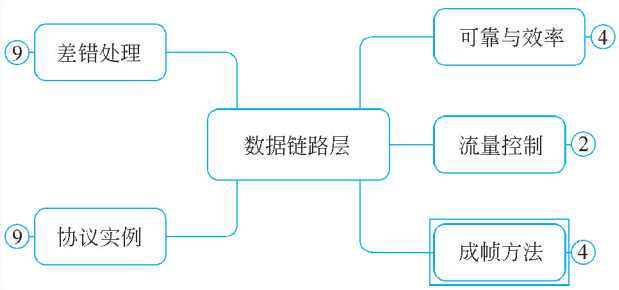
第 3 章

数据链路层

数据链路层(Data Link Layer,DLL)位于物理层之上。相对于物理层主要负责 0 和 1 在通信信道上的表示和传输,数据链路层主要解决的问题是如何在**相邻的两台机器间实现可靠的数据帧通信**。所谓“相邻”并非指两台并排放置的机器,而是指两台机器通过一条通信信道相互连接。例如,通过点到点线路直连的两台机器;或者在总线拓扑结构中,通过共享的信道相连的多台机器。**“数据帧”**则是数据链路层**协议数据单元**(PDU)的专用名称,它被用来描述在两个数据链路层实体之间传输的信息块。实现“可靠”的通信,是很多网络数据传输应用中极其重要的刚性需求。可惜“可靠”并非唾手可得,因为信号的衰减、畸变等在真实的通信信道中普遍存在,也因为传输的时延和噪声的干扰不可避免,这些因素都让我们的网络变得不那么“可靠”。因此,我们需要设计一系列协议和算法处理传输中的错误。有了这些协议和算法,对于数据链路层来说,才可能为其之上的网络层提供“可靠”的通信服务。

本章将首先介绍数据链路层的主要功能;在此之后,将介绍该层的协议数据单元(数据帧),如何将物理层传输的二进制流转换为一个个数据帧,即成帧的方法;然后,将围绕通信中的错误介绍检测错误和纠正错误的方法;之后将解释如何在可能出错的信道上,实现可靠的数据帧传输,引入肯定确认与重传技术、窗口技术、回退 n 帧技术、选择性重传技术等;最后,将介绍一些数据链路层协议的实例。

图 3-1 中的数字代表其下的主要知识点个数;读者可扫描二维码查看本章全部知识点的组织思维导图,并根据需要收起和展开。



第3章 思维导图

图 3-1 本章主要内容框架的思维导图上层

3.1　数据链路层的主要功能

在网络分层模型中，数据链路层的最终目的是为网络层提供服务，而为了实现这些服务，特别是当这些服务对数据传输的可靠性有所要求时，就要求数据链路层实现如下功能。

（1）成帧：把从物理层获得的二进制流拆分为分离的数据帧。

（2）错误处理：检测或纠正传输中发生的错误，以及处理错误。

（3）流量控制：协调发送方与接收方的数据收发速度，以免数据发送太快，超过接收方的处理能力，而导致接收方缓存溢出被“淹没”；或者是发送太慢，而导致互相等待浪费时间。

在实现上述功能的基础上，数据链路层为网络层提供三类服务。

（1）无确认的无连接服务。

（2）有确认的无连接服务。

（3）有确认的有连接服务。

确认(acknowledgment)是指接收方在完成一次成功的数据帧接收后，回复信息给发送方，以表示完成了该数据帧的接收。**有连接**则是指在通信的过程中，需要在发送方与接收方之间建立并保持一条物理的或虚拟的链路，一直到数据传输结束之后，这个连接才被释放。

对于无确认的无连接服务，发送方与接收方之间不需要建立或保持连接，也不需要接收方对接收到的数据帧进行确认。这就意味着，数据链路层不会对数据帧的丢失或错误进行任何处理。无确认的无连接服务虽然无法实现可靠的数据传输，但是其实现简单且快捷。

对于有确认的无连接服务，同样不要求在发送方和接收方之间建立连接，但要求接收方对已正确接收的数据帧回复确认，类似于我们在日常生活中签收快递。这样可以使发送方了解每个数据帧是否正确到达目的地，从而决定是否重发该数据帧。相比无确认的无连接服务，有确认的无连接服务虽然在过程中更复杂，回复和等待确认也使通信效率更低，但却能实现可靠的数据通信。

对于有确认的有连接服务，则完全采取不一样的策略。在发送数据之前，需要在发送方与接收方之间建立一个连接，发送方发出的每个数据帧都会被编号，并且沿着同一个连接发送给接收方，接收方会对已收到的数据帧回复确认，这样数据链路层可以保证这些数据帧按照被发送的顺序正确地被接收并交给接收方的网络层。这个连接可以被理解为一个在发送方与接收方之间的管道，为双方网络层实体之间提供可靠的数据帧通信。

3.2　成帧的方法

前面章节已经反复提到了“数据帧”这个名称，本节就来详细介绍数据帧的概念以及如何生成数据帧。从发送方的角度来看，当数据链路层从网络层获得数据分组时，数据链路层的进程会将这些数据分组进行封装，加上帧头和帧尾，得到一个完整的数

据帧。数据链路层的一些协议会定义这些帧头和帧尾的格式以及对应的含义。数据帧以二进制的形式表示，并交给物理层进行传输，当数据被接收方收到时，接收方的数据链路层又要负责将从物理层收到的二进制流进行拆分，恢复成原本的数据帧。数据帧的封装和拆分生成的过程如图 3-2 所示。

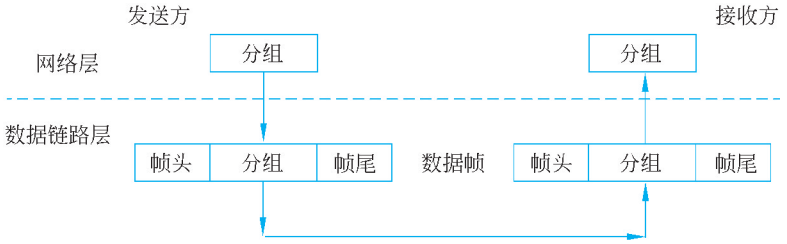


图 3-2 数据帧的封装和拆分生成的过程

在接收方，之所以要将物理层传输的二进制流拆分为离散的数据帧，主要是为了处理错误。数据帧是数据链路层的协议数据单元，是错误处理的单位；每收到一个数据帧就进行错误的检测和处理，稍后会详细介绍检错方法和纠错方法（参考 3.3 节）。

将二进制流拆分为离散的数据帧，看起来是一件简单的事，但实际并不简单。比如发送方发送时在数据帧与数据帧之间插入时间间隔作为一个数据帧开始和结束的标志，并不是一个可行方法，因为在传输过程中，可能存在时钟漂移，经过一段时间的传输，这些间隔可能已经消失，接收方根据漂移了的间隔来分隔数据帧，可能偏离原来数据帧的样子，由此造成错误。数据链路层用来成帧的方法主要有字节计数法、带字节填充的字节标志法、带位填充的位标志法和物理层编码违例法 4 种。

3.2.1 字节计数法

字节计数法利用在帧头的一个字段记录该帧包含的字符数，如图 3-3 所示。接收方读到第一个字段时，便可知该数据帧在哪里结束，也因此知道了下一数据帧从哪里开始。如果一切顺利，则二进制流中的每个数据帧都会被正确地识别出来，如图 3-3(a)所示。

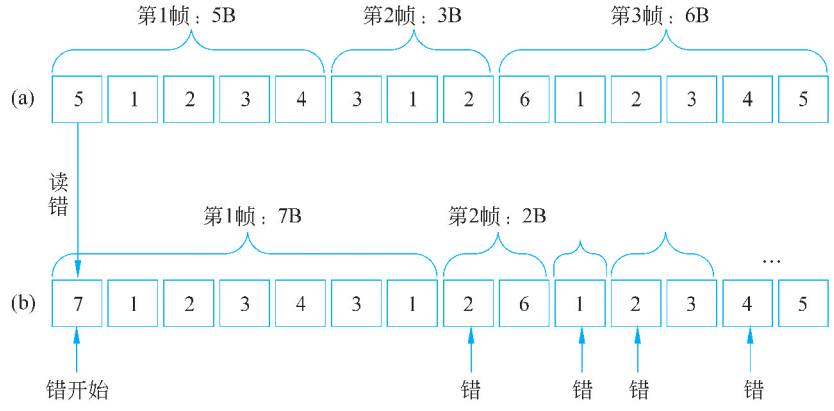


图 3-3 字节计数法示意图

但传输中的错误在所难免。一旦包含帧长度的字段出错，则后续所有的数据帧将被错误分隔，如图 3-3(b)所示。第一个数据帧中的首字段本来是“101”，却被错误地认为是“111”，5 字节的数据帧长变为了 7 字节，从此开始，后续的数据帧都被认错，且无恢复的可能。所以，虽然字节计数法简单，却存在致命的缺陷，因此很少单独用于真实的网络传输中。

3.2.2　带字节填充的字节标志法

带字节填充的字节标志法克服了字节计数法固有的出错无法恢复的缺陷，它采用一个特殊的字节 FLAG 作为帧界的标志，这个特殊的字节 FLAG 被称为数据帧的定界符，一旦出错，可以扫描定界符进行再次同步，也就是可以从错误中恢复，如图 3-4(a)所示。

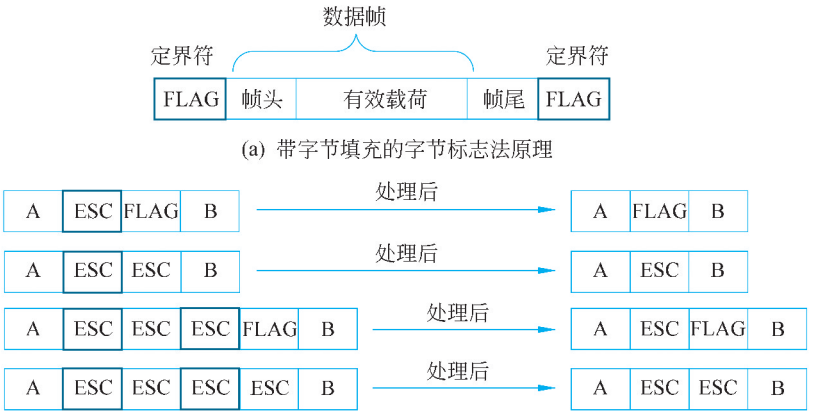


图 3-4　带字节填充的字节标志法原理和特例

这种方法唯一可能的问题是当作字节标志的特殊字节有可能出现在数据帧所承载的数据域中；一旦出现这种情况，本是数据的特殊字节被当作帧界了，于是数据帧就被错误地拆分了。

有一个类似的问题和解决方案出现在编程领域。比如在 C++ 中，我们可以用“\n”输出一个换行符，但是如果用户确实希望输出“\n”这两个字符，而不是一个换行符，则可用“\\n”表示。“\”被定义为 C++ 的转义符。类似地，我们可以定义一个特殊的转义字节(ESC)，在数据域中出现特殊字节时，要求在前面填充一个转义字节表示其仅仅是普通数据，而非数据帧的起始标志。同样地，如果数据域本身恰好包含了转义字节，则要求在其前也填充一个转义字节表示其为普通数据，不承载转义的特殊含义。

接收方接收到数据时，需要将数据域中额外填充的转义字节去除，从而使该数据与发送方发出的原始字节序列保持完全一致。在图 3-4(b)所示的 4 个例子中，A、B 表示普通数据，第一个特例的普通数据中包含了一个 FLAG，第二个特例的普通数据中包含了一个转义字节 ESC，第三个特例的普通数据中既包含了 FLAG，也包含了 ESC，第四个特例的普通数据中包含了两个 ESC，发送方在 FLAG 或 ESC 前填充 ESC 作为转义，如图 3-4(b)中左边数据帧中的蓝色框所示；图 3-4(b)右边对应的 4 个数据

帧是接收方收到之后，删除了转义符之后剩下的数据帧，这正是发送方要传输的填充前原始字节序列。

字节填充的方式要求帧长度必须是 8 位的整数倍，而且还限定必须使用 8 位作为一个字符的表示；这给编码和通信都带来了效率损失。举个极端的例子，假如发送方待传输的数据全部是 FLAG，每个 FLAG 前都要添加一个转义字节 ESC，这些 ESC 都是开销，真正的数据仅占填充 ESC 后全部字节数的约 50%。

稍后介绍的 PPP 在异步传输时，采用了带字节填充的字节标志法。

3.2.3 带位填充的位标志法

位填充的方法允许数据帧包含任意长度的位，而且还不限制字符的编码长度。带位填充的位标志法也要求使用一个特殊的位串作为每个数据帧的起始和终止标志，我们约定的帧界标志位串为“01111110”。如果在数据中出现了与帧界相同的位模式“01111110”，也会被误认为是帧界，进而引发错误的帧界分隔。为了解决这个问题，发送方的数据链路层在发送数据时，一旦发送了 5 个连续的“1”，则强制在数据中插入一个“0”。通过这种强制的位填充，“01111110”仅可能作为标志出现在帧头或帧尾。这种成帧方法也称为零比特填充法。

接收方收到数据帧时，会逐一检查收到的位，一旦发现有 5 个连续的“1”，则检查后续紧跟的位是否为“0”：如果是“0”，则自动删除该“0”，从而将位串复原为原始数据；而“01111110”的位串则是帧界标志。

相比带字节填充的字节标志法，带位填充的位标志法的开销更小，因而信道效率更高。

PPP 工作在同步模式时，采用了带位填充的位标志法。

3.2.4 物理层编码违例法

在很多局域网中，在物理层使用两种不同电平的转换表示一个二进制位。比如，使用“高-低”电平转换表示“1”位，而使用“低-高”电平转换表示“0”位。通过这样的编码，每个数据位中间的电平突变，将被用来作为发送方和接收方之间的同步。在这种编码中，连续的高电平或连续的低电平都不是有效的编码。因此，我们可以利用连续的高电平或连续的低电平表示一个数据帧的开始或者结束，这就是物理层编码违例法的工作原理。需要注意的是，物理层编码违例法只能使用在物理层编码包含足够冗余信息的网络中，因为我们需要用两个物理电平位编码一位数据。

在现实的网络中，我们可将字节计数法和其他某种方法结合使用，从而进一步确保成帧的正确性。数据帧的接收方先利用计数字段定位某帧尾，再检查该位置是否存在正确的分界标志。比如，在 IEEE 802.3 数据帧（参考 4.3.2 节）中，有 8 字节的前导码，相当于帧界；数据帧中还有一个长度字段，指明了数据帧的长度。

3.3 检错和纠错

成帧方法可以让我们准确地识别每个数据帧的起始和结束位置。我们面临的下一个问题是通信中的干扰和噪声。在数据传输中，由于干扰和噪声造成的错误是困扰

数据通信的一个长期问题。即便是物理层使用有屏蔽的传输线路，传输数字信号的是光纤网络，且由于传输技术的进步，传输错误率已经大大降低，但错误的发生还是存在一定的概率。尤其是无处不在的无线网络，由于数据传输在开放空间中进行(广播)，信号更容易受到干扰和损坏。因此，错误的处理将是一个需要长期关注的问题。

通信过程中错误的处理有两种策略。一种是错误检测(error detection)，即**检错**。检错的工作思路是：由接收方进行错误的检测，一旦发现传输数据中有错误，并不尝试恢复，而是想办法通知发送方，由发送方重新发送该数据帧。另一种是错误纠正(error correction)，即**纠错**。纠错的处理只由接收方进行，不但需要发现数据帧中的错误，还要通过算法自动对错误进行纠正，恢复出正确的数据。

两种错误处理的策略，各有适用的场景。直观的感受是，检测到 1 位的错误，难度会小于纠正 1 位的错误。当然，检测到 1 位的错误所需的冗余信息，也应小于纠正 1 位错误所需的冗余信息。但是，检错可能引起数据帧的重发，这又是额外的传输开销。

一般而言，在高质量的通信信道(例如光纤网络等低错误率的信道)中，由于错误极少发生，采用检错和重发的方式效率更高。而在错误率比较高的网络(例如无线网络)中，则更倾向使用纠错的方式，尽量减少错误数据帧的重发，以免引起信道的使用恶化。

不管是检错还是纠错，我们都需要通过设计编码规则在数据中添加冗余信息来实现。在下文中，我们将介绍较为简单和常见的几种检错码与纠错码的编码方案。

3.3.1　检错

1. 奇偶校验

奇偶校验(parity check)是一种较为简单和常见的校验数字传输正确性的方法。奇偶校验可以通过在数据中添加 1 位奇偶校验位实现。以奇校验为例，假如采用奇校验，发送方需要确保在一个数据帧中“1”的个数一定为奇数。接收方对接收到的帧中“1”的个数进行统计，若统计出“1”的个数为偶数，则意味着传输过程中有错误发生，需要发送方对该帧进行重传。

以 ASCII 中的字母 A 为例，发送方采用奇偶校验对其进行编码，对应的二进制编码为 1000001。若按照偶校验规则，校验位设置为 0，码字应为 10000010，则满足偶数个 1 的要求；若使用奇校验，则校验位应该设置为 1，码字应该为 10000011。如果采用偶校验的接收方收到一个码字为 10000011，发现其中有 3 个 1，不满足偶数个数的要求，表明发生了错误，所以，接收方可以告诉发送方重传。

奇偶校验是一种弱检错码，如果发生错误的个数不是 1 个或者奇数个，而是发生了偶数个错误，则接收方无法检出错误。

在通信中，如果一个数据帧中的消息位数为 m 位，添加的校验位是 r 位，总长度为 n，则 $n=m+r$，我们称其为一个 n 位的**码字**(codeword)。两个码字之间的差别，可以用海明距离(Hamming distance)表示。海明距离指的是两个码字对应位不同的位数，若海明距离为 d，则表示两个码字之间有 d 位的数值不同。如图 3-5 所示，10011100 和 11001001 之间有 4 位不同，则它们之间的海明距离为 4。将两个码字进行异或(XOR)逻辑运算，然后只需要统计结果中 1 的数量，就可以得到两个码字之间的海明距离。因此，利用简单的逻辑电路就可以很容易在硬件上实现海明距离的

计算。

```
    1 0 0 1 1 1 0 0
XOR 1 1 0 0 1 0 0 1
    ---------------
    0 1 0 1 0 1 0 1
```

图 3-5 计算两个码字之间的海明距离

现实中的海明距离判断，更多的不是在两个码字之间，而是在所有有效编码的集合内考虑海明距离。对于码字的集合，其海明距离的定义为，计算集合内每对码字之间的海明距离，其中最小的海明距离值，被定义为该集合的海明距离。如图 3-6 所示，如果有效的码字为 11111111、11110000、00001111、00000000，则该码字集合的海明距离为 4。

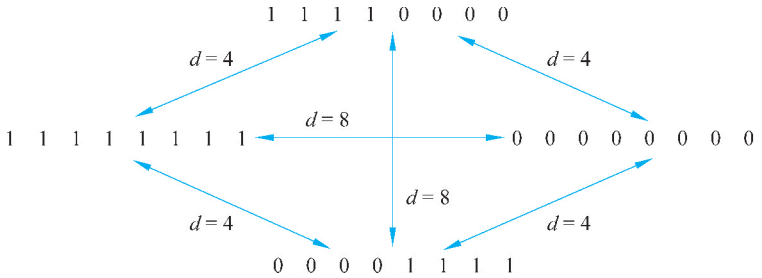


图 3-6 计算码字集合的海明距离示例

对于 1 位的奇偶校验来说，码字集合之间的海明距离为 2，这意味着，对于任何一个有效的码字，如果在传输过程中出现了一位的错误，则该码字都不会被误认为另一个有效的码字。关于检错，更为普适的结论是，**海明距离为 $d+1$ 的编码方案，最多可以实现 d 位错误的检测**。因为对于任何一个码字，如果出现 d 位的错误，必将变成一个无效的码字。

为了提高奇偶校验的检错能力，在 1 位奇偶校验的基础上，可采用**双向奇偶校验**（row and column parity）的方案。该方案也称为双向冗余校验，其工作原理是将一维的二进制数据串，作为一个 $m \times n$ 的矩阵处理，对每行和每列分别添加 1 位奇偶校验位。这种编码方案的检错能力更强。

2. 循环冗余校验

循环冗余校验（Cyclic Redundancy Check，CRC）是另一种在现实网络中被广泛使用的检错编码方案，比如以太网。循环冗余校验检错的基本思想是通过在数据位后添加若干位的校验和（checksum），以实现错误的检测。

在循环冗余校验中，二进制串被表示为系数为 0 或 1 的多项式，例如二进制串 110011，可以表示为多项式 x^5+x^4+x+1。假设有 m 个数据位需要被传输，则 m 个数据位可以表示为多项式 $M(x)$，添加校验和后的码字则可以表示为多项式 $T(x)$。此外，发送方和接收方在传输之前，需要事先确定一个生成多项式 $G(x)$。发送方在生成校验和时，需要满足 $T(x)$ 可以被 $G(x)$ 整除。对于接收方，在收到码字后，只需检查该码字是否还是能被 $G(x)$ 整除，即可判断是否出现传输错误。循环冗余校验的具体步骤如下：

第一步：数据传输前，双方确定 $G(x)$，假定其为 r 阶多项式。

第二步：发送方对每个数据帧计算其校验和。

(1) 对于数据帧 $M(x)$，在其尾部添加 r 个 0，表示为 $x^rM(x)$。

(2) 按照模 2 除法运算规则，将 $x^rM(x)$对应的二进制串除以 $G(x)$对应的二进制串，得到余数 c。

(3) 按照模 2 减法运算规则，将 $x^rM(x)$对应的二进制串减去余数 c，所得的结果即为需要传输的完整码字，可表达为多项式 $T(x)$。

第三步：接收方对收到的每个数据帧进行错误检测。

(1) 对于接收到的二进制串按照模 2 除法，除以 $G(x)$对应的二进制串，若不能整除，则认为该数据帧存在错误。

(2) 若能整除，则表明此传输过程未发生错误。

在上述步骤中，余数 c 就是计算得出的校验和，减去余数后，可以保证得到的 $T(x)$对应的二进制串能被 $G(x)$对应的二进制串整除。模 2 减法等效于模 2 加法操作，发送方如果用计算得到的校验和(余数 c)替换并添加在帧尾的 r 个 0，便可得到完整的码字。

我们以生成多项式 $x^8+x^5+x^4+1$(1 0011 0001)为例，来计算 1 字节(0001 0001)的 CRC 校验码。

由于该生成多项式最高阶为 8，按照 CRC 规则，需要在数据位尾部添加 8 个 0 进行升阶，得到的 $x^rM(x)$，其对应二进制串为 0001 0001 0000 0000。

以 $x^rM(x)$为被除数，以生成多项式为除数，进行模 2 除法计算，可得到余数为 0111 0010，这就是 CRC 校验位。以余数替换在数据尾部添加上的 8 个 0，则可得到经过 CRC 编码后的完整码字为 0001 0001 0111 0010。

接收方只要将收到的码字除以同样的生成多项式，如果余数为 0(即能整除)，便确认传输无误，可以放心接收该数据。

循环冗余校验计算较为简单，只需要进行模 2 的数学计算，也容易通过简单的硬件实现。该方案被广泛地应用于以太网等现实网络中进行传输错误的检测。

在实际 CRC 中，较为常用的生成多项式有 CRC-8、CRC-16 和 CRC-32 等。以 CRC-8 为例，其对应的标准生成多项式分别有

CRC-8：$x^8+x^5+x^4+1$　(1 0011 0001)

CRC-8：$x^8+x^2+x^1+1$　(1 0000 0111)

CRC-8：$x^8+x^6+x^4+x^3+x^2+x^1$　(1 0101 1110)

3. 互联网校验

互联网校验和(Internet checksum)则是另一种广泛使用的检错算法。互联网校验的基础是**二进制反码求和运算**。反码求和的计算过程并不复杂：将二进制数从低位到高位逐位相加，若产生进位，则需在相邻高位加 1；当最高位相加出现进位时，则需要在结果的最低位加 1；将最后计算所得的二进制数取反，就是二进制反码求和的结果。图 3-7 描述了对两个二进制数进行反码求和运算的简单示例。

在计算互联网校验和时，数据按照每 16 位作为一个单位被划分成一个序列。如果数据的字节长度为奇数，则在数据尾部填充 1 字节的 0 以凑成完整的 16 位。发送

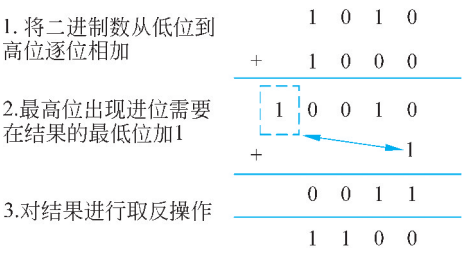


图 3-7 二进制反码求和运算示例

方按照反码求和运算规则将所有 16 位字的序列进行相加，然后再对相加的结果取反码，便得到校验和。

接收方进行检错的过程就是将收到的数据（包括校验和），再次以每 16 位作为一个单位进行反码求和运算。如果运算结果为 0，则表明数据在传输过程中没有错误；如果运算结果为非 0，即认为检测到错误，接收方将丢弃该数据。

你知道吗？ 本书后续章节将陆续介绍的 IP、ICMP、TCP、UDP 等重要协议都使用了互联网校验进行错误检测。其中，IP 校验和只是计算 IP 报文的头部，ICMP 校验和计算则包括了 ICMP 头部和 ICMP 数据。TCP 和 UDP 校验和计算除了包括头部和数据，还需要包括伪头部。这些协议的具体格式都将在后续章节进行详细介绍。

3.3.2 纠错

相对于检错，纠错可以减少重传，但是也意味着对于每个数据帧，都需要额外传输更多的冗余数据以及更复杂的纠错算法。纠错本身并不神秘，我们在日常生活中经常不自觉地对所接收到的信息进行纠错处理。比如一张字迹不清的字条，一块显示不全的屏幕，一通断断续续的电话。很多情况下，虽然数据不全或因为干扰产生错误，但在不自觉地进行纠错之后，我们仍然可以获得数据所包含的信息。最简单的纠错方式是寻找最相似或最可能的匹配。例如图 3-8 显示的电子钟屏幕，由于故障，只能显示出如左侧所示的图案，但是对于我们来说，可以毫不费力推断出数字 8（如右侧显示）是最可能的数字。生活常识告诉我们，对于错误的信号，可以通过找与之最接近的正确信号，进行直接且有效的纠错。

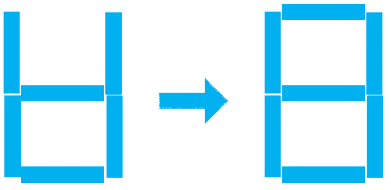


图 3-8 对一个无法正确显示的数字的推断

结合前文有关海明距离的知识，对于编码的纠错，我们可以得到一个更为普适的结论：**海明距离为 $2d+1$ 的编码方案，可以实现对 d 位错误的纠错**。可见纠错付出的冗余代价要比检错大得多。下面介绍几种常见的纠错方法。

1. 纠 1 位错的海明码

纠 1 位错的海明码是一种经典的自动纠错编码，该编码方案具备 1 位错误的纠正能力。整个海明码分为在发送方的编码阶段，以及在接收方的解码阶段（纠错）。对于需要发送的 m 位数据，首先需要确定所需的校验位 r 的数值，从而确定编码中整个码字的长度 n，且 $n=m+r$。

对于 m 位数据，可以组合出 2^m 个不同的消息。如果只考虑纠正 1 位错，而该位错误可能在 n 位中的任何一位出现，因此，对于 2^m 条数据对应的正确码字和出现 1 位错误的码字，其总数为 $(n+1)\times 2^m$。同时由于码字总长度为 n 位，则这些码字的个数必须满足 $(n+1)\times 2^m \leqslant 2^n$。考虑到 $n=m+1$，该不等式可变换为 $(m+r+1)\leqslant 2^r$。例如，当消息位长度 m 为 7 时，海明码方案中要求校验位的长度 r 必须大于或等于 4，才可实现该编码。

发送方的编码过程按照如下步骤进行。

(1) 将码字中的所有位从 1 到 n 编号，其中编号为 2 的幂次方的位设置为校验位(如 1,2,4,8,…)，其余位(如 2,3,5,6,7,…)为数据位。依次将 m 个数据位按照顺序填入。

(2) 校验位的数值，由对某组位使用奇偶校验得到(见图 3-9)。具体的规则如下。

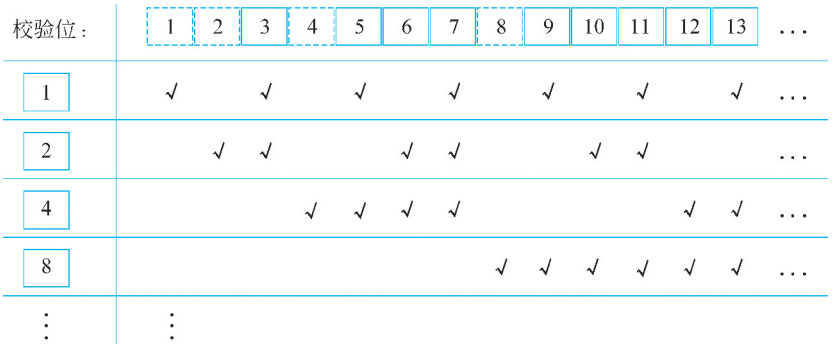


图 3-9　海明码进行校验位奇偶校验的规则示意图

- 将数据位的编号 k 用 2 的幂次方的和表示，如 $11=8+2+1$。
- 所有编号 k 的展开式中出现 2^r 的位，用来对 2^r 校验位进行奇偶校验。例如，编号为 2 的校验位数值，将由编号为 2、3、6、7、10、11 等位的奇偶校验决定是 0 或者 1。

(3) 完成所有校验位的计算，最终形成该报文的海明码表示。

当接收方收到 $m+r=n$ 位的海明码码字后，需要对该码字进行校验和纠错。海明码纠错过程，实际就是计算出码字中出错的那位，然后对该出错位进行取反操作，便完成了 1 位纠错的任务。接收方具体的纠错过程如下。

(1) 收到一个码字时，接收方初始化计数器 c 为 0。

(2) 对每个校验位按照海明码的计算方法进行重新计算，判断其是否满足奇偶校验的要求。

(3) 若校验位 r 不满足奇偶校验条件，则更新计数器 $c=c+r$。

(4) 完成所有校验位的计算后，检查计数器 c 的数值。若 c 为 0，则表示该码字无错误，可被正确地接收；若 c 非 0，则对码字中的第 c 位进行取反，从而实现海明码的 1 位纠错。

以数据 0111 1110 为例，采取偶校验的海明码实现 1 位纠错。

根据不等式 $(m+r+1)\leqslant 2^r$，可以确认对于 8 位数据最少需要的校验位 r 为 4，分别记为 P_1、P_2、P_3 和 P_4，它们在码字中的位置编号则分别为 1、2、4、8，将数据位依次填入对应的位，对于 4 个校验位，我们有如下偶校验计算：

$$P_1 = M_3 \oplus M_5 \oplus M_7 \oplus M_9 \oplus M_{11} = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$$
$$P_2 = M_3 \oplus M_6 \oplus M_7 \oplus M_{10} \oplus M_{11} = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$$
$$P_3 = M_5 \oplus M_6 \oplus M_7 \oplus M_{12} = 1 \oplus 1 \oplus 1 \oplus 0 = 1$$
$$P_4 = M_9 \oplus M_{10} \oplus M_{11} \oplus M_{12} = 1 \oplus 1 \oplus 1 \oplus 0 = 1$$

因此，最终完成偶校验后海明码的码字为 **0001** 1111 1110。

对于接收方来说，若收到的码字为 0001 1101 1110，则接收方需要重新计算 4 位校验位。对于 P_1、P_2、P_3 和 P_4 分别有如下计算：

$$P_1 = M_3 \oplus M_5 \oplus M_7 \oplus M_9 \oplus M_{11} = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$P_2 = M_3 \oplus M_6 \oplus M_7 \oplus M_{10} \oplus M_{11} = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$$
$$P_3 = M_5 \oplus M_6 \oplus M_7 \oplus M_{12} = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 1$$
$$P_4 = M_9 \oplus M_{10} \oplus M_{11} \oplus M_{12} = 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

在 4 位校验位中，P_1、P_2、P_3 三位均不符合偶校验的要求，其对应的位置编号 1、2、4 累加所得结果为 7，即 $c = 1+2+4 = 7$。接收方依此判断第 7 位出错，需要进行取反操作，纠正后的码字为 0001 1111 1110。

在较为可靠的传输信道中，传输错误只是偶发性的小概率事件，具备 1 位纠错能力的海明码方案完全可以胜任纠错的需求。但是对于容易受干扰、可靠性较差的网络（如无线网络），由于传输错误发生率较高，一个码字中可能出现连续多位的错误，显然传统的海明码方案无法满足纠错任务。这种连续位的错误，又称为突发错误。

但可以通过一个小改进，让海明码实现对一个码字中多位连续错误的纠错，即实现纠正突发错误。简单的改进方案（见图 3-10）就是将 i 个长度为 j 的码字组成一个 $i \times j$ 的矩阵。发送方在发送这些码字时，并非将这些码字逐个发送，而是按照矩阵的列逐列发送。通过这种方式，传输中连续多位的错误将可能出现在不同行的码字中。当接收方收到这 $i \times j$ 个位后，重新将这些数据按照 i 个码字逐个进行海明码的纠错。如果运气足够好，则连续位的错误都有可能被成功纠正。

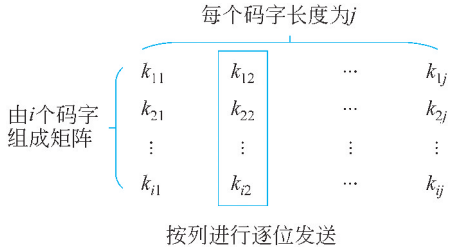


图 3-10 实现连续错误纠正的海明码改进方案

2. 里德-所罗门码

里德-所罗门码（简称**里所码**，Reed-Solomon code）是另一种使用广泛的纠错编码方案，比如，应用于 CD 存储、DVD 存储、DSL 传输、WiMAX 传输技术等。里所码是定义在有限域（也称为伽罗华域，Galois Field，GF）中的。里所码的相关计算也遵从有限域中的加减乘除运算法则。

里所码与海明码显著的不同在于，海明码以码字中的位为计算对象，而里所码则以“符号”（symbol）为单位，一个符号可以是 1 字节，即 8 位信息。里所码的编码结构如图 3-11 所示，一个完整的里所码包含 n 个里所码符号，每个里所码符号由 m 位构成，通常 m 为 2 的幂。n 个里所码符号中，k 个为消息符号，即为真正需要被发送的信息，另外的 $2t$ 个符号为校验符号（$2t = n-k$）。里所码具备纠正 t 个任何错误的能力，这 t 个错误可以是连续的突发性错误，也可以是分布在不同符号中的离散错误。

图 3-11 中的一列表示 m 位的符号，记为 S_1，S_2，…。

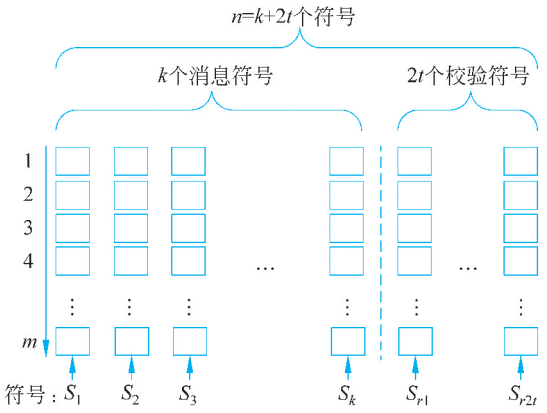


图 3-11　里所码的编码结构

里所码编码原理的数学描述并不复杂，主要分为 4 个步骤。

（1）以多项式的形式表示 k 个消息符号，记为 $M(x)$。

（2）将 $M(x)$ 升为 $2t$ 阶，即 $M(x)x^{2t}$。

（3）将 $M(x)x^{2t}$ 除以生成多项式 $G(x)$，余下的多项式为校验多项式，记为 $P(x)$。

（4）计算 $M(x)x^{2t}+P(x)$，即为生成的里所码。

编码过程的本质是将 k 维的信息数据映射到了 n 维空间中，并且保证 n 维空间中的 k 维数据子空间与 $2t$ 维校验空间是正交的。

若在传输过程中没有出现错误，则接收方所收到的里所码保证能被生成多项式 $G(x)$ 整除。若传输过程中出现不超过 t 个错误，则接收方可以通过解码确定出错符号位置，以及所对应的正确数据，从而实现纠错。

接收方收到的编码若以 $R(x)$ 表示，则 $R(x)$ 被生成多项式 $G(x)$ 除后，所得的余记为 $S(x)$，也称为 Syndrome。$S(x)$ 就是因为错误产生的，定义一个错误多项式（error locator）$E(x)=(1+X_1x)(1+X_2x)\cdots(1+X_vx)$，根据 Syndrome 值，最后可以计算得到 $E(x)$ 多项式，其中包含了出错符号的位置以及正确数值的信息。正确的里所码 $C(x)$ 可以通过计算 $R(x)-E(x)$ 获得。

需要强调的是，**里所码中提及的所有数学运算，均使用有限域 GF(2^q) 中的模运算法则**。对具体计算过程感兴趣的读者，需要对有限域中的数学运算进行一定的学习。

> **你知道吗：**有限域是数学家伽罗瓦于 18 世纪 30 年代研究代数方程根式求解问题时引入的概念。有限域在密码学、近代编码、计算机理论、组合数学等方面有着广泛的应用。有限域是指仅含有限个元素的域，在有限域中进行加法、减法、乘法和除法运算，其结果不会超出域集合。

3. 低密度奇偶校验

低密度奇偶校验（Low Density Parity Check，LDPC）码是另外一种较为常用的纠错编码，用于存储过程和无线网络通信中。LDPC 码的基本思想是通过一个生成矩阵 $\boldsymbol{G}$ 进行编码，通过一个校验矩阵 $\boldsymbol{H}$ 实现译码，即为实现编码的纠错。此时，生成矩阵 $\boldsymbol{G}$ 和校验矩阵 $\boldsymbol{H}$ 之间满足 $\boldsymbol{G}\times\boldsymbol{H}^{\mathrm{T}}=0$ 的条件。

在 LDPC 码方案中，发送的码字由信息位序列 $\boldsymbol{M}$ 与生成矩阵 $\boldsymbol{G}$ 相乘获得。假设 $\boldsymbol{M}$ 为(1 1 0)，$\boldsymbol{G}$ 为 3×6 的矩阵，如图 3-12 所示，则编码得到的码字 $\boldsymbol{C}$ 为(1 1 0 0 1 0)。注意，LDPC 码中的元素与元素是进行模二加法(异或)运算。

LDPC 码的译码有硬判决和软判决两种方法。其中，硬判决是指对信道输出位直接做出是 1 或 0 的判决，**比特翻转算法**是一种典型的硬判决算法；软判决不是直接输出 1 或 0，而是给出某位是 0 或 1 的概率，这个方法也称为**和积译码**，置信传播算法则是典型的软判决算法。

我们以位翻转算法为例进行讨论。由于生成矩阵 $\boldsymbol{G}$ 和校验矩阵 $\boldsymbol{H}$ 之间满足 $\boldsymbol{G}\times\boldsymbol{H}^{\mathrm{T}}=0$的关系，对应于图 3-12 的生成矩阵，有与之对应的校验矩阵 $\boldsymbol{H}$，如图 3-13 所示。通过 $\boldsymbol{HC}^{\mathrm{T}}$，校验矩阵定义了三个校验方程。

消息位$\boldsymbol{M}$ $\begin{bmatrix}1&1&0\end{bmatrix}$

生成矩阵$\boldsymbol{G}$ $\begin{bmatrix}1&0&0&1&0&1\\0&1&0&1&1&1\\0&0&1&0&1&1\end{bmatrix}$

LDPC码$\boldsymbol{C}=\boldsymbol{MG}$ $\begin{bmatrix}1&1&0&0&1&0\end{bmatrix}$

图 3-12 LDPC 码编码示例

校验矩阵$\boldsymbol{H}$ $\begin{bmatrix}1&1&0&1&0&0\\0&1&1&0&1&0\\1&1&1&0&0&1\end{bmatrix}$

图 3-13 LDPC 码校验矩阵 $\boldsymbol{H}$ 示例

校验方程 1：

$$\boldsymbol{C}_1 \oplus \boldsymbol{C}_2 \oplus \boldsymbol{C}_4$$

校验方程 2：

$$\boldsymbol{C}_2 \oplus \boldsymbol{C}_3 \oplus \boldsymbol{C}_5$$

校验方程 3：

$$\boldsymbol{C}_1 \oplus \boldsymbol{C}_2 \oplus \boldsymbol{C}_3 \oplus \boldsymbol{C}_6$$

当接收方收到码字 $\boldsymbol{C}$ 后，需要将其与校验矩阵 $\boldsymbol{H}$ 相乘，进行译码操作。若 $\boldsymbol{HC}^{\mathrm{T}}$ 得到一个零向量，即三个校验方程的结果均为 0，则译码成功，接收方已经接收到正确信息。

假设在传输过程中 $\boldsymbol{C}$ 码字发生一位错误，接收到的码字为(0 1 0 0 1 0)，则校验方程 1 和 3 结果为非 0。由于我们只处理一位错误的情况，校验方程 1 和 3 中，同时出现的 $\boldsymbol{C}_1$ 和 $\boldsymbol{C}_2$ 均可能出错。但由于校验方程 2 的结果为 0，则证明 $\boldsymbol{C}_2$ 并未出错，那么可以判定出错位必然是 $\boldsymbol{C}_1$。将 $\boldsymbol{C}_1$ 进行翻转，则译码得到正确的码字为(1 1 0 0 1 0)。

你知道吗？ 除了 LDPC 码之外，涡轮码(Turbo code)和极化码(Polar code)是另两种在通信领域得到广泛使用的前向纠错编码技术。其中，极化码由土耳其数学家 Erdal Arikan 教授于 2008 年在国际信息论 ISIT 会议上提出，是人类已知的第一种能够被严格证明达到信道容量的信道编码方法。2016 年，我国的华为公司主推的极化码方案，成为国际无线标准化机构 3GPP 通过的 5G 通信控制信道编码方案。

3.4　可靠传输数据帧的技术

为了简化和明确本节所讨论的问题范围，我们先明确如下三点假设。

（1）分层独立假设。本节讨论的数据传输系统严格遵循网络分层模型，网络层和物理层作为数据链路层相邻的上下层，三者为彼此独立的进程。相邻层的进程之间通过消息传递实现上下层之间的通信。

（2）可靠服务假设。数据链路层要为网络层提供可靠的传输服务。在接收方，数据链路层以正确的顺序将正确的数据交给网络层。同时为了简化起见，我们假设在发送方的数据链路层所需发送的数据随时可以从网络层获得。

（3）通信错误假设。在本节，我们仅处理通信错误，而不考虑由于设备重启、异常断电、崩溃等产生的数据错误。

就像前文介绍的一样，在进行数据传输的过程中，错误是不可避免的。除了环境噪声干扰、传输时延、信号衰减等因素造成的错误之外，也可能是由于接收方不能及时处理所接收的数据，在缓存耗尽之后，不得不丢弃数据。总而言之，数据传输从网络本身的物理属性来看，是不可靠的。而数据链路层所要实现的基本功能之一，就是要为网络层提供可靠的数据传输服务。因此，除了上文所说的检错纠错之外，需要发送方和接收方共同遵循一些重要的协议规则。

前文也提到，在发送方和接收方的数据链路层之间，交换的数据单元是数据帧。因此，在这部分，我们来讨论如何通过设计协议保证这些数据帧在通信双方的数据链路层之间正确可靠地传输，并分别把数据帧内的有效载荷数据提交给网络层。

3.4.1　流量控制

如果数据传输的信道是完美的，接收方的处理速度无限快，或者说接收方拥有无限大的缓存，则传输的数据帧既不会受损也不会丢失。这种被学者们称为“乌托邦”的场景不需要我们进行任何错误控制或流量控制，便可实现可靠的数据传输。当然这是现实中几乎不可能出现的通信场景。真实的通信错误主要来自两方面：一是通信信道上产生的错误，例如噪声干扰、衰减时延等；二是由于接收方无法及时处理到达的数据而导致数据的丢弃。

在本节，我们先考虑后者，即在真实的数据通信场景中，接收方处理到达的数据帧需要一定时间。当处理数据帧的速度低于数据帧到达的速度时，接收的数据帧就只能存放在接收方的缓存中，而缓存的大小必然是有限的，因此数据帧被丢弃的可能必然存在。

解决这一问题的基本思想非常简单，就是需要对发送方和接收方之间数据流量进行控制，避免发送方以超过接收方处理能力的速度发送大量数据帧，最后导致接收方被“淹没”(overwhelming)。有两种策略可以用来实现发送方和接收方之间的流量控制：基于固定传输速率的流量控制和基于反馈机制的流量控制。

第一种策略，需要通信双方在进行数据传输之前约定好固定的数据帧传输速度，确保发送数据帧的速度不超过接收方的处理能力。虽然理论上这种流量控制方式很容易被实现，但是在现实中，因为接收方可能需要同时与不确定数量的发送方进行通

信，接收方的处理能力很难事先被准确地估计。如果通信双方约定一个过低的数据帧传输速度，会导致低效的传输效率和宽带资源的浪费。

因此，基于反馈机制的流量控制相对来说更合理和可行。反馈机制要求接收方在完成一个数据帧的接收，并成功提交给网络层后，向发送方进行反馈，告知其可以进行下一个数据帧的发送。在实现中，通常由接收方在完成到达数据帧的处理后，向发送方发送一个**空帧**(dummy frame，也称为**哑帧**)作为成功接收的反馈确认。

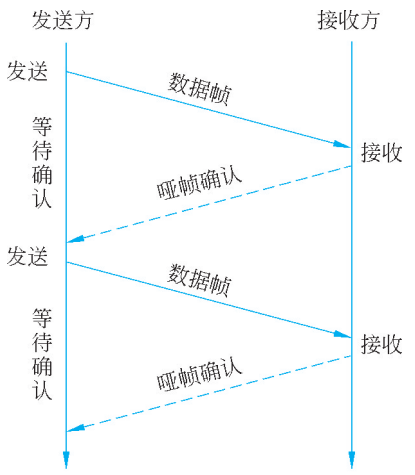


图 3-14 停-等机制的流量控制流程

站在发送方的角度来看，这一过程可以表述为，向接收方发送一个数据帧后即停下(阻塞)，等待来自接收方的反馈确认到达，空帧到达后，再开始新一数据帧的发送，并等待新的确认。因此这种流量控制机制也称为**停-等**(stop-and-wait)协议。

利用停-等机制的流量控制流程如图 3-14 所示，数据帧是由发送方到接收方单向传输，作为确认的空帧，则在相反方向传输。为了实现流量控制，所付出的代价是发送方在每个数据帧发送之后，进行一段时间的等待。

3.4.2 肯定确认与重传

传输中导致数据错误或丢失的另一个原因则是传输信道并不总是完美可靠。对于发生错误的数据帧，利用编码方案进行自动纠错是一种选择。但纠错能力的增强，也意味着需要传输更多冗余信息和花费更多的计算资源。相对来说，采用错误检测策略，并要求对方对错误数据帧进行重传，则是一种妥协后更经济的方案。那如何在发送方和接收方之间进行控制，从而实现错误数据帧和丢失数据帧的重发呢？

借鉴停-等协议的思想，很容易找到如下一个简单的解决方案。

对于发送方，我们只需要再设置一个**计时器**(timer)，每发送一个数据帧，发送方就开启这个计时器，只要在计时器**超时**(timeout)之前，能收到来自接收方的确认，则认为该数据帧被正确收到；否则一旦计时器超时，则自动重发当前数据帧。

对于接收方，如果收到的数据帧被检测出错误，则保持沉默。只有到达数据帧是无误的，接收方才发送确认帧。

该方案的主要思想是，只要发送方超时，即进行当前数据帧的重发。但是，如果我们考虑超时产生的原因，就会发现该方案可能导致问题。由于当前考虑的是真实的不可靠的通信信道，数据帧和作为确认的空帧都面临错误或丢失的风险。如果超时是由于确认帧无法正确到达而产生，则此时被重发的数据帧其实已被接收方正确收到(见图 3-15)。对于接收方，当重发的数据帧再次到达时，根本无从判断该数据帧是重发数据帧还是下一个新数据帧。

那接收方能否从数据帧的内容上进行是否为重发数据帧的判断呢？答案也是否定的。首先，这破坏了分层独立性，数据帧承载的数据内容对于数据链路层应该是透

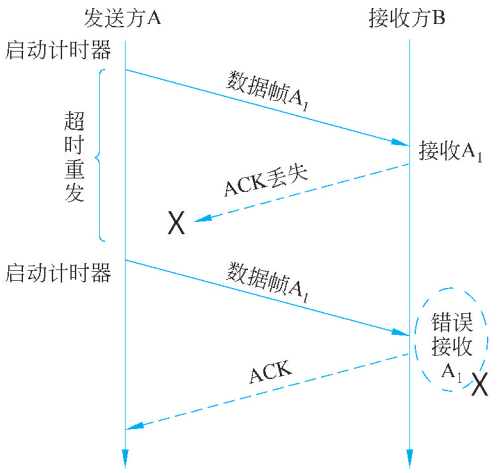


图 3-15　重发帧因为能被识别而被错误接收

明不可见的。其次，即便数据内容一样，也不能确认该数据帧为重传的数据帧，也可能发送方确实需要连续发送两个相同内容的数据帧。

为了解决确认帧丢失引发的重复接收重复数据帧的问题，我们需要在帧头附加一个可以标识不同数据帧的字段，称为数据帧的**序列号**(sequence number)，或者**帧号**(frame number)。由于传输采取停-等机制，帧号仅需要用来识别连续的两个数据帧是否是重传的同一数据帧。因此仅需用一位二进制数，让帧号在 0 和 1 之间切换便可满足要求。当接收方收到帧号连续为 0 或连续为 1 的数据帧时，便可判断这是重传的数据帧，数据需要丢弃，而不是交给自己的网络层。

这种需要收到一个成功接收确认再进行新数据帧发送，否则超时重传的协议，称为**自动重传请求**(Automatic Repeat reQuest，ARQ)协议，或**支持重传的肯定确认**(Positive Acknowledgement with Retransmission，PAR)协议。该协议在正常和异常情况下的工作示例分别如图 3-16 和图 3-17 所示。

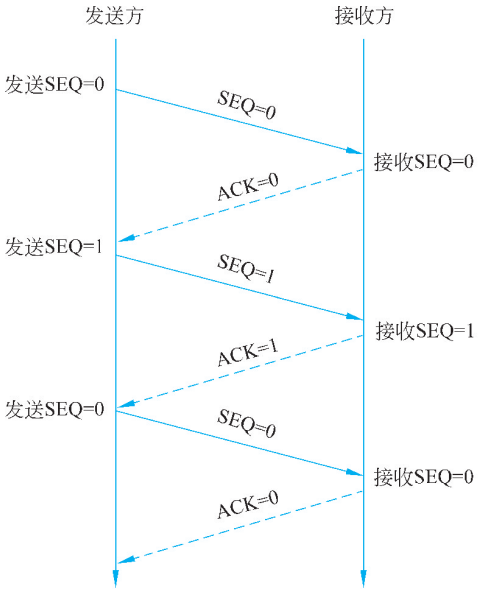


图 3-16　在正常情况下 PAR 工作流程

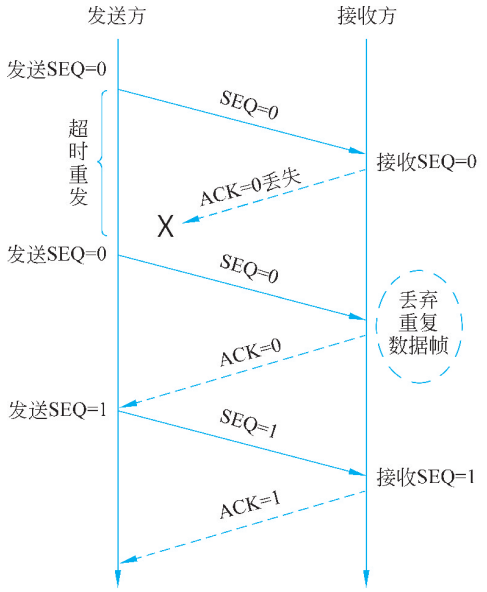


图 3-17　PAR 在出现确认帧丢失时的工作流程

3.4.3 传输效率评估和提升

让发送方每发送一个数据帧就停下来等待确认或超时，以决定是否开始发送新数据帧或者重传，这种方法虽然可以实现可靠的传输，但很明显它是极其低效的，因为有相当多的时间都被用来等待，导致网络带宽的浪费。

如图 3-18 所示，假设每个数据帧的长度固定为 f 位，传输信道的带宽为 b b/s，则发送方需要逐位发送完一个数据帧的时间 $T=f/b$ s。而在这之后，发送方被阻塞需要等待确认帧到达的时间(D)由信号在两端之间来回传输的时延(r)，以及接收方完成处理该数据帧并生成确认帧的时间(p)决定。表示传输效率的**信道利用率**(channel utilization)可以计算为 $T/(T+r+p)$。如果忽略数据帧的处理时间 p，信道利用率则可表示为 $f/(f+rb)$。由此可见，传输距离越远，传输时延越大，网络带宽越高，信道利用率越低，因为大多数的时间都被用来等待确认的到达。

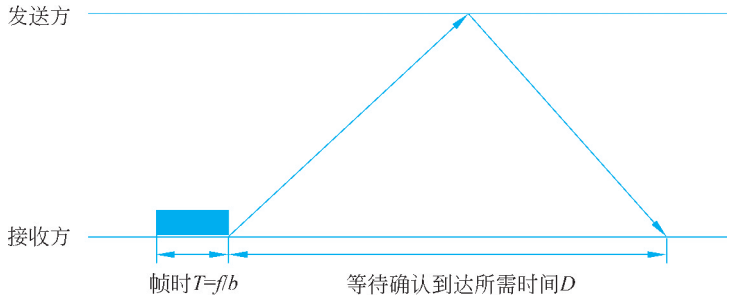


图 3-18 信道利用率计算示意图

有效提升信道利用率的方法是不再拘泥于停-等协议的机制，而让发送方可以连续发送 w 个数据帧。如图 3-19 所示，最理想的状态下，当发送方完成第 w 个数据帧的发送时，第一个数据帧的确认恰好到达，发送方可以立刻进行第 $w+1$ 个数据帧的发送。这时候的信道利用率理论上可以达到 100%。

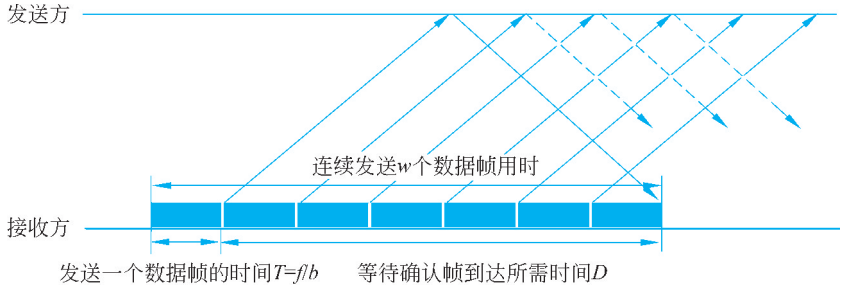


图 3-19 连续发送 w 个数据帧时信道利用率计算示意图

这就是**滑动窗口**(sliding window)协议的思想。具体实现时，需要在发送方维护一个**发送窗口**(sending window)，发送窗口的大小 w，即为允许不等待确认而连续发送数据帧的最大数量。发送窗口装载的是已经发送但其确认尚未到达的数据帧，这些数据帧都有可能由于传输错误或丢失而被要求重传。当发送窗口被填满时，发送方必须停止发送，而等待关于这些数据帧的确认帧的到达。当发送窗口内的数据帧被确认接收后，该数据帧需要被移出窗口，同时发送窗口向前滑动，继续发送后续的数据帧。

接收方也需要维护一个**接收窗口**(receiving window),用来判断哪些数据帧可以直接交给网络层,哪些数据帧需要暂时缓存,哪些数据帧需要丢弃。接收方需要确保所有正确接收的数据帧,以正确的顺序交给网络层。在某个数据帧的数据被提取出来交给网络层后,该数据帧也需要被移出接收窗口,并将接收窗口向前滑动。

由于不再使用单纯的停-等协议工作机制,如果发送窗口的尺寸大于 1,意味着可能有多个数据帧同时到达,或可能同时被确认。为了区分这些数据帧,此时的**帧号**不能仅用 1 位二进制数表示,更普适性的表示方法是用 n 位表示帧号,帧号按照顺序从 0 到 2^n-1 之间被循环使用。

其他提升传输效率的思路,是关于确认帧的传输。在停-等协议中,数据帧和作为确认的空帧都是单向传输的。而在实际的网络通信中,数据的传输通常都是双向进行的,发送方本身也是数据的接收方,接收方也需要发送数据给发送方。那么,为什么还要专门发送空帧作为确认的反馈呢?对收到数据帧的确认帧,完全可以包含在反向传送的数据帧头中,以一个附加字段的形式发送。这种“搭便车”发送确认帧的方式,称为**捎带确认**(piggybacking)。关于“搭便车”,这里又产生如下两个问题。

第一,反向的数据帧并不是时刻发送的。为了搭便车,可能需要等待很长时间才能被发送,由此也会造成传输效率的下降。解决的方案通常是设定一个计时器,在计时器超时前,如果未能实现捎带确认,则不得不使用空帧进行确认帧的传输。

第二,在等待“搭便车”的过程中,也可能有新的数据帧到达。由于滑动窗口机制的采用,发送方可能连续发送多个数据帧。比如帧号为 0 的数据帧从主机 A 到达主机 B 后,关于 0 号数据帧的确认帧正在等待“搭便车”。在等待的过程中来自 A 且帧号为 1、2、3 的数据帧也纷纷到来。这时候等到了一个需要由 B 反向传输到 A 的数据帧,那搭便车的应该是关于哪个数据帧的确认帧呢?解决的方法也很简单,我们只需要对最后一个成功接收并且交给网络层的数据帧进行确认即可。在上述例子里,主机 B 只需要对 3 号数据帧进行确认即可。当 A 收到来自 B 关于 3 号数据帧的确认帧时,便可以推断之前的 0、1、2 号数据帧已经被 B 成功接收,于是可以将 0、1、2、3 号数据帧均移出 A 的发送窗口,因为已经确保不需要重发它们了。发送窗口向前滑动后,可以用来发送更多后续的数据帧。这种确认方式,也称为**累积确认**(cumulative acknowledgement)。

关于捎带确认以及滑动窗口具体的实现,我们将在下一节具体展开说明。

3.4.4　滑动窗口协议

滑动窗口协议允许发送方连续发送 w 个数据帧,而不需要完成每个数据帧的发送后停下来等待该数据帧的确认帧。这可以对传输效率进行有效提升,但同时也带来新的问题:如果连续发送的 w 个数据帧中间有一个数据帧被丢失或者发生错误,该如何处理呢?

滑动窗口协议通常有两种处理错误数据帧或丢失数据帧的策略:**回退 n 帧**(go-back-n)协议或**选择性重传**(selective repeat)协议。

顾名思义,回退 n 帧协议在发送窗口中某个数据帧出错超时时,该数据帧以及其后续所有数据帧将要重新发送。对于回退 n 帧协议,需要维护一个较大的发送窗口,以保证传输效率。当接收一个数据帧时,只需要判断该数据帧是否为坏数据帧(数据

帧内容错误，或者并非按顺序到达的数据帧）。若该数据帧无误，则将该数据帧交给网络层的进程处理，否则就抛弃该数据帧。

图 3-20 展示了回退 n 帧协议的一个例子。在该例子中，接收窗口大小为 1。数据帧 F_0、F_1 被成功接收时，将分别发送确认帧，接收窗口相应地往前滑动。当数据帧 F_2 出错时，该数据帧将被丢弃，且不会发送任何确认帧。后续的 F_3、F_4 等数据帧虽然正常到达，由于其并非满足接收窗口的要求，因此也将被丢弃，接收方将再次发送关于最后一个成功交给网络层的数据帧的确认帧（ACK＝1），此时接收窗口保持不变。

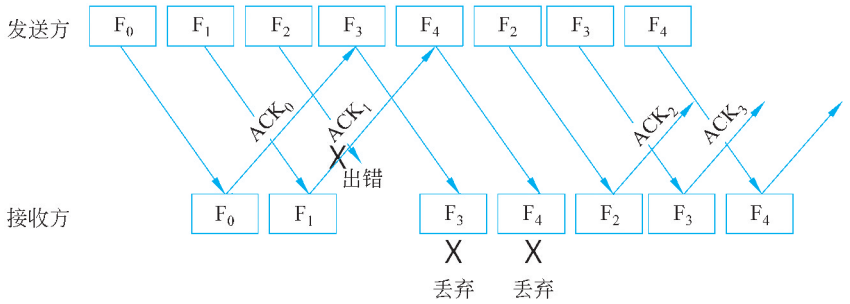


图 3-20 回退 n 帧协议工作示意图

当发送窗口中的 F_2 对应计时器超时后，F_2 以及后续的 F_3、F_4 等数据帧将被重传。如果它们能被正确接收并确认，发送窗口和接收窗口都将继续滑动，直至所有数据发送完毕。

回退 n 帧协议虽然对到达数据帧的处理简单，但在出错时，要求将发送窗口中发生错误的数据帧以及后续数据帧全部重传。在错误率比较高的网络中，这将导致大量的数据要求重传，造成带宽资源的浪费。

选择性重传协议要求只将发送窗口中的错误或超时数据帧进行重传。接收方对于到达的数据帧，需要分如下两种情况处理。

第一种情况，对于到达的数据帧，若其帧号并非在接收窗口，则直接丢弃该数据帧，再次发送关于最近成功接收并交给网络层的数据帧的确认帧。

第二种情况，若到达数据帧的帧号落在接收窗口区间内，则需要判断是否需要将该数据帧的数据提交给网络层。若数据帧并非按正确顺序到达，则需要暂时缓存该数据帧，发送关于最近成功接收并交给网络层的数据帧的确认帧。在等到缺失数据帧到达后，才逐次提交给网络层。同时，接收方需要向前滑动接收窗口，并发回确认帧。

图 3-21～图 3-23 展示了选择性重传协议工作的三种场景。接收方接收窗口大小为 4，当前允许接收的帧号为 1、2、3、4。在图 3-21 的场景中，到达的数据帧的帧号为 5，接收方判断其不在接收窗口内，因而丢弃该数据帧，并再次发送关于 0 号数据帧的确认帧。

在图 3-22 的场景中，在 1 号数据帧到达后，接收窗口向前滑动，窗口中的帧号更新为 2、3、4、5。

在图 3-23 的场景中，帧号为 3 的数据帧到达后，将被缓存，直到 1 号、2 号数据帧到达，1 号、2 号和 3 号数据帧将依次被提交给接收方的网络层，并被确认，同时接收窗口向前滑动，更新为 4、5、6、7。

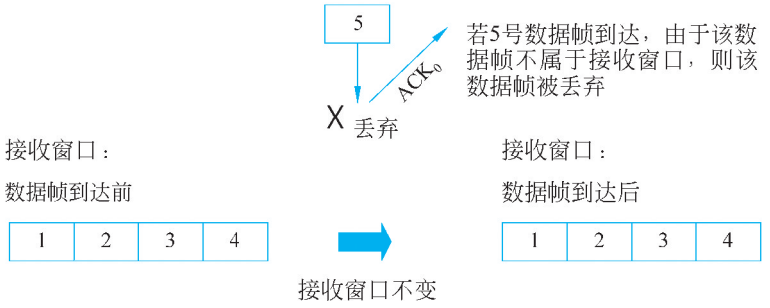


图 3-21　选择性重传协议中收到不属于接收窗口帧号的情况

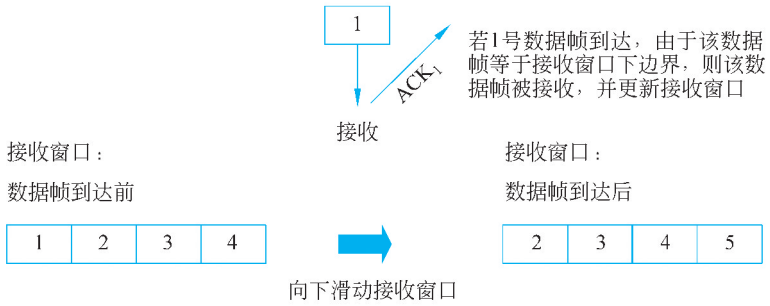


图 3-22　选择性重传协议中收到期望帧号后滑动接收窗口

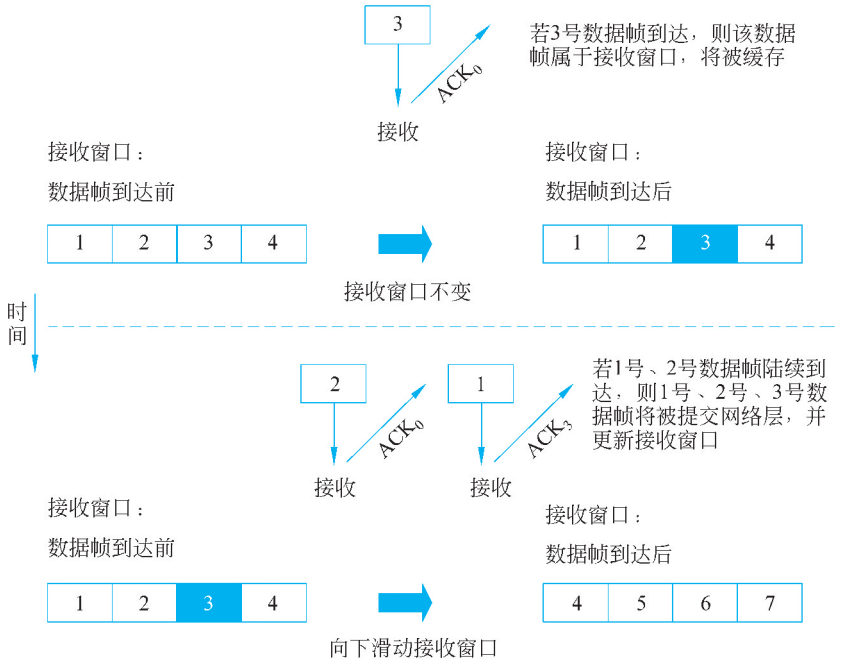


图 3-23　选择性重传协议中接收的数据帧被缓存于接收窗口后的情况

3.4.5　滑动窗口与帧号

滑动窗口协议允许连续发多个数据帧从而提高信道利用率，发送窗口的大小 w 决定了可以不等待确认连续发送数据帧的数量。那么是否是发送窗口越大，信道利用