1 简单协议 —— 乌托邦式的单工协议

乌托邦式的单工协议是一个简单的无连接协议，它既没有流量控制也没有差错控制。我们假设接收方能够立即处理它所收到的任何分组。换言之，接收方永远不会被接收到的分组淹没。它非常简单。发送方一个接一个地发送分组，甚至不用考虑接收方能否承受。

发送方的传输层从发送方的应用层接收到报文，从中建立一个分组并发送它。接收方的传输层从网络层接收到这个分组，从分组中提取报文并传递到应用层。发送方和接收方的传输层都为应用层提供传输服务。

2 有错停等协议

有错停等协议，它使用了流量和差错控制。发送方和接收方都使用大小为 1 的滑动窗口。发送方在某一时刻发送一个分组，并且在发送下一个分组之前等待确认。为了发现被破坏分组，我们需要在每个数据分组中加入校验和。当一个分组到达接收端时，它就被检测。如果校验和不正确，分组就是被破坏的并被悄悄地丢弃。接收方的沉默对发送方来说是一种信号，即那个分组不是被破坏就是丢失了。每当发送方发送一个分组时，它都开启一个计时器。如果在计时器超时之前接收到确认，那么计时器就被关闭并且发送下一个分组（如果它有待发送分组）。如果计时器超时，发送方就认为分组丢失或被破坏，于是重发之前的分组。这意味着在确认到来之前，发送方都需要存储分组的副本。

协议使用序号和确认号来防止重复分组。一个字段被加入分组头部来保存那个分组的序号。一件需要着重考虑的事情就是序号的范围。由于想使分组大小最小化，所以我们寻找能提供无歧义通信的最小的序号范围。让我们来讨论一下所需要的序号范围。假设我们使用 x 作为序号；我们只需要在之后使用 x + 1，不需要 x + 2。

为了表示这种情况，假设发送端已经发送了带有序号 x 的分组。可能发生三件事：

1．分组安全完整地到达接收端；接收方发送一个确认。确认到达发送端，使发送端发送下一个序号为 x + 1 的分组。

2．分组被破坏或未到达接收端；发送方在超时后重新发送分组（序号 x）。接收方返回一个确认。

3．分组安全完整到达接收端；接收方发送一个确认，但是确认被破坏或丢失了。发送方在超时后重传分组（序号 x）。注意，这里分组是重复的。接收方可以认出这个事实，因为它等

待分组

x + 1，但是收到了分组 x。

我们可以看到，由于接收方需要区分情况 1 和 3，因此需要序号 x 和 x + 1。但是不需要一个编号为 x + 2 的分组。在情况 1 中，分组可以再次被编号 x，由于分组 x 和 x + 1 被确认，两端都不会产生歧义。在情况 2 和 3 中，新的分组是 x + 1 而不是 x + 2。如果仅仅需要 x 和 x + 1，我们可以令 x = 0 且 x + 1 = 1。这意味着序号是 0、1、0、1、0，等等。这称为模 2 运算。

确认号

由于序号必须适合于数据分组和确认，因此我们使用这种惯例：确认号总声明接收方预期接收的下一个分组（next packet expected）序号。例如，如果 0 号分组已经安全完整到达，接收方发送一个确认号为 1 的 ACK（意味着 1 号分组是预期接收的下一个分组）。如果 1 号分组已经安全完整到达，接收方发送一个确认号为 0 的 ACK（意味着 0 号分组是预期接收的下一个分组）。

3 回退 N 帧协议

为了提高传输效率，当发送端等待确认时，必须传输多个分组。换言之，当发送端等待确认时，我们需要让不止一个分组处于未完成状态，以此确保信道忙碌。为此有两种协议。第一个协议称为回退 N 帧协议（Go-Back-N，GBN）。回退 N 帧的关键是我们在接收到确认之前，可以发送多个分组，但是接收端只能缓冲一个分组。我们保存被发送分组的副本直到确认到达。注意，很多数据分组以及确认可以同时处于信道中。

序号

如前所述，序号是模 2m 的，这里 m 是序号字段的大小，单位是比特（位）。

确认号

这个协议中的确认号是累积的，并且定义了预期接收的下一个分组序号。例如，如果确认号（ackNo）是 7，这意味着序号在 6 以内的分组都已经安全完整到达，并且接收方等待序号为 7 的分组。

在回退 N 帧协议中，确认号是累积的并且定义了预期接收的下一个分组序号。

发送窗口

发送窗口是一个想象的盒子，它覆盖了处于运送途中的以及可以被发送的数据分组序号。在每个窗口位置，某些序号定义了已经被发送的分组；其他序号定义了可以被发送的分组。

窗口最大为 2m - 1。协议的窗口大小可以变化。

在任何时候，发送窗口都可能将序号分成四部分。第一部分，窗口左侧，定义了已经确认的分组的序号。发送方不需要担心这些分组并且不需要保存它们的副本。第二部分，定义了已经被发送的分组的序号，但是这些分组状态未知。发送方需要等待，从而发现这些分组究竟是已经被接收还是丢失。我们把这些分组称为未完成（outstanding）分组。第三部分，定义了可以发送的分组的序号；然而，相应数据还没有从应用层接收到。最后，第四部分，窗口右侧，定义了直到窗口滑动前都不能使用的序号。

接收窗口

接收窗口确保正确的数据分组被接收，并且确保正确的确认被发送。在回退 N 帧中，接收窗口的大小总是 1。接收方总是寻找特定分组是否到达。任何失序分组到达都会被丢弃并需要被重发。注意，我们只需要一个变量，即 Rn（接收窗口，预期接收的下一个分组），来定义这种抽象窗口。窗口左侧的序号属于已经被接收和确认的分组；窗口右侧的序号定义了不能被接收的分组。任何序号在这两区域中的分组都被丢弃。只有序号符合 Rn 值的分组才能被接收和确认。接收窗口也滑动，但是一次只滑动一个槽。当正确的分组被接收时，窗口滑动 Rn =（Rn + 1）modulo 2m。

4 选择性回传协议

回退 N 帧协议简化了接收方的进程。接收方只记录一个变量，没有必要缓冲失序分组；它们被简单地丢弃。然而，如果下层网络层丢失很多分组，那么这个协议是低效的。每当一个分组丢失或被破坏，发送方要重新发送所有未完成分组，即使有些失序分组已经被安全完整地接收了。如果网络层由于网络拥塞，丢失了很多分组，那么重发所有这些未完成分组将会使得拥塞更严重，最终更多的分组丢失。这具有雪崩效应，可能导致网络全部瘫痪。另一个协议，称为选择性回传协议（Selective-Repeat（SR）protocol），已经被设计出来，正如其名字所示，只是选择性重发分组，即那些确实丢失的分组。

选择性回传协议也使用两个窗口：一个发送窗口和一个接收窗口。然而，这些窗口与回退 N 帧中的不同。首先，发送窗口的最大值更小；它是 2m -1。

选择性回传协议接收窗口与回退 N 帧中的接收窗口完全不同。接收窗口的大小和发送窗口等大（最大 2m - 1）。选择性回传协议允许和接收窗口一样多的分组失序到来并被存储，直到有一组连续分组被传递到应用层。因为发送窗口和接收窗口的大小是相同的，在发送窗口的所有分组可以失序到达并被存储，直到它们可以被传递。然而，我们需要强调的是，在

可靠协议中，接收方从不向应用层传递失序分组。

计时器

理论上讲，选择性重复为每个未完成分组使用一个计时器。当一个计时器终止，只有一个相应分组被发送。换言之，GBN（回退 N 帧）将未完成分组看做一个组；SR（选择性重复）将它们单独处理。然而，绝大多数实现了 SR 的传输层协议只使用一个计时器。出于这个原因，我们只使用一个计时器。

确认

这两个协议之间还有一点不同。在 GBN 中 ackNo 是累积的；它定义了下一个预期分组的序号，确认了之前的分组都安全完整到达。在 SR 中确认的语义是不同的。在 SR 中，ackNo 定义了被安全完整接收的一个分组；对其他分组没有反馈信息。

窗口大小

为什么发送窗口和接收窗口最多为 2m 的一半。例如，我们选择 m = 2，这意味着窗口大小为 2m/2 或 2(m (1) = 2。如果窗口大小为 2，并且所有确认丢失，那么分组 0 的计时器超时且分组 0 被重发，因此这个重复分组被正确地丢弃（序号 0 不在窗口内）。当窗口大小为 3，并且所有确认丢失，那么发送方发送分组 0 的副本。然而，这次，接收方窗口期待接收分组 0（0 是窗口的一部分），因此它接收了分组 0，并且不把它看做一个重复分组，但是作为下一个循环中的分组。这明显是一个错误。因此在选择性重复中，发送方和接收方窗口的大小最多为 2m 的一半。