3.

1）

0 1 0 1 0 0 1 1

+ 0 1 1 0 0 1 1 0

1 0 1 1 1 0 0 1

1 0 1 1 1 0 0 1

+ 0 1 1 1 0 1 0 0

1 0 0 1 0 1 1 0 1

1 0 0 1 0 1 1 0 1 + 1 = 1 0 0 1 0 1 1 1 0

溢出，需要取反码。

反码为1 1 0 1 0 0 0 1

2）

使用反码有以下好处：

1.不依赖系统是大端还是小端

2.计算检验和比较简单快速

3）

接收方检验差错的方法是将三个字节与检验和相加，如果任何一个位为 0，说明出错

4）

1 比特的差错肯定会导致结果不同

2 比特的差错可能会检测不出，比如题中第一、二字节变为 01010010，01100111，即最后一个比特反转，检验和最后不会出现0。

11.

如果从“等待来自下层的 1”中删除，不会影响正常工作，因为 sndpkt 已经被生成了。

但是如果从“等待来自下层的 0”中删除，而且接收方刚刚启动（处于初始状态），sndpkt 是一个错误的值，那么发送方会认为 ACK 损坏并重发分组，接收方会继续发送错误值，这将导致一个死锁。

14.

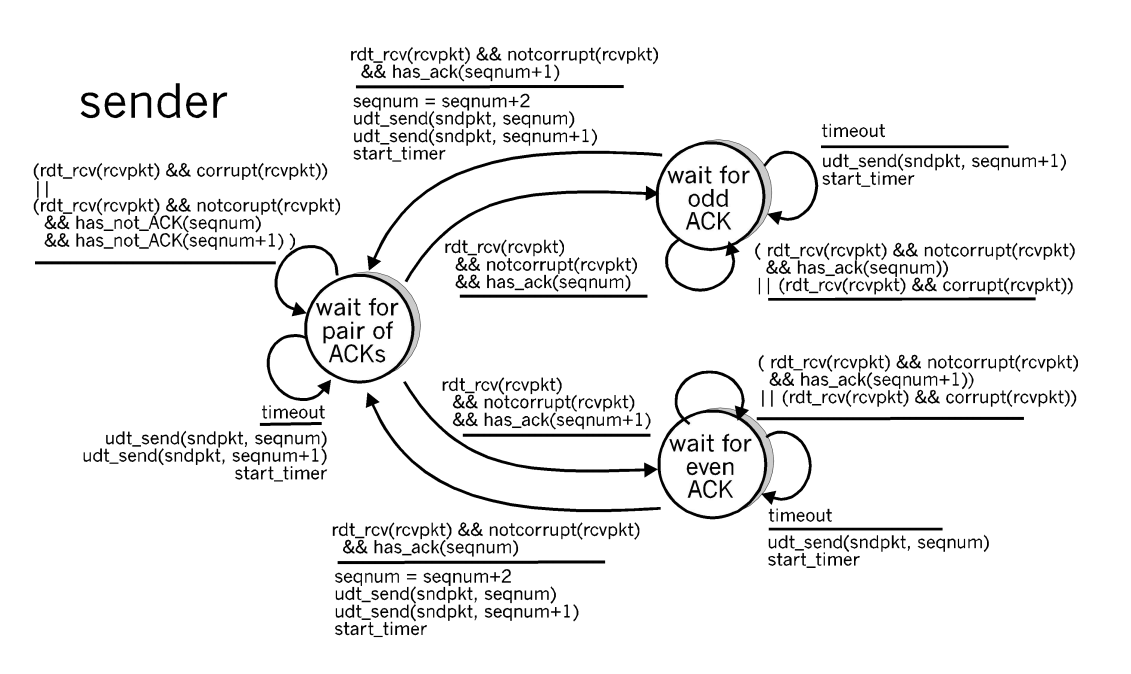
在一个仅有NAK的协议中，只有当接收方收到数据包x+1时，才会发现数据包x的丢失。也就是说，接收者先收到x-1，然后再收到x+1，只有当x+1被收到时，接收者才意识到x被遗漏了。如果x的传输和x+l的传输之间有很长的延迟，那么在仅有NAK的协议下，x将有很长的时间可以被恢复。

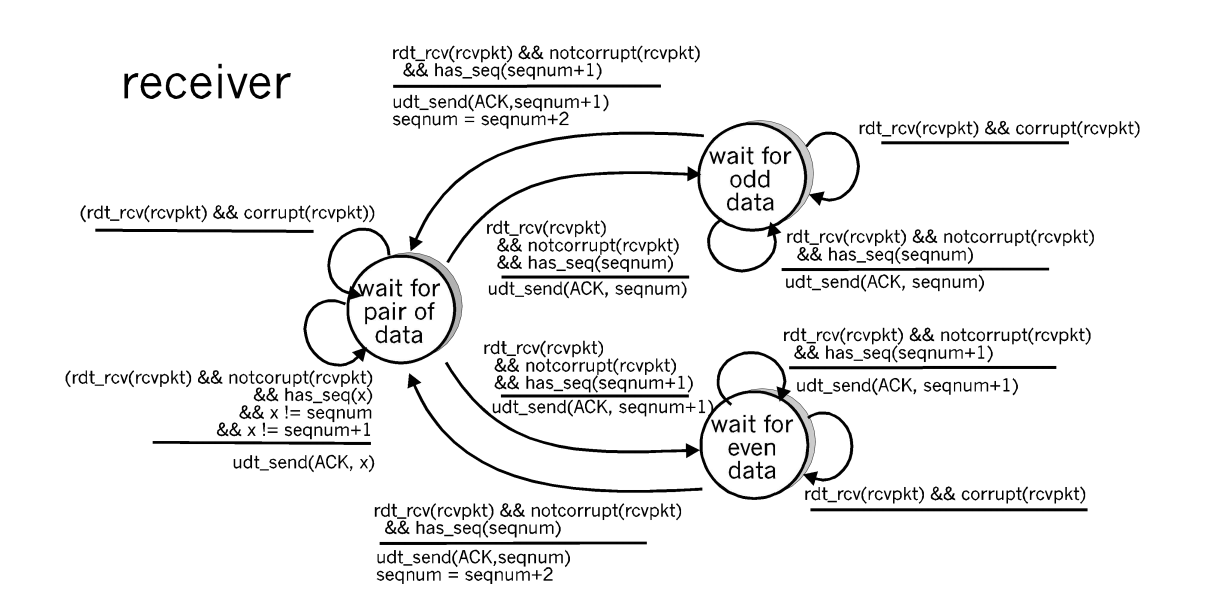
如果发送方只很少发送数据，仅使用 NAK 的协议相比使用 ACK 的协议更可取。因为在这种情况下，由于数据传输量较小，可能出现的错误应答也较少，使用 NAK 可以减少不必要的协议开销和网络带宽占用。

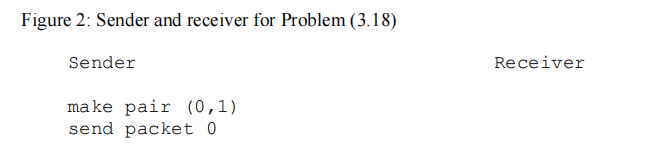
当发送方需要发送大量数据时，端到端连接很少丢失，仅使用 NAK 的协议就不如使用 ACK 的协议更可取了。这是因为在这种情况下，由于数据量大，发生错误的概率也会增加。如果仅使用 NAK 的协议，会导致发送方需重复发送相同的数据包，造成数据传输效率低下。使用 ACK 的协议，尽管每个数据包都需要发出确认响应，但可以保证数据传输的正确性和效率。

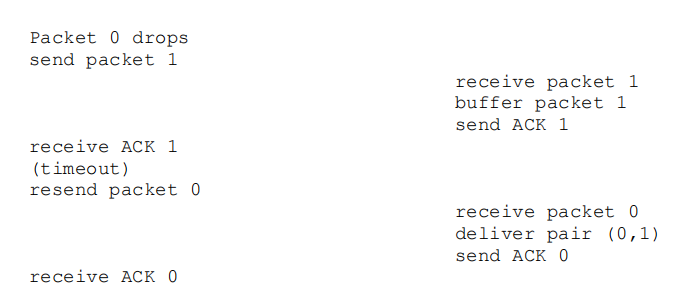
18.

在我们的解决方案中，发送方将等待收到一对消息（seqnum和seqnum+1）的ACK，然后再继续转移到下一对消息。数据包有一个数据字段，并携带一个两位的序列号。即，有效的序列号分别是0、1、2和3。（注意：您应该考虑为什么只有0,1的1位序列号空间在下面的解决方案中不起作用。）ACK消息携带着它们所承认的数据包的序列号。发送方和接收方的FSM如图2所示。注意，发送方状态记录是否(i)没有接收到当前对的ACK，（ii）接收了seqnum（仅）的ACK，或接收了seqnum+1（仅）的ACK。在这个图中，我们假设seqnum最初为0，并且发送方已经发送了前两条数据消息（为了让事情继续进行）。从丢失的数据包中恢复的时间线跟踪如下所示：









21.

由于 A-to-B 通道可能会丢失请求消息，因此 A 需要超时并重新传输其请求消息（以便能够从丢失中恢复）。由于信道延迟是可变的和未知的，因此 A 可能会发送重复的请求（即，重新发送 B 已经接收到的请求消息）。为了能够检测重复的请求消息，协议将使用序列号。1 位序列号足以满足停止和等待类型的请求/响应协议。

A（请求者）有 4 种状态：

1. “ 等待来自上面的请求 0。”在这里，请求者正在等待来自上面的调用以请求一个数据单元。当它接收到来自上面的请求时，它向 B 发送一条请求消息 R0，启动一个计时器并转换到“等待D0”状态。当处于“等待来自上面的请求 0”状态时，A 将忽略从 B 接收到的任何内容。
2. “ 等待 D0”。在这里，请求者正在等待来自 B 的 D0 数据消息。计时器始终在此状态下运行。如果计时器过期，A 将发送另一条 R0 消息，重新启动计时器并保持此状态。如果从 B 接收到 D0消息，a 将停止时间并过渡到“等待来自上述的请求 1”状态。如果 A 在此状态下收到 D1 数据消息，则会忽略该消息。
3. “ 等待来自上面的请求 1。”在这里，请求者再次等待来自上面的调用以请求一个数据单元。当它接收到来自上面的请求时，它向 B 发送一条请求消息 R1，启动一个计时器并转换到“等待D1”状态。当处于“等待来自上面的请求 1”状态时，A 忽略从 B 接收到的任何内容。
4. “ 等待 D1”。在这里，请求者正在等待来自 B 的 D1 数据消息。计时器始终在此状态下运行。如果计时器过期，A 将发送另一条 R1 消息，重新启动计时器并保持此状态。如果从 B 接收到 D1消息，a 将停止计时器并过渡到“等待来自上面的请求 0”状态。如果在该状态下接收到 D0 数据消息，则忽略该消息。

数据供应商（B）只有两种状态：

1. “发送 D0。”在此状态下，B 通过发送 D0 继续响应接收到的 R0 消息，然后保持在此状态。如果 B 收到 R1 消息，则它知道其 D0 消息已正确接收。因此，它将丢弃该 D0 数据（因为它已在另一端接收到），然后转换到“发送 D1”状态，在该状态下，它将使用 D1 发送下一条请求的数据。
2. “发送 D1。”在此状态下，B 通过发送 D1 继续响应接收到的 R1 消息，然后保持此状态。如果B 接收到 R1 消息，则它知道其 D1 消息已被正确接收，并因此过渡到“发送 D1”状态。

22.

a. 考虑两种极端情况：

发送方发送 k-4，k-3，k-2，k-1，接收方都完整得接收并发送 ACK，但 ACK 全都未传到发送方，接收方的期待序号为 k，而发送方窗口序号为 [k-4, k-1]。如果 ACK 全都传回，则发送方更新 base，其序号为 [k, k+3]。因此序号可能是 [k-4, k+3]

b. 如果接收方期待 k，则它一定将比 k-1 小的 ACK 发送出去了，如果要使发送方发送 k-1，那么它至少已经接收到了 k-5 的 ACK。因此正在传播回发送方的 ACK 序号可能是 [k-4, k-1]

23.

设序号为0（第一个0），1，...，k-1，0（第二个0）对于SR，要使其序号发生混杂，至少是当接收方刚刚包含第二个0，即接收方窗口为[k-N-1,0]，也就是说k-N(包括k-N)之前的都接收）过了。要使序号混杂还有一个条件就是第一个0在发送方窗口且恰好其ACK丢失，需要重发。0(第一个0)~k-N为k-N+1个值，如果窗口长度不足k-N+1，则第一个0和第二个0不会同时包含在发送或接收窗口中。所以窗口长度N≤k-N，即N≤k/2。GBN同理，得出结果N≤k。

26.

1. 因为TCP是字节流编号的，所以L的最大值为232byte
2. 设报文数为N。。

总头部长度为N × 66byte=528857934 byte

总字节数为byte + 218857934byte = 4.824 × 109 byte

可得

36.

考虑当分组错序到达会发生什么，分组 1 正确到达，目的地发送 ACK，分组 3 提前于分组 2 到达，发送一个冗余 ACK，这将导致发送方重传分组 2，即使分组 2 并未丢失。

37.

a.

GBN：

A：首先发送分组 12345，后来重发 2345，总共 9 个分组

B：首先发送 ACK 1111，后来发送 2345，总共 5 个 ACK

SR：

A：首先发送分组 12345，后来重发 2，总共 6 个分组

B：首先发送 ACK 1345，后来发送 2，总共 5 个 ACK

TCP：

A：首先发送分组 12345，后来重发 2，总共 6 个分组

B：首先发送 ACK 2222，后来发送 6，总共 5 个 ACK

b.

GBN、SR 需要等待超时，而 TCP 使用快速重传，故 TCP 最快