第 3 章　数据链路层

在本章，我们将学习网络模型中的第二层（即数据链路层）的设计原则。学习内容涉及两台相邻机器实现可靠有效的完整信息块（称为帧）通信的一些算法，而不像物理层那样只关注单个比特传输。这里的相邻指两台机器通过一条通信信道连接起来，通信信道在概念上就像一条线路（比如同轴电缆、电话线或者无线信道）。信道像一条线路的本质特性使得信道上传递的比特顺序与发送顺序完全相同。

刚开始，你可能认为这个问题非常简单，似乎没有什么内容需要学习——机器 A 把比特放到线路上，然后机器 B 将这些比特取下来。不幸的是，通信线路偶尔会出错。而且，它们只有有限的数据传输率，并且在比特的发送时间和接收时间之间存在一个非零延迟。这些限制对数据传输的效率有非常重要的影响。通信所采用的协议必须考虑所有这些因素。这些协议正是本章的主题。

在介绍了数据链路层的关键设计问题之后，我们将通过考察错误的本质以及如何检测和纠正这些错误来开始数据链路层协议的学习。然后，我们将学习一系列复杂性逐步递增的协议，每个协议解决了本层中越来越多的问题。最后，我们将给出一些数据链路层协议的例子来结束本章。

3.1　数据链路层的设计问题

数据链路层使用物理层提供的服务在通信信道上发送和接收比特。它要完成一些功能，包括：

（1）向网络层提供一个定义良好的服务接口。

（2）处理传输错误。

（3）调节数据流，确保慢速的接收方不会被快速的发送方淹没。

为了实现这些目标，数据链路层从网络层获得数据包，然后将这些数据包封装成**帧**（frame）以便传输。每个帧包含一个帧头、一个有效载荷（用于存放数据包）以及一个帧尾，如图 3-1 所示。帧的管理构成了数据链路层工作的核心。在后面的章节中，我们将详细地讨论上面提到的这些问题。

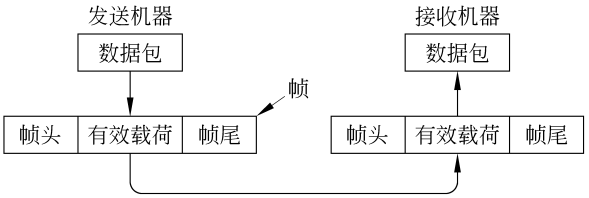

图 3-1　数据包和帧的关系

虽然本章明确讨论数据链路层及其协议，但是，我们在本章中学习的许多原理，比如错误控制和流量控制等同样可以在传输层和其他协议中寻觅到类似的踪迹。这是因为可靠性是网络的总目标，这个目标的实现需要各层次的紧密配合。实际上，在许多网络中，这些功能最常出现的地方是上层，数据链路层只要做很少的一点工作就已经“足够好”。然而，不管它们出现在哪里，原理是非常一致的。在数据链路层中，它们通常表现出最为简单和纯粹的形式，因此，数据链路层是详细学习这些原理的绝佳之地。

3.1.1 提供给网络层的服务

数据链路层的功能是为网络层提供服务。最主要的服务是将数据从源机器的网络层传输到目标机器的网络层。在源机器的网络层有一个实体（称为进程），它将一些比特交给数据链路层，要求传输到目标机器。数据链路层的任务就是将这些比特传输给目标机器，然后再进一步交付给网络层，如图 3-2（a）所示。实际的传输过程则是沿着图 3-2（b）所示的路径进行的，但很容易将这个过程想象成两个数据链路层的进程使用一个数据链路协议进行通信。基于这个原因，在本章中我们将隐式使用图 3-2（a）的模型。

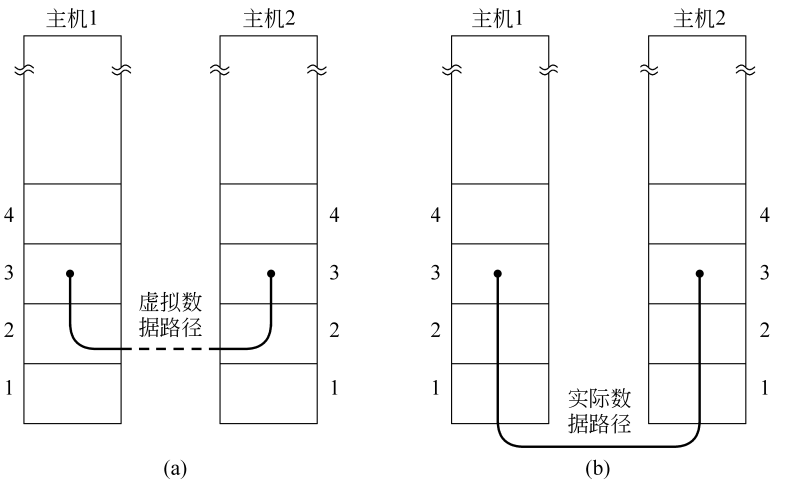

图 3-2
（a）虚拟通信；（b）实际通信

数据链路层可以设计成向上提供各种不同的服务。实际提供的服务因具体协议的不同而有所差异。一般情况下，数据链路层通常会提供以下 3 种可能的服务：

（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。

无确认的无连接服务是指源机器向目标机器发送独立的帧，目标机器并不对这些帧进行确认。以太网就是一个提供此类服务的数据链路层极好实例。采用这种服务，事先不需要建立逻辑连接，事后也不用释放逻辑连接。若由于线路的噪声而造成了某一帧的丢失，数据链路层并不试图去检测这样的丢帧情况，更不会去试图恢复丢失的帧。这类服务合适两种场合，第一种是错误率很低的场合，此时差错恢复过程可以留给上层来完成；第二种

是实时通信，比如语音传输，因为在实时通信中数据迟到比数据受损更糟糕。

迈向可靠性的下一步是有确认的无连接服务。当向网络层提供这种服务时，数据链路层仍然没有使用逻辑连接，但其发送的每一帧都需要单独确认。这样，发送方可知道一个帧是否已经正确地到达目的地。如果一个帧在指定的时间间隔内还没有到达，则发送方将再次发送该帧。这类服务尤其适用于不可靠的信道，比如无线系统。802.11（WiFi）就是此类服务的一个很好例子。

或许有一点值得强调，那就是在数据链路层提供确认只是一种优化手段，永远不应该成为一种需求。网络层总是可以发送一个数据包，然后等待该数据包被确认。如果在计时器超时之前，该数据包的确认还没有到来，那么发送方只要再次发送整个报文即可。这一策略的麻烦在于它可能导致传输的低效率。链路层对帧通常有严格的长度限制，这是由硬件所决定的；除此之外，还有传播延迟。但网络层并不清楚这些参数。网络层可能发出了一个很大的数据包，该数据包被拆分并封装到（比如说）10 个帧中，而且 20%的帧在传输中被丢失，那么这个数据包可能需要花很长的时间才能传到接收方。相反地，如果每个帧都单独确认和必要时重传，那么出现的差错就能更直接并且更快地被检测到。在可靠信道上，比如光纤，重量级数据链路协议的开销可能是不必要的；但在无线信道上，由于信道内在的不可靠性，这种开销还是非常值得的。

我们再回到有关服务的话题上，数据链路层向网络层提供的最复杂服务是面向连接的服务。采用这种服务，源机器和目标机器在传输任何数据之前要建立一个连接。连接上发送的每一帧都被编号，数据链路层确保发出的每个帧都会真正被接收方收到。它还保证每个帧只被接收一次，并且所有的帧都将按正确的顺序被接收。因此，面向连接的服务相当于为网络层进程提供了一个可靠的比特流。它适用于长距离且不可靠的链路，比如卫星信道或者长途电话电路。如果采用有确认的无连接服务，可以想象丢失了确认可能导致一个帧被收发多次，因而将浪费带宽。

当使用面向连接的服务时，数据传输必须经过三个不同的阶段。在第一个阶段，要建立连接，双方初始化各种变量和计数器，这些变量和计数器记录了哪些帧已经接收到，哪些帧还没有收到。在第二个阶段，才真正传输一个或者多个数据帧。在第三个也是最后一个阶段中，连接被释放，所有的变量、缓冲区以及其他用于维护该连接的资源也随之被释放。

3.1.2　成帧

为了向网络层提供服务，数据链路层必须使用物理层提供给它的服务。物理层所做的只是接收一个原始比特流，并试图将它传递给目标机器。如果信道上存在噪声，就像大多数无线链路和某些有线链路那样，物理层就会在它的信号中添加某种冗余，以便将误码率降到一定程度。然而，数据链路层接收到的比特流不能保证没有错误。某些比特的值可能已经发生变化，接收到的比特个数可能少于、等于或者多于发送的比特数量。检测错误和纠正错误（有必要的话）的工作正是数据链路层该做的。

对于数据链路层来说，通常的做法是将比特流拆分成多个离散的帧，为每个帧计算一个称为校验和的短令牌（本章后面将讨论校验和算法），并将该校验和放在帧中一起传输。当帧到达目标机器时，要重新计算该帧的校验和。如果新算出来的校验和与该帧中包含的

校验和不同，则数据链路层知道传输过程中产生了错误，它就会采取措施来处理错误（比如丢掉坏帧，可能还会发回一个错误报告）。

拆分比特流的实际工作比初看上去的要复杂得多。一个好的设计方案必须使接收方很容易发现一个新帧的开始，同时所使用的信道带宽要少。我们将考察下列 4 种方法：

（1）字节计数法。

（2）字节填充的标志字节法。

（3）比特填充的标志比特法。

（4）物理层编码违禁法。

第一种成帧方法利用头部中的一个字段来标识该帧中的字符数。当接收方的数据链路层看到字符计数值时，它就知道后面跟着多少个字节，因此也就知道了该帧在哪里结束。这项技术如图 3-3（a）所示，其中 4 帧的大小分别为 5、5、8 和 8 个字节。

这个算法的问题在于计数值有可能因为一个传输错误而被弄混。例如，如果第 2 帧中的计数值 5 由于一个比特反转而变成了 7，如图 3-3（b）所示，则接收方就会失去同步，它再也不可能找到下一帧的正确起始位置。即使校验和不正确，接收方知道该帧已经被损坏，它仍然无法知道下一帧从哪里开始。在这种情况下，给发送方发回一个帧，要求重传也无济于事，因为接收方并不知道应该跳过多少个字节才能到达重传的开始处。正是由于这个原因，字节计数方法本身很少被使用。

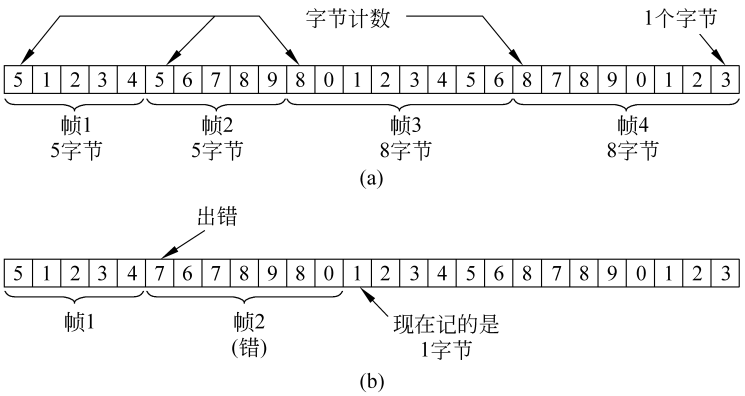

图 3-3　字节流

（a）没有错误；（b）有一个错误

第二种成帧方法考虑到了出错之后的重新同步问题，它让每个帧用一些特殊的字节作为开始和结束。这些特殊字节通常都相同，称为**标志字节**（flag byte），作为帧的起始和结束分界符，如图 3-4（a）中的 FLAG 所示。两个连续的标志字节代表了一帧的结束和下一帧的开始。因此，如果接收方丢失了同步，它只需搜索两个标志字节就能找到当前帧的结束和下一帧的开始位置。

然而，还有问题必须要解决。当标志字节出现在数据中时，尤其是当传输二进制数据（比如照片或歌曲）时，这种情景往往会严重干扰到帧的分界。有一种方法可以解决这个问题，发送方的数据链路层在数据中“偶尔”出现的每个标志字节的前面插入一个特殊的转义字节（ESC）。因此，只要看它数据中标志字节的前面有没有转义字节，就可以把作为帧分界符的标志字节与数据中出现的标志字节区分开来。接收方的数据链路层在将数据传递

给网络层之前必须删除转义字节。这种技术就称为**字节填充**（byte stuffing）。

当然，接下来的问题就是：如果转义字节也出现在数据中，那该怎么办？答案是同样用字节填充技术，即用一个转义字节来填充。在接收方，第一个转义字节被删除，留下紧跟在它后面的数据字节（或许是另一个转义字节或者标志字节）。图 3-4（b）给出了一些例子。在所有情况下，去掉填充字节之后递交给网络层的字节序列与原始的字节序列完全一致。我们仍然可以通过搜索两个标志字节来定位帧的边界，无须顾虑撤销转义字节的原意。

图 3-4 中描述的字节填充方案是 **PPP 协议**（Point-to-Point Protocol）使用的略微简化形式，该协议通常用在通信链路上传送数据包。我们将在本章后面讨论 PPP 协议。

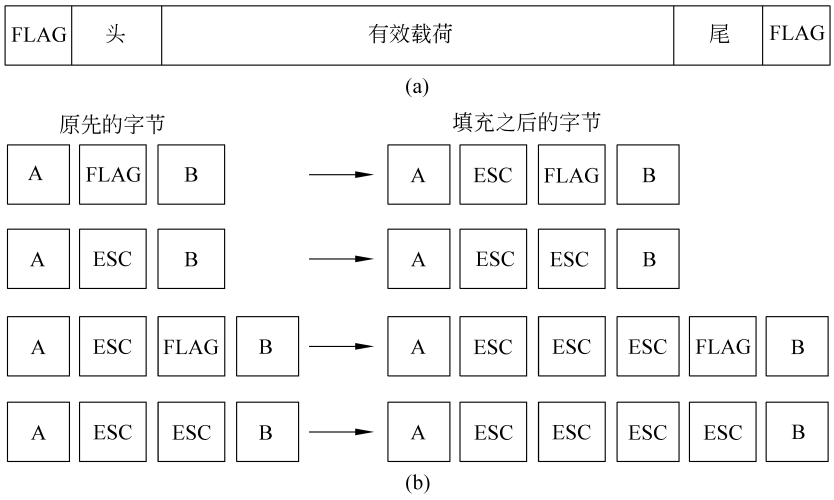

图　3-4

（a）由标志字节分界的帧；（b）字节填充之前和之后的字节序列示例

第三种区分比特流边界的方法考虑了字节填充的缺点，即只能使用 8 比特的字节。帧的划分可以在比特级完成，因而帧可以包含由任意大小单元（而不是只能以 8 比特为单元）组成的二进制比特数。这种方法是为曾经非常流行的 HDLC（高级数据链路控制）协议而开发的。每个帧的开始和结束由一个特殊的比特模式，01111110 或十六进制 0x7E 标记。这种模式是一个标志字节。每当发送方的数据链路层在数据中遇到连续五个 1，它便自动在输出的比特流中填入一个比特 0。这种比特填充类似于字节填充，在数据字段的标志字节之前插入一个转义字节到出境字符流中。比特填充还确保了转换的最小密度，这将有助于物理层保持同步。正是由于这个原因，USB（通用串行总线）采用了比特填充技术。

当接收方看到 5 个连续入境比特 1，并且后面紧跟一个比特 0，它就自动剔除（即删除）比特 0。比特填充和字节填充一样，对两台计算机上的网络层是完全透明的。如果用户数据中包含了标志模式 01111110，这个标志传输出去的是 011111010，但在接收方内存中存储还是 01111110。图 3-5 给出了一个比特填充的例子。

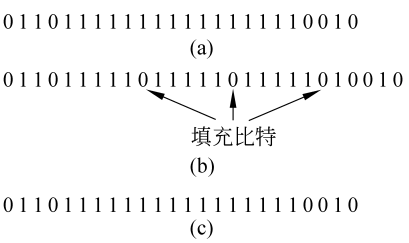

图 3-5　比特填充

（a）原始数据；（b）出现在线路上的数据；（c）存储在接收方内存中的数据

有了比特填充技术，两帧之间的边界可以由标志模式明确区分。因此，如果接收方失去了它的接收轨迹，它所要做的只是扫描输入比特流，找出其中的标志序列；因为这些标志只可能出现在帧的边界，而决不会出现在帧内的数据中。

采用比特填充和字节填充的一个副作用是一帧的长度现在要取决于它所携带的数据内容。例如，如果数据中没有标记字节，100 个字节或许被一个大约长为 100 字节的帧所携带。然而，如果数据完全由标志字节组成，每个标志字节都要被转义，那么最后发送出去帧的长度就变成大约 200 个字节。采用比特填充技术，帧的长度增幅大约为 12.5%，因为每个字节增加 1 个比特。

成帧的最后一种方法是一条使用物理层的捷径。我们在第 2 章看到比特编码成信号通常包括一些冗余比特，以便帮助接收器同步接收。这种冗余意味着一些信号将不会出现在常规数据中。例如，在 4B/5B 线性编码模式下，4 个数据位被映射成 5 个信号比特，通过这种方法确保线路上的信号有足够的跳变。这意味着 32 个可能的信号中有 16 个是不会被使用的。我们可以利用这些保留的信号来指示帧的开始和结束。实际上，我们使用“编码违法”来区分帧的边界。这种方案的优点在于，因为这些用作分界符的信号是保留不用的，所以很容易通过它们找到帧的开始和结束，而且不再需要填充数据。

许多数据链路协议为安全起见综合使用了这些方法。以太网和 802.11 使用了共同的分界模式，即用一个定义良好的比特模式来标识一帧的开始，该比特模式称为**前导码**（preamble）。这种定界模式可能很长（802.11 典型使用 72 位），目的是让接收方准备接收输入的数据包。前导码之后是头的长度字段（即计数），这个字段将被用来定位帧的结束处。

3.1.3 差错控制

解决了如何标识每一帧的起始和结束位置之后，我们现在来看下一个问题：如何确保所有的帧最终都被传递给目标机器的网络层，并且保持正确的顺序。现在假设接收方可以知道它收到的帧包含了正确的或者错误的信息（我们将在 3.2 节考察用于检测和纠正传输错误的编码）。对于无确认的无连接服务，不管发出去的帧是否正确抵达目标机器，发送方只要把出境帧留存就可以了。但是对于可靠的、面向连接的服务，这样做肯定还远远不够。

确保可靠传递的常用方法是向发送方提供一些有关线路另一端状况的反馈信息。通常情况下，协议要求接收方发回一些特殊的控制帧，在这些控制帧中，对于它所接收到的帧进行肯定的或者否定的确认。如果发送方收到了关于某一帧的肯定确认，那么它就知道这帧已经安全地到达了。另一方面，否定的确认意味着传输过程中产生了错误，所以这帧必须重传。

更为复杂的是存在这样的可能性，有时候由于硬件的问题，一个帧被完全丢失了（比如一个突发噪声）。在这种情况下，接收方根本不会有任何反应，因为它没有根据做出反应。类似地，如果确认帧丢失，发送方也不知道该如何处理。显然，如果在一个协议中，发送方发出了一帧之后就等待肯定的或者否定的确认，那么，若由于硬件故障或通信信道出错等原因而丢失了某一帧，则发送方将永远等待下去。

这种可能性可以通过在数据链路层中引入计时器来解决。当发送方发出一帧时，通常

还要启动一个计时器。该计时器的超时值应该设置得足够长，以便保证在正常情况下该帧能够到达接收方，并且在接收方进行处理后再将确认返回到发送方。一般情况下，在计时器超时前，该帧应该被正确地接收，并且确认帧也被传了回来。这种情况下，计时器被取消。

然而，如果帧或者确认被丢失，则计时器将被触发，从而警告发送方存在一个潜在的问题。一种显然的解决方案是重新发送该帧。然而，当有的帧被发送了多次之后，可能会出现这样的危险：接收方将两次或者多次接收到同一帧，并且多次将它传递给网络层。为了避免发生这样的情形，一般有必要给发送出去的帧分配序号，这样接收方可以根据帧的序号来有效区分原始帧和重传帧。

管理好计时器和序号，以便保证每一帧最终都恰好一次地被传递给目标机器的网络层，这是数据链路层（以及上层）工作的重要组成部分。在本章后面，我们将通过一系列复杂性逐渐增加的例子，来考察如何做好计时器和序号的管理工作。

3.1.4　流量控制

在数据链路层（以及更高的各层）中，另一个重要的设计问题是如果发送方发送帧的速度超过了接收方能够接受这些帧的速度，发送方该如何处理。当发送方运行在一台高速并能量强大的计算机上，而接收方运行在一台慢速并且低端机器上时，这种情况很容易发生。一种常见的场景是一个智能手机向一个超强服务器请求一个 Web 页面。这就像突然打开了消防栓一样，大量的数据涌向可怜无助的手机，直到它被彻底淹没。即使传输过程不会出错，接收方也无法以数据到来的速度那样快地处理持续到来的帧，此时必然会丢弃一些帧。

很显然，必须要采取某种措施来阻止这种情况发生。常用的办法有两种。第一种方法是**基于反馈的流量控制**（feedback-based flow control），接收方给发送方返回信息，允许它发送更多的数据，或者至少告诉发送方自己的情况怎么样。第二种方法是**基于速率的流量控制**（rate-based flow control），使用这种方法的协议有一种内置的机制，它能限制发送方传输数据的速率，而无须利用接收方的反馈信息。

在本章，我们将学习基于反馈的流量控制方案，因为基于速率的方案仅在传输层（第 5 章）的一部分中可见，而基于反馈的方案则可同时出现在链路层和更高的层次。后者在近来较为常见，在这种情况下，链路层硬件设计的运行速度足够快到不会造成丢帧。例如，作为链路层硬件实现的**网络接口卡**（NIC，Network Interface Cards），有时声称能以“线速”运行，这意味着它们能以帧到达的速度来处理帧。因而，任何过载不再是链路层的问题，它们必须由高层来处理。

基于反馈的流控制方案有许多种，但是绝大多数使用了同样的基本原理。协议包含了许多定义良好的规则，这些规则规定了发送方什么时候可以发送下一帧。这些规则通常在没有得到接收方许可（隐式或者显式的）之前，禁止继续发送帧。例如，当建立一个连接时，接收方可能会这样说：“你现在可以给我发送 n 个帧，但是在发送完 n 个帧之后就别再发送，直到我告诉你可以继续发送。”稍后我们将讨论这些细节。

3.2 差错检测和纠正

正如我们在第 2 章中所看到的那样，通信信道有许多不同的特征。某些信道，例如电话系统中的光纤，其错误率很低，因而很少发生传输错误。但是其他信道，尤其是无线链路和老化的本地回路错误率高出光纤好几个数量级。对于这些信道而言，传输出错是常态。从性能的角度来看，这些错误不能在合理的成本开销内彻底解决。因此结论是传输错误非常普遍。我们必须知道如何处理传输错误。

网络设计者针对错误处理已经研究出两种基本策略。这两种策略都在发送的数据中加入冗余信息。一种策略是在每一个被发送的数据块中包含足够多的冗余信息，以便接收方能据此推断出被发送的数据是什么。另一种策略也是包含一些冗余信息，但这些信息只能让接收方推断出是否发生了错误（而推断不出哪个发生了错误），然后接收方可以请求发送方重传。前一种策略使用了**纠错码**（error-correcting code），后一种策略使用了**检错码**（error-detecting code）。使用纠错码的技术通常也称为**前向纠错**（FEC，Forward Error Correction）。

这里的每一项技术都占据着不同的生态位置。在高度可靠的信道上（比如光纤），较为合算的做法是使用检错码，当偶尔发生错误时只需重传整个数据块。然而，在错误发生很频繁的信道上（比如无线链路），更好的做法是在每一个数据块中加入足够的冗余信息，以便接收方能够计算出原始的数据块。FEC 被用在有噪声的信道上，因为重传的数据块本身也可能像第一次传输那样出错。

这些编码的一个关键考虑是可能发生的错误类型。无论是纠错码还是检错码都无法处理所有可能的传输错误，因为提供保护措施的冗余比特很可能像数据比特一样出现错误（可危及它们的保护作用）。如果信道给予冗余比特的待遇好于数据比特那当然好，可惜事实并非如此。对信道而言，它们都是同样的比特。这意味着为了避免漏检错误，编码必须强大到足以应付预期的错误。

错误模型有两种。在第一种错误模型中，偶尔出现的极端热噪声快速淹没了信号，引起孤立的单个比特错误。在另一种错误模型中，传输错误往往呈现突发性而不以单个形式出现，这种错误源自物理过程，比如无线信道上的一个深衰落，或者有线信道上的瞬态电气干扰。

两种错误模型实际上造成的问题以及处理方式有不同的权衡。突发的错误相比单个比特错误有自己的优势，也有不足。从优势方面来看，计算机数据总是成块发送。假设数据块大小为 1000 个比特，误差率为每比特 0.001。如果错误是独立的，大多数块将包含一个错误。但如果错误以 100 个比特的突发形式出现，则平均来说 100 块中只有一块会受到影响。突发错误的缺点在于当它们发生时比单个错误更难以纠正。

同时还存在着其他类型的错误。有时候，一个错误的位置可以获得，或许因为物理层接收到的一个模拟信号远离了 0 或 1 的预期值，因而可以宣布该比特被丢失。这种情形称为**擦除信道**（erasure channel）。擦除信道比那些把比特值翻转的信道更易于纠错，因为即使某个比特被丢失，至少我们还能知道哪个比特出了错。然而，我们往往不能从信道的擦

除性质上受益。

下面我们将同时考察纠错码和检错码。请记住两点。首先，我们在链路层讨论这些编码有客观因素，因为这里是面临数据可靠传输相关问题的首要地方。然而，这些编码方案被广泛使用的根本原因在于可靠性是整个系统所关注的问题。纠错码也会出现在物理层，特别是有噪声干扰的信道，同时还会出现在更高的层次，特别是实时流媒体应用和内容分发应用。检错码更是经常被用在链路层、网络层和传输层。

第二点要记住的是差错编码是应用的数学。除非特别熟悉伽罗瓦（Galois）领域或稀疏矩阵的性质，否则，应该从可靠的来源获得性质更优良的编码，而不是自己设计编码方案。事实上，这正是为何很多协议标准一次又一次采用了相同的编码方法。在下面的材料中，我们将学习一个简单的编码方法，然后再简要描述先进的编码方法。这样，我们可以从简单编码中来理解如何权衡，并且通过先进编码来讨论实际使用的编码方案。

3.2.1　纠错码

我们将考察以下 4 种不同的纠错编码：

（1）海明码。

（2）二进制卷积码。

（3）里德所罗门码。

（4）低密度奇偶校验码。

上述所有编码都将冗余信息加入到待发送的信息中。一帧由 m 个数据位（即信息）和 r 个冗余位（即校验）组成。在**块码**（block code）中，r 个校验位是作为与之相关的 m 个数据位的函数计算获得的，就好像在一张大表中找到 m 位数据对应的 r 校验位。在**系统码**（systematic code）中，直接发送 m 个数据位，然后发出 r 个校验位，而不是在发送前对它们进行编码。在**线性码**（line code）中，r 个校验位是作为 m 个数据位的线性函数被计算出来的。异或（XOR）或模 2 加是函数的流行选择，这意味着编码过程可以用诸如矩阵乘法或简单逻辑电路来完成。除非另有说明，我们在本节中考察的是线性码、系统块状码。

令数据块的总长度为 n（即 n=m+r）。我们将此描述为（n, m）码。一个包含了数据位和校验位的 n 位单元称为 n 位**码字**（codeword）。**码率**（code rate）或者简单地说速率，则定义为码字中不包含冗余部分所占的比例，或者用 m/n 表示。实际上这个码率变化很大。在一个有噪声信道上码率或许是 1/2，在这种情况下接收方所收到的信息中有一半是加入的冗余位；而在高品质的信道上码率接近 1，只有少数的校验位被添加到一个大块消息中。

为了理解发生传输错误后如何进行处理，有必要先来看一看错误到底是什么样的。给定两个被发送或接收的码字，比如说 10001001 和 10110001，完全可能确定这两个码字中有多少个对应位是不同的。此时，上述两个码字有 3 位不同。为确定有多少个不同位，只需 XOR 两个码字，并且计算结果中 1 的个数。例如：

$$\begin{array}{r} 10001001 \\ \underline{10110001} \\ 00111000 \end{array}$$

两个码字中不相同的位的个数称为**海明距离**（Hamming distance）（Hamming，1950）。

它的意义在于，如果两个码字的海明距离为 d，则需要 d 个 1 位错误才能将一个码字转变成另一个码字。

给定计算校验位的算法，完全可以构建一个完整的合法码字列表，然后从这个列表中找出两个具有最小海明距离的码字。这个距离就是整个编码的海明距离。

在大多数数据传输应用中，所有 2^m 种可能的数据报文都是合法的；但是，根据检验位的计算方法，并非所有 2^n 种可能的码字都会被用到。事实上，对于 r 校验位，可能的报文中只有很少一部分 $2^m/2^n$ 或 $1/2^r$ 是合法的码字。正是这种空间稀疏方法，即报文被嵌入到码字空间中，使得接收方能检测并纠正错误。

块码的检错和纠错特性跟它的海明距离有关。为了可靠地检测 d 个错误，需要一个距离为 d+1 的编码方案，因为在这样的编码方案中，d 个 1 位错误不可能将一个有效码字改变成另一个有效码字。当接收方看到一个无效码字时，它就知道发生了传输错误。类似地，为了纠正 d 个错误，需要一个距离为 2d+1 的编码方案，因为在这样的编码方案中，合法码字之间的距离足够远，即使发生了 d 位变化，结果也还是离它原来的码字最近。这意味着在不太可能有更多错误的假设下，可以唯一确定原来的码字，从而达到纠错的目的。

看一个纠错码实例，考虑一个只有下列 4 个有效码字的编码方案：

0000000000，　0000011111，　1111100000，　1111111111

该编码方案的距离是 5，这意味着它可以纠正 2 个错误或者检测双倍的错。如果接收到码字 0000000111 并且期望只有单个或者 2 个错误，则接收方知道原始的码字一定是 0000011111。然而，如果发生了三个错误，0000000000 变成了 0000000111，则以上编码就无法正确地纠正错误了。另外，如果我们期望所有这些错误都会发生，我们也可以检测出它们。只要没有收到合法的码字，就必然发生了错误。很明显在这个例子中，我们不能同时纠正 2 个错误和检测 4 个错误，因为这需要我们以两种不同的方式来解释接收到的码字。

在我们的例子中，解码的任务就是找出最接近接收码字的合法码字。不幸的是，大多数情况下全部码字都将作为候选被评估，这是一件非常耗时的搜索。相反，实际的代码被设计成允许使用快捷方式找出最有可能的原始码字。

设想我们要设计一种编码方案，每个码字有 m 个消息位和 r 个校验位，并且能够纠正所有的单个错误。对于 2^m 个合法消息，任何一个消息都对应有 n 个非法的码字，它们与该消息的距离为 1。这些非法的码字可以这样构成：将该消息对应的合法码字的 n 位，逐个取反，可以得到 n 个距离为 1 的非法码字。因此，每个 2^m 中的合法消息需要 n+1 个位模式来标识它们。由于总共只有 2^n 个位模式，所以，我们必须有 $(n+1)2^m \leqslant 2^n$。由 n=m+r，这个要求变成了

$$(m+r+1) \leqslant 2^r \qquad (3\text{-}1)$$

在给定 m 的情况下，这个条件给出了纠正单个错误所需要的校验位数的下界。

事实上这个理论下限可使用海明方法（1950）获得。在海明码中，码字的位被连续编号，从最左端的位开始，紧跟在右边的那位是 2，依次从左到右编号。2 的幂次方的位（1，2，4，8，16 等）是校验位，其余位（3，5，6，7，9 等）用来填充 m 个数据位。这种模式如图 3-6 所示的（11,7）海明码，其中 7 个数据位和 4 个校验位。每一个校验位强制进行模 2 加，或对某些位的集合，包括其本身进行偶（或奇）校验。一位可能被包括在几个校验位的计算中。若要查看在数据 k 位上的校验位，必须将 k 改写成 2 的幂之和。例