传。因此，接收方只需丢弃分组 $n+1$ 即可。这种方法的优点是接收缓存简单，即接收方不需要缓存任何失序分组。因此，虽然发送方必须维护窗口的上下边界及 nextseqnum 在该窗口中的位置，但是接收方需要维护的唯一信息就是下一个按序接收的分组的序号。该值保存在 expectedseqnum 变量中，如图 3-21 中接收方 FSM 所示。当然，丢弃一个正确接收的分组的缺点是随后对该分组的重传也许会丢失或出错，因此甚至需要更多的重传。

图 3-22 给出了窗口长度为 4 个分组的 GBN 协议的运行情况。因为该窗口长度的限制，发送方发送分组 0～3，然后在继续发送之前，必须等待直到一个或多个分组被确认。当接收到每一个连续的 ACK（例如 ACK 0 和 ACK 1）时，该窗口便向前滑动，发送方便可以发送新的分组（分别是分组 4 和分组 5）。在接收方，分组 2 丢失，因此分组 3、4 和 5 被发现是失序分组并被丢弃。

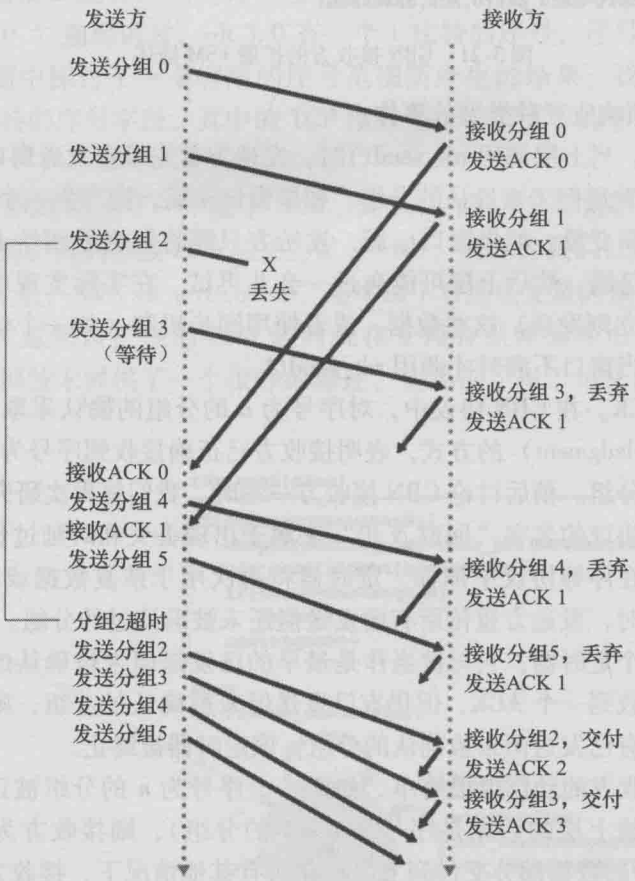

图 3-22 运行中的 GBN

在结束对 GBN 的讨论之前，需要提请注意的是，在协议栈中实现该协议可能与图 3-20 中的扩展 FSM 有相似的结构。该实现也可能是以各种过程形式出现，每个过程实现了在响应各种可能出现的事件时要采取的动作。在这种**基于事件的编程**（event-based programming）方式中，这些过程要么被协议栈中的其他过程调用，要么作为一次中断的结果。在发送方，这些事件包括：①来自上层实体的调用去调用 rdt_send()；②定时器中断；③报文到达时，来自下层的调用去调用 rdt_rcv()。本章后面的编程作业会使你有机会在一个模拟而真实的网络环境中实际实现这些例程。