相信读者朋友们看到这个标题后，不禁会想到前面提到的TCP之流量控制，而本文要说的拥塞控制又是什么？跟流量控制有啥不一样？它又有哪些值得我们学习的呢？不得不说，**拥塞控制比流量控制重要多了**，我们也逐步探索到了TCP的核心地带，将学习慢开始算法、拥塞避免算法、快重传和快恢复，下面待我细细道来。

一、拥塞控制解决了什么问题

前面我们介绍了 TCP 利用滑动窗口来做流量控制，流量控制这种机制确实可以防止发送端向接收端过多的发送数据，但是它只关注了发送端和接收端自身的状况，而没有考虑整个网络的通信状况。

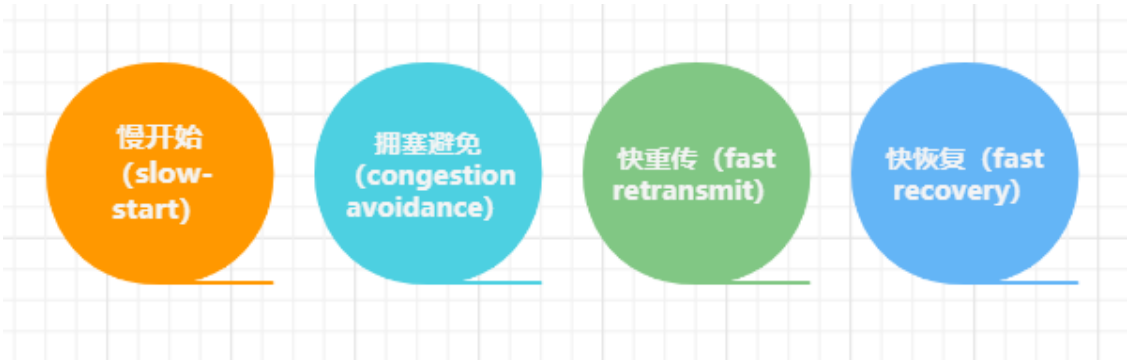
为了考虑整个网络环境的变化带来的影响，比如当前网速很快，因为网络并不拥挤，数据可以尽可能快点发送出去，但是到了某个时间点，网络又变得拥塞起来，这个时候需要进行一定的限制，否则会造成网络更重的负担，而更重的负担会造成更多的时延和丢包，形成雪崩的网络风暴，因此需要引入拥塞控制来全盘考虑。

总的来说流量控制和拥塞控制的区别是：

拥塞控制问题是一个全局性的问题，涉及到所有的主机、所有的路由器、以及与降低网络传输性能有关的所有因素，保障网络能承受现有的网络负荷。若出现拥塞而不进行控制，整个网络的吞吐量将随输入负荷的增大而下降。

流量控制往往指的是点对点通信量的控制，是个端到端的问题。流量控制所要做的就是控制发送端发送数据的速率，以使得接收端来得及接受。

可以看的出来，拥塞控制的难度可要比流量控制要高一个层级，考虑的是动态的、全局的问题。



为了实现拥塞控制，因特网建议标准RFC2581定义了进行拥塞控制的四种算法，分别是：

- **慢开始（slow-start）**
- **拥塞避免（congestion avoidance）**
- **快重传（fast retransmit）**
- **快恢复（fast recovery）**

为了聚焦拥塞控制，我们有必要忽略其他因素的干扰，我们假定有如下条件：

- 数据是单方向传送，而另一个方向只传送确认；
- 接收方总是有足够大的缓存空间，因而发送方发送窗口的大小由网络的拥塞程度来决定，这里主要是为了忽略流量控制的干扰；
- 以最大报文段MSS的大小为讨论问题的单位，而不是以字节为单位；

好了，下面我们逐一来看看这些拥塞控制算法，在说明拥塞控制算法前，有必要说一下拥塞窗口。

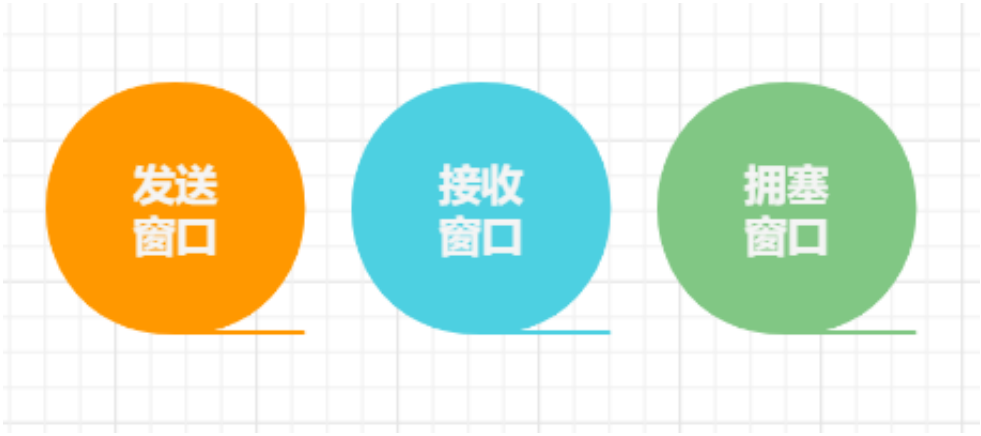
二、拥塞窗口

英文名叫做Congestion Window，我们下面简称为cwnd。

发送方维护一个叫做拥塞窗口cwnd的状态变量，其值取决于网络的拥塞程度，并且动态变化。

- 拥塞窗口cwnd的维护原则：只要网络没有出现拥塞，拥塞窗口就再增大一些；但只要网络出现拥塞，拥塞窗口就减少一些。
- 判断出现网络拥塞的依据：没有按时收到应当到达的确认报文（即发生超时重传）

提到拥塞窗口，我们自然会想到之前学习的发送窗口和接收窗口。



接收窗口rwnd和拥塞窗口cwnd的区别是：

- 接收窗口：是接收端的限制，是接收端还能接收的数据量大小
- 拥塞窗口：是发送端的限制，是发送端在还未收到对端 ACK 之前还能发送的数据量大小

拥塞窗口cwnd和发送窗口swnd的区别是：

- 发送窗口：真正的发送窗口大小 = 「接收端接收窗口大小」 与 「发送端自己拥塞窗口大小」 两者的最小值

这很好理解，发送窗口能发送多少数据，取决于：

- 对方能接收多少数据（接收窗口）
- 自己为了避免网络拥塞主动控制不要发送过多的数据（拥塞窗口）

在这里，我们假定的是：发送窗口的大小仅由网络的拥塞程度来决定，因此swnd=cwnd。

除了cwnd外，发送方还需要维护一个叫做慢开始门限ssthresh的状态变量：

- 当cwnd < ssthresh时，使用慢开始算法；
- 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，既可使用慢开始算法，也可使用拥塞避免算法。

三、慢开始和拥塞避免

首先来看慢开始算法，假设慢开始门限ssthresh为16，假设初始的拥塞窗口cwnd值为1。

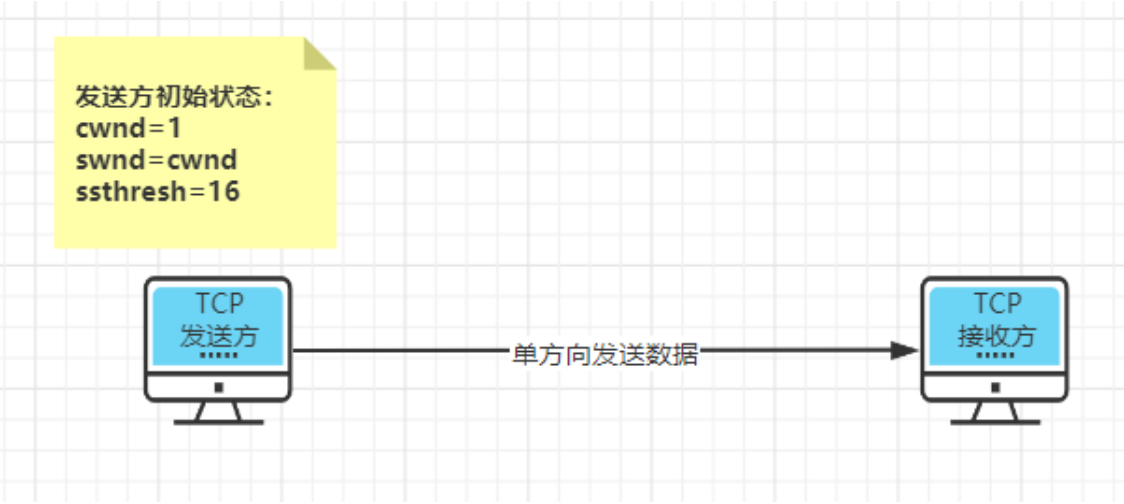
即我们假定发送方的条件如下：

- cwnd = 1
- swnd = cwnd
- ssthresh = 16

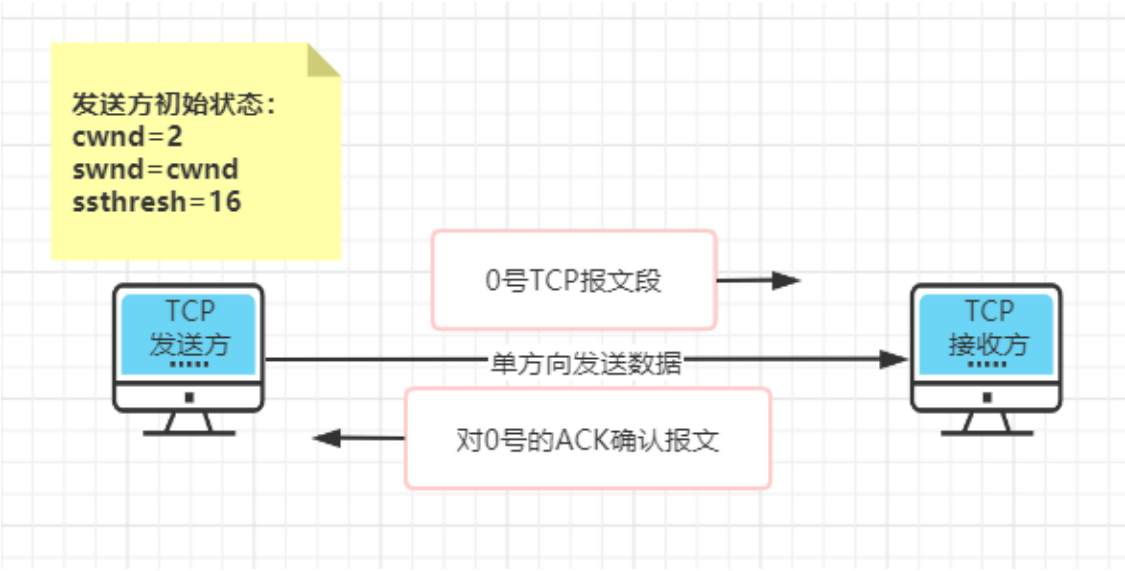
在执行慢开始算法时，发送方每收到一个对新报文段的确认时，就把拥塞窗口值加1，然后开始下一轮的传输。

当拥塞窗口值增长到慢开始门限值时，就改为执行拥塞避免算法。

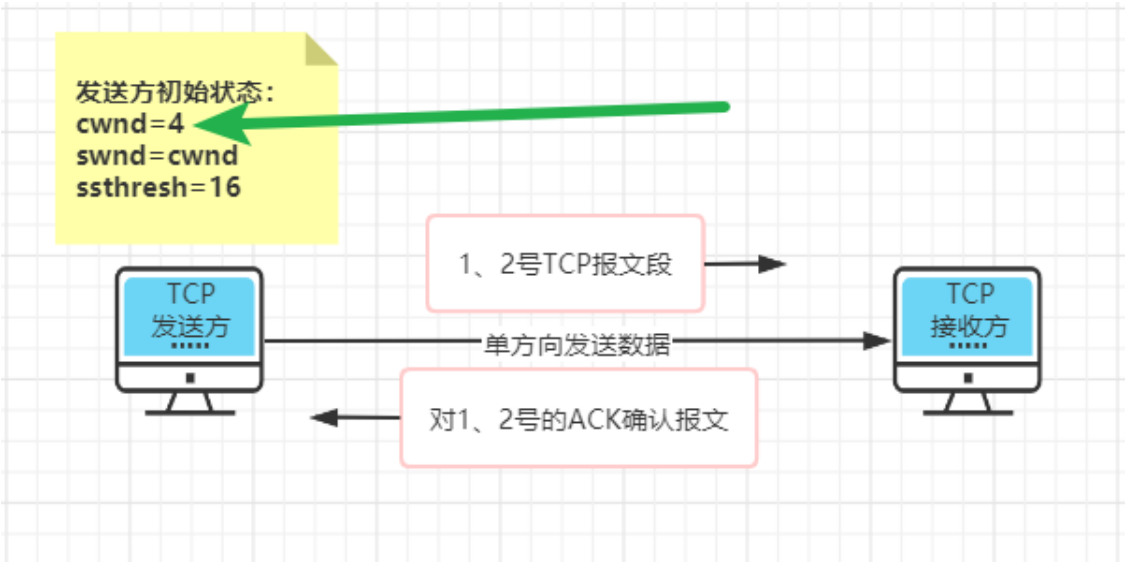
当前初始情况是拥塞窗口值为1，由于发送窗口大小等于拥塞窗口大小，因此只能发送一个报文段，换句话说，在我们这个情况下，拥塞窗口值是几，就能发送几个数据报文段。



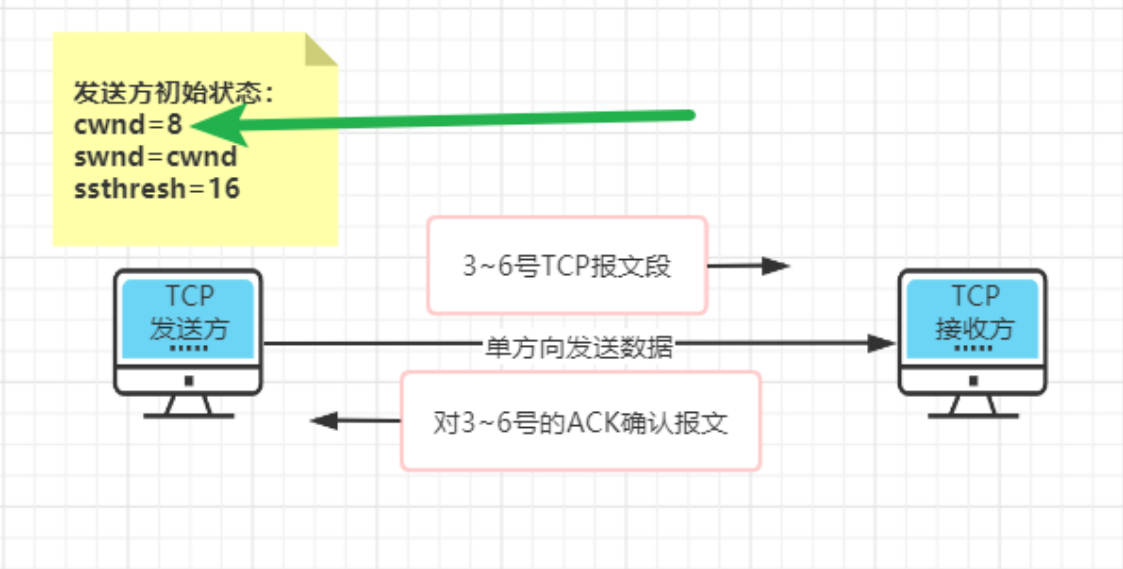
如图所示，首先发送方发送0号报文段，接收方收到该0号报文段后返回确认报文段，发送方收到确认报文段后，**将拥塞窗口值加1增大到2**。



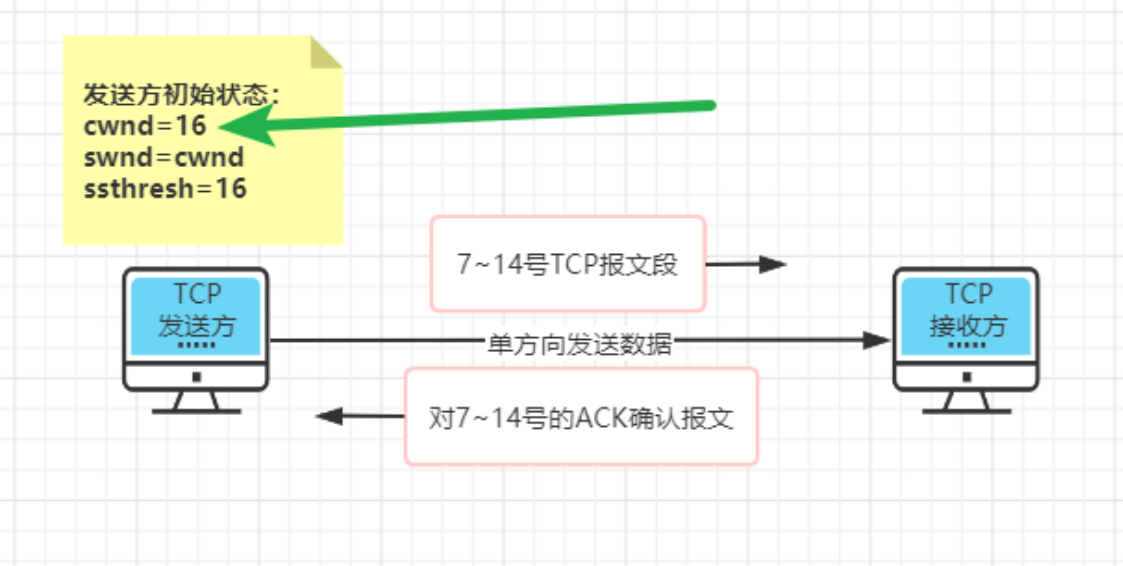
下面就可以直接一口气发送1号和2号两个TCP报文段，并且发送方收到了来自接收方的这2个报文段的ACK确认报文段，**下面拥塞窗口将被调整为4**：



此时发送方可以发送4个报文段了，即3-6号报文段，并且发送方收到了来自接收方的这4个报文段的ACK确认报文段，**下面拥塞窗口将被调整为8**：

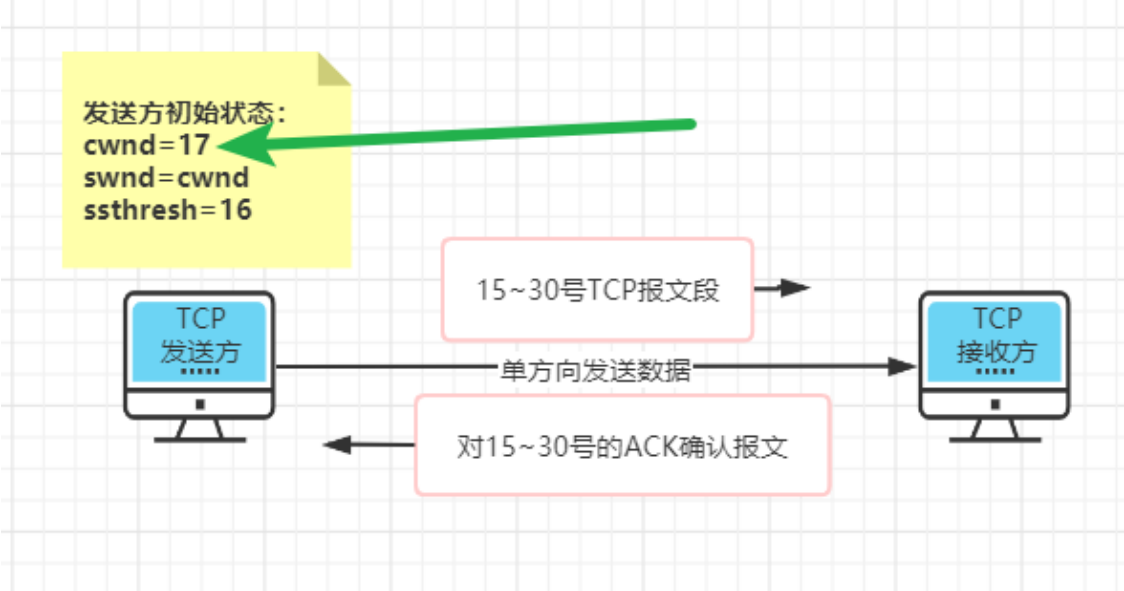


此时发送方可以发送8个报文段了，即7-14号报文段，并且发送方收到了来自接收方的这8个报文段的ACK确认报文段，**下面拥塞窗口将被调整为16**：



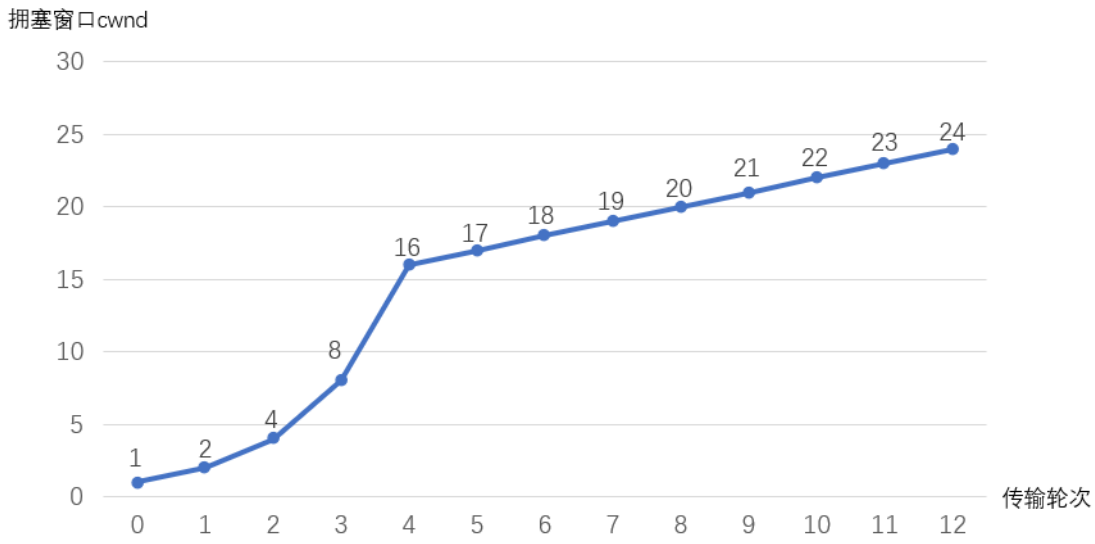
注意，**此时发送方的拥塞窗口值已经增大到了慢开始门限值，之后我们要改用拥塞避免算法，后续拥塞窗口值只能线性加1，而不像慢开始算法那样，每轮结束后，拥塞窗口值按指数规律增大。**

下面发送方可以发送15-30号共16个TCP报文段，接收方收到后，给发送方发送对15~30号报文段的确认报文段，发送方收到后，**将拥塞窗口值增加到17**。



后续如果发送方每轮次都可以及时收到接收方的ACK确认报文段，则继续将拥塞窗口加1增大。

用线型图来表示如上过程，横坐标是传输轮次，纵坐标是拥塞窗口值，那么以上拥塞窗口值的增长过程如下：



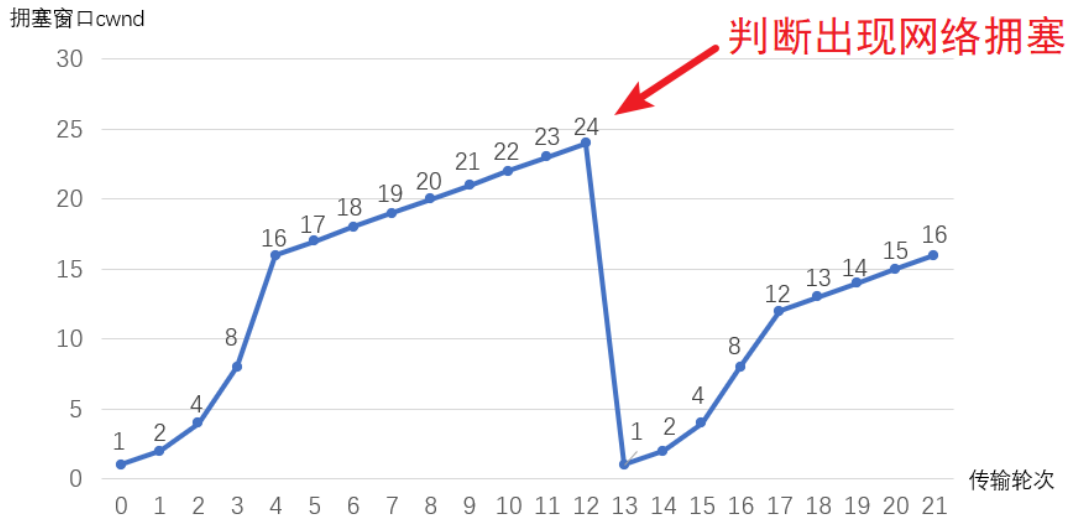
要是一切都那么顺利该多好啊，可惜没高兴多久，就出问题了。

目前拥塞窗口值增加到了24，发送方现在可以发送171~184共24个数据报文段，假设传输过程中丢失了几个报文段，这必然会造成发送方对这些丢失报文段的超时重传。

发送方依此判断网络很可能出现了拥塞，会进行以下操作：

- 1、将慢开始门限值更新为发生拥塞时拥塞窗口值的一半；
- 2、将拥塞窗口值减小为1，并重新开始执行慢开始算法。

那么接下来发生的事情就可以用折线图来表现：



第一步是将慢开始门限值更新为12，因为出现拥塞时窗口值是24；接下来重新开始慢开始算法将拥塞窗口进行指数增长，直到增长到慢开始门限值后，改为拥塞避免算法线性加1。

总结：TCP发送方一开始使用慢开始算法，让拥塞窗口值从1开始按指数规律增大，当拥塞窗口值达到慢开始门限值时，停止使用慢开始算法，转而使用拥塞避免算法，让拥塞窗口值按线性加1的规律增大。当发生超时重传时，发送方判断网络很可能出现了拥塞，采取相应措施，一方面将慢开始门限值更新为发生拥塞时拥塞窗口值的一半，另一方面将拥塞窗口值减小为1，并重新开始执行慢开始算法，调整之后，拥塞窗口值又从1开始按指数规律增大，当增大到了新的慢开始门限值时，转而使用拥塞避免算法，让拥塞窗口值按线性加1的规律增大。

“慢开始”的“慢”实际上指的是一开始向网络注入的报文较少，而并不指拥塞窗口值增长速度慢。

“拥塞避免”也不能完全“避免”拥塞，而是指在拥塞避免阶段将拥塞窗口控制为线性规律增长，使网络比较不容易出现拥塞。

四、快重传和快恢复

慢开始和拥塞避免算法是1988年提出的TCP拥塞控制算法，称为TCP Tahoe版本。

改进的思路：有时个别报文段会在网络中丢失，但实际上网络并未发生拥塞：

- 这将导致发送方超时重传，并误以为网络发生了拥塞；
- 发送方把拥塞窗口cwnd又设置为最小值1，并错误地启动慢开始算法，因而降低了传输效率。

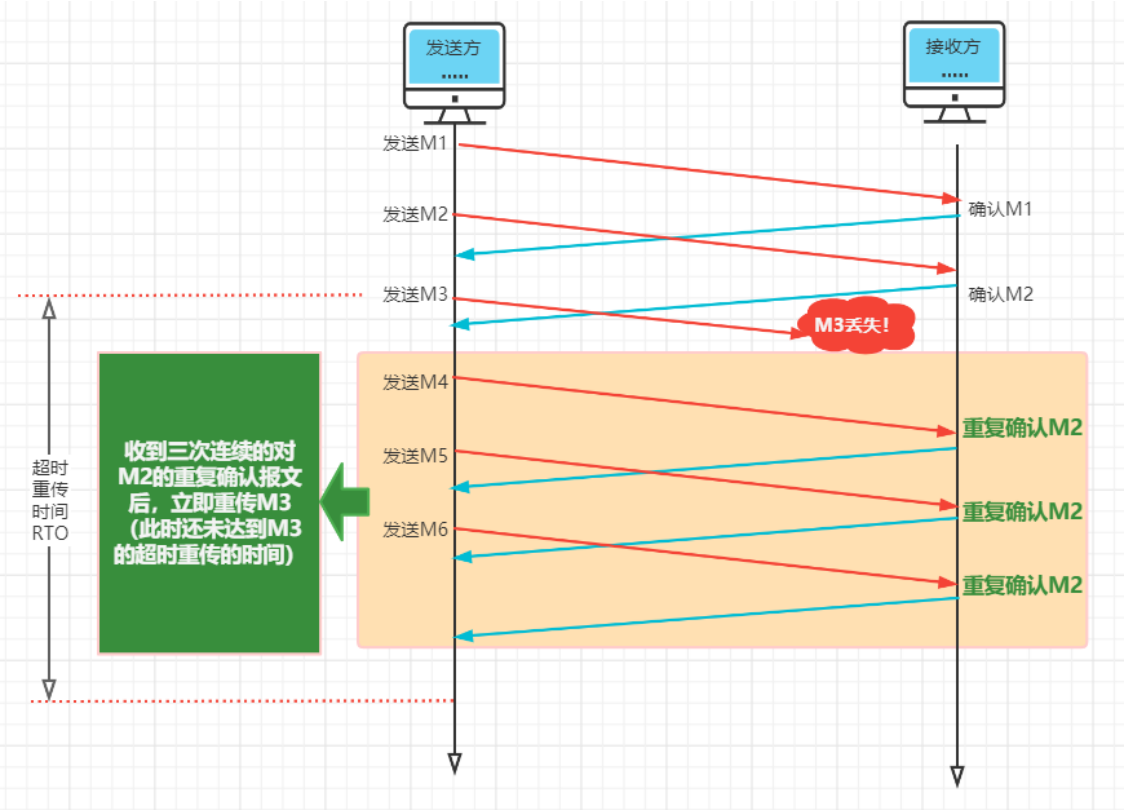
为了改进TCP性能，1990年于TCP Reno版本中又增加了两个新的拥塞控制算法，那就是快重传和快恢复算法。

采用快重传算法可以让发送方尽早知道发生了个别报文段的丢失。

所谓快重传，就是使发送方尽快进行重传，而不是等超时重传计时器超时再重传。

- 要求接收方不要等待自己发送数据时才进行捎带确认，而是要尽快确认；
- 即使收到了失序的报文段也要立即发出对已收到报文段的重复确认；
- 发送方一旦收到3个连续的重复确认，就将相应的报文段立即重传，而不是等该报文段的超时重传计时器超时再重传。
- 对于个别丢失的报文段，发送方不会出现超时重传，也就不会误以为出现了拥塞，进而降低拥塞窗口cwnd为1，使用快重传可以使整个网络的吞吐量提高约20%。

发送方一旦收到3个重复确认，就知道现在只是丢失了个别的报文段，于是不启动慢开始算法，而执行快恢复算法。



如图所示，发送方发送了M1和M2都顺利收到了M2的ACK，但是发送M3却在传输过程中丢失，不够发送方的拥塞窗口比较发，可以继续发送M4，但是接收方发现M3还未确认，因此返回M2的ACK；发送方继续发送M5，接收方发现M3还未确认，因此返回M2的ACK；发送方继续发送M6，接收方发现M3还未确认，因此返回M2的ACK；

当发送方收到三次连续的对M2的确认，就立即重传M3报文段，接收方收到后，发回针对M6的确认报文，表明序号到6为止的所有报文段都正确接收了。

以上过程就不会造成对M3的超时重传，而是提早进行了重传。

对于个别丢失的报文段，发送方不会出现超时重传，也就不会误认为出现了拥塞（进而将拥塞窗口降为1），使用快重传可以使整个网络的吞吐量提高约20%。

发送方一旦收到3个重复确认，就知道现在只是丢失了个别的报文段，于是不启动慢启动算法，而执行快恢复算法。

- 发送方将慢开始门限ssthresh值和拥塞窗口cwnd值调整为当前窗口的一半，开始执行拥塞避免算法
- 也有的快恢复实现是把快恢复开始时的拥塞窗口cwnd值再调大些，等于ssthresh+3（加3的理由是：既然发送方收到3个重复的确认，就表明有3个数据报文段已经离开了网络，可见现在网络中不是堆积了报文段而是减少了3个报文段，因此可以适当把拥塞窗口扩大些）

下面来看看使用了慢开始算法、拥塞避免算法、快重传、快恢复的一个整体拥塞窗口的变化情况：