岔，客户向邻近前端连接一条 TCP 连接，并且该前端以非常大的窗口向数据中心维护一条 TCP 连接［Tariq 2008，Pathak 2010，Chen 2011］。使用这种方法，响应时间大致变为 $4 * RTT_{FE} + RTT_{BE}$ + 处理时间，其中 RTT_{FE} 是客户与前端服务器之间的往返时间，RTT_{BE} 是前端服务器与数据中心（后端服务器）之间的往返时间。如果前端服务器邻近客户，则该响应时间大约变为 RTT_{BE} 加上处理时间，因为 RTT_{FE} 小得微不足道并且 RTT_{BE} 约为 RTT。总而言之，TCP 分岔大约能够将网络时延从 4 * RTT 减少到 RTT，极大地改善用户感受的性能，对于远离最近数据中心的用户更是如此。TCP 分岔也有助于减少因接入网丢包引起的 TCP 重传时延。今天，Google 和 Akamai 在接入网中广泛利用了它们的 CDN 服务器（参见 7.2 节），为它们支持的云服务来执行 TCP 分岔［Chen 2011］。

2. 拥塞避免

一旦进入拥塞避免状态，cwnd 的值大约是上次遇到拥塞时的值的一半，即距离拥塞可能并不遥远！因此，TCP 无法每过一个 RTT 再将 cwnd 的值翻番，而是采用了一种较为保守的方法，每个 RTT 只将 cwnd 的值增加一个 MSS［RFC 5681］。这能够以几种方式完成。一种通用的方法是对于 TCP 发送方无论何时到达一个新的确认，就将 cwnd 增加一个 MSS（MSS/cwnd）字节。例如，如果 MSS 是 1460 字节并且 cwnd 是 14 600 字节，则在一个 RTT 内发送 10 个报文段。每个到达 ACK（假定每个报文段一个 ACK）增加 1/10MSS 的拥塞窗口长度，因此在收到对所有 10 个报文段的确认后，拥塞窗口的值将增加了一个 MSS。

但是何时应当结束拥塞避免的线性增长（每 RTT 1MSS）呢？当出现超时时，TCP 的拥塞避免算法行为相同。与慢启动的情况一样，cwnd 的值被设置为 1 个 MSS，当丢包事件出现时，ssthresh 的值被更新为 cwnd 值的一半。然而，前面讲过丢包事件也能由一个三个冗余 ACK 事件触发。在这种情况下，网络继续从发送方向接收方交付报文段（就像由收到冗余 ACK 所指示的那样）。因此 TCP 对这种丢包事件的行为，相比于超时指示的丢包，应当不那么剧烈：TCP 将 cwnd 的值减半（为使测量结果更好，计及已收到的 3 个冗余的 ACK 要加上 3 个 MSS），并且当收到 3 个冗余的 ACK，将 ssthresh 的值记录为 cwnd 的值的一半。接下来进入快速恢复状态。

3. 快速恢复

在快速恢复中，对于引起 TCP 进入快速恢复状态的缺失报文段，对收到的每个冗余的 ACK，cwnd 的值增加一个 MSS。最终，当对丢失报文段的一个 ACK 到达时，TCP 在降低 cwnd 后进入拥塞避免状态。如果出现超时事件，快速恢复在执行如同在慢启动和拥塞避免中相同的动作后，迁移到慢启动状态：当丢包事件出现时，cwnd 的值被设置为 1 个 MSS，并且 ssthresh 的值设置为 cwnd 值的一半。

快速恢复是 TCP 推荐的而非必需的构件［RFC 5681］。有趣的是，一种称为 TCP Tahoe 的 TCP 早期版本，不管是发生超时指示的丢包事件，还是发生 3 个冗余 ACK 指示的丢包事件，都无条件地将其拥塞窗口减至 1 个 MSS，并进入慢启动阶段。TCP 的较新版本 TCP Reno，则综合了快速恢复。

图 3-53 图示了 Reno 版 TCP 与 Tahoe 版 TCP 的拥塞控制窗口的演化情况。在该图中，