第3章　数据链路层

本章首先介绍数据链路层的相关基本概念和三个重要问题；然后介绍点对点信道上最常用的点对点协议；之后讨论共享式以太网及其相关设备集线器，交换式以太网及其相关设备网桥和以太网交换机；最后介绍802.11无线局域网。

本章重点

（1）数据链路层的三个重要问题：封装成帧和透明传输、差错检测、可靠传输。

（2）三种实现可靠传输的机制：停止–等待协议，回退N帧协议，选择重传协议。

（3）点对点信道上最常用的点对点协议PPP。

（4）共享以太网使用的CSMA/CD协议。

（5）网络适配器、集线器、网桥以及以太网交换机的作用，特别是网桥和以太网交换机的工作原理。

（6）802.11无线局域网的组成和CSMA/CA协议。

3.1　数据链路层概述

数据链路层是作为法律标准的OSI体系结构自下而上的第二层，其主要任务是**实现帧在一段链路上或一个网络中进行传输的问题**。本节对数据链路层进行概述，首先介绍数据链路层在网络体系结构中所处的地位，然后介绍链路、数据链路和帧的基本概念。

3.1.1　数据链路层在网络体系结构中所处的地位

下面通过两台主机通过互联网进行通信的例子，来看看数据链路层在网络体系结构中所处的地位。

图3-1（a）所示的是局域网1中的主机H1经过路由器R1、广域网以及路由器R2连接到局域网2中的主机H2。当主机H1向H2发送数据时，从网络体系结构的角度看，数据的流动如图3-1（b）所示。

主机H1和H2中都有完整的网络协议栈（这里以五层协议栈为例），而路由器在转发数据包时仅使用协议栈的物理层、数据链路层和网络层。H1向H2发送数据的流程如下：

（1）待发送的数据在主机H1中按网络体系结构自上而下逐层封装，物理层将数据链路层封装好的协议数据单元看作比特流，并将其转化成相应的电信号发送出去。

（2）数据包进入路由器后，会从物理层开始被逐层解封，直到解封出网络层协议数据单元PDU。路由器从该PDU的首部中取出目的地址（属于网络层地址），根据目的地址在转发表中找到相应的下一跳地址后，将该PDU向下逐层封装后通过物理层发送出去。

（3）主机H2收到数据包后，按网络体系结构自下而上对其逐层解封，最终解封出主机H1发送的数据。

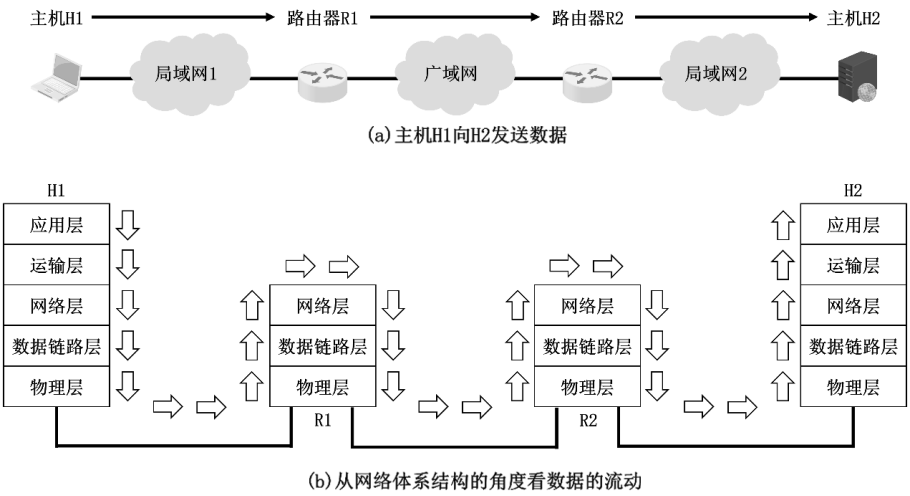


图3-1　数据链路层在网络体系结构中的地位

为了简单起见，当学习数据链路层的相关问题时，在大多数情况下，我们可以只关注数据链路层本身，而忽略网络体系结构中的其他各层。对于本例，当主机H1给H2发送数据时，可以想象数据是在各相关设备的数据链路层之间沿水平方向传送，如图3-2所示。仅从数据链路层的角度看，主机H1到H2的通信由三段不同的数据链路层通信组成，即：H1→R1、R1→R2以及R2→H2。请读者注意，这三段不同的数据链路层可能采用不同的数据链路层协议。

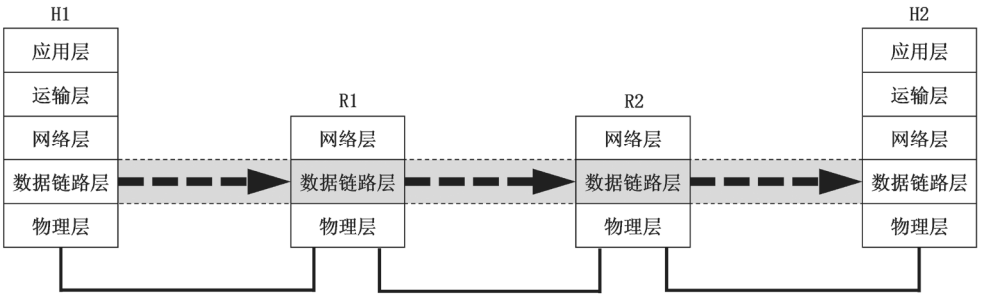


图3-2　仅考虑数据在数据链路层的逻辑传输

3.1.2　链路、数据链路和帧

请读者注意，链路与数据链路是有区别的。

1. 链路

链路（Link）是指从一个节点到相邻节点的一段物理线路（有线或无线），而**中间没有任何其他的交换节点**。

网络中各主机之间的通信路径一般由多段链路构成。例如在图3-1中，主机H1与H2的通信路径就包含了H1→R1、R1→R2以及R2→H2共三段链路。

2. 数据链路

数据链路（Data Link）是基于链路的。当在一条链路上传送数据时，除需要链路本身，还需要一些必要的**通信协议**来控制这些数据的传输，把**实现这些协议的硬件和软件加到链路上**，就构成了数据链路。

计算机中的**网络适配器**（俗称网卡）和其相应的软件驱动程序就实现了这些协议。**一般的网络适配器都包含了物理层和数据链路层这两层的功能**。

3. 帧

帧（Frame）是数据链路层对等实体之间在水平方向进行逻辑通信的协议数据单元PDU。如图3-3所示，发送方的数据链路层给网络层交付下来的分组添加首部和尾部，使之封装成为帧，然后将帧交付给物理层进行发送。接收方的数据链路层从物理层交付上来的帧中解封出分组，并将其上交给网络层。

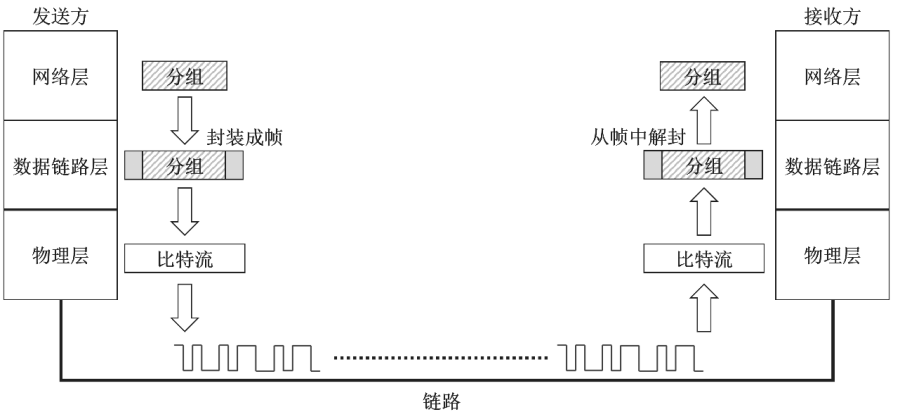


图3-3　帧是数据链路层对等实体间的通信单元

数据链路层不必考虑物理层如何实现比特流的传输。为了简单起见，可以认为帧是在通信双方数据链路层的对等实体之间沿水平方向直接传送，如图3-4所示。

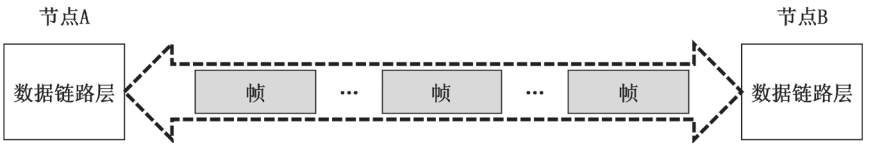


图3-4　只考虑数据链路层帧的传输

3.2　数据链路层的三个重要问题

尽管数据链路层有多种协议，但在这些数据链路层协议中，有三个重要的问题是共同的。这三个重要问题是**封装成帧、透明传输、差错检测**。近些年，随着无线局域网（例如Wi-Fi）的应用逐渐普及，**可靠传输**也成为了数据链路层的重要问题。由于透明传输与封装成帧是相关的，因此本书将封装成帧和透明传输合并为一个重要问题。下面对上述重要问题进行详细介绍。

3.2.1　封装成帧和透明传输

1. 封装成帧

所谓封装成帧（framing），就是给网络层交付下来的分组添加一个首部和一个尾部，这样就构成了一个帧，如图3-5所示。

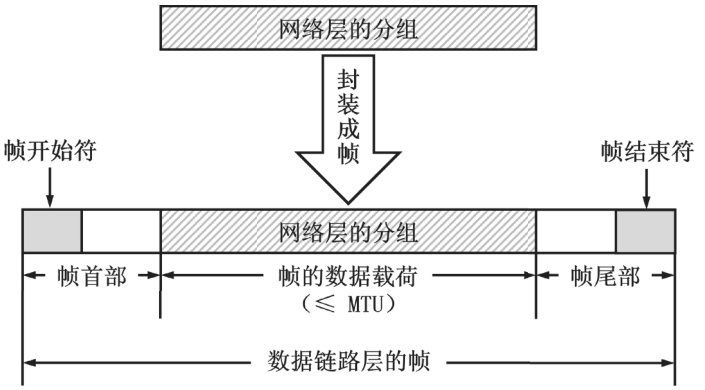


图3-5　封装成帧

帧的首部和尾部中包含有一些重要的控制信息。例如，帧首部中往往包含帧开始符、帧的源地址和目的地址（属于数据链路层地址），而帧尾部中往往包含帧校验序列和帧结束符。接收方的数据链路层在收到物理层交付上来的比特流后，根据帧首部中的帧开始符和帧尾部中的帧结束符，从收到的比特流中识别出帧的开始和结束，也就是进行**帧定界**。

各种数据链路层协议都对帧首部和尾部的格式有明确的定义。**为了提高数据链路层传输帧的效率，应当使帧的数据载荷的长度尽可能地大于首部和尾部的长度**。考虑到对缓存空间的需求以及差错控制等诸多因素，每一种数据链路层协议都规定了帧的数据载荷的长度上限，即**最大传送单元**（Maximum Transfer Unit，MTU），例如以太网的MTU为1500个字节。

2. 透明传输

如果在帧的数据载荷部分恰好也出现了帧定界符（帧开始符或帧结束符），就会造成接收方数据链路层出现**帧定界的错误**，如图3-6所示。

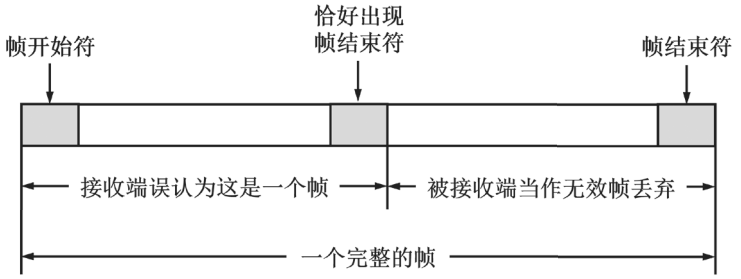


图3-6　帧中数据载荷部分恰好出现了帧定界符

如果不解决上述问题，则数据链路层就会对上层交付的协议数据单元（PDU）的内容有所限制，即PDU中不能包含帧定界符。显然，这样的数据链路层没有什么应用价值。如果能够采取措施，使得数据链路层对上层交付的PDU的内容没有任何限制，就好像数据链路层不存在一样，就称其为**透明传输**。

实现透明传输的方法一般有两种：**字节填充**和**比特填充**。

1）**字节填充**

当使用**面向字节的物理链路时，使用字节填充的方法实现透明传输**。

借助图3-7理解字节填充法：

- 发送方的数据链路层在数据载荷中出现帧定界符的前面，插入一个转义字符“ESC”（其十六进制编码是1B）。如果转义字符自身也出现在数据载荷中，则在转义字符的前

面也插入一个转义字符。

- 接收方的数据链路层在把数据载荷向上交付网络层之前，删除先前发送方数据链路层插入的转义字符。

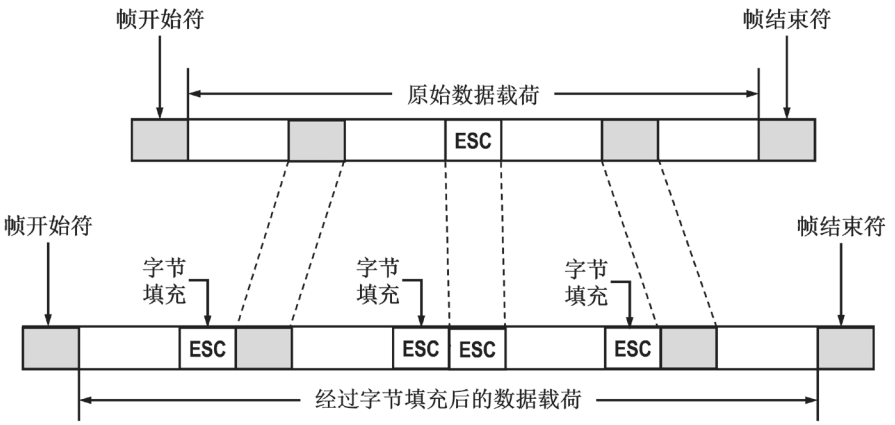


图3-7　采用字节填充法实现透明传输

2）比特填充

当使用**面向比特的物理链路**时，**使用比特填充的方法实现透明传输**。

假设某数据链路层协议采用8个比特构成的特定位串0111 1110作为帧定界符，借助图3-8理解比特填充法：

- 发送方的数据链路层扫描数据载荷，只要出现5个连续的比特1，就在其后填入一个比特0。经过这种比特0填充后的数据载荷，就可以确保其不会包含帧定界符。
- 接收方的数据链路层在把数据载荷向上交付网络层之前，对数据载荷进行扫描，每当发现5个连续的比特1时，就把其后的一个比特0删除，这样就可以还原出原始的数据载荷。

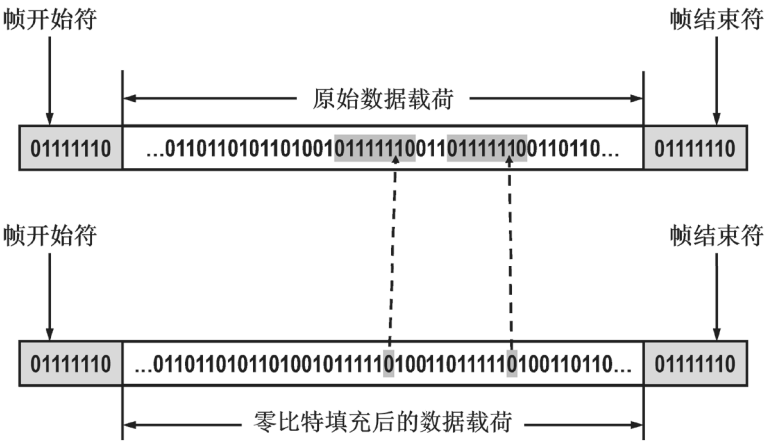


图3-8　采用比特填充法实现透明传输

请读者注意，本节介绍的字符填充法和比特填充法只是实现透明传输的一般原理性方法。各种数据链路层协议都有其实现透明传输的具体方法，其中有的是基于字符填充法或比特填充法的，而有的并未采用这两种方法。

以太网的数据链路层协议没有采用字符填充法或比特填充法来实现透明传输。这是因为在

以太网帧的首部和尾部中，并没有包含帧定界符，因此并不存在透明传输的问题。然而没有帧定界符的情况下，接收方的数据链路层又是如何从物理层交付的比特流中提取出一个个以太网帧的呢？

实际上，以太网的数据链路层封装好以太网帧后，将其交付给物理层。物理层还会在以太网帧前添加8字节的前导码，如图3-9所示。前导码中的前7个字节为前同步码，作用是使接收方的时钟同步，之后的1字节为帧开始符，表明其后面紧跟着的就是以太网帧。另外，以太网还规定了帧间间隔时间为96比特的发送时间（对于带宽为10Mb/s的传统以太网，96比特的发送时间为9.6μs）。因此，以太网帧并不需要帧结束符。

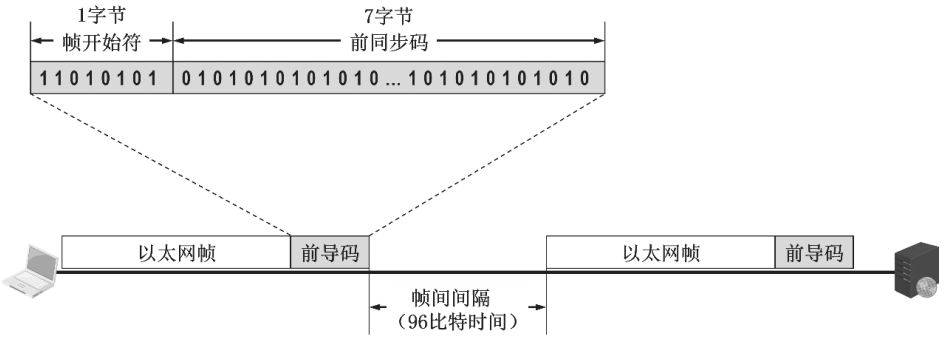


图3-9　以太网帧的前导码和帧间间隔

3.2.2　差错检测

实际的通信链路都不是理想的。表示比特的信号在信道上传输时，不可避免地会产生失真，甚至出现更严重的错误。这就会导致比特在传输过程中产生差错：比特1可能会变成比特0，而比特0也可能变成比特1。这种传输差错称为**比特差错**。

在某段时间内，出现传输错误的比特数量占这段时间内传输比特总数量的比例，称为**误码率**（Bit Error Rate，BER）。提高链路的信噪比，可以降低误码率。但在实际的通信链路上，不可能使误码率下降到零。

接收方的数据链路层从物理层交付的比特流中提取出一个帧后，如何知道这个帧在传输过程中是否出现了误码呢？这就需要采用差错检测措施。例如，发送方的数据链路层采用某种检错技术，根据帧的内容计算出一个检错码，将检错码填入帧尾部。接收方的数据链路层从帧尾部取出检错码，采用与发送方相同的检错技术，就可以通过检错码检测出帧在传输过程中是否出现了误码。帧尾部中用来存放检错码的字段称为**帧检验序列**（Frame Check Sequence，FCS）。

下面介绍两种常用的检错技术：**奇偶校验**和**循环冗余校验**。

1. 奇偶校验

奇校验是在待发送的数据后面添加1个校验位，使整个数据（包括所添加的校验位在内）中“1”**的个数为奇数**。

偶校验是在待发送的数据后面添加1个校验位，使整个数据（包括所添加的校验位在内）中“1”**的个数为偶数**。

下面举例说明奇偶校验。如图3-10所示，发送方要给接收方发送的比特流为101101，各个例子的情况说明如下。

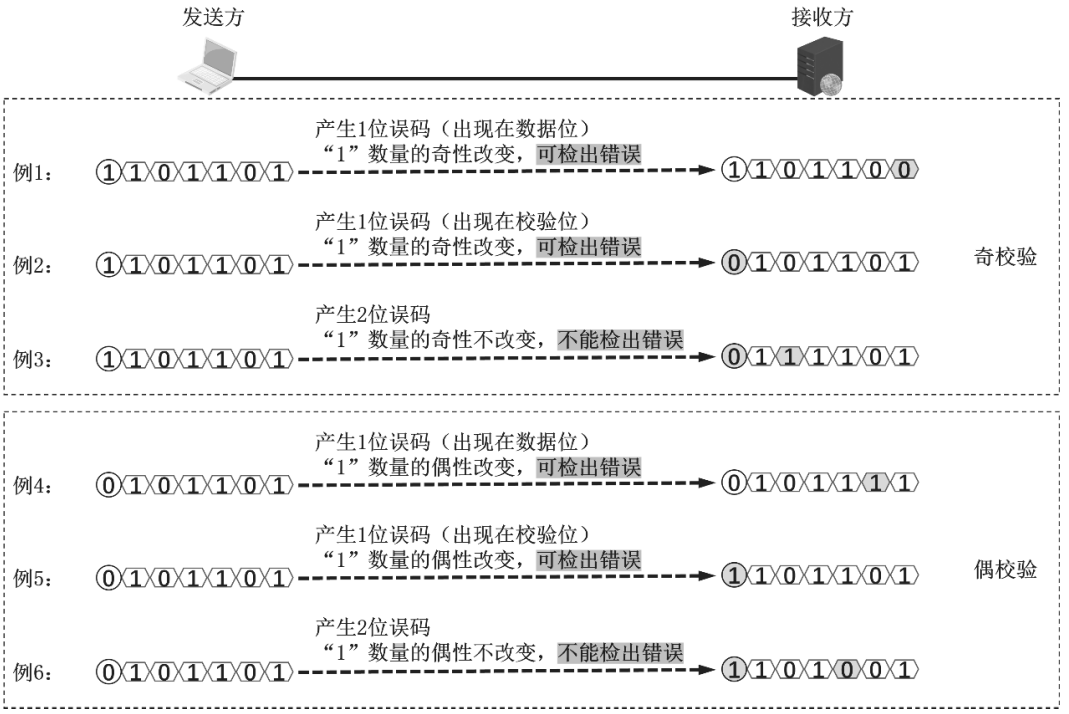


图3-10　奇偶校验举例

例1、例2和例3是收发双方约定进行奇校验的情况。为了使待发送比特流添加1个校验位后所包含“1”的个数为奇数，所添加的校验位应为1。

例4、例5和例6是收发双方约定进行偶校验的情况。为了使待发送比特流添加1个校验位后所包含“1”的个数为偶数，所添加的校验位应为0。

例1展示了在数据位产生1位误码的情况，这使得整个数据中“1”的数量的奇性发生改变，因此接收方可以检测出误码。

例2展示了校验位误码的情况，这使得整个数据中“1”的数量的奇性发生改变，因此接收方可以检测出误码。

例3展示了出现2位误码的情况，这使得整个数据中“1”的数量的奇性不发生改变，因此接收方无法检测出误码。

例4、例5和例6展示了偶校验的情况，与例1、例2和例3所展示的奇校验的情况类似，就不再赘述了，请读者自行分析。

通过上述例子可以看出：在所传输的数据中，如果**有奇数个位发生误码**，则所包含“1”的数量的奇偶性会发生变化，**可以检测出误码**；如果**有偶数个位发生误码**，则所包含“1”的数量的奇偶性不会发生变化，**不能检测出误码**，也就是**漏检**。

在实际使用时，奇偶校验又可分为垂直奇偶校验、水平奇偶校验以及水平垂直奇偶校验，有兴趣的读者可自行查阅相关资料。

2. 循环冗余校验

目前，在计算机网络的数据链路层，广泛使用漏检率极低的**循环冗余校验**（Cyclic Redundancy Check，CRC）检错技术。

循环冗余校验CRC的基本思想如下：

- 收发双方约定好一个**生成多项式**$G(X)$。
- 发送方基于待发送的数据和生成多项式$G(X)$，计算出差错检测码，即**冗余码**，将冗余码添加到待发送数据的后面一起传输。
- 接收方收到数据和冗余码后，通过生成多项式$G(X)$来计算收到的数据和冗余码是否产生了误码。

1）发送方使用CRC的操作

如图3-11所示，**采用二进制模2除法来计算余数**。二进制模2除法既不向上位借位，也不比较除数和被除数的对应位数值的大小，只要以相同位数进行相除即可，相当于**对应位进行逻辑异或运算**。具体步骤如下：

（1）将待发送的数据作为被除数的一部分，后面添加生成多项式$G(X)$最高次个0以构成被除数。

（2）生成多项式$G(X)$各项系数构成的比特串作为除数。

（3）进行二进制模2除法，得到商和余数。余数就是所计算出的冗余码。冗余码的长度应与生成多项式$G(X)$最高次数相同。

（4）将冗余码添加到待发送数据的后面一起发送。

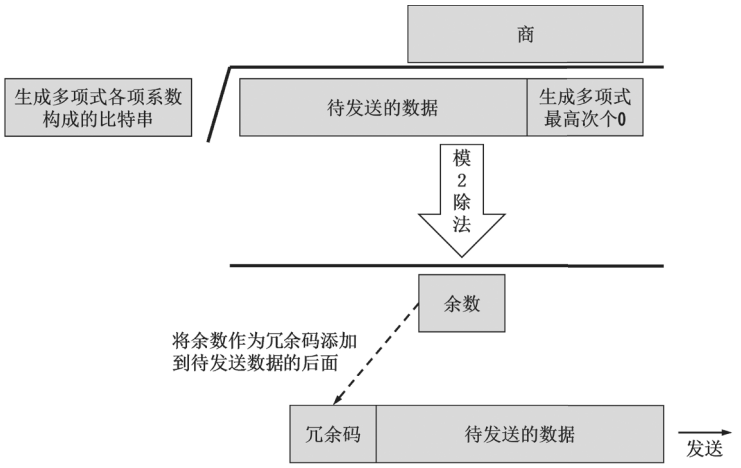


图3-11　发送方使用CRC的操作

2）接收方使用CRC的操作

如图3-12所示，采用二进制模2除法来计算余数。具体步骤如下：

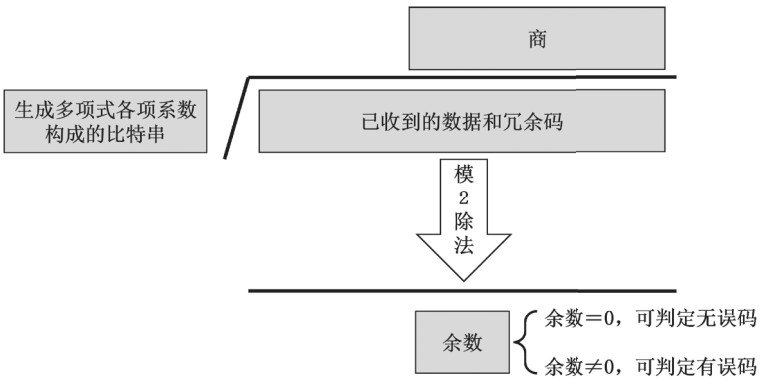


图3-12　接收方使用CRC的操作

（1）将收到的数据和冗余码作为被除数。

（2）生成多项式$G(X)$各项系数构成的比特串作为除数。

（3）进行二进制模2除法，得到商和余数。

（4）如果余数为0，就可判定数据和冗余码中未出现误码。如果余数不为0，则可判定数据或冗余码中出现了误码。

下面通过一个简单的例子进一步说明循环冗余校验的原理。

假设待发送的数据为101001，收发双方约定好的生成多项式为$G(X)=X^3+X^2+1$。

图3-13展示了发送方的操作步骤：

（1）**构造被除数**：在待发送数据101001后面添加生成多项式最高次数个0（最高次数为3），得到被除数101001000。

（2）**构造除数**：生成多项式$G(X) = X^3 + X^2 + 1$各项系数构成的比特串作为除数。$G(X)= X^3 + X^2+1 = 1 \cdot X^3 + 1 \cdot X^2 + 0 \cdot X^1 + 1 \cdot X^0$，得到除数为1101。

（3）**进行二进制模2除法**：二进制模2除法既不向上位借位，也不比较除数和被除数的对应位数值的大小，只要以相同位数进行相除即可，相当于对应位进行逻辑异或运算。计算出余数为1。

（4）**检查余数**：余数的位数应与生成多项式$G(X)$的最高次数相同。如果位数不够，则在余数前补0来凑足位数。在余数1前面补两个0，得到余数001。

（5）**将余数作为冗余码添加到待发送数据的后面进行发送**：将余数001作为冗余码添加到待发送数据101001的后面进行发送，即实际发送数据为101001001。

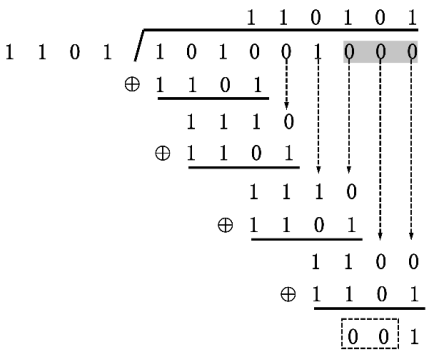


图3-13　发送方CRC举例

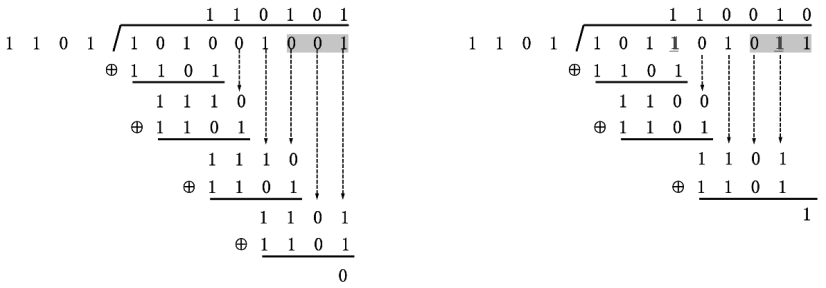
图3-14展示了接收方的操作步骤：

（1）**构造被除数**：接收到的数据和冗余码作为被除数。假设传输过程没有产生误码，则被除数为101001001，如图3-14（a）所示。假设传输过程产生了两位误码，例如收到的数据和冗余码为101101011，如图3-14（b）所示。

（2）**构造除数**：生成多项式$G(X) = X^3 + X^2 + 1$各项系数构成的比特串作为除数。$G(X)= X^3 + X^2 + 1 = 1 \cdot X^3 + 1 \cdot X^2 + 0 \cdot X^1 + 1 \cdot X^0$，得到除数为1101。

（3）**进行二进制模2除法，计算出余数**。

（4）**检查余数**：余数为0，可判定传输过程没有产生误码，如图3-14（a）所示。余数不为0，可判定传输过程产生了误码，如图3-14（b）所示。



(a) 余数为0，可判定传输过程无误码　　(b) 余数不为0，可判定传输过程产生误码

图3-14　接收方CRC举例

生成多项式$G(X)$直接关系到循环冗余校验的漏检率。在实际应用中，常采用以下几种已成为国际标准的生成多项式$G(X)$：

$\text{CRC}-16 = X^{16} + X^{15} + X^2 + 1$

$\text{CRC}-\text{CCITT} = X^{16} + X^{12} + X^5 + 1$

$\text{CRC}-32 = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

采用上述生成多项式$G(X)$进行循环冗余校验，出现漏检的概率是非常非常低的。请读者注意：**循环冗余校验要求生成多项式中必须包含最低次项X^0（也就是1）**。

接收方的数据链路层使用循环冗余校验差错检测技术，可以检测出接收到的帧在传输过程中是否误码，但无法检测出究竟是帧中的哪个或哪些比特出现了误码。因此，接收端的数据链路层也就无法纠正误码。需要说明的是，循环冗余校验只有在选择合适的生成多项式并在满足一定条件的前提下，才能定位单个比特的误码，进而对这个误码的比特进行纠错。因此，循环冗余校验码一般被划归为**检错码**，而不是**纠错码**。要想纠正帧在传输过程中产生的误码，可以使用冗余信息更多的纠错码（例如海明码）进行前向纠错。但纠错码的开销比较大，在计算机网络中较少使用。

循环冗余校验有**很好的检错能力**，虽然**软件计算比较复杂**，但**非常易于用硬件实现**，因此被广泛应用于计算机网络的数据链路层。

差错检测的目的在于辅助接收方判断所收到的数据帧是否产生了误码，为了实现差错检测，需要在数据传输中加入冗余比特。这种冗余比特就像我们生活中的“未雨绸缪”，在面对未知的风险时，我们需要有预见性地做好准备，以防止可能出现的问题。正如在我们的工作和学习中，也需要时刻保持高度的责任感和使命感，提前做好准备和规划，才能在面对挑战时从容应对，确保工作的顺利进行。

3.2.3　可靠传输

1. 可靠传输的相关基本概念

使用差错检测技术（例如循环冗余校验），接收方的数据链路层就可检测出帧在传输过程中是否产生了误码（比特错误）。如果检测出了误码，那么接下来该如何处理呢？这取决于数据链路层向其上层提供的服务类型。

- 若数据链路层向其上层提供的是**不可靠传输服务**，则接收方的数据链路层丢弃有误码的帧即可，其他什么也不用做。
- 若数据链路层向其上层提供的是**可靠传输服务**，这就需要数据链路层通过某种机制实现发送方发送什么，接收方就能收到什么。