TCP拥塞控制

1.拥塞

在计算机网络中的链路容量（即带宽）、交换节点（如路由器）中的缓存和处理机等，都是网络的资源。在某段时间内，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏，从而导致吞吐量将随着输入负荷增大而降低。这种情况就叫做拥塞。通俗来说，就跟交通拥堵性质一样。

网络拥塞的原因有很多，如**交换节点的缓存容量太小、输出链路的容量和处理机的速度。**

> **拥塞常常趋于恶化。**如果一个路由器没有足够的缓存空间，它就会丢失一些新到的分组，但当分组被丢失时，发送这一分组的源点就会重传这一分组，甚至重传多次，这样会引起更多的分组流入网络和被网络中的路由器丢失。可见拥塞引起的重传并不会缓解网络的拥塞，返回会加剧网络的拥塞。

2.拥塞控制

拥塞控制就是防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致于过载。拥塞控制是一个**全局性的过程。**涉及网络中所有的主机、所有的路由器，以及与降低网络传输性能有关的所有因素。

拥塞控制和流量控制的关系密切，但是流量控制往往是指点对点的通信量控制，是个端对端的问题。流量控制所要做的就是抑制发送方发送数据的速率，以便使接收端来得及接收。

> 如上图所示，对于左侧图，如果多台主机通过链路上的路由器向接收方发送数据，如果主机的发送数据过快，就可能导致网络拥塞，发送方的数据迟迟到不了接收方，那么接收端就会察觉到网络发生拥塞，但是接收端不清楚是哪台或哪些主机发送速率过快导致的拥塞。而对于右图，由于是端对端的通信，如果接收方发现来不及接受数据，它会知道是与它通信的主机发送速率过快导致的。

3.拥塞控制算法

TCP进行拥塞控制的算法有四种，即**慢开始（slow-start）、拥塞避免（congestion-avoidance）、快重传（fast retransmit）、快恢复（fast recovery）。**

为了讨论问题方便，提出以下假定：

> (1) 数据是单方向传送的，对方只传送确认报文。
> (2) 接收方总有足够大的缓存空间，因而发送窗口的大小由网络的拥塞程度来决定。

3.1 慢开始和拥塞避免

拥塞控制也叫做**基于窗口**的拥塞控制。为此，发送方维持一个叫作**拥塞窗口**cwnd（congestion window）的状态变量。**拥塞窗口的大小取决于网络的用谁程度，并且动态的变化。发送方让自己的发送窗口等于拥塞窗口。**

> 在流量控制一文中还提到一个接受窗口值rwnd，从接收方流量控制角度考虑，**发送方窗口一定不能超过对方给出的接收方窗口值rwnd。**

> 本节讨论拥塞控制假定了接收方有足够大的缓存空间，所以无需考虑rwnd。但是在实际中，两者都需要考虑，并且发送方窗口的上限值应当取接收方窗口值rwnd和拥塞窗口cwnd这两个变量中较小的值。即发送方窗口上限值 = min { rwnd , cwnd }。

接收方窗口值rwnd和拥塞窗口值cwnd的区别：

> **接收窗口值**：接收方根据缓存设置的值，并告知给发送方，**反映接收方容量。**
> **拥塞窗口值**：发送方根据自己估算的网络拥塞程度而设置的窗口值，**反映网络当前容量。**

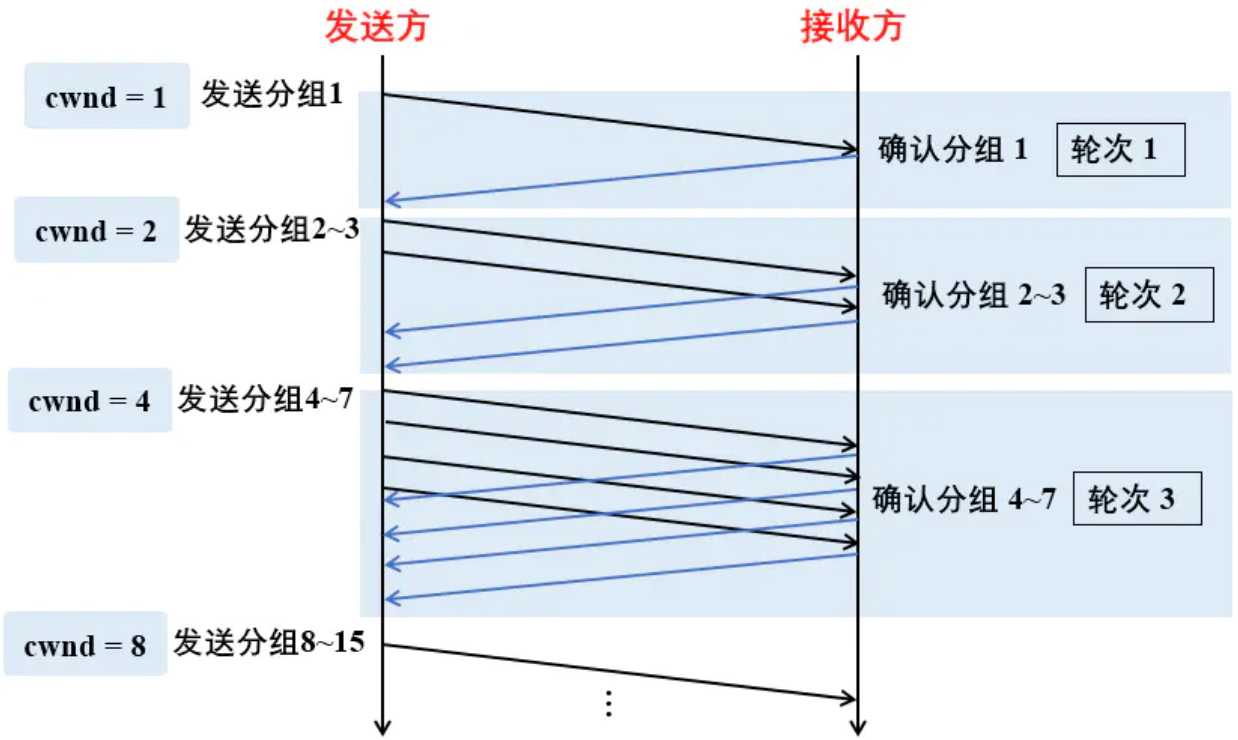
发送方控制拥塞窗口的原则是：只要网络没有出现拥塞，拥塞窗口就可以再扩大一些，以便让更多的分组发送出去，如果网络出现了拥塞，就必须将拥塞窗口减小一些，以减少分组的发送。**判断网络拥塞的依据就是出现了超时。**

3.1.1 慢开始算法

慢开始算法的思路：刚开始发送数据时，不一下向网络中注入大量数据，而是先探测一下，即**由小到大逐渐增大发送窗口**，也就是说，**由小到大逐渐增大拥塞窗口数值。**

慢开始算法具体规定：刚开始发送数据时，先把拥塞窗口cwnd根据**发送方的最大报文段SMSS**（Sender Maximum Segment Size）数值的大小设置为不超过2-4个SMSS的数值。在**每收到一个对新的报文段的确认后，可以把拥塞窗口增加最多一个SMSS的数值。**用这样的方法逐步增大发送方的拥塞窗口rwnd，可以使分组注入到网络中的速率更加合理。

下面举例说明一下，虽然实际上TCP是用字节数作为窗口大小的单位，但为了方便描述，下面使用报文段的个数来作为窗口的大小的单位，并且假设所有的报文段大小相等。



> 刚开始时发送方先将窗口大小cwnd设置为1（这里的1是指的是一个报文段的字节数，下同），发送第一个分组，接收方接收后确认分组0。发送方接收到对分组0的确认后，将cwnd增加到2（即收到一个确认就把拥塞窗口增加一个报文段长度）。

> 发送方接着发送分组2 ~ 3，接收方接收后确认分组2 ~ 3，发送方收到2个确认后将cwnd从2增加到4。

> 发送方接着发送分组4 ~ 7，接收方接收后确认分组4 ~ 7，发送方收到4个确认后将cwnd从4增加到8。 ....

所以，**慢开始算法每经过一个传输轮次（transmission round），拥塞窗口cwnd就加倍。**

> **一个传输轮次**：发送方从发送一批分组到接收到它们的确认所经历的时间，即往返时间RTT（RTT并非是恒定的数值）。

注：在TCP实际运行时，发送方只有收到一个确认就可以将cwnd加1并发送新的分组，并不需要等一个轮次所有的确认都收到后再发送新的分组。

从上面可以看出，慢开始算法虽然起始的窗口很小，但是每过一个轮次，窗口大小翻倍，呈指数爆炸增长，所以必须要对其进行一个限制，防止其增长过大引起网络拥塞。这个限制就是**慢开始门限ssthresh**状态变量。慢开始门限ssthresh的用法如下：

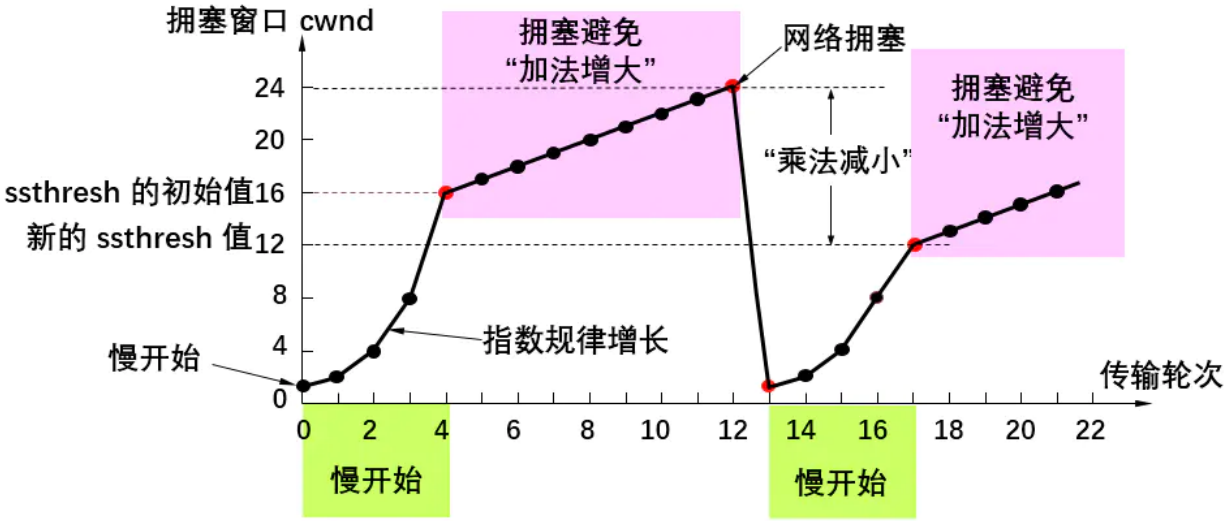
> 当cwnd < ssthresh时，使用上述慢开始算法。
> 当cwnd > ssthresh时，停止使用慢开始算法而改用拥塞避免算法。

3.1.2 拥塞避免算法

拥塞避免算法的思路是让拥塞窗口cwnd缓慢增大，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是像慢开始阶段那样加倍增长。因此在拥塞避免阶段就有“**加法增大**”AI（Additive Increase）的特点。这表明在拥塞避免阶段，拥塞窗口**cwnd按线性规律增长**，比慢开始算法的拥塞窗口增长速率缓慢得多。

下面用一个具体的例子来说明拥塞控制的过程，下图假设TCP发送窗口等于拥塞窗口，慢开始初始门限设置为16个报文段，即ssthresh = 16。

> 注：**横坐标是传输轮次。**



> (1) 在执行慢开始算法时，发送方每收到一个对新报文段确认的ACK，就把拥塞窗口的值加1，然后开始下一轮的传输。刚开始cwnd呈指数规律增长，当cwnd = 16时（即第一个红色的点

> 处），拥塞窗口到达慢开始门限。

> (2) 当拥塞窗口达到慢开始门限时，即进入避免拥塞阶段，**拥塞窗口按线性规律增长，使网络比较不容易出现拥塞。**

> (3) 当拥塞窗口 cwnd = 24时，网络出现超时（第二个红点处），发送方判断为网络拥塞。于是调整门限值 ssthresh = cwnd / 2 = 12（即发送窗口的一半），同时设置拥塞窗口cwnd = 1，进入慢开始阶段。

> (4) 重新进入慢开始阶段后，当拥塞窗口再次达到新的门限ssthresh = 12时（第四个红点处），改为执行拥塞避免算法，拥塞窗口按线性规律增长，每经过一个往返时延就增加一个MSS的大小。

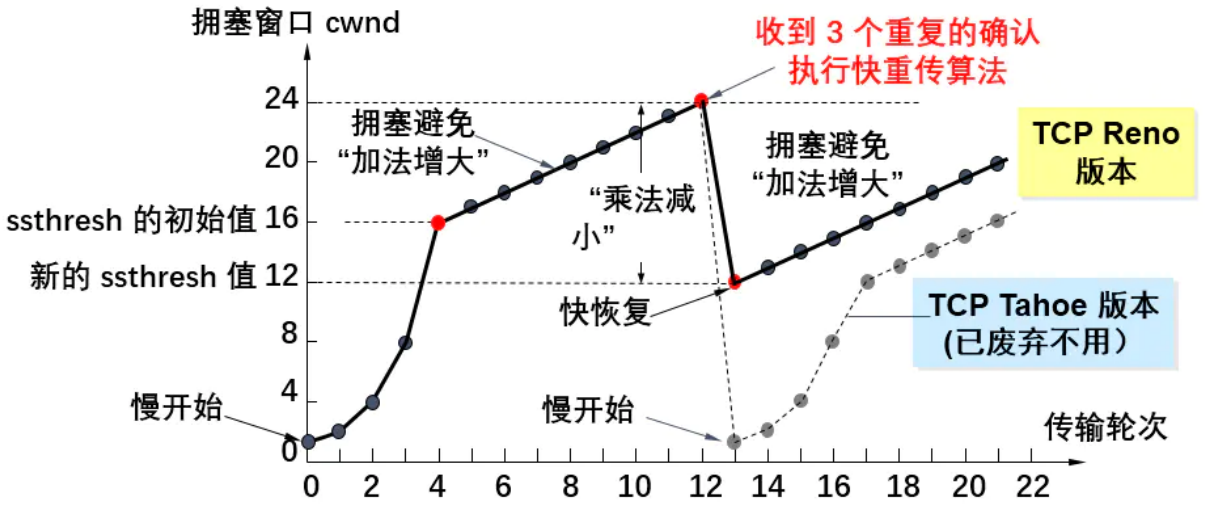
在拥塞避免阶段，拥塞窗口是按照线性规律增大的，这常称为**加法增大AI。**无论在慢开始阶段还是拥塞避免阶段，只要出现一次超时（即出现一次网络拥塞），就把慢开始门限值 ssthresh 设置为当前拥塞窗口的一半，这叫做**乘法减小 MD（Multiplication Decrease）。**

当网络频繁出现拥塞时，ssthresh 值就下降的很快，以大大减少注入网络中的分组数。

3.2 快速重传和快恢复

快速重传机制之前介绍过，这里简单概括下，即发送方如果收到连续3个冗余ACK，那么发送方就会执行快重传算法，立即重传这个被确认过3次的报文段之后的报文段，这样可以让发送方在超时事件之前知道报文发生了丢失。

快恢复算法，如果发送方连续接收到3个冗余ACK，发送方知道现在只是丢失了个别的报文段，此时调整门限值 ssthresh为当前拥塞窗口的一半，同时设置拥塞窗口 cwnd为新的门限值（发生报文段丢失时拥塞窗口的一半），而不是从1开始。



如上图所示，当cwnd = 24时，收到了3个冗余ACK，于是不启动慢开始，而是执行快恢复算。这时，发送方调整门限值 ssthresh = cwnd / 2 = 12，同时设置拥塞窗口 cwnd = ssthresh = 12，并开始拥塞避免算法。

TCP对这种丢包事件的行为，相比于超时指示的丢包，不那么剧烈，所以对于连续收到3个冗余ACK，拥塞窗口不会从1开始开始。

4.总结