**sender limits transmission**:

LastByteSent-LastByteAcked <= cwnd

**rate** : cwnd/RTT bytes/sec

拥塞窗口：

**congestion window(cwnd)**

LastByteSent-LastByteAcked<=min{cwnd,rwnd}

拥塞控制基本方法：发送方不断提高传输速率（窗口大小），直到发生数据丢失。

sshthresh: 发生丢失时的发送速率的一半。

拥塞控制的算法包括三部分：

慢启动 slow start

拥塞避免 congestion avoidance

快速恢复 fast recovery

**慢启动**

在连接刚建立时，指数地提高发送速率直到第一次丢包发生。

initially cwnd = 1MSS

double cwnd every RTT

done by incrementing cwnd for every ACK received

慢启动的结束方式：

第一种：检测到一个由超时指示的丢包事件，发送方将cwnd设为1，并重新开启慢启动过程，同时设置ssthresh 为丢包发生时cwnd的一半。

第二种：当检测到慢启动时cwnd到达ssthresh后，继续翻倍比较不妥，结束慢启动并转移到**拥塞避免**模式。

第三种：如果检测到三个冗余ACK，TCP发送方执行一种快速重传（3.5.4）并进入**快速恢复**状态。

**拥塞避免**

cwnd线性增长，每过一个RTT，cwnd增加1MSS。

出现超时时，cwnd被设为1MSS，设置ssthresh

出现三个冗余ACK时，将cwnd减半（再加上3MSS），接下来进入快速恢复状态。

**快速恢复**

对于收到的duplicate ACK，cwnd的值增加一个MSS。

TCP Tachoe:不论是超时还是三个冗余ACK，都将cwnd设为1，然后指数增长到ssthresh开始线性增长。

TCP Reno: 将cwnd减半+3MSS，然后开始线性增长。

**吞吐量（throughput）**

ignore slow start,assume always data to send

W：window size where loss occurs

平均吞吐量：0.75 W/RTT

**公平性**：

考虑K条TCP连接，每条都有不同的端到端路径，但是都经过一段传输速率为R bps的瓶颈链路（对于每条连接，沿着该连接路径上的所有其他链路都不用拥塞，且有充足的传输容量）假设每条连接都在传输一个大文件，如果每条连接的平均传输速率接近R/K，则认为该拥塞控制机制是公平的。

