东华大学学位论文原创性声明

本人郑重声明：我恪守学术道德，崇尚严谨学风。所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已明确注明和引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品及成果的内容。论文为本人亲自撰写，我对所写的内容负责，并完全意识到本声明的法律结果由本人承担。

学位论文作者签名：王云涛

日期：2010 年 1 月 30 日

东华大学学位论文版权使用授权书

学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅或借阅。本人授权东华大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密 □，在 ____ 年解密后适用本版权书。

本学位论文属于

不保密 ☑。

学位论文作者签名：王云涛　　指导教师签名：

日期：2010年1月30日　　日期：2010年1月30日

网络拥塞控制算法研究

摘要

自从互联网出现以来，网络资源的缺乏和网络流量分配的不均使得网络拥塞问题一直困扰其发展，伴随着网络规模的扩大和业务量的激增，网络拥塞问题更加严重，已经成为阻碍互联网进一步发展的瓶颈。

目前，网络拥塞问题仍然不可避免。互联网发展要求我们必须采取有效的控制机制来降低网络拥塞发生的可能性，确保发生网络拥塞后能及时将网络恢复到正常状态，保证网络运行的稳定和畅通。网络拥塞控制算法是避免网络拥塞、改善网络性能、提高网络质量的主要手段，对网络拥塞控制算法的研究具有重要的理论意义和应用价值。

本文分别从 TCP 端和网络端两部分对网络拥塞控制算法进行了较深入的研究，并提出了改进算法。在基于 TCP 端的网络拥塞控制算法中，主要针对 TCP Vegas 算法进行了详细分析，从网络非对称性和算法兼容性两个方面对 TCP Vegas 算法进行了改进；在基于网络端的拥塞控制算法中，主要针对队列管理算法和队列调度算法进行了详细分析。论文的主要内容和创新点如下：

(1)基于 TCP Vegas 网络拥塞控制算法兼容性问题的研究

TCP Vegas 算法是一种基于测量技术的拥塞控制算法，相比传统的拥塞控制算法而言，TCP Vegas 算法在“慢启动”、“拥塞避免”和“快速重传”三个方面都做了改进。特别是在拥塞避免机制上通过比较实际吞吐量和期望吞吐量来调整拥塞窗口大小，将被动的拥塞避免机制变为主动预防的拥塞避免机制，极大地提高了算法的适用性。TCP Reno 算法是目前网络中的主流算法，它采用了被动的拥塞避免机制，通过持续增加自己的拥塞窗口，直到网络过载来保证有效利用网络资源；而 TCP Vegas 算法采用了主动的拥塞避免机制，其目标是将网络带宽维持在一个稳定的水平，所以它不会持续扩展自己的拥塞窗口。在实际应用中，TCP Reno 算法将会窃取 TCP Vegas 算法的带宽，导致 TCP Vegas 算法机制不能起到实际作用。为解决这个问题，本文对 TCP Vegas 算法设置的两个参数 α 和 β 进行了分析，将其由固定值变为动态变动数值，此外将拥塞窗口的指数级增长变为线性增长，仿真结果表明改进算法提高了 TCP Vegas 算法的带宽竞争能力和数据流量的稳定性。

(2)基于 TCP Vegas 网络拥塞控制算法网络非对称性问题的研究

TCP Vegas 算法是一种基于时间的算法，因此往返时延(RTT)的准确性至关重要，粗略测量的 RTT 会导致对拥塞窗口的粗略调整。目前 TCP Vegas 算法只是考虑了在数据发送方向上发生网络拥塞时采取措施,但是若拥塞发生在回路方向，也即 ACK 方向，TCP Vegas 的算法机制同样会采取拥塞避免措施，这样会引起对实际吞吐量的过低估计，导致拥塞窗口不必要的减小。实际应用中我们要求能够辨认网络拥塞是发生在哪个方向上，进而采取正确的措施保证网络的稳定性。为解决这个问题，本文分析了 TCP Vegas 算法拥塞避免机制,得到影响 TCP Vegas 算法 RTT 最大的因素是反向排队时间,为此可以在计算时除去该部分对 RTT 的影响。仿真结果表明，改进算法消除了 TCP Vegas 算法的网络非对称性问题。

(3)基于 TCP Vegas 算法中的反向链路拥塞和兼容性问题的融合性研究

我们分别针对 TCP Vegas 算法的兼容性问题和网络非对称问题提出自己的改进算法，仿真试验也分别证明了改进算法的有效性。在实际应用中我们都希望找到一种尽可能“完美”的算法，能够解决 TCP Vegas 算法中的各种问题，以适应目前网络的需要。本文综合分析了针对 TCP Vegas 算法中反向链路拥塞问题的改进算法，以及兼容性问题的改进算法，将它们融合在一起，提出了 TCP NewVegas 算法。仿真结果表明，该算法可以在和 TCP Reno 算法并用时保持较高的带宽竞争能力，也可消除反向链路的网络拥塞对算法机制的影响。

(4)基于自适应虚拟队列算法的区分服务问题的研究

自适应虚拟队列（AVQ）算法是根据系统负载情况来判断网络拥塞的，其算法机制是维持一个容量小于实际链路容量的虚拟队列。和传统主动队列管理算法相比，该算法解决了死锁问题，保持了业务流的公平性，可以减小排队延迟。但该算法不能提供区分服务，这对于目前网络业务种类不断增加，网络环境日益复杂的发展趋势来说是个很大的缺陷。为解决这个问题，本文首先分析了动态阈值算法和队列长度阈值算法，综合了这两种算法的思想用于 AVQ 算法。具体来讲是利用动态阈值算法的思想对 AVQ 算法的队列管理进行改进,利用队列长度阈值算法的思想对 AVQ 算法的队列调度进行改进,通过对不同的业务设置不同的优先级,根据不同的优先级给予不同的服务等级,从而达到保证区分服务来满足实时性业务。

关键词：网络拥塞控制，TCP Vegas 算法，队列管理，队列调度，AVQ 算法

RESEARCH OF THE NETWORK CONGESTION CONTROL ALGORITHM

ABSTRACT

Since the Internet has appeared, the development of Internet has been encumbered with the congestion problem caused by the lack of the network resources and the unbalance distribution of the network flows. The network congestion becomes more and more serious because of the increasingly expansion of Internet scale and the rapid growth of traffic flow.Congestion control has become the bottleneck witch hinders the Internet further develops.

Network congestion is still inevitable in the present internet. The development of the internet requests us to adopt the effective control mechanism to reduce the possibility of the network congestion occurs as far as possible,attempt to make the network resume its normal work promptly even if the network congestion has happened,ensure the network stability and unimpeded. The network congestion control algorithm is the main way to avoid the network congestion, reform network performance and improve the network quality.The investigation on the network congestion control is important not only in the theory but also in the application.

This work makes an intensive study of the network congestion control algorithm. We mainly carry on the multianalysis aimed at the TCP Vegas algorithm which belongs to the network congestion control algorithm based on the TCP port. We have made an improvement on the TCP Vegas algorithm in two aspects including the network asymmetry and the algorithm compatibility;

Then we mainly research the queue management algorithm and the queue dispatch algorithm, witch belongs to the network congestion control algorithm based on the network port. the main research contents and innovative points are the follows:

(1)Research of the algorithm compatible question based on TCP Vegas network congestion control algorithm

TCP Vegas algorithm is one kind of network congestion control algorithms which is based on the measuring technology, TCP Vegas algorithm has made improvement in "slow start", "congestion avoided" and "fast re-transmit" three aspects compared with thetraditional congestion

control algorithms.Specially it adjusts the congestion window size through the comparison between the actual throughput and the expectated throughput in the congestion avoided mechanism,which change the passive congestion avoided mechanism to the initiative prevention congestion avoided mechanism,thus enhanced the algorithm serviceability enormously.TCP Reno algorithm is the mainstream algorithm in the present network,it used the passive congestion avoided mechanism , through increase its congestion window continuously,until the network overload to guarantee the effective use network resource to avoid the mechanism; But TCP Vegas algorithm used the initiative congestion avoided mechanism,its goal is maintains the network band width at a stable level, therefore it will not continue expansion its congestion window. In the practical application, In the practical application,TCP Reno algorithm will steal the band width from TCP Vegas algorithm, which will cause TCP Vegas algorithm not to be able to play the actual role. In order to solve this problem, this article analysis the two parameters Alpha and Beta established by TCP Vegas,changes the fixed value to dynamic value,in addition changes the congestion window's exponential order growth linearity growth,the simulation result indicated that the improvement algorithm enhanced TCP Vegas algorithm's band width competitive ability and band width throughput stability.

(2)Research of the network asymmetrical question based on TCP Vegas network congestion control algorithm

TCP Vegas algorithm is a time-based algorithms, therefore the round-trip latency (RTT) accuracy is very important,Sketchy survey's RTT possibly causes an sketchy adjust to the congestion window.At present TCP Vegas algorithm had only considered take measures when the network congestion occurs on the data sending terminal,but if congestion occurs in the return route direction, also called the ACK direction, TCP Vegas algorithm mechanism will adopt the congestion avoided mechanism similarly, this will cause an underestimation to the actual throughput, thus causes congestion window nonessential reduction. In the practical application we are requested to be able to identify which direction do the congestion are occur, then adopts the correct measure to guarantee the network stability.In order to solve this problem, this article first analyzed the congestion avoid mechanism of TCP Vegas algorithm, obtained affects TCP Vegas algorithm RTT biggest factor is the reverse waiting time, therefore we can except this part to RTT's influence when computation. Simulation result indicated that the improvement algorithm

eliminated the network asymmetrical problem of TCP Vegas algorithm.

(3)Research of the Fusion of the reverse link jam and compatible question based on TCP Vegas algorithm

We have proposed corresponding improvement algorithm to the compatible problem and the network asymmetrical problem of TCP Vegas algorithm separately, the simulation testing has also proven the improvement algorithm validity. In the practical application we hope to find a "perfectly"algorithm as far as possible, this algorithm could solve all problems of TCP Vegas algorithm, met the present network development request.This article generalized analysis the improvement algorithm to the reverse link congestion problem,as well as the improvement algorithm to the compatible problem of TCP Vegas algorithm,then fusion them together,proposed TCP NewVegas algorithm. Simulation result indicated that this algorithm could maintain high band width competitive ability when uses with TCP Reno algorithm, also eliminate the influence of reverse link network congestion to the algorithm mechanism.

(4)Research of the discrimination service question based on auto-adapted hypothesized formation algorithm

The adaptive virtual queuing (AVQ) algorithm judges the network congestion based on the system load situation,its algorithm mechanism is maintains an virtual queuing that capacity is smaller than the actual link capacity,compared with the traditional initiative queue management algorithm, this algorithm has solved the deadbolt lock problem, maintained the fair performance of the service class, may reduce the queuing detention. But this algorithm cannot provide discrimination service, this is a very big flaw to the increasing network service type and the network environment complex trend of development.In order to solve this problem,this article has analyzed the dynamic threshold value algorithm and the queue size threshold value algorithm firstly, then synthesized these two algorithm'thought to the AVQ algorithm.Specifically speaking,we makes an improvement to the queue management of AVQ algorithm with the dynamic threshold value algorithm,use the queue size threshold value algorithm's thought to improve the AVQ algorithm's dispatch,through setting different priority to the different service, gives the different grade of service according to the different priority,thus achieves the guarantee discrimination service to satisfy the timely service.

Yuntao Wang (Communication and information systerm)

Supervised by Jian'an Fang

KEYWORDS: network congestion control, TCP Vegas algorithm,queue management, queue dispatch ,AVQ algorithm.

目录

第一章 绪论

1.1 引言

随着经济全球化和信息技术的迅速发展,作为当今全球性信息基础设施的计算机网络呈爆炸式增长,特别是以TCP协议为基础的Internet发展速度更是令人难以置信。目前，伴随着新型网络的不断涌现和网络用户数量的激增，Internet上的通信流量急剧增长，由此造成的网络问题不断涌现，其中最严重的问题就是网络拥塞。

自从互联网诞生以来，网络资源和网络流量分布的不均衡使的网络拥塞问题一直困扰着其发展。最早的网络拥塞现象发生在1986年10月,Van Jacobson在LBL和UCB之间的链路上观察到网络吞吐量从32Kbps骤然下降到40bps[1]，而且网络和协议都处于异常忙碌状态。通过分析网络分组，发现由于网络负载陡然增大，造成数据在网络中继或端节点的缓存溢出，丢失的分组导致数据重新发送，进一步恶化网络拥塞，形成“丢失-重发”的恶性循环。

伴随着网络规模的日益扩大和应用类型的丰富，网络拥塞问题变得越来越严重，如2007年2月26日，我国南海海域发生的强烈地震导致海底光缆中断，中国电信第一时间启动海缆故障应急处理预案，通过“天地大挪移”，多方组织其他国际海、陆缆资源，调配陆上光缆和卫星线路，全力疏通因地震受阻的国际通信业务。但是由于访问量过大，而可用资源的带宽不足，造成中国大陆和北美之间网络仍然严重拥塞。最近的网络拥塞现象发生在去年 5 月我国四川大地震期间，虽然地震对中国移动在重庆地区的通讯设备未造成严重损坏，但是由于震后手机拨打量激增，达到平常的2000倍，致使可用带宽资源远远小于人们的需求，导致网络拥塞现象的发生，市民通话不畅。重庆移动紧急开通了应急通信车，并通过增加容量、疏通话务量等技术处理工作保持通话正常。时时发生的网络拥塞现象已经成为制约网络发展和应用的一个瓶颈,网络拥塞问题也成为学术界关注的热点之一。

所谓网络拥塞(congestion)是指在分组交换网络中传输分组的数目太多时，由于存储转发节点的资源有限而造成网络传输性能下降的情况。当发生网络拥塞时，一般会出现数据包时延增加，丢包概率增加，网络吞吐量下降，网络效率下降，严重的网络拥塞会造成“拥塞崩溃”现象[1] [2]。

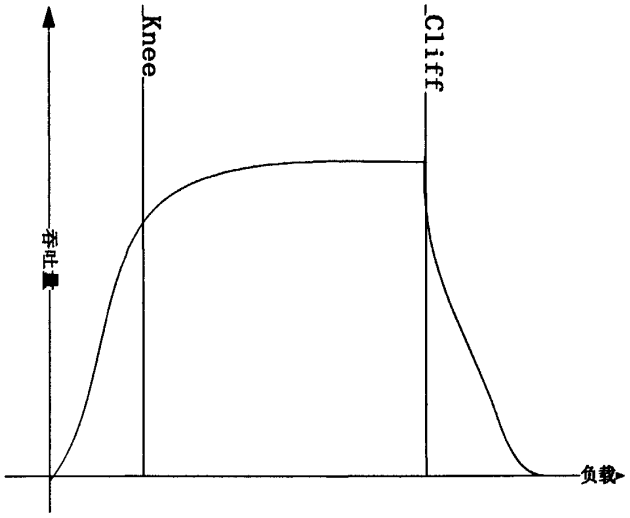

图 1-1 吞吐量随负载的变化

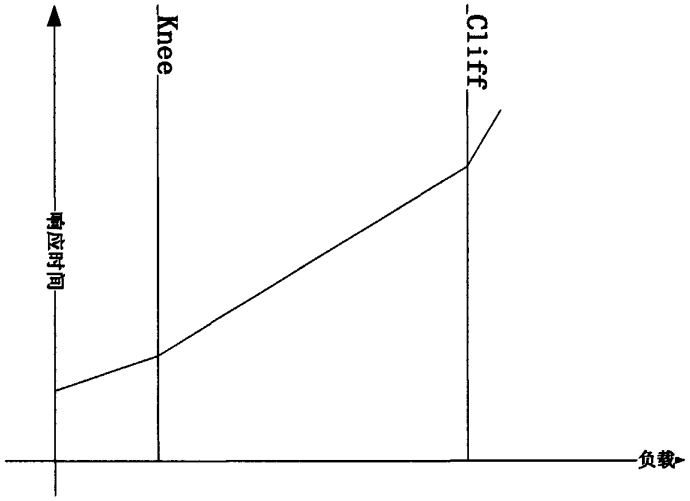

图 1-2 响应时间随负载的变化

对于拥塞现象，可以进一步用图 1-1 和图 1-2 来描述。当网络负载量较小时，吞吐量随着负载的增加而增长，呈线性关系，响应时间增长缓慢。当负载量达到网络容量时，吞吐量增长缓慢，而响应时间急剧增加，这一点称为 Knee。如果负载在此基础上继续增加，则增大到一定量时，吞吐量急剧下降，响应时间迅速增加，这一点称为 Cliff。从两图中可以看出，当吞吐量维持在 Knee 点附近时网络的使用效率最高[3]。

拥塞控制就是网络节点采取措施来避免拥塞的发生或者对拥塞的发生做出反应。拥塞控制机制实际上包含拥塞避免和拥塞缓解。前者的目的是使网络运行在 Knee 点附近，避免拥塞的发生；而后者则是使得网络运行在 Cliff 的左侧区

域。前者是一种“预防”措施，维持网络的高吞吐量、低延迟状态，避免进入拥塞；后者是一种“恢复”措施，使网络从拥塞中恢复过来，进入正常的运行状态。

网络拥塞发生的主要原因在于网络能够提供的资源不足以满足用户的需求，这些资源包括缓存空间、链路带宽容量和中间节点的处理能力。由于互联网的设计机制导致其缺乏“接纳控制”能力，因此在网络资源不足时不限制用户数量，而只能靠降低服务质量来继续为用户服务，也就是“尽力而为”的服务[4]。

资源的相对不足是引发网络拥塞的根本原因。这些资源包括链路带宽、可分配的处理器时间、缓冲区、内存等。考虑某个具体的流，如果在某个时间内对所到达的流量控制不力，使之超出了可分配的网络资源，那么将引发网络拥塞。总之，产生网络拥塞的主要原因可小结如下[1]：

(1)存储空间不足。当一个输出端口收到几个输入端口报文时，接收的报文就在这个端口的缓冲区排队。如果输出端口没有足够的存储空间存储，在缓冲区占满时，报文就会被丢弃，对突发的数据流更是如此。适当的增加存储空间在某中程度上可以缓解拥塞，但是如果过于增加存储空间，报文会因为在缓冲区排队时间过长而超时，源端会认为它们已经被丢弃因而选择重发，从而浪费了网络资源，并且进一步加重了网络拥塞。

(2)带宽容量不足。高速的数据流通过低速链路时也会产生拥塞。根据香农信息理论，任何信道带宽最大值即信道容量$C = B\log_2(1+S/N)$，所以节点接收数据流的速率必须小于或等于信道容量，才有可能避免拥塞。否则，接收的报文在节点的缓冲区中排队，在缓冲区占满时，报文被丢弃，导致网络拥塞。因此，网络中的低速链路将成为带宽的瓶颈和拥塞产生的重要原因之一。

(3)CPU处理速度慢，处理能力弱。如果节点在执行缓冲区中排队、选择路由的时候，CPU处理速度跟不上链路速度，也会导致拥塞。

(4)不合理的网络拓扑结构路由选择。这也会导致网络拥塞。

因此，解决拥塞问题，首先可以通过适当合理的增加网络资源来缓解，但是需要从全局系统的角度出发对以上影响原因综合考虑。其次，由于拥塞一旦发生往往会形成一个不断加重的恶性循环过程，最后会导致网络崩溃，因此必须从TCP/IP内部机制上采取措施，增加拥塞控制机制来避免和缓解拥塞。

1.2 研究背景及现状

中外学者针对网络拥塞进行了长期深入的研究，特别是在网络拥塞控制算法上提出许多新的机制来适应日益发展的因特网。

拥塞控制算法根据其作用和实现位置的不同，可以分为两大类：基于源端的TCP 拥塞控制算法和基于网络端的拥塞避免算法。TCP（Transmission Control Protocol）是当今 internet 广为使用的传输协议，K.Thomopson[5]等人指出，当前网络中 90%的数据流和大部分业务，如 FTP，HTTP 都是基于 TCP 的，而 TCP 所采用的拥塞控制算法是保证当今网络不断发展同时又避免拥塞崩溃的主要方法，所以，针对 TCP 人们提出不少拥塞控制算法,包括 TCP Tahoe[6]算法,TCP Reno[7]算法,TCP New Reno[8][9]算法,TCP SACK[10][11]算法,TCP Vegas[12]算法，FAST TCP[13]算法等等。这些拥塞控制算法大多数都采取了慢启动、拥塞避免、快速重传和快速恢复四种机制。基于网络端的拥塞控制算法主要集中在路由器中的队列管理和调度，提出了包括主动队列管理技术和区分服务技术。无论是基于TCP 端的拥塞控制算法还是基于网络端的拥塞避免算法，目的都是更好的解决网络拥塞问题。

拥塞控制机制在当前 internet 中发挥了重要的作用，但是由于在 internet 这样复杂的异构系统中不能期望所有用户在网络应用中兼容包括端到端的所有链路，针对 TCP 端各种拥塞控制算法的不足，人们提出许多改进机制，简述如下[1][14]：

(1)对慢启动的改进，原来的慢启动算法总是从cwnd=1开始，每收到一个ACK，cwnd增加1，对于RTT时间长的网络，为使cwnd达到一个合适的值，需要花很长的时间，特别是网络容量很大时，会造成很严重的浪费。为此可采用大的初始窗口，这样可以避免了延迟ACK机制下单个报文段初始窗口的等待超时问题，缩短了小TCP流的传输时间和大延迟链路上的慢启动时间。

在慢启动阶段，cwnd是按指数增长的，对于某些容量小的网络来说，很容易达到拥塞状态，即对于这类网络来说，慢启动的增长方式可能过于迅速，为此可采用较慢的增长方式来探测网络拥塞。

(2)对快速重传的改进。原来的快速重传算法是源端检测到拥塞后，要重传自丢失的数据包，至检测到丢失时发送的全部数据包，而实际上二者之间有些数据包已经正确传到接收端，不必重传。Matins 等人提出了选择确认算法 SACK，对数据包进行有选择的确认和重传，这样源端就能准确知道哪些数据包已正确传到

接收端，从而避免了不必要的重传，减少了时延，提高了网络吞吐量。

(3)对快速恢复的改进。最初的快速恢复算法是如果分组非顺序到达接收方，也会产生重复 ACK,而只有收到连续 3 次重复的 ACK 时才会激发快速重传，导致一定的时延和某些数据不必要的重传，在快速恢复阶段又会减少发送量，导致不必要的带宽浪费。而且当一个窗口有多个分组丢失时，如果没有足够的重复 ACK 到达，快速重传算法便会失效。有限传输机制，即当发送端收到 1 到 2 个重复的 ACK 后，如果被允许，发送端就发送一个新报文段。在许多情况下，有限传输允许小窗口的 TCP 连接不用等到超时发生就可以从一个窗口的数据丢失中恢复过来。

(4)对控制策略的改进。大部分 TCP 拥塞控制算法采用窗口控制策略，这种策略有一些缺陷：容易导致报文突发的出现；速率受到窗口大小的限制；一个窗口内多个报文丢失不容易恢复等等。为此，一些学者提出 RBP（Rate-Based Pacing），将基于窗口的控制和基于速率的控制结合起来以克服上述的缺陷。

除了上述几种改进之外，人们针对 TCP 端拥塞控制算法还提出许多其他方面的改进，包括对公平性的改进，对算法兼容性的改进，采用 ACK 过滤机制，采用显式拥塞通知等等一系列新方法，新措施，经过仿真证明这些举措在一定程度上提高了 TCP 端拥塞控制算法的性能。

由于仅仅依靠 TCP 拥塞控制机制不能确保网络的 QoS，要求网络层必须参与资源的控制和分配，早期人们提出在路由器中采用队列调度算法和缓存管理技术，伴随着网络条件的复杂化，人们对网络端的拥塞控制算法要求也越来越高，提出不少改进措施，简述如下[15]：

(1)路由器缓冲管理策略的改进。传统的路由器缓冲管理“尾部丢失”策略，报文到达时，如果缓冲队列已满，路由器则丢弃该报文。但是这种算法很容易产生持续的满队列状态，甚至导致业务流对缓存的死锁和业务流的全局同步。Floyd 提出了随机早期拥塞控制策略，有效改进了“尾部丢失”算法的缺陷，此后，许多学者相继提出如比例/比例积分（P/PI）、REM、AVQ、BLUE 等，用来改善网络拥塞控制的效果，提高了网络流量的公平性和拥塞控制系统的稳定性和鲁棒性。

(2)调度算法的改进。传统路由器采用“先来先服务”的调度算法，它按照占用度队列的多少分配带宽，无法实现公平性。对此，有的学者提出 GPS（Generalized

Processor Sharing）的分组调度策略，数据分组位于不同的逻辑队列，在有限时间间隔内，每一个非空队列都有机会接受服务，至少一次。如果为队列赋予了一定权限，接受服务与权值成正比，那么 GPS 便可以实现最大最小比例公平性。GPS 策略的一些简单和近似的实现算法有轮询、加权公平排队、最劣情况 WFQ、自时钟同步公平排队等等。

此外，许多改进还利用了控制理论和动力学的许多知识，建立网络流量的随机模型和决定模型，力求更好的控制网络拥塞。

1.3 研究目的及意义

网络拥塞控制研究是近些年来的研究热点。设计一种可靠、稳定的拥塞控制算法又是网络拥塞控制领域的研究重点，对算法的研究曾在短时间内取得很大的进展。但是早期的算法设计往往凭借观察来发现问题，利用直觉来设计解决方案，凭借仿真验证来达到目的，这种算法的设计在一定程度上限制了拥塞控制的发展。目前由于控制理论、优化理论、动力学、神经网络等被引入到拥塞控制算法中来，尝试利用其他知识来综合解决网络拥塞控制算法的情况日益增加，这些做法往往利用严格的数学模型描述一个简单的通信系统，对算法的稳定性、通信量进行分析，在一定程度上扩展了解决网络拥塞的渠道，但是由于网络的复杂性，这些理论分析还受到很多限制，尚有设计问题和算法机制有待进一步研究。

网络拥塞控制的研究致力于解决目前日益严重的网络拥塞问题，目前主要在 TCP 端和网络端两方面对拥塞算法机制、拥塞控制过程、拥塞恢复等问题进行分析，希望能够通过改进算法机制、优化控制过程等各种手段减轻网络拥塞对目前网络造成的危害。目前网络的复杂性日益增加，用户数量和业务量呈指数级上升，拥塞控制对于保证互联网的稳定具有十分重要的意义。网络中的拥塞源于网络资源和网络流量分布的不均衡性，且拥塞不会随着网络处理能力的提高而消除。拥塞控制算法的分布性、互联网的复杂性和对拥塞控制算法性能的高要求，使得拥塞控制算法设计具有很高的难度。虽然学术界在拥塞控制领域已经开展了大量的研究工作，但是到目前为止网络拥塞问题还没有得到很好的解决。

1.4 论文的主要研究内容及创新点

本文主要研究了网络拥塞控制领域的相关算法，包括 TCP 端拥塞控制算法和网络端拥塞控制算法两部分。目前 TCP 端的主要拥塞控制算法，包括 TCP

Tahoe 算法、TCP Reno 算法、TCP New Reno 算法、TCP SACK 算法和 TCP Vegas 算法，本文分析了各个算法的机制和优缺点，着重研究了 TCP Vegas 算法，根据在它在实际应用中出现的问题加以改进，力图在网络的非对称性和算法兼容性两方面提高算法性能，解决算法的反向拥塞判断失误问题和算法共存时竞争力不足的问题。对网络端拥塞控制算法的研究主要是从队列管理和队列调度两方面来进行的。队列管理算法中主要介绍了自适应虚拟队列(AVQ)算法，提出了一种改进的 AVQ 算法，并分析了控制技术在主动管理算法中的应用；队列调度方面主要介绍了几种主流的算法，在实际应用中队列管理和队列调度算法经常是配合使用，共同避免网络中的拥塞问题。论文的主要研究工作和创新点如下：

(1)基于 TCP Vegas 网络拥塞控制算法兼容性问题的研究

TCP Vegas 算法是一种基于测量技术的拥塞控制算法，相比传统的拥塞控制算法而言，TCP Vegas 算法在“慢启动”、“拥塞避免”和“快速重传”三个方面都做了改进。特别是在拥塞避免机制上通过比较实际吞吐量和期望吞吐量来调整拥塞窗口大小，将被动的拥塞避免机制变为主动预防的拥塞避免机制，极大地提高了算法的适用性。TCP Reno 算法是目前网络中的主流算法，它采用了被动的拥塞避免机制，通过持续增加自己的拥塞窗口，直到网络过载来保证有效利用网络资源；而 TCP Vegas 算法采用了主动的拥塞避免机制，其目标是将网络带宽维持在一个稳定的水平，所以它不会持续扩展自己的拥塞窗口。在实际应用中，TCP Reno 算法将会窃取 TCP Vegas 算法的带宽，导致 TCP Vegas 算法机制不能起到实际作用。为解决这个问题，本文对 TCP Vegas 算法设置的两个参数 α 和 β 进行了分析，将其由固定值变为动态变动数值，此外将拥塞窗口的指数级增长变为线性增长，仿真结果表明改进算法提高了 TCP Vegas 算法的带宽竞争能力和数据流量的稳定性。

(2)基于 TCP Vegas 网络拥塞控制算法网络非对称性问题的研究

TCP Vegas 算法是一种基于时间的算法，因此往返时延(RTT)的准确性至关重要，粗略测量的 RTT 会导致对拥塞窗口的粗略调整。目前 TCP Vegas 算法只是考虑了在数据发送方向上发生网络拥塞时采取措施，但是若拥塞发生在回路方向，也即 ACK 方向，TCP Vegas 的算法机制同样会采取拥塞避免措施，这样会引起对实际吞吐量的过低估计，导致拥塞窗口不必要的减小。实际应用中我们要求

能够辨认网络拥塞是发生在哪个方向上，进而采取正确的措施保证网络的稳定性。为解决这个问题，本文分析了 TCP Vegas 算法拥塞避免机制,得到影响 TCP Vegas算法RTT最大的因素是反向排队时间,为此可以在计算时除去该部分对RTT的影响。仿真结果表明，改进算法消除了 TCP Vegas 算法的网络非对称性问题。

(3)基于 TCP Vegas 算法中的反向链路拥塞和兼容性问题的融合性研究

我们分别针对TCP Vegas算法的兼容性问题和网络非对称问题提出自己的改进算法，仿真试验也分别证明了改进算法的有效性。在实际应用中我们都希望找到一种尽可能“完美”的算法，能够解决TCP Vegas算法中的各种问题，以适应目前网络的需要。本文综合分析了针对 TCP Vegas 算法中反向链路拥塞问题的改进算法，以及兼容性问题的改进算法，将它们融合在一起，提出了 TCP NewVegas 算法。仿真结果表明，该算法可以在和 TCP Reno 算法并用时保持较高的带宽竞争能力，也可消除反向链路的网络拥塞对算法机制的影响。

(4)基于自适应虚拟队列算法的区分服务问题的研究

自适应虚拟队列（AVQ）算法是根据系统负载情况来判断网络拥塞的，其算法机制是维持一个容量小于实际链路容量的虚拟队列。和传统主动队列管理算法相比，该算法解决了死锁问题，保持了业务流的公平性，可以减小排队延迟。但该算法不能提供区分服务，这对于目前网络业务种类不断增加，网络环境日益复杂的发展趋势来说是个很大的缺陷。为解决这个问题，本文首先分析了动态阈值算法和队列长度阈值算法，综合了这两种算法的思想用于 AVQ 算法。具体来讲是利用动态阈值算法的思想对 AVQ 算法的队列管理进行改进,利用队列长度阈值算法的思想对 AVQ 算法的队列调度进行改进,通过对不同的业务设置不同的优先级,根据不同的优先级给予不同的服务等级,从而达到保证区分服务来满足实时性业务。

1.5 论文的结构安排

本文的结构安排如下：

第一章，绪论。本章主要介绍了网络拥塞控制研究的背景和内容，网络拥塞控制算法研究的现状，给出了本文的主要研究工作和创新点。

第二章，网络拥塞控制及网络仿真器 NS2 介绍。本章首先介绍了 TCP 协议，TCP/IP 协议是目前网络中应用最广泛的协议，了解 TCP 协议的实现原理和相关

结构是研究 TCP 端网络拥塞控制的基础。接下来我们介绍了基于 TCP 端的网络拥塞控制，一般包括慢启动、拥塞避免、拥塞恢复和快速重传四种算法，最后介绍了本文所用的网络仿真软件 NS2，并进行了仿真演示。

第三章，基于 TCP 端的网络拥塞控制算法研究。对 TCP 端拥塞控制的常用算法，如 TCP Tahoe 算法、TCP Reno 算法、TCP New Reno 算法、TCP SACK 算法和 TCP Vegas 算法进行了分析，总结了各个算法的实现机制和优缺点，并在不同网络环境下仿真比较了各种算法的性能。TCP 端拥塞控制算法主要是在源端通过对网络流量的调节来避免网络拥塞问题。

第四章，TCP Vegas 拥塞控制算法的研究和改进。本章主要研究了 TCP Vegas 拥塞控制算法。首先针对 TCP Vegas 算法的网络非对称性问题进行了深入的研究，分析了算法无法判断反向拥塞出现的原因，根据数学模型分析提出自己的改进措施，通过对 TCP Vegas 拥塞避免算法的改进，可以有效解决由于反向回路出现拥塞导致网络吞吐量下降的问题。然后针对 TCP Vegas 算法与 TCP Reno 算法共存时的不兼容问题进行了研究，得出 TCP Vegas 算法阈值限制了算法网络带宽竞争能力的结论，提出了改进的 TCP NewVegas-A 算法，经仿真验证可以有效解决算法的兼容性问题。最后本文综合了上述两个问题的解决方案，力图利用一种算法解决上述两个问题，提出了 TCP NewVagas 算法，仿真分析证明了该算法的有效性。

第五章，基于网络端的拥塞控制算法研究。本章详细研究了网络端的拥塞控制算法，主要包括队列管理算法和队列调度算法。队列管理算法的主要作用是管理路由器中的队列，兼顾队列公平并有效的与 TCP 端的拥塞控制配合。在本章的一到四小节主要介绍了队列管理算法，包括主动管理算法以及控制技术在其中的应用，并提出了一种改进的自适应虚拟队列算法。随后介绍了网络端拥塞控制的另一主要算法---队列调度算法。队列调度算法主要用于管理各流之间带宽的分配，在第五小节中分析了几种典型的队列调度算法，本章最后介绍了目前队列管理和队列调度的研究进展，并提出自己的见解。在实际应用中队列管理和队列调度算法经常是配合使用的，共同避免网络中的拥塞问题。

第六章，总结和展望。对本文的研究内容进行了了总结，对以后网络拥塞控制方面的研究进行了展望。

第二章 网络拥塞控制及网络仿真器 NS2 介绍

2.1 引言

网络拥塞控制问题是目前网络控制领域的研究热点。拥塞控制就是网络节点采取措施来避免拥塞的发生或者对拥塞的发生做出反应。K.Thomopson[17]等人指出，当前网络中 90%的数据流和大部分业务，如 FTP，HTTP 都是基于 TCP 的。而 TCP 采用的拥塞控制算法又是保证网络畅通的重要机制，所以基于 TCP 端网络拥塞控制算法的研究是网络拥塞控制领域的重要研究方向之一。

研究 TCP 端的拥塞控制算法必须对 TCP 协议有完全的了解，TCP 报文段结构、TCP 传输连接机制、TCP 实施的发送策略、交付策略、接收策略、重传策略、确认策略等都是研究 TCP 拥塞控制算法的基础。本章首先对 TCP 协议的基础知识进行了详细的介绍，目的是为以后算法的研究奠定基础。在了解 TCP 协议的基础上，对 TCP 拥塞控制算法机制的构成进行了初步介绍，主要包括慢启动、拥塞避免、快速重传和快速恢复四个相关联的算法，这些算法在后面章节中会有详细的研究。

本章主要介绍了基于 TCP 端网络拥塞控制的基本知识，包括：TCP 协议的相关知识，它的传输报文结构和连接机制；TCP 网络拥塞控制机制，它的主要过程：慢启动、拥塞避免、快速重传和快速恢复，并对各个阶段的实现过程做了较为详细的阐述。最后介绍了网络仿真软件 NS2[18]，由于 NS2 的使用比较复杂，而且本文中大量使用了 NS2 来进行算法仿真试验，所以本章简单介绍了 NS2 的使用过程，也是为以后的研究工作奠定基础。

2.2 传输控制协议（TCP）简介

TCP（Transmission Control Protocol）是一种面向连接的传输控制协议，是目前网络上最主要的网络协议，它提供了可靠的数据传输服务，在保障网络通信性能方面起着非常重要的作用。

每台支持 TCP 的机器都有一个 TCP 传输实体，它管理 TCP 流以及与 IP 层之间的接口。TCP 传输实体接收本地用户进程的数据流，并将它们分割成大小不超过 64KB 的分片，然后以单独的 IP 数据报的形式发送每个分片。当包含 TCP 数据的数据报到达目的主机时，它们被递交给 TCP 传输实体，TCP 传输实体重

构出原始的字节流。两台机器上 TCP 实体之间传输的数据单元称为报文段，TCP 通过报文段的交互来建立连接、传输数据、发出确认、通告窗口大小等。TCP 报文段的结构如图 2-1 所示[1]：

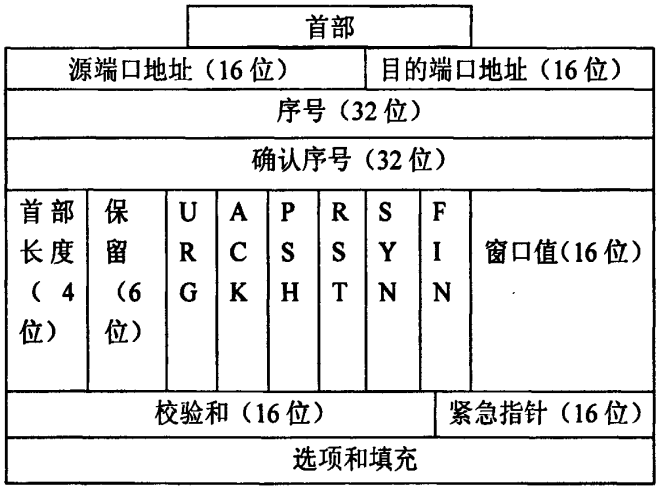

图 2-1 TCP 报文段的结构

报文段主要结构包括：

1.源端口地址（16 位）：定义了在主机发送该报文段的应用程序的端口号，以识别本地 TCP 用户。

2.目的端口地址（16 位）：定义了在主机中接收该报文段的应用程序的端口号，以识别目的 TCP 用户。

3.序号（32 位）：指明当前数据段在发送方数据字节流中的位置。

4.确认序号（32 位）:指明本机希望接受下一个数据字段的序号。

5.ACK:确认比特位。如果此比特位被置为“1“，表示确认序号是有效的。如果 ACK 被置为 0，则该数据段不包含确认信息，确认将被忽略。

6.SYN：同步比特位。用于建立连接的过程，它与确认比特 ACK 配合使用。SYN=1 和 ACK=0 表明这是一个请求建立连接的的报文段，若对方同意建立连接，则发回的确认报文段中将设置 SYN 为“1”，ACK 也设置为“1”。

7.FIN：终止比特位。用于释放一个连接。当 FIN=1 时，表示发送方已经没有数据要传输了，要释放传输连接。

8.窗口值（16 位）：流量控制使用的信用量，表示接受主机能够接收的数据大小，单位是字节。TCP 的流量控制是通过一个可变大小的滑动窗口来完成的。窗口的大小指定了发送端通知接收端在没有收到发送端的确认报文段时，接收端可以发送的数据的最大字节数。

TCP 是面向连接的协议，在传输 TCP 用户数据报之前，首先要建立传输连接，在用户数据报传输的过程中需要维护传输连接，通信结束后要释放连接。

TCP 基于客户机/服务器的模式，采用“三次握手”的方式建立连接。主动发起连接的应用进程叫做客户机，被动等待连接建立的应用进程叫做服务器。“三次握手”建立连接的过程一般为[1]:

第一次握手，源端机发送一个带有本次连接序号的请求；第二次握手，目的主机收到请求后，如果同意连接则发回一个带有本次连接序号和源端机连接序号的确认；第三次握手，源端机收到含有两次初始序号的应答后，再向目的主机发送一个带有两次连接序号的确认。当目的主机收到确认后，双方就建立了连接。图 2-2 示：

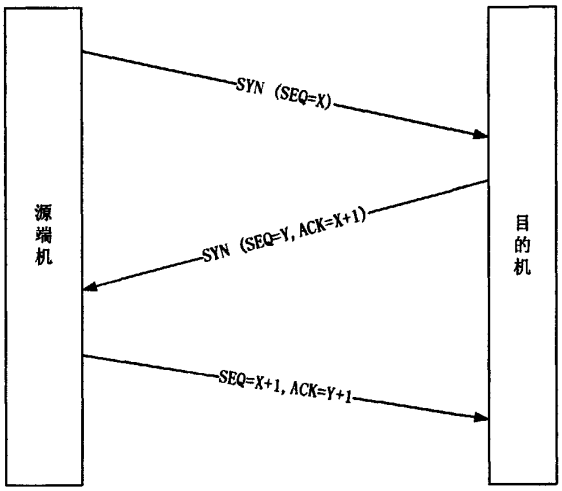

图 2-2 TCP 连接的建立过程

TCP 连接的释放同样采用“三次握手”的方式，图 2-3 示：

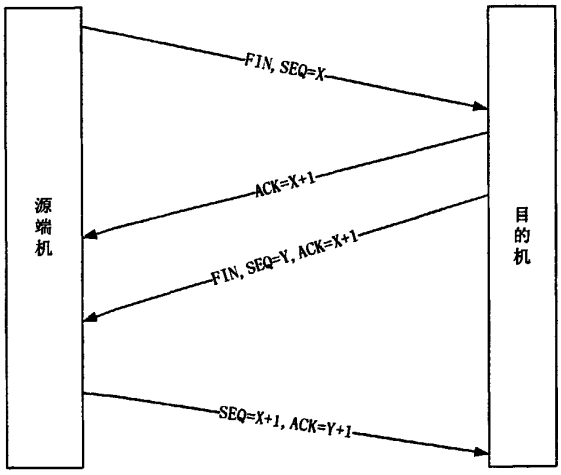

图 2-3 TCP 连接的释放过程

TCP 实施的策略，TCP 标准对于 TCP 实体之间使用的协议制定了精确的规约。然而，协议的某些方面允许几种可能的实施选项。虽然两种选择不同选项的实现方案可以互相操作，但性能可能受到影响。TCP 实施的策略主要包括[1]：发送策略、交付策略、接收策略、重传策略、确认策略。

发送策略：TCP 实体通过发送窗口传输数据，数据被用户发出后，就被缓冲在传输缓存中。TCP 可以对用户提供的每一批数据都构成一个报文段，也可以等积累一定数量的数据之后再构造和发送一个报文段。实际策略取决于性能考虑，如果传输不经常而每次传输较大，则报文段产生和处理开销就较低。另一方面，如果传输比较经常而每次传输较小，则系统反应速度就较快。

交付策略：TCP 实体可以在每次报文段按序收到的时候就交付给用户，也可以将多个报文段缓存在接收缓存中然后再交付。实际的策略取决于性能考虑。如果交付不经常而每次交付的数据较多，用户可能就不像它希望的那样及时收到数据。另一方面，如果交付频繁而每次交付的数据量较小，则不仅会造成不必要的操作系统中断，而且会造成 TCP 和用户软件不必要的处理开销。

接收策略：接收端 TCP 实体将接收到的数据放在一个缓存中，以便交付给用户。然而，也可能发生报文段到达失序的情况。这时，接收端 TCP 实体会有两种可选策略，一是仅接收按序到达的报文段，丢弃失序到达的报文段。二是接受所有处于接收窗口中的报文段。

重传策略：TCP 发送端维持一个已经发送但尚未得到确认的报文段的队列。规定：如果 TCP 发送端没能在一个给定的时间内收到确认，那么它就要重传这个报文段。TCP 实体可以采用的重传方式包括：单个传送、成批传送、只重传头一个等。

确认策略：当一个数据报文段按顺序到达时，接收 TCP 实体有两种可能的确认方式。一是立即确认，也即当收到数据时，立即传输一个包含相应确认序号的空报文段。二是累积确认，当收到数据时，把确认的需要记录下来等待一个携带数据的出向报文段，以便将确认捎带在这个报文段上。

TCP 差错控制：TCP 是一个可靠的传输层协议，差错控制包括以下一些机制：检测收到损伤的报文段、丢失的报文段、失序的报文段和重复的报文段。TCP 差错控制在拥塞控制中应用广泛。

2.3 TCP网络拥塞控制机制

基于TCP端的拥塞控制对internet的稳定性起到了关键作用。互联网中，TCP协议使用一种滑动窗口机制控制源端数据分组的发送以调整发送数率。TCP拥塞控制的基础是加性增/乘性减（ATMD：additive-increase multiplicative-decrease）[19]策略，主要包括以下四个相关联的算法：

慢启动：Jacobson推荐了慢启动的规程，即在一个网络连接刚初始化的时候，TCP使用了一个很小的拥塞窗口cwnd来探测网络的使用情况，以保证不会把太多的数据分组发送进一个已拥塞的环境。一般TCP实体初始化cwnd=1，即TCP只被允许发送一个报文段，然后就必须等待确认再传输第2个报文段。随着每个确认帧（ACK）的到达，cwnd的值就被加1，一直到某个最大值为止。显然，cwnd是以指数规律增长的。

慢启动算法通常用于连接开始阶段和超时引起的重传丢包阶段，作用是探测当前网络可提供的容量，避免由于发送大量突发数据导致网络拥塞。

拥塞避免：Jacobson[20]指出网络进入饱和状态很容易，但让网络从饱和状态中恢复却很难。意思是说一旦拥塞发生了，要清除拥塞可能需要很长时间。因此，慢启动中的cwnd的指数增长就可能太激进，它可能会使拥塞更加严重。Jacobson提出了拥塞避免算法，首先，设置慢启动门限为目前拥塞窗口的一半大小，即ssthresh=cwnd/2。其次，设置cwnd=1并执行慢启动过程直到cwnd=ssthresh。在这个阶段，cwnd在每收到一个ACK时都加1。当cwnd大于慢启动阀值ssthresh时，在此阶段，每个RTT时间内，若发送方收到对当前数据的应答后，cwnd就比原来增加一个最大TCP数据包所对应的字节数，即cwnd线形增长。

拥塞避免的作用是通过减缓cwnd值的增加速率推迟网络拥塞的发生，这样使发送端能在较长一段时间内保持较高的数据传输率。图2-4说明了慢启动和拥塞避免算法[1]：

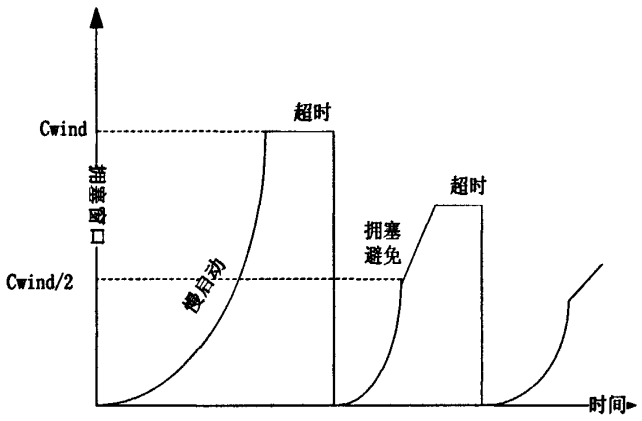

图2-4 慢启动和拥塞避免

快速重传：在数据传输过程中，如果有一个报文段丢失了，TCP 可能不能及时重传。通常发送 TCP 实体会设定一个重传定时器（RTO），用来确定什么时候对一个报文段进行重传。一般说来,RTO 会比该报文段 ACK（acknowledgement: 确认）到达发送方的所花的实际往返时延（RTT）长许多。

快速重传利用了利用了 TCP 中的下列规则[21]：如果一个 TCP 实体收到一个失序报文段，它必须立即发出一个对于最后一个收到的按序报文段的 ACK。对于每一个到来的报文段，TCP 将继续重复发送这个 ACK，直到丢失的报文段到达填补了缓存中的空隙。空隙填补后，TCP 就对所有迄今为止按序收到的报文段发送一个积累 ACK。当源端 TCP 收到一个重复的 ACK 时，意味着一是被确认的报文段后面的报文段被延迟，以致它最终失序到达。一是这个报文段丢失。为了确信这个报文段的却是丢失了，Jacobson 建议 TCP 发送方要等待收到一个报文段的 3 个重复 ACK，在这种情况下，随后的报文段被丢失的可能行性就很大，因此，应该被重传，而不是等着超时重传。

快速重传主要用来检测和恢复数据包的丢失。

快速恢复：当使用快速重传算法重传一个报文段时，它知道一个报文段丢失了，按照前面算法来讲，这意味着发生了网络拥塞,TCP 实体应采取拥塞避免措施，使网络从拥塞状态中恢复。Jacobson 认为这种做法有点保守，提出了快速恢复算法[19]。即快速重传可能丢失的数据包之后，TCP 就执行拥塞避免算法（而不是慢启动算法），此后，快速恢复算法将控制新数据的传送，直到收到一个非重复的 ACK 为止。

快速恢复算法可使大窗口下发生中度拥塞时网络仍具有较高的吞吐量，此算

法也可减少快速重发造成的 cwnd 急剧振荡。图 2-5 说明了快速重传和快速恢复算法。

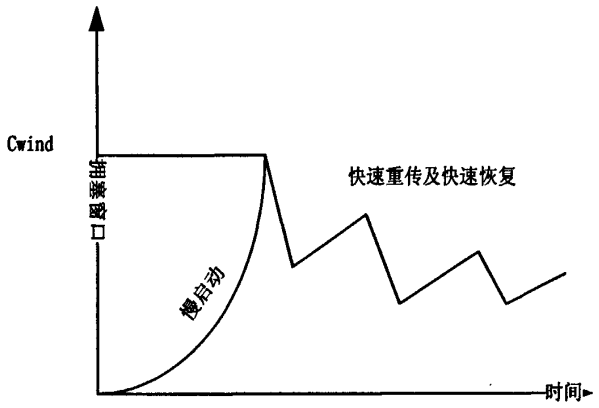

图2-5 快速重传和快速恢复

2.4 NS2 网络仿真器简介

NS（Network Simulator）网络仿真是一个近年来提出的新概念。它可以建立网络模型，根据相关网络协议和网络算法进行仿真。它一个离散事件模拟器。所谓离散事件模拟是指事件规定了系统状态的改变，状态的修改仅在事件发生时进行。在网络模拟器中，典型的事件包括分组到达、时钟超时等等。

NS是由LNBLLNBL（Lawrenc Berkeley National Laboratory）的网络研发小组开发，前身是REAL仿真器，这是一个可扩展、易配置、可编程的事件驱动仿真引擎，NS带有大量协议库支持，支持TCP Reno、TCP SACK、TCP Vegas等网络拥塞算法。它含有开放的源代码，提供用户接口，是一个具有开放体系结构的网络仿真软件。

2.4.1 NS2体系结构

NS2是NS的当前版本，它的主要包括Tcl/ Tk、OTcl 、NS、Tclcl 四部分。它的体系结构[21]可用图2-6表示

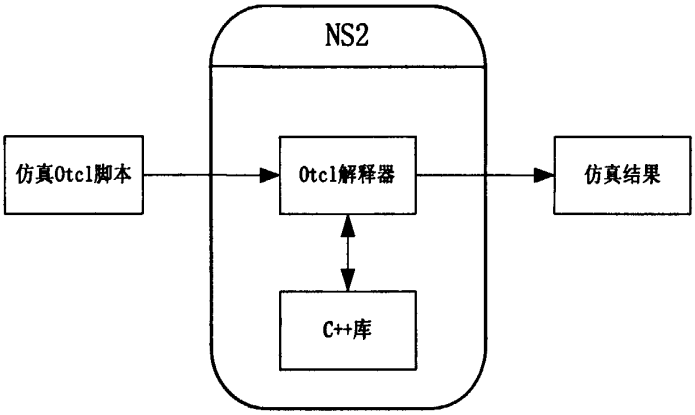

图2-6 NS2仿真器结构

NS2 是使用 C++和 Otcl 两种面向对象的语言编写的。C++是众所周知的一门高级编程语言，而 Otcl 是 MIT 开发的 ObjectTCL，即 Tcl 的面向对象的扩展。Tcl 的全称是 Tool Command Language，它是一种灵活的、交互式的脚本语言，Otcl 则在 Tcl 的基础上增加了类、实例、继承等面向对象概念。NS2 的常用构件通常是作为一个 C++类来实现的，同时有一个 Otcl 类与之相适应。NS2 同时使用 C++和 Otcl 两种面向对象的语言编写是出于兼顾模拟性能和灵活性两方面的考虑。一方面 NS2 使用 C++编写基本的组件对象和时间调度器，用 C++来实现功能的模拟，可以大大减少处理报文和时间的时间，更迅速高效的执行模拟过程。另一方面 NS2 使用 Otcl 编写脚本命令和配置接口，用 Otcl 进行模拟配置和解释执行的模拟过程，可以在不必从新编译的情况下随意修改模拟参数和模拟过程，提高了模拟的效率。

NS2 中进行模拟时，模拟过程是由一个名为 Simulator 的 Tcl 类来定义和控制的，Simulator 类提供了一系列的模拟配置接口，其中就包括“事件调度器”的接口。进行模拟时首先需要创建一个 Simulator 类的实例对象，并调用该对象的一系列方法来创建节点、链路、拓扑等模拟对象。

NS2 模拟器封装了许多功能模块，包括节点、链路、分组、代理、流量发生器、应用模拟器等，利用这些对象，可以方便快捷的构建仿真环境。下面简单介绍一下这些对象[22]。

事件调度器是 NS2 中的调度中心，它用来跟踪记录整个仿真的时间，调度时间队列中的事件，激活相应的网络组件来处理当前时刻提出处理要求的事件。它一般会从所有事件中选择发生最早的事件执行，调用它的 handle 函数，把该事

件执行完毕，然后从剩余的所有事件中选择发生时间最早的执行，如此反复。NS2 只支持单线程，如果多个事件安排在同一时刻，那么会按照事件代码插入的先后次序执行。

节点用来表示端节点和路由器，它主要由端口分类器、地址分类器、多播分类器和复制分类器等构件组成。分类器从逻辑上匹配一个分组，并基于匹配结果把该分组传递给相应的对象。复制器是生成一个分组的多分拷贝，并把这些拷贝转发到各个订阅了某一多播组 G 的输出链路。

分组是对象交互的基本单元。由一系列分组头和一个可选的数据空间组成。分组头的结构在 Simulatord 对象创建时候就被初始化了，同时每个分组头相对于分组的起始地址的偏移量也被记录下来，提供用户来存取各个头部所包含的信息。

链路用来连接网络节点，它主要是由 DelayLink、Queuse 和 TTLChcker 等连接器构成。DelayLink 构造链路带宽和延迟特征；Queuse 构造和模拟与该链路相连的路由器的输出缓冲；TTLChcker 对该链路的数据包的 TTL 字段减 1 操作，并丢弃 TTL 的值为 0 的数据包。所有链路都是以队列的形式来管理分组的到达、离开和丢弃。

代理代表了网络层分组的起点和终点，并被用于实现如 TCL 和 UDP 等网络协议。代理类支持分组的产生和接收，C++的代理包含了一系列的内部状态变量来表示分组的各个域。代理可以实现多个层次的协议，对于一些运输层的协议，分组的大小和发送时间通常由代理提供的应用程序接口（API）来控制，对于在低层使用的代理，分组的大小和发送时间通常由代理自己控制。

流量发生器、应用模拟器是构建在运输层代理之上，流量发生器是模拟应用程序产生网络通信量，有 4 类：EXPOO_Traffic、POO_Traffic、CBR_ Traffic、TrafficTrace，他们一般用在 UDP 代理之上，应用模拟器有 FTP 、Telnet 一般用在 TCP 代理之上。

2.4.2 NS2 网络模拟

利用 NS2 进行网络模拟一般涉及两个层次。一个是基于 Otcl 编程的层次，利用 NS2 已有的网络元素，编写相关的 Otcl 脚本即可进行网络模拟，无需对 NS2 本身作任何修改；另一个是基于 C++和 Otcl 编程的层次，在这个层次中，首先

需要对 NS2 进行扩展，添加 NS2 中没有但自己所需的网络元素。这需要自己编写 C++类和 Otcl 类，然后编写 Otcl 脚本。

NS2 的网络模拟过程可用图 2-7 如下表示：

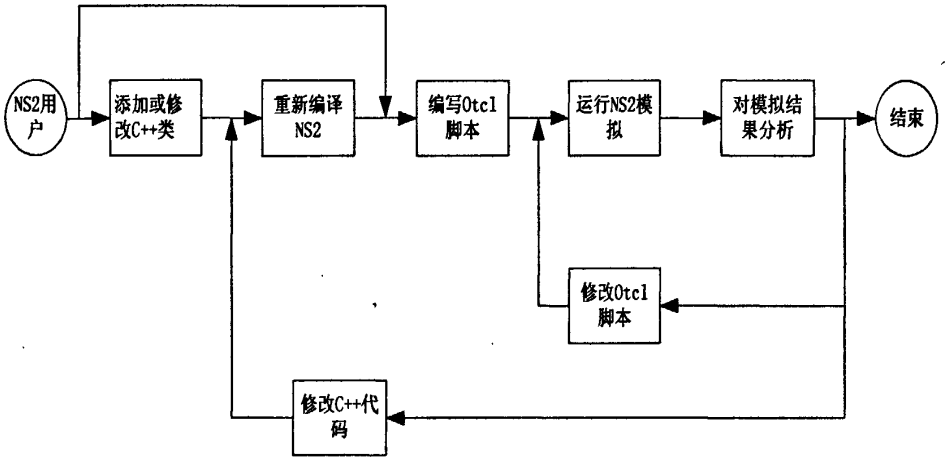

图 2-7 NS2 进行网络模拟过程示意图

用 NS2 的进行网络模拟的一般步骤大致如下[23]：

1 建立网络环境：建立网络拓扑图，确定基本参数，包括链路带宽，延迟时间等。

2 建立跟踪模型：包括网络传输协议和通信业务量模型的建立，配置业务量模型的参数，从而确定网络上业务量分布。

3 设置跟踪对象：跟踪对象可以记录模拟过程中发生的特定类型事件，并将其记录在跟踪文件中。一般用户会在模拟完成后对跟踪文件进行分析研究。

4 编写辅助过程，设定模拟开始时间，运行 Otcl 脚本文件。

5 编写辅助过程，设定模拟结束时间，至此 Otcl 脚本文件运行完成。

6 用 NS2 对刚才编写的 Otcl 脚本进行解释。

7 分析跟踪结果：对跟踪文件进行分析，可以利用 gwak、xgraph、gunplot 等画图工具得出有用的数据曲线，或者利用 NAM 观看网络模拟过程。

8 调控配置网络拓扑结构和业务量模型，重新进行上述模拟过程。

2.4.3 NS2 仿真示例

本节将通过一个简单的例子来介绍如何用 NS2 进行网络模拟,以及对仿真后的文件进行分析。我们构建的网络拓扑结构如图 2-8 所示，网络环境共包括 4 个节点（n0、n1、n2、n3），各个节点之间的链路带宽和延迟时间如图中标记，每

条链路均采用 Drop Tail 策略，且在 n2 和 n3 之间的最大队列长度是 10 个封包的长度。在节点 n0 和 n3 之间有一条架构在 TCP 之上的 FTP 连接，在 n1 和 n3 上有一条构架在 UDP 上的 CBR 连接。设定 CBR 的传送速度是 1Mbps,每个封包的大小是 1Kbytes, CBR 在 0.1 秒开始发送，4.5 秒结束。FTP 是在 1.0 秒开始发送，在 4.0 秒结束。

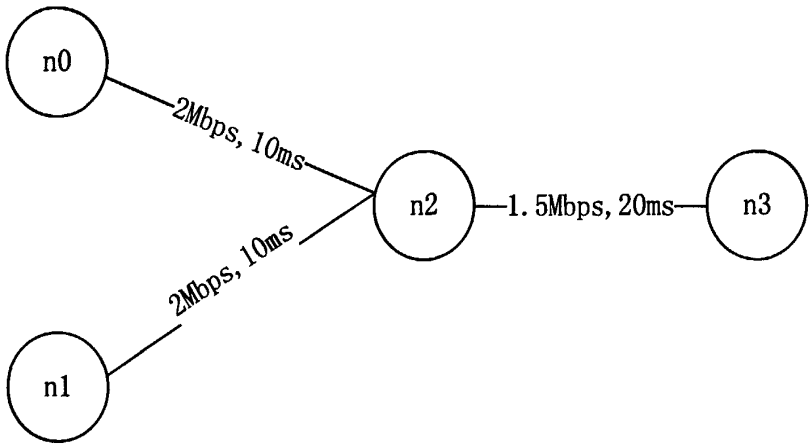

图 2-8 示例模拟网络结构

在此例中我们并没有添加和修改 NS2 中已有的类和协议，所以只需要使用 Otcl 语言来编写模拟脚本文件即可，具体过程如下：

```
#创建一个仿真对象
set ns[new simulator]
#定义不同颜色的数据流，这主要是供 NAM 使用
$ns color 1 Blue
$ns color 2 Red
#打开一个 NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf
#打开一个 trace file 记录数据包的传送过程
set nd[open out.tr w]
$ns trace-all $nd
#定义一个结束程序
Proc finish{} {
    Global ns nf nd
```

```
    $ns flush-trace
close $nf
close $nd
#以后台方式执行 NAM
exec nam out.nam&
exit 0
}
#创建 4 个网络节点
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#把节点连接起来
$ns duplex-link $n0 $n2 2Mb 10 ms Drop Tail
$ns duplex-link $n1 $n2 2Mb 10 ms Drop Tail
$ns duplex-link $n2 $n3 1.5Mb 20 ms Drop Tail
#设定 n2 到 n3 之间的最大队列长度为 10 个封包大小
$ns queue-limit $n2 $n3 10
#设定节点的位置，这是供 NAM 用的
$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
#观测 n2-n3 之间队列的变化，供 NAM 使用
$ns duplex-link-op $n2 $n3 queuePos 0.5
#建立一条 TCP 连接
set tcp [new Agent/TCP]
$tcp set class_2
set attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
```

```
$ns connect $tcp $sink
#在 NAM 中，TCP 的连接以蓝色表示
$tcp set fid_1
#在 TCP 连接上建立 FTP 应用
set ftp [new Application/FTP]
$ftp attach-agent $ftp
$ftp set type_FTP
#建立一条 UDP 的连接
set udp[new Agent/TCP]
set attach-agent $n1 $ udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $ null
#在 NAM 中，UDP 的连接以红色表示
$udp set fid_2
#在 UDP 连接上建立 CBR 应用
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_flase
#设定 FTP 和 CBR 的开始和结束时间
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
#在 5.0 秒调用 finish 过程结束模拟
$ns at 5.0 "finish"
```

#执行模拟

$ns run

结束仿真后，会生成 2 个文件，一个是 out.nam 文件，这是供 NAM 用的，可以将仿真过程用动画的方式呈现出来，可以让读者直接观察来了解数据包的传送过程。截取的图片如 2-9 所示。另一个文件是 out.tr，它记录了方阵过程中数据包传送中的所有事件，这一般是分析的重点。

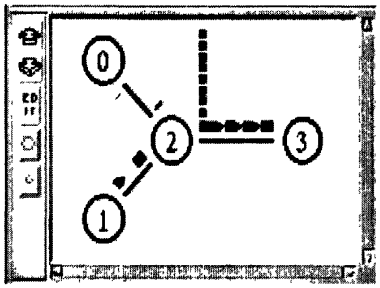

图 2-9 NS2 仿真动画

我们可以用文档的方式打开 out.tr 文件，发现里面的数据是有规律排列的，表 2-1 是截取的一段数据（为了便于观察，我们将数据排放在表格中，实际文件中并无表格）

表 2-1 out.tr 文件数据截取图

+	0.1	1	2	cbr	1000	------	2	1.0	3.1	0	0
-	0.1	1	2	cbr	1000	------	2	1.0	3.1	0	0
+	0.108	1	2	cbr	1000	------	2	1.0	3.1	1	1
-	0.108	1	2	cbr	1000	------	2	1.0	3.1	1	1
r	0.114	1	2	cbr	1000	------	2	1.0	3.1	0	0
+	0.114	2	3	cbr	1000	------	2	1.0	3.1	0	0
-	0.114	2	3	cbr	1000	------	2	1.0	3.1	0	0
+	0.116	2	2	cbr	1000	------	2	1.0	3.1	2	2
-	0.116	1	2	cbr	1000	------	2	1.0	3.1	2	2
r	0.122	1	2	cbr	1000	------	2	1.0	3.1	1	1

在仿真后生成的 out.tr 文件中，数据就是这样一行行的整齐排列的，其中的每一行都代表了一个离散事件，每一列都遵循了固定的格式，代表了一定的意义。具体来讲，第一列表示事件发生的类型，若是“+”表示报文进入队列，若是“-”表示离开队列，若是“r”表示报文被节点接收，”d”表示报文被结点丢弃；第 2 列指出事件发生的时间；第 3 列表示事件发生的地点，也就是报文发出的节点；第 4 列表示报文被接收的节点；第 5 列表示报文的类型；第 6 列表示报文的大小；第 7 列表示报文的棋标标志，在仿真动画中可以利用它来为不同的流设置不同的颜色，使得动画更加直观；第 8 列表示报文属于哪个资料流；第 9 列和第 10 列分

别 表示报文的来源端和目的端；第 11 列表示报文的序号；最后一列表示报文的惟一标志号。

根据以上含义，则表 4-1 中第一行数据的含义即为：一个报文的序号为 0，长度为 1000bytes，类型属于 cbr，封包号为 0，资料流 id 号为 2，它是从源端 1.0 到达目的端口 3.1,在时间为 0.1 秒时，从节点 1 进入了结点 2 的队列中。

大部分利用 NS2 进行仿真的试验在得到.tr 文件后都要对其中的数据进行分析，这就要利用到一种新的程式语言，简称为 awk 语言，它专门用于处理成行排列的规范化数据，我们可以利用 awk 语言来编写程序，用来在.tr 文件中取出想要的数据来得到希望的变量结果，例如，利用编写的程序可以得到关于吞吐量、延迟变化、丢包率、拥塞窗口变化等等的文件。

例如下面的 awk 程序记录了 cbr 报文的延迟时间变化。

```
  BEGIN {
#程序初始化
    old_time=0;
    old_seq_no=0;
    i=0;
}
{
    action = $1;
    time = $2;
    node_1 = $3;
    node_2 = $4;
    type = $5;
    pktsize=$6;
    flow_id = $8;
    node_1_address = $9;
    node_2_address = $10;
    seq_no = $11;
    packet_id = $12;
```

```
#判断是否为 n2 传送到 n3，且封包型态为 cbr，动作为接受封包

    if(node_1==2 && node_2==3 && type=="cbr" && action=="r")
{
#求出目前封包的序号和上次成功接收的序号差值
                dif=seq_no-old_seq_no;
#处理第一个接收封包
        if(dif==0)
        dif=1;
#求出 jitter（延迟变化量）

            jitter[i]=(time-old_time)/dif;
            seq[i]=seq_no;
            i=i+1;
            old_seq_no=seq_no;
            old_time=time;
    }
}
END {
    for (j=1; j <i;j++)
    printf("%d %f ",seq[j],jitter[j]);
}
```

由于网络的流量是不断变化的，每个报文从发送端到目的端的传输时间不会完全一样，当网络中流量大时，可能会导致过多的报文段排队等待，传输时间就会相应较长，传输时间的差异性能就是延迟时间变化，通常延迟时间越大，表明网络越不稳定。所以我们可以利用这个程序来判断网络的稳定性。

单是利用 awk 程序得到的仍然是一个数据文件，通常会再利用 xgraph 或 gnuplot 画图工具将这个文件转换为图像，这样我们就能在图片中直接观察得到想要的结果。本例中我们选择了 gnuplot，利用 gnuplot 命令可以设定相应的图片

名称，调节坐标轴的长度和和间隔量等工作，这些命令比较简单，不详细赘述，这里利用 gnuplot 画出的图片如图 2-10 所示

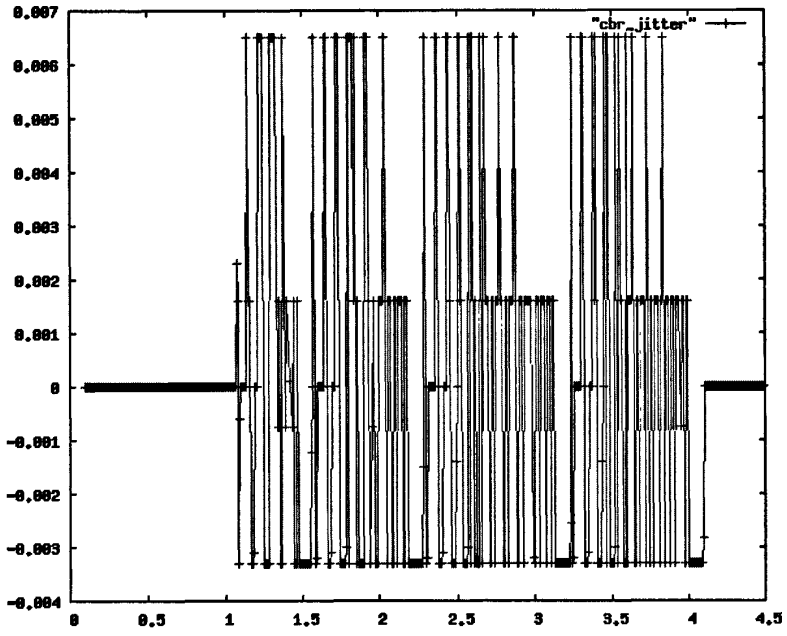

图 2-10 CBR 报文延迟时间变化

综上所述，利用 NS2 做仿真试验的完整步骤大致包括建立网络拓扑结构、建立或修改 NS2 中的网络协议或 c++类、编写 TCL 脚本文件、运行脚本文件、编写 awk 程序提取.tr 文件中的数据并得到目标数据文件、利用画图工具将目标数据文件转换为仿真图像。

2.5 小结

本章的内容是后面各章节的基础。本章主要介绍了基于 TCP 端网络拥塞控制的基本知识，包括：TCP 协议的相关知识，它的传输报文结构和连接机制；TCP 网络拥塞控制机制，它的主要过程：慢启动、拥塞避免、快速重传和快速恢复，并对各个阶段的实现过程做了较为详细的阐述。最后介绍了网络仿真软件 NS2。

第三章 基于 TCP 端的网络拥塞控制算法研究

3.1 引言

基于 TCP 端网络拥塞控制算法的研究一直是网络拥塞控制领域的重点研究方向，自从网络拥塞出现以来，人们一直期望能在 TCP 源头上来遏制此问题的发生，所以各种各样的 TCP 拥塞控制算法不断被提出。经典的 TCP 网络拥塞控制算法，包括 TCP Tahoe 算法、TCP Reno 算法、TCP New Reno 算法、TCP SACK 算法和 TCP Vegas 算法,都基于共同的算法机制，包括慢启动、拥塞避免、拥塞恢复、快速重传四个算法，只是在某些算法的具体实现机制、采用策略上有所不同。目前实际应用中 TCP Reno 算法是主流算法，其他算法可能在某些方面优于 TCP Reno 算法，但是并不适合实际应用。

本章分析了以上各种算法的数学模型，尤其是对各种算法的优缺点进行了详细研究，通过利用 NS2 仿真软件对各种算法进行了比较，分析了它们在网络吞吐量、丢包率方面的差异。通过比较可知，目前还不存在一种完美的拥塞控制算法能适应复杂的网络环境，对 TCP 端网络拥塞控制算法的研究仍任重道远。

3.2 经典 TCP 网络拥塞控制算法简介

TCP 协议经过多年的发展，有了多种具体的实现。TCP 拥塞控制中比较有代表性的方法有包括 TCP Tahoe，TCP Reno，TCP New Reno，TCP SACK 和 TCP Vegas，这些控制算法一般都包含慢启动、拥塞避免、拥塞恢复、快速重传四个过程。下面具体介绍下各个算法。

3.2.1 TCP Tahoe 算法

TCP Tahoe[25]模型是最早的TCP协议之一，它是由Jacobson在1988年提出的，其针对最初的没有拥塞控制机制的问题而提出来的，目的是在保持良好的用户通信吞吐量的同时，控制网络拥塞。

早期的TCP实现算法是基于积极响应并通过重传超时来重发丢失数据。当丢包时，TCP减小拥塞窗口，并重传被丢失的分组。当时的网络主要是有线网络。Jacobson观察到TCP报文段丢失的主要原因是网络阻塞，针对这种情况，Jacobson提出TCP Tahoe算法，它主要对原有协议进行了性能优化，其特点是，在正常情况下，通过重传计时器是否超时和是否收到重复确认信息这两种丢包检测机制来判断是否发生丢包。

具体到TCP Tahoe算法本身，它引入了指数阶回退（Exponential Back）的传输超时时长估计方法：规范了慢启动和拥塞避免算法，提出快速重传机制以有效检测报文丢失并合理地调整拥塞窗口。

指数阶回退的传输超时时长估计对重传超时时长的有效估计可以使TCP终端在及时发现报文丢失的同时减少超时误判的发生。一般说来，TCP发端在发出数据报文后会启动一个计时时长为RTO的定时器，若在定时器超时后仍没有收到ACK报文，则触发超时事件：TCP发端认为报文在传输过程中被丢弃，需要对其进行重新传输。

TCP Tahoe中引入了基于指数避退的超时时长估计方法：TCP Tahoe采用(3-1)和(3-2)对环回时间RTT的统计特性进行估计，并根据(3-3)计算RTO。这里RTT_E是环回时间的估计值，V_{RTT}是环回时间的标准偏差。

$$RTT_E \leftarrow \frac{7}{8}RTT_E + \frac{1}{8}RTT_E \tag{3-1}$$

$$V_{RTT} \leftarrow \frac{3}{4}V_{RTT} + \frac{1}{4}(|RTT_M - RTT_E| - V_{RTT}) \tag{3-2}$$

$$RTO = \beta(RTT_E + 4V_{RTT}) \tag{3-3}$$

需要说明的是式（3-3）中的因子并非常量，它根据通信的实际情况变化。如果网络中发生拥塞现象，为了缓解拥塞，TCP Tahoe终端在连续的超时发生时，将β值加倍（即将RTO倍增），直到β达到64；一旦TCP发送端收到ACK报文，则说明网络拥塞情况缓解，此时被设置为1，β恢复到拥塞发生之前的水平。

TCP Tahoe算法主要包括了慢启动、拥塞避免和快速重传3个主要部分，各个算法的详细实现过程已在前面陈述，不再赘述。为直观的理解的TCP Tahoe算法，给出了算法在不同阶段的拥塞窗口变化情况，如图3-1所示

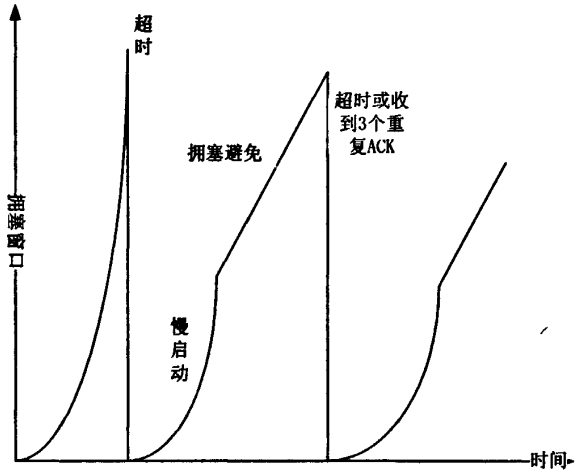

图3-1 TCP Tahoe拥塞窗口变化示意图

3.2.2 TCP Reno算法

TCP Reno[26]算法出现在1990年，它是由Jacobson对Tahoe提出的改进版本。TCP Reno算法在TCP Tahoe基础上增加了“快速恢复”算法,它是目前internet上应用最为广泛的TCP。

由于TCP Tahoe 算法在重传丢失报文后要将拥塞窗口重新设置为初始窗口大小，这样，TCP发端需要若干轮次才能恢复到报文丢失前的发送速率。这种特性造成网络发生拥塞后多个TCP之间出现“呼吸”效应，而这一过程中网络资源无法被充分利用。为了提高拥塞发生后TCP算法的吞吐量，TCP Reno算法在TCP Tahoe基础上增加了“快速恢复”算法。

在快速恢复阶段，当旧的数据包离开网络后，才能发送新的数据包进入网络，即同一时刻在网络中传输的数据包量是恒定的。如果发送方收到一个重复的ACK,则认为已经有一个数据包离开网络，于是将拥塞窗口加1。新算法避免了网络拥塞不够严重时采用慢启动算法而造成过大减小发送窗口尺寸的问题，从而避免了在快速重传之后通道为空的现象。

为了更直观的理解的TCP Reno算法，给出了该算法在不同阶段的拥塞窗口变化情况，如图3-2所示

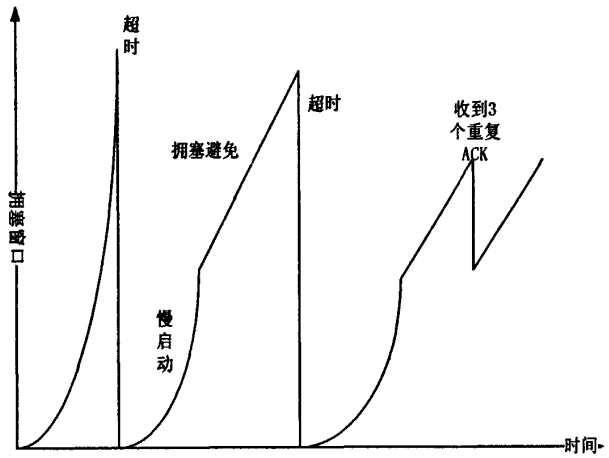

图3-2 TCP Reno算法拥塞窗口变化示意图

3.2.3 TCP New Reno 算法

虽然 TCP Reno 算法广泛应用于目前的互联网中，但是这种算法并非完美。TCP Reno 算法最大的缺点是仅考虑了每次发生拥塞时只丢失一个报文的情形，然而在实际应用中，一旦发生网络拥塞现象，路由器会丢失大量的报文，TCP 在一次拥塞中丢失多个报文的情形非常普遍。在这种情况下，采用 TCP Reno 算法的终端会多次将拥塞窗口和慢启动门限减半，结果是 TCP 的发送速率呈指数级降低，吞吐量急剧下降，造成网络性能的严重下降。

TCP Reno 性能恶化的原因在于网络中一次拥塞丢失的多个报文被错误的分析为发生了多次拥塞现象，进而过度的窗口减小导致了传输超时的发生。因此为了提高一次拥塞丢弃多个报文情形下 TCP 的性能，必须使 TCP 终端减少盲目削减发送窗口的行为。

Floyd 等人于 1996 年提出了 TCP NewReno[27]算法，对 Reno 中的“快速恢复/快速重传”进行了补充。在原“快速恢复”算法中，发送方收到一个不重复的应答后就退出“快速恢复”状态，而在 TCP NewReno 中，只有当所有报文都被应答后才退出“快速恢复”状态。

TCP NewReno 利用了部分应答（Partical ACK）在快速恢复阶段触发重传。Partical ACK 是当丢失发生时，确认了部分数据的确认包。例如报文段 N 被快速重传，报文段 M 是报文段 N 重传的时候发送端发送的最后一个报文段，如果只是报文段 N 丢失了，最后收到的 ACK 就是一个 Partical ACK，它给出的确认号应该是报文段 N+1 的第一个字节，这表明报文段 N+1 也需要重传。就这样，

NewReno 在每个 RTT 重传一个丢失的分组，直到发生多个报文段丢失的窗口中所有分组被重传，而不是等到重传超时。期间如果还有其他重复的 ACK 到来，则继续快速恢复算法，直到在快速恢复初始时所有未确认报文段都被确认。

NewReno 修正后的快速恢复/快速重传算法步骤如下：

(1)当收到第三个重复 ACK 并且发送端还没有处于快速恢复过程时，按式(3-4)设置 ssthresh,用变量"recover"记录传送的最大序列号。

$$ssthresh = \max(FlightSize / 2, 2 * MSS) \quad (3\text{-}4)$$

(2)重传丢失的段并设置 cwnd 为 ssthresh 加上 3*MSS,这将人为地按已经离开网络的报文段数目和接收端缓冲数据量来扩充拥塞窗口。

(3)对每个接收到的额外的重复 ACK，将 cwnd 增大 SMSS 字节，这将人为地扩充拥塞窗口以反映已离开网络的附加数据段。

(4)发送一个数据段，如果 cwnd 和接受端的通知窗口的值允许的话。

(5)当一个确认新数据的 ACK 到达时（此 ACK 可能是由步骤(2)中的重传引发的确认，或者是由稍后的一次重传引起的）。

如果这个 ACK 确认了所有数据（包括 recover 中所记录的序列号的数据），那么这个 ACK 确认了所有丢失段的原始传送和第三个重复 ACK 接收之间的段。设置 cwnd 为 min(ssthresh,FlightSize+MSS)或者 ssthresh，这里的 ssthresh 是步骤 1 中设置的值；这称为"deflating"窗口（注意到步骤 1 中的 FlightSize 指的是当进入快速恢复时步骤 1 中发送的数据量，步骤(5)中的 FlightSize 指的是当退出快速恢复时发送的数据量）。退出快速恢复过程。

如果这个 ACK 没有确认到所有数据（包括 recover 中所记录的序列号的数据），就产生一个部分 ACK。在这种情况下，重传递一个没有确认的数据段。按确认的新数据量来减小拥塞窗口，然后加回一个 MSS 并发送一个新数据段，如果 cwnd 的新值允许的话。这个"部分窗口缩减"试图确定这一点，当快速恢复最终结束时，大约 ssthresh 数量的数据还在向网络中传输。不要退出快速恢复过程(也就是说，如果任何重复 ACK 随后到达，执行上面的步骤(3)和步骤(4))，对在快速恢复期间第一个到达的部分 ACK,也要重设重传定时器。

3.2.4 TCP SACK 算法

TCP SACK[28]算法是 M.Mathis 等人于 1995 年提出的，它是对 TCP Reno 算

法的扩展，它在 TCP Reno 算法的基础上增加了 SACK（Selective Acknowledgments）选项，可以有针对性的重发丢失的报文段。

前面我们讲述的算法都没有 SACK 选项，因此若一个窗口丢失了多个数据包，发送端要么只重传一个丢失的数据包，要么连同已经正确接收的报文一起重传。TCP Reno 算法就采用了第一种重传策略，而 TCP Tahoe 则采用了第二种重传策略。TCP SACK 算法选用了 SACK 选项，在数据传输过程中，若接收端缓存队列中出现序号不连续的数据，就向源端发送带有 SACK 选项的重复 ACK，源端在收到这些确认后可以清楚的知道哪些数据被正确的接收，哪些数据包被丢失，从而有选择性的进行重发丢失的数据包，这样可以提高 TCP 的性能，这是目前公认的最好的 ACK 反馈机制之一。

SACK 选项使用“管道”（pipe）变量表示在发送路径上损失的数据报数量。并在发送端保存一个变量 scoreboard，用来记录上一个 SACK 选项中记录的确认报文。SACK 选项的结构如图 3-3 所示

种类=5	长度
第一个 SACK 块的左边界	
第一个 SACK 块的右边界	
……	
第 N 个 SACK 块的左边界	
第 N 个 SACK 块的右边界	

图 3-3 SACK 选项结构图

TCP 选项中包含有 40 个字节，而一个 SACK 块需要 8 个字节，除去时间戳选项，TCP SACK 算法中最多包含 3 个 SACK 块，用于标志目的端最近收到的连续 3 个数据包。在这 3 个连续的数据包中，末尾序列号为“左边界—1”的数据段和初始序列号为右边界的数据段均为没有接收到的数据，需要重传。TCP SACK 改进的快速恢复算法如下：

(1)若 piple 的数值小于拥塞窗口大小，则发送端发送新数据包或重传丢失的数据包，pipe 的数值相应的增加 1；发送端收到一个确认 ACK 后，pipe 的数值减 1。

(2)当发送端发送数据包之前，它首先参考 scoreboard 来确定是否需要重传丢失的数据包，若不需要重传，则直接发送新数据包。

(3)若发送端接收到 PACK 时，表示两个数据包离开了 pipe，一个是重传数据包，一个是丢失的数据包裹，pipe 的数值相应的减 2。

(4)若发送端收到 RACK 后结束快速恢复算法。

3.2.5 TCP Vegas 算法

TCP Vegas[29]算法是 L.S.Brakmo 等人于 1994 年提出的。与前面讲述的算法相比，TCP Vegas 算法是一种与众不同的算法，它创新性的采用了基于时间值的拥塞避免机制，并采用新的重传机制和扩展的“慢启动”机制，与目前流行的 Reno 算法相比，TCP Vegas 算法具有更高的数据吞吐量和更强的稳定性。

TCP Vegas 算法是通过观察回路响应延时 RTT 的变化情况来控制拥塞窗口的大小。它首先在理论上估算 RTT 的数值，然后用这个估算值和实际观测的 RTT 相比较，若实际的观测值大于估算值，则算法就认为网络拥塞发生，开始减小拥塞窗口；若实际的观测值小于估算数值，则认为网络拥塞正在解除，就相应的增加拥塞窗口的大小。

传统的 TCP 端的拥塞控制算法都是在拥塞发生之后采取措施来解除拥塞状态，它们都不能提前预测网络拥塞的发生，从而能在拥塞发生之前采取措施防止拥塞的发生。TCP Vegas 算法采用了基于时间的机制，通过在网络中保持一定的额外数据包，尽力维护网络的高吞吐量状态而又避免触发网络拥塞问题。

TCP Vegas 算法采用的重传策略也与传统算法不同[30]，TCP Vegas 不是等到连续收到 3 个 ACK 后才进行重传，而是当收到第一个重传 ACK 后就比较数据包的发出时间和当前时间，然后决定是否重发，这样可以显著提高响应速度，也减小了不必要的冗余数据，TCP Vegas 算法具有丢包率低，窗口稳定性高，负荷效率高的优点。

3.3 TCP 拥塞控制算法仿真分析

前面讲述了传统的 TCP 网络拥塞控制算法，虽然各个算法的实现机制不一样，但是它们都对网络的稳定性起到了关键作用。然而就算法本身而言，它们又都存在自己的不足之处。

TCP Tahoe 算法是最早的 TCP 拥塞控制算法之一，算法性能稳定，由于算法本身缺少“快速恢复”阶段，会出现暴发文现象，已经很难适应目前网络发展的要求。

TCP Reno 算法是目前网络的主流算法，通过前面算法机制的描述，它可以用数学形式表示如下：

```
Initial();                          //发送窗口(win); 源端预设的发送窗口大小（awin）
Win=min(cwnd,awin),cwnd=1;                          // ssthresh=64KB(确省值)
If (cwnd<sstresh)
        cwnd=cwnd+1;                                                    //慢启动
Else
        cwnd=cwnd+1/cwnd;                                               //拥塞避免
Relay timeout;
Ssthresh=max (2, min (cwnd/2, awin));
Cwnd=1;
```

TCP Reno 算法包含了快速恢复阶段，与 TCP Tahoe 算法相比性能有很大的提高，但是随着网络带宽延迟的持续增长，TCP Reno 算法越发暴露出以下几个缺点[31]，这也是 TCP Reno 算法在高速增长延迟网络中性能下降的主要原因：

(1)在 TCP 发送窗口日益增大的情况下，每个 RTT 内只增加一个报文显得很慢，而每丢弃一个报文却以减半方式减小拥塞窗口又显得过于剧烈。

(2)TCP Reno 算法采用了二元拥塞控制信号，窗口大小的不断更新将导致周期性振荡，产生大的延迟波动，不能有效利用带宽。

(3)理论上维持一个大的平均拥塞窗口需要极小的均衡丢失概率，这在实际网络中很难做到。

(4)不能公平的分配带宽。

TCP NewReno 算法在 TCP Reno 算法的基础上考虑了一个发送窗口内多个报文丢失的情况，在原“快速恢复”算法中，发送方收到一个不重复的应答后就退出“快速恢复”状态，而在 TCP NewReno 中，只有当所有报文都被应答后才退出“快速恢复”状态。TCP NewReno 算法的应用前提是用于不支持 SACK 的 TCP 实现，效果比 TCP Reno 算法好一些，但是现在不支持 SACK 的 TCP 实现已经很少了，因此这种算法的实际意义不是很大。

TCP SACK 算法采用了 SACK 选项，可以针对性的重发丢失的数据包，提高了数据传输效率。但是 TCP SACK 算法有时不能有效利用网络所提供的带宽，特别是在那些传输通信量小于带宽—延迟积（即网络的带宽与往返时间的乘积，这个乘积就是发送端到接收端的管道的容量）的链路中。此外，在高带宽，大延

迟网络（如卫星网络）中，TCP SACK 算法表现不理想。

综合来讲，这几种算法在不同网络中有着差异的表现，而没有一种算法可以在不同的环境中均获得最佳性能。下面通过仿真试验，来进一步分析各个算法的性能和缺点：

3.3.1 单个算法分析

图 3-4 是单个算法网络拓扑结构图。其中节点 n0 代表发送端，n1 代表中间缓存，n2 代表接收端，n0 与 n1、n1 与 n2 之间都由链路所连接，设置 n0 与 n1 之间带宽容量为 1Mb,传输延迟为 10ms，n1 与 n2 之间带宽容量为 50Kb, 传输延迟为 10ms。下面我们通过改变 n1 与 n2 之间的最大缓存来分析各种算法的吞吐量和丢包量。

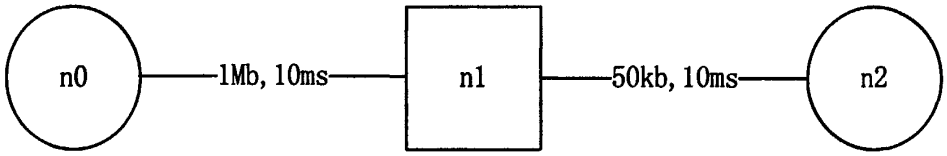

图 3-4 单个算法网络拓扑结构图

表 3-1 各个算法性能比较

n1,n2 缓存（包）	Tahoe			Reno			New Reno			SACK			Vegas		
	吞吐量	丢包率	丢包量	吞吐量	丢包率	丢包量	吞吐量	丢包率	丢包量	吞吐量	丢包率	丢包量	吞吐量	丢包率	丢包量
20	65	1.5%	1	70	1.4%	1	70	1.4%	1	70	1.4%	1	62	0%	0
10	55	20%	11	44	23%	10	65	15%	10	68	16%	11	62	0%	0
5	57	12%	7	56	16%	9	64	14%	9	66	5.6%	9	62	0%	0

从表 3-1 中可以看出当缓存包数量为 20 时，各种算法的吞吐量相差不多。除 TCP Vegas 算法外，其余算法均存在丢包现象。当缓存减小时，除 Reno 算法外，其余算法的吞吐量都未发生剧烈变化，之所以会出现这种情况，是由 Reno 算法本身的机制造成的，若源端口丢失 2 个数据包，Reno 算法就要连续启动快速重传和快速恢复机制，若丢失数据包达到 3 个或 4 个，则必须等待重传定时器超时来恢复数据传输。所以，当连续丢包时，Reno 算法恢复数据传输所花费的时间远大于其它算法所用的时间，这也就使得它的吞吐量在这种情况下是最低的。注意到 TCP Vegas 算法的丢包量一直为 0，这是由于算法本身的机制使得它可以预测网络带宽，预防网络拥塞的发生。

3.3.2 算法综合比较

多个算法比较使用的拓扑图[32]如图 3-5 所示,其中 n0,n1,n2,n3 与 n4 之间的带宽均为 1Mb,时延是 10ms，最大发送窗口设为 20 个数据包，程序运行时间为 10 秒，当 n4 与 n5 之间的最大缓存变化时，四种算法的不同仿真结果如下：

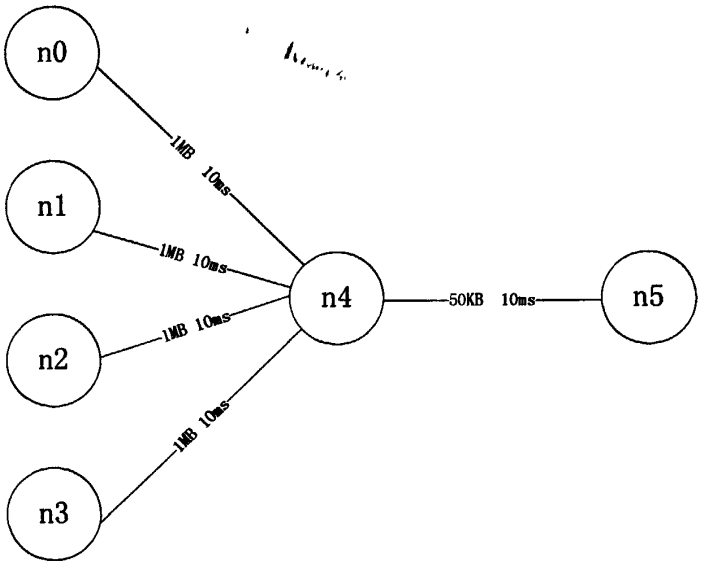

图 3-5 多个算法网络拓扑结构图

表 3-2 多种算法性能比较

n4,n5 缓存（包）	Tahoe			Reno			New Reno			SACK			Vegas		
	吞吐量	丢包率	丢包量	吞吐量	丢包率	丢包量	吞吐量	丢包率	丢包量	吞吐量	丢包率	丢包量	吞吐量	丢包率	丢包量
10	20	30%	6	12	33%	4	12	33%	4	25	20%	5	10	50%	5
20	21	24%	5	24	25%	6	24	8%	2	24	21%	5	10	20%	2
40	42	2%	1	124	0	0	126	0	0	27	0	0	11	0	0

从表 3-2 中可以看出在缓存较小的时候,Tahoe,Reno NewReno 和 SACK 的表现相差不多，而 Vegas 的吞吐量最少，远不及之前算法运行的表现，这是由于 Vegas 的算法机制和其他算法的机制不一致，导致其在争夺带宽方面存在不公平所致。随着缓存越来越大，例如达到 40 以上时候，五种算法都没有丢包，相对而言，Reno 和 New Reno 的吞吐量比较大。

综合单机算法和多个算法的仿真来看，在连续丢包的情况下，Reno 类算法表现是几种算法中最差劲的。但在缓存较高时，Reno 类算法的吞吐量最高。SACK 的表现比较稳定，一直保持在一定的水平，但在缓存较大时候竞争力不及 Reno 类算法。Vegas 的算法由于本身机制的问题，带宽竞争力最差，这也是它无法广泛应用的重要原因之一。

3.4 小节

本章介绍了几种基于 TCP 端的经典网络拥塞控制算法，包括 TCP Tahoe、TCP Reno、TCP New Reno、TCP SACK 和 TCP Vegas 等算法，详细阐述了各种算法的实现机制，分析了每种算法的优缺点，并进行了仿真试验。

通过理论阐述和仿真分析，可知目前几种基于 TCP 端的网络拥塞控制算法都不完美，每种算法机制都存在自身的缺陷，虽然新提出的算法在旧算法的基础上作了进一步改进，性能上都有一定提高，但是面对日益复杂的网络环境，每种算法无法展现出最好的性能。TCP Vegas 算法虽然在理论上证明了自己的优越性，但是算法实现机制的保守以及竞争带宽的劣势导致它在与其他算法共存时的性能严重下降，这也是其不能广泛应用的一个重要原因。在下一章中，我们将会对 TCP Vegas 算法做深入的讨论。总之，伴随着因特网业务量的增多和复杂性的加大，试图找到一个完美的拥塞控制算法来控制网络拥塞仍是一项艰巨的任务。

第四章 TCP Vegas 拥塞控制算法的研究和改进

4.1 引言

TCP Vegas 算法是一种基于测量技术的拥塞控制算法，它在算法机制上和传统的拥塞控制算法有很大不同，它从本质上改变了传统的被动式拥塞控制算法的实现原理，采取的主动防御实现策略为 TCP 拥塞控制算法的发展指明了新的道路。由于 TCP Vegas 算法的广阔发展前景，本章对 TCP Vegas 算法进行了详细的分析，从它的算法机制到算法各方面的优缺点进行了深入研究。

网络非对称性问题是困扰 TCP 端拥塞控制算法的主要问题，尤其是对准确性要求较高的 TCP Vegas 算法的影响更加突出。本章首先分析了 TCP Vegas 算法中的网络非对称性问题，产生原因是原算法在计算期望速率和实际速率时，使用的 BaseRTT 和 RTT 都是包括链路来回往返时间的,这样就把反向链路的情况包括在算法之中了,用数学分析拥塞避免阶段的算法,得出反向排队时间的增加,会导致拥塞窗口减小，使得网络吞吐量的下降。我们创新性的将反向排队时间从链路往返时间中排除出去,经数学理论验证忽略反向排队时间不会对正向链路产生很大影响,仿真试验也证实了改进算法能有效解决降低由于反向链路发生拥塞导致的吞吐量下降问题。

算法之间的兼容性同样是阻碍 TCP Vegas 算法应用的主要问题之一。TCP Reno 算法是目前网络中应用的主要算法，但 TCP Vegas 算法与 TCP Reno 算法并不兼容，本章针对该问题进行了深入分析，得出造成不兼容性的主要原因是由两种算法不同的实现机制。为解决这个问题,我们分析了 α 和 β 对算法兼容性的影响，而 TCP Vegas-A 算法将 α 和 β 参数由静态变为动态，有效的解决了兼容性问题。然而经过分析 TCP Vegas-A 算法并非完美，在仿真分析基础上我们提出了 TCP NewVegas-A 算法,仿真证明该算法在吞吐量和稳定性上都超过了 TCP Vegas-A 算法，并且在和 TCP Reno 算法共存时，该算法的吞吐量明显大于 TCP Reno 算法，这也大大提高了新算法可应用性。

本章主要研究了 TCP Vegas 算法，针对其网络非对称性问题和算法兼容性问题进行了分析，分别提出了改进算法，仿真对比也证实了改进算法的可行性。本章最后对两种问题的各自解决算法进行了融合性分析，力图在一个算法中解决上述两个问题，提出的 TCP NewVagas 算法也经过仿真试验的验证，可以解决 TCP

Vegas 算法的非对称性问题和算法兼容性问题。

4.2 TCP Vegas 拥塞控制算法详述

在第三章中,已经简单介绍了 TCP Vegas 算法，TCP Vegas 算法是一种基于测量技术的网络拥塞控制算法，它在三个方面对传统的 TCP 拥塞控制算法作了改进，这三种新机制分别是扩展的“慢启动”机制、新的拥塞避免机制和新的重传机制。这三种新机制的使用大大提高了 TCP Vegas 算法的性能，特别是在数据的吞吐量上，它比目前流行的 Reno 算法提高了 37%~71%[33]。下面，分别讲述下这三种新的算法机制。

新的慢启动机制。在传统的 TCP 端拥塞控制算法中，算法在链路连接之初使用慢启动机制。当一个网络连接刚初始化的时候， TCP 使用了一个很小的拥塞窗口 cwnd 来探测网络的使用情况，以保证不会把太多的数据分组发送进一个已拥塞的环境。一般 TCP 只被允许发送一个报文段，然后就必须等待确认再传输第 2 个报文段。随着每个确认帧（ACK）的到达，cwnd 的值就以指数规律增长到某个最大值为止。在链路刚建立的时候，没有有关可用带宽的预知信息，因为在每个 RTT 时间内拥塞窗口都按照指数级进行增长，那么会很快达到拥塞阀值，吞吐量很低。

为了找出一个不会导致数据包丢失的可用带宽，TCP Vegas算法要求每隔一个RTT才进行一次指数增长，而之间的RTT，拥塞窗口保持不变，并比较实际吞吐量和期望的吞吐量。这种新的慢启动的机制，增长了慢启动的周期，可以有效的找出合适的带宽，对网络拥塞的发生也起到了预防的作用。

新的拥塞避免机制[34]。TCP Vegas 算法最大的改变之处在于提出了基于回路相应延时 RTT 的拥塞避免机制。传统的 TCP 端网络拥塞控制算法的最大弊端在于它们只能被动的对拥塞做出反应，而不能对网络拥塞进行预防。原因很简单，传统算法都是将丢包作为发生网络拥塞的唯一标志，如果数据包在传输过程中丢失就认为网络发生网络拥塞现象，进而采取拥塞避免措施，减小拥塞窗口大小将网络从拥塞状态中恢复。网络恢复到正常之后，又采用慢启动机制，逐步增大拥塞窗口，耗尽连接通路上的缓存，最后又导致网络拥塞的发生。可以说传统的 TCP 拥塞控制算法机制就好像一个“恶性循环”过程，不断引起拥塞，又不断消除拥塞状态。

而 TCP Vegas 算法通过比较实际吞吐量和期望吞吐量来调整拥塞窗口大小。它的目的就是在保持一定量的数据包在网络中，如果源端了过量的额外数据，就认为会导致网络拥塞；反之，如果源端发送的额外数据包过少，就认为网络带宽没有完全利用。TCP Vegas 算法的拥塞避免算法可表示如下：

$$Expected = cwnd / BaseRTT; \tag{4-1}$$

$$Actual = cwnd / RTT; \tag{4-2}$$

$$Diff = Expected - Actual \tag{4-3}$$

$$cwnd(t+\Delta t) = \begin{cases} cwnd(t)+1; Diff < \alpha / BaseRTT \\ cwnd(t); \alpha / BaseRTT < Diff < \beta / BaseRTT \\ cwnd(t)-1; Diff > \beta / BaseRTT \end{cases} \tag{4-4}$$

(4-1)式表示计算期望的吞吐量，用 Expected 来表示这个值。其中的 BaseRTT 是所有观测回路响应时间的最小数值。它一般是建立连接后所发送的第一个数据包的 RTT，此时路由器的缓存队列长度还没有因为该连接产生的流量而增加。cwnd 表示目前的拥塞窗口大小，即发送的数据包数。(4-2)式表示计算实际的吞吐量，用 Actual 表示这个数值。RTT 是通过观测回路得到的响应时间。(4-3)式用 Diff 表示计算期望吞吐量和实际吞吐量的差值，这个值只能是正的或者零。(4-4)式是拥塞避免算法中的核心，它主要是由 t 时刻的拥塞窗口大小来推测 t+△t 时刻的拥塞窗口大小。这里，我们预先定义了两个阈值 α 和 β（α<β）这两个阈值大体对应了网络上存在过少或过多的额外数据量。当 Diff<α/BaseRTT 时，Vegas 在下一个 RTT 中线性增大拥塞窗口；当 Diff>β/BaseRTT 时，Vegas 在下一个 RTT 中线性减小拥塞窗口；当 α/BaseRTT<Diff<β/BaseRTT 时，Vegas 在下一个 RTT 中将拥塞窗口保持不变。

理论上来讲，如果我们实际得出的吞吐量和期望的吞吐量相差越大，那么网络拥塞发生的几率就越大，这就意味着发送速率要降低，阈值 β 触发了这种降低；相反，如果实际吞吐量离期望吞吐量越近，那么意味着网络中的带宽没有被完全利用，说明源端口的发送速率要提高，而阈值 β 触发了这种提升。总之，算法的目的是保持 α~β 的额外数据在网络上。

α 和 β 阈值是由数据包的数量决定的。在实际应用中一般设 α=1，β=3，可以解释为在连接中希望至少保留一个包，至多保留三个包。设 α=1 说明至少保留

一个数据包在网络中，因为在实际应用中经常发生链路上其他连接总体速率下降的情况，此时该连接可以利用这个额外的数据包来保证链路带宽的使用，而不必等待一个 RTT 延迟。而一般在线性增长算法中这种等待是必需的。设 β=3 说明最多保留三个数据包在网络中,因为如果过多的数据包在网络中存在,不仅占用网络带宽缓存在容量,而且影响数据传输速率,造成网络资源的浪费。使用 α 到 β 这个区域起到震荡阻尼的作用，由于 TCP Vegas 算法中使用的估算可用带宽的机制没有故意引起数据包的丢失，所以这种机制为 TCP Vegas 赢得较高的吞吐量和传输效率。

新的重传机制。传统的重传机制是利用重传定时器为来回时程计时的机制进行数据包重复，例如 TCP Reno 算法的重传定时器时钟是每 500ms 滴答一次，仅仅在每次滴答的时候进行超时检测，它使用了一种粗粒度时钟。然而试验表明，这种粗粒度时钟造成从开始发送一个丢失数据包到超时重发这个数据包所经历的时间大于必要的时间。

相对于 TCP Reno 算法的重传机制，TCP Vegas 算法在以下几个方面作了改进，首先对发送的每个数据包采取了计时的措施，即当发送一个数据包的时候，读取并记录系统时间，当一个确认到达时，再次读取系统时钟并以该时间和先前记录下的时间差计算 RTT。然后，TCP Vegas 采用这个精确的 RTT 估计值在下面两种情形下决定是否重发：

第一种情形是接收到非重复的 ACK 时，如果它是重发之后的第一或第二个确认，则计算发送首个未被确认的数据包距离目前时间的间隔，然后将这个时间间隔与超时值做比较，如果大于超时值，则 TCP Vegas 重发该数据包，这样做能够避免一些新的重复 ACK 的到来。而且为了避免在一个 RTT 内由于多个数据报包丢失而触发快速重传，拥塞窗口只是在第一次重传时被缩减。

第二种情形是接收到重复的 ACK 时，计算记录时间距离目前时间的间隔，然后将这个时间间隔与超时值做比较，如果大于超时值，则 TCP Vegas 立即重发该数据包，而不会等待第三个重复 ACK 的到来。在实际应用中，由于很多传输数据量很大，很多发送方根本不可能收到三个重复 ACK，所以 TCP Vegas 会根据相关时间的比较决定是否重发该数据包。并且当源端收到重传数据包的应答时，TCP Vegas 是以 3/4 的比例缩减拥塞窗口，而不是传统算法的拥塞窗口减半。

在 TCP Reno 算法中，如果一个 RTT 内发生多个丢包，那么会引起拥塞窗口的多次减小。而在 TCP Vegas 中只有在最近一次窗口缩减之后，已经发送了需要重新发送的数据包的情况下才会缩减拥塞窗口。

4.3 TCP Vegas 拥塞控制算法缺点分析

基于TCP端网络拥塞控制算法的研究是网络拥塞领域的研究热点之一，但设计一种合理稳定，竞争公平，可实施的算法存在一定的难度。TCP Vegas算法虽然是一种基于回路传输响应延时的新颖算法，在理论上它已经被证明为是一种吞吐量更高，稳定性更好的拥塞控制算法，但是自从它提出以来，始终未能广泛的应用于目前的互联网中，这是由于算法本身存在着几个方面的缺陷。

(1)TCP Vegas算法的兼容性问题[35]。目前，TCP Reno算法是网络中应用最广泛的算法，如果我们想用TCP Vegas算法替换目前的TCP Reno算法，那么必须首先实施局部的推广，然后逐步替代，最后达到完全覆盖的目的。这是一种新的机制取代旧机制的必由之路。但是这两种算法之间存在着不兼容性问题，导致TCP Vegas无法广泛应用。

TCP Vegas算法是一种采用了主动的拥塞避免机制，在发生丢包以前会减小拥塞窗口的大小，当网络中有可用带宽之时，会利用额外的数据包占用空余带宽，但是TCP Vegas算法的目标是将网络带宽维持在一个稳定的水平，所以它不会持续的扩展自己的拥塞窗口。而TCP Reno算法采用了反应式的拥塞避免机制，它持续增加自己的窗口，直到网络过载产生频繁的丢包来保证有效利用网络资源，这必然产生周期性的窗口振荡和缓存区的满队列特性。因此，当两种算法共存时，Reno会窃取Vegas的带宽。这种算法之间的不兼容性限制了TCP Vegas算法的应用。

(2)网络的非对称性问题[36]。TCP Vegas算法是一种基于时间的算法，因此RTT准确性至关重要，粗略测量的RTT可能导致对cwnd的粗略调整。TCP Vegas算法只是考虑了在数据传送方向上发生网络拥塞而采取的措施。如果拥塞发生在回路方向，也即ACK方向，算法用样会采取拥塞避免措施，这样会引起对实际吞吐量的过低估计，引起拥塞窗口不必要的下降。而理想状态是返回路径上的拥塞不应该影响数据传输方向的网络吞吐量。这就要求控制机制必须能够辨认拥塞是发生在哪个方向上，进而采取正确的措施保证网络的稳定性。目前的 Reno算法和

Vegas算法都不能解决这个问题，在遇到返回路径拥塞的情况下性能都严重退化。

(3)网络持续拥塞问题。如前所述，Vegas以BaseRTT作为路由的传播时延，BaseRTT的值直接决定了Vegas算法的性能，假设一个新的连接在网络拥塞时建立，此时，由于存在其他连接在网络中的包积累量，这些包积累量将造成较大的排队时延，使新的连接将经历比实际期望的BaseRTT大很多的往返延迟RTT，也就是估计的BaseRTT远远大于实际的BaseRTT，这样新的Vegas连接将设置较大的cwnd以保持连接在路由缓存中的占有量处于规定值之间，这样的情形在每个新的连接重复。这样很有可能导致整个系统的持续和永久拥塞。当然Reno算法也会遇到这种情况，但是由于Vegas算法采用的主动的拥塞避免机制，会导致它更容易遇到这种情况。

(4)算法公平问题[37]。TCP Vegas是根据实际吞吐量和期望吞吐量的差值来调整拥塞窗口大小，目的是保持额外数据在阈值α和β之间。α和β之间的范围导致赢得连接吞吐量的不确定性；而且，期望吞吐量的不正确计算也可能导致不公平性。在实际情况中，当新的连接建立时开始发送数据而其他连接仍然活跃时，就会引起固定时延的过高估计，这将影响BaseRTT的计算，而期望吞吐量的计算是基于BaseRTT的，所以这将会直接导致Vegas在连接带宽中的不公平性分配。

(5)线路变更问题。当网络环境改变时也会对TCP Vegas算法造成影响，例如当一个连接的路径被转换器改变时，那么没有来自转换器的信号，端主机不能直接发现它。如果新路径有一个较短的传输延时，则很有可能一些包将经历较短的回路响应延时并更新BaseRTT，这对Vegas来说不会引起任何严重的问题；另一方面，如果连接的新路径有较长的BaseRTT,那么算法将不能判断新BaseRTT的增加是由于路径改变造成的，而错误的认为是网络拥塞导致了回路响应延时BaseRTT的增加，从而减小了网络拥塞窗口，而这和源端该做的恰恰相反。

总之，TCP Vegas算法目前还不具备大规模应用的条件，它本身还存一定的缺陷。当然这些问题不是完全不能解决的，例如上面提到的问题中，有相当一部分是由于不能精确估计回路响应延时BaseRTT而造成的，如果我们可以得到一种精确的计算方法，那么这些问题将很容易解决。

4.4 TCP Vegas 算法的网络非对称性问题

从 4.2 节我们知道 TCP Vegas 算法机制本身存在不少缺陷，而其中的相当一

部分问题是由于对最小回路响应时间 BaseRTT 的不精确计算造成的。如果我们能够提出一种精确计算 BaseRTT 的方法，那么 TCP Vegas 算法的网络持续拥塞、算法不公平、线路变更等问题都可以得到很好的解决。在实际应用中网络的非对称性问题对 TCP Vegas 算法影响较严重，而且由于算法本身的机制又使得这种问题容易发生，因此我们试图找到一种合适的算法对 TCP Vegas 进行改进，希望能够消除当反向链路发生网络拥塞时对算法造成的影响。

4.4.1 针对网络非对称性做出的改进

由前面介绍可知，TCP Vegas 算法中 RTT 代表的是数据报在整个链路上的往返时间，它包括正向链路和反向链路传输时间的总和，这也说明如果在正向链路没有拥塞但在反向链路上发生拥塞，必然会使得往返时间 RTT 变大，从而导致计算所得的实际速率 Actual 变小，引起 Actual 和 Expected 的差值与 BaseRTT 的乘积超过上限 β，这时 Vegas 算法会错误的认为是自身发送速率过快造成的网络拥塞，从而减小发送端的拥塞窗口大小，降低发送速率。但是由于正向链路并未发生拥塞，根本不必采拥塞避免措施，这就造成吞吐量无谓的下降。

TCP Vegas 算法之所以会出现上述情况，原因是在计算期望速率和实际速率时，使用的 BaseRTT 和 RTT 都是包括来回链路往返时间的，这样就不自觉的把反向链路的情况包括在算法之中了。 通常一个 RTT 可以细分为 4 个部分，分别是正向固定延迟时间（用 T1 表示），反向固定延迟时间（用 T2 表示），正向排队时间（用 T3 表示）,反向排队时间（用 T4 表示）。按照原来的 TCP Vegas 算法，拥塞避免阶段的算法表示如下：

$$BaseRTT = T1 + T2 \tag{4-5}$$

$$RTT = T1 + T2 + T3 + T4 \tag{4-6}$$

$$Expected = cwnd / BaseRTT = cwnd / (T1 + T2) \tag{4-7}$$

$$Actual = cwnd / RTT = cwnd / (T1 + T2 + T3 + T4) \tag{4-8}$$

$$
\begin{aligned}
&\mathrm{Diff} \\
&= \mathrm{Expected} - \mathrm{Actual} \\
&= \mathrm{cwnd}\left(\frac{1}{T1+T2} - \frac{1}{T1+T2+T3+T4}\right) \\
&= \mathrm{cwnd}\left(\frac{T3+T4}{(T1+T2)(T1+T2+T3+T4)}\right) \\
&= \mathrm{cwnd}\left(\frac{T3+T4}{(T1+T2)^2+(T1+T2)(T3+T4)}\right)
\end{aligned}
\tag{4-9}
$$

$$
\begin{aligned}
&\mathrm{Diff} * \mathrm{BaseRTT} \\
&= \mathrm{cwnd}\left(\frac{T3+T4}{(T1+T2)^2+(T1+T2)(T3+T4)}\right) * (T1+T2) \\
&= \mathrm{cwnd}\left(\frac{T3+T4}{(T1+T2)+(T3+T4)}\right) \\
&= \mathrm{cwnd}\,\frac{1}{1+\dfrac{T1+T2}{T3+T4}}
\end{aligned}
\tag{4-10}
$$

$$
\begin{aligned}
\mathrm{cwnd}(t+\Delta t) &= \begin{cases} \mathrm{cwnd}(t)+1; \mathrm{Diff} < \alpha / \mathrm{BaseRTT} \\ \mathrm{cwnd}(t); \alpha / \mathrm{BaseRTT} < \mathrm{Diff} < \beta / \mathrm{BaseRTT} \\ \mathrm{cwnd}(t)-1; \mathrm{Diff} > \beta / \mathrm{BaseRTT} \end{cases} \\
&= \begin{cases} \mathrm{cwnd}(t)+1; \mathrm{Diff} * \mathrm{BaseRTT} < \alpha \\ \mathrm{cwnd}(t); \alpha < \mathrm{Diff} * \mathrm{BaseRTT} < \beta \\ \mathrm{cwnd}(t)-1; \mathrm{Diff} * \mathrm{BaseRTT} > \beta \end{cases} \\
&= \begin{cases} \mathrm{cwnd}(t)+1; \dfrac{\mathrm{cwnd}}{1+\dfrac{T1+T2}{T3+T4}} < \alpha \\ \mathrm{cwnd}(t); \alpha < \dfrac{\mathrm{cwnd}}{1+\dfrac{T1+T2}{T3+T4}} < \beta \\ \mathrm{cwnd}(t)-1; \dfrac{\mathrm{cwnd}}{1+\dfrac{T1+T2}{T3+T4}} > \beta \end{cases}
\end{aligned}
\tag{4-11}
$$

从上面的算法中可以看出当反向链路发生拥塞时，会使反向排队时间 T4 增

加，使得$\frac{T1+T2}{T3+T4}$变小，导致$\frac{cwnd}{1+\frac{T1+T2}{T3+T4}}$变大，从而使得$\frac{cwnd}{1+\frac{T1+T2}{T3+T4}}$大于β，算法相应采取拥塞避免措施，会将拥塞窗口减小，导致网络吞吐量的下降。

从上面的分析可知道为了消除反向链路的影响，简单直接的做法就是在计算期望速率和实际速率的时候消除反向排队时间 T4 对算法造成的影响，那么修改后的拥塞避免算法可表示如下：

$$BaseRTT = T1 + T2 \tag{4-12}$$

$$RTT = T1 + T2 + T3 \tag{4-13}$$

$$Excepted = cwnd / BaseRTT = cwnd / (T1 + T2) \tag{4-14}$$

$$Actual = cwnd / RTT = cwnd / (T1 + T2 + T3) \tag{4-15}$$

$$\begin{aligned} Diff &= Expected - Actual \\ &= cwnd\left(\frac{1}{T1+T2} - \frac{1}{T1+T2+T3}\right) \\ &= cwnd\left(\frac{T3}{(T1+T2)(T1+T2+T3)}\right) \\ &= cwnd\left(\frac{T3}{(T1+T2)^2 + (T1+T2)T3}\right) \end{aligned} \tag{4-16}$$

$$\begin{aligned} Diff * BaseRTT &= cwnd\frac{T3}{(T1+T2)^2 + (T1+T2)T3} * (T1+T2) \\ &= cwnd\left(\frac{T3}{(T1+T2)+T3}\right) \\ &= cwnd\frac{1}{1+\frac{T1+T2}{T3}} \end{aligned} \tag{4-17}$$

$$
\begin{aligned}
cwnd(t+\Delta t) &= \begin{cases} cwnd(t)+1; Diff < \alpha / BaseRTT \\ cwnd(t); \alpha / BaseRTT < Diff < \beta / BaseRTT \\ cwnd(t)-1; Diff > \beta / BaseRTT \end{cases} \\
&= \begin{cases} cwnd(t)+1; Diff * BaseRTT < \alpha \\ cwnd(t); \alpha < Diff * BaseRTT < \beta \\ cwnd(t)-1; Diff * BaseRTT > \beta \end{cases} \\
&= \begin{cases} cwnd(t)+1; \dfrac{cwnd}{1+\dfrac{T1+T2}{T3}} < \alpha \\ cwnd(t); \alpha < \dfrac{cwnd}{1+\dfrac{T1+T2}{T3}} < \beta \\ cwnd(t)-1; \dfrac{cwnd}{1+\dfrac{T1+T2}{T3}} > \beta \end{cases}
\end{aligned}
\tag{4-18}
$$

经上面的数学推导可知，采用修改的拥塞避免算法后即使反向链路发生拥塞，由于忽略了反向排队时间，所以不会对正向链路产生很大影响，正向链路主要受到 T1、T2、T3 的影响，由于 T1 表示正向固定延迟时间，T2 表示反向固定延迟时间，在一个相对稳定的网络环境中两者基本都是常量，不会产生很大的波动，故拥塞窗口大小主要由正向排队时间 T3 决定，如果 T3 变大，表示数据包排队时间延长，这很有可能是正向链路发生了网络拥现象，则根据算法拥塞窗口会相应减小，减小网络吞吐量。

由上面的推导得到新的拥塞避免机制，此时网络回路传输延迟时间是 T1,T2,T3 三者之和，由于每个数据报的正向固定传输延迟和反向固定传输延迟基本相等，所以可以得到

$$T1 = T2 = BaseRTT / 2 \tag{4-19}$$

由于 BaseRTT 基本是回路第一次传输测量的时间，所以 T1、T2 比较容易计算出。为了计算 RTT,我们必须知道正向排队时间 T3 的数值，我们提出了相对延迟的概念，所谓相对延迟 T_m，是指发送端连续发送两个数据报 m 和 n，当这两个数据报到达接收端时所用的时间差。具体的计算过程为，发送端首先记下发送报文 m 的时间 A_m，发送报文 n 的时间 A_n，计算出两者的时间间隔 S_m，当接收端收到报

文 m 和 n 时，分别记录下接收时间B_m和B_n，然后把这个时间分别放入数据报 m 和 n 的 ACK 确认报文的 TCP 选项中，这样当发送方收到这个 ACK 确认后，就能够从 TCP 报文头中取出接收时间，计算出两个报文的接收时间间隔R_{mn}，则这两个数据报的相对延迟时间为

$$T_{mn} = R_{mn} - S_{mn} \tag{4-20}$$

根据

$$S_{mn} = A_m - A_n \tag{4-21}$$

$$R_{mn} = B_m - B_n \tag{4-22}$$

则$T_{mn} = R_{mn} - S_{mn} = (B_m - B_n) - (A_m - A_n) = (B_m - A_m) - (B_n - A_n)$ (4-23)

（4-23）中$(B_m - A_m)$即为数据报 m 在正向路径上的传输时间SR_m，同理，$(B_n - A_n)$即为 数据报 n 在正向路径上的传输时间SR_n，则式（4-23）可写为

$$T_{mn} = SR_m - SR_n = (T1 + T3_m) - (T1 + T3_n) = T3_m - T3_n \tag{4-24}$$

（4-24）中$T3_m$即为数据报 m 的正向排队时间，$T3_n$即为数据报 n 的正向排队时间，这样就可通过相对时延的来计算修改后算法每个数据报的回路传输时间 RTT.

根据（4-13）（4-24）得到数据报 n 在回路上的传输时间公式：

$$RTT_m = T1 + T2 + T3_m = T1 + T3_n + T_{mn} + T2 = SR_n + T_{mn} + BaseRTT / 2 \tag{4-25}$$

即每个数据报文的传输时间等于它上一个数据报的正向传输时间加上相对时迟，再加上最小传输时间的一半。其中的正向传输时间、相对时延均可计算得到，最小传输时间可以测量得到，这样就可按照递归规律计算出每个数据报的传输时间。

4.4.2 改进算法仿真试验

利用仿真软件 NS2 验证改进算法的有效性，我们设计的网络拓扑结构如图 4-1 所示

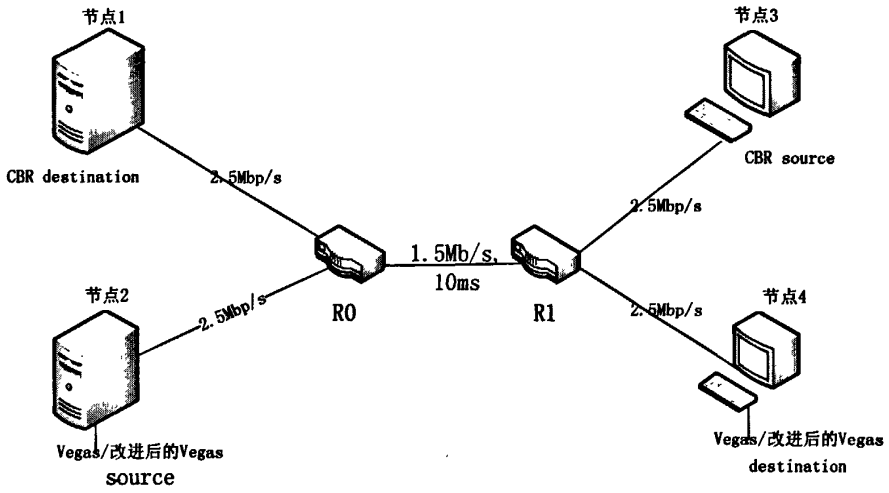

图 4-1 网络拓扑结构图

如图 4-1 所示，节点 2 到节点 4 之间是一条 TCP 链路，发送端分别采用 TCP Vegas 算法和改进后的算法，节点 3 到节点 1 之间是一条 UDP 链路，作为反向链路使用，节点 3 是一个 CBR 源，以固定速率发送数据报，设定发送速率为 2MB/S。各节点之间的数据发送速率如图中标识。实验的目的就是观察在反向链路持续发送数据的时候，正向链路中 TCP Vegas 算法和改进算法的性能表现。

仿真时间为 30 秒，从第 10 秒钟开始反向链路发送数据，直到第 20 秒结束，则 TCP Vegas 算法和改进算法的网络吞吐量随时间变化图如 3-2 所示：

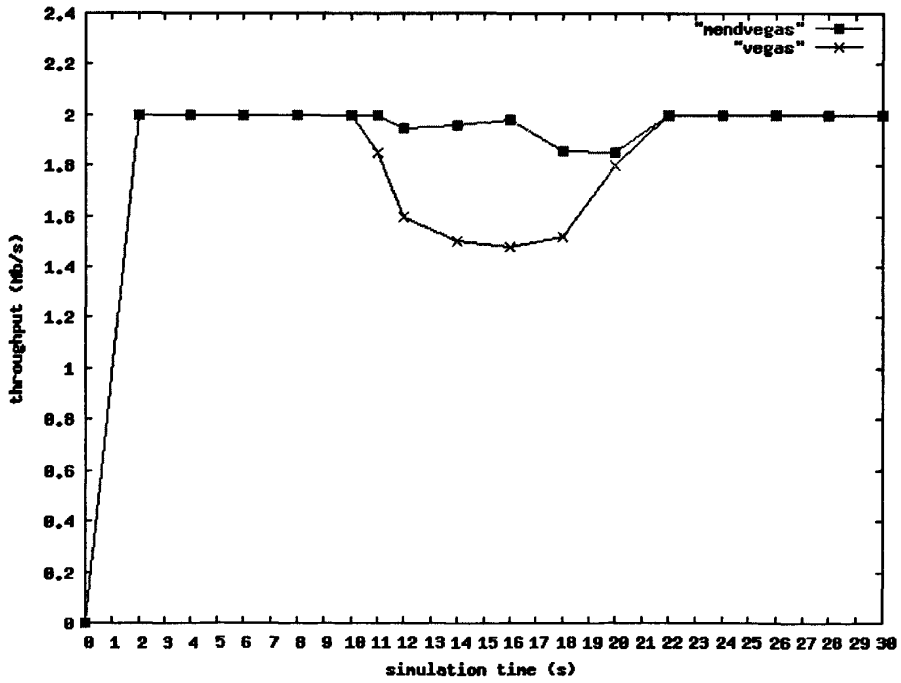

图 4-2 TCP Vegas 算法与改进算法的吞吐量比较图

从图 4-2 中可以看出，当网络中有持续的反向链路流量时，TCP Vegas 算法会减小吞吐量，而改进算法的吞吐量比 TCP Vegas 算法要高，基本稳定在一定的范围内，这证明了改进算法的有效性。

4.5 TCP Vegas 算法与 TCP Reno 之间的兼容性问题

4.5.1 算法之间兼容性问题分析

在 4.3 节中我们分析了 TCP Vegas 算法存在的问题，首要的问题便是它和目前 TCP Reno 算法的不兼容问题。TCP Reno 算法是一种被动的反应式算法，虽然它是目前网络上应用最广的算法，但是随着网络负载量的增大和网络环境的负载化，它的弊端也越来越突显，使用更合适的拥塞控制算法来替代 TCP Reno 的趋势日益明显。

TCP Vegas是一种主动式的拥塞控制算法，提前预测，保持稳定是其最大的特色。理论上已证明TCP Vegas较TCP Reno可提高37%~71%的吞吐量。实际应用中我们对网络负载量的要求越来越大，对网络稳定性的要求也越来越高，在这两方面TCP Vegas算法更适合目前的网络环境，但是它和TCP Reno算法的不兼容性又使得替换的操作难度太大，所以解决TCP Vegas和TCP Reno算法的不兼容问题是当今的一个热点问题[38]。

经过前面对TCP Vegas算法的分析，我们知道之所以TCP Vegas算法在与TCP Reno算法共存时竞争力不足是因为TCP Vegas设置的两个参数α和β是固定值，一方面α和β这个区域可以起到震荡阻尼的作用，保持了网络的稳定性；另一方面α和β数值的固定又限制了算法的带宽竞争能力，导致了两算法在共存时TCP Vegas处于劣势。

为了证实α和β对算法兼容性的影响，我们采用了如图1所示的网络拓扑图做了仿真试验，通过设定不同的α和β数值来观察两算法共存时吞吐量的变化情况，得到的结果如表4-1所示：

表 4-1 α 和 β 对 TCP Vegas 算法的影响

α	1	2	4	6	8	12
β	3	4	6	8	10	14
Reno 的平均吞吐量	238	214	186	153	147	54
Vegas 的平均吞吐量	78	103	134	168	183	284

从表4-1中可以看出，当α=1，β=3时，TCP Vegas的平均吞吐量远远小于TCP Reno的平均吞吐量。当α和β的数值逐步增大时，TCP Vegas平均吞吐量也逐渐增

大，这说明增大α和β的确对增强TCP Vegas的带宽竞争能力有较大作用。但是α和β是算法在网络中额外设定的数据包的数量，不可能设定很大的数值。

4.5.2 TCP Vegas-A 算法及仿真分析

针对 α 和 β 对 TCP Vegas 算法的限制问题，许多学者提出了 α 和 β 动态设定的方法，其中代表性的有 TCP Vegas-A[39]算法。

TCP Vegas-A 算法也叫自适应 TCP Vegas 算法，它是一种针对 TCP Vegas 算法的改进拥塞控制算法。在 TCP Vegas 中 α 和 β 分别被设置为 1 和 3，而 TCP Vegas-A 算法随着网络情况动态调整 α 和 β 的值，使之随时适应网络的要求。它主要修改了 TCP Vegas 在拥塞避免阶段的策略实现，慢启动和拥塞恢复阶段的策略与 TCP Vegas 保持一致。

TCP Vegas-A在拥塞避免阶段的算法可表示如下：

```
If diff <α{
    If α>1 and Th(t)>Th(t-rtt){
    cwnd=cwnd+1;
    }
    Else if α>1 and Th(t)<Th(t-rtt){
    cwnd=cwnd-1; //减小拥塞窗口
    α=α-1,β=β-1;
    }
    Else ifα==1
    cwnd=cwnd+1;//增大拥塞窗口，拥塞窗口呈指数级增长
    }
Else if α< diff <β{
    If Th(t)>Th(t-rtt){
    cwnd=cwnd+1;
    α=α+1,β=β+1;
    }
    Else if Th(t)<=Th(t-rtt){
    No update of cwnd, α, β;//cwnd, α, β 均保持不变
    }
```

```
}
Else if diff >β{
    If(α>1){
    cwnd=cwnd-1;
    α=α-1, β=β-1;
    }
    Else{
        No update of cwnd, α, β
     }
}
```

上式中的 diff =(Expected-Actual)*BaseRTT, Th(t)是在时刻 t 时的实际吞吐速率，Th(t-rtt)是在 t 时刻之前的一个 RTT 时刻的实际吞吐速率，为了验证 TCP Vegas-A 算法在与 TCP Reno 算法兼容性上的提高，我们做了仿真对比试验，试验 1 仍采用图 1 中网络拓扑结构，首先对 TCP Vegas 与 TCP Reno 共存时两种算法的拥塞窗口进行分析，仿真结果如图 4-3 所示

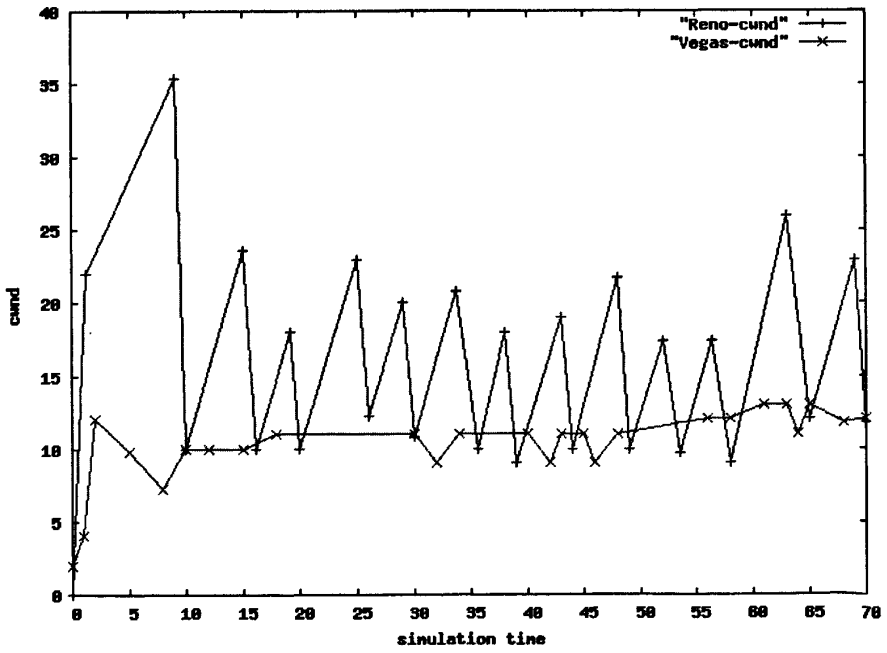

图 4-3 TCP Vegas 与 TCP Reno 共存时的拥塞窗口变化

从图4-3中可以看出当TCP Vegas与TCP Reno共存时，TCP Reno的拥塞窗口的稳定性较好，但是吞吐量远远小于TCP Vegas算法，这也是机制本身的原因造成

的结果。试验2采用图4-1中的网络拓扑结构，在链路1中用TCP Vegas-A算法代替TCP Vegas算法，其他保持不变，对TCP Vegas-A算法与TCP Reno算法共存时的拥塞窗口进行分析,仿真结果如图4-4所示：

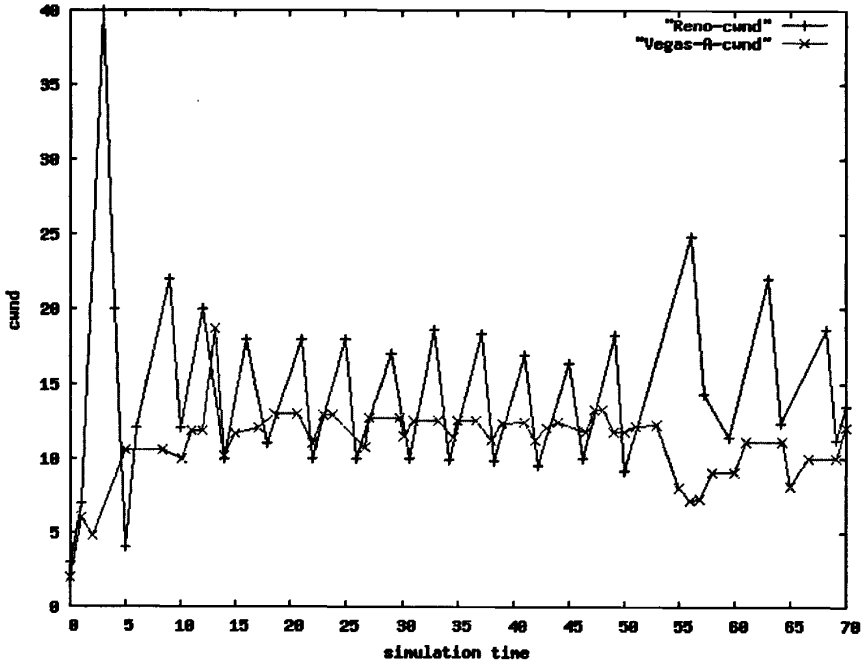

图 4-4 TCP Vegas-A 与 TCP Reno 共存时的拥塞窗口变化

从图 4-4 中可以看出，虽然与 TCP Vegas 的窗口相比，TCP Vegas-A 拥塞窗口的吞吐量有一定的增加，但是仍然保持一个较小的数值，使得在与 TCP Reno 共存时仍处于劣势。尤其是在仿真的最后 50-70 秒之间拥塞窗口有较大的波动，影响了算法本身的稳定性。故此算法存在一定的不足。

通过对 TCP Vegas-A 的分析和仿真试验的比较，我们知道 α 和 β 的动态变化机制是可以提高 TCP Vegas 算法共存时的竞争力的，但是诸如 TCP Vegas-A 算法并没有达到我们期望的水平，因此我们试图在 TCP Vegas-A 算法的基础上找到一种合适的机制对 TCP Vegas 拥塞避免机制进行改进，期望新的机制可以达到理想竞争能力。

4.5.3 改进算法以及仿真分析

TCP Vegas-A 算法采取了动态变化的 α 和 β 数值，根据不同的网络状况 α 和 β 的数值可以动态调整，这种措施提高了 TCP Vegas 算法的带宽竞争力，也提高了 TCP Vegas 算法和 TCP Reno 算法之间的兼容性。

和 TCP Vegas 算法相比，TCP Vegas-A 算法不只依赖于 α 和 β 作为拥塞窗口的判断准则，而是将网络中的瞬时吞吐率同样加入进来。当 Diff 小于 α 时，若 t

时刻实际吞吐率大于前一个 RTT 时刻的吞吐率则拥塞窗口增大，但若 t 时刻实际吞吐率小于前一个 RTT 时刻则同时减小窗口和 α，β 的数值；当 Diff 介于 α 和 β 之间时，若吞吐率相比前一个 RTT 时刻增大则同时增加拥塞窗口窗口和 α，β 的数值，否则保持不变。当 Diff 大于 β 时，无论 t 时刻实际吞吐率如何变化，Vegas-A 在减小窗口的同时也减小 α，β 的数值以进行有效的拥塞控制。

TCP Vegas-A 算法的思想是若 t 时刻实际吞吐率大于前一个 RTT 时刻的吞吐率，说明网络中的带宽没有被完全利用，此时不必考虑 α, β 的限制，增加拥塞窗口的大小以便充分探测网络可用空间;若 t 时刻实际吞吐率相比前一个 RTT 时刻的吞吐率下降，说明网络中可能发生网络拥塞现象，即使此时 Diff 小于 α 的数值，算法同样减小拥塞窗口的大小，同时也适当减小 α 和 β 的数值以适应网络的变化。但是算法在 Diff 大于 β 时，并未考虑吞吐率的变化情况，似乎稍显不妥。另外从仿真图 3-4 中可以看出算法的稳定性不好，吞吐量变化过快，这主要是由于拥塞窗口呈指数级的增减程度变化过大,尤其在算法的最后由于单纯的减小拥塞窗口的大小和 α, β 数值，使得该算法占有带宽能力显著下降。

在 TCP Vegas-A 算法的基础上，本文提出了 TCP NewVegas-A 算法。算法的思想是针对它的稳定性问题,可以考虑适当减小拥塞窗口的变化幅度。在 TCP Vegas-A 算法中，拥塞窗口的变化都是指数级的增加或减小，显的过于剧烈，可以考虑使其呈线性变化，这样可以保持拥塞窗口的稳定性。针对算法在最后部分对带宽的占有显著下降问题，可以加入吞吐量的变化来进一步进行控制，TCP NewVegas-A 算法可表示如下：

```
If diff<α{
    Ifα>1 and Th(t)>Th(t-rtt){
    cwnd=cwnd+1;
    }
    Else if α>1 and Th(t)<Th(t-rtt){
    cwnd= cwnd-1/cwnd;//减小拥塞窗口，拥塞窗口呈线性变化
     No update of α，β;
    }
    Else ifα==1
    cwnd=cwnd+1;
```

```
    }

Else if α<diff<β{
    If Th(t)>Th(t-rtt){
    cwnd=cwnd+1;
    α=α+1, β=β+1;}
    Else if Th(t)<=Th(t-rtt){
    No update of cwnd, α, β; // cwnd, α, β 均保持不变
    }
}
Else if diff>β{
      Ifα>1 and Th(t)>Th(t-rtt){
      cwnd=cwnd+1;
      α=α+1,β=β+1;
}
        Else Ifα>1 and Th(t)< Th(t-rtt){
        cwnd =cwnd-1/cwnd;
        No update of α，β;
}
```

NewVegas-A 算法的思想是当 diff<α 时，若吞吐率继续增大，则增大拥塞窗口（只增大一个 cwnd）；若吞吐量减小，说明有一定程度的网络拥塞发生则减小拥塞窗口（减小 1/cwnd）;当 α<diff<β 时，若吞吐量增大，则增大拥塞窗口，同时增大 α 和 β 的数值，否则保持不变；当 diff>β 时，若吞吐量继续增加，则增大拥塞窗口，α 和 β;否则，减小拥塞窗口的数值。算法的目的是最大程度的利用网络空余带宽，始终保持带宽占有的稳定性。

采用图 4-1 中的网络拓扑结构，在链路 1 中用 TCP NewVegas-A 算法代替 TCP Vegas-A 算法，其他保持不变，对 TCP NewVegas-A 算法与 TCP Reno 算法共存时的拥塞窗口进行仿真，结果如图 4-5 所示：

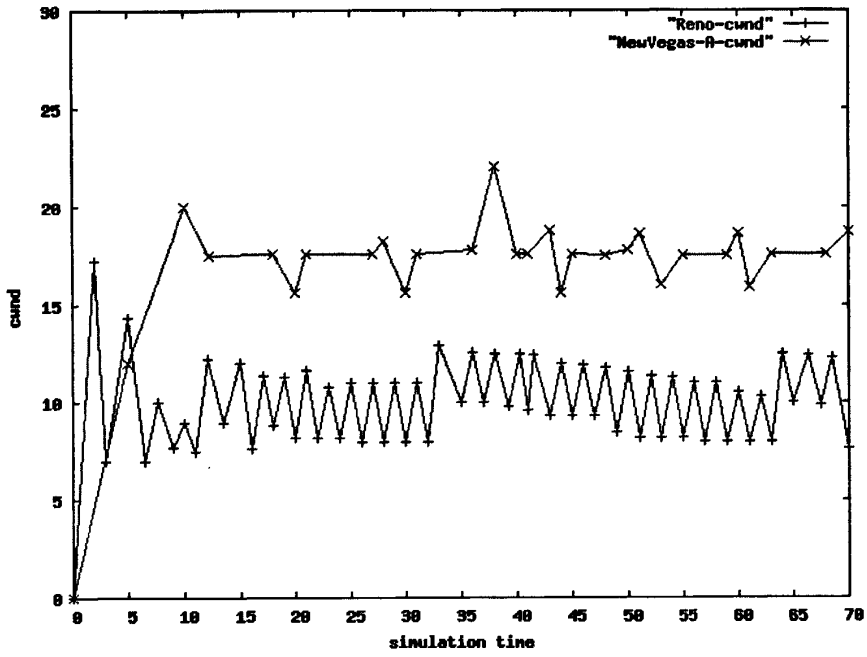

图 4-5 TCP NewVegas 与 TCP Reno 共存时的拥塞窗口变化

从图 4-5 可以看出，由于 TCP NewVegas-A 算法采取了线性增减拥塞窗口的策略，使得其吞吐量稳定性有了一定的提高；同时根据 α，β 动态变化的数值，我们在各个阶段都观测了实际吞吐速率的变化情况然后采取适当的措施来改变拥塞窗口，从而大大增强了算法的带宽竞争能力，提高了和 Reno 算法的兼容性，从仿真图中我们可以看到 TCP NewVegas-A 算法的吞吐量已超过 Reno 算法，证明了算法的可靠性。

为验证改进算法的稳定性，仍采用图 4-1 中的网络拓扑结构，在链路 1 中采用 TCP Vegas-A 算法，链路 2 中采用 TCP NewVegas-A 算法，设置瓶颈链路 R0 到 R1 之间的链路容量为 1Mbps，其余链路的容量为 10Mbps，设置路由器的缓存为 50 个数据包，仿真时间为 30s。对 TCP NewVegas-A 算法与 TCP Vegas-A 算法共存时的丢包率进行分析，结果如图 4-6 所示：

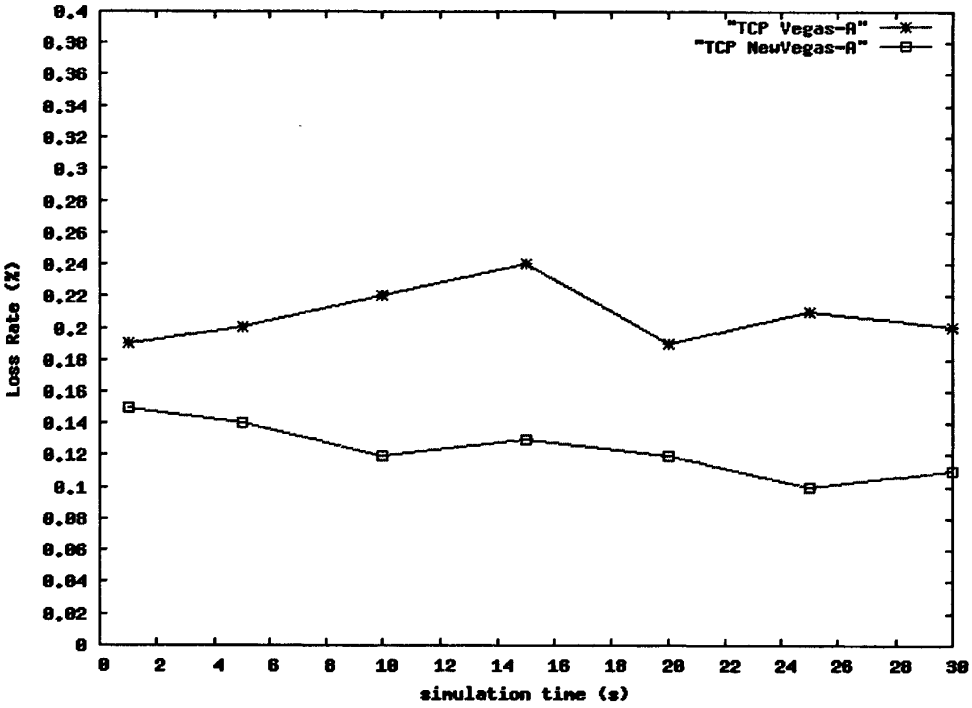

图 4-6 TCP NewVegas-A 与 TCP Vegas-A 算法共存时的丢包率分析

从图 4-6 中可以看出，TCP NewVegas-A 算法的丢包率和 TCP Vegas-A 算法相比有一定减小，并且在变化幅度上比较稳定，这主要是由于 TCP NewVegas-A 算法在发送窗口上采用了线性增减策略，保持了数据量的稳定性，减小了丢包概率。

4.6 TCP NewVagas 算法

上两节中，我们分别针对 TCP Vegas 算法中的反向链路拥塞问题和算法兼容性问题提出了自己的解决方案，并利用 NS2 仿真证明了解决方法的可靠性。在实际应用中，我们总是希望找出一种"完美"的拥塞控制算法，能够智能解决出现的所有问题，始终保持网络的高吞吐量、高稳定性和无差错性。虽然目前针对网络拥塞中出现的各种问题都给出了许多解决办法，但是由于各种解决方案思想不统一，很难将他们整合进一个算法之中。本节中我们希望能将上两节中的解决办法融合进一个算法之中，由此提出了 TCP NewVagas 算法。

在计算回路往返延迟时间的时候，我们去除了反向链路排队时间，在此基础上算出实际速率和期望速率的差值 DIFF, 进而利用标准值 diff（diff=DIFF*BaseRTT）和瞬时吞吐率共同决定拥塞窗口的变化情况，TCP NewVagas 算法机制如下：

$$BaseRTT = T1 + T2 \tag{4-26}$$

$$RTT = T1 + T2 + T3 \tag{4-27}$$

$$Excepted = cwnd / BaseRTT = cwnd / (T1 + T2) \tag{4-28}$$

$$Actual = cwnd / RTT = cwnd / (T1 + T2 + T3) \tag{4-29}$$

$$\begin{aligned} Diff &= Expected - Actual \\ &= cwnd(\frac{1}{T1+T2} - \frac{1}{T1+T2+T3}) \\ &= cwnd(\frac{T3}{(T1+T2)(T1+T2+T3)}) \end{aligned} \tag{4-30}$$

$$\begin{aligned} diff &= Diff * BaseRTT \\ &= cwnd(\frac{T3}{(T1+T2)(T1+T2+T3)}) * (T1+T2) \\ &= cwnd(\frac{1}{\frac{BaseRTT}{T3}+1}) \end{aligned} \tag{4-31}$$

则 $diff<\alpha$ 即 $cwnd < (\frac{BaseRTT}{T3}+1)\alpha$

$\alpha<diff<\beta$ 即 $\alpha(\frac{BaseRTT}{T3}+1) < cwnd < (\frac{BaseRTT}{T3}+1)\beta$

$diff>\beta$ 即 $cwnd > (\frac{BaseRTT}{T3}+1)\beta$

故在网络拥塞避免阶段算法可写为:

```
If cwnd<(BaseRTT/T3+1)α
{
 Ifα>1 and Th(t)>Th(t-rtt){
 cwnd=cwnd+1;
    }
 Else if α>1 and Th(t)<Th(t-rtt){
 cwnd= cwnd-1/cwnd;//减小拥塞窗口，拥塞窗口呈线性变化
  No update of α，β;
    }
 Else ifα==1
 cwnd=cwnd+1;
 }
```

```
Else if (BaseRTT/T3+1)α<cwnd<(BaseRTT/T3+1) β
{
    If Th(t)>Th(t-rtt){
    cwnd=cwnd+1;
    α=α+1, β=β+1;}
    Else if Th(t)<=Th(t-rtt){
    No update of cwnd, α, β; // cwnd, α, β 均保持不变
    }
}
Else if cwnd>(BaseRTT/T3+1) β
{
      Ifα>1 and Th(t)>Th(t-rtt){
      cwnd=cwnd+1;
      α=α+1,β=β+1;
         }
        Else Ifα>1 and Th(t)< Th(t-rtt){
        cwnd =cwnd-1/cwnd;
        No update of α，β;
        }
}
```

由于正向固定延迟时间 T1 和反向固定延迟时间 T2 相对来说是稳定，所以 BaseRTT 可看作是一个常量。那么初始拥塞窗口只和正向排队时间 T3 和阈值 α，β 相关。所以上面算法的思想可表述为：

当 cwnd<(BaseRTT/T3+1)α，并且 α>1 时， 若 t 时刻发送速率 Th(t)>Th(t-rtt)，则说明正向排队时间 T3 的数值相对较小，(BaseRTT/T3+1)α 数值相对较大，网络中带宽还未充分利用，所以必须增大拥塞窗口 cwnd 的数值来增加吞吐量。当 cwnd<(BaseRTT/T3+1)α，并且 α>1 时， 若 t 时刻发送速率 Th(t)<Th(t-rtt)，则说明正向排队时间较长，减小了发送端的速率。说明(BaseRTT/T3+1)α 数值相对较小，网络中可能发生拥塞现象，我们采取线形减小拥塞窗口 cwnd 的数值来降低

吞吐量，减缓拥塞程度。同样当(BaseRTT/T3+1)α<cwnd<(BaseRTT/T3+1) β 时，若在这阶段 t 时刻发送速率 Th(t)>Th(t-rtt)同样说明网络有空闲带宽可以利用，此时增大 cwnd 的数值，为使 cwnd 继续保持在这个范围内，同时增大 α 和 β，扩大阈值(BaseRTT/T3+1)α 和(BaseRTT/T3+1)β。

总之,该算法根据正向排队时间 T3 和动态阈值 α，β 来调整拥塞窗口的大小，以此来提高拥塞判断的准确性，保持网络状态的稳定性和高吞吐量。

为验证 TCP NewVagas 算法的有效性，我们采用图 4-1 中网络拓扑结构，在链路 1 中用 TCP NewVagas 算法代替 TCP Vegas 算法，其他保持不变，对 TCP NewVagas 算法与 TCP Reno 算法共存时的拥塞窗口进行分析,仿真结果如图 4-7 所示：

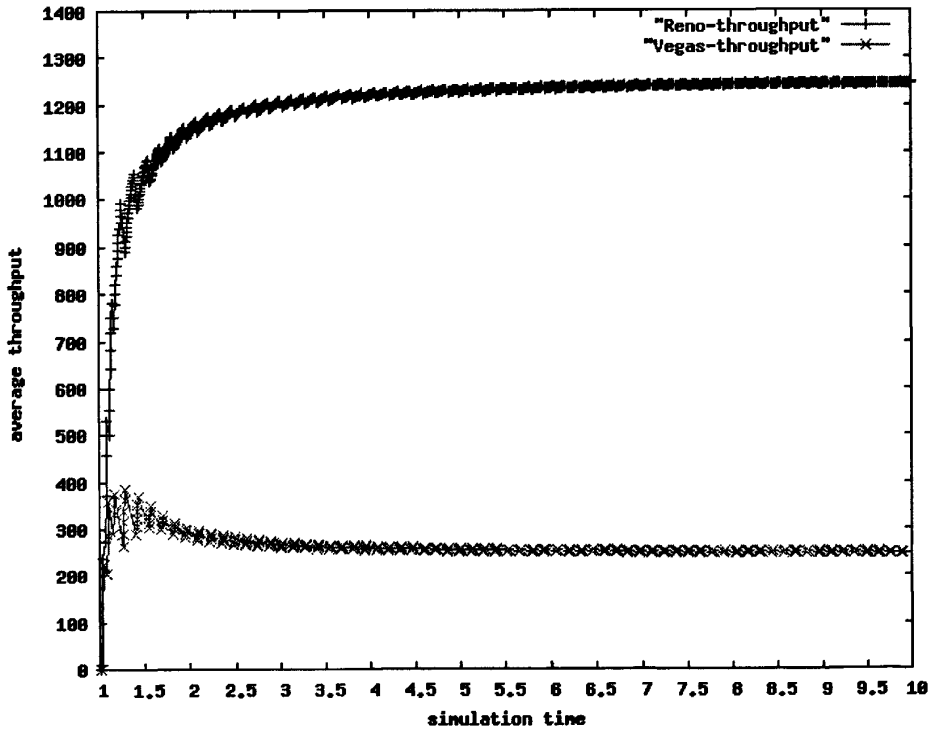

图 4-7 TCP NewVagas 算法与 TCP Reno 算法共存时的拥塞窗口变化

从图 4-7 中可以看出，在一定时间内，TCP NewVagas 算法的平均吞吐量远大于 TCP Reno 算法的平均吞吐量，解决了算法兼容性的问题，其次由于在算法机制中排除了反向排队时间对算法的影响，可以避免网络非对称性问题。

为验证 TCP NewVagas 算法的稳定性，采用图 4-1 中的网络拓扑结构，在链路 1 中采用 TCP Reno 算法，链路 2 中采用 TCP NewVagas 算法，设置瓶颈链路 R0 到 R1 之间的链路容量为 1Mbps，其余链路的容量为 10Mbps，设置路由器的

缓存为 50 个数据包，仿真时间为 30s。TCP NewVagas 算法与 TCP Reno 算法共存时的丢包率进行分析，结果如图 4-7 所示：

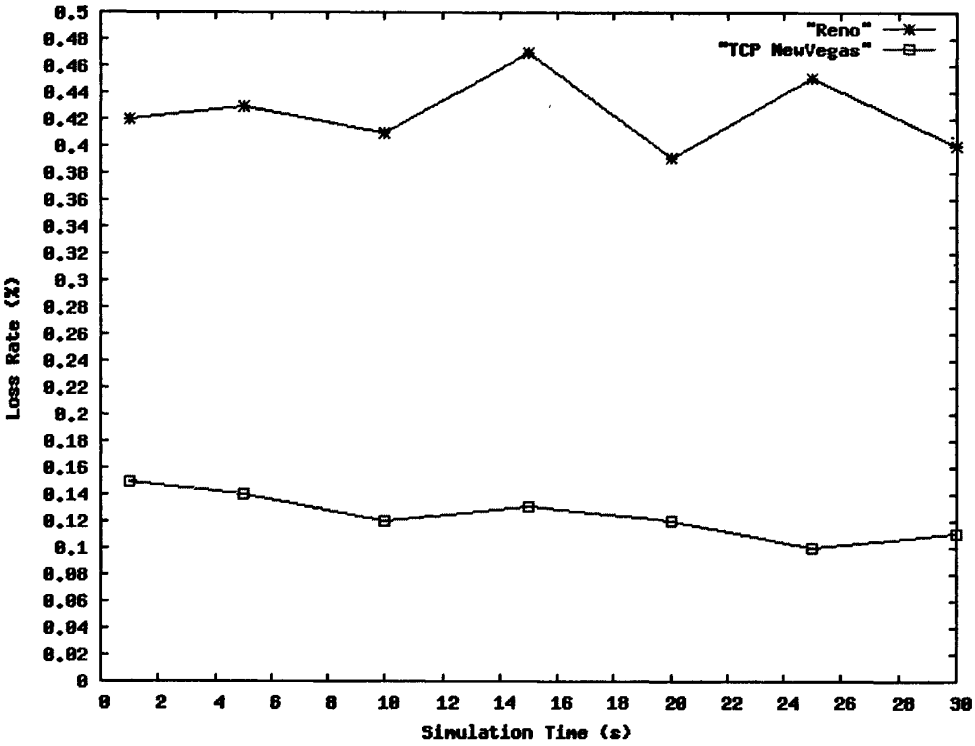

图 4-8 TCP NewVagas 算法与 TCP Reno 算法共存时的丢包率变化

从图 4-8 中可以看出，TCP NewVagas 算法在一定时间内的丢包率远小于 TCP Reno 算法，随着数据量的增加，这种优势会更加明显。总之，TCP NewVagas 算法是一种可行性算法，具有一定的应用价值。

4.7 本章小结

本章主要分析了 TCP Vegas 算法，TCP Vegas 算法是一种基于测量技术的网络拥塞控制算法，它采用了扩展的“慢启动”机制、新的拥塞避免机制和新的重传机制。尤其是采用新的拥塞避免机制使得该算法在网络吞吐量上大大增加。然而，新机制的采用也带来了包括不兼容性、非对称性、持续拥塞、公平性等问题，正是这些问题严重阻碍了 TCP Vegas 算法的应用。

本文首先分析了 TCP Vegas 算法的网络非对称性问题，经过数学模型的分析，创新性地提出将反向排队时间排除在链路往返时间之外的算法,经数学理论验证忽略反向排队时间不会对正向链路产生很大影响。仿真试验也证实了改进算法能有效解决降低由于反向链路发生拥塞导致的吞吐量下降问题。随后对 TCP Vegas 算法与 TCP Reno 算法之间的兼容性问题进行了分析，这种不兼容性是由

两种算法不同的实现机制产生的。在仿真分析基础上我们提出了 TCP NewVegas-A 算法,该算法在吞吐量和稳定性上都超过了 TCP Vegas-A 算法，并且在和 TCP Reno 算法共存时，该算法的吞吐量明显大于 TCP Reno 算法，这也大大提高了新算法的可应用性。

本章最后综合了前面两个问题的解决方法，提出了 TCP NewVagas 算法，目的是在一个算法中解决多个问题。经仿真分析，TCP NewVagas 算法能够有效解决 TCP Vegas 算法中存在的非对称性问题和兼容性问题，具有一定的应用价值。

第五章 基于网络端的拥塞控制算法研究

5.1 引言

近几年来,基于网络端的拥塞控制研究日益引起人们的重视。在传统的网络拥塞控制研究领域，人们一直将 TCP 端拥塞控制技术作为重点研究方向，TCP 端拥塞控制技术可以有效的解决网络拥塞问题，它可以调节拥塞窗口的变化来降低发送速率，利用慢启动和增量算法来探测网络带宽，它对因特网的鲁棒性起到了关键性的作用。但是随着网络结构的复杂和规模的扩大，研究者认识到仅仅依靠 TCP 端的拥塞控制是不够的，它存在着一定的缺陷。例如，TCP 端的拥塞控制的能力依赖于源的协作，这对目前的因特网来说不是一个很好的假设。再者，TCP 端的拥塞控制不能有效控制隔离网络流量，如果我们希望不同传输流的速率不相互干扰，那么单纯依靠 TCP 拥塞控制是不能实现的。总之，网络的发展要求我们必须同时利用源端拥塞控制和网络端的拥塞控制。

目前基于网络端的拥塞控制主要是在路由器上应用队列管理和队列调度。由于路由器位于拥塞的发生点上，因此在路由器上进行拥塞控制是非常有意义的，具体来说，在路由器上主要利用以下几种方式来实现拥塞控制[5]。

1：管理缓存。路由器可以利用管理缓存来实现对报文的管理，由于路由器有很多队列，用于接受和存放突发的大量报文，路由器需要确定何时发送拥塞信号，如果确定了还需要确定哪个报文需要标志或丢弃。这两个问题都是由路由管理算法决定的，一个好的缓存管理机制可以为源端拥塞控制提供好的激励。

2：拥塞信号。当链路中发生网络拥塞时，路由器可以给网络中的源节点发送拥塞信号，这个拥塞信号可以有多种形式，例如标志报文、丢弃报文、给源节点发送确定信息等。

3：调度。当多个连接共享一个路由器时，需要确定先发送哪个报文，这是由路由调度决定的。

路由器拥塞控制技术中常用的两种路由算法是队列管理算法和队列调度算法。前者是为到达的分组分配存储空间并控制队列长度，队列管理同时也是提供反馈的源头；后者是对等待服务的队列进行选择，按照一定的标准选择合适的报文到网络中去。一般来说，队列管理算法和队列调度算法紧密联系，对它们的共同使用可以实现网络流量的合理分配、最佳公平性、减小延迟等目标，对网络拥

塞的控制有很大的帮助。

本章主要研究了主动队列管理算法，在对各种算法的实现机制和数学模型分析的基础上，提出了一种改进的自适应虚拟队列(AVQ)算法，实现了对网络中不同数据流的区分服务，满足了实时性业务。随后，介绍了网络端的队列调度算法，对队列调度算法的研究进展进行了总结，并对网络端拥塞控制算法的研究前景做了展望。

5.2 队列管理算法

早期的队列管理技术采用“Drop-tail”即尾部丢弃的方法来管理分组，“Drop-tail”方式首先通常在每个队列中设置一个最大值 Q,然后接受封包进入队列直到队列长度达到最大值 ，随后到达的封包就被丢弃 ，直到队列长度下降才接受新的封包进入队列。“Drop-tail”算法反馈发送端的丢包信号 p 只能为 0 或 1

$$P_i = \begin{cases} 0 \rightarrow q_i <= Q \\ 1 \rightarrow q_i >= Q \end{cases} \qquad (5\text{-}1)$$

为更直观的了解“Drop-tail”算法,我们利用 NS2 对其进行仿真分析。采用图 5-1 所示的网络拓扑结构，发送端均采用输入 TCP 流，各条链路的带宽如图所示，其中 R0 到 R1 的时延设置为 10ms，其余链路的时延设置为 3ms，设置仿真时间为 30 秒。

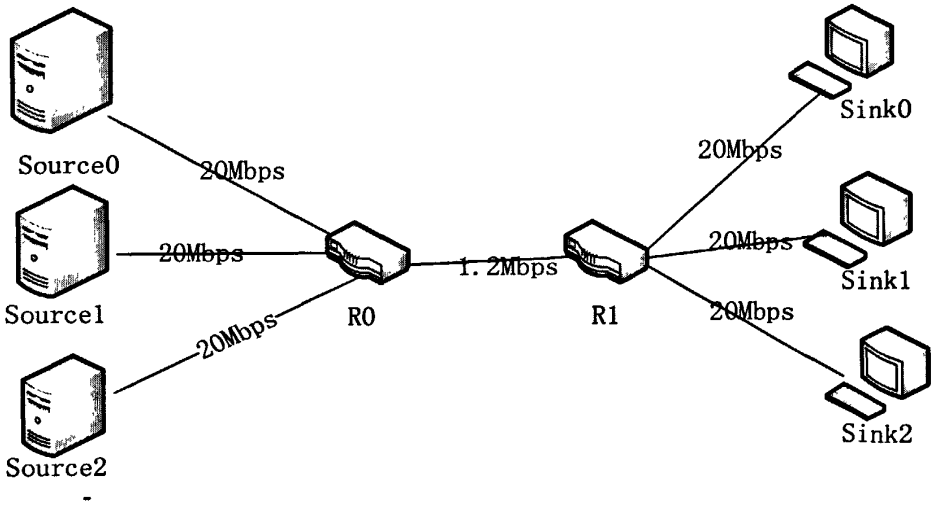

图 5-1 网络拓扑结构图

我们对“Drop-tail”算法的队列长度进行仿真，仿真结果如图 5-2 所示

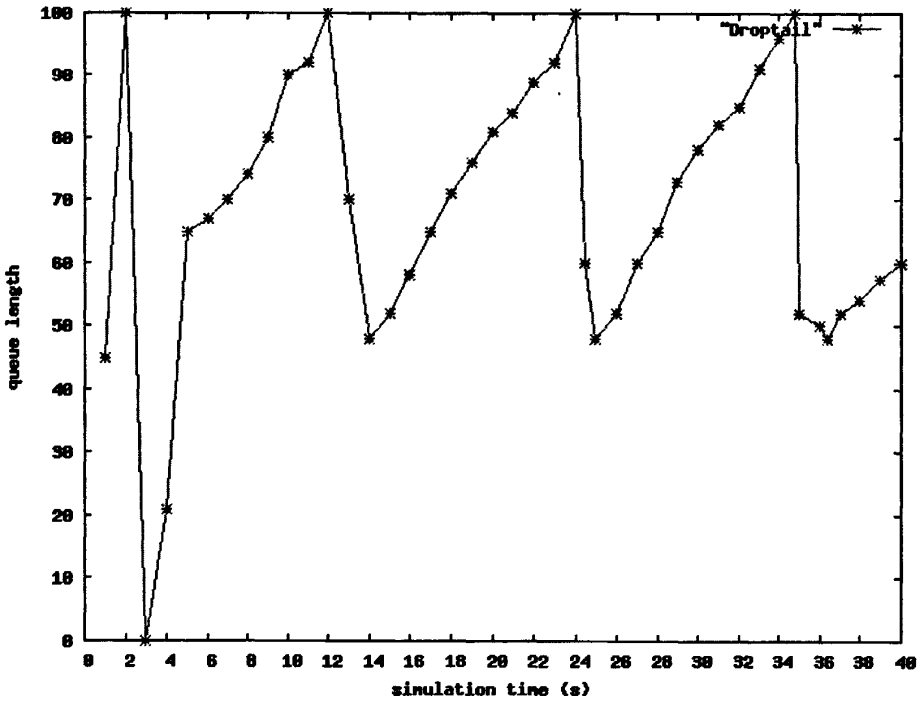

图 5-2 “Drop-tail”算法队列长度

从图 5-2 中可以看出“Drop-tail”算法的队列长度较长，并且队列长度的变化幅度较大，这是由于“Drop-tail”算法采用的尾部丢弃机制，只在队列排满时才会发出拥塞信号，当发生拥塞时，发送端会降低发送速率，使得各条链路的队列长度急剧减少，链路缓存处于空闲状态；当拥塞缓解时，发送端会提高发送速率，造成链路缓存急剧增加，如此反复又造成了全局同步问题，导致链路利用率的下降。

“Drop-tail”算法曾广泛的应用在互联网上，但是它存在着几个严重缺陷[40]，总结如下：

(1)满队列问题。由于“Drop-tail”算法只在队列排满时才会发出拥塞信号，因此会使得队列在相当长的时间内处于排满状态，并且链路设备端口的换粗容量越大，影响越坏，由于队列管理的目标之一是降低稳定状态下队列的长度，因为端到端的延迟主要是由于在队列中排队等待造成的。此外，满队列也容易造成突发数据的丢失。

(2)全局同步问题。由于网络的稳定性问题，现实中会常常出现突发数据，这也会导致到达路由器中的封包也是突发的。如果队列处于排满的状态，就会导致在短时间内连续大量的丢包。根据 TCP 端的的拥塞控制机制，发送端发现丢包就会急剧减小发送窗口，以次来减轻网络拥塞，随后发送端再逐步加大发送窗口，

直到达到网络拥塞，这就产生的“恶性循环”，从而会在周期性的一段时间内是网络处于低吞吐量、低利用率的状态，这就是所谓的全局同步现象。

(3)死锁问题。在某些情况下，“Drop-tail”算法会产生流量分配不均问题，让某个流或少部分流垄断队列空间，阻止其他流的分组进入队列。

除去“Drop-tail”算法，早期的队列管理算法还有随机丢弃算法和从前丢弃算法。当队列排满时，随机丢弃算法从队列中随机找出一个包丢弃以让新来的包进入队列；后者从队列头部丢包以让新到达的数据包进入队列。这两种方法都解决了死锁问题，但都没解决满队列问题。

5.3 主动队列管理技术

早期的队列管理算法都是被动的反应式丢弃分组，如果要有效的防止网络拥塞，应采用预见式的队列管理算法，这种算法称为“主动队列管理算法”(AQM)[41]。AQM 算法是目前网络中广泛应用的算法，它采用了分组标记来通知源端调节发送速率，目的是拥塞避免，但是也可以避免全局同步、消除对突发流量的歧视、即使存在非协作流也能维持队列的高端限制、减小分组丢弃、减小网络延迟等。具体来说，AQM 算法解决了以下几个问题[42]。

(1)解决了死锁问题。AQM 算法能够确保到来的封包总是有可用的队列空间，从而避免死锁行为的发生。也正如此，AQM 能防止路由器对低带宽高突发流的偏见行为。

(2)公平的处理了包括突发性、持久性、间歇性的各种业务流。

(3)可以减小路由器中丢弃的封包的数量，AQM[43]通过维持较小的队列长度，提供了更大的容量吸收突发数据包，从而大大减少了丢包数。

(4)减小了排队延迟。AQM 通过维持较小的队列长度来减小封包的排队延迟，这对交互应用比如 web 浏览、Telnet 业务和视频会议等非常主要。

目前有许多应用主动队列管理技术的算法，它们对网络拥塞的控制起到十分重要的作用，接下来我们会详细介绍几种典型的“AQM”算法。

5.3.1 随机早期检测算法（RED）

随机早期检测算法[44]是一种典型的 AQM 算法，它是于 1993 年由 S.Floyd 和 V.Jacobson 提出，该算法的基本思想是通过路由器输出端队列的平均长度来探测拥塞，一旦发现拥塞可能发生，就随机选择源端来通知拥塞，以使得源端通过调

节拥塞窗口的大小来降低发送数据速度，从而缓解网络拥塞。RFC2309[5]将 RED 推荐为 AQM 唯一的候选算法，因此 RED 算法已经被许多厂商所支持，并且由于 RED 是基于发 FIFO 队列调度策略的，并且只是丢弃正进入路由器的数据包，因此实施起来比较为简单。

RED 算法主要包括两部分算法，第一部分算法计算平均队列长度，以描述网关允许流量突发的程度；第二部分计算包丢弃/标记的概率，以决定在当前拥塞程度下路由器该以多大的概率丢弃数据包。

平均队列长度是由指数加权滑动平均（EWMA）低通滤波器来计算的。由于因特网的突发数据问题，如果一个队列很多时候是空的，然后被迅速充满，又很快被取空，这时就不能说明路由器发生拥塞而需要向源端发送拥塞指示。而指数加权滑动平均（EWMA）低通滤波器可以解决这个问题，具体计算公式为：

$$Q_{avg} = (1 - w_q) * Q_{avg} + w_q * q \tag{5-2}$$

其中，w_q为权值，q 为采样测量时的当前队列长度。w_q相当于低通滤波器的时间常数，它决定了路由器对输入流量变化的反应程度，因此对w_q的大小选择非常重要。如果w_q过大，RED 将不能有效地过滤短暂的拥塞；如果w_q过小，Q_{avg}会对实际队列长度的变化反应过慢，不能合理反映拥塞状况，在这种情况下路由器将不能有效检测到早期拥塞。所以，w_q的值应根据不同情况预先设置，它一般由路由器允许发生的突发业务的大小和持续时间决定。

选择公式（5-1）会使得因特网数据的突发或者短暂拥塞导致的实际队列长度暂时的增长不 会使得平均队长有明显的变化，从而“过滤”掉短期的队长变化，尽可能反映长期的拥塞变化。所以这是一种有效的计算平均队列长度的方法。

包丢弃/标记的概率的计算是基于两个和队列长度有关的阈值min_{th}，max_{th}。当有包到达路由器时，RED 计算出平均队长Q_{avg}，若Q_{avg}小于min_{th}，则没有数据包需要丢弃；若$min_{th} \le Q_{avg} \le max_{th}$，则按照概率 p 丢弃封包；若$Q_{avg} \ge max_{th}$，则所有的包都被丢弃；其中丢包概率 p 的计算公式如下：

$$P = \frac{Q_{avg} - min_{th}}{max_{th} - min_{th}} * P_{max} \tag{5-3}$$

由于 RED 算法使用的是基于时间的平均队长，有可能会发生实际队长大于平均队长的情况，如果队列已满，则随后到达的包只能被丢弃。平均队长和丢包概率的关系可用图 5-3 表示。

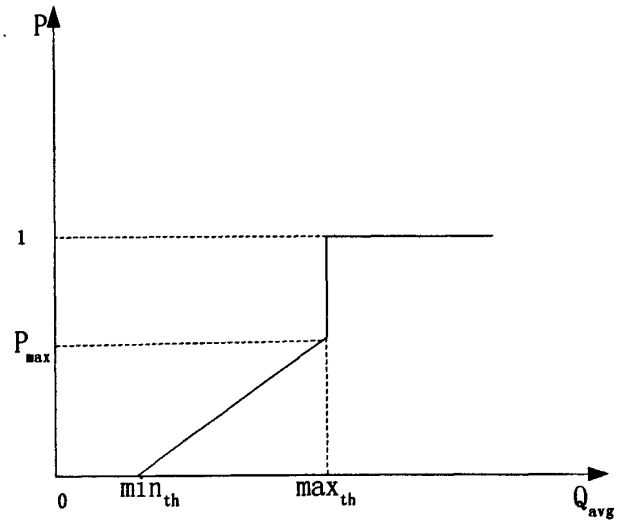

图 5-3 RED 平均队长和丢包概率的关系

队列管理面对的一个主要问题是排队延迟和吞吐量是一对矛盾，如果平均队长能够达到理想的状态，那么它必须要解决好吞吐量最大化和延迟最小之间的矛盾，这样才能在总体性能上得到较为理想的结果。阈值 min_{th} 和 max_{th} 是由理想平均队长决定的。在目前的因特网中一般将 max_{th} 设为 min_{th} 的 2 倍，这是由目前网络上的数据流的特点决定的。然而由于目前网络种类的增多，网络异构性和复杂性增强，这也意味着确定一个普遍适用的平均队长仍是一个有待研究的问题。

为更直观的了解 RED 算法,我们利用 NS2 对其进行仿真分析。仍采用图 5-1 所示的网络拓扑结构，各条链路的参数配置不变，该算法的平均队列长度变化如图 5-4 所示

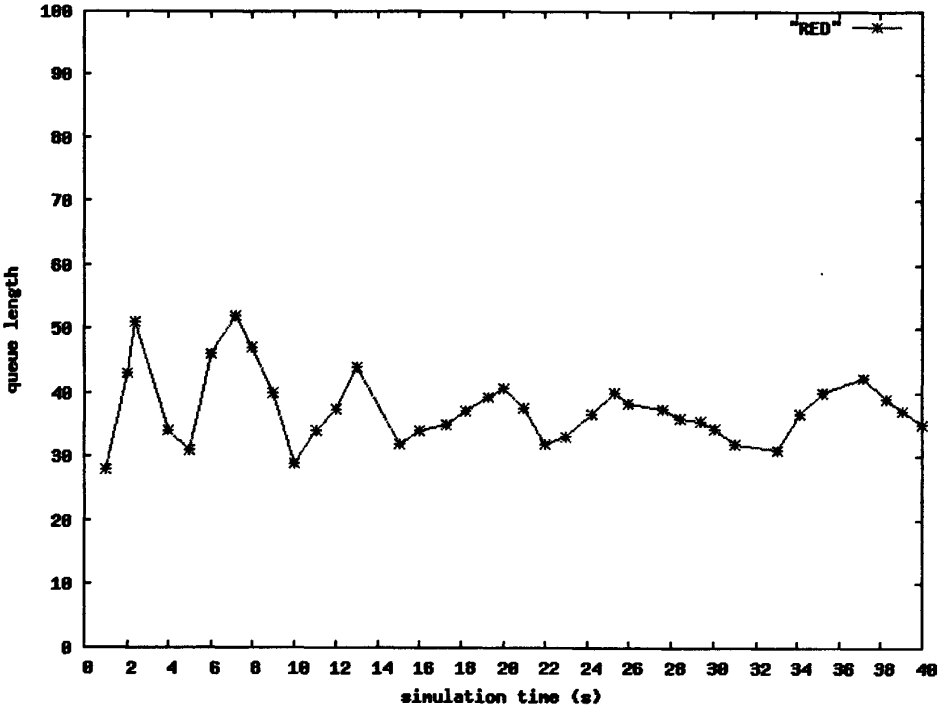

图 5-4 RED 算法队列长度

从图 5-4 中可以看出，RED 算法的平均队列长度变化较小，这是因为该算法可以提前对队列进行丢包，使得队列在达到某个长度之前就可以通知发送方降低发送速率，起到对网络拥塞起到预防作用，也可以避免全局同步问题。

和“Drop-tail”算法相比，RED 算法可以有效控制平均队列长度，可以限制时延的大小，在拥塞发生时，RED 标记封包的概率依赖于拥塞程度，并均匀间隔的丢包，可以避免连续丢包导致的全局同步现象，总体来讲，它是一种更为有效的路由器端拥塞控制算法。但是，RED 算法本身同样存在几个问题[45]。

(1)参数问题。RED 算法的性能对网络参数非常敏感，预先设置的 max_{th}、min_{th} 和 w_q 等参数或许在丢包、延迟的影响之下未必是当时的最优参数了，而算法参数的变化会对总体性能带来很大影响，所以能否找到一组动态变化的、适合当时网络环境的 RED 参数是目前研究的热点问题。

(2)公平性问题。RED 算法机制本身存在带宽竞争的公平性问题，由于不同的拥塞窗口大小、不同的回路延迟时间、不同的数据包大小等因素均会影响 TCP 流对带宽的使用。例如当两个 TCP 流竞争带宽时，若使用的拥塞窗口不同，则当拥塞发生时，RED 会使得小窗口源端陷入多重超时。这就产生了带宽使用的

公平性问题。

(3)拥塞程度问题。RED 算法不能有效估计网络拥塞的严重程度，目前 RED 算法存在的一个问题是当路由器开始丢包到源端检测到丢包的时间很长，而在这段时间内源端会继续以原速率发送数据，这就导致了更多的包被丢弃。为解决这个问题，RED 必须配置足够大的缓冲空间，但是当有大量突发报文时，RED 往往无法来得及做出很好的行动。这就导致了大量的报文被丢弃，无法断定拥塞程度。

近年来，许多学者提出了针对 RED 算法的改进算法，包括 ARED[46]算法、SRED[47]算法、FRED[48]算法、BLUE[49]算法等，这些算法在某些方面的改进是十分有效的。

5.3.2 自适应虚拟队列(AVQ)算法

自适应虚拟队列算法[50]是主动管理算法的一种，但是和传统的主动队列管理算法的实现机制有所不同。传统的 AQM 算法例如 RED 算法是根据平均队长来判断拥塞状况，而 AVQ 算法的判断依据是系统的负载情况。

AQM 算法基于一个虚拟队列的概念，每个虚拟队列的缓冲容量和实际队列一致，但是其虚拟带宽小于实际队列对应的输出链路带宽。当分组到达时，如果虚拟队列中有足够的缓冲空间，则被允许进入队列；反之，该分组会被丢弃。AQM 算法维持一个容量小于实际链路容量的虚拟队列，虚拟队列的容量按照式 5-4 进行计算

$$\Delta C=\alpha(rC-\beta)$$
$$Gc(s)=K+\frac{K}{T_i s}=K\frac{1+T_i s}{T_i s} \tag{5-4}$$

其中 ΔC 为虚拟队列容量，C 为链路实际容量，a 和 r 是数值在 0 到 1 之间的给定参数，a 代表衰减因子，b 是分组到达数率。AVQ 算法就是通过自适应参数的调整来对输出链路的容量利用率进行控制，目的是保持链路中高吞吐量的同时，也可以有效控制队列长度，进而获得较短的排队延迟。

为直观的观察 AVQ 算法的性能，我们利用 NS2 模拟器对该算法的平均队列长度进行分析，仍采用图 5-1 所示的网络拓扑结构，仿真结果如图 5-5 所示

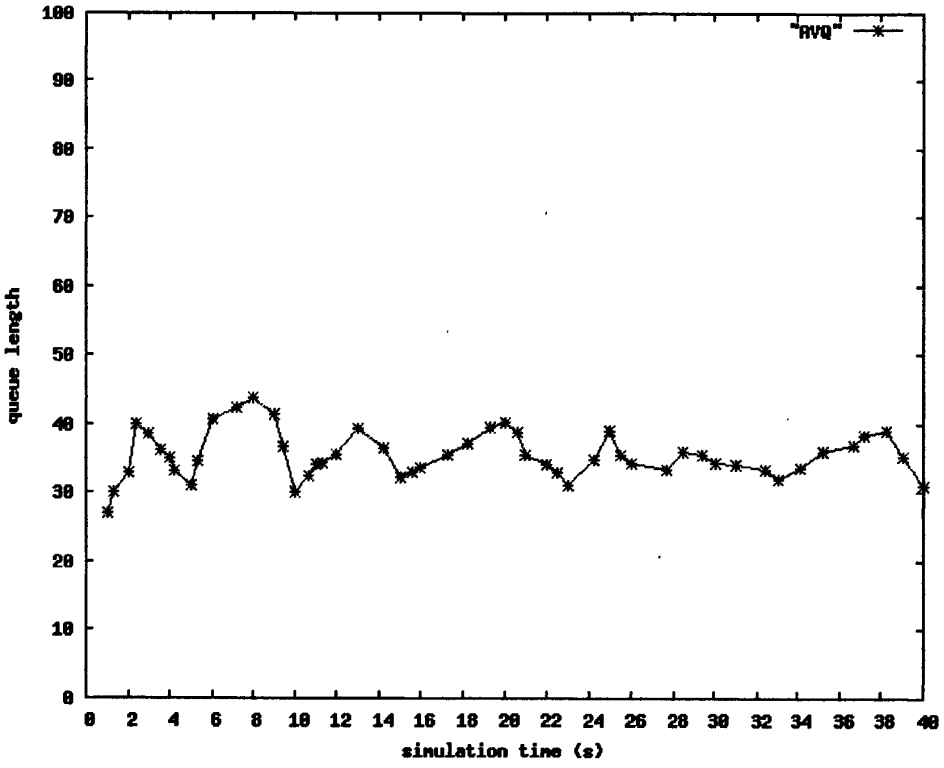

图 5-5 AVQ 队列长度

从图 5-5 中可以看出，与 RED 算法相比，AVQ 算法的队列长度变化幅度较小。前面我们提到过诸如 RED 的传统主动队列管理算法存在着过于依赖参数的问题，而 AVQ 算法主要通过自适应参数的调整来对输出链路的容量利用率进行控制，它可以获得更高的链路带宽，更快的相应速度，在保持链路的高吞吐量的同时，也可以有效的控制队列长度。因此它的实际应用价值日益受到人们的重视。

5.3.3 一种改进的 AVQ 算法

自适应虚拟队列算法存在着参数设定困难、队列长度抗干扰能力弱、存在一定链路损失的缺点，尤其是该算法不能提供区分服务，这对于目前日益复杂的网络环境来说是个很大的缺陷，因此许多对 AVQ 算法的改进算法也不断出现。通过分析动态阈值算法[51]和队列长度阈值算法[52]，我们发现可以综合利用这两种算法的思想来解决 AVQ 算法的区分服务问题。

改进算法的思想是将缓存队列管理算法和分组调度算法相结合,利用动态阈值算法的思想对AVQ 算法的队列管理进行改进,利用队列长度阈值算法的思想对AVQ算法的调度进行改进,通过对不同的业务设置不同的优先级,根据不同的优先级给予不同的服务等级,从而达到保证区分服务来满足实时性业务。

改进算法在路由器上维持多个虚拟队列，每个虚拟队列对应一个实际队列，虚拟队列和实际队列之间共享链路带宽和缓冲资源。设虚拟队列的虚拟带宽为c_i，实际队列输出链路带宽为C,每个虚拟队列的长度q_i，虚拟队列i中分组的优先级别为p_i，到达分组的长度为a_i，总的缓冲空间为B。在实际的操作中,我们只需要维护每个虚拟队列的长度q_i 即可实现算法，根据动态阈值算法，其中缓冲资源在每个队列间的分配实行动态阈值分配。

$$\left.\begin{aligned} p_i &= \alpha_i(B-\sum_i q_i) \\ q_i + a &\le p_i \end{aligned}\right\} \tag{5-5}$$

式（5-5）中的a_i代表优先级调节因子，根据动态算法的思想，每个优先级的调节因子是不同的。如果到达分组a_i的加入不会引起虚拟队列的溢出，则虚拟队列长度更新为

$$p_i = p_i + a_i \tag{5-6}$$

在实际的操作中,不需要对每个虚拟队列进行入队和出队操作,只需要维护即可。其中缓冲资源在每个队列间的分配实行动态阈值分配。

5.4 控制理论在主动管理算法中的应用

经过多年的发展控制理论已经形成了一套系统的理论，除了传统的 PID 控制，近年来智能控制技术也飞速发展，模糊控制技术、神经网络控制技术已得到广泛的应用。将成熟的控制理论应用到网络拥塞控制技术中可以得到控制机制更加有效，特别是在主动队列管理算法中应用控制技术已成为目前的拥塞控制领域的一个研究热点。

5.4.1 PI 控制在 AQM 算法中的应用

尽管新型的智能控制技术飞速发展，但是 PI 控制器仍然是目前工业中应用最多的控制器。PI 控制器结构简单，容易实现，并且是许多高级控制器的实现基础，所以目前仍受青睐。PI 控制器包含三个部分[53]比例（P）部分按比例控制误差；积分（I）部分消除静态控制误差；微分(D)部分对对象进行预测。PI 控制器的基本结构如图 5-2 所示

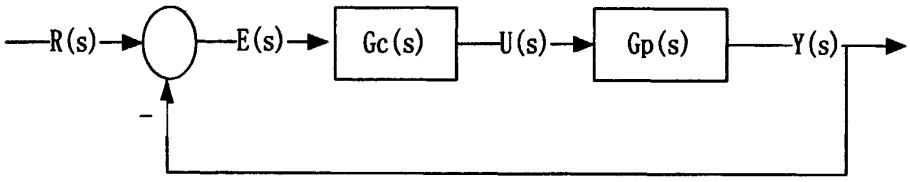

图 5-2 PI 控制器结构图

图 5-2 中，R(s)代表输入，Y(s)代表输出，Gc(s)为 PI 控制器，Gp(s)为控制进程 PI 控制器的基本结构为：

$$u(t)=K(e(t)+\frac{1}{T_i}\int e(t)dt) \tag{5-7}$$

其中 u(t)是控制信号，e(t)是控制误差，K 是控制器增益，T_i是积分时间。如果对式 5-5 进行拉氏变化那么可以得到：

$$U(s)=K(E(s)+\frac{E(s)}{T_i s} \tag{5-8}$$

那么

$$Gc(s)=K+\frac{K}{T_i s}=K\frac{1+T_i s}{T_i s} \tag{5-9}$$

$$E(s)=R(s)-Y(s)=R(s)\frac{1}{1+Gc(s)Gp(s)} \tag{5-10}$$

设$Gp(s)=\frac{Ke^{-\theta s}}{a(s)+1}$，其中 a(s)为包含 s 的多项式，$e^{-\theta s}$的一阶拉氏变化约为 $1-\theta s$，R(s)的阶跃拉氏变换为 1/s,代入（5-8）得

$$E(s)=\frac{T_i(a(s)+1)}{T_i s(a(s)+1)+K*K(1+T_i s)(1-\theta s)} \tag{5-11}$$

比例积分(PI)拥塞控制器是由 C.Hollot 等人提出[54]，与 RED 算法相比，PI 算法具有快速的动态相应能力，而且能将缓冲队列尽量保持在期望值附近。以控制论为基础的 TCP/AQM（PI）系统如图 5-3 所示[1]

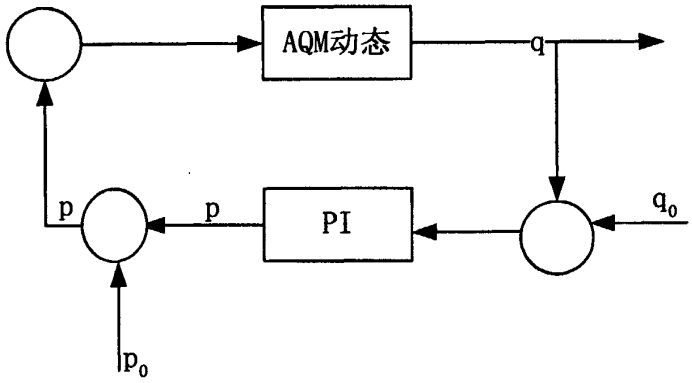

图 5-3 TCP/AQM 控制系统（PI）

在图 5-3 中，p 是报文丢失率，q_0是目标队列长度。如果系统的负载为 N，连接的 RTT 为 R，则在图 5-4 中的 PI 模块可以表示为 $G1(s)=\frac{1+\tau s}{Ts}$（t 、T 是待确定参数），AQM 动态模块可表示为 $G2(s)=\frac{K_m e^{-sR}}{(T_1 s+1)(T_2 s+1)}$，其中 $K_m=\frac{(RC)^2}{4N^2}$，$T_1=\frac{R^2C}{2N}$，$T_2=R$。则整个系统的开环传递函数为：

$$G(s)=G1(s)*G2(s)=\frac{1+\tau s}{Ts}\frac{K_m e^{-sR}}{(T_1 s+1)(T_2 s+1)} \qquad (5\text{-}12)$$

则根据上述各式，TCP/AQM（PIP）系统如图 5-4 所示

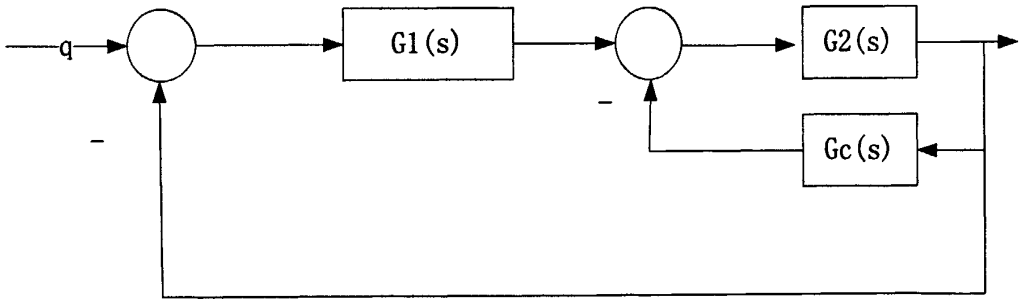

图 5-4 TCP/AQM 控制系统（PIP）

取 $R \le R_{max}, N \ge N_{min}$ 时系统的稳定条件[49]，PI 算法计算报文丢弃概率为式

$$P(k)=(a-b)(q(k)-q_0)+b(q(k)-q(k-1))+p(k-1) \qquad (5\text{-}13)$$

在式 5-11 中,P(k)是 k 时刻的报文丢弃概率，q（k）是 k 时刻的瞬时队列长度，a,b 是t 和 T 的函数，这两个参数一般是固定设置的，不会随着网络动态变化而

变化，这也对 PI 算法带来了一定缺陷。如果 PI 算法的参数能够随着网络的动态变化而灵活调整，那么可以取得更好的队列特性。

5.4.2 模糊控制在 AQM 中的应用

模糊的概念首先是由美国控制学家 L.A.Zadeh 于 1965 年提出的。模糊控制技术是智能控制的主要分支之一。传统的经典控制理论主要依赖精确的数学模型来控制对象，实际中许多对象的精确数学模型难以建立，这就可以应用模糊控制技术。模糊控制技术主要解决复杂系统控制问题，目前模糊控制技术广泛的应用到生活当中，如热交换过程的控制、污水处理过程控制、电梯控制[55]等，近年来在网络拥塞控制中的应用也日益广泛。

模糊系统的输入输出都是精确的数值，模糊系统的原理是采用语言的形式进行推理，将输入数据变成语言值，这就是所谓的“模糊化”。经过模糊规则推理得到的结果又变成为一个实际的精确值，这就是“反模糊化”。模糊控制的实质是将相关领域的专家知识和成果经验转化成模糊化后的语言规则,再通过模糊推理与模糊决策实现对复杂系统的控制。模糊控制器的设计一般步骤是模糊化，模糊规则推理，反模糊化，模糊控制系统的模型如图 5-5 所示

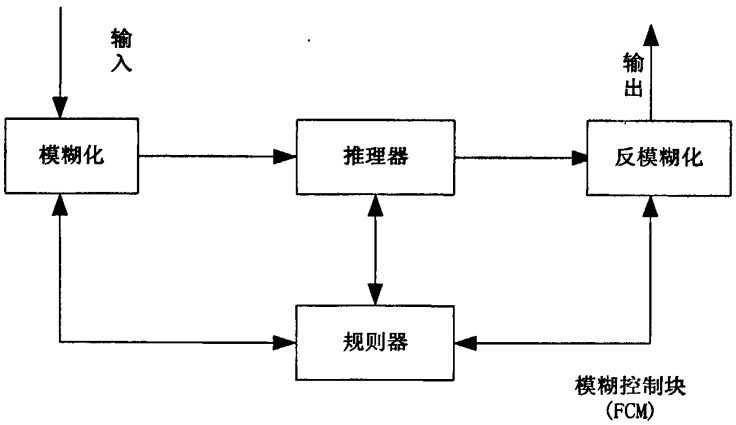

图 5-5 模糊控制系统

在实际应用中，模糊控制器的效果主要取决于网络的实际特性，能否得到准确的模糊控制规则，使得模糊推理更合乎实际情况。

模糊理论在 AQM 中的应用主要集中在队列长度、丢包方式、连接处理等方面。在队列长度方面 Chrysostomo[56]等人提出了一种针对 AQM 算法的新方案—FEM（Fuzzy Explicit Marking），该方法通过预先设定一个目标队列长度来调节路由器中的队列长度，同时获得较高的网络利用率和较低的平均延时。FEM

算法的模糊逻辑控制结构如图 5-6 所示[1]。

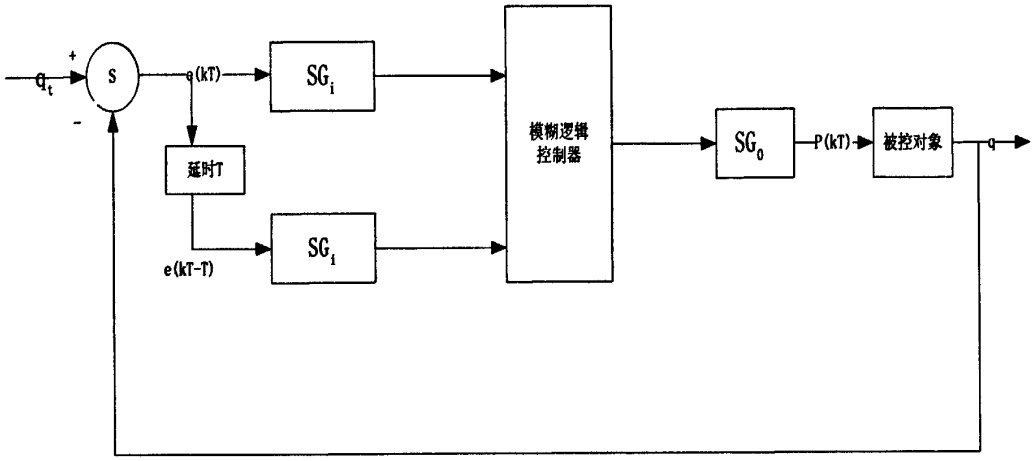

图 5-6 FEM 结构示意图

在图 5-6 中，q_t 代表目标队列长度，q 为队列平均长度，p(kT)代表标记概率，当 FEM 控制器的输入为 e(kT)和 e(Kt-T)，e(kT)= q_t-q(kT)为目标队列长度与当前值的差值，e(Kt-T)为上次采样时间的差值。为了使输入 FEM 控制器范围为[-1,1],设定：

$$SG_i = \begin{cases} \dfrac{1}{q_t}, q \le q_t \\ \dfrac{-1}{q_t - Q}, q \ge q_t \end{cases} \qquad (5\text{-}14)$$

式（5-12）中 Q 代表队列缓存大小。同时设定 SG_0 的值来确定 p(kT)最大值。Chrysostomou 在此基础上还提出了一种用于 TCP/IP 区分服务网络的 AQM 方案，对于确保型和尽力而为型两种不同的服务使用不同的 FEM 控制器，且每个控制器使用不同的目标队列长度。一般为确保型目标队列长度要大于尽力而为型的。

模糊理论也可以用在丢包方式上，例如参考文献 57 [57]提出了利用模糊逻辑设计一种智能分组丢弃机制，离线的合成推理使得分组丢弃判断仅需简单查表和比较运算即可完成。通过定义了拥塞指数这个新的测量变量来描述网络拥塞状态，进一步优化了路由器的传输性能。

模糊理论可用于不同连接的处理，例如文献 58[58]提出的算法采用模糊理论，根据缓冲队列长度的大小得出丢包概率。该模糊算法采用局部拥塞控制和全局拥

塞控制结合的策略来对队列的丢包进行控制，该算法使得路由器能够分清各个连接的状态，正确处理处于拥塞状态的数据包，保护和隔离非拥塞状态的 TCP 连接，优化网络利用率。参考文献 58 是基于加权公平队列原理提出的模糊控制拥塞算法，能够根据各连接的权值分配相应的队列长度和宽度，同时当局部拥塞时可部分增加某队列长度，使队列退出拥塞状态，以免在网络负载较轻时，长队列过早受到阻塞。

总之，模糊控制理论可以应用在 AQM 算法中的各个方面，并取得理想的效果。模糊理论同样可以和现有算法相结合，以改进算法的性能。如参考文献 59[59] 中提出了基于模糊 RED 算法和区分服务的一种新的 IP 拥塞控制机制，在不改变 RED 队列管理本身操作的情况下，使改进算法具有收敛性，改善 TCP 流量的吞吐时延特性，提高了网络利用率。

5.5 队列调度算法

5.5.1队列调度算法概述

队列调度是网络端拥塞控制技术的另一主要组成部分。队列调度技术主要是按照一定标准选择等待服务的队列并将其输出到网络链路中去，它的主要作用是管理各流之间带宽的分配，在实际应用中，队列管理和队列调度常常共同使用，它们可以有效地避免网络拥塞和控制拥塞。

队列调度算法按照是按照一定的规则来对网络节点中发生冲突需要排队的业务流进行调度和服务，目的是使所有业务流按照预定方式共享交换节点的输出链路带宽,队列调度算法的原理图如图5-7所示 [1]

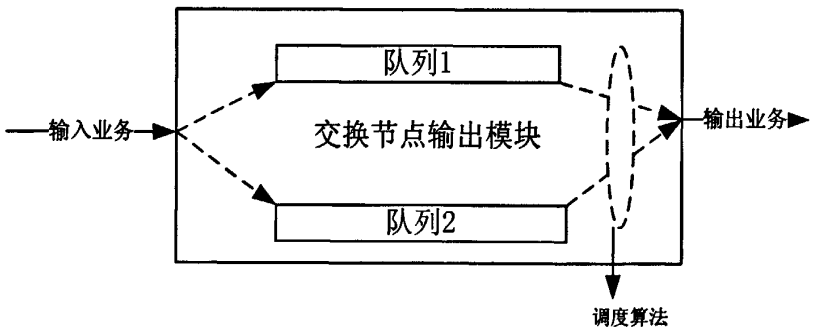

图5-7 队列调度算法原理图

如图 5-7 所示，假设有 N 个输入业务流到达交换节点后，分别暂存到 N 个

队列中，队列调度算法就会根据算法机制从这 N 个队列中选择下一个要传输的分组。不同的队列调度算法有不同的机制将输出业务流对应到不同队列中，目前常见的算法机制包括先到先服务算法，即根据业务流的到达时间对其进行服务，这种机制比较简单，它会把所有输入流都放在一个队列中。较复杂的调度算法会按照一定规则将不同业务流对应到不同的队列里面，这在不同的网络环境中是不同的。下面介绍几种主流队列调度算法

5.5.2 基于时延的调度算法

这种算法是以直接提供时延保证为目的，代表算法是早期优先算法(EDF[60]Earliest Deadline First)。在 EDF 算法中，系统会为每个队列提供一个时延参数 t，来代表时延的最大值。EDF 会为每个分组业务流计算一个时签，一般这个数值为业务流到达时间和其所属队列的时延参数之和，EDF 算法就根据这个时签的数值来选择服务顺序，时签最小的最先得到服务，这其中就涉及到对时签的排序问题，EDF 能很好的提供时延保证，但输入业务必须满足一定的属性要求。

EDF 对列调度算法在某些网络环境中存在着一定的缺陷，例如在实际应用中进入到下游的业务属性会发生变化，导致 EDF 无法提供端到端的时延保证。为此许多改进算法相继被提出，例如 H Zhang[61]等人提出了 RCS 算法，它在 EDF 的基础上引入了整形机制，即在业务流到达下游之前，首先进入整形器，经过整形后的业务需满足一定的属性要求，然后进入 RCS 中的 EDF 调度器，该算法可以提供端到端的时延保证，但是该算法在连路利用率上有一定的下降。

5.5.3 基于 GPS 模型的队列调度算法

GPS（Generalized Processor Sharing）流体调度模型可以对所有活动队列同时服务，并保证有明确的端到端的时延，保证了业务流的带宽共享。GPS 模型的实际系统是分组系统，它控制实际系统中分组的发送顺序，调度过程中系统维护一个“虚拟时间”，GPS 是一种理想的流体服务模型，在 GPS 会对共享的同一链路的每个会话建立一个独立的 FIFO 队列，每个会话被分配了不同的服务份额，对每个会话按它们的服务份额的比列来提供服务，假设存在 N 个会话，分配的服务份额分别为 q_1，q_2，q_3……q_n，其中的 q_n 称为队列 n 的权值。若服务器是以恒定速率 r 运行，时间 t_1 到 t_2 内会话 i 的业务数量表示为 $W_i(t_1, t_2)$，则一个 GPS

服务器在时间 t_1 到 t_2 内对会话 i,j 的服务满足式(5-15)

$$\frac{W_i(t_1,t_2)}{W_j(t_1,t_2)}=\frac{q_i}{q_j}=\frac{r_i t}{r_j t}=\frac{r_i}{r_j} \tag{5-15}$$

如果在时间 t_1 到 t_2 内有剩余的会话集合 $B(t_1)$，则在这个时间内服务会话 i 的速率为

$$g_i(t_i,t_2)=\frac{q_i}{\sum_{j\in B(t_i)} q_j}r \tag{5-16}$$

因为 GPS 模型只能在某一时间内保证一个分组得到服务，基于此出现了一类基于 GPS 模型队列调度算法：分组公平排队算法(PFQ)，目前应用较广泛的 PFQ 算法有加权公平队列算法 WFQ[62](Weight Fair Queueing), (Worst_case Fair Weighted Fair Queueing), WF2Q+等算法。

加权公平队列算法 WFQ 是 GPS 算法的近似算法，它是根据路由器的输出缓冲区划分队列的算法，它采用了依据业务流的调度机制，每个流占用其中一个队列，调度器采用轮换的原则发送各个队列中的分组，这种机制使不同的流分组完全分开，确保了每个流的传输质量。假设 Fp 是数据包 P 在 GPS 算法中的离开时间，那么 WFQ 算法就是一个模拟 GPS 算法并按升序调度数据包的工作保留算法，概括来讲，它的算法机制如图 5-8 所示

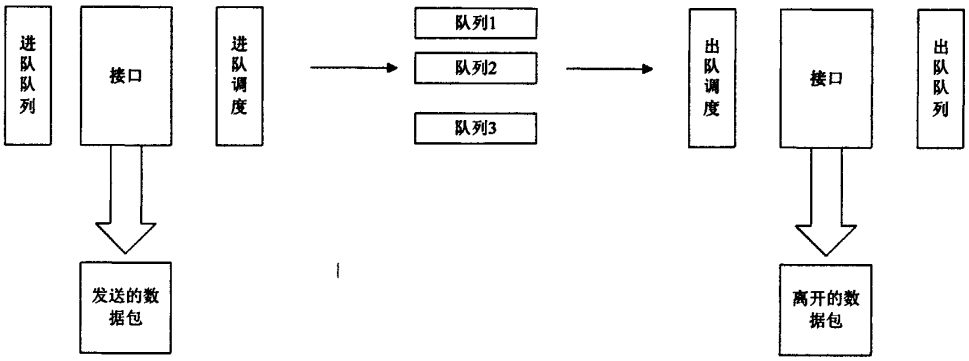

图 5-8 加权公平队列示意图

如图 5-8 所示，WFQ 算法对报文按流进行分类，即含有相同源 IP 地址、目的 IP 地址、源端口号、目的端口号、协议号、TOS 的报文属于同一个流，每个流被分配到一个不同的队列，出队时 WFQ 算法按流的优先级来分配每个流应占有的出口带宽，优先级数值越大得到的带宽越多，这样就保证了相同优先级业务之间的公平性，体现了不同业务流之间的权值。举例来说，假设当前有 5 个业务

流，优先级分别为 1，2，3，4，5，则带宽分配额是所有业务流的优先级加 1 之和，即 2+3+4+5+6=20，每个流所占带宽比例分别为(自己优先级别数值+1)/带宽分配额，则在此例中，每个流可得带宽比例分别是 2/20，3/20，4/20，5/20，6/20。由此可见，WFQ 算法可以对不同优先级的业务体现权值，这个权值依赖于 IP 报文头中所携带的 IP 优先级，WFQ 算法不仅保证了时延性，还提供了很好的公平性，然而该算法计算复杂度很大，对它的改进算法也不断提出。

SFQ(Stochastic Fairness Queueing 随机公平队列)是 WFQ 算法的一个改进，它的精确性不如其它的方法,但实现了高度的公平,需要的计算量亦很少.在 SFQ 算法中流量被分到数量很多的 FIFO 队列中，每个队列对应一个会话，数据采用轮转的方式发送，每个会话都按照顺序得到发送的机会，这样就体现了很高的公平性。SFQ 算法的精确性不高，主要是因为采用了散列算法把所有的会话映射到有限的队列中，这就可能出现一个队列分配几个会话的情形，从而可能会出现几个会话共享带宽，共享发包机会的情况，为了降低这种情形的出现几率，SFQ 算法会频繁的改变散列算法，这就影响了算法的精确度。

5.6 本章小结

本章主要介绍了基于网络端的拥塞控制算法。目前基于网络端的拥塞控制主要是在路由器上应用队列管理和队列调度。

本章首先介绍了网络中的队列管理技术。早期的队列管理算法基本是被动式算法，主要包括尾部丢弃算法，随机丢弃算法等，由于这些算法在应用中会出现全局同步和死锁问题，因此这种算法逐渐被主动队列管理算法(AQM)算法所取代,主动队列管理算法是一种预见式的队列管理算法,它可以有效解决死锁问题,有效减小排队延迟、具备更好的公平性。随后，本章详细介绍了几种主要的主动队列管理算法，包括随机早期检测（RED）算法、显示拥塞通知（ECN）算法、自适应虚拟队列(AVQ)算法等，对各算法的实现机制和数学模型进行了仿真分析，并在此基础上提出了一种改进的 AVQ 算法，即将缓存队列管理算法和分组调度算法相结合，通过对不同的业务设置不同的优先级,根据不同的优先级给予不同的服务等级,从而达到保证区分服务来满足实时性业务。

AQM 算法是目前实际应用中最广泛的算法，而目前的拥塞控制算法更多的是将控制技术应用到该算法中。为此，本章介绍了 PI 控制技术和模糊控制技术

在 AQM 算法中的应用，对其中的算法实现机制做了详细分析，并提出自己的观点。本章最后对队列调度算法进行了介绍，队列调度算法的主要作用是管理各流之间带宽的分配，其代表性算法包括基于时延的调度算法和基于 GPS 模型的队列调度算法，我们分别对其进行了深入的研究。在实际应用中，队列管理算法和队列调度算法经常是一起使用的，共同维持网络稳定和避免网络拥塞问题。

第六章 总结和展望

6.1 研究成果总结

自从互联网出现以来，网络拥塞问题就一直困扰其发展。伴随着网络规模的日益扩大和网络业务类型的不断丰富，网络拥塞问题变得越发严重，从早期的数据包时延增加，丢包概率增加，网络吞吐量下降问题到目前严重的“拥塞崩溃”现象，可以说网络拥塞问题已成为制约互联网进一步发展的瓶颈。

互联网的发展是基于 TCP/IP 协议的，网络中 90%的数据流和大部分业务，如 FTP，HTTP 都是基于 TCP 协议的，而且 TCP 所采用的拥塞控制算法是保证当今网络不断发展同时又避免拥塞崩溃的主要方法，所以对 TCP 端网络拥塞控制算法的研究一直是网络拥塞控制领域的重点，而近年来人们发现网络拥塞的控制不仅要基于源端，更需要网络端特别是路由器发挥重要的调节作用，对网络中的数据流进行管理和合理分配，共同达到避免网络拥塞，维持网络稳定可靠的目的。

本文分别从 TCP 端和网络端两部分对网络拥塞控制算法进行了较深入的研究，并提出了改进算法。在基于 TCP 端口的网络拥塞控制算法中，主要针对 TCP Vegas 算法进行了详细分析，在网络非对称性和算法兼容性两个方面对原算法进行了改进；在基于网络端的拥塞控制算法研究中，主要对队列管理算法和队列调度算法进行了研究。论文的主要研究内容和创新点总结如下：

1.基于 TCP Vegas 网络拥塞控制算法兼容性问题的研究

TCP Vegas 网络拥塞控制算法是一种基于测量技术的网络拥塞控制算法，相比传统的拥塞控制算法而言，TCP Vegas 算法在“慢启动”、“拥塞避免”和“快速重传”三个方面都做了改进，特别是在拥塞避免机制上采用了通过比较实际吞吐量和期望吞吐量来调整拥塞窗口大小的策略，将被动的拥塞避免机制改变为主动预防式的的避免机制大大提高了算法的适用性。目前网络中的主流算法是 TCP Reno 算法，它采用了反应式的拥塞避免机制，它持续增加自己的窗口，直到网络过载产生频繁的丢包来保证有效利用网络资源，而 TCP Vegas 算法采用的主动的拥塞避免机制，其目标是将网络带宽维持在一个稳定的水平，所以它不会持续的扩展自己的拥塞窗口。因此在实际应用中，TCP Reno 算法将会窃取 Vegas 的带宽，导致 TCP Vegas 算法不能起到实际作用。为解决这个问题，本文在分析

TCP Vegas-A算法的基础上,对TCP Vegas设置的两个参数α和β数值进行了分析，将其由固定值变为动态变动数值，此外将拥塞窗口的指数级增长变为线性增长，仿真结果表明改进算法提高了 TCP Vegas 算法的带宽竞争能力和带宽流量的稳定性。

2.针对 TCP Vegas 网络拥塞控制算法网络非对称性问题的研究

TCP Vegas 算法是一种基于时间的算法，因此 RTT 准确性至关重要，粗略测量的 RTT 可能导致对 cwnd 的粗略调整。目前 TCP Vegas 算法只是考虑了在数据传送方向上发生网络拥塞时采取的措施。如果拥塞发生在回路方向，也即 ACK 方向，TCP Vegas 算法的机制同样会采取拥塞避免措施，这样会引起对实际吞吐量的过低估计，引起拥塞窗口不必要的下降。实际应用中我们要求能够辨认拥塞是发生在哪个方向上，进而采取正确的措施保证网络的稳定性。为解决这个问题，本文首先 TCP Vegas 算法计算 RTT 的机制进行了分析,根据数学模型分析得到影响算法往返时延最大的是反向排队时间，为此可以在计算时除去该部分对 RTT 的影响，经过仿真表明，改进算法消除了 TCP Vegas 算法的网络非对称性问题。

3.基于 TCP Vegas 算法中的反向链路拥塞和兼容性问题的融合性的研究

TCP Vegas 网络拥塞控制算法兼容性问题和网络非对称问题，我们分别针对两者提出自己的改进方法，仿真也证明了改进算法的有效性能。在实际应用中我们都希望找到一种尽可能“完美”的算法，能够解决TCP Vegas算法中的各种问题，以适应目前网络的需要。然而目前针对 TCP Vegas 算法出现的各种问题的解决办法很难兼容，将他们完全融合到一个算法之中比较困难。本文综合分析了对 TCP Vegas 算法中的反向链路拥塞问题的解决算法，以及兼容性问题的算法，将他们融合在一起，提出了 TCP NewVegas 算法，仿真分析表明，该算法可以在和 TCP Reno 算法并用时保持较高的带宽竞争能力，也可消除反向链路的网络拥塞对算法机制的影响，相比 TCP Vegas 算法而言，在实用性上有很大提高。

4.针对自适应虚拟队列算法的区分服务问题的研究

自适应虚拟队列算法是基于系统负载情况来判断拥塞情况的算法，该算法的实现机制是维持一个容量小于实际链路容量的虚拟队列，和传统主动队列管理算法相比，该算法解决了死锁问题，保持了业务流的公平性能，可以减小路由器中丢弃的封包的数量，减小了排队延迟。但该算法的最大不足之出在于不能提供区

分服务，这对于目前网络业务种类不断增加，网络环境日益复杂的发展趋势来说是个很大的缺陷，使其实用性大大降低。为解决这个问题，本文首先分析了动态阈值算法和队列长度阈值算法，综合了这两种算法的思想用于原算法，具体来讲是利用动态阈值算法的思想对 AVQ 算法的队列管理进行改进,利用队列长度阈值算法的思想对 AVQ 算法的调度进行改进,通过对不同的业务设置不同的优先级,根据不同的优先级给予不同的服务等级,从而达到保证区分服务来满足实时性业务。

6.2 研究前景展望

上节中总结了本文的主要研究内容，分析了 TCP 端和网络端主流的网络拥塞控制算法，并深入研究了 TCP Vegas 算法和 AVQ 算法，通过对大量相关算法的研究和总结，数学模型的分析和大量仿真实验证明的基础上，在拥塞控制算法方面提出了自己的见解。

展望未来，拥塞控制算法领域还有很广阔的研究前景，特别是在控制科学突飞猛进的背景下，将控制技术、计算机技术、通讯技术结合起来综合解决网络拥塞问题的趋势越加明朗，也为以后网络拥塞控制的研究指明了方向。

下面给出今后研究工作的一些建议：

(1)对 TCP Vegas 算法的研究已经持续不短时间了，该算法设计机制新颖，是一种很有应用潜力的算法。本文中已解决了该算法的兼容性问题和网络非对称性问题，但在 TCP Vegas 算法中仍存在着诸如网络持续拥塞、算法公平性、线路变更等一系列有待解决的问题，这些问题在实际应用中都会时常出现，对算法的影响很大，如不解决这些问题，TCP Vegas 算法是不能可靠的应用到实际中去的。所以，未来对 TCP Vegas 算法的研究应着重解决该算法的各种缺陷，并提出一种能够兼顾各方面情况可以实际应用的综合算法。

(2)目前，网络端的拥塞控制算法日益得到人们的重视。在网络端拥塞控制研究中，目前主要集中在路由器上，发挥路由的队列管理和队列调节作用。在队列管理算法中目前应用较多的是自适虚拟队列（AVQ）算法，本文重点分析了该算法，并提出了一种能够提供区分服务的改进算法，但是自适虚拟队列算法还有许多待改进之处，如参数设定困难、队列长度抗干扰能力弱、存在一定链路损失等缺点，虽然本文提出了一种解决其区分业务的算法，但该算法在实际应用是否

可以和队列掉算法有效兼容，是否会产生新的问题等都未考虑，所以，今后对网络端拥塞控制算法，特别是 AVQ 算法的研究可以重点放到对以上问题的解决上。

(3)近年来控制理论飞速发展，控制技术的应用也日益广泛。本文分析了 PI 控制、模糊控制等控制技术在网络拥塞控制算法中的应用，但智能控制技术没有完全应用到网络拥塞控制中来，如果能将智能控制技术运用到网络拥塞控制算法中去，那么网络拥塞技术将实现质的飞跃。今后研究智能控制技术的相关理论，并将其中的适用的部分应用到网络拥塞控制中将是研究工作的新课题。

参考文献

[1] 徐昌彪, 鲜永菊.计算机网络中的拥塞控制与流量控制[M].北京：人民邮电出版社. 2007.10

[2] Jain R.Congestion Control in Computer Networks [J]. Issues and Trends, IEEE Network Magazine, 1990, 4(3):24-30.

[3] Peterson L L, Davie B S.Computer Networks: A Systerm Approach, Morgan Kaufmann Publishers, 2000, 12(5):37-42.

[4] 朱利,周俊辉.计算机网络的拥塞控制技术研究[J].计算机科学.1999, 6(12):76-79.

[5] K.Thompson, GJ Miller, R.Wilder, Wide-Area Internet traffic patterns and characteristics [J].IEEE Network, 1997, 11(6):10-23

[6] Cerf Viking R.A Protocol for Packet Network Intercommunication [J].IEEE transactions on communications. May 1974, COM-22:637-648

[7] RFC, TCP Congestion Control

[8] RFC2582, The New Reno Modification to TCP's Fast Recovery Algorithm

[9] RFC2582,S Floyd Henderson. The New Reno Modification to TCP's Fast Recovery Algorithm [Z], 1999

[10] Mathis M,Mahdavi J,et al.TCP Selective Acknowledgement Options[S].RFC 2018,1996:10

[11] Lee K, Renchandran K.An Integrated Source Coding and Congestion Control framework for Video streaming in the Internet [J].IEEE Computer Society, 2000, 8(11), 32-35.

[12] Andrea De Vendicitis, Andrea Baiocchi, Michela Bonacci, Analysis and enhancement of TCP Vegas congestion control in mixed TCP Vegas and TCP Reno network scenario [J]. Performance Evaluation, 2003, 53:225~253.

[13] Padhye J. Modeling TCP Throughput: A Simple Model and It s Empirical Validation [J]. Computer Communication Re-view, 1998 (4):457-462.

[14] Anirban Mahanti, Derek L. Eager, Mary K. Vernon.Improving multirate congestion control using a TCP Vegas throughput model [J]. Computer Networks, 48 (2005) 113–136.

[15] Hashem E, Analysis of random drop for gateway congestion control. Tech.Rept. MIT/LCS/TR-465, MIT Lab. For Comp.Sci, MA, Nov,1989

[16] C. Parsa and J. J. Garcia2Luna2Aceves, "Improving TCP congestion control over internet with heterogeneous transmission media", in Conf. Rec.1999 IEEE Int. Conf. Network Protocols, pp. 213-221.

[17] K.Thompson, GJ Miller, R.Wilder, Wide-Area Internet traffic patterns and characteristics [J]. IEEE Network, 1997, 11(6):10-23.

[18] Marc Greis' Tutorial for the UCB-LBNL-VINT Network Simulator ns. http://www.isi.edu/nsnam/ns/tutorial/.

[19] 孟川杰,黄宏光.TCP Vegas 的拥塞控制算法研究[J].网络与应用.2006, 12(10):54-59.

[20] Jacobson Van, Karels Michael J, Congestion Avoidance and Control.ACM Computer Communication Review, in Proceedings of ACM Sigcomm"88 Symposium in Stanford, CA, 1988

[21] Xiao Li, Lara-Rosario.Modeling an Electronic Component Manufacturing System Using Object Oriented Colored Petri Nets[C].Proceedings of ICECS'99.The 6th IEEE International Conference on Electronics, Circuits and System, 1999.12.

[22] Kevin Fall, Kannan Varadhan.The NS Manual [M].October 14, 2000.

[23] 徐雷鸣，庞博，赵耀.NS 与网络模拟[M].北京:人民邮电出版社,2003: 67-72.

[24] Eitan Altman, Tania Jimenez.NS Simulator for beginners [M].December 4, 2003.

[25] Jacobson V.Modification to TCP Congestion Avoidance Algorithm, email to the end 2 end list, April 1990.

[26] RFC .The Algorithm Research of RTCP-Based Real-time Streaming Transport Congestion Control .

[27] Hoe J.Improving the Start-Up behavior of a Congestion scheme for Tip in Sigcomm Symposiums on Communications[J], Architectures and Protocol, August 1966, 21(1)： 56-58.

[28] Lee K, Renchandran K.An Integrated Source Coding and Congestion Control framework for Video stream in the Internet[J].IEEE Computer Society,2000,4(6):62-67.

[29] Lawrence S. Brakmo, Larry L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet[J]. IEEE Journal on selected areas in communications, October 1995, 13(8):45-50.

[30] Liu Gang,Zhang DeYun,Liu Jing.Improving Throughput for TCP Vegas,The Journal of China University of Posts and Telecommunication.2004,11(2):60-65.

[31] Lowekamp B.Discovery and application of network Information school of computer science[J].Pittsburgh:Carnegic Mellon University,2001.

[32] 蔡小玲.网络拥塞控制的若干问题研究[D].2005.11

[33] Thomas Bonald.Comparision between TCP Reno and TCP Vegas: efficiency and fairness[J]..

Performance Evaluation, 1999.36(37):307-332.

[34] Kyungsup Kim, Chong-Ho Choi.Queue delay estimation and its application to TCP Vegas[J]. Computer Networks, 2003, 43:619-631.

[35] Chan Yi Cheng, Chan Chia Tai, Chen Yaw Chung, RoVegas: a router-based congestion avoidance mechanism for TCP Vegas[J].Computer Communication,2004,27:1624-1636.

[36] Padhye J. Modeling TCP Throughput: A Simple Model and It s Empirical Validation [J]. Computer Communication Re-view, 1998 (4):457- 462.

[37] Brakmo L, Peterson L.TCP Vegas: End-to-End Congestion Control in a Global Internet[J]. IEEE Journal on Selected Areas in Communications, 1999, 53(8):1465-1480.

[38] Anirban Mahanti, Derek L. Eager, Mary K. Vernon. Improving multirate congestion control using a TCP Vegas throughput model[J]. Computer Networks 48 (2005) 113–136.

[39] K.N. Srijith, Lillykutty Jacob, A.L. Ananda. TCP Vegas-A: Improving the Performance of TCP Vegas[J]. Computer Communications 28 (2005) 429–440.

[40] Braden B.Recommendations on queue management and congestion avoidance in the Internet[J]. RFC 2309, 1998.

[41] Misra V,Gong W,Towsley D.Fluid-Based analysis of a network of AQM routers supporting TCP flows with an application to RED[J].ACM SIGCOMM Computer Communication Review, 2000,30(4):151-160.

[42] Floyd S, Jacobson V,"Traffic phase effects in packet-switched gateways," ACM Comp. Commum. Rev, Apr.1991, 21(2):26-42.

[43] Tinnakornsrisuphap, P.Richard J L.Characterization of queue fluctuations in probabilistic AQM mechanisms.ACM SIGMETRICS Permance Evaluation Review,2004,32(1):283-294.

[44] Floyd S, Jacobson V.Random early detection gateways for congestion avoidance[J]. ACM/IEEE Transactions on Networking, 1993, 1(4):397-413.

[45] S.Floyed .Recommendation on using the"gentle" variant of RED. http://www.aciri.org/floyd/red/gentle.html.March 2000.

[46] Floyd S, Gummadi R, Adaptive RED: all algorithm for increasing the robustness of RED'S active queue management.Aug.2001.

[47] Ott T J,Lakshman T V.SRED:Stabilized RED.In: Doshi, B.ed. Proceedings of IEEE INFOCOM.New York, USA: IEEE Communications Society, 1999, 23(7):1346-1355.

[48] Woo-June Kim, Byeong Gi Lee.FRED-fair random early detection algorithm for TCP over

ATM networks.Electronics Letters. 34(2):152-154.

[49] W.Feng, D.Kandlur, D.Saha, K.Shin. Blue: A New Class of Active Queue Management Algorithms.Technical report.UM CSE-TR-387-99.1999.

[50] Kunniyur S, Srikant R.Analysis and Design of an Adaptive Virtal Queue (AVQ) Algorithm for Active Queue Management.ACM Computer Communication Review, 2001,31(4):123-134.

[51] Kunniyurs, Srikant R. Analysis and design of an adaptive virtual queue (AVQ) algorit hm for active queue management（R）. IEEE SIGCOMM, 2001.

[52] Choudhury A K, Hahne E L. Dynamic queue lengt h t hresholds for shared2memory packet switches[J]. IEEE/ ACM Trans Networking ,1998 , (6) :130-140.

[53] JP Gerry, Comparison of PID Control Algorithms, Control Engineering, 1999, 34(3):102-105.

[54] Hollot C V,Misra V,Toesley D.On designing improved controllers for AQM routers supporting TCP flows[A].Proc of IEEE INFO COM'01[C].2001:1726-1734.

[55] 汤兵勇，路林吉，王文杰. 模糊控制理论与应用技术[M].北京:清华大学出版社.2002.

[56] Chrysostomou C, Pitsilillides A.A Fuzzy Logic Congestion in TCP/IP Best-Effort Networks. In: Proceedings of the Australian Telecommunication Networks and Application Conference (ATNAC03), Melbourne, Australia, 2003.

[57] 冯宪林,龙腾飞.一种基于模糊理论的拥塞控制方法[J].微机发展.2004,14(8).

[58] 宋学军,刘民.一种基于模糊理论的拥塞控制算法[J].计算机工程与应用.2003,39(11):78-81.

[59] 薛质,潘理,李建华.基于模糊 RED 算法的 IP 拥塞控制机制[J].计算机工程. 2002,(10):78-90.

[60] Stiliadis D, Varma A.Efficient fair queueing algorithms for packet switched network [J]. IEEE/ACM Trans.on Networking, 1998, 6(2):175-185.

[61] Chonggang W, Li B, Sohraby K.AFRED:An Adaptive Fuzzy-based Control Algorithm for Active Aueue Management.In: The 28th Annual IEEE Conference on Local Computer Networks (LCN2003),2003.

[62] Parek A K,Gallager B.A generalized processor sharing approach to flow control in integrated services networks:the single node case[J].IEEE/ACM Trans.on Networking,June 1993,(7):344-357.

攻读硕士学位期间发表和完成的论文

[1] 王云涛,方建安,张晓辉,严伟锋,基于TCP Vegas的网络拥塞控制改进算法,计算机应用研究,2009,26(12):4645-4647.

[2] 严伟锋,方建安,王云涛,张晓辉,基于耦合混沌映射的彩色图像加密算法,微计算机信息 2010.7.(已录用)

[3] 张晓辉,方建安,严伟锋,王云涛,基于DSP的焊接机器人控制系统设计，微计算机信息 2010.10.(已录用)

致 谢

两年半的硕士生活一晃而过，论文即将完成之日，回首过去，思绪万千。首先，衷心感谢我的恩师方建安教授对我的淳淳教诲和悉心关怀，在我攻读硕士学位期间，他给予了我生活上、学习上无微不至的关心。他严肃的科学态度，严谨的治学精神，精益求精的工作作风，深深地感染和激励着我。从论文的开题到论文的定稿过程中，自始至终都倾注着恩师的心血。从导师身上我不仅学的许多专业知识,更学会如何为人处事，导师对我的教导将会使我受益终身。在此，我要向方导师致以最崇高的敬意和衷心的感谢！

感谢东华大学信息理学楼202实验室所有的同学。实验室拥有很浓厚的学术交流氛围，很多的构思都是和他们交流中得到的。特别感谢师哥唐漾博士对我论文的指导和帮助，给出了很多宝贵的意见。感谢师姐于冬梅博士，师哥郑达博士平常在学习和生活上对我的关心和帮助，感谢实验室的所有同学。

特别要感谢父母以及亲人的谆谆教诲、支持和理解，感谢他们在生活上给予我无微不至的关心。感谢所有给予我关心、帮助、指导的师长、亲人、同学和朋友们！