《计算机网络原理》

课程编号：40240513

讲课教师：吴建平 徐明伟 尹霞

本科生必修课

计算机科学与技术系

第五章 数据链路层

第一部分

主要教学内容和学时分配

第一章	引言	3
第二章	计算机网络的体系结构	6
第三章	数据通信的基本原理	3
第四章	物理层	3
第五章	数据链路层	9
第六章	局域网技术	6
第七章	网络互联和访问控制	6
第八章	运输层的可靠传输	3
第九章	计算机网络应用	6
第十章	计算机网络新技术/复习	3
共计		48

第五章　数据链路层

5.1　定义和功能
- 5.1.1　定义
- 5.1.2　为网络层提供服务
- 5.1.3　成帧
- 5.1.4　差错控制
- 5.1.5　流量控制

5.2　错误检测和纠正
- 5.2.1　纠错码
- 5.2.2　检错码

5.3　基本的数据链路层协议
- 5.3.1　无约束单工协议
- 5.3.2　单工停等协议
- 5.3.3　有噪声信道的单工协议

第五章　数据链路层

5.4　滑动窗口协议

5.4.1　一比特滑动窗口协议

5.4.2　退后n帧协议

5.4.3　选择重传协议

5.5　协议说明与验证

5.5.1　通信协议中的形式化描述技术

5.5.2　有限状态机模型

5.5.3　Petri网模型

5.6　常用的数据链路层协议

5.6.1　高级数据链路控制规程 HDLC

5.6.2　X.25 LAPB

5.6.3　Internet数据链路层协议

5.1 定义和功能（1）

5.1.1 定义

- 要解决的问题：如何在有差错的线路上，实现无差错传输。
- ISO关于数据链路层的定义：

  数据链路层的目的是为了提供功能上和规程上的方法，以便建立、维护和释放网络实体间的数据链路。
- 数据链路：从数据发送点到数据接收点所经过的传输途径。虚拟数据通路，实际数据通路。

  Fig. 3-1

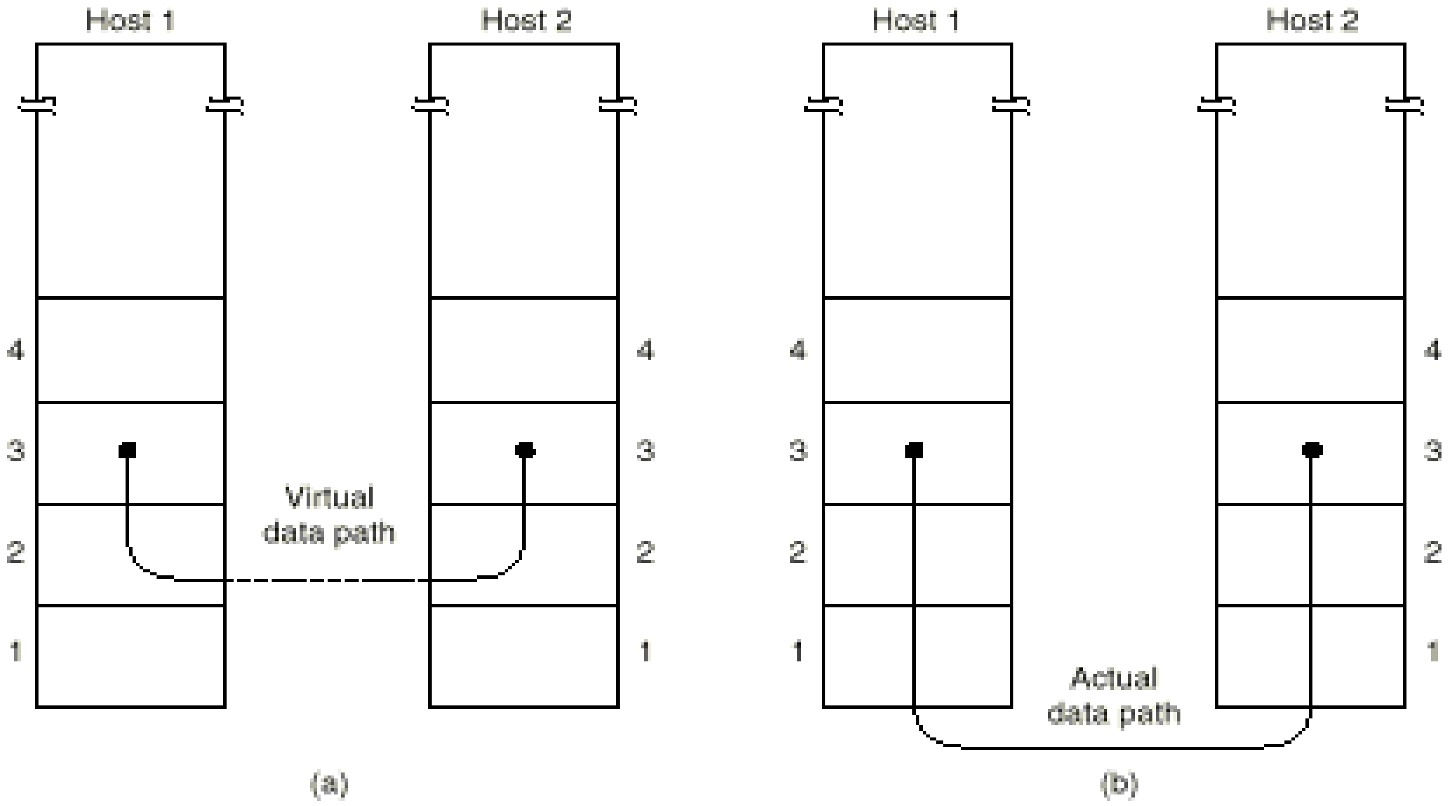


Fig. 3-1. (a) Virtual communication. (b) Actual communication.

5.1 定义和功能（2）

- 数据链路控制规程：为使数据能迅速、正确、有效地从发送点到达接收点所采用的控制方式。
- 数据链路层协议应提供的最基本功能
  - 数据在数据链路上的正常传输（建立、维护和释放）
  - 定界与同步，也处理透明性问题
  - 差错控制
  - 顺序控制
  - 流量控制

5.1 定义和功能（3）

5.1.2 为网络层提供服务

为网络层提供三种合理的服务

- 无确认无连接服务

  适用于

  - 误码率很低的线路，错误恢复留给高层；
  - 实时业务
  - 大部分局域网

- 有确认无连接服务

  适用于不可靠的信道，如无线网。

- 有确认有连接服务

5.1 定义和功能（4）

5.1.3 成帧（Framing）

将比特流分成离散的帧，并计算每个帧的校验和。

成帧方法：

- 字符计数法
  - 在帧头中用一个域来表示整个帧的字符个数
  - 缺点：若计数出错，对本帧和后面的帧有影响。
    Fig. 3-3
- 带字符填充的首尾字符定界法
  - 起始字符 DLE STX，结束字符DLE ETX
  - 字符填充
    Fig. 3-4
  - 缺点：局限于8位字符和ASCII字符传送。

5.1 定义和功能（5）

– 带位填充的首尾标记定界法
  - 帧的起始和结束都用一个特殊的位串“01111110”，称为**标记**
  - “0”比特插入删除技术

    Fig. 3-5
– 物理层编码违例法
  - 只适用于物理层编码有冗余的网络
– 注意：在很多数据链路协议中，使用字符计数法和一种其它方法的组合。

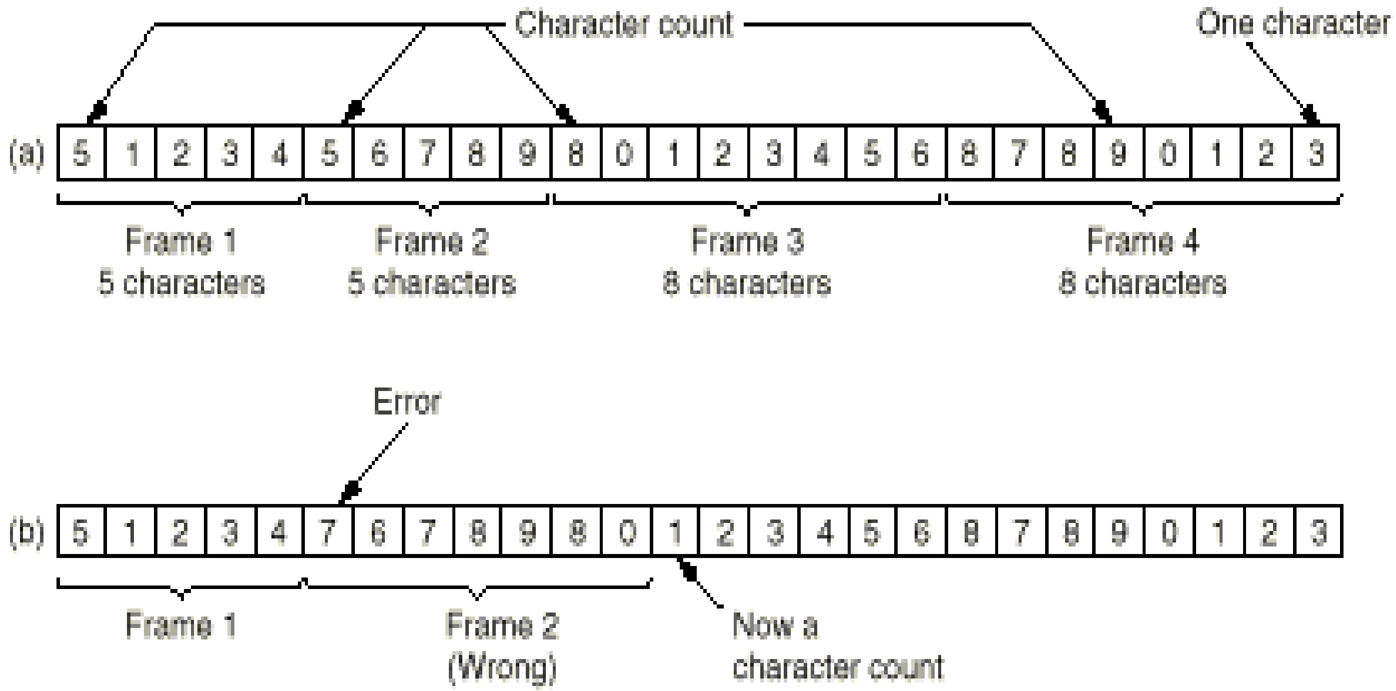


Fig. 3-3. A character stream. (a) Without errors. (b) With one error.

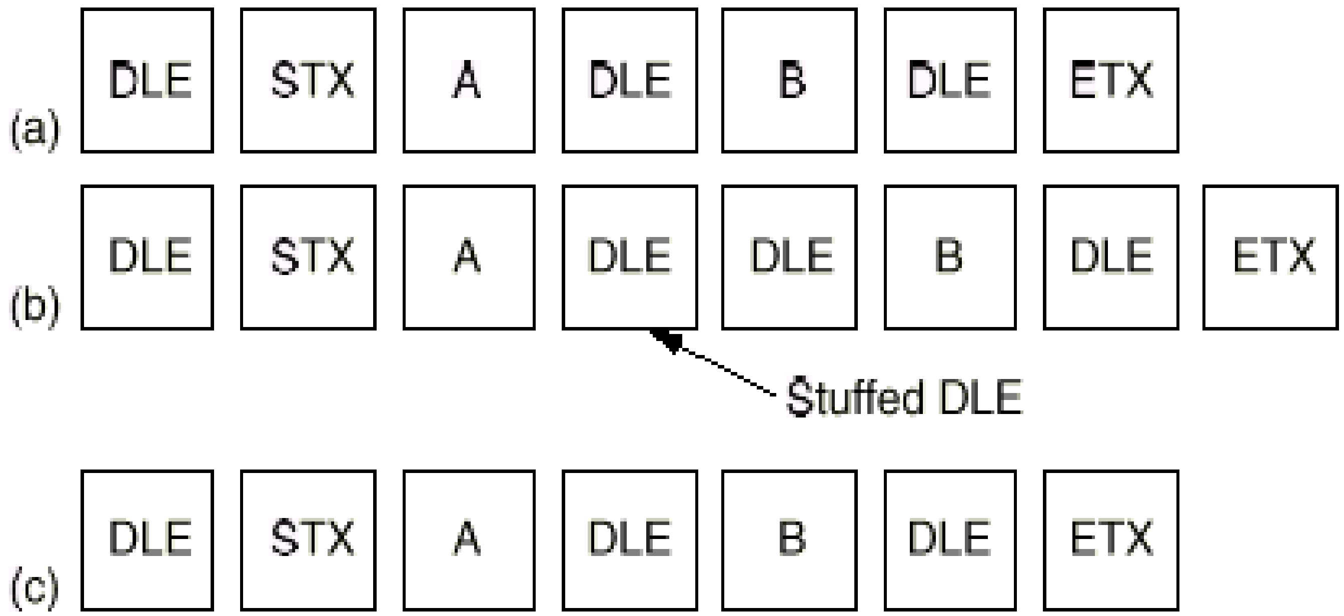


Fig. 3-4. (a) Data sent by the network layer. (b) Data after being character stuffed by the data link layer. (c) Data passed to the network layer on the receiving side.

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Fig. 3-5. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

5.1 定义和功能（6）

5.1.4 差错控制

- 一般方法：接收方给发送方一个反馈（响应）。
- 出错情况
  - 帧（包括发送帧和响应帧）出错；
  - 帧（包括发送帧和响应帧）丢失
- 通过计时器和序号保证每帧最终交给目的网络层仅一次是数据链路层的一个主要功能。

5.1.5 流量控制

基于反馈机制

流量控制主要在传输层实现。

5.2 错误检测和纠正（1）

- 差错出现的特点：随机，连续突发（burst）
- 处理差错的两种基本策略
  - 使用纠错码：发送方在每个数据块中加入足够的冗余信息，使得接收方能够判断接收到的数据是否有错，并能纠正错误。
  - 使用检错码：发送方在每个数据块中加入足够的冗余信息，使得接收方能够判断接收到的数据是否有错，但不能判断哪里有错。

5.2.1 纠错码

- **码字**（codeword）：一个帧包括m位数据，r个校验位，$n = m + r$，则此n比特单元称为n位码字。
- **海明距离**（Hamming distance）：两个码字的不同比特位数目。

5.2 错误检测和纠正（2）

例:　　　　0000000000 与
0000011111
的海明距离为5

- 如果两个码字的海明距离为d，则需要d个单比特错就可以把一个码字转换成另一个码字；
- 为了检查出d个错（单比特错），需要使用海明距离为 d + 1 的编码；
- 为了纠正d个错，需要使用海明距离为 2d + 1 的编码；
- 最简单的例子是奇偶校验，在数据后填加一个奇偶位（parity bit）。

例：使用偶校验（“1”的个数为偶数）

10110101 ——> 101101011

10110001 ——> 101100010

奇偶校验可以用来检查单个错误。

5.2 错误检测和纠正（3）

- 设计纠错码
  - 要求：m个信息位，r个校验位，纠正单比特错；
  - 对2^m个有效信息中任何一个，有n个与其距离为1的无效码字，因此有：

$$(n + 1)\ 2^m \le 2^n$$

利用 $n = m + r$，得到 $(m + r + 1) \le 2^r$

给定 m，利用该式可以得出校正单比特误码的校验位数目的下界。

- 海明码
  - 码位从左边开始编号；
  - 位号为2的幂的位是校验位，其余是信息位；
  - 每个校验位强迫包括自己在内的一些位的奇偶值为偶数（或奇数）。
  - 为看清数据位k对哪些校验位有影响，将k写成2的幂的和。

    例：$11 = 1 + 2 + 8$

5.2 错误检测和纠正（4）

- 海明码工作过程
  - 每个码字到来前，接收方计数器清零；
  - 接收方检查每个校验位k (k = 1, 2, 4 ...)的奇偶值是否正确；
  - 若第 k 位奇偶值不对，计数器加 k；
  - 所有校验位检查完后，若计数器值为0，则码字有效；若计数器值为m，则第m位出错。

    若校验位1、2、8出错，则第11位变反。

    Fig. 3-6
- 使用海明码纠正突发错误
  - 可采用k个码字（n = m + r）组成 k × n 矩阵，按列发送，接收方恢复成 k × n 矩阵
  - kr个校验位，km个数据位，可纠正最多为k个的突发性连续比特错。

Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	11111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	00101011111
d	1100100	11111001100
e	1100101	00111000101

Order of bit transmission

Fig. 3-6. Use of a Hamming code to correct burst errors.

5.2 错误检测和纠正（5）

5.2.2 检错码

- 使用纠错码传数据，效率低，适用于不可能重传的场合；大多数情况采用检错码加重传。
- 循环冗余码（CRC码，多项式编码）

  110001，表示成多项式 $x^5 + x^4 + 1$
- 生成多项式G(x)
  - 发方、收方事前商定；
  - 生成多项式的高位和低位必须为1
  - 生成多项式必须比传输信息对应的多项式短。
- CRC码基本思想：校验和加在帧尾，使带校验和（checksum）的帧的多项式能被G(x)除尽；收方接收时，用G(x)去除它，若有余数，则传输出错。

5.2 错误检测和纠正（6）

- 校验和计算算法
  - 设G(x)为 r 阶，在帧的末尾加 r 个0，使帧为m + r位，相应多项式为$x^rM(x)$；
  - 按模2除法用对应于G(x)的位串去除对应于$x^rM(x)$的位串；
  - 按模2减法从对应于$x^rM(x)$的位串中减去余数（等于或小于r位），结果就是要传送的带校验和的多项式T(x)。

    Fig. 3-9 循环冗余检验码计算实例。
- CRC的检错能力

  发送：T(x)

  接收：T(x) + E(x)

  (T(x) + E(x)) / G(x) = 0 + E(x) / G(x)

  若 E(X) / G(x) = 0，则差错不能发现；否则，可以发现。

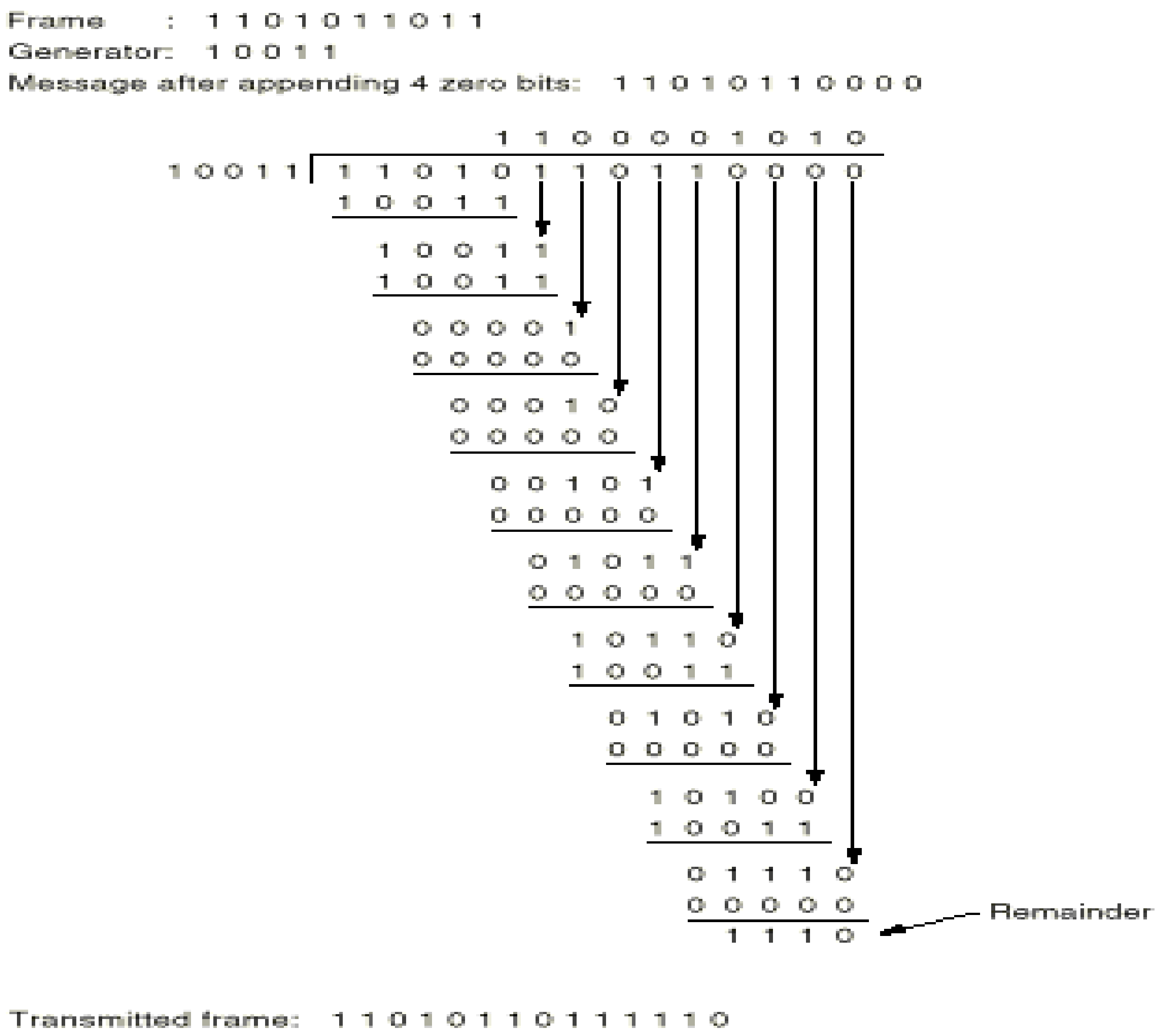


图 3-9 循环冗余检验码计算实例

5.2 错误检测和纠正（7）

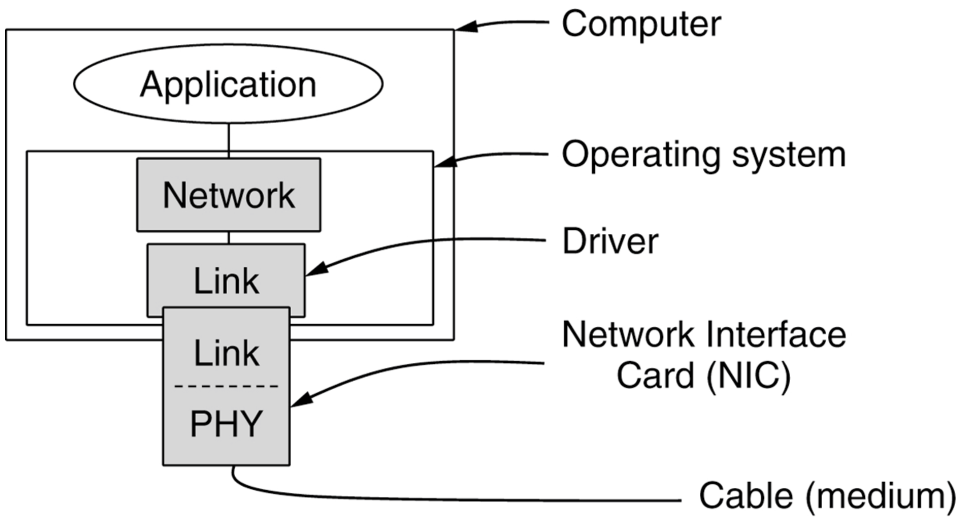
- 如果只有单比特错，即E(x) = x^i，而G(x)中有两项，E(x) / G(x) ≠ 0，所以可以查出单比特错；
- 如果发生两个孤立单比特错，即E(x) = $x^i + x^j = x^j (x^{i-j} + 1)$，假定G(x)不能被x整除，那么能够发现两个比特错的充分条件是：$x^k + 1$不能被G(x)整除 ($k \le i - j$)；
- 如果有奇数个比特错，即E(x)包括奇数个项，G(x)选(x + 1)的倍数就能查出奇数个比特错；
- 具有r个校验位的多项式能检查出所有长度 ≤ r 的差错。长度为k的突发性连续差错（并不表示有k个单比特错）可表示为 $x^i (x^{k-1} + \dots + 1)$，若G(x)包括$x^0$项，且 k - 1小于G(x)的阶，则 E(x) / G(x) ≠ 0;
- 如果突发差错长度为 r + 1，当且仅当突发差错和G(x)一样时， E(x) / G(x) = 0，概率为$1/2^{r-1}$;
- 长度大于 r + 1的突发差错或几个较短的突发差错发生后，坏帧被接收的概率为 $1/2^r$。

5.2 错误检测和纠正（8）

- 三个循环冗余检验码生成多项式已成为国际标准
  - CRC-12 $= x^{12} + x^{11} + x^3 + x^2 + x + 1$
  - CRC-16 $= x^{16} + x^{15} + x^2 + 1$
  - CRC-CCITT $= x^{16} + x^{12} + x^5 + 1$
- 硬件实现CRC校验。

5.3 基本的数据链路层协议（1）

- 三个最简单的链路层协议
  - 协议1：无流量控制和错误控制。也叫乌托邦（Utopia）或无约束协议
  - 协议2：增加流量控制：停-等协议
  - 协议3：增加错误控制：顺序号和ARQ
- 最基本的简单假设
  - 独立的进程
    - 物理层、数据链路层、网络层
  - 单项通信
  - 可靠的计算机和进程



数据链路层协议算法的通用定义

```
#define MAX_PKT 1024                                   /* determines packet size in bytes */
typedef enum {false, true} boolean;                    /* boolean type */
typedef unsigned int seq_nr;                           /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;  /* packet definition */
typedef enum {data, ack, nak} frame_kind;              /* frame_kind definition */

typedef struct {                                       /* frames are transported in this layer */
  frame_kind kind;                                     /* what kind of frame is it? */
  seq_nr seq;                                          /* sequence number */
  seq_nr ack;                                          /* acknowledgement number */
  packet info;                                         /* the network layer packet */
} frame;
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Figure 3-11. Some definitions needed in the protocols to follow. These definitions are located in the file *protocol.h*.

5.3 基本的数据链路层协议（2）

- 5.3.1 无约束单工协议（An Unrestricted Simplex Protocol）
  - 工作在理想情况，几个前提：
    - 单工传输
    - 发送方无休止工作（要发送的信息无限多）
    - 接收方无休止工作（缓冲区无限大）
    - 通信线路（信道）不损坏或丢失信息帧
  - 工作过程
    - 发送程序:取数据，构成帧，发送帧；
    - 接收程序：等待，接收帧，送数据给高层
    - 图3-12

协议1：无约束协议

```
/* Protocol 1 (Utopia) provides for data transmission in one direction only, from
   sender to receiver. The communication channel is assumed to be error free
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender1(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */
  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;                  /* copy it into s for transmission */
      to_physical_layer(&s);            /* send it on its way */
  }                                     /* Tomorrow, and tomorrow, and tomorrow,
                                           Creeps in this petty pace from day to day
                                           To the last syllable of recorded time.
                                              – Macbeth, V, v */
}
void receiver1(void)
{
  frame r;
  event_type event;                     /* filled in by wait, but not used here */
  while (true) {
      wait_for_event(&event);           /* only possibility is frame_arrival */
      from_physical_layer(&r);          /* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
  }
}
```

Figure 3-12. A utopian simplex protocol.

5.3 基本的数据链路层协议（3）

- 5.3.2 单工停等协议（A Simplex Stop-and-Wait Protocol）
  - 增加约束条件：接收方不能无休止接收。
  - 解决办法：接收方每收到一个帧后，给发送方回送一个响应。
  - 工作过程
    - 发送程序：取数据，成帧，发送帧，等待响应帧；
    - 接收程序：等待，接收帧，送数据给高层，回送响应帧。
    - 图 3-13

协议2：单工停等协议

```
/* Protocol 2 (Stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender2(void)
{
  frame s;                               /* buffer for an outbound frame */
  packet buffer;                         /* buffer for an outbound packet */
  event_type event;                      /* frame_arrival is the only possibility */
  while (true) {
      from_network_layer(&buffer);       /* go get something to send */
      s.info = buffer;                   /* copy it into s for transmission */
      to_physical_layer(&s);             /* bye-bye little frame */
      wait_for_event(&event);            /* do not proceed until given the go ahead */
  }
}
void receiver2(void)
{
  frame r, s;                            /* buffers for frames */
  event_type event;                      /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);            /* only possibility is frame_arrival */
      from_physical_layer(&r);           /* go get the inbound frame */
      to_network_layer(&r.info);         /* pass the data to the network layer */
      to_physical_layer(&s);             /* send a dummy frame to awaken sender */
  }
}
```

Figure 3-13. A simplex stop-and-wait protocol.

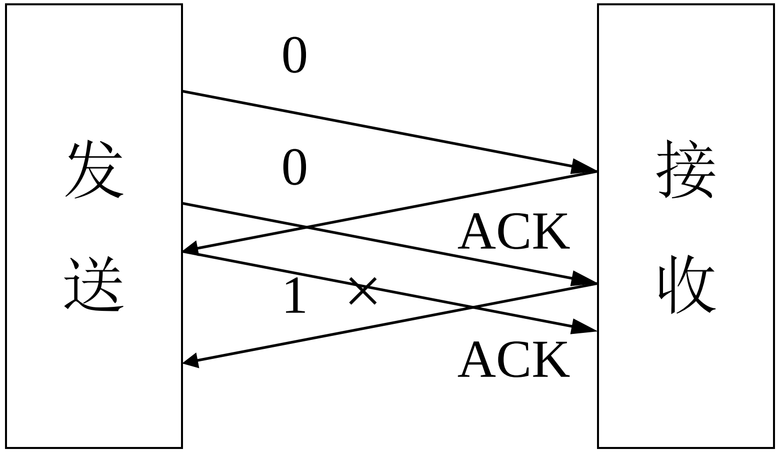
5.3 基本的数据链路层协议（4）

5.3.3 有噪声信道的单工协议（A Simplex Protocol for a Noisy Channel）

- 增加约束条件：信道（线路）有差错，信息帧可能损坏或丢失。
- 解决办法：出错重传。
- 带来的问题：
  - 什么时候重传 —— 定时
  - 响应帧损坏怎么办（重复帧）—— 发送帧头中放入序号
  - 为了使帧头精简，序号取多少位 —— 1位
- 发方在发下一个帧之前等待一个肯定确认的协议叫做PAR（Positive Acknowledgement with Retransmission）或ARQ（Automatic Repeat reQuest）

5.3 基本的数据链路层协议（5）

– 工作过程
– 图 3-14
  注意协议3的漏洞
  由于确认帧中没有序号，超时时间不能太短，否则协议失败。因此假设协议3的发送和接收严格交替进行。



协议3：有噪声信道的单工停等协议

```
/* Protocol 3 (PAR) allows unidirectional data flow over an unreliable channel. */

#define MAX_SEQ 1                          /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;               /* seq number of next outgoing frame */
  frame s;                                 /* scratch variable */
  packet buffer;                           /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;                  /* initialize outbound sequence numbers */
  from_network_layer(&buffer);             /* fetch first packet */
  while (true) {
      s.info = buffer;                     /* construct a frame for transmission */
      s.seq = next_frame_to_send;          /* insert sequence number in frame */
      to_physical_layer(&s);               /* send it on its way */
      start_timer(s.seq);                  /* if answer takes too long, time out */
      wait_for_event(&event);              /* frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) {
          from_physical_layer(&s);         /* get the acknowledgement */
          if (s.ack == next_frame_to_send) {
              stop_timer(s.ack);           /* turn the timer off */
              from_network_layer(&buffer); /* get the next one to send */
              inc(next_frame_to_send);     /* invert next_frame_to_send */
          }
      }
  }
}
```

```
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
      wait_for_event(&event);              /* possibilities: frame_arrival, cksum_err */
      if (event == frame_arrival) {        /* a valid frame has arrived */
          from_physical_layer(&r);         /* go get the newly arrived frame */
          if (r.seq == frame_expected) {   /* this is what we have been waiting for */
              to_network_layer(&r.info);   /* pass the data to the network layer */
              inc(frame_expected);         /* next time expect the other sequence nr */
          }
          s.ack = 1 – frame_expected;      /* tell which frame is being acked */
          to_physical_layer(&s);           /* send acknowledgement */
      }
  }
}
```

Figure 3-14. A positive acknowledgement with retransmission protocol.

5.4 滑动窗口协议（1）

- 滑动窗口协议说明
  - 单工 ——> 全双工
  - 捎带（piggybacking）：暂时延迟待发确认，以便附加在下一个待发数据帧的技术。
    - 优点：充分利用信道带宽，减少帧的数目意味着减少“帧到达”中断；
    - 带来的问题：复杂。
  - 本节的三个协议统称滑动窗口协议，都能在实际（非理想）环境下正常工作，区别仅在于效率、复杂性和对缓冲区的要求。
  - Fig-15 为一个3位顺序号和接收窗口为1的滑动窗口协议

5.4 滑动窗口协议（2）

- 滑动窗口协议（Sliding Window Protocol）工作原理：
  - 发送的信息帧都有一个序号，从0到某个最大值，$0 \sim 2^n - 1$，一般用n个二进制位表示；
  - 发送端始终保持一个已发送但尚未确认的帧的序号表，称为发送窗口。发送窗口的上界表示要发送的下一个帧的序号，下界表示未得到确认的帧的最小编号。发送窗口 = 上界 - 下界，大小可变；
  - 发送端每发送一个帧，序号取上界值，上界加1；每接收到一个正确响应帧，下界加1；
  - 接收端始终保持一个允许接收的帧的顺序号表，称为接收窗口，大小固定，但不一定与发送窗口相同。接收窗口的上界表示允许接收的序号最大的帧，下界表示希望接收的帧；
  - 接收窗口表示允许接收的信息帧，落在窗口外的帧均被丢弃。序号等于下界的帧被正确接收，并产生一个响应帧，上界和下界同时加1。接收窗口大小不变。

滑动窗口协议：3位顺序号和接收窗口为1

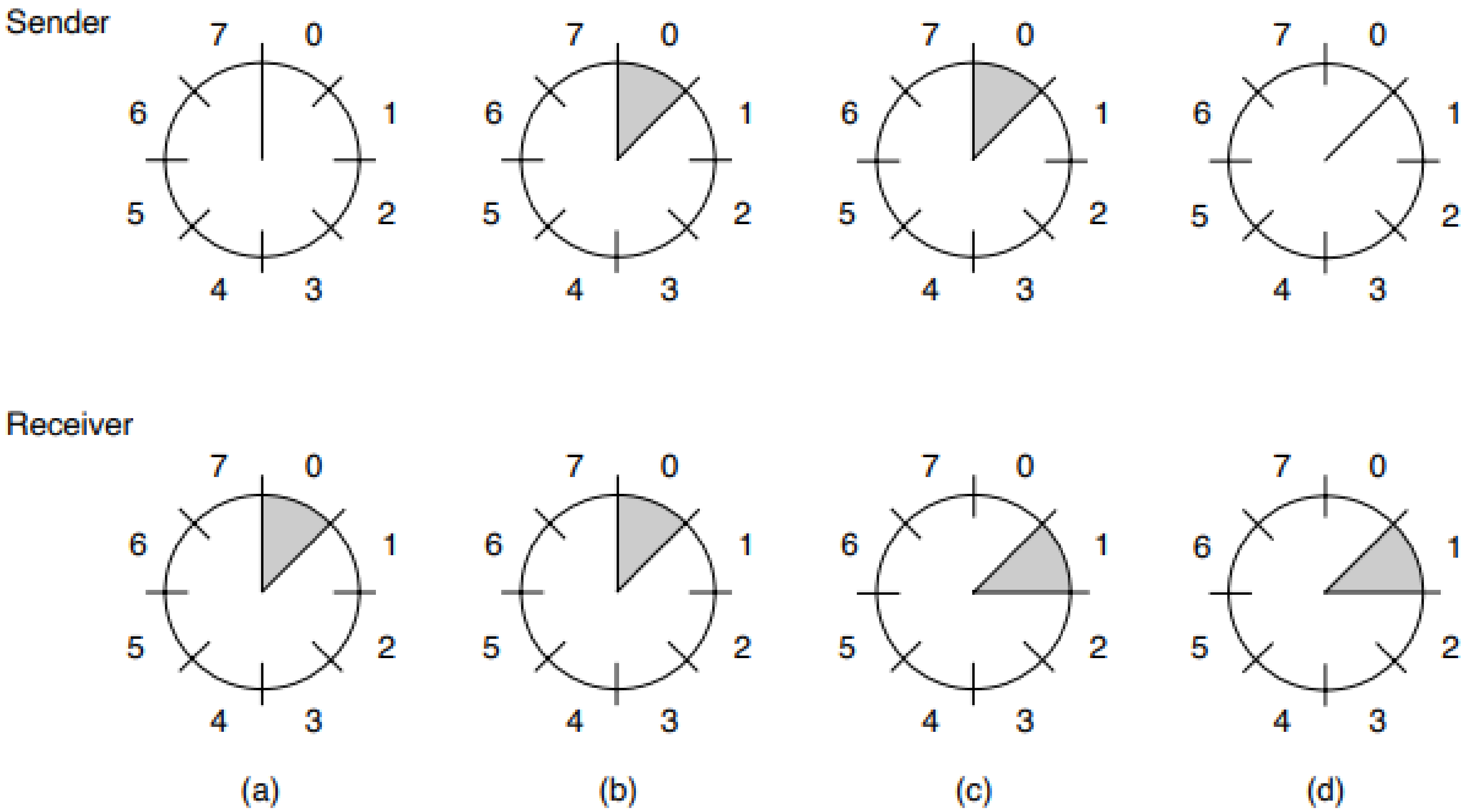


Figure 3-15. A sliding window of size 1, with a 3-bit sequence number. (a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgement has been received.

5.4 滑动窗口协议（3）

5.4.1 一比特滑动窗口协议（协议4）（A One Bit Sliding Window Protocol)

- 协议特点
  - 窗口大小：N = 1，发送序号和接收序号的取值范围：0，1；
  - 可进行数据双向传输，信息帧中可含有确认信息（piggybacking技术）；
  - 信息帧中包括两个序号域：发送序号和接收序号（已经正确收到的帧的序号)
- 工作过程: Fig. 3-16
- 存在问题
  - 能保证无差错传输，但是基于停等方式；
  - 若双方同时开始发送，则会有一半重复帧；
  - 效率低，传输时间长。
  - Fig-17 给出协议4的正常情况（a）和异常情况（b）

协议4：一比特双工滑动窗口协议

```
/* Protocol 4 (Sliding window) is bidirectional. */

#define MAX_SEQ 1                              /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                   /* 0 or 1 only */
  seq_nr frame_expected;                       /* 0 or 1 only */
  frame r, s;                                  /* scratch variables */
  packet buffer;                               /* current packet being sent */
  event_type event;
  next_frame_to_send = 0;                      /* next frame on the outbound stream */
  frame_expected = 0;                          /* frame expected next */
  from_network_layer(&buffer);                 /* fetch a packet from the network layer */
  s.info = buffer;                             /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                  /* insert sequence number into frame */
  s.ack = 1 – frame_expected;                  /* piggybacked ack */
  to_physical_layer(&s);                       /* transmit the frame */
  start_timer(s.seq);                          /* start the timer running */
```

```
  while (true) {
     wait_for_event(&event);                   /* frame_arrival, cksum_err, or timeout */
     if (event == frame_arrival) {             /* a frame has arrived undamaged */
          from_physical_layer(&r);             /* go get it */
          if (r.seq == frame_expected) {       /* handle inbound frame stream */
               to_network_layer(&r.info);      /* pass packet to network layer */
               inc(frame_expected);            /* invert seq number expected next */
          }
          if (r.ack == next_frame_to_send) {   /* handle outbound frame stream */
               stop_timer(r.ack);              /* turn the timer off */
               from_network_layer(&buffer);    /* fetch new pkt from network layer */
               inc(next_frame_to_send);        /* invert sender's sequence number */
          }
     }
     s.info = buffer;                          /* construct outbound frame */
     s.seq = next_frame_to_send;               /* insert sequence number into it */
     s.ack = 1 – frame_expected;               /* seq number of last received frame */
     to_physical_layer(&s);                    /* transmit a frame */
     start_timer(s.seq);                       /* start the timer running */
  }
}
```

Figure 3-16. A 1-bit sliding window protocol.

协议4的正常情况（a）和异常情况（b）

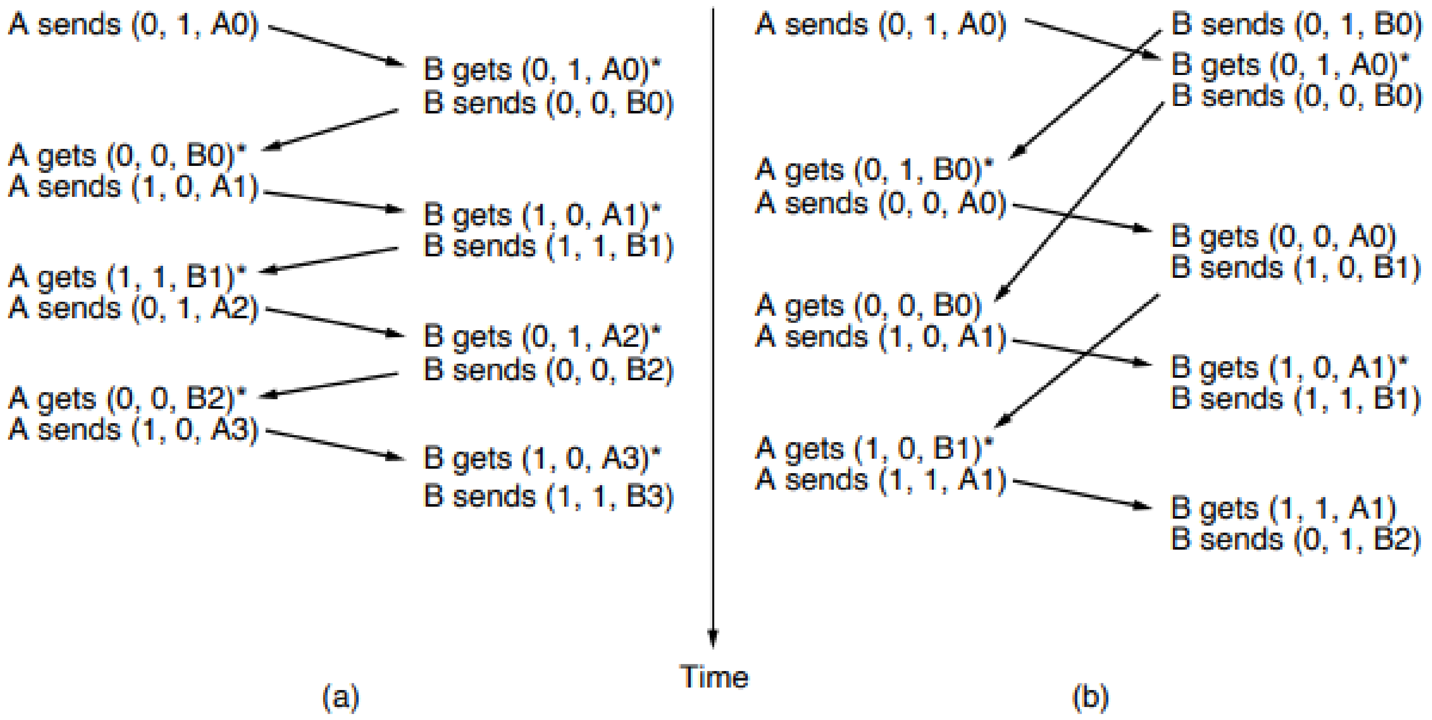


Figure 3-17. Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

5.4 滑动窗口协议（4）

5.4.2 退后n帧重传协议（协议5） （A Protocol Using Go Back n）

- 设计提高传输效率
  - 例：

    卫星信道传输速率50kbps，往返传输延迟500ms，若传1000bit的帧，使用协议4，则传输一个帧所需时间为：

    发送时间 + 信息信道延迟 + 确认信道延迟（确认帧很短，忽略发送时间） = 1000bit / 50kbps + 250ms + 250ms = 520ms

    信道利用率 = 20 / 520 ≈ 4%
  - 一般情况

    设：窗口大小为w=2BD，其中：B为带宽，D为单向传送时间；则信道利用率为：

$$\text{信道利用率} \le \frac{w}{1+2BD}$$

传输延迟大，信道带宽高，帧短时，信道利用率低。

5.4 滑动窗口协议（5）

- 解决办法

  连续发送多帧后再等待确认，称为流水线技术（pipelining），Fig-18
- 带来的问题

  信道误码率高时，对损坏帧和非损坏帧的重传非常多。

– 两种基本方法

  - 退后n帧（go back n）

    接收方从出错帧起丢弃所有后继帧；

    接收窗口为1；

    对于出错率较高的信道，浪费带宽。
  - 选择重传（selective repeat）

    接收窗口大于1，先暂存出错帧的后继帧；

    只重传坏帧；

    对最高序号的帧进行确认；

    接收窗口较大时，需较大缓冲区。

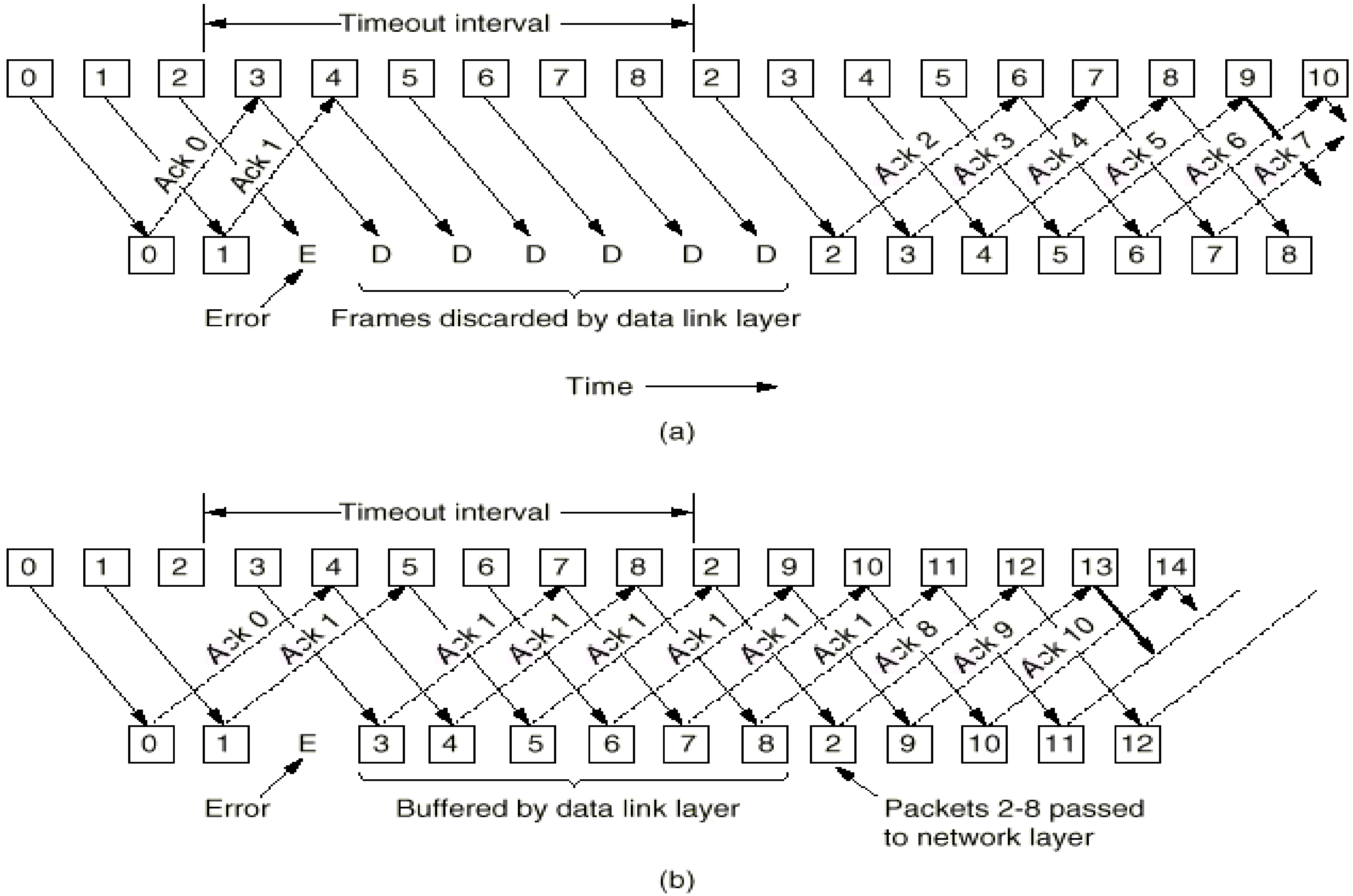


Fig-18 流水线和错误回复（a）接收窗口为1时的错误影响

（b）接收窗口很大时的错误影响

5.4 滑动窗口协议（5）

退后n帧重传协议

– 协议特点

- 发送方有流量控制，为重传设缓冲；
  发送窗口未满，EnableNetworkLayer
  发送窗口满，DisableNetworkLayer
- 发送窗口大小 < 序号个数（MaxSeq + 1）；
  考虑MaxSeq = 7的情况
  1 发送方发送帧 0 ~ 7;
  2 序号为 7 的帧的确认被捎带回发送方;
  3 发送方发送另外 8 个帧，序号为 0 ~ 7;
  4 另一个对帧 7 的捎带确认返回。
  问题：第二次发送的 8 个帧成功了还是丢失了?
- 退后n帧重发;
- 由于有多个未确认帧，设多个计时器。

5.4 滑动窗口协议（6）

- 工作过程

  Fig. 3-19
- 存在的问题

  隐含着信道负载重的假设。若一个方向负载重，另一个方向负载轻，则协议阻塞。
- 计时器实现

  Fig. 3-20

协议5：退回到n重传的滑动窗口协议 （1）

```
/* Protocol 5 (Go-back-n) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike the previous protocols,
   the network layer is not assumed to have a new packet all the time. Instead, the
   network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7        /* should be 2^n - 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"


static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if (a <=b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
        return(true);
    else
        return(false);
}


static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data frame. */
  frame s;      /* scratch variable */

  s.info = buffer[frame_nr];     /* insert packet into frame */
  s.seq = frame_nr;      /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);    /* piggyback ack */
  to_physical_layer(&s);        /* transmit the frame */
  start_timer(frame_nr);        /* start the timer running */
}
```

协议5：退回到n重传的滑动窗口协议（2）

```
void protocol5(void)
{
  seq_nr next_frame_to_send;     /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;  /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;         /* next frame expected on inbound stream */
  frame r;       /* scratch variable */
  packet buffer[MAX_SEQ+1];      /* buffers for the outbound stream */
  seq_nr nbuffered;      /* # output buffers currently in use */
  seq_nr i;      /* used to index into the buffer array */
  event_type event;

  enable_network_layer();        /* allow network_layer_ready events */
  ack_expected = 0;      /* next ack expected inbound */
  next_frame_to_send = 0;        /* next frame going out */
  frame_expected = 0;    /* number of frame expected inbound */
  nbuffered = 0;         /* initially no packets are buffered */


  while (true) {
     wait_for_event(&event);    /* four possibilities: see event_type above */

     switch(event) {
        case network_layer_ready:       /* the network layer has a packet to send */
                /* Accept, save, and transmit a new frame. */
                from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
                nbuffered = nbuffered + 1;      /* expand the sender's window */
                send_data(next_frame_to_send, frame_expected, buffer);  /* transmit the frame */
                inc(next_frame_to_send);        /* advance sender's upper window edge */
                break;

        case frame_arrival:     /* a data or control frame has arrived */
                from_physical_layer(&r);        /* get incoming frame from physical layer */

                if (r.seq == frame_expected) {
                        /* Frames are accepted only in order. */
                        to_network_layer(&r.info);      /* pass packet to network layer */
                        inc(frame_expected);    /* advance lower edge of receiver's window */
                 }
```

协议5：退回到n重传的滑动窗口协议（3）

```
while (true) {
   wait_for_event(&event);    /* four possibilities: see event_type above */

   switch(event) {
      case network_layer_ready:       /* the network layer has a packet to send */
              /* Accept, save, and transmit a new frame. */
              from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
              nbuffered = nbuffered + 1;      /* expand the sender's window */
              send_data(next_frame_to_send, frame_expected, buffer);  /* transmit the frame */
              inc(next_frame_to_send);        /* advance sender's upper window edge */
              break;

      case frame_arrival:     /* a data or control frame has arrived */
              from_physical_layer(&r);        /* get incoming frame from physical layer */

              if (r.seq == frame_expected) {
                      /* Frames are accepted only in order. */
                      to_network_layer(&r.info);      /* pass packet to network layer */
                      inc(frame_expected);    /* advance lower edge of receiver's window */
               }
```

协议5：退回到n重传的滑动窗口协议（4）

```
        /* Ack n implies n - 1, n - 2, etc.  Check for this. */
        while (between(ack_expected, r.ack, next_frame_to_send)) {
                /* Handle piggybacked ack. */
                nbuffered = nbuffered - 1;      /* one frame fewer buffered */
                stop_timer(ack_expected);       /* frame arrived intact; stop timer */
                inc(ack_expected);      /* contract sender's window */
        }
        break;

    case cksum_err: ;       /* just ignore bad frames */
        break;

    case timeout:   /* trouble; retransmit all outstanding frames */
        next_frame_to_send = ack_expected;      /* start retransmitting here */
        for (i = 1; i <= nbuffered; i++) {
                send_data(next_frame_to_send, frame_expected, buffer);  /* resend 1 frame */
                inc(next_frame_to_send);        /* prepare to send the next one */
        }
  }

  if (nbuffered < MAX_SEQ)
     enable_network_layer();
  else
     disable_network_layer();
 }
}
```

Fig-19 退回到n重传的滑动窗口协议

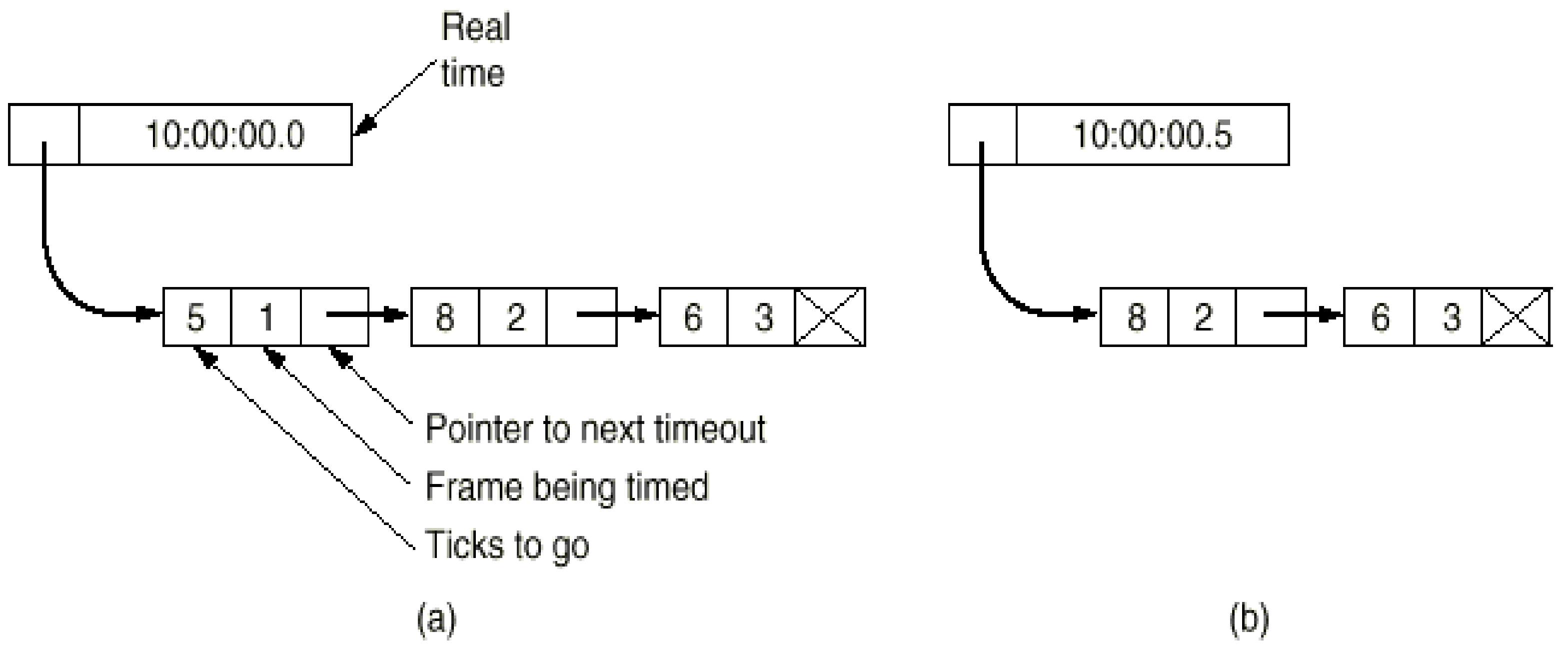


Fig. -20 软件实现的多个计时器

5.4 滑动窗口协议（7）

5.4.3 选择重传协议（协议6）（A Protocol Using Selective Repeat)

– 目的

在不可靠信道上有效传输时，不会因重传而浪费信道资源，采用选择重传技术。

– 基本原理

- 发送窗口大小：MaxSeq，接收窗口大小：(MaxSeq + 1) / 2

  保证接收窗口前移后与原窗口没有重叠；

  设 MaxSeq = 7, 若接收窗口 = 7

  发方发帧 0 ~ 6，收方全部收到，接收窗口前移（7 ~ 5），确认帧丢失，发方重传帧0，收方作为新帧接收，并对帧6确认，发方发新帧 7 ~ 5，收方已收过帧 0，丢弃新帧 0，协议出错。

  Fig. 3-21

  发送窗口下界：AckExpected，上界：NextFrameToSend

  接收窗口下界：FrameExpected，上界：TooFar

5.4 滑动窗口协议（8）

- 缓冲区设置
  发送方和接收方的缓冲区大小应等于各自窗口大小；
- 增加确认计时器，解决两个方向负载不平衡带来的阻塞问题；
- 可随时发送否定性确认帧NAK。

– 工作过程：Fig.-21
– 工作过程举例：Fig.-22

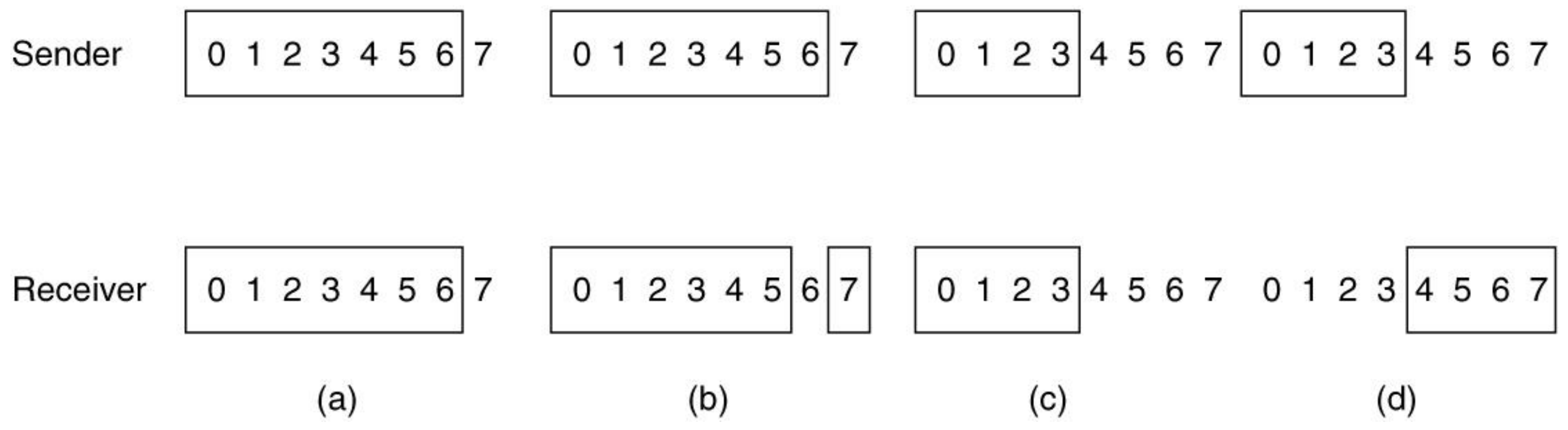


Fig.-22 选择重传的滑动窗口协议工作过程举例

协议6：选择重传的滑动窗口协议（1）

```
/* Protocol 6 (nonsequential receive) accepts frames out of order, but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   goes off, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7        /* should be 2^n - 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;  /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ+1;        /* init value is for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;      /* scratch variable */

  s.kind = fk;  /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;     /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;        /* one nak per frame, please */
  to_physical_layer(&s);        /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();     /* no need for separate ack frame */
}
```

协议6：选择重传的滑动窗口协议（2）

```
void protocol6(void)
{
  seq_nr ack_expected;  /* lower edge of sender's window */
  seq_nr next_frame_to_send;    /* upper edge of sender's window + 1 */
  seq_nr frame_expected;        /* lower edge of receiver's window */
  seq_nr too_far;       /* upper edge of receiver's window + 1 */
  int i;        /* index into buffer pool */
  frame r;      /* scratch variable */
  packet out_buf[NR_BUFS];      /* buffers for the outbound stream */
  packet in_buf[NR_BUFS];       /* buffers for the inbound stream */
  boolean arrived[NR_BUFS];     /* inbound bit map */
  seq_nr nbuffered;     /* how many output buffers currently used */
  event_type event;

  enable_network_layer();       /* initialize */
  ack_expected = 0;     /* next ack expected on the inbound stream */
  next_frame_to_send = 0;       /* number of next outgoing frame */
  frame_expected = 0;   /* frame number expected */
  too_far = NR_BUFS;    /* receiver's upper window + 1 */
  nbuffered = 0;        /* initially no packets are buffered */


  for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
  while (true) {
     wait_for_event(&event);    /* five possibilities: see event_type above */
     switch(event) {
        case network_layer_ready:       /* accept, save, and transmit a new frame */
                nbuffered = nbuffered + 1;      /* expand the window */
                from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
                send_frame(data, next_frame_to_send, frame_expected, out_buf);  /* transmit the frame */
                inc(next_frame_to_send);        /* advance upper window edge */
                break;
```

协议6：选择重传的滑动窗口协议（3）

```
         case frame_arrival:     /* a data or control frame has arrived */
                from_physical_layer(&r);        /* fetch incoming frame from physical layer */
                if (r.kind == data) {
                        /* An undamaged frame has arrived. */
                        if ((r.seq != frame_expected) && no_nak)
                            send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
                        if (between(frame_expected, r.seq, too_far) && (arrived[r.seq%NR_BUFS] == false)) {
                                /* Frames may be accepted in any order. */
                                arrived[r.seq % NR_BUFS] = true;        /* mark buffer as full */
                                in_buf[r.seq % NR_BUFS] = r.info;       /* insert data into buffer */
                                while (arrived[frame_expected % NR_BUFS]) {
                                      /* Pass frames and advance window. */
                                      to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                                      no_nak = true;
                                      arrived[frame_expected % NR_BUFS] = false;
                                      inc(frame_expected);    /* advance lower edge of receiver's window */
                                      inc(too_far);   /* advance upper edge of receiver's window */
                                      start_ack_timer();      /* to see if (a separate ack is needed */
                                }
                        }
               }

               if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_to_send))
                       send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

               while (between(ack_expected, r.ack, next_frame_to_send)) {
                       nbuffered = nbuffered - 1;       /* handle piggybacked ack */
                       stop_timer(ack_expected % NR_BUFS);      /* frame arrived intact */
                       inc(ack_expected);       /* advance lower edge of sender's window */
               }
               break;

        case cksum_err: if (no_nak) send_frame(nak, 0, frame_expected, out_buf); break; /* damaged frame */
        case timeout: send_frame(data, oldest_frame, frame_expected, out_buf); break;   /* we timed out */
        case ack_timeout: send_frame(ack,0,frame_expected, out_buf);    /* ack timer expired; send ack */
     }

     if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
  }
}
```

Fig.-21 选择重传的滑动窗口协议

第五章　数据链路层

第一部分　结束