TCP/IP 拥塞控制算法的改进

Improvement in TCP/IP Congestion Control Algorithm

邓亚平　叶凌伟　陈　雁
（重庆邮电学院计算机系　重庆400065）

Abstract　Congestion control is one of the important things of the research of TCP/IP. With the worldwide flooding of the Internet, congestion control algorithms for TCP/IP become more and more important. After the introduction to the basic concept of network congestion and the discussion of standardized TCP congestion control algorithms, we propose an improvement of slow start algorithm——Larger Initial Window, and extensively investigate the advantages and disadvantages of the improvement. In this paper we also present some network simulations and experiments to specify that the improvement can improve network performance in many cases at last.

Keywords　Congestion control, Slow start, Larger initial window

一、引言

在 Internet 上，随着信息传送量的逐渐增大和网络组成的日益复杂，由于网络负载超过网络的处理能力，网络发生拥塞的可能性也越来越大。如果不对网络拥塞进行有效的控制和在网络发生拥塞时使网络能及时恢复到正常状态，就会造成严重的网络拥塞，甚至导致网络崩溃。因此，网络拥塞的避免和控制成为越来越重要和亟待解决的问题。

拥塞控制是 TCP/IP 协议研究的重要内容。在 Internet 中，拥塞控制的大部分工作是由 TCP 完成的。目前，标准的 TCP 协议实现都包含了一些避免和控制网络拥塞的算法[1]。但是，现有的拥塞控制算法都有一些局限性。因此，对 TCP/IP 拥塞控制算法进行进一步的研究具有重要的理论和应用价值。

二、TCP/IP 拥塞控制技术

Internet 拥塞控制的大部分工作是由传输层来完成的，因为解决网络拥塞的真正办法就是降低数据发送率[2]。目前，在各种 TCP 协议实现中一般都包含有四个相互关联的拥塞控制算法[3]：slow start（缓慢启动），congestion avoidance（拥塞避免），fast retransmit（快速重传），fast recovery（快速恢复）[3]。

2.1　Slow Start 算法

Slow Start 在发送方 TCP 中增加了一个拥塞窗口，称为 cwnd。当主机同网络中的另一台主机建立连接后，cwnd 的值被初始化为一个分组（分组的大小由另一端说明，或者使用缺省值，典型为536或512）。发送方每接收到一个 ACK，cwnd 的值就增加1。发送方可以发送的最大窗口为拥塞窗口（cwnd）和接收方声明窗口（rwnd）之间的最小值。

一开始，发送方发送1个分组，然后等待该分组的 ACK。当接收到相应的 ACK 后，拥塞窗口的值就从1增加至2，此时，发送方一次就能发送2个分组。当这两个分组的 ACK 都到达后，拥塞窗口的值就增加至4。这种方式使发送方可发送的信息量呈指数增长的趋势，尽管它并不是真正的指数增长，因为接收方可能会延迟它的 ACK，特别当接收方采用每收到两个分组才发送一个 ACK 的确认机制时。

当拥塞窗口增加到某一个值时，发送方一次所发送的信息量就达到或超过了网络的容量，而且中继路由器也开始丢包。这种现象就告诉发送方它的拥塞窗口值已经太大了。

2.2　Congestion Avoidance 算法

算法假设由于损坏而丢失包的比例是非常小的（远远小于1%）。因此，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：超时和接收到重复 ACK。

在拥塞避免中，在每个 RTT 时间内，若发送方收到对当前 cwnd 对应数据的应答后，cwnd 比原来增加

邓亚平　教授，主要研究方向：计算机网络与通信，分布式计算机系统。叶凌伟　硕士生，研究方向为计算机网络与通信。陈　雁　硕士生，研究方向为计算机网络与通信。

一个最大 TCP 分组所对应的字节数，即 cwnd 线性增长。

拥塞避免和缓慢启动是两个独立的算法，两者的目的和作用范围是不一样的。但是，当网络发生拥塞时，TCP 必须减慢数据包进入网络的速率，然后激发缓慢启动算法使发送过程继续进行。实际上，这两种算法是一起实现的。要实现这两个算法，每个 TCP 连接要保持3个状态变量：拥塞窗口：cwnd；声明窗口：rwnd；缓慢启动临界值：ssthresh。

在缓慢启动阶段，cwnd 的值呈指数增长，直至 cwnd＞ssthresh（或 cwnd＝ssthresh）或拥塞发生时，缓慢启动阶段就结束，在拥塞避免阶段，cwnd 的值呈线性增长，直至 TCP 检测到拥塞，拥塞避免阶段才终止。

2.3 Fast Retransmit（快速重传）

无论缓慢启动或拥塞避免，cwnd 总是增长，这样负载就逐渐增大并最终导致网络拥塞。一旦数据分组丢失，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。所幸的是因数据丢失，接收方将返回对相同数据的重复应答。发送方获得多个重复应答就能获悉应答之后的下一个分组已经丢失，不必等待定时超时，从而提前进行数据重发。

发送方在收到重复 ACK 时并不能确定是丢失了分组还是发生了分组失序。通常假定，发生分组失序时，在接收方解决此问题之前，发送方只可能收到一个或者二个重复 ACK。如果发送方连续收到三个以上的重复 ACK，就很可能表明一个分组丢失了。这时发送方不等到重传定时器超时就重新传送那个可能丢失的分组，这就叫做快速重传算法。TCP 发送方就是利用快速重传算法来检测和恢复分组丢失的。

2.4 Fast Recovery（快速恢复）

在快速重传可能丢失的分组之后，TCP 就执行拥塞避免算法，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 ACK 为止。实际上，接收到重复 ACK 只是告诉 TCP 仅仅只有一个包丢失了，既然接收方只有在收到另一个分组时才会发回重复的 ACK，那么就表明该分组已经离开网络并已进入接收方的缓冲区，也就是说，此时发送方和接收方之间仍然存在着数据流，所以发送方不必采用缓慢启动算法而急剧降低数据流。这样做可使大窗口下中度拥塞发生时网络仍然具有较高的吞吐量。

同样，快速重传算法和快速恢复算法通常也一起实现。

三、对 Slow Start 算法的改进

在标准的 Slow Start 算法中，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 rwnd 的值所需要的时间为公式(1)所示：

$$\text{slow start time} = \text{RTT} \log_2 \text{rwnd} \quad (1)$$

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 rwnd 的值所需要的时间就增加至大约两倍，如公式(2)所示：

$$\text{slow start time} = 2R \log_2 \text{rwnd} \quad (2)$$

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中继路由器的资源损耗，并可提高网络吞吐量。但是，延迟确认机制在 cwnd 的初始值为1时会产生性能问题。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延迟 ACK 定时器超时才发送 ACK。也就是说，当发送方 cwnd 的初始值为1时，发送方在发送一个分组后不得不等待直到延迟 ACK 定时器超时。

因此，标准 Slow Start 算法并不能总是保证网络获得较高的性能，这种机制在下列几种情况下显得效率不高：

1）当接收方采用延迟 ACK 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量并不大时（2～4分组）时，也必须经过2到3个 RTT 时间才能完成数据发送。

3）在高带宽大延迟的网络（如卫星通信网）中，发送方在发送一个分组后，需要等待较长的时间（因为 RTT 较大）才能发送第二个分组。

可以看出，如果 Slow Start 算法的初始窗口值略大一些，上述3种情况可能就不会发生。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。

3.1 算法改进——使用大初始窗口

显然，要解决上述关于初始窗口值为1所带来的一些问题，必须增加 Slow Start 算法的初始窗口值。根据实际情况不同，初始窗口值的增加也不同。算法改进建议如下：

·初始窗口可以为2个分组

·如果分组大小至多为1460字节，初始窗口可以为3个分组

·如果分组大小至多为1095字节，初始窗口可以为4个分组

把 Slow Start 算法的初始窗口值（称为 IW）由一

个分组增至2～4个分组。在多数情况下这个变化会导致IW的值大约为4K字节，IW的值可由公式(3)精确定义如下：

$$IW = \min(4 * MSS, \max(2 * MSS, 4380\ bytes)) \quad (3)$$

也就是说，IW的上限是由MSS(maximum segment size)决定的，如下所示：

```
If (MSS <= 1095 bytes)
  then IW <= 4 * MSS;
If (1095 bytes<MSS<2190 bytes)
  then IW <= 4380;
If (2190 bytes<=MSS)
  then IW <= 2 * MSS
```

实际上，TCP在以下三种情况下使用Slow Start算法：1)开始一个新的连接(初始窗口)；2)在经过一段较长空闲期后重新启动传输(重启窗口)；3)传输超时后的重传(丢失窗口)。

本文所提议的改变Slow Start初始窗口值主要是针对第一种情况。此外，TCP也可把"restart window"的值设为min(initial window,cwnd)，因为这样并不会增加cwnd的值。但是，这些变化不能改变丢失窗口(第三种情况)的值，丢失窗口的值必须被设为一个分组。

对于使用大初始窗口，一个附加的限制是初始窗口的值最多为初始分组大小的4倍，同时接收方声明窗口对发送方发送窗口上限的限制必须保留(TCP流量控制所必需的)。因此，TCP连接开始可以发送3个1460字节的分组或4个512字节的分组，而且这个增加的初始窗口值是可选的。

3.2 大初始窗口的优点

1)当初始窗口为一个分组时，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分组时，那么接收方在第二个分组到达时就可生成一个ACK。这就减少了等待超时的时间(一般情况下可达到200 msec)。

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

3)对那些可能采用大拥塞窗口的连接而言，在数据传送的缓慢启动阶段，大初始窗口减少3个RTT和一个延迟ACK时间。在高带宽大广播延迟链路中，如卫星链路，这个变化将给TCP连接带来较大的益处。

此外，大初始窗口并不缺乏公平性。这意味着同未使用大初始窗口的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字节的分组的TCP连接和一个初始窗口为3个1460字节的分组的TCP连接在任何情况下都得到相同的吞吐量。

3.3 大初始窗口弊端

同时，在Slow Satrt算法中使用大初始窗口也会带来一些负面影响。在高度拥塞的网络环境中，大初始窗口会增加一个TCP连接中丢失分组的数量。这个附加的丢失可能会降低网络的性能。在多个TCP连接竞争共享的高度拥塞的瓶颈时，使用大初始窗口也会导致附加的分组丢失。因此，在某些环境下，使用大初始窗口可能会降低网络性能，而且会产生不公平的竞争通信量。

3.3.1 大初始窗口对单个连接产生的弊端

在高度拥塞的网络环境中，特别是网络中有些路由器不能很好地处理突发通信量(如路由器采用典型的drop tail router queues)时，一个TCP连接采用一个分组的IW有时会更好一些。有可能会发生这样一种现象：一个TCP连接使用一个分组的IW开始进行传送不会产生包丢失，而如果该TCP连接使用四个分组的IW可能会由于路由器不能处理少量的突发通信量而导致不必要的数据重传。这可能会导致一个不必要的传输超时，即使该TCP连接可以在传输超时前就恢复，也会使TCP过早从缓慢启动阶段进入拥塞避免阶段。而在具有足够缓存的非拥塞网络中、或在路由器使用"active queue management"(如随机早期检测)的中等拥塞的网络环境中，这些过早的分组丢失现象就不会发生。

在某些TCP连接中，即使大初始窗口的突发通信量导致过早的分组丢失，仍能获得更高的性能。这在以下两种情况中是完全可以达到的：1)TCP连接不等到传输超时就可从分组丢失中恢复过来；2)由于网络拥塞或接收方rwnd限制，TCP连接基本上被限制在一个小的拥塞窗口。

3.1.2 大初始窗口对网络产生的弊端

在拥塞崩溃的潜在性方面，我们要考虑大初始窗口的两个单独的对网络的潜在危险。第一，聚集在拥塞链路上的大量分组是已经被接收方接收的却还留在网络中的重复分组。第二，聚集在拥塞链路上的大量分组是那些在到达目的地前会被网络丢弃的分组。从对网络中其他连接产生的负面效果来看，大初始窗口的一个潜在不利因素是它增加了网络的包丢失率。下面我们来详细分析这三个潜在危险：

·重复分组。前面说过，大初始窗口有时会引起分组丢失，而这些分组当TCP发送方采用初始窗口为一个分组的slow start算法时不会丢失。在目前的网络环境中，并没有明确的可标识网络发生拥塞的标志，所

有 TCP 实现都使用分组丢失作为网络发生拥塞的标志。但是，实际上大初始窗口并不会导致发送方重传大量的已被接收方接收的重复分组。

如果初始窗口中有一个分组丢失了，TCP 有3种方法进行恢复：1)从一个分组的窗口缓慢启动，和重传超时后的恢复一样，或者和 Taboe TCP 中快速重传一样；2)快速恢复(不使用 SACK)，和 Reno TCP 中收到3个重复 ACK 后的处理一样；3)快速恢复(使用 SACK)，只对发送方和接收方都支持 SACK 选项的 TCP 有效。在上述3种情况中，如果初始窗口中有一个分组丢失了，不会重传重复的分组。

如果初始窗口中有多个分组丢失呢？TCP 可使用第一种恢复方法，从一个分组的窗口缓慢启动，不必要的重传分组数目是有限的。如果使用第二种恢复方法，通常会产生一个重传超时，而且不必要的重传分组数目是很小的。如果使用第三种恢复方法，只有当网络出现严重的分组失序情况才会有不必要的重传分组，因此，在任何情况下，由大初始窗口引起的不必要的重传分组数目是很小的。

结论：在不使用 SACK 的前提下，初始窗口中的多个分组丢失有时会导致一个重复分组的重传。在任何情况下，均不会有重传2个重复分组的现象发生。因此，我们的结论是，由大初始窗口导致的重传重复分组数目是很小的。

·丢弃分组。到底 TCP 采用大初始窗口会增加多少聚集在拥塞链路上的在到达目的地前会被网络丢弃的分组呢？这个问题只可能发生在那些有多条拥塞链路的连接中，一些分组可能会在源端到目的端的路径上的第一条拥塞链路使用稀有的带宽，这些分组在以后沿着该路径的传送过程中被丢弃。首先，大多数 TCP 连接在源端到目的端的路径上只有一条拥塞链路。这些连接上的分组丢失并不会"浪费"稀有带宽，也不会引起拥塞崩溃。然而，一些网络路径上会有多条拥塞链路，这样，初始窗口中的丢失分组将使用路径上前面拥塞链路上的稀有带宽并在后继的拥塞链路上被丢弃。

对一个分组丢失率较高的网络来说，增加 TCP 的初始窗口将进一步增加包丢失率。这部分是由于那些采用 Drop Tail 队列管理的路由器在网络发生拥塞时对突发通信量的处理有较大的困难。但是，如果假设多个 TCP 连接的到达没有关联，大 TCP 初始窗口并不会显著地增加分组丢失率。

四、仿真和实验结果

1. *研究使用大初始窗口的 TCP 连接* 这个部分主要是观察大初始窗口对 TCP 连接的影响。第1组实验检验在卫星链路上的性能。实验表明，大初始窗口可以改进在卫星信道上的 TCP 连接的性能[4]。在这个研究中，初始窗口值为4个分组(MSS 为512字节)，结果是吞吐量增加了30%(取决于传输信息量大小)。第2组实验检验在拨号 MODEM 链路上的性能[5,6]。在一个 28.8b/s 的拨号信道中使用4个分组的初始窗口，可使一个16kB 文件的传送时间减少大约10%，而且包丢失率没有增加。

在低速的 tail 电路中，(如拨号 MODEM 链路)，而且路由器的缓存容量较小时，大初始窗口对 TCP 性能的影响值得特别关注。一个仿真实验[7]研究在一个连有低速 MODEM 链路的主机和一个只有3个包缓存容量的路由器中使用大初始窗口对 TCP 性能的影响。研究得出结论，在这种情况下，使用大初始窗口不会损害 TCP 的性能。

2. *研究使用大初始窗口的网络* 这个部分通过一些仿真和实验研究大初始窗口对共享路径的其他 TCP 连接的影响[5,6]。实验表明，对100个 Internet 主机进行的16kB 文件传送，4个分组的初始窗口会使包丢失率有少量的增加，大约为0.04分组/传送。在包丢失率有轻微增加的同时，使用4个分组(MSS 为512字节)的初始窗口同使用一个分组的初始窗口相比传送时间却大约减少了25%。

另一个研究在一个非常拥塞的网络中使用大初始窗口传送20k 大小的信息量。当网络中所有 TCP 连接都使用4个分组的初始窗口，其包丢失率比都使用1个分组初始窗口的情况增加1-2%。在一个分组初始窗口的网络的包丢失率在1%至11%之间，这个结论都成立。这些实验表明，在一个高度拥塞的网络环境中，如果每个连接共享的瓶颈带宽仅接近一个分组大小，使用大初始窗口会导致包丢失率和重传超时的显著增加。

参考文献

1 Jacobson V. Congestion Avoidance and Control. Computer Communication Review, 1998, 18(4): 314~329

2 Tanenbaum A S. Computer Networks. Prentice Hall, Inc., 1996

3 Stevens W, Allman M, Paxson V. TCP Congestion Control. RFC 2581, April 1999

4 Allman M. Improving TCP Performance Over Satellite Channels: [Master's thesis]. Ohio University, June 1997

5 Allman M. An Evaluation of TCP with Larger Initial Windows. 40th IETF Meeting--TCP Implementations WG. Dec. 1997

6 Allman M, Hayes C, Ostermann S. An Evaluation of TCP with Larger Initial Windows. March 1998

7 Shepard T, Partridge C. When TCP Starts Up With Four Packets Into Only Three Buffers. RFC 2416, Sep. 1998