rdt 2.0 的发送端有两个状态。在最左边的状态中，发送端协议正等待来自上层传下来的数据。当产生 rdt_send(data)事件时，发送方将产生一个包含待发送数据的分组（sndpkt），带有检验和（例如，就像在 3.3.2 节讨论的对 UDP 报文段使用的方法），然后经由 udt_send(sndpkt)操作发送该分组。在最右边的状态中，发送方协议等待来自接收方的 ACK 或 NAK 分组。如果收到一个 ACK 分组（图 3-10 中符号 rdt_rcv(rcvpkt)&& isACK(rcvpkt)对应该事件），则发送方知道最近发送的分组已被正确接收，因此协议返回到等待来自上层的数据的状态。如果收到一个 NAK 分组，该协议重传最后一个分组并等待接收方为响应重传分组而回送的 ACK 和 NAK。注意到下列事实很重要：当发送方处于等待 ACK 或 NAK 的状态时，它不能从上层获得更多的数据；这就是说，rdt_send()事件不可能出现；仅当接收到 ACK 并离开该状态时才能发生这样的事件。因此，发送方将不会发送一块新数据，除非发送方确信接收方已正确接收当前分组。由于这种行为，rdt 2.0 这样的协议被称为**停等**（stop-and-wait）协议。

rdt 2.0 接收方的 FSM 仍然只有一个状态。当分组到达时，接收方要么回答一个 ACK，要么回答一个 NAK，这取决于收到的分组是否受损。在图 3-10 中，符号 rdt_rcv(rcvpkt) && corrupt(rcvpkt)对应于收到一个分组并发现有错的事件。

rdt 2.0 协议看起来似乎可以运行了，但遗憾的是，它存在一个致命的缺陷。尤其是我们没有考虑到 ACK 或 NAK 分组受损的可能性！（在继续研究之前，你应该考虑怎样解决该问题。）遗憾的是，我们细小的疏忽并非像它看起来那么无关紧要。至少，我们需要在 ACK/NAK 分组中添加检验和比特以检测这样的差错。更难的问题是协议应该怎样纠正 ACK 或 NAK 分组中的差错。这里的难点在于，如果一个 ACK 或 NAK 分组受损，发送方无法知道接收方是否正确接收了上一块发送的数据。

考虑处理受损 ACK 和 NAK 时的 3 种可能性：

- 对于第一种可能性，考虑在口述报文情况下人可能的做法。如果说话者不理解来自接收方回答的“OK”或“请重复一遍”，说话者将可能问“你说什么?”（因此在我们的协议中引入了一种新型发送方到接收方的分组）。接收方则将复述其回答。但是如果说话者的“你说什么?”产生了差错，情况又会怎样呢？接收者不明白那句混淆的话是口述内容的一部分还是一个要求重复上次回答的请求，很可能回一句“你说什么?”。于是，该回答可能含糊不清了。显然，我们走上了一条困难重重之路。
- 第二种可能性是增加足够的检验和比特，使发送方不仅可以检测差错，还可恢复差错。对于会产生差错但不丢失分组的信道，这就可以直接解决问题。
- 第三种方法是，当发送方收到含糊不清的 ACK 或 NAK 分组时，只需重传当前数据分组即可。然而，这种方法在发送方到接收方的信道中引入了**冗余分组**（duplicate packet）。冗余分组的根本困难在于接收方不知道它上次所发送的 ACK 或 NAK 是否被发送方正确地收到。因此它无法事先知道接收到的分组是新的还是一次重传！

解决这个新问题的一个简单方法（几乎所有现有的数据传输协议中，包括 TCP，都采用了这种方法）是在数据分组中添加一新字段，让发送方对其数据分组编号，即将发送数据分组的**序号**（sequence number）放在该字段。于是，接收方只需要检查序号即可确定收