基于 TCP 的流量控制和拥塞控制分析

Control of Flow and Congestion Based on TCP

杨华甫 倪子伟 （三峡大学职业技术学院 443000）

摘要：网络的信息流量协调与控制是 Internet 的 QoS 的重要内容，本文着重讨论了 TCP/IP 协议流量控制和拥塞控制的策略与算法，分析了在具体的网络传输中重传时间的确定，并对基于 TCP 的流量和拥塞控制策略进行了对比性探讨。

关键词：滑动窗口 慢开始 拥塞控制 快重传 往返时延

随着 Internet 技术的发展，越来越多的不同系统、不同速度的网络正在接入 Internet，网络堵塞也越来越严重。那么，如何调配不同网络中不同速度的系统间端到端的通信以匹配网络的运载容量，就成了 Internet 的 QoS 的重要内容。TCP 是 TCP/IP 体系中面向连接的运输层协议，它提供全双工的可靠交付的服务。据统计，Internet 上 95% 的数据流使用的是 TCP 协议，UDP 业务占据很小的份额，围绕着 TCP 流量控制的拥塞控制一直是 Internet 研究的一个热点。本文分析了滑动窗口技术在 TCP 协议中实现流量控制、拥塞控制的工作机制对 Internet 的 QoS 的保证。

1 滑动窗口的传输机制

为了提高报文段的传输效率，TCP 采用大小可变的滑动窗口进行流量控制。停止等待协议是最基本的数据链路层协议，其基本原理是发送端收到接受端的确认帧后在发送新的数据帧。停止等待协议比较简单，即每发送一个数据帧后都等待确认后再发送新的数据帧，由于等待占用了较长的时间，使整个通信信道的利用率不高。连续 ARQ 协议的特点是在发送完一个数据帧后，不是停下来等待确认帧，而是可以连续再发送若干个数据帧。如果这时收到了接受端发来的确认帧，那么还可以接着发送数据帧。由于减少了等待时间，整个通信的吞吐量就提高了。但其缺点是：只要有一帧出了差错，就可能有很多的数据帧需要重传，这必然要白白花费较多时间，因而增大开销，同时对每个数据帧的编号也要占用较多的比特数。

因此，为了提高报文段的传输效率，在连续 ARQ 协议中，应当将已经发送出去但未收到确认的数据帧的数目加以限制，这就是在网络传输中广泛采用的滑动窗口。TCP 采用大小可变的滑动窗口进行流量控制。窗口大小的单位是字节。在 TCP 报文段首部的窗口字段写入的数值就是当前给对方设置的发送窗口数值的上限。

如图所示：图 1(a) 要表示发送端要发送 900 字节长的数据，划分为 9 个 100 字节长的报文段，而发送窗口确定为 500 字节。发送端只要收到了对方的确认，发送窗口就可前移。每发送一个报文段，指针就向前移动一个报文的距离。当指针移动到发送窗口的最右端时就不能再发送报文段了。

图 1(b) 表示已发送了 400 字节的数据，但只收到对前 200 字节数据的确认，同时窗口大小保持不变。因此，现在发送端还可以发送 300 字节。

图 1(c) 表示发送端受到了对方对前 400 字节数据的确认，但对方将窗口减小到 400 字节。于是，发送端最多还可以发送 400 字节的数据。

发送窗口在连接建立由双方商定。但在通信的过程中，接收端可根据自己的资源情况，随时动态地调整对方的发送窗口上限值（可增大或减少）。

发送断利用发送窗口（这个窗口取决于对方的接受窗口）调节向网络注入分组的速率不仅仅是为了使接受端来得及接受，而且还是为了对网络进行拥塞控制。拥塞控制发生在通过网络传输的分组数量开始接近网络对分组的处理能力时。若网络中的分组数量超

过这个水平，网络的性能就会急剧恶化。

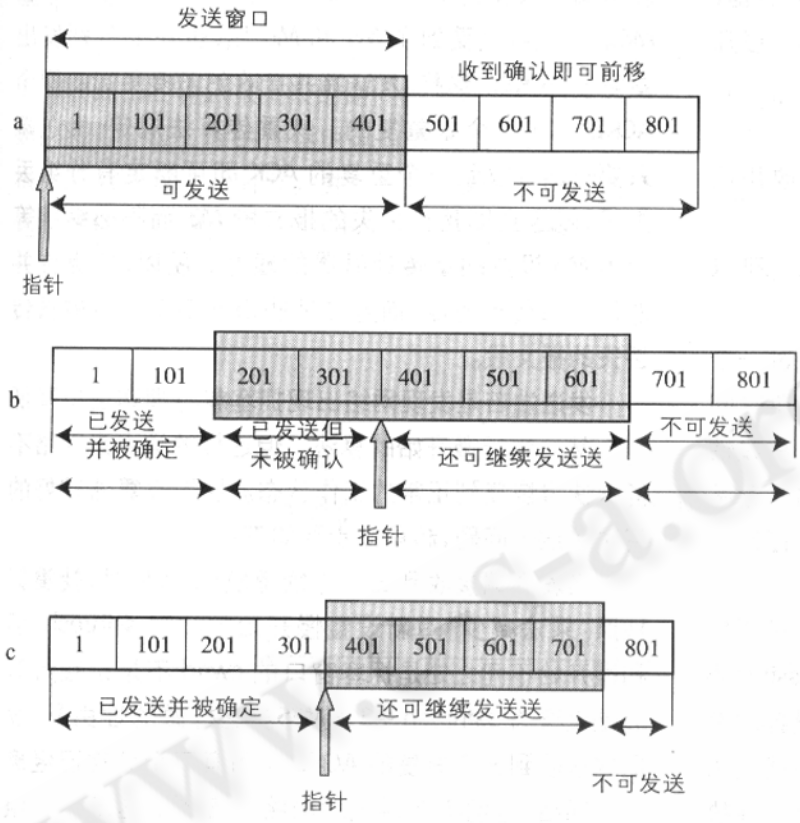

图 1　TCP 中的窗口概念

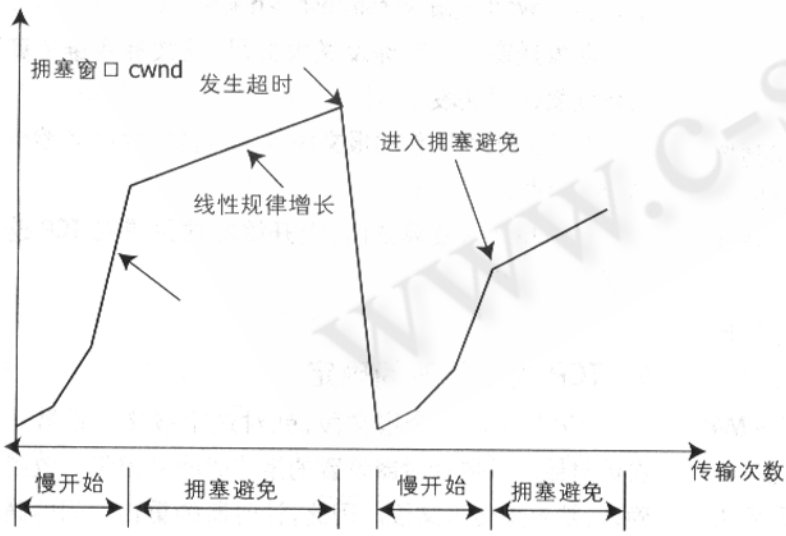

图 2　慢开始和拥塞避免算法

2　慢开始和拥塞控制（如图 2）

由于发送端的主机在确定发送报文段的速率时，既要根据接受端的接受能力，又要从全局考虑不要使网络发生拥塞。因此，对于每一个 TCP 连接，需要有以下两个状态变量：

（1）接收端窗口 rwnd（receiver window）。这是接收端根据其目前的接受缓存大小所许诺的最新的窗口值。是来自接受端的流量控制。接收端将此窗口值放在 TCP 报文的首部中的窗口字段，传送给发送端。

（2）拥塞窗口 cwnd（congestion window）。这是发送端根据自己估计的网络拥塞程度而设置的窗口值，是来自发送端的流量控制。

在发送时，发送端并不能确定拥塞窗口的大小，因此发送端确定拥塞窗口的原则是：只要网络没有出现拥塞，发送端就使拥塞窗口在增大一些。以便将更多的分组发送出去。但只要网络出现拥塞，发送端就使拥塞窗口减小一些，以减小注入到网络中的分组数。由于发生拥塞时，路由器要丢失分组。因此只要发送端没有按时接受到应当到达的报文 ACK，就可以认为网络出现了拥塞。

综上所述，发送端的发送窗口的上限应当取接收窗口和拥塞窗口这两个变量中较小的一个，即按以下公式确定：

发送窗口的上限值 = Min[rwnd, cwnd]

慢开始算法的原理是这样的。当主机开始发送数据时，如果立即将较大的发送窗口中的全部数据字节都注入到网络，那么由于这时还不清楚网络的状况，因而就有可能引起网络拥塞。但如果是由小到大逐渐增大发送端的拥塞窗口的数值，通常在刚刚开始发送报文段时先将拥塞窗口设置为一个最大报文段 MSS 的数值。即每受到一个对新的报文的确认后，将拥塞窗口增加至多一个 MSS 的数值。用这样的方法逐步增大发送端的拥塞窗口 cwnd，可以使分组注入

到网络的速率更加合理。

为了防止拥塞窗口 cwnd 的增长引起网络拥塞，还需要另一个状态变量，即慢开始门限 ssthresh。慢开始门限 ssthresh 的用法如下：

当 cwnd < ssthresh 时，使用上述慢开始算法。

当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。

当 cwnd = ssthresh 时，既可以使用慢开始算法，也可以使用拥塞避免算法。

拥塞避免算法使发送端的拥塞 cwnd 每经过一个往返时延 RTT 就增加一个 MSS 大小（而不管在时间 RTT 内收到了几个 ACK）。这样拥塞窗口 cwnd 按线性规律缓慢增长，比慢开始算法的拥塞窗口的增长速率缓慢得多，由于拥塞避免将拥塞窗口控制窗口控制为按线性增长，使网络比较不容易出现拥塞。

无论在慢开始阶段还是在拥塞避免阶段，只要发送端发现网络出现拥塞，就要将慢开始门限 ssthresh 设置为出现拥塞时发送窗口值的一半。这样设置的考虑就是：既然出现了网络拥塞，那就要减小向网络注入的分组数。然后将拥塞窗口 cwnd 重新设置为 1，并执行慢开始算法。

由于迅速减小主机发送到网络中的分组数，使得发生拥塞的路由器有足够的时间把队列积压的分组处理完毕。

3　快重传和快恢复

虽然慢开始和拥塞避免的算法可以对拥塞起到控制作用，但有时一条 TCP 连接会因等待重传计时器的超时而空闲较长的时间。因此又增加了两个新的拥塞控制算法。即快重传和快恢复。

快重传的算法是：当接收端每收到一个报文段后都要立即发出确认 ACK 而不是等待自己发送数据时才将 ACK 捎带上，假定发送端发送了报文段 M1 ~ M4 共 4 个报文段，当接收端受到了 M1 和 M2 后，就发出确认的 ACK1 和 ACK2。假定由于网络拥塞使 M3 丢失了。接收端后来收到一个 M4，发现其序号不对，但仍收下放在缓存中，同时发出确认，不过发出的是重复的 ACK2（不能发送 ACK4，因为 ACK4 表示 M3 和 M4 都已经收到了）。这样发送端知道现在可能是网络出现了拥塞造成分组丢失。但也可能是报文段 M3 尚滞留在网络的某处，还要经过较长的时延才能达到接受端。发送端接着发送 M5 和 M6。接收端接受到了 M5 和 M6。接收端接受到了 M5 和 M6 后，也还要分别发出重复的 ACK2。这样，发送端共接收到了接受端的四个 ACK2，其中三个是重复的。快重传算法规定，发送端只要一连接收到三个重复的 ACK 即可断定有分组丢失了，就应立即重传丢失的报文段 M3 而不必继续等待为 M3 设置的重传计时器的超时。所以，快重传并非取消重传计时器，而是在某些情况下个更早的重传丢失的报文段。

发送端若是发现网络出现拥塞就将拥塞窗口降低为 1，然后执行慢开始的算法。但这样的缺点是网络不能很快得恢复到正常的工作状态。快恢复算法较好的解决了这一问题，起具体步骤如下：

当发送端接收到三个连续重复的 ACK 时，就重新按照“乘法减少”重新设置慢开始的门限 ssthresh；但和慢开始不同之处是拥塞窗口的 cwnd 不是设置为 1，而是设置为 ssthresh + 3 * MSS。这样做的理由是：发送端接收到三个重复的 ACK2 表明有三个分组已经离开了网络，他们不会再消耗网络的资源。这三个分组是停留在接受端的缓存总。可见现在网络中并不是堆积了分组而是减少了三个分组。因此，将拥塞窗口扩大些并不会加剧网络的拥塞。若收到的重复 ACK 为 N 个，则将 cwnd 设置为 ssthresh + n * MSS。

若发送窗口还容许发送报文段，就按拥塞避免算法继续发送报文段。

若收到了确认新的报文段的 ACK，就将 cwnd 缩小到 ssrhresh。

在采用快恢复算法时，慢开始算法只是在 TCP 连接建立时使用。

4　TCP 的重传时间确定

TCP 每发送一个报文段，就对这个报文段设置一次计时器。只要计时器设置的重传时间到但没有收到确认，就要重传报文段。因此，计时器的重传时间的确定就非常重要了。

由于 TCP 的下层是一个互连网环境，其中可能有高速局域网，也可能经过多个低速广域网，并且 IP 数据报所选择的路由还可能发生变化，时间设置过短，则许多报文的重传时间是太早了，给网络增加许多不必

要的负担，但若超时的时间太长，又会使网络的传输效率降低很多。

为此，TCP 采用了一种自适应算法。发送报文时间和收到相应确认报文段的时间差为报文段的往返时延。将各个报文段的往返时延样本加权平均，就得到报文段的平均往返时延。并且，每测量到一个新的往返时延样本，就重新计算一次平均往返时延 RTT：

平均往返时延 RTT = α *（旧的 RTT）+（1—α）* *（新的往返时延样本）

文重传了，就不采用其往返时延样本。这样得出的平均往返时延 RTT 和重传时间就较准确，同时，避免往返时延的突然增大而造成的重传时间无法更新，对 Karn 算法进行了修正。即报文重传一次，就将重传的时间增大一些。当不在发生报文段的重传时，才根据报文段的往返时延 RTT 更新平均往返时延和重传时间的数值。（如图 3 所示）

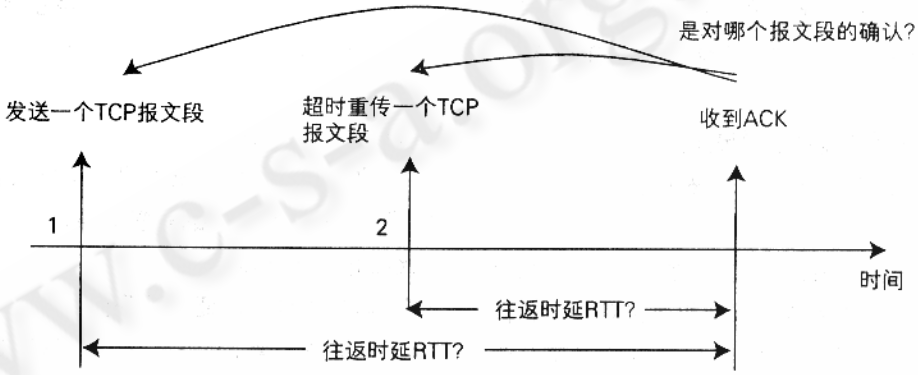

图 3 确认报文段 ACK 无法判断是对哪一个报文段的确认

在上试中，α 若接近于 1，表示新算出的平均往返时延 RTT 和原来的值相比变化不大，RTT 更新较慢。若 α 接近于零，则表示加权计算的平均往返时延 RTT 受新的往返时延的影响较大 RTT 更新较快。典型的 α 值为 7/8。

为此，计时器设置的超时重传时间应略大于上面得出的平均往返时延 RTT，即：

RTO = β * RTT

β 显然是一个大于 1 的系数。β 很接近于 1，发送端可以及时地重传丢失的报文段，因此效率得到提高。但若报文段并未丢失而仅仅增加了一点时延，那么过早地重传未收到确认的报文段，反而会增加网络的负担。因此 TCP 的 β 值一般取值为 2。

由于重传时间到，重发的报文后收到的确认报文段 ACK，无法判断是对报文段 1 的确认还是对重传报文段 2 的确认，就会造成往返时延和重传时间难以确认，如图所示。因此，Karn 提出了一个算法：即只要报

5 结束语

总之，通过运输层的流量控制、拥塞控制，协调发送端和接受端的工作速度，避免发送端发送速度过快，而接收端来不及处理而丢失数据。同时，在网络层的随机早期丢弃 RED（Random Early Discard）策略，保证了 TCP/IP 在网络的传输过程中的提供可靠的交付服务和和较高的网络吞吐量。

参考文献

1 enneth d. Reed 著，孙坦译，协议分析，电子工业出版社，2004 年 4 月。
2 德森、戴维、叶新铭、贾波著，计算机网络，机械工业出版社，2001 年 6 月。
3 谢希仁，计算机网络，电子工业出版社，2003 年 6 月。
4 YouLu Zheng Shakil Akhtar 著，彭旭东译，计算机网络，清华大学出版社，2004 年 5 月。