到的分组是否一次重传。对于停等协议这种简单情况，1 比特序号就足够了，因为它可让接收方知道发送方是否正在重传前一个发送分组（接收到的分组序号与最近收到的分组序号相同），或是一个新分组（序号变化了，用模 2 运算“前向”移动）。因为目前我们假定信道不丢分组，ACK 和 NAK 分组本身不需要指明它们要确认的分组序号。发送方知道所接收到的 ACK 和 NAK 分组（无论是否是含糊不清的）是为响应其最近发送的数据分组而生成的。

图 3-11 和图 3-12 给出了对 rdt 2.1 的 FSM 描述，这是 rdt 2.0 的修订版。rdt 2.1 的发送方和接收方 FSM 的状态数都是以前的两倍。这是因为协议状态此时必须反映出目前（由发送方）正发送的分组或（在接收方）希望接收的分组的序号是 0 还是 1。值得注意的是，发送或期望接收 0 号分组的状态中的动作与发送或期望接收 1 号分组的状态中的动作是相似的；唯一的不同是序号处理的方法不同。

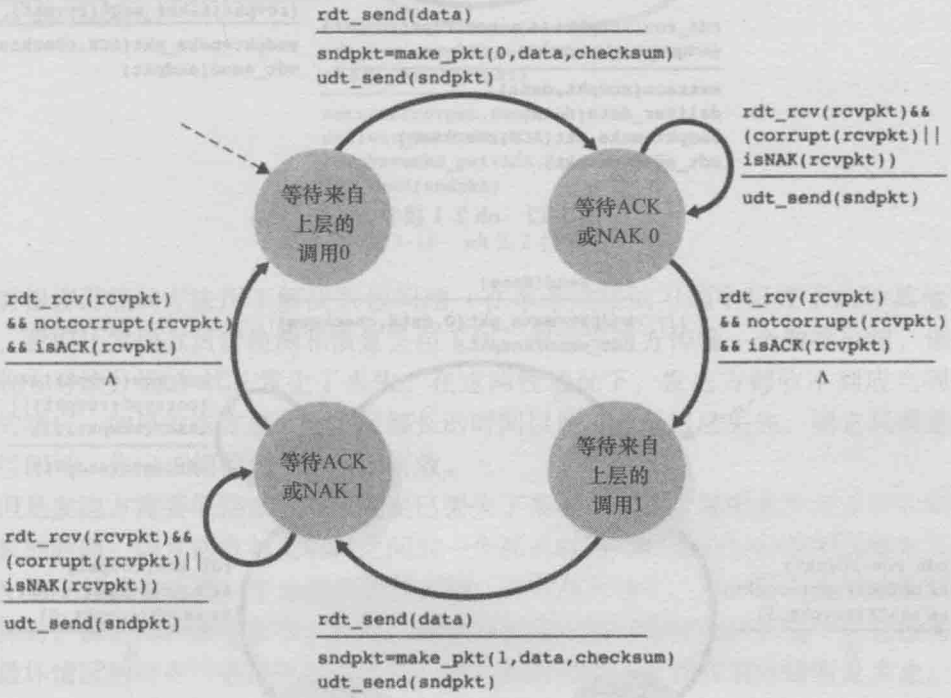

图 3-11 rdt 2.1 发送方

协议 rdt 2.1 使用了从接收方到发送方的肯定确认和否定确认。当接收到失序的分组时，接收方对所接收的分组发送一个肯定确认。如果收到受损的分组，则接收方将发送一个否定确认。如果不发送 NAK，而是对上次正确接收的分组发送一个 ACK，我们也能实现与 NAK 一样的效果。发送方接收到对同一个分组的两个 ACK（即接收**冗余 ACK**（duplicate ACK））后，就知道接收方没有正确接收到跟在被确认两次的分组后面的分组。rdt 2.2 是在有比特差错信道上实现的一个无 NAK 的可靠数据传输协议，如图 3-13 和图 3-14 所示。rdt2.1 和 rdt2.2 之间的细微变化在于，接收方此时必须包括由一个 ACK 报文所确认的分组序号（这可以通过在接收方 FSM 中，在 make_pkt() 中包括参数 ACK 0 或 ACK 1 来实现），发送方此时必须检查接收到的 ACK 报文中被确认的分组序号（这可通过在发送方 FSM 中，在 isACK() 中包括参数 0 或 1 来实现）。