拥塞的信号不再是惟一的分组丢弃,而可以采用标记分组来通知信源减速,这样会进一步提高连接的有效吞吐量。AQM 的主要技术目标是在减小排队时延的同时保证较高的吞吐量,具体分析,AQM 解决的问题主要包括以下 4 个方面:

(1) 早期探测路由器可能发生的拥塞,并通过随机丢弃或标记分组来通知源端采取措施避免可能发生的拥塞。

(2) 公平地处理包括突发性、持久性和间隙性的各种 TCP 业务流。

(3) 避免多个 TCP 连接由于队列溢出而造成同步进入"慢启动"状态。

(4) 维持较小的队列长度,在高吞吐量和低时延之间作出合理平衡。

虽然 Braden 等人在 IETF 提出 AQM 的研究动议是在 1998 年[24],但与其密切相关的 RED(random early detection)算法的研究却由来已久。早在 1993 年,Floyd 和 Jacobson 就提出了 RED[26],当时的主要目的是克服"早期随机丢弃"(early random drop,ERD)网关偏袒突发业务而造成的不公平问题。因为在提出 AQM 的研究时,RED 是惟一能实现其技术目标的算法,所以 RFC 2309[24] 将其推荐为 AQM 的惟一候选算法,随后,围绕着 AQM 和 RED 的研究逐渐丰富起来。

## 6.7 主动队列管理

### 6.7.1 AQM 与 RED

与 Tail Drop 相比,RED 为队列管理增添了两种新机制。其一,不是等队列全满后再丢弃到来的分组,而是利用概率判定机制事先丢掉部分分组来预防可能发生的拥塞;其二,通过平均队列而非即时队列调整分组丢弃概率,由此来尽可能地吸收部分短暂的突发流量。平均队列长度是用指数加权滑动平均(EWMA)来计算的:

$$avg \leftarrow (1 - w_q) \times avg + w_q \times q \qquad (6.7.1)$$

当分组到达队列时,如果平均队列长度小于最小门限值 $min_{th}$,分组安全进入队列;如果平均队列长度位于 $min_{th}$ 和 $max_{th}$(最大门限值)之间,如下计算分组丢弃概率 $P_b$:

$$P_b = max_p(avg - min_{th}) / (max_{th} - min_{th}) \qquad (6.7.2)$$

$$P_a = P_b / (1 - count \times P_b) \qquad (6.7.3)$$

其中,$max_p$ 是最大丢弃概率,avg 是加权平均队列长度,式(6.7.3)用于处理保证被丢弃分组间隔的均匀分布。

RED 的有效性经过了一些实践的验证,但依旧存在一些缺陷。比如,RED 算法的性能敏感于设计参数和网络状况,在特定的网络负载状况下依然会导致多个 TCP 的同步,造成队列振荡、吞吐量降低和时延抖动加剧[25]。RED 算法的公平性和稳定性[27,28] 也存在问题。自 RED 被首次提出来之后,它的参数配置就是一个没有彻底解决的问题。有关 RED 的最初文献中表述了配置参数对性能的影响,并给出了合理设置参数的建议。随后,许多关于参数优化的研究给出了当前广泛使用的参数配置方案[29],其中因为权值 $w_q$ 太小,不能及时探测拥塞,被从 0.001 增加到 0.002;又因为 2% 的最大分组丢失率不足以迫使多个 TCP 业务源有效地减小它们的窗口,0.02 的 $max_p$ 被 0.1 替代了。对于 RED 的

另一种改进是在丢弃函数中引入了一个新的参量 $2max_{th}$，如果平均队长大于 $max_{th}$ 而小于 $2max_{th}$，拥塞概率 $P_b$ 不再保持为 1，而是依据下式计算：

$$P_b = 2 \times max_p - 1 + (1 - max_p) \times avg/max_{th} \qquad (6.7.4)$$

具有这种丢弃函数的 RED 算法被称为“gentle_RED”[30]，图 6.7.1 描述了 RED 算法的演化过程，两次修正的主要目的都是为了解决 RED 算法存在的稳定性问题。

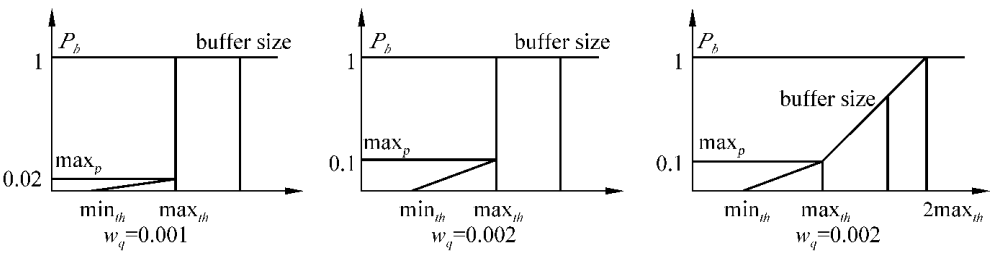



图 6.7.1　RED 算法的演化

## 6.7.2　RED 的变种算法

大多数有关 RED 算法的研究都将注意力集中到了改进和完善存在的不足与缺陷上，产生了不少 RED 的变种算法，较有影响力的有 WRED[31]，Stabilized-RED[32]，Self-configuration RED[33]，Adaptive RED[34]，FRED[35] 和 Balanced-RED[36]，其中 FRED 和 Balanced-RED 侧重解决 RED 存在的公平性问题，其余均意在增强 RED 的稳定性。

WRED 通过为不同的业务等级设定不同的最大丢弃概率来提供不同的服务质量。为了克服负载变化对 RED 稳定性的影响，Stabilized-RED（SRED）通过判定新到来的分组是否“命中”（hit），从称为 Zombies 的列表中任意选出的记录项来估计网络中激活的连接数，并用它修正分组丢弃概率，使 SRED 不再像 RED 一样对连接数敏感，能工作在较为广泛的网络环境中。SRED 采用瞬时队长，而非平均队长探测拥塞，分组丢弃概率的计算也与 RED 完全不同。

Self-configuration RED[30] 将 AQM 的工作区间用于 $min_{th}$ 和 $max_{th}$，划分为三部分。在轻载区间上（队列小于 $min_{th}$），用 $max_p/\alpha$ 修正最大分组丢弃概率；在重载区间上（队列大于 $max_{th}$），用 $max_p \times \beta$ 修正最大分组丢弃概率；当工作在 $min_{th}$ 和 $max_{th}$ 之间时，丢弃概率的调节规则与 RED 相同，一般分别将 $\alpha$ 和 $\beta$ 设定为 3 和 2，这样的自适应调整增强了 RED 的鲁棒性，但并非完全意义上的自适应策略，因为它仅仅依赖队列长度估计网络中激活的连接数，虽然有利于扩展性，但算法是粗糙的。

作为对 Self-configuring RED 的进一步改进，在调整丢弃概率时，Adaptive RED 用加性增加倍乘减小（AIMD）策略替代了倍乘增加倍乘减小（MIMD）策略，并提出了目标队长的概念。

FRED 是对 RED 所作的另一种改进，主要解决 RED 在多种类型流共存的网络中存在的不公平问题，防止了非响应性流对带宽资源的贪婪侵占。FRED 与 RED 的基本操作是相似的，只是为了区分不同类型的流而引入了一些新的变量，当然由此也会影响 FRED 的

扩展性。

Balanced RED 的目的也在于解决 RED 的公平性问题,主要处理响应性业务流和非响应性业务流之间的带宽均衡分配问题,它是一种 per-flow 的方案,路由器为每个业务流维护两个状态变量 $W_i$ 和 $qlen_i$,分别记录流 i 在缓存中的最大占用量和实际分组数,在概率函数的确定上使用了类似于 gentle-RED 的两阶段模式。

### 6.7.3 AQM 新策略

除了改进的 RED 算法,新的 AQM 策略也不断涌现,诸如 BLUE[37],PI 控制器[38],REM[39],GREEN[40],GKVQ[41],AVQ[42],SMVS[43] 和 FLC[44] 等。

BLUE 通过监测链路的空闲状态和分组丢失事件,以增量形式动态调整分组丢弃概率,稳定性虽有所增强,但在某些状态下队列依旧会发生振荡。

在文献[38]中,Hollot 等人对 AQM 的非线性模型[45] 进行局部线性化处理后将其等效为带时滞的二阶系统,在忽略时滞的前提下,用经典的频域分析方法,设计了用于主动队列管理的比例(P)控制器和比例-积分(PI)控制器,增强了系统的稳定性和适应能力,但依旧存在不少问题,比如瞬态性能差、过分依赖缓存空间的大小等。在后面的章节中,我们将具体分析,并提出相应的改进算法。

PI 控制器虽增强了系统的稳定性和适应能力,但却存在不少问题,比如瞬态性能差、过分依赖缓存空间的大小。为了克服这一问题,我们加入了微分环节,提出了应用于 AQM 的 PID 调节器,缩短了系统的调节时间[46]。REM 利用了 F. Kelly 提出的网络流量优化理论中“价格(price)”的概念[47] 来探测和控制网络的拥塞状态。“价格”的变化由两方面的因素决定,一方面是实际队长与期望队长之差,另一方面是聚合流的输入速率与端口输出速率之差。在“价格”的演化算法中,REM 使用了简单的比例关系。单节点上分组的丢弃概率与“价格”之间呈指数关系。对于端到端的业务流而言,其上分组的丢弃概率与链路上“价格”的总和呈近似的比例关系。因为 REM 算法是梯度寻优的解,虽然能实现 AQM 的技术目标,但目前的性能还不甚理想。

Gibbens 和 Kelly 提出了虚拟队列(virtual queue)的概念,并将之应用到主动队列的管理中,便有了 GKVQ 策略[41]。所谓虚拟队列是路由器维护的实际上并不存在的一种逻辑队列,与其连接的虚拟链路容量小于实际容量,而缓存大小与实际队列相同。当有分组进入实际队列时,更新虚拟队长以反映新分组的到达。如果虚拟队列溢出,则丢弃或标记实际队列中的分组。因为 GKVQ 采用了确定的虚拟容量 $\tilde{c}$,原理上,链路利用率总小于 $\tilde{c}/c$。为了克服这一缺陷,Kunniyur 和 Srikant 提出了自适应虚拟队列的 AQM 算法[42],用简单的微分方程在线调节虚拟容量,借助利用率因子和阻尼因子的调整,在高利用率和小队长之间实现恰当的平衡。从本质上讲,虚拟队列策略也是早期分组丢弃技术的一种,因为没有对队长的显式控制,很难在高吞吐量和低时延之间做出合理的平衡。

### 6.7.4 我们的研究思路与成果[50]

网络的研究大致包括体系结构、协议标准和机制算法这三方面的内容。体系结构的定义依赖于承载业务的特点和已有经验的积累,是网络研究需要首先解决的问题。在确

定了合理的体系结构之后，必须建立适当的协议和标准来保证整个网络按预想的目标运行。为了提高资源的利用率和增强业务的服务质量，采取必要的管理和控制机制是必不可少的。有了机制，当然需要相应的算法来加以实现。对于以上几部分的研究，大多采用依赖于知觉的启发设计加通过仿真实验验证的模式来进行。不可否认，这种模式为网络设计积累了不少经验，发掘了不少新的方法与策略。在体系结构、协议和机制的研究中无疑是一种最为理想的方式，但在有关算法的研究中，我们有理由怀疑这种设计模式的有效性，因为启发式的设计不免带有盲目性和片面性。RED 算法两次修正设计参数[29] 的事实充分说明了这一点。1993 年，在 Floyd 最初提出 RED 算法时，Internet 主干的链路带宽不过几百 Kbps，当时通过仿真确定的典型配置参数自然无法适应当前兆吉级的主干链路，出现队列振荡等不稳定现象也是可以理解的了，这一点已被相关的理论分析所证明[49]。也就是说，如果我们对所设计的算法在理论上没有一个准确的理解和把握，一旦应用的背景发生变化，性能将得不到保障。以 RED 算法为例，当前运行良好的配置参数在将来太比特（$10^{12}$）级的网络中再次发生不稳定现象将是必然的，到时再一次通过仿真实验来修正参数的设计方法肯定是不科学的。

总结 AQM 中各种策略与算法的研究，除 PI，REM 和 AVQ 以外，绝大多数算法的研究在很大程度上是依赖于直觉的、启发性的、针对局部个别问题的，没有全面、系统地运用理论工具对算法本身加以分析和研究，从而得到一个评价算法性能的有说服力的结论，为实际运用算法时的参数设置和算法本身的改进给出有价值的指导建议。借助适当的理论以保证算法的性能是非常必要的。PI，REM 和 AVQ 在这方面做了探索性的研究，只是它们各自所依据的理论是否切合 AQM 系统的实际情况还值得考虑。REM 依赖的全局流量优化理论是准静态的；PI 和 AVQ 分别将网络视为定常线性系统和时不变非线性系统，但实际网络的状态参数却随着新连接的建立和旧连接的拆除等诸多因素而瞬息万变，静态的、定常系统的处理方法的有效性值得怀疑，至少它不是最佳的。寻求恰当的理论来支持 AQM 策略的设计应该是一个有生命力的研究方向。

AQM 策略在高吞吐量和低时延之间作出合理平衡的关键在于始终将队列长度维持在一个较小的期望值，从控制系统的角度分析，这是典型的调节系统的技术目标。考虑到网络状态参数的时变性，鲁棒控制理论中控制器的设计与分析方法应该对设计性能良好的 AQM 策略具有很强的理论指导性。为此，我们借助滑模变结构控制器的设计方法，提出了综合性能优越的 SMVS 算法[43]。我们对于 PI 控制器的改进[46] 也是从分析控制系统性能的角度出发的。这些都从不同的侧面说明，恰当地运用控制理论中的分析和设计方法对网络拥塞控制和流量控制的研究是非常有帮助的，应该是一个值得进一步探索的方向。

分析已有的 AQM 算法，除 GKVQ 和 AVQ 之外，其余算法无一例外地沿用了 RED 的分组概率丢弃机制，它的有效性毋庸置疑。但是从本质上讲，判定分组丢弃的过程实质上是一个基于一定目标的决策过程，在智能决策技术的支持下，应该能发掘出新颖而高效的 AQM 机制，因为分组概率丢弃机制中随机数的生成毕竟需要不小的计算开销，不利于优化网络设备的性能。这也将是 AQM 研究需要解决的问题之一。

因为主机终端和网络中间节点在地理空间上的分散性，对路由器在拥塞控制中产生

的任何激励,源终端系统作出响应都有一定的滞后,已有的 AQM 策略在设计时几乎都忽略了这种响应的滞后效应,在小时延的 LAN 或 MAN 上可能有一定的合理性,但控制理论中的研究结论表明,系统的稳定性对纯延迟环节非常敏感,因此在大时滞的 WAN 环境下,延时必定会给算法的性能造成不良影响。我们初步的仿真结果也证明了这一点。如何克服或补偿时延对 AQM 算法性能的影响也将是一个需要关注的问题。

在算法的性能分析和评价方面,虽已有不少的研究工作,且已给出诸多有价值的结论[28,48,49],但我们认为依然有进一步深入分析的必要。以 RED 的稳定性分析为例,文献[28]强调:在一定的条件下是 RED 的非线性结构诱发了队列的振荡,而文献[48]却将 RED 算法等效为线性的放大环节,着重讨论连接数、链路容量和往返时间的变化给稳定性带来的影响。结合两方面的因素,在文献[49]中,我们应用非线性控制理论中描述函数的分析方法,给出了 RED 稳定性的新判据。

## 参考文献


1 Nagle J. Congestion control in IP/TCP internetworks. RFC 896, 1984

2 Jacobson V. Congestion avoidance and control. ACM Computer Communication Review, 1988,18(4):314~329

3 Floyd S, Fall K. Promoting the use of end-to-end congestion control in Internet. IEEE/ACM Trans Networking, 1999,7(4):458~472

4 Caserri C, Meo M. A new approach to model the stationary behavior of TCP connections. In: IEEE INFOCOM 2000. Tel Aviv, Israel, CA, 2000. 367~375

5 Stevens W. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, Jan, 1997

6 Mathis M, Mahdavi J, Floyd S, Romanow A. TCP selective acknowledgement options. RFC 2018, 1996

7 Brakmo L S, Peterson L L. TCP Vegas: End-to-end congestion avoidance on a global Internet. IEEE Journal on Selected Areas in Communications, 1995,13(8):1465~1480

8 Chiu D, Jian R. Analysis of the increase and decrease algorithm for congestion avoidance in computer networks. Computer Networks and ISDN System, 1989,17(1):1~14

9 Mahdavi J, Floyd S. TCP-friendly unicast rate-based flow control. http://www.psc.edu/networking/tcp_friendly.html,January 1997

10 Padhy J, Firoiu V, Towsley D, Kourose J. Modeling TCP throughput: A simple model and its empirical validation. In: Proc SIGCOMM Symposium on Communications Architecture and Protocol. Vancourver, Canada, August, 1998. 303~314

11 Floyd Sally, Handley Mark, Padhye Jitendra, Widmer Joerg. Equation-Based congestion control for unicast applications. In: Proc SIGCOMM Symposium on Communications Architecture and Protocol. August, 2000. 43~56

12 Bansal Deepak, Balakrishnan Hari. Binomial congestion control algorithms. In: IEEE INFOCOM 2001. Anchorage, AK, April 2001. 631~640

13 Yang Y, Lam S. General AIMD congestion control. Technical Report TR-2000-09, University of Texas at Austin, May 2000. available at http://www.cs.utexas.edu/users/lam/NRL/

14 Rhee I, Ozdemir M, Yi Y. TEAR: TCP emulation at receiver-flow control for multimedia streaming. Tech Rep, NCSU, April 2000. available at http://www.csc.ncsu.edu/

15 Rejaie R, Handley M, Estrin D. RAP: An end-to-end rates-based congestion control mechanism for realtime streams in the Internet. In: IEEE INFOCOM 1999. New York, 1999. 1337~1345

16 Floyd S, Handley M, Padhye J. A comparison of equation-based and AIMD congestion control. May 2000. http://www.aciri.org/tfrc/

17 Ramakrishnan K K, Floyd S. A proposal to add explicit congestion notification (ECN) to IPv6 and to TCP. Internet Draft draftkksjf-ecn-03.txt, October 1998. http://citeseer.nj.nec.com/ramakrishnan99proposal.html

18 Parker A K, Gallagher R G. A generalized processor sharing approach to flow control in integrated services networks: The single node case. IEEE Network, 1993, 1(3): 344~357

19 Demers A, Keshav S, Shenker S. Analysis and simulation of a fair queuing algorithm. In: Proceedings ACM SIGCOMM. Austin, TX, 1989. 1~12

20 Bennett J C R, Zhang H. W2FQ: Worst-case fair weighted queuing. In: IEEE INFOCOM'96. San Francisco, CA. March 1996. 120~128

21 Golestani S J. A self-clocked fair queuing scheme for broadband applications. In: IEEE INFOCOM'94. Toronto, Canada, 1994. 120~128

22 Lakshman T V, Neidhardt A, Ott T J. The drop from front strategy in TCP and in TCP over ATM. In: IEEE INFOCOM'96. San Francisco, CA, 1996. 1242~1250

23 Hashem E S. Analysis of random drop for gateway congestion control. Tech. Report MIT/LCS/TR-465, MIT Lab for Computer Science, 1989

24 Braden B, et al. Recommendations on queue management and congestion avoidance in the Internet. RFC 2309, 1998

25 Floyd S. A report on some recent developments in TCP congestion control. IEEE Communication Magazine, April 2001

26 Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. ACM/IEEE Transactions on Networking, 1993, 1(4): 397~413

27 Christiansen M, Jeffay K, Ott D, Smith F D, Tuning RED for Web traffic. In: Proc of the ACM SIGCOMM Conference 2000. Stockholm, Sweden, 2000. 139~150

28 Firoiu V, Borden M. A study of active queue management for congestion control. In: IEEE INFOCOM 2000. Tel Aviv, Israel, CA, 2000. 1435~1444

29 Jacobson V. Notes on using RED for queue management and congestion avoidance. ftp://ftp.ee.lbl.gov/talks/vj-nanog-red.ps.gz

30 Floyd S. Recommendation on using the "gentle_" variant of RED algorithm. http://www.icir.org/floyd/red/gentle.html

31 Cisco System. Distributed weighted random early detection. http://cco.cisco.com

32 Ott Teunis J, Lakshman T V, Wong Larry H. SRED: Stabilized RED. In: IEEE INFOCOM'99. New York, USA, 1999. 1346~1355

33 Feng W, Kandlur D, Saha D, Shin K. A self-configuring RED gateway. In: IEEE INFOCOM'99. New York, USA, 1999. 1320~1328

34 Floyd Sally, Gummadi Ramakrishna, Shenker Scott. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. http://www.cs.berkeley.edu/

35 Lin D, Morris R. Dynamics of random early detection. In: Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York, USA, 1997. 127 ~ 138

36 Anjum F, Tassiulas L. Balanced-RED: An algorithm to achieve fairness in Internet. In: IEEE INFOCOM'99. New York, USA, 1999

37 Feng W, Kandlur D, Saha D, Shin K. Blue: A new class of active queue management algorithm. In: IEEE INFOCOM 2001. Anchorage, Alaska, 2001. 1520 ~ 1529

38 Hollot C, Misra V, Towsley D, Gong W B. On designing improved controllers for AQM routers supporting TCP flows. In: IEEE INFOCOM 2001. 1726 ~ 1734

39 Athuraliya Sanjeewa, Low Steven H, Li Victor H, Yin Qinghe. REM: Active queue management. In: IEEE Network, May/June 2001. 48 ~ 53

40 Wydrowski Bartek, Zukerman Moshe. GREEN: An active queue management algorithm. In: Proceedings of ICC2002. New York, 2002. 2368 ~ 2372

41 Gibbens R J, Kelly F P. Distributed connection acceptance control for a connectionless network. In: Proc of the 16th International Teletraffic Congress. Edinburgh, Scotland, June 1999

42 Kunniyur S, Srikant R. Analysis and design of an adaptive virtual queue algorithm for active queue management. In: ACM SIGCOMM 2001. San Diego, CA, USA, 2001

43 Ren Fengyuan, Lin Chuang, Yin Xunhe, Shan Xiuming. A robust active queue management algorithm based on sliding mode variable structure control. In: IEEE INFOCOM 2002, San Francisco, CA, 2002. 13 ~ 20

44 Ren Fengyuan, Ren Yong, Shan Xiuming. Design of a fuzzy controller for active queue management. Journal of Computer Communications, 2002, 25(9): 874 ~ 883

45 Misra V, Gong Weibo, Don Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In: ACM SIGCOMM 2000. Stockholm, Sweden, 2000. 151 ~ 160

46 任丰原, 王福豹, 任勇, 山秀明. 主动队列管理中 PID 控制器的设计. 电子信息科学, 2003, 25(1): 94 ~ 99

47 Gibbens R J, Kelly F P, Resource pricing and the evolution of congestion control. 1998, http://www.statslab.cam.ac.uk/~frank/evol.html

48 May M, Bonald T, Bolot T. A control theoretic analysis of RED. In: IEEE INFOCOM 2000. Tel Aviv, Israel, CA, 2000

49 任丰原, 林闯, 王福豹. RED 算法的稳定性: 基于非线性控制理论的分析. 计算机学报, 2002, 25(12): 1302 ~ 1307

50 任丰原, 林闯, 刘卫东. IP 网络中的拥塞控制. 计算机学报, 2003, 26(9): 1025 ~ 1034

# 第7章

# 报文分类

Internet 网络应用的发展要求下一代路由器必须有能力支持 QoS、网络入侵检测、传输测量与记账、负载平衡、拥塞控制等一系列功能，因此要求采用不同的机制来实现这些功能。虽然实现这些功能的技术可能不尽相同，但它们都有一个公共的要求，即路由器应能够基于报文头的某些字段（也可能是报文的内容）对报文进行分类，因此报文分类（packet classification）是许多网络关键技术的基础，它涉及到网络的控制、性能、安全、管理等多方面内容，报文分类速度的快慢、功能的强弱都直接影响到这些网络技术的性能。

已有的研究表明，实现高速多维报文分类算法是非常困难的。在现有的算法中，一维、二维的分类算法比较多，而对高维分类支持的算法要么要求的内存空间过大无法满足低成本的要求，要么分类的速度较低无法满足高速网络环境的应用需求，它已成为路由器的新的瓶颈，随着网络的发展和 IPv6 的出现，这种情况会更加突出。一方面为满足未来高速网络的需求，要求分类速度越来越快，另一方面网络应用的发展要求路由器根据更多的报文头字段（甚至是报文的内容）以及更复杂、更灵活的分类组合对报文进行分类（这将导致分类速度更慢），这种矛盾使报文分类成为当今网络技术的一个研究热点，近年来吸引了许多研究人员的注意。

## 7.1 报文分类基础

### 7.1.1 报文分类概述

报文分类是指 Internet 路由器基于一个或多个报文头的字段（或报文的内容），把报文分类到相应的流/服务类的过程。所有的属于同一个流的报文应遵守事先规定的规则，并由路由器按照相同或相似的方式进行处理或标记。

事先规定的规则称为过滤规则，所有规则的集合称为分类器（或规则库）。每个分类规则关联一个行为（action），以便对符合该规则的报文做相应的处理或标记。简单地说，把报文映射到不同的服务类的过程就称为报文分类。例如，所有具有相同源 IP 地址、目的 IP 地址的报文被定义成一个流，事先定义好的源 IP 地址、目的 IP 地址对就是规则，相关的行为可以是保证一定的传输带宽或将报文丢弃等。

下面我们从一个 ISP 所用的实际例子来简单说明报文分类的应用[1]。图 7.1.1 说明

ISP1 连接三个不同的点：两个企业网络 E1 和 E2 以及网络访问点 NAP(network access point)，网络访问点又连接到 ISP3 和 ISP2。ISP1 向其用户提供一系列不同的服务，详见表 7.1.1。


图 7.1.1 一个 ISP(ISP1)的网络

表 7.1.1 由 ISP1 所提供的客户服务

| 服 务 | 例 子 |
|---|---|
| 报文过滤 | 拒绝所有从 ISP3 (在接口 X)到 E2 的传输 |
| 基于策略的路由 | 让所有从 E1(在接口 Y)到 E2(在接口 Z)的 IP 语音传输通过某个特定的 ATM 网络 |
| 记账 | 对所有的到 E1 的视频传输(通过接口 Y)赋予最高的优先权，并对这种类型的传输执行记账 |
| 传输速率限制 | 确保 ISP2 在接口 X 的输入不超过 10Mbps 的 E-mail 传输和不超过 50Mbps 的总量传输 |
| 传输整形 | 确保进入 ISP2(在接口 X)的 Web 传输不超过 30Mbps |

表 7.1.2 给出了在接口 X 流入到路由器的报文必须被分类的例子，同时也给出了需要分类的报文头字段。

表 7.1.2 在接口 X 流入到路由器的报文必须被分类的例子

| 流 | 需要考虑的相关报文字段 |
|---|---|
| 从 ISP2 来的 E-mail 报文 | 源链路层地址，源端口号 |
| 从 ISP2 来的普通报文 | 源链路层地址 |
| 从 ISP3 来的到 E2 去的报文 | 源链路层地址，目的网络层地址 |

一般报文分类处在通信子网的节点或路由器中，路由器分为识别流与非识别流路由器。识别流路由器跟踪流的传输并对在同一个流的报文进行相同或相似的处理；非识别流的路由器(也称报文-报文路由器)单独处理每个进入的报文。报文分类在网络流识别路由器中的位置如图 7.1.2 所示，分类的基本原理如图 7.1.3 所示。

每个报文在流识别路由器中的处理步骤为：

（1）进入路由器；
（2）路由表的查找（routing lookup）；
（3）报文分类（packet classification）；
（4）特殊处理（special processing）；
（5）交换调度（switching scheduling）；
（6）退出路由器。


图 7.1.2　流识别路由器的基本结构组成

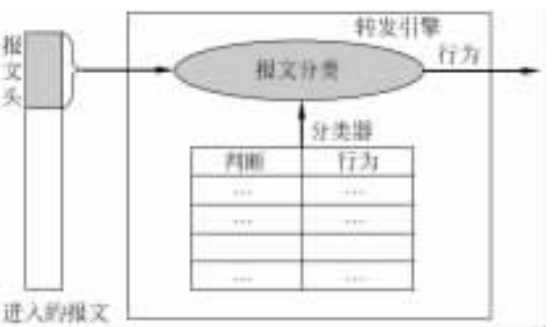



图 7.1.3　报文分类的基本原理图

## 7.1.2　相关符号术语的定义

为了便于后面的论述，我们先定义一些将要用到的符号术语。

- Field（字段）：Field 字段 $F_i$ 是一个连续的位的集合，它可以是报文的一部分，也可以是与报文相关的信息，例如，时间戳或者进入报文的端口号。
- Field list（字段列表）：Field 字段列表 $F = [F_0, F_1, \cdots, F_n]$ 是一个有顺序的字段集合，不同的顺序将产生不同的分类效果。
- Address（地址）：地址 D 是一个长度为 W 的位串。
- Prefix（前缀）：前缀 P 是一个长度为 0…W 的位串，用符号 length（P）表示一个前缀的位的个数。
- Packet head（报文头）：报文头 H 是 K 个字段的集合，这些头字段分别表示为 H[1]，H[2]，…，H[K]，这里每个字段都是一个位串。
- Filtering rule（过滤规则）：过滤规则 R 由 K 个前缀/范围组成，每个前缀/范围对

应报文头的相应字段,在更复杂的报文分类中可能要包含报文的内容,规则的字段用 R[i] 表示。一些规则用前缀表示(如,IP 源目的地址),一些规则用范围表示(如,FTP 的端口号为 20 ~ 21),还有一些规则用具体的值表示(如协议类型的值)。

- Classifier(分类器):分类器/规则库 C 是规则的集合,也称策略数据库、流分类器等。规则的总个数记为 N,这些规则的排列具有一定的顺序关系。当一个报文同时匹配多个规则时,许多算法默认排在前面的规则具有较高的优先权,即,报文匹配在所有匹配的规则中排在最前面的规则。
- Exact matching(精确匹配):精确匹配是对规则字段 i 的值的说明,一个报文头字段 H[i] 与一个规则字段 R[i] 是一个精确匹配是指,任给一个 i,当且仅当 H[i] = R[i]。
- Prefix matching(前缀匹配):前缀匹配是对规则字段 i 的前缀说明,一个报文头字段 H[i] 与一个规则字段 R[i] 是一个前缀匹配是指,任给一个 i,当且仅当 R[i] 是 H[i] 的一个匹配前缀。前缀匹配也可以表示成地址/掩码的形式,在这种形式中,掩码中位置 x 若为 0/1,表示在地址中的对应位为任意/有效位(必须相同),而且这种掩码是连续的,即掩码中所有位为 1 的都在所有位为 0 的左边。
- Range matching(范围匹配):范围匹配是对规则字段 i 的范围的说明,是指一个报文头字段 H[i] 落在规则字段 R[i] 的范围之内。
- Matching rule(匹配规则):一个规则 R 是一个报文头的匹配规则,当且仅当报文头的每个字段 H[i] 匹配对应的规则字段 R[i]。匹配的类型由 R[i] 说明,可以是精确匹配、前缀匹配或范围匹配。
- Cost function(代价函数):每一个规则联系一个代价函数。记为 Cost(R),以便当一个报文同时匹配多个规则时,通过寻找最低代价函数的规则作为最终匹配规则来解决规则的冲突问题。
- Action(行为):每一个规则联系一个行为,记为 A(R),当一个报文匹配一个规则时,就按照 A(R) 对报文进行处理。
- 规则更新:对分类器 C 进行规则的增加或删除等所做的更新操作。
- 规则重叠:指两个或多个规则所确定的范围、前缀有相互重合的部分。
- 规则冲突:规则冲突指两个或多个规则重叠,从而导致一个模糊的分类问题[2]。即,当有一个报文同时匹配多个过滤规则,并且这些规则所相联系的行为不一致时就产生规则的冲突。
- 最佳匹配:最佳匹配是指在分类器 C 中查找满足下列条件的规则 $R_{best}$。其中,$R_{best}$ 是与报文头 H 匹配的规则,并且在 C 中不存在其他的规则 R,R 与 H 匹配并且满足:$Cost(R) < Cost(R_{best})$,换句话说,$R_{best}$ 是在所有与 H 匹配的规则中,代价函数最低,或者说优先级最高的规则。
- 一维分类:假定一个有 N 个规则的分类器,每一个规则只有一个在 [1…u] 范围内的字段,且每个规则联系一个代价函数。一维分类是指在分类器 C 中查找一个包含点 $q \in [1 \cdots u]$ 且具有最小代价函数的规则的过程。最常见的一维分类是 IP

Lookup(IPL),分类规则是目的 IP 地址或者其前缀,它已被广泛用于路由表查找中。

- 多维分类:多维分类的含义与一维分类的含义相似,只不过每个规则中可以包含两个或多个字段。

可以用上述符号和术语对报文分类进行形式化定义。

定义 7.1.1　报文分类

假定一个分类器 C 含有 N 个规则 $R_j(1 \leqslant j \leqslant N)$,如果对报文头的 K 个字段进行分类,这里的规则 $R_j$ 由三部分组成:

(1) 关于报文头第 i 个字段的正则表达式 $R_j[i]$,$1 \leqslant i \leqslant K$,通常用具体的值、范围表达式或者前缀表达式表示;

(2) 一个整数 $pri(R_j)$,表示这个规则在分类器中的优先级,用来表示当一个报文同时匹配多个规则时来决定优先匹配哪个规则;

(3) 一个行为 $A(R_j)$,表示当这个规则被匹配后应对报文所作的操作。

对于一个进入路由器的报文 P,假设考虑报文头的 K 个字段$(P_1, P_2, \cdots, P_K)$,则 K 维报文分类是指在所有的匹配$(P_1, P_2, \cdots, P_K)$的规则 $R_j$ 中找到一个具有最高优先权的规则 $R_m$。即,$\forall j \neq m, 1 \leqslant j \leqslant N$,都有 $pri(R_m) > pri(R_j)$,且满足 $P_i$ 匹配 $R_j[i]$,$1 \leqslant i \leqslant K$,则称 $R_m$ 是报文 P 的最佳匹配规则。

## 7.1.3　报文分类的可用字段

用于报文分类的字段可能涉及报文头字段,也可能涉及报文的内容,下面从网络 OSI 参考模型的角度对可以用来进行报文分类的字段进行描述。

1. 第 2 层的报文分类可用字段

从 IEEE 802.3 MAC 层的格式(如图 7.1.4 所示)可以看出,可以用于分类的字段有源 MAC 地址、目的 MAC 地址(用以确定报文发送的源、目的主机),对于以太网还可以按类型来分类,用以确定上层传输所使用的协议(如图 7.1.5 所示)。


图 7.1.4　IEEE 802.3 MAC 层格式


图 7.1.5　以太网 MAC 层格式

为了使以太网帧和 IEEE 802.3 帧能够兼容,要处理好类型和长度的区别。其方法是,如果该字段的值大于帧的最大长度(1518),则表示为类型,否则表示为长度,即 802.3

标准。具体的做法是,以1536(十六进制0600)为界限,大于或等于该值认为是以太网,这时可以按类型处理,如IP为0800H,XNS为0600H,IXP为8137H等。

表7.1.3总结了第2层可以用来分类的字段。

表7.1.3 第2层可以用于分类的字段

| 层 | 长度 | 字段名称 |
|---|---|---|
| 2 | 8b | 目的MAC地址 |
| 2 | 48b | 源MAC地址 |
| 2 | 48b | 类型(以太网) |

2. 第3层的报文分类可用字段

从IPv4报文的格式[3](如图7.1.6所示)可以看出,在第3层可以分类的字段有源IP地址、目的IP地址(用以确定主机)、传输层协议类型(用以确定第4层所使用的协议,例如TCP,UDP)和服务类型。服务类型是一些指示服务质量的参数,这些参数用于在特定网络指示所需要的服务,所以也可以用作分类的字段。


图7.1.6 IP报文的格式

表7.1.4总结了第3层可以用来分类的字段。

表7.1.4 第3层可以用于分类的字段

| 层 | 长度 | 字段名称 |
|---|---|---|
| 3 | 8b | 服务类型 |
| 3 | 32b | 源IP地址 |
| 3 | 32b | 目的IP地址 |
| 3 | 8b | 传输层协议 |

3. 第4层的报文分类可用字段

从图7.1.7的TCP报文格式[3]可以看出,第4层可以分类的字段有源端口和目的端口,它包含了表示连接两端应用程序的TCP端口号。TCP报文段有多种应用,包括用于传输数据、携带确认信息、携带建立或关闭连接请求等。TCP使用6位长的标识字段来指示报文段的应用目的和内容,所以标识字段也可以用作分类的字段,每个标识字段的详细

含义见表 7.1.5。表 7.1.6 总结了第 4 层可以用来分类的字段。


图 7.1.7　TCP 报文头格式

表 7.1.5　TCP 报文头标识字段的含义

| 位(从左到右的标识) | 该位置的含义 | 位(从左到右的标识) | 该位置的含义 |
|---|---|---|---|
| URG | 紧急指针字段可用 | RST | 连接复位 |
| ACK | 确认字段可用 | SYN | 同步序号 |
| PSH | 请求急迫操作 | FIN | 发送方字节流结束 |

表 7.1.6　第 4 层可以用于分类的字段

| 层 | 长度 | 字段名称 |
|---|---|---|
| 4 | 16b | 源端口 |
| 4 | 16b | 目的端口 |
| 4 | 6b | 标识 |

4. IPv6 的报文分类可用字段

在同时支持 IPv4 和 IPv6 的机制中，IPv6 的版本号也可以作为分类的字段，用来区分 IPv4 和 IPv6。IPv6 的报文格式如图 7.1.8 所示[4]。优先级字段用来区分哪些分组能进行流量控制，哪些不能。值 0～7 表示在拥塞时可以慢下来的传输，值 8～15 是给恒速传送的实时通信量所用的，即使所有的分组都丢失也要保持恒速，所以优先级可以用作分类字段。流量标识用于使源端口和目的端口之间建立一条有特殊属性和需求的伪连接。不同的流量标识表示具有不同属性的伪连接，所以也可以作为报文分类的字段。源地址、目的地址与 IPv4 一样也可以作为报文分类的字段。

从上面的讨论可以知道哪些报文的头字段可以用作分类字段，除此之外，诸如入侵检测还需要检测报文的净负荷、部分 URL 等相关内容。

从上面的讨论可知，在报文分类中，可供分类的字段分布在从 OSI 参考模型的数据链路层一直到应用层的不同层中，但目前比较常见的分类组合有两种[5]：

(1) 源 IP 地址、目的 IP 地址的两维分类；

(2) 源 IP 地址、目的 IP 地址、传输层协议类型号、源端口号、目的端口号五维分类。


图 7.1.8 IPv6 的报文头格式

## 7.1.4 报文分类的几何解释

报文分类可以用几何的方式进行解释。我们知道,前缀可以用来表示线段上连续的间隔,那么两维的规则在二维欧几里得空间上就表示为 $2^{w_1} \times 2^{w_2}$ 的长方形,其中 $w_1$,$w_2$ 分别表示每个维的宽度。类似地,三维的规则在欧几里得空间上表示一个立方体。一般地,一个 d 维规则在 d 维空间上表示为一个 d 维超长方形。一个分类器就是一个具有优先级的超长方形集合。报文头表示一个点,这个点的值就是对应的报文头 d 个字段的值。

对一个报文的分类相当于寻找一个包含报文点且具有最高优先级的长方形,图 7.1.9 是表 7.1.7 分类器的几何解释,点 P(011,110) 由规则 5 来分类。

表 7.1.7 一个分类器的例子

| 规则 | 字段 1 | 字段 2 | 规则 | 字段 1 | 字段 2 |
|---|---|---|---|---|---|
| $R_1$ | 00 * | 00 * | $R_4$ | 00 * | 0 * |
| $R_2$ | 0 * | 01 * | $R_5$ | 0 * | 1 * |
| $R_3$ | 1 * | 0 * | | | |

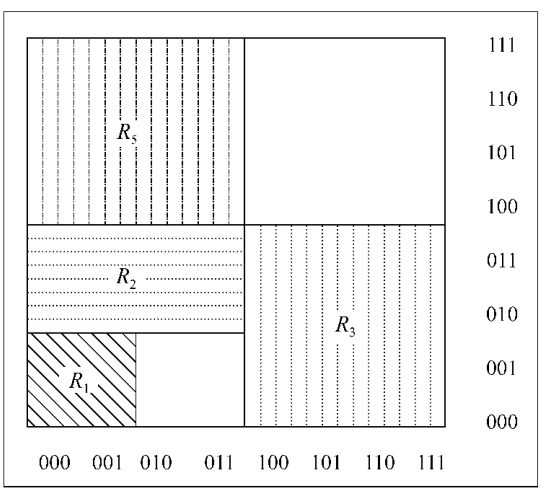



图 7.1.9 表 7.1.7 分类器的几何解释

注意:高优先级的规则可以覆盖低优先级的规则,例如,在图 7.1.9 中,规则 4 被规则 1 和规则 2 所覆盖。

## 7.1.5　报文分类规则的冲突问题

**定义 7.1.2（规则冲突）　规则冲突是指两个或多个规则重叠，从而导致一个模糊分类的问题[2]。即当有一个报文同时匹配多个分类规则，并且这些规则所联系的行为不一致时就产生规则冲突。**

**假定 166.111.＊代表清华大学的 IP 地址，162.105.＊代表北京大学的 IP 地址，又设规则仅涉及到报文头的两个字段（源 IP 地址，目的 IP 地址）。现考虑两个规则 $R_1$ =（166.111.＊，＊），它所相联系的行为 A（$R_1$）=｛1000bps 带宽｝；$R_2$ =（＊，162.105.＊），它所相联系的行为 A（$R_2$）=｛100bps 带宽｝。第一个规则表示分配所有的从清华大学来的报文千兆带宽，而第二个规则表示分配所有到北京大学的报文百兆带宽。现在如果路由器接收了一个源地址为清华大学、目的地址为北京大学的报文该怎么分配带宽呢？是分配给该报文千兆带宽还是百兆带宽呢？这时就产生了规则的冲突问题。矛盾的产生是由于一个报文同时匹配了两个规则（$R_1$ 和 $R_2$），并且这两个规则所联系的行为也不一致（一个是百兆带宽，另一个是千兆带宽）造成的。**

**最基本的报文分类规则的冲突解决方法是给每个规则都赋予一个优先级 pri（$R_j$），当一个报文同时匹配多个规则时，通过比较它们的优先级来确定最终报文应匹配哪个规则，即 $\forall j \neq m, 1 \leqslant j \leqslant N$，都有 $pri(R_m) > pri(R_j)$，且满足 $P_i$ 匹配 $R_j[i]$，$1 \leqslant i \leqslant K$。**

**通过对规则的优先级字段从高到低进行排序，可以将规则中的优先级 pri（$R_j$）字段省略，即当一个报文同时匹配多个分类规则时，取第一个匹配的分类规则就可以了，从而减少分类器的规模。**

## 7.1.6　报文分类举例

**1. 一维报文分类的例子**

**一个最简单的报文分类的例子是 IP Lookup（IPL），它执行最长地址匹配查找，每一个目的 IP 地址前缀就是一个规则 R，对应的下一跳就是规则相关联的行为 A（R），分类器 C 是转发表。假定在转发表中较长的前缀总出现在较短的前缀前面，即规则的优先级由规则在分类器中所处的位置先后来确定，则它是一个一维的报文分类。所以 IP Lookup（IPL）是报文分类的特例（见表 7.1.8），它是典型的一维报文分类问题，而报文分类则是 IPL 的一般化。**

**表 7.1.8　报文分类与 IP Lookup 的对照表**

| 报文分类 | IP Lookup | 报文分类 | IP Lookup |
|---|---|---|---|
| 分类器 | 路由表 | 规则表达方式 | 前缀 |
| 分类维数 | 一维 | 行为 | 下一跳地址，端口号 |
| 规则 | 路由表项 | 优先级 | 最长匹配规则 |

**2. 多维报文分类的例子**

**二维报文分类问题要在一维分类的基础上加上源 IP 地址，三维分类要加上传输层协**

议,五维分类要加上源端口和目的端口,有时为了惟一地确定一台计算机,要采用 IP 地址与网卡 MAC 地址绑定的办法,这就要用到源 MAC 地址和目的 MAC 地址,有时还要用到报文的内容,例如在入侵检测时就要检测报文的内容等。所以根据分类的不同应用目的,报文分类所涉及的分类字段是不同的。表 7.1.9 是一个多维分类器的通用表达形式。注意:这里,规则的优先级省略,默认情况是按规则的位置确定规则的优先级,即前面规则的优先级高于其后面规则的优先级。

表 7.1.9 多维分类器的通用表达形式

| 规则 | 字段 $F_1$ | 字段 $F_2$ | … | 字段 $F_K$ | 行为 A(R) |
|---|---|---|---|---|---|
| $R_1$ | 166.111.* | 202.192.* | … | UDP | $A_1$ |
| $R_2$ | 162.105.* | 166.111.* | … | TCP | $A_2$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $R_N$ | 202.114.* | 202.192.* | | * | $A_N$ |

表 7.1.10 是一个四维分类的具体例子,其中源 IP 地址、目的 IP 地址用前缀表示,目的端口号用范围表示,第 4 层协议用具体值表示,* 表示该字段值可以是任意值,不进行任何限制。

表 7.1.10 一个实际的四维分类器

| 规则 | 目的 IP 地址 | 源 IP 地址 | 目的端号 | 第 4 层协议 | 行为 |
|---|---|---|---|---|---|
| $R_1$ | 152.163.190.69/<br>255.255.255.255 | 152.163.80.11/<br>255.255.255.255 | * | * | 拒绝 |
| $R_2$ | 152.168.3/<br>255.255.255 | 152.163.200.157/<br>255.255.255.255 | 等于 www | UDP | 拒绝 |
| $R_3$ | 152.168.3/<br>255.255.255 | 152.163.200.157/<br>255.255.255.255 | 范围在 20 ~ 21 之间 | UDP | 容许 |
| $R_4$ | 152.168.3/<br>255.255.255 | 152.163.200.157/<br>255.255.255.255 | 等于 www | TCP | 拒绝 |
| $R_5$ | * | * | * | * | 拒绝 |

表 7.1.11 是表 7.1.10 的分类结果,报文 A 因为匹配规则 $R_1$ 而被路由器拒绝转发,报文 B 因为匹配规则 $R_2$ 也被路由器拒绝转发。

表 7.1.11 表 7.1.10 的分类结果

| 报文头 | 目的 IP 地址 | 源 IP 地址 | 目的端口 | 第 4 层协议 | 规则,行为 |
|---|---|---|---|---|---|
| 报文 A | 152.163.190.69 | 152.163.80.11 | www | Tcp | $R_1$,拒绝 |
| 报文 B | 152.168.3.21 | 152.163.200.157 | www | Udp | $R_2$,拒绝 |

## 7.2　报文分类算法

### 7.2.1　报文分类算法综述

下面讨论在分类算法中比较有影响的几个多维分类算法，同时对它们的性能和优缺点也进行了分析。

表 7.2.1 是一个有 6 个过滤规则的两维分类器，后面讨论的算法都将用它作为例子。两个字段（维）分别记为 $F_1$ 和 $F_2$，所有字段说明的长度最长为 3 位。按传统方法确定规则的优先级，即在分类器中，按规则的代价函数/优先级从上到下升序/降序排列，取所有匹配规则中排在最前面的第一个规则作为查找的最终结果。

表 7.2.1　一个有 6 个过滤规则的两维分类器

| 规则 | $F_1$ | $F_2$ | 规则 | $F_1$ | $F_2$ |
|---|---|---|---|---|---|
| $R_1$ | 00 * | 00 * | $R_4$ | 00 * | 0 * |
| $R_2$ | 0 * | 01 * | $R_5$ | 0 * | 1 * |
| $R_3$ | 1 * | 0 * | $R_6$ | * | 1 * |

#### 7.2.1.1　线性（linear）查找算法

它是一个按优先级降序排列（或按代价函数升序排列）的分类规则链表，一个报文顺序地与每个规则进行比较，直到找到第一个匹配的规则，因为规则已经事先按优先级降序排列好了，所以第一个匹配的规则就是最佳匹配规则。算法的优点是简单，而且存储器的利用率很高，但显然这种算法的扩展性很差。报文分类的时间随着规则数的增加呈线性增加。

#### 7.2.1.2　交叉组合（cross-producting）算法

交叉组合[9]是一个适用于任意维数的报文分类方法。报文分类的主要思想是，通过组合在每个维的单独的范围查找结果来对进入的报文进行分类。

预处理步骤用来建立数据结构，也包括计算范围集合 $G_k$（它的大小用 $s_k$ 表示，即 $s_k = |G_k|$），它是由每个维 k 的过滤规则映射得到的，其中 $1 \leqslant k \leqslant d$。设 $r_k^j$，$1 \leqslant j \leqslant s_k$，表示在 $G_k$ 中的第 j 个范围。这样，一个大小为 $\prod_{k=1}^{d} s_k$ 的交叉组合表 $C_T$ 就建立起来了。这个表中的每一项 $(r_1^{i_1}, r_2^{i_2}, \cdots, r_d^{i_d})$ $(1 \leqslant i_k \leqslant s_k)$ 的最佳匹配规则就被预先计算出来并存储起来。

对进入报文 $(v_1, v_2, \cdots, v_n)$ 的分类请求，首先在每个维 k 执行一个范围查找用来确定范围 $r_k^{i_k}$ 是否包含点 $v_k$。元组 $\langle r_1^{i_1}, r_2^{i_2}, \cdots, r_d^{i_d} \rangle$ 就直接在交叉组合表 $C_T$ 中被找到以便得到预先计算的最优匹配规则。

由交叉组合算法从表 7.2.1 过滤规则产生的范围见表 7.2.2，其几何解释如图 7.2.1 所示。例如，对于进入的报文 P（001，110），存取地址为 $(r_1^2, r_2^3)$ 的表项对应的规则为 $R_5$。

表 7.2.2 由交叉组合算法从表 7.2.1 过滤规则产生的范围

| | | | |
|---|---|---|---|
| $(r_1^1, r_2^1)$ | $R_1$ | $(r_1^2, r_2^3)$ | $R_5$ |
| $(r_1^1, r_2^2)$ | $R_2$ | $(r_1^3, r_2^1)$ | $R_3$ |
| $(r_1^1, r_2^3)$ | $R_5$ | $(r_1^3, r_2^2)$ | $R_3$ |
| $(r_1^2, r_2^1)$ | — | $(r_1^3, r_2^3)$ | $R_6$ |
| $(r_1^2, r_2^2)$ | $R_2$ | | |

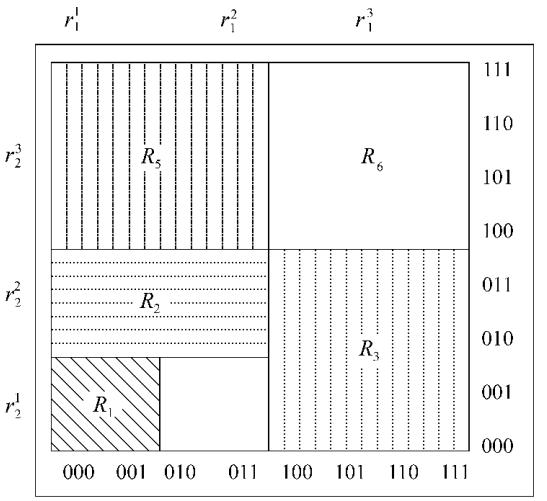



图 7.2.1 由交叉组合算法从表 7.2.1 过滤规则产生的范围的几何解释

我们看到,N 个前缀最多引起 2N 个范围,因此,$s_k \leqslant 2N$,$C_T$ 的大小为 $O(N^d)$。查找的时间复杂度为 $O(dt_{RL})$,其中 $t_{RL}$ 是在一个字段上执行一次范围查找的时间复杂性。因为交叉组合在最坏情况下的存储复杂性太大,所以此算法只适合于分类器非常小的报文分类。同时因为增加一个过滤规则将会改变映射的范围,并且需要重新计算交叉组合表,所以交叉组合是一个报文分类的静态解决方案。

### 7.2.1.3 Hierarchical tries 算法

一个 d 维的 Hierarchical tries[1] 是一维数据结构树的简单扩充,它是按照下面的方法进行递归构造的。

如果 d 大于 1,首先在规则集的第 1 个字段上构造一个一维的 $F_1$-trie 树。对于每一个在 $F_1$-trie 中的前缀 p,再在 $F_1$ 上递归构造(d - 1)维分层树 $T_p$,前缀 p 通过 next-trie 指针连接到树 $T_p$ 上。对于具有 N 个规则的分类器来说,存储空间的复杂度为 $O(Ndw)$,其中 w 为分类域宽。对于表 7.2.1 的分类器的数据结构如图 7.2.2 所示,分层树也称为多级树、逆向搜索树。

对于进入的报文$(v_1, v_2, \cdots, v_d)$进行分类的过程如下:查找算法首先基于 $v_1$ 遍历 $F_1$-trie,在遇到的每个 $F_1$-trie 结点上,算法沿着 next-trie 指针(如果存在)再遍历(d - 1)维树。因此搜索 d 维的时间复杂度为 $O(w^d)$。因为更新规则的每个组成部分可以被精确

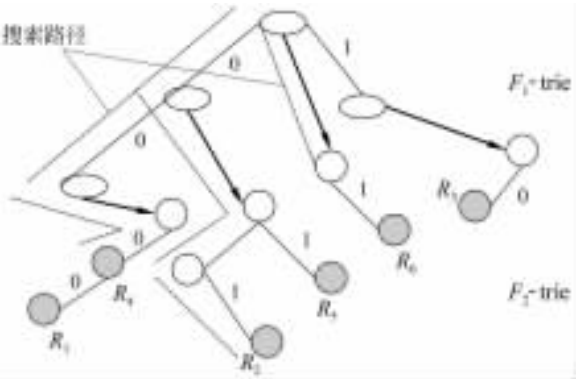



**图 7.2.2　一个 Hierarchical trie 的数据结构，灰色的点是 next-trie 指针，路径是算法关于报文 P(000,010) 对树的遍历**

地存储在一个最大深度为 O(dw) 的位置上，所以增量更新可以在 $O(d^2w)$ 时间内完成，方法与分类过程相似。其中 w 代表分类域宽，d 为分类的维数。

### 7.2.1.4　Bitmap-Intersection 算法

由 Lakshman 和 Stiliadis[10] 提出的 Bitmap-Intersection 算法是基于匹配报文头的规则集合 S 是 d 个集合 $S_i$ 的交集，这里，$S_i$ 是在第 i 个维上单独匹配报文的规则集合。Cross-Producting 是预先计算 S 并存储最高优先级的匹配规则，而此算法的机制是在每次分类操作的过程中计算 S 和最高优先级的匹配规则。

为了有效地利用硬件来计算集合的交集，每个集合被编码成一个 N 位的位映射，1 个位又对应 N 个规则。匹配的规则集合就是那些在位映射中对应位置为 1 的规则集合。一个对报文 P 的分类请求用类似于交叉组合算法进行相似的处理，首先对每个维单独地进行范围查找。每个范围查找返回一个对应维的编码成匹配规则的位映射。d 个集合相交得到匹配报文 P 的规则集，这只需要进行简单的硬件布尔与操作。从这个集合中就可以计算出最佳匹配规则。图 7.2.3 显示了表 7.2.1 分类器对应的位映射，同时也说明了 $R_5$ 是报文 P(001,110) 的最佳匹配规则。

由于每个位映射是 N 位宽度，在每个维上有 O(N) 个范围，所以总的存储空间的消耗是 $O(dN^2)$。分类的时间复杂度为 $O(dt_{RL} + dN/w)$，其中 $t_{RL}$ 是执行一次范围查找所需要的时间，w 是内存的宽度，这样，用 N/w 次内存存取操作得到一个位映射。时间复杂度可以通过用硬件并行单独查找每个维降下来，此算法不支持增量更新。同样的机制也可以用软件执行，但是分类的时间可能因为没有硬件特性（例如，并行和位映射相交）而变得较长。

文献[10] 显示这种机制每个维用 1MB SRAM 的 33MHz FPGA 设备可以支持到 512 条规则，每秒分类一百万个报文。在多维分类中，当规则个数很少时这个算法工作得很好。但随着规则的增加，存储空间平方倍地增加，内存带宽也要求线性增加（因此分类时间也线性增加），导致此算法不适合大规模的分类器。文献[10] 提出了一种在增加分类

时间的基础上降低存储空间的一种改进方法,但仍不能从根本上解决问题。

维 1

| | | |
|---|---|---|
| $r_1^1$ | $\{R_1, R_2, R_4, R_5, R_6\}$ | 110111 |
| $r_1^2$ | $\{R_2, R_5, R_6\}$ | 010011 |
| $r_1^3$ | $\{R_3, R_6\}$ | 001001 |

维 2

| | | |
|---|---|---|
| $r_2^1$ | $\{R_1, R_3, R_4\}$ | 110111 |
| $r_2^2$ | $\{R_2, R_3, R_4\}$ | 010011 |
| $r_2^3$ | $\{R_5, R_6\}$ | 000011 |

Query on P(011,110):

| | |
|---|---|
| 010011 | 维 1 |
| 000011 | 维 2 |
| 000011 | 相交的位映射 |
| $R_5$ | 最佳匹配规则 |

图 7.2.3 对应表 7.2.1 规则库的位映射表及报文 P(001,110)的分类过程

### 7.2.1.5 Tuple space search 算法

由 Srinivasan, Suri 和 Varghese[11] 提出的元组空间查找(tuple space search)算法的基本思路是把一个分类请求分解为一系列的精确匹配请求。算法首先将每个 d 维过滤规则映射成 d 元组,它的第 i 个组成部分用来存储第 i 维的前缀长度(这种机制只支持前缀说明)。因此,映射到同一个元组的过滤规则具有固定的长度,它们存储在一个 Hash 表中用来执行精确的匹配请求操作。一个分类请求是通过对所有可能元组的 Hash 表进行精确的匹配操作来完成的。表 7.2.1 分类器所对应的元组及其 Hash 表如图 7.2.4 所示。

| 规则 | 描 述 | 元组 |
|---|---|---|
| $R_1$ | (00 * ,00 * ) | (2,2) |
| $R_2$ | (0 ** ,01 * ) | (1,2) |
| $R_3$ | (1 ** ,0 ** ) | (1,1) |
| $R_4$ | (00 * ,0 ** ) | (2,1) |
| $R_5$ | (0 ** ,1 * * ) | (1,1) |
| $R_6$ | ( *** ,1 *** ) | (0,1) |

| 元 组 | Hash 表项 |
|---|---|
| (0,1) | $\{R_6\}$ |
| (1,1) | $\{R_3, R_5\}$ |
| (1,2) | $\{R_2\}$ |
| (2,1) | $\{R_4\}$ |
| (2,2) | $\{R_1\}$ |

图 7.2.4 表 7.2.1 分类器在元组空间查找算法中的元组和 Hash 表

对此算法的一个改进是运用启发式机制来避免查找所有的 Hash 表,基本思想与在“关于前缀长度的二进制查找算法”中所(详细内容参见文献[11])使用的机制相类似。

此算法中的分类时间等于 M 次 Hash 内存的存取时间,其中 M 是分类器的元组个数。因为每个过滤规则被精确地存储在一个 Hash 表中,所以此算法所用的存储空间为 O(N)。此算法支持增量更新,并只需要一次 Hash 内存存取时间存取与被修改元组相联系

的 Hash 表。总之，当元组的个数较少时，在平均情况下元组空间查找算法对多维分类来说工作得很好。然而，由于 Hash 的应用使得查找和更新的时间复杂性变得不确定，元组的个数可能很大，在最坏情况下达到 $O(w^d)$。而且，由于此机制只支持前缀，对于每个范围将被分成 $O(w)$ 个前缀（详细的范围转化为前缀的方法见 7.3.3 节），所以存储空间的复杂性为 $O(w^d)$。

#### 7.2.1.6　Modular 算法

正确地讲，Modular 算法[5]应当算作一类算法，它假定与每个过滤规则 $F_i$ 相联系的一个权值 $W_i$。在知道了 $W_i$ 的情况下，Modular 算法试图建立更有效的查找数据结构。Modular 算法将查找过程分为三个层次。

（1）跳转表：所有过滤规则依据规则位串中的某些位划分不同的组，该位串通常是过滤规则不同域的前缀，其选择也是根据统计特性做出的。

（2）查找树：（1）中同组的过滤规则组成一棵 $2^m$ 叉查找树。通过每次检查过滤规则中的 m 位，将其分为 $2^m$ 组。这 m 位是从当前规则位串中尚未检查的各位中选出的。其选择通常遵循两个原则，即减少重复拷贝（节省空间）和 $2^m$ 棵子树的平衡（节省平均查找时间和最坏情况下的查找时间）。建立查找树的过程实际上就是一个不断分组的过程，分组中止于叶子节点，即 filter bucket。

（3）filter bucket：当剩下的过滤规则数目小于某个给定的值时，则不再做进一步的划分，该节点即为叶子节点。由于叶子节点中的过滤规则数目比较少，因此可用比查找树更为简单的查找算法进行搜索。

整个报文头的全部域视为一个比特串，称为报文头位串。查找过程如下：首先以报文头位串中的特定位为索引查跳转表，得到查找树的入口地址，然后，沿着查找树，每次根据报文头位串中 m 位的值不断“下降”，直至到达叶子节点为止，然后在叶子节点中，查找最小代价的过滤规则。filter bucket 是最终存储过滤规则的地方，对 filter bucket 中过滤规则的查找，可以有不同的方法，例如，可以用硬件流水线来进行线性查找。

filter 的时间复杂性和空间复杂性比较复杂，在极坏情况下，时间和空间都可能达到无穷（不收敛）。

#### 7.2.1.7　RFC 算法

RFC（recursive flow classification）算法[12]是一种多维报文分类快速查找算法，其主要思想是将 IP 分类问题看成是一个将报文头中的 S 比特数据映射到 T 比特的 classID 的映射问题，这里 $T = \log N$，N 是过滤规则的总数。一个简单但不可行的报文分类方法是，预先计算出包头中的这 S 位共 $2^S$ 种不同情况中每种情况所对应的 classID 值，那么每个包只需一次查表即可，但这样会消耗极大的空间。RFC 的思想是通过多步或者说是多个阶段完成这个映射过程。每个阶段的结果将一个较大的集合映射成一个较小的集合，称为一次缩减。算法的描述见图 7.2.5。

图 7.2.6 是 5 个阶段的缩减树，对报文的查找过程如下：

（1）在第 1 个阶段（phase 0），报文头的 K 个域被分成若干个块（chunks），每一个块

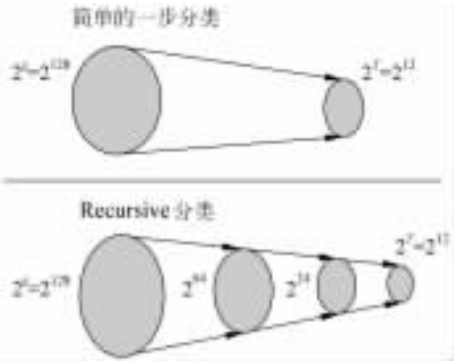



图 7.2.5 显示 RFC 基本的思想,缩减是在多个阶段执行的

被用来作为并行查找的索引;

(2)在后面的几个阶段,每次查找内存的索引值都是由前几个阶段的查找结果通过特定的方式合并而成的;

(3)在最后一个阶段,查找的结果得到一个确定的值,这个值就是该报文所对应的 classID。

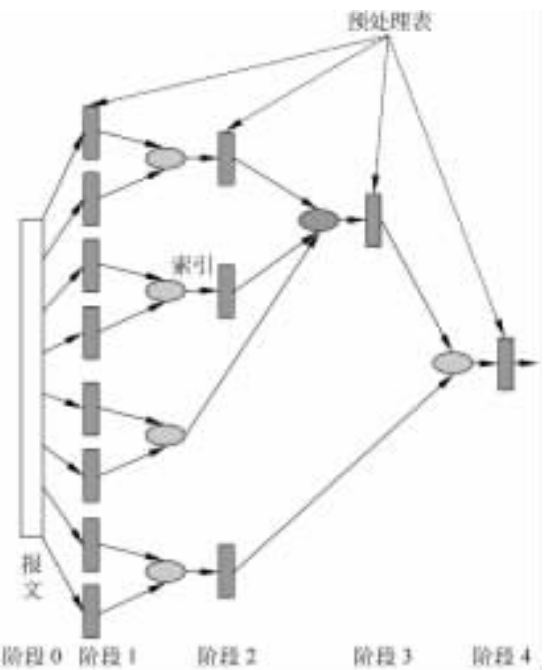



图 7.2.6 RFC 中的报文流

RFC 算法的性能受到两个参数的影响:

(1)选取阶段的数目 P;

(2)在给定 P 的情况下,“缩减”树的形状。也就是后面阶段的索引值从前面哪几个阶段的查找结果进行合并。在 P 和缩减树固定的情况下,随着过滤规则数 N 的增加,消

耗的内存量也相应增加。对于同样的过滤规则数据库，一般而言，P 越大，消耗的空间越少，但是查找时间也越长。

RFC 查找的时间复杂度是 O(d)，但是消耗内存非常大，有时候甚至是不切合实际的。

文献[12]指出，在四维的含有 1700 条规则的分类器中，RFC 用硬件实现可以达到 10Gbps 线速，用软件实现可以达到 2.5Gbps。但是对于大于 6000 条规则的分类器来说，存储空间和预处理时间增长得很快。在文献[12]中提出了一种优化措施，对于含有 15000 条规则的四维分类器来说，存储空间的要求可以低于 4MB。

### 7.2.2　报文分类算法的评价标准

衡量报文分类算法的好坏有许多技术指标，例如，时间复杂度、空间复杂度和更新复杂度等，下面分别进行论述。

1. 时间复杂度(time complexity)

该指标用于定义执行分类所需要的步骤或循环的数目的最大边界 O(f)，f 是规则个数 n、域宽 w 和维数 d 的函数。该指标用于确定找到匹配/不匹配规则所需要的顺序的步骤数。

2. 空间复杂度(space complexity)

该指标用于定义报文分类保存分类器及其相关数据结构所需要的空间的最大边界 O(f)，与时间复杂度相同，f 也是规则个数 n、域宽 w 和维数 d 的函数。分类应要求较少的存储空间，因为较少的存储空间要求意味着可以使用更快的内存，例如，静态随机存储器(SRAM)或者芯片存储器等。

3. 更新复杂度(update complexity)

该指标用于定义对分类器进行原子的插入、删除或更新所需要的步骤或循环数目的最大边界 O(f)。分类规则改变，数据结构也相应更新，不同的应用要求不同的更新速率，像防火墙这种由人工来增加规则，其增加规则的频率不是很高，一个很低的更新速率就可以满足了，但是像对每个流都有队列的路由器来说可能要求有很高的更新速率。一般有三种可能的更新：

(1) 完全更新，是指对全部数据结构进行重建；

(2) 增量更新，是指在运行过程中向分类器中插入或删除一条过滤规则；

(3) 结构重组更新，是指随着过滤规则的不断插入和删除，可能会造成查找数据结构效率变低，需要对数据结构进行适当调整，以提高查找效率。

4. 扩展性(extension)

可扩展性包括以下三个方面的内容：

(1) 分类器规则个数的可扩展性，是指分类器的规则个数 n 可以很大，而不是限定在

固定的数目上；

(2) 分类维数的可扩展性,是指分类可以包含任意多个分类字段,而不是限定在固定的某几个字段上,例如常见的有源 IP 地址、目的 IP 地址的两维分类和源 IP 地址、目的 IP 地址、协议类型、源和目的 TCP/UDP 端口号的五维分类等；

(3) 分类层次上的可扩展性,是指分类可包含 OSI 模型的多层内容,理想的算法应该是能够容许匹配任意层内的字段,包括数据链路层、网络层、传输层,在一些特殊情况下可能要包括应用层的内容。

5. 最坏情况下的性能分析

一个分类算法的好坏,除了给出平均的性能分析以外,还应该给出在最怀情况下的性能分析。平均的性能分析有时不能完全真实地反映分类的性能。最近的研究表明,75% 的报文要比典型的 552 字节的 TCP 报文要小,而且将近一半的报文是 40 ~ 44 字节的长度,主要由 TCP 确认报文和 TCP 控制报文组成[21]。由于分类算法在执行过程中不能用缓冲来处理报文的变化,即使报文的大小都是 44 字节,分类算法也必须以线速处理报文[22]。这就意味着算法应能够提供最坏情况下报文最小的执行时间。

6. 规则表达的灵活性

规则表达的灵活性是指一个分类算法应该支持通用规则说明,除简单的前缀说明、具体的值说明以外,还应包括范围说明(如小于、大于、等于)和通配符等。

7. 预处理时间

一般的分类算法都有一个预处理时间,通常要求预处理时间也比较低,但一些算法往往通过增加预处理时间来提高分类的速度,所以此项要求达到合理的程度就可以了,视不同的情况而定。

8. 算法执行的灵活性

算法执行的灵活性是指算法在软硬件上均可实现。因为分类操作必须是高速的,所以算法除了可以用软件实现以外,也应易于用硬件实现,一个报文分类算法的执行不应该只限定为软件执行。

对于报文分类算法而言,一般期望时间复杂度、空间复杂度和更新复杂度越小越好,并具有良好的可扩展性,不仅平均性能良好,而且在最坏情况下的性能也良好。但在许多情况下,报文分类算法无法满足全部指标达到最优,而是要根据算法的使用场合加以折中。

### 7.2.3 报文分类算法的性能比较

目前,报文分类算法从实现的角度可以分为两种,一种是硬件算法(hardware-based algorithms, HW),另一种是软件算法(software-based algorithms, SW)。

硬件算法,如 Ternary-CAMs (content-addressable memory) 算法[13~15],将所有规则用

TCAM 硬件实现，利用硬件并行对报文进行分类，并报告最高优先级匹配规则。这种方法分类速度较快，但它要求大量的逻辑元素，并受到功率消耗的限制，不能扩展到大规模的分类器上，因此这种方法有维数少、扩展性差、价格比较昂贵等缺点。同时，一些运算符不能直接支持，内存数组的利用率也不是很有效，对基于 PC 机的路由器来说不实用[16]。

现有的软件算法一般只具备特定维数的分类能力，特别是单维分类和二维分类的算法较多（例如 Binary Rang Search 算法、Grid-of-tree 算法[9]、AQT 算法[17] 等）。能够提供对高维分类支持的算法在空间复杂度和时间复杂度上都不能很好地满足应用要求，例如 RFC 算法[12]、Hierarchical Cuttings 算法[18] 等在最坏情况下空间复杂度很大。另一些算法，例如 Bit Vector Search 算法、Back Tracking 算法时间复杂度较大。而基于 Cache 的报文分类方法[19] 因命中率较低、流持续时间较小等原因在实际使用中也不能很好地满足要求。

传统的报文分类算法主要分为以下 4 类[1]。

1. 基本数据结构算法（basic data structure algorithms，BS）

（1）Table 算法，它构造全部可能的情况表，用可能情况作为随机寻址的下标，属于单维分类，时间复杂度为 1，空间复杂度为 $2^w$。

（2）Linear search 算法，用最简单的数据结构，将所有规则按优先级降序排列为一个链表，一个分组与每一个规则一一比较，直到找到匹配所定义的相关字段的规则为止。时间复杂度为 n，空间复杂度为 n。优点：算法简单，能充分利用存储器 缺点：扩展性差。

（3）Hierarchical tries 算法[1]，将每维分类均排列为按比特的搜索树，并将树相互连接，时间复杂度为 $w^d$，空间复杂度为 ndw，此算法维数一般不超过 2，规则数少。

（4）Set-pruning tries 算法[9]，是 Hierarchical tries 算法的改进。通过复制部分树，可以不回溯搜索，时间复杂度为 dw，空间复杂度为 $n^d$，此算法维数一般不超过 2，规则数可较多。

2. 几何算法（geometric algorithms，GA）

（1）Grid-of-tree 算法[9]，是 Set-pruning tries 算法的改进。将复制功能通过指针实现，节约空间，时间复杂度为 $w^{d-1}$，空间复杂度为 ndw，属于二维分类，更新复杂度高。

（2）AQT 算法[17]，二维树查找，时间复杂度为 w，空间复杂度为 nw，属于二维分类。

（3）FIS-tree 算法[20]，利用规则之间的覆盖关系构造 Fat Inverted Segment Tree，时间复杂度为 w(L+1)，空间复杂度为 $Ln^{(L+1)/L}$。

3. 启发式算法（heuristics algorithm，HA）

（1）RFC 算法[12]，分布并行运算，逐步求精，时间复杂度为 d，空间复杂度为 $n^d$，此算法需要并行计算能力，速度快，存储和分类维数相关。

（2）Hierarchical cuttings 算法[18]，通过几何搜索获得一个规则的子集，其中规则数目少于一个常数，继续搜索获得最终规则，时间复杂度为 d，空间复杂度为 $n^d$，此算法空间需要量大，速度快，可多维分类，更新复杂度低。

(3) Tuple-space search 算法[11],预计算规则前缀元组,比较时进行 Hash 匹配操作。时间复杂度为 n,空间复杂度为 n,虽然最坏时间复杂度为 O(n),但平均分类速度较快,节省空间。

4. 基于硬件的算法(hardware-based algorithm, HW)

(1) Ternary CAM 算法[13],将所有的规则用 TCAM 硬件实现,时间复杂度为 1,空间复杂度为 n,速率要求较高,容许成本高,维数较少。

(2) Bitmap-intersection 算法[10],分类利用投影关系计算位表,利用位表合成运算结果,时间复杂度为 $dw+\frac{n}{memwidth}$,空间复杂度为 $dn^2$,需要硬件实现,存储空间大。

## 7.3 报文分类器的设计

### 7.3.1 报文分类器的特性

要开发出高效可行的分类算法,必须考虑下面两个方面的因素:

(1) 充分利用硬件开发平台的特性来提高报文分类的速度。例如,因为内存访问是分类速度变慢的主要原因,所以要了解硬件平台的内存访问的速度,内存数据总线的带宽以及硬件的并行性、流水线特性等。

(2) 充分利用分类规则的特性并根据分类要求开发出高速的符合实时要求的分类算法,如分类规则的分布规律,分类规则的个数范围,分类使用的不同目的和场合,分类速度的要求,规则更新速度的要求等。

本节主要讨论分类器的特性,下面的结论是通过分析来自 101 个不同的 ISP 的 793 个分类器中的规则得出的[12]。在设计分类算法时,应该充分利用这些特性来加快分类算法的速度。

1. 分类器不包含大量的规则

只有 0.7% 的分类器包含的规则数大于 1000 条,一般平均规则个数为 50 条。分类器的规则个数相对较少也是有一定道理的,现今的大多数网络,规则的配置是由网络管理员手动配置的,要确保配置的正确性是一件不容易的事情。

2. 传输层协议字段限制在较小的范围内

虽然传输层协议字段是 8 位,共有 128 种可能性,但在所检查的报文分类器中,仅包含 TCP,UDP,ICMP,IGMP,(E)IGRP,GRE,IPNIP 和通配符 *(* 表示除前面 7 种协议类型以外的其他协议类型)8 种可能性,这样分类时就可以只用 3 位而不需用 8 位来表示协议的类型。

3. 传输层字段有各种说明

许多(10.2%)是范围说明(例如大于 1023 或者范围在 20 ~ 24 之间)。特别地,大于

1023 的规则就占 9%。这样,这个共同的范围描述如果要表示成前缀描述,则要求表示为 6 个独立的前缀(1024 ~ 2047,2048 ~ 4095,4096 ~ 8191,8192 ~ 16383,16384 ~ 32767,32768 ~ 65535),这将导致分类器规则数的增加,所以范围表示改为前缀表示会导致分类器规则数的增加,但前缀表示改为范围表示不会导致分类器规则数的增加。

4. 在分类器中,8% 的规则是冗余的

如果在分类器中出现冗余规则,可以从分类器中删除掉,从而减少分类器的规模。后面将会更详细介绍如何判断一个规则是冗余规则的方法。

5. 在同一分类器中,许多不同规则共享一系列的字段说明

这个现象的出现是因为网络操作员经常想对子网或者主机的通信组指定相同策略。

6. 接近一半报文的字节数是 40 ~ 44 字节

典型的 TCP 报文的大小是 552 字节,但接近一半报文的字节数是 40 ~ 44 字节,所以评价报文的分类速度应按每秒处理多少个报文来衡量,而不能以每秒处理多少字节数来衡量[10]。

## 7.3.2　报文分类器的设计原则

1. 线速转发原则

报文分类必须以线速转发报文,这要求路由器能够及时转发从线路进入的最小分组(64 字节),这对基于报文分类的各种应用来说是非常重要的,否则,路由器就有可能在路由器还未知道报文的重要性之前就将一些重要的报文丢掉了[23]。

2. 折中原则

评价分类算法的标准有很多方面,而这些方面又往往是相互矛盾、相互冲突的。设计算法就是要根据分类的实际要求选取所需要的速度、内存、规则更新等各方面因素的折中,如分类的速度和所需的内存就是一对矛盾,在不可能两者都兼顾的情况下,必须根据实际情况找到一个折中点[24]。

3. 满足实时需要原则

不同的应用所需要的分类、规则更新速度不一样(如边缘路由器要求规则更新速度较慢,而核心路由器要求规则更新速度较快),所以要根据实际需要设计符合实时需要的分类算法。满足所有要求的算法是很难找到的。

4. 遵循算法的简单性原则

过分复杂的分类算法不仅会导致算法的实现复杂,而且算法本身的运行也要消耗一定的资源和时间,在理论上提出了大量的‘ 高效 ”分类算法,但在实际中却用得很少或者

根本不用,其原因与分类算法本身是否简单可行有很大关系。所以提倡研究在满足实时需要的前提下的简单分类算法。

## 7.3.3 报文分类算法的基本设计思路

### 7.3.3.1 范围查找

分类算法的本质是多维查找,路由表查找是分类的特殊情况(一维分类)。由于路由表查找方法已经比较成熟,并且已有一些快速的路由表查找算法研究出来[25~29],因此,分类算法能否利用这些已经成熟的查找算法来解决分类问题是一个基本的解决报文分类的思路。如果在某个维上字段的说明都是前缀说明,则路由表查找技术就很有可能用上。或者查找所有的前缀匹配,或者查找最长前缀匹配,这可以利用任何以前的路由表查找算法。然而,分类字段的说明不仅仅只是前缀说明,它可以是任意的范围说明(如 FTP 协议使用的端口号为 20~21)。因此,定义一个长度为 w 的一维的范围查找问题是很有用的,它可以将范围说明等价为前缀说明,因此是解决报文分类的一个基本思路。

**定义 7.3.1(范围查找)** 给定一个包含 N 个不相交的范围集合 $G=\{G_i=[l_i,u_i]\}$,组成对线段$[0,2^w-1]$的一个分割,即 $l_i$ 和 $u_i$ 满足 $l_1=0$,$l_i\leqslant u_i$,$l_{i+1}=u_i+1$,$u_N=2^w-1$,范围查找问题是查找一个包含进入的点 P 的范围 $G_p$。

范围查找问题可以通过首先把每个范围转换成最大前缀集合,然后在这些前缀组合的基础上解决前缀的匹配问题来解决。范围转换成前缀利用这样观察到的事实,设长度为 s 的前缀对应的范围为[l,u],l 是(w-s)个最低有效位全为 0 的二进制数,u 是(w-s)个最低有效位全为 1 的二进制数。因此,如果把一个给定的范围分成满足这个特性的最小个数的子范围,将得到一个与原范围等价的最大前缀集合。表 7.3.1 是一些将 4 位字段的范围说明转换为前缀说明的例子。

表 7.3.1 一些 4 位字段的范围表示转换为前缀表示

| 范 围 | 连续的最大前缀 |
|---|---|
| [4,7] | 01** |
| [3,8] | 0011, 01**, 1000 |
| [1,14] | 0001, 001*, 01**, 10**, 110*, 1110 |

可以看到,一个关于 w 位的范围可以被分成最多 2w-2 个最大前缀,虽然范围查找问题可以被转换成前缀匹配问题,但内存的存储要求增长了 2w 倍。Feldmann 和 Muthukrishnan[20] 提出了一种可以减少存储空间的方法,增长的倍数只是常量 2,然而,这种存储空间减少技术不能被用在所有的多维分类机制上。所以,直接通过把所有字段中的范围说明转换为前缀说明,然后利用路由表查找技术进行报文分类是行不通的,必须与其他技术结合(如分类规则的特性等)才有可能设计出较为有效的报文分类算法。

### 7.3.3.2 计算几何的上下界

在计算几何领域内有几个标准的问题[30~32],例如点的定位问题、长方形的包围问题

等非常类似报文的分类问题，所以利用计算几何理论解决报文分类问题也是一种思路。

我们知道，前缀可以用来表示线段上连续的间隔，那么二维的规则在二维欧几里得空间上就表示为 $2^{w_1} \times 2^{w_2}$ 的长方形，其中 $w_1$，$w_2$ 分别表示每维的宽度。类似地，三维的规则表示在三维欧几里得空间上的立方体。一般地，一个 d 维的规则表示在 d 维空间上的 d 维超长方形。一个规则库/分类器就是一个具有优先级的超长方形集合。报文头表示一个点，这个点的值就是报文头 d 个字段的值，例如图 7.3.1 是表 7.3.2 分类器的几何解释。

表 7.3.2　一个分类器的例子

| 规则 | $F_1$ | $F_2$ | 规则 | $F_1$ | $F_2$ |
|---|---|---|---|---|---|
| $R_1$ | 00 * | 00 * | $R_4$ | 00 * | 0 * |
| $R_2$ | 0 * | 01 * | $R_5$ | 0 * | 1 * |
| $R_3$ | 1 * | 0 * | | | |

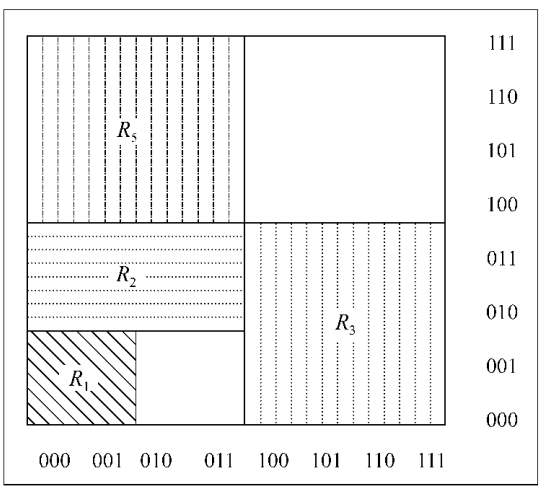



图 7.3.1　表 7.3.2 所示分类器的几何解释

在这种几何解释的前提下，对一个报文的分类就相当于在所有的长方形中，寻找一个包含代表报文分类的点且具有最高优先级的长方形。如果较高优先级的长方形总是画在较低优先级的前面，报文分类就等价于找一个包含给定点的最前面的可见的长方形。例如图 7.3.1 中的点(011，110)将由规则 5 来分类，注意，规则 4 被规则 1 和规则 2 所覆盖。

多维空间点的定位问题要求在一组互不重叠的范围内查找一个包围给定点的范围。由于报文分类的超长方形可以相互重叠，所以解决报文分类问题的难度要大于计算几何中点的定位问题。在最坏情况下，具有 d 维($d > 3$)的 N 个长方形区域的最好点的定位的边界是：时间复杂度为 $O(\log N)$，空间复杂度为 $O(N^d)$；或者时间复杂度为 $O((\log N)^{d-1})$，空间复杂度为 $O(N)$[31,33]。显然，对于高速的路由器的报文分类来说是不可行的，对于仅有 100 条过滤规则的 4 维分类，$N^d$ 所要求的空间为 100MB；$(\log N)^{d-1}$ 是大约 350 次内存存取次数。所以，纯粹用计算几何的理论解决报文分类问题也是行不

通的,必须与其他技术结合(如分类规则的特性等)才有可能设计出较为有效的报文分类算法。

### 7.3.3.3 规则个数的压缩

规则个数的增加是分类算法速度减慢的原因之一。大多数分类算法的速度都与规则个数 N 有关,如果能够减少规则个数,则可以加快分类的速度,同时也能节省内存空间。研究表明,在分类器中,8% 的规则是冗余的,这是由于网络操作员日积月累地增加新规则但又没有删除可能是冗余的规则造成的。如果报文分类算法能够识别冗余规则并将其删除,则可以提高报文分类的性能。删除冗余规则的前提是如何判断哪些规则是冗余的,下面给出判断冗余的两条规则。

1. 向后冗余的规则

规则 7.3.1(向后冗余规则的判断) 设在分类器 C 中的两个规则 $R_t$ 和 $R_s$,符合下面条件:

(1) $t < s$;

(2) $\forall i$,都有 $R_t[i] \supset R_s[i]$,$1 \leqslant i \leqslant k$,其中 k 为分类的维数;

(3) $\forall j > m$,$pri(R_m) > pri(R_j)$,其中 $pri(R_j)$ 表示规则 $R_j$ 的优先级。

这样就永远不会有报文匹配规则 $R_s$ 了,这时规则 $R_s$ 就是冗余的了。我们称这样的冗余为向后冗余。实际调查表明,大约有 4.4% 的规则是向后冗余的。

2. 向前冗余的规则

规则 7.3.2(向前冗余规则的判断) 设在分类器 C 中的两个规则 $R_t$ 和 $R_s$,符合下面条件:

(1) $t > s$;

(2) $\forall i$,都有 $R_t[i] \supset R_s[i]$,$1 \leqslant i \leqslant k$,其中 k 为分类的维数;

(3) $\forall m \in [s,t]$,都有 $A(R_s) = A(R_m)$ 或者 $R_m[i] \cap R_s[i] = \varnothing$,$1 \leqslant i \leqslant k$;

(4) $\forall j > m$,都有 $pri(R_m) > pri(R_j)$,其中 $pri(R_j)$ 表示规则 $R_j$ 的优先级。

这时规则 $R_s$ 就是冗余的了,我们称这样的冗余为向前冗余。实际调查表明,大约 3.6% 的规则是向前冗余的。

可以从分类器中删除掉冗余的规则,从而减少分类器的规模。这样既可以提高分类的速度,也可以减少内存的需求。另外,一些具有相同行为的规则如果大部分规则字段的说明是相同和相近的,则可以进行合并。当把多个规则合并为一个规则之后,同样可以得到像删除规则一样所带来的好处。

### 7.3.3.4 分类域宽的压缩

分类域宽的增加是分类算法速度减慢的原因之一,随着分类字段的增加,分类的域宽也在增加,大多数分类算法的速度都与分类域宽 w 有关,如果能够减少域宽,则可加快分类的速度,同时也能节省内存空间。

第 4 层协议类型长度是 8 个位(可表示 256 种协议类型),但真正在实际中使用的协议类型很少,从 101 个不同的 ISP 收集来的 793 个分类器的研究表明,协议类型仅包含 TCP,UDP,ICMP,IGMP,(E)IGRP,GRE 和 IPNIP,这样就不需要 8 个位表示协议类型,只需要 3 个位就够了。一旦协议类型所需要的位数少了,不但可以减少内存的需求,而且还可以加快分类的速度。

在绝大多数客户-服务器结构中,所有端口可以分为两类[34],一类是 Reserved ports,端口号为 1 ~ 1023,另一类是 Ephemeral ports,端口号大于 1023。Ephemeral ports 通常用在客户端,大多数是由 kernel 指定的,它除了标识一个连接的端点之外并没有其他意义。过滤规则几乎不可能对单独某个大于 1023 的端口感兴趣,比较常见的是用‘ > 1023 ”表示大于 1023(小于 65535)的所有端口。Reserved ports 端口也主要集中在 20 ~ 21(用于 FTP),23(Telnet),25(SMTP(email)),110(POP3(email)),53(DNS),80(HTTP(Web)),68(DHCP),161(SNMP)等十多种常用的端口号上[34],因此用 5 位表示协议类型足够了,而且还留有余地,可以进行扩充。这与协议本身定义的 16 位相差很大,可以节约出来十多位,其余端口则在过滤规则中较少出现,可用 * 表示。另外,客户-服务器结构本身的特性决定了在绝大多数情况下,通信双方的端口一个为 Reserved ports,另一个为 Ephemeral ports。假设我们用 6 位表示端口号,这样,源端口号、目的端口号共用 12 位,协议类型用 3 位,就可以用两个字节表示这三个分类字段,最高位保留。把原来的五维分类缩减为三维分类。表 7.3.3 和表 7.3.4 规定一种将协议值、源目的端口号的不同值分别映射到 3 位和 6 位上,映射后的 16 位字段如图 7.3.2 所示。

表 7.3.3　8 种协议类型号映射成 3 位二进制数

| TCP(6) | UDP(17) | ICMP(1) | IGMP(2) | (E)IGRP(88 | GRE(47) | IPNIP(4) | * |
|---|---|---|---|---|---|---|---|
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

表 7.3.4　17 种端口号映射成 6 位二进制数

| 十进制编码 | 端口号 | 用　途 | 6 位二进制代码 |
|---|---|---|---|
| 0 | 20 | FTP data | 000000 |
| 1 | 21 | FTP control | 000001 |
| 2 | 23 | Telnet | 000010 |
| 3 | 25 | SMTP (E-mail) | 000011 |
| 4 | 53 | DNS | 000100 |
| 5 | 70 | Gopher | 000101 |
| 6 | 79 | Finger | 000110 |
| 7 | 80 | HTTP (Web) | 000111 |

续表

| 十进制编码 | 端口号 | 用　途 | 6 位二进制代码 |
|---|---|---|---|
| 8 | 88 | Kerberos | 001000 |
| 9 | 110 | POP3（E-mail） | 001001 |
| 10 | 111 | Remote procedure call（RPC） | 001010 |
| 11 | 119 | NNTP（News） | 001011 |
| 12 | 68 | Dynamic host configuration protocol（DHCP） | 001100 |
| 13 | 69 | Trivial file transfer protocol（TFTP） | 001101 |
| 14 | 161 | Simple network management protocol（SNMP） | 001110 |
| 15 | 2049 | Network file system（NFS） | 001111 |
| 16 | * | Other ports | 010000 |

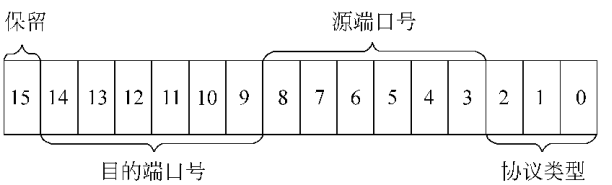



图 7.3.2　协议类型、源端口号、目的端口号映射到 16 位字段内的一种分配图

**下面例子说明如何将表 7.3.5 中五维分类压缩为三维分类，从而减少分类的域宽。**

表 7.3.5　5 个五维的过滤规则

| 规则 | 目的端口号 | 源端口号 | 协议类型 | 目的 IP 地址 | 源 IP 地址 |
|---|---|---|---|---|---|
| 规则 0 | 20～21<br>（0x14～0x15） | 21<br>（0x15） | 6（tcp）<br>（0x06） | 166.111.68.22<br>（0xa66f4416） | 166.111.68.22<br>（0xa66f4416） |
| 规则 1 | 21<br>（0x15） | 21<br>（0x15） | 6（tcp）<br>（0x06） | 166.112.68.23<br>（0xa6704417） | 166.112.68.23<br>（0xa6704417） |
| 规则 2 | 23<br>（0x17） | 20<br>（0x14） | 17（udp）<br>（0x11） | 166.113.68.24<br>（0xa6714418） | 166.113.68.24<br>（0xa6714418） |
| 规则 3 | 20<br>（0x14） | 23<br>（0x17） | 6（tcp）<br>（0x06） | 16.113.68.24<br>（0xa6714418） | 166.112.68.23<br>（0xa6704417） |
| 规则 4 | 23<br>（0x17） | 23<br>（0x17） | 17（udp）<br>（0x11） | 166.112.68.23<br>（0xa6704417） | 166.113.68.24<br>（0xa6714418） |

表 7.3.6　按表 7.3.3 和表 7.3.4 的映射规则将源目的端口、协议类型压缩成一个端口协议字段

| 规则 | 端口协议字段 | 目的 IP 地址 | 源 IP 地址 |
|---|---|---|---|
| 规则 0 | 0x0008，0x0208 | 166.111.68.22<br>（0xa66f4416） | 166.111.68.22<br>（0xa66f4416） |
| 规则 1 | 0x0208 | 166.112.68.23<br>（0xa6704417） | 166.112.68.23<br>（0xa6704417） |

续表

| 规则 | 端口协议字段 | 目的 IP 地址 | 源 IP 地址 |
|---|---|---|---|
| 规则 2 | 0x0401 | 166. 113. 68. 24<br>(0xa6714418) | 166. 113. 68. 24<br>(0xa6714418) |
| 规则 3 | 0x0010 | 16. 113. 68. 24<br>(0xa6714418) | 166. 112. 68. 23<br>(0xa6704417) |
| 规则 4 | 0x0411 | 166. 112. 68. 23<br>(0xa6704417) | 166. 113. 68. 24<br>(0xa6714418) |

## 7.3.4　高速可行的报文分类算法的设计思路

1. 最慢的报文分类算法——线性查找算法

线性查找算法是一个按优先级降序排列的规则链表,一个报文顺序地与每个规则进行比较,直到找到第 1 个与报文匹配的规则(因为规则已经事先按优先级降序排列好了,所以第 1 个匹配的规则就是最佳匹配规则)。算法的优点是简单,而且存储器的利用率很高,但显然这种算法扩展性很差。报文分类的时间随着规则数的增加呈线性增加,所以此算法因分类速度太慢而不可行。

2. 最快的报文分类算法及其缺点

最快的报文分类算法是利用 Hash 函数或其他方法直接计算出每个报文分类字段所对应的规则的位置,例如可以将所要分类的所有字段作为一个整体进行排序,这样用报文头各个字段的值作为索引值进行查找,只要一次内存访问时间就可以找到所匹配的规则。但这种算法要求非常大的内存空间,以 5 维分类为例,IP 源目的地址(各 32 位)、源目的端口号(各 16 位)、第 4 层协议类型(8 位),至少需要 $2^{104}$ 个存储单元。即使是按 IP 源地址的一维分类,如果用这种方法也需要 $2^{32}$ = 4 294 967 296 个存储单元,如果每个存储单元是 32 位,则至少需要内存 17 179 869 184B,所以此算法因内存爆炸而不可行。

一个高速可行的报文分类算法是对前面提到的两个极端化解决方法的折中,是时间和空间的折中,但要找到一个既符合实时需要且以线速转发,同时又对内存的要求比较合理的报文分类算法并不是一件简单的事情。前言中已经谈到,在现有的算法中,一维、二维的分类算法比较多,而对高维(指二维以上)分类支持的算法要么要求的内存空间过大无法满足低成本的要求,要么分类的速度较低无法满足高速网络环境的应用需求。一个高速可行的报文分类算法应该从以下 3 ~ 7 加以考虑。

3. 充分利用分类规则的特性

虽然第 4 层协议类型长度是 8 个位(可表示 256 种协议类型),但真正在实际中使用的协议类型只有 7 种,这样就不需要 8 个位而只需要 3 个位就可以了。一旦协议类型所需要的位数少了,不但可以减少内存的需求,而且还可以加快分类的速度。

再例如,研究了分类规则字段的分布规律之后,可以先查找分类规则中分布最均匀的字段,有可能只查找报文前几个字段就可以提前找到匹配的规则,从而提前结束查找的整个过程。

4. 排序索引思想

通过排序,可以根据索引值进行快速查找,但正如前面所提到的那样,排序需要大量的内存空间。要注意内存空间的爆炸问题,找到一个合适的平衡点是非常重要的。换句话说,分类是一个多目标的优化问题,而不是单目标优化问题。例如可以对要分类的报文字段进行分别排序,这样不仅可以加快分类的速度,同时也可以解决内存空间的爆炸问题。把这种方法与其他方法结合起来就有可能设计出一个好的算法来。

5. 通过增加预处理时间来增加报文分类的速度

因为预处理在真正分类前执行且只执行一次,所以不影响分类时的操作,这样把一些可以在分类时做的工作放在分类之前进行,就可以减少分类时所需要的时间。例如,提前对分类规则的某些分类字段进行预处理(如排序),从而使得在分类时可以利用索引值进行查找,加快分类的速度。再例如,通过在预处理部分统计协议类型字段的种类个数,通过减少表示协议类型的二进制位数,减少整个分类的域宽,从而加快分类的速度。通过删除在预处理部分统计出的冗余规则,减少分类时分类的规则数,也加快了分类的速度。所有这些操作都是通过增加预处理时间来加快分类速度的。

6. 提高内存的访问速度

内存访问是分类算法速度比较慢的主要原因。除了减少内存访问的次数外,还可以利用加快存储器访问速度的技术来提高访问内存的速度。例如,利用交叉访问存储器、并行访问存储器等技术加快内存访问的速度。算法中用到的变量尽量用寄存器而少用内存,在用内存时尽量用芯片内存(on-chip),其次是 SRAM,SDRAM 等。但如果像 SDRAM 技术能被用上,内存的访问代价则可以相对降低。因为这种设备不仅可以提供非常大的容量,而且只要存取是连续的就可以提供较高的存取速度。一个报文分类算法执行时如果要求比较多的对高速的 SDRAM 连续访问,则这种报文分类算法可能比使用高代价、低容量内存(例如 SRAM)的报文分类算法更可行。另外,一个好的数据结构、巧妙的设计思路都可以减少内存的访问次数。需要注意的是,内存是按字长 w 组织的,存取一个字中任何位的子集的代价与存取一个字的代价是相同的。

7. 适当增加内存空间

前面已经提到分类速度与内存的需求是一对矛盾,在内存容许的前提下,通过适当增加内存空间可以提高报文分类的速度。例如前面提到的利用计算几何来研究点的定位问题,如果要求最好的时间复杂度为 $O(\log N)$,则在没有其他技术的优化下,最坏的空间复杂度 $O(N^d)$ 就不可避免 如果要求最好的空间复杂度为 $O(N)$,则在没有其他技术的优化下,同样最坏的时间复杂度 $O((\log N)^{d-1})$ 就不可避免。

8. 充分利用实现算法的硬件平台特性

算法都是在一定的硬件平台上实现的，不同的硬件平台有不同的硬件特性，如是否具有并行部件、是否提供流水线功能（如网络处理器 IXP1200 每条指令的执行都划分为 5 个阶段的流水线）、Cache 的大小也不完全相同等，算法的实现要充分利用硬件平台的这些特性来提高分类算法的速度。

9. 其他加快报文分类的技术

其他加快分类的技术包括是否能够找到好的无冲突的 Hash 函数来快速定位规则的位置，从而加快分类的速度；如果某些规则或者报文命中率比较高，可以考虑 Cache[35] 或者分层次技术[36]；另外，设计好的数据结构始终是加快分类速度的主要技术之一；基于智能的启发式的分类算法可能是未来彻底解决分类问题的一个方向。

10. 理想的报文分类方法

理想的报文分类方法不仅要满足高速度（以线速度分类）、低存储要求（支持大量的分类规则）、快的更新速度、在执行上是可行的（低成本），同时要满足报文头分类字段在个数上的可扩展性和在表达上的灵活性。由于缺乏足够大量的现实规则库的统计数据，所以很难量化这些参数的值[22]。然而不难想象，一个通信公司的边缘路由器支持 1024 个 ISP（Internet service provider）客户，每个客户又有 256 个五维的过滤规则，这样要求路由器的分类引擎要支持 256 × 1024 个过滤规则。

在实践中，理想的报文分类的解决方法很难找到，我们相信，可能的解决办法应该是基于智能的启发式方法。

## 7.4　报文分类的应用

### 7.4.1　区分服务体系结构中的报文分类

在区分服务体系结构中[6]，专门对报文的分类和调节机制进行了阐述，报文的分类主要在边缘路由器中进行，在逻辑上分为分类器（classifier）与调节器（conditioner）两个模块[7]。

1. 分类器

报文分类策略将要接受区分服务的流量分成若干子集，具体可采用对流量进行调节或将其映射至一个或多个行为聚合体。报文分类器根据报文报头某些字段的内容在业务流中选择报文。已经定义两种分类器：行为聚合体分类器，仅根据 DSCP 进行报文分类；多域（multi-field，MF）分类器，根据报文报头多个字段的组合选择报文，如 DS 字段、源地址、目的地址、源端口号、目的端口号以及输入网络接口等。分类器必须在管理过程中根据适当的 TCA 进行配置。

2. 流量调节器

分类器的作用是控制符合某些约束规则的报文进入流量调节器(traffic conditioner)以进行进一步处理。流量调节器则是执行流量调节功能的实体,它包括计量器、标记器、丢弃器和整形器。流量调节器一般在 DS 域的边界节点上实现。流量调节器可能重新标记一个业务流或对业务流的分组进行丢弃和整形以改变其当前属性使其符合某个流量特征描述。分类器选择业务流并将其中的分组导向流量调节器的逻辑实体。计量器根据流量特征描述对业务流进行监测,对于特定的分组,计量器的状态将影响后序的标记、丢弃和整形活动。分类和流量调节一般在 DS 域的入口节点中完成。当报文离开边界节点时,每个报文的 DSCP 必须被设置成一个适当的值。

第 3 章中的图 3.2.3 显示了分类器和流量调节器的逻辑组成框图。应当指出,流量调节器可能不必包含上述全部 4 个组成部分。例如在没有流量特征描述的情况下,分组可能直接通过分类器和标记器。

总之,通过在边缘路由器中设置分类、监控(包括计量和整形)、标记、调度模块,就能在 IP 网络上为不同的业务量提供有区别的 QoS,服务的种类由 IP 报文上的 DS 码点和路由器的 PHB 定义。区分服务与综合服务(IntServ)相比具有以下主要特点:

(1) 区分服务不需要事先预留带宽,因此实现简单;

(2) 区分服务的服务种类由 DSCP 定义,每个 PHB 与一种或几种 DSCP 相对应,转发行为的状态数受限于 DSCP 的个数,与业务流的个数无关,因此具有较好的可扩充性。

业务量的分类、计量、整形、标记在边缘路由器上进行,核心路由器只需根据 IP 报文的 DSCP 分类,进行相应的 PHB 转发,因此使核心路由器的功能简单,易于实现高速转发。

## 7.4.2 报文分类在网络技术领域中的应用

在网络技术领域中需要用到报文分类的地方很多,如防火墙、入侵监测、区分服务等,下面分析一些应用到报文分类的例子。

1. 虚拟专用网(VPN)与拥塞控制

随着 Internet 的迅速发展,大量企业内部网络与 Internet 互联,从而使现代企业网的概念发生了根本性的变化,虚拟专用网(VPN)就此成为建立企业通信网的一种新方法。VPN 通过 IP 网来传输私用数据,如利用 Internet 主干网,采用专用的网络加密技术与通信协议相结合,可以使企业在公共网络上建立安全的虚拟专网。这一功能是通过将安全性与隧道协议相结合来实现的。以往企业的跨地区网络互连通过专线实现,费用很昂贵。利用 VPN,企业远端用户接入是从远程公共网,通过虚拟的加密通道与企业的内部网络连接,而公共网络上的用户则无法穿过虚拟通道访问该企业的内部网络,这样就大大减少了企业构建其 IT 基础设施的费用,同时,网络的灵活性、安全性及全球接入性等方面都有所提高。VPN 的一种实现技术称为路由过滤技术,也称受控路由过滤(controlled router leaking),在这种机制中,服务提供商在路由器中根据报文头的源地址和目的地址字段将

用户划分为不同的通信域，每个通信域构成一个 VPN。路由器控制不同通信域间的信息传送，达到 VPN 的逻辑分割。例如，考虑一个 ISP 想为 VPN 提供带宽保证，为这个应用的报文分类的规则可以是源网络前缀、目的网络前缀、保证的带宽。

对于拥塞的报文分类经常是为选择的协议配置一系列的规则来定义队列的优先级，以便网络设备进行报文的管理。典型地，一个队列算法基于传输层参数，如使用协议、端口号、连接状态，也可能用到用户配置的参数等，对报文进行分类以确保优先级较高的报文不被丢弃或延迟。

2. 基于安全存取列表控制的防火墙与网络入侵检测监控

随着 Internet 的迅速发展和广泛应用，网络安全的重要性日益显现出来。目前网络安全已成为影响网络技术进一步发展和应用的关键因素之一。防火墙作为一种边界安全，能较有效地保护网络的内部安全，例如，采取与安全相关的行为把从某个特定的子网来的报文丢弃。典型的防火墙有包过滤防火墙和应用层代理防火墙。包过滤防火墙工作在 IP 层，它根据 IP 数据包中的有关信息和所定义的规则对数据包进行处理；应用层防火墙工作在应用层，它可以理解高层协议，能针对某类应用制定严格的安全策略。

数据包过滤路由器如图 7.4.1 所示，它检测每个数据包，确定其是否与过滤规则相匹配，根据这些规则，路由器做出容许或禁止的决定。


图 7.4.1　数据包过滤路由器示意图

过滤规则的根据是 IP 数据包报头信息。这些信息包括：IP 源地址、IP 目的地址、封装协议（TCP，UDP，ICMP 或者 IP 隧道协议）、TCP/UDP 源端口、TCP/UDP 目的端口、ICMP 消息类型、数据包引入接口和数据包导出接口等。

如果路由器找到容许数据包通过的规则，它就按照路由表信息发送数据包；若发现禁止通行的规则，则丢弃数据包。若没有规则应用于数据包，则按照用户默认定义的参数决定是发送数据包还是丢弃数据包。

按服务要求过滤的防火墙是按照具体的服务要求决定容许或禁止通信。因为大多数服务监听器都驻留在已知的 TCP/UDP 端口，例如，Telnet 服务器监听 TCP 端口为 23 的远程连接，SMTP 服务器监听 TCP 端口为 25 的引入连接，要阻止所有引入 Telnet 连接，路由器仅需将 TCP 目的端口值等于 23 的数据包丢弃。

入侵检测系统是一个试图确定和隔离计算机系统入侵的一项安全技术。不同的入侵检测系统有不同的报文分类方法，可能用到第 4 层协议，如 TCP，UDP 协议，也可能根据监测目的的不同要用到报文的其他信息。例如，如果想监测攻击 Web 服务器，可能要考虑不怀好意的 HTTP 请求；如果想监测动态路由协议，可能只考虑用到 RIP 欺骗。入侵检测

系统利用报文分类和记录来跟踪传输对象。

3. 资源预留(RSVP)与 QoS 路由

传统 IP 的服务方式是尽力而为,不支持资源预留,甚至不支持简单的优先级方案,新一代的 IP 网将支持资源预留。资源预留与包分类和流标识密切相关,即为预先指定的流作资源预留。在有效的分类器工作基础上,资源预留还需要制定一些策略来限制单个独立的流的要求,或隔离包调度,以阻止超出限制的流。

在提供诸如接入控制、每个流的排队和公平调度机制的 QoS 的路由器中,是根据预先定义的规则,用快速分类算法对每个报文进行分类,进而对分类后的报文头进行匹配以标识和区分不同的流,为流与其性能保证相联系提供基础。例如,采取区分的输出调度机制把 VoIP(voice over IP)报文分配到一个较高的优先权队列中。采用的规则可以是根据数据包中的源地址和目的地址、业务类型、端口号、协议类型和要求等参数来制订,且具有优先级标识。

4. 网络地址转换、传输测量与记账

网络地址转换容许一个单独的设备(例如路由器)在 Internet(公共网)和本地(私有)网之间充当代理,这就说明只用一个 IP 地址就可以代表整个组的计算机,从而为解决 Internet 网 B 类地址耗尽、路由表爆炸和整个地址耗尽的危机提供可行的方法。路由器通过分析公共网的报文头来分类报文,所用的分类规则就是内外网对应的地址、端口转换表,可能用到的分类字段有网络源地址、目的地址、源端口号、目的端口号等。

网络计费是网络管理的重要组成部分,开发一套完善的网络计费系统是每个网络运营部门的首要工作。网络计费主要是对登录用户进行认证和传输量的测量(例如测量两个子网间的传输量),目前网络普遍采用的是以 IP 地址标识用户,采用路由器管理信息数据库中数据,统计该 IP 地址的数据流通量的计费方法。为防止 IP 盗用,采用 IP 地址与网卡 MAC 地址绑定的办法惟一地确定一台计算机,从而为传输记账打下基础。所以分类字段涉及到 IP 地址、MAC 地址等信息。

除了上面介绍的应用以外,报文分类在负载平衡(例如,将报文路由到不同的服务器上)、收集统计数据、路由器和交换机的设计、第 4 层交换、MPLS(multiprotocol label switching)通道的流合并、传输整形以及未来的 IPv6 流标识、异步传输模式(ATM)信元交换等方面都有广泛的应用。

前面讨论了报文分类的各种具体应用,可以依据应用的目的把报文分类的应用进一步归纳为以下三种类型[8]:

(1) 基于报文传递的应用

这方面的例子包括基于第 2 层(L2)的 MAC 地址交换、异步传输模式(ATM)信元交换、多协议表交换(MPLS)以及第 3 层(L3)的 IP 报文传递,实例见表 7.4.1。这些应用的分类操作要求通常是报文头的单个字段,例如第 3 层 IP 传递是根据报文头的目的地址执行一个最长前缀匹配策略。

（2）基于报文过滤的应用

这些例子包括防火墙的报文过滤，虚拟专用网（VPN）的执行和服务质量的应用如综合服务（IntServ）和区分服务（DiffServ）。在典型情况下，这些应用所采用的策略是基于报文头的 L2/L3/L4 字段。这些应用要求规则与报文头的多个字段精确或者前缀/范围匹配，实例见表 7.4.1。

（3）需知道报文内容的应用

有许多新的应用要求分类不仅要基于报文头而且还要基于报文的本身内容。这样的例子包括服务器的负载平衡、入侵检测和病毒扫描等（见表 7.4.1）。

表 7.4.1　三种基本报文分类的应用例子

| 层 | 应　用 | 字段个数 | 规 则 举 例 |
|---|---|---|---|
| 2 | 交换、MPLS | 单个 | 发送具有目标 MAC 地址 68:10:D1:ab:12:7a 的包直接传输到终端主机 |
| 3 | 传递 | 单个 | 发送具有目标 IP 地址 192.168.0.* 的包到 ISP 的路由器 |
| 4 | 流标识、IntServ | 多个 | 让具有源 IP 地址、目的 IP 地址、源端口、目的端口为（192.168.4.10，166.111.10.1，*，21）的报文具有最高的优先权 |
| 4 | 过滤、DiffServ | 多个 | 丢弃所有源 IP 地址为 201.* 并且源端口大于 1020，目的端口小于 5100 的报文 |
| 7 | 负载平衡 | 多个 | 把报文数据里含有以‘.ra’结尾的文件名重新定向到音频服务器上 |
| 7 | 入侵检测 | 多个 | 在数据到达时当报文里有‘get.*vbs’时产生报警 |

尽管不同的服务变化可能很大，但对这些服务来说，一个公共的要求是路由器能够基于报文的头部将这些报文分类到一个等效的称为流的类中。一个流由一个规则定义，例如所有源地址为 S，目的地址为 D 的报文作为一个流。每个流关联一个行为，例如所有属于前面所定义的流的报文被发送到一个具体的队列，或者丢弃它，或者复制它等。所以报文分类不仅关系到网络的控制、性能、安全、管理等方面，同时也是未来网络发展主要研究的基础内容之一。报文分类成为 QoS、区分服务和满足各种不同用户的需要的基本技术，已经成为网络方面的研究热点。

## 7.5　进一步的研究工作

1. 报文分类与网络安全问题

虚拟专用网（VPN）的安全问题、基于安全存取列表控制的防火墙、网络入侵检测监控（network intrusion detection systems，NIDS）、网络地址转换（network address translation，NAT）等都与网络安全有关，同时它们都是基于报文分类的。报文分类速度的快慢、分类能力的强弱都直接影响到这些网络技术所提供的网络安全的强弱。例如，如果分类仅对

源 IP 地址、目的 IP 地址分类,那么防火墙的能力就非常有限,但如果不仅可以对源 IP 地址、目的 IP 地址进行分类,而且还可以对源端口、目的端口、协议类型等进行分类,甚至可以对源 MAC 地址、目的 MAC 地址和报文的净负荷进行分类,则防火墙的能力将大大增强。所以报文分类能力的增强将有利于网络安全的提高。但是一些安全技术,例如加密技术,要对报文的一些信息进行加密,这使得路由器难以提取报文分类所需要的信息,不利于分类,反过来又影响了诸如防火墙的安全技术的能力。如何解决这些冲突,也要根据不同的网络应用环境找到一个平衡点。

2. IPv6 的报文分类问题

IPv6 可分类的字段有流量标识、优先级、源地址、目的地址,在同时支持 IPv4 和 IPv6 的情况下,版本号也可以作为分类的字段,详见图 7.1.8。由于 IPv6 源、目的地址均为 128 位,是 IPv4 地址的 4 倍,所以报文分类的域宽增加得比较大,而一般的报文分类速度都随着分类的域宽增大而减慢,故对 IPv6 的分类是报文分类的一个挑战。由于缺乏对 IPv6 规则特征的统计数据,目前还没有提出针对 IPv6 的好的报文分类办法。我们认为可以考虑下面几个方面:①利用 IPv6 地址的规律特性来减少分类的域宽;②利用智能的启发式方法和分类器的特性来加快分类的速度;③对分类器的特性和规律进行统计和研究也是解决 IPv6 报文分类的前提和基础。

3. 通用报文分类方法的设计

现有的报文分类方法多是针对固定维数的,并且只根据报文头某些字段进行分类,但随着网络的发展,新的网络应用技术将不断产生,不同的应用所要求的分类的维数不同,即使分类维数相同,分类的字段也可能不同。一些基于安全策略的防火墙、负载平衡技术不仅要求针对报文头的某些字段进行分类,而且要针对报文的净负荷进行分类,不同的应用可能要求规则库的更新速度也不一样。总之,到目前为止,还没有设计出一种通用的报文分类方法来满足各种不同的要求。是设计出适合不同应用的分类算法好,还是设计出一种适用于多种网络技术需要的、并可根据需要进行组合模块化的、可用参数进行灵活配置的报文分类方法好,现在还不能盲目下结论,有待进一步研究,这要在报文分类的性能和通用性上找到一个平衡点。

4. 理想的报文分类问题

理想的报文分类方法不仅要满足高速度(以线速度分类)、低存储、更新速度快、在执行上是可行的(低成本)等要求,同时还要满足报文头分类字段在个数上的可扩展性和在表达上的灵活性。由于缺乏足够多的现实规则库的统计数据,所以很难量化这些参数的值[22]。然而不难想象一个通信公司的边缘路由器支持 1024 个 ISP 客户,每个客户又有 256 个五维的过滤规则,这样要求路由器的分类引擎要支持 256 × 1024 个过滤规则。

在实践中,理想的报文分类的解决方法很难找到,我们相信可能的解决办法应该是基于智能的启发式方法。

5. **报文分类模型及其综合性能分析**

**随着各种报文分类算法的不断出现,如何评价这些报文分类算法的性能也会逐渐显得重要起来。前面已经给出了一些衡量报文分类算法性能的参数,这些参数是分析报文分类算法性能的基础,但不同的算法性能的侧重点又不一样,用不同的参数衡量算法的性能会得出不同的结论,很难说哪个好哪个坏。如何给出一种综合的评价标准,如何设计出好的报文分类模型,确定出哪些模块是报文分类通用的,可以固化到硬件中实现,哪些模块是随不同的应用变化的,可以用软件实现,这些都是需要进一步研究的问题。报文分类是其他网络技术(如 buffer 管理,报文整形,调度等)的基础,所以必须与其他网络控制机制算法协调工作。如何进行协调才能达到系统性能的最优化也是研究报文分类问题的一个方向。因为若它们在同一个硬件平台上实现的话,则会出现争夺系统资源的情况。两个各自性能都比较满意的算法,因为没有协调好对系统资源的利用策略,结合起来性能可能很差。**

**总之,报文分类是许多网络技术的基础,它关系到网络的控制、性能、安全、管理等多方面内容,分类速度的快慢、功能的强弱都直接影响到这些网络技术的性能。已有的研究表明,实现高速多维报文分类算法是非常困难的。在现有的算法中,一维、二维的分类算法比较多,而对高维分类支持的算法要么要求的内存空间过大无法满足低成本的要求,要么分类的速度较低无法满足高速网络环境的应用需求,如果再加上要求更新复杂度小、规则个数具有可扩展性、分类维数具有可扩展性、在分类层次上具有可扩展性、规则表达具有灵活性、预处理时间少、算法执行的灵活性等,则分类是一个很复杂的多目标的优化问题,它已成为路由器的新瓶颈。**

## 参考文献


1　Gupta P, McKeown N. Algorithms for packet classification. IEEE Network, 2001,15(2):24 ~ 32

2　Hari A, Suri S, Parulkar G. Detecting and resolving packet filter conflicts. In: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proc IEEE, 2000,3:1203 ~ 1212

3　Comer D E. Internetworking with TCP/IP Vol. Ⅰ: Principles, Protocols, Architectures. 4th ed. Prentice-Hall, Inc., 2000. 66 ~ 73

4　Tanenbaum A S. Computer Networks. 3rd edition. Prentice-Hall, Inc., 2000. 437 ~ 448

5　Woo T Y C. A modular approach to packet classification: Algorithms and results. In: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proc IEEE, 2000,(3):1213 ~ 1222

6　Blake S, Black D, Carlson M, Davies E, Wang Z, Weiss W. An architecture for differentiated services. IETF Internet RFC 2475, December 1998

7　**林闯, 单志广, 盛立杰, 吴建平**. Internet **区分服务及其几个热点问题的研究**. **计算机学报**, 2000, 23(4):419 ~ 433

8　Iyer S, Rao Kompella R, Shelat A. ClassiPl: An architecture for fast and flexible packet classification.

IEEE Network, 2001,15(2):33 ~ 41
9 Srinivasan V, Varghese G, Suri S, Waldvagel M. Fast and scalable layer four switching. In:Proc of ACM SIGCOMM'98. August 1999. 191 ~ 202
10 Lakshman T V, Stiliadis D. High speed policy-based packet forwarding using efficient multi-dimensional range matching. In:Proc of ACM SIGCOMM'98. Vancouver, Canada, September 1998. 203 ~ 214
11 Srinivasan, Suri S, Varghese G. Packet classification using tuple space search. In:Proc of SIGCOMM'99. 1999
12 Gupta P, McKeown N. Packet classification on multiple fields. In:Proc of ACM SIGCOMM'99. August 1999. 147 ~ 160
13 Memory-memory. http://www.memorymemory.com, 2000
14 Netlogic Microsystems. http://www.netlogicmicro.com
15 www.openskytech.com/ContentAddressableMemory.htm
16 Kohler E, Morris R, Chen B, Jannotti J, Kaashoek M F. The click modular router. ACM Trans on Computer Systems, 2000(18):263 ~ 297
17 Buddhikot M M, Suri S, Waldvogel M. Space decomposition techniques for fast layer-4 switching. In: Proc Conf Protocols for High Speed Networks. Salem, MA, USA:Kluwer Academic Publishers, August 1999. 25 ~ 41
18 Pankaj Gupta, Nick McKeown. Packet classification using hierarchical intelligent cuttings. In:Proc ACM Sigcomm'98. September 1998
19 Xu J, Singhal M, Degroat J. A novel cache architecture to support layer-four packet classification at memory access speeds. In:Proc Infocom, 2000,(3)
20 Feldmann A, Muthukrishnan S. Tradeoffs for packet classification. In:Proc INFOCOM 2000. IEEE, 2000,(3):1193 ~ 1202
21 Thompson K, Miller G J, Wilder R. Wide-area Internet traffic patterns and characteristics. IEEE Network, 1997,11 (6):10 ~ 27
22 Kijkanjanarat T. Fast routing lookup and packet classification for next-generation router. Ph. D. Dissertation. Polytechnic University. January 2002
23 Lakshman T V, Stidialis D. High speed policy-based packet forwarding using efficient multi-dimensional range matching. In:Proc of ACM S'98. September 1998. 203 ~ 214
24 Kohler E, Morris R, Chen B, Jannotti J, Kaashoek M F. The click modular router. ACM Trans on Computer Systems, 2000(18):263 ~ 297
25 Brodnik A, Carlsson S, Degermark M, Pink S. Small forwarding tables for fast routing lookups. In:Proc ACM SIGCOMM 1997. Cannes, France. 1997. 3 ~ 14
26 Lampson B, Srinivasan V, Varghese G. IP lookups using multiway and multicolumn search. In:Proc Conf on Computer Communications (IEEE INFOCOMM). San Francisco, California, March/April 1998,(3): 1248 ~ 1256
27 Waldvogel M, Varghese G, Turner J, Plattner B. Scalable high-speed IP routing lookups. In:Proc ACM SIGCOMM 1997. Cannes, France, 1997. 25 ~ 36
28 Gupta P, Lin S, McKeown N. Routing lookups in hardware at memory access speeds. In:Proc Conf on Computer Communications (IEEE INFOCOMM). San Francisco, California, March/April. 1998,(3): 1241 ~ 1248
29 Srinivasan V, Varghese G. Fast IP lookups using controlled prefix expansion. ACM Trans on Computer

Systems，1999，17(1)：1～40
30 de Berg M，van Kreveld M，Overmars M. Computational Geometry：Algorithm and Applications. 2nd rev ed. Springer-Verlag，2000
31 Preparata F，Shamos M I. Computational Geometry：An Introduction，Springer-Verlag，1985
32 Rose C. Rapid optimal scheduling for time-multiples switches using a celluar automation. IEEE Trans on Communication，1989，37(5)：500～509
33 Overmars M H，van der Stappen A F. Range searching and point location among fat objects. Journal of Algorithms，1996，21(3)：629～656
34 Steven W R. UNIX Networking Programming vol 1. 2nd ed. Englewood Cliffs，NJ：Prentice-Hall，1998
35 Ata S，Murata M，Miyahara H. Efficient cache structures of IP routers to provide policy-based services. IEEE Int'l Conf on Communications，2001(5)：1561～1565
36 Baboescu F，Varghese G. Scalable packet classification. In：ACM SIGCOMM'01. San Diego，California，USA，2001. 199～210
37 **田立勤，林闯**. IP **报文分类技术的研究及其应用**. **计算机研究与发展**，2003，40(6)：765～775

# 第8章

# 流量整形与监测

在数据传输的过程中，路由器将根据传输好的传输模式对流量进行整形。所谓传输整形是调整数据传输的平均速率，是数据按照传输模式规定的速率进行传输，尽量避免突发性通信量导致的拥塞问题[1]。流量整形主要采用两种基本算法：漏桶（leaky bucket）算法和令牌桶（token bucket）算法。

## 8.1 漏桶算法

漏桶算法[2]是将流量整形操作形象地比喻成一个底部带有一个小孔的水桶，无论流入桶中的水速有多大，从底部小孔流出的水速是恒定的。如果桶中无水，则速率为0；如果桶中水满，则流入桶中的水将从桶边溢出而流失掉。

漏桶相当于一个分组队列（缓冲区），当队列满时，分组被丢弃，队列按规定的速率向网络发送分组，如图8.1.1所示。如果分组长度固定，则以分组为单位，每隔一定的时间发送一个分组，否则，应规定队列每次发送的最大字节数（字节计数器）。

这里举一个漏桶算法的例子。假设主机和网络的数据速率都是25MB/s，路由器在短时间内也能处理这样高的速率，但其处理能力较弱，在较长的时间里，进入路由器的平均数据速率最好不超过2MB/s。假设主机产生的数据是突发性的，每秒产生一个突发数据块，数据块长度为1MB。为了限制数据进入网络的平均速率，应选择漏桶的传输速率为2MB/s，漏桶的容量为1MB/s，漏桶的容量为1MB，这样，漏桶每次最多可装入1MB的数据，不会造成数据丢失。如果不采用漏桶算法，1MB的数据块在1MB÷25MB/s=40ms时间内就会全部流入网中；而采用漏桶算法之后，需要1MB÷2MB/s=500ms时间才能全部进网，平滑了原来波动很大的流量曲线。

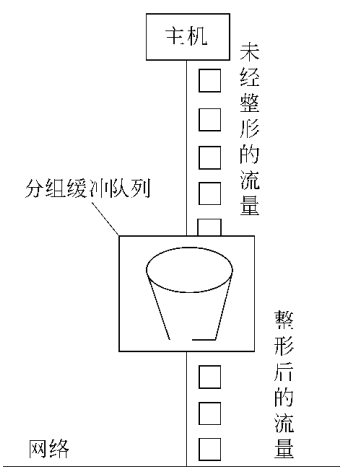



图8.1.1 漏桶算法示意图

## 8.2　令牌桶算法

漏桶算法[2]的缺点是，无论数据量多大，总以恒定的速率发送数据，且当漏桶满时，数据丢失。而大多数应用程序希望有一种更好的算法，在突发数据到来时，能较快地给予响应，同时最好不要丢失数据，令牌桶就具有这样的特点。如图 8.2.1 所示，令牌桶装的是令牌而不是分组，每个 Δt 时间，产生一个令牌，放入令牌桶，令牌桶满后，新产生的令牌将丢弃。分组在桶外缓冲区中排队，桶中有多少个令牌就可以发送相应于令牌个数的分组，当桶空时，停止发送数据，新来的分组就要等待生成新令牌或被丢弃。由于突发性的输入流往往导致拥塞的发生，因此获得令牌的分组将被快速地输出，使突发性的输入流得以迅速地疏导。

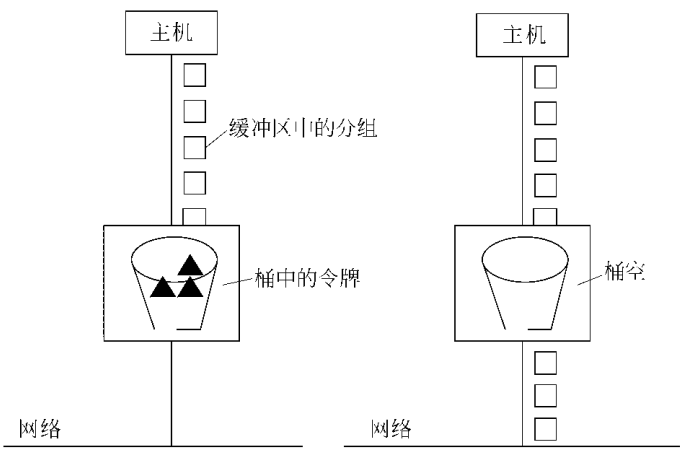



图 8.2.1　令牌桶算法示意图

一个令牌表示可发送一个分组，也可表示发送一定数量的字节。若一个令牌表示允许发送一个分组，则每个 Δt 时间，令牌计数器的值加 1（受桶容量的限制）。每发一个分组，计算器的值减 1，当计数器的值等于 0 时，则停止发送分组。若一个令牌表示允许发送 K 个字节，则令牌桶相当于一个字节计数器，每个 Δt 时间，令牌计数器的值加 K（受桶容量的限制）。每发一个分组，计算器的值减去分组长度（字节），当计数器的值小于下一分组长度时，则停止发送。

令牌桶的优点是：当主机空闲时，可积累令牌，积累的令牌数量由令牌桶容量来确定，因此，令牌桶有利于发送突发性数据，突发数据量由令牌数量决定。当令牌桶满时，丢弃的是令牌而不是分组。

假设突发数据到达时，令牌桶是满的。设突发长度为 S 秒，其容量为 C 字节，令牌产生速率为 r 字节/秒，数据的峰值速率为 M 字节/秒，则 MS = C + rS，即突发长度 S = C/（M - r），表明在开始时，可按峰值速率 M 连续发送 S 秒，然后按速率 r 继续发送，直到结束。

## 8.3 滑动窗口协议

滑动窗口协议[3]在数据链路层和传输层都可以用来进行流量控制。它们的区别在于:在数据链路层,流量控制是作用在单条链路上的;在传输层,流量控制是端到端执行的。而且传输层中的窗口在大小上可以发生变化以适应缓冲区的占用。下面分别进行讨论。

### 8.3.1 数据链路层的滑动窗口协议

在大多数协议中,流量控制是一组过程,用来告诉发送方在等待接收方的确认信号之前最多可以传送多少数据。数据流不能使接收方过载。任何接收设备都有一个处理输入数据的速率限制,并且存储输入数据的存储器的容量也是有限的。接收设备必须在达到这些限制之前通知发送设备,并且请求发送设备发送较少的数据帧或是暂停一会儿。在使用输入数据之前必须对它们进行校验和处理。这种处理的速率通常比传输速率要低。因此,每个接收设备都有一块存储器,叫作缓冲区,用来在进行处理之前保存输入数据。如果缓冲区即将满,接收方必须能够通知发送方暂停传输,直到接收方又能接收数据。

流量控制是一组过程,用来限制发送方在等待确认前可以发送的数据量。

在流量控制的滑动窗口方法中,发送方在需要发送前可以发送若干帧。帧可以直接依次发送,这意味着链路上可能同时承载了几个数据帧,从而充分有效地使用了链路的能力。接收方只对其中的一些帧进行确认,使用一个 ACK 帧来对多个数据帧的接收进行确认。

在流量控制的滑动窗口方法中,一次可以传输多个帧。

滑动窗口是指发送方和接收方都要创建的额外的缓冲区。这个窗口可以在两端存储数据帧,并且对收到确认之前可以传输的数据帧的数目进行限制。可以不等待窗口被填满而在任何一点对数据帧进行确认,并且只要窗口未满就可以继续传输。为了记录哪些帧已经被传输以及接收了哪些帧,滑动窗口引入了一个基于窗口大小的标识机制。帧以模 n 的方式标号,也就是说从 0 到 n - 1 编号。例如,如果 n = 8,则帧标号就为:0,1,2,3,4,5,6,7,0,1,2,3,4,5,6,7,0,1,…。窗口的大小是 n - 1(在本例中是 7)。换一种说法,也就是窗口不能覆盖所有编号(8 帧),能覆盖的数量为所有编号的帧数减 1。在本节的后面将讨论这样设置的原因。

当发送方发出一个确认帧时,它就在其中包含了预期接收的下一帧的编号。也就是说,为了对以帧 4 结尾的一串数据帧进行确认,接收方发送了一个包含有编号 5 的确认帧。当发送方收到含有编号 5 的确认帧时,它就知道了到编号 4 为止的所有数据帧均已被接收了。

在两端的窗口都可以存储 n - 1 帧。因此在必须接收一个 ACK 帧之前最多可以发送 n - 1 帧。图 8.3.1 表明了窗口和主缓冲区之间的关系。


图 8.3.1　滑动窗口

1. 发送方窗口

在传输的开始，发送窗口有 n - 1 个帧。随着数据帧的发送，窗口的左边界向内移动，不断缩小窗口。如果窗口的大小是 w，并且自从最近一次确认以来已经发送了 3 帧，那么在窗口中剩余的帧数是 w - 3。一旦一个确认帧到来，窗口根据确认帧确认的数据帧的数量对窗口进行相同数目的扩展。图 8.3.2 显示了一个大小为 7 的发送方的窗口。

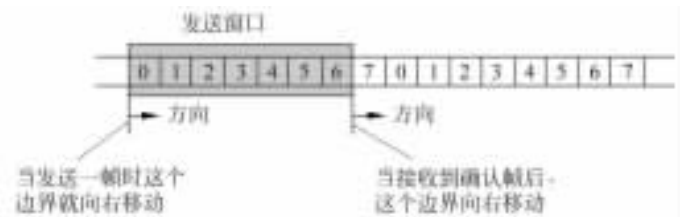



图 8.3.2　发送方滑动窗口

如图 8.3.2 所示，假定窗口大小为 7，若第 0 ~ 4 帧已经发送而且还没有收到确认，那么发送方窗口就含有两帧（5 号和 6 号）。现在，如果收到编号为 4 的确认帧，就可以知道有 4 帧（0 ~ 3 帧）已经无损坏地到达，因此发送方就扩展其窗口以包括它的缓冲区中接下来的 4 帧。此时，发送方窗口包含有 6 帧（编号为 5，6，7，0，1，2）。如果接收了一个编号为 2 的确认帧，则发送方窗口只会扩展两帧，总共包含 4 帧。

从概念上讲，当数据帧发送出去时，发送方滑动窗口从左边开始收缩。当收到确认时，发送方滑动窗口向右扩展。

2. 接收方窗口

在传输开始时，接收方窗口不是包含有 n - 1 个帧而是包含有 n - 1 个帧空间。随着新数据帧的到来，接收方窗口不断缩小。因此接收方窗口并不代表接收的数据帧数，而是代表在必须发送确认帧前还可以接收的帧的数目。假定窗口的大小是 w，如果在没有返回确认前接收了 3 帧，那么在窗口中剩余的空间数是 w - 3。一旦发送了一个确认，窗口就按照确认的帧的数量来扩展。

图 8.3.3 显示了一个大小为 7 的接收方窗口。在图中，窗口中含有容纳 7 帧的空间，意味着在发送 ACK 帧之前可以接收 7 个数据帧。随着第 1 个帧的到来，接收窗口开始收缩，边界从空间 0 变为空间 1。窗口缩小了一帧，因此接收方在要求发送确认帧之前还可以接收 6 帧。如果 0 号帧到 3 号帧已经到达但是未进行确认，窗口就只含有三帧的空间。

从概念上来说，当收到数据帧时，接收方滑动窗口从左边开始收缩。当发送确认时，接收方滑动窗口向右扩展。

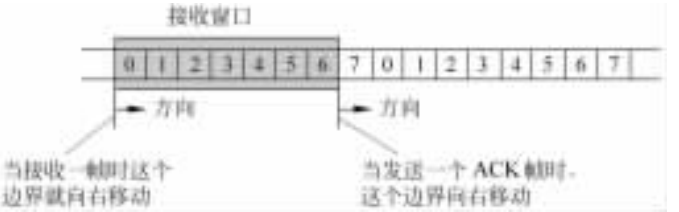



图 8.3.3 接收方滑动窗口

随着每个 ACK 帧的发出,接收窗口按照新确认的帧的数量扩展相同数目的空位。窗口扩展的数目等于最近的确认帧中包含的编号减去上一次确认帧中包含的编号。在一个 7 帧窗口中,如果上一次的确认是对 2 号帧而当前的确认是对 5 号帧,那么窗口就扩展 3 个空间(5 - 2)。如果上一次的确认是对 3 号帧而当前的确认是对 1 号帧,那么窗口的扩展数是 6(1 +8 - 3)。

3. 实例

图 8.3.4 显示了利用一个 7 帧窗口的滑动窗口流量控制进行传输的例子。在该例中,假设所有的帧都无差错到达。在传输开始,发送方和接收方的窗口大小都完全扩展到 7 帧大小(发送方窗口中有 7 个可传输帧,接收方窗口中有 7 个帧空位)。窗口中的帧从 0 ~ 7编号,并且是更大的数据缓冲区的一部分,其中的 13 帧显示在图中。

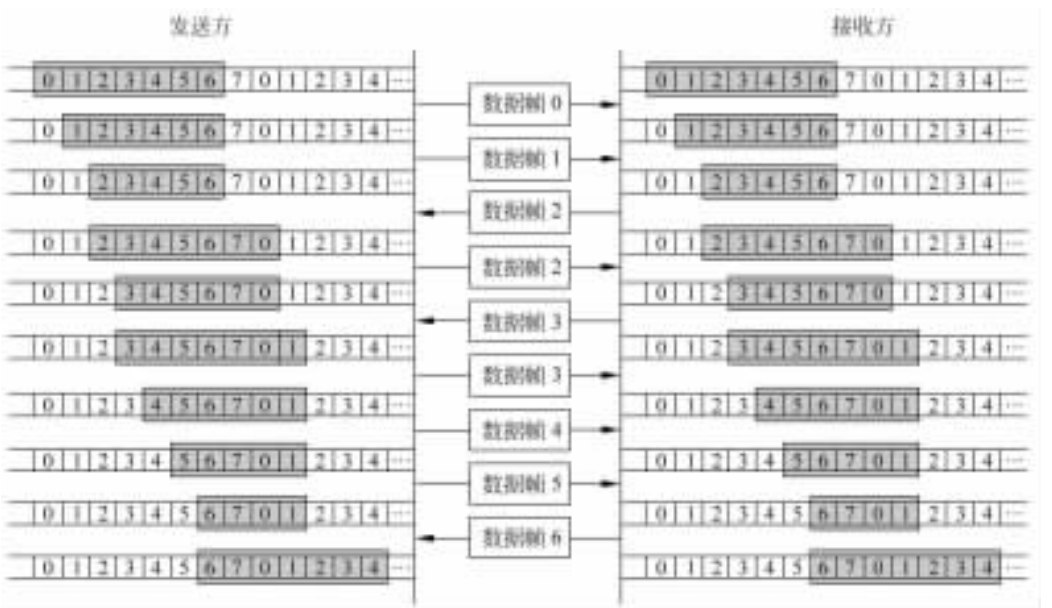



图 8.3.4 滑动窗口实例

4. 关于窗口大小的其他内容

在流量控制的滑动窗口方法中，为了避免在接收的帧的确认中出现二义性，窗口的大小比模数要小 1。假定帧的序列号是模 8，并且窗口大小也是 8。现在假定发送了 0 号帧却收到编号 1 的确认帧（ACK1）。发送方扩展窗口并发送帧 1，2，3，4，5，6，7 以及 0。如果此时又收到了 ACK1，它就不知道到底这是前一次 ACK1 的重复（网络重复）还是确认最近发送的 8 帧的新的 ACK1。但如果窗口的大小是 7（而不是 8），这种情况就不会发生。

## 8.3.2　传输层的滑动窗口协议

正如前面所说，在传输层，流量控制是端到端执行的，而且传输层中的窗口在大小上可以发生变化以适应缓冲区的占用。

由于窗口的大小是可变的，因此，窗口实际可以容纳的数据量是可以协商的。在大多数情况下，窗口大小的控制是接收方的责任。接收方在确认包中可以指定窗口的大小是递增的（或递减的，但多数协议不容许递减）。在多数情况下，传输层的滑动窗口是基于接收方可能容纳的字节数，而不是帧数。一对通信实体可以使用能够容纳 y 帧的 x 字节大小的缓冲区。

滑动窗口用来使数据传输更加有效，同时也用来控制数据的流量，使接收方不会变得过分拥挤。在传输层中使用的滑动窗口通常是面向字节而不是面向帧的。

传输层中滑动窗口有如下一些特点：

（1）发送方不必发送满这个窗口的数据；

（2）一个确认可以基于确认数据段的序列号来扩展窗口的大小；

（3）窗口的大小可以由接收方来递增或递减；

（4）接收方可以随时发送确认。

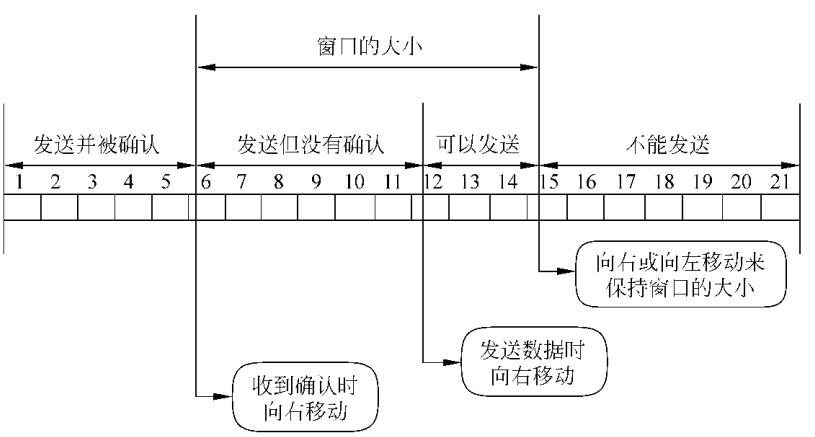



图 8.3.5　滑动窗口

为了适应大小上的变化,传输层滑动窗口使用三个指针(其作用就像虚拟墙一样)来识别缓冲区(如图 8.3.5 所示)。当收到确认时,左边界向右移动,当发送数据时,中间边界向右移动,而右边界向左或向右移动来调整窗口的大小。如果收到确认,而窗口的大小没有变化,则第 3 个边界将向右移动以保持窗口大小的恒定(因为左边界的墙向右移动了)。例如,如果 5 个字节被确认了,而窗口的大小没有改变,那么左边界向右移动 5 个字节,缩小了窗口,因此右边界必须向右移动 5 个字节以保持窗口大小不变。如果 5 个字节被确认,而接收方也将窗口大小扩大了 10 个字节,这时右边界必须向右移动 15 个字节来适应新的窗口大小。

## 参考文献

1 Lin L, Lo J, Ou F. A generic traffic conditioner. IETF Internet Draft 〈draft-lin-diffserv-gtc-01.txt〉, August 1999

2 谢晓尧. 计算机网络. 重庆:重庆大学出版社, 2001

3 吴时霖,周正康,吴永辉等译. 数据通信与网络. 北京:机械工业出版社,2002.1

# 第9章

# 队列管理

从排队论的角度来看，可以把分组（或信元等）流经网络传输节点（路由器、交换机等）的过程用一个顾客-服务器模型来描述。而在网络传输节点的控制策略中设置队列缓冲的目的旨在通过一定程度上增大顾客的等待时间——即分组的排队延迟，提高服务器的利用率——即输出链路的带宽利用率。

随着网络技术的发展，用户对于基于网络的实时、多媒体应用的兴趣和需求的不断提高 如何发展、规划、改造现有的网络，以适应这样的变化，并满足用户多样的需求也成为研究领域和业界亟待解决的问题。服务区分——即提供不同等级的传输服务，以满足不同类型的用户需求——的思想逐渐浮出水面。基于多队列的控制方案在业务隔离、资源公平分配等方面表现出极大的优势，已经成为网络传输控制的必然选择。

广义的队列管理是指以队列为依托，对到达的数据分组、数据流进行传输控制的多种机制，主要涵盖缓冲管理和分组调度两个方面的内容。而狭义的队列管理仅指对网络传输节点中队列缓冲资源的管理和分配——即缓冲管理。本章将仅限于缓冲管理展开讨论，而关于分组调度部分的内容将在第 10 章中给出。

## 9.1 缓冲管理的意义

缓冲管理即是对网络传输节点中队列缓冲资源的管理。在分组传输过程中，其流经的网络传输节点通常采用队列缓存、延迟转发的服务方式以提高输出链路的带宽利用率。缓冲管理机制在分组到达队列前端时依据一定的策略和信息决定是否允许该分组进入缓冲队列，从另一个角度看，也就是做出是否丢弃该分组的决策，因此也称为丢弃控制。

缓冲管理在网络传输控制中发挥着相当大的作用，是网络服务质量（QoS）控制的核心技术之一，也是实现网络拥塞控制的重要手段。

### 9.1.1 对于 QoS 控制的意义

就单个网络传输节点而言，其控制目标在于解决输出链路的带宽资源分配问题，把有限的资源公平而有效地分配给不同的服务类别（或用户流等）。而在众多网络传输节点构成传输网的基础上，网络传输控制需要整合、规划所有的网络带宽资源的使用，为用户

提供端到端的有服务质量保障的网络传输服务,这也就是 QoS 控制的目标。

带宽资源的分配在网络传输节点上主要通过基于多个队列的分组调度机制来实现。虽然缓冲管理机制直接涉及到的是节点中的队列缓冲资源,直接影响到的也只是分组丢失率性能,然而其对系统带宽分配的性能有着不可忽视的影响:

(1) 合理的系统队列缓冲容量,对于平衡系统吞吐量和分组排队延迟起着至关重要的作用;

(2) 在多队列情况下,缓冲资源在不同队列(服务类别)之间的分配只有在与输出带宽的分配相互一致时,才能获得最佳的缓冲效果。

因此,更多的研究者注意到把传统上相互独立、缺乏联系的缓冲管理机制和分组调度机制两者相结合,研究更优的资源分配方案,以期获得更优的系统性能[19,20]。如何与分组调度机制有效配合,解决网络带宽资源的有效、公平分配问题,是缓冲管理机制设计的关键。

## 9.1.2 对于拥塞控制的意义

在早期计算机网络中,还没有网络传输控制的概念,网络传输节点采用先入先出队列(first in first out,FIFO)配合尾部丢弃的队列控制策略对到达的数据分组排队。1986 年,计算机网络遭遇了历史上的第一次拥塞崩溃(congestion collapse),从伯克利到 LBL 的链路有效网络流量从 32Kbps 降到 40bps,下降了将近 1000 倍。从那时起,人们意识到有效利用网络资源的意义,并开始研究解决网络拥塞问题的机制和方案。

最早的拥塞控制机制出现在 TCP 协议中,由网络用户(通常为用户使用的传输协议,对于用户而言是透明的)根据丢包、超时等现象来判断网络是否出现拥塞,并采用慢速启动和拥塞避免算法来调整自己的数据发送速率,缓解传输网络的压力。随着显示拥塞通告(explicit congestion notification,ECN)和主动队列管理(active queue management,AQM)的思想的提出,拥塞控制不再只是网络用户的责任,在网络的传输节点中也引入了拥塞控制的机制。如何有效配合网络用户采用的协议,尽量避免拥塞的出现 如何区分出不遵守拥塞控制规范的网络用户,并限制其不至于影响其他用户和整个网络的有效运行等,都成为网络传输节点中的队列缓冲管理机制需要解决的关键问题。

图 9.1.1 给出了采用输入输出队列结构路由器中的分组处理流程。通常情况下,为了提供有效的 QoS 支持,队列结构都放在输出端。缓冲管理机制位于队列的输入端,负

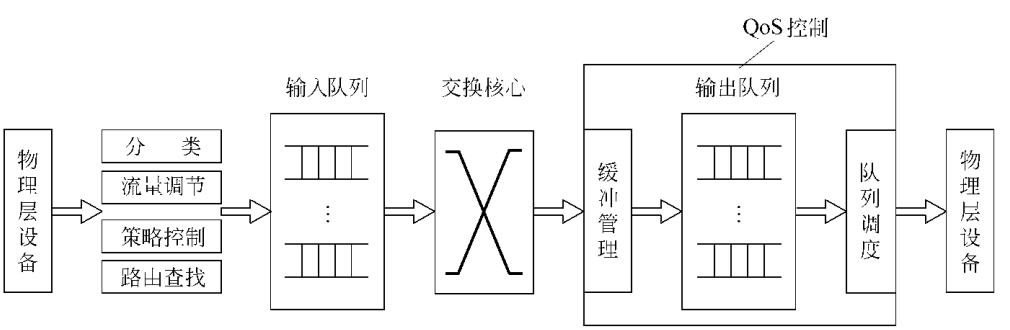



图 9.1.1 具有 QoS 支持的输入输出排队路由器中的分组处理流程

责管理系统中缓冲资源的分配，根据系统策略和到达分组的信息来决定是否允许其进入队列；而在队列的输出端，则有队列调度机制负责带宽分配和延迟调整，两者互相配合完成完整的队列操作。而在队列操作机制之前，根据系统采取的策略和控制算法，可以辅助以分组分类、流量整形/调节等机制来配合队列管理机制的要求。

## 9.2　缓冲管理的目标

### 9.2.1　系统吞吐量与分组排队延迟

网络传输控制中通常采用的队列结构模型是为了提高输出链路的带宽利用率而设置的。在没有排队的情况下，到达的分组或者被丢弃，或者立刻获得服务，分组排队延迟可以降到最低。然而这是以低系统吞吐量和高分组丢失率为代价的——大量分组到达时将因为服务器忙而被丢弃，而在链路空闲时服务器又将因为没有服务对象而闲置，造成系统资源的极大浪费。

队列缓冲的设置使链路的带宽利用率和系统吞吐量得以改善，然而同时也增大了分组排队延迟，在队列缓冲空间的容量增大时情况尤为严重。传统的 Internet 网络提供的数据传输业务只保证数据的正确性和完整性，并不提供基于数据传输延迟的保障。而随着网络实时应用的发展，用户对数据传输延迟的要求越发苛刻，要求网络传输业务能够提供尽可能低的、稳定的传输延迟。而分组在网络中传输遭遇的延迟的最主要部分，也是最容易控制的部分，就来源于网络传输节点的排队延迟。因此，如何设置合适的队列缓存空间容量，如何在网络运行期间合理地控制队列长度的动态变化，以平衡系统吞吐量和分组排队延迟之间的矛盾，是缓冲管理乃至整个网络 QoS 控制需要解决的重要问题。

### 9.2.2　系统的缓冲与带宽资源

显而易见，缓冲管理是管理系统中队列缓冲资源的分配机制，然而仅从缓冲资源的管理这一个方面来看待缓冲管理却有失偏颇。

在网络传输节点中，最重要的资源是输出链路的带宽资源，配合以处理资源和缓冲资源，实现完整的交换、排队等控制机制。因此，网络传输控制机制的首要目标是要在到达的用户数据流之间合理地分配输出链路的带宽资源，这主要是通过基于队列的分组调度机制来实现的。相应地，缓冲管理机制需要完成两个层面的控制需求：

（1）与分组调度机制的带宽分配相配合，提高缓冲资源的利用效率

当缓冲管理机制的缓冲分配和分组调度机制的带宽分配不相一致时，将会导致缓冲资源的利用效率降低，甚至影响到带宽的公平分配。理想情况下，瞬时的队列长度应该与其分配的系统带宽成比例，这样才能获得最大效率的缓冲资源利用。然而，在实际网络环境中，网络流量具有很强的突发性，随着到达的用户数据流量的变化，系统的带宽分配也是实时变化的，这些都给缓冲管理机制的设计带来了相当大的困难。

（2）缓冲管理机制作为一种资源分配机制，提高缓冲资源的利用率也是其中一个重要目标

在系统队列缓冲容量经过合理设计的情况下,如何尽可能地提高所有缓冲资源的利用率成为缓冲控制面临的又一大问题。与上面(1)的不同之处在于,前者关注的是如何分配缓冲资源能够获得更大的系统效率,而后者的关注点在于如何尽可能完全地把缓冲资源分配出去,以降低分组丢失率。

传统的 Internet 网络采用完全共享的队列控制策略,不提供服务区分,虽然无法提供服务质量保障和用户数据流之间的公平性,但确实获得了最佳的队列缓冲资源利用率。为了提供服务质量保障,目前缓冲管理方案主要采用以下两种不同的控制策略:

(1) 动态资源预留

众多缓冲管理方案采用各种形式的资源预留机制为较高等级的应用提供更好的服务保障,同时不可避免地降低了队列缓冲资源利用率。因而,动态资源预留和流量预测成为其提高性能的关键。

(2) 选择性丢弃

另一种有效的控制策略是不预留任何队列缓冲资源,在可能的情况下把缓冲资源完全分配出去,而在需要做丢弃决策时依据一定的策略从系统缓冲的分组中选择丢弃的对象。这种控制策略可以最大限度地利用队列缓冲资源,然而却增加了处理时的难度,实时性较差,详细说明将在后文中给出。

### 9.2.3 用户的公平性

从用户的角度来看,缓冲管理方案是一种接纳(或丢失)控制策略。通常情况下,网络资源总是相对有限的,无法完全满足所有用户的需求。这种情况下,不同用户之间网络资源的公平分配成为必需、并且必要的手段。

传统的 Internet 网络采用尾部丢弃的先入先出队列,当一个用户流的数据分组突然增加时,会占据大部分、乃至耗尽全部的缓冲资源,无法保证用户之间的公平性。公平队列(fair queue,FQ)为每个用户流维护一个单独的队列,在用户流之间公平分配所有的带宽资源和缓冲资源,同时也能抑制不符合协议规范的用户流,避免其独占系统资源。

如前所述,公平性的解决依赖于系统 QoS 控制的粒度,只有基于用户流的细粒度控制才能完全解决公平性问题,这同时也带来了控制方案的复杂度和系统的可扩展性问题。

公平性还体现在平稳流量与突发流量之间以及不同长度的分组之间,这同样是需要考虑的问题[23]。

### 9.2.4 与端系统配合——拥塞控制

Internet 网络采用 TCP 协议来实现端到端的拥塞控制,而 TCP 协议的拥塞控制机制对于网络 QoS 控制机制的设计也有着不可忽视的影响。

从端系统的角度考虑,用户无法确切了解网络的运行状态,只能从分组丢弃、往返时间等现象来推测,这使得协议采取的拥塞控制响应颇有些盲目。实践证明,单纯地依靠端到端的控制协议,无法完全解决这个问题,必须有路由器等网络核心设备的介入。传统的 Internet 网络总是在系统资源不足的情况下(拥塞已经发生)丢弃数据分组,这时遭遇分组丢失的所有端系统将认为网络中发生拥塞,相应地降低分组的发送速率,使得网络流量

急剧变小,造成网络带宽资源的浪费。而随后又一致地逐渐增大分组发送速率,将可能导致又一次网络拥塞,这也被称为全局同步效应。从时间跨度来看,网络流量更显突发性,导致网络性能下降。

在缓冲管理和拥塞控制的重叠处,形成了一类主动队列管理方案[23~32]。主动队列管理方案采用明确拥塞指示的拥塞避免策略:由路由器等网络核心交换设备预测拥塞的到来,并在到达的分组上添加标记(或者丢弃该分组),把拥塞信息传递给端系统,在端系统采用的拥塞控制策略的配合下避免或减轻网络拥塞,提高网络系统资源的利用率。

## 9.3　缓冲管理的控制策略

近年来,有关缓冲管理机制的研究一直是一个热点问题,研究领域内也提出了许多基于不同目标设计、采取不同控制策略、适应不同情况的方案。然而,从缓冲管理控制的基本原理出发,其控制策略可以从两个层次分析:

在数据流的层次上,我们援引"流"(flow)的概念——所谓流,指的是具有相同源、目的地址(包括 IP 地址和端口号),并且流经相同网络路径的一组数据分组。不同的流隶属于不同的用户,并且有不同的服务质量要求,归属于不同的服务类别。从系统资源管理的角度来看,缓冲管理机制需要采用一定的资源管理策略,在流经网络传输节点的流之间公平而有效地分配系统中的队列缓冲资源。

而在数据分组层次上,缓冲管理机制需要采用一定的丢弃控制策略,决定在什么情况下丢弃分组,如何选择需要丢弃的分组,这是从分组丢弃控制的角度来看的。在网络传输控制中,有一个基本的假定——相同服务类别的分组的重要性是等同的,不存在哪个分组具有更高的特权或更重要的意义。考虑到端系统的拥塞控制响应,丢弃不同分组的瞬态影响可能相差很大,然而从长期运行的统计结果来看,其间的差距就微乎其微了,绝大多数情况下可以完全忽略不计。

缓冲管理机制的资源管理策略和丢弃控制策略之间有着密切的联系。尽管在一些特定的情况下,如完全共享的资源管理策略可以配合尾部丢弃、头部丢弃等丢弃策略达到不同的有效的控制效果,然而在绝大多数情况下,两者之间相互影响非常大,在设计缓冲管理方案时需要综合考虑,以获得最佳的控制效果。

### 9.3.1　资源管理策略

缓冲管理机制分配的主体是系统中的队列缓冲资源,而分配的对象则包括服务类别、用户以及网络传输节点的物理端口等。为方便起见,在本节中仅以服务类别作为分配对象进行描述和分析。

关于基于端口的缓冲资源管理,这里需要交待一下:由于交换功能的需要,网络传输节点必然支持多个数据传输端口。而在这些数据传输端口之间如何分配系统的缓冲资源,则涉及到系统采用的交换机制。目前主流的交换机制主要有两大类:基于共享存储器和基于交换网络。前者采用惟一的物理存储器缓冲所有到达的数据分组,因其控制机制简单又易于提高资源利用率而得到广泛的青睐;然而存储器共享的同时也限制了网络

传输节点的交换带宽,虽然通过提升存储总线频率以及增加存储总线条数的方法可以缓解一部分压力,但却无法真正解决问题,尤其是对于骨干系统中的高速路由器而言。而后者采用空间换时间的策略,采用每个数据传输端口独立的存储空间,提供基于定长数据段的 $N \times N$ 的交换能力,把交换带宽提高到原来的 $N$ 倍,以其强大的潜力迅速成为骨干路由器的首选;然而这一切也不是没有代价的,为了尽可能高效地利用交换网络的交换能力,需要在交换网络入口设置队列进行调度,增大了系统控制的复杂度和处理开销。只有在共享存储器交换模式下,基于端口的缓冲资源管理的说法才有意义;在交换网络模式下,每个端口可用的缓冲资源完全由硬件条件限制,而不是灵活可控的。这些已经超出了缓冲管理专题的讨论范围,在此不赘述。

就队列缓冲资源管理而言,有共享和划分两种典型策略。总体而言,共享策略的资源利用率高,但是难以保障不同用户的公平性;而划分策略虽然易于提供服务质量保障,却降低了资源的利用率。完全共享和完全划分的控制策略都难以达到满意的效果,由此出现了众多介于完全共享和完全划分策略之间的方案。

假定系统中可用的缓冲资源总量为 $B$,分配给 $K$ 个不同的服务类别使用。为了有效而公平地使用缓冲资源,需要为每个服务类别设定资源使用限额,分别为 $B_i\ (i = 1, \cdots, K)$。调整 $B_i$ 的设定,即得到不同的资源管理策略[5,6]。

1. 完全划分策略

$$\sum_{i=1}^{K} B_i = B \tag{9.3.1}$$

由公式(9.3.1)可知,完全划分策略把系统中所有的队列缓冲资源静态划分给不同的服务类别,不同服务类别的数据分组只能使用预先分配的队列缓冲资源,即使仍然有大量缓冲资源闲置。这种方案将不同的服务类别完全隔离,虽然在最坏情况下也可以保障不同服务类别的服务质量,但同时又会造成大量缓冲资源的浪费,降低了系统效率。

完全划分策略可能使得分配给某些服务类别的缓冲资源一直闲置,而其他服务类别却因为缓冲资源不足而丢弃大量分组,导致系统资源分配的严重失衡,效率低下。这是由于系统资源的静态划分造成的,近年来针对这个问题提出了一些改进方案。

文献[9]中提出的虚拟划分方案(virtual partition,VP)基于虚拟划分的概念[10],其核心思想在于,初始时的缓冲资源划分只是一种名义上的划分,在做分组丢弃控制时不但要依据初始的名义划分,还需要参考当时系统资源的占用情况。

假定初始系统资源划分为 $B_i\ (i = 1, \cdots, K)$,并满足完全划分策略条件。为了简单起见,假定一个服务类别与系统中一个队列相关联,满足一一对应关系,后文如无特别说明均视为同样情况。当队列长度(即该队列对应服务类别所占用的缓冲资源)低于名义划分时,称为轻载(underloaded),否则称为超载(overloaded)。同时定义两个静态的系统资源预留参数 $R_u$ 和 $R_o$,分别对应轻载状态和超载状态,并且满足 $0 \leqslant R_u \leqslant R_o \leqslant B$。当到达分组满足且只满足以下条件之一时即可被接收:

$$\begin{cases} Q_i(t) < B_i, & Q(t) < B - R_u \\ Q_i(t) \geqslant B_i, & Q(t) < B - R_o \end{cases} \tag{9.3.2}$$

虚拟划分方案的思想显而易见：在系统缓冲资源整体占用较少的情况下，即使超过名义划分，依然可以更多地占用系统资源；而当资源整体大部分被占用的时候，即使尚未到达名义分配，也不允许其再占用剩余的缓冲资源。虚拟划分方案在一定程度上提高了划分方案的资源利用率。

动态划分方案（dynamic partition）基于前端流量整形保证到达的数据流量符合规范，并依据流量约定计算需要为每个服务类别 i 预留的缓冲上限 $b_{0,i}$ 和缓冲下限 $b_i$。同时，动态划分方案也提供对 best-effort 数据流的支持，并采取一定的策略保证其和规范数据流之间的公平性。在运行期间，动态划分方案动态调整为每个服务类别设定的缓冲空间上限，见公式（9.3.3），其中 B 为系统缓冲容量，$\gamma$ 为预设的参数，接近于 B，Q(t) 为当前队列长度，$\alpha$ 为一调节因子。

$$\begin{cases} T_i(t) = b_{0,i} + \alpha(\gamma - Q(t)), & \text{if } Q(t) \leqslant \gamma \\ T_i(t) = b_{0,i} - \dfrac{(b_{0,i} - b_i)}{(B - \gamma)}(Q(t) - \gamma), & \text{if } \gamma < Q(t) \leqslant B \end{cases} \tag{9.3.3}$$

当在系统中存在较多缓冲资源空闲时，动态划分方案允许不同服务类别的数据流占用的缓冲资源超过其上限；而当网络流量增大、缓冲资源的占用超过预设的参数 $\gamma$ 时，就开始降低所有服务类别的缓冲空间上限，但最终仍按照缓冲资源下限为其保留相应的缓冲资源。

2. 完全共享策略

$$B_i = B(i = 1, \cdots, K) \tag{9.3.4}$$

由公式（9.3.4）可以看出，完全共享的资源管理策略允许所有数据流使用系统中所有的缓冲资源。传统的 Internet 网络采用完全共享策略，不作服务区分，让所有数据流共享所有的缓冲资源。这样虽然获得了相当高的系统资源利用率，但无法为用户应用提供服务质量支持，同时也无法提供公平性保障。

出于提供服务质量控制和公平性保障的考虑，推出方案在完全共享策略的基础上提供了基于优先级的服务类别区分控制。在推出方案中，只要队列中仍有足够的空闲缓冲资源，就允许到达分组进入队列，并占用该缓冲资源；而当系统中的缓冲资源被完全占用时，只有最低优先级的分组将被无条件丢弃，而具有较高优先级的分组则可以依据一定的策略“推出”（覆盖）队列中的已经缓冲的较低优先级的分组（如果存在），并占用相应的缓冲资源，这也是推出方案名称的由来。低优先级优先（low priority first，LPF）是一种最简单的推出方案——在系统中所有缓冲资源都被占用时，丢弃最低优先级的缓冲分组以满足新到达的高优先级分组的需求。

推出方案本质上是把完全共享策略和选择性丢弃的分组丢弃策略相结合，既可以获得最理想的缓冲资源利用率，同时采用不同的丢弃策略也可以实现不同的控制目标，为不同的服务类别提供区分服务保障。推出方案存在的问题在于，选择性丢弃机制的实现复杂度高，如何合理平衡控制的有效性和实现的难度仍是一个棘手的问题。

3. 部分共享策略

$$\begin{cases} \sum_{i=1}^{K} B_i > B \\ B_i \leqslant B \quad (i = 1, \cdots, K) \end{cases} \tag{9.3.5}$$

部分共享的缓冲资源管理策略介于前两者之间,既允许系统中一部分缓冲资源被所有服务类别的分组共享,又为高优先级服务类别的分组预留一部分资源以满足其突发需求。同时,由于算法实现非常简单,因而也受到了相当广泛的关注。

部分缓冲共享方案(partial buffer sharing, PBS)基于部分共享策略[7],采用单队列结构提供两级区分服务,分组丢弃控制阈值满足 $0 < TH = B_1 < B_2 = B$。当队列长度(此处为被占用缓冲资源的总量)小于控制阈值 TH 时,所有到达的分组都可以进入队列;反之,则只有高优先级分组可以进入队列,低优先级的分组将被丢弃。文献[11]扩展了初始 PBS 方案,提出了基于嵌套的多优先级方案(nested PBS),采取多个控制阈值提供对多个服务类别的传输控制支持。

然而,部分缓冲共享及绝大多数改进方案都采用静态设定的控制阈值,一方面控制阈值的静态设定,类似于完全划分策略,当网络流量变化剧烈时将导致缓冲资源利用率下降,分组丢失率上升;另一方面,如何设置控制阈值也是一个难题。

我们在部分缓冲共享方案的基础上,提出了动态部分缓冲共享方案 DPBS(dynamic PBS),以和前面的 SPBS 相区别。DPBS 方案同样基于缓冲阈值,但是缓冲阈值将依据队列分组的丢弃行为作实时调整,以适应当前的网络流量环境。DPBS 方案能够保证不同优先级分组的丢失比率,并且能够保证不同优先级分组丢失率的时间平稳性。同时,DPBS 方案的初始阈值设定与稳态性能无关,在一定时间内,阈值能够自动调整到合适的位置,这是由方案自身的负反馈机制保证的。

## 9.3.2 分组丢弃策略

从分组的角度来看,缓冲管理也是分组丢弃机制[4]。当数据分组到达队列前端时,缓冲管理方案根据一定的控制信息和当前系统状态决定是否丢弃该分组。

### 1. 尾部丢弃与头部丢弃

尾部丢弃是所有丢弃策略中最简单的一个。尾部丢弃不需要选择丢弃的分组,只是在系统中没有空闲缓冲资源时丢弃到达的分组。尾部丢弃策略不需要保留任何与用户流相关的状态信息,然而尾部丢弃策略无法解决公平性问题,同时也无法避免 TCP 流的全局同步问题。

在缓冲资源被完全占用并且有新的分组到达队列前端时,头部丢弃策略丢弃队列最前端的分组,用以接纳新到达的分组。头部丢弃策略可以保证一定程度的公平性——分组丢弃正比于所消耗的系统带宽,当一个用户流发送越多的分组进入队列时,该用户流分组被选择丢弃的概率也就越大。同时,头部丢弃策略可以提高 TCP 端系统对网络拥塞响应的性能,缩短网络拥塞的时间。

2. 阈值丢弃

上述的尾部丢弃和头部丢弃策略对应着完全共享的缓冲资源分配策略，则基于阈值的丢弃策略对应着完全划分和部分共享的缓冲资源分配策略。基于阈值控制参数的丢弃策略由于其机制简单、易于实现，是影响最广的、被广泛接受和采纳的控制策略。

基于阈值的丢弃策略，无论是静态完全划分，还是静态部分共享，都存在一个严重的问题——资源预留导致的利用率下降。众多的研究算法都偏向于动态调整控制阈值，如何依据当前状态和流量特性有效地调整阈值，同时保证算法的稳定性是这类算法的关键所在。

3. 随机丢弃

传统的基于阈值的丢弃策略，在队列长度的阈值位置上有一个突变——从接收到直接丢弃，从控制理论的角度来看，这将使得队列长度趋于不稳定，甚至产生振荡。文献[23]中提出的 RED 算法首先提出了基于概率的随机丢弃思想，旨在平滑路由器上的分组丢弃行为。文献[28]中提出的 REM 算法更延伸了这种思想，提出了指数平滑的丢弃策略。

4. 非参数控制丢弃

非参数方案模型在优化问题求解、控制器设计等领域都有着非常广泛的应用。然而由于网络处理的实时性要求很高，这些方案的应用都有相当大的难度。文献[16]中提出了一种基于模糊控制思想的丢弃策略。其主体思想和随机丢弃策略非常接近，利用模糊控制的平滑过渡特性平滑分组丢弃行为，同时保持队列长度的稳定性。

这类方案的主要问题在于实现过于复杂，如果没有专用硬件的支持，软件实现的难度和复杂性将远远超出现有网络设备的处理能力。

5. 选择性丢弃

如前所述，绝大多数丢弃策略都基于一个队列操作的简单规则——进入队列的分组将不再被丢弃，而选择性丢弃策略则打破了这种约束。每当新的分组到达时，如果缓冲空间已经被完全占用，则依据一定的策略选择出需要丢弃的分组（包含到达的新分组）。前述的头部丢弃策略正是选择性丢弃策略的一个简单实现。

由于选择性丢弃策略在数据流量足够大的情况下，总能保证所有缓冲资源都被使用，因此可以获得最大的缓冲资源利用率。然而选择性丢弃策略需要一种选择丢弃分组的机制，在支持多个服务等级的情况下，实现将更为复杂。

## 9.4　缓冲管理的典型算法

### 9.4.1　RED 及其衍生算法

早期 IP 网中的拥塞控制机制在 TCP 传输控制协议中实现并由端系统执行，根据发送

分组的丢弃估计网络中可能出现的拥塞,并相应地降低发送速率以减轻网络的负担。然而这样的机制缺乏网络传输节点的配合,存在以下两个问题:

(1) 端系统采取的拥塞控制机制根据分组丢弃来判断是否出现网络拥塞,而此时拥塞已经发生,并已影响了网络的运行效率,因而没有达到避免拥塞的效果。

(2) 早期 IP 网络中网络传输节点采用尾部丢弃策略,在缓冲资源被完全占用时将造成大量分组丢弃。受此影响,大量 TCP 数据源将几乎同时降低自己的发送速率,在短期内造成网络负载过轻,降低了网络资源的利用率;然后,所有 TCP 数据源又将同时逐步增大自己的发送速率,导致下一轮网络拥塞的出现,周而复始地影响网络的运行效率。这被称为 TCP 流的全局同步问题。

因此,把拥塞控制引入到网络传输节点的控制机制中,提高网络资源的利用率成为该研究领域内所关注的话题,Floyd 和 Jacobson 正是由此而提出了影响相当广泛的随机早期检测(random early detection, RED)算法[23]。RED 算法基于平均队列长度预测可能到来的网络拥塞,并采用随机选择的策略对分组进行标记(或丢弃该分组,这是为了与早期 TCP 协议中拥塞控制机制相兼容),在拥塞尚未出现时提示端系统降低其发送速率,以达到避免拥塞的目的。同时,由于 RED 算法随机标记到达的数据分组,使不同 TCP 流的拥塞相应异步化,因而解决了 TCP 流的全局同步问题。

RED 算法不提供服务区分,采用完全共享策略和单队列结构对到达的分组进行排队。分组到达时,RED 算法采用指数加权平均算法计算系统的平均队列长度,作为拥塞预测的依据,并依此计算该分组的标记(或丢弃)概率。平均队列长度的计算公式如下:

$$avg = (1 - w_q) \times avg + w_q \times q \tag{9.4.1}$$

其中 q 为当前队列长度,$w_q$ 为当前队列长度加权系数,满足 $0 < w_q < 1$,avg 为平均队列长度。从公式(9.4.1)中可以看出,RED 算法采用平均队列长度作为判断是否拥塞的依据,平滑了突发流量到来时对算法的影响。在网络流量突然增大的情况下,由于平均队列长度的计算采用指数平均算法,同时 $w_q$ 参数的数值一般设置得较小,使得平均队列的长度变化得很缓慢,不会突然增大而导致大量分组丢弃,提高了系统对于突发流量的适应性。

RED 算法设定两个控制阈值 $min_{th}$ 和 $max_{th}$,当平均队列长度 avg 小于最小阈值 $min_{th}$ 时,所有到达分组都将被允许进入队列,当 avg 超过最大阈值 $max_{th}$ 时,所有到达分组将被直接标记(或丢弃);而当 avg 介于两个控制阈值之间时,将依据一定的概率标记(或丢弃)到达的分组,标记(或丢弃)概率的计算公式如下:

$$P_b = max_p \times (avg - min_{th}) / (max_{th} - min_{th}) \tag{9.4.2}$$

其中 $max_p$ 是预先设置的标记概率,$P_b$ 为当前分组标记概率的计算值,如图 9.4.1 所示。

从公式(9.4.2)中可以看出,RED 算法的标记概率随平均队列长度的变化满足分段线性关系。在算法的实际实现中,为了使被标记的分组散布得更均匀,对标记概率 $P_b$ 作如下修正以得到 $P_a$ 作为实际标记概率:

$$P_a = P_b / (1 - count \times P_b) \tag{9.4.3}$$

为了简化考虑,假定 $P_b$ 在一段时间内保持为常数,且 $1/P_b$ 为一整数。假定 X 为两次分组

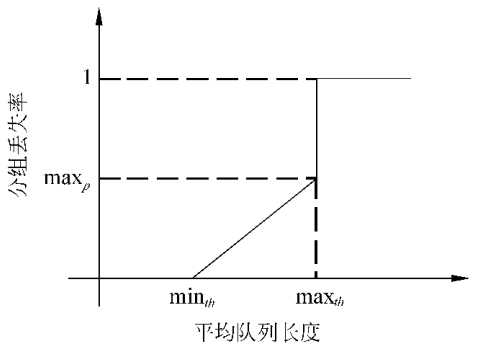



图 9.4.1　RED 算法丢弃概率分布

丢弃之间到达的分组个数，则应满足公式(9.4.4)。易见，经过优化处理后的分组标记概率使得两次分组丢弃之间到达的分组个数满足平均分布，有效地平滑了分组的标记过程。

$$\begin{aligned}\text{Prob}[X = n] &= \frac{P_b}{1 - (n-1) \times P_b} \times \prod_{i=1}^{n-1}\left(1 - \frac{P_b}{1 - (i-1) \times P_b}\right) \\ &= \begin{cases} P_b, & 1 \leqslant n \leqslant 1/P_b \\ 0, & n > 1/P_b \end{cases}\end{aligned} \tag{9.4.4}$$

RED 算法的优点包括：

(1) 设计基于拥塞避免的思路，在网络尚未发生拥塞时提示端系统减小发送速率以避免网络拥塞的出现。同时，RED 算法不是采用源抑制策略，立刻把反馈信息返回给发送端，而是通过设置标志位提示接收端，再由接收端传递给发送端。这样，对于网络传输节点而言，至少需要经历一次往返时间才能看到到达速率的降低。这使得 RED 算法判断拥塞的时间尺度和端系统响应拥塞指示的尺度大致吻合。

(2) 采用随机早期标记的策略处理到达的分组，使得不同端系统对拥塞指示的响应更加分散，有效地避免了 TCP 流全局同步现象的出现，提高了网络资源的利用率。

(3) 有效地控制平均队列长度，使得系统吞吐率和分组时延达到很好的平衡，获得优化的 Power① 性能。

(4) 虽然 RED 算法没有基于服务类别或用户流来设计，但是其随机选择标记的机制使得发送速率越大的用户流得到标记的概率也越大——与占用的带宽成正比。这样，如果需要的话，在网络发生拥塞时，可以根据标记分组的计数情况确定占用大量系统带宽的用户流，并采取一定的附加机制来保障系统的公平性。

(5) RED 算法基于简单的控制机制，如果采取合适的参数设置，平均队列长度和标记概率的计算都可以转化为加法和移位操作完成，易于实现，因此也得到了业界的广泛认可和支持。

RED 算法存在的问题有：

---

① 用于衡量系统综合性能的参数，具体内容参见第 12 章性能评价部分内容，$\text{Power} = \frac{\text{Throughput}}{\text{Delay}}$。

(1) 对控制参数过于敏感,难以优化参数设定。算法的性能对控制参数和网络流量负载的变化非常敏感。在用户流增大的情况下,RED 算法的性能会急剧下降。

(2) 不支持服务区分,基于 best-effort 服务模型,没有考虑不同等级服务之间、不同用户流之间的差别,无法提供有效的公平性保障。

针对 RED 算法存在的问题,研究领域和业界提出了一系列改进算法,包括稳定 RED 算法(SRED)、自适应 RED 算法(adaptive-RED)、带 In/Out 标记的 RED 算法(RED with In/Out bit, RIO)、随机指数标记算法(random early marking, REM)、比例积分控制器算法(proportional integral controller)等[24~27, 29~31],下面仅选择几个有代表性的作简要介绍。

1. 自适应 RED 算法

RED 算法通过有效地控制系统的平均队列长度,可以同时获得较高的系统吞吐量性能和较低的分组排队延迟。然而在 RED 算法中,平均队列长度受网络拥塞程度和控制参数的设置的影响非常巨大。如果网络流量负载较轻,或者最大标记概率参数 $max_p$ 设置得较大,则平均队列长度将会趋于最小阈值 $min_{th}$;而如果网络负载较重或者 $max_p$ 值设置得较大,则会导致平均队列长度趋于甚至超过最大阈值 $max_{th}$。这就破坏了平均队列长度的稳定性,进而使得系统排队造成的延迟无法预先估测,同时也降低了系统的有效吞吐量。

出于提高 RED 算法性能稳定性的考虑,文献[25, 26]提出并改进了一种自适应调整控制参数的自适应 RED(adaptive-RED)算法。自适应 RED 算法的思路非常简单。为了控制平均队列长度稳定地保持在最小、最大阈值之间,当网络拥塞程度较大时,相应地增大标记概率 $max_p$,而当网络拥塞程度较小时,则减小标记概率的数值,以达到保持平均队列长度稳定的目的。文献[25]提出的改进算法如公式(9.4.5)所示:

$$
\left.\begin{array}{l}
\text{每次分组到达时,} \\
\text{if } (min_{th} < Q(ave) < max_{th}) \\
\quad \text{status} = \text{Between}; \\
\text{if } Q(avg) < min_{th} \text{ and status} \neq \text{Below} \\
\quad \text{status} = \text{Below}; \\
\quad max_p = max_p / \alpha, (\alpha > 1) \\
\text{if } Q(avg) > max_{th} \text{ and status} \neq \text{Above} \\
\quad \text{status} = \text{Above}; \\
\quad max_p = max_p * \beta, (\beta > 1)
\end{array}\right\} \qquad (9.4.5)
$$

在此基础上,文献[26]更严格地限定了平均队列长度的允许变化范围和标记概率参数的调整范围,同时把标记概率参数的调整算法由乘法增大乘法减小(MIMD)改为加法增大乘法减小(AIMD)。

自适应 RED 算法把 RED 算法的控制参数动态化,提高了 RED 算法的鲁棒性,使之更能适应网络流量的变化,获得更加稳定的性能。然而另一方面,自适应算法也增加了处

理的复杂度，同时引入的调整参数的设置也是在实现中需要考虑的问题。

2. RED with I/O 和 WRED

为了提供更有效的公平保障，满足用户服务区分的需要，文献[28]提出了基于 In/Out 标记的 RIO(RED with In/Out bit)算法。RIO 算法实际上是两个 RED 算法的重叠，其中一个用于符合规范的 TCP 流(标记为 In 的分组)，另一个用于不符合规范的 TCP 流(标记为 Out 的分组)。在适当的控制参数设置下，RIO 算法可以对两种 TCP 流区别对待，以保障符合规范的 TCP 流的优先权。RIO 算法类似于 RED 算法，易于实现；然而如何优化设定两组控制参数以获得期望的性能区分依然是需要解决的问题。

RIO 算法需要在前端实现分组分类，对到达的分组作相应的“In”和“Out”标记。具体来说，需要由两部分机制来实现：①速率估测，计算 TCP 流在最近一段时间内的发送数据速率；②标记算法，在检查 TCP 流的发送速率超过约定值时将其标记为“Out”。RIO 算法的设计体现了网络核心与边缘相分离的趋势，把与用户流相关的需要大量信息的操作放在网络边缘，而网络核心只提供尽量简单的高速传输服务。

Cisco 公司在其 7000 系列和 7500 系列路由器产品中采用的 WRED 算法是对 RIO 算法的进一步扩展，实现了更灵活的服务优先级控制[29]。WRED 算法最多支持 8 个优先级，每个优先级都有一组独立的控制参数，可以灵活配置以达到预期的控制目标。

## 9.4.2　AVQ 算法

自适应虚拟队列(adaptive virtual queuing, AVQ)算法属于另一类主动队列管理算法，与 RED 算法依据(平均)队列长度判断系统的拥塞状态不同，AVQ 算法的判断依据是当前系统的负载情况。同时，AVQ 算法认为直接对队列长度进行控制是导致算法鲁棒性较差的原因，在系统中 TCP 流数量增大的情况下尤为严重，因此 AVQ 算法把输出链路的带宽利用率作为首要的控制目标[32]。

AVQ 算法基于 Gibbens 和 Kelly 提出的虚拟队列的概念，虚拟队列的虚拟带宽小于实际队列对应的输出链路带宽，而其缓冲容量和实际队列的一致。当分组到达时，如果在虚拟队列中还有足够的缓冲空间，就允许其进入队列；而如果该分组将导致虚拟队列溢出，则在虚拟队列中将该分组丢弃，并在实际队列中标记(或丢弃)该分组(取决于系统的队列管理策略)。虚拟队列的带宽也取决于系统策略——是保持高吞吐量还是保持短队列长度，以获得较低的分组延迟，如公式(9.4.6)所示：

$$\tilde{C} = \gamma \times C \tag{9.4.6}$$

其中 $\tilde{C}$ 为虚拟队列带宽，C 为实际队列带宽，$\gamma$ 为系统期望的带宽利用率。AVQ 算法同时在分组到达的过程中，不断修正虚拟队列带宽参数，以获得更高的系统效率，修正算法如公式(9.4.7)所示，其中 $\alpha$ 为衰减因子，$\lambda$ 为分组到达速率：

$$\Delta\tilde{C} = \alpha(\gamma C - \lambda) \tag{9.4.7}$$

AVQ 算法的具体操作如公式(9.4.8)所示。其中 VQ 为虚拟队列长度，t，s 分别为当前分组和前一个分组的到达时间，b 为当前分组长度，B 为系统缓冲容量。

每次分组到达时，

$$
\begin{aligned}
&VQ = \max(VQ - \tilde{C}(t - s), 0)\\
&\text{if } VQ + b > B\\
&\quad \text{标记(或丢弃)该分组}\\
&\text{else}\\
&\quad \text{允许分组进入队列，} VQ = VQ + b\\
&\text{endif}\\
&\tilde{C} = \min(\max(\tilde{C} + \alpha * \gamma * C(t - s), C) - \alpha * b, 0)\\
&s = t
\end{aligned}
\tag{9.4.8}
$$

AVQ 算法通过自适应参数调整来控制输出链路的带宽利用率，在保证高的系统吞吐量的同时，也可以获得相当有效的队列长度控制，进而获得更短的排队延迟。

图 9.4.2 和图 9.4.3 是文献[32]中提供的一组性能数据，从图中可以明显看出 AVQ 算法的带宽利用率很高，而队列长度也低于其他 AQM 算法。

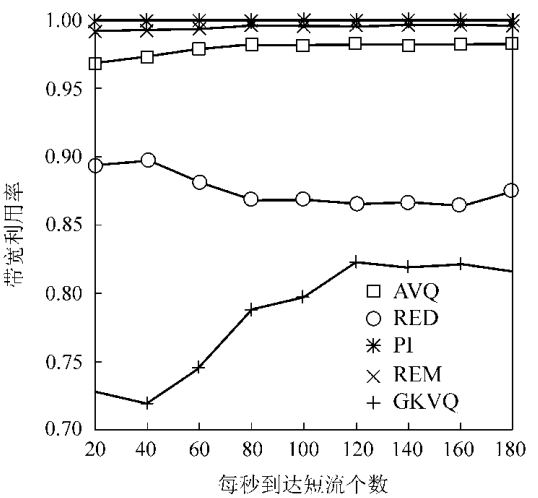



图 9.4.2 AVQ 算法和其他算法带宽利用率比较

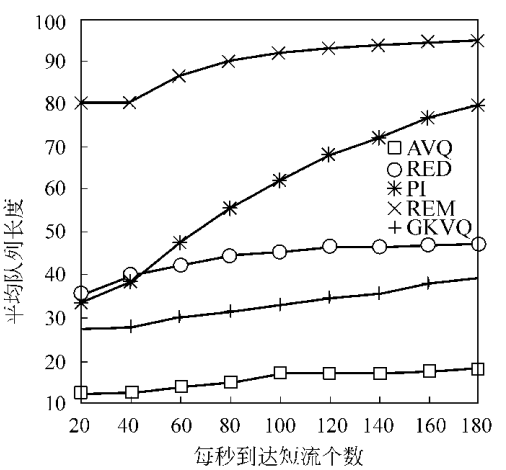



图 9.4.3 AVQ 算法和其他算法队列长度比较

在 AVQ 算法中有两个关键的控制参数：$\gamma$ 是系统设定的带宽利用率的期望值，允许实现时在系统吞吐量和排队延迟之间做出一定的权衡，同时也影响着系统的鲁棒性，尤其是在系统中存在大量维持时间很短的 TCP 流的情况下；$\alpha$ 是一个衰减控制因子。这两个控制参数都对算法的稳定性有着相当大的影响。而在参数设置合适的情况下，AVQ 算法能够适应 TCP 流反馈延迟、大量短期 TCP 流并存的负面影响，保持算法的稳定性和鲁棒性。

### 9.4.3　动态阈值算法

前面介绍的 RED 和 AVQ 两种算法都是从拥塞控制和主动队列管理的角度出发的，力求获得最佳的系统吞吐量和排队延迟的平衡，而系统缓冲资源的利用率则放在其次。同时，这两种算法都没有把服务区分作为重点来考虑，使之难以提供有效的公平性保障。下面将要介绍的动态阈值算法（dynamic threshold，DT）则是完全从系统资源分配的角度来设计的，力求最大化缓冲资源利用率，同时提供不同类别的服务区分[12, 13]。

DT 算法基于共享存储器模式的分组交换方式，首先提出了在多个物理端口之间公平分配缓冲资源的方案，如公式（9.4.9）所示。其中 B 为系统缓冲空间的容量，$Q^i(t)$ 为端口 i 的队列长度，$Q(t)$ 为当前系统总队列长度，即为总的缓冲空间占用量，$T(t)$ 为控制分组丢弃时采用的阈值参数，$\alpha$ 为一调节因子。从公式可以看出，DT 算法根据系统状态动态调整控制阈值，阈值的大小与当前系统中空闲缓冲资源成正比，当端口占据的缓冲空间超过控制阈值时，将阻塞该端口，不允许其接收更多的到达分组。

$$\begin{cases} T(t) = \alpha(B - Q(t)) = \alpha\left(B - \sum_i Q^i(t)\right) \\ Q^i(t) \leqslant T(t) \text{ 时允许接收该分组} \end{cases} \tag{9.4.9}$$

假定取 $\alpha$ 值为 2，则当系统中仅有一个端口工作时，其可用的缓冲资源为系统资源的 1/3，而随着系统负载的增大，当有 4 个端口同时工作时，每个端口最大可用空间为系统资源的 2/9。DT 算法总是预留一部分缓冲资源，虽然造成了一定的资源浪费，但是对于调整系统的瞬态性能提供了有效的支持。

为了适应服务区分的需求，DT 算法在物理端口的基础上，引入了对不同优先级的服务的支持。DT 算法为每一个优先级设置一个不同的调节因子 $\alpha_P$，算法如公式（9.4.10）所示。$Q_P^i(t)$ 为端口 i 优先级 P 对应的队列长度，$T_P(t)$ 为优先级 P 对应的控制阈值。

$$\begin{cases} T_P(t) = \alpha_P(B - Q(t)) = \alpha_P\left(B - \sum_i \sum_P Q_P^i(t)\right) \\ Q_P^i(t) \leqslant T_P(t) \text{ 时允许接收该分组} \end{cases} \tag{9.4.10}$$

DT 算法为低优先级分组提供了一定的缓冲资源，使得其总能获得一定的服务，而且不同优先级之间的缓冲资源配比可以通过对 $\alpha$ 因子的调节来实现。只要 $\alpha$ 因子的值满足一定的条件，就能保证系统性能的稳定性。

DT 算法通过动态控制参数的调整，能够在一定程度上适应网络流量的变化，同时通

过预留缓冲资源来保证不同优先级服务之间的公平性。从文献[13]给出的性能数据图9.4.4中可以看出,DT动态阈值方案在丢失率性能上优于静态阈值方案,比采用选择性丢弃策略的pushout方案稍差,然而DT算法的复杂性却要比pushout方案小很多。

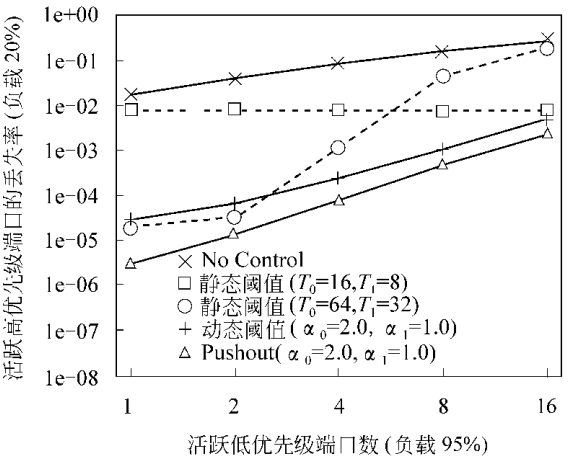



图9.4.4 动态阈值方案与静态阈值、Pushout方案的丢失率性能比较

## 9.4.4 成比例丢失率控制算法

成比例丢失率控制算法(proportional loss ratio, PLR)是基于成比例区分服务模型提出来的[17, 18],目的是通过队列管理算法在不同等级的服务类别之间保证稳定的相对丢失率,即满足如下关系:

$$\frac{\bar{l}_i}{\bar{l}_j} = \frac{\sigma_i}{\sigma_j} \quad (1 \leqslant i,j \leqslant N) \tag{9.4.11}$$

其中$\bar{l}_i$为第i类服务的平均分组丢失率,而$\sigma_i$则为相应的控制因子。考虑到服务等级随着序号的增加而提高,应该满足关系$\sigma_1 > \sigma_2 > \cdots > \sigma_N > 0$。

PLR算法有两种形式:PLR( )和PLR(M),两者在统计分组丢失率的时间区间上有所不同——前者统计整个算法运行期间内不同服务类别的丢失率,而后者只关心最近一段时间的情况。PLR算法采取选择性丢弃策略,为每个服务类别保存丢失率的估计值$l_i$,在到达分组将使得缓冲空间溢出的时候,选择使比率$l_i/\sigma_i$最小的类别(如果在队列中该类没有分组,则选择第二小的类别,依此类推)并丢弃其中的一个分组。由于PLR算法总是尽可能丢弃加权丢失率最小类别的分组,在不同服务类别竞争网络资源的时候,PLR算法总能在不同服务类别之间保持基本一致的加权丢失率,即按照相对比率丢弃分组。

PLR( )的算法描述如公式(9.4.12)所示,PLR(M)则需要额外保存最近到达的M个分组的状态信息作为计算丢失率的依据,算法的时间和空间复杂度都略大于PLR( )算法。

$$
\left.\begin{array}{l}
\text{第 i 类服务的分组到达} \\
\quad \text{更新到达分组计数 } A_i + + ; \\
\quad \text{if qlen} < B \\
\quad \text{允许该分组进入队列} \\
\quad \text{else} \\
\quad \text{选择服务类别 j，满足 } j = \arg\min_{i \in [1,N]} \dfrac{D_i}{A_i \sigma_i} ; \\
\quad\quad \text{丢弃该类别的一个分组，更新丢弃分组计数 } D_j + + 。
\end{array}\right\} \quad (9.4.12)
$$

由于采取 Pushout 的选择性丢弃策略，PLR 算法可以获得极佳的系统缓冲资源利用率，同时，PLR 算法又能够保证不同服务类别之间分组丢失率的公平性，提供有效的服务区分。另一方面，PLR 算法的计算复杂性相当大，每完成一次分组丢弃操作，需要做 N 次浮点乘运算（$A_i\sigma_i$）、N 次浮点除运算（$\dfrac{D_i}{A_i\sigma_i}$）和比较操作，尤其在服务类别较多的情况下，如何提高实现性能是一个很关键的问题。

### 9.4.5　缓冲管理和调度联合算法

缓冲管理和调度联合算法（joint of buffer management and scheduling，JoBS）把队列管理的两个核心机制——缓冲管理算法和分组调度算法相结合，体现了队列管理方面研究的新思路[19]。传统的缓冲管理算法主要关注系统缓冲资源的分配，考察参数主要是分组丢失率、队列长度对分组延迟的影响等，而带宽（速率）的分配则交给分组调度算法来完成。然而从整个队列管理的角度来看，缓冲管理和分组调度算法是相辅相成的，两者之间有着内在的联系，相互配合才能获得优化的控制效果。

JoBS 算法以用户的 QoS 需求为基本出发点，将其作为约束条件，来寻求队列操作的优化方案。JoBS 算法考虑的 QoS 约束包括：

（1）绝对丢失率约束（absolute loss ratio constraint，ALC）；

（2）相对丢失率约束（relative loss ratio constraint，RLC）；

（3）绝对延迟约束（absolute delay constraint，ADC）；

（4）相对延迟约束（relative delay constraint，RDC）。

在尽量满足这些 QoS 约束条件的基础上，JoBS 算法的优化目标包括：

（1）尽量保持静态的带宽（速率）分配；

（2）尽量保持分组不丢弃。

JoBS 方案采取的修正算法流程如图 9.4.5 所示：当到达分组使系统缓冲溢出时，依据丢失率约束选择一个服务类别，从其缓冲的分组中选择一个丢弃，并开始检查是否违反了延迟约束 如果一直没有分组被丢弃，则在每到达 N 个分组之后作一次延迟约束检查。对延迟约束的检查从绝对延迟约束开始，如果被违反，则通过调整带宽分配和丢弃部分分组来调节 ;其次检查相对延迟约束是否被违反，如果有通过带宽分配的调节就可以解决。

JoBS 算法可以提供绝对和相对的 QoS（延迟和丢失率）保证，并且性能与到达的数据流量特性无关。JoBS 算法把速率分配引入到缓冲管理算法的设计中，增强了与调度算法

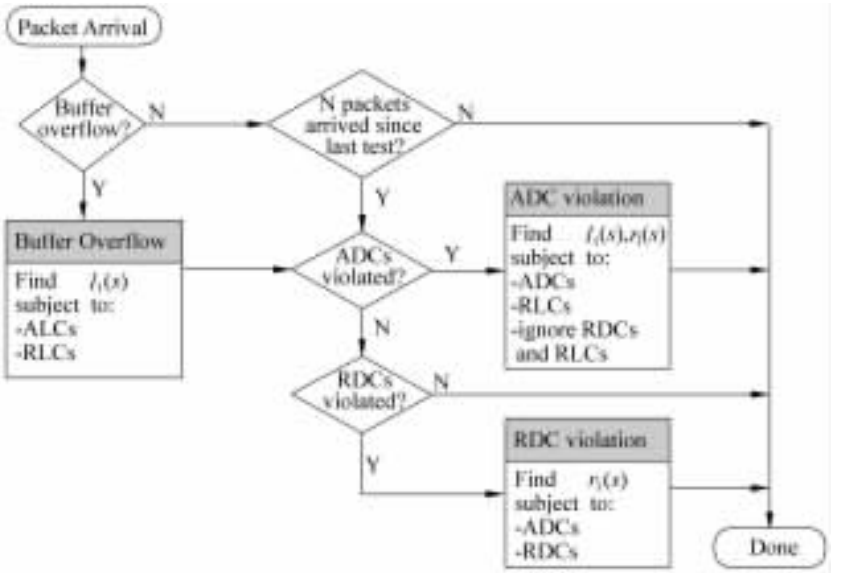



图 9.4.5 JoBS 方案修正算法流程图

的配合,进一步提高了整个系统的运行效率。

## 9.4.6 动态部分缓冲共享算法

动态部分缓冲共享(dynamic partial buffer sharing, DPBS)算法[33]在基于静态阈值的 PBS 算法的基础上将阈值动态化,更能适应网络流量的变化,提高缓冲资源的利用率,也解决了 PBS 算法的参数难以优化设定的问题;同时,在对流量统计特性已知的情况下,可以提供对不同服务类别的区分服务,保证它们之间可确定的稳定的分组丢失率之比。

假定系统的队列缓冲空间最多可以容纳 N 个分组,同时假定所有到达分组都属于 D 个分组丢弃优先级,则相应设定 D 个丢弃阈值 $TH_d$ ($1 \leqslant d \leqslant D$),并且满足关系 $0 \leqslant TH_1 \leqslant \cdots \leqslant TH_{D-1} \leqslant TH_D = N$。优先级为 d 的到达分组,当且仅当其进入队列后队列长度不超过相应丢弃阈值 $TH_d$ 时,即满足条件 $qlen < TH_d$ 时才可以进入队列,否则将被丢弃。而一旦分组进入队列后,将不会被后续到达的分组推出队列。如图 9.4.6 所示,为多优先级情况下的 DPBS 方案队列模型。

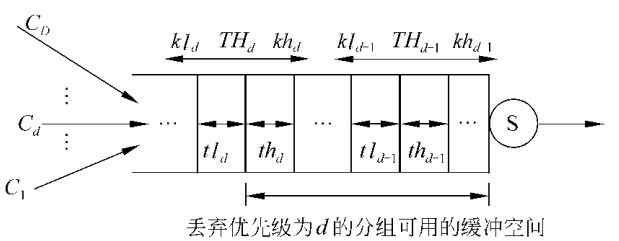



图 9.4.6 DPBS 方案队列模型

为了简化起见,假定只有两个分组丢弃优先级,丢弃阈值 TH 的初始值设置在 0 和 N

之间(图中,0 的位置在靠近服务器 S 的队列右端)。只要队列中还有足够大的容纳空间,高优先级的分组就可以进入队列;而如果队列长度超过丢弃阈值,则到达的低优先级分组将被丢弃。

为了实现丢弃阈值的动态调整,每个优先级对应一个丢弃计数,用来记录该优先级丢弃分组的个数,丢弃计数的累计将会推动阈值逐渐调整。以高优先级为例,当丢弃记数超过相应的丢弃上限 kh 时,算法将控制 TH 减小相应的修正步长 th。反之,对于低优先级也有类似的机制。当丢弃记数超过相应的丢弃上限 kl 时,算法将控制 TH 增大相应的修正步长 tl。在多个分组丢弃优先级的情况下,算法也非常类似,只是除了最低和最高优先级以外,所有优先级都需要两组丢弃上限和修正步长参数,分别对应其上端阈值和下端阈值(以优先级 d 为例,分别为 $TH_d$ 和 $TH_{d-1}$)的动态调整。

DPBS 算法中采用的动态阈值调整的基本思想在于,数据流的到达在短时间内是相对平稳的,根据当前的丢失量,在一定程度上可以预测下一阶段的数据流量。如果一段时间内某优先级的分组丢失量较大,则相应的下一时间段内该优先级到达流量大的概率也较高,反之亦然。同时,如果高优先级的分组丢弃过多,则应该减小丢弃阈值以压缩低优先级可用的缓冲空间,而如果低优先级的分组丢弃过多,则应该增大丢弃阈值,增大低优先级分组的可用缓冲空间,保证相对丢失率维持在期望值附近。当系统进入稳定状态时,丢弃阈值将会依据丢弃行为不断调整,达到动态平衡。

丢弃阈值 TH 的调整速度是由设定的控制参数 kh,kl,th,tl 决定的。随着 kh 和 kl 的增大,阈值的调整速度会越来越慢,当增大到一定程度时,将不再出现丢弃阈值的变化,这时 DPBS 方案退化成 PBS 方案,因此可以说 PBS 方案是 DPBS 方案的一个特殊情形。

在 DPBS 算法执行过程中,不断的分组丢弃逐渐积累影响着丢弃阈值的调整,如式(9.4.13)所示,其中 $v_{inc}$ 为丢弃阈值增大的速度,$P_1$ 为低优先级分组丢失率,$\lambda_1$ 为低优先级分组的平均到达速率;而 $v_{dec}$ 为丢弃阈值减小的速度,$P_2$ 为高优先级的分组丢失率,$\lambda_2$ 为高优先级分组的平均到达速率。

$$\begin{cases} v_{inc} = P_1\lambda_1(tl/kl) \\ v_{dec} = P_2\lambda_2(th/kh) \end{cases} \tag{9.4.13}$$

通常情况下,到达稳态时丢弃阈值不会停留在队列的两端(即 $TH \neq 0$,$TH \neq N$)。这样,阈值的增大、减小必然满足动态平衡的条件。再考虑极限情况,当 $TH = N$ 时,低优先级分组的可用缓冲空间是整个队列空间,和高优先级分组享有相同的服务等级,这时 $P_2/P_1$ 取最大值 1;而当 $TH = 0$ 时,所有到达的低优先级分组都将被丢弃,丢失率 $P_1$ 为 1,这时 $P_2/P_1$ 取最小值 $P_2$。这样必然满足式(9.4.14)的描述:

$$P_2 < P_2/P_1 = \frac{kh \times \lambda_1 \times tl}{kl \times \lambda_2 \times th} < 1 \tag{9.4.14}$$

上式中,$\lambda_i$($i = 1,2$)是到达数据流的特性,与控制算法本身无关,因此在已知到达数据流统计特性的情况下,可以通过 4 个控制参数的选取,来获得期望的相对丢失率 $P_2/P_1$。需要注意的是,两个优先级分组的整体丢失率受到队列容量和其他条件的影响,如果设定的相对丢失率期望值太低,即使低优先级分组全部丢失也无法满足要求,这时阈值 TH 将近似稳定在队列右端($TH \equiv 0$)。这将导致算法的调整机制失效,应当在具体实现中避免

出现。对于多优先级的 DPBS 方案,也有完全类似的结果,这里不再赘述。

## 9.5 缓冲管理的研究方向

### 9.5.1 基于流量预测提高系统资源利用率

为了提供不同等级的区分服务,几乎所有的缓冲管理算法都采用不同形式的资源预留,为高优先级的服务提供性能保证。资源预留策略带来的结果是缓冲资源利用率的下降。为了提高资源利用率,动态资源预留是惟一可行的出路,然而如何动态、如何调整相关的控制参数,则是算法设计的关键。

一个理想的缓冲管理算法应该是根据即将到来的数据流量最优地调整资源分配,在满足服务质量要求的同时,最大限度地利用系统的缓冲资源。然而在实际情况下,下一时间段内的流量到达情况是无法确切获得的,而采用流量预测可以在一定程度上预知到达流量的特性,其精度依赖于采集的信息和计算的复杂度。

大量研究结果表明,传统的认为网络流量服从 Poisson 分布的观点是不确切的,实际的网络流量具有更强的突发性和相关性,服从自相似分布。与 Poisson 分布不同,自相似分布是一种长相关分布,即下一时间段内流量到达的情况是与历史情况相关的,这论证了流量预测的可行性。

多时间段的加权平均是流量预测的一种经典方法,即记录最近 N 个时间段内的流量到达情况,N 决定了预测精度和算法的空间复杂度(存储信息量),同时,时间段大小的选择也对预测结果有一定的影响,假定 $H_i(1 \leqslant i \leqslant N)$ 为历史流量到达情况,$\lambda_i(1 \leqslant i \leqslant N)$ 为归一化权重系数,则下一段时间内的数据流量预测为

$$E = \sum (H_i \times \lambda_i) \left( \sum \lambda_i = 1, 0 \leqslant \lambda_i \leqslant 1 \right) \tag{9.5.1}$$

权重系数 $\lambda_i(1 \leqslant i \leqslant N)$ 同样对预测的准确性有着相当大的影响,需要采用一定的算法来优化。

目前,多数算法采用指数平均(exponential estimation)的方法来预测网络流量的到达,即采用如下公式:

$$\begin{aligned} E(n+1) &= \lambda \times R(n) + (1-\lambda) \times E(n) \\ &= \lambda \times R(n) + \lambda(1-\lambda) \times R(n-1) + (1-\lambda)^2 E(n-1) \\ &= \lambda \times R(n) + \lambda(1-\lambda) \times R(n-1) + \cdots + (1-\lambda)^n E(1) \\ &= \lambda \times R(n) + \lambda(1-\lambda) \times R(n-1) + \cdots + (1-\lambda)^n R(0) \end{aligned} \tag{9.5.2}$$

指数平均预测算法是一个迭代算法,利用上一次预测结果和当前实际流量情况来估算下一时段内的流量特性。指数平均预测算法也是一种多时段加权平均算法,如式(9.5.2)所示,各项系数之和也是归一化的,同时,由于加权系数的影响,越早的历史情况对预测结果的影响也越小。指数预测算法的优点在于,实现的复杂度非常小,而且在流量变化频率不是很高的情况下的性能较好。在文献[34]中的分析报告指出,指数预测算法不能反映网络流量变化具有很强趋势性的特征,并引入了增量预测算法(delta estimation),然而增量预测导致对流量的细微变化非常敏感,这会造成预测结果的剧烈抖

动,如式(9.5.3)所示,$\Delta(n)$和$\Delta_E(n)$分别为时段n的实际流量增量和预测流量增量:

$$\begin{aligned}\Delta_E(n+1) &= \lambda \times \Delta(n) + (1-\lambda) \times \Delta_E(n)\\ E(n+1) &= R(n) + \Delta_E(n+1)\end{aligned} \tag{9.5.3}$$

提高系统缓冲资源利用率的另一种选择是在可能的情况下完全使用所有缓冲资源,并采用选择性丢弃策略为不同服务等级提供性能保证(主要是分组丢失率)。然而这种策略需要为不同服务等级保存大量的状态信息,同时丢弃分组(服务等级)的选择也带来相当大的计算复杂度,可扩展性差。

### 9.5.2 与分组调度相结合融入带宽分配

目前,在绝大多数队列管理方案中,缓冲管理算法和分组调度算法缺乏一定的联系,导致缓冲资源和带宽资源的分配相互独立,协作性差。显而易见,只有缓冲资源和带宽资源的分配相互一致,才能最大程度地利用缓冲队列的特性,使得系统吞吐量和分组丢失率的综合性能最优。因此,把缓冲管理和分组调度结合起来考虑,在缓冲管理算法中引入带宽(速率)分配是一个很有价值的研究方向。

带宽的分配基于不同类别(用户流或不同服务等级)的服务质量要求和到达的流量。通常服务质量要求是静态的(或者是基于某种策略的,变化极慢,相对于到达流量可以认为是静态的),而到达流量特性则是不断变化的。为了优化带宽分配——即在保证服务质量的同时提高链路利用率,同样可以采用流量预测算法。

### 9.5.3 队列长度的控制与维护

缓冲管理算法同样关注分组的延迟特性。在路由器的分组处理流程中,排队延迟是影响最大的,同时也是惟一可以控制的。在单队列情况下,分组排队延迟只与队列长度相关(考虑输出链路保持稳定的速率);在多队列情况下,分组排队延迟还与队列输出端的调度策略相关。

无论在哪一种情况下,队列长度对分组延迟都有着相当大的影响。在系统吞吐量和分组排队延迟的综合考虑下,需要选择合适的队列长度提供综合性能。同时,队列长度的稳定性也影响着分组延迟抖动性能。

## 参考文献


1 Parekh A K, Gallager R G. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. IEEE/ACM Trans Networking, 1993,1(3)

2 Floyd S, Jacobson V. Link-sharing and resource management models for packet networks. IEEE/ACM Trans Networking, 1995,3(4)

3 Massoulie L, Roberts J. Bandwidth sharing: Objectives and algorithms. IEEE/ACM Trans Networking, 2002,10(3)

4 Labrador M A, Banerjee S. Packet dropping policies for ATM and IP networks. IEEE Communications Surveys,1999,2(3)

5 Kroner H, Hebuterne G, Boyer P, Gravey A. Priority management in ATM switching nodes. IEEE Journal

on Selected Areas in Communications, 1991,9(3)
6 Causey J W, Kim H S. Comparison of buffer allocation schemes in ATM switches: Complete sharing, partial sharing, and dedicated allocation. International Conference on Communications, Vol. 2, May 1994
7 Kang C G, Tan H H. Queuing analysis of explicit priority assignment partial buffer sharing schemes for ATM networks. In: IEEE INFOCOM '93. March 1993
8 Kang C G, Tan H H. Queuing analysis of explicit priority assignment push-out buffer sharing schemes for ATM networks. In: IEEE INFOCOM '94. June 1994
9 Kumaran K, Mitra D. Performance and fluid simulations of a novel shared buffer management scheme. In: IEEE INFOCOM '98. Vol. 3, San Francisco, CA, March 1998
10 Mitra D, Ziedins I. Virtual partitioning by dynamic priorities: Fair and efficient resource-sharing by several services. Broadband Communications, Proc International Zurich Seminar on Digital Communications. Springer-Verlag, 1996
11 Krishnan S, Choudhury A K, Chiussi F M. Dynamic partitioning: A mechanism for shared memory management. In: IEEE INFOCOM '99. Vol. 1, March 1999, 144 ~ 152
12 Choudhury A K, Hahne E L. Dynamic queue length thresholds for shared-memory packet switches. IEEE/ACM Trans Networking, 1998,6: 130 ~ 140
13 Hahne E L, Choudhury A K. Dynamic queue length thresholds for multiple loss priorities. IEEE/ACM Trans Networking, 2002,10(3)
14 Petr D, Frost V. Nested threshold packet discarding for ATM overload control: Optimization under packet loss constraints. In: IEEE INFOCOM '91. April 1991
15 Lee H W, Ahn B Y. Queuing model for optimal control of partial buffer sharing in ATM. Computers & Operations Research, 1998,25(2)
16 Ascia G, Catania V, Panno D. An efficient buffer management policy based on an integrated fuzzy-ga approach. In: IEEE INFOCOM 2002. June 2002
17 Dovrolis C, Stiliadis D, Ramanathan P. Proportional differentiated services: Delay differentation and packet scheduling. In: ACM SIGCOMM '99. Cambridge, MA, September 1999
18 Dovrolis C, Ramanathan P. Proportional differentiated services, Part Ⅱ: Loss rate differentiation and packet dropping. In: Proc IWQoS 2000. Pittsburgh, PA., June 2000. 52 ~ 61
19 Liebeherr J, Christin N. JoBS: Joint buffer management and scheduling for differentiated services. In: Proc IWQoS 2001. Karlsruhe, Germany, June 2001. 404 ~ 418
20 Liebeherr J, Christin N. Rate allocation and buffer management for differentiated services. Computer Networks, 2002,40(1)
21 Nabeshima M. Improving the performance of active buffer management with per-flow information. IEEE Communications Letters, 2002,6(7)
22 Cho J-W, Cho D-H. Dynamic buffer management scheme based on rate estimation in packet-switched networks. Computer Networks, 2002,39(6)
23 Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking, 1993,1(4)
24 Ott T J, Lakshman T V, Wong L H. SRED: Stabilized RED. In: IEEE INFOCOM '99. March 1999
25 Feng W, Kandlur D, Saha D, Shin K. A self configuring RED gateway. In: IEEE INFOCOM '99. March 1999
26 Floyd S, Gummadi R, Shenker S. Adaptive RED: An algorithm for increasing the robustness of RED's

active queue management. ACIRI Technical Report, 2001

27 Feng W, Kandlur D, Saha D, Shin K. BLUE: A new class of active queue management algorithms. U. Michigan CSE-TR-387-99, April 1999

28 Clark D D, Fang W, Explicit allocation of best effort packet delivery service. IEEE/ACM Transactions on Networking, 1998,6(4)

29 Technical Specification from Cisco. Distributed weighted random early detection. http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.pdf

30 Athuraliya S, Lapsley D E, Low S H. Random early marking for internet congestion control. In: IEEE Globecom'99. December 1999

31 Hollot C V, Misra V, Towlsey D, Gong W. On designing improved controller for AQM routers support TCP flows. In: IEEE INFOCOM 2001. April 2001

32 Kunniyurs, Srikant R. Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. In: IEEE SIGCOMM 2001. August 2001

33 Lin Chuang, Li Yin. Dynamic partial buffer sharing scheme: Proportional packet loss rate. In: 2003 International Conference on Communication Technology (ICCT 2003). April 2003, Beijing, China. 255~258

34 Roland Balmer, Manuel GÜhter, Torsten Braun. A fast trend-sensitive function for the estimation of near-future data network traffic characteristics. Institute of Computer Science and Applied Mathematics (IAM), April 2001

# 第10章

# 分 组 调 度

调度是系统资源管理的核心机制之一，是解决多个业务竞争共享资源问题的有效手段。网络系统资源大致包含三个部分：缓冲区、链路带宽、处理器资源。对缓冲区的管理已在第9章进行了讨论，对于处理器资源的管理涉及到系统实现方面的内容，将在第三部分进行讨论。本章所述分组调度实现对链路带宽的管理，是指按照一定的规则来决定从等待队列中选择哪个分组进行发送，使得所有输入业务流能够按照预定的方式共享输出链路带宽。它影响的主要性能参数包括带宽分配、时延和时延抖动等，是实现网络服务质量控制的核心技术之一。

## 10.1 分组调度概述

### 10.1.1 分组排队策略

在分组到达某个网络节点之后，经过存储，再转发到下一节点。根据分组在系统中的存储位置，可以有三种排队策略[15]。

1. 输出排队（output queued，OQ）

一个分组到达输入接口时，被立即经过交换机构转移到相应的输出接口进行缓存。这种排队方式在概念上比较容易理解，分组在发送前排队，由调度算法进行调度输出，可以方便地用于提供服务质量控制。由于分组仅在输出接口排队等待发送而无须在交换机构中经历延迟，其性能分析是比较容易进行的。因此，大多数理论研究成果均是针对输出排队模型的。输出排队的缺点是，要求交换机构有很高的加速比。在最坏情况下，所有输入接口都可能有分组要转移到同一输出接口。因此，交换机构需要以N（输入接口的个数）倍于接口单链路线速的速率工作，而且需要输出端有较大的缓冲。由于交换机构的各种实现方式都有速度上的限制，当输入接口数较多、接口速率也较高时，输出排队就无法实现。

2. 输入排队（input queued，IQ）

把分组首先存储在输入接口中，交换机构只需以线速实现交换即可，因而可以极大地

降低所需的加速比，具有良好的可扩展性。但是，可能会发生各个输入接口争用交换机构的情况，并存在 HOL（head of line）blocking 问题，在输出端口数较多时最多可以达到 58% 的吞吐率，而周期性的 HOL 可能导致性能的大幅度降低。为了处理这个问题，需要引入复杂的交换机构调度机制以仲裁各输入接口的请求。由于分组在输入接口会因为等待交换机构可用而被延迟，系统的行为变得难以分析。

3. 输入输出排队（combined input-output queued，CIOQ）

将上面两种排队方式结合起来，就形成了输入输出排队策略。这种排队方式综合了输入排队和输出排队的优点，既不需要很大的加速比，便于扩展，又可以有效地避免拥塞。但这种排队策略尚处在研究阶段，还有许多问题需要解决。

## 10.1.2　分组调度的功能

典型地，分组调度发生在路由器去往下一个路由器或主机的输出接口，但可以潜在地存在于路由器内部的任何发生资源竞争而需要排队等待调度的地方。当一个分组到达网络节点后，分类器根据分组（或业务类）的上下文和粒度确定它所在的队列，分组进入相应队列排队等候，直至调度器将其选择发送。

如何把输入业务流对应到不同的队列中，不同的调度算法在不同的网络环境下有不同的方法，需要分类功能和调度规则的配合。先到先服务（first come first served，FCFS）只根据分组的到达时间对之进行服务，队列数为 1。这种调度算法的粒度较大，因为是把所有输入业务流无区别地放在一个队列里。而较复杂的调度算法则会根据一定的规则把输入业务流对应到不同的队列里，从而对输入业务进行有区别的服务，称为 CQS（classification，queuing，scheduling）结构[16]。如图 10.1.1 所示。比如在 Internet 中，可以基于 IP 源/目的地址、传输层/源/目的端口和协议类型对输入业务进行分类，也可以根据 IP 头中的 TOS 字段进行分类，每一类可能对应一个队列。而在 ATM 网络中，可基于虚通道标识（virtual channel identifier，VCI）和虚通路标识（virtual path identifier，VPI）对输入业务进行分类。分类的结果使得不同队列中的分组有不同的服务质量要求，比如第一个队列要求排队时延不超过 5 毫秒，第二个队列要求排队时延小于 20 毫秒即可；或者第一个队列的服务速率为第二个队列服务速率的 2 倍，等等。虽然分类和调度紧密相关，但本章主要讨论对不同业务流所属队列的调度。假定在调度器之前存在一个性能良好的分类器，能够根据调度规则对分组进行分类，并把分组存入到相应的队列中（这涉及到缓冲管理的功能，参见第 9 章）。关于分类的详细讨论参见第 7 章。

传统的路由器在每个输出链路接口只有单个队列，所以调度任务很简单——只要底层链路能传输，它就能从队列中尽快地取出分组。支持 QoS 的网络环境要求路由器具有 CQS 结构：每个接口有一个调度器，每个调度器对应多个队列，这些队列之间共享输出链路容量。何时和如何频繁地从每一队列中取出分组进行传输是由调度器运行分组调度算法完成的。当某个队列中的分组被调度发送时，其余队列只能等待。而在每个队列内部，仍然采用 FCFS 的服务方式。分组长度可以是变化的，而且一个分组的服务不能被抢断。

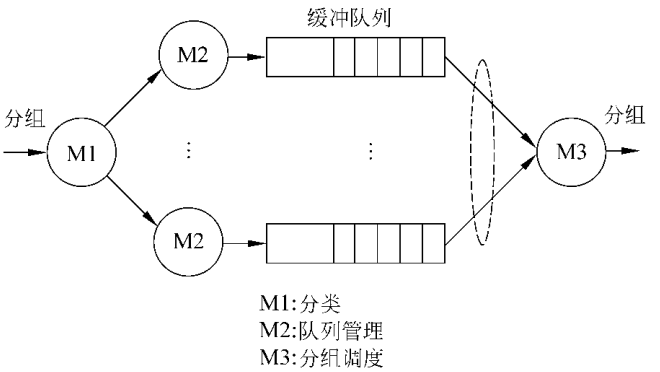



图 10.1.1 多队列系统图

分组调度规定了分组从每个队列离开的瞬时特性,通过流隔离,使得不同业务类的分组得到不同等级的服务。具体表现为带宽分配、时延范围和抖动控制、不同业务类之间公平性和相对优先级。

(1) 带宽分配:一个调度器可为特定的业务类提供最低的带宽保证,以确保分组规则地从该类所在队列中取出(即确保队列被规则地服务)。一个调度器也可以提供速率整形(控制一些特殊业务类的最大允许带宽)以限制该类的队列被服务的频率。根据调度器设计,它可能对每一队列既强调上限带宽又强调下限带宽,或对某些队列只强调上限或只强调下限带宽。

(2) 时延控制:某些实时业务要求严格的时延范围保证和抖动控制,而某些非实时业务则需要较为宽松的时延控制。对于某个业务类,其平均服务速率(服务带宽)R(s)与经历的排队时延 D 存在着关系 $Q = D \cdot R(s)$,其中 Q 为该服务类所在队列的平均长度。这个关系可以从两个方面来理解:①对于具有一定到达速率的业务类,在不考虑分组丢弃的情况下,带宽分配可以通过时延特性来反映。事实上,有些研究者并不把带宽作为一项基本的服务质量指标来考察。②调度器通过控制各个业务类的服务速率分配,来实现对实验特性的控制。分组公平排队 PFQ(packet fair queuing)算法就采用了这种思想。

(3) 相对优先级控制:调度器使得重要的分组得到最好的服务,次要分组得到较差的服务,表现出使用共享资源的一定的相对优先级。这种优先级关系可以在系统初始化时设置为静态优先级,也可以在运行过程中根据系统状态进行调节,即计算动态优先级。

## 10.2 分组调度算法本质分析

调度算法是研究分组调度的核心问题,本质上可以理解为从多个对象 $O_i, 0 < i \leqslant N$ 中选择一个符合某种条件 c 的对象 $O_j$ 进行服务。具体到分组调度,就是从多个队列 $Q_i, 0 < i \leqslant N$ 中选择一个符合某种条件 C 的队列 $Q_j$ 进行服务(分组发送),即从系统中找到一个关系(C,j),其中 j 为队列编号。寻找关系(C,j)的基本方法有两种:

(1) 先确定 j,即假设下一个要服务的队列为 $Q_j$,然后判断该队列是否满足条件 C。如果满足,则服务之;如果不满足,则再判断下一个队列。这种方法可称为基于轮循的

方法。

（2）通过比较判断条件 c 来确定 j。按照条件 c 的要求为每个队列动态地计算一个优先级参数 $p_j$，每次调度具有最大或最小 $p_j$ 值的队列。这种方法可称为基于优先级的方法。大部分现有算法可归结为此类方法，不同之处在于如何设定条件 c 以及如何计算优先级 $p_j$。

另外，从分组调度的控制过程来看，分组调度就是根据某些与队列相关的信息进行判断和控制，从而改变队列和系统的状态，实现某种控制目标。

调度算法所依据的系统信息又分为两个层面：时间层面和空间层面。时间层面是指所考虑的信息可以是静态的（比如静态优先级、权值等），可以是当前的状态信息，也可以是历史状态信息（比如平均值）。空间层面是指考虑不同对象的信息，如数据到达速率、队列长度、分组的等待时间等。一般来讲，考虑的信息越多，算法越有效，当然复杂性也就越高。有些简单算法，如轮循 RR（round-robin）、优先排队 PQ（priority queuing），只考虑队列的静态信息（静态优先级）；有些更先进的算法，如 WFQ（weighted fair queuing）、EDF（earliest deadline first），则考虑了分组的服务时间、到达时间等状态信息。

而调度算法的控制目标在某种程度上是与系统服务模型相关的。算法可以进行带宽分配，例如 DRR（deficit round robin），可以提供严格的时延范围的保证（比如 WFQ，EDF 等，适用于综合服务模型），也可以提供服务等级的比例关系，比如 WTP（waiting time priority），HDP（historical date priority），适用于区分服务模型。

分组调度算法还有一种特性：连续工作（work-conserving）或断续工作（non-work-conserving）[35]。连续工作是指只要系统中有等待分组，调度器就可以选出一个分组进行服务。连续工作的调度算法可以获得很高的链路利用率。断续工作是指即使系统中有等待分组，调度算法也可能暂时不对其进行调度。这一类算法一般是在输入业务流被调度之前对其进行整形（shaping）处理。好处是，可以对端到端时延和时延抖动进行控制；缺点是，链路利用率较低。只有少数算法是断续工作的，比如 HRR（hierarchical round robin），Jitter-EDD（jitter-earliest- due-date），Stop-and-Go。

## 10.3　分组调度算法的性能指标

分组调度算法在不同环境下可能有不同的应用。比如分组调度算法可能被用于隔离恶意业务流来为正常业务流提供服务质量保证，还可能用来让用户平等地使用共享链路的可用带宽。有效的分组调度算法应该拥有诸多良好的特性，这里将其总结为有效性、公平性和复杂性三个方面。

### 1. 有效性

分组调度算法应该能够实现预期的控制目标，使得各个数据流（业务类）能够得到事先约定的等级的服务。有效性可以包括资源利用率和时延特性等。资源利用率体现了系统对调度算法的要求，时延特性体现了用户对调度算法的要求。分组调度算法应为不同的业务流提供端到端的时延保证，而且只与此业务流的某些参数（如带宽需求）有关，而

与其他业务流无关,或者保证不同业务流时延之间的优先级关系。Stiliadis 和 Varma 首先提出了一种分析网络中不同分组调度算法带来的端到端时延的模型:时延速率服务器(latency rate server, LRS)[40]。Francini 随后又提出了另一种分析端到端时延的模型:速率分隔时签调度器(rate spaced timestamp scheduler, RST)[41,42],此模型的限制条件比 LRS 少,而且在定长分组环境下应用时更加有效。

2. 公平性

可用的链路带宽必须以公平的方式分配给共享此链路的各个业务流,并且必须能够隔离不同的业务流,让不同的流只享用自己可以享用的带宽,这样即使存在恶意或高突发性业务,它也不致影响到其他的正常业务流。然而,对于公平性,不同的人有着不同的理解。有人认为,公平就是指所有用户均等地占用资源,也有人认为,公平是相对而言的,与用户的需求和系统状态有关。关于算法公平性的定义有:服务公平指数(service fairness index, SFI)[22,43]、最坏公平指数(worst-case fairness index, WFI)[20, 22,43]和 Raj Jain 公平指数(Raj Jain s fairness index)[44]以及成比例公平原则(见第 12 章)。SFI 表示任意两个活动队列在任意时间间隔内收到的规格化(normalized)服务量(等于服务量与其分配的服务速率的比值)的最大差值;WFI 用来表示一个队列在分组级系统和相应流系统上接收到的服务量的最大差值,较大的 WFI 意味着调度输出业务较大的突发性;Raj Jain 公平指数表示所有队列收到的服务量之和的平方与这些服务量平方和的比值再除以队列个数,可用来评价系统中所有业务流的带宽分配或时延公平性。

3. 复杂性

速度的不断提高和规模的不断扩大是网络发展的一个趋势。在高速网络中,分组调度算法必须能够在很短的时间内完成分组的调度转发。这要求调度算法应该比较简单,易于实现。另外,当业务流数量增加和链路速率变化范围较大时,调度算法仍应有效工作,即要求算法具有良好的可扩展性。然而,算法的有效性和复杂性之间是存在矛盾的,一般来讲,考虑的信息越多,算法越有效,复杂性也就越高。因此分组调度算法通常是在实现简单性和有效性之间进行折中。

有些更复杂的评价方法把多个指标综合在一起,参见第 12 章内容。

## 10.4 常用的分组调度算法比较

分组调度算法的研究一直是一个热门课题,到目前为止,产生的算法已有几十种,分别有着不同的服务规则、控制目的和复杂度。有些综述性文章[38, 39]对常用算法进行了归类比较。实际上,对于某一特定的调度算法,根据不同的归类规则,可以属于多个不同的类别,因此并没有一种统一的归类标准。本章结合对分组调度算法的本质分析,根据工作原理和控制目标并参考文献[38, 39]把算法归为以下几类:基于静态优先级、基于轮循、基于 GPS(generalized processor sharing)模型(PFQ 算法)、基于时延、分层链路共享算法、核心无状态算法、基于服务曲线、比例区分算法、结合缓冲管理的算法。这里,重点讨论各

类算法的基本思想，关于具体算法的详细内容，有兴趣的读者可以阅读相关参考文献。

### 10.4.1　基于静态优先级的算法

常见的基于静态优先级的分组调度算法有 PQ 和 QLT（queue length threshold）[17]。

PQ 算法给每个队列赋予不同的优先级，每次需要调度时，具有最高优先级的非空队列中的分组最先被选择服务。如果最高优先级的队列为空，则服务具有次优先级的队列，如此类推。这样，最重要的分组就能得到最好的服务，比如最小的时延。此类算法简单，容易实现，然而在高优先级队列源源不断地有分组到达时，低优先级的队列容易被“饿死”，即在很长时间内得不到服务，因而公平性很差。而且，优先级是静态配置的，不能动态适应变化的网络要求。

解决 PQ 算法中低优先级队列“饿死”现象的一个有效手段就是设置调度阈值。QLT 给每个队列设置调度阈值，需要进行调度时从最高优先级开始比较队列的长度和调度阈值。当最高优先级队列的长度大于等于其调度阈值时，该队列头部的分组首先被选择服务。当最高优先级队列的长度小于其调度阈值时，不再服务调度该队列，而是检查具有次高优先级的队列，如此类推。通过设置合理的调度阈值，QLT 算法在保证优先级关系的基础上，提高了公平性。

### 10.4.2　基于轮循的算法

常见的基于轮循的算法有 RR，WRR（weighted round-robin）[18] 和 DRR[1]。

传统的 RR 只是简单地对所有队列进行轮循调度，一次调度发送一个分组，使得不同队列在某种程度上“平等”地使用带宽资源。然而由于分组长度不固定，使得长分组队列可能比短分组队列得到更多的服务，获得更高的带宽，因而其公平性受到很大限制，而且不能提供时延保证。为了提高 RR 算法的性能，出现了许多改进算法，常见的有 WRR 和 DRR。

WRR 给队列赋予不同的权值，代表一次完整循环队列被服务的分组数。同时为每个队列维护一个计数器，初始化为权值。每次轮循时，计数器为非零的队列允许发送一个分组，并把计数器减 1。当所有队列的计数器均为零时，重置为权值。WRR 可以比 RR 更灵活地控制带宽在不同队列的分配，而且能以比较平滑的方式调度输出业务，但仍然存在由于分组变长带来的不公平性。

为了解决 RR 和 WRR 算法中由于分组变长带来的不公平性，DRR 算法以字节为单位为每个队列分配一个带宽配额，该配额的比例对应于队列服务速率的比例。同时，为每个队列维护一个计数器，初始化为其带宽配额。每次轮循时，如果待发分组长度小于或等于计数器值，则允许发送，并把计数器减去此分组长度值；如果待发分组长度大于计数器值，则检查下一个队列，同时把该队列计数器的差值累计到下一次循环（即下次调度该队列之前把此剩余值和配额之和赋予计数器）。DRR 由于考虑了分组变长这一信息，很好地解决了带宽分配的公平性问题；缺陷是不能很好地满足业务的时延特性，不能像 WRR 那样以较为平滑的方式调度输出。

除此之外，还有许多更复杂的基于轮循的算法，比如 URR（urgency-based round

robin)[19],SRR(surplus round robin)[14]等。

### 10.4.3 基于GPS模型的算法(PFQ算法)

常见的基于GPS模型的算法有WFQ[3],WF2Q(worst-case fair weighted fair queueing)[20],WF2Q+[21],VC(virtual clock)[2],SCFQ(self-clocked fair queueing)[22],SFQ(stochastic fair queueing)[23],SPFQ(starting potential fair queueing)[24],FFQ(frame-based fair queueing)[24]。此类算法是对理想流体模型GPS[36,37]的逼近,称为PFQ(packet fair queueing)[20]算法。

GPS模型是一个理想化的流模型,假定每一个队列中的业务元可以无限小,队列之间具有相同的优先级,因而调度器可以根据各队列的共享比例同时服务所有的队列,能使各业务流真正公平地共享服务器,为每个业务流提供明确的端到端时延上限保证。GPS模型在实际网络系统中是无法实现的,因为业务元的最小粒度是分组,而且该分组的服务不能被抢断。

PFQ是把GPS模型应用于网络系统的一类近似算法,在分组进入各自业务流的队列时,为其加上调度优先级标记,并依此顺序逐次调度各分组进行服务。PFQ代表了一类算法,其中包括许多具体算法,常见的有WFQ,WF2Q,WF2Q+等。

此类算法的基本思想是维护一个系统虚拟时间$V(t)$,并为每一个队列维护一个虚拟开始时间标签$S_i(t)$(代表队列i头部分组的虚拟开始发送时间)和一个虚拟完成时间标签$F_i(t)$(代表队列i头部分组的虚拟完成发送时间),并当队列中有分组发送完毕,或者空队列有分组到达时分别计算如下:

$$S_i(t) = \begin{cases} \max(V(t), F_i(t^-)), & \text{如果队列 } i \text{ 在前一时刻为空} \\ F_i(t^-), & \text{如果分组 } p_i^{k-1} \text{ 被发送完毕} \end{cases},$$

$$F_i(t) = S_i(t) + L_i^k / r_i,$$

其中$F_i(t^-)$是队列i更新之前的虚拟完成时间,$L_i^k$是当前队列i第1个分组$P_i^k$的长度,$r_i$是预先定义的服务速率。

当每次需要调度时,系统根据时间标签的大小选择一个分组。系统通过给用户链接提供最小带宽保证,进而提供了时延范围的保证。

所有PFQ算法都是在虚拟时间函数的基础上根据一定的选择策略进行调度的。其目的是尽量使$S_i(t)$和$V(t)$之间的差别达到最小,从而尽量精确地逼近GPS模型。$V(t)$的作用是当一个队列变为活动时重置这个队列的虚拟开始时间,以保证算法的公平性。各种具体算法的不同之处在于系统虚拟时间函数$V(t)$的计算和分组选择策略,因而也导致了各种算法在精确度和复杂度上的差异。

(1) 系统虚拟时间函数$V(t)$的计算:WFQ和WF2Q定义了与GPS最接近的虚拟时间函数$\frac{dV(t)}{dt} = \frac{C}{\sum_{j\in\theta(r)} r_j}$,其中C表示服务器的服务速率(链路总带宽),$\sum_{j\in\theta(r)} r_j$表示t时刻有分组等待调度的所有队列的服务速率之和。但这是以高复杂性$O(N)$为代价的。WF2Q+算法是对WF2Q的改进,把$V(t)$的计算复杂度降到了$O(\log N)$,却保持了与WF2Q相近的

精确度和性能：$V_{WF2Q+}(t+\tau) = \max(V_{WF2Q+}(t)+\tau, \min_{i\in B(t)}(S_i(t)))$，其中 B(t) 是 WF2Q + 系统在 t 时刻非空队列的集合。SPFQ[24] 和 MD-SCFQ 具有与 WF2Q + 相类似的复杂度和性能。其他一些算法也试图降低虚拟时间函数计算的复杂性，但其精确度较低。比如 SCFQ 定义虚拟时间函数为：当 t 时刻有分组正被服务时，等于该分组的完成时间；而当队列无分组时，重置为 0。

（2）分组选择策略主要有三种：最小虚拟完成时间优先（smallest virtual finished time first，SFF）——即系统中具有最小虚拟完成时间的队列的对头分组被选择发送、最小虚拟开始时间优先（smallest virtual started time first，SSF）和最小合法虚拟完成时间优先（smallest eligible virtual finished time first，SEFF），所谓合法是指分组的虚拟开始时间不大于当前的系统虚拟时间。前两种策略对分组的选择只使用一个时间标签，因而可能会产生与 GPS 模型较大的偏差，影响系统的公平性。在具体算法中，WFQ，SCFQ 和 FBFQ（flow-based fair queueing）使用了第 1 种，SFQ 使用第 2 种，WF2Q 和 WF2Q + 使用第 3 种。但这些算法都具有相同的排序复杂度 O(logN)。

## 10.4.4　基于时延的算法

常见的基于时延的算法有 EDF[4]，RCS（rate-controlled service）[25]，Delay-EDD[26]，Jitter-EDD[27]，RC-EDF（rate controlled EDF）[28]，DC-EDF（deadline-curve based EDF）[29]，EEDF（earliest effective deadline first）[30]。

基于轮循和 GPS 模型的调度算法可以看成是基于速率的调度算法，即通过为每一队列提供一定的速率保证来获得时延保证。而基于时延的算法则是直接以排队时间作为参数，并以提供时延保证为目的。

此类算法的基本思想是给每一个队列分配一个时延参数 $D_i$ 作为时延上界，为每一个到达的分组计算时间标签 $T_i = A_i + D_i$ 作为到期时间（deadline），其中 $A_i$ 是分组到达时间。每次调度具有最小到期时间 $T_i$ 的分组，因而具有 O(logN) 的排序复杂度。EDF（有时又称为 EDD，earliest due date）是此类算法的代表。但当应用于网络环境时，为了在业务流量发生突发时保证端到端的时延，通常要求引入抖动控制或整形等规范化机制对输入业务进行调节（关于流量整形参见第 8 章）。其一般的模型为：每个网络节点由规范化器（速率控制器）和 EDF 调度器组成，其中第一个网络节点的规范化器进行流的重新整形，后续网络节点的规范化器进行流量特征的恢复。分组的到期时间标签 $T_i$ 是将分组从规范化器中输出的时间加上其所属业务流的最大允许延迟 $D_i$。这样，系统增大了处理时延：总的时延 = 规范化时延 + 调度时延，并容易引入断续工作的特性，使得链路利用率下降。但是也可以改造成连续工作的方式：计算出分组进行规范化所需的延迟时间，再据此计算出分组的到期时间标签 $T_i$，然后就按照此到期时间将分组直接放入 EDF 调度器中等待传送，而不是先对其进行延迟再放入调度器中。这实际上相当于分组提前进入了调度器，但其到期时间与断续工作方式调度策略是相同的。

目前有许多 EDF 的改进算法，比如 Delay-EDD 采用流量整形的方法，为连续工作；Jitter-EDD 采用延迟抖动控制方法，为断续工作。Delay-EDD 算法中的到期时间称为 Expected Deadline（用 ExD 表示）。算法中假定所调度的经过整形满足（$X_{min}$，$X_{ave}$，I，$S_{max}$）

模型。当有分组到达时,根据 $X_{min}$ 等参数计算并修改 ExD = max(A + D),ExD + $X_{min}$,系统把修改过的 ExD 值赋予相应队列作为调度优先级。($X_{min}$,$X_{ave}$,I,$S_{max}$)模型的含义是流中的分组必须同时满足以下条件:任意两相邻分组之间的到达时间间隔必须大于 $X_{min}$;在任意的长度为 I 的时段内,相邻分组的到达时间间隔的均值必须大于 $X_{ave}$;流中的最大分组长度必须小于 $S_{max}$。在 Jitter-EDD 算法中,当有分组到达时,首先对该分组进行时延抖动调节,根据该分组中记录的(由上一交换节点标记)提前输出时间值 Ahead,计算分组的可被调度时间 $E_i = A_i$ + Ahead,并在调节器中保持此分组直到时刻 $E_i$。随后调度算法以此时刻作为分组到达时刻,按照 Delay-EDD 算法中所述的类似调度机制进行。但在分组被输出时,必须将其输出时刻与希望输出时刻之间的差值赋予 Ahead,并记入该分组中,以便后续算法运行。

## 10.4.5 分层链路共享算法

常见的分层链路共享算法有 CBQ(class-based queueing)[5],H-PFQ(hierarchical packet fair queueing)[31],HFSC(hierarchical fair service curve)[32]。

“实时性”是许多业务的一个典型服务质量要求,满足这一要求的一般做法是为其提供足够的资源保证,包括缓冲区和链路。“链路共享”反映了在总体资源有限的情况下,为提高资源利用率,并在多个业务之间公平、合理地分配资源的要求。从不同的角度来看,链路共享存在着多种情况。比如同一条链路可以被多个公司、组织或代理共享;也可以被 IP,SNA 等多种协议共享;还可以被 WWW,FTP,Telnet 等多种应用所共享。随着网络规模的发展,单一层次的链路共享方式已经不能满足复杂的系统要求。因此导致了分层链路共享策略的出现。

分层链路共享把不同类型的业务流划分成不同级别的类,并同时考虑业务的实时性和链路共享的需求,按照不同的层次进行分组调度,从而提供了在分层共享树内进行不同等级的共享链路带宽资源的方法。如图 10.4.1 所示,其中百分比表示该业务类预先分配的带宽比例。

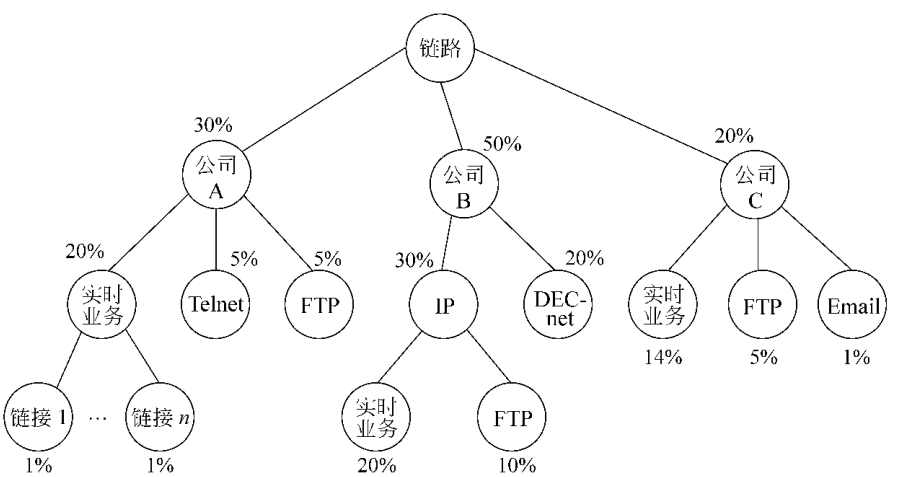



图 10.4.1 分层链路共享结构图

分层链路共享的目标：

（1）如果某个业务类有数据到达，系统应该能够为其提供预先分配的带宽保证。这样，保证了某些高优先级业务的实时要求，即使在系统出现拥塞的时候。

（2）某个类未用完的带宽可以根据一定的规则被同级别的其他“兄弟”类共享，而不是被所有其他类共享。这样使得某些低优先级业务不会被“饿死”，提高了系统的公平性。

为实现这一控制目标，系统从概念上设计了两级调度器：常规调度器（general scheduler）和链路共享调度器（link-sharing scheduler）。当系统没有出现拥塞时，使用常规调度器，使得各个业务类都能得到所需要的服务。这里，常规调度器可以采用许多具体的分组调度算法，比如轮循调度、静态优先级调度等。当系统出现拥塞时，启动链路共享调度器，对某些超出其额定带宽的业务类进行限制，对其“规则化”。实际上，这两级调度只是从概念上进行了划分，系统在具体实现的时候不必设计两个单独存在的调度器，而可以只使用同一块代码，并通过参数选择根据系统状态完成不同的功能。

此类算法的优点是，能够在保持高链路资源利用率的情况下提供很好的带宽、时延保证以及公平性。

### 10.4.6　核心无状态算法

常见的核心无状态算法有 CSFQ（core stateless fair queueing）[33] 和 CJVC（core stateless jitter virtual clock）[34]。

前文所讨论的分组调度算法把网络各节点看成具有相同的地位，在网络各个节点中执行相同的功能，而且最初都是基于每个业务流提出的，它们需要交换节点维护每个业务流的一些状态信息（比如分组的时间标签）。尽管这样可以达到很好的调度性能，但却带来了不易扩展的缺点。即随着网络规模的增长，网络核心节点中数据流的数量急剧上涨。如果调度算法继续为所有的数据流都维护状态信息，则会增大处理难度，使得系统性能急剧降低。为此出现了一类类似于区分服务模型的分组调度算法：核心无状态算法。此类算法是对传统的基于每个流的调度策略的改造，即通过研究在调度过程中需要用到哪些状态信息，并考虑如何在核心网络节点中消除它们。

类似区分服务的体系结构，此类算法把网络分为边缘和核心两个域，同时又需要边缘节点和核心节点之间的有效配合。在边缘节点为每个流维护一定的状态信息，并给每一个分组打上调度策略所需的信息标签。而核心节点则不必进行复杂的流分类和为每一个流都维护状态信息，而是在分组传输的每一跳上，根据分组携带的信息标签进行调度。信息标签由边缘网络节点生成，一部分信息在传输过程中保持不变，另一部分则在分组转发之前进行修改以供下一节点使用。这种在分组转发过程中修改分组头状态信息的方法称为 DPS（dynamic packet state）技术，是核心无状态调度策略的关键部分。

此类算法的优点是避免了在核心交换节点的基于每个业务流的调度，从而使算法复杂度降低，并具有良好的可扩展性。缺点是算法的有效运行依赖于边缘交换节点和核心交换节点的有效配合，需要修改分组头（附加信息标签并在每一条进行修改），进而可能会影响某些协议。

## 10.4.7 基于服务曲线的算法

常见的基于服务曲线的算法有 SCED(service curve-based earliest deadline)[6] 和 HFSC(hierarchical fair service curve)[32]。

早期在 Parekh 和 Gallager 的工作中[36]就出现了服务描述函数的概念。他们在其调度算法中引入了一种通用的服务描述函数。这种方案的一个优点是,它能把对服务质量的要求通过一个简单的描述函数表示出来,并将一个网络连接的服务特性与其他网络连接区别开来,另一个重要特点是给予服务器更大的灵活性以为具有不同时延和带宽要求的业务流分配资源。Cruz 在文献[7, 8]中也提出了一个更为宽松的服务曲线概念,采用了更宽松的服务定义作为对服务特性的通用描述框架。每一种业务类都被赋予一条服务曲线,这条服务曲线指定了它不同时刻应该收到的最小服务量。

如果对于有分组需要调度的业务流 i 的任意时刻 $t_2$,存在一个时刻 $t_1 < t_2$($t_1$ 为业务流 i 有分组需要调度的某期间的开始时间,且该期间不一定包含时刻 $t_2$),并满足关系式 $S_i(t_2 - t_1) \leqslant w_i(t_1, t_2)$,其中 $w_i(t_1, t_2)$ 表示业务流 $t_2$ 在时间间隔 $(t_1, t_2]$ 内得到的服务量,那么业务流 $t_2$ 称为保证了服务曲线 $S_i(\cdot)$,且 $S_i(\cdot)$ 是一个非递减函数。如果对于任意时刻 $t_1$ 和 $t_2$,以及任意 $\alpha \in (0,1)$ 存在 $S_i(\alpha t_1 + (1-\alpha)t_2) \leqslant \alpha S_i(t_1) + (1-\alpha) S_i(t_2)$,则 $S_i(\cdot)$ 称为凸曲线 如果存在 $S_i(\alpha t_1 + (1-\alpha)t_2) \geqslant \alpha S_i(t_1) + (1-\alpha) S_i(t_2)$,则 $S_i(\cdot)$ 称为凹曲线。

根据这一模型,产生了几种基于服务曲线的算法,比如基于服务曲线的最早期限优先 SCED 和分级的公平服务曲线 HFSC。

SCED 算法把 $s(t)$ 定义为 $S(t) = \begin{cases} 0, & \text{if} \quad 0 \leqslant t \leqslant d^{max} - 1 \\ b(t - d^{max}), & \text{if} \quad t \geqslant d^{max} \end{cases}$,其中 $b(t)$ 为到达曲线,$d^{max}$ 为最大时延:即服务曲线是直接由到达曲线平移 $d^{max}$ 来确定,因此 SCED 能够保证业务的时延特性。当有分组到达时,SCED 根据其服务曲线计算此分组应该被发送的一个到期时间,然后按照由小到大的顺序依次发送各个分组。SCED 能保证所有服务曲线的前提是所有服务曲线之和不大于系统总的服务曲线(等于 $R \times t$,其中 R 为输出链路总带宽)。SCED 能保证业务的实时性,但却不能同时保证业务之间的公平性。

在 HFSC 算法中,服务曲线由三个参数来确定:最大分组长度、最大时延和平均速率。时延要求严格的业务分配一个凸的服务曲线,而时延要求宽松的业务分配一个下凹的服务曲线,其实质在于给需要小时延的业务提供一个短时的高速率(大于其平均速率),而让时延要求宽松的业务在一个较短的时间里不接受服务或服务速率小于平均速率。

这种利用服务曲线为业务流提供服务保证的方法具有更大的通用性。比如基于 GPS 模型的算法可以看作是保证了一条过原点的线性服务曲线,但由于该线性服务曲线只用一个参数(速率)来指定业务的质量要求而不能单独指定时延要求,使得时延与带宽的分配互相耦合在一起,灵活性很差。相比之下,通过综合考虑时延和带宽要求,设计非线性

的服务曲线就可以实现时延和带宽的解耦，实现灵活的资源管理和较高的资源利用率。

## 10.4.8　比例区分算法

常见的比例区分算法有 PAD（proportional average delay）[9]，WTP（waiting-time priority）[9] 和 HPD（hybrid proportional delay）[9]。

比例区分服务（proportional differentiated services）[10] 是一种相对区分服务。在相对区分服务（relative differentiated services）[10] 中，数据流被组合成服务类（service classes），这些服务类按照其分组转发的服务质量要求进行排序，以决定其排队延迟、分组丢失率等转发行为。比如排序结果使得类 i 应该有更好于（或者至少不低于）类（i - 1）（1 < i < N）的服务质量（注意，“至少不低于”是必须的，因为在低负载的情况下所有分组获得的服务质量可能相同，即网络能满足所有数据流的服务要求）。网络不提供接纳控制和动态资源预留，而是由用户或应用程序选择最符合它们质量要求和价格限制的服务类。网络应用和用户不能获得绝对的服务质量保证，但网络能够保证高级别的类可以得到比低级别的类更好的服务。在比例区分服务模型中，任意两个数据类在每一跳（per-hop）所获得的服务满足确定的比例，比例参数由网络管理者设定，而且与类的负载无关。即对于任意 $1 < i, j < N$，有 $q_i(t, t+\tau) / q_j(t, t+\tau) = c_i / c_j$，其中 $q_i(t, t+\tau)$ 为数据类 i 在时间段 $[t, t+\tau]$ 内所获得的服务，$c_1 < c_2 < \cdots < c_N$ 为网络管理者设定的比例参数。

WTP，PAD 和 HPD 是针对比例区分服务模型设计的算法，其控制目标是使得任意两个类的排队时延保持在恒定的比例，以在最大程度上逼近成比例时延区分（proportional delay differentiation，PDD）模型。在 WTP 算法中，假定在时刻 t 业务类 i 有分组需要调度，$w_i(t)$ 是该队列队头分组的等待时间，则定义规格化的队头等待时间为 $\tilde{w}_i(t) = w_i(t) / \delta_i$，其中 $\delta_i$ 是预先定义的比例系数。当每次需要调度时，选择具有最大规格化队头等待时间的非空队列：$j = \arg \max_{i \in B(t)} \tilde{w}_i(t)$。WTP 能在很短的时间尺度内逼近 PDD 模型，但是在轻负载的环境下容易偏离 PDD 模型。相比之下，PAD 算法采用已发送分组的平均等待时间进行计算比较，因而能在较大的时间尺度下逼近 PDD 模型，但是当遇到数据突发时则容易偏离。HPD 算法把前两者进行综合，利用已发送分组的平均等待时间和队头分组的等待时间的加权之和进行计算比较，因而具有更好的性能。

## 10.4.9　结合缓冲管理的算法

常见的结合缓冲管理的分组调度算法有 C-DBP-Delay-Loss[11] 和 JoBS[12]。

到目前为止，关于缓冲管理和分组调度的研究绝大部分都是分开进行的，前面讨论的分组调度算法很少考虑分组的丢失率这一服务质量的指标。分组丢失率通常由缓冲管理来控制。然而，缓冲管理作为分组入队列的操作，分组调度作为分组出队列的操作，两者从对系统的影响来看有着紧密的内在联系，对系统性能的影响是统一的。比如，某队列被调度的机会越多，该队列所对应的数据流越能获得高带宽、低时延和低丢失率，同时也会影响不同队列之间获得服务的公平性。又如，在缓冲管理方案中，增加队列容量可以在某种程度上降低丢失率，提高吞吐率，但同时也增加了分组的排队时延。因此，为了获得较好的综合性能，需要把缓冲管理和分组调度结合起来考虑，设计综合算法。

有些更先进的算法,比如 C-DBP-Delay-Loss 和 JoBS 试图把丢失率和时延综合在一起进行控制。

C-DBP-Delay-Loss 是把(m,k)模型和用于调度实时数据流的 DBP(distance based priority)算法[13]应用于相对区分服务网络的一种改进算法。(m,k)模型提供了一种用以计算某个流的服务质量特性的方法:在某给定流中每 k 个连续分组中,至少有 m 个分组能满足其端到端时延限制。在 DBP 算法中,每个网络节点的每个流都关联一个状态机,其中的状态描述了这 k 个分组的传输情况。一个流的 DBP 值等于到达某失败状态所需发送的分组数目,其中失败状态是指发送分组数小于丢弃分组数。DBP 值越小,该流的优先级越高。系统总是选择具有最高优先级的流进行服务。如果把 DBP 算法中的流改为业务类(class),即系统中的队列对应业务类,而不是原来的微流,则该算法可被改造成 C-DBP(class DBP)算法。C-DBP-Delay-Loss 调度器通过综合考虑时延优先级和丢失优先级,可同时实现时延区分和丢失区分的控制。但 C-DBP-Delay-Loss 本质上只是一种考虑了丢失率的调度算法,而没有提供分组丢弃的控制方案。

JoBS 算法实现了基于区分服务类的服务速率分配和丢弃控制,可同时提供单节点的时延区分和丢失区分,其特点为:①把缓冲管理和分组调度综合到一步完成;②可同时支持相对和绝对的服务质量(时延和丢失率)保证。JoBS 算法的基本工作原理如下:定义一个系统目标函数,并为每个队列设定 QoS 约束(包括时延和丢失率,可以是几个队列之间的相对约束,也可以是单个队列的绝对约束)。当有分组到达时,对所有队列计算时延估计,然后对各个队列的服务速率分配进行调整,使得所有 QoS 约束都能得到满足。如果不存在满足所有 QoS 约束的速率分配方案,则选择分组(从当前队列或者其他队列)进行丢弃。JoBS 算法把速率分配归结为一个多目标优化问题。其中约束条件为前面所述的关于时延和丢失率的 QoS 约束以及关于链路和缓冲容量的系统约束;目标函数旨在使丢失率最小,其次保持当前的速率分配不变。第一个目标是为了防止某些不必要的分组丢弃,第二个目标是为了避免服务速率的抖动。该优化问题的解就构成了队列的服务速率分配和分组丢弃方案。但是 JoBS 算法仍然具有类似 PFQ 算法的服务速率和时延的耦合问题。

### 10.4.10 分组调度算法小结

分组调度是实现网络服务质量控制的核心机制之一,是网络资源管理的重要内容,通过控制不同类型的分组对链路带宽的使用,使不同的数据流得到不同等级的服务。

分组调度的基本过程是根据一定的系统信息做出判断,控制各个队列占用链路带宽的频率,进而影响时延、丢失等服务质量空间状态。一般来讲,考虑的信息越多,算法越有效,复杂性也就越高。先进的队列调度算法都是把分组存放在不同的队列里,然后为其计算一个动态优先级作为控制参数,根据这些参数进行调度。

未来网络有两个发展趋势:带宽高速化和业务多样化。分组调度算法的研究要适合这些发展趋势,既要保证高的分组调度速度,又要提供时延等服务质量和公平性保证。也就是说,在追求简单性和易实现性的同时,考虑算法的综合性能。

# 参考文献


1　Shreedhar M，Varghese G. Efficient fair queueing using deficit round-robin. IEEE Transactions on Networking，1996，4(3)：375 ~ 385

2　Zhang L. Virtual clock：A new traffic control algorithm for packet switching. ACM Trans Comp Syst，1991，101 ~ 124

3　Parekh，Gallager R. A generalized processor sharing approach to flow control in integrated services networks：The single node case. IEEE INFOCOM92，1992

4　Liu，Layland J. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of ACM，1973，20(1)：46 ~ 61

5　Floyd Sally，Jacobson Van. Link-sharing and resource management model for packet networks. IEEE/ACM Trans on Networking，1995，3(3)

6　Sariowan H，Cruz R L，Polyzos G C. Scheduling for quality of service guarantees via service curves. In：Proc Intl Conf on Computer Communications and Networks (ICCCN) 1995. Sept. 1995，512 ~ 520

7　Cruz R L. Service burstiness and dynamic burstiness measures：A framework High Speed Networks，1992，1(2)：105 ~ 127

8　Cruz R L. Quality of service guarantees in virtual circuit switched network. IEEE J Select Areas Comm，1995，13：1048 ~ 1056

9　Dovrolis C，Stiliadis D，Ramanathan P. Proportional differentiated services：Delay differentiation and packet scheduling. IEEE/ACM Trans on Networking，2002，10(1)：12 ~ 26

10　Dovrolis C，Ramanathan P. A case for relative differentiated services and the proportional differentiation model. IEEE Network，1999，13：26 ~ 34

11　Striegel A，Manimaran G. Differentiated services：Packet scheduling with delay and loss differentiation. Computer Communications，2002，25(1)：21 ~ 31

12　Liebeherr J，Christin N. JoBS：Joint buffer management and scheduling for differentiated services. In：Proc of IWQoS 2001. Karlsruhe，Germany，June 2001：404 ~ 418

13　Hamdaoui M，Ramanathan P. A dynamic priority assignment technique for streams with (m，k)-firm guarantees. IEEE Trans Computer，1995，44(12)：1443 ~ 1451

14　Guo C. SRR：An O(1) time complexity packet scheduler for flows in multi-service packet networks. In：Proc of ACM SIGCOMM01. 2001，pp. 211 ~ 222

15　**江勇，吴建平，徐恪. 高性能交换体系结构及其调度算法分析. 电子学报，**2000，S1

16　Armitage G. **隆克平等译**. IP **网络的服务质量. 北京：机械工业出版社，**2001. 22 ~ 23

17　Chiopalkatti R，Kurose J，Towsley D. Scheduling policies for real-time and non-real-time traffic in a statistical multiplexer. In：Proc of the IEEE INFOCOM89. Ottawa，Canada，April 1989. 774 ~ 783

18　Shimonishi H，Yoshida M. An improvement of weighted round robin cell scheduling in ATM networks. In：Proc of IEEE Globecom97. vol. 2，1997. 1119 ~ 1123

19　Altintas O，et al. Urgency-based round robin：A new scheduling discipline for packet switching networks. In：Proc IEEE INFOCOMM98. vol. 2，1998. 1179 ~ 1183

20　Bennett J. R，Zhang H. WF2Q：Worst-case fair weighted fair queueing. In：Proc of IEEE INFOCOM96. vol. 1，San Francisco，CA，Mar. 1996. 120 ~ 128

21 Bennett J C R, Zhang Hui. Hierarchical packet fair queueing algorithms: IEEE/ACM Trans on Networking, 1997, 5(5): 675 ~ 689
22 Golestani S J. A self-clocked fair queueing scheme for broadband applications. In: Proc of IEEE INFOCOM94. Toronto, June 1994. 634 ~ 646
23 Goyal P, Vin H M, Cheng H. Start-Time fair queueing: A scheduling algorithm for integrated services packet switched networks. IEEE Trans on Networking, 1997, 5: 690 ~ 704
24 Stiliadis D, Varma, A. Efficient fair queueing algorithms for packet-switched networks. IEEE/ACM Trans on Networking, 1998, 6(2): 175 ~ 185
25 Zhang H, Ferrari D. Rate-controlled service disciplines. Journal of High Speed Networks, 1995, 3(4): 389 ~ 412
26 Ferrari D, Verma D C. A scheme for real-time channel establishment in wide-area networks. IEEE JSAC, 1990, 8(3)
27 Verma D, Zhang H, Ferrari D. Delay jitter control for real-time communication in a packet switching network. In: Proc of the TriComm 91. Chapel Hill, NC, Mar. 1991
28 Georgiadis L, Guerin R, Peris V, Sivarajan K. Efficient network QoS provisioning based on per node traffic shaping. IEEE/ACM Trans on Networking, 1996, 4(4)
29 Zhu Kai, Zhuang Yan, Viniotis Yannis. Achieving end-to-end delay bounds by EDF scheduling without traffic shaping. In: Proc IEEE Infocom 2001
30 Liebeherr J, Yilmaz E. Work-conserving vs. non-workconserving packet scheduling: An issue revisited. In: Proc IEEE/IFIP 7th International Workshop on Quality of Service (IWQoS 99). June 1999
31 Bennett J C R, Zhang H. Hierarchical packet fair queueing algorithms. IEEE/ACM Trans on Networking, 1997, 5(5). Also In: Proc of SIGCOMM96, Aug. 1996 675 ~ 689
32 Stoica Ion, Zhang Hui, Eugene N T S. A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. IEEE/ACM Trans on Networking, 2000, 8(2): 185 ~ 199
33 Stoica Ion, Shenker Scott, Zhang Hui. Core-Stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high speed networks. In: SIGCOMM98
34 Stoica Ion, Zhang Hui. Providing guaranteed services without per flow management. In: ACM SIGCOMM99. Boston, MA, Sept. 1999
35 Zhang Hui. Service disciplines for guaranteed performance service in packet-switching networks. In: Proc of the IEEE, 1995, 83(10): 1374 ~ 1396
36 Parekh A K, Gallagher R G. A generalized processor sharing approach to flow-control in integrated services networks: The single-node case. IEEE/ACM Trans on Networking, 1993, 1(3): 344 ~ 357
37 Parekh A K, Gallagher R G. A generalized processor sharing approach to flow-control in integrated services networks: The multiple node case. IEEE/ACM Trans on Networking, 1994, 2(2): 137 ~ 150
38 **王重钢,隆克平等. 分组交换网络中队列调度算法的研究及其展望. 电子学报,**2001,29(4)
39 **王宏宇,顾冠群. 集成服务网络中的分组调度算法研究综述. 计算机学报,**1999,22(10)
40 Stiliadis D, Varma A. Latency-Rate servers: A general model for analysis of traffic scheduling algorithms. IEEE/ACM Trans on Networking, 1998, 6(5)
41 Chiussi F M, Francini A. Implementing fair queueing in ATM switches-Part 1: A practical methodology for the analysis of delay bounds. In: IEEE GLOBECOM97. 1997. 509 ~ 518
42 Chiussi F M, et al. Implementing fair queueing in ATM switches-Part 2: The logarithmic calendar queue. In: IEEE GLOBECOM97. 1997. 519 ~ 526

43　Stiliadis D，Varma A. A general methodology for designing efficient traffic scheduling and shaping algorithm. In：IEEE INFOCOMM97. 1997. 326～335

44　Peterson L L，Davie B S. Computer Networks：A System Approach. 2nd ed. Morgan Kaufmann. 2000. 456～457

# 第11章

# QoS 路由

路由包含两个基本功能：一是搜集网络的状态信息并不断更新；二是根据所搜集的信息计算可行路径。QoS 的概念用来刻画服务提供者与用户间用数量或质量来定义的性能约定，一次连接的服务质量由一系列约束条件给出，如带宽约束、时延约束、抖动约束等。QoS 路由（QoS routing）的基本任务是为一次连接寻找一条有足够资源、能满足 QoS 要求的可行路径。QoS 路由不同于尽力而为的（best effort）路由，因为 QoS 路由通常是面向连接、有资源预留功能，并且能够提供有质量保证的服务；而后者有可能是面向连接的，也可能是无连接的，根据当前可获得的资源的不同，服务质量方面也有所不同。

QoS 路由比尽力而为的路由要复杂得多，寻找有两个独立路径约束的可行路问题属于 NP 完全问题。近年来，QoS 路由逐渐受到研究者的关注，许多启发式算法被提出来，文献[1]是关于 QoS 路由算法的一个综述。

路由算法的设计目标通常包括以下内容。

（1）最优化：是指路由算法选择最佳路径位置的能力。不同的优化目标及其权值大小决定了选择不同的最佳路径，例如路由算法可能考虑节点数和预留带宽，路由协议必须严格地定义它们待优化的度量标准。

（2）简单性：路由算法应被设计成尽可能地简单，即必须以最少的开销和使用费用获得高效的功能。当路由算法由软件实现、并在物理资源受限制的计算机上运行时，效率显得特别重要。

（3）健壮性：路由算法必须是健壮的，在异常的或者无法预料的情况下（如硬件失败、高负载条件和不正确的安装和使用等），要求算法仍能正确运行。因为路由器处于网络连接点位置，一旦出现故障时它们会导致整个网络产生严重的问题。最好的路由算法应经得住时间的考验，在各种网络条件下仍能保证稳定可靠地运行。

（4）快速收敛性：路由算法必须在短时间内收敛。收敛是指所有的路由器关于最佳路由能获得一致的过程，当一个网络事件使得路由过程失败或成功时，路由器发送路由更新消息。路由更新消息扩散至整个网络，导致重新计算最佳路由，并最终使所有的路由器一致接受这些路由。路由算法收敛过慢会产生路由循环或网络损耗。

（5）灵活性：路由算法应迅速准确地适应各种各样的网络情况。例如，假定一段网络失效，多数路由算法一旦监测到该问题，则要求很快地为使用该段网络的路由选择次优

的路径。路由算法应被设计成能够适应网络状态变化，如网络带宽、路由器队列大小、网络延迟或其他链路状态信息。

## 11.1　基本路由算法

### 11.1.1　路由算法概述

在传统的数据网中，由于没有建立连接电路，传送的每个分组携带着传送目的地址以及到达终点的各中间节点的路由表信息。构造路由表可以使用集中式算法（最典型的是 Dijkstra 算法）或者分布式算法（如距离矢量算法和链路状态算法）。由于节点或链路失效以及有新的节点或链路动态加入时，可能导致网络拓扑的变化，集中式算法不能处理这种情况。分布式算法知道网络拓扑的改变，根据更新的拓扑信息来计算路由，因而更具有动态性。

可以根据不同情况对路由算法进行分类：静态或动态；单路或多路；平坦的或分级的；主机智能或路由器智能；域内或域间；链路状态或距离矢量。

1. 静态或动态路由

静态路由算法的过程相对简单，静态路由表是网络管理员在路由运行之前建立的，只有网络管理员才能改变路由表目。静态路由算法设计简单，适于工作在网络运行相对可预测、网络设计相对简单的环境。因为静态路由系统不能对网络变化做出反应，所以一般被认为不适用于大型的、不断改变的网络。20 世纪 90 年代大多数路由算法皆为动态的。

动态路由算法靠分析收到的路由更新消息来实时自调整，以适应变化的网络情况。如果消息显示相关网络发生了改变，路由软件就重新计算路由，向外发送新的路由更新消息。这些信息扩散至整个网络，引发路由器重新运行它们的算法来调整相应的路由表。动态路由算法可作为静态路由的适当补充。

2. 单路或多路路由

一些复杂的路由协议支持到相同目的地的多条路径，这些多路算法允许可多路复用，单路算法则只选择一条路由。多路算法的优点是能够提供较好的网络负载均衡，合理地使用网络资源。

3. 平坦的或分级的路由

路由算法可以运行于平坦的空间，或者使用路由分层。在平坦的路由系统中，所有的路由器相互平等；在分级路由选择系统中，一些路由器构成路由主干（backbone）。包从非主干（nonbackbone）路由器发送到主干路由器，再沿主干发送，直至到达目的地所在区域，然后从最后的主干路由器通过一个或多个非主干路由器到达最终目的地。路由系统通常指定节点的逻辑组，称为域（domain）、自治系统（autonomous system，AS）或者区域（area）。在非常大的网络中会存在附加的分级层，最高分级级别的路由器构成路由主干。分级路

由选择的主要优点是模仿了多数公司的组织形式,因而会很好地支持他们的通信形式。多数网络通信产生于小工作组(域)的内部,域内路由器只需知道域内部的其他路由器,因此它们的路由算法能够得以简化,路由更新导致的通信量也相应地被减少。

4. 主机智能/路由器智能路由

源端路由(source-routing)由源端节点决定整个路径,路由器仅仅担任存储转发设备的功能,简单地发送包到下一个节点,此时主机拥有路由智能。如果主机对路由一无所知,算法要求路由器自己计算并决定路径,则路由器拥有路由智能。主机智能系统更多地选中较好的路由,因为它们通常在真正发出包之前就搜索出所有可能的路由,然后选择系统定义的最佳路径。

5. 域内或域间路由

域内路由算法仅工作在域内部,域间路由算法则工作在域内部和域之间。最佳的域内路由算法并不必然是最佳的域间路由算法。

6. 链路状态或距离矢量路由

链路状态算法(也称为短路径优先算法)向网络的所有节点散发路由信息,各路由器仅发送描绘它自己连接状态的那部分路由表。距离矢量算法(也称为 Bellman-Ford 算法)要求各路由器发送它整个的路由表格或者其中一部分,但只发送到相邻的路由器。实质上,链路状态算法向各处发送短小的更新,距离矢量算法仅向附近的路由器发送较大的更新。链路状态算法较距离矢量算法收敛得更快,不容易产生路由循环,但链路状态算法比距离矢量算法需要更强的 CPU 处理能力和更多的存储能力,其实现和支持更昂贵。两种算法类型在大多数情况下都工作得很好。

路由算法使用许多不同的度量标准来决定最佳路径,一个完善的路由算法能够综合多个度量标准进行路径选择。一些常用的路由优化度量标准包括路径长度(path length)、可信度(reliability)、时延(delay)、带宽(bandwidth)、负载(load)、通信代价(communication cost)等。

路径长度是最常用的路由度量标准。通常,路由协议允许网络管理员给各网络连接赋以任意的耗散值,则路径长度为传输路径上各连接耗散值之和。有的路由协议定义路径长度为节点计数,亦即包从源到达目的地途中经过的网络设备(比如路由器)的数目。

可信度在路由算法中是指网络连接可信度(通常以误码率表征)。某些网络连接可能比其他出故障概率要大,或者一些网络连接可能比其他连接更容易或更快地被修复。很多可信度因素影响着可信度级别,可信度级别通常由网络管理员指定给网络连接。

路由时延是指包从源到达目的地所要求的时间。时延取决于多种因素,包括中介网络连接的带宽、路径上各路由器端口的包队列、中介网络连接的堵塞以及传输的物理距离等。时延是一种常用和有效的度量标准。

带宽是指链接可用的通信容量。例如,在所有其他条件相同的情况下,10Mbps 以太网链接就比 64Kbps 专线具有更好的通信能力。尽管带宽表征链接可达到的最大传输能

力,但并不意味着通过高带宽连接的路由一定就比通过较慢连接的路由要好。例如业务都去使用高带宽的链路,使之比较忙碌,则实际传送包所要求的时间可能比较慢的连接要长。

负载是指网络资源(如一个路由器)的繁忙程度。负载能够从各种各样的参数中计算得来,包括 CPU 利用率和每秒被处理的包的数目。对这些参数持续不断地监视本身也会加重资源负担。

通信代价是另一个重要的度量标准。一些公司可能关心运行开支更甚于关心性能。有时即使线路时延更长,它们也会通过自己的线路传送包,而不使用需计时付费的公共线路。

### 11.1.2　Dijkstra 最短路径算法

通信网络可以模型化为一个图形,其中顶点和边代表通信网络的节点和链路。通信链路的代价表示为图形中相关边的权值。Dijkstra 算法为集中式,需要完整的网络(图形)拓扑和链路(边)权值,以此来计算从一个指定源节点到网络中所有其他节点的最短路径。一个图形 G 表示为 G = (V, E),其中 V 和 E 分别表示图形的顶点和边。从顶点 u 到顶点 v 的边的权值表示为 w(u, v),须为非负的。一条边的权值度量方法可以包括跳数(即 w(u, v) = 1)、链路带宽、平均传输时延、链路排队时延或者它们的组合。算法使用一种权值度量方式来计算最短的路径。

Dijkstra 算法的步骤如下:

(1) 每个节点用从源节点沿已知最佳路径到本节点的距离来标注,标注分为临时性标注和永久性标注;

(2) 初始时,所有节点都为临时性标注,标注为无穷大;

(3) 将源节点标注为 0,且为永久性标注,并令其为工作节点;

(4) 检查与工作结点相邻的临时性结点,若该节点到工作节点的距离与工作节点的标注的和小于该节点的标注,则用新计算得到的和来重新标注该节点;

(5) 在整个图中查找具有最小值的临时性标注节点,将其变为永久性节点,并成为下一轮检查的工作节点;

(6) 重复第(4)、(5)步,直到目的节点成为工作节点。

Dijkstra 算法的标准流程如下。

Dijkstra(G, w, s)　G:图形,w:边的权值,s:源节点

begin

d[v] 表示到目前为止从 s 到 v 的最短路径的权值.

path[v] 表示在最短路径中 v 的前一节点.

$\forall v \in V$, d[v] = ∞, path[v] = nil;

d[s] = 0; S = ∅; T = V;

While (T ≠ ∅) do

　取 d[u] 为最小值的 u 为 T 中的一个顶点.

　T = T - {u}; S = S ∪ {u};

```
    对于 u 相邻的每个节点 v:
      If (d[v] > d[u] + w(u, v))
        d[v] = d[u] + w(u, v);
        path[v] = u;
end.
```

通过跟踪 path[v],可以获得从 s 到 v 的最短路径的顶点序列。d[v] 的最终值即为从 s 到 v 的最短路径的权值。

### 11.1.3 距离矢量路由算法

距离矢量路由也称为"旧 ARPANET 路由"或称"Bellman-Ford 路由",它让每个路由器维护一张表(即矢量),表中给出了到每个目的地已知的最佳距离和路线。在此类路由算法中,每个路由器维持有一张子网中每一个以其他路由器为索引的路由选择表,表中的每一个项目都对应于子网中的每个路由器。此表项包括两个部分,即希望使用的到达目的地的输出线路和估计到达目的地所需时间或距离。距离度量标准可为跳数、时延、路径排队长度或者类似的值。如果度量标准为跳数,则其距离为一跳 如果度量标准是队列长度,则路由器会简单地检查每个队列 如果度量标准为时延,则路由器可直接发送一个特别"响应"(ECHO)分组来测出时延,接收者只对它加上时间标记后就尽快返回。

距离矢量路由算法的工作流程如下:

(1) 路由器启动时,对路由表进行初始化,包含所有去往与本路由器直接相连的网络路径,这些网络路径不经过中间节点,初始路由表中各项的路径长度为 0。

(2) 各路由器周期性地向外广播其路由表报文,与某路由器 $R_j$ 直接相连的路由器 $R_i$ 收到 $R_j$ 的报文后,逐项检查报文内容,遇到下述表目之一,须对本地路由表的相应内容进行修改:

① 路由器 $R_j$ 列出的表目在 $R_i$ 路由表中没有,则 $R_i$ 路由表中增加相应表目,其信宿是 $R_j$ 表目中的信宿,距离为 $R_j$ 表目中的距离加 1,下一跳为 $R_j$。

② $R_j$ 去往某信宿的距离 $+1 < R_i$ 去往某信宿的距离,说明 $R_i$ 去往该信宿若经过 $R_j$,其路径会更短,则 $R_i$ 修改本表目,其信宿域不变,距离为 $R_j$ 中相同信宿表目的距离值 +1。

③ $R_i$ 去往某信宿的路径经过 $R_j$,而 $R_j$ 去往该信宿的路径发生如下变化:

(i) $R_j$ 的路由表中不再含有去往该信宿的表目,则删除 $R_i$ 路由表中相应表目;

(ii) $R_j$ 的路由表中去往某信宿的距离发生变化,则对 $R_i$ 中相应表目的距离项进行修改,以 $R_j$ 中距离 +1 取代。

距离矢量路由的主要问题是,网络拓扑改变时算法收敛慢以及无穷计算问题,所谓无穷计算是指节点强迫其他节点单调地增加它们的距离矢量。加速距离矢量收敛的一个流行方法是水平分裂(split horizon)算法,但水平分裂算法不能解决所有的无穷计算问题。

### 11.1.4 链路状态路由算法

每条链路具有一套有关 QoS 度量准则的被测量状态。如图 11.1.1 所示,链路状态是一个包含剩余带宽、时延和代价的三元组。节点也有状态信息,节点的状态信息可以单独

地测量出来，或者像图 11.1.1 那样合并到相邻链路的状态中。

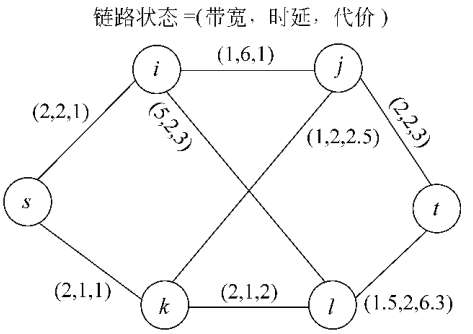



图 11.1.1　网络链路状态

链路状态路由算法的工作流程如下：

（1）每个路由器发现它的邻居节点，并知道其网络地址；

（2）测量到各邻居节点的延迟或者开销；

（3）组装一个分组以告之路由器刚知道的所有信息；

（4）将该分组发送给所有其他路由器；

（5）计算到每个其他路由器的最短路径。

现在各种各样的链路状态路由选择算法得到了广泛的应用，例如在 Internet 上越来越多地使用开放最短路径优先（open shortest path first，OSPF）协议就是使用链路状态算法的一个例子。

## 11.2　QoS 路由问题

### 11.2.1　QoS 路由问题的基本结论

在实时网络中，对于时间上的任意一点，很多信道建立的条件是活动的，其目标在于找到一条从各自的源到目的节点的符合要求的路径。路由发现中的路径选择属于典型的最短路径优化问题。优化的目标函数可以是一些参数，诸如跳数、代价、时延以及其他与被选路径上链路的某参数数值之和的度量标准。QoS 路由问题就是找到一条满足一个或多个 QoS 条件的路径。研究表明，寻找一条路径，使之满足两个或者多个加法和乘法组合的约束条件属于 NP 完全问题。

网络服务时被要求提供的 QoS，对于给定路径相对于其成分链路而言一般表现如下三类性质：①可加性：总 QoS 等于构成这条路径的所有链路 QoS 值之和（如跳数、时延等）；②可乘性：总 QoS 等于构成这条路径的所有链路 QoS 值之积（如误差率，丢包率等）；③最大最小性：总 QoS 等于构成这条路径的所有链路 QoS 值中的最小者（如费用等），或者等于构成这条路径的所有链路 QoS 值中的最大者（如流量、带宽等）。

由于要同时满足这些性质各异的 QoS 是比较复杂的，因此对于最小性 QoS，进行路径选择之前不满足 QoS 的链路将不作为路径选择对象；对于乘法性 QoS，可以将各链路的

QoS 值进行对数变换,转换为加法性 QoS,保证在进行路径选择时只包括加法性 QoS,以便于处理。

通常,分组级的 QoS 特征度量集中在分组丢失率、分组时延和时延抖动上,它们与预留带宽和网络的突发流量相关。一般来说,时延比其他指标更重要,通常,网络时延 = 分组处理时延 + 排队时延 + 传输时延 + 传播时延。前两个时延具有很大的不确定性,对于后两个时延,如果确定了端口和传输介质,则成为确定因素。在不确定因素中,排队时延是主要矛盾,它与网络节点的调度策略、端口处理速度、链路剩余带宽以及流的通信量特征有关。

一般可将网络应用大致分为 4 类:①非实时数据(或正文)传送;②实时图像传送;③实时声音传送;④视频会议传送。这种分类与 IETF 提出的"尽力而为型服务"、"质量保证型服务"和"可控负载型服务"是相对应的,可以找到某种映射关系。

第一类应用对数据的正确性要求较高,而不关心时间的快慢,其服务质量是确保信息的丢失率为 0。第二类应用对数据丢失不太敏感,对时延和抖动较敏感,由于此类应用一般数据量较大,常常对带宽要求较高,可以忽略短距离传播时延,因此对这种应用应优先考虑满足时延要求,在多个可行路径中尽量选择具有最大瓶颈带宽的路径。第三类应用对数据丢失和时延较敏感,但若是双向实时通话,则数据丢失可以通过实时交互进行重发。由于声音传输对带宽要求并不高,因此在找到满足时延约束的可行路径集之后,可选择其中跳数最小或者瓶颈带宽最小的路径,以减少资源的占用。第四类应用对服务质量要求最高,因为声音和图像要求同步,因此比较强调时延抖动,其次是时延、带宽和丢失率,可将时延抖动作为第一 QoS 指标,结合带宽来选择最佳路径。

在通信网络中,使用 Dijkstra 和 Bellman-Ford 算法计算最短路是很有效的,但如果要求满足不同的 QoS 条件、将约束引入优化问题,则算法会变得十分复杂。通常约束条件分成两类:链路约束(link constraint)和路径约束(path constraint)。链路约束是指一条路径上链路的使用限制,例如可得到的链路容量(如带宽)必须大于或者等于呼叫所需的要求。路径约束是指在选定路径上性能度量标准值的加性或乘性组合的界限,例如路径提供的端到端时延不能超过呼叫所能容忍的范围。路径约束使得路由问题变得难以处理,例如寻找一条路径使其满足两个独立的路径约束条件是一个 NP 完全问题。因此通常使用启发式方法来解决 QoS 路由问题。

目前,多路路由是 QoS 路由算法的一个发展趋势。当网络流量很少、网络资源总有剩余时,多路路由的首要任务是平衡网络负载以便更好地利用网络资源。平衡的流量分布将增加未来连接请求的连通率,并且使无须 QoS 保证的数据流有更好的回应(response)时间。当网络负载较重且不断变化时,多路路由可增加资源竞争情况下的连通率。多路路由算法主要有三类:第一类算法是利用回绕(crankback)来依次搜索多条路径,当被选择的路径不能满足 QoS 要求时,寻路过程返回到源点,再搜索下一条路径。该方法能适应网络的动态性,但需要较长的连接建立时间。另一类算法,即所谓的并行多路路由方法被用来克服这一缺陷,其路由信息并行地沿多路出发,沿途预留资源。如果多于一个寻路信息到达目的地,则最好的一条或者其寻路信息最先到达的路径被选定,其他路上预留的资源被释放,但目前尚没有标准来规定如何选择并行路径。第三类多路路由

算法是为每次连接寻找多条路径而不仅仅是一条路径，当存在的可行路径没有足够的资源时，算法试图使用其他路径来替代以满足通信要求。多路路由存在的问题是沿多条路径传送连续数据（如语音、视频）将会使信息的同步问题变得复杂。另外，在接收端需要更多缓冲空间来减少因路径不同而导致的时延抖动。

### 11.2.2　QoS 路由算法的主要特征

对于一个在实际中运行效果良好的 QoS 路由算法，除了考虑路由的优化方面以外，还要考虑其他问题，如整个网络的性能、路由表信息过时的可能性、链路参数的频繁变化以及信道建立期间的资源预留。

一般认为，路由算法在用于广域网实时信道建立的路由选择时，必须具备下列特征[2]：

（1）算法必须在不牺牲任何呼叫所需条件的前提下，使整个网络性能最大化；

（2）算法在实现路由策略时必须设计成能够保证资源预留；

（3）算法必须在几乎不具备全局状态信息的情况下能够正常运行；

（4）算法必须适应链路状态（如链路时延和可获得带宽）的变化；

（5）算法必须能够优化 QoS 路由所需的多个约束条件。

QoS 路由选择有两个基本目标：

（1）所选择的路径必须是满足 QoS 约束的可行路径；

（2）所选择的路径必须尽量有效地使用网络资源并使网络资源利用率最大。

QoS 路由问题一般由三个部分组成：

（1）获得满足应用 QoS 请求所必须的 QoS 路由计算信息；

（2）建立一条满足 QoS 请求的路径；

（3）维护已建立的路径。

对于第（1）部分，可以扩充现有的链路状态协议（如 OSPF），目前的 OSPF 协议将 QoS 编码位扩展为 5 位，分别指示路由度量指标为费用代价、可靠性、带宽和时延，全‘0’表示无特殊要求。常用的路由算法对于第（2）部分都使用基于 Dijkstra 或者 Bellman-Ford 的最短路径算法，以带宽、传播时延和跳数为主要度量准则。目前尚无较好的方法来解决问题（3），一般认为，RSVP 可以充当 QoS 路由请求和路由维护的信令协议，采用预计算方式周期性地计算几种类型的路由。

作为网络传输 QoS 体系的重要组成之一，QoS 路由选择有两种应用背景，一种是为流量工程，另一种是为动态请求。对于前者来说，其操作主要以长程的通信量变化为基础，以聚集流为处理对象，考虑粗粒度的性能需求，此时的 QoS 路由选择的目标是在缓慢改变通信量模式的情形下获得最大的网络整体性能（如普遍地减少时延）。这要求通过持续地测量通信量的图谱来计算流量聚集的路径，优化各种性能的测度。对于后者来说，QoS 路由是为每个请求而计算的，这些请求被显式地表达成资源的需求，路由计算更加频繁，资源分配的粒度更小。因此，这种背景下的 QoS 路由选择的目标是满足单个请求的性能约束，即为了提供有力的保证，不仅在更新网络状态和路由计算上有更大的开销，而且还要为此付出附加的信令开销。

### 11.2.3 QoS 路由的性能度量标准

在实时通信中,传统的度量标准,如平均报文时延和单个链接的路由距离并没有实质意义。因为这些标准没有对收到报文的及时性做出描述,因此不能应用到面向连接的实时服务中。对于一个可接受的呼叫请求 R,定义函数如下:

- accepted(R) = 1
- cost(R) = 对于 R 所选路径的代价
- setup(R) = 呼叫建立分组访问过的顶点数目
- dist(R) = 对于 R 选择的路径长度(根据跳数)

对于一个被拒绝呼叫请求 R,所有的函数值为 0。假设产生的整个呼叫请求数目为 N,使用下列量化的性能评价指标可以分析 QoS 路由算法的性能和优化实时网络的整体性能:

(1) 平均呼叫接收率(average call acceptance rate, ACAR):实时信道建立所接受的请求所占百分率

$$ACAR = \frac{\sum_{i=1}^{n} accepted(R)}{N}$$

(2) 平均呼叫建立时间(average call setup time, ACST):建立一个实时信道所需的平均时间,度量方法是根据呼叫建立分组访问的顶点数目

$$ACST = \frac{\sum_{i=1}^{n} setup(R)}{\sum_{i=1}^{N} accepted(R)}$$

(3) 平均路由距离(average routing distance, ARD):所建立信道的平均跳数

$$ARD = \frac{\sum_{i=1}^{N} dist(R)}{\sum_{i=1}^{N} accepted(R)}$$

(4) 平均代价(average cost, AC):所建立信道的平均代价

$$AC = \frac{\sum_{i=1}^{N} cost(R)}{\sum_{i=1}^{N} accepted(R)}$$

平均呼叫接收率是最重要的指标,因为它衡量了对呼叫的总处理能力。在实时、交互式多媒体应用和分布式实时系统的负载均衡算法中,平均呼叫建立时间的指标很重要,因为它们要求快速的信道建立。平均路由距离是不可缺少的指标,因为短的路由具有更少的开销。平均代价指标也很重要,因为追求最小的代价是 QoS 路由算法的关键目标之一。

## 11.3　路由选择方法

路由选择有两种方法：集中式和分布式。

### 11.3.1　集中式路由选择方法

集中式路由选择方法假设存在一个全局网络管理器来维护所有已建立信道和网络拓扑的信息，因此能够对每个实时信道请求选择一条合适的路由。在集中式方法中，每个实时信道请求必须经过网络管理器的获准。虽然这种方法可以比分布式方法更能有效地选择符合要求的路由，但是由于很难得到完整的网络状态信息以及集中控制的本质，导致在性能和可行性方面比较差。另外，集中式方法不具有可扩展性。

### 11.3.2　分布式路由选择方法

相对于集中式方法来说，分布式路由方法具有更好的性能、可扩展性和可靠性。由于在源和目的之间可能存在的路由数目很多，对于实时信道来说，选择一条合格的路由并不容易。任何路由算法的目标都是利用最小的操作开销来找到合格的路径。最近，研究人员对该问题进行了研究，提出了一些启发式路由算法。目前主要的分布式路由算法可分为两大类：洪泛（flooding）法和优选邻居（preferred neighbor）法。

1. 洪泛法

基于洪泛的算法属于静态路由算法，其基本思想是将收到的所有包，向除了该包到来的线路以外的所有输出线路发送。该算法以牺牲平均呼叫接收率为代价，在平均呼叫建立时间和平均路由距离方面具有优势。导致平均呼叫建立时间和平均路由距离值很小的原因在于，洪泛没有沿原路返回。存在的问题是洪泛要产生大量重复包，有可能是无穷多个分组。一种解决措施是，每个包头包含站点计数器，每经过一站计数器减 1，为 0 时则丢弃该包；另一种方法是记录包经过的路径，防止它第二次再发送到已发送的路径中。

洪泛法的一个改进版本是选择性洪泛法（selective flooding），它不将每一个进来的分组从每一条输出线路上发出，而是仅发送到与正确方向接近的线路。

洪泛法在很多应用中并未实际采用，但它仍然有些用途。例如，在军事应用中，大批的路由器随时都可能被炸毁得所剩无几，因此希望采用很可靠的洪泛方式。另外，洪泛法可作为一种标准来衡量其他路由选择算法的性能，因为洪泛并行地选择每一条可能的路径，最终产生最短路径。如果不考虑洪泛过程本身产生的开销，其他算法不会产生一个比它更短的延迟。

2. 优选邻居法

优选邻居路由方法基本上是使用基于返回的路由选择方法。一旦某节点收到一个呼叫建立或拒绝的分组时，它需要完成一系列操作。当节点 v 收到一个呼叫建立分组时，它

将该分组发送至最优先选择的链路。如果链路的另一端送回一个拒绝分组,则节点 v 将该分组发送至次最优先选择的链路,如此下去,直到呼叫分组到达终点为止,才表明该呼叫已成功建立链接。如果所有寻找成功链接的努力均告失败,则节点 v 向它收到呼叫建立分组的那个节点发送一个拒绝分组。

在优选邻居法中优选邻居的选择是基于启发式的,如最短路径优先(shortest path first, SPF)和轻负载链路优先(lightly loaded link first, UF)。相对于洪泛法来说,优选邻居法以牺牲 ACST 和 ARD 为代价,具有较高的 ACAR 值。优选邻居法具有较高的 ACAR 值是因为它仅沿一条路径预留资源,而洪泛法使用了多条路径;而具有较差的 ACST 值和 ARD 值的原因在于,它是沿原路返回的。

优选邻居法有两类:①基于局部/静态信息;②基于动态非局部信息。SPF 和 LIF 算法就属于前者,这两种启发式算法的处理开销相差无几,因为它们都使用局部链路信息或相对静态的全局信息。LIF 的优选邻居表是依据局部链路状态,而 SPF 则基于相对静态的网络拓扑。

## 11.4 分布式时延受限的路由算法

求解最小代价的时延受限路由问题属于典型的约束优化问题[3],时延受限在很多实时应用中具有普遍性。

### 11.4.1 网络模型

为了研究最小代价的时延受限路由问题,假设网络模型为一个无向图 $G=(V, E)$,其中 V 为节点集合,E 为互连的链路集合。对于每条 $e \in E$,定义如下 4 个函数:

时延函数 $D: E \rightarrow R^+$

代价函数 $C: E \rightarrow R^+$

整个带宽函数 $TB: E \rightarrow R^+$

可利用带宽函数 $AB: E \rightarrow R^+$

网络中的路径 $P=(v_0, v_1, v_2, \cdots, v_n)$ 具有两个相关的特征函数:

路径代价函数:$C(P) = \sum_{i=0}^{n-1} C(v_i, v_{i+1})$

路径时延函数:$D(P) = \sum_{i=0}^{n-1} D(v_i, v_{i+1})$

在静态网络模型中,对于每条 $e \in E$,其 $C(e)$,$D(e)$ 和 $TB(e)$ 值是固定的,$AB(e)$ 则随链路利用率的不同而发生变化。但在动态网络模型中,$C(e)$,$D(e)$ 也允许变化。当某链路 e 的参数 $C(e)$ 和 $D(e)$ 发生变化时,假设这种变动对于 e 的两端节点是透明的。做出这种假设是合理的,因为两端节点能够管理它们相连的链路状态和瞬时记录链路参数的变化。通常这种信息直接(如使用链路状态分组)或间接地(如执行分布式 Bellman-Ford 算法)传播到其他节点是需要时延的。

### 11.4.2　问题描述

网络中信道建立请求(或称呼叫)可表示为一个五元组：

$$R = (id, s, d, B, \Delta)$$

其中 id 为呼叫请求标识码，$s \in V$ 为呼叫的源节点，$d \in V$ 为呼叫的目的节点，B 为所需要的带宽，$\Delta$ 为必须满足的时延约束。设源节点 s 和目的节点 d 之间的路径表示为 $P = (v_0, v_1, v_2, \cdots, v_n = d)$，$P_{sd}$ 表示其所有路径的集合，则 P 满足以下两个条件：

$$\begin{cases} AB(e) \geqslant B, & \forall e = (v_i, v_{i+1}), 0 \leqslant i \leqslant n-1 \\ D(P) \leqslant \Delta \end{cases}$$

最小代价的时延约束路由问题可表示如下：

寻找如此 $P' \in P_{sd}$，使其满足 $C(P') = \min\{C(P) : P \in P_{sd}\}$。

## 11.5　Internet 路由协议

根据 Internet 具有的结构特征，可按照路由协议的作用范围将其分为两大类——内部网关协议和外部网关协议；根据路由算法所需满足的要求，可分为单播(unicast)、组播(multicast)和任播(anycast)路由协议三种；根据采用的路由调度策略，可分为“QoS 保证型”和“尽力而为型”的路由协议。

### 11.5.1　内部网关协议

内部网关协议主要包括 RIP(routing information protocol)，OSPF，IGRP(interior gateway routing protocol)，IS-IS(intermediate system-intermediate system)等。RIP 协议是距离矢量算法在局域网上的直接实现，RIP 将协议的参加者分为主动方(active machine)和被动方(passive machine)。主动方主动向外广播路由更新报文，被动方被动接收路由更新报文。通常网关为主动方，主机为被动方。网关周期性地向外广播一个(V,D)报文，报文信息来自本地路由表。报文中的距离 D 以站数来计算，与报宿直接相连的网关规定为一个跳数，相隔一个网关则增加一个跳数。一条路径的距离为从源端主机到目的主机经过的网关数。为防止出现路由环路，RIP 规定长度为 16 的路径为无限长路径，即不存在路由。该规定使 RIP 只适用于小规模的局域网。

开放最短路径优先协议 OSPF 是目前在 Internet 上最广泛使用的核心路由协议[4]，它将一个网络或一系列相邻的网络分为编号区域，一个区域的拓扑结构对于自治系统的其余部分是不可见的。这种信息的隐藏可带来路由信息量的显著降低，且域内路由只由域本身的拓扑结构决定，使其不受域外错误信息的影响。OSPF 定义了一个特殊的域，称为主干(backbone)，编号为 0，所有区域均与主干相连，主干负责向所有的非主干区域分发路由信息。由于域的引入，OSPF 要区分 4 类路由器：①域内路由器：与该路由器相连的所有网络均属于同一个域，该路由器只运行一套基本的路由算法；②域间路由器：连接两个或多个区域的区域边界路由器，域间路由器运行多种基本的路由算法，每种算法对应于与它相连的一个区域，域间路由器将与它们相连的域的拓扑结构信息发送至主干，由主

干将信息分发到各个域;③主干路由器:与主干有接口的路由器,包括所有的域边界路由器;④自治系统边界路由器:与其他自治系统交换路由信息的路由器,这些路由器向整个自治系统广播自治系统以外的路由信息。

IGRP 是由 Cisco 公司在 20 世纪 80 年代中期开发的,90 年代开发了增强版来提高 IGRP 的可操作性。IGRP 采用组合的路由度量方法,对网络时延、网络带宽、网络可靠性和负载均衡均有考虑,网络管理人员可为每种度量方法设置不同的权值,通常 IGRP 使用设定的或缺省的权值来计算最佳路由。IGRP 为每种度量方法提供很宽的取值范围,例如网络可靠性和网络负载可取 1 ~ 255 之间的任意值,网络带宽可取 1200bps ~ 10Gbps 之间的任意值。这种较宽的度量可以反映网络性能的微小变化。

中间系统到中间系统(IS-IS)协议是基于 DEC 公司的第五代 DECnet 所做的工作,尽管 IS-IS 最初是为了 OSI 的 CLNP 进行路由选择,但新版本的 IS-IS 支持 CLNP 和 IP 两种网络,该版本通常称为集成 IS-IS 或双 IS-IS。IS-IS 为链路状态路由协议,使用链路状态信息来刷新整个网络,从而建立一个完整的网络拓扑图。每个中间系统发送刷新信息给与之相邻的中间系统或终端系统,链路状态刷新消息中含有度量信息。相邻的系统收到刷新消息后向其相邻节点转发,直到遍布全网为止。

### 11.5.2 外部网关协议

外部网关协议或边界网关协议(border gateway protocol, BGP)是用于自治系统之间的路由协议,它们的主要功能是在各个已实现 BGP-4 协议的系统之间交换网络可达性信息。这些信息包括一个路由所穿越的自治系统的列表,用来建立一个表示连接状态的图。外部网关协议与内部网关协议的目的不同,所有内部网关协议的内存均是将分组尽量高效地从源端发送到目的端,不必考虑策略。但外部网关协议须考虑很多策略问题,典型的策略问题涉及政治、安全和经济方面的考虑。

BGP 路由器既保证提供可靠性通信,又隐藏所有网络的细节。BGP 基本上属于距离矢量协议,但与其他此类协议(如 RIP)有所不同,因为每个 BGP 路由器记录的是使用的确切路由,而不是到每个目的地的开销。另外,每个 BGP 路由器并不是周期地向它的每个邻居提供到每个可能目的地的信息,而是向邻居说明正在使用的确切路由,并且 BGP 很容易解决困扰其他距离矢量算法的无穷计算问题。

目前的路由协议大多是为每一个目的地址寻找一条最短路径,当数据包到达路由器时,路由器根据其包头的目的地址,将其沿路由表中给出的路径发送。在这种情况下,路由器实际上只是一个中转站,当最短路径阻塞或发生故障时,路由器只能缓存数据包以等待线路畅通 或者若路由表中给出了另一条路径,它将数据包沿此替代路径发送。

## 11.6 组播路由问题

QoS 组播路由技术是网络多媒体信息传输的关键技术之一,目前已有不少的研究成果。组播是一种可由源节点同时向多个目的节点发送信息的通信方式。随着计算机网络技术的发展,新兴的大量多媒体应用,如电视会议、远程教学等,均涉及多个用户参与,这

不仅需要消耗大量的网络资源，而且视频、音频等多媒体业务对网络服务质量也提出了更高的要求。由于基于多个不相关可加度量的 QoS 组播路由问题是 NP 完全问题，因此目前采用的方法多为启发式算法。

组播路由通常采用树型结构。因此一方面保证信息到不同信宿的并行传输，另一方面保证数据复制最少，从而减少冗余信息的传递并降低网络资源的消耗。目前，对于组播路由问题已经提出了多种算法。最常见的方式是将组播路由问题形式化为图论中的斯泰纳（Steiner）问题，通过求解斯泰纳最小树来求解代价最小的组播树。

## 11.6.1　组播路由问题的网络模型

在组播路由问题中，计算机网络表示为有向赋权图 $G=(V,E)$，其中 $V=(v_1, v_2, \cdots, v_N)$ 是图 G 中节点的集合，表示网络中的主机或路由器；$E=(e_1, e_2, \cdots, e_L)$ 是边的集合，每条边为连接网络节点的通信链路。E 上每条从 i 到 j 的边都定义了两个正实数加权值 $(C_{ij}, D_{ij})$，$C_{ij}$ 为由节点 i 到节点 j 传送信息的代价，反映链路资源的利用情况；$D_{ij}$ 为由 i 到 j 传送信息的时延，包括排队时延、传输时延和交换时延等。为了简化问题，这里假设节点 i 到 j 的边和节点 j 到 i 的边上的权值相等，即 $C_{ij}=C_{ji}$ 以及 $D_{ij}=D_{ji}$。

考虑一个源节点到多个目的节点的组播问题，假设信息由一个源节点 $s\in V$ 传送到一组目的节点集 $D\subseteq V-\{s\}$。组播树 $T=(V_T, E_T)$，其中 $V_T\subseteq V$，$E_T\subseteq E$，T 中存在由源节点 s 到每个目的节点 $d\in D$ 的通路 $P_T(s, d)$。一般代价和时延的定义如下：

- 组播树 T 的代价：$C(T)=\sum_{e\in E_T(s,d)} C(e)$
- 组播树 T 的时延为由源节点 s 到各目的节点路径时延的最大值，即

$$D(T)=\max\left(\sum_{e\in P_T(s,d)} D(e), d\in D\right)$$

如图 11.6.1 所示为组播通信结构框架的一个示例，其中包括一个源节点、两个接收节点和两个中间路由器节点。一个组播会话的相关事件按序执行如下：①初始化一个组播会话是先产生一个组播组；②构造组播分布树；③考虑资源预留；④开始数据传输；⑤会话时间结束，源节点启动会话拆除进程。

## 11.6.2　组播路由算法

组播路由中使用两种分布树[5]：① 最短路径树（shortest path tree，SPT）；②共享树（shared tree，ST）。最短路径树的方式针对于每个组播组的源（source-based）来处理生成树。基于源的分布树使用反向路径正向发送（reverse path forwarding，RPF）的机制，当某路由器接收一个带源的组播包后，它在除了接收该包的端口以外的所有端口转发该包，转发只发生在提供最短路径返回发送者的链路上，如果该包到达一个不是最短的路径上则被丢弃。在共享树方式中，所有的包沿着分布树发送到组播组而不考虑发送的源，这种方式减少了处理时间，但导致较大的端到端延时。不同的组播组定义不同的分布树，如果一个设备想接收数据，它必须加入这个组的共享树，组播路由协议通过检测一个点播的路由可达来建立分布树。

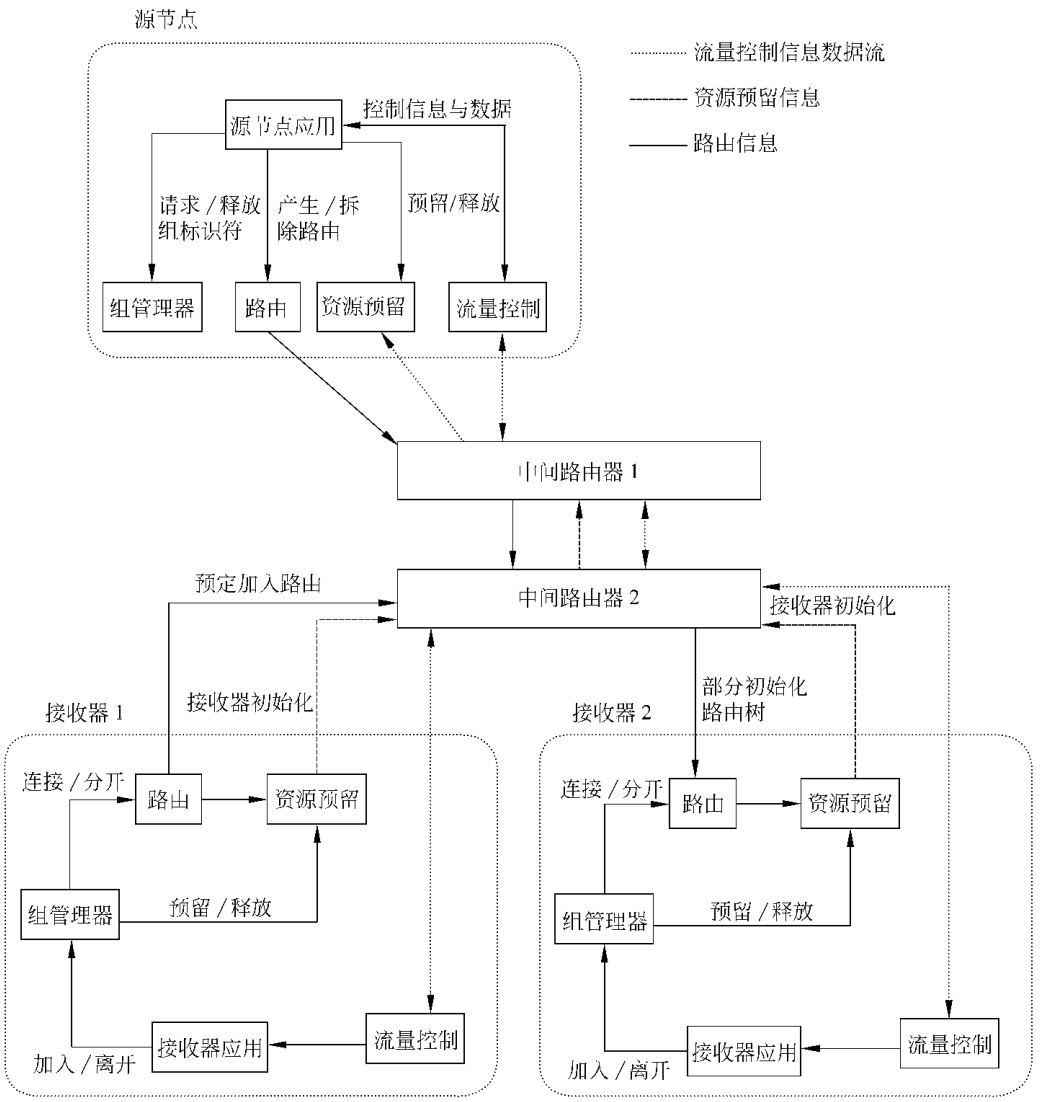



**图 11.6.1　组播通信的结构框图示例**

1. **最短路径树算法**

**构造 SPT 有多种算法,当前在 Internet 上常用的算法有以下两种。**

**(1) 反向路径正向发送(RPF)**

**RPF 是已经在 IP 组播中广泛应用的 SPT 路由算法,过程如下:当节点接收到一个分组后,检查接收到分组的端口是否是该节点到分组信源的最短路由上的端口。如果是,则该节点向该端口之外的所有端口转发此分组 否则抛弃该分组。RPF 假定信道对称,即按照最小时延的原则构造一棵生成树。**

（2）Dijkstra 算法

Dijkstra 算法的基本思想是每一次将距离信源最近的节点加入到树上，构造一棵生成树，然后将非成员树叶删除，因此要求信源具备所有信宿的地址信息。Dijkstra 算法实现简单，但算法集中执行，需要全网拓扑，一次性花费时间长，大量信源同时发送数据时 CPU 负载重。

2. 共享树算法

（1）网络中的斯泰纳问题（SPN）

使用斯泰纳路由算法的目的是使用最小化网络代价来构造一棵共享树。斯泰纳问题是指在一个无向图中，参照网络的边上定义的代价，在最小化网络代价前提下，寻找一棵树来覆盖组中的所有成员。SPN 属于图论中的 NPC 问题，即最优算法无法在多项式时间内完成。因此需要寻求有效的启发式算法来降低算法难度，而在性能上逼近理论最优算法。由于构造最小生成树（MST）相对简单，许多算法是将 SPN 问题降低为 MST 进行的，其中 KMB 算法因其简捷有效而著名。

KMB 算法过程简述如下：对于给定连通图 G = (V,E)，给定点集 D 来构造完全图 G' = (D, E')，其中 ∀ (u,v) ∈ E' 满足代价函数 $cost_G$ (u,v) 是 u,v 在 G 中的最短距离，u, v∈ D 在 G' 中构造最小生成树 T'，将 T' 中的边转换成 G 中的路径，得到问题的解。KMB 算法平均代价是在最优算法树代价 105% 以内。

（2）中心树（center-based trees，CBT）

CBT 路由策略是只寻找实际性能优良但并不遵从最优性准则。CBT 的基本思想是以选定中心为根，其他组成员按照最短路径原则与中心建立连接，构造成为一棵由所有发送节点共享的树。CBT 不具备广播特性，即数据只发向明确发出加入组请求的节点，避免了 RPF 类型算法运行在广域网上大量无效分组的扩散，因而适于接收者分布稀疏的网络情况。

## 11.6.3　组播路由协议

组播路由协议用于发现组播组和建立每个组播组的分布树，有两种方式：密集方式路由（dense-mode routing）和稀疏方式路由（sparse-mode routing），如何选用取决于组播组的成员在整个网络中的分布。如果网络中几乎所有的路由器都为每个组播组分发组播信息，则使用密集方式。密集方式为了维护分布树，每隔一段时间洪泛网络组播信息。密集方式适用于组成员密集地分布于整个网络，且有足够的带宽来容忍洪泛信息。密集方式路由协议包括：Distance Vector Multicast Routing Protocol（DVMRP），Multicast Open Shortest Path First（MOSPF），Protocol Independent Multicast Dense Mode（PIM DM）。

稀疏方式路由协议用于每个组播只有很少的路由器（并不意味着每个组播组只有很少的成员），组播组成员被广泛地分散，例如 Internet 上的组播。稀疏方式假设网络带宽很有限，因而不使用洪泛方式，开始先建立一个空的分布树，只有当成员请求加入组播组时，才向分布树添加分支。

1. 距离矢量组播路由协议

距离矢量组播路由协议(distance vector multicast routing protocol, DVMRP)是 Mbone 上广泛运用的组播路由协议,它是 RIP 的扩展。两者均采用距离向量算法,不同的是 RIP 直接根据指向信宿的路由表向发送数据;而 DVMRP 则采用 RPM 算法实现数据的转发。DVMRP 大多用于组播主干路由器,使用反路径洪泛(reverse path flooding)。当 DVMRP 接收一个包时,在它连接的所有路径上(除了接收路径)洪泛该包,该包可以到达所有的局域网,如果某个网段没有任何组播组的成员,则路由器发送一个削减信息返回分布树,使用削减信息可防止后来的包发送到此没有成员的区域。DVMRP 使用自己集成的路由协议去决定包返回源的路径,为了让新主机加入组播组,DVMRP 周期性地实施洪泛。DVMRP 的扩展性并不好,很少在大的网络中使用。

2. 组播开放式最短路由优先

组播开放式最短路由优先(multicast open shortest path first, MOSPF)是利用点到点的链路状态数据库,以 OSPF V2 为基础的组播路由协议。由于 OSPF 是一个链路状态路由协议,应用 Dijkstra 算法进行路由选择,MOSPF 把组播信息加入 OSPF 链路状态通告,因此每个节点都要保存全网的拓扑信息。在 OSPF/MOSPF 网络中,每个路由器基于链路状态信息维护详细的网络拓扑,MOSPF 路由器使用链路状态通告去学习在连接局域网中有某个组播组被激活,通过该信息来构造分布树。MOSPF 依赖于其集成的 OSPF,适用于单独的路由域,例如一个网络被一个单独的组织控制。它与 DVMRP 不同,不需要发送任何分组,节点可以根据全网链路状态表计算每个信源的 SPT。因此链路利用率比 DVMRP 高。MOSPF 可按需执行算法以便减少计算量,即只有当一个节点收到一个信源关于某个组的第一个分组时,才执行路由算法。这种做法的缺点是对第一个分组带来较大时延。该算法依赖于点到点路由协议,很难适用于广域互联网上的组播通信,须定期扩散大量的路由控制信息。MOSPF 基于包的源和目的地址来转发包,分布树当网络拓扑发生变化时被重新计算,MOSPF 不适用于不稳定的环境。

3. 协议无关的组播

协议无关的组播(protocol independent multicast, PIM)设计的出发点在于广域网范围之内同时支持共享树与 SPT,并能完成两者之间的灵活转换,因而集中两者的优点同时避免它们的缺点,在组员密集时以广播形式传送数据,再从树上删除不存在接收节点的分支;而在组员分布稀疏时,构造共享树传送,避免分组的广播开销。

PIM DM 和 DVMRP 都使用反向路径洪泛。当 PIM DM 接收一个包时,在它连接的所有路径上洪泛该包。如果某个网段没有任何组播组的成员,则路由器发送一个削减信息返回分布树。协议独立意味着它不依赖任何一个指定的点播路由协议,该原则适用于密集方式和稀疏方式。PIM 可使用所有的点播路由协议,适用于发送者和接收者的距离很近的场合,也适用于很少的发送者和很多的接收者以及流量很高的情况。

稀疏方式有两种组播路由协议:PIM SM(Protocol Independent Multicast Sparse Mode)

和 CBT(Core-Based Trees )。PIM SM 适用于只有较少的接收者以及流量较少的环境,该协议可同时处理几个组播数据流,非常适合应用于广域网和 Internet。该方式定义一个集合点(rendezvous point),一个发送者必须发送数据到此集合点,接收者在接收数据之前先要在集合点登记,路由器自动优化路径。PIM 可在某些组播组中使用密集方式的同时,在另外一些组中使用稀疏方式。

在 CBT 方式中,所有的组成员共享一个单独的树,组播流在相同的分布树上传输而不考虑源。CBT 与生成树相似,除了为每个组播组创建一个分离的树以外,基于核心的树可使用一个单独的路由器或一组路由器作为核心,路由器通过发送加入信息加入核心,核心发送一个确认返回路由器,加入信息无须被核心确认,此路由器就成为分布树的一个分支。

## 11.7　无线网络中的路由算法

前面所述均为有线网络中的路由算法。由于无线网络的拓扑结构时刻发生着变化,因此其路由算法相对复杂,下面介绍主要的几种。

### 11.7.1　自适应树型算法

自适应树型路由算法(STAR)是应用于在 ad hoc 网络中的基于连接状态信息的路由方法,它有两个优点:①充分利用带宽;②性能优于按需(on-demand)算法。STAR 的路由器根据自己到达目的地的路径连接状态发出更新信息,这些路径形成一棵生成树。每一个路由器根据连接状态和由邻居报告的源树共同形成对该路由器为已知的部分网络拓扑结构,同时 STAR 不像层次式路由需要骨干网。另外,其他路由算法需要路由器准确地通知正在连接的邻居和停止连接的邻居,而 STAR 只需更新源树的部分,因为每一个目的地只有一个先辈(predecessor),路由器只需更新部分链接和惟一的子树的表项。

STAR 的网络模型可以看作是离散图 G = (V,E),V 是节点的集合,E 是链接节点的边。在网络中邻居的定义是节点之间可连接的(link-level)。每个路由器向自己的邻居报告与其相邻的链路状态,由这些针对于某个路由器的链路组成该路由器的生成树。一个路由器的相邻链路和由其邻居报告的生成树形成了网络部分拓扑图,路由器利用这个拓扑图来生成自己的生成树,每个路由器利用生成树的路由选择算法形成自己的路由表。当 STAR 的路由器发现新的链接或者丢失某个连接以及某连接的变化导致路由时延过大,则向邻居发送自己的生成树的更新信息。

优化路由和最小负荷优化路由是两种被 STAR 采用的路由信息更新算法。优化路由试图尽可能快地更新路由表,以提供优化的路由路径;而最小负荷优化路由则尽可能地减少由于更新路由而带来的网络中的负载。优化路由可提供满足各种业务要求的路径,如最小时延路径、最大带宽路径。

### 11.7.2 SP 和 DSDSP

无线网络中的路由效率取决于位置信息的传播时延,扩展最短路径算法(ESP)和动态依赖状态的最短路径(DSDSP)则是两种分别基于动态传输和状态依赖的路由算法。ESP 算法包含三个部分:呼叫初始化、路径搜寻和网络控制。呼叫初始化在每一个源交换中心执行,是先到先服务机制(FCFS)。如果同时有多个呼叫请求到达,则交换中心根据其优先级排队。通常有三种呼叫等级:最高的是扇形切换呼叫,其次是非扇形切换呼叫,最低是非切换呼叫。若呼叫等级相同,则按照最小的 ID 来选择。路径搜寻也是在每个交换中心执行,主要过程包括非切换的路由路径搜寻和切换的重新路径搜寻。前者形成最小生成树,后者形成扩展的路径树。网络控制算法接收来自源基站、源交换中心或目的交换中心的消息。如果经从源基站收到消息,则送往源交换中心,此时呼叫初始化算法启动。同时网络控制检测呼叫的状态,如果呼叫结束,则释放占用该呼叫的信道。

DSDSP 的主要思想是通过最小生成树来搜寻路由路径,也包括呼叫初始化、优化路径搜寻和网络控制三个过程。DSDSP 在搜索扩展路径树时,它的范围覆盖整个网络,当网络半径很大时,搜索时间的开销远大于 ESP。研究结果表明,DSDSP 切换的丢失率比 ESP 小 20% 左右,但切换的平均建立时间比 ESP 要长。

### 11.7.3 PNNI

PNNI 是一种在无线 ATM 网络中常用的路由算法,可收集到网络时刻变化的拓扑信息。PNNI 将网络中的节点分成 PG(Peer Group)。每个 PG 有一个发布式的数据库来管理 PG 的有关信息,通过上下层的信息交换使得网络中的每个节点都知道网络的整个拓扑结构,因此单独一个节点即可计算出最短路由。

### 11.7.4 ZRP

ZRP(zone routing protocol)是将网络分成互连的邻居,称为 Zone,其中运行的是叫作“IntrAzone”的路由协议(IARP),而网络的其他部分运行的是“IntErzone”的路由协议(IERP)。如果分组的目的地在一个 Zone 中,则可直接送到其中的任意一个节点,否则由 IERP 建立并寻找该目的节点。

## 参考文献

1 Shigang C, Nahrstedt K. An overview of quality of service routing for next-generation high-speed networks: Problems and solutions. IEEE Network, 1998, 12(6): 64 ~ 79

2 Murthy C S R, Manimaran G. Resource Management in Real-Time Systems and Network. Cambridge, MA: MIT Press, 2001

3 Salama H F, Reeves D S, Viniotis Y. A distributed algorithm for delay-constrained unicast routing. In: Proc IEEE INFOCOM97, Vol. 1. Kobe, Japan, April, 1997. 84 ~ 91

4　Apostolopoulos G，Williams D，Kamat S，et al. Routing mechanisms and OSPF extensions. RFC 2676，August 1999

5　Salama H F，Reeves D S，Viniotis Y. Evaluation of multicast routing algorithms for real-time communication on high-speed networks. IEEE Journal on Selected Areas in Communications，1997，15(3)：332 ~ 345

# 第三部分

QUALITY OF SERVICE OF COMPUTER NETWORKS

# QoS 的性能评价与应用扩展

# 第12章

# QoS 控制的综合性能评价标准

## 12.1 概述

网络 QoS 控制的本质在于资源的管理，即控制缓冲队列、链路带宽等网络资源的分配和使用。QoS 控制算法的设计需要满足一定的性能要求。而系统性能的好坏是针对一定的性能评价标准而言的，应用不同的评价标准往往会导致不同的评价结果和 QoS 控制方案设计。因此性能评价标准对于分析（比较）现有方案、优化原有方案以及设计新方案都非常重要。在网络 QoS 控制策略和算法的设计中，性能评价的标准是一个关键问题。

随着网络技术的不断发展，各种新的网络应用对网络 QoS 控制策略的转发效率、带宽、延迟和丢失率等都提出了新的要求，如何同时满足这些要求是当前的研究难点。其困难在于：网络中存在资源和策略机制方面的限制，如网络带宽资源、处理器速度以及控制策略的设计是否合理等，而且吞吐率、分组延迟和丢失率等多种服务质量要求很难同时满足，有时候这些目标是相互抵触的，比如过分要求提高吞吐率可能导致分组延迟的大幅度增大，因而任何一种实际的网络 QoS 控制策略都只能对多个服务质量要求进行折中。

这就引出了一个问题，如何评价一种网络 QoS 控制策略，或者说资源分配机制，是否合理，一个网络是否有效而公平地分配了它的资源，更重要的问题在于，对单个性能目标（如吞吐率或者延迟等等）的评价和比较都是片面的，对网络 QoS 控制策略的性能评价应该针对多个性能目标，是综合性的，而这在目前的网络技术研究中还是开放问题。

对网络 QoS 控制进行综合性能评价的关键问题在于找到一种性能评价模型，该模型要能同时反映多个 QoS 目标，并体现其内在联系，进而通过对性能模型的研究提出对网络 QoS 控制策略进行综合性能评价的标准。

以前对网络 QoS 控制策略的性能研究工作主要集中在问题的某个方面，比如某一个性能目标的要求或者某些特定领域的综合性能研究。例如，描述吞吐率和延迟之间关系的 Power 公式[1]，但该公式只考虑了单链路缓冲队列的网络，同时假设缓冲队列可以无限长，这使得 Power 公式不能很好地评价网络 QoS 控制策略的有效性；文献[2]中研究了传统强迫优化算法（classical constrained optimization）和遗传算法（genetic algorithm）在吞吐率、公平性和时间复杂度方面的性能差异，作者随后提出了一种综合的折中方案，但它主要关注的是带宽分配。关于公平性，Raj Jain 提出了一个公平指数公式[1]，但该指标公式

假定公平就是所有用户均等地占用资源,而忽略了系统状态和用户需求。

到目前为止还缺乏有效的网络 QoS 控制的综合性能评价标准。在本章中,我们力图在这方面作一些理论上的探讨,提出了几种评价标准。标准 1 对 Power 公式作了改进,引入了多服务类的竞争机制,标准 2 引入比例公平性原则对用户要求的延迟、丢失率和资源分配的公平性进行综合评价,这两个标准由于其相互的独立性可以结合作为评价标准。标准 3 从服务质量、服务数量和系统公平性三个方面出发,有效地将系统吞吐率、时延、丢包率等指标统一在一起,并对时间尺度以及评价力度等问题进行了讨论。我们的研究工作[10,11]有效地综合了网络吞吐率、用户服务质量(QoS)要求以及系统公平性等多目标的性能要求,从而提供网络 QoS 控制策略选择和优化的量度和目标,这些工作也提供了对网络性能研究有益的理论探索。

## 12.2 网络 QoS 控制策略的性能目标

网络 QoS 控制性能研究中常考虑的主要目标有:

(1) 网络的整体效率性能:QoS 控制策略要考虑的一个重要性能要求是如何平衡负载、提高资源的利用率、避免拥塞,达到提高网络的整体效率性能的目的。

(2) 服务质量要求:一个好的 QoS 控制策略必须是能够满足应用要求,如应用对带宽、时延和丢失率等方面的要求。

(3) 公平性:是指不同应用之间的相关性,是否能够相互隔离,其指标是否能够在不同的流或者流聚类间保持服务的公平性,不会因为一个坏行为的流而影响其他流的性能。

这三个目标可以通过网络的一些性能参数来体现和衡量:①网络的整体效率性能可以用网络的吞吐率 T 来代表,要求有高的网络性能也就是希望网络节点的吞吐率尽可能地高;②应用服务的 QoS 要求包括带宽、延迟、丢失率和延迟抖动等,这里主要考虑两个一般采用的参数:延迟 D 和丢失率 L;③对公平性 F 的要求则要复杂一些,原因在于怎样才算公平并没有一般性的答案,比如有人认为所有网络流得到的带宽等资源相对平均就是公平,而有人认为应该根据负载情况进行资源的平均分配,在这里,我们借鉴比例区分模型[3]的思想,考虑用户的服务质量要求,在重负载情况下按 QoS 参数比例进行资源分配。

需要说明的一点是,之所以选取了延迟 D 和丢失率 L 作为 QoS 要求的参数,是因为我们注意到大部分应用的服务质量要求主要就是延迟和丢失率,例如实时应用以及多媒体网络应用等都希望网络的传输延迟和丢失率要尽可能地小。因而这里选取这两个参数作为评价参数。

这些个别的性能评价标准可以较好地评价和比较,但因为这些要求有时是互相抵触的,而对单个性能目标的量度和评价并不能全面地反映网络 QoS 控制策略的整体性能,因此人们希望能有一个综合的性能评价标准,可以对 QoS 控制策略多方面的性能目标进行综合评价。本章提出了一套综合性能评价标准,包括标准 1 和标准 2,它们只是这套综合评价标准的两个量度,可以结合起来共同作为评价标准。下面我们来详细讨论这两个标准以及它们的相对独立性。

## 12.3　综合性能评价标准 1：吞吐率 T + 延迟 D

网络的两个基本参数——吞吐率和延迟，可以作为研究分组调度策略有效性评价标准的一个很好的出发点。人们希望吞吐率越大越好，而延迟要尽可能地小。有人可能会觉得增加吞吐率的同时也会减小延迟，但事实上却不是如此：一个 QoS 控制策略增加吞吐率的当然做法就是让尽可能多的分组通过网络，让底层驱动达到较高的利用率，但在这种情况下，随着网络中分组数目的增加，每个路由器中等待队列的长度也会增加，而队列变长也就意味着等待延迟的增加。

为了描述吞吐率和延迟间的关系，有人提出了一个吞吐率和延迟的公式来评价资源分配策略的有效性[1]，这个公式称为网络的 Power 公式：

$$\text{Power} = \frac{\text{Throughtput}^{\alpha}}{\text{Delay}}, \quad 0 < a < 1 \tag{12.3.1}$$

但显然式(12.3.1)并不能很好地评价资源分配的有效性。一方面，式(12.3.1)的理论基础是 M/M/1 队列网络以及队列长度是无限的。另一方面，式(12.3.1)定义在单个连接上，不能应用在多个相互竞争的连接上。但由于以前没有很好的替代方法，所以式(12.3.1)仍然得到广泛的使用。

由于式(12.3.1)只考虑了 M/M/1 队列在缓冲区大小没有任何限制的情况(如图 12.3.1 所示)，而随着网络技术和应用的发展，像以前一样仅使用一种服务方式不再适应，不同的应用和用户对网络提供的服务有着不同的要求。在此情况下，可以把网络流按照它们对服务质量(QoS)的不同要求分类，这样就有了多个网络 QoS 服务类对带宽资源的竞争，针对式(12.3.1)的 M/M/1 模型，考虑用多服务类模型进行改进。

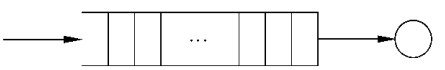


图 12.3.1　M/M/1 模型

多队列模型如图 12.3.2 所示。图 12.3.2 中每一个队列代表一个 QoS 服务类。同时不考虑缓冲区大小的影响，假设队列长度没有限制。

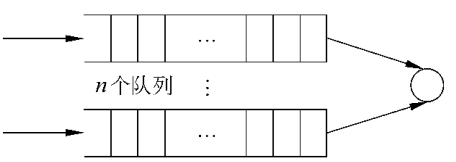



图 12.3.2　多队列 M/M/n 模型示意图

假定所有服务类分组长度分布相同且服务器持续工作，用 $\lambda_i$ 代表服务类 i 分组的平均到达速率，$\bar{d}_i$ 代表服务类 i 分组的平均排队延迟，$\lambda = \sum_{i=1}^{N} \lambda_i$ 代表系统总的平均到达速

率,则服务类 i 对系统平均延迟的贡献为$\frac{\lambda_i}{\lambda}\bar{d}_i$,系统总的平均延迟为

$$\bar{d}(\lambda) = \frac{1}{\lambda}\sum_{i=1}^{N}\lambda_i\bar{d}_i \tag{12.3.2}$$

则原 Power 公式变为

$$\text{Power} = \frac{\text{Throughput}^a}{\bar{d}(\lambda)} = \frac{(\sum T_i)^a}{\frac{1}{\lambda}\sum\lambda_i\bar{d}_i}, \quad 0 < a < 1 \tag{12.3.3}$$

从这个公式出发,要提高网络资源分配的有效性就要尽量让式(12.3.3)取得更大的值,而这是一个随网络负载变化的函数。在这里,负载大小是由分配策略决定的,图 12.3.3 描述了网络的 Power 曲线。理论上分配策略应该取得曲线的峰值。处于峰值左边的分配策略表现得过于谨慎,也就是说,还可以增加进入网络的负载以提高链路利用率;而对在右边的分配策略来说,过多的分组被允许进入网络而增加了队列延迟,吞吐率却没有显著提高。

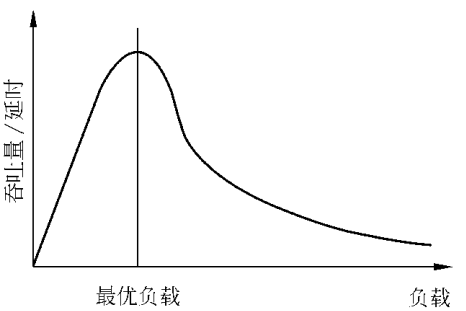



图 12.3.3 Power 公式示意图

有趣的是,Power 曲线看起来很像在多道程序计算机系统中的系统吞吐率曲线。系统吞吐率随着任务数的增加而不断提高,直到到达某个阈值(所有的时间片和内存页都在使用)时吞吐率才开始降低。

其实传输控制策略只可能粗略地控制负载,而不可能以很小的数量微调分组的变化。而且网络设计者关心,当网络系统的负载极度增大时的网络运行状况——也就是说,当处于图 12.3.3 曲线的极右边时,不希望因为系统的颠簸而使吞吐率下降为零。换句话说,我们希望系统是稳定的——分组即使在网络重负载的情况下仍能不断通过。若一种策略是不稳定的,则网络会出现拥塞崩溃现象[4]。

其实系统的分组丢失率和吞吐率之间的关系非常类似于上面延迟与吞吐率的关系情况,因此可以仿照延迟和吞吐率的式(12.3.3)得出分组丢失率的公式。采用与上面一样的所有服务类分组长度分布相同且服务器持续工作的假定,用 $\bar{l}_i$ 代表服务类 i 的分组丢失率,这时系统总的平均丢失率为

$$\bar{l}(\lambda) = \frac{1}{\lambda}\sum_{i=1}^{N}\lambda_i\bar{l}_i \tag{12.3.4}$$

得出另一个 Power 公式(注意,在无分组丢失时定义丢失率为一非零的极小值 $\varepsilon$):

$$\text{Power} = \frac{\text{Throughput}^a}{\bar{l}(\lambda)} = \frac{\left(\sum T_i\right)^a}{\frac{1}{\lambda}\sum \lambda_i \bar{l}_i}, \quad 0 < a < 1 \qquad (12.3.5)$$

可以发现，分组丢失率 Power 公式和上面关于延迟的 Power 公式十分类似。进一步的分析表明，这两个 Power 公式在性质上也十分相似，有着相近的曲线。

## 12.4　综合性能评价标准 2：QoS 要求 + 公平性 F

Internet 应用之间以及使用者之间有着非常不同的服务要求，这使得目前的同一服务模型在某些情况下存在着较大的局限性。在相对区分服务（relative differentiated services）[5] 中，网络流被组合成一个个的服务类（service class），这些服务类按照其分组转发质量要求进行排序以决定其排队延迟、分组丢失率等转发行为。假定网络流被分类组合成排序的服务类，类 i 应有更好于（或者至少不差于）类（i - 1）（1 < i < N）的服务质量（排队延迟和分组丢失率）。注意‘至少不低于’是必须的，因为在低负载的情况下所有分组的服务质量是相同的，即满足所有流的服务要求。这样，Internet 用户或者应用程序能够选择最符合它们要求的质量和价格限制的服务类。在这种情况下，由于没有准入控制和资源预留，Internet 应用程序和使用者不能获得绝对的服务质量保证，如端到端的延迟界限和带宽等，但网络能保证高级别的类享有比低级别类相对更好的服务。在文献[3]中，作者提出了一种比例区分模型（proportional differentiation model）来达到相对区分服务的要求。该比例区分模型着眼于提供网络管理者以与服务类负载无关的、调节服务类间的服务质量差别的方式，而这是在其他的相对区分服务模型（如严格优先级或者容量区分）中所不能解决的。

我们借鉴文献[3]中的思想，提出对网络 QoS 控制进行性能评价的比例公平性原则。正如前面所说，考虑到例如实时应用、多媒体应用等对服务质量的要求主要在于延迟和丢失率等 QoS 参数，因而在对公平性和服务质量要求的综合评价中采用延迟 D 和丢失率 L 作为评价参数。对于用户要求和系统公平性的考虑，分以下情况来分别加以讨论：

（1）在提供保证服务质量的条件下，如基于资源预留和准入控制的分组调度以及绝对区分服务[6] 的情况下，要求有 $\phi_i \geqslant \bar{d}_i$，其中 $\phi_i$ 为用户要求的延迟上界，在此代表用户的服务质量要求。这时由于各个服务类的服务质量要求均得到了满足，故不存在服务类间公平性问题。

（2）在低负载的相对区分服务的条件下，网络系统能够提供满足用户要求的网络资源，类似于（1）中的情况，不存在服务类间的公平性问题。而且在此情况下，从后面的分析讨论可知，比例公平性原则一般也是不可行的。

（3）在重负载的相对区分服务的情况下，由于网络资源不足，需要根据用户的要求进行合理分配，因此必须综合考虑 QoS 控制策略的满足用户要求和公平性问题。这里将主要考虑在重负载情况下网络 QoS 控制策略的满足用户要求和公平性的综合评价标准，下面分别从分组排队延迟和分组丢失率两个重要的服务质量参数角度进行讨论。

需要说明的一点是，这里考虑的是采用相对区分服务的网络在重负载情况下的公平

性和 QoS 要求问题,对于有资源预留和准入控制机制的情况,由于各网络流均得到了其预约的服务质量,也就是说,网络能够满足该流的服务质量要求,否则不允许该流进入,因而在此情况下不存在公平性的问题。

### 12.4.1 延迟 D + 公平 F

比例公平性原则按照网络管理者给定的区分参数按比例分配网络资源,从而得到相应的服务性能。若用 $q_i$ 代表服务类 i 的性能量值,则比例公平性原则给每对服务类加入如下的限制:

$$\frac{q_i}{q_j} = \frac{c_i}{c_j} \quad (i,j = 1,\cdots,N) \tag{12.4.1}$$

其中 $c_1 < c_2 < \cdots < c_N$ 是一般的服务质量区分参数。因此,即使每个类的服务质量随其负载发生变化,但类间的服务质量比值是不变的,与负载无关。

考虑基于排队延迟的公平性,采用排队延迟作为比例公平性原则的性能参数。最简单的服务类的延迟计量是在长时间段内的平均队列延迟,若用 $\bar{d}_i$ 代表服务类 i 分组的平均队列延迟,比例公平性原则可以表述为对所有服务类对 i 和 j 有

$$F = \frac{\bar{d}_i}{\delta_i} = \frac{\bar{d}_j}{\delta_j} \quad (i,j = 1,\cdots,N) \tag{12.4.2}$$

其中参数 $\{\delta_i\}$ 是用户要求的延迟区分参数(delay differentiation parameter,DDP),由于高级别类有更好的服务性能,故有 $\delta_1 > \delta_2 > \cdots > \delta_N > 0$。既然比例公平性原则独立于类的负载,因此在所有比例公平性原则可行的条件下可以有相同的定义。例如,网络管理者可以指定类 1 的平均延迟是类 2 平均延迟的两倍,而与平均延迟是几个分组的还是几百个分组的无关。关于比例公平性原则的可行性将在后面加以讨论。

我们还希望比例公平性原则不仅在长时间段的平均延迟下适用,在较短的时间片内也应有效。因为长时间段的平均值并不总是我们所需要的,特别是当网络流突发性很强时,或者说用户和应用程序数据流很短的情况下。为了说明这个问题,假定用户产生了一个很短的服务类 j 的分组流(例如一个 Web 请求),希望该流有比服务类 i 更小的延迟(i < j)。即使在长时间段内服务类 j 的平均延迟比服务类 i 的平均延迟要小,也可能在服务类 j 的分组流产生的短时间内其延迟反而比服务类 i 的大,这正是因为分组流的突发性造成的,而如果这种情况频繁发生,那么这种相对区分模型就是不一致和不可预测的了。对于短时间片内的比例公平性原则的表达式如下:用 $\bar{d}_i(t,t+\tau)$ 代表服务类 i 在时间间隔 $(t,t+\tau)$ 内的平均延迟,这里,$\tau > 0$ 是监测时间片。若在该时间间隔内没有分组,则 $\bar{d}_i(t,t+\tau)$ 作为未定义。在 $\bar{d}_i(t,t+\tau)$ 和 $\bar{d}_j(t,t+\tau)$ 都有定义的时间间隔 $(t,t+\tau)$ 里,比例公平性原则可表述为

$$F = \frac{\bar{d}_i(t,t+\tau)}{\delta_i} = \frac{\bar{d}_j(t,t+\tau)}{\delta_j} \tag{12.4.3}$$

**延迟比例公平性原则的可行性分析**

这里,我们进一步讨论在长时间段里延迟比例公平性原则的可行性条件。考察有 N

个队列(每个服务类一个)的无丢失的持续工作的传输控制策略。无丢失的假定需要系统运行在稳定状态,也就是说工作负载要小于系统的服务能力,否则队列长度会无限增加。这个条件可以通过一些拥塞控制机制来实现。更一般的情况应该是考虑丢失率,这将在后面加以讨论。服务器持续工作的假定也是非常重要的,因为非持续工作型的调度策略可能导致服务类间的延迟间隔为任意值,而在实际运行中,大部分也是持续工作型。

假定存在一种调度策略能够满足长时间段内的延迟比例公平性原则(见式(12.4.2)),同时,假定比例公平性原则是可行的。若分组长度分布在所有服务类间是相同的,则持续工作假定意味着:

$$\sum_{i=1}^{N} \lambda_i \bar{d}_i = \lambda \bar{d}(\lambda) \tag{12.4.4}$$

其中 $\bar{d}(\lambda)$ 是总的平均队列延迟,假定服务器采用持续工作的先来先服务(FCFS)法则。由 Little 定律可知,持续工作系统的平均等待队列与调度策略无关。对于分组长度分布一样的更一般的持续工作模型,可参见文献[7]。注意到,$\bar{d}(\lambda)$ 与流的行为(如突发性)密切相关,而这需要对每个链路的实际流行为作详尽的测量。

结合式(12.4.2)和式(12.4.4),服务类 i 的平均队列延迟为

$$\bar{d}_i = \frac{\delta_i \bar{d}(\lambda)}{\delta_1 \frac{\lambda_1}{\lambda} + \delta_2 \frac{\lambda_2}{\lambda} + \cdots + \delta_N \frac{\lambda_N}{\lambda}}, \quad (i = 1, \cdots, N) \tag{12.4.5}$$

由式(12.4.4)可知,延迟比例公平性原则有如下特性:

(1) 服务类 i 的平均延迟应该随着服务类 j 的平均到达速率的增加而增加。

(2) 增加高级别类的负载会比增加低级别类的负载更大地影响平均延迟的增加。

(3) 若一个服务类的延迟区分参数增大,则所有其他服务类的平均延迟应减小,而该服务类的平均延迟应增大。

(4) 假定把服务类 i 的一部分负载换到服务类 j,而总负载不变。若 $i < j$,则每类的平均延迟增加,反之则减少。

前面假定延迟比例公平性原则是可行的,也就是说存在持续工作的调度器能够实现式(12.4.2)中的限制。而显然这并不总是成立的,例如,在没有其他服务类的情况下,一个服务类的平均延迟不可能低于它在 FCFS 服务器中的平均延迟。假定类的到达速率为 $\{\lambda_i\}$,整个延迟为 $\bar{d}(\lambda)$,若存在持续工作型调度策略使每个服务类的平均延迟满足式(12.4.5),则可以说 DDPs $\{\delta_i\}$ 是可行的,这时,式(12.4.2)也成立。在给定负载情况下平均延迟可行性成立的充要条件在文献[8]中已有叙述,这里不再赘述。

注意到,可行性条件能用于实际,只有在通过监测特定链路状态就能够估计平均延迟 $\bar{d}(\sum_{i \in \phi} \lambda_i)$ 的情况下,这还是一个富有挑战性的开放式问题。

## 12.4.2　丢失率 L + 公平 F

我们研究如图 12.4.1 所示的 N 个服务类的网络传输控制模型。

每个逻辑分组队列对应着一个服务类。每一个到达分组根据它的分类标识进入这些队列中的一个。分组调度器(packet scheduler)决定哪个类将被服务,以获取类间必要的

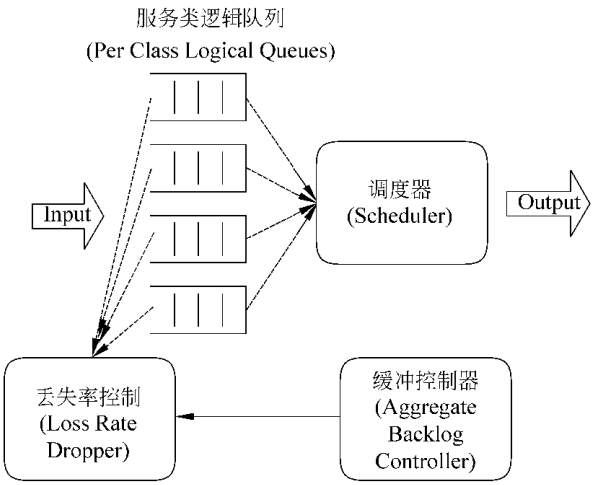



图 12.4.1 N 服务类网络传输控制模型

延迟区分。另一个模块,我们称其为缓冲控制器(backlog controller),管理在转发引擎中等待发送的分组的缓冲队列,它决定是否应该有分组被丢弃。最简单的缓冲控制器是尾部丢弃(drop-tail)的缓冲管理,当没有可用的缓冲资源时丢弃分组。更好的办法是可以采用如 RED[9] 的丢弃策略。当一个分组不得不被丢弃时,分组丢弃模块(packet dropper)选取一个缓冲队列。也就是说,缓冲控制器管理转发引擎整个的缓冲容量,而分组丢弃模块控制类间的分组丢失率区分。丢弃模块从相应队列的末尾移除一个分组。虽然从队列头丢弃分组可以增大分组的平均排队延迟,但在此不考虑这种方案。

比例公平性原则要求服务类的性能指标(如排队延迟和分组丢失率等)应该正比于相应的网络管理者设定的区分参数。通过这些区分参数,网络管理者能够控制服务类间相对的服务质量区别,比如基于服务价格或策略要求的区别。如前所述,假设用 $\bar{d}_i$ 代表无丢失的情况下离开分组的平均排队延迟,延迟比例公平性原则要求服务类间的平均延迟满足式(12.4.2)。同样,用 $\bar{l}_i$ 代表类 i 的平均丢失率,比如在长时间段内服务类 i 分组被丢弃的比例。基于分组丢失率的比例公平性原则要求服务类间的丢失率满足:

$$F = \frac{\bar{l}_i}{\sigma_i} = \frac{\bar{l}_j}{\sigma_j} \quad (i,j = 1,\cdots,N) \tag{12.4.6}$$

其中,$\sigma_i$ 是丢失率区分参数(loss rate differentiation parameter, LDP),它们的大小顺序为 $\sigma_1 > \sigma_2 > \cdots > \sigma_N > 0$。一般考虑高级别的类应该有比低级别类更好的延迟和丢失率性能,而不考虑两极化的要求,如更小的延迟但有更大丢失率的服务类,这在实际中也是很少遇到的。

在式(12.4.6)中,比例公平性原则表明丢失率比值 $\frac{\bar{l}_i}{\sigma_i}$ 应该是对所有服务类都是相等的,这样才说系统的 QoS 控制策略是公平的。比例丢失率模型的一个基本思想是为每类保留一个实时的丢失率估计 $l_i$,当需要一个分组被丢弃时,从缓存区的服务类中选取一个 $\frac{l_i}{\sigma_i}$ 最小者,然后从中丢弃一个分组,这样就减小了服务类间丢失率比值 $\frac{l_i}{\sigma_i}$ 的差距而使之趋

于相等。

存在两个重要问题。首先，丢失率估计值 $l_i$ 怎样定义和测量？其次，当要丢失分组时，$\frac{l_i}{\sigma_i}$最小的服务类没有待发分组怎么办？第一个问题涉及到时间计量的长度、缓冲区大小和算法复杂性等因素，而第二个问题涉及到丢失率比例公平性原则的可行性分析，如特定情况下的丢失率参数、缓冲区大小和负载情况等，这些因素都可能影响到一个服务类在应该丢弃分组时是否为空，因而会影响丢失率比例公平性原则的可行性。

下面对几个问题分别加以讨论。

1. 负载分布变化和不变的比较

对于不变或者变化较小的负载分布情况，无论是记录时间长还是短的算法都可以达到较好的丢失率比例公平性原则；而对于那些随时间变化较大的负载分布，只记录较短时间段内的负载情况，能较好地符合负载的变化作相应调整。相反，对于记录的是长期负载情况的算法，虽然负载分布发生了变化但其总的负载变化并不大。因此，对于负载变化剧烈的情况，记录时间短比记录时间长的算法有更好的适应性。

2. 时间记录长短对比例区分的偏差

对于短时间段内的比例丢失率区分来说，并不总能符合给定的比率，存在着比较明显的‘噪声’的影响。这种偏差出现的原因有两点，第一个原因与分组丢失的操作有关，而第二个原因则关系到比例公平性原则的可行性。在一个短时间段内，比如每 $T = 100000$ 个分组的测量，分组的突发性会使它的到达速率和长时间段内的平均速率有很大的偏差。对于记录时间比 T 长的分组丢失操作能够较少受突发的影响，而对于短时间段内的分组丢失操作来说则不可避免地存在与前者的较大偏差。若考虑可行性的影响，分组丢失策略不可能从空的服务类中丢弃分组，但有时这是为了获得指定的丢失率比例所应该进行的操作。如果在时间段 T 内的大部分分组丢弃操作发生在空服务类 i 中，那么就不可能得到服务类 i 和其他服务类的正常的丢失率比例。另一个可行性问题是，如果在时间段 T 内丢失率太小，则它不可能从每个服务类中丢弃足够的分组来保持丢失率的比例调节。在这些情况下，都可以通过增加观测时间 T 来减小偏差。

3. 内存长度的影响

由前所述，增加内存长度能使比例公平性原则的偏差更小。原因是，基于长时间段内分组到达的丢失率估计的鲁棒性更好，到达速率的突发性和丢失时间趋于缓和。我们还发现，在某个容错要求的情况下达到比例丢失率限制的内存长度随以下情况而有所增加：

（1）丢失率参数的比率增加；

（2）总的丢失率减小；

（3）服务类间的负载比率增加。

4. 丢失率比例公平性原则的可行性分析

若分组丢弃模块在要丢弃分组时发现该服务类没有缓存的分组，则它就不能达到所

要求的比例公平性原则。这种情形可能发生在多种情况下,这取决于特定的丢失率参数(LDP)、延迟区分参数(DDP)、平均缓冲区大小以及服务类负载分布等。下面用一些例子来分析一下这些参数的影响。第一种情形,假定高丢失率区分参数 LDP($\sigma_1/\sigma_2 = 64$)和较小差别的负载分布($\lambda_1/\lambda_2 = 10/90$),这时显然丢失率比例公平性原则是不可行的,因为低负载的服务类 1 没有足够的分组来满足类 1 和类 2 丢失率间的 64 倍的比率;另一种情形,假定负载分布同上例,但其延迟区分参数 DDP 更低($\delta_1/\delta_2 = 2$),这时,调度器不可能达到如此大的类 1 分组的延迟,因此类 1 将不能获得足够多的缓存分组数来达到要求的丢失率比例。相似的情形在 DDP 太高时也会发生,因为最高级别的服务类能很快得到服务,故没有足够的缓存分组数,这样就会导致丢失率的比例比要求的还要大;对于到达速率小于预定的情况,此时到达的分组很少的服务类就没有足够多的分组来满足丢失率比例公平性要求的丢失分组数;同样,当总的缓冲区数目减小时,必然使某些服务类的分组数为空的可能性增加,而我们知道,在某些服务类空闲时是无法达到要求的丢失率区分的。

在通常情况下,我们无法准确地知道丢失率比例公平性限制是否是可行的,因为它取决于上述的多种条件参数,因此在这里仅提供了一个可行性问题的讨论。

## 12.5 标准 1 和标准 2 的结合

前述的标准 1 和标准 2 一般是相互独立的,可以结合作为对 QoS 控制策略的评价标准,而不会相互制约和影响。以标准 1 和标准 2 中延迟的情况为例。

在前面讨论的 N 队列持续工作条件下,假定比例公平性原则是可行的,根据 Little 定律可知,持续工作系统的平均等待队列与策略无关。同时,在没有其他服务类的情况下,一个服务类的平均延迟不可能低于它在 FCFS 服务器中的平均延迟。由比例公平性原则的可行假定可知,对于给定的流类型,N 个平均延迟 $\{\bar{d}_i\}$ 可行的充要条件是下列的 $2^N - 2$ 个不等式成立:

$$\sum_{i\in\phi}\lambda_i\bar{d}_i \geqslant \left(\sum_{i\in\phi}\lambda_i\right)\bar{d}\left(\sum_{i\in\phi}\lambda_i\right), \quad \forall\phi \in \Phi \tag{12.5.1}$$

其中 $\Phi$ 是 $\{1,2,\cdots,N\}$ 的 $2^N - 2$ 个非空真子集的集合。$\bar{d}\left(\sum_{i\in\phi}\lambda_i\right)$ 是 $\phi \in \Phi$ 中全部服务类流在 FCFS 服务器中的整个的平均延迟。这些条件表明,N 个服务类的子集的平均待办服务量不会少于这些服务类流在 FCFS 服务器中的待办服务量,这与具体的网络 QoS 控制策略无关。

因此,在通常情况下,系统总的平均延迟与调度策略无关,也就是说,对标准 2 中比例公平性的要求不会影响到标准 1 中的系统平均延迟;同样,系统的吞吐率在持续工作的条件下只与负载情况和服务器性能等有关,而与实行延迟区分的具体调度策略没有多大关系。这就说明,标准 1 的要求和标准 2 的要求是相对独立的,可以结合作为网络 QoS 控制策略的评价标准,以对 QoS 控制策略作出更为全面的综合性能评价。

对标准 2 中的分组丢失率的讨论与此类似。由前面的讨论可知,系统总的平均丢失率与负载情况、服务器能力和缓冲资源大小等有关,而一般与具体的丢弃选择策略无关。

注意,这里没有考虑丢弃队列头分组会增大系统平均延迟的情况,因为这种情况在实际的 QoS 控制策略中一般是很少见的。

综上可以得出,标准 1 和标准 2 是相互独立的,应该结合作为对 QoS 控制策略的综合性能评价标准。这是因为标准 1 关注的是 QoS 控制策略对网络和系统整体性能和服务质量要求的影响,而标准 2 关注的是在多个相互竞争的服务类间的用户要求和公平性的满足。

## 12.6　综合性能评价标准 3

### 12.6.1　几个基本问题

#### 12.6.1.1　性能评价的多指标

评价计算机网络 QoS 控制系统的性能,包含有效性和公平性两个方面。从用户的角度来看,有效性是指端到端时延、时延抖动、分组丢失率等服务质量问题;从系统的角度来看,有效性是指吞吐率、接入的用户数等服务数量问题。服务质量和服务数量是矛盾的统一体,一般而言,当数量上升到一定程度时,质量就会有所下降。公平性也是 QoS 控制的一个重要性能指标。然而,对于公平性,不同的人有着不同的理解。有人认为,公平就是所有用户均等地占用资源;也有人认为,公平是相对而言的,与用户的需求和系统状态有关。本章定义了规格化性能函数这一概念,它包括规格化服务质量函数和规格化服务数量函数,并且认为用户间的公平性是指其规格化性能函数相等。

各种网络应用的服务要求是多种多样的。有的对时延和抖动有严格的要求而对分组丢失率的要求比较宽松,有的则相反。为了表示这些性能指标之间的关系,可以把它们组成一个多维空间,每一维分别代表一项性能指标。具有不同性能要求的应用对应着空间中的不同坐标(区域),如图 12.6.1 所示。

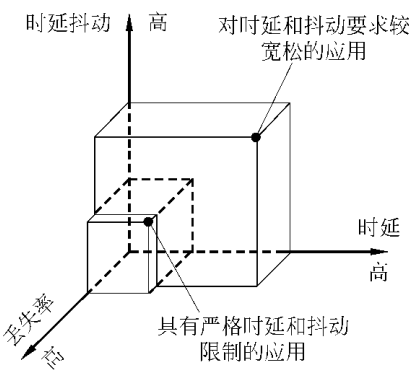



图 12.6.1　性能指标的多维空间

同时满足多个性能指标的要求是非常困难的,因为在某些情况下,有的指标之间可能存在着矛盾。比如,通过增加队列长度可以降低分组丢失率,但同时却可能增大了时延。因此,需要在具有矛盾性的指标之间寻求最佳的折中(tradeoff)。

#### 12.6.1.2 性能评价的时间尺度

进行系统性能评价,通常要选择一定的时间间隔 $\Delta t = t_2 - t_1$ 作为评价的时间尺度,而 $\Delta t$ 的大小会影响到评价结果。如果 $\Delta t$ 较大,评价结果代表了长期的平均性能(时间均值),可以反映相对稳定的变化趋势。如果 $\Delta t$ 很小,则其结果代表了瞬时性能,可以反映短期的变化状态。

只评价长期的平均性能或者短期的瞬时性能有时候并不能完整地反映系统的性能,尤其是当数据流存在突发性或者数据流是短连接时,系统的瞬时性能可能会偏离于平均性能而产生抖动。因此,需要同时考虑两个时间尺度。

#### 12.6.1.3 性能评价的粒度

QoS 控制一般基于一定的系统服务模型,而不同的服务模型有着不同的控制粒度。比如综合服务(IntServ)以流为对象,而区分服务(DiffServ)则以类(流的聚合,aggregate)为对象。本章所研究的性能评价标准不以特定的服务模型为基础,而是考虑了一般性,对流、类等分别加以讨论。

在这里,流(或者叫微流)是指一串具有相同源 IP 地址、目的 IP 地址、协议和传输层端口号的分组。对于单个流而言,一般有确定的起止时刻。

流的聚合——类:关于类,有两种描述方法。一是统计性描述,表现为随机过程;二是确定性描述,表现为一种"宏"流,即把某些具有相同特征(比如相同的时延要求)的流会聚在一起,组成更一般的"宏"流。前者通过一定的数学模型可以较为精确地描述数据流的行为特征,因而比较适用于理论研究,但在实际使用中却有些困难;而后者则是通过捕捉某些可测量的参数来描述数据流,因而更适合于实践应用。

类的集合:在多队列单服务器的模型系统中,多个队列需要竞争服务器资源。比如在输出排队的实际系统中,每个端口一般对应着多个逻辑队列,每个逻辑队列中存放一个或几个类的分组,这些类竞争链路带宽。对于这些类的调度将会影响系统的性能,比如分组在该端口的平均排队时延。

### 12.6.2 综合性能评价标准

#### 12.6.2.1 有效性的评价

1. 针对流的评价

设类 Class_i 中流 Flow_j 的某项服务质量指标的目标值为 $\psi_{i,j}$,在时刻 t 获得的实际值为 $\psi_{i,j}(t)$。

定义 12.1 如果 $\phi_{i,j}(t)$ 的值越大,则代表该项服务质量越好。定义规格化服务质量函数为

$$\varphi_{i,j}(t) = \frac{\psi_{i,j}(t)}{\Psi_{i,j}} \tag{12.6.1}$$

定义 12.2 如果 $\psi_{i,j}(t)$ 的值越小代表该项服务质量越好(比如时延和分组丢失率),

则定义规格化服务质量函数为

$$\varphi_{i,j}(t) = \frac{\psi_{i,j}^{-1}(t)}{\Psi_{i,j}^{-1}} \tag{12.6.2}$$

**定义 12.3**　在等式(12.6.2)中,如果 $\psi_{i,j}(t)=0$,则等式无意义。为此,定义规格化服务质量函数的界:如果 $\psi_{i,j}(t)=0$,或 $\varphi_{i,j}(t)>\Phi$,则令 $\varphi_{i,j}(t)=\Phi$。$\Phi$ 称为该项规格化服务质量函数的界。

$\varphi_{i,j}(t)$ 可以用来表示用户的满意程度。如果 $\varphi_{i,j}(t)<1$,则表示用户得到不满意的服务;若 $\varphi_{i,j}(t)=1$,则表示用户得到满意的服务;若 $\varphi_{i,j}(t)>1$,则表示用户得到非常满意的服务,比如某用户获得了比其保证的时延界(delay bound)更小的时延。

对于 n 项服务质量指标,有:

$$\varphi_{i,j}(t) = \sum_{k=1}^{n} w_k \times \varphi_{(i,j),k}(t) \tag{12.6.3}$$

其中 $w_k$ 为服务质量指标函数 $\varphi_k$ 的权重,且有 $\sum_{k=1}^{n} w_k = 1$。

**定义 12.4**　设系统服务数量的目标值和在时刻 t 获得的实际值分别为 $\Phi_{i,j}$和$\phi_{i,j}(t)$,则定义规格化服务数量函数为

$$\theta_{i,j}(t) = \frac{\phi_{i,j}(t)}{\Phi_{i,j}} \tag{12.6.4}$$

服务质量和服务数量是矛盾的统一体。虽然对规格化服务质量和规格化服务数量的期望是都能获得较高的值,但一般而言,当数量上升到一定程度时,质量就会有所下降。因此单独考虑服务质量或者服务数量都是不全面的,不能反映系统的整体性能。为此,这里定义规格化性能函数,把服务数量和服务质量统一在一起。

**定义 12.5**　类 Class_i 中流 Flow_j 在时刻 t 的规格化性能函数为

$$\delta_{i,j}(t) = \theta_{i,j}(t) \cdot \varphi_{i,j}(t) \tag{12.6.5}$$

如果服务质量代以时延 Delay,服务数量代以吞吐率 Throughput,则该规格化性能函数转化为 Power 公式[1]:

$$\delta(t) = \theta(t) \cdot \varphi(t) = \frac{\text{Throughput}(t)}{T} \cdot \frac{\text{Delay}^{-1}}{D^{-1}} = K\frac{\text{Throughput}(t)}{\text{Delay}(t)} \tag{12.6.6}$$

可见,Power 公式是这里规格化性能函数的一个特例。

当满足下面的条件时,认为在时刻 t,类 Class_i 中流 Flow_k 获得的综合性能优于流 Flow_j:

$$\delta_{i,k}(t) > \delta_{i,j}(t) \tag{12.6.7}$$

但以上反映的只是短期(或者说瞬时)性能,不够充分,还需要比较长期的平均性能(时间均值)。

设流的起止时刻分别为 $t_s$和 $t_e$,则类 Class_i 中流 Flow_j 的平均性能为

$$P_{\delta,(i,j)} = \frac{1}{t_e - t_s}\int_{t_s}^{t_e} \delta_{i,j}(t) \tag{12.6.8}$$

当满足下面的条件时,认为类 Class_i 中流 Flow_k 获得的综合平均性能优于流 Flow

_j：

$$P_{\delta,(i,k)} > P_{\delta,(i,j)} \tag{12.6.9}$$

2. 针对类的评价

如果把类描述为随机过程,则类 Class_i 的规格化性能函数可以表示为该类中所有流的规格化性能函数的统计均值(集均值)：

$$\delta_i(t) = \bar{\delta}_i(t) = E[\delta_{i,j}(t)] \tag{12.6.10}$$

如果把类看作是‘宏’流,则类 Class_i 的规格化服务质量函数和规格化服务数量函数分别为

$$\varphi_i(t) = \frac{\psi_i(t)}{\Psi_i} \text{ 或 } \varphi_i(t) = \frac{\psi_i^{-1}(t)}{\Psi_i^{-1}} \tag{12.6.11}$$

$$\theta_i(t) = \frac{\phi_i(t)}{\Phi_i} \tag{12.6.12}$$

其中 $\Psi_i$,$\Phi_i$,$\psi_i(t)$ 和 $\phi_i(t)$ 分别为类 Class_i 的服务质量和服务数量的目标值以及获得的实际值。

那么,规格化性能函数为

$$\delta_i(t) = \theta_i(t) \cdot \varphi_i(t) \tag{12.6.13}$$

当满足下面的条件时,认为在时刻 t,类 Class_k 获得的综合性能优于类 Class_j：

$$\delta_k(t) > \delta_j(t) \tag{12.6.14}$$

取时间间隔[t,t + Δt],则类 Class_i 的平均性能为

$$P_{\delta,i} = \frac{1}{\Delta t}\int_t^{t+\Delta t}\delta_i(t) \tag{12.6.15}$$

当满足下面的条件时,认为类 Class_k 获得的综合平均性能优于类 Class_j：

$$P_{\delta,k} > P_{\delta,j} \tag{12.6.16}$$

3. 针对类的集合的评价

采用与前面针对类的方法相似的方法,可以分析类的集合的性能。设其服务质量、服务数量函数和总体性能函数分别为 $\varphi(t)$,$\theta(t)$ 和 $\delta(t)$,则：

$$\delta(t) = \bar{\delta}(t) = E[\delta_i(t)] \tag{12.6.17}$$

或者由式(12.6.18)和式(12.6.19)得出式(12.6.20)：

$$\varphi(t) = \frac{\psi(t)}{\Psi} \text{ 或 } \varphi(t) = \frac{\psi^{-1}(t)}{\Psi^{-1}} \tag{12.6.18}$$

$$\theta(t) = \frac{\phi(t)}{\Phi} \tag{12.6.19}$$

$$\delta(t) = \theta(t) \cdot \varphi(t) \tag{12.6.20}$$

### 12.6.2.2 公平性的评价

本节从规格化性能函数这一概念出发来分析系统的公平性,认为规格化性能函数相等即为公平,此时也意味着用户的满意程度相等。

关于公平性的评价有两个层次：①把类（或类的集合）看作一个整体，公平性是该整体的特性之一，但不直接反映具体某几个流（或类）之间的公平性的好坏；②比较具体的某几个流（或类）之间的公平性。实际上，由于采用了规格化性能函数这一概念，使得第二层次的评价非常简单：当规格化性能函数相等时，两个流（或类）之间是公平的；否则不公平，而且规格化性能函数值较大的流（或类）获得更好的服务。这一点类似于文献[9]中提出的成比例公平的评价方法。因此，这里主要讨论第一层次的公平性问题。

1. 流之间的公平性

（1）方法 1：利用均值和方差

在随机过程中，方差表示随机变量的实际值对于均值（集均值）的偏离程度。因此，可以基于均值和方差的关系来研究公平性。

设类 Class_i 中流 Flow_j 的规格化服务质量函数、服务数量函数和总的性能函数分别为 $\varphi_{i,j}(t)$，$\theta_{i,j}(t)$ 和 $\delta_{i,j}(t)$，则类 Class_i 中所有流的均值（集均值）为式（12.6.10），方差为

$$\sigma_{\delta,i}(t) = \mathrm{Var}[\delta_{i,j}(t)] \tag{12.6.21}$$

则类 Class_i 中所有流的公平系数为

$$\eta_{\delta,i}(t) = \frac{\sigma_{\delta_i}(t)}{\bar{\delta}_i(t)} \tag{12.6.22}$$

$\eta_i(t)$ 值越小，表示类 Class_i 中所有流获得的性能差距越小，则公平性越好。

由于不同类型的应用有着各自适应的随机模型，而且难于验证一个数据流是否符合某种随机模型，因此在实践中计算均值和方差有一定的困难。这使得前面运用随机过程均值和方差间的关系来分析公平性的方法虽然适合于理论研究，但在实际使用中却存在一些困难。

下面将提出另外一种分析方法。与前者相比，该方法可基于系统测量，因而更适合于实践应用。

（2）方法 2：基于 Raj Jain 的公平指数

Raj Jain 提出了一个评价拥塞控制机制公平性的公式[1]。假定公平意味着所有用户均等地占用资源，又假定所有路径长度相等。若系统中有 n 个流，其吞吐率分别表示为 $(x_1, x_2, \cdots, x_n)$，则系统公平指数（fairness index）公式为

$$f(x_1, x_2, \cdots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2} \tag{12.6.23}$$

本节基于 Raj Jain 的公平指数公式，提出新的公平指数。根据我们提出的公平即为规格化性能函数值相等这一假定，得出类 Class_i 中所有流的公平指数为

$$f_{\delta,i}(\delta_{i,1}, \delta_{i,2}, \cdots, \delta_{i,n}) = \frac{\left(\sum_{j=1}^{n} \delta_{i,j}\right)^2}{n\sum_{j=1}^{n} \delta_{i,j}^2} \tag{12.6.24}$$

可以看出,当所有 $\delta_{i,j}$ 都相等时,

$$f_{\delta,i}(\delta_{i,1},\delta_{i,2},\cdots,\delta_{i,n}) = \frac{\left(\sum_{j=1}^{n}\delta\right)^2}{n\sum_{j=1}^{n}\delta^2} = 1 \tag{12.6.25}$$

表示类 Class_i 的公平性最好。而且 $f_{\delta,i}$($f_{\delta,i} \leqslant 1$)的值越小,表示公平性越差。

假设某个流的规格化性能函数变为 $\delta+\Delta$,$\Delta>0$,则:

$$f'_{\delta,i}(\delta_{i,1},\delta_{i,2},\cdots,\delta_{i,n}) = \frac{(n\delta+\Delta)^2}{n(n\delta^2+2\delta\Delta+\Delta^2)} = \frac{n^2\delta^2+2n\delta\Delta+\Delta^2}{n^2\delta^2+2n\delta\Delta+n\Delta^2} < 1 \tag{12.6.26}$$

表示类 Class_i 的公平性较差。

或者,假设某个流的规格化服务质量为 $\delta+\Delta$,而另外某个流的规格化服务质量为 $\delta-\Delta$,则

$$f''_{\delta,i}(\delta_{i,1},\delta_{i,2},\cdots,\delta_{i,n}) = \frac{n^2\delta^2}{n^2\delta^2+2n\Delta^2} < 1 \tag{12.6.27}$$

而且

$$\frac{n^2\delta^2}{n^2\delta^2+2n\Delta^2} < \frac{n^2\delta^2+2n\delta\Delta+\Delta^2}{n^2\delta^2+2n\delta\Delta+n\Delta^2} \tag{12.6.28}$$

即

$$f''_{\delta,i}(\delta_{i,1},\delta_{i,2},\cdots,\delta_{i,n}) < f_{\delta,i}(\delta_{i,1},\delta_{i,2},\cdots,\delta_{i,n}) \tag{12.6.29}$$

表示公平性更差。

以上反映的是短期(或者说瞬时)的公平性,下面分析长期(时间均值)的公平性。由于流一般都有比较短的持续时间,有确定的起止时刻,因此在较长的时间间隔[t,t+Δt]内,可能会不断地有旧的流结束,又有新的流产生。另外,所谓公平性,应该是指同时处于活动状态的流之间的公平性。因为只有同时处于活动状态的流,才会竞争资源。所以,不能直接把流的平均性能代入公平指数公式。而采取如下方法:

类 Class_i 在较长时间间隔[t,t+Δt]内的公平指数为

$$F_{\delta,i} = \frac{1}{\Delta t}\int_{t}^{t+\Delta t} f_{\delta,i}(\delta_{i,1},\delta_{i,2},\cdots,\delta_{i,n};\tau)\,d\tau \tag{12.6.30}$$

2. 类之间的公平性

(1) 方法 1:利用均值和方差

设类 Class_i 的规格化服务质量函数、服务数量函数和总的性能函数分别为 $\varphi_i(t)$,$\omega_i(t)$ 和 $\delta_i(t)$,则系统中所有类的均值(集均值)为式(12.6.17),方差为

$$\sigma_\delta(t) = \mathrm{Var}[\delta_i(t)] \tag{12.6.31}$$

则系统中所有类的公平性系数为

$$\eta_\delta(t) = \frac{\sigma_\delta(t)}{\bar{\delta}(t)} \tag{12.6.32}$$

$\eta(t)$ 值越小,表示系统中所有类获得的性能差距越小,则公平性越好。

（2）方法 2：基于 Raj Jain 的公平指数

系统中所有类的公平指数为

$$f_\delta(\delta_1,\delta_2,\cdots,\delta_n)=\frac{\left(\sum_{i=1}^{n}\delta_i\right)^2}{n\sum_{i=1}^{n}\delta_i^2} \tag{12.6.33}$$

当所有 $\delta_i$ 相等时，$f_\delta=1$，则系统公平性最好。而且 $f_\delta$（$f_\delta\leqslant 1$）值越小，公平性越差。

### 12.6.2.3　应用

本节所提出的综合性能评价标准可用于两个目的：①比较现有网络 QoS 控制方案的优劣，进行选择、优化或设计新的方案；②判断网络状态，更新系统配置，进行反馈控制和调节。

比如，利用与文献[12～14]中类似的方法，从很轻的网络负载开始，逐步提高注入网络的数据流量，测量并计算式（12.6.20）和式（12.6.33）的值。当式（12.6.20）到达峰值时，如果再继续增加流量，则网络会因为负载过重而导致性能下降。因此，式（12.6.20）处于峰值时的网络负载是最佳的，网络管理者可以根据式（12.6.20）的值的变化情况来控制网络负载。另外，如果式（12.6.33）的值接近于 1，则表明目前网络中各个数据类之间具有良好的公平性，即所获得的服务水平（满意程度）基本相等；反之，如果式（12.6.33）的值远小于 1，则表明目前网络中各个数据类之间的公平性较差。

当网络处于不良状态（比如负载过重，出现拥塞，公平性较差）时，可采取短期或长期的调节措施。作为短期措施，可以通过拒绝接纳新的流来限制网络流量；作为长期措施，可以更新系统配置，比如改变队列的数量或容量、改变控制策略、加入流量整形机制、改变队列管理算法或分组调整算法参数等。

总之，由于多目标性能研究的复杂性，目前还缺乏有效的网络 QoS 控制的综合性能评价标准，本章的研究是对这方面理论研究的有益探索，研究成果可广泛应用于对网络 QoS 控制策略的评价、选择和优化。网络 QoS 控制策略的综合性能评价标准中多个性能目标之间的相互影响和最佳的综合评价标准，以及相应的实施策略方面还有许多问题仍然是开放的，这些问题是值得进一步研究的方向。

## 参考文献


1　Peterson L L，Davie B S. Computer Networks：A System Approach. 2nd edition. Morgan Kaufmann Publisher，Inc，2000. 454～457

2　Pitsillides A，Stylianou G，et al. Bandwidth allocation for virtual paths（BAVP）：Investigation of performance of classical constrained and genetic algorithm based optimisation techniques. In：Proceedings of INFOCOM 2000. Tel Aviv，2000. 1379～1387

3　Dovrolis C，et al. Proportional differentiated services：Delay differentiation and packet scheduling. In：Proceedings of SIGCOMM99. September 1999

4　Jacobson V. Congestion avoidance and control. In：Proceedings of ACM SIGCOMM88. August 1988. 314

~329

5 Dovrolis C, Stiliadis D. Relative differentiated services in the internet: Issues and mechanisms. In: Proceedings of ACM SIGMETRICS 99. May 1999

6 Stoika I, Zhang H. LIRA: An approach for service differentiation in the Internet. In: Proceedings of NOSSDAV. 1998

7 Bolch G, Greiner S, et al. Queueing Networks and Markov Chains. John Wiley and Sons, 1999

8 Coffman E G, Mitrani I. A characterization of waiting time performance realizable by single-server queues. Operations Research, 1980, 28: 810~821

9 Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking, 1993, 1: 397~413

10 江勇,林闯,吴建平. 网络传输控制的综合性能评价标准. 计算机学报, 2002, 25(8): 869~877

11 林闯,周文江,田立勤. IP 网络传输控制的性能评价标准研究. 电子学报, 2002, 30(12A): 1973~1977

12 Paxson V. Measurements and analysis of end-to-end Internet dynamics. Ph. D. Thesis. Berkeley: University of California, April 1997

13 Paxson V. End-to-end Internet packet dynamics. IEEE/ACM Trans on Networking, 1999, 7(3): 277~292

14 Paxson V, et al. Framework for IP performance metrics. RFC 2330. May 1998

# 第13章

# Web QoS 控制

Internet 上 Web 应用和 HTTP 请求的爆炸性增长，使得目前许多热门的 Web 站点都经常面临服务器超载的问题，为此，如何对各类 Web 用户提供满意的服务性能和质量保证已成为迫切需要解决的问题。另一方面，随着电子商务应用的兴起，Internet 的服务模式正由传统的数据通信与信息浏览向电子交易与服务转变，Web 服务器系统作为支持电子商务的核心设施，迫切需要对不同类型的用户或服务请求进行区分优先级别的处理，从而提供区分的服务性能。因此，仅靠网络 QoS 机制并不能完全解决端到端的 QoS 控制问题，Web 服务器系统作为端到端网络中不可缺少的一个重要环节，必须同样具备建立和支持 QoS 的机制与策略。

近年来，网络传输的 QoS 控制技术研究已十分活跃，但是目前通用的 Web 服务器尚无 Web QoS 控制机制，无法为 Web 应用提供服务区分和性能保证。因此，如何在 Web 服务器及其集群系统中引入和实现 QoS 控制的机制与策略，已经成为实现下一代网络 QoS 控制技术不可或缺的关键环节。

Web QoS 控制的研究越来越受到重视，其主要的研究方向包括：Web 请求的分类机制，Web 服务器应用软件的 QoS 控制机制，操作系统的 Web QoS 控制机制，中间件的 Web QoS 控制机制以及 Web 服务器集群 QoS 控制的机制、策略与性能评价，等等。

## 13.1 引言

### 13.1.1 Web QoS 控制的研究背景

QoS 控制技术作为下一代网络的核心技术之一，多年来一直是计算机网络中研究与开发的热点问题[1~4]。一般而言，QoS 是指网络在传输数据流时要求满足的一系列服务请求，强调端到端或网络边界到边界的整体性，具体可以量化为带宽、延迟、延迟抖动、丢失率、吞吐量等性能指标。QoS 反映了网络元素（例如应用程序、主机或路由器）在保证信息传输和满足服务要求方面的能力。QoS 控制的基本目标是为 Internet 应用提供性能保证和服务区分。为此，IETF 已经提出了两种不同的 Internet QoS 体系结构：综合服务（integrated services，IntServ）[3] 和区分服务（differentiated services，DiffServ）[4]。截至目前，QoS 控制技术的研究和开发都进展得非常迅速，并且已经取得了许多基本的成果。

随着网络 QoS 技术研究和应用的不断深入,近期一种面向 Web 客户和 HTTP 请求以提供性能保证和服务区分的技术——Web QoS 应运而生,并且越来越受到国际上学术界和产业界的瞩目,成为 QoS 技术的一个新的研究领域和重要的学术分支。

随着计算机网络和多媒体技术的迅猛发展,Internet 上的 Web 应用一直呈现爆炸性的增长趋势。目前,Web 流量在 Internet 总流量中所占的比例已经超过了 60%,成为 Internet 上信息传输的主流。由于 HTTP 请求的指数性增长,Internet 上的许多热门站点都经常面临服务器超载的问题。通常,人们期望的 Web 站点的理想响应时间为 1 秒,这与人类的响应时间大体相当。美国 HP 实验室的 Bouch 等人对用户感知的 QoS 及其影响进行了研究[5],发现普通的 Web 用户通常不会忍受超过 8 ~ 10 秒的等待时间。而根据 Zona 研究中心[6]的统计,一个电子商务网站必须保证用户在 7 秒内得到响应,否则将损失 30% 或者更多的客户。

具体而言,Web 服务的响应时间由两方面的因素决定:一是网络传输的质量,另一个是服务器的处理性能。近年来,网络传输的 QoS 研究已经十分活跃,包括建立 IntServ 体系结构来提供端到端的 QoS 保证,建立 DiffServ 体系结构来提供可选择的区分服务级别。然而,如果 Web 服务器不支持任何 QoS 控制,那么仅依靠网络层 QoS 机制可能仍然不足以为用户提供满意的服务性能。例如,在服务器过载的情况下,具备端到端 QoS 保证的高级流可能仍然会遭到服务拒绝,或者 Web 服务的平均响应时间比用户的期望高出多个数量级,从而导致事实上的‘拒绝服务’效果。据估计,仅在美国的电子商务市场,慢的通信速度、延迟的响应时间、频繁的连接中断等问题已经导致了每年大约 400 亿美元的销售损失。

由此可见,由于服务器的超载问题,Web 服务器已经在某种程度上成为实现端到端 QoS 的瓶颈。因此,如何在 Web 服务器系统实现有效的 QoS 控制,为用户提供满意的服务性能保证,已经成为一个迫切需要解决的问题。

另一方面,随着电子商务应用的发展,Internet 的服务模式正由传统的数据通信与信息浏览向电子交易与服务转变,这种变化使得 Web 服务器及其系统成为支持电子商务的核心设施。企业和服务提供商都越来越崇尚将重要的服务转移到 Web 上去。例如,在线银行、股票交易、网上预定、网上购物等,都是目前流行的通过 Web 前端提供的电子交易与服务的形式。

与传统的 TCP/IP 和 HTTP 服务的平均主义哲学不同,电子商务应用通常要求对用户或服务进行区分优先级别的处理,这是因为所有的 Web 事务不可能对客户或服务器而言都同等重要。例如,对客户而言,在线进行股票投资交易的 HTTP 请求显然要比简单的浏览或下载请求更加紧要,因此需要更加严格的实时保证;对 Web 服务器而言,它需要为付费的用户提供较免费用户更好的服务级别。随着 Web 应用资源需求的不断增加,电子商务类增值服务迫切要求为其提供基于利润收益的有竞争力的区分服务,而不应该再遵循尽力而为型(best-effort)的服务规范。

因此,如何在 Web 服务器和 Web 服务器系统中为不同类型的用户或请求提供区分的、而非‘一视同仁’的 Web QoS 已经成为支持电子商务的一个至关重要的问题。

总之,Web QoS 概念的提出反映了当今 Web 应用和电子商务应用对于 QoS 控制的迫切需求,可以说,Web QoS 技术的出现是 Web 应用和电子商务应用飞速发展的必然结果。

如何在 Web 服务器及其系统中引入和实现 QoS 控制的机制与策略，从而满足不断增长的 Web 性能需求，为不同类型的用户或请求提供服务区分和性能保证，这是目前 Web 发展所迫切需要解决的问题，也是实现下一代网络 QoS 控制不可或缺的关键环节。

### 13.1.2　Web QoS 控制的研究概况

目前，通用的 Web 服务器尚无 Web QoS 支持机制，无法为 Web 应用提供服务区分和性能保证。现代 Web 服务器都是根据 Internet 尽力而为的服务模型平等地处理所有到来的请求，大多数 UNIX 内核的 Web 服务器采用 FIFO（first-in-first-out）的调度策略，在超载的情况下不加区别地丢弃高优先级的请求分组，使得通过 IntServ 和 DiffServ 等 Internet QoS 机制实现的性能改进受到严重损害。因此，仅靠网络 QoS 机制并不能完全解决端到端的 QoS 控制问题，Web 服务器作为端到端网络中不可缺少的一个重要环节，必须同样具备建立和支持 QoS 的机制与策略[7]。

从网络分层的角度而言，Web QoS 属于应用层的 QoS，它量度的是用户在与 Web 站点进行交互时所感受到的服务性能。例如，下载时间、交易时间（如银行结算、股票交易、网上购物等）、服务器的可用性、遇到的错误（如失败的连接、丢失的页面或组件、中断的链路、交易失败），等等。

由于 Web 基础设施的复杂性，影响 Web QoS 的因素有许多。实际上，Web QoS 控制涉及到构成 Web 的每一个元素，从网络技术和协议，到 Web 服务器（以及代理服务器）的硬件、软件（包括服务器应用软件、操作系统以及中间件）体系结构，等等。由于大多数 Web 基础设施的组件通常都无法轻易进行控制，因此 Web QoS 的实现并非一件容易的事情。相对而言，网络通信公司对其主干网具有完全的控制能力，因而能够向其客户提供基于网络可用性和保证网络响应时间的服务水平协议，而 Web 服务供应商则无法提供类似的服务保证契约，因为他们只能对 Web 基础设施的一小部分进行处理和操作。

概括地讲，Web 服务供应商可以实施的 Web 系统解决方案大体分为以下两类：

（1）区分的 Web 服务机制与策略。主要方法是定义用户或服务请求的类别，确定优先级，利用基于优先级的请求分配策略和资源监控与调度机制来保证不同的服务水平协议。

（2）Web 服务器系统体系结构设计。目标是确定能够向所有的 Web 用户或请求提供服务水平协议保证的正确的体系结构。分为三个方面：通过增加内存和 CPU 来扩大单个服务器的处理能力；在局域范围内通过复制服务器内容建立 Web 服务器集群来增强本地处理能力；在地理上广域分布的范围内通过复制服务器集群来扩大全局处理能力。

实际上，上述两个方面也是当前 Web QoS 控制技术研究的两个主要切入点。具体地讲，当前 Web QoS 技术的研究方向主要有：

（1）Web 服务器应用软件的 QoS 控制技术；

（2）操作系统的 Web QoS 控制技术；

（3）中间件的 Web QoS 控制技术；

（4）Web 服务器集群系统中的 QoS 控制技术；

（5）Web QoS 控制策略和算法的性能分析与评价技术，等等。

目前，国际上 Web QoS 控制的研究已经越来越受到网络学者和知名公司的重视。许多

著名的国际会议,如 International Workshop on Quality of Service(IWQoS),International World Wide Web Conference 等,都已经开始将 Web QoS 列为一个重要的会议议题。HP 公司已经推出了在 Web 服务器中支持 QoS 控制机制的名为‘ WebQoS ’的服务质量软件[8],而且,他们还与制造 LocalDirector [9] 和 DistributedDirector [10] 等网络负载均衡设备的 Cisco 公司结成战略同盟[11],联手推出面向 Web 服务器集群系统的 Web QoS 总体解决方案。IBM 公司也推出了支持 Web QoS 控制机制的名为‘ WebSphere ’的软件平台[12]。此外,许多生产第 7 层交换机(也称第 5 层交换机、Web 交换机、内容交换机)的公司也不断推出能够在服务器集群环境中支持基于 QoS 的负载均衡的硬件产品,如 Cisco 公司(收购了原来生产 CS-800 等 Web 交换机的 ArrowPoint 公司)的 CSS 11000 系列内容服务交换机[13],Fore(已被 GEC 公司收购)的 ESX-2400/4800 交换机[14],Foundry 的 ServerIron 系列 Web 交换机[15],以及 Alteon(已被 Nortel Networks 合并)的 Web 交换机等产品[16]。

但是,目前国内还很少见到关于 Web QoS 的研究论文,这方面的产品和专利技术也基本上属于空白。因此,紧密跟踪国际上 QoS 领域的技术前沿,积极开展 Web QoS 技术方面的研究与开发,是摆在我国科研工作者和企业面前的一项紧要任务。

## 13.2 Web 服务器概述

在对 Web QoS 控制技术进行详细讨论之前,本节将首先给出有关 Web 服务器的必要的背景知识。

随着 Internet 在全球的迅速发展,Web 服务器的应用已经非常普遍。从功能上来讲,Web 服务器监听来自用户的 HTTP 请求,根据请求的类型提供相应的服务。用户端使用 Web 浏览器与 Web 服务器通信。Web 服务器在收到用户的 HTTP 请求后,处理请求并返回 HTTP 应答。应答数据通常以格式固定、含有文本和图像等组件的页面出现在用户端浏览器。浏览器收到应答后加以处理并提供给用户。所以,从应用软件的角度来看,Web 服务器只是简单的程序:等待客户端的 HTTP 请求,然后提供相应的应答。图 13.2.1 简单地描述了这一过程。


图 13.2.1 Web 请求与应答的过程

### 13.2.1 Web 应答内容的编码与生成

Web 服务器返回给客户端的 HTTP 应答内容可以有多种格式,典型地是,这些内容是使用超文本标记语言 HTML(hypertext markup language)组织起来的。HTML 是 Web 的关键所在,它使用一种简单的标记语言来限定文本的格式,并生成独立于平台的超文本文档。HTML 标记可以表示超文本新闻、邮件、文档和超媒体 选项菜单 数据库查询结果;

带有内嵌图形的简单结构化文档,等等。HTML 文档支持 GIF 和 JPEG 等格式的内嵌图像。HTML 是一种格式化语言,只能定义信息的显示格式,却无法描述信息的语义,因此无法实现 Internet 上跨平台的数据交换,尤其是数据结构和语义的交换。

目前,扩展标记语言 XML(extensible markup language)在编码 Web 文档方面越来越显示出取代 HTML 的趋势。XML 通过标签定义信息的含义和结构,从而可以直接利用这些信息实现 Internet 上跨平台的信息交换以及计算机的自动处理。XML 对电子商务应用具有极为重要的意义,它使得公司之间很容易地以标准化的、连续的方式来描述并交换大规模的结构化信息数据。

Web 服务器返回给客户端的应答内容大致可以分为两类。第一类是静态内容,这类数据在与请求的服务时间相比的较长时间内保持不变,例如,保存在服务器文件系统中的文档内容。第二类是动态内容,这类响应是在接收到客户请求时在服务器端动态生成的。这种动态内容的生成基于在 Web 服务器运行辅助的第三方程序所产生的数据。目前已有一些 Web 技术可支持动态内容的生成,如公共网关接口 CGI(common gateway interface)、FastCGI、超文本预处理器 PHP(hypertext preprocessor,一种服务器端、跨平台、嵌入 HTML 文件的脚本语言)和 Java Servlet 等。

代理(proxy)服务器从 Internet 上的其他 Web 服务器获取其内容。通常地,代理服务器的位置靠近客户端,它截取客户发出的 HTTP 请求,并且代表客户向远程服务器提交请求。在与客户交互时,代理服务器就像服务器那样工作;而当与服务器交互时,代理服务器就如同客户。通过缓存(caching)从远程服务器获取内容,代理服务器可以减少 Web 请求的响应时间。

### 13.2.2 HTTP 协议

Web 服务器使用超文本传输协议 HTTP(hypertext transfer protocol)[17,18] 与浏览器通信,HTTP 是一种面向事务(transactions)的客户/服务器协议,规定了发送和处理请求的标准方式,从而允许不同种类的客户端相互通信而不存在兼容性问题。HTTP 协议已从最初的 0.9 版发展到 1.1 版[18],目前最常用的是 1.0 版[17]。HTTP 协议建立在传输控制协议 TCP(transmission control protocol)的有连接的、可靠的服务之上,其协议层对应于 OSI 七层协议中的会话层(session layer)。但 HTTP 是‘无状态’的协议:每个事务都独立地处理。因此,通常它在实现时,要在客户端和服务器之间为每个事务都创建一个新的 TCP 连接,然后在这个事务完成之后立刻结束连接,尽管在规约中没有要求实现这种事务生存期和连接生存期上的一对一的关系。这样,为了获取一个包含多个嵌入式元素(如 FRAME 结构文件、GIF 或 JPEG 图像文件、Java 的 Applet 等)的 HTML 文件,客户端典型地需要向 Web 服务器发送多个 HTTP 请求。

在 HTTP/1.0 中,HTTP 协议的一次会话由一个请求过程和一个应答过程组成,一个 TCP 连接只能包含一次会话。换言之,每个 HTTP 会话由一个单独的 TCP 连接完成,客户浏览器每发出一个 HTTP 请求都要建立一个新的 TCP 连接。例如,一个包含一首背景音乐和三幅图像的 Web 页面就要求 5 个独立的服务器请求来检索 4 个对象(背景音乐和三幅图像)以及带有这些对象的页面。图 13.2.2(a)描述了 HTTP/1.0 请求与应答的会话

**过程：在一个 TCP 连接上仅允许发送和应答一个请求。这种频繁的 TCP 连接与关闭操作增加了每个请求对服务器资源的需求以及每个请求的网络分组数量，结果会导致服务器系统性能的降低和响应时间的延长。**

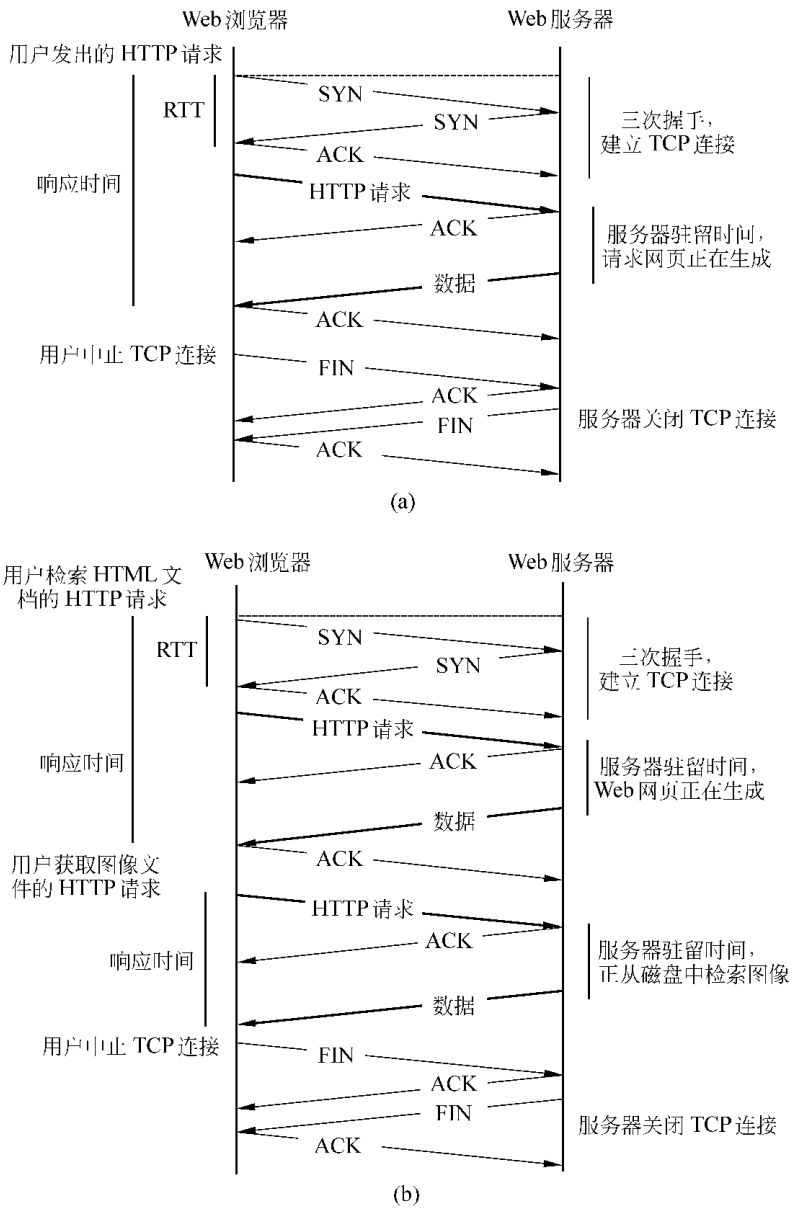



**图 13.2.2 HTTP/1.0(图(a))和 HTTP/1.1(图(b))的请求与应答过程**

**HTTP/1.1 采用了一项称为“持续的 HTTP 连接(persistent HTTP connections)”的新技**

术。该技术允许 Web 服务器在一个 TCP 连接中为客户端的多个请求提供服务，即一个 TCP 连接可以包含多个会话（每个会话仍由一个请求及其应答组成）。Web 服务器在接收到一个请求之后仍然在一个可配置的时间间隔（典型的为 15 秒）内保持该 TCP 连接的开放状态，以期进一步接收更多来自同一客户端浏览器的请求，直到该时限超过或者客户端主动切断 TCP 连接为止。图 13.2.2（b）描述了 HTTP/1.1 的请求与应答的会话过程：在同一个 TCP 连接上可以发送和响应多个请求。这种方法可以有效地减少为多个请求建立 TCP 连接所需的系统开销（CPU、内存和网络资源），并且可以实现对请求的流水线操作。此外，在一个 TCP 连接上接连发送多个服务器应答响应避免了多个 TCP“慢启动（slow-starts）”[19]，因此增加了网络的利用率和客户感知的有效带宽。慢启动是在 TCP 建立阶段通过快速增加（随着时间指数增加）TCP 的分组窗口以完全利用网络可用带宽的一种拥塞控制机制。

## 13.2.3　Web 服务器体系结构

运行 Web 服务器软件的最简单的系统体系结构就是图 13.2.1 中所示的一台单独的工作站。更复杂的 Web 服务器集群（cluster）系统将在 13.7 节中加以介绍。

由于 HTTP 协议运行于 TCP 层之上，因此客户机的 Web 浏览器必须与运行在工作站上的服务器软件建立一个 TCP 连接。为了接收客户端的 TCP 连接请求，服务器应用软件对某一知名端口（典型地为端口 80）进行监听。一旦建立起与客户的 TCP 连接，操作系统的内核就使用 accept( ) 系统调用将连接传递给服务器软件。然后服务器软件就等待客户在此连接上发来 HTTP 请求。当接收到请求时，服务器应用软件就分解请求，并将请求的内容通过该连接返回给客户。HTTP/1.0 仅允许客户在每个 TCP 连接上发送一个单独的请求，而 HTTP/1.1 则允许在一个 TCP 连接上发送和响应多个请求。Web 服务器

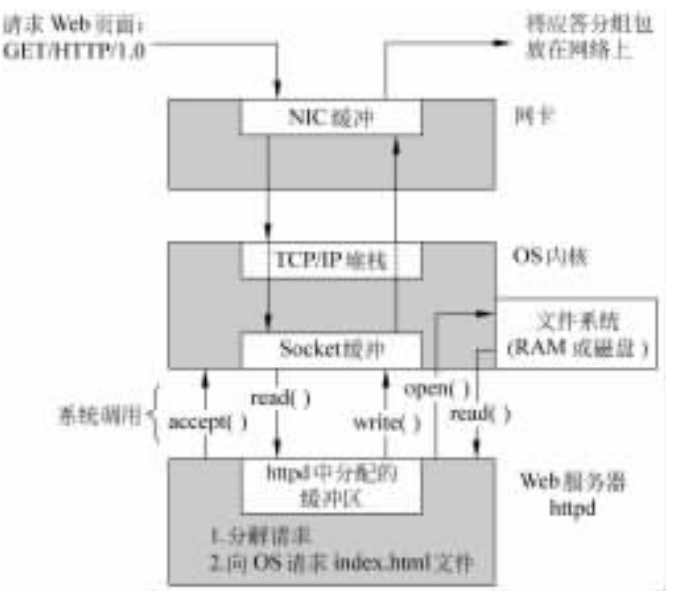



图 13.2.3　操作系统和 Web 服务器处理每个请求的工作流程

典型地从本地文件系统获取静态请求的内容,而代理服务器则从其他 Web 服务器获取内容(如果本地文件系统的缓存中没有该内容)。这两种服务器都可以使用存储器缓存来加速内容的检索。图 13.2.3 显示了操作系统和 Web 服务器处理每个请求的工作流程。

多年来,Web 服务器应用软件的设计已经经历了多次根本性的转变。根据传统的 UNIX 模型,早期的服务器为每个 HTTP 连接派生(fork)一个新的进程来进行处理,但派生进程的系统开销很大。后来的服务器(如 NCSA-HTTPD [20])使用一组预先派生的(pre-forked)进程,其中的一个进程被指派为主进程(master process),负责接收新到的连接。当收到一个连接后,主进程将该连接移交给某一预先派生的工作进程(worker process,又称为从进程 slave process)以完成剩余的 HTTP 处理。每个进程,不论是主是从,都只有一个控制线程。这种具有一个主进程的‘进程每连接’(process-per-connection)服务器的体系结构如图 13.2.4 所示。

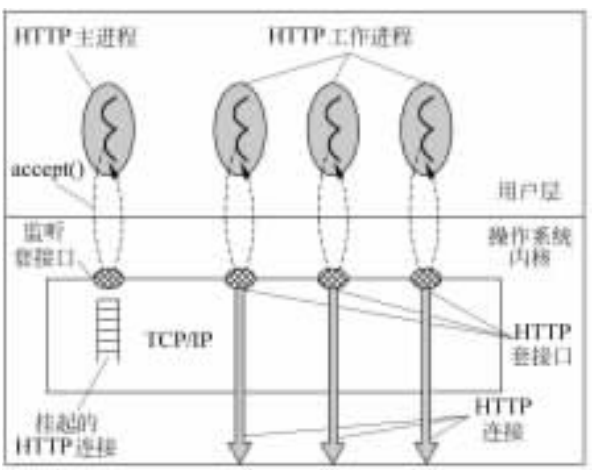



图 13.2.4 具有一个主进程的‘进程每连接’服务器

再以后设计上的改进进一步取消了主进程,这样,每个预先派生的工作进程都可以直接调用 accept( )系统调用。操作系统内核负责将一个新的连接移交给一个已经在监听套接口(listen socket)发出接收要求的工作进程。这种没有主进程的‘进程每连接’服务器的体系结构如图 13.2.5 所示,典型的例子如当今 Internet 上使用最广泛的 Apache [21] Web 服务器。

多进程服务器的上下文交换(context-switching)和进程间通信(inter-process communication, IPC)具有很大的系统开销。为了减少这些系统开销,许多新近的高性能服务器都使用单进程的体系结构。由于只有一个进程,几乎不必进行上下文交换和进程间通信。在一个事件驱动的模型中,服务器使用单一进程中的单一线程来管理所有的连接和执行所有的 HTTP 处理。服务器进程在使用 select( )系统调用的同时等待它所管理的所有连接上的事件。这种单进程事件驱动服务器的体系结构如图 13.2.6 所示,使用这

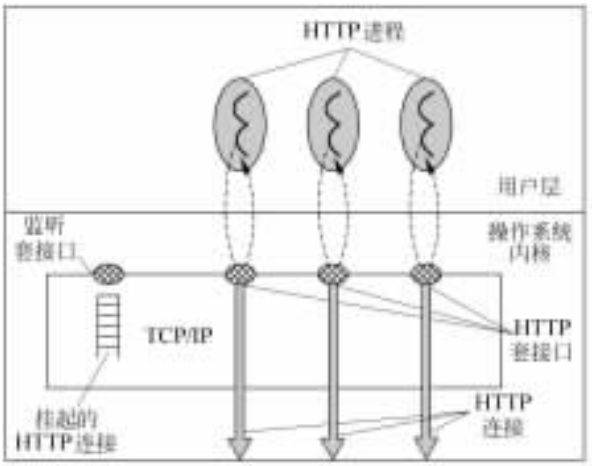



图 13. 2. 5　没有主进程的‘ 进程每连接 ’服务器

种体系结构的现代 Web 服务器有 Squid [22]，Zeus [23] 和 thttpd [24] 等。

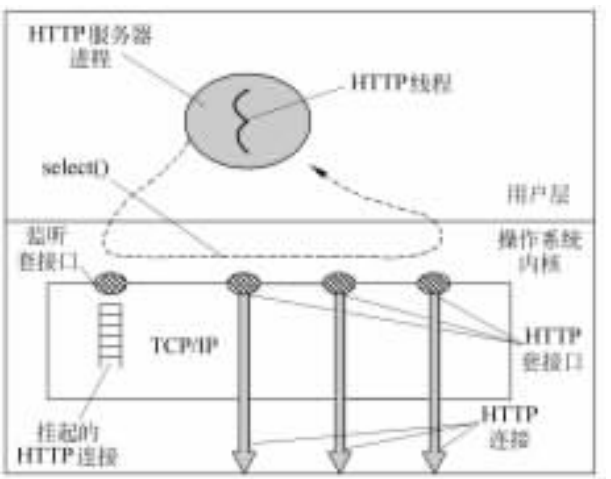



图 13. 2. 6　单进程事件驱动的服务器

为了有效地利用多处理器系统(multiprocessor system)并降低上下文交换与进程间通信的系统开销,一些服务器采用了在一个进程中设置多个控制线程的体系结构。在这个模型中,每个连接分配给一个单一的内核线程,线程运行于服务器应用进程的地址空间内。线程调度器负责不同服务器线程对 CPU 的分时共享。空闲的线程从监听套接口接收新的连接。这种单进程多线程的服务器体系结构如图 13. 2. 7 所示,Alta Vista 搜索引擎的前端就使用了具有这种体系结构的服务器。然而,并非所有的操作系统内核都支持在一个进程中设置多个线程,这限制了多线程服务器的可移植性。

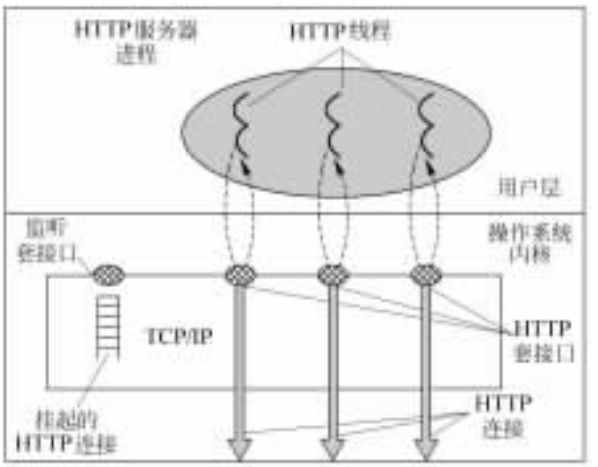



图 13.2.7 单进程多线程的服务器

## 13.3 Web 请求的分类机制

支持 Web QoS 的首要环节是对到来的 HTTP 请求进行区别与分类。目前已有多种请求分类的机制[7],这些机制大体上可以归纳为两类:基于客户类别的分类和基于目标类别的分类。

### 13.3.1 基于客户的分类

基于客户的分类是根据客户某些特有的属性和特征来进行分类,具体包括:

(1) 基于客户端 IP 地址的分类

客户端的 IP 地址可以用来辨别不同的客户。这种方法最易实现,但缺点是客户端的 IP 地址会被代理服务器或防火墙所屏蔽,因此其应用受到限制。

(2) 基于 HTTP cookie 的分类

HTTP cookie 是 Web 服务器返回给客户用来收集客户信息的数据块,也是一种具有惟一性的标识符,它可以嵌入 HTTP 请求内以表明客户所属的类别。例如,对特定商品或服务的网上订购就是作为持久的 cookie 来实现的。另外,cookie 也可以用来识别一个已经建立起来的会话(session),从而实现基于会话的请求分类。

(3) 基于浏览器 plug-ins 的分类

浏览器 plug-ins(插件程序,用于扩充 Web 浏览器的功能)也可以在每个 HTTP 的请求体中嵌入特定的客户标识符。这样的插件程序允许付费购买高级服务的客户下载。

总之,使用上述这些基于客户的分类方法可以设置优先的客户组,从而向其提供较其他客户更好的 Web QoS。

### 13.3.2 基于目标的分类

基于请求目标的分类是根据请求的目标所特有的一些属性和特征来进行分类,具体

包括：

（1）基于请求 URL 的分类

URL 请求类型或文件名路径可以用来区分不同请求的相对重要程度，因此可以作为一种请求分类的依据。

（2）基于目标 IP 地址或端口号的分类

当多个 Web 站点放置于同一 Web 服务器节点时，服务器可以利用请求的目的 IP 地址或端口号来进行请求的分类。

这类基于目标的分类方法可以用来实现为重要的请求（如信用卡支付请求）或者付费更多的虚拟站点提供更好的 Web QoS，也可以用来控制资源紧要型请求（如动态文件的请求）所消耗的资源。

一般情况下，上述两种分类机制对于请求分类而言通常是足够的，当然也可以使用基于客户和基于目标的特定属性的组合来进行请求分类。

## 13.4　Web 服务器应用软件的 QoS 控制机制

传统的 Web 服务器应用软件对客户的请求不加识别和区分，接收到一个请求便可立即进行处理。这样，分配给某个服务类（即一组服务，其所消耗的系统资源的计算和调度与其他组服务相独立）的系统资源就与该服务类所处理的 HTTP 请求的数量成正比，从而接收较高到达速率请求的服务类就可以获得比其他服务类更高比例的系统资源。因此，这种一视同仁的服务无法为高优先级的请求提供更好的 Web QoS。

通过改进 Web 服务器应用软件来提供 Web QoS 支持是一种典型的 Web QoS 控制技术。目前，很多研究项目和技术成果都试图通过这种途径来为不同的客户或请求提供区分的 Web QoS。这类研究解决了在 Web 服务器中实施服务区分和优先化处理的问题，并且与网络层的 QoS 支持机制相互配合和补充。其主要方法是将客户的 HTTP 请求进行分类，并且实现优先化调度、接纳控制（admission control）、资源分配等机制。

下面讨论一些典型的基于服务器应用软件提供 Web QoS 控制的研究工作。

### 13.4.1　服务器的优先调度

Web 服务器中的优先化调度已被证明可以有效地改善高优先级请求的延迟性能，但是对低优先级请求的影响却相当小。

美国威斯康星-麦迪逊大学（University of Wisconsin-Madison）的 Alemeida 等人实现了在用户级和内核级分别通过基于优先级的请求调度来提供区分的 Web QoS[25]。其中，用户级的方法是对 Apache Web 服务器软件进行修改，增加一个调度器进程，由它决定对请求进行服务的顺序，并且此调度器进程限制了每类优先级进程的最大并发数目；内核级的方法需要对 Apache 服务器软件和 Linux 内核同时进行修改，增加两个系统调用以实现请求优先级到进程优先级的映射机制和记录各请求所对应的活动进程的优先级别。

美国波士顿大学（Boston University）的 Crovella 等人研究了 Web 服务器的连接调度问题，针对处理静态文件的 Web 服务器提出了一种优先处理短连接的策略——最短连接优

先(shortest connection first)[26]。在此基础上,美国卡内基梅-隆大学(Carnegie Mellon University)的 Harchol-Balter 等人对"最短剩余处理时间"SRPT(shortest remaining processing time)调度策略进行了详尽的分析,并且在 Web 服务器上进行了实现[27,28],证明了 SRPT 对于最小化平均响应时间而言是最优的连接调度策略,并且显示了该策略相比于其他连接调度策略的优越性。

### 13.4.2 选择性的资源分配

当 Web 服务器不能为所有的请求提供满意的服务时,选择性的资源分配可以实现为更加重要的请求提供 QoS 保证,因而是一项很有前景的技术。

当今的 Web 服务器不允许对服务器资源进行选择性分配,所以,当服务器繁忙时,检索普通页面的请求通常会淹没检索更重要页面的请求。为此,美国加州大学戴维斯分校(University of California, Davis)的 Pandey 等人提出了一个 Web 服务器的 QoS 模型[29],该模型允许一个 Web 站点对 HTTP 服务器如何响应外部的请求进行定制,其方法是为不同的(或不同组的)页面请求设置不同的优先级并相应地分配服务器资源。该模型支持对服务器资源的预订,预订量可以是全部服务器系统资源的一个固定比例,或者是一个速率/带宽的保证。

美国密歇根大学(University of Michigan)的 Li 和 Jamin 提出了一种为 Web 客户按比例分配网络带宽的服务器模型[30]。该模型使用基于测量的方法为系统中不同的服务类提供带宽估计值。当某个服务类的带宽估计值低于其目标值时,则对过度分配带宽的服务类进行节流,适当地延迟处理这些服务类的请求,从而实现在各请求服务类之间按比例地分配网络带宽。

美国南加州大学(University of Southern California)的 Eggert 和 Heidemann 提出了三种简单的、服务器端的、仅是应用层的机制来提供两类不同级别(常规优先级和低优先级)的 Web 服务[31],具体的机制包括限制进程池的大小、区分进程的优先级以及限制传输速率。这些机制可以获得明显的性能改进,并且容易实现。

### 13.4.3 有效的接纳控制

一般地,在过载的情况下,有效的接纳控制机制[7]能够保证丢弃速率和任务的延迟界线。

HP 实验室的 Cherkasova 和 Phaal 提出了基于用户的会话(session)而非每个请求的接纳控制机制[32]。一个会话是指由某一用户在一个预先规定的时间段(如 30 分钟)内发出的一系列请求。由于目前的许多 Web 服务本质上都是事务性质的,包含许多申请 Web 服务的请求,因此基于会话的接纳控制具有重要的意义。例如,对于一次银行的交易结算,用户需要登录银行的主页、注册、检查账户余额、付账、最终退出登录。在过载的情况下,基于会话的接纳控制方案允许现有的用户会话继续进行,而新到的会话或者被重定向给其他的 Web 站点或者被拒绝服务。

### 13.4.4 Web 内容自适应

Web 内容自适应(adaptation)是一种比接纳控制更加灵活的机制,它可以在服务器重

载的情况下自适应地提供连续的内容降级服务而不是简单地拒绝请求，从而能够更好地为用户提供 Web QoS。

美国密歇根大学（University of Michigan）的 Abdelzaher 和 Bhatti 提出了一种依赖于 Web 内容自适应机制来实现服务区分的 QoS 管理体系结构[33]。其具体策略是在每个服务器上存储多份不同质量的 Web 内容，在服务器超载的情况下，可以使服务器有选择地为客户提供某种质量的 Web 内容：以体面的方式为低优先级客户提供平滑的服务降级，而保证高优先级的客户不会受到降级服务。他们根据负载条件和 QoS 需求，使用反馈控制技术，通过服务器利用率控制回路（如图 13.4.1 所示）来实现自适应的 Web 内容传送。这种 QoS 管理方法可以通过直接修改服务器软件来实现（也可以使用对服务器透明的中间件）。但这种方法的缺点是需要同时维护多份 Web 内容，因此需要较大的存储空间，费用较大。

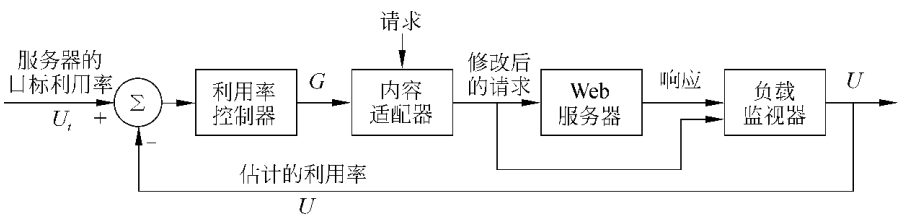



图 13.4.1　服务器利用率控制回路

在图 13.4.1 中，$U_t$ 代表服务器的目标利用率，即期望的最优服务器利用率，U 代表通过负载监视器计算得到的当前利用率。负载监视器测量当前的请求速率和发送带宽，并将其转换成利用率值 U，利用率控制器对 $U_t$ 与 U 进行比较得到利用率误差，并且采用积分控制器产生输出 G，即所需的内容自适应程度。内容适配器对 G 进行翻译，通过修改请求的 URL 对部分请求进行相应的降级等处理。负载监视器更新负载估计值并且形成闭环控制回路，使得 Web 服务器维持期望的目标利用率。

## 13.4.5　基于控制理论的方法

使用控制理论的方法解决 Web QoS 控制的问题是一个新颖的思路，并且已经引起研究者们的关注。

美国维吉尼亚大学（University of Virginia）的 Web QoS 研究小组[34]提出了基于控制理论的方法在 Web 服务器中实现相对的延迟保证[35]。他们提出了一种基于反馈控制回路的自适应的 Web 服务器体系结构（如图 13.4.2 所示），利用动态的连接调度和进程重新分配机制来实现 Web 服务器的比例区分服务，从而为不同的服务类提供相对的延迟保证。

他们使用了传统控制理论的方法来设计反馈控制回路，系统地设计和实现了一个自适应的 Web 服务器。所使用的控制理论方法有：通过离线系统辨识（system identification）建立一个 Web 服务器模型以进行性能控制；使用基于控制理论的性能规范（如稳定性、调节时间、稳态误差等）来描述 Web 服务器的性能需求；使用根轨迹（root

locus)方法设计一个反馈控制器以满足 Web 服务器的性能要求等。实验表明,可以用一个二阶差分方程来对 Web 服务器进行建模。他们通过修改 Apache Web 服务器软件实现了这种自适应的体系结构。最终的性能实验结果显示:即使在工作负载剧烈变化的情况下,该自适应 Web 服务器仍然可以实现鲁棒的相对延迟保证,而且能够确保稳定性,可以有效和准确地实现期望的相对延迟区分。

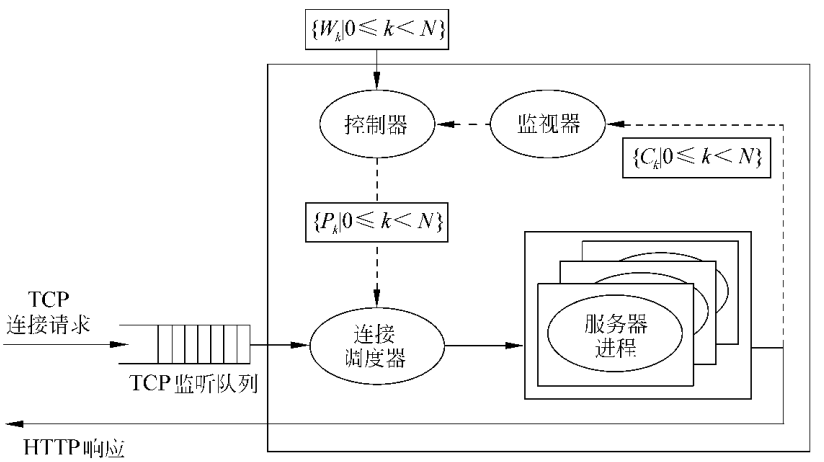



图 13.4.2 自适应 Web 服务器的反馈控制体系结构

在图 13.4.2 中,服务器进程是该系统的控制对象;$\{W_k \mid 0 \leqslant k < N\}$ 是该闭环控制的给定量,代表各服务类(共有 k 类)期望的相对延迟;$\{C_k \mid 0 \leqslant k < N\}$ 是该闭环控制的反馈量,代表各服务类的实际连接延迟;$\{P_k \mid 0 \leqslant k < N\}$ 是该闭环控制的偏差量,代表各类进程的数量在总进程数量中所占的比例。相对延迟保证的控制目标是要求任意两个服务类 j 和 k ($j \neq k$)在任意第 m 个采样时刻满足:$C_j(m) / C_k(m) = W_j(m) / W_k(m)$。

应该指出的是,这种设计方法仍存在明显的不足,即由于采用离线系统辨识来为动态的 Web 服务器建立模型,因而模型的准确性不高,影响了 QoS 控制的效果;而且,这种控制方法仅局限于单一的 Web 服务器,而未扩展到服务器集群。进一步的工作可以考虑采用在线系统辨识来调节控制器的参数,实现 Web QoS 的自适应控制,并且可以研究将这种基于控制理论的 Web QoS 控制方法从单一的 Web 服务器扩展到 Web 服务器集群甚至地理上广域分布的 Web 服务器集群系统。总之,采用控制理论的方法解决 Web QoS 控制问题是一种新颖的交叉学科方法,具有进一步深入研究和开发的价值。

## 13.4.6 典型软件产品实现

值得指出的是,目前已经出现了一些利用上述某些 Web QoS 机制与策略的商用 Web 服务器产品,比较典型的有 HP 公司的“WebQoS”服务器软件[8,36] 和 IBM 公司的 WebSphere 软件平台[12] 等。

下面以 HP 公司的 WebQoS 软件为例,对这类产品进行介绍。

具体而言,HP 公司研究开发出的‘WebQoS’的服务质量控制软件[36] 是基于服务器应

用软件提供 Web QoS 支持的一个典型的例子。图 13. 4. 3 具体显示了 HP 公司的 WebQoS 的体系结构。

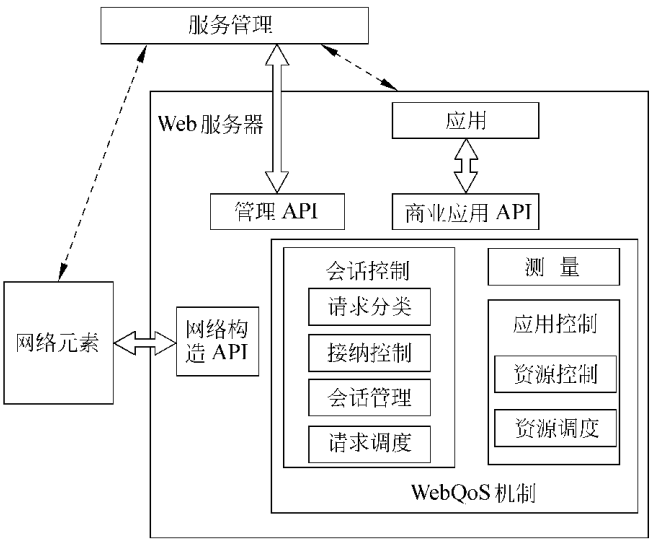



图 13. 4. 3　HP 公司的 WebQoS 的体系结构

在该体系结构中，首先对到达的请求根据其重要程度进行分类，基于系统管理员设定的过滤器和相关策略设置请求的 QoS 属性。分类后的请求根据其对应的调度策略可能被服务或拒绝，这由接纳控制组件决定。如果请求被接纳，则请求调度组件就会利用其分类属性来决定如何对请求进行排队和调度。会话管理组件为状态无关的 HTTP 协议提供会话语义并维持会话的状态。资源调度组件能够使高优先级的请求在主机操作系统中由高优先级的进程来执行，从而为高优先级的请求提供更好的实时性能。资源控制组件能够控制服务器的资源分配。此外，该体系结构还支持与网络 QoS 机制的集成，例如读取和标识 IP 服务类型（type of service，ToS）或者区分服务标记域，从而确定 TCP 连接的丢弃优先级。该体系结构还支持与管理系统的集成，从而实现 WebQoS 配置参数的远程设置，并且能够输出内部的测量结果以利于实施监控。

基于这种体系结构，HP 实验室的研究人员对 Apache Web 服务器应用软件进行了修改，提出了一种支持区分的 Web QoS 的服务器原型系统[7]，其工作原理如图 13. 4. 4 所示。

具体而言，该服务器系统实现了以下 5 种重要的 Web QoS 控制组件。

（1）连接管理器

连接管理器是通过修改 Apache 服务器软件生成的一个新的惟一的接收器进程。它负责截取所有到来的 HTTP 请求，并将其分为不同的 QoS 类别。然后将分类后的请求放入相应的 QoS 分级队列中等待服务。Apache 工作进程从这些多级队列中接收请求，而不直接从 HTTP 套接口接收。工作进程使用 UNIX 系统中的共享存储器设施共享这些请求队列。工作进程基于请求调度策略选择下一个进行服务的请求。例如，对于优先级调度

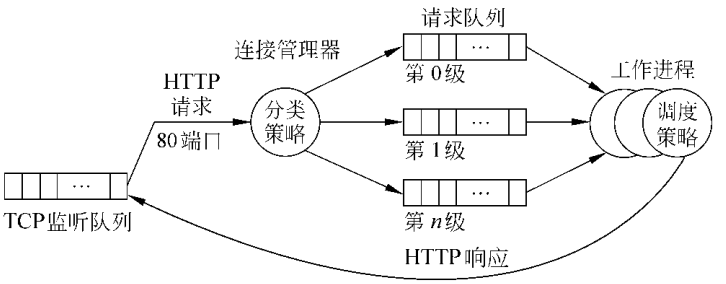



图 13.4.4 WebQoS 服务器的工作原理

策略,高级请求队列中的请求将首先得到处理。这种间接性的请求接收方式允许调度策略决定哪些请求被优先处理以及哪些请求被拒绝服务。

由于所有的请求必须首先由连接管理器进行接收,因此要求连接管理器必须运行得足够快,以保证填满请求队列。反之,如果连接管理器的执行速度不够快,则即使仍有高优先级的请求在等待被接收,但是如果所有高级请求队列中的请求已被处理完,工作进程也会执行低优先级队列中的请求。在这种情况下,服务器便很难快速响应新到达的高优先级请求,从而破坏了优先级服务策略。而且,如果连接管理器的执行速度不够快,新到的请求可能很快填满 TCP 监听队列,导致该 HTTP 端口拒绝任何新的 TCP 连接,从而高优先级的请求无法建立新的 TCP 连接而被最终拒绝。

(2) 请求分类

支持区分的 Web QoS 的一个关键的需求就是服务器能够对每类新到的请求进行识别与分类。可以采用前述的基于客户类别或基于目标类别的请求分类机制。

(3) 接纳控制

当服务器的处理速率低于客户请求的到达速率时,对高优先级和低优先级的请求,服务器都无法进行快速响应。在这种情况下,为了保护服务器免受过高的请求负载,一些到达的请求就必须被拒绝。很明显,应该首先拒绝低优先级的请求而保护高优先级的请求,并且维护已经建立的会话。所以在服务器过载的情况下,针对低优先级请求的接纳控制就会被触发。接纳控制的目的是保证低优先级的高速率请求不会影响高优先级请求的服务性能,使得在过载的情况下,服务器仍然能够维持现有客户会话的 Web QoS。在图 13.4.4所示的系统中,接纳控制方案可以根据队列的阈值来决定是否接收新到的请求,如果排队等待的请求达到队列的阈值,则将新到的请求丢弃而不予接收。

(4) 请求调度

请求经过分类和接纳控制后进入相应的请求队列,工作进程使用请求调度策略从中选择请求进行服务,即通过选择请求执行的顺序,为每类请求提供不同的服务级别。工作进程是基于调度策略选择请求进行服务的自治进程。为了保持 Apache 工作进程的设计,当工作进程选定某一请求之后它将持续运行直到完成为止,然后才能够处理其他请求。以下是一些可用的调度策略:

- 绝对优先级(strict priority):总是优先调度优先级最高的请求,在高优先级请求被处理完之前,较低优先级的请求只能等待。

- 加权优先级(weighted priority):根据客户请求的加权重要程度进行选择服务。例如,如果某类请求的优先权重是另一类请求的两倍,则该类请求得到服务的数量就将是另一类请求的两倍。具有高优先级的请求会获得较高的服务概率,但不是绝对的优先概率1.0。
- 共享容量(shared capacity):根据某一事先设定好的系统资源容量对每类请求进行调度,任何未使用的系统资源可以分给其他类请求。每类请求也可以设定一个不可分配给其他类请求的最小的资源预定容量。
- 固定容量(fixed capacity):根据某一固定的、不可与其他类请求共享的系统资源容量对每类请求进行调度。
- 最早截止优先(earliest deadline first):根据每个请求的完成服务的截止时间来进行调度。这种策略能够提供预测的响应时间保证。

如图13.4.3所示,使用何种策略可以通过一个包含网络构造API的策略管理接口加以配置。

(5)资源调度

保证主机操作系统将高优先级的请求通过高优先级的进程来进行处理,即服务器为工作进程设置的优先级应与其处理的请求的优先级相匹配。UNIX中的nice系统调用可以为高优先级的进程分配更多的CPU资源。

综上所述,通过改进Web服务器应用软件来提供Web QoS支持是一种非常直观的Web QoS控制思路,而且通常也比较有效,因此这类机制与策略是当前Web QoS研究和开发的热点,类似的研究和实现还有许多[37,38]。但是由于这种方法需要对Web服务器软件进行一定程度的修改,所以目前在实际应用中仍存在通用性和可扩展性等方面的局限。

## 13.5 操作系统的Web QoS控制机制

通过修改Web服务器应用软件虽然能在一定程度上为客户或请求提供Web QoS支持,但仅在服务器应用层执行服务区分,对于实现Web QoS控制而言仍具有一定的局限性。实际上,一个HTTP请求的很重要的一部分处理是在操作系统的内核中完成的。

操作系统和Web服务器软件处理每个请求的工作流程如图13.2.3所示(见13.2.3节)。从中可见,由于客户请求中很重要的一部分处理是在操作系统的内核中完成的,因此,缺少了这一层次上的服务区分会导致对这部分处理的计算不准确。例如,在内核中缺少服务区分会导致某些服务类占有比其应得的份额更多的系统资源。为了实现区分的Web QoS控制,Web服务提供者要求操作系统的内核能对不同请求类别所消耗的系统资源进行有效的计量和控制,从而实现在Web服务器中为不同的请求提供基于类别的区分的性能。

另外一个问题关系到当多个进程或线程对某一服务类进行处理时的应用层资源调度。大多数传统的操作系统将每个进程或进程中的线程既作为调度的实体,也作为资源分配和管理的单元(或称资源代理resource principals),并且在这些代理之间多路复用系

统资源。在这种以进程为中心的操作系统中,进程是组成一个服务类的基本单元。在需要进行服务区分的 Web 服务器中,当多个进程/线程与一个单一的服务类相关联时,对分配给多个进程/线程的资源进行控制是很困难的,从而很难对该服务类实现期望的资源分配。例如,如果该服务类的某些进程/线程没有完全用尽所分得的资源,操作系统就会将这些剩余的资源分给系统中所有竞争资源的代理,而不是仅分给与该服务类相关联的进程/线程。

操作系统的研究者们已经意识到了操作系统内核的机制对于支持服务区分和实现 Web QoS 的重要性[39]。Banga 等人提出了"资源容器(resource container)的抽象这一概念,将资源代理与进程/线程的概念从原来的一体化中分离开来,为操作系统中精细粒度的资源管理提供了支持。资源容器可以用来计算和控制不同请求类别所消耗的操作系统资源,这种操作系统的控制机制能够在 Web 服务器中提供基于类别的区分的性能[40],从而在 Web 服务器中有效地支持服务区分和 Web QoS[39,40]。

具体而言,资源容器是资源代理的最好的操作系统抽象,并且与进程和线程相独立。一个资源容器逻辑上包含一个服务类所消耗的所有系统资源。例如,对于一个与某服务类相关联并由 Web 服务器所管理的 HTTP 连接,其消耗的资源包括分配给该连接以及 Socket 和协议控制块等内核对象的 CPU 时间、该连接所使用的网络缓冲区,等等。使用资源容器,Web 服务器可以将一个网络连接、一个服务器线程、一个 CGI 进程等与一个单一的资源代理(代表正被处理的服务类)相关联。此代理与代表其他服务类的代理竞争服务器的资源。资源容器允许在内核和用户层为某一服务类所消耗的资源进行准确的计算与调度,并且与适当的资源调度程序相结合,能够为不同类型的客户或请求提供性能隔离和区分的 Web QoS。

使用资源容器,一个进程/线程与一个资源代理的绑定是动态的,并且直接在应用的控制之下。内核将进程/线程所消耗的资源对资源容器进行记账。一个容器可与多个进程/线程相关联。一个在多个服务类之间分时复用的进程/线程可以动态地改变其与资源容器(对应于正被处理的服务类)的绑定。系统调度程序为资源容器而不是进程/线程提供可用的资源。然后这些资源通过特定的二级容器调度程序进一步由相关的进程/线程多路复用。

此外,资源容器允许一个 Web 服务器或代理服务器的操作员将一个资源代理与位于该服务器/代理上的每个虚拟站点或客户团体相关联。为了向不同的虚拟站点或客户群有效地提供区分的服务,资源容器还必须针对每类资源(CPU、存储器、磁盘和网络带宽)与一种适当的调度策略相结合。因此,使用资源容器能够为位于同一 Web 服务器上的多个虚拟站点或客户团体实现性能隔离和服务区分,实现区分的 Web QoS。

除了资源容器,研究者们已经针对实验室的实时和多媒体操作系统提出了其他一些相关的资源代理的抽象来提供精细粒度的资源管理,从而提供操作系统的 QoS 支持。例如,微软实验室的 Rialto 实时操作系统中的 Activity[41] 与资源容器相似,也是一种优秀的资源代理的操作系统抽象。软件性能单元 SPU(software performance units)[42] 是共享内存多处理器环境下的一种资源代理,它与资源容器在很多方面具有相似性,SPU 可以在基于大型对称多处理器的计算服务器环境中对 CPU、内存和磁盘带宽等资源实现可控的分

配。Eclipse 操作系统中的预留域(reservation domains)[43]允许系统控制一组进程所消耗的全部资源,为进程组提供资源保证,从而满足多种实时条件下的 QoS 需求。Scout 操作系统支持一种路径(paths)[44]抽象,可以代表多层系统中的 I/O 通道(如 TCP 连接)。Path 抽象已经在网络和 Web 服务器设备中使用,用以实现精细粒度的资源管理。

## 13.6　中间件的 Web QoS 控制机制

前面讨论了 Web 服务器应用软件以及操作系统的 Web QoS 控制机制,为了向不同类型的客户提供区分的 Web QoS,这些机制通常要求对 Web 服务器应用软件或者操作系统进行根本上的修改,使之具备特有的体系结构以实现对 Web QoS 控制的支持。然而,这类修改在具体实现上具有一定的难度,而且改进后的系统也存在灵活性、通用性和可扩展性等方面的问题,因此具有一定的局限性。

目前,中间件(middleware)技术已经用来为 Web 服务器提供 QoS 支持。基于中间件的 Web QoS 控制机制不存在上述两种方法的问题,无须对 Web 服务器应用软件或者操作系统进行修改而能够实现对 Web QoS 的支持。

中间件是基础软件的一大类,属于可复用软件的范畴。顾名思义,中间件处于操作系统软件与用户应用软件的中间。中间件在操作系统、网络和数据库之上,在应用软件的下层,总的作用是为处于自己上层的应用软件提供运行与开发的环境,帮助用户灵活、高效地开发和集成复杂的应用软件。

在众多关于中间件的定义中,普遍接受的是美国 IDC 公司的表述:"中间件是一种独立的系统软件或服务程序,分布式应用软件借助这种软件在不同的技术之间共享资源,中间件位于客户机/服务器的操作系统之上,负责管理计算资源和网络通信"。该定义表明,中间件是一类软件,而非一种软件;中间件不仅实现互连,还要实现应用之间的互操作;中间件是基于分布式处理的软件,最突出的特点是其网络通信功能。

在众多的研究中,HP 实验室的 Bhoj 等人设计和实现的 Web2K 服务器[45]是一个典型的例子。Web2K 是一种基于 QoS-aware 中间件来实现 Web QoS 功能的服务器,该服务器的体系结构如图 13.6.1 所示。

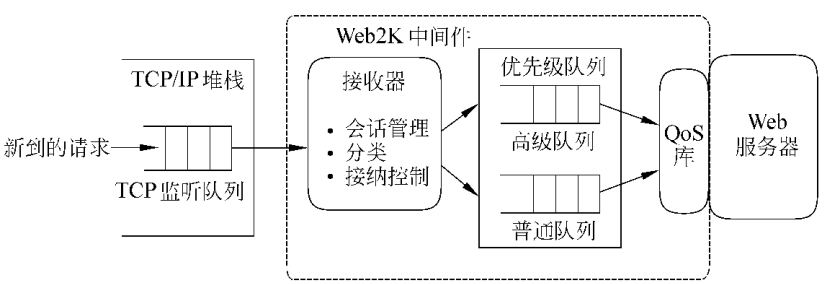



图 13.6.1　Web2K 服务器的体系结构

在该服务器中,QoS-aware 中间件层保证 Web2K 服务器可以向用户提供区分的 Web QoS,而无须对操作系统和 Web 服务器应用软件进行修改。为此,中间件层截取服务器的

TCP Socket 调用,并直接与服务器的 TCP/IP 堆栈对接,接收请求并进行优先化处理,然后将请求转发给 Web 服务器进行服务。

中间件层的主要组件如下。

(1) 接收器

中间件层的接收器组件与服务器的 TCP/IP 堆栈对接。接收器对服务器的 TCP 端口(通常是端口 80 或者 443)进行监听,并负责接收新到的连接。对于到达连接上的每个请求,接收器首先执行会话管理(session management),以确定该请求是否属于一个已经存在的会话。如前所述,一个会话是由某一用户在一个预先规定的时间段内发出的一系列请求。为了维护会话信息,Web2K 服务器的中间件在返回给客户的每个 HTTP 响应头部引入了一个特殊的 Cookie。该 Cookie 能够帮助接收器区分到来的请求是属于已有的会话还是新建的会话。

在会话管理之后,接收器负责对请求进行分类。每个会话与一个类(高级的或普通的)相关联,一个会话中的所有请求属于同一类。为了确定一个会话所属的类,接收器可以使用多种分类准则。例如,根据建立会话的用户的标识以及用户的 IP 地址等。用户的会话也可以由于与其关联的事务的变化而得以升级。例如,当用户决定从 Web 站点购买物品时,其已建立的会话就可以升为高级类。后端的应用(例如,结算应用逻辑模块)也可以通过定义好的 API 将分类信息传递给 Web2K 服务器。在分类之后,接收器进行接纳控制以决定是否接纳请求进行服务。如果请求被拒绝服务,则接收器负责向客户端返回一个拒绝响应以表明该请求不能得到服务。所有被接纳的请求由接收器负责放置到两个优先级队列(即如图 13.6.1 中所示的高级队列和普通队列)中去排队。

接收器所实现的上述会话管理、分类、接纳控制等机制联合起来称为“Traffic-aware 请求处理模型”,该模型的具体实现如图 13.6.2 所示。

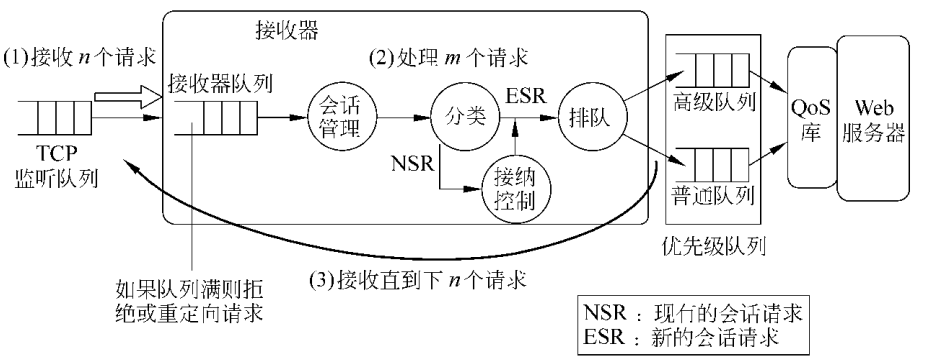



图 13.6.2 Web2K 接收器实现的 Traffic-aware 请求处理模型

(2) QoS 库

如图 13.6.1 所示,Web2K 服务器中间件层通过 QoS 库与 Web 服务器对接。在服务器启动的时候,Web2K 服务器通过一个称为“qosification”的进程将 Web 服务器的 TCP Socket 调用映射成对 QoS 函数库的相应的调用。Web 服务器在一个 TCP Socket 上执行的每一个操作(如 accept, read, send, close 等),QoS 库都有一个相应的内部函数与之对应。

这些函数使 Web 服务器从优先级队列中而不是从 TCP Socket 库中接收新到的 HTTP 请求。

概括地讲，Web2K 服务器的设计与以往其他研究工作相比，其关键性的创新主要有：对到达的请求实行一种新颖的 Traffic-aware 处理方案，优先保证高级请求的服务性能；请求的及早优先化（early prioritization）处理，利用用户会话的过去历史信息来对请求实施优先化以及分类；基于控制理论中比例积分 PI（proportional integral）控制技术进行预测性队列控制，确定可以接纳的各类请求的比例；实现多级接纳控制机制。

Web2K 服务器的性能评价结果表明，即使在极严重的超载情况下（如负载超过服务器处理能力的 3 倍）该服务器仍然能够为高优先级的用户提供响应时间等方面的 Web QoS 保证[45]。

类似的研究还有很多。文献[33]中给出了利用中间件方法实现 Web 内容自适应机制的过程。Virginia 大学的 Web QoS 研究小组开展了一系列基于中间件的 Web QoS 机制研究[34]。

这类基于中间件的方法可以在 Web 服务器中实现透明的负载监测、过载保护、动态 QoS 自适应、QoS 隔离和服务区分，无须修改服务器软件代码、下层协议、操作系统的调度及通信资源管理机制。因此，这种方法无须对客户和服务器进行修改，可以直接应用于目前的 Internet 环境，具有很强的实用性和应用前景。

## 13.7 Web 服务器集群的 QoS 控制

目前，集群（clustering）技术[46,47]已经成为解决服务器超载和提供高性能服务器的一种有效手段。

Web 服务器集群系统是由分布在局域网（LAN）或广域网（WAN）上的多台 Web 服务器主机（同构的或异构的）相互联结而成的一种服务器体系结构，采用负载均衡策略将到达的请求分配给集群中的某台服务器进行服务。一般地，集群系统可以分为由局部范围内的多台服务器组成的局域集群和由多个局域集群在地理上广域分布而形成的广域集群。与单一的工作站相比，集群系统具有更高的性能/价格比、良好的可扩展性、更好的系统可靠性和容错性，但是其系统规模更大，涉及的因素更多，Web QoS 控制问题也更为复杂。

Web 服务器集群需要设计和实现有效的请求分配（request dispatching）机制和负载均衡（load balancing）策略[47,48]，将客户的请求分配到集群中最适宜的 Web 服务器节点进行处理，从而获取可得到的最佳 Web QoS 性能，并且通过负载均衡策略，使集群中各台 Web 服务器的负载处于均衡状态（即服务器分得的请求负载与其处理能力成正比），使得整个集群系统的效率最高。因此，对于 Web 服务器集群系统，有效的请求分配机制和负载均衡策略对其实现高系统性能和为用户提供 QoS 性能保证具有决定性作用。

目前，常见的 Web 服务器集群的体系结构主要有镜像站点、基于 DNS（domain name server）的集群和基于请求分配器（dispatcher）的集群。

### 13.7.1 镜像站点

镜像站点的客户是直接从多个具有独立 URL 的镜像站点中进行选择并接受服务,这种体系结构对客户不具有透明性,而且集群也无法对请求分配进行任何控制,因此这里不对其进行详细介绍。

而基于 DNS 和基于请求分配器的集群体系结构则均对客户具有透明性,而且可以实现各种请求分配机制和负载均衡策略。

### 13.7.2 基于 DNS 的集群

在基于 DNS 的集群体系结构中,集群所在域的授权域名服务器(称为“集群 DNS”)向外部提供一个单一的 URL 主机名作为整个集群的虚拟接口,使得集群对客户端具有透明性。集群中的每台 Web 服务器都具有一个真实的 IP 地址。集群 DNS 作为集群系统的集中式请求调度器,在 Web 站点域名(URL)到服务器节点 IP 地址的映射过程中,能够实现多种负载均衡策略,从而为客户请求选择最适宜的目标服务器。

但是,由于中间域名服务器的地址映射缓存机制(TTL > 0),当地址映射的有效期未过时,地址映射请求就会被中间域名服务器直接处理并返回给客户端,而不会最终到达集群 DNS。因此,集群 DNS 只能处理很少一部分地址映射请求,从而只能实现一种粗粒度的负载均衡。

由于集群 DNS 对到达请求的控制能力有限,加之来自不同客户域负载的高度不一致性和真实 Web 工作负载的高可变性,DNS 调度器通常需要使用复杂的负载均衡算法来实现可接受的性能。这些算法典型地依赖于额外的状态信息,如每个域的隐藏的负载权(表示来自该域的请求速率)、客户的分布位置、服务器的状态信息,等等。

另外,由于 TTL 的值越小,集群 DNS 控制地址映射请求的机会就越大,所以,自适应地设置每个地址映射请求的 TTL 值也能够显著地提高系统的性能。

文献[47]和[48]对这种集群体系结构及其负载均衡算法进行了综述和分析,主要是针对局域集群。Colajanni 等人研究了广域分布的异构 Web 服务器集群的动态负载均衡策略[49],基于 DNS 请求分配器提出了一种动态改变 TTL 的自适应算法。Cardellini 等人研究了广域分布的 Web 服务器集群的负载均衡问题[50],提出了一种基于 DNS 邻近(proximity)调度和 HTTP 请求重定向的综合机制,能够为地理上广域分布的 Web 站点提供 QoS 支持。采用这种基于 DNS 的集群体系结构的典型例子有 NCSA 的 Round-Robin DNS(DNS-RR)[51]和 Cisco 公司的 DistributedDirector[10]等。

### 13.7.3 基于请求分配器的集群

与基于 DNS 的集群相对照,基于请求分配器的 Web 服务器集群能够完全控制所有到来的请求,并且实现精细粒度的负载均衡[47]。

在基于请求分配器的集群系统中,前端的请求分配器作为到达请求的代理,负责集中地接收所有到达的 HTTP 请求,并且按照特定的负载均衡策略将客户的请求均衡、透明地分配给集群中的后端服务器。整个集群具有单一的虚拟 IP 地址,即集群地址,使集群中

的服务器对客户端具有透明性。实际上,集群地址就是请求分配器的IP地址。

Schroeder等人对Web服务器集群技术进行了综述[46],他们根据请求分配策略将Web服务器集群分为以下三类:

- L4/2——第4层交换,第2层分组转发;
- L4/3——第4层交换,第3层分组转发;
- L7——第7层交换。

在这种体系结构中,早期的研究或产品典型地采用第4层交换机(L4/2,仅处理到达的请求分组)或TCP路由器(L4/3,处理到达的请求分组和返回的应答分组)作为请求分配器,能够根据后端服务器的私有IP地址或MAC地址对之进行独一无二的识别,并在TCP层执行请求分配(即第4层处理)。所采用的典型请求分配机制包括分组重写(packet rewriting)[9, 52~54]、分组转发(packet forwarding)[55~57]、HTTP重定向(redirection)[10, 58]等。其负载均衡策略典型地根据后端服务器的活动连接数、运行队列中的进程数量、到达的请求速率等情况来选择负载最轻的服务器。采用这种集群体系结构的典型例子有Cisco公司的负载均衡产品LocalDirector[9]和DistributedDirector[10]、IBM公司的Network Dispatcher[55, 56]和Bell实验室的ONE-IP[57]等。

最近的研究和产品已经开始采用第7层交换机(也称为第5层交换机、Web交换机或内容交换机)[59]作为集群的前端请求分配器(L7),实现Content-aware(基于请求内容)的请求分配。Content-aware请求分配策略既能提高集群后端服务器主存储器缓存的命中率(hit rates),又能实现负载均衡,从而提高集群系统的性能[60, 61]。与不考虑请求内容的分配机制相比,Content-aware请求分配的实现机制要复杂得多,这是因为在对目标服务器进行选择时必须首先检查客户HTTP请求的内容或服务的类型。为此,客户必须首先与不对请求实施服务的分配器建立一个TCP连接,并且要求请求分配器执行比不考虑请求内容的第4层(TCP层)处理复杂得多的第7层(应用层)处理。

目前已经提出了一些能够支持Content-aware请求分配的机制,典型的有HTTP重定向、前端中继(relaying front-end)[60]、TCP接合(splicing)[59, 62]、TCP转接(handoff)[61]等。目前典型的支持Content-aware请求分配的第7层交换机产品有:Cisco公司(收购了原来生产CS-800等Web交换机的ArrowPoint公司)的CSS 11000系列内容服务交换机[63],Fore(已被GEC公司收购)的ESX-2400/4800交换机[14],Foundry的ServerIron系列Web交换机[15],以及Alteon(已被Nortel Networks合并)的Web交换机[16]等。

第7层交换机和Content-aware请求分配机制的出现,使得在Web服务器集群中实现基于QoS的负载均衡成为可能。与第4层交换机和TCP路由器等不考虑请求内容的分配器相比,第7层交换机除了能够获得第2~4层信息以外,还能获得应用层信息(如URL),从而能够执行基于HTTP内容的请求分类和优先化,实现QoS-aware请求分配和负载均衡,进而与后端服务器的Web QoS控制机制协调一致。在这种情况下,第7层交换机必须连续地建立两个TCP连接:一个与客户端,另一个与选定的服务器节点。最后使用TCP接合、TCP转接等机制实现客户与服务器的直接通信。

以往的研究工作对Web服务器QoS控制和集群负载均衡策略的研究是相互独立的:Web QoS控制主要通过在服务器实现基于请求优先级的调度策略等来实现区分的QoS;

集群负载均衡策略主要利用服务器的负载信息等来实现请求的均衡分配,从而提供最好的服务性能。两者的分离使得分配器的负载均衡策略没有考虑到 HTTP 请求的 QoS 类型,即是 QoS-unaware 的,因此不能实现基于 QoS 的请求分配和负载均衡。在这种情况下,客户的请求在集群的前端被一视同仁地执行分配处理,但在后端服务器却被分为不同的 QoS 类执行基于优先级的进程调度。这种前后过程控制目标的不一致和不协调将导致不能实现面向 QoS 的请求分配的最优化,从而影响系统的 Web QoS 性能。据此,我们提出了在 Web 服务器集群的负载均衡过程中考虑 HTTP 请求的内容和优先级的 QoS-aware 综合控制思想[64],并且针对局域集群和广域集群提出了多种 Web QoS 控制的综合策略[65, 76]。提出了一种采用 Content-aware 请求分配器的、能够实现 QoS-aware 综合控制策略的 Web 服务器集群体系结构,在此基础上,提出了单个局域集群的多服务器多队列(multiserver multiqueue, MSMQ)系统模型[66]和广域集群的多服务器多队列网络(multiserver multiqueue network, MSMQN)系统模型[67],使用随机高级 Petri 网(stochastic high-level Petri nets, SHLPN)建模和分析技术,为之建立了 SHLPN 性能模型,并进行了性能分析[65]。性能分析的数值结果表明,与常用的 QoS-unaware 控制策略相比,我们提出的 QoS-aware 综合控制策略能够为不同类型的客户或请求提供区分的 Web QoS。具体情况请见本章 13.8 节的描述。

另外,Zhang 等人提出了一种面向全球分布的 Web 服务器集群系统的 QoS-aware 负载均衡算法[68],该算法基于两级分布式 Web 集群体系结构,在为一个到达的请求选择服务器节点的同时利用了失效的负载信息和基于内容的请求调度,从而达到减少响应时间的目的。他们将不同的客户服务类作为负载均衡算法中的一个参数,从而能够更好地实现端到端的 QoS。

目前,制造 LocalDirector[9] 和 DistributedDirector[10] 等网络负载均衡设备的 Cisco 公司已经和生产 WebQoS 服务质量软件的 HP 公司结为战略同盟[11],联手推出支持 Web QoS 的集群系统解决方案。

应该指出的是,从系统科学的观点来看,局域集群和广域集群系统都属于规模庞大、结构复杂、功能综合、因素众多的大系统。因此,进一步地可以从大系统控制论的角度,研究这些 Web 服务器集群大系统 Web QoS 的多级-递阶、分解-协调的综合控制,实现 Web 服务器集群大系统 Web QoS 控制的最优化和协调化。

此外,一个完整的端到端 QoS 解决方案需要网络 QoS 控制与 Web QoS 控制的结合。因此,进一步的研究工作可以考虑将 Web QoS 控制机制和策略与网络 QoS 机制(如 DiffServ)相互综合,实现统一的一体化的 QoS 控制机制。

## 13.8 Web 服务器集群 QoS-aware 负载均衡的策略、模型与性能分析

本节中,我们提出了将 Web 服务器进程调度中所使用的 HTTP 请求的内容和优先级与 Web 服务器集群的负载均衡策略相结合的综合控制思想。这种综合控制策略能够同时实现负载均衡和 Web 服务质量(QoS)控制。本节的另外一个贡献是提出了一种基于

随机高级 Petri 网(stochastic high-level Petri net, SHLPN)[69]的性能模型与性能分析技术,能够对 Web 服务器集群的 QoS-aware 负载均衡策略进行有效的性能建模与性能评价。为了简化模型求解的复杂性,我们提出了一种基于模型分解和迭代的近似性能分析技术,以处理模型求解的状态空间爆炸问题。感兴趣的读者可参考文献[64, 65]。

在本节中,我们将首先讨论可扩展 Web 服务器的体系结构,并提出一种负载共享模型,然后给出 SHLPN 的非形式化描述,并建立 Web 服务器集群的 SHLPN 模型。描述有关的服务器进程调度策略和负载均衡策略,并在此基础上提出一种 QoS-aware 负载均衡策略,同时对性能分析所使用的性能评价指标进行描述,然后通过两个例子的数值结果展示 QoS-aware 负载均衡策略的优越性。接下来,提出一种近似分析技术来处理模型求解的状态空间爆炸问题,给出与精确求解相对应的近似数值结果,并进行验证。最后进行总结,并讨论进一步的研究工作。

### 13.8.1 可扩展的 Web 服务器体系结构与负载共享模型

Arlitt 和 Lin 对 1998 年法国世界杯足球赛官方网站的 Web 服务器工作负载进行了统计和分析[70]。根据他们的研究,世界杯期间该网站的 Web 服务器平均每分钟接收 10756 个 HTTP 请求,两个月期间共收到 1 352 804 107 个请求,传输的全部数据量将近 5TB,服务器存储的数据为 0.3GB。由此可见,一个单独的 HTTP 服务器显然无法处理数据量如此庞大的请求负载,因此需要可扩展的 Web 服务器体系结构来解决服务器超载和提供高性能的服务器。

下面分别讨论 Web 服务器可扩展性的实现方法,Web 服务器集群中的负载均衡问题以及端到端 QoS 和 Web 服务器 QoS 的概念。

实现 Web 服务器可扩展性的主要方法是对 Web 服务器的内容进行镜像,各个镜像服务器可以放置于地理上位置不同的区域并且具有各自的 IP 地址。各服务器之间的负载均衡可以通过 DNS(domain name services)服务器来实现,如图 13.8.1 所示。在图中,我们描述了将一个请求的 URL 映射到某 IP 地址的过程。当多个镜像服务器同时可用时,集群所在域的授权域名服务器将从后端服务器的 IP 地址列表中进行选择,并且将之作为请求分配的目的地。

如图 13.8.1 所示,使用 DNS 将一个 URL 映射成 IP 地址的过程如下:(1)包含 Web 服务器 URL 的浏览器的请求从客户发往本地域名服务器;(2)请求从本地域名服务器发往根域名服务器;(3)根域名服务器应答,提供一个授权域名服务器的地址;(4)请求从本地服务器发往授权域名服务器;(5)授权域名服务器应答,为本地域名服务器提供 Web 服务器的 IP 地址;(6)本地域名服务器为浏览器提供应答,一旦客户得到服务器的 IP 地址即可发送 HTTP 请求(7)并且接收 Web 服务器返回的应答(8)。

为了减少网络的传输量和客户请求的响应时间,客户端和本地域的 DNS 服务器都可以对 Web 站点名到服务器 IP 地址的映射进行缓存。这种地址映射的缓存机制(TTL > 0)导致了集群所在域的 DNS 只能处理很少一部分地址映射请求,所以基于 DNS 的 Web 服务器集群体系结构只能实现一种粗粒度的负载均衡[47]。

这里,我们仅考虑另外一种基于请求分配器的可扩展的 Web 服务器体系结构,如

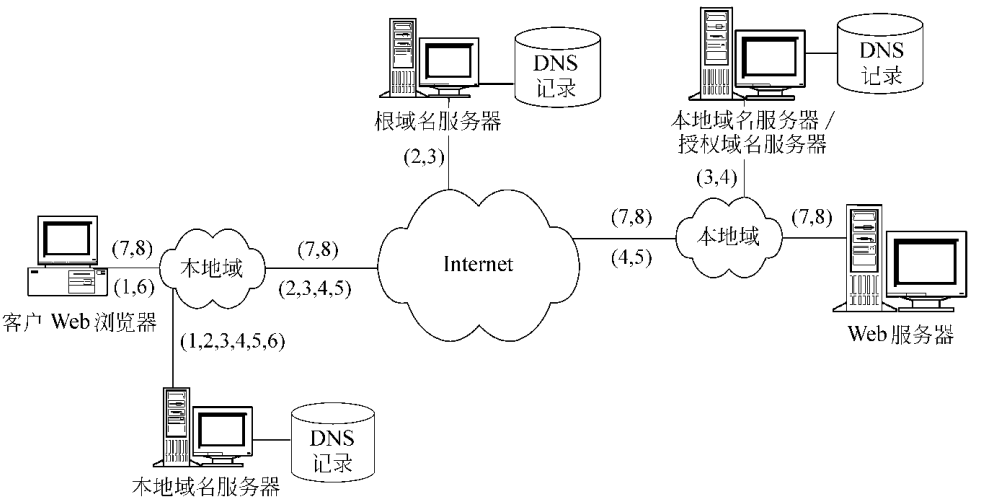



**图 13.8.1　使用 DNS 将 URL 映射成 IP 地址**

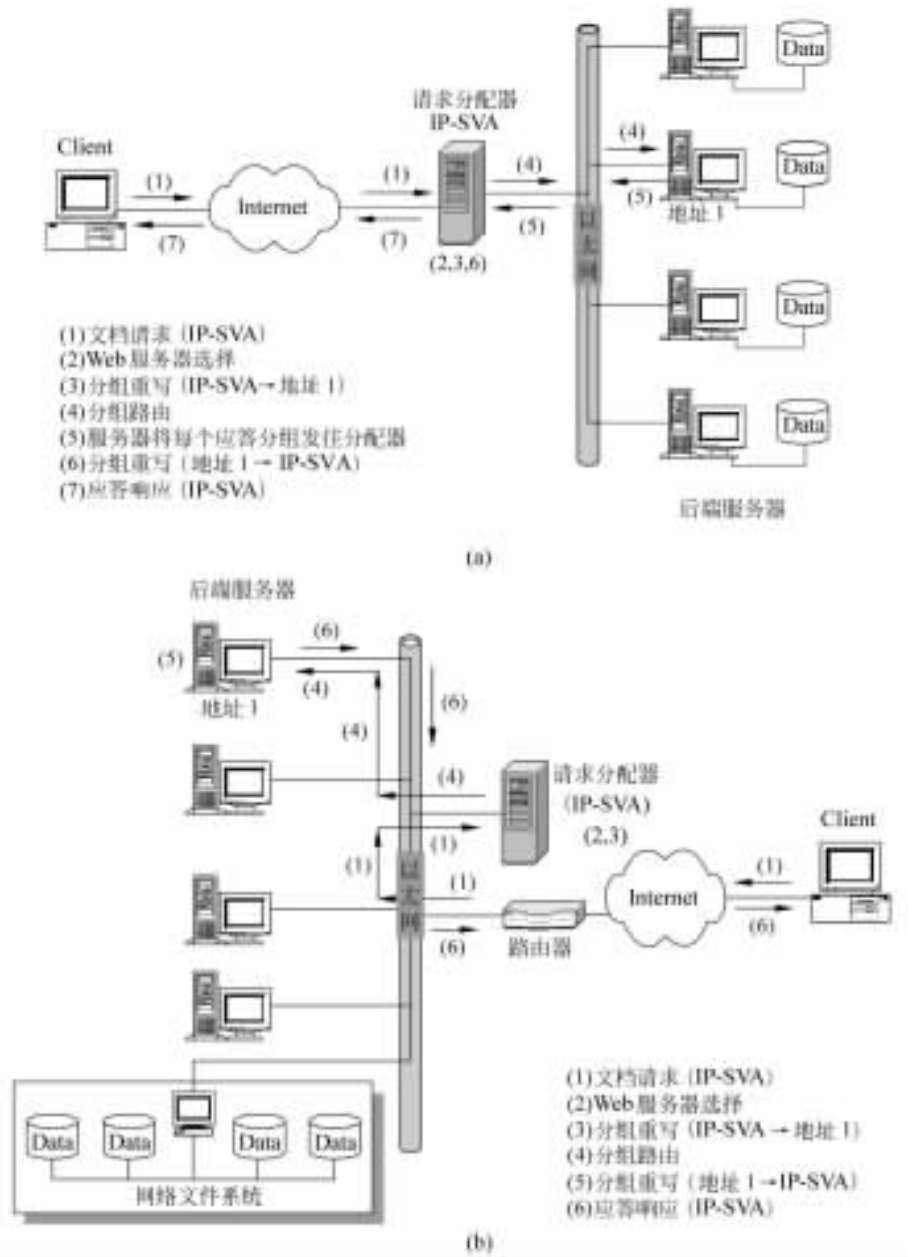
图 13.8.2 所示。在该体系结构中,集群前端的请求分配器作为到达请求的代理,负责集中地接收所有到达的 HTTP 请求,然后按照特定的负载均衡策略,将客户的请求均衡、透明地分配给集群中的后端服务器节点。该请求分配器具有一个单一的虚拟 IP 地址,能够完全控制所有到来的请求,并且实现精细粒度的负载均衡[46,47]。

如图 13.8.2 所示,Web 服务器集群系统是由分布在高速局域网上的多台后端 Web 服务器主机相互联结而成的一种服务器体系结构,采用负载均衡策略将到达的请求分配给某台后端服务器进行处理。集群中的每台服务器都能够响应任何客户的请求。通常地,信息在服务器节点间的分布方式有以下两种:

(1) 在每个服务器的本地磁盘复制全部的内容树,即镜像各台服务器的内容;

(2) 通过管理全部 Web 文件树的分布式文件系统(如 Andrew 文件系统)来共享信息。

这里只考虑第一种信息分布方式,并且假设采用第 7 层请求分配方式(L7),即集群中的每台服务器都有一份复制的内容,并且请求分配器可以采用应用层信息进行请求分配。早期的研究或产品典型地采用第 4 层交换机或 TCP 路由器作为请求分配器,即采用 L4/2 或 L4/3 集群方式,而最近的研究和产品已经开始使用第 7 层交换机(也称为 Web 交换机或内容交换机)作为请求分配器,即采用 L7 集群方式[46,59]。与第四层交换机和 TCP 路由器等 Content-unaware 请求分配器相比,第 7 层请求分配器除了能够获得第 2 ~ 4 层信息以外,还能获得应用层信息(如 URL),从而实现基于内容的请求分配。为此,第 7 层请求分配器必须连续地建立两个 TCP 连接:一个与客户端,另一个与选定的目标服务器节点。最后使用 TCP 接合[59,62]等机制将两个 TCP 连接接合起来,实现客户与服务器的直接通信。这样,第 7 层请求分配器能够实现基于内容的请求分类和优先化,从而与集群后端服务器的基于优先级的 HTTP 进程调度策略等 Web QoS 控制机制相互协调一致。由于能够实现 QoS-aware 负载均衡策略,因此在电子商务领域,第 7 层请求分配器




（a）HTTP 请求和应答均经过前端的请求分配器，请求分配器的作用相当于一个代理服务器；
（b）HTTP 响应直接返回给客户端

图 13. 8. 2　HTTP 服务器集群，前端的系统请求分配器将请求分配给后端的服务器

是比其他 Content-unaware 请求分配器更好的分配器选择。

我们提出了一种能够实现 QoS-aware 负载均衡的 Web 服务器集群系统模型，如图 13. 8. 3 所示。

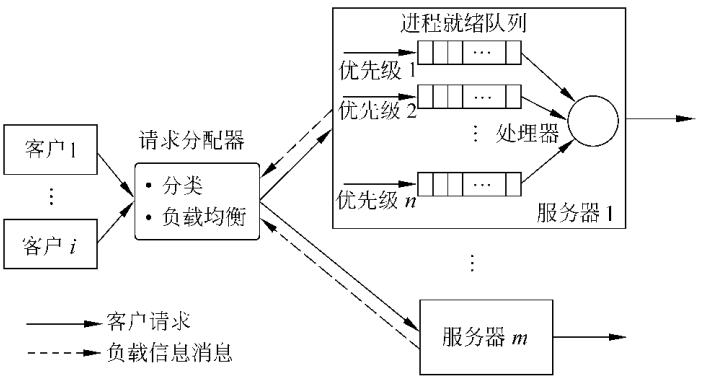



图 13.8.3 一种实现 QoS-aware 负载均衡的 Web 服务器集群模型

在此系统模型中,一个 Content-aware 请求分配器(如第 7 层交换机)作为集群系统的前端负责执行负载均衡策略。它接收所有到达的请求,并且根据所获得的应用层信息,如 URL 请求类型、文件名路径等,将请求分成多个类别。不同类别的请求分配不同的优先级。请求分配器将这些优先级别运用到负载均衡算法中,从而实现基于优先级的请求分配,即 QoS-aware 负载均衡。应该指出的是,前端请求分配器和后端服务器节点遵守相同的请求分类和优先化规则。

具体而言,为了获得请求的应用层信息,请求分配器必须首先截取每个来自客户的 TCP 连接建立请求,并且返回响应,与客户建立一个 TCP 连接。然后分配器尽可能地读取所需的应用层信息,对请求执行分类和优先化,并且基于请求的优先级作出请求分配的决定。之后,分配器与选定的后端服务器节点建立第二个 TCP 连接。最后可以使用 TCP 接合机制[59, 62]将两个 TCP 连接拼接起来,从而实现客户端到服务器节点的直接通信,即拼接后的 TCP 连接上的 IP 分组可以通过网络层直接从一个端点转发给另一个端点,无须再穿过请求分配器的 TCP 层到达其应用层。

集群的各个后端服务器节点具有相同的内容,每台服务器都能够响应任何客户的请求。所有的服务器都采用‘进程每请求’(process-per-request)的体系结构,因此每个到达的请求均对应于一个单独的进程进行处理。与请求分配器一样,服务器也要执行相同的请求分类和优先化操作。请求的优先级被映射成处理该请求的 HTTP 进程的优先级。每个服务器中都有一组进程就绪队列(ready queue),每个就绪队列对应于一个优先级别。具有相同优先级的 HTTP 进程进入相同的就绪队列进行排队,并且在内核中按照 FCFS 的规则接受处理。每个就绪队列都设有一个阈值来限定其可以容纳的最大进程数。每个服务器仅有一个处理器(CPU),它根据优先级选择不同就绪队列中的进程加以执行。这种 Web 服务器集群模型能够通过修改 Linux 操作系统下的 Apache 服务器[21]得以实现。Apache 服务器采用多进程模型并且具有‘进程每请求’的体系结构:服务器启动时生成一定数量的进程(进程池),一个请求到来时就分配一个单独的进程进行处理。在 Linux 操作系统中,实时进程的 FIFO 调度策略可以通过 Linux 提供的 SCHED_FIFO 调度类[71]得以实现。此外,该体系结构中采用了反馈机制,从而将服务器的负载信息(如每个就绪

队列中的等待进程数量)动态地报告给请求分配器,以实现负载均衡策略(如图13.8.3中的虚线所示)。

应该指出的是,这里的研究仅限于处理器的优先排队。实际上,经验显示应该对进程或者控制线程所使用的全部计算资源实施优先排队,具体而言,除了CPU,为客户请求提供响应所需的计算资源还有存储内容对象的辅助存储器以及向客户发送响应的网络设备等。但是,考虑多处优先排队的系统必须使用排队网络(queueing network)模型,因此其建模与分析相比之下就要复杂得多。另外,一旦进程超过了内核的限度,包括Linux在内的大多数操作系统都无法再基于进程的优先级而只能以FIFO的顺序处理I/O请求,因此,为了有效地执行优先排队,对操作系统内核的修改是十分必要的。

最后简单说明一下该集群系统模型中的QoS性能量度。在13.2.2节中,图13.2.2(b)已经描述了一个持续的HTTP连接的响应时间。其中,服务器的响应时间(response time)是指从客户发出HTTP请求到响应返回到达客户所经历的时间。服务器的驻留时间(residence time)是指从服务器收到HTTP请求到将应答响应移交给TCP连接的服务器端传输协议所经历的时间,该时间是由从辅助存储器或缓存中检索页面所需的时间或者获取页面中的对象所需的时间决定的。在图13.8.3所示的集群模型中,端到端的QoS指的是保证一个Internet服务响应时间的某种属性(如最大值或期望值)的能力;而Web QoS主要关注的是如何控制服务器的驻留时间。从图13.2.2(b)中可见,端到端的响应时间包括往复传输时间RTT(round trip time),因此可见,端到端的QoS保证需要网络QoS机制的支持与协作。但应该指出的是,这里的研究主要是解决Web服务器集群系统的QoS保证问题,而未考虑网络QoS的因素。

## 13.8.2 SHLPN模型

我们将采用随机高级Petri网(SHLPN)的建模与分析技术对所提出的Web服务器集群QoS-aware负载均衡策略进行性能分析与评价。在本节中,我们首先给出SHLPN的非形式化介绍,然后建立图13.8.3所示的Web服务器集群的SHLPN模型,并且为了减少模型求解计算的复杂性,对其进行精化设计以获得更为紧凑的模型。

### 13.8.2.1 SHLPN的非形式化介绍

首先简要介绍SHLPN的主要特性以及相关的概念,例如标记(token)类型、标记变量、标识(marking)和稳定状态概率。我们假设读者具备一些Petri网的基本知识,感兴趣的读者可参考文献[72]。

SHLPN[69]是一种具有指数分布的变迁实施时间的高级Petri网(high-level Petri net,HLPN)[72]。SHLPN模型中具有标记类型和标记变量的概念。SHLPN模型中的标记既可以是原子(atomic)标记,也可以是具有多个属性的复合(compound)标记。原子标记可由一个二元向量来表示:第一个变量表示其类型,第二个变量表示其标志(identity)。复合标记变量可由原子变量构成向量,其向量表示中的每一元素为一个原子标记变量。将变量引入到标记的意义是不仅可以增加网的图形描述能力,而且状态空间的抽象能力也得以进一步增强。相同类型的标记在同一子网中运行,并且具有相同的行为和相同的实施

速率。在 SHLPN 中,不带属性的标记通常用来进行同步操作。

SHLPN 模型的标识是指模型中各个位置(place)的标记的分布。变迁表示标识可能发生的变化。这种变化又称为变迁实施(firing),其过程是根据相应弧上标明的权值从该变迁的输入位置移出标记并且向其输出位置增加标记。当弧上没有标明权值时,则意味着权值为 1。变迁可能联系到一个可实施谓词(enabling predicate),该谓词可以使用相关的标记变量或者位置标识表达式进行描述。每个变迁联系一个优先级。如果变迁的可实施谓词不满足或者该变迁的优先级在多个可实施的变迁中不是最高,则该变迁不可实施。模型的状态空间包括从初始标识直到变迁实施所有可达标识的集合。

在 SHLPN 模型中,标记和变迁均可以分为两类:

- 标记:类型 1 标记用来表示环境变量(如服务器的处理器)或者用于并发活动中的同步操作。类型 2 标记用来表示作业或任务,如本节中讨论的 HTTP 进程。通常,性能研究主要关注的是类型 2 标记的行为。
- 变迁:类型 1 变迁用来表示某种逻辑关系或者决定某些条件是否满足,这类变迁的实施时间为零,称为瞬时变迁(immediate transition)。类型 2 变迁用来表示对作业的操作或对信息的处理,这类变迁具有指数分布的实施时间,称为时间变迁(timed transition)。通常,瞬时变迁的优先级高于时间变迁。

几个常用符号的意义如下:$M(x)$ 代表位置 x 的标识,即标识为 M 情况下的位置 x 中标记的数量。$M_0$ 代表模型的初始标识。$R(x)$ 是变迁 x 的实施速率。

在 SHLPN 中,如果 H 是在标识 M 下的具有相同优先级的所有可实施变迁的集合,并且 $R(t_i)$ 为 $\lambda_i$,则 H 中的任何变迁均可实施,但是这些变迁的实施概率(firing probability)却不相同,可由如下表达式描述:

$$P(t_i \mid M) = \lambda_i \Big/ \left(\sum_{t_k \in H} \lambda_k\right) \tag{13.8.1}$$

任何一个有限位置、有限标记颜色和有限变迁的 SHLPN 同构于一个一维的、连续时间的有限马尔可夫链(Markov chain, MC),SHLPN 中的标识与 MC 中的状态具有一一对应的关系。

假设一个 SHLPN 的状态空间为 $M_0$,并且具有 n 个状态,即其对应的 MC 具有 n 个状态,则可以为该 SHLPN 建立转移速率矩阵 $R = \| r_{ij} \|$,$1 \leqslant i, j \leqslant n$。令一个行向量 $X = (x_1, x_2, \cdots, x_n)$ 表示其 MC 中 n 个状态的稳定状态概率,则这些稳定状态概率可以通过下面的线性方程进行求解:

$$\begin{cases} XR = 0, \\ \sum_{i=1}^{n} x_i = 1. \end{cases} \tag{13.8.2}$$

变迁的吞吐量是性能分析中的重要指标。假定 E 是一组标识的集合,并且变迁 t 在这些标识下实施,$P[M]$ 是标识(状态)的稳定状态概率,$\lambda_m$ 是标识下变迁 t 的实施速率,则变迁 t 的吞吐量可以表示为

$$TH(t) = \sum_{M \in E} P[M] \lambda_m \tag{13.8.3}$$

位置中期望的标记数量是另外一个有用的性能度量。如果初始标识 $M_0$ 的所有可达

标识的集合为 S，并且 P[M] 是标识 M 的稳定状态概率，则位置 s 中的平均标记数量D(s)可以表示为

$$D(s) = \sum_{M \in S} P[M] M(s) \tag{13.8.4}$$

### 13.8.2.2　系统模型

在 13.8.1 节中，我们提出了一种能够实现 QoS-aware 负载均衡的 Web 服务器集群系统模型，下面给出该系统模型的具体规定和描述：

（1）到来的请求被分为 n 类，并且相应地分配 n 个优先级别。对于 $1 \leqslant i < k \leqslant n$，优先级 i 高于优先级 k。优先级为 i 的请求表示为 $r_i$；

（2）集群包含 m 个后端 Web 服务器节点，其中第 j 个服务器表示为 $S_j$（$1 \leqslant j \leqslant m$）；

（3）每个 Web 服务器具有 n 个进程就绪队列，每个队列对应于一个优先级别。请求的优先级被映射成处理请求的 HTTP 进程的优先级，因此同一等待队列中的进程负责处理相同优先级的请求。在同一队列中排队的进程按照 FIFO 的方式依次执行，所有 n 个就绪队列共享服务器的同一个处理器。每个就绪队列设有一个阈值以限制挂起（pending）进程的最大数量；

（4）客户请求的到达为泊松过程（Poisson process）。请求 $r_i$ 的平均到达速率为 $\lambda_i$，换言之，请求 $r_i$ 的到达间隔时间服从均值为 $1/\lambda_i$ 的指数分布。当某个进程就绪队列已满时，它拒绝接收任何新到的该类请求；

（5）不同就绪队列中的 HTTP 进程的服务时间服从均值不同的指数分布。服务器 $S_j$ 对请求 $r_i$ 进行处理的平均服务速率为 $\mu_{ij}$。

在图 13.8.4 中，我们给出了如图 13.8.3 所示的能够实现 QoS-aware 负载均衡的 Web 服务器集群的 SHLPN 模型。

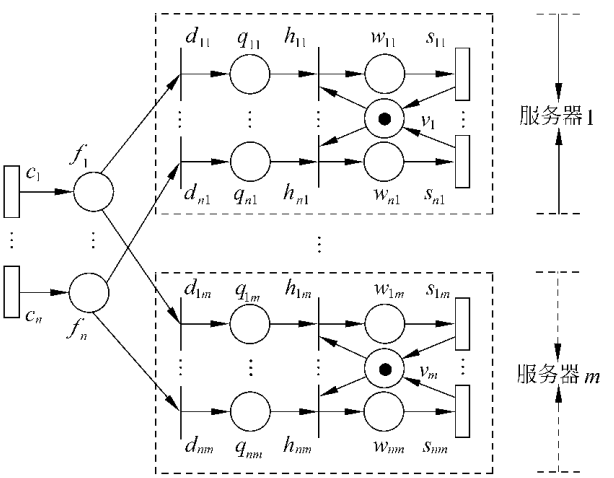



图 13.8.4　图 13.8.3 中 Web 服务器集群的 SHLPN 模型

在此 SHLPN 模型中，长方形代表时间变迁，粗实线代表瞬时变迁。请求的到达和服务均由时间变迁表示，并且联系指数分布的实施时间。服务的速率可与系统的状态相关。

请求分配到后端服务器节点和竞争处理器资源可由瞬时变迁表示,它们不占用处理时间(即实施时间为 0),而且可以联系随机开关(即实施概率)。模型中的变迁有实施优先级:当几个变迁同时可实施时,优先级高的可实施,而优先级低的则不能实施;相同优先级的变迁则都有实施的可能性。瞬时变迁比时间变迁有更高的实施优先级。在模型中,允许变迁实施的条件用变迁的可实施谓词规定,当变迁谓词条件不能满足时,变迁不能实施。对于没有可实施谓词的变迁,其可实施谓词为永真。模型中圆圈代表位置,位置中包含由黑点表示的标记。HTTP 进程就绪队列由位置表示,队列中挂起进程的数量由这些位置的标识表示。该模型中的标记既可以表示 HTTP 服务器进程,又可以表示处理器资源,并且不同优先级的进程可以由不同颜色的标记表示。图 13.8.4 中仅标出了处理器位置中的标记,而其他位置中标记的表示从略。模型中符号的第一个下标表示请求的类别(即进程的优先级别),第二个下标表示请求分配的目标服务器节点。如果符号只有一个下标,则它或者表示请求的类别,或者表示目标服务器,视具体符号而定。

下面给出图 13.8.4 模型中各变迁和位置的具体意义(其中 $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$):

- 变迁

$c_i$:表示请求 $r_i$ 的到达,其平均实施速率为 $\lambda_i$。

$d_{ij}$:表示将请求 $r_i$ 分配给后端服务器节点 $S_j$。由于一个到达的请求可以有多个可选的目标服务器,因此该变迁可以联系一个实施概率。

$h_{ij}$:表示服务器 $S_j$ 选择优先级为 i 的进程进行处理。

$s_{ij}$:表示服务器 $S_j$ 对请求 $r_i$ 进行处理,其平均服务速率为 $\mu_{ij}$。

- 位置

$f_i$:表示对请求 $r_i$ 进行分配的位置,它仅代表了请求分配器的部分功能。全部 $f_i$ $(1 \leqslant i \leqslant n)$ 的集合代表了整个请求分配器的功能。

$q_{ij}$:表示服务器节点 $S_j$ 中优先级为 i 的进程就绪队列,其容量为 $b_{ij}$。

$w_{ij}$:表示服务器 $S_j$ 中优先级为 i 的进程的运行状态。因为每个服务器只有一个处理器,该位置的容量为 1,每次至多只能有一个进程处于此运行状态。

$v_j$:表示服务器 $S_j$ 的处理器资源,该位置中的标记表示惟一的处理器。

### 13.8.2.3 模型精化

一般情况下,为了分析图 13.8.4 中 SHLPN 模型的性能,可以根据该模型构造相应的马尔可夫链。基于马尔可夫链和状态转换速率,可以构造状态转换矩阵并且进而求得所有状态的稳定状态概率,从而最终计算出模型系统的各性能参数。随机 Petri 网软件包 SPNP(stochastic Petri net package)[73] 就是基于这种思路设计出来的。当模型的规模不大时,这种方法可以直接用来对 SHLPN 模型进行性能分析。但是一般情况下,具有 n 个队列位置的 SHLPN 模型等价于一个 n 维的马尔可夫链,所以图 13.8.4 中的模型具有 $m \times n$ 维的马尔可夫链。随着系统模型参数 m,n 和 $b_{ij}$ 的增加,该马尔可夫链的状态空间呈指数性增长。当状态数量超过一定限制后,当前的一般计算机的存储和计算能力将无法忍受,从而使问题成为不可实际求解。这种情况又称为状态空间爆炸(state-

space explosion)。

在本节中,我们首先使用SPNP软件对经过初步精化设计的SHLPN模型进行性能分析,从而考察QoS-aware负载均衡策略的性能优越性。进一步,后面的13.8.5节将提出一种基于模型分解和迭代的近似分析技术来处理模型的状态空间爆炸问题。

本节首先使用变迁可实施谓词和实施速率函数对图13.8.4中模型的结构进行精化处理,将一个复杂的模型转变成一个等价的、紧凑的、具有减小的状态空间的模型,从而达到初步实现减少模型求解复杂性的目的。模型精化技术的一个重要手段就是删除瞬时变迁,转移实施谓词,削减子模型图形中的联系。图13.8.5显示了经过精化设计后的图13.8.4中的模型。

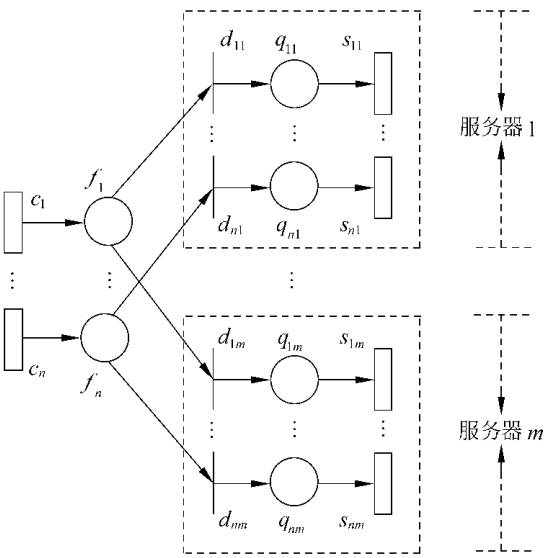



图13.8.5 精化设计后的图13.8.4中的SHLPN模型

与图13.8.4中的SHLPN模型相比,图13.8.5已经删除了位置$w_{ij}$,$v_j$,变迁$h_{ij}$以及一些相关的弧和标记($1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$)。原来由被删除的瞬时变迁$h_{ij}$所表示的多个队列对单一处理器的竞争关系现在直接由图13.8.5中变迁$s_{ij}$的可实施谓词和实施概率描述。所以,在图13.8.5中,变迁$s_{ij}$是否可实施不但依赖于位置$q_{ij}$中的标识,而且还依赖于同一服务器节点中其他队列位置中的标识情况。

## 13.8.3 QoS-aware负载均衡策略及其性能评价指标

对于图13.8.3所示的Web服务器集群系统而言,其请求分配器可以采用多种不同的负载均衡策略,后端Web服务器集群节点也可以采用多种不同的HTTP进程调度策略。本节首先介绍所采用的HTTP进程调度策略和负载均衡策略,然后基于这两种策略提出一种QoS-aware负载均衡策略,并且描述性能分析中所使用的性能评价指标,给出具体的性能求解公式。

### 13.8.3.1 策略描述

1. HTTP 服务器进程调度策略

考虑在 Web 服务器集群的后端服务器采用“绝对优先级”(strict priority, SP)策略对 HTTP 服务器进程进行调度。这是一个基于优先级的 QoS-aware 策略。在下面的“3. QoS-aware 负载均衡策略”一节中,此策略所使用的请求优先级将被集成到请求分配器的负载均衡策略中去,从而实现一个 QoS-aware 负载均衡策略。

SP 策略

此策略按照优先级从高到低的顺序执行调度,具有最高优先级的 HTTP 服务器进程总是最先得到处理。这样,对于一个就绪进程而言,如果比其优先级高的进程就绪队列中尚有未被执行的进程,则该低优先级进程就只能等待。

SP 调度策略可以由图 13.8.5 中变迁 $s_{ij}$ 的可实施谓词和实施速率加以描述。

变迁 $s_{ij}$ 的可实施谓词可以表达为

$$(M(q_{ij}) > 0) \wedge (\forall k, 1 \leqslant k \leqslant i, M(q_{kj}) = 0)$$

特别地,变迁 $s_{1j}$ 的可实施谓词为 $M(q_{1j}) > 0$。

变迁 $s_{ij}$ 的实施概率可以表达为

$$P(s_{ij}) = \begin{cases} 1, & \text{若 } i \in Q; \\ 0, & \text{其他} \end{cases}$$

其中,$Q = \{k \mid M(q_{kj}) > 0$ 且 $M(q_{lj}) = 0$,对于 $\forall l, 1 \leqslant l < k \leqslant n\}$。

特别地,变迁 $s_{1j}$ 的实施概率可以表达为

$$P(s_{1j}) = \begin{cases} 1, & M(q_{1j}) > 0; \\ 0, & M(q_{1j}) = 0. \end{cases}$$

2. QoS-unaware 负载均衡策略

考虑典型的负载均衡策略“最少服务器进程优先”(fewest server processes first, FSPF)。该策略本质上是 QoS-unaware 的,请求分配过程没有考虑 HTTP 请求的 QoS 类别。在下面 3 中该策略将被修改成 QoS-aware 负载均衡策略。

FSPF 策略

此策略选择具有最少 HTTP 服务器进程(即具有最少客户请求负载)的集群后端服务器作为请求分配的目的地。请求分配器未考虑请求的优先级别,请求分类和优先化仅在 Web 服务器节点处实现。这样,每个到达请求的类别和优先级别只有在服务器节点处才可知道。FSPF 负载均衡策略可以由图 13.8.5 中变迁 $d_{ij}$ 的可实施谓词和实施速率加以描述。服务器节点 $S_j$ 中所有进程就绪队列的集合由符号 $q_j$ 表示,其总体容量为 $b_j$,并且 $b_j = \sum_{i=1}^{n} b_{ij}$,其中 $M(q_j) = \sum_{k=1}^{n} M(q_{kj})$。

变迁 $d_{ij}$ 的可实施谓词可以表达为

$(M(q_j) < b_j) \wedge$（对于 $\forall k, 1 \leqslant k \neq j \leqslant m, (M(q_j) \leqslant M(q_k)) \vee (M(q_k) = b_k)$.

变迁 $d_{ij}$ 的实施概率为

$$P(d_{ij}) = \begin{cases} \dfrac{1}{\| Q \|}, & \text{若 } j \in Q; \\ 0, & \text{其他} \end{cases}$$

其中 $Q = \{k \mid M(q_k) = \min(M(q_1), M(q_2), \cdots, M(q_m)) \text{且} M(q_k) < b_k\}$。

FSPF 负载均衡策略基于服务器的负载条件，因此需要一种动态反馈机制将每个后端服务器节点中的全部 HTTP 进程数量及时通报给前端的请求分配器。如图 13.8.3 所示。

3. QoS-aware 负载均衡策略

一般情况下，QoS-unaware 负载均衡策略与 QoS-aware 服务器进程调度策略具有控制目标的不一致性和不协调性，二者的联合使用有可能导致不能实现面向 QoS 的请求分配的最优化，从而可能无法实现真正的负载均衡。

一方面，QoS-unaware 的 FSPF 负载均衡策略有可能将一个新到的请求分配到一个虽然全部进程总数最少，但对应于该请求的优先级队列却已满的实际上不可用的服务器节点，从而最终导致该请求被拒绝服务。具体而言，由于 FSPF 策略未考虑新到请求的优先级，那么假如一个后端服务器节点与其他节点相比，其所有就绪队列中的挂起进程总数最少，但其对应此新到请求的优先级队列却已满，则分配器还是会根据 FSPF 策略将该请求分配到这个实际上不可用的服务器，虽然其他服务器中的该优先级队列并未满甚至可能负载很轻。

另一方面，一个后端服务器节点具有最少的 HTTP 就绪进程总数并不能说明该节点对于一个新到的请求负载最轻，即不能保证向该请求提供最好的服务。这是因为，对于一个被分配到某服务器的新到的请求，只有服务器中优先级高于或等于该请求的就绪进程才能影响到它的等待时间，而优先级低于它的就绪进程根本影响不了它的执行，因为服务器的 SP 进程调度策略总是优先执行优先级高的服务器进程。

综上所述，为了实现基于 QoS 的负载均衡，请求分配器的负载均衡策略必须要考虑到客户请求的优先级别。这里，基于前述的 FSPF 和 SP 策略提出了一种扩展的 FSPF（extended FSPF，E-FSPF）策略，从而实现了基于优先级的 QoS-aware 负载均衡。

E-FSPF 策略

此策略是通过修改前述的 FSPF 策略使之在负载均衡过程中考虑到客户请求的优先级别而实现的，目的是更好地与后端服务器节点的基于 QoS 的 SP 进程调度策略相配合，使集群系统同时实现系统均衡和 Web QoS 控制。前面的图 13.8.3 已经给出了一个能够实现此 QoS-aware 负载均衡策略的 Web 服务器集群模型，在该系统模型中，Content-aware 的请求分配器能够区分请求的优先级。对于一个新到的请求，E-FSPF 策略并非根据每个服务器中的所有 HTTP 进程计算该服务器的负载，而只是对优先级高于或等于新到请求的进程进行统计作为对应该请求的服务器负载，而对优先级低于新到请求的进程则不予考虑。打一个形象的比喻，在 E-FSPF 策略中，一个新到的请求只能‘看见’每个后端服务

器中优先级高于或等于它的那些就绪进程,并且据此对每个服务器当时的负载情况进行评估,而根本“看不见”那些优先级低于它的就绪进程。

同样,E-FSPF 负载均衡策略可由图 13.8.5 中变迁 $d_{ij}$ 的可实施谓词和实施速率加以描述。

定义变量 $M_i(q_j)$ 表示服务器 $S_j$ 中优先级高于或等于新到请求 $r_i$ 的所有服务器进程的数量,即 $M_i(q_j) = \sum_{k=1}^{i} M(q_{kj})$,且称其为对于请求 $r_i$ 的“有效队列长度”。

变迁 $d_{ij}$ 的可实施谓词可以表达为

$(M(q_{ij}) < b_{ij}) \wedge (\text{对于 } \forall k, 1 \leqslant k \neq j \leqslant m, (M_i(q_j) \leqslant M_i(q_k)) \vee (M(q_{ik}) = b_{ik}))$.

变迁 $d_{ij}$ 的实施概率为

$$P(d_{ij}) = \begin{cases} \dfrac{1}{\| Q \|}, & \text{若} \quad j \in Q; \\ 0, & \text{其他} \end{cases}$$

其中 $Q = \{k \mid M_i(q_k) = \min(M_i(q_1), M_i(q_2), \cdots, M_i(q_m)) \text{且} M(q_{ik}) < b_{ik}\}$。

E-FSPF 负载均衡策略基于服务器的负载条件,因此同样需要一种动态反馈机制,将每个后端服务器节点中各优先级队列的 HTTP 进程数目及时通报给前端的请求分配器。如图 13.8.3 所示。

### 13.8.3.2 性能评价指标

在一般的 SHLPN 中,性能评价指标可以基于模型的稳定状态概率求得。下面介绍本节在对 QoS-aware 负载均衡策略进行性能分析和评价时所考察的一些性能指标。

请求的吞吐量(throughput)是一个重要的性能指标。状态(标识)M 的稳定状态概率用 P[M] 表示。稳定状态下变迁 $s_{ij}$ 的吞吐量 $TH(s_{ij})$ 可以表示为

$$TH(s_{ij}) = \mu_{ij} \sum_{M \in E} P[M] \tag{13.8.5}$$

其中 E 是能使变迁 $s_{ij}$ 实施的所有标识的集合,变迁 $s_{ij}$ 的可实施条件在其可实施谓词中得以描述。

Web 服务器集群中请求 $r_i$ 的吞吐量 $TH_i$ 可以表达为

$$TH_i = \sum_{j=1}^{m} TH(s_{ij}) \tag{13.8.6}$$

整个 Web 服务器集群系统的吞吐量 TH 为

$$TH = \sum_{i=1}^{n} \sum_{j=1}^{m} TH(s_{ij}) \tag{13.8.7}$$

Web 服务器的响应时间(response time)是另外一个重要的性能指标。应该指出,这里的响应时间实际上是指请求在服务器的驻留时间,即从服务器收到 HTTP 请求到将应答响应移交给 TCP 连接的服务器端传输协议所经历的时间,包括请求的排队时间和处理时间,而对从客户到服务器的网络上的传输延迟则未予考虑。

某一队列位置 q 中的平均标记数量表示为 D(q),其定义参见式(13.8.4)。服务器

变迁 $s_{ij}$ 的响应时间 $RT_{ij}$ 可以表示为

$$RT_{ij} = D(q_{ij}) / TH(s_{ij}) \tag{13.8.8}$$

集群系统中请求 $r_i$ 的响应时间为

$$RT_i = \frac{\sum_{j=1}^{m} D(q_{ij})}{\sum_{j=1}^{m}} TH(s_{ij}) \tag{13.8.9}$$

拒绝概率是另外一个有用的性能指标，因为客户的请求被不可用的服务器拒绝将严重影响用户感知的 Web QoS。就绪队列位置 $q_{ij}$ 的拒绝概率 $RP_{ij}$ 可以表示为

$$RP_{ij} = P[M(q_{ij}) \geqslant b_{ij}] \tag{13.8.10}$$

集群系统中请求 $r_i$ 的拒绝概率 $RP_i$ 可以表示为

$$RP_i = 1 - \frac{TH_i}{\lambda_i} \tag{13.8.11}$$

整个 Web 服务器集群系统的拒绝概率 RP 为

$$RP = 1 - \frac{TH}{\sum_i \lambda_i} \tag{13.8.12}$$

## 13.8.4　数值结果

在本节中，我们使用随机 Petri 网软件包 SPNP 对图 13.8.5 中的 SHLPN 模型进行性能分析。在性能分析的数值例子中，将 HTTP 进程调度策略 SP 分别与 QoS-unaware 负载均衡策略 FSPF 和 QoS-aware 负载均衡策略 E-FSPF 进行组合，通过性能对比，考察了 QoS-aware 负载均衡策略给系统带来的 QoS 性能改进。

在性能分析的数值例子中，对于图 13.8.5 中 SHLPN 模型的系统参数值，如集群中服务器节点的数目、优先级队列中挂起进程数的最大阈值、服务器的服务速率等，可以根据 13.8.1 节中所讨论的实际的 Apache 服务器系统进行选取。为了避免状态空间爆炸并简化模型的求解，不失一般性，这里给出的数值例子仅考虑了具有两个后端服务器节点的 Web 服务器集群系统。此外，在真实的系统实现中，可以根据实际情况的需要为客户的请求配置多个优先级别。在下面的例子中，我们分别考察了客户请求（服务器进程）分为两个和三个优先级别的情况。

### 13.8.4.1　两个优先级的例子

本例分析客户请求分为两种优先级别情况下的图 13.8.5 中 SHLPN 模型的性能。

假定第 1 类请求 $r_1$ 具有高优先级，第 2 类请求 $r_2$ 具有低优先级。各服务器节点的就绪进程队列的阈值为 $b_{11} = b_{21} = b_{12} = b_{22} = 12$。假定两个后端服务器是异构的，并且每个服务器可以为不同类型的请求提供不同的服务速率：$\mu_{11} = 250.0$ 请求/s，$\mu_{21} = 200.0$ 请求/s，$\mu_{12} = 200.0$ 请求/s，$\mu_{22} = 160.0$ 请求/s，假设这些服务速率可以通过统计的方法而事先获知。在性能分析中，请求 $r_1$ 和 $r_2$ 的到达速率同步地从 50.0 请求/s 单调递增到

400.0 请求/s,两类请求的到达速率总是保持一致,即 $\lambda_1 = \lambda_2$。因此,两类请求的联合到达速率的最大值为 $\lambda = \lambda_1 + \lambda_2 = 800.0$ 请求/s。

在这个数值例子中,SP 策略分别与 FSPF 策略和 E-FSPF 策略进行组合,实现性能的对比分析,从而考察使用 QoS-aware 和 QoS-unaware 负载均衡策略之间的性能差异。

图 13.8.6 显示了请求的吞吐量(单位:应答数/s)随着全部请求到达速率(即第 1、2 类请求的到达速率之和)的增加而变化的情况。图 13.8.7 显示了请求的拒绝概率随着全部请求到达速率的增加而变化的情况。

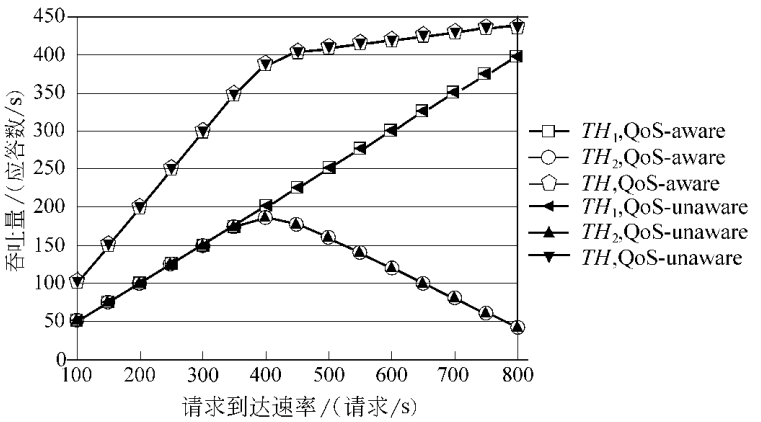



图 13.8.6 采用 QoS-aware 和 QoS-unaware 负载均衡策略的吞吐量比较

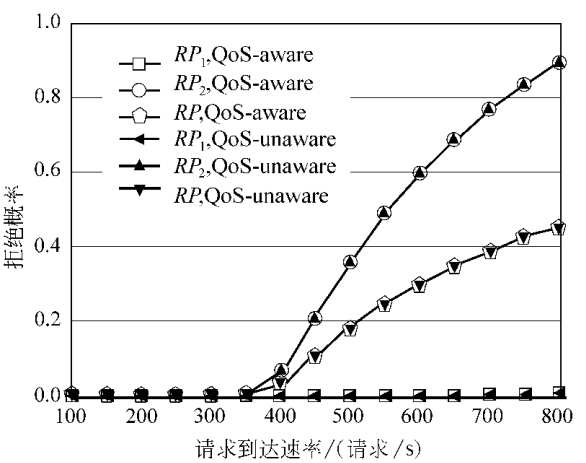



图 13.8.7 采用 QoS-aware 和 QoS-unaware 负载均衡策略的拒绝概率比较

由图 13.8.6 可见,将采用 QoS-aware 负载均衡策略 E-FSPF 和 QoS-unaware 负载均衡策略 FSPF 的两种情况相比较,$TH_1$,$TH_2$和 TH(分别是集群系统的第 1、2 类请求和全部请求的吞吐量)均没有显著的区别。更精确地讲,根据所获得的具体性能分析结果,在图 13.8.6 中,$TH_1$在采用 E-FSPF 策略情况下的数值实际上总是大于采用 FSPF 策略情况下的数值,但是最多只增加了 0.05%,而 $TH_2$在采用 E-FSPF 策略情况下的数值实际上总是

小于采用FSPF策略情况下的数值，但是最多只减少了0.09%。由此可见，采用QoS-aware负载均衡策略E-FSPF能给高优先级的请求带来的吞吐量方面的性能改进是不明显的。与图13.8.6中的结论相同，图13.8.7显示了将采用E-FSPF策略与采用FSPF策略的两种情况相比较，$RP_1$，$RP_2$和RP（分别是集群系统的第1、2类请求和全部请求的拒绝概率）均没有显著的区别。

但是，由图13.8.6和图13.8.7可见，无论是采用E-FSPF策略还是采用FSPF策略，高优先级请求与低优先级请求的吞吐量和拒绝概率之间均有显著的区别。具体而言，当全部请求的到达速率小于350请求/s时，$TH_1$，$TH_2$和TH均与第1、2类请求以及全部请求的到达速率相等，而$RP_1$，$RP_2$和RP则都趋近于0。这表明，当请求的到达速率较低时，服务器能够轻易地处理所有到来的客户请求，因此无论是高优先级请求还是低优先级请求均能获得良好的服务。但是随着请求速率的不断增加，服务器集群的处理能力在请求到达速率大约为350请求/s时达到饱和。从这时开始，低优先级的请求开始被越来越多地拒绝：图13.8.6中的$TH_2$曲线在横坐标大约400左右达到最大值，然后随着请求速率的增加而迅速下降，图13.8.7中的$RP_2$曲线在横坐标大于400以后开始急剧上升；而高优先级的请求却直到请求速率达到800请求/s（大约是集群处理能力的2倍）仍没有遭受明显的拒绝：图13.8.6中$TH_1$曲线一直随着请求速率的增加而呈直线上升，图13.8.7中的$RP_1$曲线则一直趋近于0。在请求速率为800请求/s时，高优先级的请求$r_1$仅受到0.78%（QoS-aware情况）和0.79%（QoS-unaware情况）的拒绝，而低优先级的请求$r_2$的拒绝概率却已达到89.38%（QoS-aware情况）和89.37%（QoS-unaware情况）。

图13.8.8中显示了在采用E-FSPF策略和FSPF策略两种情况下高优先级请求$r_1$的响应时间$RT_1$的比较。

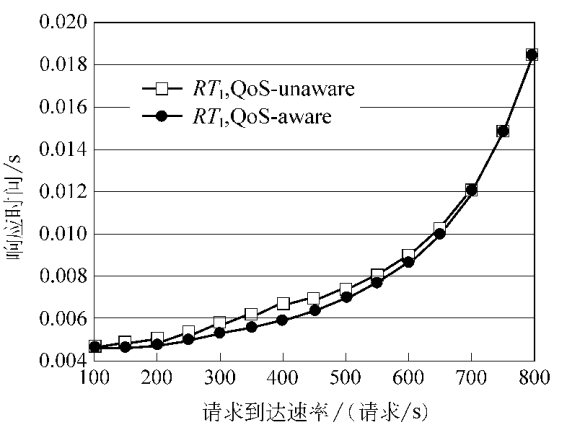



图13.8.8 采用QoS-aware和QoS-unaware负载均衡策略的响应时间$RT_1$比较

根据图13.8.8中的数值结果，高优先级请求的响应时间$RT_1$在采用QoS-aware负载均衡策略E-FSPF情况下的数值总是显著地小于采用QoS-unaware负载均衡策略FSPF情况下的数值，并且最多时减少了11.94%。

图 13.8.9 显示了在采用 E-FSPF 策略和 FSPF 策略两种情况下低优先级请求 $r_2$ 的响应时间 $RT_2$ 的比较。图 13.8.10 对图 13.8.9 的左半部分进行了放大,目的是使两条曲线之间的性能差别看得更加清楚。

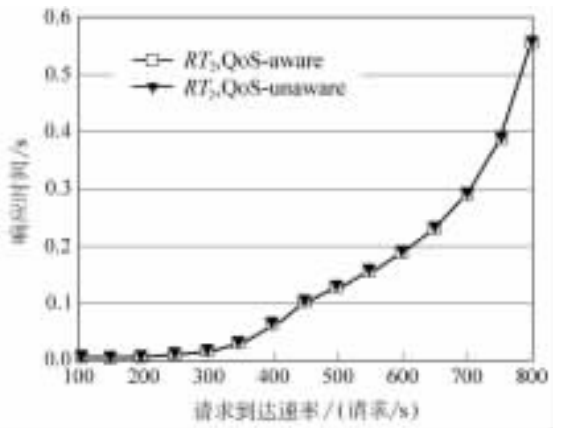



图 13.8.9 采用 QoS-aware 和 QoS-unaware 负载均衡策略的响应时间 $RT_2$ 比较

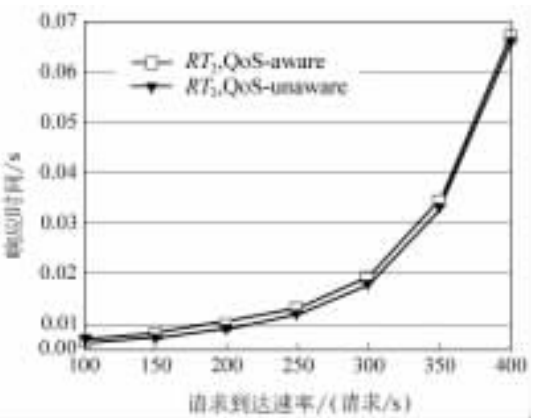



图 13.8.10 对图 13.8.9 的左半部分进行放大

从图 13.8.10 中可见,低优先级请求的响应时间 $RT_2$ 在采用 QoS-aware 负载均衡策略 E-FSPF 情况下的数值总是略微大于采用 QoS-unaware 负载均衡策略 FSPF 情况下的数值,并且根据图 13.8.9 中的数值结果,最多时增加了 9.89%。

根据图 13.8.8 ~ 图 13.8.10 可以得出结论:采用 QoS-aware 负载均衡策略 E-FSPF 能够显著地改善高优先级请求的响应时间,但作为代价,低优先级请求的响应时间却受到了一些影响。

由于图 13.8.6 中已经显示了请求的吞吐量受不同负载均衡策略的影响并不显著,而请求的响应时间却受到了显著的影响,那么根据前面的排队论公式(13.8.9)可以推断:高优先级就绪队列的平均长度在采用 QoS-aware 负载均衡策略 E-FSPF 的情况下必定小

于采用 QoS-unaware 负载均衡策略 FSPF 的情况。我们在性能分析中获得的平均队列长度 $D(q_{ij})$ 的数值结果证实了这一点。

表 13.8.1 列出了一些平均队列长度和吞吐量的数值结果。$D_i(i=1,2)$ 表示集群中第 i 类请求的联合平均队列长度，即 $D_1 = D(q_{11}) + D(q_{12})$，$D_2 = D(q_{21}) + D(q_{22})$。如表 13.8.1 中的数据所示，$D_1$ 在采用 QoS-aware 负载均衡策略 E-FSPF 情况下的数值确实小于采用 QoS-unaware 负载均衡策略 FSPF 的情况，而 $D_2$ 的情况恰好与 $D_1$ 相反。

表 13.8.1　一些平均队列长度和吞吐量的数值结果

| 性能度量 | | $\lambda(=\lambda_1+\lambda_2)$ | E-FSPF | FSPF |
|---|---|---|---|---|
| 平均队列长度 | $D_1$ | 300.0 | 0.792 0 | 0.864 7 |
| | | 350.0 | 0.975 8 | 1.087 5 |
| | | 400.0 | 1.188 0 | 1.331 0 |
| | $D_2$ | 300.0 | 2.970 3 | 2.680 1 |
| | | 350.0 | 6.043 5 | 5.720 1 |
| | | 400.0 | 12.638 3 | 12.375 7 |
| 吞吐量 | $TH_1$ | 300.0 | 150.0 | 150.0 |
| | | 350.0 | 175.0 | 175.0 |
| | | 400.0 | 200.0 | 200.0 |
| | $TH_2$ | 300.0 | 149.974 2 | 149.976 9 |
| | | 350.0 | 173.905 8 | 173.970 1 |
| | | 400.01 | 86.987 3 | 187.239 9 |

综上所述，可以得出如下结论：与 QoS-unaware 负载均衡策略 FSPF 相比，QoS-aware 负载均衡策略 E-FSPF 能够更好地对高优先级的请求实现负载均衡，从而为高优先级的请求提供更好的 Web 服务性能。

### 13.8.4.2　三个优先级的例子

本例分析客户请求分为 3 种优先级别情况下的图 13.8.5 中 SHLPN 模型的性能。

假定第 1，2，3 类请求 $r_1$，$r_2$，$r_3$ 分别具有高、中、低优先级。各服务器节点的进程就绪队列的阈值为 $b_{11} = b_{21} = b_{31} = b_{12} = b_{22} = b_{32} = 4$。两个异构的后端服务器可以为不同类型的请求提供不同的服务速率：$\mu_{11} = 180.0$ 请求/s，$\mu_{21} = 210.0$ 请求/s，$\mu_{31} = 240.0$ 请求/s，$\mu_{12} = 240.0$ 请求/s，$\mu_{22} = 280.0$ 请求/s，$\mu_{32} = 320.0$ 请求/s，假设这些服务速率可以通过统计的方法而事先获知。在性能分析中，第 1，2，3 类请求的到达速率同步地从 20.0 请求/s 单调递增到 300.0 请求/s，三类请求的到达速率保持一致，即 $\lambda_1 = \lambda_2 = \lambda_3$。因此，所有请求的最大联合到达速率为 $\lambda = \lambda_1 + \lambda_2 + \lambda_3 = 900.0$ 请求/s。

在这个数值例子中，SP 策略同样分别与 FSPF 策略和 E-FSPF 策略组合，通过性能的对比分析，从而考察 QoS-aware 和 QoS-unaware 负载均衡策略之间的性能差异。

图 13.8.11 显示了请求的吞吐量随着全部请求到达速率(即第 1,2,3 类请求的到达速率之和)的增加而变化的情况。图 13.8.12 显示了请求的拒绝概率随着全部请求到达速率的增加而变化的情况。

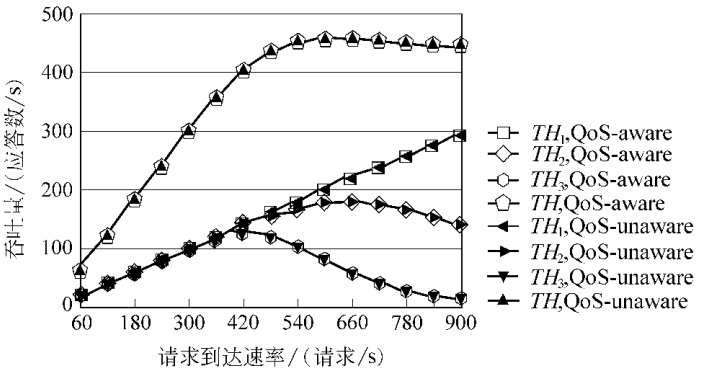



图 13.8.11 采用 QoS-aware 和 QoS-unaware 负载均衡策略的吞吐量比较

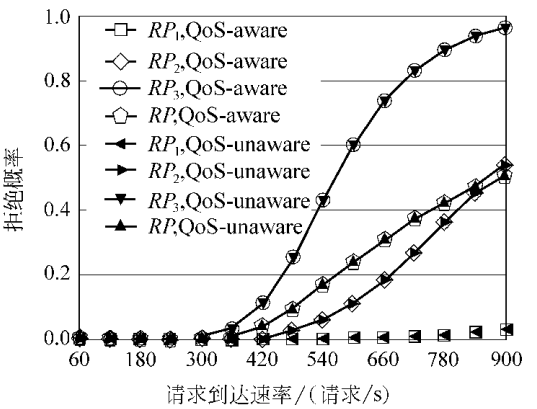



图 13.8.12 采用 QoS-aware 和 QoS-unaware 负载均衡策略的拒绝概率比较

与 13.8.4.1 节中的情形相似,在图 13.8.11 中,将采用 QoS-aware 负载均衡策略 E-FSPF 与 QoS-unaware 负载均衡策略 FSPF 的两种情况相比较,$TH_1$,$TH_2$,$TH_3$和 TH(分别对应于第 1,2,3 类请求和全部请求的吞吐量)均没有显著的区别。具体而言,根据图 13.8.11 中的具体数值结果,$TH_1$在采用 E-FSPF 策略情况下的数值实际上总是大于采用 FSPF 策略情况下的数值,但是最多只增加了 0.12%;而 $TH_2$在采用 E-FSPF 策略情况下的数值实际上总是小于采用 FSPF 策略情况下的数值,但是最多只减少了 1.12%;$TH_3$的情况稍微有些复杂:当请求的到达速率小于 660 请求/s 时,$TH_3$在采用 E-FSPF 策略情况下的数值总是小于采用 FSPF 策略情况下的数值,但是最多只减少了 1.38%;当请求的到达速率大于 660 请求/s 时,采用 E-FSPF 策略情况下的数值总是大于采用 FSPF 策略情况下的数值,但是最多只增加了 8.3%。由此可见,采用 QoS-aware 负载均衡策略 E-FSPF 给高优先级的请求带来的吞吐量方面的性能改进是不明显的。

同样，图 13. 8. 12 显示了将采用 E-FSPF 策略与采用 FSPF 策略的两种情况相比较，$RP_1$，$RP_2$，$RP_3$和 RP（分别对应第 1，2，3 类请求和全部请求的拒绝概率）均没有显著的区别。

同样，从图 13. 8. 11 和图 13. 8. 12 中可见，无论是采用 E-FSPF 策略的情况还是采用 FSPF 策略的情况，高优先级请求与低优先级请求的吞吐量和拒绝概率之间总有显著的区别：高优先级的请求总是能比低优先级的请求遭受更少的拒绝服务，从而实现更多的吞吐量。更详细的讨论与 13. 8. 4. 1 节中的情况类似，这里从略。

图 13. 8. 13 ~ 图 13. 8. 15 分别显示了在采用 E-FSPF 策略和 FSPF 策略两种情况下的高、中、低优先级请求的响应时间 $RT_1$，$RT_2$和 $RT_3$。

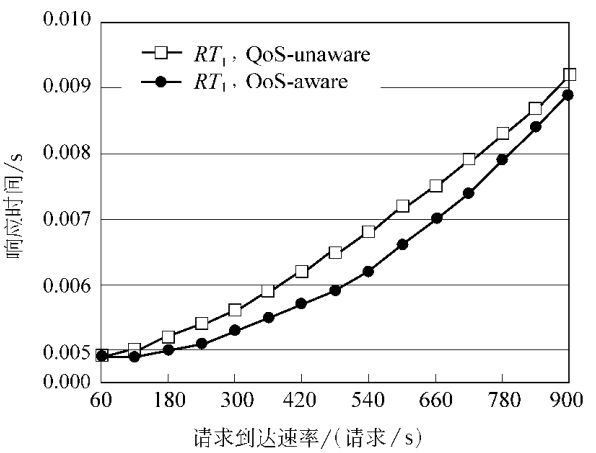



图 13. 8. 13　采用 QoS-aware 和 QoS-unaware 负载均衡策略的响应时间 $RT_1$比较

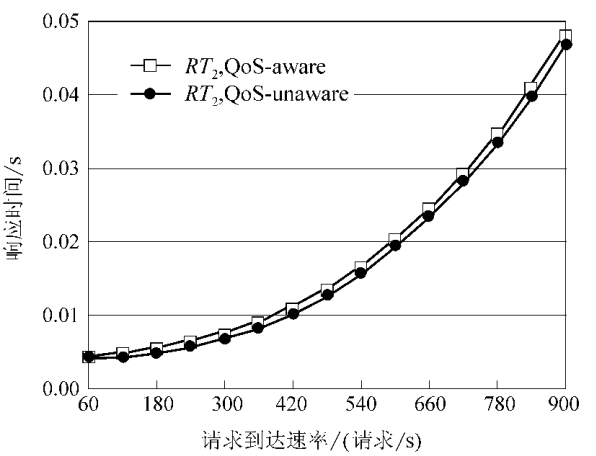



图 13. 8. 14　采用 QoS-aware 和 QoS-unaware 负载均衡策略的响应时间 $RT_2$比较

根据图 13. 8. 13 中的数值结果，高优先级请求的响应时间 $RT_1$在采用 QoS-aware 负载均衡策略 E-FSPF 情况下的数值总是显著地小于采用 QoS-unaware 负载均衡策略 FSPF 情

况下的数值,并且最多时减少了 9.23%。

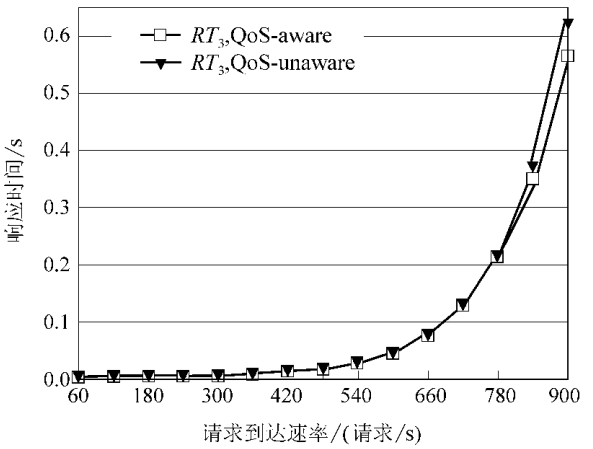



图 13.8.15 采用 QoS-aware 和 QoS-unaware 负载均衡策略的响应时间 $RT_3$ 比较

从图 13.8.14 中可见,中等优先级请求的响应时间 $RT_2$ 在采用 QoS-aware 负载均衡策略 E-FSPF 情况下的数值总是略微大于采用 QoS-unaware 负载均衡策略 FSPF 情况下的数值,并且根据图 13.8.14 中的数值结果,最多时增加了 5.88%。

图 13.8.15 中的低优先级请求的响应时间 $RT_3$ 的情况有些复杂:当请求的到达速率小于 660 请求/s 时,$RT_3$ 在采用 E-FSPF 策略情况下的数值总是大于采用 FSPF 策略情况下的数值,但是最多只增加了 5.88%;当请求的到达速率大于 660 请求/s 时,采用 E-FSPF 策略情况下的数值总是小于采用 FSPF 策略情况下的数值,但是最多时只减少了 6.36%。

从图 13.8.13 ~ 图 13.8.15 可以得出结论:采用 QoS-aware 负载均衡策略 E-FSPF 能够显著地改善高优先级请求的响应时间,但作为代价,低优先级请求的响应时间却受到了一些影响。与 13.8.4.1 节中对图 13.8.8 ~ 图 12.8.10 的讨论类似,根据本例子中的平均队列长度 $D(q_{ij})$ 的数值结果,同样可以得出如下结论:与 QoS-unaware 负载均衡策略 FSPF 相比,QoS-aware 负载均衡策略 E-FSPF 能够更好地对高优先级的请求实现负载均衡,从而为高优先级的请求提供更好的 Web 服务性能。

## 13.8.5 近似性能分析

在 13.8.2.3 节中我们已经介绍过,状态空间爆炸问题对于复杂和大规模系统模型的分析和求解仍然是极大的挑战。由于 SHLPN 模型的状态空间随着模型规模的增长而呈指数级增长,当状态数量超过一定的限制后,当前的一般计算机的存储和计算能力将无法忍受,因而使问题成为不可实际求解。

一般情况下,SHLPN 模型中实存标识(tangible marking)的数量可以看成是对应的马尔可夫过程中的状态数目,因此可以反映模型的复杂性。对于图 13.8.5 中的 SHLPN 模

型,其状态(实存标识)的数目可以通过公式 $\prod_{i=1}^{n}\prod_{j=1}^{m}(b_{ij}+1)$ 计算得出,由此可见,该模型的状态数量随着系统参数 m,n 和 $b_{ij}$ 的增加而呈指数级增长。具体而言,在 13.8.4.1 节的例子中,模型的状态数为28 561,在使用 SPNP 对模型进行分析求解时,每个数值结果的平均计算时间大约为 3 分 44 秒。在 13.8.4.2 节的例子中,模型的状态数为15 625,求解每个数值结果的平均计算时间大约为 1 分 34 秒;但是,当仅把参数 $b_{ij}$($1\leqslant i\leqslant 3$, $1\leqslant j\leqslant 2$)的值从 4 增加到 5 时,则求解每个数值结果的平均计算时间竟然超过了14 分钟;当仅把参数 $b_{ij}$ 的值增加到 8 时,我们的计算机竟然因为状态空间爆炸而无法算出结果,最终出现了一个"Abort:Not enough memory"(中止:内存不足)的出错信息。作者计算机的配置情况为:700MHz AMD Duron 处理器,128MB RAM,SPNP 运行于 Linux 操作系统下。

为了克服 SHLPN 模型的状态空间爆炸问题,本节我们提出一种基于 SHLPN 模型的分解、精化和子模型之间相互迭代方法的近似性能分析技术,以简化模型求解的复杂性。

### 13.8.5.1 近似分析技术

我们提出的 SHLPN 模型的近似分析技术主要包括以下步骤:

(1)模型的精化设计:利用 SHLPN 模型的变迁可实施谓词和变迁实施速率函数去简化模型的结构和暴露子模型的独立性。

(2)模型的分解:将一个具有接近无关的 SHLPN 模型 A 分解成 n 个子模型 $A_1$, $A_2$, …, $A_n$。

(3)确定子模型之间的输入和输出:对于每个子模型 $A_i$,设置它来自于 $A_j$($1\leqslant j\neq i\leqslant n$)的输入参数。当 $A_i$ 求解后,规定它的输出参数。

(4)迭代求解:根据输入和输出的关系,确定子模型求解的顺序。如果 $A_i$ 和 $A_j$ 属于一个循环输入关系,那么它们当中的一个必须被首先求解。但此时它的初始输入参数未知,只能通过猜测来提供。在求解过程中,每个子模型被求解后,需要执行更多的迭代,每次都使用最新的结果作为迭代执行的输入参数。如果在连续两次迭代中,所有输入参数的相对变化已在给定的容许范围内,则认为解已收敛。停止迭代。

下面以图 13.8.5 中的 SHLPN 模型为例说明如何实现上述的近似分析技术。

在 13.8.2.3 节中,图 13.8.5 模型已经过了精化设计处理。现在,将此接近无关的模型分解成多个子模型,方法是使每个子模型都成为仅包含一个进程就绪队列的独立结构。然后,对分解的子模型进行精化设计以获得紧凑的模型,并且揭示各子模型在原模型中的独立性和相互关系。图 13.8.5 中的模型经过精化和分解处理之后的模型如图 13.8.16 所示,每个独立的结构都是一个子模型,例如由时间变迁 $c_{ij}$,$s_{ij}$ 和位置 $q_{ij}$ 组成的子模型 $A_{ij}$。这样,原来图 13.8.5 中的 SHLPN 模型就被 $m\times n$ 个 $A_{ij}$ 这样的子模型所等价地表示。

与图 13.8.5 中的 SHLPN 模型相比,图 13.8.16 中的模型已经删除了位置 $f_i$、变迁 $c_i$ 以及一些相关的弧,并用符号 $c_{ij}$ 替换了 $d_{ij}$,表示请求 $r_i$ 被分配到服务器节点 $S_j$ 的时间变迁。变迁 $c_{ij}$ 的可实施谓词就是原来联系图 13.8.5 中变迁 $d_{ij}$ 的可实施谓词,变迁 $c_{ij}$ 的实施速率就是原来图 13.8.5 中变迁 $c_i$ 的实施速率。图 13.8.16 中各子模型之间的输入和输

出关系在变迁 $c_{ij}$ 和 $s_{ij}$ 的可实施谓词和实施速率中进行描述,变迁 $c_{ij}$ 和 $s_{ij}$ 的可实施与否不但依赖于该子模型中就绪队列位置 $q_{ij}$ 中的标识(状态),而且依赖于其他子模型中队列位置的标识情况。值得指出的是,在图 13.8.16 的模型中,由于队列位置的标识(状态)是时间的函数,因此不能直接作为子模型之间的输入输出参数,但是稳定状态下的标识的概率或者简单地稳定状态下队列位置中的平均标记数量,都可以作为子模型之间的输入输出参数。

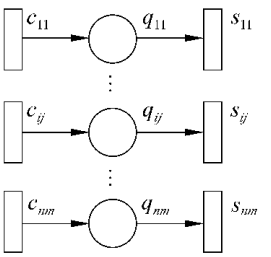



图 13.8.16 图 13.8.5 的精化、分解模型

下面对图 13.8.16 中 SHLPN 模型的时间变迁 $c_{ij}$ 和 $s_{ij}$ 的可实施谓词和实施速率进行描述,以使用上述的近似分析技术对 QoS-aware 负载均衡策略进行近似性能分析。

对于 FSPF 和 E-FSPF 两种策略,变迁 $c_{ij}$ 的可实施谓词均为 $M(q_{ij}) < b_{ij}$。

定义函数 $G[M(q_{ij}), k]$ 为子模型 $A_{ij}$ 中的变迁 $c_{ij}$ 能与任何其他 $k$ 个子模型中的变迁 $c_{ir}$ $(1 \leqslant r \leqslant m, r \neq j)$ 同时可实施的概率。在 QoS-unaware 负载均衡策略 FSPF 中,概率函数 $G[M(q_{ij}), k]$ 可以表达为

$$G[M(q_{ij}),k] = \frac{1}{k+1}\sum\left(\prod_{x\in E_j(k)} P[M(q_x) = M(q_j)] \prod_{y\in \bar{E}_j(k)} P[M(q_y) > M(q_j)]\right) \tag{13.8.13}$$

其中,$0 \leqslant M(q_{ij}) < b_{ij}$ 定义集合 $E_j = \{S_1, \cdots, S_{j-1}, S_{j+1}, \cdots, S_m\}$,它是不包括服务器节点 $S_j$ 的集群服务器索引集合。$E_j(k)$ 是 $E_j$ 的子集,由对请求 $r_i$ 进行分配时可选的 $k$ 个目标服务器组成。$\bar{E}_j(k)$ 是 $E_j(k)$ 的补集,即 $\bar{E}_j(k) = E_j - E_j(k)$,$\bar{E}_j(k)$ 中的元素是对请求 $r_i$ 而言不可用的目标服务器。

在 QoS-aware 负载均衡策略 E-FSPF 中,函数 $G[M(q_{ij}), k]$ 可以表达为

$$G[M(q_{ij}),k] = \frac{1}{k+1}\sum\left(\prod_{x\in E_j(k)\wedge(M(q_{ix})<b_{ix})} P[M_i(q_x) = M_i(q_j)] \prod_{y\in E_j(k))\wedge(M(q_{iy})<b_{iy}} P[M_i(q_y) > M_i(q_j)] \prod_{y\in \bar{E}_j(k)} P[M(q_{iy}) = b_{iy}]\right) \tag{13.8.14}$$

其中,$0 \leqslant M(q_{ij}) < b_{ij}$ 服务器 $S_j$ 对于请求 $r_i$ 的"有效队列长度" $M_i(q_j)$ 已经在 13.8.3.1 节(3)中定义。

对于 FSPF 和 E-FSPF 两种策略,当 $M(q_{ij}) = b_{ij}$ 时,$G[M(q_{ij}), k] = 0$。

进一步,对于 FSPF 和 E-FSPF 两种策略,变迁 $c_{ij}$ 的实施概率函数可以描述为

$$G[M(q_{ij})] = G[M(q_{ij}),0] + G[M(q_{ij}),1] + \cdots + G[M(q_{ij}),m-1] = \sum_{k=0}^{m-1} G[M(q_{ij}),k] \quad (13.8.15)$$

其中，$0 \leqslant M(q_{ij}) < b_{ij}$。当 $M(q_{ij}) = b_{ij}$ 时，$G[M(q_{ij})] = 0$。

因此，在 FSPF 和 E-FSPF 两种策略中，变迁 $c_{ij}$ 的实施速率为 $\lambda_i \times G[M(q_{ij})]$。

对于 HTTP 进程调度策略 SP，变迁 $s_{ij}$ 的可实施谓词为 $M(q_{ij}) > 0$。

变迁 $s_{ij}$ 的实施概率函数 $K[M(q_{ij})]$ 可以表示为

$$K[M(q_{ij})] = \begin{cases} 1, & i = 1 \\ \prod_{k=1}^{i-1} P[M(q_{kj}) = 0], & i > 1 \end{cases} \quad (13.8.16)$$

其中 $0 < M(q_{ij}) \leqslant b_{ij}$。

当 $M(q_{ij}) = 0$ 时，$K[M(q_{ij})] = 0$。

因此，变迁 $s_{ij}$ 的实施速率为 $\mu_{ij} \times K[M(q_{ij})]$。

下面对图 13.8.16 中的 SHLPN 子模型 $A_{ij}$ 进行分析。考虑到时间变迁 $c_{ij}$ 和 $s_{ij}$ 的实施速率指数性分布的假定，SHLPN 子模型 $A_{ij}$ 同构于一个连续时间马尔可夫链，并且就绪队列位置 $q_{ij}$ 中 HTTP 进程的随机过程构成一个生灭过程（birth-death process），其生速率和灭速率分别为上面推导的 $\lambda_i \times G[M(q_{ij})]$ 和 $\mu_{ij} \times K[M(q_{ij})]$。因此，根据排队论，就绪队列位置 $q_{ij}$ 中标识的稳定状态概率具有乘积形式解，并且相邻两个稳定状态之间的转移概率方程可以表达为

$$P[M(q_{ij}) = y] = \frac{\lambda_i \times G[M(q_{ij}) = y-1]}{\mu_{ij} \times K[M(q_{ij}) = y]} \times P[M(q_{ij}) = y-1], \quad y \geqslant 1 \quad (13.8.17)$$

而任一标识的稳定状态概率的乘积形式解表达如下

$$P[M(q_{ij}) = y] = \prod_{k=0}^{y-1} \frac{\lambda_i \times G[M(q_{ij}) = k]}{\mu_{ij} \times K[M(q_{ij}) = k+1]} \times P[M(q_{ij}) = 0], \quad y \geqslant 1 \quad (13.8.18)$$

其中

$$P[M(q_{ij}) = 0] = \left[1 + \sum_{y=1}^{b_{ij}} \prod_{k=0}^{y-1} \frac{\lambda_i \times G[M(q_{ij}) = k]}{\mu_{ij} \times K[M(q_{ij}) = k+1]}\right]^{-1} \quad (13.8.19)$$

至此，图 13.8.5 中的 SHLPN 模型所对应的 $m \times n$ 维的、具有非乘积解的 MC 已经被分解成图 13.8.16 所示的 $m \times n$ 个具有乘积解的简单生灭过程。进而，可以按照本节提出的近似分析技术，通过执行 $m \times n$ 个子模型之间的参数迭代，有效地求解系统模型的性能，对 QoS-aware 负载均衡策略进行性能评价。

### 13.8.5.2　近似分析的数值结果

我们使用 $m \times n$ 个 SPNP 程序进行迭代求解和近似分析，实现了上述的近似性能分析技术。我们使用上述的近似性能分析技术对包括 13.8.4.1 节和 13.8.4.2 节中的例子在

内的许多数值例子进行了分析求解。由于篇幅的限制,本节只给出 13.8.4.1 节中例子的近似性能分析结果。

在这个数值例子中,SP 策略分别与 FSPF 策略和 E-FSPF 策略进行组合,实现性能的对比分析,从而考察 QoS-aware 和 QoS-unaware 负载均衡策略之间的性能差异。所有的系统参数值仍与 13.8.4.1 节中的保持一致。在迭代时,可按 $A_{11}$,…, $A_{n1}$,…, $A_{1m}$,…, $A_{nm}$ 的顺序求解。在初始迭代中,首先求解子系统 $A_{11}$,由于此时它的输入参数未知,因此只能通过猜测来提供,可以将所有稳定状态概率的初值设置为 0 和 1 之间的任一值。当某个子模型被求解后,将其最新的稳定状态概率结果输出给后续迭代模型,作为后续模型的迭代输入参数。我们假定如果在连续两次迭代中,系统吞吐量参数 TH 的相对变化小于给定的容许范围 $10^{-5}$,则认为解已收敛,令迭代停止。应该指出的是,在我们进行的所有近似性能分析的实验中,迭代求解总能收敛。

图 13.8.17 显示了使用近似分析技术求得的吞吐量的数值结果。

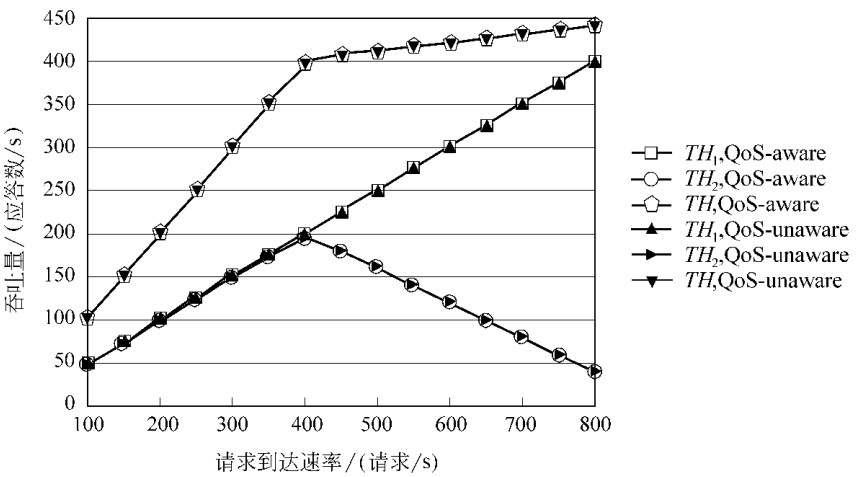



图 13.8.17 采用 QoS-aware 和 QoS-unaware 负载均衡策略的吞吐量比较(近似分析结果)

根据图 13.8.17 中的近似分析的数值结果,可以得出与 13.8.4.1 节中的图 13.8.6 相同的结论,即将采用 QoS-aware 负载均衡策略 E-FSPF 和 QoS-unaware 负载均衡策略 FSPF 的两种情况相比较,$TH_1$,$TH_2$ 和 TH(分别是集群系统的第 1 类请求、第 2 类请求和全部请求的吞吐量)均没有显著的区别。可见,采用 QoS-aware 负载均衡策略 E-FSPF 给高优先级的请求带来的吞吐量方面的性能改进是不明显的。

图 13.8.18 和图 13.8.19 分别显示了采用近似性能分析技术所求得的高优先级和低优先级请求的响应时间。

根据图 13.8.18 中的数值结果,高优先级请求的响应时间 $RT_1$ 在采用 QoS-aware 负载均衡策略 E-FSPF 情况下的数值总是显著地小于采用 QoS-unaware 负载均衡策略 FSPF 情况下的数值,并且最多时减少了 16.17%。

根据图 13.8.19 中的数值结果,低优先级请求的响应时间 $RT_2$ 在采用 QoS-aware 负载

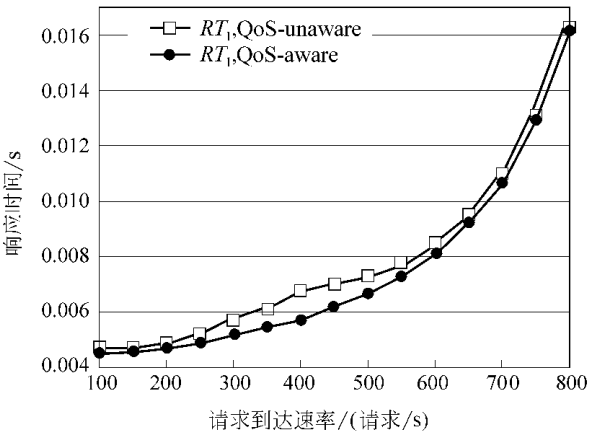
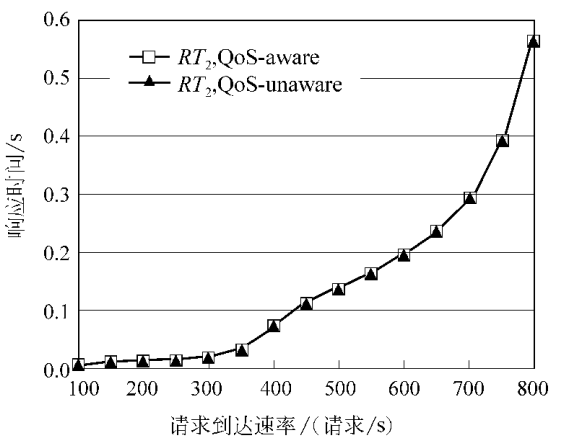



图 13.8.18　采用 QoS-aware 和 QoS-unaware 负载均衡策略的响应时间 $RT_1$ 比较(近似分析结果)




13.8.19　采用 QoS-aware 和 QoS-unaware 负载均衡策略的响应时间 $RT_2$ 比较(近似分析结果)

均衡策略 E-FSPF 情况下的数值总是略微大于采用 QoS-unaware 负载均衡策略 FSPF 情况下的数值,并且根据图 13.8.19 中的数值结果,最多时增加了 10.18%。

从图 13.8.18 和图 13.8.19 中的近似性能分析结果,可以得出与 13.8.4.1 节相同的结论:采用 QoS-aware 负载均衡策略 E-FSPF 能够显著地改善高优先级请求的响应时间,但作为代价,低优先级请求的响应时间却受到了一些影响。

与 13.8.4.1 节中的分析相似,根据平均队列长度 $D(q_{ij})$ 的数值结果,同样可以得出如下结论:与 QoS-unaware 负载均衡策略 FSPF 相比,QoS-aware 负载均衡策略 E-FSPF 能够更好地对高优先级的请求实现负载均衡,从而为高优先级的请求提供更好的 Web 服务性能。

但是,使用分解模型、迭代求解的近似性能分析技术所得到的数值结果与使用完整模型、精确求解所得到的数值结果相比存在着相对误差。图 13.8.20 ~ 13.8.22 分别对采用 QoS-aware 负载均衡策略 E-FSPF 情况下的吞吐量和响应时间的精确解与近似解进行了比

较。通过考察近似解与精确解的接近程度,对近似性能分析的数值结果的准确性进行了验证。

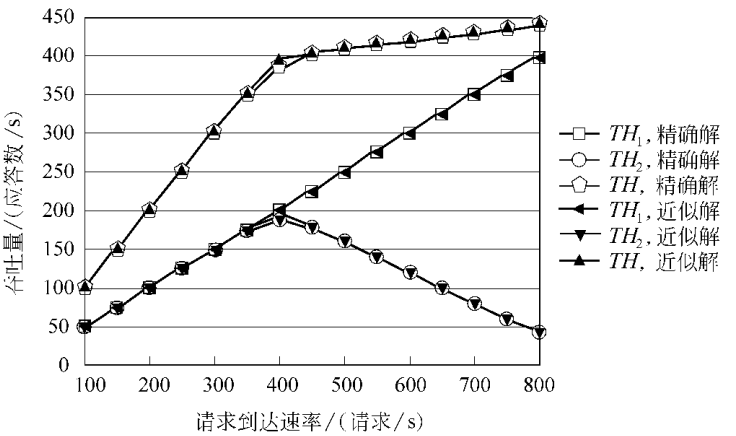



图 13.8.20 采用 E-FSPF 策略的吞吐量的近似解和精确解的比较

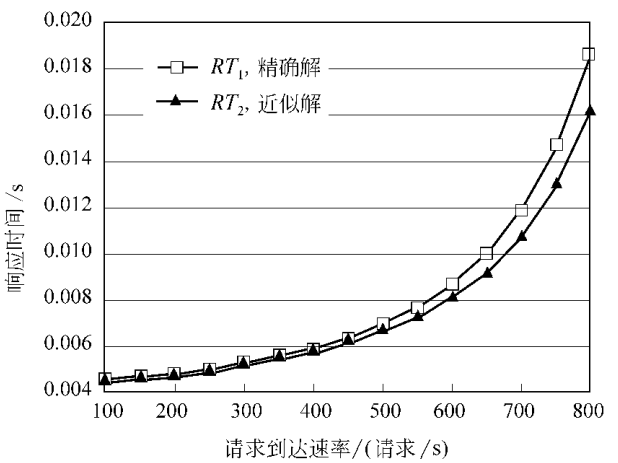



图 13.8.21 采用 E-FSPF 策略的响应时间 $RT_1$ 的近似解和精确解的比较

根据图 13.8.20 中的数值结果,吞吐量 $TH_1$,$TH_2$ 和 TH 的近似解与精确解的最大相对误差分别为 0.79%,4.75% 和 2.30%。

根据图 13.8.21 和图 13.8.22 中的数值结果,响应时间 $RT_1$ 和 $RT_2$ 的近似解与精确解的最大相对误差分别为 12.76% 和 13.79%。这些数值结果显示近似性能分析技术所得到的数值结果是足够准确的。

此外,我们还对许多其他例子进行了验证,结果同样显示,我们提出的近似性能分析技术能够得到足够准确的数值结果。

此外,在近似性能分析中,图 13.8.16 中每个子模型的状态数为 13,一共有 4 个经过分解精化的子模型,所以整个 SHLPN 模型共有 52 个状态。与精确求解时的图 13.8.5 中

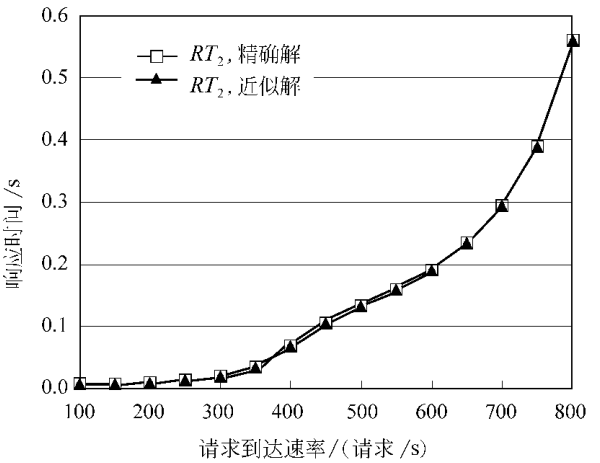



**图 13.8.22　采用 E-FSPF 策略的响应时间 $RT_2$ 的近似解和精确解的比较**

SHLPN 模型的28 561 个状态相比，本节提出的近似性能分析技术显著地降低了模型状态的复杂性。而且，在近似分析中，求解收敛所需要的平均迭代次数为 17 次，求解每个数值结果的平均计算时间大约为 15 秒，与精确求解时所需的大约 3 分 44 秒相比，模型求解的时间大大减少。可见，本节提出的近似性能分析技术有效地克服了 SHLPN 模型求解的状态空间爆炸问题。

## 13.8.6　结论

本节提出了在 Web 服务器集群系统中将 HTTP 进程调度中所使用的请求优先级与请求分配器所使用的负载均衡策略相结合的综合控制思想，以期同时实现负载均衡和 Web QoS 控制。我们提出了一种能够实现 QoS-aware 负载均衡的 Web 服务器集群体系结构，并且采用随机高级 Petri 网性能模型与分析技术，对 QoS-aware 负载均衡策略的性能优越性进行了分析和研究。为了克服模型求解的状态空间爆炸问题，简化模型求解的复杂性，我们提出了一种基于模型分解、精化和迭代的近似性能分析技术，并且以此对具体实例进行了近似性能分析。所得近似数值结果显示，该近似性能分析技术能够得到足够准确的数值结果，并且显著地降低了模型状态的复杂性，大大减少了模型求解的时间，有效地克服了模型求解中的状态空间爆炸问题。我们的分析结果表明，与 QoS-unaware 负载均衡策略 FSPF 相比，本节提出的 QoS-aware 负载均衡策略 E-FSPF 能够更好地对高优先级的请求实现负载均衡，从而为高优先级的请求提供更好的 Web QoS 性能。因此，这种 QoS-aware 负载均衡策略能够为不同类型的客户提供区分的 QoS，能够满足电子商务等应用的 Web QoS 需求。

我们的研究只是 Web 服务器集群 QoS-aware 负载均衡机制的初步探索。我们相信，仍有许多其他方面的机制和策略值得进一步深入研究，如 Web 服务器集群中的缓存(cache)性能问题[74] 等。Content-aware 请求分配器还允许不同的集群后端服务器节点具

有分割的 Web 内容数据库,因此,如何保证后端服务器节点主存储器缓存的高度局部性已成为一个重要的问题[75]。未来的工作将进一步考虑集群中的缓存性能问题。此外,一个完整的端到端 QoS 解决方案需要网络 QoS 控制和 Web QoS 控制的结合。因此,进一步的研究工作可以考虑地理上广域分布的 Web 服务器集群,将 Web QoS 控制的机制和策略与网络 QoS 机制(如 DiffServ)相互综合,实现统一的一体化的 QoS 控制机制[77]。

## 参考文献


1 林闯. 多媒体信息网络 QoS 的控制. 软件学报, 1999,10(10):1016~1024
(Lin Chuang. On QoS control of multimedia information networks. Journal of Software, 1999, 10(10): 1016~1024)

2 林闯, 单志广, 盛立杰, 吴建平. Internet 区分服务及其几个热点问题的研究. 计算机学报, 2000, 23(4):419~433
(Lin Chuang, Shan Zhi-guang, Sheng Li-jie, Wu Jian-ping. Differentiated services in the Internet: A survey. Chinese Journal of Computers, 2000, 23(4):419~433)

3 IETF working group on integrated services. http://www.ietf.org/html.charters/intserv-charter.html

4 IETF working group on differentiated services. http://www.ietf.org/html.charters/diffserv-charter.html

5 Bouch A, Kuchinsky A, Bhatti N. Quality is the eye of the beholder: Meeting users requirements for Internet quality of service. In: Proceedings of ACM Conference on Human Factors in Computing Systems. April 2000

6 Zona Research. http://www.zonaresearch.com/

7 Bhatti N, Friedrich R. Web server support for tiered services. IEEE Network, September/October 1999, 64~71

8 HP Labs. WebQoS. http://www.internetsolutions.enterprise.hp.com/webqos/

9 Cisco System Inc. Cisco LocalDirector 400 Series. http://www.cisco.com/warp/public/cc/pd/cxsr/400/index.shtml

10 Cisco System Inc. Cisco DistributedDirector. http://www.cisco.com/warp/public/cc/pd/cxsr/d/index.shtml

11 Cisco-HP Strategic Alliance. http://www.cisco.com/warp/public/756/partnership/hp/

12 The IBM WebSphere software platform and patterns for e-business—Invaluable tool for IT architects of the new economy. IBM Corporation White Paper, http://www-4.ibm.com/software/info/websphere/docs/offers/wsplat.pdf

13 Cisco CSS 11000 series content services switches. http://www.cisco.com/warp/public/cc/pd/si/11000/

14 FORE Systems, Inc. http://www.marconi.com/html/homepage/home.htm

15 Foundry Networks. ServerIron. http://www.foundrynet.com/products/webswitches/

16 Alteon WebSystems. http://www.nortelnetworks.com/corporate/acquisitions/alteon/index.html

17 Berners-Lee T, Fielding R, Frystyk H. Hyertext transfer protocol—HTTP/1.0. RFC 1945, May 1996

18 Fielding R, Gettys J, Mogul J, et al. Hyertext transfer protocol—HTTP/1.1. RFC 2068, Jan. 1997,

ftp://ftp.isi.edu/in-notes/rfc2068.txt

19 Stevens W B. TCP/IP Illustrated, Volume 1: The Protocols. Reading, MA: Addison Wesley, 1994

20 NCSA. httpd. http://hoohoo.ncsa.uiuc.edu/

21 Apache Software Foundation. http://www.apache.org/

22 Squid. http://squid.nlanr.net/Squid/

23 Zeus. http://www.zeus.co.uk/

24 Thttpd. http://www.acme.com/software/thttpd/

25 Alemeida J, Dabu M, Manikutty A, Cao P. Providing differentiated level of service in Web content hosting. In: Proc 1998 SIGMETRICS Workshop on Internet Server Performance. Piscataway, NJ: IEEE Press, 1998

26 Crovella M E, Frangioso R, Harchol-Balter M. Connection scheduling in Web servers. In: Proc 1999 USENIX Symp on Internet Technology and Systems (USITS 99). Boulder, Colorado, 1999. 243~254

27 Harchol-Balter M, Bansal N, Schroeder B, Agrawal M. Implementation of SRPT scheduling in Web servers. Technical Report CMU-CS-00-170, School of Computer Science, Carnegie Mellon University, Oct. 2000

28 Bansal N, Harchol-Balter M. Analysis of SRPT scheduling: Investigating unfairness. In: Proc ACM SIGMETRICS 2001 Conf on Measurement and Modeling of Computer Systems. Boston, MA, June 2001

29 Pandey R, Barnes J F, Olsson R. Supporting quality of service in HTTP servers. In: Proc 17th Annual SIGACT-SIGOPS Symp on Principles of Distributed Computing. 1998. 247~256

30 Li K, Jamin S. A measurement-based admission controlled Web server. In: Proc IEEE INFOCOM Conference, Tel-Aviv, Israel, 2000

31 Eggert L, Heidemann J. Application-level differentiated services for Web servers. World Wide Web Journal, 1999, 2(3): 133~142

32 Cherkasova L, Phaal P. Session-based admission control—A mechanism for improving performance of commercial Web sites. In: Proc IEEE/IFIP IWQoS 99. London, UK, June 1999

33 Abdelzaher T, Bhatti N. Web server QoS management by adaptive content delivery. In: Proc of the 7th Int l Workshop on QoS. May 1999

34 The Web QoS Group. Department of Computer Science, The University of Virginia. http://www.cs.virginia.edu/~zaher/qos/index.htm

35 Lu C, Abdelzaher T, Stankovic J, Son S. A feedback control approach for guaranteeing relative delays in Web servers. In: Proc IEEE Real-Time Technology and Applications Symposium. TaiPei, Taiwan, 2001

36 HP WebQoS Technology Overview. Hewlett-Packard Company White Paper. http://www.webqos.hp.com/infolibrary/whitepapers/qos2_whitepaper.pdf

37 Chen X, Mohapatra P, Chen H. An admission control scheme for predictable server response time for Web access. In: Proc WWW10. Hong Kong, 2001

38 Rhee Y, Hyun E, Kim T. Differentiated service on the Web by managing TCP connection. In: Proc 2001 International Conferences on Info-Tech and Info-Net (E). Beijing, Oct. 2001. 54~59

39 Banga G. Operating system support for server applications. Ph. D. Thesis. Rice University, Houston, Texas, May 1999

40 Banga G, Druschel P, Mogul J C. Resource containers: A new facility for resource management in server

systems. In:Proc 3rd USENIX Symp on Operating Systems Design and Implementation (OSDI). New Orleans, LA, 1999

41 Jones M B, Leach P J, Draves R P, Barrera J S. Modular real-time resource management in the Rialto operating system. In:Proc 5th Workshop on Hot Topics in Operating Systems (HotOS-V). Orcas Island, WA, May 1995

42 Verghese B, Gupta A, Rosenblum M. Performance isolation:Sharing and isolation in shared-memory multiprocessors. In:Proc 8th Conference on Architectural Support for Programming Language and Operating Systems. San Jose, CA, Oct. 1998

43 Bruno J, Gabber E, Ozden B, Silvberschatz A. The Eclipse operating system:Providing quality of service via reservation domains. In:Proc USENIX 1998 Annual Technical Conference. Berkeley, CA, June 1998

44 Spatscheck O, Peterson L L. Defending against denial of service attacks in Scout. In:Proc 3rd USENIX Symp on Operating Systems Design and Implementation, Feb. 1999

45 Bhoj P, Ramanathan S, Singhal S. Web2K:Binging QoS to Web servers. Hewlett-Packard Laboratories Technical Report HPL-2000-61, May 2000

46 Schroeder T, Goddard S, Ramamurthy B. Scalable Web server clustering technologies. IEEE Network, 2000, 38~45

47 Cardellini V, Colajanni M, Yu P S. Dynamic load balancing on Web-server systems. IEEE Internet Computing, 1999, 3(3):28~39

48 Colajanni M, Yu P S, Dias D M. Analysis of task assignment policies in scalable distributed Web-server systems. IEEE Trans on Parallel and Distributed Systems, 1998, 9(16) 585~600

49 Colajanni M, Yu P S, Cardellini V. Dynamic load balancing in geographically distributed heterogeneous Web servers. In: Proc IEEE 18th International Conference on Distributed Computing Systems (ICDCS 98). Amsterdan, The Netherlands, IEEE Computer Society, May 1998. 295~302

50 Cardellini V, Colajanni M, Yu P S. Geographic load balancing for scalable distributed Web systems. In: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2000). San Francisco, CA, USA, IEEE Computer Society, Aug. 2000, 20~27

51 Katz E D, Bulter M, McGrath R. A scalable HTTP server:The NCSA prototype. Computer Networks and ISDN Systems. 1994, 27:155~163

52 Dias D M, Kish W, Mukherjee R, Temari R. A scalable and highly available Web-server. In:Proc 41st IEEE Computer Society Int l Conf (COMPCON 96). Feb. 1996. 85~92

53 Egevang K, Francis P. The IP network address translation (NAT). RFC1631, May 1994. http://www.ietf.org/rfc/rfc1631.txt

54 Anderson E, Patterson D, Brewer E. The Magicrouter, an application of fast packet interposing. University of California, Berkeley, May 1996. http://www.cs.berkeley.edu/~eanders/projects/magicrouter/osdi96-mr-submission.ps

55 IBM Network Dispatcher. http://www-3.ibm.com/software/network/dispatcher/

56 Hunt G D H, Goldszmidt G S, King R P, Mukherjee R. Network dispatcher:A connection router for scalable Internet services. Computer Networks and ISDN Systems, Sept. 1999

57 Damani O P, Chung P Y, Huang Y, et al. ONE-IP:Techniques for hosting a service on a cluster of

machines. Computer Networks and ISDN Systems, 1997, 29:1019~1027

58 Garland M, Grassia S, Monroe R, Puri S. Implementing distributed server groups for the World Wide Web. Technical Report CMU-CS-95-114, School of Computer Science, Carnegie Mellon University, Jan. 1995

59 Apostolopoulos G, Aubespin D, Peris V, Pradhan P, Saha D. Design, implementation and performance of a content-based switch. In: Proc INFOCOM 2000, Vol. 3. Piscataway, NJ: IEEE Press, 2000: 1117~1126

60 Aron M, Sanders D, Druschel P, Zwaenepoel W. Scalable content-aware request distribution in cluster-based network servers. In: Proc ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Santa Clara, CA, June 2000

61 Pai VS, Aron M, Banga G, Svendsen M, et al. Locality-aware request distribution in cluster-based network servers. In: Proc 8th Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, CA, Oct. 1998

62 Cohen A, Rangarajan S, Slye H. On the performance of TCP splicing for URL-aware redirection. In: Proc 2nd USENIX Symposium on Internet Technologies and Systems. Boulder, CO, 1999

63 Cisco CSS 11000 series content services switches. http://www.cisco.com/warp/public/cc/pd/si/11000/

64 Shan Zhiguang, Lin Chuang, Marinescu D C, Yang Y. Modeling and performance analysis of QoS-aware load balancing of Web server clusters. Computer Networks, 2002, 40(2): 235~256

65 **单志广**. Web **服务质量**(QoS)**控制的策略、模型及性能分析**, [**博士学位论文**]. **北京**: **北京科技大学**, 2002 **年** 6 **月**
(Shan Zhiguang. Web QoS control: Policies, modeling and performance analysis [Ph. D Thesis], Beijing: University of Science and Technology Beijing, June 2002)

66 Shan Zhiguang, Lin Chuang, Yang Y, Wang Y. Performance Modeling and approximate analysis of multiserver multiqueue systems with Poisson and self-similar arrivals. Journal of University of Science and Technology Beijing, 2001, 8(2): 145~151

67 Shan Zhiguang, Lin Chuang, Yang Y. A multiserver multiqueue network: modeling and performance analysis. Journal of University of Science and Technology Beijing, 2002, 9(5): 389~395

68 Zhang J, Hamalainen T, Joutsensalo J, Kaario K. QoS-aware load balancing algorithm for globally distributed Web systems. In: Proc 2001 International Conferences on Info-Tech and Info-Net (Conference E: Networking). Beijing, China, Oct. 2001. 36~41

69 Lin C, Marinescu D C. Stochastic high-level Petri nets and applications. IEEE Trans on Computers, 1988, 37(7): 815~825

70 Arlitt M, Liu C. A workload characterization study of the 1998 World Cup Web Site. IEEE Network, 2000, 30~37

71 Stallings W. Operating Systems: Internals and Design Principles. 4th ed. Prentice-Hall International, Inc., 2001

72 Murata T. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, 1989, 77(4): 541~580

73 Ciaodo G, Muppala J, Trivedi K S. SPNP: Stochastic Petri net package. In: Proc Petri Nets and

Performance Models. 1989,142 ~ 151

74 Bunt R B, Eager D L, Oster G M, Williamson C L. Achieving load balancing and effective caching in clustered Web servers. In: Proc 4th International Web Caching Workshop. 1999

75 Aron M, Sanders D, Druschel P, Zwaenepoel W. Scalable content-aware request distribution in cluster-based network servers. In: Proc 2000 Annual Usenix Technical Conference. 2000

76 单志广,戴琼海,林闯,杨扬. Web 请求分配和选择的综合方案与性能分析. 软件学报,2001,12(3):355 ~ 366

77 单志广,林闯,肖人毅,杨扬. Web QoS 控制研究综述. 计算机学报,2004,27(2):145 ~ 156

# 第四部分

# QoS 的仿真与实现

QUALITY OF SERVICE OF COMPUTER NETWORKS

# 第14章

# 基于 NS2 的网络仿真

## 14.1 网络仿真工具 NS2 概述

NS2(Network Simulator,version 2)是一种面向对象的网络仿真器,由 UC Berkeley 开发而成。它本身有一个虚拟时钟,所有的仿真都是由离散事件驱动的。目前 NS2 可以用于仿真各种不同的 IP 网,已经实现了的一些仿真有:网络传输协议,比如 TCP 和 UDP;业务源流量产生器,比如 FTP, Telnet, Web, CBR 和 VBR;路由队列管理机制,比如 DropTail, RED 和 CBQ;路由算法,比如 Dijkstra 等。NS2 也为进行局域网的仿真而实现了多播以及一些 MAC 子层协议。NS2 使用 C ++ 和 Otcl 作为开发语言。

NS 的基本结构如图 14.1.1 所示。

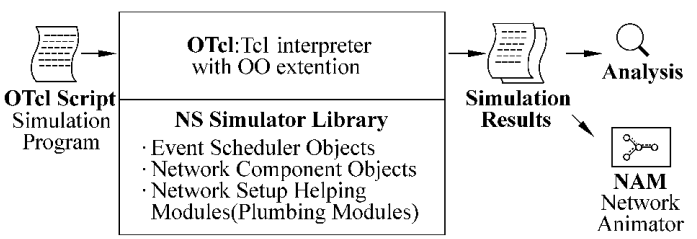



图 14.1.1 简化的 NS 仿真过程

NS 可以说就是 OTcl 的脚本解释器,它包含仿真事件调度器、网络组件对象库以及网络构建模型库等。

除了网络组件对象以外,NS 的另一个主要组件就是事件调度器。NS 中的一个事件就是一个具有惟一 ID 的分组,这个分组内包含调度时间以及指向处理该事件的对象的指针。事件调度器计算(keep track of)仿真时间,并且激活事件队列中的当前事件(主要是通过激活适当的网络组件),执行一些相关的事件(通常都是那些发出事件的对象),网络组件通过传递分组(packet)来互相通信,但是这并不耗费仿真时间。所有需要花费仿真时间来处理分组的网络组件(比如一个延迟部件)都必须要使用事件调度器。它先为这个分组发出一个事件,然后等待这个事件被调度回来之后,才能做下一步的处理工作。

比如,一个网络转换器,它具有 20 毫秒的转换延迟,为一个分组发出一个事件。

20 毫秒后,调度器就会出队这个事件,把它激活并传递给那个转换器。转换器才能执行它的操作,把这个分组传送给适当的输出链路组件。

事件调度器的另一个用处就是计时。比如,TCP 协议就需要一个计时器来跟踪分组的传送时间,看它是否超时 如果超时,它就会进行重传(传送一个具有相同 TCP 分组序列号和不同 NS 分组 ID 的分组)。计时器使用事件调度器的方式与延迟部件相同。惟一的不同就是,计时器测量与分组有关的时间值,并且在一定时间之后进行一些相关的处理,但延迟部件却不是这样。

NS 是用 OTcl 和 C ++ 编写的。由于效率的原因,NS 将数据通道与控制通道的实现相分离。为了减少分组和事件的处理时间(并非仿真时间),事件调度器和数据通道上的基本网络组件对象都是用 C ++ 写出并编译的,这些对象通过映射(linkage)对 OTcl 解释器可见。那么如何进行映射呢?首先为每一个 C ++ 对象创建一个相应的 OTcl 对象,再将可配置变量作为相应的 OTcl 对象中的成员变量,最后将控制函数作为相应的 OTcl 对象中的成员函数。这样,对于 C ++ 对象的控制就完全由 OTcl 负责了。如果对由 C ++ 映射过来的 OTcl 对象添加成员函数和变量也是可能的。当然,在仿真中不受控制和被内部使用的 C ++ 对象就不必映射到 OTcl 对象;一个对象(不在数据通道上)也可以完全由 OTcl 来实现。

从图 14. 1. 2 中可以看出,对于 C ++ 中具有映射的对象,如果它们形成了一定的层次结构,那么在 OTcl 对象中,也会有相似的层次结构。

NS 的一般体系结构可以如图 14. 1. 3 所示。

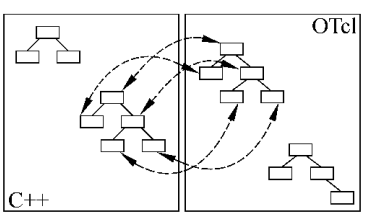



图 14. 1. 2 C ++ 与 OTcl 的对应

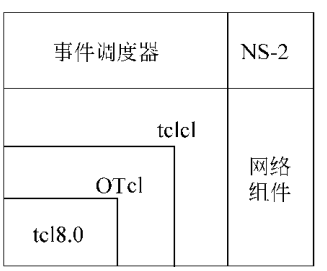



图 14. 1. 3 NS 的体系结构

一般用户被认为是站在图 14. 1. 3 的左下角。事件调度器和大多数网络组件都是用 C ++ 实现的,并且通过映射对 OTcl 可见 而映射本身是用 tclcl 实现的。所有的部分合起来就构成了 NS——一个具有网络模型库的 OTcl 解释器。

当某个仿真完成以后,如何获得仿真数据呢?如图 14. 1. 1 所示,仿真结束后,NS 将会产生一个或多个基于文本的跟踪文件。只要在 Tcl 脚本中加入一些简单的语句,这些文件中就会包含详细的跟踪信息。这些数据可以用于下一步的分析处理,也可以使用 NAM 将整个仿真过程展示出来。NAM 是一个很好用的软件,它具有类似于 CD 播放器的图像界面(播放、快进、倒带、暂停等)和一个播放速率控制器。此外,它还能够图形化地展示诸如链路吞吐率以及分组丢失数之类的信息,尽管这些信息并不能用于准确的仿真分析。

## 14.2　NS 仿真基础

### 14.2.1　用户编程语言 OTcl

在 14.1 节的概述里面已经提到过，NS 基本上就是一个具有网络仿真对象库的 OTcl 脚本解释器。所以，要想使用 NS 就必须学会如何使用 OTcl 语言编程。下面的两个例子将会给读者提供一些关于 OTcl 编程的基本概念。

```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
       if {$k < 5} {
           puts "k < 5, pow = [expr pow($d, $k)]"
       } else {
          puts "k >= 5, mod = [expr $d % $k]"
       }
    }
}

# Calling the "test" procedure created above
test
```

例 14.2.1　一个 Tcl 脚本的例子

例 14.2.1 是一个一般的 Tcl 脚本的例子。它虽然很简单，但还是向我们展示了一个程序语言的基本要素，比如如何创建一个过程并调用它，如何给一个变量赋值以及如何做一个循环，等等。我们已经知道，OTcl 就是 Tcl 语言引入面向对象的扩展，那么显然，所有的 Tcl 命令都可以在 OTcl 中使用，它们的关系就像 C 和 C ++ 一样。要运行这个脚本，只要将下面的代码存入一个文件，比如 ex-tcl.tcl，然后在 shell 提示符后面键入“ns ex-tcl.tcl”就可以了（假定已经安装了 NS）。如果也安装了 tcl8.0 的话，键入“tcl ex-tcl.tcl”将会得到同样的结果。

在 Tcl 语言中，关键字 proc 用来定义一个过程，后面跟着过程名和参数（花括弧中）；关键字 set 用于给一个变量赋值；[expr…] 用来让解释器计算方括弧中表达式的值；要获取一个变量的值，必须在变量名前加 $；关键字 puts 打印出双引号中的字符串。下面是例 14.2.1 的运行结果：

```
k < 5, pow = 1.0
k < 5, pow = 1120.0
k < 5, pow = 1254400.0
k < 5, pow = 1404928000.0
```

```
k < 5, pow = 1573519360000.0
k >= 5, mod = 0
k >= 5, mod = 4
k >= 5, mod = 0
k >= 5, mod = 0
k >= 5, mod = 4
```

图 14.2.2 是 OTcl 中面向对象编程的一个例子。这个例子也很简单,它向我们展示了在 OTcl 中创建并使用一个对象的方法。如果只是作为 NS 的一个普通用户,则自己动手编写对象的机会可能很少。但是,既然仿真过程中所有的 NS 对象,无论它们是用C++编写然后通过 Tcl 映射在 OTcl 中可见,还是直接使用 OTcl 语言编写的,都是最基本的 OTcl 对象,对它们有所了解将会不无裨益。

```
# Create a class call "mom" and
# add a member function call "greet"
Class mom
mom instproc greet {} {
    $self instvar age_
    puts "$age_ years old mom say:
    How are you doing?"
}

# Create a child class of "mom" called "kid"
# and overide the member function "greet"
Class kid -superclass mom
kid instproc greet {} {
    $self instvar age_
    puts "$age_ years old kid say:
    What's up, dude?"
}

# Create a mom and a kid object set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function "greet" of each object
$a greet
$b greet
```

例 14.2.2　一个 OTcl 脚本的例子

例 14.2.2 是一个 OTcl 脚本的例子,它定义了两个对象类,“mom”和“kid”,其中“kid”是“mom”的一个子类。每一个类都有一个叫作 greet 的成员函数。类定义之后,每个类都申明了一个实例,它们的变量 age 被分别置为 45 和 15,其成员函数 greet 也被分别

调用。关键字 class 用于创建一个对象类，instproc 用来定义一个成员函数，-superclass 用于描述类的继承。在定义成员函数的时候，$ self 与 C ++ 中的“ this ”指针作用相同。Instvar 检查变量是否已经在超类中被申明，如果是，则引用之；否则，申明一个新的变量。关键字 new 用于建立一个对象实例。将上面的代码存入一个新的文件 ex-otcl. tcl，并且运行“ ns ex-otcl. tcl ”，就会得到下面的结果：

```
45 year old mom say:
    How are you doing?
15 year old kid say:
    What s up, dude?
```

## 14.2.2　网络仿真

这一节我们将会展示一个简单的 NS 仿真脚本，并且说明每行代码都完成什么样的动作。例 14.2.3 中的 OTcl 脚本创建了如图 14.2.1 所示的网络结构，并运行其中的仿真配置。

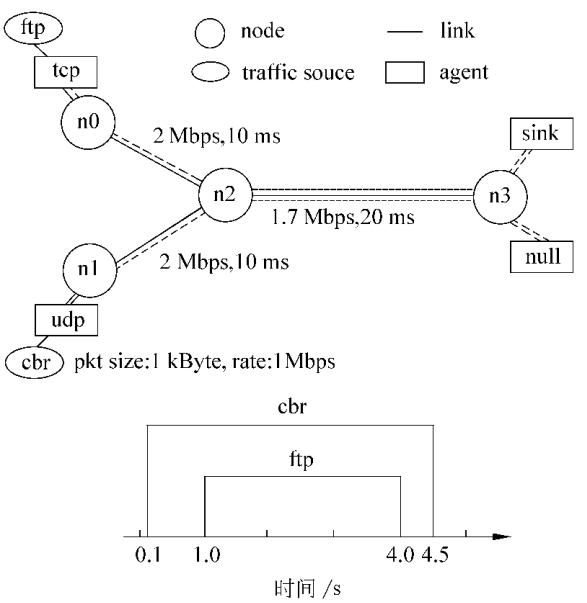



图 14.2.1　一个简单的网络拓扑和仿真配置

这个网络由 4 个节点组成，如图 14.2.1 所示。节点 n0 和 n2，n1 和 n2 之间的全双工链路的带宽为 2Mbps，延时为 10ms。节点 n2 和 n3 之间的全双工链路的带宽为 1.7Mbps，延时为 20ms。所有节点都使用 DropTail 队列，队列容量为 10。节点 n0 上有一个 tcp 代理，它与节点 n3 上的 tcp sink 代理之间建立了一个连接。Tcp 代理产生的最大的 tcp 分组默认为 1KB。Tcp sink 代理产生并发送 ACK 分组给 tcp 代理，它还将接收到的 tcp 分组释放掉。节点 n1 上的 udp 代理与节点 n3 上的 null 代理之间也建立了一个连接。Null 代理只是释放接收到的分组，也就是说，它并不产生并传送 ACK 分组。一个 ftp 和一个 cbr 流量发生器被分别连接到 tcp 代理和 udp 代理，cbr 业务源以 1Mbps 的速率产生大小为 1KB 的分组。Cbr 业务开始于 0.1s，在 4.5s 结束；ftp 业务在 1.0s 开始，结束于 4.0s。

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
       global ns nf
       $ns flush-trace
       #Close the NAM trace file
       close $nf
       #Execute NAM on the trace file
       exec nam out.nam &
       exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_2
$ns attach-agent $n0 $tcp
```

**例 14.2.3　一个简单的 NS 仿真脚本**

```
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1


#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

**例 14.2.3(续)**

下面,我们对上面的脚本作出解释。一般说来,NS 脚本一开始首先创建一个仿真器对象实例:

- set ns [new Simulator]:产生一个仿真器对象实例,并把它赋给变量 ns,这行命令主要做下面一些动作:
  - 初始化 packet 格式
  - 创建一个调度器
  - 选择默认的地址格式
- "Simulator"对象拥有以下功能的成员函数:
  - 创建组合对象,比如节点 node 和连接 link
  - 链接已建立的网络组件对象,比如 attach-agent
  - 设定网络组件参数
  - 在代理之间建立连接(tcp 和 sink)
  - 指定显示选项
  - 其他

大多数这些函数都用于仿真的建立和调度,也有一些是作为 NAM 显示用的。

- $ns color fid color:用于描述流 fid 中分组的颜色,纯粹为了 NAM 显示;
- $ns namtrace-all file-descriptor:让模拟器跟踪仿真,并按 NAM 的格式记录下来。后面的 $ns flush-trace 将会把跟踪信息写入文件。相似地,成员函数 trace-all 以通用格式记录跟踪信息;
- proc finish {}:在仿真结束后,用命令 $ns at 5.0 "finish"调用之。这个函数主要描述仿真的后续处理;
- set n0 [$ns node]:创建一个节点;NS 中的节点是一个组合对象,包括地址和端口分类器。用户可以先分别创建一个地址和一个端口对象,然后再将它们合并成为一个节点;
- $ns duplex-link node1 node2 bandwidth delay queue-type:创建一对指定带宽和延迟的单项链接,并连结两个节点。在 NS 中,节点的输出队列是作为 link 的一部分来实现的,所以,用户在创建 link 时需要指定队列类型;上面的仿真中用的是 DropTail 类型的队列,如果想使用 RED 队列,只要把 DropTail 替换为 RED 就可以了。与 node 一样,link 也是一个组合对象,用户可以创建它的子对象并将其与 node 连接;
- $ns queue-limit node1 node2 number:设定了 node1 和 node2 之间两个单项链接的队列容量;
- $ns duplex-link-op node1 node2 ...:用于 NAM 显示。

  既然基本的网络组件都已完成,则下一件事情就是要建立业务代理(traffic agent)(比如 TCP 和 UDP)、业务源(traffic source)(比如 FTP 和 CBR),并且把它们分别连接到 node 和 agent。
- set tcp [new Agent/TCP]:创建一个 TCP 代理。一般说来,用户可以这样创建任何代理和业务源。代理和业务源实际上是简单对象(不是组合对象),通常由 C++

实现并映射到 OTcl，所以，没有专门的成员函数来创建这些对象实例。若要创建一个代理或业务源，用户需要知道这些对象的类名（Agent/TCP，Agnet/TCPSink，Application/FTP 等）；

- $ns attach-agent node agent：成员函数 attach-agent 将一个对象 agent 连接到对象 node，实际上，这个函数所做的就是调用指定节点的 attach 函数，它把那个指定的代理连结到自己。所以，用户也可以这样做：$n0 attach $tcp。相似地，每一个代理对象都有一个成员函数 attach-agent，将业务源连结到自己；
- $ns connect agent1 agent2：在两个即将互相通信的代理被创建之后，下面一件事情就是要在它们之间建立一条逻辑连接，这条命令是通过将对方的网络和端口地址对设定为自己的目标地址来确立网络连接的。

假定所有的网络配置都已完成，下一件要做的事情就是写仿真配置（比如，仿真的时序安排）。仿真器对象有许多关于时序安排的成员函数，最常用的是：

- 在 $ns at time“string”：这个成员函数使得调度器（scheduler_是一个指针变量，它指向在一开始由[new Scheduler] 命令创建的调度器对象）在指定时间调度执行指定的“string”，此如，$ns at 0.1“$cbr start”就会使得调度器调用 CBR 业务源对象的 start 成员函数，它启动 CBR 来传输数据。在 NS 中，一个业务源通常都不传送实际的数据，但是会通报下层的代理它有多少数据要传，那些代理只要知道有多少数据要传，便会创建 packet 并送出它们。
- 在所有的网络配置、调度和后续处理的描述都完成之后，剩下的惟一一件事情就是运行这个仿真，只要简单的 $ns run 就可以完成。

## 14.2.3　事件调度器

下面我们将讨论 NS 的离散事件调度器。事件调度器的主要用户是仿真分组处理延迟或需要计时器的网络组件对象。图 14.2.2 展示了一个网络对象使用事件调度器的例子。注意，发布事件的网络对象就是后来在预定事件处理事件的那个网络对象，网络对象之间的数据链路不同于事件链路。实际上，分组从一个网络对象传递到另一个网络对象是通过两个方法来实现的：发送者的方法 send(Packet * p) {target_->recv(p)} 和接收者

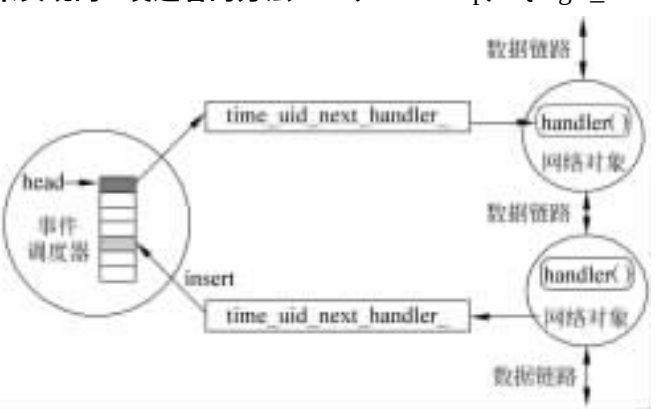



图 14.2.2　离散事件调度器

的方法 recv(Packet * , Handler * h = 0)。

NS 实现了两类不同的事件调度器,实时的和非实时的。对于非实时的事件调度器来说,NS 有三种实现(List, Heap 和 Calendar),它们在逻辑上是等价的. 实时的调度器用来仿真,这就允许仿真器与实际的网络交互。

下面是一个选择特定事件调度器的例子:

```
…
set ns [new Simulator]
$ns use-scheduler Heap
…
```

事件调度器的另一个用处是用来调度仿真事件,比如何时开始 FTP 应用,何时结束仿真,等等。一个事件调度器对象本身就有仿真调度成员函数,比如 at time“ string ”,它在指定的仿真时间 time 内发送一个名叫 AtEvent 的专用事件。一个 AtEvent 事件实际上是 Event 的子类,它有一个额外的变量来保存指定的 string。在事件调度器中,它与标准 Event 的处理相同。一个仿真开始后,当 AtEvent 事件的调度时间到来时,它就被传递给“ AtEvent handler ”;“ AtEvent handler ”被一次创建并处理所有的 AtEvent,同时执行 AtEvent 中由 string 指定的命令。

下面是前面例子的扩展:

```
…
set ns [new Simulator]
$ns use-scheduler Heap
$ns at 300.5 "complete_sim"
…
proc complete_sim {} {
…
}
```

从这个例子也许可以注意到,at time“ string ”是仿真器对象(set ns [new Simulator])的一个成员函数。但是要记住,仿真器对象只是一个用户接口,它实际上是调用的网络对象或调度器对象的成员函数,它们才是做实际工作的。下面是仿真器对象部分成员函数列表:

- Simulator instproc now # return scheduler s notion of current time
- Simulator instproc at args # schedule execution of code at specified time
- Simulator instproc at-now args # schedule execution of code at now
- Simulator instproc after n args # schedule execution of code after n secs
- Simulator instproc run args # start scheduler
- Simulator instproc halt # stop (pause) scheduler

## 14.2.4 网络组件

下面介绍 NS 中的网络组件,主要是复合网络组件。图 14.2.3 展示了 NS 中 OTcl 类

的层次结构(部分),这对我们理解基本的网络组件将会很有帮助。

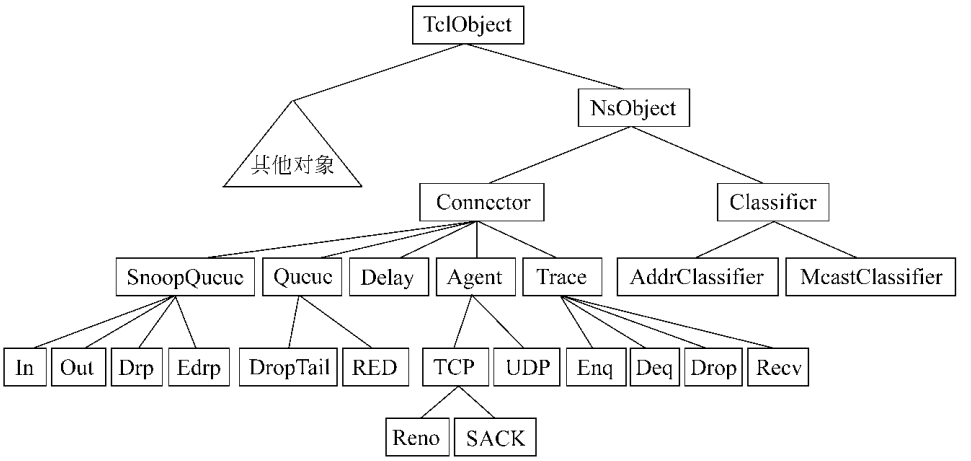



图 14. 2. 3　NS 中 OTcl 类的层次结构(部分)

层次的根是 TclObject 类,它是所有 OTcl 库对象(调度器,网络组件,计时器以及其他包括 NAM 相关的对象的超类。作为 TclObject 的子类,NsObject 类是所有进行分组处理的基本网络组件对象的超类。这些基本网络组件对象包括复合网络对象,比如 node 和 link 之类。

基本网络组件对象又被进一步划分为 Connector 和 Classifier 两个子类。这种划分是基于可能的输出数据路由的数目进行的。只有一个输出数据路由的基本网络对象被划分到 Connector 类下,像交换对象(switch)之类的有可能的多个输出数据路由的网络对象被划分到 Classifier 下。

1. 节点和路由

一个节点是一个组合对象,它包括节点入口(node entry)对象和 classifiers(如图 14. 2. 4所示)。NS 中有两种不同的节点类型:单播节点(unicast)和多播节点(multicast)。一个单播节点有一个进行单播路由的地址分选器(classifier)和一个端口分选器。多播节

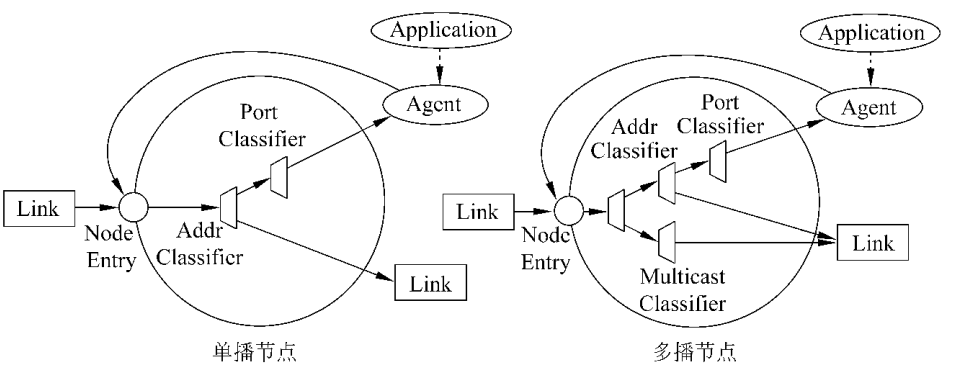



图 14. 2. 4　单播节点和多播节点

点另外还有一个分选出多播分组的分选器和一个进行多播路由的多播分选器。

在 NS 中,默认的节点是单播的。要建立多播节点,用户必须在创建了调度器之后在输入脚本中明确地描述这一点。这样,以后建立的节点都将是多播的节点了。在指定节点类型之后,用户还可以选择一个特定的路由协议来代替默认的协议。

- Unicast

- $ ns rtproto type
- type:Static, Session, DB, cost, multi-path ?

- Multicast

- $ ns multicast (right after set $ ns [new Scheduler])
- $ ns mrtproto type
- type:CtrMcast, DM, ST, BST

2. 链路

链路(link)是 NS 中另外一个组合的网络对象。当用户使用成员函数 duplex-link 创建一个链路时,两个方向的单向链路都被创建了(如图 14.2.5 所示)。要注意的是,一个节点的输出队列实际上是作为单向链路对象的一部分来实现的。

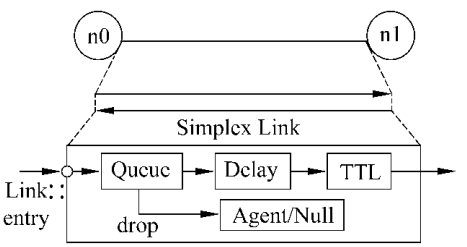



图 14.2.5 链路

另外,需要注意的是,一个节点的输出队列是作为单项链路的一部分来实现的。从一个队列中出队的分组被传递到仿真链路延迟的延迟(delay)对象。从队列中丢弃的分组被发送到 Null Agent,并在那里被释放。最后,TTL 对象为每一个收到的分组计算 Time To Live 参数,并更新每一个分组的 TTL 域的值。

(1) 跟踪技术(tracing)

在 NS 中,对网络行为的跟踪是沿着单向链路进行的。如果用户要求仿真器跟踪网络行为(用 $ ns trace-all file 或者 $ ns namtrace-all file 来描述),以后创建的 link 将会插入如图 14.2.6 所示的跟踪对象。用户也可以使用 create-trace {type file src dst} 命令专门在指定的 src 和 dst 之间创建类型 type 的跟踪对象。当插入的跟踪对象(EnqT, DeqT, DrpT 或 RecvT)收到一个分组时,它就会把这个情况写入指定的跟踪文件,并把分组传递给下一个网络组件。而这些都不耗费任何仿真时间。

(2) 队列监视器(queue monitor)

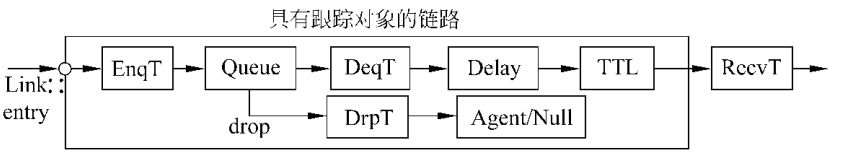



图 14.2.6　具有跟踪对象的链路

跟踪对象主要用来记录分组到达它们所在位置的时间。虽然一个用户可以从跟踪对象中获得很多信息，他也许会对某个特定输出队列内部所发生的事情感兴趣。比如，一个对 RED 队列行为感兴趣的用户也许想测量平均队列长度的动态值，或者一个特定 RED 队列的当前值。可以使用队列监视器（queue monitor）对象或者队列探听（snoop queue objects）对象来达到对于队列的监控（如图 14.2.7 所示）。当一个分组到达时，队列探听对象就会将此事通知队列监视器对象。队列监视器就是用这个信息监控队列的。

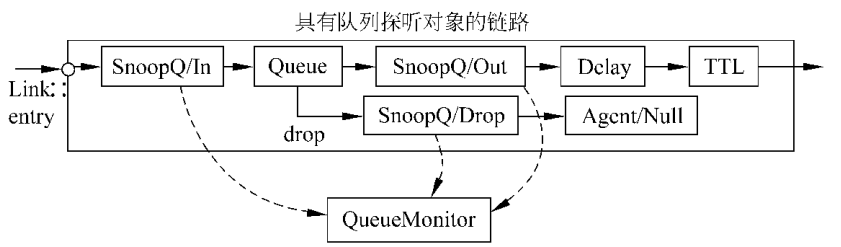



图 14.2.7　具有队列探听对象的链路

## 14.2.5　分组

一个 NS 的分组包含一个分组头堆栈和一个可选的数据域（如图 14.2.8 所示）。当一个仿真器对象被创建时，分组头格式就已经完成了初始化。这时，分组头堆栈中所有已注册的分组头，像公共（common）分组头、IP 分组头、TCP 分组头、RTP 分组头（UDP 使用）和 trace 分组头，都已经被定义，并且每个分组头在栈中的偏移都已被记录下来。如果需要，公共分组头可以被任何一个对象使用。

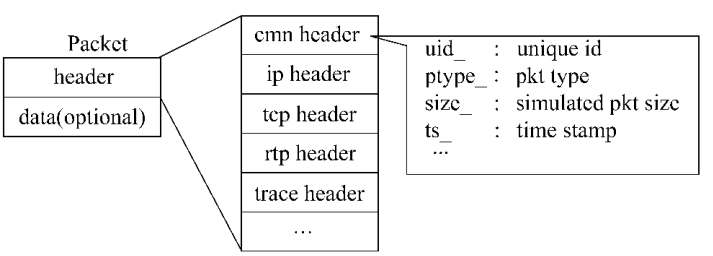



图 14.2.8　NS 分组结构

这就意味着，无论特定的分组头是否被使用，当一个代理分派一个分组时，一个包含所有注册分组头的堆栈就被创建了。一个网络对象可以使用偏移值访问分组头堆栈中的任何一个分组头。通常，一个分组只拥有分组头堆栈（数据域指针为空 null）。

## 14.3 仿真后续处理

### 14.3.1 跟踪分析

这一节将展示一个跟踪分析的例子。例 14.3.1 是由例 14.2.3 改编而来的。

```
...
#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish { } {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $tf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
...
```

例 14.3.1 激活了跟踪功能的简单仿真脚本

例 14.3.1 在例 14.2.3 的基础上添加了几行代码,打开跟踪文件并写入跟踪数据。运行上面的脚本将会产生一个将用作 NAM 输入的 NAM 跟踪文件和一个用于仿真分析的跟踪文件 out.tr 。图 14.3.1 展示了跟踪文件的格式和 out.tr 中的跟踪数据。

跟踪文件的每一行都开始于一个事件描述符(+,-,d,r);后面紧跟着事件的仿真时间(以秒计)、源节点和目标节点,用于识别事件发生的链路;一行中在标记(flag,如果没有设置就显示为"------")之前的下一个信息是分组的类型和大小(以字节计);当前,NS 只实现了 ECN(explicit congestion notification)位,其余位暂时还没有用;下一个域是 IPv6 的 fid(flow id),用户可以在 OTcl 脚本输入时设定它。即使 fid 域在仿真中没有被用到,用户也可以把它用于分析上。当描述 NAM 显示中的流颜色时,fid 也可以派上用场。下两个域是以"node.port"格式显示的源地址和目标地址。再下一个域指示网络层的分组序号。注意,即使 UDP 实现中不用分组序号,为了达到分析的目的,NS 仍旧使用 UDP 分组序号。最后一个域展示了数据分组的惟一的分组 ID。

如果已经有了仿真跟踪数据,那么所有要做的事情就是把所感兴趣的数据转换成好

| event | time | from node | to node | pkt type | pkt size | flags | fid | src addr | dst addr | seq num | pkt id |
|---|---|---|---|---|---|---|---|---|---|---|---|

```
r:receive (at to_node)
+:enqueue (at queue)      arc_addr:node.port(3.0)
-:dequeue (at queue)      arc_addr:node.port(0.0)
d:drop    (at queue)

 r 1.3556 3 2 ack 40-------1 3.0 0.0. 15 201
 + 1.3556 2 0 ack 40-------1 3.0 0.0. 15 201
 - 1.3556 3 2 ack 40-------1 3.0 0.0. 15 201
 r 1.35576 0 2 tcp 1000-------1 0.0 3.0. 29 199
 + 1.35576 2 3 tcp 1000-------1 0.0 3.0. 29 199
 d 1.35576 2 3 tcp 1000-------1 0.0 3.0. 29 199
 + 1.356 1 2 cbr 1000-------2 1.0 3.1. 157 207
 - 1.356 1 2 cbr 1000-------2 1.0 3.1. 157 207
```

**图 14.3.1　跟踪文件数据格式**

理解的信息并分析之。下面是一个数据转换的例子。这个例子使用一个叫“ column ”的 perl 命令，以选择给定输入数据中的一列。这是一个使用管道的 shell 命令，它使用“ out.tr ”中的数据计算在接收节点（n3）的 CBR 流量抖动，并把结果数据储存到“ jitter.txt ”中去。

```
cat out.tr | grep " 2 3 cbr " | grep r | column 1 10 | awk {dif = $2 - old2 ;if(dif = =0) dif = 1 ;if(dif > 0) {printf("%dht%fhn", $2, ( $1 - old1) / dif) ;old1 = $1 ; old2 = $2}} > jitter.txt
```

这个 shell 命令挑选在 n3 的“ CBR packet receive ”事件，选择时间（column 1）和分组序（column 10），并对每一个包序号计算在分组间隔和包序间隔（对于丢失的包而言）的比值。

下面图 14.3.2 就是使用 gnuplot 产生的相应的抖动图。其中，X 轴指示分组序号，Y 轴指示用秒表示仿真时间。

## 14.3.2　队列监测

本节主要想通过一个例子（例 14.3.2）来展示如何进行队列的监测。图 14.3.3 展示了一个 RED 队列监测例子的网络拓扑配置。注意到，我们要监测的队列就是在节点 r1 ~ r2之间的链路，它能够容纳 25 个分组。我们的目标是想通过队列长度当前值和平均值的动态情况来了解 RED 队列是怎样工作的。

在上面的脚本中，有两个事情要注意。首先，更高级的仿真器对象成员函数 create-connection 被用来建立 TCP 连接。其次，在脚本中的队列跟踪部分，用一个变量指向 RED 队列对象，调用其成员函数 trace 来监测当前队列长度（curq_）和平均队列长度（avg_），并让它们把结果写入文件“ all.q ”中。队列跟踪输出格式为

```
a time avg_q_size
```

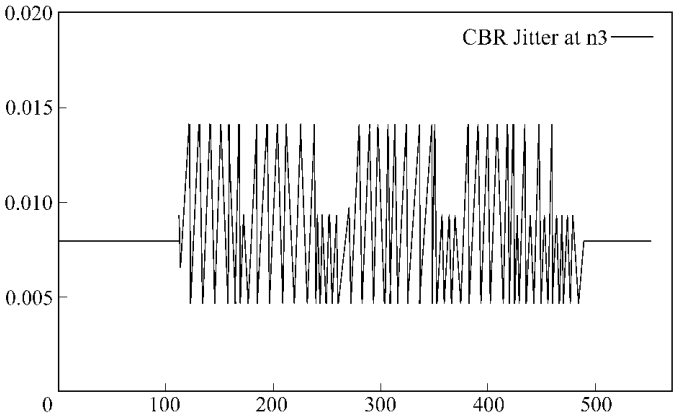



**图 14.3.2　在接收节点 n3 的 CBR 抖动图**

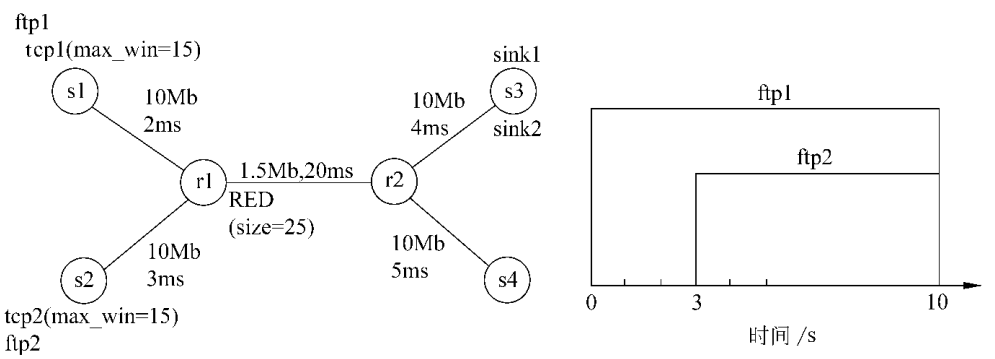



**图 14.3.3　RED 队列监测的例子**

```
$ns duplex-link $ node_(r1) $ node_(r2) 1.5Mb 20ms RED
set tcp1 [$ns create-connection TCP/Reno $node_(s1) TCPSink $node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Reno $node_(s2) TCPSink $node_(s3) 1]
$tcp2 set window_ 15
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]

# Tracing a queue
set redq [[$ns link $node_(r1) $node_(r2)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_

$ns at 0.0 "$ftp1 start"
$ns at 3.0 "$ftp2 start"
$ns at 10 "finish"
```

**例 14.3.2　RED 队列监测仿真脚本**

```
# Define 'finish' procedure (include post-simulation processes)
proc finish {} {
    global tchan_
    set awkCode {
        {
            if ($1 == "Q" && NF>2) {
            print $2, $3 >> "temp.q";
                set end $2
            }
        else if ($1 == "a" && NF>2)
            print $2, $3 >> "temp.a";
    }
    }
    set f [open temp.queue w]
    puts $f "TitleText: red"
    puts $f "Device: Postscript"

    if { [info exists tchan_] } {
     close $tchan_
    }
    exec rm -f temp.q temp.a
    exec touch temp.a temp.q

    exec awk $awkCode all.q

    puts $f \"queue
    exec cat temp.q >@ $f
    puts $f \n\"ave_queue
    exec cat temp.a >@ $f
    close $f
    exec xgraph -bb -tk -x time -y queue temp.queue &
    exit 0
}
$ns run
```

**例** 14. 3. 2**(续)**

Q time crnt_q_size

**既然所有的仿真工作已经完成,那么下一件事情就是进行后续处理。在这个脚本中使用的是** awk **来作后续处理,它把监测信息转换成** Xgraph **输入格式,并且启动它去绘出指定的信息(如图** 14. 3. 4 **所示)。**

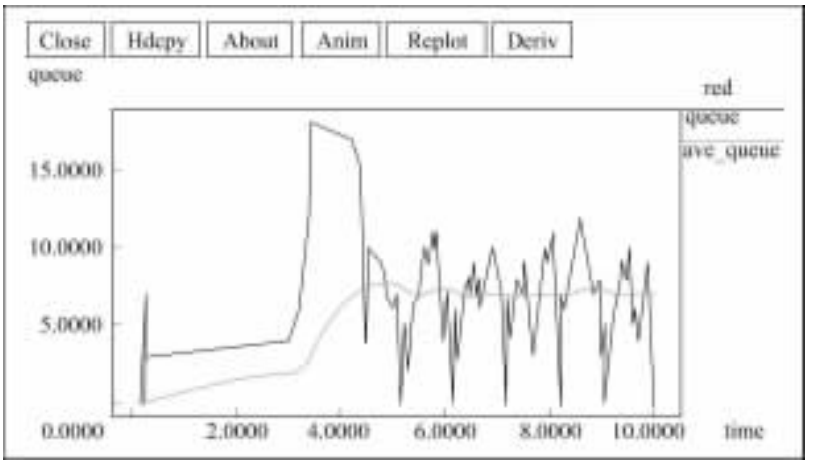



图 14.3.4 RED 队列跟踪图

## 14.4 NS 的扩展

### 14.4.1 NS 软件的相关内容

在讨论如何扩展 NS 之前,我们先来简要地看一看在各个文件或目录里都有一些什么样的信息。图 14.4.1 展示了使用 ns-allinone-2.1b 软件包安装的仿真器目录结构的一部分。

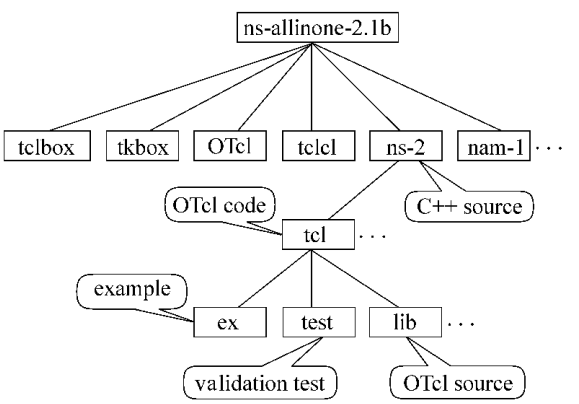



图 14.4.1 NS 文件目录结构

在 ns-allinone-2.1b 的子文件夹中,ns-2 包含所有仿真器的 C++ 或 OTcl 实现、OTcl 测试脚本和例子脚本。在这个文件夹里,所有的 OTcl 代码和测试、例子脚本都放在子文件夹 tcl 中,大多数实现事件调度器和基本网络组件对象类的 C++ 代码,除了与 www 相关的以外,都放在当前层里。比如,如果想了解 UDP 代理的实现,就应当去‘ns-allinone-2.1b/ns-2”文件夹,打开‘udp.h”,“udp.cc”以及包含 UDP 祖先类实现的文件。要了解

网络组件的层次结构，去网络组件一节的图 14.2.3。从现在开始，你就被认为是已经在文件夹 ns-allinone-2.1b 之下了。

文件夹 tcl 还含有子文件夹，其中一个就是 lib。它包含大多数关于 NS 实现（agent，node，link，packet，address，routing，等等）的基本部分的 OTcl 源代码，用户和开发者都需要经常去访问。注意到，关于 LAN，Web 和 Multicast 实现的 OTcl 源代码放在单独的子文件夹 tcl 下。下面就是“ns-2/tcl/lib”目录中的部分文件列表：

- ns-lib.tcl：除了与 LAN，Web 和 Multicast 相关的以外，仿真器类和大多数成员函数的定义都放在这儿。如果想知道哪一个成员函数可用以及它们是如何工作的，就来这里。
- ns-default.tcl：各种不同网络组件可配置参数的缺省值都放在这里。既然大多数网络组件是用 C++ 实现的，可配置参数实际上就是 C++ 变量。它们通过 OTcl 映射函数 bind（C++ _variable_name，OTcl_variable_name）对 OTcl 可见。下一节将介绍如何从 C++ 代码作 OTcl 映射。
- ns-packet.tcl：分组头格式的初始化实现放在这里。当你创建一个新的分组头时，应当在这个文件中登记这个分组头，以便使分组头初始化进程能将分组头包括到分组头栈中并给出分组头在栈中的偏移。
- other OTcl files：这个文件夹中的其他 OTcl 文件包含了复合网络对象的实现和用 C++ 写出的网络对象前端（控制部分）的实现。FTP 应用的 OTcl 完全实现和源代码都放在文件 ns-source.tcl 中。

如果用户想知道怎样设计一个具体的仿真，也许他会对 tcl 的两个子文件夹 ex 和 test 感兴趣。前一个文件夹包含各种例子仿真脚本，后一个文件夹包含的是确认脚本，它通过运行各种不同的脚本并与预期的结果作比较来确认 NS 是否已经被成功地安装到机器上了。

## 14.4.2　Tcl 映射

通过添加新的基本网络对象可以扩展 NS，但是既然由于效率的原因，数据通道上的对象类都是用 C++ 实现的，那么通常就需要在 C++ 代码中作 OTcl 映射。这一节通过一个例子向大家介绍一下从 C++ 到 OTcl 的映射（linkage）。这个例子只是创建了一个简单的名叫“MyAgent”的代理，它只是一个没有实际动作行为的空白代理（比如，没有分组的创建和传输）。图 14.4.2 ~ 图 14.4.5 展示了“MyAgent”的部分源代码，它们一起构成了一个完整的实现（外加 3 个头文件行）。

1. 输出 C++ 类到 OTcl

假定已经用 C++ 创建了一个新的网络对象类“MyAgent”，它是“Agent”的子类，并且希望能够在 OTcl 中创建这个对象的实例。要做到这一点，我们必须定义一个映射对象，叫作“MyAgentClass”，它继承于“TclClass”。这个映射对象创建指定名称（在这个例子中“Agent/MyAgentOTcl”）的 OTcl 对象，并且在 OTcl 对象和 C++ 对象（“MyAgent”）之间建立映射。其中实例启动程序由成员函数“create”描述。图 14.4.2 展示了“MyAgent”类的

定义和映射类的定义。

最初,当 NS 被启动时,它运行静态类 class_my_agent 的构造函数,从而创建了“MyAgentClass”的一个实例。在这个过程中,“Agent/MyAgentOTcl”类以及它的某些方法都在 OTcl 中被创建了。如果一个 OTcl 用户试图使用命令‘new Agent/MyAgentOTcl’创建这个对象的一个实例,它就调用‘MyAgentClass::create”函数来创建“MyAgent”的一个实例并返回地址。注意到,从 OTcl 中创建 C++ 对象类的实例并不意味着能够从 OTcl 中调用 C++ 对象实例的成员函数和访问 C++ 对象实例的成员变量。

```
class MyAgent: public Agent {
public:
          MyAgent();
protected:
          int command(int argc,const char*const* argv);
private:
          int    My_var1;
          double My_var2;
          void   MyPrivFunc(void);
};
```

```
static class MyAgentClass:public TclClass{
public:
          MyAgentClass (): TclClass("Agent/MyAgentotcl"){}
          Tclobject* Create (int,const char*const*){
                  return(new MyAgent ());
          }
} Class_my_agent;
```

图 14.4.2 “MyAgent”类定义和映射类的定义

2. 输出 C++ 类的成员变量到 OTcl

假定新的 C++ 对象‘MyAgent”有两个参变量‘my_var1’和‘my_var2”,若很想在 OTcl 输入脚本中自由地配置它们,应当对每一个 C++ 成员变量使用 binding 函数。一个 binding 函数在相应的 OTcl 对象类(“Agent/MyAgentOTcl”)中创建一个指定名称的成员变量(第一个参数)与一个在 OTcl 类变量和 C++ 变量(它的地址由第二个变量指定)之间的双向绑定。图 14.4.3 展示出如何对图 14.4.2 中的两个变量‘my_var1’和‘my_var2”进行绑定的。

注意到,binding 函数放在‘MyAgent’构造函数中,当创建这个对象的一个实例时,这个绑定就被建立起来了。NS 支持对 5 种不同的变量类型的 4 种不同的绑定:

- bind():real or integer variables
- bind_time():time variable
- bind_bw():bandwidth variable

```
MyAgent::MyAgent():Agent (PT_UDP) {
        bind ("My_var1_otcl",&my_var1};
        bind("my_var2_otcl",&my_var2};
}
```

图 14.4.3　变量绑定的例子

- bind_bool()：boolean variable

通过这种方法，使用 OTcl 脚本定义并运行一个仿真器的用户，可以改变并访问由C ++ 实现的网络组件的可配置参数。注意，无论何时，当输出一个 C ++ 变量时，推荐的做法是同时在文件“ ns-2 /tcl /lib /ns-lib. tcl ”设定这个变量的默认值。否则，当创建新类的一个对象时，将会出现一个警告信息。

3. 输出 C ++ 对象控制命令到 OTcl

用户不仅可以输出 C ++ 成员变量，而且也可以把对 C ++ 对象的控制放到 OTcl 中。通过在用户的 C ++ 对象中定义一个作为命令解释器的成员函数可以轻易地完成这种控制转移。事实上，对一个用户来说，在 C ++ 对象的“ command ”成员函数中定义的 OTcl 命令与相应的 OTcl 对象中的成员函数是一样的。图 14.4.4 展示了图 14.4.2 中“ MyAgent ”对象的一个“ command ”成员函数定义的例子。

```
int MyAgent::Command(int argc,const char*const*argv){
      if(argc==2{
           if(strcmp(argv[1],"call-my-priv-unc")==0){
                 MyPrivFunc();
                 return(TCL_OK);
           }
      }
      return(Agent::command(argc,argv));
}
```

图 14.4.4　OTcl 命令解释器的例子

当在 OTcl 空间中创建匹配对象“ MyAgent ”的 OTcl 影子对象的一个实例（即 set myagent[new Agent/MyAgentOTcl]），并且用户试图调用这个对象的成员函数（即 $ myagent call-my-priv-func）时，OTcl 就会在 OTcl 对象中寻找给定的成员函数名称。如果没有发现，它就调用“ MyAgent ::command ”，并传递被调用的 OTcl 成员函数名和用 argc/argv 格式表示的参数。如果在成员函数“ command ”中被调用的成员函数名有一个动作，它将执行这个动作并返回结果。如果没有，它的祖先对象的“ command ”函数被递归调用直到发现指定名字为止。如果这个名字没有在任意一个祖先对象中被发现，一个错误信息将会被返回到 OTcl 对象，OTcl 对象将这个信息传递给用户。这样，OTcl 用户就可以控制 C ++ 对象的行为了。

4. 在 C ++ 对象中执行 OTcl 命令

当用户用 C ++ 实现一个新的网络对象时,也许想在 OTcl 中执行来自于 C ++ 的 OTcl 命令。图 14. 4. 5 展示了图 14. 4. 2 中‘ MyAgent ’的成员函数‘ MyPrivFunc ’的实现,它使得 OTcl 解释器执行打印私有成员变量‘ my_var1 ’和‘ my_var2 ”的值的操作。

```
void MyAgent::MyPrivFunc(void){
      Tcl& tcl=Tcl::instance ();
      tcl.eval("puts\"Message From MyPrivFunc\"");
      tcl.evalf("puts\"     my_var1=%d\",my_var1);
       tcl.evalf("puts\"    my_var2=%f\",my_var2);
}
```

图 14. 4. 5 在 C ++ 对象中执行 OTcl 命令

要在 C ++ 对象中执行 OTcl 命令,应当参考一下‘ Tcl ::instance() ”,它被声明为一个静态的成员变量。它提供了几个函数,通过这些函数,可以把一个 Otcl 命令传递给解释器(函数‘ MyPrivFunc ’的第一行就执行这个命令)。这个例子展示了两种传递 Otcl 命令给解释器的方式。

5. 编译、运行和测试‘ MyAgent ”

(1) 下载文件‘ ex-linkage. ce ”,保存到文件夹‘ ns-2 ”下;

ex-linkage.tcl

```
# Create MyAgent (Theis will give two warning messages that
# no default vaules exist for my_var1_otc1 and my_var2_otc1)
set myagent [new Agent/MyAgentotc1]


# Set configuable parameters of MyAgent
$myagent set my_var1_otc1 2
$myagent set my_var2_otc1 3.14

# Give a command to MyAgent
$myagent call-my-priv-func
```

result

```
warning: no class variable Agent/MyAgentOtc1::my_var1_otc1
         see tc1-object.tcl in tclcl for info about this warning.
warning: no class variable Agent/MyAgentOtc1::my_var2_otc1

Message From MyPrivFunc
     my_var1=2
     my_var2=3.140000
```

图 14. 4. 6 测试脚本和结果

（2）打开“ Makefile ”,把“ ex-linkage. o ”添加到对象文件列表的末尾；

（3）用“ make ”命令重新编译 NS；

（4）下载包含“ MyAgent ”测试命令的文件“ ex-linkage. td ”（如图 14. 4. 6 所示）；

（5）使用命令“ ns ex-linkage. tcl ”运行 OTcl 脚本。

## 14. 4. 3　添加新的应用和代理

（work with ns-2. 1b8a）

1. 目标

我们想要建立一个基于 UDP 连接的多媒体应用,它仿真一个虚构的五速率媒体（five level media scaling）应用的动作行为。这个应用通过改变与 scale 参数值相关的编码和传输策略对,可以在一定程度上对网络拥塞做出反应。

2. 应用描述

在这个实现中,默认的是一旦连接建立,发送方和接收方都具有 5 套不同的编码和传输策略对以及相关的 5 个 scale 值。出于简化的原因,同样假定对每一个编码和传输策略对都只有一个常量传输速率。并且不论编码方式如何,每一对使用的分组大小都相同。基本上,“ 五速媒体 ”是按照如下方式工作的：发送者一开始使用 scale 0 的速率发送数据分组,然后不断地按照接收者通报的 scale 值调整传输速率值。接收者有责任监控网络拥塞并决定 scale 因子。至于拥塞监控,要用到一个简单周期的（对每一个 TTR 间隔）分组丢失率。在一个周期中,即使只有一个分组丢失,也认为网络具有拥塞。一旦拥塞被检测到,接收者就减少一半 scale 参数值,并通报发送者。如果没有检测到分组丢失,接收者将 scale 值增加 1 点并通知发送者。

3. 问题分析

既然 UDP 代理分派并发送网络分组,应用层通信的所有信息就都应当作为数据流传递给 UDP 代理。然而,UDP 的实现分派只有头栈的分组。所以,我们需要修改 UDP 实现,添加机制来发送来自于应用的数据。同样注意到,我们也许会使用这个应用做一些关于 IP 路由器队列管理机制的深入研究。所以,我们需要一种方法将这种多媒体数据流类型与其他流类型相区别。即,需要修改 UDP 代理,在 IP 分组头某个域中记录数据类型。

4. 设计与实现

对于应用,我们决定修改 CBR 实现,使之具有五速媒体的特征。我们决定将这个应用命名为“ MmApp ”,并作为“ Application ”的一个子类来实现。相应的 OTcl 层次名叫作“ Application / MmApp ”。发送者和接收者被一起放在“ MmApp ”中实现。对于支持“ MmApp ”的代理,我们决定称其为“ UdpMmAgent ”,并把它作为“ UdpAgent ”的子类来实现。相应的 OTcl 层次名为“ Agent / UDP / UDPmm ”。

- MmApp Header：出于应用层通信，我们决定定义一个分组头(header)，它的结构叫作“hdr_mm”。只要应用层有信息要传，它就会把信息用结构“hdr_mm”的格式传给“UdpMmAgent”。然后，“UdpMmAgent”分派一个或更多的分组，并把数据写到每一个多媒体分组的分组头。图 14.4.7 展示了分组头的定义，其中包括一个分组头结构和一个分组头类“MultimediaHeaderClass”，它应当继承于类“PacketHeaderClass”。在定义这个类时，分组头的 OTcl 名字(“PacketHeader/Multimedia”)和分组头结构的大小都被列了出来。注意，在这个类的构造函数中必须调用函数 bind_offset()。

```
//Multimedia Header Structure
stract hdx_nm{
      int ack;           //is it ack packet?
      int seq;           //nm sequence number
      int nbytes;        //kytes for mm pkt
      doukle time;       //current time
      int scale;         //scale (u-4) associated with cata races

      //Facket header access functions
      static int offset_;
      inline staeic int & offset() (return offset_;}
      inline staeic hdr_mm* access(const packet* p){
                  recult (hdr_mm*1 p->access(offset_);
      }
};
```

```
// Multimedia Header Class
staeic class MultimediaHeaderClass: public PacketHeaderClass{
public:
 MultimediaHeaderClass():packetHeaderClass("PacketHeader/Multimedia",
                                          siteof(hdc_mm)){
          bind_offset(shdr_mm::offset_);
       }
}class_mmhuk;
```

图 14.4.7　MM 分组头结构和类

(in“udp-mm.h”&“udp-mm.cc”)

我们还需要在 packets.h 和 ns-packet.tcl 中添加几行代码来将我们的“Multimedia”分组头添加到分组头栈中，如图 14.4.8(a)和(b)所示。这时，分组头创建的过程就已完成。“UdpMmAgent”将会通过 hdr_mm::access() 成员函数访问新的分组头。

- MmApp Sender：发送者使用一个计时器来调度下一个应用数据分组的传送。我们定义了一个“TimerHandler”的子类“SendTimer”，并且让它的“expire()”成员函数来调用“MmApp”对象的成员函数“send_mm_pkt()”。然后，我们将这个对象的一个实例作为私有成员包含到对象“MmApp”中，以“snd_timer_”来引用。图 14.4.9 展示了 SendTimer 的实现。

在设定这个计时器之前，“MmApp”重新计算下一个传输时间。计算时要用到传输

```
enum packet_t {
      PT-tcp,
      ...
      PT Multimedia,
      PT_NITYPE//This MUST be the LAST one
};

class p_info {
public:
      p_info() {
            name_[PT_TCP]="tcp";
            ...
            name_[PT_Multimedia]="Multimedia";
            name_[PT_NTYPE]=}undefined";
      }
      ...
};
```

（a）　添加到文件“ packet. h ”中（C ++）

```
foreach prot{
      AODV
      ...
      Multimedia
}{
      add-packet-header $prot
}
```

（b）　添加到文件“ ns-packet. tcl ”中（OTcl）

图 14. 4. 8

速率连同与之相关的当前 scale 值和应用数据分组的大小，它已经在输入仿真脚本中给定了（或者使用默认的大小）。那个“ MmApp ”发送者，当来自于接收方 ACK 应用分组到达时，将会刷新 scale 参数。

- MmApp Receiver：接收者也使用计时器，叫做“ ack_timer_ ”，来调度下一个应用 ACK 分组的传输，它的传输间隔等于平均的 RRT。当接收到来自发送方的应用数据分组时，接收方就计数收到的数据分组，并且使用分组序号来计算丢失的分组数。当“ ack_timer_ ”到期时，它就调用“ MmApp ”的成员函数“ send_ack_pkt ”。它根据收到和丢失的分组的计数信息来调整 scale 值，清空收到和丢失的分组计数为零，并且发送具有新的 scale 值的 ACK 分组。
- UdpMmAgent：“ UdpMmAgent ”是通过修改“ UdpAgent ”而得到的，它具有以下新的特征：①把收到的来自“ MmApp ”的信息写入即将发送数据分组的 MM 分组头（或者从收到的数据分组的 MM 分组头中读出信息，并把它传递给“ MmApp ”；②分段并重新装配（“ UdpAgent ”只实现分段）；③为“ MmApp ”分组设定优先权比特位（IPv6），最大优先级为 15。
- Modify“ agent. h ”：为了产生一个新的代理和应用，应当添加两个新的方法到“ Agent ”类作为公共成员。在“ MmApp ”类的成员函数“ command ”中，定义了一个 OTcl 命令“ attach-agent ”。当从 OTcl 接收到这个命令时，“ MmApp ”尝试着把它自

已链接到下层的代理。在链接之前,它调用下层代理的“ supportMM() ”成员函数来检查底层代理是否支持多媒体传输业务(比如,看它是否支持应用层到应用层的数据传送)。如果支持,就调用“ enableMM() ”。即使这两个成员函数在类“ UdpMmAgnet ”中都已定义了,但是它们并没有在基类“ Agent ”中定义,并且普通“ Agent ”类中的两个成员函数将会被调用。所以,当用户试图编译这些代码时,它将会给出错误信息。将这两种方法作为公共成员函数插入“ Agent ”类将会解决这一问题,如图 14. 4. 10(a)所示。

```
class SendTimer : public TimerHandler {
 public:
        SendTimer(MmApp* c): TimerHandler(),t_(t)(}
        inline virtual void expire (Event* );
 protected:
        MaApp* t_;
};

void SendTimer::expire (Event* )
{
  t_- >send_mm_pkt();
}
```

```
class MnApp:public Application{
public:
       MnApp();
       ...
private:
       ...
       SendTimer snd_timer_;
       ...
};
```

```
MnApp ::MmApp():running_(0),and_timer_(this),ack_
timer_(this)
{
 bind_bw ("rate0_",&rate[0]};
 ...
 bind("rate4_",&rate[4]);
 bind("pktsize_",&pktsize_);
 bind_bool("random_",&random_);
}
```

```
void MmApp::send_mm_pkt()
(
 hdr_mm mh_buf;

 if (running_){
   ...
   agent_ - >sendmsg(pktsize_,(char* ) * mh_buf);//
 send to UDP
   // Reschedule the send_pkt timer
   do uble next_time_=next_sand_time();
   if(next_time_>0) snd_timer_.resched(next_time_);
 }
}
```

图 14. 4. 9 SendTimer 的实现

```
class Agent:public Connector {
 public:
        Agent (int pktType);
        ...
        virtual int supportMM() { return 0;}
        virtual void enableMM() {}
        virtual void sendmsg (int nbytes, const char *
        flages =0);
        virtual  void  send  ( int  nbytes )  {  sendmsg
        (nbytes); }
        ...
}
```

（a）添加两个成员函数到类“ Agent ”中

```
class Application : public Process {
public:
        Application ();
        virtual void send (int nbytes);
        virtual void recv {int nbytes);
        virtual void recv_msg(int nbytes,const char *
        msg =0){};
        virtual void resume();
        ...
};
```

（b）添加成员函数到类“ Application ”

图 14. 4. 10

- Modify“ app. h ”：用户也需要添加新的成员函数“ recv_msg（int nbytes，const char ＊msg）”到类“ Application ”中。如图 14. 4. 10（b）所示。
- 在“ ns-default. tcl ”中为新的参数设定默认值：在实现了应用和代理的所有零件之后，最后一件事情就是在文件“ ns-default. tcl ”中为新引入的可配置参数设定默认值。图 14. 4. 11 展示了一个设定默认值的例子。

```
...
Application/MmApp set rate0 _
0.3mb
Application/MmApp set rate0 _
0.6mb
Application/MmApp set rate0 _
0.7mb
Application/MmApp set rate0 _
1.2mb
Application/MmApp set rate0 _
1.5mb

Application/MmApp set pktsize
_1000
Application/MmApp  set  random
_false
...
```

图 14. 4. 11　缺省参数值的设定

5. 下载和编译

下面是在重新编译 NS 之前的动作序列：

(1) 下载“ mm-app. h ”,“ mm-app. cc ”;“ udp-mm. h ”和“ udp-mm. cc ”到“ ns-2 ”文件夹；

(2) 确认通过修改“ packet. h ”和“ ns-packet. tcl ”而注册了新的应用层分组头,如图 14. 4. 8(a)和(b)所示；

(3) 修改“ agent. h ”,将新的方法 supportMM() 和 enableMM() 添加到类“ Agent ”中,如图 14. 4. 10 (a)所示；

(4) 修改“ app. h ”,将新的方法 recv_msg() 添加到类“ Application ”中,如图 14. 4. 10 (b)所示；

(5) 在文件“ ns-default. tcl ”中为新引入的可配置变量设定默认值,如图 14. 4. 11 所示。

在做完这一切之后,根据需要修改文件“ Makefile ”(将文件“ mm-app. o ”和“ udp-mm. o ”包含入对象文件列表)并重新编译 NS 。一般来说,在对“ Makefile ”加以改变之后,重新编译之前,做一个“ make depend ”是一个好习惯。

6. 仿真测试

图 14. 4. 12 展示了仿真的网络拓扑及其配置来测试“ MmApp ”。图 14. 4. 13 展示了测试仿真脚本。

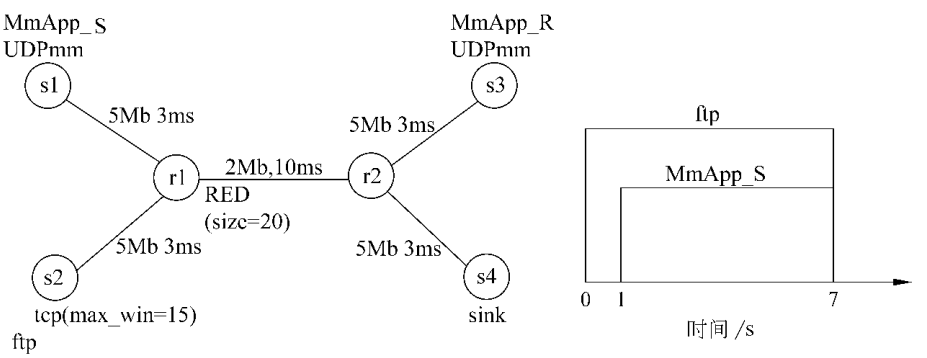



图 14. 4. 12 仿真的拓扑及其配置

## 14. 4. 4 添加新的队列

1. 目标

若要建立一个简单的 drop-tail 路由器的输出队列,这个队列对于优先级为 15 的分组(来自于“ UDPmm ”之上的“ MmApp ”)和队列中的其他分组使用 round-robin 的出队机制。即,当优先级为 15 的分组和其他分组共存于一个队列中时,它就一个接一个地轮流出队每一种类型队列中最早的分组。

```
set ns [new Simulator]

$ns duplex-link $node_(r1) $node_(r2) 2Mb 10ms RE

#Setup RED queue parameter
$ns queue-limit $node_(r1) $node_(r2) 20
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set q_weight_ 0.002
Queue/RED set ave_ 0

#Setup a MM UDP connection
set udp_s [new Agent/UDP/UDPmm]
set udp_r [new Agent/UDP/UDPmm]
$ns attach-agent $node_(s1) $udp_s
$ns attach-agent $node_(s3) $udp_r
$ns connect $udp_s $udp_r
$udp_s set packetSize_ 1000
```

**图** 14. 4. 13 **“** MmApp **”测试模型脚本**

```
$udp_r set packetSize_ 1000
$udp_s set fid_ 1
$udp_r set fid_ 1

#Setup a MM Application
set mmapp_s [new Application/MmApp]
set mmapp_r [new Application/MmApp]
$mmapp_s attach-agent $udp_s
$mmapp_r attach-agent $udp_r
$mmapp_s set pktsize_ 1000
$mmapp_s set random_ false


#Simulation Scenario
$ns at 0.0 "$ftp start"
$ns at 1.0 "$mmapp_s start"
$ns at 7.0 "finish"

$ns run
```

**图** 14. 4. 13**（续）**

2. **设计**

**这个队列有两个逻辑** FIFO **队列，分别叫作** LQ1 **和** LQ2**，它们总的队列长度等于物理队列长度（**PQ**），也即** LQ1 + LQ2 = PQ**。为了实现一个标准的** drop-tail **入队行为，当一个分组即将入队时，入队管理器检测** LQ1 + LQ2 **是否小于** PQ **的最大允许值。如果小**

于,那么这个分组将被入队到适当的逻辑队列。为了实现 round-robin 出队机制,出队管理器试图轮流从一个队列出队一个分组,从另一个队列出队下一个分组。即,如果两个逻辑队列都不为空,那么它们中的分组将会以 1: 1 的比率出队。

3. 实现

我们把这个队列对象的 C ++ 名字命名为" DtRrQueue(drop-tail round-robin queue)",它是类" Queue "的子类。其相应的 OTcl 名字叫作" Queue/DTRR "。当在类" Queue "中(in" queue. cc ")实现的成员函数" recv "接收到一个分组时,它就调用队列对象的入队成员函数。如果 link 对象没有被阻塞,它还会调用出队成员函数。若一个 link 脱离阻塞状态,它也调用队列对象的出队成员函数。所以,我们需要写出" DtRrQueue "对象类的出队和入队成员函数。图 14. 4. 14 展示了" DtRrQueue "的类定义以及它的入队、出队成员函数。

```
class DtRrQueue: public Queue {
 public:
        DtRrQueue() {
              q1_=new PacketQueue;
              q2_=new PacketQueue;
              pq_=q1_;
              deq_turn_=1;
        }
 protected:
         void enque(Pqcket* );
         packet* deque();

         packetQueue * q1_;      // First FIFO queue
         packetQueue * q2_;      // Second FIFO queue
         int deq_turn_;           // 1 for First queue 2
         for Second
};
```

```
void DtRrQueue::enque(Packet* p)
{
  hdr_ip* iph=hdr_ip::access(p);

  // if IPv6 priority=15 enqueue to queue1
  if (iph->prio_==15){
    q1_->enque(p);
    if ((q1_->length()+q2_->length())>q1im_)(
     q1_->remove(p);
  drop(p);
    }
  }
  else{
    q2_->enque(p);
    if((q1_->length()+q2_->length())>q1im_){
     q2_->remove(p);
     drop(p);
    }
  }
}
```

**图** 14. 4. 14　tRrQueue **类的实现**

```
Packet* DtRrQueue::deque()
{
 Packet *p;
 if (deq_turn_==1){
       p=q1_->deque();
       if (p==0){
           p=q2_->deque();
           deq_turn_=1;
       }
       else
           deq_turn_=2;
 }
 else {
      p=q2_->deque();
      if (p==0){
          p=q1_->deque();
          deq_turn_=2;
      }
      else
          deq_turn_=1;
 }
 return(p);
}
```

**图** 14. 4. 14**(续)**

4. **模型测试**

**我们使用前一节测试基于“** UDPmm **”的“** MmApp **”的模型脚本,并将其中** r1 ~ r2 **链路**

上的 RED 队列改为 DTRR 队列。图 14.4.15 展示了这个变化。

```
Set ns [new Simulator]
...
$ns duplex - link $node _ ( s1 ) $node _ ( r1 ) 5Mb
3ms DropTail
$ns duplex - link $node _ ( s2 ) $node _ ( r1 ) 5Mb
3ms DropTail
$ns duplex -link $node_(r1) $node_(r2) 2Mb 10ms DTRR
$ns duplex - link $node _ ( s3 ) $node _ ( r2 ) 5Mb
3ms DropTail
$ns duplex - link $node _ ( s4 ) $node _ ( r2 ) 5Mb
3ms DropTail

# Set DTRR queue size to 20
$ns queue -limit $node_(r1) $node_(r2) 20
...
#Simulation Scenario
$ns at 0.0 "$ftp start"
$ns at 1.0 "$mmapp_s start"
$ns at 7.0 "finish"

$ns run
```

图 14.4.15 “DtRrQueue”模型测试脚本

# 第15章

# 基于网络处理器平台的实现

## 15.1 网络处理器综述

随着 Internet 主干网络流量的指数性增长，主干路由器的处理线速已从 622Mbps 增加到 2.4Gbps，并且很快将达到 10Gbps[1]，特别是大量的多媒体和高清晰度电视应用的出现，促使这一增长进一步加快。同时，复杂的 QoS 控制诸如 DiffServ（differented services）[2]，RED（random early detection）[3] 以及新的网络协议和服务已成为研究热点。而基于 ASIC（application integrated specific circuit）和 GPP（general purpose processor）的传统网络处理方案已经不能同时满足处理速度和灵活性这两方面的要求。为此，基于 ASIP（application specific instruction processor）技术的网络处理器（network processor）得到了广泛的发展。近来，一块网络处理器芯片上已经集成了多个为网络处理优化过的处理器，同时还集成了一些专用硬件处理单元如 CRC 校验、Hash 单元等，并且在处理线速上已经达到或超过 10Gbps。它的出现兼顾了 GPP 的灵活性和 ASIC 的执行效率，为从第 2 层～第 7 层的多种应用提供了良好的支持。

与此同时，围绕着网络处理器应用所开展的相关研究也得到了飞速的发展，一些企业和学校也给予了足够的重视。比如 Intel 公司专门投资支持全球 100 所大学进行网络处理器及其相关应用的研究，并每年召开一次 Workshop 进行交流与总结。华盛顿州立大学 2002 年也专门召开了以网络处理器为主题的学术会议，并对网络处理器硬件体系结构和性能评价分析等方面作了深入的探讨。综合来看，与网络处理器相关的工作主要包括两个方面：一方面是针对网络处理器硬件本身，另一方面则是面向网络处理器的应用。前者侧重于网络处理器的硬件系统设计和性能改善与提升，比如硬件体系结构的研究[4]、网络处理器的集成设计[5] 等。后者则是基于网络处理器的相关应用研究，其核心问题就是要充分发挥网络处理器灵活性和高性能的特点，构建并实现一定的网络服务及应用。例如基于网络处理器的可编程路由器研究[6]、集成 QoS 控制[1]、网络处理器的性能分析与测试[7] 等。

本节[68] 据此首先从网络处理器的自身特性入手，介绍了其硬件结构和基本技术特点；接下来，从应用的角度，分析了系统设计和应用所面临的主要问题及挑战；最后，在网络处理器现有研究成果的基础上展望了网络处理器的发展方向及相关工作。

### 15.1.1 网络处理器的硬件结构及基本处理技术

随着 DWDM[8] 等光纤技术在主干网络的广泛应用,每 12 个月光纤链路的容量就增大一倍,而按照摩尔定律,电处理器的处理速度则是每 18 个月增长一倍,因此网络上以电处理器为核心的 IP 路由器已经成为网络的瓶颈。与此同时,随着大量新网络协议和网络服务的出现又进一步增加了网络处理的复杂度。文献[9]中将网络处理分为三个层次:低层、IP 层、应用层。低层(low or micro level),比如以太网中的 CRC 校验和表查找(table lookup)等;IP 层(IP level),如 IP 路由、DRR、NAT 等;应用层(application level),如 MD5,SSL,URL 交换等。为了弥补处理能力和链路带宽爆炸之间的差距以及兼顾处理所需要的足够高的可编程性和灵活性,网络处理器本身必须具备:①拥有网络分组并行处理能力;②具有高效的处理速度,能够达到分组的实时处理;③拥有一定数目的网络专用协处理器;④具有高度的可编程性和可扩展性;⑤能够快速投向市场,尽量减小再开发周期。

为了迎合这些要求,网络处理器将网络处理任务划分为控制层和数据层两个层面,控制层面专门负责非实时性的管理和策略控制等,数据层面承载高速易变的数据实时处理。总体上,网络处理器主要采用以下几种硬件处理技术。

(1) 两种处理机制

网络处理器内部一般为多内核(multi-core)结构[10],这些内核通常可以分为两种:一种是具有一般运算能力和指令存储能力的处理单元 PE(processing element);另一种是能够完成特定处理任务的功能模块 FU(function unit),如 CRC 校验单元等。在现有的商业网络处理器中,这两种单元一般采用以下两种组织机制:

- 流水线:每一个内核被设计成具有特定处理功能的模块,这些模块以流水线方式组织在一起完成对分组的处理。这种结构的网络处理器主要有 Cisco 的 PXF,Motorola 的 C-5 DCP 等。
- 并行处理:每个处理单元 PE 都可以完成相似的任务,多个处理单元彼此之间可以并行执行。这种结构的网络处理器主要有 Intel 的 IXP1200,IBM 的 PowerNP 以及 Agere 的 PayloadPlus 等。

(2) 优化的内存管理和 DMA 单元

在一般的多处理器系统中,内存操作往往是系统开销的一大瓶颈。而一般的网络处理器设备通常要对分组进行存储和复制等处理,需要执行大量的存储器操作。网络处理器为了优化这一操作往往引入经过优化的存储器接口和一些 DMA 单元,以提高存储器的操作效率。比如,在 Intel 的 IXP1200 中,提供了 SRAM 的空闲堆栈以及相应的处理指令。

(3) 优化的运算逻辑单元 ALU

不同于一般的处理器,网络处理器提供了一些用于网络处理的专用指令[11],比如位(字)扩展、压缩指令。同时,大部分网络处理器采用了 RISC 技术,可以使任何 ALU 运算在一个时钟周期内完成。

（4）网络专用的协处理器（co-processors）

现在，大部分网络处理器为一些特定的网络操作提供专用的硬件协处理[12]，比如专用的路由表查找引擎、硬件分类引擎、缓冲和队列管理引擎。这些协处理器有些集成在网络处理器片内，有些则安排在片外。

（5）硬件多线程技术

为了进一步提高处理器的利用率，网络处理器通常引入硬件多线程技术，并且线程间的切换开销为零。比如 Intel IXP1200 的每一个微引擎中拥有 4 个线程，每个线程拥有独立的程序计数器 PC。IBM PowerNP 中也有类似的结构。

下面以 Intel 的 IXP1200 为例来说明上面的特点。IXP1200 属于第一代成熟的网络处理器，由 Intel 公司提出设计要求，DEC 公司完成实现。IXP1200 是 Intel IXA（Internet exchange architecture）结构的最早组成部分，同时，它具有一般网络处理器的各种特点，并在实践中已经得到了一定的应用。IXP1200 主要是针对网络中第 2 层 ~ 第 4 层的应用，并且处理速率在 64 字节分组的情况下可以达到 2.5Mpackets/s。IXP1200 由 0.28μm 工艺制造，工作时钟频率最高 232MHz，功耗为 5W，制造时充分体现了 SoC（system on chip）的思想。其体系结构如图 15.1.1 所示。

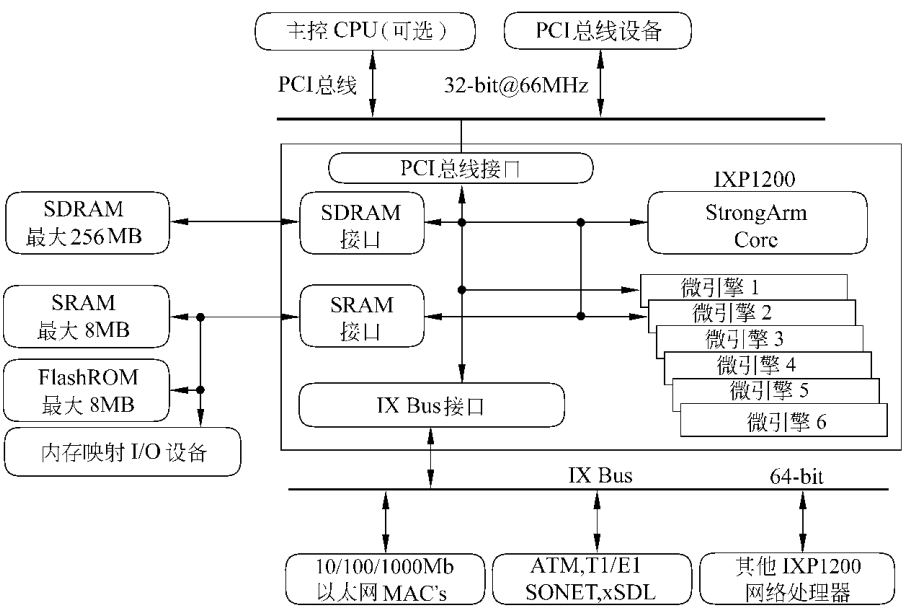



图 15.1.1　IXP1200 硬件系统结构

IXP1200 主要由 6 个可编程微引擎（microengine）和一个工作在 200MHz 的 StrongArm 处理器组成。峰值带宽为 6.6Gbps 的 64 位 IX Bus 将微引擎、StrongArm 处理器、存储器以及 MAC 等片外设备连接到一起。PCI 总线则用来和外部设备通信，完成 4 层以上的更高层网络处理。微引擎主要用来完成数据层面分组的实时处理，每个微引擎拥有 4 个硬件线程，并且这 4 个线程共用一个寄存器堆，寄存器的分配可由软件编程控制。微引擎具有单指令 ALU 功能，此外还拥有一些用于分组处理的特殊指令，比如比特提取、设置等。每

个微引擎最多可以存储 2K 的程序指令,指令的装载与微引擎的配置由 StrongArm(SA) Core 完成。此外,StrongArm 还承担其他一些慢速数据处理,它是控制层面的主要载体。为了提高 StrongArm 的性能和微引擎的通信能力,片上为 StrongArm 集成了一个 8KB 的数据缓存和一个 16KB 的指令缓存以及一个 4KB 的 ScratchPad SRAM。为了增强分组的处理能力,IXP1200 还提供了一个专用的硬件可编程 HASH 引擎以及专用 FIFO 单元用于从 MAC 接收和发送数据。同时,IXP1200 片上还集成了 SDRAM,SRAM 控制器和 PCI 接口控制器,它最多可以同时支持 256MB 的 SDRAM 及 8MB 的 SRAM。

综上所述,IXP1200 拥有网络处理器的一般特点。从系统角度来看,IXP1200 属于一个并行式的多处理器共享总线的计算机系统,多处理器(微引擎和 StrongArm)共享同一个存储器,多处理器彼此间的数据交换通过共享总线完成,其他网络协处理器等资源也通过共享总线完成资源共享。

## 15.1.2 系统设计与应用所面临的问题

### 15.1.2.1 系统处理特性

网络处理器所处理的是从第 2 层 ~ 第 7 层的任务,包括从简单的 IP 路由直到复杂的加密和压缩程序,因此它具有独特的系统处理特性。为了获得这一特性,Tilman Wolf 和 Jonathan S. Turner 在文献[7]中使用了 8 种不同层次的应用程序,进行了实际运行测试,得到了以下的系统处理复杂度的数据(见表 15.1.1),其中复杂度使用“RISC 指令/字节”来描述。

表 15.1.1 不同应用的处理复杂度

| 应用 | 类型描述 | 复杂度 | 应用 | 类型描述 | 复杂度 |
|---|---|---|---|---|---|
| RTR | 路由表查找 | 2.1 | JPEG | 图像压缩 | 81 |
| DRR | 分组调度 | 4.1 | CAST | 数据加密 | 104 |
| FRAG | 分组重组 | 7.7 | ZIP | 数据解压 | 226 |
| TCP | 流量监测 | 10 | REED | 传送纠错 | 603 |

同时,文献[7]中还宏观地总结了系统所处理任务的三个重要特性:①短任务:大部分任务需要 1 ~ 100 个 RISC 指令/字节;②大的任务数目:一般每秒要处理大于1 000 000 分组;③高异构的应用:任务彼此间的差异非常大,比如 IP 路由和 MPEG 编码。

这里,我们根据 Tilman Wolf 等人的数据分析一下常用分组处理的基本特点。以路由查找(RTR)和数据加密(CAST)为例简单估算一下这两种任务对处理器的要求。假设系统吞吐量为 1.2Gb/s(其余的数据见表 15.1.1):

RTR 所需要的运算量为 $1.2/8 \times 2.1 = 315$ MIPS,

CAST 需要的运算量为 $1.2/8 \times 104 = 15\ 600$ MIPS。

而现在网络处理器中单个处理元的处理能力仅为 1 500MIPS 左右,例如 IBM 的 NP2G 为 1 596MPIS、Intel 的 IXP1200 为 1 396MIPS。可见,系统负载处理的开销是相当庞

大的，并且处理的层次越高其开销越大。

#### 15.1.2.2　系统并行性要求

从 15.1.2.1 节中的分析可以看出，由于处理负载的开销大大超过了现有网络处理器内部单个处理元的处理能力，因此在系统的处理策略上广泛使用了并行处理机制。具体来讲，并行处理的核心就是要充分利用处理任务间的彼此独立性，将不同的任务同时交给不同的服务员处理。从网络处理器所面向的处理任务来看，不同的处理层面其并行性特点也不尽相同。对于控制层面和管理层面的任务一般是实时性较差的管理及控制操作，例如，路由表的维护和端口状态轮循等。对于数据层面，则是实时性较强的分组处理等操作，各自的系统并行性特点见表 15.1.2。

表 15.1.2　网络处理器的系统并行特性

| 系统层面 | 性能要求 | 数据并行性 | 范　例 |
|---|---|---|---|
| 数据层面 | 高 | 高 | 分组路由 |
| 控制层面 | 低 | 低 | ATM 虚拟链路控制 |
| 管理层面 | 低 | 低 | SNMP 处理 |

因此可以认为，网络处理器系统的并行性要求集中体现在数据层面，为了迎合这一要求，一般采用以下三个层次的并行处理技术[13]：

（1）分组级并行（packet level parallelism，PLP）

大部分网络应用都是基于分组级（packet level）的，不同的分组一般经过类似的处理，分组间一般是独立的，这样分组可以由不同的服务员并行服务。然而这种处理也有一定的局限性，即当属于同一个流（flow）的分组经过不同服务员并行处理后必须有一个分组的重组保序过程。虽然上层的 TCP 等协议允许分组乱序，但是为了减小在终端上重组分组的时间，一般还应保证分组的顺序。

（2）分组内并行（intra packet parallelism，IPP）

在对分组内部处理的过程中，有些任务也是彼此独立的。这种思路类似于微处理器设计中的线程级并行（thread level parallelism）。例如，一个二层交换程序，源 MAC 地址的处理和目的 MAC 地址的查找是完全独立的，可以并行处理。同时还可以引入一些专用的硬件协处理器，以增加 IPP 的并行效果。另外，增加一些专用的处理指令也是提高 IPP 效果的有效手段[13]。

（3）指令级并行（instruction level parallelism，ILP）

在网络处理器内部，通过特殊的硬件结构完成处理器指令的并行执行，比如 VLIW 和超标量技术。此外，有些处理器还使用了线程级并行（thread level parallelism）技术[14,15]。然而，模拟研究表明，在一般非特定的整数科学计算应用中 ILP 是非常有限的，因此，现在大部分网络处理器出于成本的考虑并没有大量使用 ILP 技术。

以上三种技术同时出现在网络处理器硬件结构和应用设计中。现在的商用网络处理器大都采用 PLP 并行技术，即将多个简单的 RISC 处理引擎（PE）集成到一个芯片上，并通

过集成软件开发环境完成对多个处理引擎的编程[16]。为了提高集成度,RISC 内核被设计得尽量简单,并且一般都不具备浮点运算指令及超标量等 ILP 技术。

在以上的三种并行技术中,硬件实现 PLP 是最简单的,但是对于相同的芯片面积存在着 PE 数目和 PE 复杂度的一个设计均衡(tradeoff)问题。如果单个 PE 很简单,则可以在相同的面积上集成多个 PE,但是单个 PE 的处理能力会比较差 如果减少 PE 的集成数目,提高单个 PE 的处理能力(包括引入 IPP 和 ILP 机制),将减小分组在单个 PE 中的处理时间,不过,系统的 PLP 并行度会受到一定的损失。一些模拟的结果表明[13],通过增加 PE 数目的做法虽然可以增加 PLP 效果,但是随着 PE 的增加其利用率将下降,同时,增大分组重组缓冲将有助于提高 PE 的利用率。不过,随着缓冲区的增大,PE 利用率很快达到饱和。因此,对于网络处理器本身在设计时将尽量减小 PE 的数目,同时要求 IPP 与 ILP 两种策略的合理应用。

对于应用设计,一般是依靠网络处理器提供的硬件并行特点,在分组处理的各个阶段适当地引入并行策略以提高分组处理效率。由于现在的商用网络处理器都提供 PLP 硬件支持,因此现有的大部分应用设计主要体现的是 PLP 技术。

#### 15.1.2.3 建立 Gigabit 链路系统的挑战

在使用网络处理器构建 Gigabit 链路系统(如可编程 QoS 路由器等)时,在系统的结构设计、策略使用、应用开发等各个层次存在着如下问题。

1. 并行策略所引入的问题

如 15.1.2.2 节所述,在网络处理器中大量采用并行处理技术,虽然提高了分组处理的效率,但却带来了以下三个问题:

(1) 同步问题

在 PLP 策略中,分组交由不同的 PE 处理。但由于分组的特点,不同 PE 所处理的分组并不总是相互独立的,有时分组间也存在着一定的制约关系。这种制约关系主要体现在服务时序和资源操作两个方面,对于服务时序引起的同步问题可以通过维护分组处理状态来解决,对于资源操作冲突则需要通过加锁机制来解决。同样的同步问题在 IPP 和 ILP 策略中也存在。

(2) 处理器资源调度问题

简单地讲,即当一个待处理的分组到达时将由哪一个 PE 来服务,如何来服务的问题。在调度时需要考虑包括系统负载和流量特性在内的多种信息,并要保证效率和公平性。目前,此种资源调度算法一般分为静态和动态两类。文献[6,17]中采用了静态算法,即 PE(或硬件线程)静态地对应不同的端口,不同的 PE(或硬件线程)只负责特定端口的收发操作。文献[18]采用了基于 EHDA(enhanced hash-based distribution algorithm)的动态分配算法,算法考虑了 TCP 与 UDP 两种流的特性以及 PE 的负载情况。

实际上,网络处理器内的其他共享资源(如网络协处理器等)也存在着同样的问题。

(3) 模块间通信问题

为了解决同步以及系统控制等问题，网络处理器内部 PE 间的通信以及与外部设备间的通信越来越多，甚至成为系统性能的瓶颈。为了提高系统集成度，大部分网络处理器内部采用共享总线的通信方式，内部没有其他专用的数据通路。但是这种结构却严重地影响了系统的同步及通信策略。不过，这一问题可以通过增加专用的数据总线得以改善，然而它却会增加系统设计的开销。

（4）并行处理与流水线处理的关系

并行处理虽然可以提高分组处理的效率，但是它将引入大量的分组状态维护工作，如果分组状态存取过于频繁，反而会降低分组处理的效率。例如，在 DiffServ 边界路由器中引入了基于类的调度（class based scheduling）和基于流的调度（per-flow scheduling）。对于基于流的调度使用 WRR 算法[19]，文献[1]中给出了 2.4Gbps DiffServ 边界路由器在进行普通报头处理和 WRR 调度时，并行处理与流水线处理的数据量，如图 15.1.2 所示。对于 WRR 调度，由于使用并行处理要频繁地对分组状态进行更新反而增大了运算量。因此，何时采用并行处理也是十分讲究的。

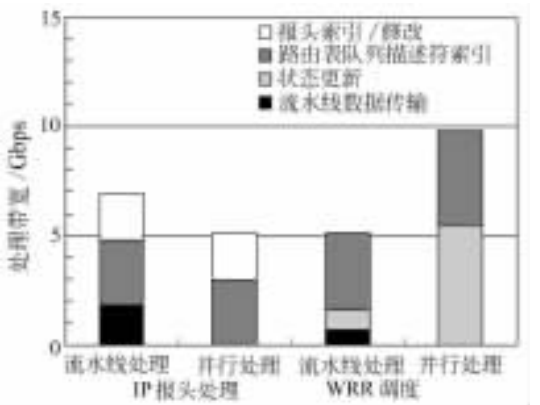



图 15.1.2　并行处理与流水线处理数据

2. 系统复杂度和 SoC 优化问题

为了适应新一代 Gigabit 网络的灵活性和高性能，通常要求网络处理器系统采用 SoC 的设计方式。SoC 的基本目的就是为了降低系统的成本和提高整体性能。同时，为了满足复杂的实时性处理要求，系统还需要在网络处理器内部储存一定数目的指令代码，如 Intel 的 IXP1200 为每个微引擎提供了 2KB 的指令空间以及一些用于高速运算和分组状态存储的存储区域[15]。然而，目前芯片设计技术还存在着诸多困难[20]，在一块硅片上集成特性不同的异种功能单元电路增加了 SoC 的困难，同时不同单元所占芯片面积也存在着彼此的均衡问题，其中最突出的就是 Cache 大小和 I/O 通道数目的设计。文献[5]对网络处理器的 SoC 优化得出了一些结论，并通过模型的方法分析了多处理器进行 SoC 时的性能，认为对于特定面积的网络处理器，其 Cache 的大小和 I/O 通道数目随着片上处理器的增加而减少，并存在一个最优配置的问题。

3. **延时隐藏问题**(latency hiding)

对于 Gigabit 的链路来说,分组到达的间隔为几百纳秒,通常的 SRAM 访问时间在 10ns 左右,而处理器在访问存储器时处于空闲状态,因此存储器访问延时将带来巨大的处理器周期浪费。同样,这种延时出现在对协处理器以及其他一些外部资源的访问过程中,例如 Intel 的 IXP1200 对于 SRAM 的访问速度为 30 个微引擎周期,对片外 FIFO 的访问速度在 40 个微引擎周期左右。

通过调整并行执行的时序可以有效地将这些延时隐藏掉,其基本思路如图 15.1.3 所示。图中实线代表处理器实际运行的时间,虚线代表访问延时等待时间。通过调整处理时序,合理地利用访问等待时处理器的空闲时间,在宏观上可以达到隐藏访问延时的效果。

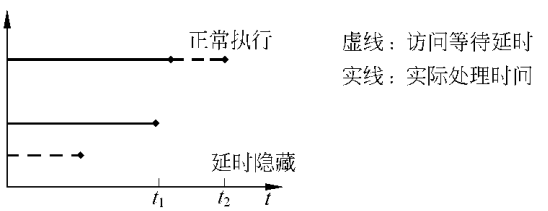



图 15.1.3 延时隐藏的基本原理

对于一般的计算机系统,解决内存访问延时可以通过预取内存地址来实现。但这种方法对于网络处理器中具有大量流操作的数据层面处理是没有什么效果的,而对其控制层面的诸如路由表更新等一些操作却是非常有效的。此外,这种方法不需要额外的寄存器或线程调度等额外开销。在数据层面上,大部分商用网络处理器使用硬件多线程技术来解决内存访问延时的问题。在运行时,这些线程一部分处于活动状态,另一部分处于等待状态,若一条活动线程遇到访问延时就立即交出执行权,由另一条等待线程获得处理器资源。如 Intel 的 IXP1200 每个微引擎拥有 4 个线程,其中一条可执行;IBM 的 PowerNP 每个 PE 拥有 4 个线程,其中两个可执行。此外,网络处理器一般还引入零开销的硬件线程切换。

虽然多线程技术得到了广泛的应用,但是线程间的调度和使用策略仍对系统公平性和利用率起着重要作用。例如,在文献[6]的设计中线程被静态地指派,当一个线程触发内存访问等操作时,由系统硬件仲裁器决定下一个可以激活的线程。然而,模型分析显示,这种简单的被动线程调度方式的处理器利用率受外部负载的影响重大,当分组长度增加时,处理器利用率将下降,其原因就是随着分组长度的也增加,内存访问量增加,这种被动的线程分配方式往往导致处理器内部所有线程都处于访问延时等待状态。

## 15.1.3 网络处理器的应用研究

### 15.1.3.1 基于网络处理器的现有研究工作

网络处理器被认为是推动下一代网络向灵活性和高性能发展的核心技术,围绕其本身及其相关应用展开了大量的研究并且已成为热门话题[21,22]。OPENARCH 2002 专门为

网络处理器开设了专栏进行讨论，INFOCOM 等重要国际会议也在可编程路由器等领域的研究中加大了网络处理器的比重。综合起来，现有的关于网络处理器研究的成果主要有以下几个方面。

1. 基于网络处理器的路由器体系结构研究

随着网络处理器的出现，将其用于可编程 QoS 路由器的研究[14,15] 得到了飞速的发展。其中主要是关于体系结构和相关算法的研究，并且在体系结构研究上有了完整的结论和部分的实现。其中比较成功的有 Princeton 大学的可扩充路由器 VERA[6] 以及 Columbia 大学的 Genesis[17]。以 Princeton 大学的 VERA 为例，它主要是基于 DiffServ 的体系结构，并兼顾可扩展性、兼容性、效率性三方面特点，不过总的看来，VERA 更侧重于服务的灵活性和可扩充性。其分组处理逻辑流程如图 15. 1. 4 所示。图中 C 代表一个层次化的分类器（classifier）。F 代表传送处理单元（forward），它负责完成分组的数据处理。S 为一个输出调度器（scheduler）。在具体实现上，该项目采用一种基于 Intel IXP1200 和 PC 的层次化体系结构，如图 15. 1. 5 所示。Intel IXP1200 中的微引擎主要负责数据层面的实时数据处理，StrongArm 和 PC 负责更高层次的非实时数据处理。此外，该项目组还在体系结构的性能评价方面做了一定的工作，文献[23]通过实际测量的手段考察了 VERA 各个层面的数据吞吐量、存储器访问频率、代码长度，并分别使用了三种不同的队列模型（单队列、多私有队列、多共用队列）。在性能测试的同时，他们还对三种不同队列模型所引入的 QoS 调度机制和同步控制策略做了一定的研究。在可扩展应用方面，VERA 给出了一些在 IXP1200 上实现的典型应用扩展所带来的开销，见表 15. 1. 3，进一步证明了其扩展的可行性。

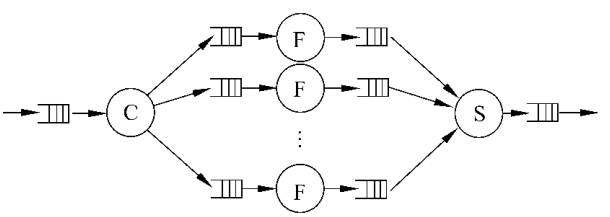



图 15. 1. 4　VERA 的分组处理流程

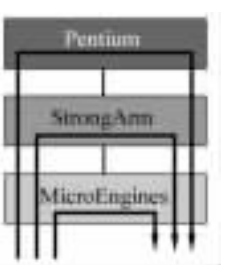



图 15. 1. 5　VERA 的层次化结构

表 15.1.3 一些典型应用扩展的时间及空间开销

| 典型应用 | SRAM 读写 (Byte) | 寄存器操作指令 | 所需寄存器数目 |
|---|---|---|---|
| TCP slicing | 24 | 45 | 7 |
| ACK monitor | 12 | 15 | 4 |
| SYN monitor | 4 | 5 | 0 |
| Smart dropper | 8 | 28 | 4 |

不过,VERA 在系统资源和处理器调度策略上只采用了简单的静态分配,一些代码的运行加载和卸载等动态特性与 Columbia 大学的 Genesis 中采用的 NetBind[24] 相比还存在着不足。同时,VERA 虽然在体系结构上具有完整的声明,但是在综合 QoS 控制策略以及系统的理论建模等工作方面还不尽完善。

2. 综合 QoS 控制策略的研究

QoS 控制简单来讲,是指网络能够提供有保证的、可预测的数据传输服务,满足不同用户的应用需求。最近,围绕着分组处理 QoS 控制已经展开了很多的研究,比如任务模型[25]、任务调度[26]、操作系统相关问题[27]、分组处理器(packet processor)体系结构[28]。

然而这些研究大部分都面向网络处理器分组处理中的某一阶段。方案对一个阶段是最优未必对全局就是最优。比如从系统的角度考虑,我们要提高系统的利用率使之获得最大的吞吐量,同时还要确保不同用户间的服务公平性;从用户的角度看,对一些分组延时抖动及丢失率则要给予充分的重视。

清华大学参加的 Intel IXA 大学项目就是以 QoS 控制中最关键的缓冲管理及队列调度为对象进行综合优化方案的研究。该项目以 Intel 的 IXP 系列网络处理器为实现平台,在研究综合方案的同时,对其在网络处理器上的优化也作了一定工作。该项目采用如图 15.1.6 所示的基于分类(class based)的多队列系统,在系统的前端,M1 是一个多维分类器(classifier),它是整个系统的重要基础,并且要能够满足多维性(报文分类的字段)和实时性。根据充分利用分类规则、索引排序、增加预处理时间以及网络处理器的硬件特性,该项目已经构建了五维的实时分类算法。M2 是缓冲管理模块,它主要负责缓冲区的分配,并根据一定的策略对分组进行丢弃,这主要会影响到分组丢失率和公平性。目前,该项目已经在 Intel 的 IXP1200 上实现了 Tail Drop[29] 和 PBS(partial buffer sharing)[30] 等经典算法,同时还在 PBS 的基础上提出了动态阈值的 DPBS(dynamic partial buffer sharing)算法[31]。M3 是分组的输出调度器,它主要是实现对链路带宽的管理,即按照一定的规则来决定从队列中选择分组进行发送,它影响的主要性能参数包括带宽分配、时延和时延抖动等。同样,项目在实现 WRR(weighted round robin)[19],DRR(deficit round robin)[32] 等经典调度算法的同时还对这些算法与缓冲管理及分类的综合组合方案进行了优化研究。在整体方案上还提出了基于 DiffServ 的拥有不同延时丢弃优先级的集成队列调度和缓冲管理方案[33],并将 QLT(queue length threshold)[34] 调度算法与 PBS 及 RED 缓冲管理算法相结合作为分析实例。

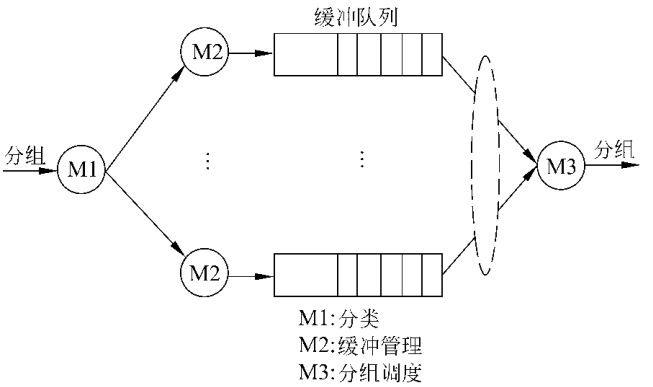



图 15.1.6　多队列系统图

另外，在综合方案的性能评价方面，该项目也取得了一定的成果。文献[31]指出，综合 QoS 方案是一个多目标问题，每个目标不可能同时满足，同时，文中还给出了综合吞吐量、队列延时、分组丢失率以及公平性在内的综合评价方案。其中比较重要的就是对 Power 公式[35]的改进。原始的 Power 公式是基于单队列系统的，增大 Power 值被认为是计算机网络的一个主要目标：

$$\text{Power} = \frac{\text{Throughput}^a}{\text{Delay}},\ 0 < a < 1$$

改进后的 Power 公式是基于多服务多类的，并在原有基础上综合考虑延时公平参数（delay fairness parameters，DFP）和丢失率公平参数（loss rate fairness parameters，LFP）：

$$\text{Power} = \sum \text{Power}_i = \sum \frac{T_i^a}{w_1 \cdot \bar{d}_i + w_2 \cdot \bar{l}_i},\ 0 < a < 1$$

式中 $T_i$，$\bar{d}_i$，$\bar{l}_i$ 分别为分类 i 的吞吐量、平均队列延时、平均丢失率；$w_1$，$w_2$ 为权重，它根据不同的环境和应用而发生变化。

3. 局部资源调度算法的研究

现在，大部分商用网络处理器系统是一个典型的多 RISC 内核的并行处理系统，同时存在着大量共享资源，如共享存储器、共享内存和协处理器等。依据系统特性对处理器、存储器等共享资源的有效调度已成为当前的热门课题，其中围绕着资源调度所展开的研究主要表现在两个方面：一方面是围绕处理器所展开的负载平衡和处理器调度算法的研究，另一方面是根据系统特性（外部特性和内部特性）所展开的分组调度算法的研究。

关于负载平衡方面的算法分类一般遵循 Casavant 和 Kuhl 在文献[36]中所采用的基于策略的分类规则。另外，Lazowska 和 Zahorjan[37]在研究了特定的适应性（adaptive）负载平衡策略后发现，简单的基于阈值的负载平衡策略可以在本质上改善系统的性能。在网络领域，随着 Web Server，Web Caching 和分布式数字图书馆的出现[38,39]，一些基于 HRW（highest random weight）的负载平衡算法也被应用到网络处理器中。不过，这些基于 HRW 的方法大多从用户请求空间角度入手，系统适应性较差。但也有些算法从服务空间出发，同时结合 TCP/IP 分组在路由器中并行处理算法[40]，并加上了一些动

态执行预测的技术,很好地解决了适应性的问题。综合来看,这类算法具有以下几个特性[41]:

(1) 基于一些随机映射(random mapping)算法,如 Hash-Based 和 HRW 等。同时要保证一定的容错性(faults tolerance)。

(2) 定义具有动态特性的系统利用率函数,即将系统利用率定义为时间的函数。

(3) 定义触发阈值,只有当系统利用率达到阈值时才触发负载平衡策略。这样是为了避免平衡策略触发得过于频繁而降低系统的性能。

但这类算法也存在着它的局限性,它通常是基于流映射的(flow mapping)[42],虽然不需要对流状态进行保留,可是它假定流与流之间是同构的,对流处理预测时采用相同的算法。这对于采用 DiffServ 结构的路由器等基于多分类的应用是不适宜的。

对于分组调度算法,简单地讲,就是当一个分组到来时将进行何种处理,哪一个分组先处理,哪一个分组后处理。通常,基于网络处理器的分组处理系统逻辑结构如图 15.1.7所示[43]。

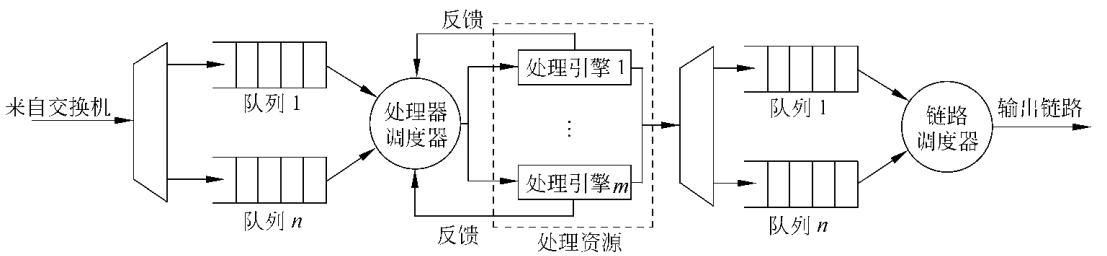



图 15.1.7 分组处理系统的结构图

系统采用一种类似输入输出队列的结构(CBIO),输入输出队列间是一组并行且同构的服务员,在这组服务员前面设有一个处理器调度器(processor scheduler),负责分组的处理调度,在输出端队列后面设有一个链路调度器(link scheduler),负责分组的发送调度。分组调度及处理算法的核心就是处理器调度器和链路调度器的算法设计。而传统的分组处理系统只是简单的存储-转发过程,仅涉及到链路调度器。因此,对于网络处理器系统则要综合考虑两方面。

无论是哪种调度器,其核心问题就是对分组处理时间进行预测[44],对于具有多个通用处理器的网络处理器系统尤为明显。在链路调度器方面,分组调度策略(基于带宽调度)已经有了广泛的研究[44],并且一些性能评价参数如端到端延时、公平性等也与 GPS (general processor sharing) 策略[45] 作了相应的比较。但是一些基于 PFQ(packet fair queue)的分组调度策略则不能用来做处理器的调度,因为诸如 WFQ,$WF^2Q$[46] 等使用了虚拟时间的概念,这些虚拟时间要在调度器中被精确地更新,也就是说,调度器需要精确地知道分组处理所耗费的时间,而对于处理器这是十分困难的。虽然一些算法,诸如 SFQ (start-time fair queueing)[47] 不需要知道分组处理的时间。但是,SFQ 在最坏情况下,系统延时将随着流数目的增加而增加,并在一些流相关负载(correlated cross-traffic)下变得更糟[48]。

因此,一些研究在避免模拟 GPS 策略复杂性的同时尽力提高算法理论延时和公平性等系统性能参数[49]。为此,这些算法使用预测的分组服务时间作为调度依据,文献[43]给出了一种常用的预测分组处理开销的算法:

$$c = \alpha_a + \beta_a \cdot l_i,$$

$$\beta_a = \frac{\sum_n c_i \cdot l_i - \sum_n c_i \cdot \sum_n l_i / n}{\sum_n l_i^2 - \sum_n l_i \cdot \sum_n l_i / n},$$

$$\alpha_a = \sum_n c_i - \beta_a \cdot \sum_n l_i,$$

式中 c 为长度为 l 的分组进行 a 处理的开销,$\alpha_a$ 为平均每个分组进行 a 处理的开销,$\beta_a$ 为平均每字节进行 a 处理的开销;调度器运行时为每一种处理 a 维护如下的统计信息:$\sum c_i$, $\sum l_i$, $\sum c_i^2$, $\sum c_i \cdot l_i$。

在分组调度策略的选择上,链路调度器大部分使用现有的 PFQ 策略,而处理器调度大部分采用 FCFS(first come first serve),T-Opt(throughput-optimal),LA(locality-aware),LAP(locality-aware predictive)[50]。

4. 网络处理的性能评价

网络处理器性能评价研究的主要目的是为了在众多系统方案中选择一个最适合需要的方案,同时对现有系统方案性能的缺陷和瓶颈进行改进和提高,再对未来设计的系统进行性能预测。具体的性能评价工作包括两方面:一方面是对系统应用方案进行性能评价;另一方面是对网络处理器本身进行性能评价。前者的目的是在现有系统上评价应用方案的性能,为应用方案的设计与改进提供依据。后者则是从网络处理器本身出发,对其体系结构作性能分析,为网络处理器的设计与体系结构的改进提供必要的支持。

现在商业网络处理器正在以惊人的速度增长,已经有 30 个厂商超过 500 个设计[51],这些网络处理器的体系结构、内存及系统接口都不尽相同,这给性能评价和方案间的相互比较带来了巨大的困难[10]。因此,在具体的性能评价工作上一般采用实际测量(benchmark)和模型两种评价方式。由于网络处理器的高异构性,现有的大部分性能评价工作都是基于实际测量的。Nemirovsky 在文献[52]中讨论了实际测量的要求以及所面临的挑战,同时为网络处理器的测量方法定义了一组评价指标和测量软件的设计原则。除此之外,Mel Tsai[53] 等人还给出了用于网络处理器的通用测量软件的两项基本原则:测试标准的通用性;测试例程符合真实网络应用。

在具体测量软件方面,目前一些流行的测量软件如 SPEC[54],Dhrystone[55],MediaBench[56] 是面向同构系统的,对于网络处理器系统并不太适合,因此出现了一些专门评价网络处理器的软件[7,9,57,58],这些软件按照不同的分类提供了一系列的测试程序,并且使用它们已经得到了一些关于网络处理器的系统负载[7]、嵌入式集成[5] 等重要数据。其中比较著名的测试工具有 CommBench[7],NetBench[9],EEMBC[57] 及 NPF-BWG[58],它们的技术特性见表 15. 1. 4。

表 15. 1. 4　典型测试软件技术特性

| 技术特性 | CommBench | NetBench | EEMBC | NPF-BWG |
|---|---|---|---|---|
| 提供的测试例程 | 8 | 9 | 34 | |
| 测试例程分类 | 按负载轻重 | 按处理层次 | 按应用对象 | 按处理层次 |
| 例程执行描述 | 有 | 有 | 有 | 未知 |
| 测试例程的选择方法 | 提供 | 提供 | 提供 | 提供 |
| 环境与系统测试 | 无 | 无 | 无 | 有 |

它们基本上都是面向网络处理器应用的,除 NPF-BWG 以外都没有从系统的角度对整个系统进行评价。比如 CommBench 提供的 8 组测试程序,其中 4 个被用作报头处理测试,另外 4 个被用来作负载测试,它们很好地反映了 3.1 节中提到的网络处理器的系统特性。但是 CommBench 是基于单服务员的,且没有明确地说明如何将其应用于一些多内核的网络处理器,如 Intel 的 IXP1200。类似地,NetBench 也定义了 9 组测试程序并将这 9 组程序分为低层、IP 层、应用层三个不同处理层次,经过这种分类,NetBench 的测试在底层结构(micro-architecture)反映了网络处理器的异构特性(heterogeneous),并且已经在 SimpleScalar[59] 模拟器和 Intel IXP1200 系统上得到了一定的验证。但是对于其他层次,这种异构特性并不十分明显。EEMBC 则从网络应用对象出发,给出了面向工业、普通消费者等终端对象的不同的测试程序,其中包括一些 OSPF 等协议测试。

在理论建模方面,Patrick Crowley[60] 等人给出了一种基于可编程网络接口的网络处理器的模型,同时将模型划分为系统模型、应用模型、流量负载模型三个子模型,并对模型的负载特性做了详细的评价分析。但是他们的模型还不十分完善,只涉及到真实网络处理器系统的一小部分,并且没有将系统功能模块与环境模块分开考虑。总的来看,目前围绕着网络处理器所展开的性能评价工作大部分是基于实际测量的,模型化的方法还不十分完善。

5. 网络处理器的应用

网络处理器的应用研究得到了广泛的开展,与此同时涌现出了一些成功的应用范例。这些应用集中地体现在以下几个方面:

(1)基于网络处理器的智能路由交换设备,包括路由器体系结构的研究和功能模块的构建。比如 Alcatel and Asante Technologies 使用 IBM 的 PowerNP 构建其核心路由及交换设备[61],Broadband Access Systems,Mayan Systems,NorthChurch[62] 及 Cloudshield Technologies[63] 使用 Intel 的 IXP1200 构建起网络路由设备。

(2)智能安全设备的应用:防火墙、VPN 及入侵检测设备等。

(3)无线网络的应用:Ad Hoc 无线网络中的路由设备、3G 协议栈的实现等。

(4)系统的监测控制或模拟设备:一些研究机构将网络处理器作为其系统测试和实验设备,比如 Empirix 使用 Motorola 的 C-5 作为其网络模拟器用于在线速情况下获得网络延时、抖动、丢失率等参数[64]。

6. 网络处理器其他相关研究

（1）主动网络和可编程网络的研究

这些研究包括执行代码的动态移植和发布、代码的安全执行、资源共享调度等。其中一些研究成果也被应用到网络处理器中，比如 Columbia 大学的 Genesis 将可编程虚拟网络中的概念移植到网络处理器系统中，在 Intel IXP1200 的平台上构建了可在运行时装卸代码的动态数据层面[17]。

（2）处理器系统体系结构的研究

这部分研究与网络处理器相关的主要是对处理器内部组件的研究（如 RISC 内核、协处理器等），以及内存接口、高速数据总线等接口的研究。这些研究主要着眼于硬件体系结构上解决网络处理器的性能瓶颈。

（3）第二代商用网络处理器的开发与研制

大部分厂商都展开了对下一代商用网络处理器的研究，并将其产品定位到核心网络设备。其中以 Intel IXP 2K 系列为代表的第二代网络处理器并没有增加过多的并行处理内核，而是大幅度地提高了处理器的执行效率，同时加强了多处理器间的通信能力，并增设一些硬件指令执行单元，如乘法硬件指令。不过，这些研究与改进主要还是面向提高报头处理效率，至于加密等复杂的数据计算处理暂时还没有专门涉及。

### 15.1.3.2　网络处理器的发展方向和相关工作

随着网络应用的数据码率呈指数级增长，网络处理器的研究与发展越来越受到网络的边缘和独立节点的影响，今天的核心设备支持 2.5Gbps 链路，到了下一代就是 10Gbps 甚至是 40Gbps，而边缘设备虽然在链路带宽方面落后于核心设备一代，但是它将承载更为复杂的处理任务。围绕着网络处理器所开展的相关工作也将迎合网络的边缘设备的发展需求。下面将从网络应用、网络处理器结构和相关研究这三个方面展望网络处理器的发展。

1. 网络应用的发展

如今的网络应用除了链路速度不断提高之外，传统的 OSI 模型也已经被打破，越来越多的高层信息被用来决定低层次操作。比如，Web 交换或者 URL 负载平衡使用 TCP 端口和 HTTP 协议信息来决定相应的路由操作。但由于高层协议和应用一般遵循端到端（peer to peer）的原则，它们的变化非常频繁而且很难预测。因此，对网络处理器灵活性和性能的要求也越来越强。

另外，IPv6 的出现给现有的基于 IP 地址的查找运算带来了巨大变化，其中多维查找算法（多个字段）将会越来越普遍，平均每关键字查找数据量将大于 300 比特，而现有的协处理器最多只能处理每关键字 288 比特的数据量[10]。为此，人们对 IPv6 的高速路由查找和分类算法方面的研究正在积极地展开。此外，IPv6 需要实现 IPSec 协议，因此在通常的报头处理之余，网络处理器还需进行大量的加密、解密运算。所以，IPv6 的出现会使下一代网络处理器向能胜任大计算量任务的方向发展。

最后,随着 MPLS 的加速应用,MPLS 将作为以太网、SONET、ATM 等多种主流协议的协同工作标准,作为标签交换路由器(label switch router)将同时支持多种协议,而网络处理器的灵活性、高效性必将在标签路由系统中得以广泛应用。但是网络处理器这种通用处理计算结构是否能够很好地适应其特殊的交换式结构还有待研究,相关的算法和体系结构研究也必将得到重视。

2. 网络处理器自身体系结构的进化

随着网络应用的发展和摩尔定律的限制,网络处理器自身体系结构也在不断地进化,主要表现在并行结构、协处理器、通信体系结构三个方面。

(1) 并行结构:数据的增长将会超过摩尔定律的增长,3.2 节已分析过并行结构将是弥补这一缺陷的有效手段。随着半导体工艺的改进,新一代网络处理器的运行时钟已经超过了第一代网络处理器一倍以上,其中 Broadcom 的 Mercurian 运行速度达到了 1GHz[65]。而新一代网络处理器中 PE 的数目并不会有过多的增长,而是向着提高 PE 速度的方向发展。

(2) 协处理器的广泛应用:网络处理器本身是基于 ASIP 技术的,为了提高性能,网络处理器中部分常用单元将以 ASIC 协处理器的形式出现,同时,原先一些片外协处理器也将逐步集成到网络处理器芯片内部。另外,协处理器所处理的任务也将越来越复杂,Vissers 等人[66,67]已经展示了一种具有多媒体处理协处理器的设计。

(3) 通信体系结构:第一代网络处理器的通信体系结构一般采用简单的共享总线共享内存的方式,新一代的网络处理器不但增强了处理器内部 PE 间的通信能力(使用专用数据通路和寄存器),而且在处理器外部接口通信能力的提高方面也做了很多的工作,比如 Intel 的 IXP2400 增加了交换光纤通信接口。同时,为了解决多 PE 竞争内存的瓶颈,一些设计还采用了增加内存操作通道的方法进行改进。

表 15.1.5 以 Intel 的 IXP2400 与 IXP1200 的比较为例,说明了以上三个方面的改进。

表 15.1.5 IXP1200 与 IXP2400 网络处理器的比较

| 对比项目 | | IXP2400 | IXP1200 |
|---|---|---|---|
| 并行结构 | 内核工作频率 | 600Mhz | 200Mhz |
| | 微引擎数目 | 8 | 6 |
| 协处理器 | 硬件乘法单元 | 有 | 无 |
| | ATM 分段重组单元 | 有 | 无 |
| 通信体系结构 | 微引擎间专用数据通道及寄存器 | 有 | 无 |
| | 微引擎内局部存储器 | 640Byte | 无 |

3. 相关研究工作的开展

15.1.3.1 节总结了一些现有的工作,虽然有些已经取得了一定的成果,但是仍有很多问题有待解决,集中体现在以下几个方面:

(1) 综合资源调度方案的研究:目前,大部分资源调度都是面向某一局部的[43,50],从

系统角度的资源调度方案以及相应的综合 QoS 控制策略方面的研究，将是未来一段时间内研究的热点。

（2）PE 内硬件线程的调度以及延时隐藏技术的研究：3.3 节中提到了延时隐藏这一技术，同时 PE 内多硬件线程的结构正是为了达到延时隐藏这一要求。但现在大多数设计[6,17]只是静态地使用 PE 内的线程，仅依靠线程硬件执行特点而没有使用任何主动线程调度算法，这使得延时隐藏的效果受到了一定影响。今后将会有包括 PE 线程调度在内的更多延时隐藏技术受到重视，相应的理论模型及其分析工作也将得到广泛开展。

（3）网络处理器性能评价问题：正如 15.1.3.1 节中所提到的，目前网络处理器的性能评价主要依靠实际测量的方法，今后其性能评价研究将向模型、模拟、测量三者结合的方向发展。完整的系统模型和相应的模拟器将是性能评价的首选，同时，测量软件也将逐步完善，测量评价的标准也将得到统一。

（4）网络处理器软件开发工具和操作系统：随着网络处理的发展，一些相关的开发工具也得到了飞速发展，大部分网络处理器已经可以使用 C 语言进行开发，同时也出现了一些支持网络处理器运行的操作系统。在开发工具方面，将向着集成化开发环境（IDE）发展，如 Intel 的 WorkBench。并且在开发套件里提供一些专门用于 IP 协议的 API 和子程序，另外，Network Processing Forum 也正在为网络处理器开发通用的软件接口。在操作系统方面，将从现有的嵌入式操作系统，如 Linux，VxWorks 等向专用的网络处理器操作系统过渡，比如 Princeton 大学的 Vera 所使用的 Scout OS[27]。

## 15.2　基于 Intel 网络处理器的路由器队列管理

计算机网络的 QoS 控制模型有许多机制，比如，集成服务的资源预留（信令协议 RSVP）、接纳控制等；区分服务的流量整形/管制、分组标记等。其中不同服务模型所共有的部分，也是一个核心部分是缓冲管理和分组调度，本节统称为队列管理。

缓冲管理实现对缓冲区的管理，研究的主要内容是缓冲区如何分配和当缓冲区占用率到达一定程度时如何选择分组进行丢弃，所影响的性能参数主要是分组丢失率。而分组调度则是对链路带宽的管理，是指按照一定的规则来决定从多个等待队列中选择哪个分组进行发送，它影响的主要性能参数包括带宽分配、时延和时延抖动。目前已经有许多关于缓冲管理和分组调度的算法（参见第 9、第 10 章）。

队列管理是路由器的一个重要的子系统，因此，研究队列管理的设计和实现技术是路由器开发的一个关键部分。

从传统意义上来看，网络设备一般基于 CPU 或 ASIC。也就是说，队列管理算法的实现一般是基于 CPU 的软件实现，或者是基于 ASIC 的硬件实现。然而，基于 CPU 的软件实现虽然灵活，但很难获得高性能，而基于 ASIC 的硬件实现虽然性能较高，但灵活性很差，而且费用高。这使得传统的实现机制不能很好地满足当前网络高速度和多样化业务的发展趋势。因此有必要采用更先进的实现平台，研究新的实现技术。

网络处理器（network processor）就是一种很好的选择[68]。通过良好的体系结构设计，网络处理器将软件的灵活性和硬件的高性能特点结合在一起，是继 ASIC 之后一个新

的技术发展方向。目前,世界上有很多公司和大学在研究网络处理器技术。网络处理器的一般特点是,专门针对网络数据处理而优化了的指令集和硬件级并行处理(多个片内处理器、多硬件线程、多组高速总线、多组存储器、多个 I/O 接口单元等)。

基于网络处理器设计开发网络软件系统是一个新兴领域,面临许多技术挑战,比如多个线程之间的同步问题。因此,基于网络处理器的队列管理服务模块设计的关键是如何有效地管理系统多线程以提高任务处理的并行性,以及如何动态编程以进行灵活的系统配置。具体来讲,需要研究下列问题 如何解决系统同步问题 如何管理系统资源 如何对每个线程进行合理的任务分配 如何设计合理的模块化结构和清晰的接口等。

Intel 公司的 IXP 系列网络处理器是目前市场上的一种典型产品,是 Intel 公司 IXA (internet exchange architecture)架构的核心,其他很多公司的产品都有与之相似的结构。本节[69] 以 Intel 的 IXP1200 为平台,参照相对区分服务模型进行了队列管理模块和一种新的缓冲管理算法 DPBS(dynamic partial buffer sharing)的设计,并重点研究了几项关键技术。

## 15.2.1 体系结构设计

Intel 网络处理器 IXP1200 的基本结构如图 15.2.1 所示。包括 1 个通用处理核心 StrongArm、6 个 RISC 结构的专用可编程微引擎、1 个 IX 总线接口单元 FBI(fast bus interface,连接 MAC 等网络接口)、一个 SRAM 接口单元(外接 SRAM 存储器)、1 个 SDRAM 接口单元(外接 SDRAM 存储器)和 1 个 PCI 总线接口单元。每个微引擎具有独立的指令存储器、控制器以及寄存器,能支持 4 个硬件线程,并且线程之间可以进行零开销切换。

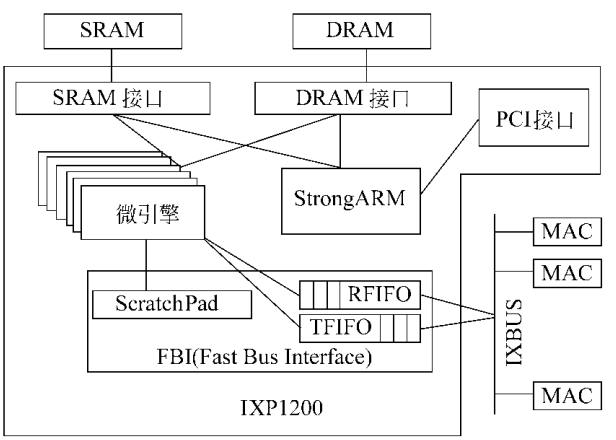



图 15.2.1 IXP1200 硬件系统结构

FBI 中含有两组缓冲寄存器 RFIFO 和 TFIFO,分别用于从/到网络接口的数据接收/发送。每组含有 16 个 64 字节的寄存器单元(element),每个寄存器单元可单独编址进行访问。另外,FBI 中还有一组 4KB 的片内 ScratchPad 存储器。

#### 15.2.1.1　软件体系结构

队列管理是软件路由器的一个重要的子系统。整个路由器是一种模块化结构，划分为两个部分：控制平面（control plane）和数据平面（data plane）。两个平面通过一组通信协议进行通信，相互协作完成网络节点的完整功能[70, 71]。在 IXP1200 系统中，StrongArm 负责控制平面的处理，微引擎负责数据平面的处理。

控制平面运行操作系统，完成诸如路由协议处理、系统管理等计算性较强的“慢通道”功能。有时也完成复杂分组，比如 IP 头部可选字段的处理。在数据平面，存在着一个或多个服务组件构成可扩展服务层来执行每个分组的“快通道”转发处理，比如分组头有效性检验、路由表查找、TTL 修改、分组分类、缓冲管理、分组调度等。数据平面从网络接收分组，分组经过转发处理（或被重定向到控制平面）之后，再发送到网络中去。

#### 15.2.1.2　模块接口

每一个服务组件构成一个软件模块，该模块的接口包含三部分内容：输入数据、输出数据、控制/配置信息。其一般化描述如下。

- 模块输入：源数据的信息（存放位置，用以描述数据的标签、类型、值等），操作者的信息（处理器位置、类型、编号等），操作规则（功能配置、规则数据库的位置等），同步信息。
- 模块输出：处理后的数据信息（数据处理后的存放位置，用以描述数据的标签、类型、值等），操作者的信息（处理器位置、类型、编号等），操作结果（错误信息等），同步信息。

#### 15.2.1.3　系统资源分配

为简化设计，本系统基本采用静态分配的资源管理模式。将系统配置为支持 8 个 10/100Mbps 和 1 个 1Gbps 以太网端口，但本文讨论以实现 8 个 10/100Mbps 为主。

微引擎/线程/FBI 缓冲寄存器：用两个微引擎负责 8 个 10/100Mbps 端口的数据接收（每个端口静态地对应于一个 FBI 接收缓冲寄存器 RFIFO element，每一个物理端口都由一个专门的硬件线程提供接收服务），用一个微引擎负责 8 个 10/100Mbps 端口的数据发送（每个端口静态地对应于一个 FBI 发送缓冲寄存器 TFIFO element，一个硬件线程循环服务两个端口）。如图 15.2.2 所示。

IXP1200 系统中有多种存储部件，根据其不同的特点，分别用于不同的存储目的。

寄存器（包括通用寄存器和传输寄存器）：速度最快，数量最少。用于同一微引擎内线程间的通信、临时变量数据。

ScratchPad：速度较快，数量较少。用于微引擎间的通信、中间结果暂存、意外处理结果记录。

SRAM：速度一般，数量较大。存放路由查找表格、分组和队列的描述信息、微引擎间的通信邮箱、微引擎和 StrongArm 间的通信管道等。

DRAM：速度最慢，数量最多。存放路由转发信息表、分组数据（payload）等。

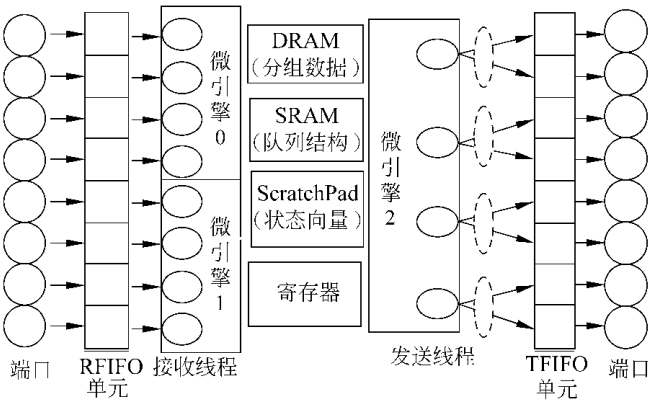



**图** 15. 2. 2　**系统资源分配**

### 15. 2. 1. 4　队列结构

**系统采用输出排队策略,即每个输出端口对应多个逻辑队列,**IP **分组按照其服务质量要求选择相应的队列。各个逻辑队列对应了不同的时延优先级,在每一个逻辑队列内部,分组又按照丢弃优先级分成多个子类。采用输出排队的原因如下:①输出排队最适合进行** QoS **控制**[72] **;②硬件系统中不存在交换机构,所有端口共享** IX bus **总线,不存在加速比问题。**

**如图** 15. 2. 3 **所示,该队列结构由下列部分组成:**

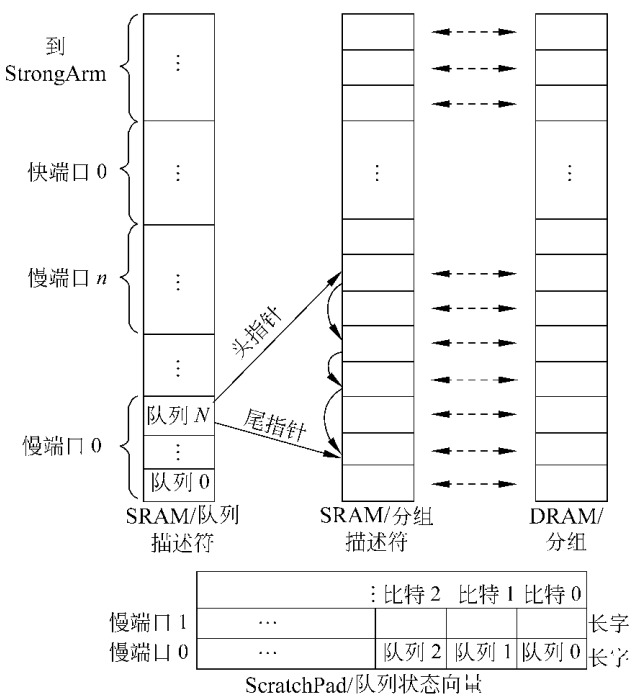



**图** 15. 2. 3　**队列结构**

- SRAM 分组描述符：存放分组的基本信息（比如分组长度）和分组调度算法参数（比如分组到达时间戳）。未分配（空闲）的分组描述符构成堆栈结构，已分配的分组描述符构成链表结构，形成逻辑队列。
- DRAM 分组缓冲区：把 DRAM 的某个地址连续区域分成固定大小（2K）的页，每一页存放一个分组（整个分组，确切地说，是以太网帧），并静态地对应 SRAM 中的一个分组描述符。
- SRAM 队列描述符：存放每个队列的基本信息（比如队列长度）、缓冲管理算法参数（比如分组丢失率）和分组调度算法参数（比如队列的权值）。
- SRAM 队列结构：一系列队列描述符组成数组，构成系统完整的队列结构。
- Scratchpad 队列状态向量：存放队列的状态（是否为空，每一比特位对应一个队列）。这一结构主要是为了提高性能。

每端口所对应的逻辑队列数量可以配置。

## 15.2.2　系统处理基本流程

系统把分组（确切地说是以太网帧）分为 64 字节长的单元（最后一个单元可能会小于 64 字节），称为 MAC-Packet，简称 MP。第一个 MP（SOP）、最后一个 MP（EOP）和中间的 MP（MOP）分别需要不同的处理。

### 15.2.2.1　输入处理

在输入端，接收线程不停地检查 MAC 端口的状态。当有分组/MP 到达时：①如果是 MOP，则简单地将其存入预先分配的 DRAM 缓冲区中。②如果是 SOP，则进行分组头校验以及其他相应处理，并进行路由表查找和分组分类。通过路由表查找，确定分组转发的物理端口号。需要本地处理的分组被重定向到 StrongArm 中进行处理；而需要转发到下一节点的分组通过分类，确定将要进入的队列编号和丢弃优先级。以上数据保存在全局寄存器中。③如果是 EOP，则根据缓冲管理算法和上述分类结果进行分组丢弃判断。如果分组不需要被丢弃，则被加入队列。

分组分类：一般来讲，分组头中有 5 个字段域可供分类：IP 地址（源、目标地址）、协议类型和传输层端口号（源、目的端口）[73]。可以根据需要从一维（一个字段域）到五维分类。当然，维数越多，处理速度越慢。

缓冲管理：缓冲管理算法有三种配置：Tail-drop，SPBS（static partial buffer sharing）和 DPBS。Tail-drop 是最简单的算法，即当队列满的时候不加选择地丢掉所有新到达的分组。而 SPBS 则对分组加以区分：给不同的数据流汇聚（类）分配不同的丢弃优先级，并为每个类维护一个丢弃阈值（队列长度阈值）。对于某个类，只有在当前的队列长度大于其丢弃阈值时，属于该类的新到达的分组才被丢弃。丢弃阈值是静态配置的，在系统运行过程中保持不变。而 DPBS 算法则通过动态调整丢弃阈值的大小，可以使得各个类的分组丢失率保持较为恒定的比率，实现更为精确的丢弃控制。

DPBS 算法：为每一个服务类 Class-i 维护一个计数器 $C_i$，记录在某时间段内该服务类的分组丢弃个数。当该计数器到达某个界值 $K_i$ 时，则把该服务类的丢弃阈值 $T_i$ 增加某个量

值 $TH_i$,同时把次优先级服务类 Class-(i - 1)的丢弃阈值 $T_{i-1}$减小某个量值 $TL_i$。当然,对于最低优先级服务类 Class-1 和最高优先级服务类 Class-n,只需要分别调整 $T_1$和 $T_{n-1}$。

缓冲管理算法的详细内容请阅读本书第 9 章。

#### 15.2.2.2 输出处理

在输出端,发送线程首先计算将要服务的物理端口号(因为两个物理端口竞争一个发送线程),然后检查端口的队列状态。

(1) 如果该端口无数据要发送,则跳过(skip)该端口继续检查下一端口。

(2) 如果该端口有数据要发送,则检查上一个分组的最后一个 MP(EOP)是否已发送。①如果没有,则继续发送该分组的下一个 MP;②如果已发送,则根据分组调度算法从多个队列中选择一个分组,发送该分组的 SOP。

分组调度:有多种分组调度算法可以配置:PQ(priority queueing),RR(round robin),WRR(weighted round robin),DRR(deficit round robin)。PQ 算法给不同的队列分配不同的调度优先级,并且每次调度具有最高优先级的队列直到该队列为空。这样保证重要的分组得到最好的服务。RR 算法只是简单地循环调度每个队列,以获得某种程度的公平性。WRR 算法在 RR 算法的基础上给队列分配不同的权值,该权值对应于每次调度发送的分组数。这样,在保持某种程度公平性的基础上,使得权值比较大的队列能够得到更好的服务。DRR 算法则考虑了 IP 网络中分组长度变化这一特点,以字节为单位为每个队列分配一个带宽配额,该配额的比例对应于队列服务速率的比例。这样,DRR 算法就能获得比 WRR 更好的公平性和更精确的控制。

分组调度算法的详细介绍请阅读本书第 10 章。

### 15.2.3 几个设计问题

基于网络处理器的队列管理模块设计的目标之一是在保持灵活性的同时能够获得较高的系统性能。由于网络处理器本身硬件级并行处理的技术特点和本系统采用的输出排队的策略,使得系统同步、系统资源管理、线程的任务分配以及其他涉及分组发送的问题对系统性能具有非常大的影响。另外,结合网络处理器的硬件特点,总结队列管理的基本操作,对于进行系统优化也有非常重要的意义。因此,本节着重讨论了一些相关的关键技术问题。

#### 15.2.3.1 系统同步

1. 系统同步问题存在的原因

(1) 系统多线程:IXP1200 具有先进的硬件多线程系统,不同的线程可以负责不同的任务处理,通过硬件级并行化来提高系统吞吐率,提高系统性能。然而,多个线程之间并不总是相互独立的,有时候具有某些相互制约的关系。主要体现在服务的时序关系和数据一致性两个方面。

(2) 服务的时序关系:有些服务的执行需要一定的条件,比如另外一个任务的处理

结果作为本任务的输入。有些服务的执行要求具有一定的先后关系，比如同一个分组的多个 MP 的发送必须保持严格的先后关系。

（3）数据一致性：数据一致性主要是由于资源共享引起的。当多个具有读取/修改权限的线程共享同一个数据结构时，必须保证多个线程对数据的修改不造成冲突。这需要同步控制，典型的是通过加锁或信号量机制。

2. 系统同步问题的影响

多线程设计的目的是任务处理的并行化，从而获得很高的系统性能。而同步控制恰恰相反，目的是通过限制某些任务的执行，以保持服务的时序关系和数据一致性。两者具有一定的矛盾性。因此，解决同步问题，需要兼顾系统运行的正确性和性能。即在保证正确的时序关系和数据一致性的基础上，提高系统性能。

3. 解决方案

（1）为保证系统处理的正确性，需要设计严格的同步控制。由于篇幅所限，本文主要讨论缓冲管理和分组调度模块之间的同步控制解决办法。

（2）应尽量避免不必要的同步关系，以提高系统吞吐率。发送 MP 时，对于 TFIFO 的软件管理模式就体现了这一点，我们将在 15.2.3.3 节中详细讨论。

缓冲管理和分组调度之间进行同步控制是为了解决由于资源共享而引起的数据一致性问题。所涉及的共享资源有三个：分组描述符、队列描述符和队列状态向量。

1）当分组入队列或出队列时，需要修改队列描述符和分组描述符中的某些信息。因此必须保证当缓冲管理模块或分组调度模块对描述符进行修改时，该数据不能被其他模块读取。队列描述符和分组描述符保存在 SRAM 中，而系统提供了 SRAM 的硬件加锁机制，因此，可以利用硬件加锁来进行同步控制，当某模块需要修改描述符时，在读取数据的同时将该描述符加锁（read_lock 指令）；而当写完数据后，将该描述符解锁（write_unlock 或 unlock 指令）。在加锁和开锁期间，任何其他模块不能再读取该描述符。如果有读请求到达，则被阻塞。

2）队列状态向量用来指示某队列是否为空。当某空闲队列有分组到达时，将向量中该队列对应的比特位置 1；当某队列最后一个分组被调度时，将向量中该队列对应的比特位清 0。分组调度模块根据队列状态向量来判断队列是否为空。因此必须保证向量的某比特位为 1 时，对应队列中确实存在着有效分组。队列状态向量也是被缓冲管理和分组调度模块共享，并存在着比 1）更复杂的数据一致性问题。队列状态向量被保存在 ScratchPad 中（为了保证较快的访问速度），而 ScratchPad 没有提供硬件加锁机制，因此只能利用软件的方法进行控制，比如通过调整指令代码的执行位置，把对状态向量读写的操作安排在 SRAM 的加锁期间，当然这样会在某种程度上降低系统效率。

综上所述，确定缓冲管理模块和分组调度模块对分组描述符、队列描述符和队列状态向量的操作规则如下：

（1）在修改队列描述符之前（读取数据时），对该描述符加锁；修改结束后（写入数据

时)解锁。为提高效率,上锁的时间要尽量短,即加锁和解锁之间安排尽量少的指令。

(2) 在缓冲管理模块中:当某空闲队列有分组到达时,分组描述符和队列描述符的更新(写)操作要早于队列状态向量的更新(写)操作,以保证状态向量中对应比特位为 1 时,队列描述符中确实存在有效的分组的相关数据。

(3) 在分组调度模块中:当某个队列最后一个分组被调度之后,状态向量要尽快更新(相应比特位清零)。

(4) 根据以上三点,在缓冲管理模块中,队列状态向量的更新应该在队列描述符解锁操作之后,在分组调度模块中,状态向量的更新应该在队列描述符解锁操作之前。

(5) 由于分组描述符的读写总是在已经读取队列描述符并计算出分组描述符地址之后,因此不必专门对分组描述符作同步控制。

### 15.2.3.2 线程分配

IXP1200 的每个微引擎支持 4 个硬件线程,通过线程之间零开销的切换可以在某种程度上隐藏内存访问的时间延迟,以并行技术提高系统的性能。

在发送端,由于用一个微引擎负责 8 个 10/100Mbps 端口的分组发送,因此一个线程需要服务多个端口。在传统的软件开发模式(IXP1200 参考设计)中,专门分配了一个线程作为调度线程,即决定另外三个发送线程分别负责哪一个端口的分组发送。

然而在本质上,调度线程的开销属于内部管理开销,对分组的转发没有直接意义,因此应使这种开销最小化。经过分析发现,专门用一个线程作为调度线程是不必要的,在某种程度上浪费了处理器资源。比如从调度线程的调度结果来看,是使得每个发送线程所服务的物理端口号按照加 3 再对 8 求模的规律进行变化(当然,也可以有其他的调度策略)。实际上,这种调度策略的执行可以交给发送线程自己,即 4 个线程都作为发送线程,并且服务的物理端口号按照加 4 再对 8 求模的规律进行变化。这样就提高了处理器资源的利用率,进而提高了系统性能。图 15.2.4(a)表示线程 T0 作为调度线程而其余三个线程作为发送线程模式下的端口服务序列,图 15.2.4(b)表示调整结构后的服务序列。可以看出,在第二种情况下,各个端口得到更快的服务。

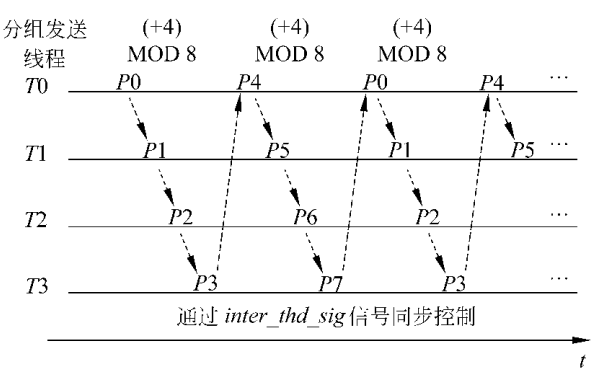



图 15.2.4(a) T0 负责线程调度

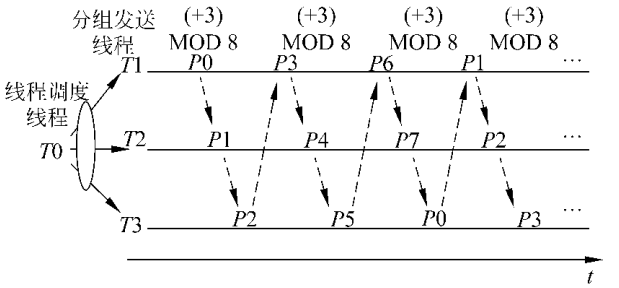



图 15.2.4(b)　4 个线程全作为发送进程

### 15.2.3.3　发送缓冲 TFIFO 的管理

在发送 MP 时,对于 TFIFO 的软件管理模式也涉及到同步问题,即系统正确性与性能之间的折中问题。传统的设计模式中用线程间的同步控制来保证各个线程的时序关系,而经过分析发现,这种同步控制是没有实际意义的。在我们的新系统中,通过解除这种同步控制,既保证了发送的正确性,又提高了发送速度。

IXP1200 通过 FBI 中的状态寄存器 XMIT_PTR 来指示当前 TFIFO 中哪一个 element 的数据将被发送到 IX 总线,并且采用轮循的服务方式。一般来讲,一个 TFIFO element 必须满足三个条件才能被服务:控制/状态寄存器已写;有效标记位(valid flag)已置位;XMIT_PTR 已指向该 element。如果有效标记位被置 1,则根据控制寄存器中的信息把 element 中的数据发送到 IX 总线上(控制寄存器指示该 element 中有数据要发送),同时 XMIT_PTR 加 1,去处理下一个 element;或者只是简单地跳过当前 element(控制寄存器指示该 element 中无数据要发送),同时 XMIT_PTR 加 1,去处理下一个 element。当 XMIT_PTR 等于 15 时,则返回到 0 继续下一次循环。

提高数据发送速度的关键在于提高 XMIT_PTR 的轮转速度。

在传统的软件设计模式中,系统通过线程间通信信号 inter_thd_sig 来进行同步控制,使得发送线程满足某种时序关系,如图 15.2.4(b)所示。这在很大程度上限制了 XMIT_PTR 的轮转速度。比如,当某个 element 准备好数据并且写入控制/状态寄存器中的信息以后,服务该 element 的线程必须被阻塞(blocked)以等待前一个线程的 inter_thd_sig 信号。只有当接收到这一信号并且 XMIT_PTR 到达时,有效标记位才能被置 1,从而该 element 被继续服务,同时发送一个 inter_thd_sig 信号给下一个线程。这样,当某个线程等待 inter_thd_sig 信号而被阻塞时,其他线程有可能由于访问存储器也被阻塞,从而使得处理器处于空闲状态,利用率下降。经过统计发现,处理器(负责发送的微引擎)存在 15%~20%的空闲状态。

实际上,对于本系统的 10/100Mbps 端口应用来讲,一个端口只对应一个 TFIFO element,各个物理端口相互独立,因而相应的 TFIFO element 之间不存在服务时序关系。这样,发送线程之间的同步控制是没有必要的。这与千兆口不同,由于一个千兆口对应着多个 TFIFO element,为保证 MP 发送的先后顺序,这些 element 需要一个服务的时序控制,

因而相应的发送线程需要同步控制。

在解除发送线程间不必要的同步关系之后,4 个发送线程的服务序列如图 15.2.5 所示,可见,端口被服务的速度有很大提高。

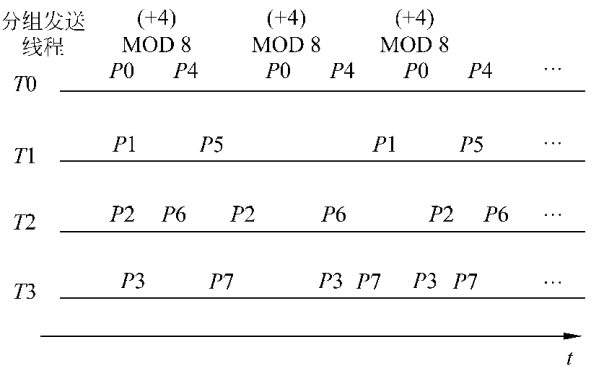



图 15.2.5 没有同步控制的线程分配

然而,由于 XMIT_PTR 对 TFIFO 轮循服务的硬件管理模式的限制,实际上,各个发送线程之间仍然隐藏着某种服务的时序问题。这是软件的方法所不能解决的。

当 XMIT_PTR 指向某个 element 时,由于该 element 数据未准备好,使得 valid flag 未置 1,则 XMIT_PTR 停下来等待,直到 valid flag 被置 1。在这种情况下,即使后面的某个 element 已准备好数据并且 valid flag 已经置 1,但由于 XMIT_PTR 未指向该 element,数据也不能被发送,服务该 element 的发送线程被阻塞等待。

一种可行的硬件解决方案是用一个 FIFO 队列代替 XMIT_PTR 的轮循服务方式:当某个 element 准备好数据及控制信息后,其编号被写入 FIFO 队列。系统不断地从该 FIFO 队列中读取 element(编号)进行服务。

#### 15.2.3.4 队列管理的几个基本操作

乘/除法运算:由于网络处理器是针对网络信息处理而优化了的专用处理器,在设计上采用了一套专用的精简指令集,没有提供乘/除法指令。如果需要,乘/除法运算只能由软件来完成,即由多条整数加/减/移位指令来软件模拟,但一般需要花费几十个指令周期,其性能是非常低的。而实际上,服务质量控制本身是一种计算性较强的服务,乘/除法运算对于缓冲管理和分组调度而言,是非常重要的基本操作。比如 RED[34] 需要用乘法计算平均队列长度,用除法计算分组丢失概率。在一类分组调度算法 PFQ(packet fair queueing)中,除法用来计算每个分组的服务时间。WTP(waiting time priority)算法用除法计算调度优先级。因此,在网络处理器中提供硬件乘/除法的功能单元,对于提高队列管理子系统的性能,是非常重要的。

查找/排序:查找/排序,更确切地说,即从数据列表中查找最大或最小值,是分组调度的另一个重要的基本操作。基于优先级的调度算法是优先级的排序,基于轮循的调度算法是逻辑队列编号的排序。这两者的排序关系一般是固定的,因此复杂度为 O(1)。而另外许多算法都具有 O(logN)的复杂度,其中 N 代表队列数量。PFQ 算法是服务时间

的排序(SFF：最小虚拟完成时间；SSF：最小虚拟开始时间；或 SEFF：最小合法虚拟完成时间)。EDF 及其变种是排队时延的排序。又如，URR 是关于紧急程度参数的排队，WTP 是关于分组等待时间优先级的排队。一般来讲，数据列表存储在片外存储器(比如 SRAM)，或片内存储器(比如 ScratchPad，如果其容量足够大的话)中。但不管怎样，都需要大量的内存访问操作。对于大多数算法来讲，O(logN)的复杂度会使内存访问成为一个性能瓶颈。因此，研究在网络处理器体系结构上的快速排序方法，或提供某种硬件支持，对于提高分组调度的执行速度是非常重要的。

## 15.2.4　性能评价

队列管理子系统的实现性能应该从两个层次来评价：局部性能和系统性能。

局部性能是指有关队列操作的性能，体现为缓冲管理和分组调度算法的控制结果，比如不同队列/服务类的分组丢失率、排队时延、带宽、队列长度等；系统性能是指整个路由器系统的性能，体现为系统中所有服务模块的综合处理性能、算法实现的复杂度等，可衡量指标为系统吞吐率等。我们通过 IXP1200 SDK 1.2 的模拟器进行了性能分析。

模拟实验基本配置：8 个全双工 100Mbps 速率端口，端口 MAC 接收和发送缓存均为 256 字节，每端口对应 8 个逻辑队列，队列容量为 32 个分组。

### 15.2.4.1　局部性能

局部性能测试配置：①数据流配置为：8 个端口接收到的数据只向 4 个端口轮流发送(本端口除外)。②数据源分组长度为 64～1500 字节随机。③PQ 算法：队列 1 优先级最低，队列 8 优先级最高，其余队列优先级随着编号的增大逐次提高；WRR 算法：各个队列权值与队列编号相同，即队列 1 权值为 1，队列 2 权值为 2，依此类推；DRR 算法：队列 1 带宽配额为 1540 字节，其余队列随着编号的增加其带宽配额逐步增加 220，队列 8 配额为 3080；SPBS 算法：把同一队列中的数据流分成两个丢弃优先级 1(低)和 2(高)，丢弃阈值分别为 24 和 32。同时，用脚本文件控制分组的到达时间间隔，以在一定程度上模仿突发性；DPBS 算法：两个丢弃优先级的阈值变化量相等，即 $TH_1 = TL_2$，丢弃计数器 $C_i$ 的界值 $K_1$ 和 $K_2$ 分别为 15 和 10。④分类结果使得队列 1 中没有分组，一直为空。

局部性能测试结果如图 15.2.6 所示(纵坐标代表各个性能参数，图 15.2.6(a)～15.2.6(d)横坐标代表逻辑队列编号，图 15.2.6(e)横坐标代表时间采样点)。

局部性能测试结果分析：

- PQ 算法：最重要的分组能得到最好的服务，然而在高优先级队列源源不断地有分组到达时，低优先级的队列容易被“饿死”，即很长时间内得不到服务。因而公平性很差。
- RR 算法：循环地服务每个队列，但由于分组长度不固定，使得长分组队列可能比短分组队列得到更多的服务，因而其公平性特点受到很大限制。
- WRR 算法：WRR 可以比 RR 更容易控制带宽在不同队列的分配，但仍然存在由于分组长度变化而带来的公平性较差的弱点。

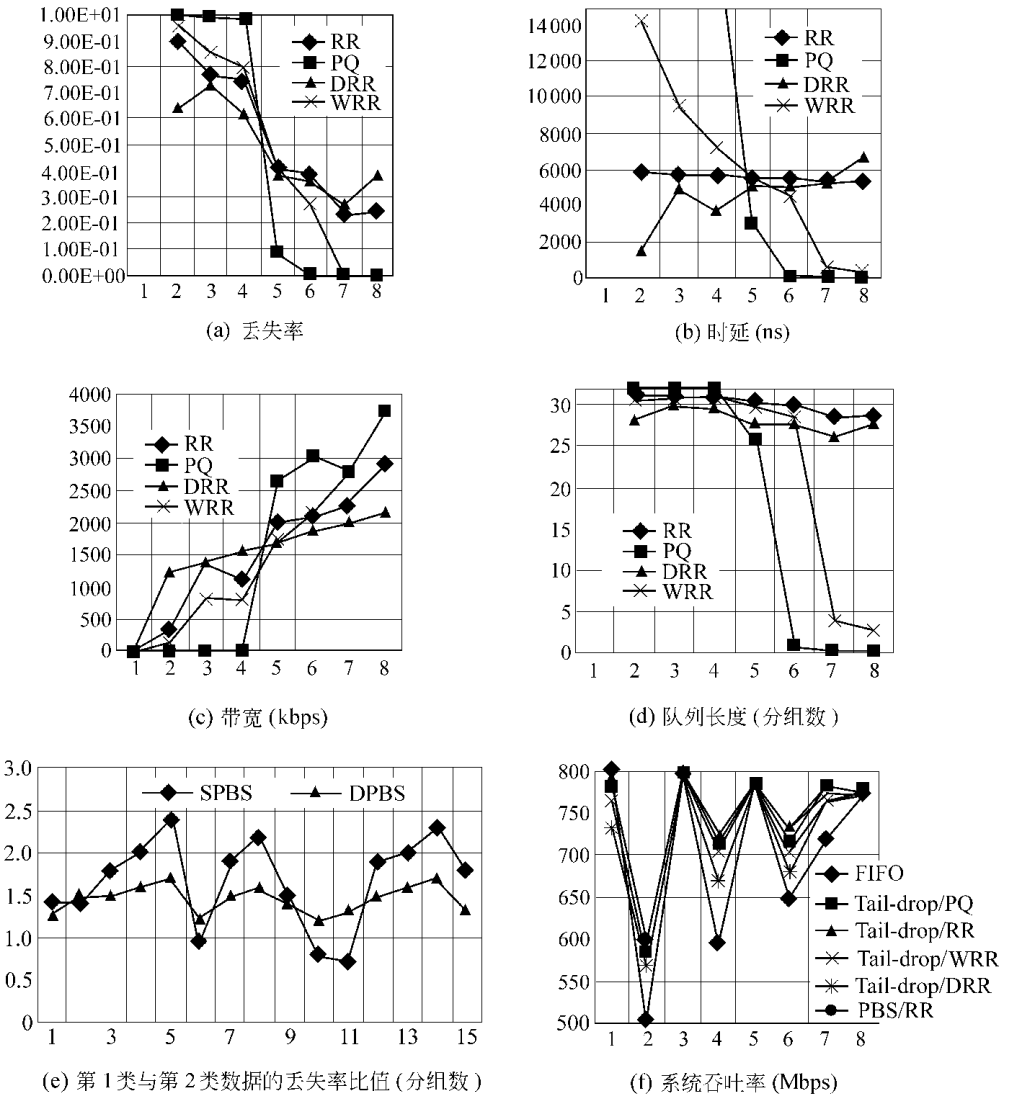



**图** 15.2.6

- DRR **算法：很好地解决了上述公平性问题，而且能实现比** WRR **算法更精细的队列间的服务速率分配。**
- SPBS/DPBS **算法：在** SPBS **算法中，丢弃阈值是静态配置的，在系统运行过程中保持不变，因此数据突发性使得丢失率的比值变化较为剧烈。而** DPBS **算法通过动态调整丢弃阈值的大小，则可以使各个类的分组丢失率保持较为恒定的比率，实现更为精确的丢弃控制。**

**以上结果是在输出端口为重负载情况下测量得到的，因而分组丢失率很高。而在轻负载情况下，某些队列的服务速率等于分组的到达速率，因而算法的控制结果几乎相同（丢失率为零，带宽相等）。限于篇幅，这里未列出其曲线图。**

#### 15.2.4.2　系统性能

系统性能测试配置：①数据流配置为：8 个端口接收到的数据向 8 个端口轮流发送（本端口除外）；②数据源分组长度配置有 9 种：固定长度 64 字节、固定长度 65 字节、固定长度 128 字节、固定长度 129 字节、固定长度 256 字节、64 ~ 128 随机长度、64 ~ 256 随机长度和 64 ~ 1500 随机长度；③算法组合有 6 种：FIFO（没有 QoS 控制和 15.2.3 节所讨论的优化），Tail-drop/PQ，Tail-drop/RR，Tail-drop/WRR、Tail-drop/DRR 和 SPBS/RR。

系统性能测试结果如图 15.2.6（f）所示（横坐标代表不同的数据源，纵坐标代表系统吞吐率，曲线为不同的算法配置）。

系统性能测试结果分析：

（1）由于系统采用 MAC-Packet（64 字节）的工作方式，即一次接收/发送一个 MAC-Packet，而分组最后一个 MAC-Packet 的有效数据可能会小于 64 字节，因此当数据源为 65，129，193，257 字节长度的分组时，系统吞吐率会有明显的下降（图中只画出了 64/65 和 128/129 的对比）。

（2）经过 15.2.3 节所述的优化后，系统吞吐率有很大提高。从图中可以看出，在某些数据源的配置下，系统吞吐率甚至比优化前 FIFO 队列管理模式下的还要高。

（3）分组调度算法的实现复杂度有如下关系：DRR/WRR 的复杂度高于 RR/PQ。而从图中可以看出，DRR/WRR 配置下的系统吞吐率低于 RR/PQ。因此，算法复杂度对系统性能的影响是不能被忽略的，这也是设计算法时需要考虑的一个重要问题。

综上，目前的队列管理子系统从结构和实现的算法上都只是单独方案，即缓冲管理和分组调度是分开的。实际上，缓冲管理和分组调度的综合方案能获得比单独方案更好的控制结果，并也已开始受到越来越多的研究者的重视。因此，有关队列管理系统的下一步工作将是设计队列管理的综合方案，并设计适合于综合方案的软件体系结构，在网络处理器上实现上述综合方案。

## 参考文献


1　Shimonishi H，Murase T. A network processor architecture for flexible QoS control in very high-speed line interfaces. In：Proceedings of the 2001 IEEE Workshop on High Performance Switching and Routing（HPSR 2001）. Dallas：IEEE Computer Society Press，2001. 402 ~ 406

2　Blake S，Black D，Carlson M，Davies E，Wang Z，Weiss W. An architecture for differentiated services. IETF RFC 2475，1998

3　Floyd S，Jacobson V. Random early detection gateways for congestion control. IEEE/ACM Transactions on Networking，1993，1（4）：397 ~ 413

4　Thiele L，Chakraborty S，Gries M，KÜhzli S. Design space exploration of network processor architectures. In：Proceedings of the 8th International Symposium on High Performance Computer Architecture. Cambridge，MA. 2002

5　Wolf T，Franklin M A，Spitznagel E. Design tradeoffs for embedded network processors. Technical Report，WUCS-00-24，St. Louis：Department of Computer Science，Washington University. 2000

6 Karlin S, Peterson L. VERA: An extensible router architecture. Computer Networks, 2002, 38(3): 277 ~ 293
7 Wolf T, Franklin M A. CommBench—A telecommunications benchmark for network processors. In: IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, 2000. 154 ~ 162
8 Hecht J. Wavelength division multiplexing. http://www.technologyreview.com/articles/hecht0399.asp, March/April 1999
9 Memik G, Mangione-Smith B, Hu W. NetBench: A benchmarking suite for network processors. In: Proceedings of the International Conference on Computer-Aided Design(ICCAD). San Jose, CA, 2001. 39 ~ 43
10 Shah N. Understanding network processors [MS Thesis]. Department of Electrical Engineering & Computer Sciences, University of California, Berkeley, 2001
11 Nie X, Gazsi L, Engel F, Fettweis G. A new network processor architecture for high-speed communications. In: Proceedings of the IEEE Workshop on Signal Processing Systems. IEEE Computer Society Press, 1999. 548 ~ 557
12 McAuley A, Francis P. Fast routing table lookup using CAMs. In: Proceedings of the INFOCOM'93, IEEE Computer Society Press, 1993. 3: 1382 ~ 1391
13 Liu H. A trace driven study of packet level parallelism. In: Proceedings of the International Conference on Communications(ICC). New York, NY, 2002. 2191 ~ 2195
14 IBM Corp. Rainier Network Processor, 2000. http://www.ibm.com/
15 Intel Corp. Intel IXP1200 Network Processor, 2002. http://developer.intel.com/design/network/products/npfamily/ixp1200.htm
16 Wolf T, Turner J S. Design issues for high-performance active routers. IEEE Journal on Selected Areas in Communications, 2001, 19: 404 ~ 409
17 Kounavis M E, Campbell A T, Chou S, Modoux F, Vicente J, Zhuang H. The genesis kernel: A programming system for spawning network architectures. IEEE Journal on Selected Areas in Communications, 2001, 19(3): 511 ~ 526
18 Wang J, Nahrstedt K. Parallel IP packet forwarding for tomorrows IP routers. In: Proceedings of the 2001 IEEE Workshop on High Performance Switching and Routing(HPSR 2001). Dallas, TX, IEEE Computer Society Press, 2001. 353 ~ 357
19 Katavenis M, Sidiropoulos S, Courcoubetis C. Weighted round-robin cell multiplexing in a general-purpose ATM switch chip. IEEE Journal on Selected Areas in Communication, 1991, 9(8): 1265 ~ 1279
20 Keutzer K. Enabling Fully programmable embedded system solutions. Presentation. Gigascale Silicon Research Center Annual Review. 1999
21 http://www.npforum.org. Network Processor Forum
22 http://www.networkprocessors.com. Network Processor Conference
23 Spalink T, Karlin S, Peterson L, Gottlieb Y. Building a robust software-based router using network processors. In: Proceedings of the 18th SOSP: ACM Press, 2001. 216 ~ 229
24 Campbell A T, Chou S T, Kounavis M E, Stachtos V D. NetBind: A binding tool for constructing data paths in network processor-based routers. In: Proceedings of the 5th IEEE Conference on Open Architectures and Network Programming(OPENARCH), IEEE Computer Society Press, 2002. 91 ~ 103
25 Thiele L, Chakraborty S, Gries M, Maxiaguine A, Greutert J. Embedded software in network processors models and algorithms. In: Proc 1st Workshop on Embedded Software. Lecture Notes in Computer

Science 2211, Lake Tahoe, CA, USA: Springer-Verlag, 2001. 416 ~ 434

26 Qie X, Bavier A, Peterson L, Karlin S. Scheduling computations on a software-based router. In: Proc SIGMETRICS. IEEE Computer Society Press, 2001. 13 ~ 24

27 Peterson L, Karlin S, Li K. OS support for general purpose routers. In: Proceedings of the 7th Workshop on Hot Topics in Operating Systems. IEEE Computer Society Press, 1999, 38 ~ 43

28 Lahiri K, Raghunathan A, Dey S. System level performance analysis for designing on-chip communication architectures. IEEE Trans on Computer Aided-Design of Integrated Circuits and Systems, 2001, 20(6): 768 ~ 783

29 Peterson L L, Davie B S. Computer Networks: A System Approach. 2nd ed. Morgan Kaufmann Publishers, Inc., 2000. 458

30 Causey J W, Kim H S. Comparison of buffer allocation schemes in ATM switched: Complete sharing, partial sharing and dedicated allocation. In: Proc Int l Conf Communications. IEEE Computer Society Press, 1994. 1164 ~ 1168

31 Jiang Y, Lin C, Wu J, Sun X. Integrated performance evaluation criteria for network traffic control. In: Proc IEEE Symp on Computers and Communications. IEEE Computer Society Press, 2001. 438 ~ 443

32 Shreedhar M, Varghese G. Efficient fair queueing using deficit round-robin. IEEE Trans on Networking, 1996, 4(3): 375 ~ 385

33 Lin C, Sheng L, Wu J, Xu M W. An integrative scheme of differentiated services. In: Proc 8th Int l Symp on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2000). IEEE Computer Society, 2000, 441 ~ 448

34 Yong J, Kim Y H. Performance analysis of a hybrid priority control scheme for input and output queueing ATM switches. In: Proc IEEE INFOCOM98. IEEE Computer Society Press, 1998, 3: 1470 ~ 1477

35 Giessler A, Haanle J, Konig A, Pade E. Free buffer allocation——An investigation by simulation. Computer Networks, 1978, 2: 191 ~ 204

36 Casavant T L, Kuhl J G. A taxonomy of scheduling in general purpose distributed computing systems. IEEE Trans on Software Engineering, 1988, 14(2): 141 ~ 154

37 Eager D L, Lazowska E D, Zahorjan J. Adaptive load sharing in homogenous distributed systems. IEEE Trans on Software Engineering, 1986, SE-12(5): 662 ~ 675

38 Barish G, Obraczka K. World wide web caching: Trends and techniques. IEEE Communications Magazine, 2000, 38(5): 178 ~ 184

39 Goldszmidt G, Hunt G. Scaling Internet services by dynamic allocation of connections. In: Proc 6th IFIP/IEEE Int l Symp on Integrated Network Management. IEEE Computer Society Press, 1999. 24 ~ 28, 171 ~ 184

40. Koufopavlou O G, Tantawy A N, Zitterbart M. Analysis of TCP/IP for high performance parallel implementations. In: Proc 17th IEEE Conf on Local Computer Networks: Minneapolis, IEEE Computer Society Press, 1992. 576 ~ 585

41 Kencl L, Le Boudec J-Y. Adaptive load sharing for network processors. In: Proc IEEE INFOCOM 2002. NY USA, IEEE Computer Society Press, 2002. 545 ~ 554

42 Thaler D G, Ravishankar C V. Using name-based mappings to increase hit rates. IEEE/ACM Trans on Networking, 1998, 6(1): 1 ~ 14

43 Pappu P, Wolf T. Scheduling processing resources in programmable routers. In: Proc IEEE INFOCOM 2002. IEEE Computer Society Press, 2002. 104 ~ 112

44 Zhang H. Service disciplines for guaranteed performance service in packet switching networks. Proc IEEE, 1995, 83(10): 1374 ~ 1396

45 Parekh A K, Gallager R G. A generalized processor sharing approach to flow control: The single node case. In: Proc IEEE INFOCOM92. IEEE Computer Society Press, 1992. 915 ~ 924

46 Bennett J, Zhang H. Worst case fair weighted fair queuing. In: Proc IEEE INFOCOM95. IEEE Computer Society Press, 1995. 120 ~ 128

47 Goyal P, Vin H M, Cheng H C. Start-time fair queuing: A scheduling algorithm for integrated services packet switching networks. In: Proc ACM SIGCOMM. ACM Press, 1996. 157 ~ 168

48 Bennett J C R, Zhang H. Hierarchial packet fair queuing algorithms. In: Proc ACM SIGCOMM. ACM Press, 1996. 43 ~ 56

49 Golestani S J. A self clocked fair queuing scheme for broadband applications. In: Proceedings of the IEEE INFOCOM94. IEEE Computer Society Press, 1994. 636 ~ 646

50 Wolf T, Franklin M. Locality-aware predictive scheduling of network processors. In: Proc IEEE Int l Symp on Performance Analysis of Systems and Software(ISPASS). IEEE Computer Society Press, 2001. 152 ~ 159

51 Matsumoto C. Technical trial-by-fire awaits NPUs. EE Times. http://www. eetimes. com/story/OEG20011221S0027

52 Nemirovsky A. Towards characterizing network processors: Needs and challenges. XStream Logic, Whitepaper, 2000

53 Tsai M, Kulkarni C, Sauer C, Shah N, Keutzer K. A benchmarking methodology for network processors. In: Workshop on Network Processors. Cambridge, MA: Morgan Kaufmann Publishers, 2002

54 Standard Performance Evaluation Corporation(SPEC). http://www. spec. org/

55 Weicker R. Dhrystone benchmark: Rationale for version 2 and measurement rules. SIGPLAN Notices, 1988, 23(8): 1 ~ 2

56 Lee C, Potkonjak M, Mangione-Smith W. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In: Proc Int l Symp on Microarchitecture. IEEE Micro-30, IEEE Computer Society Press, 1997, 330 ~ 335

57 Embedded Microprocessor Benchmark Consortium(EEMBC). http://www. eembc. org/

58 Audenaert S, Chandra P. (NPF Benchmarking Working Group co-chairs), Network processors 59. Benchmark Framework. NPF Benchmarking Workgroup, http://www. npforum. org/

59 Burger D, Austin T. The SimpleScalar tool set. Version 2. 0. Technical Report CS-TR-97-1342, University of Wisconsin, 1997.

60 Crowley P, Baer J-L. A modeling framework for network processor systems. In: Proc 8th Int l Symp on High Performance Computer Architecture. Cambridge, MA, 2002

61 Mathew J. IBM ropes in partners for network processors. Electronic News Online. 2000. http://www. e-insite. net/electronicnews /index. asp?layout = article&articleId = CA49443

62 Matthew J. Network processor companies face the same tough issues. Electronic News Online. 2000. http://www. e-insite. net /electronicnews/index. asp?layout = article&articleId = CA49900

63 Matsumoto C. CloudShield pushes net processors to next performance level. EE Times. 2001. http://www. eetimes. com/story /OEG20010625S0088

64 Austin T X, Wilmington M A. Motorola s C-Port network processor chosen as key technology in the hammer PacketSphere platform from empirix. 2001. http://apspg. motorola. com/press/022801/

cport. html

65　Atondo A. Fabless semiconductor start-up SiByte announces breakthrough MIPS64 microprocessor core—Delivers up to 1 GHz at less than 2. 5 Watts world s most power-efficient core in its performance class. Press Release, http ://www. sibyte. com/pressroom /docs/pr_20000612_sb1. html

66　van Eijndhoven J T J, Sijstermans F W, Vissers K A, Pol EJD, Tromp M J A, Struik P, Bloks R H J, van der Wolf R H J, Pimentel A D, Vranken H P E. TriMedia CPU64 Architecture. In :Proc Int l Conf on Computer Design(ICCD 1999). IEEE Computer Society Press, 1999. 586 ~ 592

67　Hekstra G J, La Hei G D, Bingley P, Sijstermans F W. TriMedia CPU64 design space exploration. In : Proc ICCD 1999, Int l Conf on Computer Design. IEEE Computer Society Press, 1999. 599 ~ 607

68　**谭章熹,林闯,任丰原,周文江. 网络处理器的分析与研究. 软件学报**, 2003, 14(2) :253 ~ 267

69　**林闯,周文江,李寅,郑波,田立勤. 基于** Intel **网络处理器的路由器队列管理:设计、实现与分析. 计算机学报**, 2003, 26(9) :1068 ~ 1077

70　Yang L, Dantu R, Anderson T. ForCES architectural framework. IETF Internet Draft, Sept 2002

71　Karlin S, Peterson L. VERA : An extensible router architecture. Computer Networks, 2002, 38(3) :277 ~ 293

72　**江勇,吴建平,徐恪. 高性能交换体系结构及其调度算法分析. 电子学报**, 2000, 28(11A) :105 ~ 109

73　**田立勤,林闯.** IP **报文分类技术的研究及其应用. 计算机研究与发展**, 2003, 40(6) :765 ~ 775

# 英汉对照术语表

A

access control **访问控制**
active queue management（AQM） **主动队列管理**
adaptation **自适应**
adaptive-random early detection **自适应随机早期检测**
adaptive virtual queue（AVQ） **自适应虚拟队列**
additive increase and multiplicative decrease （AIMD） **加性增加倍乘减小**
admission control **接纳控制**
aggregate **聚集**
anycast **任播**
application integrated specific circuit（ASIC） **特定用途集成电路**
application specific instruction processor（ASIP） **特定应用指令处理器**
assured service（AS） **确保服务**
asynchronous transfer mode（ATM） **异步传输模式**
ATM adaptation layer ATM **适配层**
authentication **鉴别，认证**
autonomous system（AS） **自治系统**
available bit rate（ABR） **可用位速率**
average call acceptance rate（ACAR） **平均呼叫接收率**
average call setup time（ACST） **平均呼叫建立时间**
average cost（AC） **平均代价**
average routing distance（ARD） **平均路由距离**

B

backbone **主干**
backoff **退避**
backward explicit congestion notification（BECN） **反向显式拥塞通告**
bandwidth broker（BB） **带宽中介服务器**
best-effort **尽力而为的**
border gateway protocol（BGP） **边界网关协议**
buffer management **缓冲管理**
burst tolerance **突发容忍量**

C

caching **缓存**
call admission control（CAC） **呼叫接纳控制**
cell delay variation（CDV） **信元延时方差**

cell transfer delay 信元传送延时
cell loss rate(CLR) 信元丢失率
center-based trees(CBT) 中心树
class-based queueing(CBQ) 基于类的排队
classifier 分类器
cluster 集群
combined input-output queued(CIOQ) 输入输出排队
common gateway interface(CGI) 公共网关接口
conditioner 调节器
congestion avoidance 拥塞避免
congestion control 拥塞控制
constant bit rate(CBR) 恒定位速率
Consultative Committee on International Telephone and Telegraph(CCITT) 国际电话电报咨询委员会
context-switching 上下文交换
controlled-load service 可控负载型服务
co-processors 协处理器
core stateless fair queueing(CSFQ) 核心无状态公平排队
core stateless jitter virtual clock(CJVC) 核心无状态抖动虚拟时钟
cost of service(CoS) 服务成本
credit-based flow control 基于信用的流控
cut-off 截止

D

deficit round robin(DRR) 逆差轮循
delay differentiation parameter(DDP) 延迟区分参数
delay fairness parameters(DFP) 延时公平参数
delta estimation 增量预测
differentiated services(DiffServ) 区分服务
differentiated services codepoint(DSCP) 区分服务标记
dispatcher 请求分配器
distance based priority(DBP) 基于距离的优先级
distance vector multicast routing protocol(DVMRP) 距离矢量组播路由协议
domain name server(DNS) 域名服务器
drop-tail 尾部丢弃
dynamic packet state(DPS) 动态分组状态
dynamic partial buffer sharing(DPBS) 动态部分缓冲共享
dynamic partition 动态划分
dynamic threshold 动态阈值

E

earliest effective deadline first(EEDF) 最早有效截止优先

earliest deadline first(EDF) **最早截止优先**
earliest due date(EDD) **最早到达时间**
early random drop(ERD) **早期随机丢弃**
enabling predicate **可实施谓词**
explicit congestion notification(ECN) **显式拥塞通告**
exponential estimation **指数预测**
extensible markup language(XML) **扩展标记语言**

F

fairness index **公平指数**
fair queueing(FQ) **公平队列**
fast recovery **快速恢复**
faults tolerance **容错性**
feedback flow control **反馈流控**
firing **实施**
firing probability **实施概率**
first come first served(FCFS) **先到先服务**
first-in-first-out(FIFO) **先入先出**
fixed capacity **固定容量**
flooding **洪泛法**
flow **流**
flow-based fair queueing(FBFQ) **基于流的公平排队**
fork **派生**
forward explicit congestion indication(FECI) **前向显式拥塞指示**
forward explicit congestion notification(FECN) **前向显式拥塞通告**
frame-based fair queueing(FFQ) **基于框架的公平排队**

G

generalized processor sharing(GPS) **广义处理器共享**
general purpose processor(GPP) **通用处理器**
genetic algorithm **遗传算法**
goodput **有效吞吐量**
guaranteed service **保证型服务**

H

hard real-time **硬实时**
Hash function **哈希函数**
hierarchical fair service curve(HFSC) **分级公平服务曲线**
hierarchical packet fair queueing(H-PFQ) **分级分组公平排队**
hierarchical round robin(HRR) **分级轮循**
historical date priority(HDP) **历史日期优先级**

hop-by-hop **逐跳**
host group model **主机组模型**
HTTP redirection HTTP **重定向**
hybrid proportional delay(HPD) **混合比例延迟**
hypertext markup language(HTML) **超文本标记语言**
hypertext transfer protocol(HTTP) **超文本传输协议**

I

identity **标志**
immediate transition **瞬时变迁**
input queued(IQ) **输入排队**
integrated services(IntServ) **综合服务**
integrated services model **综合服务模型**
interior gateway routing protocol(IGRP) **内部网关路由协议**
intermediate system- intermediate system(IS-IS) **中介系统到中介系统协议**
Internet Engineering Task Force(IETF) Internet **工程特别工作组**
internet service provider(ISP) **因特网服务供应商**
inter-process communication(IPC) **进程间通信**

J

joint of buffer management and scheduling(JoBS) **缓冲管理和调度联合**

L

latency rate server(LRS) **时延速率服务器**
leaky bucket algorithm **漏斗桶算法**
level of service(LoS) **服务层次**
listen socket **监听套接口**
link constraint **链路约束**
load balancing **负载均衡**
loss rate differentiation parameter(LDP) **丢失率区分参数**
loss rate fairness parameters(LFP) **丢失率公平参数**

M

macro-flow **宏流**
marking **标识**
Markov chain(MC) **马尔可夫链**
maximum burst size(MBS) **最大突发尺寸**
master process **主进程**
metering **计量**
metric **度量**
micro-flow **微流**

middleware **中间件**
minimum cell rate(MCR) **最小信元速率**
minimum laxity threshold(MLT) **最小松弛阈值**
multicast **点到多点通信,多播,组播**
multicast open shortest path first(MOSPF) **组播开放式最短路径优先**
multifield(MF) **多域的**
multiprocessor system **多处理器系统**
multiprotocol label switching(MPLS) **多协议标记交换**
multiserver multiqueue(MSMQ) **多服务器多队列**
multiserver multiqueue network(MSMQN) **多服务器多队列网络**

N

network access point(NAP) **网络访问点**
network address translation(NAT) **网络地址转换**
network intrusion detection systems(NIDS) **网络入侵检测系统**
network processor(NP) **网络处理器**
network simulator(NS) **网络仿真器**
nonreal-time variable bit rate(nrt-VBR) **非实时可变位速率**

O

Olympic service **奥林匹克服务**
open shortest path first(OSPF) **开放最短路径优先**
output queued(OQ) **输出排队**

P

packet **分组、报文**
packet classification **报文分类**
packet fair queueing(PFQ) **分组公平排队**
packet forwarding **分组转发**
packet rewriting **分组重写**
path constraint **路径约束**
partial buffer sharing(PBS) **部分缓冲共享**
peak cell rate(PCR) **峰值信元速率**
pending **挂起**
per-hop-behavior(PHB) **逐跳行为**
pipe **管道**
place **位置**
Poisson process **泊松过程**
policy control **策略控制**
preferred neighbor **优先邻居法**
premium service(PS) **奖赏服务**

priority queueing(PQ) 优先排队
process-per-connection 进程每连接
process-per-request 进程每请求
proportional average delay(PAD) 比例平均延迟
proportional delay differentiation(PDD) 比例时延区分
proportional integral(PI) 比例积分控制
proportional loss ratio(PLR) 成比例丢失率
protocol independent multicast dense mode(PIMDM) 协议无关的组播密集模式
proxy 代理

Q

quality of service(QoS) 服务质量
QoS routing QoS 路由
queue length threshold(QLT) 队列长度阈值
queueing network 排队网络

R

random early detection(RED) 随机早期检测
random early detection with In/Out bit(RIO) 带 In/Out 标记的随机早期检测
random exponential marking(REM) 随机指数标记
rate-based flow control 基于速率的流控
rate-controlled service(RCS) 速率可控的服务
rate spaced timestamp scheduler(RST) 速率分隔时戳调度器
ready queue 就绪队列
real-time variable bit rate(rt-VBR) 实时可变位速率
reference implementation framework 参考实现框架
relative differentiated services 相对区分服务
relaying front-end 前端中继
rendezvous point 集合点
request dispatching 请求分配
request specification(RSpec) 服务要求规范
residence time 驻留时间
resource container 资源容器
resource principals 资源代理
resource reservation protocol (RSVP) 资源预留协议
response time 响应时间
retransmission timer(RTO) 重发定时器
reverse path forwarding(RPF) 反向路径转发
root locus 根轨迹
round-robin(RR) 轮循
round trip time(RTT) 往返传输时间

routing 路由,选路
routing information protocol(RIP) 路由信息协议

S

scheduling 调度
self-clocked fair queueing(SCFQ) 自时钟公平排队
service curve-based earliest deadline(SCED) 基于服务曲线的最早截止
service fairness index(SFI) 服务公平指数
service layer specification (SLS) 服务层描述
service level agreement(SLA) 服务层协议
session 会话
session layer 会话层
session management 会话管理
shaping 整形
shared capacity 共享容量
shared tree(ST) 共享树
shortest connection first 最短连接优先
shortest path tree(SPT) 最短路径树
shortest remaining processing time(SRPT) 最短剩余处理时间
slave process 从进程
slow-start 慢启动
smallest eligible virtual finished time first(SEFF) 最小合格虚拟完成时间优先
smallest virtual finished time first(SFF) 最小虚拟完成时间优先
smallest virtual started time first(SSF) 最小虚拟开始时间优先
soft real-time 软实时
soft state 软状态
software performance units(SPU) 软件性能单元
source routing 源端路由
space complexity 空间复杂度
split horizon 水平分裂
starting potential fair queueing(SPFQ) 起始潜力公平排队
state-space explosion 状态空间爆炸
statistical multiplexing 统计多路复用
stochastic fair queueing(SFQ) 随机公平排队
stochastic high-level Petri nets(SHLPN) 随机高级 Petri 网
stream aggregate 流聚集
strict priority 绝对优先级
surplus round robin(SRR) 盈余轮循
sustained cell rate(SCR) 持续信元速率
system identification 系统辨识
system on chip(SoC) 片上系统

T

tangible marking **实存标识**
TCP handoff TCP **转接**
TCP splicing TCP **接合**
threshold **阈值**
throughput **吞吐量**
time complexity **时间复杂性**
timed transition **时间变迁**
token **标记**
token bucket algorithm **令牌桶算法**
traffic conditioning agreement(TCA) **传输调节协议**
traffic control **传输控制**
traffic shaping **传输整形**
traffic specification(TSpec) **流量规范**
transactions **事务**
transmission control protocol(TCP) **传输控制协议**
type of service(ToS) **服务类型**

U

unicast **点到点通信,单播**
unspecified bit rate(UBR) **未指定位速率**
urgency-based round robin(URR) **基于紧急的轮循**
user network interface(UNI) **用户网络接口**

V

variable bit rate(VBR) **可变位速率**
video on demand **视频点播**
virtual channel identifier(VCI) **虚通道标识**
virtual clock(VC) **虚拟时钟**
virtual connection(VC) **虚连接**
virtual path identifier(VPI) **虚通路标识**
virtual partition(VP) **虚拟划分**
virtual private network(VPN) **虚拟专用网**

W

waiting time priority(WTP) **等待时间优先级**
weighted fair queueing(WFQ) **加权公平排队**
weighted priority **加权优先级**
weighted random early detection(WRED) **加权随机早期检测**
weighted round-robin(WRR) **加权轮循队列**

worker process **工作进程**
worst-case fairness index(WFI) **最坏公平指数**
worst-case fair weighted fair queueing(WF2Q) **最坏公平加权公平队列**

Z

zone routing protocol(ZRP) **区域路由协议**