阈值初始等于8个MSS。在前8个传输回合，Tahoe和Reno采取了相同的动作。拥塞窗口在慢启动阶段以指数速度快速爬升，在第4轮传输时到达了阈值。然后拥塞窗口以线性速度爬升，直到在第8轮传输后出现3个冗余ACK。注意到当该丢包事件发生时，拥塞窗口值为12×MSS。于是ssthresh的值被设置为0.5×cwnd=6×MSS。在TCP Reno下，拥塞窗口被设置为cwnd=9MSS，然后线性地增长。在TCP Tahoe下，拥塞窗口被设置为1个MSS，然后呈指数增长，直至到达ssthresh值为止，在这个点它开始线性增长。

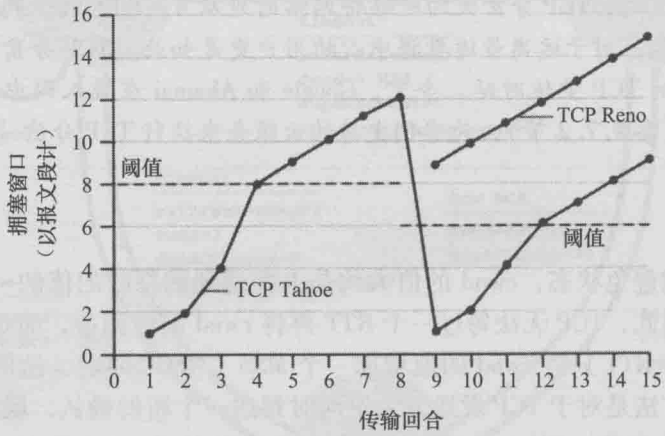

图3-53 TCP拥塞窗口的演化（Tahoe和Reno）

图3-52表示了TCP拥塞控制算法（即慢启动、拥塞避免和快速恢复）的完整FSM描述。该图也指示了新报文段的传输或重传的报文段可能出现的位置。尽管区分TCP差错控制/重传与TCP拥塞控制非常重要，但是注意到TCP这两个方面交织链接的方式也很重要。

4. TCP拥塞控制：回顾

在深入了解慢启动、拥塞避免和快速恢复的细节后，现在有必要退回来回顾一下全局。忽略一条连接开始时初始的慢启动阶段，假定丢包由3个冗余的ACK而不是超时指示，TCP的拥塞控制是：每个RTT内cwnd线性（加性）增加1MSS，然后出现3个冗余ACK事件时cwnd减半（乘性减）。因此，TCP拥塞控制常常被称为**加性增、乘性减**（Additive-Increase，Multiplicative-Decrease，AIMD）拥塞控制方式。AIMD拥塞控制引发了在图3-54中所示的“锯齿”行为，这也很好地图示了我们前面TCP检测带宽时的直觉，即TCP线性地增加它的拥塞窗口长度（因此增加其传输速率），直到出现3个冗余ACK事件。然后以2个因子来减少它的拥塞窗口长度，然后又开始了线性增长，探测是否还有另外的可用带宽。

如前所述，许多TCP实现采用了Reno算法［Padhye 2001］。Reno算法的许多变种已被提出［RFC 3782；RFC 2018］。TCP Vegas算法［Brakmo 1995；Ahn 1995］试图在维持较好的吞吐量的同时避免拥塞。Vegas的基本思想是：①在分组丢失发生之前，在源与目的地之间检测路由器中的拥塞；②当检测出快要发生的分组丢失时，线性地降低发送速率。快要发生的分组丢失是通过观察RTT来预测的。分组的RTT越长，路由器中的拥塞越严重。Linux支持若干拥塞控制算法（包括TCP Reno和TCP Vegas），并且允许系统管理员配置使用哪个版本的TCP。在Linux版本2.6.18中，TCP默认版本设置为CUBIC［Ha 2008］，这是为高带宽应用开发的一个TCP版本。对于许多特色TCP的综述参见［Afanasyev 2010］。