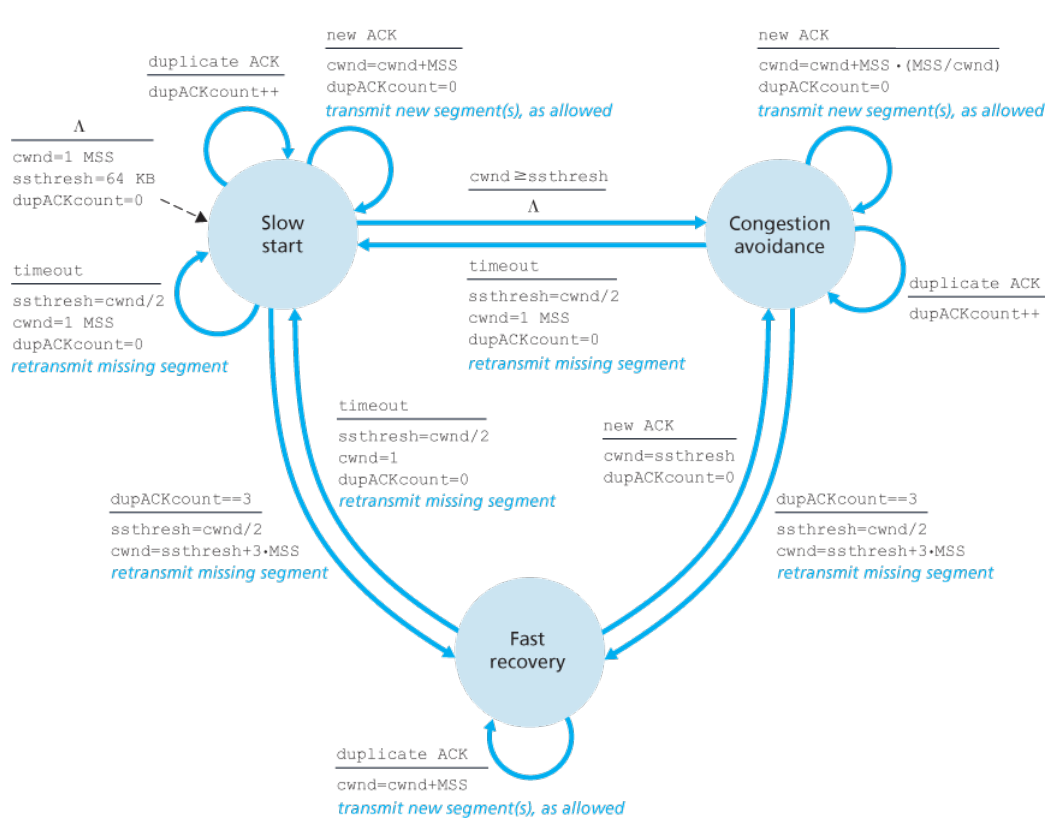
**关于TCP拥塞控制机制的说明与理解**

参考RFC 5681 TCP congestion control

https://tools.ietf.org/html/rfc5681

TCP的拥塞控制主要包括4个状态，RFC文档中的调整策略是按照**每收到一个ACK应答**(Acknowledgment)来描述操作过程的，因此在理解的时候最好结合教材(第7版)中给出的FSM图。

整个教材描述的图中，快速恢复是较让人迷惑的部分。下文中，黄色标记部分是关于“快速恢复”状态的理解。其余是关于其他部分的描述，供参考。



【关于状态转移图，圆圈表示状态；虚线箭头表示初始进入状态，以及对应的初始条件；蓝色箭头实线表示状态转移方向；黑色实线的上部分表示状态转移的触发条件，下部分表示状态转移时对应的动作。】

**相关参数：**

|  |  |
| --- | --- |
| cwnd | 拥塞控制窗口 |
| MSS | (协商的)1个最大报文段的长度，the maximum segment size |
| ssthresh | 慢启动阈值 |
| dupACKcount | 重复ACK计数标记 |
| timeout | 计数器超时 |

TCP中的状态主要包括3个，分别是：慢启动(slow start)、拥塞避免(congestion avoidance)、快速恢复(fast recovery)。根据状态转移图，启动时首先是慢启动状态（参考虚线箭头）：

* **在慢启动状态下：**

初始时，cwnd为1个报文段(MSS)，ssthresh(慢启动阈值)为64KB，dupACKcount(重复ACK计数标记为0)；

* + 慢启动状态下cwnd的增加规则为：每收到一个新的ACK，cwnd=cwnd+1MSS，即拥塞窗口增加1个最大报文段长度；

【这个规则的另一种描述为：每RTT时间，cwnd呈指数倍变化。例如第1个RTT，cwnd为1个MSS，假设接收方立即返回1个ACK，则第2个RTT开始之前，cwnd变为2个MSS，以此类推；第2个RTT，发送方发出2个MSS，接收方收到并返回2个ACK，则第3个RTT开始之间，cwnd变为4个MSS（说明，收到第1个ACK，cwnd为3，收到第2个ACK，cwnd为4）；这是书上的原文解释】



* + 当cwnd超过慢启动阈值(ssthresh)时，状态就从“慢启动”转为“拥塞避免”状态；
* **在拥塞避免状态下：**
  + cwnd的增加规则为：每收到一个新的ACK，

cwnd = cwnd + MSS\*MSS/cwnd；注意这里不是增加1个最大报文段长度，而是增加（MSS/cwnd）个MSS；

【这个规则的理解就是：每个RTT，cwnd才总共增加1个MSS，图示】



* + 在拥塞避免状态（或慢启动状态），当收到3个重复ACK，进入快速恢复状态，在状态转移的时候，执行的操作是将慢启动阈值设置为当前拥塞控制窗口的一半，即ssthresh = cwnd/2; 将新的拥塞控制窗口设置为新的慢启动阈值加3个MSS，cwnd = ssthresh + 3MSS；重新传输丢失的报文段；

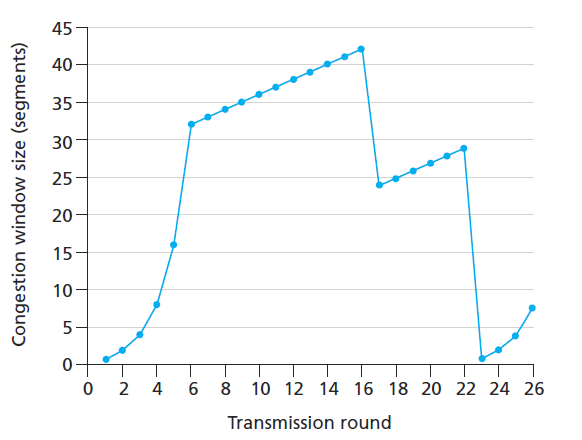
【举例说明：假设收到3个重复ACK时的cwnd为10个MSS，则新的ssthresh=5MSS,新的cwnd=5MSS+3MSS=8MSS】

* **当处于快速恢复状态时比较复杂，有两种不同的操作：**
  + 每收到1个重复的ACK，拥塞控制窗口的变化：cwnd = cwnd + MSS（注意这里是每收到一个重复ACK，就增加1个MSS）；然后传输允许的新的报文段【感觉不是很合理，但是RFC原文是这样写的】；
  + 当收到1个新的ACK时，从快速恢复状态返回拥塞避免状态，cwnd = ssthresh，这里的慢启动阈值是进入快速恢复状态时的计算值，沿用上面的例子，返回“拥塞避免”状态后，cwnd=5MSS；

**因此，很多例题/习题，只用坐标图来描述TCP进入快速恢复状态，而不说明后面到底发生的是哪种情况，这会导致分不清后续的cwnd窗口的变化。**

例题分析：

书上的课后题：（P193 第40题）



在这个题目中，[6,16]是拥塞避免状态，在第16个RTT，cwnd=42 MSS，这个时候收到3个重复ACK，因此第17个RTT时ssthresh=21MSS，cwnd=(21+3)MSS，

**如果按照书上的答案[17, 22]也处于拥塞避免状态，**说明在进入快速恢复恢复状态时，很快就收到了新的ACK确认，这个时候**cwnd=ssthresh=21MSS，然后再根据收到的新的ACK情况修改cwnd的值，因为是“拥塞避免”状态，所以这个cwnd增加最多就约1个MSS，那么第18个RTT开始时的，cwnd窗口到底是多少呢？22？这个就与书上的画法不同。准确的说应该是从cwnd=21开始线性增长。**

**所以在坐标图中，利用cwnd的变化来描述拥塞控制状态变迁时，不应过于粗粒度，每一个RTT都应该仔细考虑。**

RFC关于快速恢复算法的原文：

The "fast recovery" algorithm governs the transmission of new data until a non-duplicate ACK arrives. The reason for not performing slow start is that the receipt of the duplicate ACKs not only indicates that a segment has been lost, but also that segments are most likely leaving the network (although a massive segment duplication by the network can invalidate this conclusion).

In other words, since the receiver can only generate a duplicate ACK when a segment has arrived, that segment has left the network and is in the receiver's buffer, so we know it is no longer consuming network resources. Furthermore, since the ACK "clock" [Jac88] is preserved, the TCP sender can continue to transmit new segments (although transmission must continue using a reduced cwnd, since loss is an indication of congestion).

The fast retransmit and fast recovery algorithms are implemented together as follows.

1. On the first and second duplicate ACKs received at a sender, a TCP SHOULD send a segment of previously unsent data per [[RFC3042](https://tools.ietf.org/html/rfc3042)] provided that the receiver's advertised window allows, the total FlightSize would remain less than or equal to cwnd plus 2\*SMSS, and that new data is available for transmission. Further, the TCP sender MUST NOT change cwnd to reflect these two segments [[RFC3042](https://tools.ietf.org/html/rfc3042)].

Note that a sender using SACK [[RFC2018](https://tools.ietf.org/html/rfc2018)] MUST NOT send new data unless the incoming duplicate acknowledgment contains new SACK information.

2. When the third duplicate ACK is received, a TCP MUST set ssthresh to no more than the value given in equation (4). When [[RFC3042](https://tools.ietf.org/html/rfc3042)] is in use, additional data sent in limited transmit MUST NOT be included in this calculation.

ssthresh = max (FlightSize / 2, 2\*SMSS) (4)

where, as discussed above, FlightSize is the amount of outstanding data in the network.

3. The lost segment starting at SND.UNA MUST be retransmitted and cwnd set to ssthresh plus 3\*SMSS. This artificially "inflates" the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.

4. For each additional duplicate ACK received (after the third), cwnd MUST be incremented by SMSS. **This artificially inflates the congestion window in order to reflect the additional segment that has left the network.** 【这里解释了为什么要在快速恢复阶段增加拥塞控制窗口】

5. When previously unsent data is available and the new value of cwnd and the receiver's advertised window allow, a TCP SHOULD send 1\*SMSS bytes of previously unsent data.

6. When the next ACK arrives that acknowledges previously unacknowledged data, a TCP MUST **set cwnd to ssthresh** (the value set in step 2). This is termed "deflating" the window. 【收到新的ACK确认，则会缩小窗口】

This ACK should be the acknowledgment elicited by the retransmission from step 3, one RTT after the retransmission (though it may arrive sooner in the presence of significant out- of-order delivery of data segments at the receiver).

Additionally, this ACK should acknowledge all the intermediate segments sent between the lost segment and the receipt of the third duplicate ACK, if none of these were lost.