**第三次作业**

**P5.**

答：

假设发送方处于 "Wait for call 1 from above " 的状态，接收方（本题中显示的接收方）处于 "Wait for call 1 from below "的状态。发送方发送了一个序列号为1的数据包，并过渡到 "Wait for ACK或NAK 1"，等待ACK或NAK。假设现在接收方正确地接收了序列号为1的数据包，发送了一个ACK，并转换到 "Wait for 0 from below"状态，等待序列号为0的数据包，然而，ACK被破坏了。当rdt2.1发送方收到损坏的ACK时，它重新发送序列号为1的数据包。然而，接收器正在等待一个序列号为0的数据包，并且（如本题所示）当它没有得到一个序列号为0的数据包时，总是发送一个NAK。两者都不会从这个状态向前发展。

**P6.**

答：

发送方之所以需要序列号，是因为其便于接收方能够知道一个数据包是否是已经收到的数据包的重复部分。在ACK的情况下，发送方不需要这种信息（即ACK上的序列号）来告诉检测重复的ACK。重复的ACK对rdt3.0的接收者来说是显而易见的，因为当它收到原始ACK时，它已经过渡到下一个状态。重复的ACK不是发送方需要的ACK，因此被rdt3.0发送方忽略了。

**P7.**

答：

rdt3.0协议的发送方与2.2协议的发送方不同的是，增加了**超时**。我们已经看到，超时的引入在发送方到接收方的数据流中增加了重复数据包的可能性。然而，协议rdt.2.2中的接收方已经可以处理重复的数据包。(如果接收方发送的ACK丢失了，而发送方又重新发送了旧的数据，就会出现rdt2.2中接收方的重复数据）。因此，**rdt2.2协议中的接收方也将作为rdt3.0协议中的接收方进行工作。**

**P8.**

答：

**不可以，接收器不能绝对确定没有发生比特错误**。这是因为数据包的校验和的计算方式。如果数据包中两个16位字的对应位（将被加在一起）是0和1，那么即使这些位分别被翻转为1和0，其总和仍然是相同的。因此，接收器计算的1的补数也将是相同的。这意味着即使有传输错误，校验和也会被验证。

**P9.**

答：

假设该协议已经运行了一段时间。发送方处于 "Wait for call from above "的状态，接收方处于 "Wait for 0 from below"的状态。损坏的数据和损坏的ACK的情况分别如图1(a)、(b)所示。



**图1(a) 数据损坏情况图**



**图1(b) ACK损坏情况图**

**P10.**

答：

在这里，添加一个定时器，其值大于已知的往返传播延迟。我们在 "等待ACK或NAK0 "和 "等待ACK或NAK1 "状态中加入一个超时事件。如果超时事件发生，最近传输的数据包将被重传。让我们看看为什么这个协议在rdt2.1接收器上仍能工作。

1. 假设超时是由一个丢失的数据包引起的，即发送方到接收方通道上的一个数据包。在这种情况下，接收方从未收到之前的传输，从接收方的角度来看，如果收到超时重传，看起来与收到原始传输完全一样。
2. 假设现在有一个ACK丢失。接收者最终会在超时的情况下重传该数据包。但是，重传的动作与ACK被乱码时的动作完全相同。因此，发送方对丢失的反应和对乱码ACK的反应是一样的。rdt 2.1的接收方已经可以处理乱码ACK的情况了。

**P11.**

答：如图2.所示



**图2.**

**P12.**

答：

**该协议仍然有效，**因为如果收到的有错误的数据包真的丢失了，就会发生重传（从接收者的角度看，它永远不知道这些事件中的哪一个，如果有的话，会发生）。

为了了解这个问题背后更微妙的问题，我们必须允许过早超时的发生。在这种情况下，如果数据包的每个额外副本都被ACK，并且每个收到的额外ACK都会导致当前数据包的另一个额外副本被发送，那么数据包n的发送次数将随着n接近无穷而无限制地增加。

**P13.**

答：

在仅有NAK协议中，只有在收到*x+1*数据包时，接收方才会发现数据包x的丢失。也就是说，接收者先收到*x-1*，然后再收到*x+1*，只有在收到*x+1*时，接收者才意识到*x*被遗漏了。如果*x*的传输和*x+1*的传输之间有很长的延迟，那么在只有NAK的协议下，将需要很长的时间才能恢复*x*。

另一方面，如果数据经常被发送，那么在仅有NAK的方案下，恢复可能很快发生。此外，如果错误不经常发生，那么NAK只是偶尔发送（在需要的时候），而ACK永远不会发送--在只发送NAK的情况下，反馈比只发送ACK的情况下大大减少。