TCP/IP 拥塞控制算法研究

Researches on TCP/IP Congestion Control Algorithms

苏晓丽　陈　晶　郑明春　孟　强

（山东师范大学计算机科学系　济南250014）（南京大学计算机软件新技术国家重点实验室　南京210093）

Abstract The research on TCP/IP congestion control is one of the most active fields in the computer network. In this paper, the first discussed are the purpose and principal strategies of TCP/IP congestion control. Then, a summary of some typical algorithms and improvements on them is presented. Finally research directions and open problems in this area are also discussed.

Keywords TCP/IP, Congestion control, Algorithm

1 引言

TCP/IP 拥塞研究兴起于80年代中期，随着 TCP/IP 协议的流行，越来越多的网络互连起来，网络应用和新的网络技术在不断发展。然而，由于网络用户的增加、新旧技术的并存，以及网络的异质性导致的许多不匹配的问题，如链路速度不匹配、网络各个中间接点处包的到达速度与处理速度不匹配等，引发了排队等待和拥塞现象[3]。

1984年 Nagle 指出了现有 TCP/IP 机制在拥塞控制方面的不足[1]，详细描述了由于其错误行为所导致的拥塞崩溃现象。从1986年10月开始，Internet 出现了一系列的拥塞崩溃现象，网络吞吐量急剧降低，许多分散在各地的网点被迫长时间停止服务。

网络产生拥塞的根本原因在于用户提供给网络的负载大于网络资源容量和处理能力[2]。拥塞一旦发生若不加以控制，往往会导致恶性循环，如果路由器没有空余缓冲区，它必须丢掉到来的分组，当扔掉一个分组时，发送该分组的路由器可能会因为超时而重传此分组，或许要重发多次，这样拥塞进一步加重，出现了严重的时延。在通信量非常高的情况下，网络会完全瘫痪，几乎没有分组能够送达。

随着计算机网络规模的不断扩大，用户急剧增长，拥塞状况越来越严重，这已经成为制约网络发展和应用的一个瓶颈，拥塞控制已经成为当前网络研究的一个热点问题，许多人已经开始致力于这一方面的研究。目前，研究人员已经提出了多种拥塞控制算法，如慢启动、拥塞避免、快速重传、快速恢复[5~7]、SACK[9]、RED[20,21]、ECN[22]、NEW-ECN[23]等。在本文中，我们首先介绍拥塞控制算法的目的与任务，比较各种典型的拥塞控制算法，讨论其中存在的问题及新的发展，最后提出了未来的工作方向。

2 拥塞控制策略的目的与任务

拥塞控制的主要任务是保证子网不被它的用户发送的数据所淹没[3]。如图1(a)所示，当负载比较小时，网络的吞吐量随负载逐渐增加，当负载达到网络最大容量时，吞吐量停止增加，若负载继续增长，则开始排队、丢包，当负载超过 Cliff 这一点时，吞吐量急剧下降，网络发生拥塞。时延也是如此(见图1b)，最初随着负载缓慢增长，当开始排队时，便开始线性增长，到队列溢出时，时延急剧增长。当吞吐量接近于零时，出现拥塞崩溃现象，时延趋于无穷。

拥塞控制策略的目的就是在网络接近于拥塞崩溃点时能尽快检测到，并采取措施减小网络负载，使网络恢复到正常状态，保持在 Cliff 点左侧运行。在 Knee 点处吞吐量的增长速度减小，而网络时延加速增长，在该处采取的策略称为拥塞避免策略，其目的是在没有显著的时延时，仍可以让用户适当增加通信量，当时延明显增加时，再要求其减少通信量，从而使网络负载在 Knee 处振荡。但是当突发的通信量使网络负载接近 Cliff 点时，必须采取拥塞控制策略。进行拥塞控制的最终目的是资源的使用效率达到最佳，一般地采用如下公式计算资源利用率(efficiency)[3]：

$$\text{efficiency}=\frac{Power}{Power_{\text{知识}}}$$

在 Knee 处利用率达到100%。其中 *Power* 为：

$$Power=\frac{\text{Throughput}^{\alpha}}{\text{Delay}}$$

如图1(c) 当 $\alpha=1$时，在 Knee 处 *Power* 达到最大；当 $\alpha>1$时，倾向于高吞吐量；当 $\alpha<1$时，对时延更敏感。

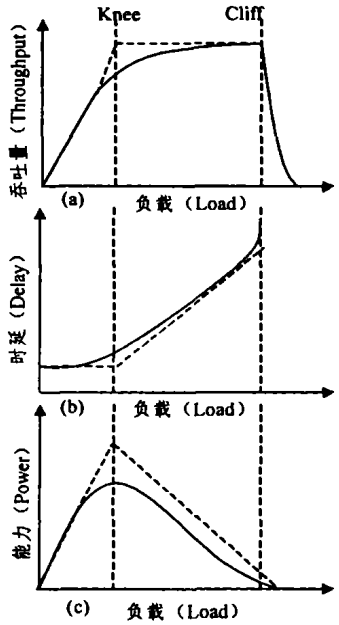

图1

3 拥塞控制的基本策略

拥塞控制算法可以分为两类[4]：开环与闭环。开环算法致

力于通过良好的设计来避免问题的出现，确保问题在一开始就不会发生，一旦系统安装并运行起来，就不再作任何中间阶段的更正，显然对网络这样不断变化的复杂系统来说，开环控制并不是理想的选择。而闭环的解决方案是建立在反馈环路的概念之上的，按反馈方式可分为：显式反馈和隐式反馈。在显示反馈算法中，由拥塞点将拥塞信号反馈给源端；而在隐式算法中，源端通过局部观察来推断是否存在拥塞。拥塞策略主要包含三个部分[2]：

(1)监视系统，检测何时何地发生了拥塞。

(2)将此信息传送到可能采取行动的地方。

(3)调整系统操作以更正问题。

在基于 TCP/IP 协议的网络中主要使用的是闭环算法。从考察整个网络系统的参与者入手，我们将分别从端系统(资源的使用者)和通信子网(资源的提供者)两个角度来分析各种算法。

3.1 TCP(端系统处)拥塞控制策略

3.1.1 *基本策略* 在端系统处主要是通过降低数据发送率来解决拥塞，目前主要算法(由 Jacboson 提出)包括以下四个部分[5~7]：

(1)慢启动阶段(slow start)。在建立连接之前，路由器处没有任何预留资源，它只能通过观察流量来了解用户对资源的要求，同时用户也不知道网络中有多少资源可供他利用，因此为了防止连接开始时用户发送过量的数据而导致网络拥塞，TCP 使用慢启动算法来逐步探测网络的容量。由于快速网络向小容量接收器输送和慢速网络向大容量接收器输送，都可能产生拥塞，因此设定了两个参数来控制网络上数据的流量：cwnd 和 rwnd。拥塞窗口 cwnd 是发送方对发向网络的数据量作出的限制，通告窗口 rwnd 是接收方对网络上可传输的数据量的限制，实际发送窗口 win 是二者的最小值，即 win＝min(cwnd，rwnd)。TCP 使用慢启动策略，当建立连接时，拥塞窗口(cwnd)初始化为一个数据报大小，一个数据报缺省为536或512字节，源端发送数据后，每收到一个 ACK 确认，cwnd 就增加一个数据报的发送量，通过这样的方式来探测网络可供使用的容量。显然，cwnd 随 RTT(指从源端发送数据包至收到接收端确认的时间间隔)呈指数级增长：1个、2个、4个、8个……，源端向网络中发送的数据量急剧增加。实际上，慢启动算法并不慢，要找到可利用的最大窗口 W，需要的时间为 Rlog2W (R 是往返时间)[5]。当 cwnd 增加到某一个值时，发送方一次发送的信息量就可能达到或超过了网络的容量，中间的路由器开始丢包，通知发送方它的拥塞窗口已经太大了，发送方需采取相应的措施进行调整，尽量使得负载保持在 Knee 附近，达到最佳吞吐量。如果一直不出现超时现象，拥塞窗口会一直增大到接收方的窗口 rwnd 的大小，此时拥塞窗口将停止增大。只要不出现超时，并且接收方窗口也保持不变，拥塞窗口将保持不变。

(2)拥塞避免阶段。当拥塞窗口 cwnd＞sstresh 时(sstresh 为慢启动阈值)，就进入了拥塞避免阶段，cwnd 增长速度减慢，每收到一个非重复的确认，令：

cwnd＝cwnd＋SMSS * SMSS/cwnd

(注：SMSS 为发送方最大报文段长度，在 TCP 实现中采用的是整数计算，所以当 cwnd＞SMSS * SMSS 时，将结果四舍五入，即 cwnd 最多增加一个字节)。如果发现超时，则表示有数据包丢失，网络已发生拥塞(TCP 这一假定是基于由传输引起的数据包损坏和丢失的概率很小(＜1%))[5]，这时要进行相应的拥塞控制，将慢启动阈值设置为：

ssthresh ＝ max(2，min(cwnd/2，rwnd))

对于保留在发送窗口中的报文段，将重传定时器的时限加倍。如果检测到超时还要将拥塞窗口 cwnd 置为1。若 cwnd≤ssthresh，TCP 重新进入慢启动阶段；若 cwnd＞ssthresh，TCP 就执行拥塞避免算法。

(3)快速重传与恢复阶段。当接收方收到重复的 ACK 确认时，表示网络已经出了一些问题，若要等到定时器超时再进行控制，中间有一定的时延，而且当数据包超时时，cwnd 要被置为1，重新进入慢启动，这会导致过大地减小发送窗口的尺寸，降低 TCP 连接的吞吐量，所以采用快速重传与恢复算法，即在收到3个或3个以上的重复 ACK 时就断定该数据包已丢失，采取下列步骤：

```
    /* 快速重传 */
set ssthresh ＝ max(2，min(cwnd/2，rwnd))；
set cwnd ＝ ssthresh＋3 * MSS；
    /* 快速恢复 */
if 一个重复的 ACK 确认到达
then cwnd ＝ cwnd＋1；
if 对新发送数据的确认到达
then cwnd ＝ ssthresh；
```

以上四个基本算法互相联系，是目前在 Internet 上被广泛使用的端系统拥塞控制机制。但是在实际应用中仍然存在一些不足，近年来出现了许多改进的算法。

3.1.2 *改进及存在的问题*

(1)对慢启动的改进

·大初始窗口 慢启动算法通过逐渐增加 cwnd 的大小来探测可用的网络容量，防止连接开始时采用不合适的发送量导致网络拥塞。然而有时该算法也会浪费可用的网络容量，因为慢启动算法总是从 cwnd＝1开始，每收到一个 ACK，cwnd 增加1，对 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，特别是网络实际容量很大时，会造成浪费。文[7，8]建议对此进行一定的扩展，使用大初始窗口，建立连接后，设初始窗口：IW＝min(4 * MSS，max(2 * MSS，4380 bytes))，其中，MSS(maximum segment size)指这一连接的最大报文长度，这样 TCP 发送端在刚开始就可以发送3个1460字节或4个512字节的分组，从而减少3个 RTT 时间和一个延迟 ACK 时间，但是同时它也会导致附加分组的丢失，降低网络性能，这一建议还在讨论之中，没有成为标准。

·估计慢启动阈值 ssthresh 在慢启动阶段，在每个 RTT 时间内，cwnd 增加一倍，这样当 cwnd 增加到一定的值时，就可能导致以网络能够处理的最大容量的两倍来发送数据，从而淹没网络。因此，在1996年 Hoe 建议使用 packet-pair 算法和测量 RTT 来为 ssthresh 估计合适的值[14]，以此来适时地结束慢启动阶段。根据 Hoe 的方法，发送方利用接收到的 ACK 确认个数和到达时间来推测瓶颈链路的带宽，令：

ssthresh ＝ max(1 segments，delay * bandwidth)

当 cwnd≥ssthresh 时，开始进入拥塞避免阶段，模拟试验已经显示，设置 ssthresh 的值在一定程度上可以提高网络性能、减少丢包率[14]。但是如何准确估计动态网络可利用的带宽仍然是一个有待解决的问题，M. Allman 等人在发送方作了一些尝试[13]。

(2)对重传与恢复的改进

·选择确认 SACK 源端检测到拥塞后，要重传自丢失数据包之后，至检测到丢失时发送的全部数据包(即 Go-back-n 算法)，而实际上二者之间有些数据包已正确传到接收端，不必重传。文[9]中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，这样源端就能准确地知道哪些数据包正确地传到接收端，从而避免了不必要的重传，减少了时延，提高了网络的吞吐量。

·**有限传输机制** 如果分组非顺序到达接受方，那么也会产生重复的 ACK，而只有等收到三次重复的 ACK 时才能激发快速重传，因而导致了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小发送量，导致不必要的带宽浪费。而且当一个窗口中多个分组丢失时，如果没有足够的(连续三个)重复的 ACK 到达，快速重传算法便会失效。文[28]中使用有限传输机制，通过对重复的 ACK 迅速做出反应，快速重传丢失的分组，但不改变 cwnd 大小，提高了 TCP 的丢失恢复能力。

另外，对超时重传定时器 RTO (retransmition timeout) 和 RTT 的估计方面也存在很多问题。文[26,30]中提到的 New-Reno 不依赖重传定时器，消除了当一个窗口中丢失多个分组时对重发计时器的等待，尽力避免了在快速恢复阶段的许多重传超时，利用一个 ACK 确认部分发送窗口，立即重传余下的分组。文[15]中 Vegas 算法通过观察以前的 TCP 连接中 RTT 改变的情况来控制拥塞窗口 cwnd，如果发现 RTT 显著变大，便认为网络发生拥塞，减小 cwnd；如果 RTT 变小，就解除拥塞再次增加 cwnd。但是 Vegas 算法还没有被普遍采用，因为它还存在一些公平性的问题未解决。

(3)对公平性的改进。在拥塞避免阶段，如果没有丢包，那么 TCP 发送方的 cwnd 在每个 RTT 时间内大约可以增加一个报文段大小，但是这种方法会造成具有不同 RTT 时间或窗口尺寸的多个连接在瓶颈处对带宽竞争的不公平性，RTT 时间长或窗口小的连接，相应的 cwnd 增长速度缓慢，所以只能得到很小的一部分带宽。

·**Constant-Rate Increase 策略** 解决上面提到的问题可以通过在路由器处使用公平队列[10]和 TCP 友好缓存管理[16]来进行控制以增加公平性。然而如果没有路由器的参与，要增加公平性则要求 TCP 发送端的拥塞控制策略进行相应的改变，文[17,18]中提到了一种稳定速度增加策略(Constant-Rate Increase)，在拥塞避免阶段中使共享同一资源的各个 TCP 连接以相同速度发送数据，从而确保了各个连接之间的公平性。

·**Increase-by-K 策略** 对具有长 RTT 时间的连接还可以选择一种每次增加 K(Increase-by-K)的策略，这种策略改变了 cwnd 线性增长的斜率，对 RTT 超过一定阈值的 TCP 连接，每个 RTT 时间内增加 K 个段，而不是一个段。文[18]表明对于共享一个瓶颈链路的少数几个连接来讲，使用一个较小值的 K，可以在保持链路的高利用率的同时减少不公平性。

上述方法都要求 TCP 发送方拥塞控制算法的变化，Constant-Rate 要求的是全局性的变化，所有的连接全部参与，而且发送方必须对每个连接的 RTT 有准确的估计。而 Increase-by-K 算法易引发一个窗口中多个分组丢失的情况，从而引起更多的超时重传，因此该算法最好与 SACK 丢失恢复算法一起使用。

(4)其他方面的改进

·**ACK 拥塞控制** 由于 TCP 采用 ACK 自时钟技术[1]，以 ACK 来触发新数据的发送，因此 ACK 的行为也直接影响了发送方的速度，尤其是在不对称的网络中 ACK 的路径可能与数据发送的路径不是同一条路，ACK 直接影响着数据的发送与 RTT 的估计[1,4]，因而 ACK 拥塞控制需要认真的研究，文[19]对此进行了初步的分析。

·**RPB(Rate-Based Pacing)算法** 当一个 TCP 连接在闲置一段时间后要重新开始传输，可以使用慢启动算法，但可能会浪费带宽，文[27]中提出了 RPB(rate-based pacing)算法，在缺少 ACK 确认时，数据发送方以一定的速度发送 TCP 分组来开始 ACK 自时钟。在接收到第一个返回的 ACK 确认后该算法停止工作，开始正常的 ACK 自时钟。调节的速度可以从最近的通信量进行估计，或者从其他外部方式获得。此时一个 TCP 连接必须使用非标准的初始 cwnd 以及增加新的机制来调节数据发送间隔，即每隔一段时间发送一次，而不是紧接着一个又一个地发送，这样便减小了数据的突发性。这种技术仅要求发送方进行一些变化，在发送方设置一个发送定时器作为调节的依据，但是如何确定 cwnd 的初值以及发送时间间隔仍有待于进一步讨论。

值得指出的是，在端系统处进行的拥塞控制从感知到拥塞采取行动之间有着明显的时延，而且它无法了解资源的使用情况，目前主要是通过降低发向网络的流量来减小网络负载从而解决拥塞，这在一定程度上可以提高效率，但是很难保证公平性，要保证公平性以及进行有效的控制，最好能在对拥塞情况有足够信息的子网中，即在拥塞发生地进行相应的控制。在 IP 层进行的控制实施在网络层，可以不依赖信源，均匀分配资源，但不可避免地增加了共享资源(主要是路由器)的复杂性。

3.2 IP(在路由器处)拥塞控制策略

在无连接的网络中，路由器对每一对传输实体的资源要求并不了解，因而无法提前预测未来的通信量，它们总是试着通过监视当前的负载来要求用户增加或减少发向网络的数据量，以便使网络能运行在最佳状态，保持较高的资源利用率和较好的公平性。但是在路由器处进行拥塞控制必须要有用户的配合，因为除非信源减少流量，否则无法缓解拥塞状况。这里主要有以下三个方面需要考虑：

3.2.1 拥塞检测

在路由器反馈任何信息之前，必须估算出当前的负载水平，是在 Knee 之下还是之上。其判断通常有两种方式，一种是根据缓存区的利用率，另一种是根据平均队列长度。

如果在缓存区满时才作出拥塞反馈，会使得网络的吞吐量振荡加剧。为加强网络的稳定性，使路由器能尽早检测出拥塞，人们又提出了一些改进办法，如在缓存区中设置一个阈值[1]，当超过缓存区容量的一半时，表示即将发生拥塞，应作出反应。或者是为队列设置一个界限，当超过这一界限时，开始通知源端。这些方法中较为著名的是 Floyd 在1993年提出的 RED(Random Early Detection)算法[20,21]，为队列设置了两个阈值 Q_{min}、Q_{max}，当一个数据报到达路由器时，计算当前平均队列长度 Qe 并将其与这两个阈值相比较来判断是否拥塞。

$$Qe=(1-\alpha)*Qe+\alpha*SampleQe$$

其中 $0\leqslant\alpha\leqslant1$，SampleQe 是采样测量时的排队队长。

3.2.2 检测间隔

在路由器估算自己的负载状态后，对用户的反馈信息应该而且必须能保持足够长的时间，使得用户能以此为依据采取相应的行动。如果该状态变化太快，在用户刚得知这一信息时，已经发生了变化，此时反馈信息就会对用户产生误导作用。因而需要对拥塞信息进行过滤，只保留那些能持续时间比较长的信息才有意义。比如在计算平均队列

时要排除瞬时队列长度，这就需要考虑计算平均队列的时间间隔，Jain 建议使用图2中的 Averaging Interval 长的间隔[3]，它包含了前一个工作周期和空闲时期以及当前工作周期的一部分。

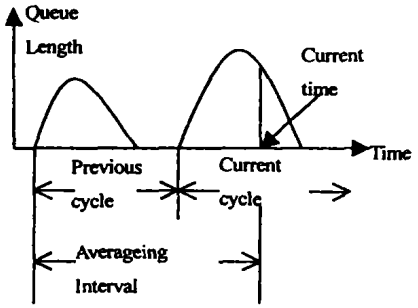

图2

3.2.3 *路由器处拥塞控制* 要解决的问题是如何对拥塞作出反应，如何将拥塞信号反馈给源端，一般地有以下策略：

·**源抑制(source quench)**。当路由器发现自身资源短缺，如缓存区满或达到容量的一半时[1]，路由器向源端发送 ICMP 源抑制报文，要求源端进行拥塞控制。但是这种算法有一个明显的缺点，它需要额外的通信代价，当网络拥塞时，发送抑制报文会加重拥塞。另外当缓存区满时，路由器工作总是丢弃最后到达的包，这种去尾(Drop tail)方式对间歇性的、低要求的用户易产生不公平问题[20]。

·**随机丢弃(Random Drop)**。该算法基于这样一个假设，随机丢弃的包属于那个在传输流中占很大比例的用户。1990年 Mankin 提出通过随机选择丢弃包[11]来要求那些通信量超过它们的公平资源份额(fair share)的用户调节流量[3]。后来 Floyd 对随机丢弃和去尾方式进行了改进，提出了 RED 算法[20]，设置了两个阈值 Q_{min}、Q_{max}，当一个数据报到达路由器时，将当前平均队列长度 Qe 与阈值相比较来判断是否拥塞，但 $Qe \leqslant Q_{min}$ 时，将此包排队，如果 $Q_{min} < Qe < Q_{max}$，计算概率 P，并以此概率丢弃包；如果 $Qe > Q_{max}$，则丢弃此包。这样 RED 可以及早通知源端减小拥塞窗口，以减少进入网络的数据量，这意味着路由器以后不必丢弃更多的包，而且可以避免丢弃属于同一连接的数据包，从而提高吞吐量。丢弃概率 P 是平均排队长度及上次丢包以来经历时间的函数：

$$T = \max_p \times (Qe - Q_{min})/(Q_{max} - Q_{min})$$

$$P = T/(1 - count \times T)$$

其中 T 是临时变量，count 表示在两个阈值之间的平均排队长度。P 值随 count 的增长而缓慢增加。

这种策略是以丢弃数据包作为拥塞信号，通过源端发现超时或重复 ACK 的方式隐式地通知拥塞情况，其间存在了一定的时延，而且由丢包引起的重传同样会加重网络的流量。

·**拥塞指示算法(Congestion Indication)**。该策略思想是使用 IP 首部的一位(称为 congestion avoidance bit)作为拥塞信号反馈至端系统，源端将发送的每一个包的该位置0，网中所有的路由器计算它们的负载，一旦发现运行在 Knee 之上，则将引起超载的用户的包首部中的拥塞避免位置位，当端系统收到置位的包后，进行相应的处理。这种策略有两种方式：基于源端的和基于目的端的[3]。

在基于目的端的算法中，由目的端决定新的流控窗口，通告给源端；在基于源端的算法是由目的端将该 bit 通过确认信息捎带至源端，在这种情况下应在 TCP ACK 包的首部保留一位以便能将拥塞避免位中的内容拷贝到其中。源端的 TCP 据此进行相应的操作。

该算法参考 DECbit (源于 the Digital Network Architecture，DNA)，在 DNA 中的 NSP 传输层协议使用的是基于源端的方法，而 ISO TP4使用的是基于目的端的方法，文[29]提出了在基于 TCP/IP 的网络中使用这种算法称为 Binary Feedback Scheme。Floyd 在 RED 基础之上提出了 ECN (Explicit Congestion Notification) 算法[21,22]，使用 IP 首部 TOS 域的最后两个保留位作为两个标志，在源端数据包中嵌入 ECN 使能发送比特位 ECT(ECN-Capable Transport bit)，由路由器根据网络情况设置 CE(Congestion Experienced)比特位。源端接收从网络中反馈回的这种 CE 置位的数据包，然后将随后发出的数据报标志为可丢弃的数据包。ECN 的优势在于不需要重传超时，也不依赖于 TCP 定时器，所以对时延有一定要求的应用效果较好。2000年 Hamann 和 Walrand 提出了一种改进的算法 New-ECN，它通过参数调整拥塞窗口 cwnd 大小，纠正了有长时间 RTT 的 TCP 连接的偏差，改进了共享瓶颈处带宽的公平性[23]。

·**选择反馈拥塞指示(Selective Feedback Congestion Indication)算法**。该算法首先为经过该路由器的每个用户计算出它可以使用的公平资源份额 (fair share)，当某些用户的数据发送量超过了这个值时，便将发向这些用户的分组首部一个比特位置1，要求它们减少发送量，这个比特位称为拥塞指示位(Congestion Experience Bit)。而对于其他用户，仍然可以适当增加发送量。该算法基于这样的假设：网络拥塞是因某些用户的发送量超过了他们的公平资源份额而造成的，所以应该进行有选择的反馈，而不是盲目地向所有的用户反馈拥塞信息。但由于必须计算每个连接的流量，该算法同时也增加了路由器的复杂性与计算量。

以上的算法均无法区分不同的数据流，实际上它们仅可以约束那些遵守拥塞控制机制的数据源，这可能使行为不良的数据流抢占大量的带宽，它们可以绕开端到端的拥塞控制机制，向路由器任意发送自己的包，从而引起其他连接的包被丢弃。

·**公平队列 FQ(Fair Queuing)算法**。针对上面提到的问题 Nagle 在文[10]中提出一种公平队列算法，为每一个经过该网关的数据流维持一个独立的队列，使不遵守拥塞控制的流不影响其他流。当拥塞发生时，总是最长队列的包被丢弃，保证了资源的公平分配，但这对长时间连接传输大块数据的用户是不公平的，有时会造成资源浪费。文[24]中提出一种比特轮循算法(The Bit-Round Fair Queuing)，每次路由器从各个数据流中取一个比特发送，从而保证了输出链路上带宽的公平分配，但这又对数据流量大的用户不太公平，因为有时会不能及时处理而造成它的队列溢出而丢包，而且需要对每个数据流进行排队处理、状态统计，对数据包进行分类、调度等额外开销。1990年 Mckenney 提出随机公平排队算法 SFQ (Stochastic Fairness Queuing)，使用哈希函数随机地将到来的包放到一组固定的队列中[12]，该算法主要目的是提高路由器处理的速度与效率。

以上几种算法只是将资源平均分配，而没有考虑到不同的用户对资源的不同要求。文[31]中的加权公平队列 WFQ (Weighted Fair Queuing)算法对此做了一点改进，它为每个流分配一个权值，根据不同的数据流对带宽的不同要求，对每

个排队队列采用加权方法分配资源，从而增加了对不同应用的适应性。这类算法的缺点是实现起来比较复杂，而且忽视了端系统的控制能力。

4 进一步的工作

随着 Internet 的广泛应用，网络的可靠性、鲁棒性越来越依赖于 TCP/IP 拥塞控制，对拥塞控制的研究是非常具有价值的。由于拥塞控制的复杂性，使得现在的控制机制还存在着许多问题。第一、这些方案一般都有一些人为设定的假设条件，如假设分组丢失有99%的概率是由拥塞引起的。但实际上在某些网络上由于传输错误丢包的概率是很大的，如果对此也进行拥塞控制，无疑会造成吞吐量和资源利用率的浪费，降低网络性能。第二、目前的 TCP/IP 拥塞控制算法与实际应用有一定的差距，大部分的研究仅局限于特定的网络环境中，通过仿真实验来测试算法性能，缺乏理论基础。第三、公平性问题仍然没有完全解决，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，会导致对网络资源的不公平使用，TCP 数据流会主动减小发向网络的数据流，而 UDP 不会对此做出任何反应，这样它会得到越来越多的资源，继续增加流量，从而进一步加重网络拥塞。此外，TCP 连接之间也存在公平性问题，一些 TCP 连接在拥塞前使用了大窗口尺寸，或者它们的 RTT 较小，或者其数据包比其他 TCP 连接的大，会占用较大的带宽。

除了上面的基本问题外，我们认为还有以下几个方向值得进一步研究：

(1) 在以上几种拥塞控制策略中如何过滤瞬时突发流量的影响，做出可靠的、正确的判断，因为如果不能应付突发流量，就会导致源端的错误反应，减小吞吐量，影响网络的性能。

(2) 结合路由技术来控制拥塞，利用选路信息、路径交换报文等重新部署资源分配，绕开个别拥塞网关，如进行分流或平衡负载等，或者考虑使用网络管理协议 SNMP (Simple Network Management Protocol)以及 MIB(Management Information Base)对象来对路由器进行监测，预防拥塞的发生。

(3)由于基于窗口的拥塞控制机制的抖动幅度比较大，因而可以考虑通过保留 TCP 连接的历史状态信息或者采用基于速度的控制机制来保持稳定的发送速率。网络拥塞在很大程度上是由于数据流的突发性、并发性和不稳定性造成的，如果能使源端注入网络的数据流保持一种稳定的状态，在一定程度上就可以避免拥塞，但是如何自适应地估计速度是一个具有挑战性的问题。

参 考 文 献

1 Nagle J. Congestion Control in IP/TCP Internetworks. RFC 896, 1984

2 Tanenbaum A S. Computer Networks. The third edition. B Pretice Hall , 1996. 374～378

3 Jain R、Ramakrishnan K K. Congestion Avoidance in Computer Networks with a Connectionless Network Layer: Concepts, Goals and Methodology. In: Proc. IEEE Computer Networking Sysposium. 1988. 134～143

4 Yang C, Reddy A. A Taxonomy for Congestion Control Algorithms in Packet Switching Networks. IEEE Network Magazine, 1995, 9(1): 34～45

5 Jacobson V. Congestion avoidance and control. IEEE/ACM Transaction On Networking, 1998,6(3): 314～329

6 Stevens W. TCP Slow Start, Congestion Avoidance , Fast Retransmit ,and Fast Recovery Algorithms. RFC 2001 , 1997

7 Allman M, Paxson V, Stevens W. TCP Congestion control. RFC 2581,1999

8 Allman M, Floyd S, Partidge C. Increasing TCP's Initial Window. RFC 2414, 1998

9 Mathis M, Mahdavi J, Floyd S, Ramanow A. TCP Selective Acknowledgment Options . RFC 2018, 1996

10 Nagle J. On Packet Switches With Infinite Storage. RFC 970 , 1985

11 Mankin A. Random Drop Congestion Control. In: Proc. of SIGCOMM,1990

12 Mckenney P. Stochastic Fairness Queuing. In: Proc. of INFOCOM , 1990

13 Allman M, Paxson V. On Estimating End to End Network Path Properties. In: Proc. of SIGCOMM '99

14 Janey Hoe. Improving the Startup Behavior of a Congestion Control Scheme for TCP. In: ACM SIGCOMM'96

15 Lawrence S, Sean W, Larry L. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In: Proc. SIGCOMM'99

16 Suter B, et al. Design Considerations for Supporting TCP with Per-flow Queuing. In: Proc. of IEEE Infocom'98 Conf. 1998

17 Floyd S. Connections with Multiple Congested Gateways in Packet-Switched Networks, Part1: one-way Traffic. ACM Computer Communication Review, 1991,21(5)

18 Henderson H, Katz R. On Improving the Fairness of TCP Congestion Avoidance. In: Proc. of IEEE Globecom conf. 1998

19 Balakrishnan H, Padmanabhan V, Katz R. The Effects of Asymmetry on TCP Performance. In: Proc. of the ACM/IEEE Mobicom, Budapest, Hungary, ACM, 1997

20 Floyd S, Jacobson V. Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking , 1993

21 Braden B, et al. Recommendations on Queue Management and Congestion Avoidance In the Internet. RFC 2309, 1998

22 Ramakrishnan K, Floyd S. A Proposal to Add Explicit Congestion Notification (ECN) to IP. RFC 2481 , 1999

23 Hamann T, Walrand J. A New Fair Window Algorithm for ECN Capable TCP (New-ECN) . http://walrandpc. eecs. berkeley. edu/Pubs. html#confs, 2000

24 Demers A, Keshav S, Shenker S. Analysis and Simulation of a Fair Queuing Algorithm. In: Proc. of SIGCOMM'89

25 Floyd S. A Report on Some Recent Developments in TCP Congestion Control. IEEE Communications Magazine, April 2001

26 Clark D, Hoe J. Start-up Dynamics of TCP's congestion control and avoidance schemes. Presentation to Internet End-to-End Research Group:[Technical Report]. 1995

27 Visweswaraiah V, Heidemann J. Improving Restart of Idle TCP Connections. http://www. isi. edu/～ johnh/PAPERS/Visweswaraiah97b. html , 1997

28 Allman M, Balakrishnan H, Floyd S. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042,2001

29 Ramakrishnan K, Jain R. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. ACM Transactions on Computer Systems, 1990 , 8(2):158～181

30 Floyd S, Henderson T. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582 , 1999

31 http://www. ciso. com/warp/public/732/tech/wfq