第 3 章

数据链路层

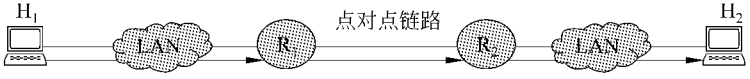
3.1 概述

1. 什么是数据链路

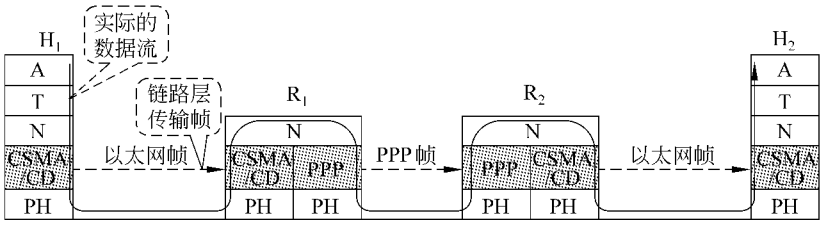
数据链路层的通信对等实体之间的数据传输通道称为数据链路(data link),它是一个逻辑概念,包括物理线路和必要的传输控制协议。

图 3.1(a)表示 H_1 向 H_2 发送数据,中间经过了 3 个数据链路,分别是 $H_1 \rightarrow R_1$、$R_1 \rightarrow R_2$ 和 $R_2 \rightarrow H_2$。其中,$R_1 \rightarrow R_2$ 是两个路由器间的点对点链路,数据链路层使用点对点协议(PPP)。另外两个是 LAN(如以太网)链路,属于广播链路,使用以太网的数据链路层协议即带冲突检测的载波监听多点接入(Carrier Sense Multiple Access with Collision Detection,CSMA/CD)。

图 3.1(b)中的实线表示实际的数据流动。在每个链路上,发送站的数据链路层将上层的数据封装成帧,通过物理层发送出去,接收站负责接收帧。图 3.1(b)中的虚线表示数据链路层上虚拟的帧传输,数据链路层的协议只作用在每个独立的链路上。



(a) H_1 向 H_2 发送数据



(b) 数据链路层上虚拟的帧传输

图 3.1 数据链路层负责在单个链路上的发送和接收结点之间传送的帧

2. 帧传输的基本问题

数据链路层负责在单个链路上的发送站和接收站之间传送以帧(frame)为 PDU 的数据。数据链路层有很多协议,但要实现正常的帧传输,有 3 个基本问题是共同的,即帧同步、透明传输和差错检验。

帧同步是为了使接收方能准确地判断一个帧的开始和结束,也就是帧定界。透明传

输要实现什么样的数据帧都可以被传输，是与帧定界相关的问题，我们将在 3.2 节介绍帧同步和透明传输的概念和方法，在 3.3 节介绍数据链路层的差错检验方法。

3. 帧传输的可靠性控制

数据链路层一个重要功能是在不可靠的物理链路上加上必要的控制规程，以实现帧的可靠传输。数据链路控制的方法称为自动请求重传（Automatic Repeat reQuest，ARQ），ARQ 综合了反馈重传机制和滑动窗口机制，进行差错控制和流量控制，实现数据的可靠传输。我们将在 3.4 节介绍 ARQ。

早年的网络传输的可靠性差，数据链路层设计了复杂的 ARQ 控制机制，当时数据链路层最重要的协议高级数据链路控制（HDLC）就实现了这些机制并曾广泛应用。但是，随着技术的发展，现代网络传输的可靠性已大大提高，数据链路层已不再使用这些复杂的控制机制，主要是由传输层负责传输的可靠性。然而这些流量控制和差错控制机制，不仅适用于单个链路的数据链路层，也可以用于跨越多个链路的端到端的传输层，TCP 协议就使用了这些机制用于传输层的可靠性传输，后面讲到 TCP 时，还将结合其特点在此基础上进一步介绍。

4. 广播链路的接入控制

图 3.1 中有两个 LAN 链路，它们是广播链路，是网上的所有结点共享的，但每一个时刻只能一个结点发送数据。数据链路层必须进行多个结点接入的协调控制，称为媒体接入控制（Medium Access Control，MAC），使得链路上的各个站点能够通过合理的竞争使用共享信道，这是广播链路数据链路层的一个非常重要的功能。不同类型的 LAN 有不同的 MAC 方式，这方面的内容将在第 4 章结合具体的 LAN 进行介绍。

3.2　帧同步和透明传输

3.2.1　帧同步

1. 同步传输

计算机网络的通信一般采用同步传输（synchronous transmission）方式。同步传输中，通信双方使用统一的定位时钟，数据是以称为帧（frame）的较大的数据块（比如以太网是 1518 个字节）为单位进行传送。

同步传输有面向字符和面向位两种方式。面向字符的同步传输方式是 20 世纪 60 年代采用的传输方式，如 ARPANET 的 IMP-IMP 协议和 IBM 的 BSC（Binary Synchronous Communication）规程。面向字符是指在通信链路上所传送的数据和控制信息都是由选定的字符集中的字符所组成的，如 ASCII 码字符集。面向字符的传输方式存在一些缺点，比如它要求所有的通信设备都要使用同样的字符集。目前普遍应用的是面向位的同步传输方式，它不限定传送的数据是某一字符集中的字符。

和同步传输方式对应的是异步传输（asynchronous transmission）方式，通信的双方各自使用独立的定位时钟，以字符为单位进行数据传输。著名的物理层规范 RS-232 接口，就是使用异步传输方式。

2. 帧同步

同步传输的PDU为帧,必须实现帧同步,即接收方能正确地判断发送方发出的每一个帧的开始和结束的位置,以便正确地接收这些帧。

第2章讲过,物理层接收比特流一般采用内同步方式实现位同步。显然帧同步是位同步的必要前提,只有明确比特流的首、尾的情况下,位同步才有意义。如果不知道比特流的首、尾,可能出现错过前面的一些位或丢掉后面的一些位等情况。

帧同步在数据链路层实现,实现的方式也比较简单,帧封装时使用特殊的帧定界符加在帧的首、尾两端,标志一个帧的开始和结束位置。首、尾帧定界符可以相同,也可以不同。

比如,面向字符的同步传输方式中,可以使用字符集中特定的控制字符作为帧定界符。面向位的同步传输方式中,可以使用特定的比特模式(bit-pattern)作为帧定界符,如HDLC和PPP中使用了比特模式01111110(十六进制表示为0x7E,其中0x表示其后是十六进制数)。

异步传输方式也有字符同步问题。它每一个字符前后各加一个起始位和一个停止位,以实现字符同步。通信的双方使用独立的定位时钟,但要约定同样的传输速率,因为一个字符包含的比特很少,定位偏差积累有限,也可以实现位同步。

3.2.2 透明传输

1. 伪同步问题

使用帧定界符解决帧同步问题的确是一个简单的方法,但是还存在所谓伪同步的问题。试想,如果帧携带的数据中恰好包含了一个或多个和帧定界符同样的字符或比特模式,就会出现各种伪同步的问题。比如,数据中有一个和帧结束定界符一样的字符或比特模式,那么接收方就提前结束接收,丢掉一部分数据。又如,数据中前面有一个和帧结束定界符一样的字符或比特模式,后面又有一个和帧开始定界符一样的字符或比特模式,那么接收方就当成两个帧来接收,CRC检验就不对了。

2. 透明传输的方法

数据链路层要实现透明传输,即首、尾帧定界符之间不管什么样的字符或比特模式都能正确传输,特别是用户数据和差错检验生成的帧检验序列,它们有可能包含帧定界符。

实现透明传输,可以用以下两种方法:

① 对用户数据中与帧定界符一样的字符或比特模式进行变换,使之与帧定界符不一样,然后再进行封装;接收方则进行逆变换。对字符和比特模式的变换方式分别称为字节填充(byte stuffing)和比特填充(bit stuffing)。

② 采用特殊的帧定界符,它在用户数据和帧检验序列中根本不可能出现。

(1) 字节填充

字节填充也称字符填充(character stuffing),基本方法是发送方数据链路层在数据中与帧定界符一样的字符前插入一个转义字符(escape character),如果数据中出现了转义字符,在其前面也插入一个转义字符,接收方数据链路层删除转义字符后上交网络层。

下面以PPP为例说明字节填充的方法。PPP可以在多种类型的链路上运行,其中一种

常用场合是住宅用户计算机通过 RS-232 和 Modem 拨号连接公共交换电话网进行 Internet 接入。这是一种异步传输的链路，数据块要逐个字符地传送，此时 PPP 使用字节填充。

PPP 字节填充使用的转义字符是 0x7D，发送方 PPP 在发送首、尾帧定界符之间的部分时，进行如下的处理：

① 将与帧定界符相同的 0x7E→0x7D，0x5E；

② 将与转义字符相同的 0x7D→0x7D，0x5D。

接收方 PPP 接收帧时，删除字符 0x7D，并将其后面的字符与 0x20 进行异或运算，还原成原来的字符。

另外，对于 Modem 使用的控制字符（ASCII 码中控制字符的数值小于 0x20），PPP 也做类似的转换处理。否则，Modem 可能把它当成控制字符引起误操作。

(2) 比特填充

比特填充方式的发送方，在发送首、尾帧定界符之间的比特流时，对与帧定界符相同的比特模式进行变换，插入额外的比特，从而变成与帧定界符不同的形式。

下面仍以 PPP 为例说明比特填充的方法。PPP 协议的另一种常用的场合是由路由器和点对点连接而成 Internet 的一些主干，路由器之间的链路可以是 SDH/SONET 等传输系统，一个数据块的一连串比特是连续发送的，此时 PPP 使用比特填充。

PPP 沿用了高级数据链路规程（HDLC）同样的比特模式 01111110 作为首、尾帧定界符，也沿用了 HDLC 的零比特填充方式。在发送数据过程中，当遇到连续 5 个 1 时，插入一个 0，在接收过程，遇到了 5 个连续的 1 时，去掉后边的 0，恢复为发送前的状态。因为插入的比特是 0，所以称为零比特填充。

比特填充可以由硬件实现，快速方便。

(3) 使用特殊的帧定界符

如果能够找到用户数据中根本不可能出现的编码作为特殊的帧定界符，显然就非常简单直接地实现了透明传输。以下是几个例子。

目前使用最广的 100Mb/s 的 100BaseTX 以太网采用 4B/5B-MLT3 两级编码。4B 码有 16 种组合，而 5B 码则有 32 种组合，选用其中 16 种组合作为数据码，而多余的 16 种可以选做控制码，包括特殊的帧定界符。

IEEE 802.5 令牌环帧采用差分曼彻斯特编码规则，而它的帧定界符则采用了与差分曼彻斯特编码不同的编码方式，8 个比特中有 4 个比特中间无跳变，作为特殊的帧定界符。

IEEE 802.3 以太网帧不使用帧结束定界符，当总线上传输信号（以太网中称为载波）消失，信道空闲，就判断一帧结束。此处载波消失也可视为特殊的帧定界符。

3.3　差错检验

3.3.1　差错检验方法

检验差错的常用方法是对被传送的信息进行适当的编码。给信息码加上冗余码，冗

余码长度一般是固定的且比信息码的长度短,过长会增加额外的传输负担。

冗余码通过一定的运算得出,它与信息码之间具备某种特定的关系。由信息码求冗余码的运算是通信双方的数据链路层协议中约定的。发送方将信息码和冗余码一起封装在帧里,通过信道发出。在数据链路层的帧结构中,冗余码常称为帧检验序列(Frame Check Sequence,FCS)。

接收方接收到帧后,检验它们之间的关系是否符合双方约定的关系,符合就认为没有传输差错,不符合就认为发现了传输差错。之所以这样讲,是因为一般差错检验算法都不能100%地检验出所有可能的传输差错。但是,由于差错检验算法的精心设计,其检错率是很高的,接近100%。冗余码长度常用的有8、12、16和32比特等,一般附加用于检验的冗余码的位数越多,检错能力就越强,但传输的额外开销也就越大。

计算机网络中,差错控制用得最广泛的方式还是反馈重传纠错。纠错的前提是进行差错检验。

数据链路层最常用的差错检验方法是循环冗余检验(Cyclic Redundancy Check,CRC)。发送方和接收方都可以用专用的集成电路硬件实现CRC算法,这样可以大大加快差错检验的速度。

3.3.2 循环冗余检验(CRC)

1. 码多项式

CRC检验使用码多项式的概念。码多项式的基本思想是任何一个二进制编码的位串都可以用一个多项式来表示,多项式的系数由该位串的码元表示,只有0和1,一个n位长度的位串$C=C_{n-1}C_{n-2}\cdots C_1C_0$,可以用下列$n-1$次码多项式表示:

$$C(x)=C_{n-1}x^{n-1}+C_{n-2}x^{n-2}+\cdots+C_1x+C_0 \tag{3.1}$$

例如位串1010001的码多项式为x^6+x^4+1。

数据后面附加上冗余码的操作可以用码多项式的算术运算来表示。一个k位的信息码后面附加上r位的冗余码,组成长度为$n=k+r$的码,它对应一个$(n-1)$次的码多项式$C(x)$,信息码和冗余码分别对应一个$(k-1)$次码多项式$K(x)$和一个$(r-1)$次的码多项式$R(x)$,那么有:

$$C(x)=x^rK(x)+R(x) \tag{3.2}$$

下面将会看到,CRC检验中,由信息码产生冗余码以及进行差错检验的过程也可以用码多项式的运算来实现。

2. 由信息码生成冗余码

由信息码产生冗余码的过程,即由已知的$K(x)$求$R(x)$的过程,也是用码多项式的算术运算来实现。方法是:通过用一个特定的r次多项式$G(x)$去除$x^rK(x)$$\left(\text{即}\frac{x^rK(x)}{G(x)}\right)$,其余数为$(r-1)$次的码多项式$R(x)$,对应的$r$位的位串作为冗余码。其中$G(x)$称为生成多项式(generator polynomial),是事先约定的。除法中使用模2减(无借位减,相当于作异或XOR运算)。实际上,进行码多项式的除法运算,只要用其相应的系数(等于对应的位串)进行除法运算就可以。

下面举例说明上述由信息码生成冗余码过程。

信息码：1010001，对应的码多项式为 $K(x)=x^6+x^4+1$　　$(k=7)$；

生成多项式：$G(x)=x^4+x^2+x^1+1$　　$(r=4)$，对应的位串为 10111；

$x^4K(x)$：$x^4(x^6+x^4+1)=x^{10}+x^8+x^4$，对应的位串为 10100010000；

那么，$R(x)$为$\frac{x^4K(x)}{G(x)}$的余数。使用由相应的系数构成的除式，进行运算如下所示。

```
              1001111
        ┌────────────
10111 ) 10100010000
        10111
        ─────
          11010
          10111
          ─────
            11010
            10111
            ─────
              11010
              10111
              ─────
                11010
                10111
                ─────
                 1101
```

4 位的余数 1101 作为冗余码，其码多项式为 $R(x)=x^3+x^2+1$。

3. 传输差错检验

若传输过程不出现差错，则接收端接收到的信息也应为 $C(x)$。接收方将接收到的 $C(x)$除以生成多项式 $G(x)$，只要余数不为零，则表明检验出传输差错，若余数为零，则可以认为传输无误。证明如下：

设 $x^rK(x)$除以 $G(x)$的商为 $Q(x)$，则

$$x^rK(x)=G(x)Q(x)+R(x) \tag{3.3}$$

将式(3.3)代入式(3.2)，得到：

$$\begin{aligned} C(x) &= x^rK(x)+R(x)=G(x)Q(x)+R(x)+R(x) \\ &= G(x)Q(x) \end{aligned} \tag{3.4}$$

在式(3.4)的推导中，因为＋为模 2 加(不进位加，相当于异或)，故 $R(x)+R(x)=0$。可见，如果传输无差错，接收到的仍为 $C(x)$，则用 $C(x)$除以 $G(x)$的余数必为零(可用上述例子进行验算)，也就是说，只要余数不为零，则表明传输出现差错。但反过来，并不等于余数为零就一定传输无差错，在某些非常特殊的比特差错组合下，CRC 也可能碰巧使余数为零。

4. 常用的生成多项式

广泛采用的生成多项式有：

CRC-8＝x^8+x^2+x+1

CRC-16＝$x^{16}+x^{15}+x^2+1$

CRC-CCITT＝$x^{16}+x^{12}+x^5+1$

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

CRC-8 用于 ATM 信元头差错检验，CRC-16 是 HDLC 规程中使用的 CRC 检验生成多项式，而 CRC-32 是 IEEE 802.3 以太网媒体接入控制帧中采用的 CRC 检验生成多项

式。这些生成多项式都是经过数学上的精心设计和实际验证的。

3.4 数据链路控制

3.4.1 数据链路控制的基本思想

一条可靠的数据链路应该满足以下两个条件:

① 传输的任何数据,既不会出现差错也不会丢失;

② 不管发送方以多快的速率发送数据,接收方总能够来得及接收、处理并上交主机。也就是接收方有足够的接收缓存和处理速度。

但实际应用的数据链路并不能满足上述条件。第①个条件不满足,就必须进行差错控制(error control),第②个条件不满足就必须进行流量控制(flow control)。

差错控制使得链路传输出现差错时得到补救。主要有两种差错发生,一是帧丢失,即一个数据帧未能到达接收端;二是帧损坏,例如其中有几位数据出错。差错控制的基本方式是反馈重传机制。

流量控制用来保证发送数据在任何情况下都不会"淹没"接收方的接收缓存(接收缓存溢出),从而不会丢失数据,而且还应使传输达到理想的吞吐率。由接收方根据其接收缓存的状况来控制发送方的数据流量是流量控制的基本思想。实现流量控制的一个重要方法是滑动窗口(sliding window)机制。

3.4.2 数据链路控制的基本机制

1. 反馈重传机制

差错控制的常用方法是反馈重传机制,也称确认-重传机制,也是数据链路控制的一个基本机制。接收方对接收到的数据进行差错检验后,以某种方式向发送方反馈差错状况,称为确认(acknowledgement),发送方根据确认信息对出现传输差错的帧进行重传。

反馈重传机制包括下面两步。

(1) 接收方反馈确认信息

确认方式包括多种:

① 正确认或肯定确认(positive acknowledgement)。接收方收到一个经检验无错的帧后,返回一个正确认。正确认简称确认,记为 ACK(ACKnowledgement)。

② 累计确认(cumulative acknowledgement)。接收方可以收到多个连续且正确的数据帧以后,才只对最后一个帧发回一个 ACK,称为累计确认。累计确认表明该帧及其以前所有的帧均已正确地收到。

③ 捎带确认(piggybacking)。在双向数据传输情况下,将确认信息放在自己的数据帧的首部字段中捎带过去。累计确认和捎带确认都可提高传输效率。

④ 负确认(Negative ACKnowledgement,NAK)。接收方收到一个有差错的帧后,返回一个对此帧的 NAK。

(2) 发送方重传差错帧

常用的重传方式包括以下两种：

① 超时重传(timeout retransmission)。发送方在发送完一帧时即启动一个重传定时器，若由它设定的重传时间到且未收到反馈的确认信息，则重传此帧。这是经常采用的重传方式。为了重传，发送方必须保存一个已发出的数据帧的副本。

② 负确认重传。发送方收到接收方对一个帧的 NAK，重传此帧。

反馈重传机制对出差错数据帧的重传是自动进行的，因此这种控制机制称为自动请求重传 ARQ。根据反馈重传方式的不同，可以分为停等 ARQ(stop-and-wait ARQ)、回退-N ARQ(go-back-N ARQ)和选择重传 ARQ(selective repeat ARQ)。

实际上，ARQ 既使用了反馈重传机制对传输过程进行差错控制，也同时使用了滑动窗口机制进行流量控制，从而保证数据链路层实现可靠的数据传输。

ARQ 是第二次世界大战期间发明的，目的是使无线电通信提供可靠的字符传送，它非常简单，但对信道的噪声有很强的适应性。

2. 滑动窗口机制

滑动窗口是数据链路控制的一个基本机制，发送方和接收方分别设置发送窗口和接收窗口，数据传输过程中在接收方的控制下向前滑动，从而对数据传输流量进行控制。

发送窗口用来对发送方进行流量控制，落在窗口内的帧是可以连续发送的，其大小 W_T 指明在收到对方 ACK 之前发送方最多可以发送多少个帧。

接收窗口控制哪些帧可以接收，只有到达帧的序号落在接收窗口之内时才可以被接收，否则将被丢弃。一般，当接收方收到一个有序且无差错的帧后，接收窗口向前滑动，准备接收下一帧，并向发送方发出一个 ACK。

当发送方收到接收方的 ACK 后，发送窗口才能向前滑动，滑动的长度取决于接收方确认的序号。向前滑动后，又有新的待发帧落入发送窗口，可以被发送。

可见，接收方的 ACK 作为授权发送方发送数据的凭证，接收方可以根据自己的接收能力来控制确认的发送时机，从而实现对传输流量的控制。

下面以图 3.2 为例说明滑动窗口机制。假设发送序号用 3 比特来编码，即发送序号可以有从 0～7 的 8 个不同的序号。又设发送窗口 $W_T=5$，发送方滑动窗口工作过程如图 3.2 所示。

① 初始状态如图 3.2(a)所示。发送窗口内共有从 0～4 的 5 个序号，这些帧现在可以连续发送，而 5 号及以后的帧是不能发送的。若发送方发送完了窗口内全部 5 个帧，仍未收到接收方的 ACK，就必须停止发送。

② 收到了接收方对 0 号帧的确认 ACK1，发送窗口向前滑动 1 个序号，如图 3.2(b)所示。现在 5 号帧已进入到发送窗口之内，可以发送。

③ 在这以后，假设又收到对 3 号帧的累计确认 ACK4，说明接收方又正确地收到了 1～3 号帧，于是发送窗口又可再向前移动 3 个序号，那么 6 号、7 号和 0 号帧又进入发送窗口，它们也可以发送，如图 3.2(c)所示。

可见，图 3.2 中发送窗口左边的数据帧是已经发送并得到确认的帧；窗口内是可以发送的帧，包括已经发送但未得到确认的帧和尚未发送的帧；窗口的右边是不可以发送的数

图 3.2 发送方的滑动窗口

据帧。

滑动窗口机制中,为控制传输流量可以设置合适大小的 W_T,一般不超过接收方接收缓存的大小,这样发送的数据就不容易淹没接收缓存。还可以使用可变滑动窗口,由接收方根据目前可用接收缓存的大小动态改变 W_T,在 TCP 流量控制中就使用这种方式。

3.4.3 自动请求重传(ARQ)

1. 停等 ARQ

(1) 停等 ARQ 工作机制

停等 ARQ 的基本思想是在发送方发出一个数据帧后停下来不再发送,等待接收方的 ACK 到达,ACK 到达后才发送下一帧。

停等 ARQ 实际上也使用了滑动窗口技术,它的发送窗口大小是 $W_T=1$,接收窗口大小也是 1,因此,在发送出去一个数据帧后,停止发送,等待接收方的 ACK。

停等 ARQ 要处理传输中可能出现的以下 3 种传输差错:

① 接收方收到了发来的数据帧,但检测出收到的帧有差错。

② 发送方发出的数据帧丢失,接收方收不到,发送方不可能收到 ACK。

③ 接收方收到正确的数据帧,但发出的 ACK 丢失,发送方也不可能收到 ACK。

对于差错①,接收方丢弃此帧,并可以考虑采取下面两种方式进行重传:

(a) 负确认重传。但如果 NAK 丢失,发送方将收不到 NAK,又有新的问题。

(b) 超时重传。

对于②和③这两种差错,即使发送方一直等下去也不会等到接收方的 ACK,这样就会出现死锁。要解决死锁,可采用上述的方法(b)。重传定时时间 T_{OUT} 应大于一个帧的正常往返传输时间,主要包括数据帧的发送时间 T_{DATA}、确认帧的发送时间 T_{ACK}、链路的往返传播时延 2τ 以及必要的处理时间 T_{PRO}(差错检验等)。

但对于差错③,超时重传会使接收方收到两个同样的数据帧,且接收方无法识别后者

是一个数据相同的新帧还是重传的旧帧。解决重复帧的方法是为数据帧和确认帧编上序号。对于停等 ARQ，用 0 和 1 交替地编号就可以区分上述两种情况，以辨别出重复的数据帧而丢弃之。接收方正确地收到 0/1 号数据帧，发回确认 ACK1/ACK0，确认序号表明期望接收的下一个序号。

停等 ARQ 采用超时重传的方式，并对 ACK 编号。对于传输差错①，接收方将不发送 ACK。这样，以上 3 种传输差错问题都可以解决。

(2) 停等 ARQ 示例

图 3.3 是一个停等 ARQ 的例子。例子中包括了正常发送和确认、发送数据帧丢失和 ACK 丢失等情况。图中的水平方向表示了发送站 A 和接收站 B 之间的距离，垂直方向为时间，向下是时间增长方向。表示数据帧和确认帧传播的带箭头水平线都向下倾斜，反映了传播时延 τ。实际上，发送的数据帧和确认帧在垂直方向上都应该有一定的宽度，在图右侧的局部放大图中可清楚地看到，其大小反映了它们的发送时间 T_{DATA} 和 T_{ACK}，发送时间与帧长度成正比，与发送的比特率成反比。

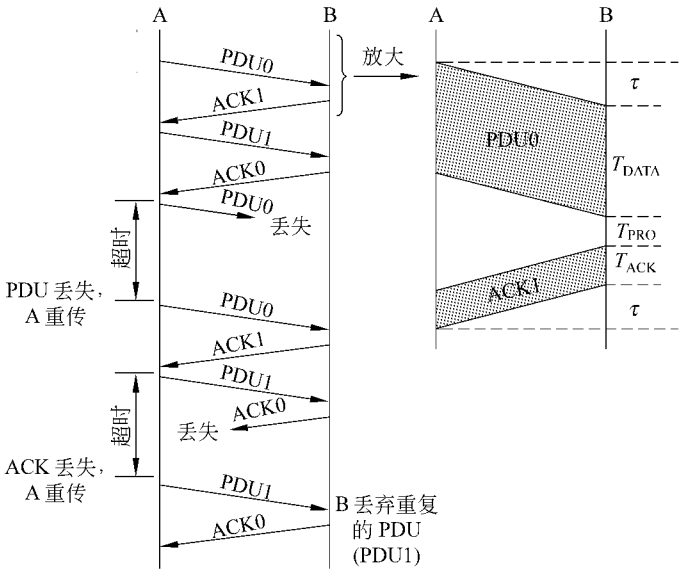


图 3.3　停等 ARQ 传输过程示例

(3) 停等 ARQ 的链路利用率

停等 ARQ 机制的问题是可能产生严重的低效率。下面对停等 ARQ 性能进行粗略的定量分析。正常传输的情况下，停等 ARQ 把传输流量控制在每个往返时间中一个数据帧的水平，一个往返时间为 $T_{DATA}+T_{ACK}+2\tau+T_{PRO}$。因此，链路利用率 η 可以表示为：

$$\begin{aligned}\eta &= T_{DATA}/(T_{DATA}+T_{ACK}+2\tau+T_{PRO})\\ &\approx T_{DATA}/(T_{DATA}+2\tau)\\ &= \text{帧的比特长度}/(\text{帧的比特长度}+2\text{ 链路的比特长度})\end{aligned}\tag{3.5}$$

式(3.5)中的第 2 式是一个近似，因为一般情况下 T_{ACK} 和 T_{PRO} 较小。将第 2 式的分子分母同乘带宽，则变为第 3 式。由式(3.5)可见，如果帧的长度很大而传播延时又小，停等