**拥塞**

即对资源的需求超过了可用的资源。若网络中许多资源同时供应不足，网络的性能就要明显变坏，整个网络的吞吐量随之负荷的增大而下降。

**定义**

tcp模块还有一个重要的任务，就是提高网络利用率，降低丢包率,并保证网络资源的对每条数据流的公平性。这就是所谓的拥塞控制。拥塞控制是一个全局性的过程，涉及到所有的主机、路由器，以及与降低网络传输性能有关的所有因素。

**拥塞控制的目标**

* 提高网络利用率
* 降低丢包率
* 保证网络资源的对每条数据流的公平性

**拥塞控制的方法**

慢启动、拥塞避免、快速重传和快速恢复

**慢启动和拥塞避免**

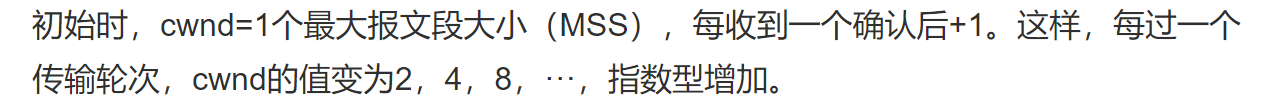
* MSS:报文段
* RTT：往返时间 (表示从发送端发送数据开始，到发送端收到来自接收端的确认（接收端收到数据后便立即发送确认，不包含数据传输时间,总共经历的时间)。
* SWND:发送窗口 (发送端向网络一次连续写入的数据量，窗口大小就是无需等待确认应答而可以继续发送数据的最大值）
* CWND:拥塞窗口 指某一源端数据流在一个RTT内可以最多发送的数据包数
* RWND：接收窗口

之前我已经介绍过[滑动窗口](https://blog.csdn.net/qq_43313035/article/details/89060259)，它的其中一个作用就是进行流量控制可以避免发送方过载接收方。但是却无法避免过载网络，这是因为接收窗口只反映了服务器个体的情况，却无法反映网络整体的情况。

发送方维持一个拥塞窗口 cwnd ( congestion window )的状态变量。拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。发送方让自己的发送窗口等于拥塞窗口。

发送方控制拥塞窗口的原则是：只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。

慢开始算法：

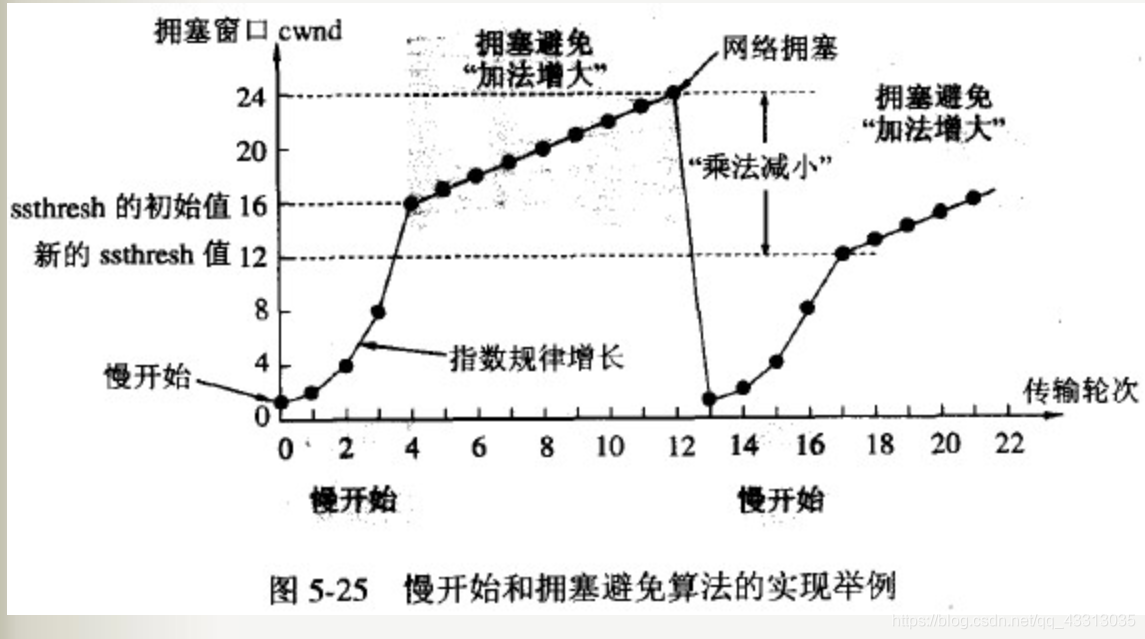
* 拥塞窗口和接收窗口共同决定了发送者的发送窗口
* 当主机开始发送数据时，如果立即所大量数据字节注入到网络，那么就有可能引起网络拥塞，因为现在并不清楚网络的负荷情况。
* 较好的方法是 先探测一下，即由小到大逐渐增大发送窗口，也就是说，由小到大逐渐增大拥塞窗口数值
* 通常在刚刚开始发送报文段时，先把拥塞窗口 cwnd 设置为一个最大报文段MSS的数值。而在每收到一个对新的报文段的确认后，把拥塞窗口增加至多一个MSS的数值。用这样的方法逐步增大发送方的拥塞窗口 cwnd ，可以使分组注入到网络的速率更加合理。  
  
* 如果不施加手段进行控制，慢启动必然使得CWBD很快膨胀，为防止拥塞窗口cwnd的增长引起网络拥塞，还需要另外一个变量，慢开始门限ssthresh  
       
  cwnd <ssthresh时,进行慢开始算法。    
  cwnd>ssthresh时,进行拥塞避免算。  
  cwnd = ssthresh时,两者皆可

拥塞避免算法：

* 让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的cwnd 拥塞窗口cwnd加1cwnd ，而不是加倍cwnd 。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增长速率缓慢得多。
* 不论是慢开始还是拥塞避免只要网络出现拥塞（没有按时到达）时，就把ssthresh的值置为出现拥塞时的拥塞窗口的一半（但不能小于2），以及cwnd置为1，进行慢开始。 目的是迅速减少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时间把队列中积压的分组处理完毕。

**注意**：

* 由指数增长拉低到线性增长，降低出现拥塞的可能。“拥塞避免”并非指完全能够避免拥塞，利用以上的措施要完全避免网络拥塞还是不可能的。

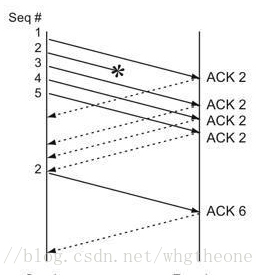
图片来源于网络  
  
以上是发送端在未检测到拥塞时所采用的积极避免拥塞的方法。接下来介绍拥塞发生时（可能发生在慢启动阶段或者拥塞避免阶段）拥塞控制的行为。

**发送端判断拥塞发生的依据：**  
1.传输超时（TCP重传定时器溢出）  
2.接收到重复的确认报文段

拥塞控制对这两种情况有不同的处理方式。对第一种情况仍然使用慢启动和拥塞避免。对第二种情况使用快速重传和快速恢复（如果是真的发生拥塞的话），注意：第二种情况如果发生在第一种情况之后，则也被拥塞控制当成第一种情况来对待

**快速恢复和快速重传**

在很多情况下，发送端都有可能接收到重复的确认报文段，比如TCP报文端丢失，或者接收端收到乱序TCP报文段并重排之等。

**拥塞控制算法需要判断当收到重复的确认报文端时，网络是否真的发生了阻塞，或者说TCP报文端是否真的丢失了。具体的做法是：发送端如果连续收到3个重复的确认报文端，就认为是拥塞发生了。然后它启用快速重传和快速恢复算法来处理拥塞。**  


**快速重传**(Fast retransmit):要求接收方在收到一个失序的报文段后就立即发出重复确认（为的是使发送方及早知道有报文段没有到达对方），而不要等到自己发送数据时捎带确认。

**如果在超时重传定时器溢出之前，接收到连续的三个重复冗余ACK（其实是收到4个同样的ACK，第一个是正常的，后三个才是冗余的），发送端便知晓哪个报文段在传输过程中丢失了，于是重发该报文段，不需要等待超时重传定时器溢出，大大提高了效率。这便是快速重传机制。**

**快速恢复**(Fast retransmit)具体过程：

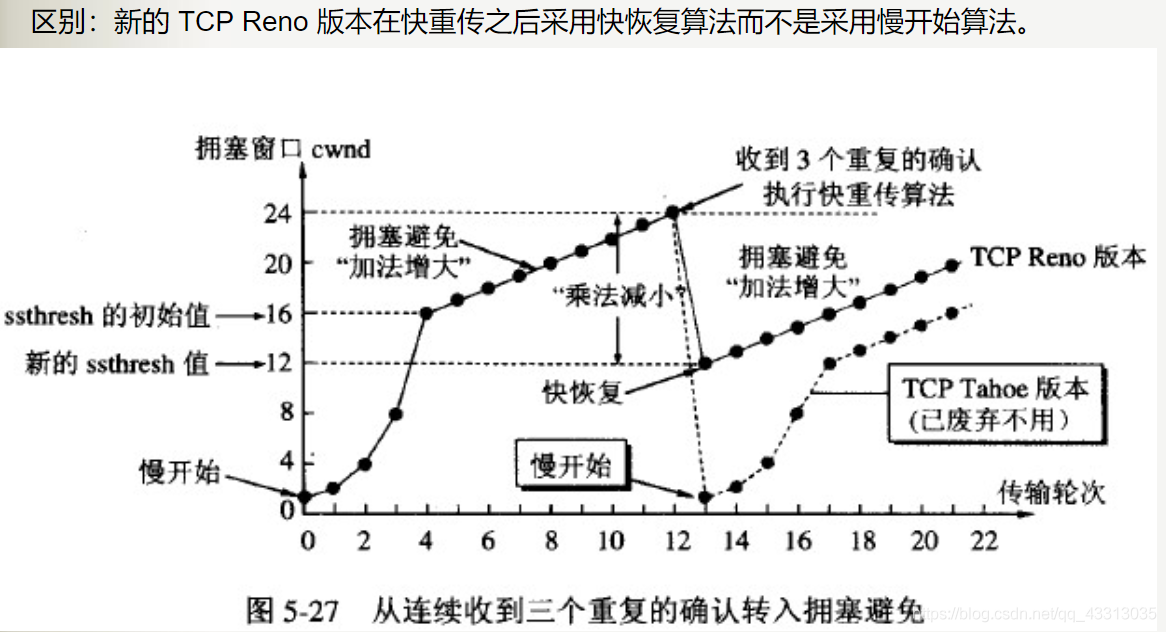
（1）当发送方连续收到三个重复确认，就执行“乘法减小”算法，把慢开始门限ssthresh减半。这是为了预防网络发生拥塞。然后立即重传丢失的报文段，并将CWND设置为新的ssthresh（减半后的ssthresh）  
请注意：接下去不执行慢开始算法

有些快重传实现是把开始时的拥塞窗口cwnd值再增大一点，即等于 ssthresh + 3 \* MSS 。这样做的理由是：既然发送方收到三个重复的确认，就表明有三个分组已经离开了网络。这三个分组不再消耗网络 的资源而是停留在接收方的缓存中。可见现在网络中并不是堆积了分组而是减少了三个分组。因此可以适当把拥塞窗口扩大了些。

（2）由于发送方现在认为网络很可能没有发生拥塞，因此与慢开始不同之处是现在不执行慢开始算法（即拥塞窗口cwnd现在不设置为1），而是把cwnd值设置为慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。

(3)每次收到一个重复的确认时，设置CWND=CWND+SMSS（拥塞窗口加1）.此时发送端可以发送新的TCP报文段

(4)当收到新数据的确认时，设置CWND=ssthresh(ssthresh是新的慢启动门限值，由第一步计算得到)  
原因是因为该ACK确认了新的数据，说明从重复ACK时的数据都已收到，该恢复过程已经结束，可以回到恢复之前的状态了，也即再次进入拥塞避免状态。

快速重传和快速恢复完成之后，拥塞控制将恢复到拥塞避免阶段.  


**总结**

* 当出现**超时重传**和**冗余ack**的时候慢**启动门限都要设置为当前发送窗口的一半**
* 不同的就是**超时重传还得将拥塞窗口大小设为1，重新进入慢启动**，而**冗余ack则是将拥塞窗口设为慢启动门限大小并且进入拥塞避免**

https://blog.csdn.net/qq\_43313035/article/details/89069962