摘 要

随着 Internet 在全球范围内呈现出爆炸式的发展趋势，TCP/IP 协议的拥塞控制算法研究也就变得更加重要和紧迫。在此背景下，本文针对 SACK 算法的缺陷及存在的问题，进行了比较系统、深入的研究，提出了一种改进型的 SACK 算法：一种高效的 SACK 算法。用 2 到 4 个分组（大约 4K 字节）的窗口代替原来一个分组的窗口，并通过理论分析和网络仿真试验结果说明：改进型的 SACK 算法在大多数的情况下确实能够改善网络性能。

由于网络传输多媒体文件的迅猛增加，UDP 文件和 TCP 文件业务之间分配的公平性，就显得特别重要了。为此，论文研究了网络公平性问题，讨论了使用改进型的 SACK 算法对于 TCP 和 UDP 文件的公平性。本文提到过的几种重要算法的性能参数对照，均是在 NS 中进行仿真分析的；并通过模拟方法验证了改进型 SACK 算法的公平性。

论文的研究工作和结论对于研究和解决 Internet 的拥塞问题具有一定的参考价值；由于还没有在广泛的网络环境中经过中大量的试验论证，因此还有待进一步的研究和改进。

关键词：拥塞控制，SACK 算法，公平性

ABSTRACT

Congestion control is one of hot spots in the research field of TCP/IP. With the worldwide upsurge of Internet in recent years, congestion avoidance and control algorithms for TCP/IP become more and more imperative and urgent. By means of surveying the current research state of congestion control in depth, some shortcomings of SACK algorithm are pointed out in detail. Thereby, some improvements for SACK are discussed and a modified SACK algorithm is presented. In the modified algorithm, 2-4 segments (about 4K bytes) are used to replace the original one segment in the initial windows. In order to confirm the correctness and efficiencies of the modified SACK algorithm, some network simulation experiments are carried out. Based on the theoretic analyses and experimental results, it is showed that the modified SACK algorithm can improve network performance in many cases.

NS is one of the most popular network simulators. It can be used to analyze and evaluate network performance. Thus, NS is chosen as the simulator tool in this thesis. Here, the architecture and key components of NS are introduced. In addition, the special topic concerning setting the simulation scripts and analyzing the simulation results are discussed.

On the side, it is very important to further investigate the congestion control of TCP/IP. The detailed discussion for the congestion control of

TCP/IP is carried out. Meanwhile, the active and significant research directions for the improvement of congestion control algorithm of TCP/IP have been concluded as well.

Congestion control is a sophisticated task that cannot be absolutely solved via single protocol or algorithm. The feasible way to guarantee the stability and efficiency of network is to use various mechanisms to control the congestion from multiple aspects of the whole network.

The study on the congestion control of TCP/IP network in this thesis has some important for the correlative study of congestion control in the Internet.

KEY WORDS: congestion control，TCP-Fairness，SACK

原 创 性 声 明

本人声明，所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了论文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得中南大学或其他单位的学位或证书而使用过的材料。与我共同工作的同志对本研究所作的贡献均已在论文中作了明确的说明。

作者签名：刘相成　　日期：04年5月17日

关于学位论文使用授权说明

本人了解中南大学有关保留、使用学位论文的规定，即：学校有权保留学位论文，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以采用复印、缩印或其它手段保存学位论文；学校可根据国家或湖南省有关部门规定送交学位论文。

作者签名：刘相成　导师签名：[illegible]　日期：04年5月17日

第一章 绪论

1.1 研究的背景与意义

近年来，随着 Internet 在全球范围内呈现出爆炸式的发展趋势，使得 Internet 已经从初始的科研工具逐渐地转变为商业应用信息基础设施。

Internet 在完全成为一个理想的商业应用信息基础设施之前，还存在许多关键性问题，其中，一个问题是需要快速的引入新的、更复杂的分布式多媒体应用及支持关键性任务，包括针对国家安全、商业应用和社会目标的网络应用，但目前 Internet 的单一服务模式不能有效提供及保证这些网络所需的服务质量，从而限制了这些网络的广泛开展；另一个问题是 Internet 的社会用户和业务量正以指数级方式增长，使 Internet 的性能和资源利用率下降，主要表现为网络时延和数据包丢失率增大。

这主要就是由于 Internet 本身网络资源的不足、网络拓扑结构或传输路径的不合理，不良的控制算法，使得它极其容易发生网络拥塞现象。网络的拥塞控制问题已经成为制约 Internet 进一步发展的重要因素之一。TCP/IP 协议的拥塞控制算法研究也就变得更加重要和紧迫。

针对以上两个问题，一种最直接的方法是使网络具有尽可能高的网络带宽。然而，实践已表明，仅依靠提高网络带宽的方法，虽然可暂时缓解部分问题，但随后产生一些新的网络应用会更迅速地占用网络带宽，而使所有网络连接的性能下降。另外，即使在网络内部可提供过量网络带宽，由于网络负载的不可预见性，网络拥塞现象还是可能在网络边缘结点上产生，因此，为解决以上问题，还需要研究一些更为有效的、优化的方法。实质上这也就是网络拥塞控制问题。

自从数据网络诞生以来，拥塞问题就一直困扰着其发展。特别是在日新月异的网络技术的刺激下，网络规模急剧膨胀。随之而来的是：新型网络应用的不断涌现和用户数量迅速增加，使得 Internet 的流量飞速增长，其中除了传统的 WWW、FTP、Telnet 等数据流外，还出现了大量的实时多媒体数据流，由于网络中不同的数据流在路由器处交汇，因而给网络各个节点造成很大的负担，那么这就形成了越来越严重的拥塞问题。

Internet 近十几年来的迅猛发展，已证明 TCP 协议在设计上是非常成功的。根据 MCI 的统计，TCP 是目前 Internet 上使用最广泛的一种传输协议，因为：在 Internet 上总字节数的 95%以及总数据包数的 90%使用 TCP 协议传输[1]，拥塞控制是确保 Internet 的鲁棒性(robustness)的关键因素，也是各种管理控制机制和应

用（如多媒体通信中 QoS 控制与区分服务）的基础，可以毫不夸张地说，没有 TCP/IP 拥塞控制算法的引入，就不会有今天 Internet 的成功[2][3]。拥塞控制正在成为当前网络研究的一个热点问题。因此 TCP/IP 的拥塞控制（congestion control）机制对控制拥塞具有特别重要的意义。

1.2 网络产生拥塞的原因

1、 网路产生拥塞的根本原因

网络产生拥塞的根本原因在于用户（即端系统）提供给网络的负载（Load）大于网络资源容量和处理能力（Overload）[4]。其典型表现就是数据包时延增加、丢弃概率增大、上层应用系统性能显著下降（服务质量降低）等。拥塞一旦发生，如果不采取正确控制，拥塞会继续加重并最终导致网络崩溃。

2、 网路产生拥塞的直接原因

拥塞产生的直接原因主要有以下的三个方面：

（1）路由器存储空间不足。如果多个输入数据流共同需要同一个输出端口，并且入口速率之和大于出口速率，那么就会在这个端口建立一个队列。当然没有足够的存储空间，数据包就会被丢弃。对突发数据流更是如此。增加存储空间在某种程度上可以缓解这一矛盾，但当路由器有无限存储量，拥塞只会变得更坏，而不是更好，因为在网络里数据包经过长时间排队完成转发时，它们早已超时，源端会认为它们已经被丢弃，而实际上这些数据包还会继续向下一个路由器转发，这样也就浪费了网络资源，使得网络更加拥塞。

（2）带宽容量相对不足。低速数据链路对高速数据流的输入会产生拥塞。根据香农信息理论：任何信道带宽最大值（即信道容量）为 $C = B \log_2 (1 + S/N)$（N 为信道白噪声的平均功率，S 为信源的平均功率，B 为信道带宽）[2]。要求所有信源发送的速率 R 必须小于或等于信道容量 C。如果 R 大于 C，则在网络低速链路处就会形成带宽瓶颈，一旦当其满足不了所有通过它的源端带宽的需求时，网络就会发生拥塞[3]。

（3）处理器处理能力弱和速度慢也能引起拥塞[4]。如果路由器的 CPU 在执行排队缓存，更新路由表等功能时，处理速度跟不上高速链路，就会产生拥塞。同样，低速链路对高速 CPU 也会产生拥塞。

从理论上说，任何网络资源都是有限的，而且这些资源又有可能是共享的，那么这类网络资源必定会成为一种竞争获取的对象。因此，网络拥塞问题是计算机网络飞速发展的必然结果。

1.3 网路拥塞控制算法的研究现状

目前主要的网络拥塞控制算法主要有：TCP Tahoe、TCP Reno、New-Reno TCP、TCP SACK、TCP Vegas。

1、TCP Tahoe

Jacobson 在 1988 年改进了原来的 TCP，提出了 Tahoe。在早期实现的基础上，Tahoe 加入了慢启动、拥塞避免（加性增加部分）、快速重传机制。重传计时器的超时值得到了改进。Tahoe 的基本思想是探测网络的可用带宽，在拥塞时急剧的降低数据发送速率[5][6]。Tahoe 具备 RTT 估计和差错恢复的功能。

2、TCP Reno

1990 年，Jacobson 又提出了 Reno，Reno 的快速重传中包括了快速恢复，在三次重复的 ACK 后不进行慢启动，这使得在仅仅一个分组丢失时的 TCP 性能得到了提高[6][7]。Reno 的基本思想是用快速恢复取代了慢启动，在收到三个和以上相同的 ACK 时进入快速重传和快速恢复，在超时的时候进入慢启动。

3、New Reno

1996 年 Fall 和 Floy 在 Reno 的基础上提出了 New Reno。在 TCP Reno 的基础上进行了小小的改变，使得在一个窗口中有多个分组丢失时的 TCP 性能有所提高。在 Reno 中，处于快速恢复状态的发送方在收到第一个正常的确认（Partial ACK）后，就会立即离开快速重传/快速恢复。当存在多个分组丢失时，这一算法很难将它们全部恢复，New Reno 中的发送方只有在收到所有丢失分组的确认后，才会离开快速恢复状态，否则就继续进行快速重传/快速恢复，每一个 RTT 发送一个丢失的分组，直到原来窗口中丢失的分组已经全部被重传过。

4、TCP SACK

1996 年 Mathis、Mahdavi、Floy 和 Romanow 还提出了 Reno 的另一变形：SACK 采用选择性确认，而不是 GO BACK N 机制，进一步提高 TCP 在拥塞较严重且一个窗口中有多个分组丢失时的性能[9]。Reno 和 New Reno 在一个 RTT 内最多只能重传一个丢失的分组，如果一个窗口中有多个分组丢失的话，这就有可能导致网络中没有分组。SACK 的基本思想是接受方 TCP 发送 SACK 分组来通知发送方接受数据的情况。这样发送方只重传丢失的分组。SACK 在进入快速重传状态时，如果网络中的所有分组已经得到确认，那就会退出快速重传状态。

5、TCP Vegas

1994 年，L.S.Brakmo 等[7]提出了一种新的拥塞控制策略：TCP Vegas。由于 RTT 值与网络运行情况有密切关系，因此，TCP Vegas 通过观察 TCP 连接中 RTT 值改变来感知网络是否发生拥塞，从而控制拥塞窗口大小。但是 TCP Vegas 在 Internet 上是不大可能被广泛使用的，因为这里有一个致命的问题没有解决：在

竞争带宽时，使用 TCP Vegas 的数据流在带宽竞争能力方面极差，远不及未使用 TCP Vegas 的数据流，因此会导致网络资源分配的严重不公平性[9][10]。

从上面的分析可以看出：目前 TCP 协议主要的四个版本是：TCP Tahoe、TCP Reno、TCP New Reno 和 TCP SACK。TCP Tahoe 包括了 3 个最基本的拥塞控制阶段："慢启动"、"拥塞避免"和"快速重传"[11][12]。TCP Reno 在 TCP Tahoe 基础上增加了"快速恢复"方式。TCP New Reno 对 TCP Reno 中的"快速恢复"方式进行了修正，它考虑了一个发送窗口内多个数据包丢失的情况。在 Reno 版中，发送端收到一个新的 ACK 后，就退出"快速恢复"阶段[12][13]；而在 New Reno 版中，只有当所有的数据包都被确认后才退出"快速恢复"阶段[14]。

很明显，SACK 是目前所使用算法中性能最优的。TCP SACK 关注的也是一个窗口内多个数据包丢失的情况，它避免了之前版本的 TCP 重传一个窗口内所有数据包的情况，包括那些已经被接收端正确接收的数据包，而只是重传那些被丢弃的数据包。但是 SACK 算法也存在缺陷。

1.4 论文的主要工作、创新点及安排

拥塞控制这个问题从 80 年代末期，就有人开始研究。拥塞控制是 TCP/IP 协议研究的重要内容，近年来，随着 Internet 在全球范围内的迅速蔓延，TCP/IP 协议的拥塞避免和控制算法的研究与改进也变得更加重要和急迫。目前，国外在 TCP/IP 网络的拥塞控制方面的研究取得了很多进展，而在国内，这方面的研究进行得很少。

该文是在当今流行的 Linux 操作系统下，模拟仿真日常生活中 Internet 上的 windows 操作系统为主的现实网路运行状况。在总结已有的理论知识的基础上，进行了计算机仿真分析，对标准 SACK 算法进行了改进，其主要内容：

1、本文在介绍了网络拥塞控制的基本概念和目前 Internet 上的拥塞控制策略的基础上；重点分析了 SACK 算法的不足之处。

2、论文提出了改进型的 SACK 算法：即用 2 到 4 个分组（大约 4K 字节）的窗口代替原来一个分组的窗口，并通过理论分析和网络仿真试验结果说明：改进型的 SACK 算法在大多数的情况下确实能够改善网络性能。

3、随着使用网络传输多媒体文件的迅猛增加，UDP 文件和 TCP 文件业务之间分配的公平性，就显得特别重要了。为此，论文讨论了使用改进型的 SACK 算法对于 TCP 文件的公平性。通过仿真试验证明了：改进型的 SACK 算法确实能够提高终端用户平均吞吐量。并且不会损害 TCP 的性能。而且稳定性还可以。

4、全面系统地研究了拥塞控制领域最新研究成果，总结出目前拥塞控制的研究主要集中在三个方向：（1、）继续改进现有的 TCP 拥塞控制算法：本课题所

研究的工作即属于这一方向；（2、）研究在中间网络设备（主要是路由器和交换机）中采用一定的策略来避免和控制网络拥塞；（3、）利用价格等经济因素，限制用户对网络资源的需求，阻止拥塞的发生。

论文的研究工作和结论对于研究和解决 Internet 的拥塞问题具有一定的参考价值；虽然这个解决方法还不是特别完善，但在很大程度上，它确实能够解决目前所面临的问题。由于还没有在广泛的网络环境中经过中大量的试验论证，距正式使用，还有待进一步的研究和改进。

本文的主要内容安排如下：

第一章概述了本文研究背景与意义，提出了研究的动机，论证分析了拥塞产生的根本原因和直接原因，最后指出了本课题研究的内容和所做的工作。

第二章主要是探讨了拥塞控制理论，详细分析了拥塞控制的各种分类方法及其分类的依据，由此推导出：一个理想的拥塞控制策略应该具有的属性；并讨论了网络服务质量与拥塞控制的关系和流量控制与拥塞控制的关系。最后介绍了研究所使用的方法和工具。

第三章主要介绍 TCP 基于窗口端到端的拥塞控制策略。首先对 TCP 的拥塞控制的四个阶段进行了深入细致的研究；然后讨论了 TCP 基于窗口端到端的几种重要的算法的实现过程。

第四章提出了一种基于 TCP/IP 拥塞控制实现算法 SACK 的改进方式。在这里仔细的分析了标准 SACK 算法中存在的重大缺陷，详细的研究了改进算法对网络可能产生的好处，并给出了仿真的结果。经过仿真实验证明，改进型的 SACK 算法确实具有良好的可伸缩性。

第五章主要研究了改进型 SACK 算法的公平性问题，公平性是衡量网络性能的重要参数，这也是评价一个算法性能的关键指标。对于 TCP 友好性问题和 TCP 公平性理论进行了深入的研究，通过仿真试验证明：改进型 SACK 算法确实具有满意的公平性。最后讨论了特定网络上的 TCP 拥塞控制。

第六章介绍了改进型 SACK 算法在 Internet 上的应用情景。分析了当前 Internet 拥塞控制的现状，指出了 Internet 上拥塞的原因。由此得出结论：目前拥塞控制的研究三个主流方向。比较了 TCP 与 IP 拥塞控制这两种方法的差异，由此对改进型 SACK 算法未来在 Internet 上的使用情况做出展望。

第七章总结了全文的工作，给出了本文的研究结论，并对下一步研究工作做出了展望。

第二章　网络拥塞控制理论

2.1　拥塞的定义

因为存在许多不同的度量来描述拥塞现象，如传输延时、数据吞吐量、队列长度和网络效率等，人们对拥塞控制并无严格定义，甚至对拥塞的定义都无法完全统一。这儿给出几个比较类似并且相对被普遍认可的拥塞的定义。

定义 1： 如果因为网络负载增加而导致用户 I 的满意度降低，用户 I 则认为网络发生拥塞[15]。（如图 2—1）

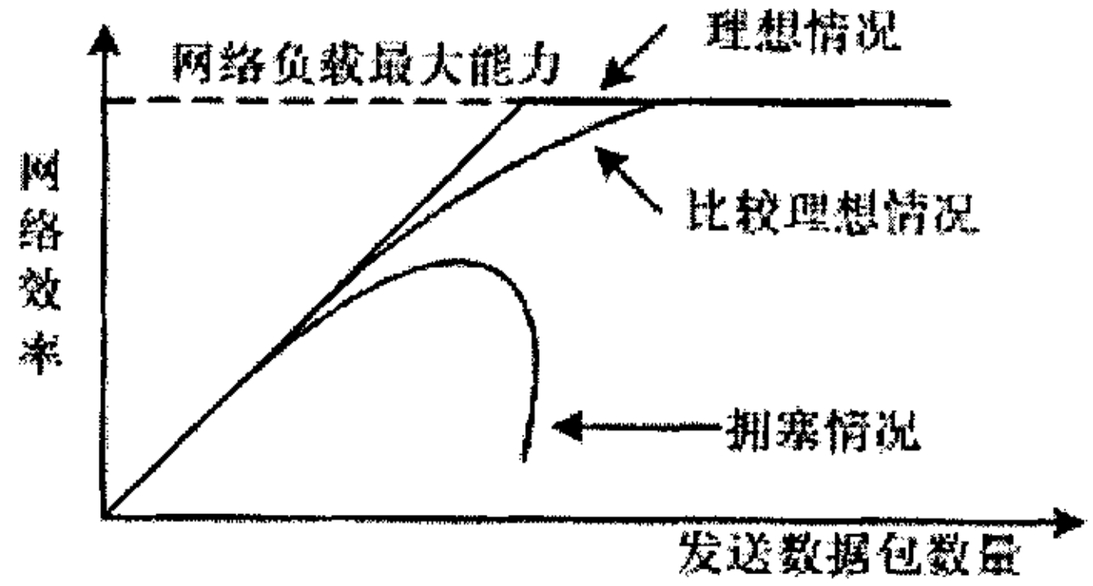


图 2—1　网络拥塞情况

上述定义并不是很完整，因为网络负载的减少在某种情况下并不一定能让用户的满意度增加。如图 2—2，假设：所有链路带宽为 1，接受者满意度即为所占用带宽。路由器队列调度规则为 WFQ，流 F1（从发送者 S1 到 R1）权值为 1，数据流 F2(从发送者 S2 到 R2)权值为 2，流 F3（从发送者 S3 到 R3）权值为 1。刚开始，三个数据源均发送数据，此时 R1 满意度为 1/3，R2 与 R3 满意度为 2/3。当 S2 停止发送数据后，R1 的满意度为 1/2，而 R3 的满意度则变为 1/2（网络负载降低，用户 R3 满意度反而降低）。

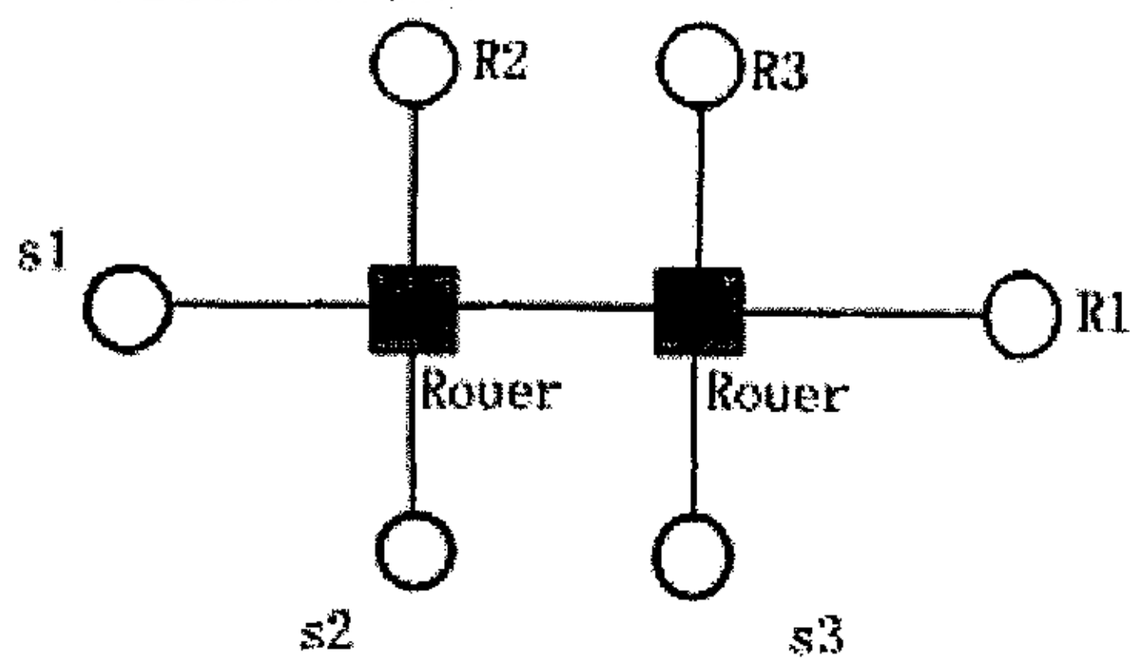


图 2－2　拥塞控制

定义 2： 如果由于用户 I 的网络连接性能（带宽、延迟、延迟抖动等）的变化而导致其满意度降低，用户 I 则认为网络发生拥塞[16]。

定义 3： 假设每个用户在抢占网络资源方面均是自私的（Selfish），当网络处于某种状态，如果每个用户都不能继续增加其满意度，此时网络处于 NASH 平衡点（NASH equilibrium）[17][18]。

2.2 拥塞控制算法分类

从网络设施上看，解决网络拥塞只能从两个方面入手：一是尽量避免拥塞发生；一是在拥塞发生时，采取控制策略来消除拥塞。我们所讨论的拥塞控制算法就包括了拥塞避免和拥塞恢复两部分。近几年来，有关文献中已提出大量的拥塞控制算法。我们对拥塞控制算法从以下几个角度分类：

1、从控制论角度出发，按作用方式分为：开环控制方式（或前动式/预防）和闭环反馈控制方式（或被动式）两类。

开环控制是在网络业务运行前，事先设计一个“好”的网络，确保它不会发生拥塞，而网络一旦运行起来，就不再采用调节措施。显然对网络这样不断变化的复杂系统，开环控制并不是理想的选择。开环控制比较典型的例子就是资源预留（如 RSVP 协议）。这类控制机制较适用于音频和活动图象业务。这种方法虽然是一种很直接的控制拥塞的方法，但由于要事先精确确定业务特性几乎是不可能的，因此为保证服务质量、避免拥塞、往往需要预留多余的网络资源（Over allocate resources），很容易造成网络资源利用率低下。另外，资源预留的复杂性和可扩展性差也是使得该方法不能被广泛采用的重要原因。

闭环控制方式是指各网络连接的发送端根据网络内部反馈信息，检测网络状态，遵循适当的策略动态的调整数据包发送率[19][20]。网络的反馈方式分为显式和隐式反馈。在显示反馈方式中，发送端直接从网络反馈信息中获取当前网络状态，如可用网络带宽等，相反，在隐式反馈方式中，发送端一般只能根据网络连接性能，如接受端返回的应答报文的时延变化或数据报丢失情况，判断网络状态。显然，网络反馈信息的准确性影响发送端调整效率。一般而言，反馈信息越多，端节点调整速率越有效，但是反馈信息过多，相应的会增加拥塞控制的复杂性。

同开环拥塞控制方式相比，闭环拥塞控制方式主要适用于无连接网络，基于反馈信息调整发送速率可更有效地利用网络资源，也更具有鲁棒性。但必须注意，闭环拥塞控制方式是否有效取决于发送端行为，只有保证发送端能快速和准确响应网络初始拥塞信息，才能真正达到拥塞控制目标。

2、从实施控制的类型上：拥塞控制可以分为基于速率（Rate based）和基于窗口（Windows based）两种类型。前者通常用数据突发的大小、速率或持续

时间的任意两个组合，而作为控制参数，它需要对即将发送的流量进行整形，以免爆发数据流（Bursts）的出现[21]；后者则给出发送方注入网络未经应答数据的最大值。即通过源端限制数据包的传送，并且不应答，这种机制在源端不需对发送的数据进行整形（Shaping）。

TCP 采用的是典型的基于窗口的控制方式，TCP 通过调整滑动窗口的大小控制发送到网络的数据量，这个窗口响应于接受者的缓存容量和网络情况，基于窗口的控制易于实现，并且可以限制注入网络的最大流量；基于速率的拥塞控制目前仍然是个开放问题。基于速率的控制方式则是按每秒发送的比特数量来控制数据流的发送，但这种算法在源端的传输率和网络相匹配的情况下，具有明显的优势，它可以缩减交换机所需的缓冲区数量[8]。所以它在本质上更适合于多媒体数据流的传输，因为多媒体数据流本质上都是基于速率的。

（3）从推断网络状态的反馈信息的类型上，可以分为显式拥塞控制（explicit）和隐式拥塞控制（implicit）[22][23]；前者，网络具有独立的拥塞控制过程，系统使用显式信号向执行流量控制的端点通告其状态（有效带宽。缓存容量等），可再细分为定性和定量控制两种；后者的拥塞控制信息是通过对数据传输的观察以获悉的，如果控制端使用流量测量或者通过诸如超时、重复 ACK 等隐含信号来推断网络状态，则为隐式控制方式。

（4）从实施控制的位置，可以分为在端系统上使用的源算法（Source algorithm）和在网络设备上使用的链路算法（Link　algorithm）[24][25]。源算法是指端对端的拥塞控制（end-to-end），它仅在网络边缘（即在 TCP 层）端系统或者主机处执行，此时中间节点仅负责向端系统产生和转发必要的反馈信息；而链路算法指路由器拥塞控制（router-based），是在网络设备（路由器或交换机）等中间节点处（即在 IP 层）执行的，多采用的是“弃尾”算法。

源算法的作用是在根据反馈信息调整发送速率；链路算法的作用主要是检测网络拥塞的发生，产生拥塞反馈信息。由于实际中网络内部的路由器和网络边缘的端系统都参与了拥塞控制，所以真正的问题在于拥塞控制的核心位置。

2.3　理想拥塞控制策略的属性

1、　拥塞控制的基本原理

造成网络拥塞的重要因素是系统存在的瓶颈和源端无限制的数据发送[5]。前者是由网络系统结构决定的，并随网络结构的发展而不断完善。然而网络的物理资源相对用户需求总是不足的，因而要解决拥塞问题必须在共享资源管理的基础上控制源端的数据发送率[6]。合理使用瓶颈处的资源，避免网络发生拥塞或将网络从拥塞状态恢复正常工作状态，就是网络拥塞控制的基本思想（见图 2—3）。

图 2－3　拥塞控制传输模型

2、　拥塞控制机制的设计目标：

也就是讲对于任何一种拥塞避免/控制策略我们希望它能同时满足以下几点要求：

（1、）　高效性（Efficiency）

高效性是指：在满足拥塞控制目标前提下，尽可能充分利用网络资源；拥塞控制的效率包括两方面含义：一方面是拥塞控制算法必须保证网络效率，使网络既不会欠载也不会过载；另一方面是拥塞控制应快速收敛于 NASH 平衡点，充分利用带宽，提高网络吞吐量，同时能够减少延时，使得对时延敏感的业务得到保证[26][27]。

（2、）　公平性(Fairness)

拥塞控制的一个重要原则就是应该能够抵御恶意用户（即不遵守拥塞控制）的行为，即能隔离恶意用户，防止这一类恶意用户降低其他用户的满意度。这个原则也就是我们讲的公平性，有时也称其为鲁棒性（Robustness）[28]。公平性是指：防止一些网络连接过度占用网络资源，而导致另一些网络连接不能公平的使用网络资源。

对 TCP（网络中主流传输协议）拥塞控制而言，主要考虑不同连接间的公平性（即协议内公平，Intra protocol fairness），对非 TCP 拥塞控制而言，则要考虑两种公平性：一是协议内公平，一是协议间公平，即 TCP 友好（TCP Friendly）[29][30]。

公平性可能是数据报网络中拥塞控制所面临的最具有挑战性的问题。它的困难在于终端用户和中间系统都不能得到网络状态的全部信息。因此，严格意义上讲，终端用户和中间系统都很难采用准确的控制措施。

（3、）　稳定性（Stability）

拥塞控制方案必须做的一个重要决定就是在瓶颈处应维持的最佳负载。当流量较小时，吞吐量就随着负载的增长而增加。当网络发生轻微拥塞时，数据包会在瓶颈处的缓冲区聚集；如果负载进一步增长，队列延迟会急剧增加。直至缓冲区发生溢出现象，缓冲区开始丢弃分组，网路有效吞吐量开始递减，此时网络进入严重拥塞状态。Keshav 建议一种最优化的定义：如果在每一个时间间隔[t1，t2]内瓶颈处没有缓冲溢出和带宽损失，这样一个信息流量是最优的。这种定义可允许更宽范围内的负载[31][32]。

实际上，很难让一个信息流正好工作在这样一个最佳的点上，一个具有合理长度的队列大小的工作点应该被认为是可接受。

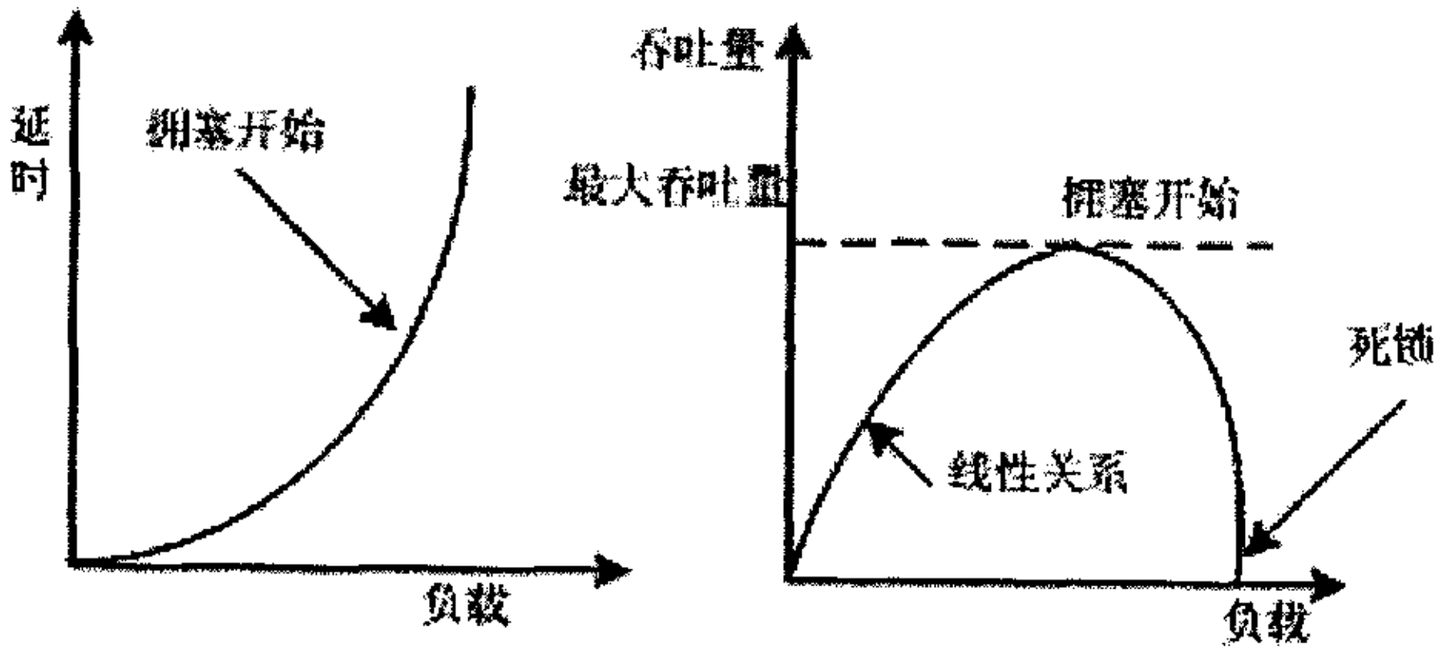
（4、） 可扩展性（Scalability）

扩展性是指：当网络连接数增多时，算法开销不会成为影响网络性能的一个主要因素；对单播（Unicast）而言，拥塞控制的可扩展性体现在对不同规模网络、不同链路状况、不同端断用户能力等均有效[32][34]。对组播（Multicast）而言，拥塞控制的可扩展性除上述外，还应能处理大量异构接受者的情况[33]。

需要指出的是，上述 4 种属性是拥塞控制的理想境界，大多数拥塞控制只能达到其中几种要求，如当具有不同传输时延的 TCP 连接竞争网络资源时，TCP 拥塞控制对具有较大传输时延的连接具有不公平性。尽管研究证明当竞争带宽的各连接的传输时延按比例减小时，小传输时延的连接对大传输时延连接的抑制性相对减弱，这种不公平性也减弱 ，但 TCP 拥塞控制的不公平性依然存在。

2.4　服务质量与拥塞控制

近年来，网络服务质量（Qos），尤其是 IP Qos 一词频繁的出现在科技文献中。实际上 Qos 控制就是通过对带宽、缓存等网络资源的管理与调度实现所传输数据流要求的一系列服务请求。如果实现得当，则拥塞就可以避免。然而，Qos 控制的目的远不是为了控制拥塞，而是要满足更高的要求，例如传输多媒体信息。随着网络多媒体及其它实时业务的大量出现，给原本不提供 Qos 的 Internet 提出了新的研究课题。于是，各种各样新的网络服务模型和 Qos 技术应运而生。其中比较突出的有综合服务模型（Int ser），区分服务模型（Diff serv）[35]，多协议标记交换（Mpls）[36]，流量工程和约束路由器等[37]。（如图 2—4）



延时-负载曲线　　输出—负载曲线

图 2－4　延迟、吞吐量与拥塞关系示意图

2.5　流量控制与拥塞控制

在实际拥塞控制中，我们往往通过控制流量的方法，预防网络处于严重超负荷状态，从而减少网络拥塞发生的可能性。可见拥塞控制与流量控制关系紧密。我们常用的流量控制算法有：漏桶算法[38]、基于窗口的流控[39]、基于速率的流控、基于信用卡的流控等[40][41]。

拥塞控制与流量控制密切相关。拥塞控制会抑制源方发送数据，起到流量控制的作用；相反，好的流量控制可以避免或推迟拥塞的发送。但是，拥塞控制与流量控制又存在一定的差异。前者是一个关系网络全局的问题，它意在解决路由器等中间链路设备的瓶颈问题，它的规则具有“社会性”，要求做到“公平合理”后者只涉及源方到目的方端到端的传播，是为了确保源方发送数据时不超过目的方的接收能力，解决目的方的瓶颈问题，这是二者的“个体”行为，不存在公平性问题。所以它们之间必须相互协调、互相约束，这样才能做到既保证网络效率又不会发生过度拥塞。

2.6　研究方法和工具

计算机网络性能评价的基本方法包括测量、试验、模拟和解析四种。测量方法和试验方法只适用于现已存在的网络系统。测量是对一个实际的运行系统作全面、真实的性能分析。试验更侧重于分析具体的实验问题，是一种初步验证。由于试验一个系统是非常复杂的，测量和试验方法一般只用于验证一个网络系统的特定方法，而不是用来比较不同方法对网络性能影响的情况。

模拟方法和解析方法用于分析一些可重构、抽象的网络模型或一个未实现的网络系统的各种抽象。解析方法是对网络系统的最高级抽象，所求得的结果是最严格的。使用解析方法，能完整的了解网络系统的可能行为。但解析方法只局限于一些比较简单的网络模型模拟模型，许多复杂模型不可能用纯粹的数学方法来求解。另外，为使模型求解，往往过分抽象一个网络系统，而忽略其中的一些关键因素，使所求得结果不可信。模拟方法是对解析方法的一种有力补充补充，不仅用于验证网络验证抽象模型的有效性和解析结果的正确性，而且还是用于分析一些不能用数学方法求解的复杂模型的行为。特别地，由于 Internet 的拓扑结构、业务模型、以及自适应性拥塞控制作用的所具有的复杂性，使模拟方法成为研究 Internet 问题的一个最主要方法，采用模拟方法，可方便的改变各种网络参数，比如网络拓扑、带宽、时延、输入负载等，而能更好的分析不同策略和算法在各种情况下的性能，却不需要实现实际的网络系统。

NS(Network simulator)是一个由 UCB/BNDNL 实验室实现的，离散事件驱动的，面向对象的多协议模拟工具[42][43]。它实现了应用当前 Internet 的各种网络协议和算法，包括单点、多点的路由算法、传输层、会话层和应用层协议，以及

一些调度和缓冲管理算法，它用 C++编写，使用 OTCL 解释器作为前端程序。在内部，它分别维持一个接 OTCL 面向对象方法表示的类层次结构和一个接 C++面向对象方法表示的类层次结构。两个类层次结构存在一对一的影射关系。使得可用任意一种语言修改一个对象的变量值。同时修改对应语言中绑定的对象变量。

用 NS 模拟分析一种算法或网络协议的基本过程为：首先用 C++语言定一个新的派生类。例如，本文为分析改进型的 SACK 算法性能，就先定义了一个新的 MSACK 类，在 MSACK 类内部实现改进型的 SACK 算法，调用 OTCL 类构造方法生成一个 OTCL 的 MSACK 类，调用 bind()过程绑定两个对象类之间的内部变量，再重新编译 NS 可执行程序。第二步，编写一个 OTCL 语言的描述文件，定义网络模型和参数，包括网络拓扑、链路带宽、时延、缓冲大小，定义调度算法和缓冲管理算法，以及统计变量或跟踪文件。第三步是分析模拟结果。

第三章　基于端对端的 TCP 拥塞控制

3.1　TCP 的拥塞控制思想

虽然协议规范和控制策略有着很大的区别，但是 TCP 拥塞控制技术必须是建立在 TCP 协议规范基础之上的。TCP 拥塞控制技术主要包括控制机制和控制策略。控制机制从宏观上确定拥塞控制的模式，如是主动的还是被动的。TCP 采用发送方主动、接受方积极参与的拥塞避免和控制机制。控制策略则从微观上明确描叙实现拥塞避免和控制所采取的策略，TCP 往往要使用拥塞控制算法来描述这些策略。

从网络设施上看，在直观上利用计算机技术，解决网络拥塞只能从两个方面入手：一是尽量避免拥塞发生；一是在拥塞发生时，采取控制策略来消除拥塞。前者是为了预防，力图使网络维持在最佳状态，从而避免拥塞；后者是为了补救，进行拥塞恢复。要真正避免网络拥塞是可能的，但这样做会牺牲网络的资源利用率。在追求公平、高速、高利用率的网络环境下，采用这样保守的方法显然不合理的。因此，采用拥塞避免与拥塞控制相结合的方法更为合理。

3.2　TCP 的拥塞控制技术

3.2.1　TCP 的拥塞控制机制

从理论上看，通过引用一个物理学方面的定律“质量守恒定律”，拥塞问题就可以迎刃而解。它的基本思想是：在网络中只有当一个旧的数据包离开（如已被发送）网络时才能向网络注入一个新的数据包。TCP 通过动态调整滑动窗口大小来试图达到这个目标。

TCP 采用发送方主动、接受方积极参与的拥塞避免和控制机制。为了论述的方便，在此先把与 TCP 相关的几个重要概念予以简单说明：

1、　TCP 的层次结构

为了降低设计的复杂性，大多数网络都由互相独立的几层组成，每一层完成不同的具体的功能，低层向高层提供服务.Internet 协议体系结构(TCP/IP 协议栈)分为四层，即链路层、网路层、传输层及应用层[44]。链路层包括低级别的元件诸如网络设备驱动器及必要的硬件、网络层负载路由功能及网络中包的投递。在 TCP/IP 协议族中，网络层协议主要指 IP 协议[45]。IP 提供一种不可靠的无连接的数据报投递业务。传输层提供可靠或不可靠的端到端的数据传输。最为大家熟知的熟知传输层协议是分别提供可靠和传输控制协议 TCP 和用户数据报协议

UDP。TCP/IP 协议栈的第四层是应用层，它由一些利用下层服务来提供应用的应用协议诸如 Telnet、FTP、SMTP 组成[46]。通常来讲前三层组成了主机操作系统的内核，应用层为用户进程。

2、 TCP 的 ACK 时钟特性

TCP 是以 ACK 为时钟的，是指发送方只能在收到接收方对其曾发送过的一个分组的确认后，才能发送一个新的分组且发送方收到 ACK 的速率同接收方的速率是一样的[47]。这样一来，原则上说，信源的发送速率同整个连接上最慢的部分的传送速率是一致的。

3、 丢失检测

分组丢失可以通过接收不到 ACK 来判断。在 TCP 中，发送方可以通过超时或三个复制的 ACK 辨别两种不同种类的分组丢失[48][49]。

4、 超时

发送方在收到一个 ACK 后确定一个最大等待时间间隔，若在这个等待时间间隔内，一直没有收到新的 ACK，则发送方认为接收方尚未确认的分组已经丢失，开始启动重传机制。该时间间隔的取值(RTO)是否合理对于 TCP 的性能好坏有重要的影响。

3.2.2 TCP 的拥塞控制策略

下面我们就再来研究一下网络拥塞控制阶段的大致工作原理：

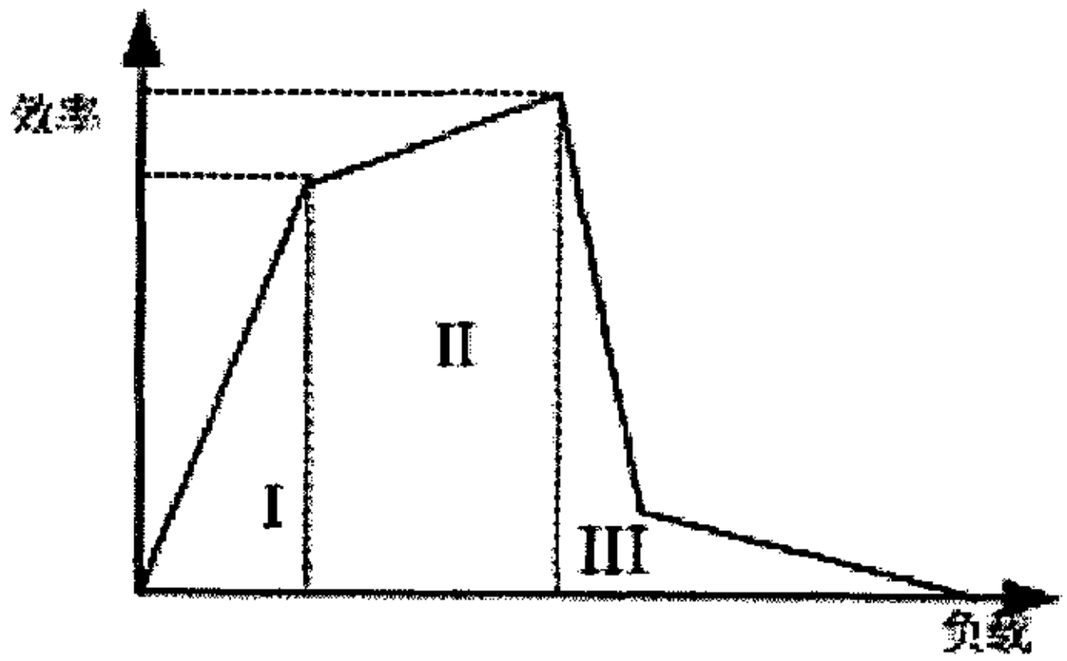


图 3-1　负载和效率的关系

1、慢启动阶段（Slow　start）

当刚开始传输数据时，TCP 在这个新连接中，并不知道当前的网络状况。因此，一开始 TCP 必须慢慢地探测当前网络的可用带宽，以免出现由于发送大量突发数据导致网络拥塞的情况。慢启动算法就是为解决这个问题而设计的，它通常用于连接的开始阶段和由于重发定时器超时而引起的重传丢失。

用来避免出现上述情况的算法称为 slow start（缓慢启动）。其运行原理是：开始，拥塞窗口被初始化为一个数据包大小(一个数据包缺省值为 536 或 512byte)[51]。源端按 cwnd 大小发送数据，每收到一个 ACK 确认，拥塞窗口 cwnd 就增加

一个数据包发送量。发送方可以发送的最大窗口为拥塞窗口(cwnd)和接收方通告窗口(awnd)之间的最小值。这样，cwnd 的增长将随 RTT 呈指数级(exponential)增长：1 个、2 个、4 个、8 个……。源端向网络中发送的数据量将急剧增加[50][52]；直到拥塞窗口增加到慢启动阀值时，源端一次所发送的信息量就达到甚至是超过了网络容量，此时路由器也开始丢包，这表明源端的拥塞窗口值已经太大了。

从上面的　3—1 可以看出：其具体的工作过程是：将要发送到网络的新的数据包发送速率等于从接收方返回的确认消息的速率。这种方式使发送方可发送的信息量呈指数增长趋势（尽管它并不是真正的指数增长，因为接受方可能会延迟它的 ACK，特别当接受方采用每收到两个报文端才发送一个 ACK 的确认机制时），这样能使会话尽快从区域 I 进入区域 II（如图 3—1）。

这个算法虽然称为慢启动算法，但实际上，它的发送率一点也不慢。要达到可利用的最大窗口 W，只要时间 RTT log2　awin（R 为往返时间）。早期 TCP 的实现只是当通信的另一端在不同的网络上才使用该算法，现在，TCP 实现总是使用该算法。

2、拥塞避免阶段（Congestion　avoidance）

当从高速局域网向低速广域网传输数据包时，或者到达一个路由器的通信量超出了它的处理能力时，就会发生拥塞现象。拥塞避免就是用来处理这种丢包的方法。当出现包丢失时，就表明源端和目的端之间的网络的某个地方出现了拥塞，包丢失有两种指示：超时和接收到重发的 ACK。

假设由于损坏而造成丢包的概率远远小于 1%，那么当发现超时或收到 3 个重复的 ACK 确认时，就表示在源端与接收方的网络中某处出现了拥塞，此时就应该进入拥塞避免阶段[19][52]。慢启动阈值被设置为当前 cwnd 的一半；如果已经超时，cwnd 被设置为 1。如果 cwnd=ssthresh，则 TCP 重新进入慢启动过程；如果 cwnd>ssthresh，则 TCP 执行拥塞避免算法，cwnd 在每次收到一个 ACK 时只增加 1/cwnd 个数据包（这里将数据包大小 segsize 假定为 1）。拥塞窗口不再是按指数级增长，而是改为按线性增长；即 cwnd 线性增长。

发送方在收到新数据的应答后，即每收到一个非重复 ACK 时线性地增长 cwnd，即

cwnd＋＝SMSS*SMSS/cwnd（单位为字节）　　公式（3—1）

其中，SMSS(Sender Maximum Segment Size)是指发送方所能传送的最大报文段长度。在遵循每个 RTT 数据 cwnd 比原来增加一个最大 TCP 报文段所对应的字节数的基本原理下，该公式提供了一个可以接受的近似值。如果接受方对收到的每个报文段都进行确认，则该公式的值在每个 RTT 中会略小于一个报文段；如果 cwnd 值非常大（>SMSS×SMSS）时，推迟网络拥塞的发生，即拥塞避免

的线性增长使得会话尽量保持在区域 II 中（如图 3—1）。这样就使发送端能够在较长的一段时间内保持较高的数据传输率。

3、**快速重传阶段**(Fast　retransmit)

无论缓慢启动或拥塞避免，cwnd 总是增长，使得会话迅速从区域 I 进入区域 II 后又不可避免地进入区域 III，这样负载就逐渐增大并最终导致网络拥塞。一旦数据包丢失，发送方只能在定时器超时之后，才启动数据重发过程，这样往往不能及时反映数据丢失。所幸的是对于源端，在收到重复的 ACK 确认后，主要是由于数据丢失或者是由于网络重新排列数据引起，所以当收到 3 个或 3 个以上重复的 ACK 确认时，就把 ssthresh 值设置为当前拥塞窗口的一半，并立即重传数据包，而不是等待重传定时器超时。图 3—2 和图 3—3 反映了拥塞控制窗口随时间在四个阶段的变化情况。

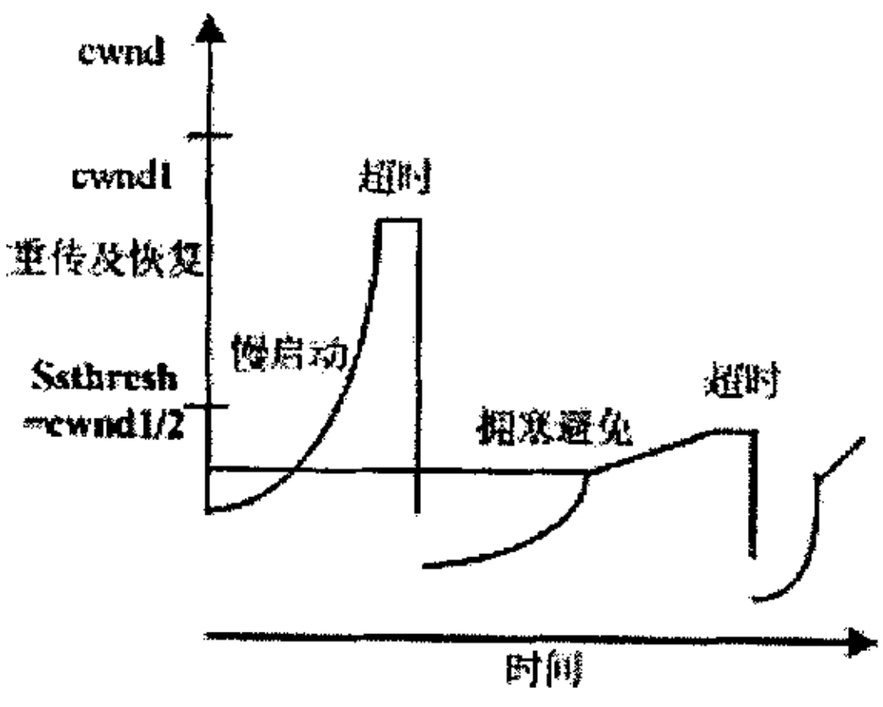


图 3－2　慢启动和拥塞避免

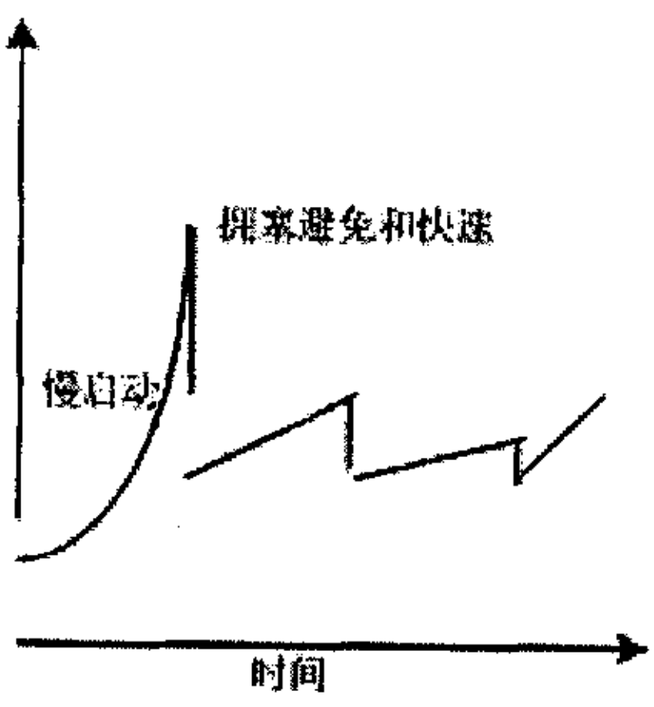


图 3－3　快速　传和恢复
重

4、**快速恢复阶段**（Fast　recovery）

在快速重传可能丢失的报文段之后，TCP 就执行拥塞避免算法（而不是缓慢启动算法），这就是快速恢复算法。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[35]。这样做可使得窗口在中度拥塞发生时，网络仍然具有较高的吞吐量。此外，快速恢复也可减少快速重发造成的 cwnd 急剧振荡，使得会话尽量能从区域 III 后退到区域 I 和区域 II 相邻的位置，而不是区域 I 的最左端。

快速恢复和快速重传算法描述如下：

```
step 1 : if (dupacks==3){
            ssthresh=max(2,cwnd/2);
            cwnd=ssthresh+3*segsize;
         }
```

step 2：重传丢失的分组

step 3：此后每收到一个重复的 ACK 确认时，cwnd=cwnd+1

step 4：当收到对新发送数据的 ACK 确认时，cwnd=ssthresh

实际上，收到重复的 ACK 只是表明有一个数据包丢失了，也就是说，此时源端与接收方之间仍然存在着数据流，因此，发送方不必进入慢启动阶段而急剧减少数据流。

最后，我们来分析一下会话负载与网络效率之间的关系，如图 3－1 所示。从图 3－1 中可以看出，若处于负载较小的区域 I，随着负载的增加，效率也相应提高：一旦负载进入某个合适的区域 II，效率不再随着负载的增加而显著提高；若负载进一步加重到 III，效率将迅速下降，也就是通常观察到的“拥塞崩溃”现象。当然，我们总是希望网络能稳定运行在区域 II，因为这样既能避免负载过低造成效率下降（区域 I），又能避免负载过高造成拥塞崩溃后的效率底下（区域 III）。更进一步，如果考虑网络中存在竞争资源的多个会话，我们希望网络能运行在区域 I 和区域 II 交界的位置，尽管区域 II 略微降低负载会减小效率，但释放负载会让处于区域 I 的其他会话获得更显著的效率提升，以优化整个网络的性能。

传统 TCP 协议仅有隐式的拥塞控制，很难保证稳定地运行在合理的区域，通常会在区域 I 和区域 III 之间振荡。现在，缓慢启动、拥塞避免、快速重发和快速恢复都已成为 TCP 协议标准实现所必须具备的拥塞避免和控制算法，通过这标准控制算法的协调操作，可以较好地解决这个问题。

3.3　TCP 拥塞控制算法设计的困难

拥塞控制算法设计的关键问题：是如何生成　馈信息和如何对反馈信息进行响应。拥塞控制算法的设计困难具体表现在以下几个方面：

（1）算法的分布性[46][50]：拥塞控制算法的实现分布在多个网络节点中，必须使用不完整的信息完成控制，并使各个节点协调工作，还必须考虑某些节点工作不正常的情况。

（2）网络环境的复杂性。Internet 中各处的网络性能有很大的差异，算法必须具有很好的适应性[47]；另外，由于 Internet 对报文的正确传输不提供保证，算法必须处理报文丢失、乱序到达等情况。

（3）算法的性能要求[42]。拥塞控制算法对性能有很高的要求，包括算法的公平性、效率、稳定性和收敛性等某些性能目标之间存在矛盾，在算法设计时，需要进行权衡。

（4）算法的开销[52]。拥塞控制算法必须尽量减少附加的网络流量，特别是

在拥塞发生时，在使用反馈式的控制机制时，这个要求增加了算法设计的困难，算法还必须尽量降低在网络节点（特别是网关）上的计算复杂性。目前的策略，是将大部分计算放在端节点上完成，在网关上只进行少量的操作，也只有这样才真正符合 Internet 的基本设计思想。

3.4　TCP 的几个重要算法

TCP 解决拥塞是采用基于窗口的端到端（end to end）的算法[8]，其典型代表是 Tahoc 算法和 Reno 算法。

1、　Tahoc 算法

（1）　慢启动（Slow Start）。窗口大小以指数速度增加：在建立连接时，发送方将拥塞窗口（cwnd）的大小设置为一个 TCP 段的最大字节数（mss），即 cwnd←mss，发送方最大的发送未应答的数据为 min{wnd，cwnd}，其中的 wnd 为接收方窗口。若发送方获得一个对新数据的应答，则 cwnd 按指数增加，即 cwnd←cwnd + mss。cwnd 按指数增长一直持续到 cwnd 超过慢启动阈值（ssthresh）为止，初始值通常设为 65535bytes[11]。ssthresh 值的设置方法为：如果分组超时或收到多个重复应答，则 ssthresh=max{2mss，min{wnd，cwnd}/2}。

（2）拥塞避免（Congestion Aviodance）。窗口大小以线性速度增加：在 cwnd 按指数增长到超过门限值时，即 cwnd>ssthresh 时，发送方收到对新数据的应答后，不再是按指数增长，而是改为按线性地增长 cwnd，即 cwnd←cwnd + mss*mss/cwnd，即为了避免拥塞的发生降低了 cwnd 增长的速度[11][12]。其对应的控制示意图见图 3－4。

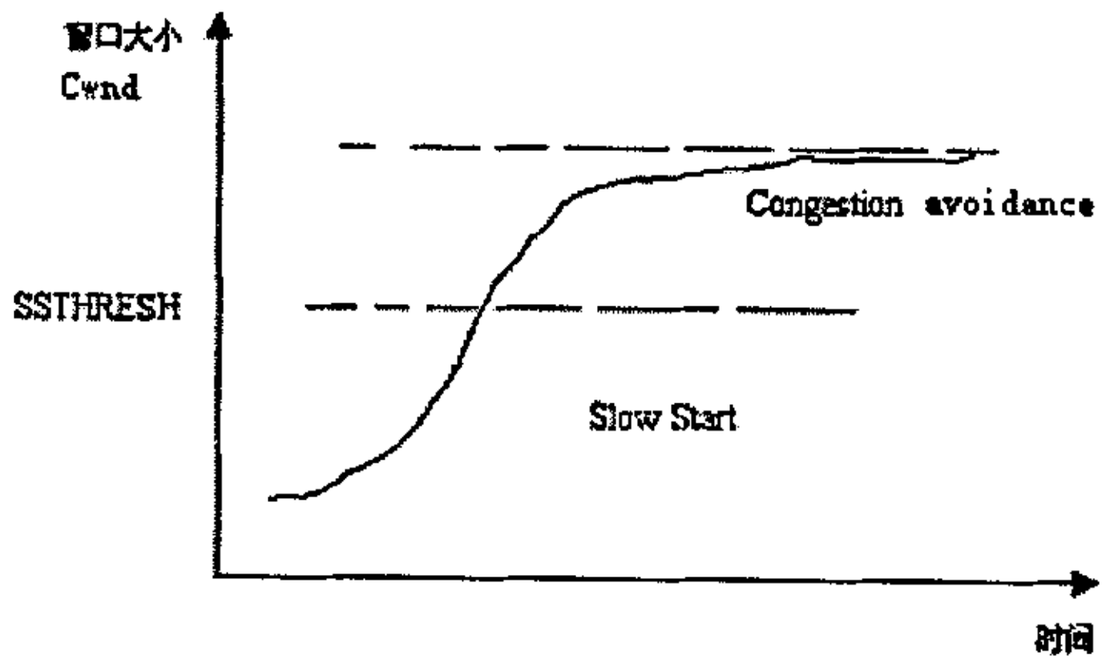


图 3－4　慢启动和拥塞避免算法控制图

上述两个过程的算法描述如下（C++语言）：

```
{
    IW=min(4*mss，MAX(2*mss，4380bytes))：      //  IW 为 cwnd 的初始值
}
Switch(cwnd)
{
    Case cwnd<ssthresh：        cwnd=cwnd+mss                    // 执行 Slow  Start
    Case cwnd>=ssthresh：       cwnd=cwnd+mss*mss/cwnd          // 执行 Congestion  Avoidance
}
```

（3）快速重传（Fast　Retranmit）。不论是按指数增长，还是按线性增长，其结果都会导致拥塞，因而在 Tahoc 算法中还引入了快速重传技术。所谓快速重传机制，就是当接受到多个对同一个 TCP 报文的相同应答时，发送方就推断已经发生了丢包，而没有必要等到超时重传计数器（RTO）超时，就立即重传相应的包[13]。Tahoc 算法能对往返时间（Rtt）的估计量进行修改，并把 Rtt 设置为 RTO 的基值。

2、 Reno 算法

Reno 算法修改了 Tahoc 算法的快速重传为快速恢复（Fast Recovery）机制[14]。当数据包超时时，cwnd 会被重置为 1，重新进入慢启动，这会导致过大地减小发送窗口尺寸，降低了 TCP 连接的吞吐量。所以快速重传和恢复就是在源端收到 3 个或 3 个以上重复的 ACK 时，就断定数据包已经丢失，重传数据包，同时将 ssthresh 设置为当前 cwnd 的一半，而不必等到超时重传计数器 RTO 超时。Reno 算法的理想情况是在一个窗口中单包丢弃时，Reno 算法中的发送方在每个往返时间（Rtt）中最多重传一个包。

以下是 Reno 拥塞控制的算法：

```
Initinal()：                          //  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(缺省值)
If(cwnd<ssthresh)     cwnd=cwnd+1;                    // Slow  Start
Else                          cwnd=cwnd+1/cwnd;          // Congestion  Avoidance
Relay timeout：                    ssthresh=max(2，min(cwnd/2，awin));
cwnd=1;
```

3、 SACK 算法

前面这两种算法，在单包丢弃时，效果是不错的，但如果在同一数据窗口中出现多包丢弃的情况下，它们的性能都有较大局限性。后来出现了基于选择应答（SACK　selective　acknowledge）的算法，它较好的解决了同一数据窗口中多包丢弃的问题[6]。

这种算法的基本原理是这样的：它是对 Reno 拥塞控制算法的一种简单改进。它们在扩大和减小拥塞窗口上使用的是相同的方法。在 TCP 中增加 SACK 并不改变拥塞控制基本阶段的方法，SACK 保留了 Tahoe 和 Reno TCP 在报文失序时的健壮性，且在必要时使用重传计时等待来进行恢复。SACK 和 Reno 的主要差别在于当多个报文从一个数据窗口丢失时的性能和所采取的对策。

SACK 和 Reno 一样，当数据发送端收到 ssthresh 个重复的 ACK 时就进入快速恢复阶段，发送端重发报文，并把拥塞窗口减小一半。在快速恢复阶段中，SACK 保持了一个变量 pipe，用它来表示出现在路由器上的报文的估计数量，这与 Reno 实现的机制不同。当 pipe 小于拥塞窗口大小时，源端只发送一个新报文分组或重发一个老报文时，pipe 值就加 1；而当发送端接收了一个带 SACK 选项的重复 ACK 报文，表明新数据已被接收者接收时，pipe 值就减 1。用 pipe 变量使何时发送报文和发送哪个报文分离开来。

如果有一个重发的报文丢失，SACK 实现可以在重发等待时间内察觉，从而重发丢失的报文，然后再进入慢启动，而在收到一个“恢复 ACK”，确认后了进入快速恢复时出现的数据后，发送端就退出快速恢复。

另外，SACK 发送端对快速恢复期间收到的“恢复 ACK”还使用了特殊的处理方法，对这些 ACK，发送端对 pipe 变量值每次减 2 而不是减 1。

4 、 Vegas 算法

在拥塞避免阶段，Vegas 算法中的 cwnd 值由以下公式（3—2）决定[17]：

$$
\begin{cases}
cwnd(t+\Delta t) = cwnd(t) + 1, & \text{当} \quad diff < (a / base\_Rtt) \\
cwnd(t+\Delta t) = cwnd(t), & \text{当} \quad (a / base\_Rtt) = diff = (\beta / base\_Rtt) \\
cwnd(t+\Delta t) = cwnd(t) - 1, & \text{当} \quad (\beta / base\_Rtt) < diff
\end{cases}
\qquad \text{公式（3—2）}
$$

其中 diff=cwnd(t)/base_Rtt－cwnd(t)/Rtt，Rtt 是观察到的回路响应时间，base_Rtt 是所观察到所有 Rtt 的最小值，a 和 β 是两个常数。上式表明如果所有数据包的 Rtt 值稳定不变，那么拥塞窗口 cwnd 将保持不变。

但是 TCP Vegas 在 Internet 上是不大可能被广泛使用的，因为这里有一个致命的问题没有解决：在竞争带宽时，使用 TCP Vegas 的数据流在带宽竞争能力方面极差，远不及未使用 TCP Vegas 的数据流，因此会导致网络资源分配的严重不公平性[22][23]。

目前 Internet 上广泛使用的拥塞控制协议是：Tahoe TCP；基于它的改进协议主要有 Reno TCP、New-Reno TCP 以及 SACK TCP 协议。深入研究以上几种协议可以看到：这些协议本质上都或多或少与慢启动阶段、拥塞避免、快速恢复以及

快速重传阶段相关联；都是使用诸如确认、超时、重复确认等隐含信号来推断网络状态，并利用反馈来修正数据源的发送窗口，控制注入网络的业务量，以达到其缓解网络拥塞的目的。

第四章 一种改进型的 SACK 算法

4.1 SACK 算法研究状况

近年来，许多科研人员致力于 TCP 拥塞控制策略的研究，针对 TCP 拥塞控制中存在的问题进行了方方面面的改进，以便使它能够更好的符合各种网络应用的需要，适应网络动态变化的特性，使网络运行在最佳状态。国外在 TCP/IP 网络的拥塞控制方面的研究取得了很多进展，而在国内，这方面的研究进行得很少。

在拥塞控制源算法的研究中，使用“管子”模型来抽象两个端节点之间的网络。“管子”的性能（包括带宽、延迟、丢失率等），在一定时间范围内是相对稳定的，这样端系统可以使用历史信息来估计未来的情况。“管子”模型的一个重要特征就是“报文守恒”（Pack conservation）.也就是讲：控制的目标是使得网络中报文的个数保持恒定，在报文离开网络之前，不向网络中加入新的报文。TCP 拥塞控制的 SACK 算法：就是典型的“管子”模型。

使用“管子”模型要求端节点之间的网络性能是相对稳定的。文献指出端系统之间的网络性能在分钟量级上是相对稳定的；另一个因素是路由的稳定性。文献指出，端节点之间的路由是相对稳定的，测试数据中保持 10 分钟以上的路由超过 95%，其中 68%的路由保持 1 天以上，所有这些数据都说明是使用“管子”模型是合适的[48]。

4.2 标准 SACK 算法的缺陷

经分析发现：按以上三种算法思想，在 TCP 实现过程中，最初的报文丢失都是在快速重传阶段所发现的。在实现过程中 Tahoc 算法性能稳定，但会出现暴发报文现象。Tahoc 在通过快速重传过程后，再用慢启动来恢复，而与从窗口中丢失的报文的数量无关。Reno 实现避免了暴发报文现象，在只有一个报文从数据窗口中丢失时，性能最佳；当有两个报文丢失时，发送端将先后两次进入快速重传和快速恢复，没有必要地两次减小了拥塞窗口的大小；当有三或四个报文丢失时，Reno 必须等待重传定时器超时，健壮性较差，丢失的报文多。

SACK 使用“管道”（pipe）变量表示在发送路径上损失的数据包的数量。用 ssthresh 判断拥塞是否发生。在文[15]中，通过一系列的仿真试验，对 Tahoc，Reno，New-Reno 和 SACK 这四种 TCP 的拥塞控制机制进行了比较研究，讨论了四种算法在丢失 1，2，3 和 4 个数据包的情况下的性能。结果表明：Reno 优于 Tahoe，New-Reno 和 SACK 则优于 Tahoe 和 Reno。由于 SACK 不像 New-Reno 一次全部

重传已发送包，而是有选择地重传，因此在一个窗口中出现数据包大量丢失时，SACK 的性能优于 New-Reno。

SACK TCP 不用等待重传定时器超时，增加了健壮性，不管丢失报文数量多少，SACK 发送端都有较好的恢复性能，并且不论是每个报文的端对端延迟还是总的延迟，SACK 解决了报文失序问题，提高了重发效率和信道利用率。

虽然 SACK 算法性能优良，但仍然存在一个较为严重的问题，因为 SACK 算法也会经历慢启动阶段，而慢启动本身存在缺陷。

互连网络中存在一些低速链路，而且一些中继路由器的缓存容量也十分有限，这些地方对突发通信量的处理能力较弱，在网络的信息传递过程往往会成为网络瓶颈，降低网络的吞吐量。当网关（如中继路由器）接收分组的速率大于其将分组转发到外向链路的速率时，分组将存入缓存队列等待发送。当网关的缓存耗尽时，接收的分组将被丢弃。SACK 算法可以防止超过网关处理能力的不适当的突发通信量，以免造成分组丢失（网络拥塞发生）。

当然 SACK 算法能较有效地解决突发通信量问题，从而防止网络拥塞的发生，而且实际上它的发送速度并不慢。但是，SACK 算法有时并不能有效地利用网络所提供的带宽，特别在那些传输通信量小于带宽延迟积（bandwidth-delay product，即网络的带宽与往返时延的乘积，这个乘积就是发送端到接收端的管道的容量）的链路中。

由于 SACK 也要经历慢启动阶段，而慢启动阶段的主要目的是用来决定可利用的网络带宽，阻止发送方发送大量的数据淹没中间网关从而引起数据丢失。从这一点来讲，它是很好的，但是，这种机制在下列情况效率很差：（1、）当接收方采用延迟的 ACK 接收机制时，发送方在发送一个分组后，不得不等待一个超时，才能发送第二个分组（2、）当发送方需要发送的数据不大时（2 到 4 个分组），必须经过两到三个 RTT 时间才能完成数据发送（3、）在高带宽，大延迟的网络（如卫星网络）中，发送方在发送一个分组后，需要等待较长的时间（因为 RTT 很大）才发送第二个分组。可以看出，如果慢启动的初始窗口稍大一些，以上情况可能就不会发生。

在标准的慢启动中，假设接收方每收到一个分组就发出一个 ACK，且没有分组丢失和 ACK 丢失的情况，这样，cwnd 从一个分组到达通告窗口（awin）的值所需要的时间为公式（4—1）所示：

$$\text{Slow start time} = \text{RTT} \log 2 \quad \text{awin} \qquad \text{公式（4—1）}$$

如果接收方采用延迟确认机制，发送方收到的 ACK 数目大约只有上面的一半，cwnd 从一个分组到达通告窗口（awin）的值所需要的时间就增加至大约两倍，如公式（4—2）所示：

Slow start time=2RTT log2　awin　　公式（4—2）

可以看出 TCP 所采用的延迟响应策略也影响了慢启动，即减慢 cwnd 的增加速度，从而影响了 TCP 性能。TCP 的延迟响应算法被证明是有损于 TCP 性能的。而目前许多 TCP 协议实现中都采用了延迟确认机制，这样可以减少主机和路由器的资源耗损并可以提高网络吞吐量，但是延迟确认机制在拥塞窗口的初始值为 1 的时候会产生性能问题。

当初始窗口为 1 时，发送方发送一个分组后就等待该分组对应的 ACK。但是由于接收方采用延迟确认机制，接收方在收到一个分组后，不是马上产生 ACK，而是要等到第二个分组到达或者延迟 ACK 定时器超时才发送 ACK，也就是说，当发送方拥塞窗口的初始值为 1 的时候，发送方在发送一个分组后不得不等待，直到延迟定时器超时。这显然严重降低了网络带宽的利用率。因此慢速启动算法经常是以大量的报文段丢失而终止的，这显然有损 TCP 的性能。

当然，在不同的网络环境下，甚至在同一网络的不同时间段，初始窗口值的大小对网络的影响是不同的，而且过大的初始窗口值也会使网络发生拥塞的机率大大增加，因此我们必须从各种不同的角度来讨论这个问题。

尽管 TCP 拥塞算法使 Internet 在全球范围内的迅速蔓延变得可能，但网络拓扑、协议、应用和实现的异构化，标准的 TCP 拥塞控制和算法并不适合各种情况。随着技术的发展，网络的带宽将越来越大，现在网络中的许多低速链路将被高速链路所代替，而且中继路由器的缓存容量也越来越大，网络中的许多瓶颈将逐逐渐消失。在这种新的网络环境下，网络中大多数链路和中继结点对突发通信量的处理能力得到了较大的提高，算法初始窗口值过小的问题也逐渐成为影响网络性能的一个因素。

4.3　改进型 SACK 算法的提出

为了消除现有 SACK 算法的缺陷，提高网络传输效率，在此提出了一种高效的 SACK 算法，我们称之为：ESACK（efficiency selective acknowledge）。具体方法是：使用扩大的初始 cwnd，增加初始的 cwnd 的大小对较短的数据流和低带宽网络最为有利。假如首先使用一个较大 cwnd 的初始值 Iw，在发生超时重传时才设置为 1，这就能提高慢启动时的网络带宽利用率。算法改进如下：

把慢启动阶段中的初始窗口值 Iw（Initial window）定为 2 个分组。

如果分组大小不超过 1460bytes，Iw 设为 3 个分组。

如果分组大小不超过 1095 bytes，Iw 设为 4 个分组。

那么，就这样 Iw 的值由一个分组扩大为 2～4 个分组大小，在多数情况下这样就会使得 Iw 的值大约在 4K 左右，Iw 的精确值就可由公式（4—3）确定：

Iw = min(4*Mss, max (2*Mss,4380bytes))　　公式 (4-3)

很明显，Iw 的上限是由一个 TCP 段的最大字节数 Mss（maximum segment size）所确定，即：

```
IF  (Mss<= 1095bytes)
    Then  Iw <= 4*Mss  ;
IF   (1095bytes <= Mss< 2190bytes)
    Then  Iw<= 4380bytes ;
IF  (2190bytes<= Mss)
    Then    Iw<=2*Mss
```

这里提出的扩大 Iw 值主要是针对慢启动阶段在数据开始传输时或检测出丢失数据后修复重传时而设置的。但是在重传超时器发生超时重传的情况下，cwnd 的值是直接设置为 1 的。

4.4　对改进型的 SACK 算法的理论分析

本文所提议的使用改进型 SACK 算法并不提倡在 Web 游览器中同时打开多个都扩大初始窗口的 TCP 连接。当游览器同时打开多个 TCP 连接时（不管 IW 的值多大），这本身就违背了 TCP 拥塞控制机制（即要保持公平性）。如果这些 TCP 连接都扩大初始窗口，将进一步加大在网络中对其他通信流量的不公平性。

对于扩大初始窗口，一个附加的限制是初始窗口的值最多为初始报文段大小的 4 倍，同时接收方通告窗口对发送方发送窗口上限的限制必须保留（TCP 流量控制所必须的）。例如，如果某个 TCP 连接的初始报文段大小为 1460 字节，那么一开始它最多只能发送 3 个报文段的数据;如果某个 TCP 连接的初始报文段大小为 1460 字节，那么一开始它最多只能发送 3 个报文段的数据；如果某个 TCP 连接的报文段大小是 512 字节，那么一开始它最多也只能发送 4 个报文段的数据。而且这个增加的初始窗口值是可选的，当然此时 TCP 发送方不一定要采用扩大的初始窗口。

现在我们来看一下，改进型的 SACK 算法的主要优点：首先，对那些仅有少量数据需要传输的连接，扩大初始窗口可以减少发送所有数据所需的时间(假设在中等包丢失率的网络中)。目前，许多 email 和 web 页面文件的大小小于 4K 字节，大初始窗口可将发送时间减少至一个 RTT 时间。

其次，可减少 cwnd 从初始值增加到 awin 所需要的时间。例如，当接收方对每个报文段都进行确认时，cwnd 从 IW 增加到 awin 所需时间为：

slow start time=R(log2awin — log2IW)　　公式（4—4）

当接收方采用延时确认机制，cwnd 从 IW 增加到 awin 所需时间为：

slow start time=2R(log2awin — log2IW)　　公式（4—5）

如上面公式（4—4）和（4—5）所示，如果 cwnd 从 IW 增加到 awin 所需的和 RTT 次数减少了，那么 TCP 连接传输数据的所需时间也相应地减少了。这个时间减少对通信流量相对于带宽延迟很小的网络中的传输会起到较大的作用。对于通信流量较大的传输而言，由于随着通信流量的增大，Slow Start 对整个性能的影响逐渐减小，所带来的益处也相对要小一些。

另外，当接收方采用延迟确认机制，使用大于 1 的初始值会减少 1 个延迟 ACK 超时的时间。例如，对那些报文段大小不大于 2K 字节的连接，这意味着发送方一开始至少可发送两个报文段，发送方就无须等待一个延迟 ACK 超时。

具体来说，扩大初始窗口在以下几个方面给 TCP 连接带来了较大的益处。

当初始窗口一个报文段时，如果接收方采用延迟 ACK 机制，这样接收方在生成一个 ACK 之前不得不等待一个超时。但是，如果初始窗口至少为两个报文段时，那么接收方在第二个报文段到达时就可生成一个 ACK。这就减少了因等待超时而浪费的时间(一般情况下可达到 200ms)。

对那些仅仅传送少量数据的连接而言，改进型 SACK 算法将减少传输时间(假设连接处于中等丢包率的网络中)。日常应用中，许多 email 和 Web 页面传送的信息量都小于 4K 字节，这样，扩大的初始窗口就可以在仅仅一个 RTT 时间内完成数据传送。

对那些采用大拥塞窗口的连接而言，在数据传送的缓慢启动阶段，改进型 SACK 算法可减少 3 个 RTT 和一个延迟 ACK 时间。在高带宽大广播延迟连接路中，如卫星链路，这个变化将给 TCP 连接带来较大的益处。

此外，改进型 SACK 算法并不缺乏公平性。这意味着同没有扩改进型 SACK 算法的 TCP 连接相比，并不会产生系统的不公平。但是，这并不意味着一个初始窗口为 1 个 1460 字节大小的报文段的 TCP 连接和一个初始窗口为 3 个 1460 字节大小的报文段的 TCP 连接在任何情况下都得到相同的吞吐量。

4.5　对改进型 SACK 算法的网络仿真研究

通过在 ns-2 网络仿真器上进行的一系列仿真实验，我们将观察改进型 SACK 算法对 TCP 连接和网络带来的影响。主要目的是研究在什么条件下使用改进型 SACK 算法可以改善网络性能，并研究扩改进型 SACK 算法的 TCP 连接对其它仅仅使用一个初始窗口的连接所产生的影响。

在仿真实验中，将初始窗口的值增加至大约 4K 字节（4380 字节，报文为 1460 节的情况），这和前面小节中所提的建议是一致的。仿真研究主要涉及到增加初始窗口值所带来的效果和对网络中其它 TCP 连接的影响。

4.5.1　NS 仿真程序的建立与运行

1、用户使用 NS 实现改进型的 SACK 算法仿真的过程如下：

（1、）建立仿真程序：用户用 Otcl 源程序来初始化一个事件调度器，定义网络拓扑结构，配置业务源，确定发送和接受传输数据包时间，然后启动 NS2 仿真器。

（2、）运行仿真程序

（3、）仿真结果分析：程序运行结束后，用户根据记录模拟数据的文档，利用 NS 中的可视化仿真软件 NAM 动态查看仿真的运行过程、观察跟踪数据，或使用 Linux 中的 Xgraph 或 Windows 下的 Origin 等软件将结果转换成带有坐标的平面图。

2、用 NS 进行仿真的具体步骤，如下：

产生事件调度器；（可选）打开跟踪机制；产生需要进行仿真的网络；设置路由选项；插入错误跟踪模块；产生传输层连接；业务生成；传输 应用层数据；开始仿真。

一、产生事件调度器

set ns [new simulator]

调度事件

$ns at <time><event>(<event>任何合法的 NS/Tcl 命令)

开始调度

$ns run

二、跟踪

跟踪所有链路上的分组

$ns trace-all [open test.out w]

以 nam-1 格式跟踪所有链路上的分组

$ns trace-all [open test.nam w]

跟踪特定链路上的分组

$ns trace-queue $n0 $n1

$ns nam-trace　$n0 $n1

三、产生仿真网络

产生节点

set n0 [$ns node]

set n1 [$ns node]

产生链路和队列

$ns duplex-link $n0 $n1 <带宽>　<延时>　<队列类型>

四、插入错误检测模块

生成错误模块

$ns loss-module [new ErrorModule]

$loss-module set rate_ 0.01

$loss-module unit pkt

$loss-module ranvar [new Randowm Variable/uniform]

$loss-module drop_target [new Agent/NULL]

插入错误模块

$ns lossmodule $loss-module $n0 $n1

五、设置路由

单播（unicast）

$ns rtproto<类型> ：类型是指静态路由、距离矢量路由等路由类型。

多播（multicast）

$ns multicast

$ns mrtproto<类型> ： 类型是指多播情况下的路由类型。

六、产生连接

TCP 连接：

Set tcp [new Agent/TCP]

Set tcpsink [new Agent/TCPsink]

$ns attach-agent $n0 $tcp

$ns attach-agent $n1 $tcpsink

$ns connect $tcp $tcpsink

UDP 连接：

Set udp [new Agent/UDP]

Set null [new Agent/Null]

$ns attach-agent $n0 $udp

$ns attach-agent $n1 $null

$ns connect $udp $null

七、产生业务

使用 UDP 的业务

set src [new Application/Traffic/CBR]

使用 TCP 的业务

FTP 业务

Set ftp [new Application/FTP]

$ftp attach-agent $tcp

TELNET 业务

Set telnet [new Application/Telnet]

$telnet attach-agent $tcp

很明显，我们利用 NS 仿真的处理流程结构就是：

set ns [new simulator];

[打开跟踪开关]；

产生仿真网络拓扑图；

产生路由方式；

产生传输 层代理；

产生应用层业务或其他的业务源；

后处理子程序；

开始仿真：

4.5.2 仿真工作

1、 模型和假设

要进行网络仿真实验，首先必须建立仿真的模型和条件。我们模拟的一个有一条瓶颈链路的网络拓扑结构如图 4－1 所示。

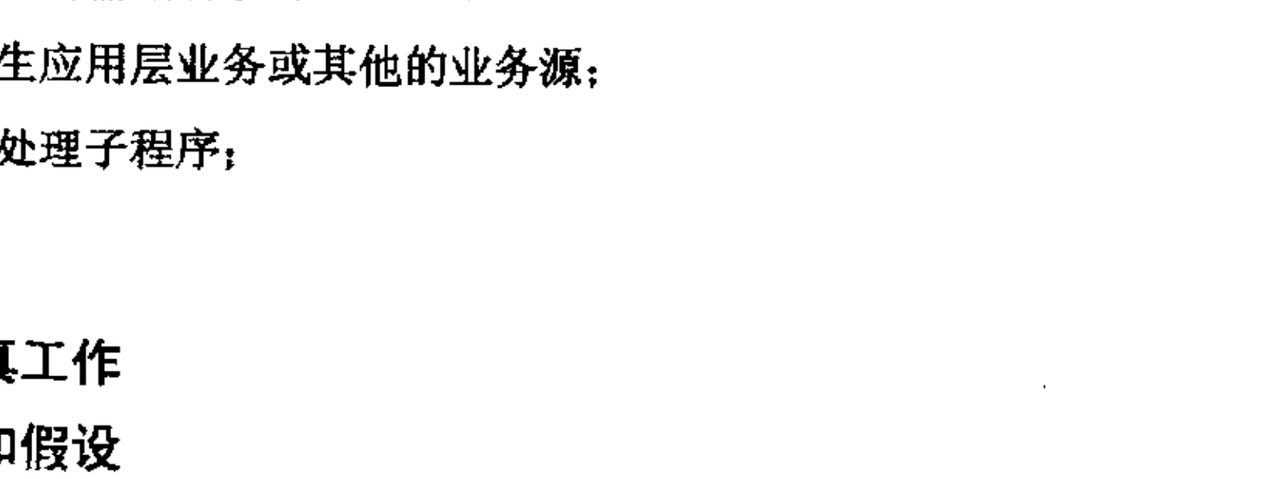


图 4－1　仿真网路的拓扑结构

在图4—1中，ftp和 web浏览器客户连接在左边的节点上(节点2～节点5)。这些客户由连接在右边的节点（节点6～节点9）的ftp和web服务器提供服务。这些节点之间的来回链路的带宽是10Mb。瓶颈链路位于节点1和节点0之间，带宽仅为1.5Mb，延迟为50ms。所有链路都是双向的，但是仅仅是ACK、SYN、FIN和URL等信息从左边流向右边。一些仿真同时还有数据通信流从右边流向左边，但它对结果没有任何影响。

在仿真实验中，我们假设所有的文件传送均为1M字节，所有WEB页面都有3个内嵌的URL。Web客户的浏览速度很快，要求在一个随机的延迟（一律分布在1至5秒内）后就出现一个新的页面。这并不是真正的去模仿单个用户的web浏览模式，而是产生一个适当的较高的通信负载。

TCP窗口的最大值被设置为11个报文，最大报文段长度为1460字节，缓冲区长度为25个报文（在ns—2中，TCP需要设置窗口大小和缓冲区大小的单位为报文)。在仿真实验中，我们改变新TCP连接的 初始窗口的报文段数目，从1中4，并保持所有报文段的大小均为1460字节，一个丢失报文段将导致一个报文段的restart　window 值被使用，这和目前实用中的TCP 协议是一样的。

在仿真实验运行中，有8或16 web客户，同时还有一定数量的ftp客户（0至3个)。初始窗口值为1至4个报文段，尽管4个报文段的情况已经超出了前面的建议值，但是还是可以用它来做一个比较。我们使用3个指标来观察实验结果：goodput；web客户的中等页面延迟；ftp客户的中等文件传输。此外，仿真的运行时间是相当长的，有360秒，这足以确保仿真实验是一个合适的例子。

2、 **仿真结果**

在仿真实验中，我们改变文件传输客户的数目，目的是为了改变链路的拥塞状况。因为在实验中ftp客户不断的请求1M字节的文件传送，因此即使在只有一个ftp客户的情况下，链路的利用率也到达了90%以上。当有 3个文件传输客户同时运行时，产生的拥塞状况却和想象的不大一样，记录的值相当稳定。尽管所有连接都使用相同的初始窗口。在进行1M字节大小的文件传输中增加IW的值，却看不出产生的影响有多大。因此我们将注意力集中在web浏览连接上。

在下面表4—1，我们用“web”表示web客户的数量，“ftp”表示连接的文件传输客户的数量，表1中显示了在web传输中随着IW值的增加，中等页面延迟的变化。从表4—1中可以看出，随着Iw的增加，web连接的传输延迟有了明显的改善，在大多数情况下可以减少30%的延迟时间。

表4－1　时间延迟情况

Web	Ftp	IW 值的大小			
		1	2	3	4
		%（时延减少百分比）　秒			
8	0	0.52	14.6	18.4	16.6
	1	1.12	19.2	26.2	32.8
	2	1.20	17.0	18.6	29.2
	3	1.32	12.5	19.0	28.2
16	0	0.68	10.0	16.2	18.2
	1	1.12	16.4	24.2	35.6
	2	1.32	17.2	20.9	25.4
	3	1.36	12.2	22.1	23.2

下面表 4—2 和表 4—3 分别显示了在同一实验下的瓶颈的利用率和包丢失率。包丢失率是随着 IW 的增加而增加的，在所有情况下包丢失率的增加不会超过 1%。在一些过载的情形中，也观察到包丢失率的减少，特别是当 ftp 传输消耗了大部分链路带宽而且大量的 web 传输只能共享剩余的链路带宽时。在这种情况下，web 传输产生了严重的包丢失，有些 IW=4 的客户在同一个窗口就有多个的包丢失，这就导致了较长的恢复时间（相对于在一个窗口中只有一个包丢失的情形）。在恢复时间内，连接处于停止状态，这就减轻了拥塞状况，从而导致包丢失率下降。特别要提出的是，上述这种情形只有在链路极度过载的情况下才会出现。

表 4－2　链路利用率

Web	Ftp	IW 值的大小			
		1	2	3	4
		%（链路利用率）			
8	0	35	36	38	39
	1	96	93	92	94
	2	98	96	95	92
	3	97	97	97	97
16	0	68	69	70	72
	1	96	94	97	94
	2	98	98	97	98
	3	99	96	98	98

从表4—2中可以看出，随着IW值从1增加到4，一般情况下链路利用率有轻微的下降，但是变化很小，基本上都是1%左右。因此，改进型SACK算法基本上不会影响瓶颈链路的利用率。为了获得一个更完整的性能分析，我们通过计算网络能力：goodput数目（中等延迟）（单位为Mb/ms），并跟IW一起对所有情形进行分析（每个情形由它的web客户数和ftp客户数来唯一确定）。可以看出增加IW的值普遍可以带来溢出。

为了获得一个清楚表明在所有测试情形下产生了什么效果，对那些正常的情形我们通过定义在IW为1的情形的网络能力作为基准来规格化网络能力中。通过分析，随着IW从1增加至4，网络能力也增加了至少15%，即使在一个有大量传输通信量的极度拥塞情形下也是如此。在仿真实验中，当web通信量占据了主要的可利用的带宽时，网络能力可增加达60%。在使用改进型的SACK算法时，网络能力的增加主要是由于吞吐量的增加和延迟的减少。

表4－3　分组丢失率

Web	Ftp	IW值的大小			
		1	2	3	4
		%（分组丢失率）			
8	0	0.0	0.0	0.0	0.0
	1	0.7	1.2	1.41	1.3
	2	1.8	2.2	2.4	2.6
	3	2.6	3.1	3.6	3.5
16	0	0.0	0.2	0.4	0.8
	1	2.0	2.4	2.8	2.7
	2	3.5	3.6	4.0	4.4
	3	4.2	4.5	5.0	5.2

相对于文件传输观察到的性能，web客户获得的性能必须被平衡。我们计算了ftp的网路能力，见表4—4。数据表明，web连接中的网络能力的改善对同时进行的文件传输的影响可以忽略不计的。从表4—4中可以看出，随着IW的增加，文件传输的网络能力有一个很小的变化，但看不出什么特别的趋势。因此，可以得出这样一个结论：文件传输的网络能力基本保持一致。但是，值得注意的是，较大的IW允许WEB传输比在较小IW时，可以获得更多的网路带宽，这对FTP的应用来讲：可能就意味着较少字节的传输，或者导致网络能力的轻微下降。

表 4－4　网络处理能力与 IW 值的关系

Web	Ftp	IW 值的大小			
		1	2	3	4
		单位：Mb/ms			
8	1	4.8	4.4	4.2	4.2
	2	3.0	2.9	3.1	2.8
	3	2.4	2.2	2.3	2.3
16	1	2.4	2.4	2.4	2.6
	2	2.0	2.2	1.9	2.0
	3	1.5	1.6	1.7	1.8

3、 分析

上面几个仿真试验研究了改进型的 SACK 算法对于竞争的网络通信量的影响。在这个研究中，HTTP 和 FTP 数据流共享一个单一的拥塞网关（当然，在不同的试验中 HTTP 和 FTP 数据流的数量是不一样的）。对每个仿真试验而言，观测瓶颈链路的利用率和包丢失率，中等大小的 web 页面的延迟。从上面的仿真结果可以看出，在大多数情况下，改进型的 SACK 算法可以增加网络的吞吐量，虽然，有时也会导致包丢失率轻微的增加。但是，从整体试验效果来看，可以认为：增加 TCP 的初始窗口到 3 个分组（或者 4380 字节）可以改善网络的性能。从安全的角度来看，这个建议增加的初始窗口值只是对 TCP 协议的一个小变化，不会产生任何安全问题。

下面我们再来看一下，改进型的 SACK 算法，在不同连接数目下的性能表现，具体网络参数设置如下：网络拓扑结构与图 4－1 类似，TCP 时钟以 0.01ms 为单位计时；源端传输文件大小为 30KB(标准的 Web 页面大小)，每个源端每隔 120ms 产生一个 FTP 流，并连续发送 20 次。

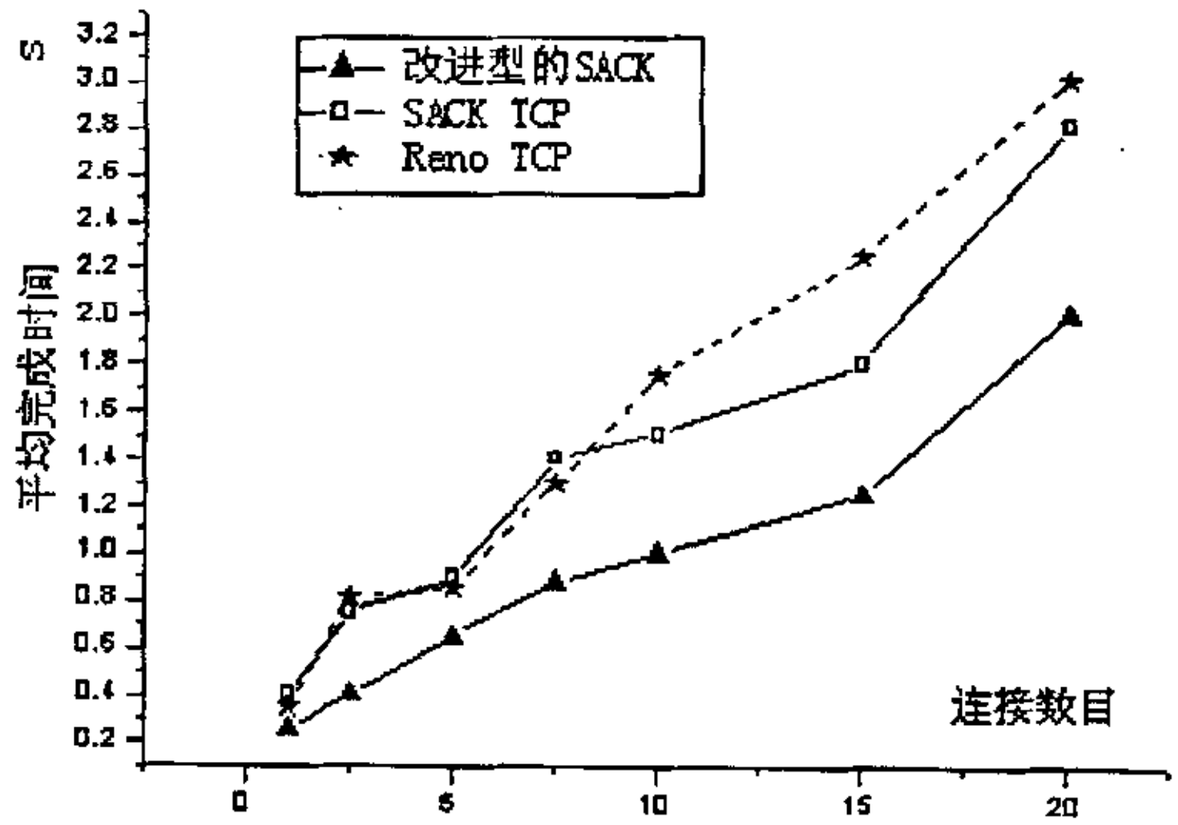


图 4—2　改变连接数目性能比较（file size=30KB，delay=50ms）

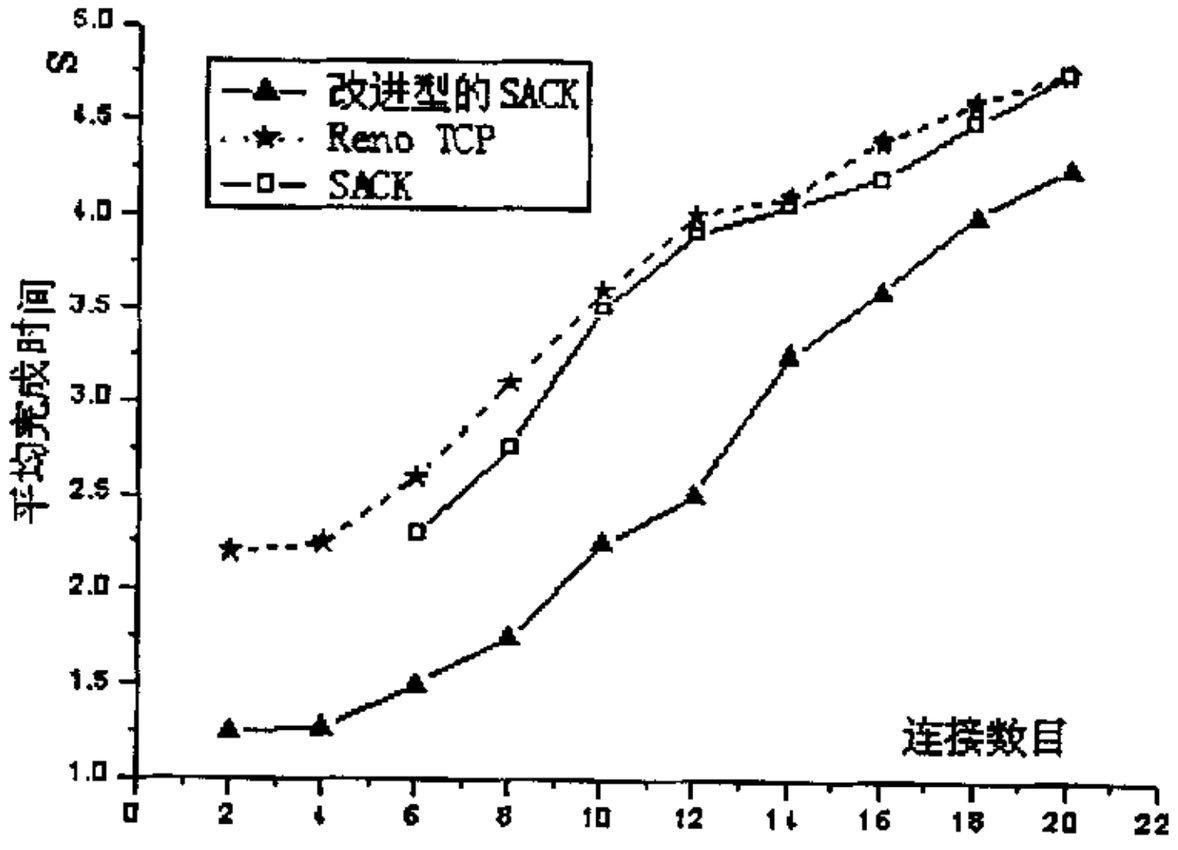


图 4—3　改变连接数目性能比较（file size=30KB，delay=200ms）

从图 4—2 和 4—3 中可以看出，改进型的 SACK 算法在两种实验环境下都明显的减少了连接的平均完成时间，当瓶颈链路延迟为 50ms 时，改进型的 SACK 算法在轻度拥塞的情况下(连接数＝<5)，比 Reno TCP 的平均传输时间减少了 30—40%左右，在中度拥塞的网络中（5<连接数＝<15），传输时间减少了 20—30%左右，而在拥塞程度较高时，减少了 15—25%；很明显，改进型的 SACK 算法算法在链路延迟增加时，能够表现出更好的性能（因为此时，网络中存在的带宽延时乘积值较大。）例如，当瓶颈链路延迟为 200ms 时，改进型的 SACK 算法的传输时间比 Reno TCP 减少了 25—40%。由此可知，改进型的 SACK 算法能够大大减少短连接的传输时间。

4、 结论

目前，控制算法中有的过于复杂不便实现，有的不够有效，大多数则公平性较差。虽然标准的 SACK 算法性能优良，但是，存在一些局限性，因此，对 SACK 的进一步研究具有及其重要的理论和应用价值。本文重点分析了 SACK 拥塞控制算法，并就针对它在慢启动阶段所存在的缺陷提出了一种改进型的 SACK 算法的思想，即扩大的初始 cwnd 改进方法，本文的研究，可望进一步提高网络传输的效率，并改善 TCP 网络传输的性能和质量。

我们着重从 SACK 拥塞控制算法入手分析它的主要缺陷，对 TCP 性能的影响，提出了改进型的 SACK 算法：即用 2 到 4 个分组（大约 4K 字节）的窗口代替原来一个分组的窗口，并通过理论分析和网络仿真试验结果说明：改进型的 SACK 算法在大多数的情况下确实能够改善网络性能。虽然这个解决方法还不是特别完善，但在很大程度上，它确实能够解决目前所面临的问题。在文中描述改进型的拥塞控制算法，还没有在广泛的网络环境中经过中大量的试验论证，距正式使用，还有待进一步的研究和改进。在将来，移动主机和无线链路会继续增加，因而对 SACK 算法在无线网络应用中的研究是一个非常有意义的工作，我们下一步要做的工作就是这一方面研究，给出具体的解决方案。

第五章　改进型的SACK算法的公平性

公平性是拥塞控制中的非常重要的一个问题。衡量网络性能的重要参数，这也是评价一个算法性能的关键指标。

5.1　拥塞控制的公平性原则

拥塞控制计划就是用来维持网络中的供需平衡，主要有两个基本目标：（1、）保证网络的瓶颈处能工作在“最佳点”；（2、）保证不同的用户间共享网络资源的公平性。通常我们把这两个目标分别称为“最佳”和“公平”。

“最佳”关心的焦点是网络瓶颈处的总的通信负载[28]。为维持网络瓶颈处工作在“最佳点”，终端用户必须能根据不断变化的网络环境来调整他们的通信流量要求。网络瓶颈处能够收集关于自身资源利用的信息，如队列长度和链路利用，并向源端反馈明确的控制信息，这样，终端用户通过从确认中获得超时和延迟等隐含信息，就可以确定当前的网络状况。

而“公平”就是要使网络中的不同连接能平等地共享网络资源，即使在网络瓶颈处也是如此[32]。在 Internet 中，尽量保证公平性原则是非常重要的，否则就会出现因为某些用户的信息传送占用大部分网络资源而严重影响其他用户的信息传送的现象。拥塞控制的公平性要求如果有 n 个用户共享一个网络资源，那么这 n 个用户必须以相等的份额占有资源。

当前 TCP/IP 协议只提供单一的“尽力而为”（best effort）服务和 FIFO 缓冲队列管理算法，没有服务质量的保证，为多种应用提供不同的服务质量已经成为 Internet 发展的必然趋势，由于不同的用户对业务具有不同的要求，网络也需要区别 Qos 等级。目前 Internet 的资源分配主要是在端系统进行，资源分配的公平性将是当前网络通信中的公平性问题。本文提出了改进型的 SACK 算法，经过仿真实验证明，该算法具有较好的公平性。

5.2　拥塞控制的公平性问题

TCP 拥塞控制的公平性是指在网络发生拥塞时各源端（或同一源端建立的不同 TCP 连接或 UDP 数据报）能公平地共享同一网络资源（如带宽、缓存等）。处于相同级别的源端应该得到相同数量的网络资源。

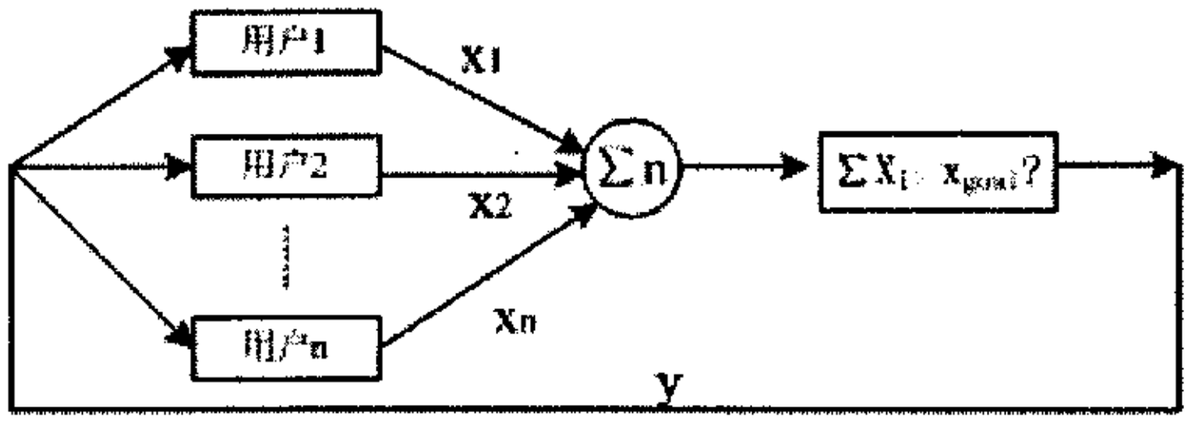


图 5－1　n 个用户共享一个网络下拥塞的闭环控制系统

如上面图 5－1 中所示，如果每个源端优先级相同，则 $x_i(t)=x_j(t)$（对于任意 i，j）。如果所有源端没有得到相同的分配，网络就缺乏公平性。Jain 在文献中将网络公平性可定义为公式（5－1）：

$$F(X)=\frac{(\sum xi)^2}{n(\sum xj^2)} \qquad \text{公式（5－1）}$$

其中 x_i 表示分配给用户 i 的网络资源，如响应时间、吞吐量、带宽、缓存等、0＝<F(x)=<1，显然完全公平时，F（x）=1，此时所有的连接平均分配网络资源；当所有的网络资源由一个连接占用时，此时 F（x）＝1/n 达到最小，当 n 趋于无穷大时，F（x）＝0；如果所有资源被 n 个用户的 k 个平均共享资源，其他 n-k 个源端没有得到任何资源，则 F（x）＝k/n。

TCP 层上的公平性问题表现在两方面：

1、　TCP 连接与 UDP 连接间

面向连接的 TCP 和无连接的 UDP 在拥塞发生时对拥塞指示的不同反应和处理，从而导致对网络资源的不公平使用问题。由于 TCP 提供端对端的可靠传输服务，在检测到拥塞发生时，有拥塞控制反应机制的 TCP 数据流会按拥塞控制步骤进入拥塞避免阶段，从而主动减小发送进入网络的数据量。但对无连接的数据报 UDP，由于没有端到端的拥塞控制机制，即使网络发出了拥塞指示（Congestion　indication，例如数据包丢失，收到重复 ACK 等），UDP 也不会像 TCP 那样减少向网络发送的数据量。结果遵守拥塞控制的 TCP 数据流得到的网络资源越来越少，没有拥塞控制的 UDP 则会得到越来越多的网络资源，这就导致了网络资源在各源端分配的严重不公平。

网络资源分配的不公平反过来会进一步加重拥塞情况，甚至可能导致拥塞崩溃（Congestion　collapse），因此如何判断在拥塞发生时各个数据流是否严格遵守了 TCP（TCP　friendly），以及如何“惩罚”不遵守拥塞控制协议（Not　TCP

friendly）的行为，成了目前研究拥塞控制的一个热点领域。在传输层解决拥塞控制的公平性问题的根本方法是全面使用端到端的拥塞控制机制。目前，判断拥塞时不遵守拥塞控制的数据流的几种方法如下：

● 如果数据流遵守 TCP 拥塞控制方式，那么在拥塞发生时，作为响应，它首先应将拥塞窗口 cwnd 减半，然后在每个 RTT 内按常数速率增加 cwnd。给定包丢失率 p，TCP 连接的最大传送速率为 T byte/s，B 为一个数据包的最大字节数，R 为最小 RTT。当某数据流的发送速率大于 T 时，则可断定该数据流没有执行拥塞控制（Not TCP friendly）。公式（5—2）主要应用于没有突发级数据包丢失的情况。

$$T \le \frac{1.5\sqrt{2/3} \times B}{R \times \sqrt{P}} \qquad \text{公式（5-2）}$$

实际使用中以大于 $1.45B/(R\sqrt{P})$判断数据流有没有执行拥塞控制，以小于 $1.22B/(R\sqrt{P})$为解除对该数据流“惩罚”的条件。

● 通过判断网络中占据高带宽的数据流是否对拥塞指示进行响应来决定其是否执行拥塞控制，也就是随着网络包丢失率的增加，其传送速率应相应降低[52]。

表明如果包丢失率 p 增加 x，则源端的发送速率应大致减少$\sqrt{x}$，举例来说，如果包丢失率增加 4 倍，那么发送速率应减少 2 倍。正是根据这一关系，通过检测数据流对包丢失率的反应，就可以大致判断该流是否执行了拥塞控制。对有 ON/OFF 特性的数据源和接收者经常变动（Subscribing or unsubscribing）的多点广播（Multicase）方式，由于传送速率本身经常变化，所以这种判断方法在以上两种情况下并不理想。

2、 TCP 连接与 TCP 连接间

一些 TCP 连接之间也存在公平性问题（Competing TCP）。产生问题的原因在于一些 TCP 的 RTT 较小，或者数据包比其他 TCP 大，这样它们也会多占带宽。另外一些上层应用为了使自己性能最佳，往往在信源端自行修改 TCP 拥塞控制策略，使 TCP 端到端的拥塞控制机制名存实亡。很明显，由于往返时间长的连接其窗口增长速度慢，因而在同样的条件下获得的带宽也就越少，导致 TCP 对传输延时长的连接的不公平性； TCP 使用的平均窗口 w 和平均包丢失率 p 关系[46]：

$$P = \frac{0.76}{W^2} \qquad \text{公式（5—3）}$$

当有 N 个 TCP 共享一个网络瓶颈（Bottleneck ，例如路由器）时，假定该瓶颈允许最大排队数为 S，那么所有 TCP 窗口大小之和应为 S，所以 w=S/N。带入公式（5—3）：

$$P = 0.76\frac{N^2}{S^2} \qquad \text{公式（5—4）}$$

公式（5—4）给出了各 TCP 在网络瓶颈处的数量关系。总之，解决 TCP 拥塞控制公平性问题的根本出路是在 Internet 上全面实行端到端拥塞控制和融合 IP 层拥塞控制的新算法。

5.3　TCP 友好性问题

Floy 在文中给出了 TCP 友好流（TCP　friendly　flows）的定义：在相同的网络环境下，如果一个流的平均发送速率不超过一个标准的 TCP 流的平均发送速率，则称这个流是 TCP 友好流，反之称它为非 TCP 友好流。

TCP 拥塞控制仅能约束遵守拥塞控制机制的数据源，行为不良的数据流会故意绕开端到端的拥塞控制机制，向网络持续发送大量的数据包，抢占带宽，从而引起其他连接的包被丢弃，甚至导致 TCP 流饿死。因此，TCP（TCP friendly）问题引起了商业界的广泛关注。

解决非 TCP 友好流与 TCP 流共存的方法之一，就是在 UDP 协议上增加拥塞控制机制，使得 TCP 和 UDP 流能够友好的共享网络资源。根据上述对 TCP 友好流的定义，其发送速率的上限应该是 TCP 数据流的发送速率。假设包丢失率为 P，一个 TCP 连接的最大发送速率 T 可以表示为公式（5—5）[33]：

$$T \le \frac{1.5\sqrt{2/3} \times B}{tRTT \times \sqrt{P}} \qquad \text{公式（5—5）}$$

其中 B 为最大的分组长度，tRTT 为往返时间。

但是实际上 TCP 流的发送速率远远小于上面公式给出的 TCP 流发送速率上限，根据上式实现的基于方程的 TCP 友好流在与 TCP 流竞争时，往往会占用更

多的带宽，表现为极不友好。文[26]精确的描述了网络稳定状态下传输数据的 TCP 拥塞控制模型，可以用来计算 TCP 数据流的最大发送速率：

$$R=\frac{B}{tRTT\sqrt{2P/3}+tRTO\min\left[1,3\sqrt{3P/8}\right]P(1+32P^2)} \qquad \text{公式（5—6）}$$

该方程不仅考虑了快速重传对吞吐量的影响，还考虑了由于超时导致的吞吐量下降的问题，在传输大量数据时，TCP 的吞吐量表示为包丢失率和平均往返时间的函数，在公式（5-6）的基础上，文[21]通过确定包的往返时间和丢失率来计算 TCP 数据流的最大发送速率，提出了一种 TCP 友好拥塞控制算法(控制速率)，使得多媒体数据流在传输时与 TCP 流分享网络带宽。另外，文中针对 Web 数据流传输时间短的特点，给出了短连接的传输时间模型，为更好的实现网络中数据流的友好传输提供了依据 。

5.4　改进型的 SACK 算法的效果

由于 TCP 是一个端到端的控制协议，RTT 对于连接的发送窗口和吞吐量起着决定性的作用，所以这种不公平性是端系统的拥塞控制机制的一种内在属性。要保证公平性以及进行有效的控制，最好能在发生拥塞的网络中间节点处采取相应的措施。

改进型的 SACK 算法从公平性出发，源端在收到拥塞指示信号后，不仅调整窗口的增长速度，纠正了有长时间 RTT 的 TCP 连接的偏差，改进了共享瓶颈处的公平性。

为了研究在不同的队列管理算法下多个 TCP 连接竞争资源的公平性，建立与图 4—1 类似的仿真结构，链路延迟均为 1Mb/s，带宽为 5Mb/s，假设源端有四个节点，与目的端的四个节点之间建立 TCP 连接，四个连接的传输延迟分别设为 10ms、50ms、100ms 和 200ms，瓶颈链路延迟为 20ms。

表 5－1　在同一个瓶颈链路处的 TCP 连接的平均吞吐量

连接位置	第一个	第二个	第三个	第四个
RTTmin（ms）	10	50	100	200
SACK 吞吐量（Kb/s）	72.8	35.2	22.5	9.28
改进型 SACK 吞吐量（Kb/s）	51.2	32.8	34.8	24.7

为了研究改进型的 SACK 算法和各种传统 TCP 协议，对终端平均吞吐量和

网络节点平均延迟等性能的影响具体情况，同样也对它们进行了仿真试验对照组测试。得出下面一组数据。

表5－2　平均吞吐量和网络节点平均处理延迟仿真试验数据

	Tahoe	Reno	New Reno	SACK	改进型SACK
平均吞吐量（Kbyte/s）	1.7557	1.8055	1.85465	1.8678	2.3015
性能提高（以Tahoe为基准）		2.84%	5.64%	6.38%	31.09%
平均延迟（ms）	85.6504	86.21212	90.2034	86.2754	88.5144
性能提高（以Tanhoe为准）		－0.66%	－5.32%	－0.73%	－3.34%

此处通过仿真试验证明了：改进型的SACK算法确实能够提高终端用户平均吞吐量。使用在扩大的初始窗口在一般情况下，确实可以提高网络的吞吐量，并且不会损害TCP的性能。

上面的表格给出了SACK 算法与改进型的SACK算法中的各个连接的吞吐量，由上面可以得出结论，各连接获得的吞吐量是不相同的。其公平性指数计算结果如下：

$F_{(sack)}(X)=0.57$

$F_{(改进型的SACK)}(X)=0.72$

四个连接的平均吞吐量的不同，通过试验得到了证明，改进型的 SACK 算法和原来的SACK算法相比，可以得到更高的平均吞吐量，而且稳定性还可以。

下面我们再来分析一下UDP流与TCP流共存时，改进型的SACK算法的公平性情况。拓扑结构类似于图4－1，仿真结构配置如下：由一个UDP连接与10个TCP连接共享同一瓶颈链路，所有链路带宽为5Mb/s,链路延迟为10ms，UDP连接以固定的速率100Kb/s发送CBR(Constant Bit Rate)流，所有的分组长度设为1000bytes。

下面就是SACK算法与改进型的SACK算法，在上述网络环境下的TCP流与UDP流吞吐量的仿真结果，从图5－2和5－3中可以看出，标准的SACK算法不能识别非TCP流，导致UDP流占用了绝大部分带宽（约85kb/s），而TCP流动平均吞吐量仅为35Kb/s；当然改进型的SACK算法，也不能识别非TCP流，但是它却能较快的占用网络带宽，使得 UDP 流所占用的带宽一定幅度的降低（42Kb/s），因此TCP流就得到65Kb/s的平均吞吐量。

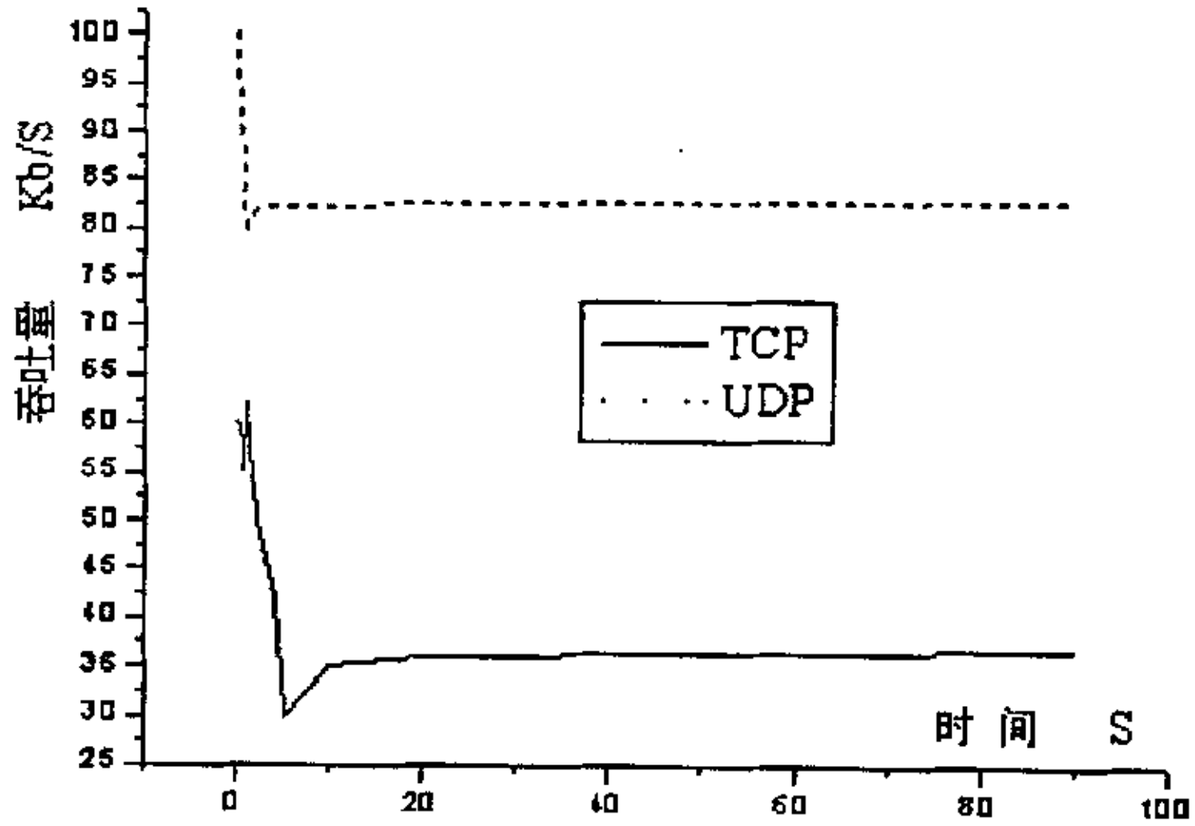


图 5—2　SACK 算法的 TCP 与 UDP 流吞吐量比较

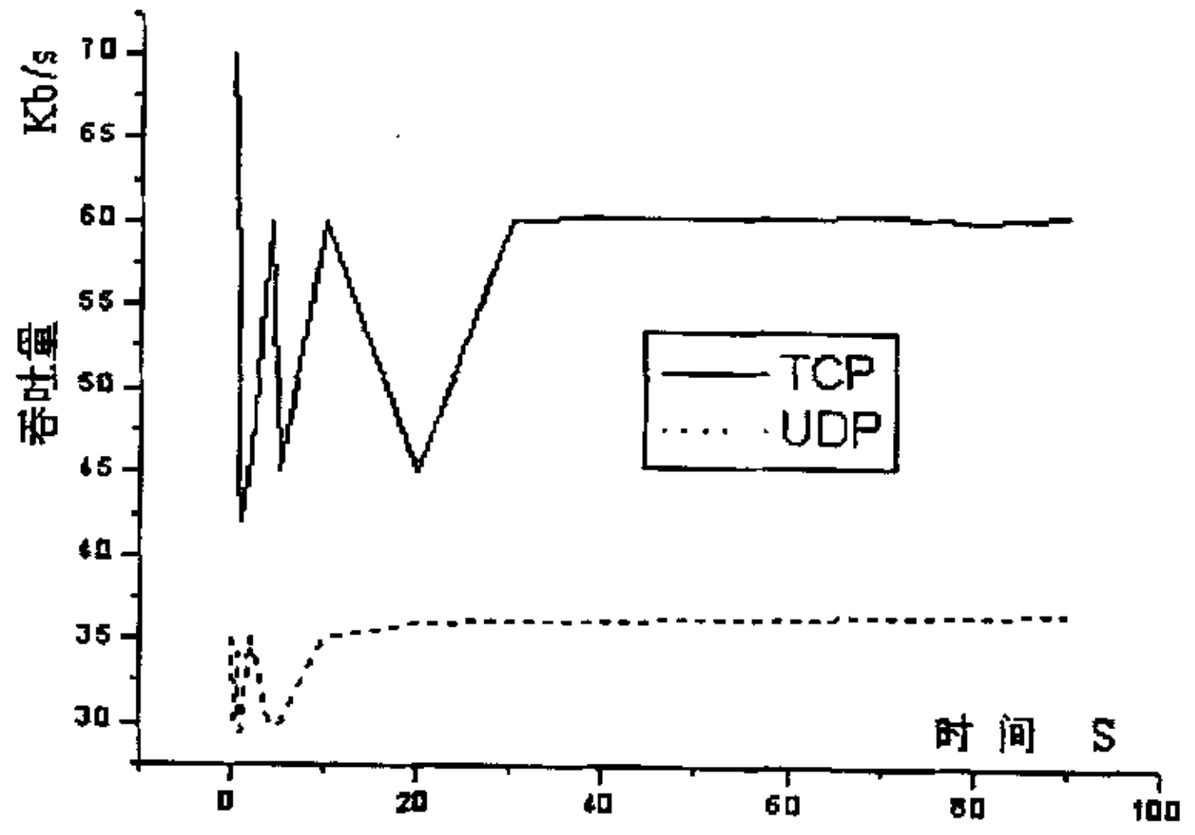


图 5－3　改进型的 SACK 算法的 TCP 与 UDP 流吞吐量比较

5.5　特定网络上的 TCP 拥塞控制

1、　TCP 在 ATM 上的拥塞控制

按计算机网络的分层模型，TCP 属于传输层协议，它的实施与下层网络有着密切的关系。其中由于 ATM 网络和无线网络的特殊性，给传统 TCP 拥塞控制机制的实现及有效性带来了新的问题。

ATM 论坛规定：ATM 网络本身采用几种拥塞控制方案[21]。这些方案包括：（1、）端到端基于速率的拥塞控制方案。这实际上是个带反馈的闭环控制，已被 ATM 论坛采纳。这种方案又分为显式正向拥塞指示（EFCI）、按比例速率控制算法（PRCA）等。（2、）基于信用（credit）的方法，如流量控制的虚通道（FCVC）

策略。(3、）早期分组丢弃算法（如 EPD)。

TCP 在 ATM 上的拥塞控制研究就是针对以上控制方法，在 ATM 不同业务类型（主要是指传输数据业务的可用比特速率 ABR，未定义比特速率 UBR 以及可变比特速率 VBR）上进行性能研究。由于 TCP 和 ATM 这两种技术本身的复杂性，目前的研究还不够深入。文献结合 Internet 上视频流研究了 TCP 在 UBR 上的 cell 丢失和 TCP 包丢失和吞吐量的关系。文献对 TCP 在 ABR 和 VBR 上上的性能进行了仿真试验研究。但由于这类仿真往往采用简化的业务模型（如 ON/OFF 模型)。使其研究离实际情况还有一段距离。目前控制算法中有的过于复杂不便实现，有的不够有效，大多数的公平性都较差。TCP 拥塞控制的参数（如窗口 Cwnd）与 ATM 的拥塞控制 算法(如 EPD)中的参数有什么关系，如何映射（mapping）？如何建立一套有效的分析理论来指导目前单纯根据经验来改进算法的不足？这些都是研究的热点和难点。

2、　TCP 在卫星信道上的拥塞控制

卫星通信具有覆盖面广、带宽分配灵活、资源利用率高、用户接入方便、故障恢复迅速的优点，因此能够满足商业、政府和个人的各种数据通信（data communication）需求，各种卫星系统（低轨 LEO、中轨 MEO、高轨 GWO）必将在下一代 Internet （NGI）中发挥关键作用。同时由卫星建立的 Internet 系统还存在未解决的技术难点，例如路由（routing）技术。其中 TCP 在卫星信道上的拥塞控制是个突出的问题，目前仍仍没有理想的解决办法。

1、卫星通信信道的链路传输时延对 TCP 拥塞控制重传超时 RTO 的影响。由于卫星信道传输距离远，所以传输时延比常用信道大得多[38]。例如局域网 LAN 单向时延为 5ms，而低轨卫星 LEO 为 100ms，高轨卫星 GEO（即同步卫星 geosynchronous orbit）为 250ms。TCP 拥塞控制的重传超时 RTO 通常为 2RTT 或 5RTT，根据计算，在 GEO 上的 Internet 完成 TCP 连接启动平均值为 29RTT，这显然与传统 TCP 的 RTO 不相符合。

2、卫星信道链路误码率对 TCP 拥塞指示的影响。如前所述，TCP 协议默认数据包丢失为网络发生拥塞的标志，卫星信道与其他通信手段如：光纤、电缆相比，更容易受气候、天象等外部环境影响，信道误码率相对较高[26]。这就是导致传输中出错的数据包的可能性大大增加，这种由传输引起丢失数据包现象与网络发生拥塞丢弃数据包并不相同。所以在卫星信道上采用传统 TCP 判断拥塞方式，从而减小 Cwnd，进入拥塞避免的方法并不合适。只会降低网络吞吐量，这个问题最明显的解决方法是设法让 TCP 知道它所处信道更多的信息，从而避免“误操作”。但 IP 并没有保留信道更多的信息（IP 主要掌握路由信息)，并且这还涉及要修改 TCP 协议，引起协议兼容性 问题，所以这种方法实施难度很大。

这也是目前 TCP　over　satellite 研究进展不大的原因之一。

拥塞控制本身是一个极其复杂的问题，任何单一的拥塞控制机制都不能完整顿解决这个问题，必须采用多种策略，从网络的各个部位、多角度全方位对拥塞加以控制，才能保证网络高效、稳定运行。

第六章　改进型的 SACK 算法在 Internet 上的应用前景

6.1　Internet 上的拥塞控制历史

Internet 最初源于美国国防部的 ARPANET 计划。在上世纪 60 年代中期，正是冷战的高峰，美国国防部希望有一个命令和控制网络能够在核战争的条件下幸免于难，而传统的电路交换的电话网络则显得太脆弱。国防部指定其下属的高级研究计划局（ARPA）来解决这个问题，此后诞生的一个新型网络便称为 ARPANET，其最大特点是采用无连接端到端的数据包交换服务。随后 ARPANET 开始与美国国家科学基金会（NSF）建成的 NSFNET 及加拿大、欧洲和太平洋地区的网络互联[31]。到了 80 年代中期，人们开始把这个网络称为 Internet。

早在 70 年代中期，ARPA 为了实现异种网络之间的互联与互通，推出了 TCP/IP 体系结构和协议规范[22][41]。拥塞控制的研究开始于 80 年代中期，1984 年，Nagle 首次指出了复杂的 TCP/IP 网络中可能存在的拥塞问题，特别是在由路由器连接的带宽差异较大的网络中，容易发生“拥塞崩溃”现象，但在科研界没有引起足够的重视。

直到 1986 年 10 月，Internet 首次出现了一系列的拥塞崩溃现象，网络吞吐量急剧下降，其间从 LBL 到 UC Berkerley 的数据流量从 32 Kbps 降到了 40bps，许多分散在各地的网点被迫长时间停止服务[16]。此后 Jacobson 等人开始对此进行研究，发现这是由于在网络拥塞状态下 TCP 的行为失常所致，为此 Jacobson 在 TCP 中增加了现在正在 TCP 协议上使用的拥塞控制算法（即 TCP Tahoe）。并于 1988 年 Van Jacobson 指出了 TCP 在控制网络拥塞方面的不足，并提出了著名的“慢启动”（Slow Start）、“拥塞避免”（Congestion Avoidance）方法、“快速重传”（Fast Retransmit）[12]；1990 年出现的 TCP Reno 版本增加了“快速重传 ”（Fast Retransmit）、“快速恢复”（Fast Recovery）方法，避免了网络拥塞不够严重时采用“慢启动”而造成大幅度减小发送窗口尺寸的现象[18]。这样 TCP 的拥塞控制就由这 4 个核心部分组成。

经过十多年的发展，目前 TCP 协议主要包含有四个版本：TCP Tahoe、TCP Reno、TCP New Reno 和 TCP SACK。TCP Tahoe 是早期的 TCP 版本，它包括了 3 个最基本的拥塞控制阶段：“慢启动”、“拥塞避免”和“快速重传”。TCP Reno 在 TCP Tahoe 基础上增加了“快速恢复”方式。TCP New Reno 对 TCP Reno 中的“快速恢复”方式进行了修正，它考虑了一个发送窗口内多个数据包丢失的情况。在 Reno 版中，发送端收到一个新的 ACK 后，就退出“快速恢复”阶段；

而在 New Reno 版中，只有当所有的数据包都被确认后才退出“快速恢复”阶段[37]。TCP SACK 关注的也是一个窗口内多个数据包丢失的情况，它避免了之前版本的 TCP 重传一个窗口内所有数据包的情况，包括那些已经被接收端正确接收的数据包，而只是重传那些被丢弃的数据包。

应该说，到目前为止 Internet 的发展与运行是非常成功的，这主要归功于支持 Internet 运行的 TCP/IP 协议在设计上的灵活性与合理性。但是，随着技术的发展与应用需求得更新，今天的 Internet 呈现出了许多新的特点，造成当前 Internet 网络拥塞的原因，已经发生了变化，主要有以下一些特点：

1、不同的应用要求有不同的服务质量（Qos），如吞吐量、传输时延、时延抖动等。目前 的 Internet 不提供服务质量保证，只许诺以点对点的“尽力而为”(Best effort)服务。如何在 Internet 中实现服务质量保证是当前通信研究领域十分关注的问题，已有许多种 Qos 服务模型和技术被相继提出了。一个合理的 Internet 的资源分配方案是实现服务质量保证的关键技术。目前 Internet 的资源分配主要在用户端和中继网络设备中进行。

2、网络多媒体的推广对 Internet 的正常运行带来了严峻的挑战。多媒体实时应用和部分用户自己编写的网络程序。这类应用在 Internet 上传输时，均要求网络能提供大量的网络带宽，而且对报文的延迟也有很高的要求。由于这些多媒体 UDP 流应用不采用拥塞控制机制，这就导致了网络资源分配的严重不公平。这种不公平最终会导致 TCP 流饿死，直至网络崩溃。

3、近年来，Internet 中涌现出大量的应用，如软件分发、视频、远程教学、信息发布等。这些应用属于组播应用。组播技术能够提高网路传输效率，但是组播通信技术还有许多急待解决的问题如：组播路由问题、组播安全问题、组播拥塞控制问题、组播数据恢复问题等、这些问题中任何一个得不到解决，推广组播应用都将遇到巨大困难。

6.2　Internet 上的拥塞控制研究方向

一般来说，不同网络层次中的控制机制具有不同的目标，拥塞现象持续的时间越长，实现控制的层次也应该越高。拥塞控制可以在网络协议的各个层次上实施。由于数据链路层靠近拥塞的发生点，所以在数据链路层进行控制可以快速响应拥塞，但它只能对短期的拥塞现象进行控制。明确的拥塞控制主要在网络层和传输层实现。

Internet 本质上是一个尽最大努力(best-effort)传送的网络，IP 层底下基本不涉及流量控制问题，更谈不上拥塞控制。尽管 IP 层也想进行拥塞管理，但是大部分工作还是由传输层来完成，因为解决网络拥塞的真正办法就是降低数据的发

送率。

因为， Internet 上的拥塞控制，是假设 IP 层在指示拥塞和控制拥塞方面不提供任何显式的支持，为了进行拥塞控制，必须使用诸如确认、否定确认和超时等隐含信号来推断网络的状态，因此，采用基于端到端的拥塞控制机制，实质上就是 TCP 的拥塞控制。

Internet 的新特点要求研究新的技术和开发新的协议对网络施加更合理、更有效的管理和控制。随着 Internet 的不断成长壮大，许许多多的专家与学者为此付出了极其艰辛的劳动，留下了辛勤的汗水，产生了大量的研究文献和实践成果。

现在，有些文献提出把已有的、成熟的控制理论引入到 Internet 的网络拥塞控制过程中来。例如：建立数学模型；利用网络的学习能力，引入神经网络；利用模糊控制理论或者是自适应控制方法。但是，这里有几个普遍问题，目前还无法解决：（1、）算法假定的情况很不全面，在设计算法时，一般只考虑一个链路，忽略了多个链路相互影响的因素；（2、）对数据源的认识不深入。特别是 UDP 流，不能用简单的 ON/OFF 来仿真；（3、）控制策略不完善。现有策略只考虑了拥塞避免，没有考虑解除拥塞。当然更多的文献提出的是各种各样的 TCP 拥塞控制机制，或者是各种变形的 IP 控制策略。

但是，无论你把什么的理论引入到 Internet 的网路拥塞控制过程中来，也不论你提出了什么样的 TCP 拥塞控制机制，或者是推导出什么变形的 IP 拥塞控制策略来。经过全面系统的研究了拥塞控制领域最新研究成果，本文总结出目前拥塞控制的研究主要集中在三个方向：

1、继续改进现有的 TCP 拥塞控制算法（具体细节已经在第 4、5 章介绍），并仍然将其作为 Internet 上的主要拥塞控制机制。本课题所研究的工作即属于这一方向；

2、研究在中间网络设备（主要是路由器和交换机）中采用一定的策略来避免和控制网络拥塞，也就是后面马上就要讨论的 IP 拥塞控制策略；

3、利用价格等经济因素，限制用户对网络资源的需求，阻止拥塞的发生。因此，要建立网络收费新模型，用经济学理论分析具有拥塞特性网络资源的计费问题。 当然这个研究方向不是本文的主题。

6.3　Internet 的网络模型

拥塞是我们许多人都经历过的：交通堵塞、超级市场长长的付款队列等等。拥塞通常被定义为类似情形的一种状态。在对资源的需求超过了可用资源的一段时间之后便会出现这种状态。以日常生活中的交通堵塞为例，拥塞的发生是由于要求驶过道路的车辆超出了能在此道路上运行的车辆数，而且又在上下班时间

（一个特定的时间段内）。

Internet 中的拥塞控制不仅是一个非常重要的问题，而且是非常复杂的。拥塞控制的目的就是使信源不能淹没在整个网络中。拥塞现象的发生和 Internet 的设计有着密切的联系。Internet 的网络模型可以用以下几点来抽象。

1、报文交换（Packet switched）网络。和电路交换（Circuit switched）相比，报文交换通过共享提高了资源的利用效率，但在 共享方式下，如何保证用户的服务质量是一个很棘手的问题。在报文交换网络中可能出现报文“乱序”现象，对乱序报文的处理增加了端系统的复杂性。

2、无连接（Connectionless）网络。Internet 的各个节点之间在发送数据之前不需要建立连接，无连接模型简化了网络的设计，在网络的中间节点上不需要保存和连接有关的状态信息。但是使用无连接模型难以引入“接纳控制”（Admission control）算法，在用户需求大于网络资源时难以保证服务质量；在无连接模型中对数据发送源的追踪能力很差，给网络的安全带来了隐患，无连接同时也是网络中出现“乱序报文”的一个重要原因。

3、Best effort 的服务模型。Best effort 的涵义就是指网络不对数据传输的服务质量提供保证 ，和早期网络中的应用有关，传统的网络应用主要是 FTP、Telnet、SMTP 等，它们对网络性能（带宽、延迟、丢失率等）的变化不敏感，Best effort 模型可以满足需要。但是 Best effort 模型不能很好的满足新出现的多媒体应用的要求。这些应用对延迟、速率等性能的变化比较敏感。这就要求网络在原有的服务模型的基础上进行扩充。

6.4 TCP 拥塞控制的相关问题

在基于 TCP/IP 的 Internet 中拥塞控制是一个复杂而困难的任务，对这个问题，在几十年中作了大量的研究，进行了不少的试验，并且产生了许多论文结果。这个任务的困难之处在于下列几个因素：

1、TCP 只提供端到端流量控制，它只能通过间接方法推测中间的 Internet[Sfloyd91]给出了 TCP 处在拥塞避免阶段时数据包丢失率 p，发送率（带宽 B）以及回路响应时间 RTT 的稳态数学分析[32][52]。

即：
$$B < \frac{MSS \times C}{RTT \times \sqrt{P}} \qquad \text{公式（6—1）}$$

这里 MSS 是 TCP 数据包的长度，C 是常数，一般取 1.22。假定在一个瓶颈

链路处带宽为 LMbs，有 N 个固定的 TCP 连接。那么分配给单个连接的带宽近似为 L/N。将其代入上式并解出 p，

$$P<(\frac{N\times MSS\times C}{L\times RTT})^2 \qquad \text{公式（6—2）}$$

公式（6—1）和（6—2）表明：所以当连接均使用 TCP 拥塞避免算法时，包丢失率的上限值与连接数的二次方成正比。利用这一特性，可以改进拥塞控制机制。

大量的实践证明这种拥塞控制机制对 Internet 上大批量文件传输等尽量作好（Best effort）型服务具有较好的适应性。

2、TCP 拥塞控制的第二个问题是自相似（Self similarity）问题。近年来的研究分分析发现，不仅 ATM 的可变比特率（VBR）业务源和以太网产生的数据源具有自相似性，TCP 的拥塞控制也会产生具有短相关（Short range dependence）和长相关（Long range dependence）的自相似数据流（Self similar traffic）（可以用 Hurst 参数描述），而这一特性与更高层的应用或协议无关。用 TCP 拥塞控制的混沌本性(Chaotic nature)从另一侧面证明了这一观点。事实上，TCP 拥塞控制本身就是一个确定性过程。由于 TCP 拥塞控制工作在自时钟（Self clocking）方式，所以每个事件(例如发送数据包或计算超时)都都完全由过去决定，即系统的未来可以由过去描述。同时，其反馈控制也具有非线形（Nonlinearly）和周期性（Periodicity）等特点（例如 RTT 的计算、慢启动和拥塞避免等）。

3、TCP 拥塞控制的第三个问题是效率（Efficiency）。网络资源的使用效率总是由源端要求的总资源与网络资源的接近程度决定的。如果源端总资源 $x(t)=\sum xi(t)$ 接近和等于网络所能提供资源 xgoal，那么这种算法就是效率就是高的；超载（x(t)>xgoal 或负载不足 x(t)<xgoal）都是效率不高的表现。显然，效率只与总资源的利用率有关，而与各个源端之间的资源利用无关。

4、TCP 拥塞控制的第四个问题是公平性（Fairness）问题。有关公平性问题的进一步叙述，已经在第五章进行进行过深入研究。

6.5　TCP 与 IP 拥塞控制比较

TCP 基于窗口的端到端的拥塞控制对于 Internet 的稳定性起到了关键性作用。然而，随着 Internet 的迅速发展，其网络规模越来越庞大，结构日趋复杂，仅仅依靠端到端的拥塞控制是不够的。

IP 拥塞控制策略，有时我们也称其为路由器的拥塞控制策略，IP 拥塞控制策略是指在路由器中采用包调度算法和缓存管理技术。当前，Internet 中的路由器一般采用简单的先来先服务（FIFO）调度算法和尾部丢弃（Drop tail）缓冲管理方法。当然还有常用的一些其他重要方法如：（1、）数据报丢弃优先级（Drop priority）（2、）显示拥塞指示算法（3、）公平排队算法（4、）加权公平排队算法（5、）随机早期检测算法（6、）主动队列管理

基于信源的控制已被成功地用于局域网，但结合网络层的控制才能将其应用于 Internet。基于信源的拥塞控制通常利用网络层协议（例如 IP）来传送反馈，利用传输层协议（例如 TCP）来减少拥塞。在基于信源的策略中，信源的复杂程度可以是不同的。基于信源的策略可以提高效率，但可能不公平。

表 6－1　TCP 与 IP 拥塞控制策略的差异

	基于 IP	基于 TCP
实例	RED、FQ	慢启动
延迟	无	反馈延迟
反馈开销	无	反馈消息或比特
头开销位置	路由器	信源端
公平性	容易实现	不易实现
突发业务适应性	较灵活	较弱
拥塞时间	短	比反馈延迟长

基于 IP 的拥塞控制策略实施在网络层，不必依赖信源来均匀地分配资源，所以不存在以上那些问题。基于 IP 的拥塞控制策略是公平的，基于 IP 控制的主要问题在于增加共享资源（主要是路由器）的复杂性。事实上，很多策略的复杂性是与共享路由器的信源数成正比的，当网络的链路速率增加时信源数量也随之增加。

当然，基于 IP 的控制也有缺点，因为除非信源减少流量，否则无法缓解拥塞情况。短期拥塞不存在这一问题。但当长时间拥塞时，拥塞将通过临近的路由器和链路在网络中扩展，并最终扩展到信源。因此，基于 IP 的策略有助于强化拥塞时的公平性，而基于信源的控制能够承受长期拥塞。

6.6　改进型的 SACK 算法在 Internet 上的应用前景

经过前面两章的理论分析和仿真试验分析，我们可以得出如下的理论结果：改进型的 SACK 算法在大多数的情况下确实能够改善网络性能；同时它确实能够提高终端用户平均吞吐量。并且不会损害 TCP 的性能。而且稳定性还可以。下面我们来仔细的分析它对 Iiternet 可能产生的益处，具体来讲，好处如下：

1、网络资源利用率高

网络资源的使用效率是由源端要求的总资源与网络资源的接近程度所决定。如果源端总资源接近或等于网络所提供的资源，那么这种算法的效果就是高的。超载和负载不足都是效率不高的表现。从图 6—1 和图 6—2 中我们可以清楚地看出改进前后的网络资源利用情况。

2、启动速度更快

由于拥塞窗口的初始值 IW 被设置为 3—4 段，所以启动速度更快，能够尽快的到达网络传输极限。

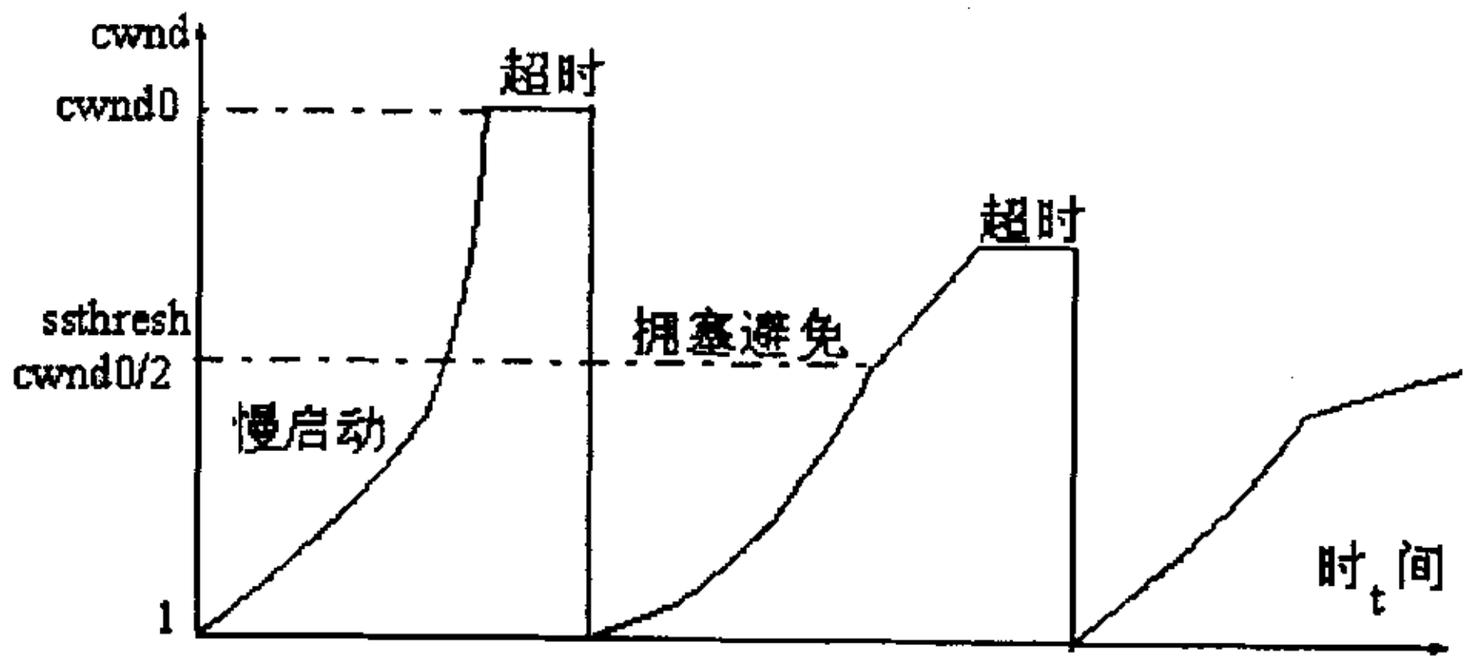


图 6－1　改进前网络利用情况

3、超时恢复更快

原有系统在超时重传后，立即将慢启动阀值（Ssthresh）减小到当前窗口的一半，并进入慢启动阶段，当 Cwnd 大于 Ssthresh 时，才进入拥塞避免阶段，直到进入拥塞避免阶段，由于省略了慢启动阶段，所以系统能够更快的恢复到网络资源的极限。这一点非常类似于快速重传/快速恢复。

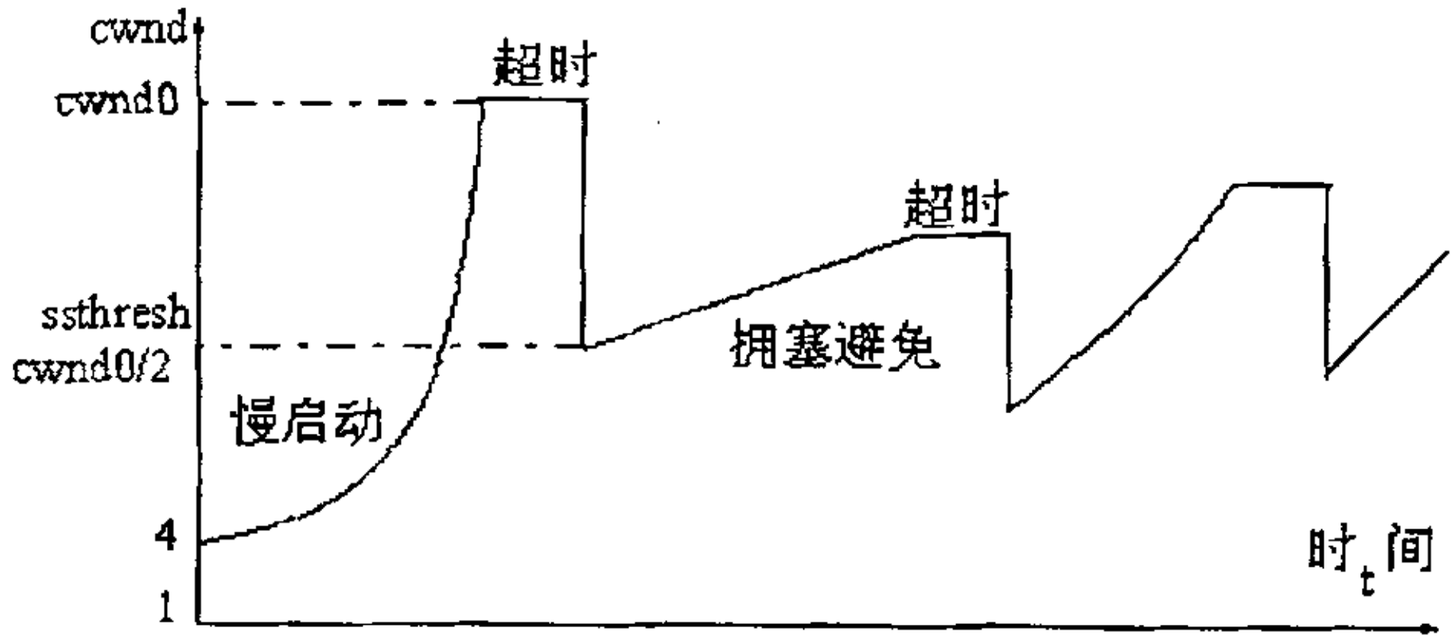


图 6－2　改进后网络利用情况

拥塞控制是 TCP/IP 协议研究的重要内容。随着 Internet 在全球范围内的蔓延，网络拥塞的避免和控制成为越来越重要和争待解决的问题，其相应 TCP/IP 协议的拥塞避免及控制算法的研究与改进也变得更加重要和急迫。

经过许许多多的专家经过十多年的深入研究，目前 TCP 协议主要包含有四个版本：TCP Tahoe、TCP Reno、TCP New Reno 和 TCP SACK。目前 Internet 上广泛采用的 TCP 基于窗口的和式增加积式减少（Additive increase multiplicative decrease， AIMD）的拥塞控制策略。TCP 的拥塞控制采用的是基于窗口的端到端的闭环控制机制。主要的实现方式就是 TCP Reno 在 TCP Tahoe 算法。在实际网络中，New Reno 和选择性应答（selective acknowledgement，SACK），这两种 TCP 实现方式目前尚未得到广泛应用。

经分析发现：New Reno 的最大缺点是：鲁棒性不好；很明显，SACK 是目前所使用算法中性能最优的。TCP SACK 关注的也是一个窗口内多个数据包丢失的情况，它避免了之前版本的 TCP 重传一个窗口内所有数据包的情况，包括那些已经被接收端正确接收的数据包，而只是重传那些被丢弃的数据包。虽然标准 SACK 算法性能优良，但仍然存在一个较为严重的问题，因为 SACK 算法也会经历慢启动阶段，而慢启动本身存在缺陷。

通过仿真试验证明了：改进型的 SACK 确实能够提高终端用户平均吞吐量。使用在扩大的初始窗口在一般情况下，确实可以提高网络的吞吐量，并且不会损害 TCP 的性能。而且稳定性还可以。虽然这个解决方法还不是特别完善，但在很大程度上，它确实能够解决目前所面临的问题。由于还没有在广泛的网络环境中经过中大量的试验论证，距正式使用，还有待进一步的研究和改进。

TCP/IP 拥塞控制的设计和实现面临许多的抉择与难题，应当有信心地期望将会研究出通用而有效的拥塞控制方法。

第七章 结论与展望

7.1 研究结论

拥塞控制是确保 Internet 的鲁棒性(robustness)的关键因素，也是各种管理控制机制和应用的基础，可以毫不夸张地说，没有 TCP/IP 拥塞控制算法的引入，就不会有今天 Internet 的成功。

计算机网络经历了过去十几年爆炸式的增长后，新型网络应用的不断涌现和用户数量的迅速增加，使得 Internet 的流量急剧增长，随之而来的就是越来越严重的拥塞问题。其相应 TCP/IP 协议的拥塞避免及控制算法的研究与改进也变得更加重要和急迫。但遗憾的是，国内对此技术进行研究的人并不多。

目前，标准的 TCP 协议实现包含了一些拥塞避免和控制的算法，但是，现在的这些算法都存在一些局限性，因此，对网络拥塞控制的进一步研究具有及其重要的理论和应用价值。

在此背景下，本文针对 SACK 算法的缺陷及存在的问题，进行了比较系统、深入的研究，提出了改进型的 SACK 算法：一种高效的 SACK 算法。即用 2 到 4 个分组（大约 4K 字节）的窗口代替原来一个分组的窗口，并通过理论分析和网络仿真试验结果说明：改进型的 SACK 算法在大多数的情况下确实能够改善网络性能。

随着使用网络传输多媒体文件的迅猛增加，UDP 文件和 TCP 文件业务之间分配的公平性，就显得特别重要了。为此，论文研究了 TCP/IP 网络的公平性问题，讨论了使用改进型的 SACK 算法对于 TCP 文件的公平性。论文是在当今流行的 Linux 操作系统下。在总结已有的理论知识的基础上，通过仿真试验证明了：改进型的 SACK 确实能够提高终端用户平均吞吐量。并且不会损害 TCP 的性能。而且稳定性还可以。

全面系统的研究了拥塞控制领域最新研究成果，总结出目前拥塞控制的研究主要集中在三个方向：（1、）继续改进现有的 TCP 拥塞控制算法；本课题所研究的工作即属于这一方向；（2、）研究在中间网络设备（主要是路由器和交换机）中采用一定的策略来避免和控制网络拥塞；（3、）利用价格等经济因素，限制用户对网络资源的需求，阻止拥塞的发生。

论文的研究工作和结论对于研究和解决 Internet 的拥塞问题具有一定的参考价值；虽然这个解决方法还不是特别完善，但在很大程度上，它确实能够解决目前所面临的问题。由于还没有在广泛的网络环境中经过中大量的试验论证，距

正式使用，还有待进一步的研究和改进。

7.2 展望

伴随着 Internet 技术的飞速发展，特别是在网络的拓扑结构、协议、应用和实现的异构化逐渐加强的情况下，原有的拥塞控制假设和算法变得不再合适，在下面的这些方面尚待继续研究：

1、目前对 TCP/IP 拥塞控制研究基本上都是先通过仿真，然后再根据经验改进算法，一直缺乏一套有效的理论来指导。所以建立一套系统的理论来有效分析、设计 TCP/IP 拥塞控制协议也是一个重要的问题。

2、对 SACK 等系列拥塞控制算法的进一步改进。在数据包的确认和重传的实现上，可以考虑采用一些智能控制策略以期进一步地减少时延提高网络吞吐量，例如，可以引入自适应控制和模糊控制等手段。

3、将拥塞控制机制与网络传输服务质量 QoS 进行有机结合[21]。目前的拥塞控制研究主要集中在竭尽全力（best-effort）型服务，对有实时 QoS 要求的数据流拥塞控制机制的研究还不多，而这种研究对支持多媒体数据流传输是极有价值的。

4、在将来，移动主机和无线链路会继续增加，因而对 SACK 算法在无线网络应用中的研究是一个非常有意义的工作，我们下一步要做的工作就是这一方面研究，给出具体的解决方案。

5、随着新型分布式多媒体和分布式处理应用的发展，多点投递成为非常急迫的通讯要求。TCP 受本身协议机制约束无法支持多点投递，而基于 UDP 的 Mbone 应用和其它协议则能在不同层次支持多点投递和群组管理。不过，无连接的 UDP 没有流量和拥塞控制，因此对 UDP 和其它无约束协议的拥塞控制也成为一个研究热点。

值得指出的是，拥塞控制并不是一个孤立的问题，解决这一问题在结构上涉及到用户端系统、网络内部传输单元(例如路由器)的协同工作，在算法策略上涉及到数据流的调度算法、缓存管理技术等网络资源的分配。TCP/IP 拥塞控制的设计和实现面临着众多的折中(例如在复杂性和性能之间找到折中)，不可能有一种设计和实现在所有环境中都是“最好的”。现有的拥塞控制思路、方法和技术在多目标的不同环境中更是面临着挑战，它们还有许多要改进的地方。

我们认为，虽然拥塞控制是很一个复杂的问题，但目前人们的努力正朝着正确地方向前进。人们面临的挑战是如何来保持一个互相关联且容易改变的全球性的网络体系结构。

参 考 文 献

[1] 罗万明，林 闯，阎保平. TCP/IP 拥塞控制研究. 计算机学报. 2001,24 (1): 1~18.

[2] Tanenbaum A S. Computer Networks (Third Edition). New York: Pretice Hall, 1998.

[3] Steves W. TCP Slows Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC2001, 1997.

[4] Meom C. A new approach to model the stationary behavior of TCP Connections. IEEE Computer Society, 2000.

[5] Jacbson V , Michael J. Karels. Congestion Avoidance and Control. IEEE/ACM Transaction on Networking, 1998, 6 (3): 314 ~329.

[6] Mark A.Smith. Formal specification and verification of safety and performance of TCP selective acknowledgement. IEEE/ACM Transaction on Networking, 2002, 4 (10): 193~207.

[7] Floy S and Jacobson V. Random Early Detection Gateways for congestion Avoidance. IEEE/ACM Transaction on Networking, Vol.1, No.4, pp397~413, August 1993. URL http://www-nrg.ee.lbl.gov/floyd/

[8] J. Bennett, D. Stephens, H. Zhang. High speed, scalable, and accurate implementation of packet fair queuing algorithms in ATM networks. In: Proc IEEE ICNP'97, Atlanta, GA, 1997. 7~14

[9] Comer D E. Internet working With TCP/IP Vol I: Principles, Protocols, and Architectures (Fourth Edition). New York: Pretice Hall, 2001.

[10] Firor V, Borden M.A. A study of active queue management for congestion control. IEEE Computer Society, 2000.

[11] Jacbson V. Congestion Avoidance and Control. IEEE/ACM Transaction on Networking, 1998, 6(3): 314 ~329.

[12] Matsuo T K, Hasegawa G. Comparison of fair service among connections on the Internet. IEEE Computer Society, 2000.

[13] Clark D D. Observations on the dynamics of a congestion control: the effect of two-way traffic. Proceedings of ACM Inforcomm, New York, 2000: 133~147.

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

[15] Lee k, Renchandran k. An Integrated Source Coding and Congestion Control framework for Video streaming in the Internet. IEEE Computer Society, 2000.

[16] MoJ, La R J, Anatharam V A. Analysis and Comparison Of TCP Rneo and Vegas. Proceedings of Infocom99, New York, 1999: 1265~1273.

[17] Floyds F K. Simulation Based Comparisons Of Tahoe, Reno, and SackTcp. Berkeley: Lawrence Berkeley National Laboratory, 2000.

[18] Morris R. Scalable TCP Congestion Control. IEEE Computer Society, 2000.

[19] http://www.path.berkely （Anatharam V. A case study for TCP Vegas）

[20] Hamann T, Walrand J. A new fair window algorithm for ECN capable TCP

(New-ECN). IEEE Computer Society, 2000.
[21] http://www.mash.cs.Berkcley.edu
[22] http://www.transarc.ibm.com/Library/documentation
[23] http://ftp.ee.lbl.gov （Jacobson V. Modified TCP Congestion Avoidance Algorithm Technical Report）
[24] D. Stephens, J. Bennett, H. Zhang. Implementing scheduling algorithms in high-speed networks. IEEE JSAC, 1999, 17(6): 1145~1158.
[25] B. Suter, T. Lakshman, D. Stiliadis, A. Choudhury, Efficient active queue management for internet routers. In: Proc. Of Interop Engineers Conference, Las Vegas, NV, May 1998.
[26] Barden R T. Extending TCP for Transactions-Concepts. RFC1379, November 1992
[27] Hos J. Improving the Startup Behavior of a Congestion Control Scheme for TCP, In ACM
[28] 刘拥民，年晓红. 对 SACK 拥塞控制算法的研究 信息技术， 2003，9：55~58.
[29] AIIman M., et. Al. On going TCP Research Related to Satellites. RFC 2760, February2000
[30] Wanng H Williamsom C. A New TCP Congestion Control Scheme: Smooth-Start and Dynamic recovery, in Proc. on IEEE MASCOTS'98, pp, 69~76.
[31] Janey C, Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP, Proceedings of SIGCOMM'96, pages 270~280,August 1996.
[32] Floyd S, Fall K. Promoting the Use of end-to-end congestion Control in the Internet. IEEE/ACM Transactions on Networking, May 1999
[33] Paxsom V, Measurements and Analysis of End-to-End Internet Dyamics. PHDthesis.U.C Berkeley 1996
[34] Henderson H, Katz R. On Improving the Fairness of TCP Congestion Avoidance. Proceedings of IEEE Globecom'98 Conference, 1998
[35] Fall K. Ns notes and Documentation: Collaboration between researchers at UC Berkeley, [M] LBL USC/ISI, and Xerox PARC, July 1, 1999
[36] UCB/LBNL/VINT Network Simulator- ns(version2). http://ww.isi.edu/nsnam/ns
[37] 杨凯锋, 虹佩琳, 束永安, 李津生. Internet 路由器中的拥塞控制策略. 小型微型计算机系统. 2000，21（4）：353~356。
[38] 王 斌, 刘增基, 李宏滨, 张 冰. 前向主动网络拥塞控制算法及其性能分析, 电子学报. 2001, 29(4) ： 438~486
[39] 刘喜成, 庞 斌, 向阳朝, 高 文. 通用拥塞控制及其在 Linux 内核中的实现, 计算机学报. 2001, 24(6) ： 657~660
[40] 吴 杰, 冯振明, 袁 坚, 刘文科. TCP 竞争资源的公平性研究， 电子学报. 2000, 28(8)
[41] M. Allman, and D.Glover, Enhancing TCP Over Satellite Channels using Standard Mechanisms, RFC2488, January 1999
[42] H.Kanakia, P.Misha, and A.Reibman. An Adaptive Congestion Control Scheme for Real Time Packet video Transport, IEEE/ACM Trans. Networking,

3:671~682, Dec 1995.

[43] H.Balakrishnan, V. N.Padmanabhan, S. Se-shan, etc. TCP Behavior of a Busy Internet Server: Analysis and Improvements. Proc. of IEEE Infocom'98, Mar. 1998. URL http://www.cs.berkeley.edu/hari/papers/infocom98.ps.gz.

[44] Thomas R. Henderson, Emile Sahouria, Steven McCanne, etc. On Improving the Fairness of TCP Congestion Avoidance, Proc. of the IEEE GLOBECOM'98, Sydney, Australia, November1998.

[45] Douglas E.Comer: Internetworking With TCP/IP Vol II: Design, Implementation, and Internals,(Second Edition), 电子工业出版社， 1998.7

[46] Douglas E.Come: Internetworking With TCP/IP Vol III:Clien-Server Pogramming and Application,(Third Edition), 电子工业出版社，1998.7

[47] V.Jacobson. Congestion Avoidance and Control, ACM Computer Communication Review (CCR), 18 (4): 314~329, August 1988

[48] L.Brakmo, S.Omalley, and L.Peterson, Peterson. TCP Vegas: New Techniques for congestion detection and avoidance, Proc. Of ACM SIGCOMM 94,Aug.1994. UCL http://http.cs.berkeley.edu/~kkeeton/Netread/tcpvegas-sigcomm94.ps.

[49] D.Chiu and R.Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. Journal of Computer Networks and ISDN, 17(1), June 1989.

[50] Floyd S. Henderson T. The New Reno Modification to TCP's Fast Recovery Algorithm. RFC 2582, February 1999

[51] Mark Allman, Sally Floyd, and Craig Patridge. Increaseing TCP's Initial Window, RFC2414, September 1998.

[52] J.Mahdavi and S.Floyd: TCP-Friendly Unicast Rate-Based Flow Control, URL http://www.psc.edu/networking/papers/tcp friendly-html, Jan.1997

附录一　网络仿真软件 NS 简介

1、1　NS 概述

NS 是 Network Simulator 的首字母缩写，它是由 LBNL(Lawrece Berkeley National Laboratory)的网络研究小组开发的仿真工具。NS 是一个面向对象的、基于离散事件驱动的网络仿真工具，它的源代码全部公开，提供开放的用户接口，并且可扩展、容易配置，用户可以很方便的将自己开发的新协议模块集成到 NS 环境中。

LBNL 的仿真软件的开发始于 1990 年 5 月对 S.Keshav 的 REAL 网络仿真程序的修改。91 年夏天，对仿真描述语言进行了修改，称为 TCPSIM，1994 年 12 月，Mccane 用 C++重写了 TCPSIM，称为 NS。

NS 仿真器由 10 万行 C++代码、7 万行 Otcl 代码、3 万行测试代码和 2 万行文本文档构成。NS 能够仿真多种局域网和广域网的性能，并支持多种协议，如传输层的 TCP，UDP 协议，应用层的 FTP，Telnet，Web 协议，支持 Droptail、RED 等几种路由器队列管理机制以及 Dijkstra、动态路由、组播路由等路由算法，此外 NS 还支持组播协议 SRM 及部分 MAC 层的协议。

免费的 NS 软件包及相关文档[35]可以在站点[36]中找到，目前的常见版本是 2.1b8a，该站点提供了两种下载方式：一种是 ns－allinone 软件包，它将 ns 中所需的所有组件全部压缩为一上文件，便于用户下载，安装过程非常简单；另一种是未经打包的一组软件。对于初学者推荐选用 ns－allinone 软件包。

NS 能在大多数 UNIX 平台下运行，包括 Free BSD、Linux 和 Sun Solaris。在 Windows 95/98/NT 平台下工作却不是很稳定，有些仿真工作能正常进行，某些则可能发生这样或那样的错误。由于 ns－allinone 软件包只能运行在 Unix 平台，因此用户可以在 Linux 平台下将 ns－allinone 软件包解压缩，ns 中的所有组件会自动安装在 ns－allinone－2.x 目录下，运行该目录下的 install 脚本，它将会自动配置、编译和安装所有的组件，完成安装过程。

1、2　NS 的安装步骤

我们选择的平台为 Linux，因为 Linux 是一种免费系统，可安装于微机上，网上有很多软件包都支持此平台。Linux 被称为 Unix 的 PC 版，是迄今唯一能够在 PC 机硬件平台上为广大用户免费提供多任务、多进程功能的操作系统。

NS 仿真器的安装步骤：（所用的 NS 版本为 NS-2.1b5，安装平台为 Linux）

1、将下载的软件包 ns-allione-2.1b5.tar.gz 拷贝到/home/luojun/目录下。

2、使用命令 cd /home/luojin，进入 NS 软件包所在目录。使用命令 tar –zxvf

ns-alinone-2.1b5.tar.gz将其解压，自动在当前目录下生成ns-allinone-2.1b5子目录。

3、用命令 cd ns-allinone-2.1b5，进入此子目录，运行 ./configure 命令生成 Makefile 文件。键入./make，编译并生成库文件和可执行文件。

4、make 完毕后，进入 ns-allinone-2.1b5 下的 ns-2.1b5 子目录，运行 validate，以验证 NS 软件中各协议安装的正确性。

1、3　NS 软件包的组成

NS 软件包 ns-allinone-2.1b5 中的主要组成部分如下：

Bin：有常用的工具 ns、xgraph、nam 等的连接。

Nam－1.0a8：动画显示工具。

Ns－2.1b5：NS 语言的核心部分。

Otcl－1.0a5：otcl 软件包。

Tcl8.0.4：tcl 语言软件包。

Tclcl－1.0b9：tcl 库文件。

Tk8.0.4：tk 工具包。

Xgraph：绘图显示工具。

1、4　NS 仿真器的常用命令

在仿真程序中的第一步是取得 Simulator 类的一个实例。

在 NS 中创建类的对象用 new 命令，删除对象实例用 delete 命令。例如，可以用下述命令生成 Simulator 类的一个对象。

Set ns [new Simulator]

进行仿真，必须从生成拓扑结构开始。NS 仿真器支持以下几种拓扑结构产生方法：

a：使用 GT-ITM 拓扑结构生成程序。

b：使用 Tiers 拓扑结构生成程序。

c：其它拓扑结构生成程序。

d：手工生成网络的拓扑结构。

本课题采用的方法是手工生成网络的拓扑结构。网络的拓扑结构可以通过指定三个基本的拓扑单位来实现：nodes、links、agents。Simulator 类中有创建和确认这些拓扑单位的方法。

Simulator 类中的 node 函数用来创建结点，并给每个结点自动分配一个唯一的地址。

Simplex-link 函数用来在两个结点之间建立单向连接，duplex-link 用来建立两个结点之间的双向连接。

Agents 是驱动仿真程序的对象。它可以被想象为在结点上运行的进程，结点

可以是端主机或路由器。网络业务源端和业务接收端、动态路由模块以及各种不同的协议模块都是 agents 的实例。通过实例化一个 Agent 类的子类对象可以创建一个 agent，Agent/type 表示子类对象，其中 type 表示是哪一种 agent。

在 NS 中用“/”表示类的继承关系，例如用 Agent/TCP 表示 TCP 是 Agent 的子类。例如，要一个 TCP 的 agent，则用下述命令：

set tcp[new Agent/TCP]

一旦创建了一个 Agent，系统也自动为其分配一个唯一的句柄。用 Simulator 将一个 agent 置于一个 node 上。

$ns detach – agent node agent

将一个 agent 从一个 node 上断开。

$ns connect src dst

建立 src 的 agent 与 dst 的 agent 之间的双向连接。返回 src 上 agent 的句柄。

$ns use – scheduler type

在仿真程序中使用 type 类型的事件调度。Type 是 List、Calendar、Real Time 中的一种。List 类型是缺省的。Calendar 事件调度使用一个 calendar 队列跟踪事件。RealTime 调度用于仿真程序和外部 agent 互相作用时的仿真模式。

$ns at time procredure

在仿真经过 time 时间执行 procedure，procedure 可以是全局局的可访问函数（proc）或是对象的方法（instproc）。这条命令可以用来开始或终止程序，动态的重新配置仿真程序。返回一个事件的 id 号。

$ns cancel edi

从事件队列中删除由事件 id 号 eid 指定的事件。

$ns now

返回当前的仿真时间。

$ns gen – map

路径仿真的拓扑结构，并列出已经生成的所有对象以及它们之间的所有路径。这有利于调试仿真的脚本程序。

1、5　NS 的常用组件

我们通过 NS 解释程序调用仿真器，NS 解释程序是 otclsh 命令 shell 的扩展。一个仿真是用一个 Otcl 脚本程序定义的。脚本程序使用 Simulator 类作为与 NS 的接口。通过此类中定义的方法，可以定义网络的拓扑结构，指定网络的业务源端和业务接收端，还可以通过 Simulator 类中的函数运行仿真程序，并收集数据。

下面介绍 NS 中常用几个组件：

1、事件调度器：由于 NS 是基于事件驱动的，调度器也就成为 NS2 的指挥

中心，它可以跟踪仿真时间，调度当前事件链表中的仿真事件（例如一个ＦＴＰ应用何时开始，何时结束仿真等），并交由产生该事件的对象处理。目前ＮＳ提供了四种具有不同数据结构的调度器，分别是链表、堆、日历队列和实时调度器，实时调度器可以将实际的网络流量导入一网络的仿真环境，从而可以在一个接近真实的环境中测试协议的性能，ＮＳ的实时调度器尚处于开发阶段。

2、节点（Node）：是由 TclObject 对象组成的复合组件，在 NS 中可以表示端节点和路由器。每个节点具有唯一的地址（id 标识），节点有单播节点和组播节点两种不同类型，通过节点内部的 nodetype 变量来区分，NS 中默认的是单播节点；节点为每一个连接到它的业务源分配不同的端口，用于模拟实际网络中的端口；另外，节点有一个路由表以及路由算法，由地址分类器（Addr Classifier）根据目的地址转发数据包。

3、链路（Link）：由多个组件复合而成，用来连接网络节点。所有的链路都是以队列的形式来管理分组到达、离开、和丢弃。在链路中增加 Trace/EnqT、Trace/Deqt、Trace/DrpT 以及 Trace/RecT 等对象可以跟踪每个数据包到达、进入、离开队列以及被丢弃的时间；还可以用队列监视器（Queue Monitor）来监测队列长度和平均队长的变化情况。

4、代理（Agent）：负责网络层分组的产生和接收，也可以用在各个层次的协议实现中。Agent 类包含源及目的节点地址、分组类型、大小、优先级等状态变量，并利用这些状态变量来给所产生的分组的各个字段赋值。每个 Agent 连接到一个网络了点上（一般是端节点），由该节点给它分配一个端口号。Agent 是实现 UDP 协议及各种版本 TCP 协议的基类。

5、包（Packet）：由头部和数据两部分组成。头部包括 cmn header、ip header、tcp header、rtp header 及 trace header 等，其中最常用的是通用头结构 cmn header，该头结构中包含一个唯一的标识符，包类型、包的大小以及时间戳等。头结构的格式是在仿真器创建时被初始化的，各头部的偏移量也被记录下来。在 Agent 产生了一个包之后，所有的头部都同时生成，用户能够根据偏移量来存取各头部所包含的信息。一般情况下，Packet 只有头部，没有数据部分。

总地来说，网络仿真软件 NS 功能强大，应用方便，能够支持多种软件协议，对于分析评价网络模型具有重要作用，NS 的缺点是当网络拓扑结构较大或仿真程序较为复杂时，其运行的效率会明显地降低。

附录二 缩 写 词

A

ACK (Acknowledgement) 确认
AIMD- Additive Increase and Multiplicative Decrement 线性增加倍乘减少
ARPA 国防部指定其下属的高级研究计划局
AS- Assured Service 确信服务
ATM (Asynchronous Transfer Mode) 异步传递方式

B

BE- Best Effort Service 尽力而为服务
BRED- Balanced Random Early Detection 平衡随机提前检测

C

CWND- Congestion Window 拥塞窗口
Congestion Control 拥塞控制机制
CCA- Congest Control Agent 拥塞控制代理
CSFQ- Core –Stateless Fair Queuing 核心无状态公平排队

D

DIFFSERV- Differentiated Service 区分服务

E

ECN- Explicit Congestion Notification 显式拥塞指示

F

FCFS- First Come First Service 先来先服务
FRED- Fair Random Early Detection 公平随机提前检测
FTP- File Transfer Protocol 文件传送协议
Flow Control 流控制机制

H

HTTP- Hyper Text Transport Protocol 超文本传输协议
HTML (hypertext markup language) 超文本标记语言

I

IETF- Internet Engineering Task Force 因特网工程任务组
IP- Internet Protocol 因特网协议
INTSERV- Integrated Service 集成服务

N

NSF 美国国家科学基金会
NS- Network Simulator ， NS 网路模拟仿真器

Q

QOS- Quality Of Service 服务质量

R

RED- Random Early Detection 随机提前检测
REM- Random Early Marking 随机提前标记
RIO- Random Early Detection With In And Out 区别随机提前检测
RSVP- Resource Reservation Protocol 资源预留协议
RTT-Round Trip Time 往返时延
Rwnd-receive window 接收窗口
Robustness 鲁棒性

S

SACK-Selective Acknowledgement 选择性应答
SCFQ-Self-Clocked Fair Queuing 自同步公平排队.
SMSS-Sender Maximum Segment Size 发送端最大段长度
Sshresh-Slow-start threshold 慢起动门限

T

TCP-Transport Control Protocol 传输控制协议

U

URL (Unshielded Resource Locator) 统一资源定位符
UDP-User Data gram Protocol 用户数据协议

W

WFQ-Weighted Fair Queuing 加权公平排队
WWW-World Wide Web 万维网

致　　谢

在本论文完成之际，首先，我要特别感谢我的导师——年晓红教授，感谢他在我攻读硕士研究生期间对我学业上的悉心培养和不倦教诲，以及生活上无微不至的关怀和照顾。年老师严谨的治学态度和忘我工作的敬业精神以及亲切宽厚的生活作风令我深受感动，生活和学习在我们研究所里让我觉得十分开心，而且很有收获。他那孜孜不倦的学习态度和勤勤恳恳的工作作风，在现在和以后都是我学习的楷模。并时时激励我在学习和工作中不断进取。

同时，在三年的研究生学习过程中，我得到了中南大学自动化工程研究中心罗大庸教授、樊晓平教授、张航博士、黄芳博士、申立博士、杨胜跃博士、张亚鸣博士、刘少强博士等老师的指导和帮助。在此，我向你们表示衷心的感谢和良好的祝愿！

另外，我还要感谢和我朝夕相处的同学和朋友们。他们对我的研究工作提出了很多建设性的意见，与他们的讨论使我受益非浅。谢谢师兄罗熊博士、同学王家凡、张恒、彭展，陈艳琴、杨玺、刘浩、周宏广、赵战克、赵明，以及我的师弟杨东候、王凌云，谢谢你们！

感谢中南大学研究生院的各位老师和信息学院的各位老师，正是他们的辛勤工作为我们顺利完成学业提供了良好的学习条件。感谢他们对我生活和学习上的关怀和帮助，在此，我向你们表示衷心的感谢！

最后，我要感谢我的父母亲和姊妹们，以及所有关心着我、爱我的人，感谢他们多年来对我学业的巨大支持。

再一次真心地谢谢大家！

作者：刘拥民

2004 年 5 月于中南大学

信息学院　自动化工程研究中心