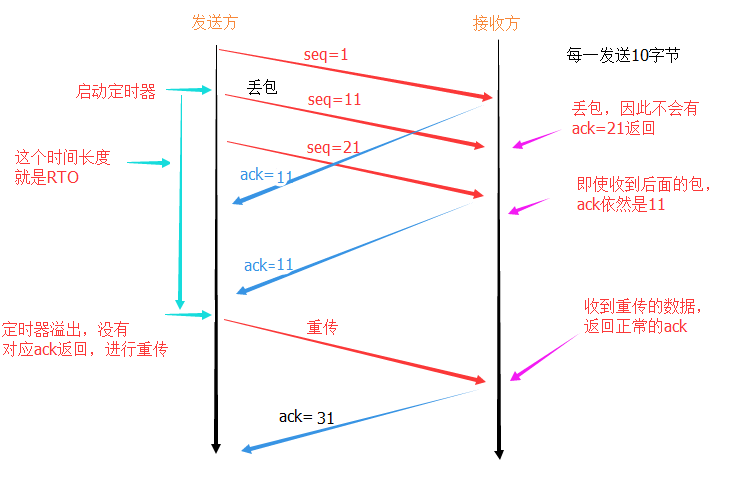
TCP超时与重传机制   
　　   
　　TCP协议是一种面向连接的可靠的传输层协议，它保证了数据的可靠传输，对于一些出错，超时丢包等问题**TCP设计的超时与重传机制**。其基本原理：**在发送一个数据之后，就开启一个定时器，若是在这个时间内没有收到发送数据的ACK确认报文，则对该报文进行重传，在达到一定次数还没有成功时放弃并发送一个复位信号。**　　这里比较重要的是**重传超时时间**，怎样**设置这个定时器的时间（RTO）**，**从而保证对网络资源最小的浪费。因为若RTO太小，可能有些报文只是遇到拥堵或网络不好延迟较大而已，这样就会造成不必要的重传。太大的话，使发送端需要等待过长的时间才能发现数据丢失，影响网络传输效率。**   
　　由于**不同的网络情况不一样，不可能设置一样的RTO，**实际中RTO是**根据网络中的RTT（传输往返时间）来自适应调整的。**具体关系参考相关算法。   
　　通过图来了解重传机制：   
　

从图可以知道，发送方连续发送3个数据包，**其中第二个丢失，没有被接收到，因此不会返回对应的ACK，每发送一个数据包，就启动一个定时器，当第二个包的定时器溢出了还没有收到ack，这时就进行重传。**

TCP慢启动

慢启动是TCP的一个拥塞控制机制，**慢启动算法**的基本思想是当**TCP开始在一个网络中传输数据或发现数据丢失并开始重发时**，**首先慢慢的对网路实际容量进行试探，避免由于发送了过量的数据而导致阻塞。**　　慢启动**为发送方的TCP**增加了另一个窗口：**拥塞窗口(congestion window)，记为cwnd。**当与另一个网络的主机建立TCP连接时，拥塞窗口**被初始化为 1个报文段**（即另一端通告的报文段大小）。**每收到一个ACK，拥塞窗口就增加一个报文段**（cwnd以字节为单位，但是慢启动以报文段大小为单位进行增加）。**发送方取（拥塞窗口与通告窗口中的最小值，下面有解释）作为发送上限。**

**拥塞窗口是发送方使用的流量控制**，而**通告窗口则是接收方使用的流量控制。**

发送方开始时发送一个报文段，然后等待 ACK。当收到该ACK时，拥塞窗口从1增加为2，即可以发送两个报文段。当收到这两个报文段的 A C K时，拥塞窗口就增加为4。这是一种指数增加的关系。

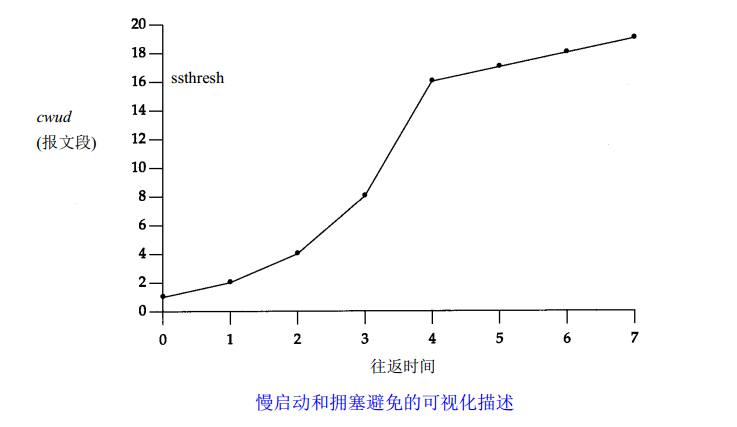
拥塞避免算法

网络中拥塞的发生**会导致数据分组丢失，需要尽量避免。**在实际中，拥塞算法与慢启动通常在一起实现，其基本过程：   
　　 1. 对一个给定的连接，初始化cwnd为1个报文段，ssthresh为65535个字节。   
　　 2. TCP输出例程的输出不能超过cwnd和接收方通告窗口的大小。

拥塞避免是发送方使用 的流量控制，而通告窗口则是接收方进行的流量控制。前者是发送方感受到的网络拥塞的估计，而后者则与接收方在该连接上的可用缓存大小有关。   
　　 3. 当拥塞发生时（超时或收到重复确认（丢包）），**ssthresh被设置为当前窗口大小的一半**（cwnd 和接收方通告窗口大小的最小值，**最少为2个报文段**）。此外，如果是超时引起了拥塞，则 cwnd被设置为1个报文段（这就是慢启动）。   
　　 4. 当新的数据被对方确认时，就增加cwnd，但增加的方法依赖于是否正在进行慢启动或拥塞避免。

如果cwnd小于或等于ssthresh，则正 在进行慢启动，否则正在进行拥塞避免。

慢启动一直持续到回到当拥塞发生时所处位置的半数才停止（因为记录了在步骤2 中制造麻烦的窗口大小的一半），然后转为执行拥塞避免。   
　　 慢启动算法初始设置cwnd为1个报文段，此后每收到一个确认就加 1。那样，这会使窗口按指数方式增长：**发送 1个报文段，然后是2个，接着是4个……。**



在该图中，假定当**cwnd为32个报文段时就会发生拥塞**。于是设**置 ssthresh为16个报文段，而cwnd为1个报文段。在时刻 0发送了一个报文段，并假定在时刻 1接收到它的ACK，此时cwnd增加为2。接着发送了2个报文段，并假定在时刻 2接收到它们的ACK，于是cwnd增加为4（对每个ACK增加1次）。这种指数增加算法一直进行到在时刻3和4之间收到8个ACK后cwnd等于ssthresh时才停止指数增长，**

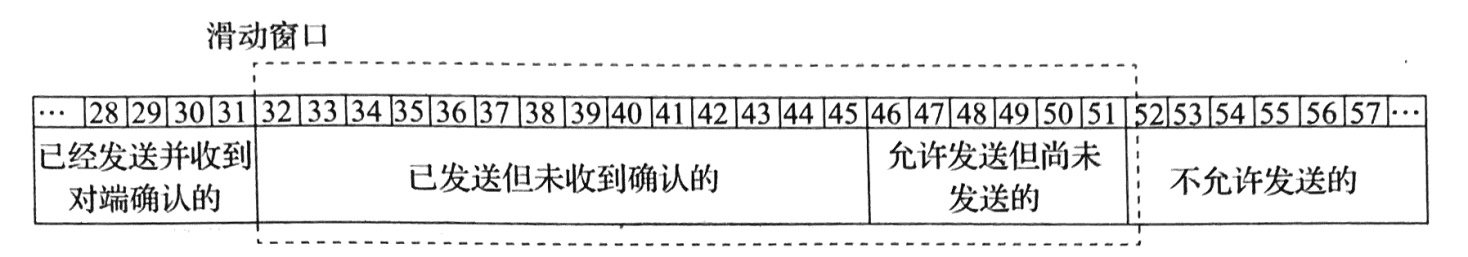
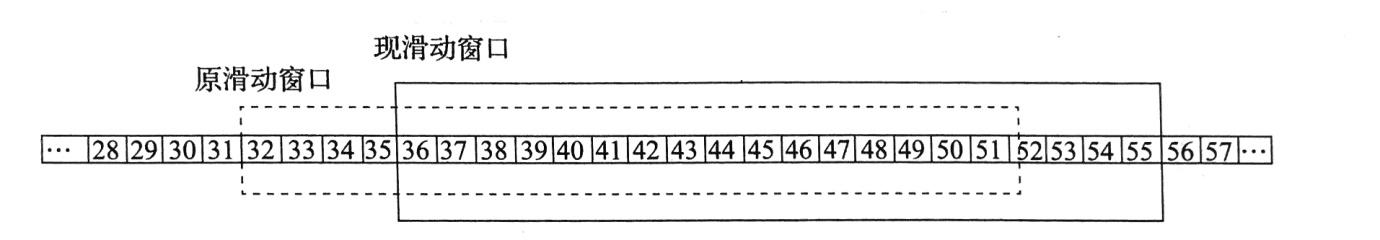
**从该时刻起，cwnd以线性方式增加，在每个往返时间内最多增加 1个报文段。**

**再熟悉一次：**

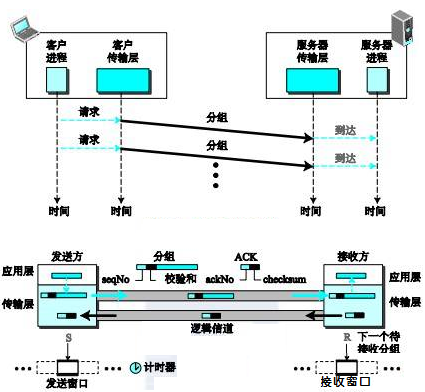
**TCP超时重传**

　　原理是在**发送某一个数据以后就开启一个计时器，在一定时间内如果没有得到发送的数据报的ACK报文，那么就重新发送数据**，直到发送成功为止。   
　　影响超时重传机制协议效率的一个关键参数是**重传超时时间**（**RTO**，Retransmission TimeOut）。RTO的值被设置过大过小都会对协议造成不利影响。   
　　（1）RTO设长了，**重发就慢，没有效率，性能差。**　　（2）RTO设短了，**重发的就快，会增加网络拥塞，**导致更多的超时，更多的超时导致更多的重发。   
　　连接往返时间（RTT，Round Trip Time），指发送端从发送TCP包开始到接收它的立即响应所消耗的时间。

**TCP滑动窗口（接收方）**

**作用：（1）提供TCP的可靠性；（2）提供TCP的流控特性**  
  


TCP的滑动窗口的可靠性也是建立在**“确认重传”**基础上的。   
发送窗口只有**收到对端对于本段发送窗口内字节的ACK确认**，才会**移动发送窗口的左边界。**  
**接收端可以根据自己的状况通告窗口大小，从而控制发送端的接收，进行流量控制。**

**停止等待ARQ协议（stop and wait）**   
   
**当发送窗口和接收窗口都等于1时，就是停止等待协议。发送端给接收端发送数据，等待接收端确认回复ACk，并停止发送新的数据包，开启计时器。**数据包在计时器超时之前得到确认，那么计时器就会关闭，并发送下一个数据包。如果计时器超时，发送端就认为数据包丢失或被破坏，需要重新发送之前的数据包，说明数据包在得到确认之前，发送端需要存储数据包的副本。   
停止等待协议是发出一个帧后得到确认才发下一个，降低了信道的利用率。

**TCP拥塞控制**

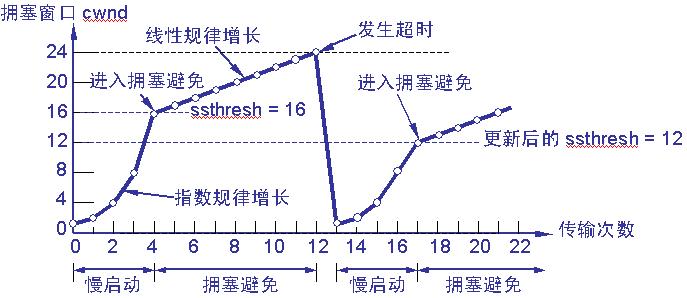
　　拥塞控制是一个全局性的过程； 流量控制是点对点通信量的控制   
　　TCP拥塞控制4个核心算法：**慢开始（slow start）、拥塞避免（Congestion Avoidance）、快速重传（fast retransmit）、快速回复（fast recovery）**　　拥塞窗口（cwnd，congestion window），**其大小取决于网络的拥塞程度，并且动态地在变化。**  
　　   
　　慢开始算法的思路就是，不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说**由小到大逐渐增加拥塞窗口的大小。**

**为了防止cwnd增长过大引起网络拥塞，还需设置一个慢开始门限ssthresh状态变量**。ssthresh的用法如下：   
**当cwnd < ssthresh时，使用慢开始算法。   
当cwnd > ssthresh时，改用拥塞避免算法。   
当cwnd = ssthresh时，慢开始与拥塞避免算法任意。**

**拥塞避免算法让拥塞窗口缓慢增长，即每经过一个往返时间RTT就把发送发的拥塞窗口cwnd加1，而不是加倍。**

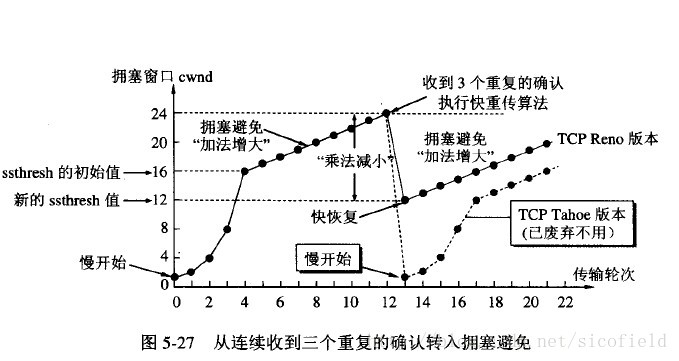
无论是在慢开始阶段还是在拥塞避免阶段，只要**发送方判断网络出现拥塞**，就把**慢开始门限设置为出现拥塞时的发送窗口大小的一半。然后把拥塞窗口设置为1，执行慢开始算法。**

**无论是在**慢开始阶段**还是在**拥塞避免阶段**，只要发送方判断网络出现拥塞（其根据就是没有收到确认，虽然没有收到确认可能是其他原因的分组丢失，但是因为无法判定，所以都当做拥塞来处理），就把慢开始门限设置为出现拥塞时的发送窗口大小的一半。然后把拥塞窗口设置为1，执行慢开始算法。如下图：**

如下图：   
   
拥塞控制的具体过程如下：   
**（1）TCP连接初始化，将拥塞窗口设置为1   
（2）执行慢开始算法，cwnd按指数规律增长，直到cwnd=ssthresh时，开始执行拥塞避免算法，cwnd按线性规律增长   
（3）当网络发生拥塞，把ssthresh值更新为拥塞前ssthresh值的一半，cwnd重新设置为1，按照步骤（2）执行**

**快重传和快恢复**

**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。   
　　快重传算法规定，**发送方只要一连收到3个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计数器时间到期。**　　 　　   
　　**快速恢复(Fast Recovery)**   
　　（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。请注意：接下去不执行慢开始算法。 （超时才重新执行慢开始算法）**（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。**①当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半。但是接下去并不执行慢开始算法。

②考虑到如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。如下图：

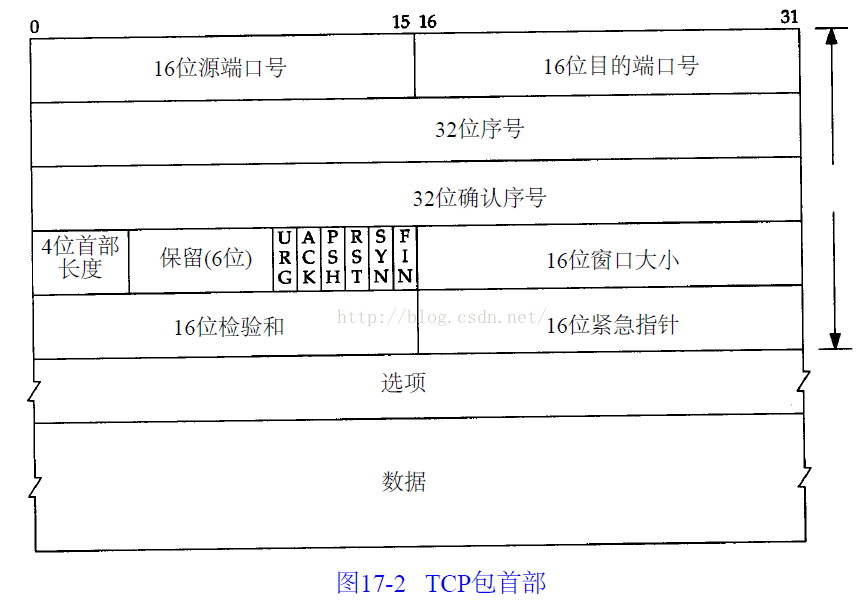
**发送方窗口的上限值 = Min [ rwnd, cwnd ]**

**当rwnd < cwnd 时，是接收方的接收能力限制发送方窗口的最大值。**

**当cwnd < rwnd 时，则是网络的拥塞限制发送方窗口的最大值。**

其中**rwnd为通告窗口大小（滑动窗口），cwnd为拥塞窗口的大小。**

 不管是多么复杂的协议原理，我想首先肯定要落实到协议的字段上面。对于TCP来说流量控制无非是就是在序号和窗口大小上面做文章。



因此我们先了解一下16位的窗口大小究竟有什么作用。

窗口分为滑动窗口和拥塞窗口。

**滑动窗口是接受数据端使用的窗口大小，用来告知发送端接收端的缓存大小**，以此可以控制发送端发送数据的大小，从而达到流量控制的目的。

那么对于数据的发送端就是拥塞窗口了，拥塞窗口不代表缓存，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](http://baike.baidu.com/view/25880.htm" \t "_blank)数

**一、滑动窗口**

滑动窗口协议是传输层进行流控的一种措施，

**接收方通过通告发送方自己的窗口大小，从而控制发送方的发送速度**，从而达到防止发送方发送速度过快而导致自己被淹没的目的。

对ACK的再认识，ack通常被理解为收到数据后给出的一个确认ACK，ACK包含两个非常重要的信息：

一是期望接收到的下一字节的序号n，该n代表接收方已经接收到了前n-1字节数据，此时如果接收方收到第n+1字节数据而不是第n字节数据，接 收方是不会发送序号为n+2的ACK的。

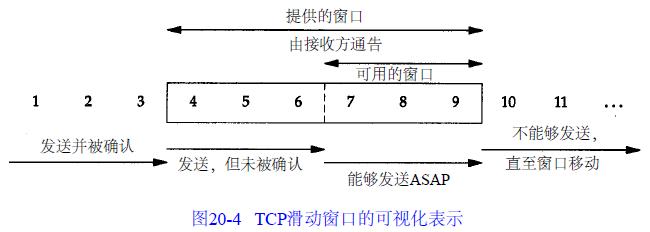
**举**个例子，假如接收端收到1-1024字节，它会发送一个确认号为1025的ACK给发送端，但是接下来收到的是 2049-3072，它是不会发送确认号为3072的ACK,而依旧发送1025的ACK。

二是当前的窗口大小m，如此发送方在接收到ACK包含的这两个数据后就可以计算出

**还可以发送多少字节的数据给对方**，假定当前发送方已发送到第x字节，则可以发送的字节数就是y=m-(x-n).这就是滑动窗口控制流量的基本原理.

（其中x-n为还未回复确认的数据，）

滑动窗口协议如图所示:



在这个图中，我们将字节从1至11进行 标号。接收方通告的窗口称为提出的窗口，**它覆盖了从第4字节到第9字节的区域**，表明接收方已经确认了**包括第3字节在内的数据，且通告窗口大小为6**。我们知道窗口大小是与确认序号相对应的。**发送方计算它的可用窗口，该窗口表明多少数据可以立即被发送。当接收方确认数据后，这个滑动窗口不时地向右移动。窗口两 个边沿的相对运动增加或减少了窗口的大小。**

我们使用三个术语来描述窗口左右边沿的运动：

* **称窗口左边沿向右边沿靠近为窗口合拢。这种现象发生在数据被发送和被确认时。**
* 当窗口**右边沿向右移动时将允许发送更多的数据**，我们称之为**窗口张开**。**这种现象发生在另一端的接收进程读取已经确认的数据并释放了T C P的接收缓存时。**
* **当右边缘向左移动时，称之为窗口收缩。**

二、拥塞窗口

迄今为止，在本章所有的例子中，**发送方一开始便向网络发送多个报文段，直至达到接收方通告的窗口大小为止。**当发送方和接收方处 于同一个局域网时，这种方式是可以的。**但是如果在发送方和接收方之间存在多个路由器和速率较慢的链路时，就有可能出现一些问题。**一些中间路由器必须缓存分组，并有可能耗尽缓存，[Jacobson 1988]证明了这种连接方式是如何严重降低了TCP连接的吞吐量的。

现在，TCP需要支持一种被称为“慢启动(slow start)”的算法。该算法通过观察到新分组进入网络的速率应该与另一端返回确认的速率相同而进行工作。

慢启动为发送方的TCP增加了另一个窗口：拥塞窗口(congestion window)，记为cwnd。当与另一个网络的主机建立TCP连接时，拥塞窗口被初始化为1个报文段（即另一端通告的报文段大小）。每收到一个ACK， 拥塞窗口就增加一个报文段（cwnd以字节为单位，但是慢启动以报文段大小为单位进行增加）。发送方取拥塞窗口与通告窗口中的最小值作为发送上限。

拥塞窗口是发送方使用的流量控制，而通告窗口则是接收方使用的流量控制。

发送方开始时发送一个报文段，然后等待ACK。当收到该ACK时，拥塞窗口从1增加为2，即可以发送两个报文段。当收到这两个报文段的ACK时，拥塞窗口就增加为4。这是一种指数增加的关系。

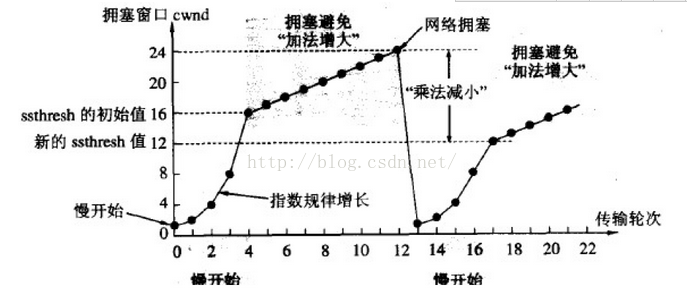
拥塞避免是发送方使用 的[流量控制](http://baike.baidu.com/view/190232.htm)，而通告窗口则是接收方进行的流量控制。

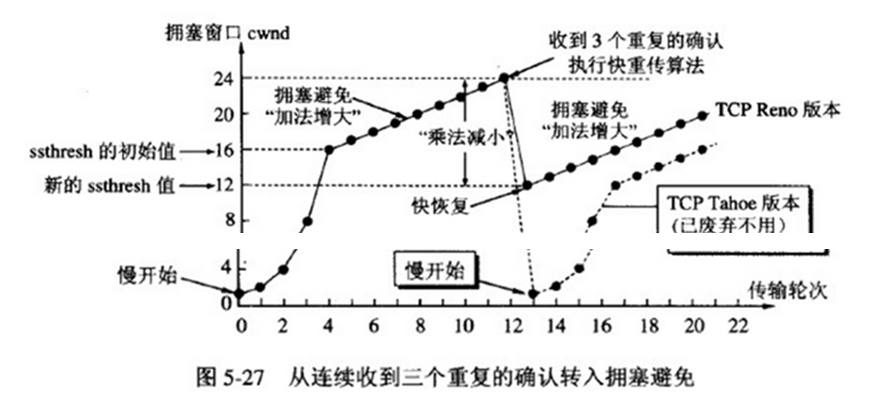
前者是发送方感受到的[网络拥塞](http://baike.baidu.com/view/1452069.htm" \t "_blank)的估 计，而后者则与接收方在该连接上的可用[缓存大小](http://baike.baidu.com/view/3749508.htm" \t "_blank)有关。

拥塞控制：防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。拥塞控制所要做的都有一个前提：网络能够承受现有的网络负荷。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。

拥塞发生有超时和收到重复确认两种情况，

**超时会启用慢启动和避免拥塞算法**

  
**无论是在**慢开始阶段**还是在**拥塞避免阶段**，只要发送方判断网络出现拥塞（其根据就是没有收到确认，虽然没有收到确认可能是其他原因的分组丢失，但是因为无法判定，所以都当做拥塞来处理），就把慢开始门限设置为出现拥塞时的发送窗口大小的一半。然后把拥塞窗口设置为1，执行慢开始算法。**

**重复确认可以避免过长时间的等待，会启用快速重传和快速恢复****①当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半。但是接下去并不执行慢开始算法。**

**②考虑到如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。**