

第五章

数据链路控制及其协议



主要内容

- **数据链路层概述**
 - 定义和功能
 - 为网络层提供的服务
- **组帧**
- **错误检测和纠正**
 - 纠错码
 - 检错码
- **基本的数据链路层协议**
 - 无约束单工协议
 - 单工停等协议
 - 有噪声信道的单工协议
- **滑动窗口协议**
 - 一比特滑动窗口协议
 - 退后n帧重传协议
 - 选择重传协议
- **协议说明与验证**
 - 协议形式化描述技术
 - 有限状态机模型
 - Petri网模型
- **常用的数据链路层协议**
 - 高级数据链路控制规程 HDLC
 - X.25的链路层协议 LAPB
 - PPP协议



定义和功能

■ 要解决的问题

- 如何在**有差错的线路上**，进行**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

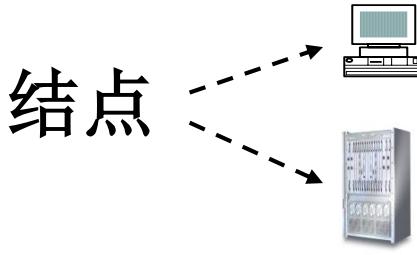


基本概念

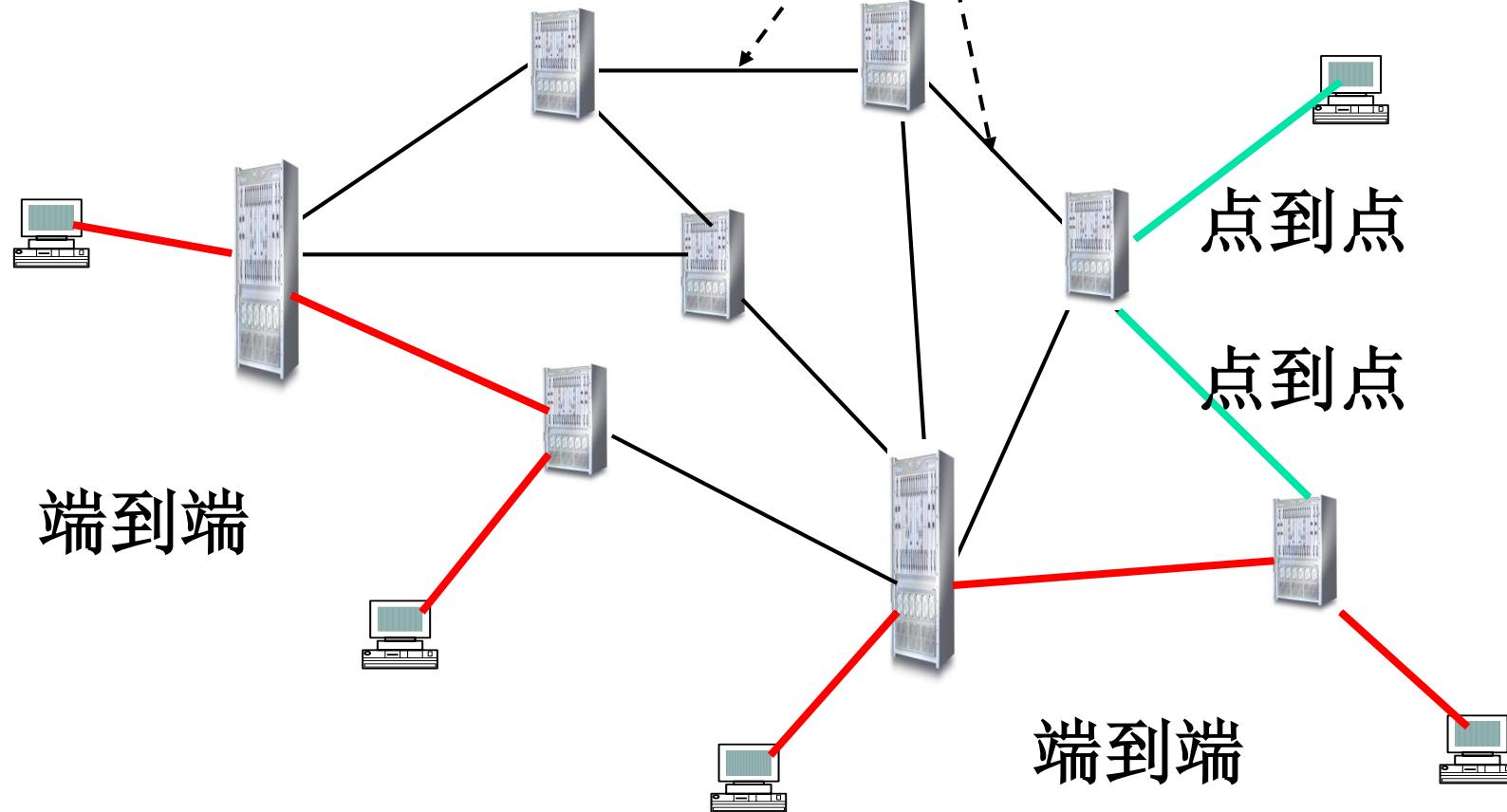
- **结点 (node)** : 网络中的**主机**和**网络设备** (路由器、交换机等)
- **链路 (link)** : 通信路径上连接相邻结点的**通信信道**
- 数据链路层协议定义了一条链路的两个结点间交换的**数据单元格式**,
以及结点**发送和接收**数据单元的动作
- **点到点 (point to point) 通信**: 一条链路上两个**相邻结点**间的通信
- **端到端 (end to end) 通信**: 从**源结点**到**目的结点**的通信, 通信路
径 (path) 可能由多个链路组成
- **实际数据通路与虚拟数据通路**



结点

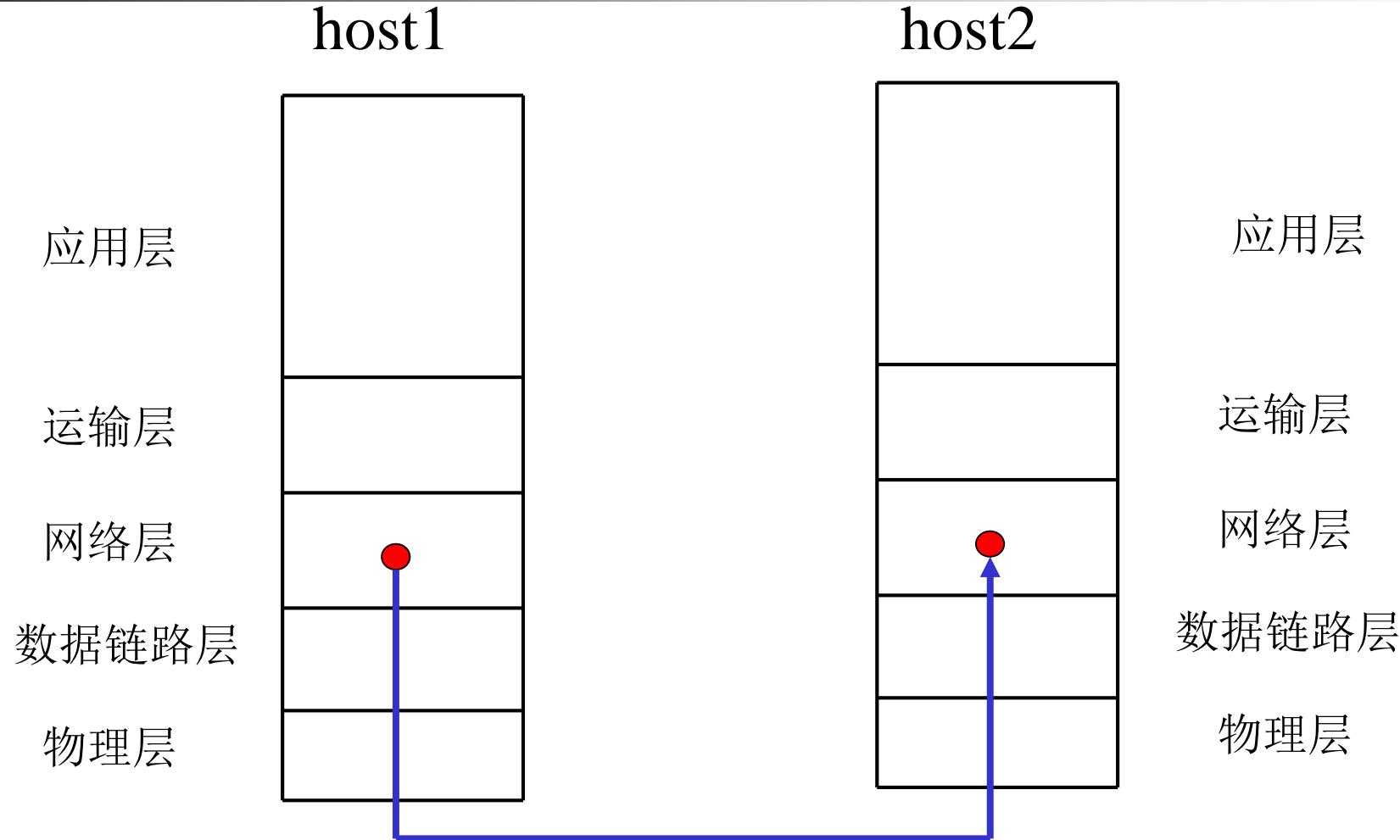


链路



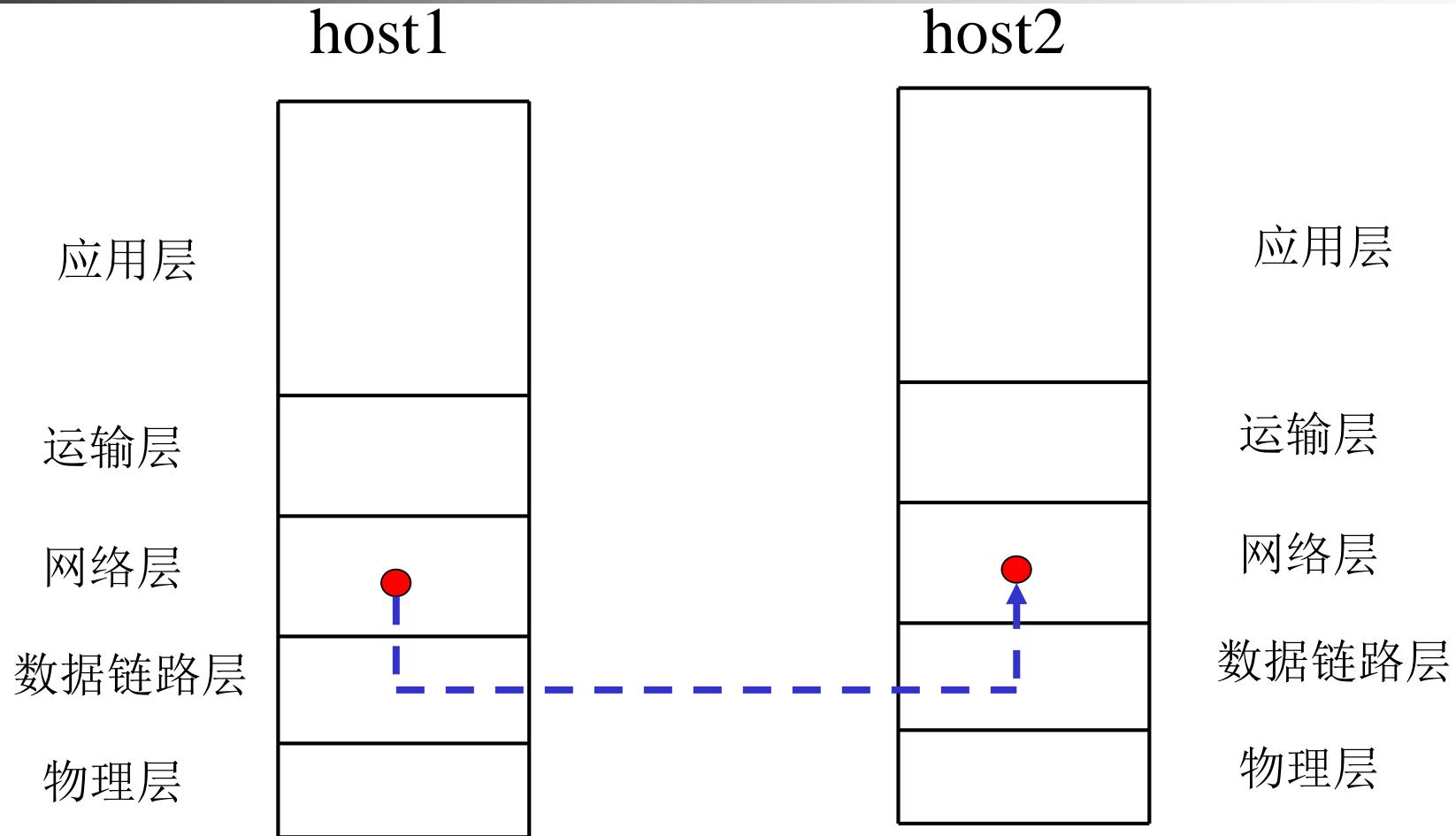


实际数据通路





虚拟数据通路





定义和功能（续）

■ 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基本功能

- 数据在数据链路上的正常传输（建立、维护和释放）
- 组帧：定界与同步，处理透明传输问题
- 差错控制：检错和纠错
- 顺序控制（可选）
- 流量控制（可选）：基于反馈机制



为网络层提供的服务

- 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



主要内容

- **数据链路层概述**
 - 定义和功能
 - 为网络层提供的服务
- **组帧**
- **错误检测和纠正**
 - 纠错码
 - 检错码
- **基本的数据链路层协议**
 - 无约束单工协议
 - 单工停等协议
 - 有噪声信道的单工协议
- **滑动窗口协议**
 - 一比特滑动窗口协议
 - 退后n帧重传协议
 - 选择重传协议
- **协议说明与验证**
 - 协议形式化描述技术
 - 有限状态机模型
 - Petri网模型
- **常用的数据链路层协议**
 - 高级数据链路控制规程 HDLC
 - X.25的链路层协议 LAPB
 - PPP协议



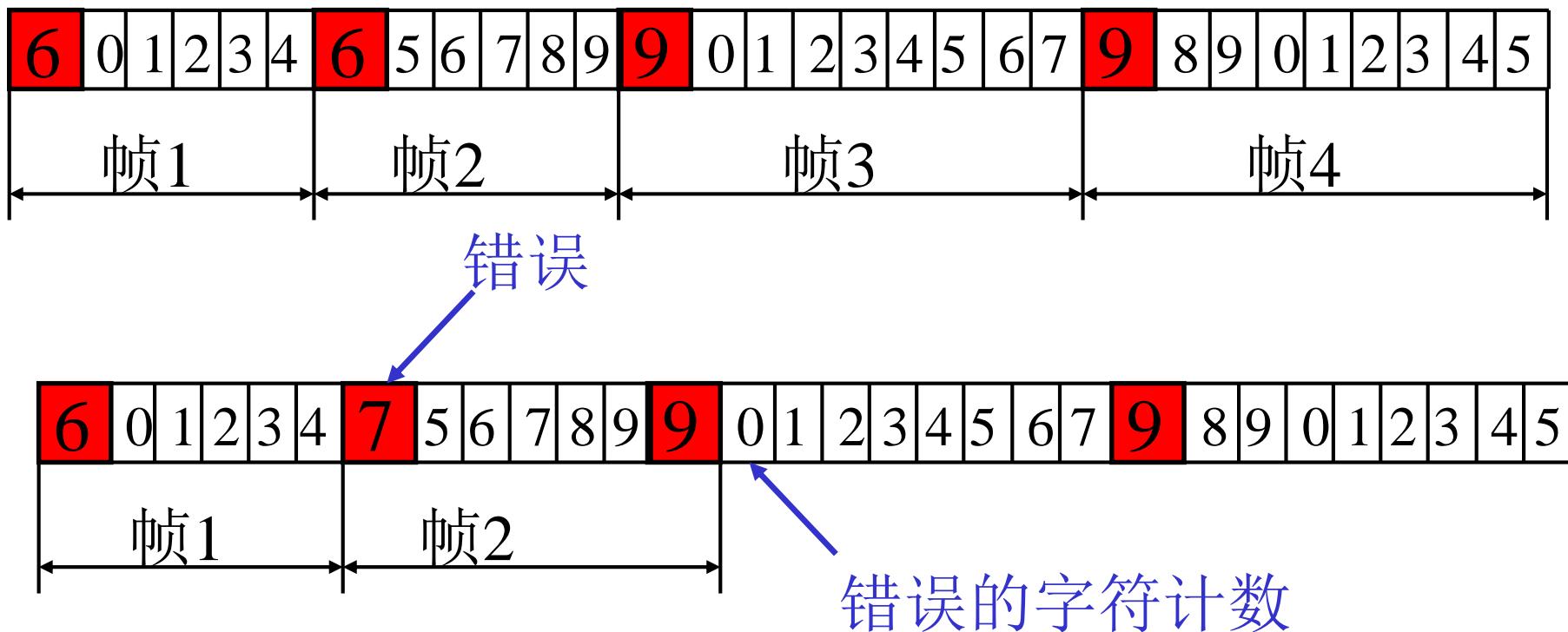
组帧 (Framing)

- 将比特流分成离散的帧，并计算每个帧的校验和
- 组帧方法
 - 字符计数法
 - 带字符填充的字符定界法
 - 带位填充的标记定界法
 - 物理层编码违例法
- 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



字符计数法

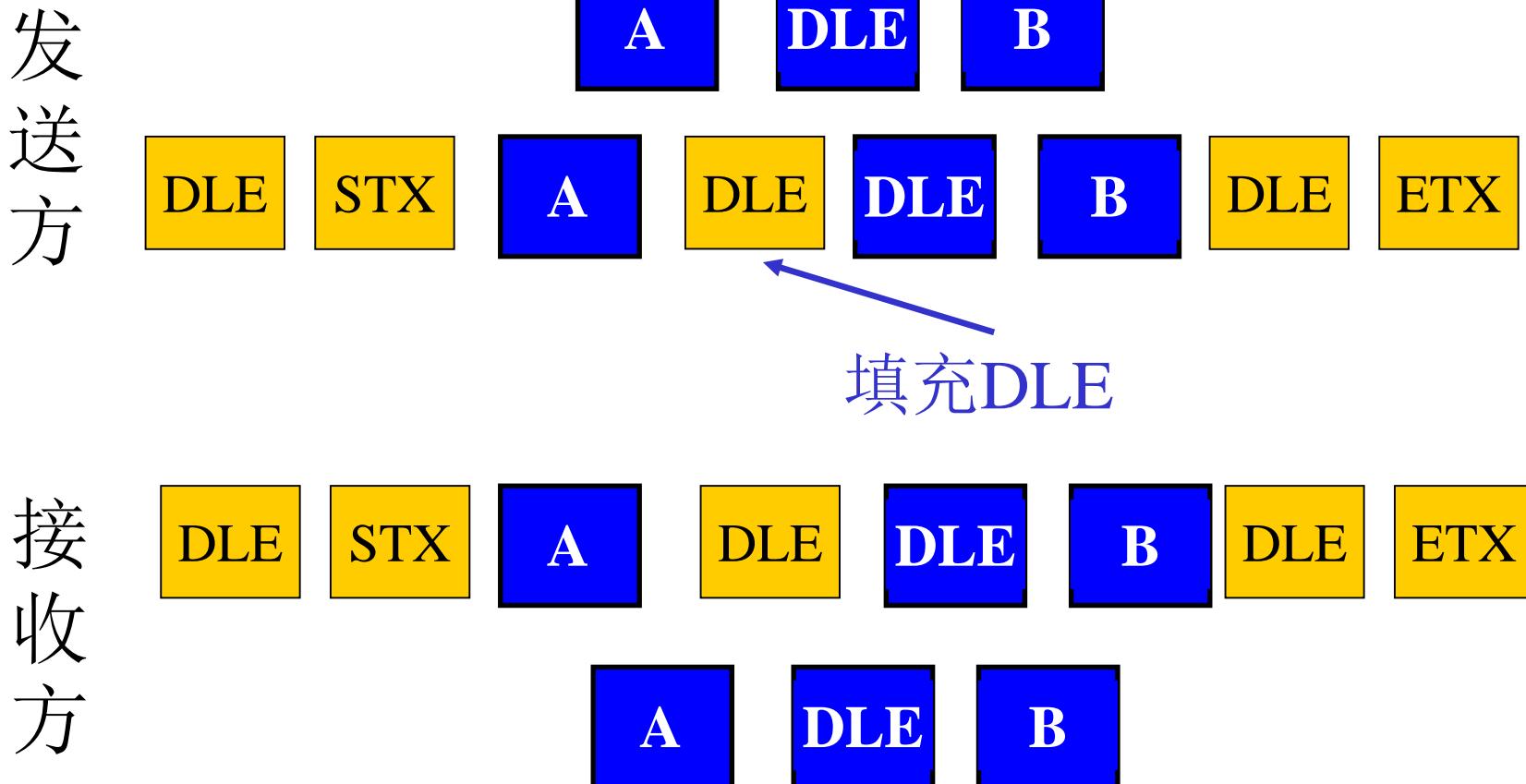
- 在帧头中用一个域来表示整个帧的字符个数
- 缺点：若计数出错，对本帧和后面的帧有影响





带字符填充的字符定界法

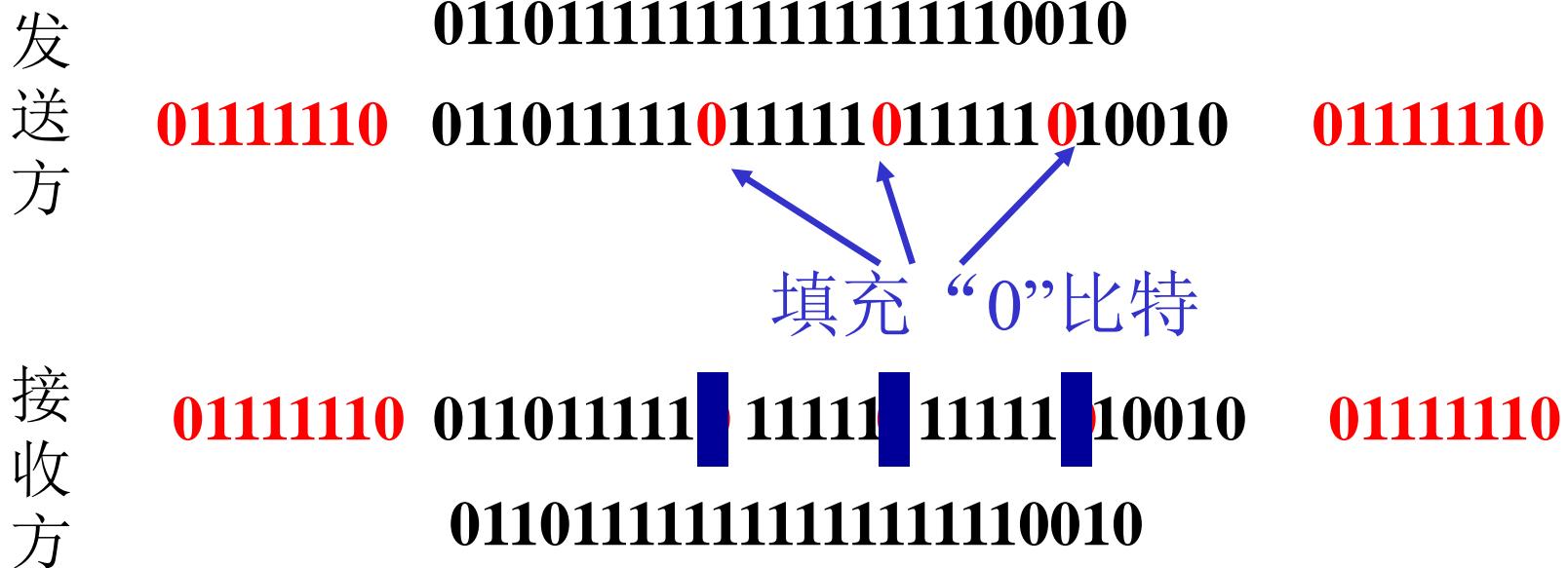
- **起始字符 DLE STX, 结束字符 DLE ETX**
 - DLE: Data Link Escape
 - STX: Start of Text
 - ETX: End of Text
- **字符填充**
- **缺点:** 局限于8位字符和ASCII字符传送





带位填充的标记定界法

- 帧的起始和结束都用一个特殊的位串“01111110”，称为标记(flag)
- “0”比特插入删除技术





物理层编码违例法

- 只适用于物理层编码有冗余的网络
- **802 LAN**: 曼彻斯特编码用 high-low pair/low-high pair 表示1/0, high-high/low-low 不表示数据, 可以用来做定界符



主要内容

- **数据链路层概述**
 - 定义和功能
 - 为网络层提供的服务
- **组帧**
- **错误检测和纠正**
 - 纠错码
 - 检错码
- **基本的数据链路层协议**
 - 无约束单工协议
 - 单工停等协议
 - 有噪声信道的单工协议
- **滑动窗口协议**
 - 一比特滑动窗口协议
 - 退后n帧重传协议
 - 选择重传协议
- **协议说明与验证**
 - 协议形式化描述技术
 - 有限状态机模型
 - Petri网模型
- **常用的数据链路层协议**
 - 高级数据链路控制规程 HDLC
 - X.25的链路层协议 LAPB
 - PPP协议



错误检测和纠正

- **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



纠错码

- **码字 (codeword)** : 一个帧包括 m 个数据位, 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)** : 两个码字之间不同的对应比特位数目
 - 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 (比特错), 可以使用海明距离为 $d + 1$ 的编码
 - 为了纠正 d 个错, 可以使用海明距离为 $2d + 1$ 的编码



纠错码（续）

■ 最简单的例子是奇偶校验，在数据后填加一个奇偶位

- 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奇数个错误

■ 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： $(n + 1) 2^m \leq 2^n$
- 利用 $n = m + r$ ，得到 $(m + r + 1) \leq 2^r$ 。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



纠错码（续）

■ 海明码

- 码位从左边开始编号，从 “1” 开始
- 位号为2的幂的位是校验位，其余是信息位
- 每个校验位使得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： $11 = 1 + 2 + 8$



Char.	ASCII	Check bits	1	2	3	4	5	6	7	8	9	10	11
H	1001000	00110010000	1										
a	1100001	10111001001		2								2	2
m	1101101	11101010101			4	4	4						
m	1101101	11101010101										8	8
i	1101001	01101011001											
n	1101110	01101010110											
g	1100111	11111001111											
Ø	0100000	10011000000											
c	1100011	11111000011											
ø	1101111	00101011111											
d	1100100	11111001100											
e	1100101	00111000101											

Order of bit transmission

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



纠错码（续）

■ 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 ($k = 1, 2, 4 \dots$)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。例：若校验位1、2、8出错，则第11位变反

■ 使用海明码纠正突发错误

- 可采用 k 个码字 ($n = m + r$) 组成 $k \times n$ 矩阵，按列发送，接收方恢复成 $k \times n$ 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



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
 - 生成多项式应该比传输信息对应的多项式短



检错码（续）

■ CRC码基本思想

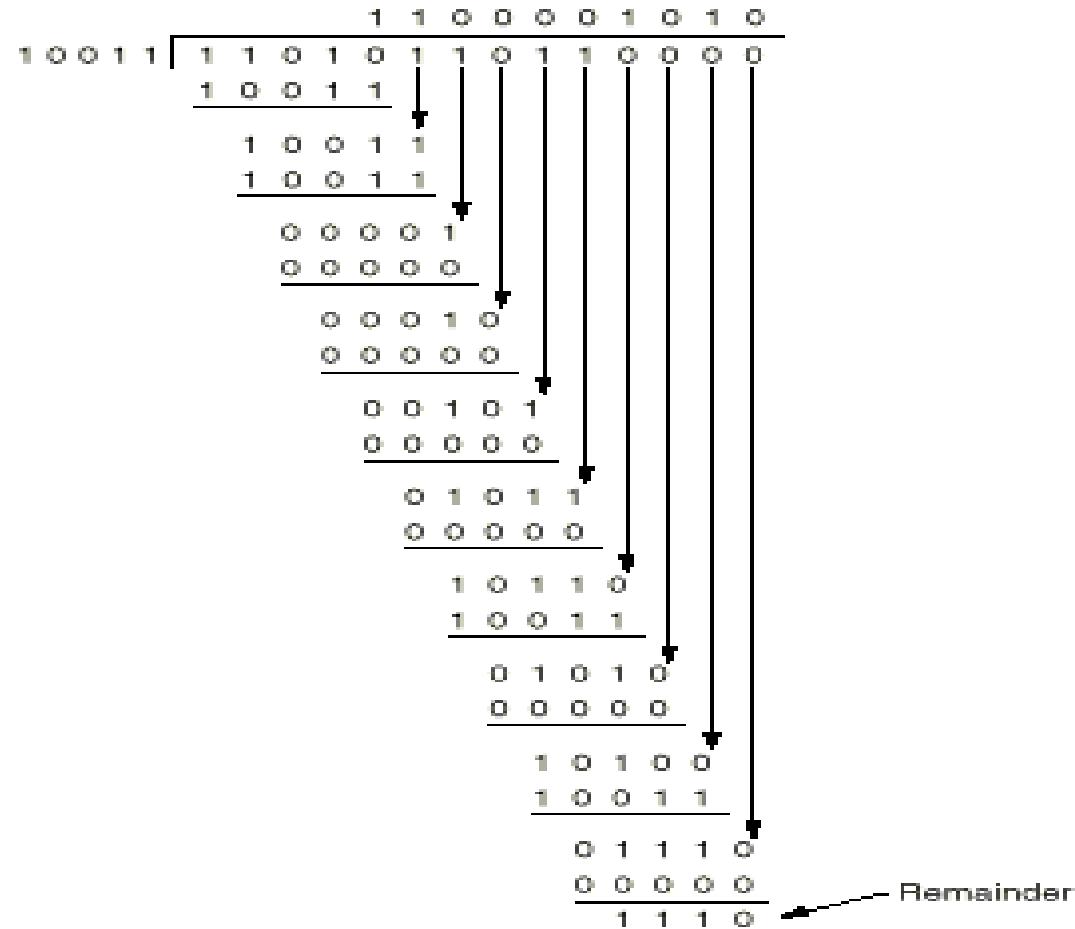
- 校验和 (checksum) 加在帧尾，使带校验和的帧的多项式能被 $G(x)$ 除尽；收方接收时，用 $G(x)$ 去除它，若有余数，则传输出错

■ 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 $x^r M(x)$
- 按模2除法用对应于 $G(x)$ 的位串去除对应于 $x^r M(x)$ 的位串
- 按模2减法从对应于 $x^r M(x)$ 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 $T(x)$



Frame : 1 1 0 1 0 1 1 0 1 1
Generator: 1 0 0 1 1
Message after appending 4 zero bits: 1 1 0 1 0 1 1 0 0 0 0



Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 0

Fig. 3-7. Calculation of the polynomial code checksum.



检错码（续）

■ CRC的检错能力

- 发送: $T(x)$
- 接收: $T(x) + E(x)$, $E(x) \neq 0$
- 余数($(T(x) + E(x)) / G(x)$) = 0 + 余数($E(x) / G(x)$)
- 若余数($E(x) / G(x)$) = 0, 则差错不能发现; 否则, 可以发现



检错码（续）

■ CRC检错能力的几种情况分析 (1)

- 如果只有单比特错，即 $E(x) = x^i$ ，而 $G(x)$ 中至少有两项，余数($E(x) / G(x)$) $\neq 0$ ，所以可以查出单比特错
- 如果发生两个孤立单比特错，即 $E(x) = x^i + x^j = x^j(x^{i-j} + 1)$ ， $G(x)$ 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 ($k \leq i - j$)
- 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



检错码（续）

■ CRC检错能力的几种情况分析 (2)

- 具有 r 个校验位的多项式能检查出所有长度 $\leq r$ 的突发性差错。长度为 k 的突发性连续差错可表示为 $x^i (x^{k-1} + \dots + 1)$, 若 $G(x)$ 包括 x^0 项, 且 $k - 1$ 小于 $G(x)$ 的阶, 则 余数($E(x) / G(x)$) $\neq 0$
- 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, 余数($E(x) / G(x)$) = 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 $1/2^r$



检错码（续）

■ 四个国际标准生成多项式

- CRC-12 $= x^{12} + x^{11} + x^3 + x^2 + x + 1$
- CRC-16 $= x^{16} + x^{15} + x^2 + 1$
- CRC-CCITT $= x^{16} + x^{12} + x^5 + 1$
- CRC-32 $= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

■ 硬件实现CRC校验

- 网卡NIC (Network Interface Card)



主要内容

- **数据链路层概述**
 - 定义和功能
 - 为网络层提供的服务
- **组帧**
- **错误检测和纠正**
 - 纠错码
 - 检错码
- **基本的数据链路层协议**
 - 无约束单工协议
 - 单工停等协议
 - 有噪声信道的单工协议
- **滑动窗口协议**
 - 一比特滑动窗口协议
 - 退后n帧重传协议
 - 选择重传协议
- **协议说明与验证**
 - 协议形式化描述技术
 - 有限状态机模型
 - Petri网模型
- **常用的数据链路层协议**
 - 高级数据链路控制规程 HDLC
 - X.25的链路层协议 LAPB
 - PPP协议

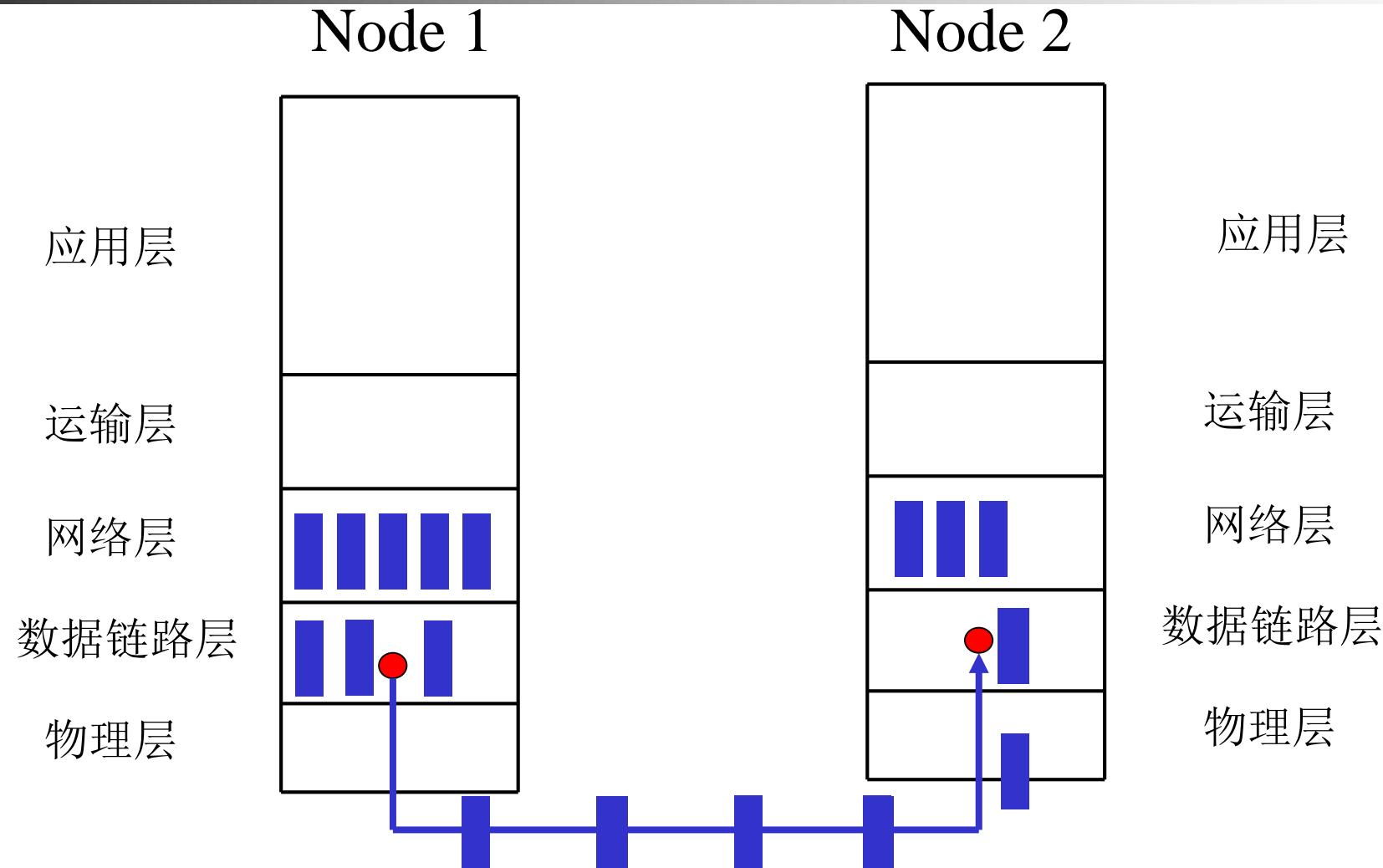


无约束单工协议

- 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



无约束单工协议（续）





/* 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 fast.

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 */
    }
}
```

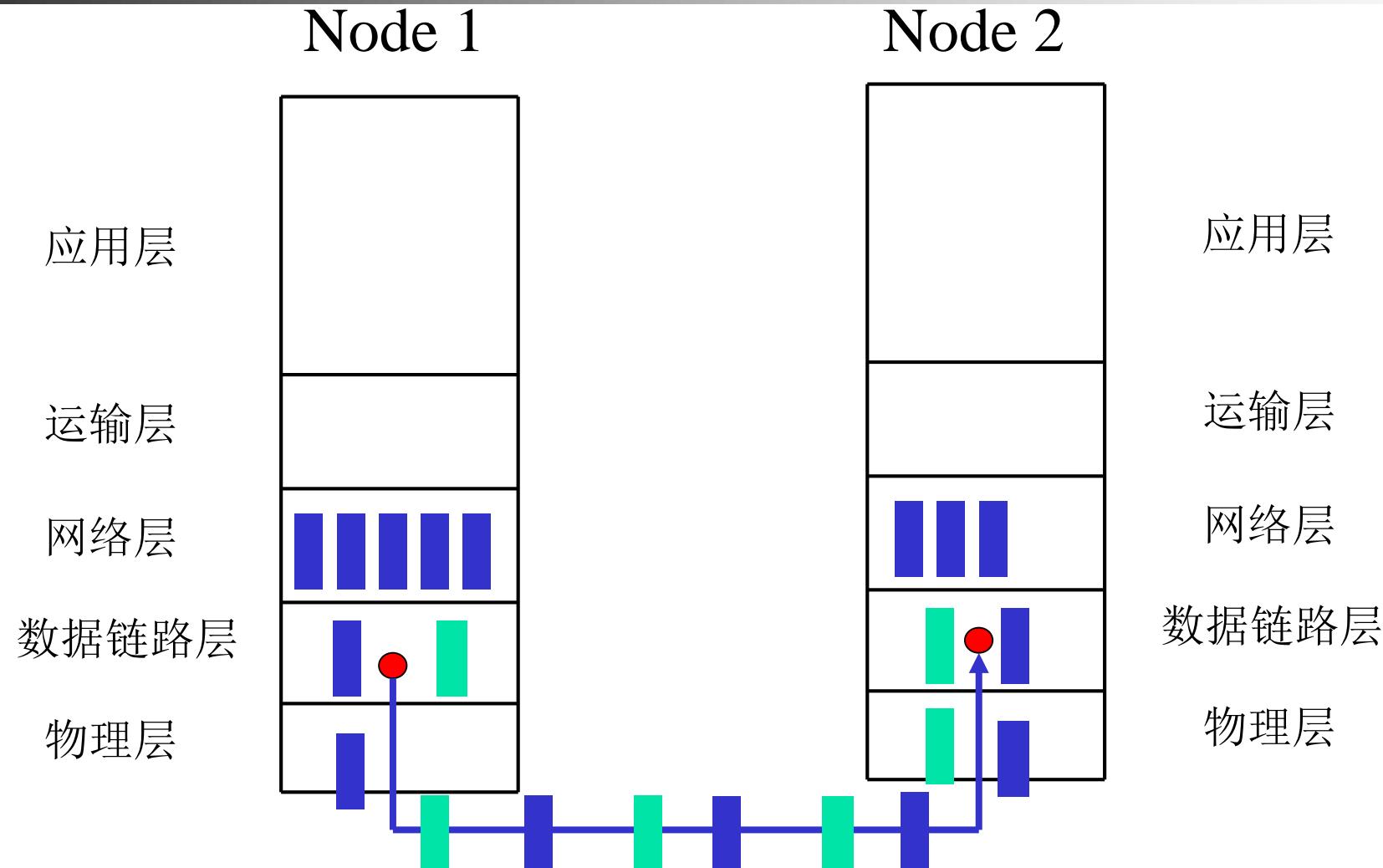


单工停等协议

- 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，组帧，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



单工停等协议（续）





```
/* 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 */
    }
}
```



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

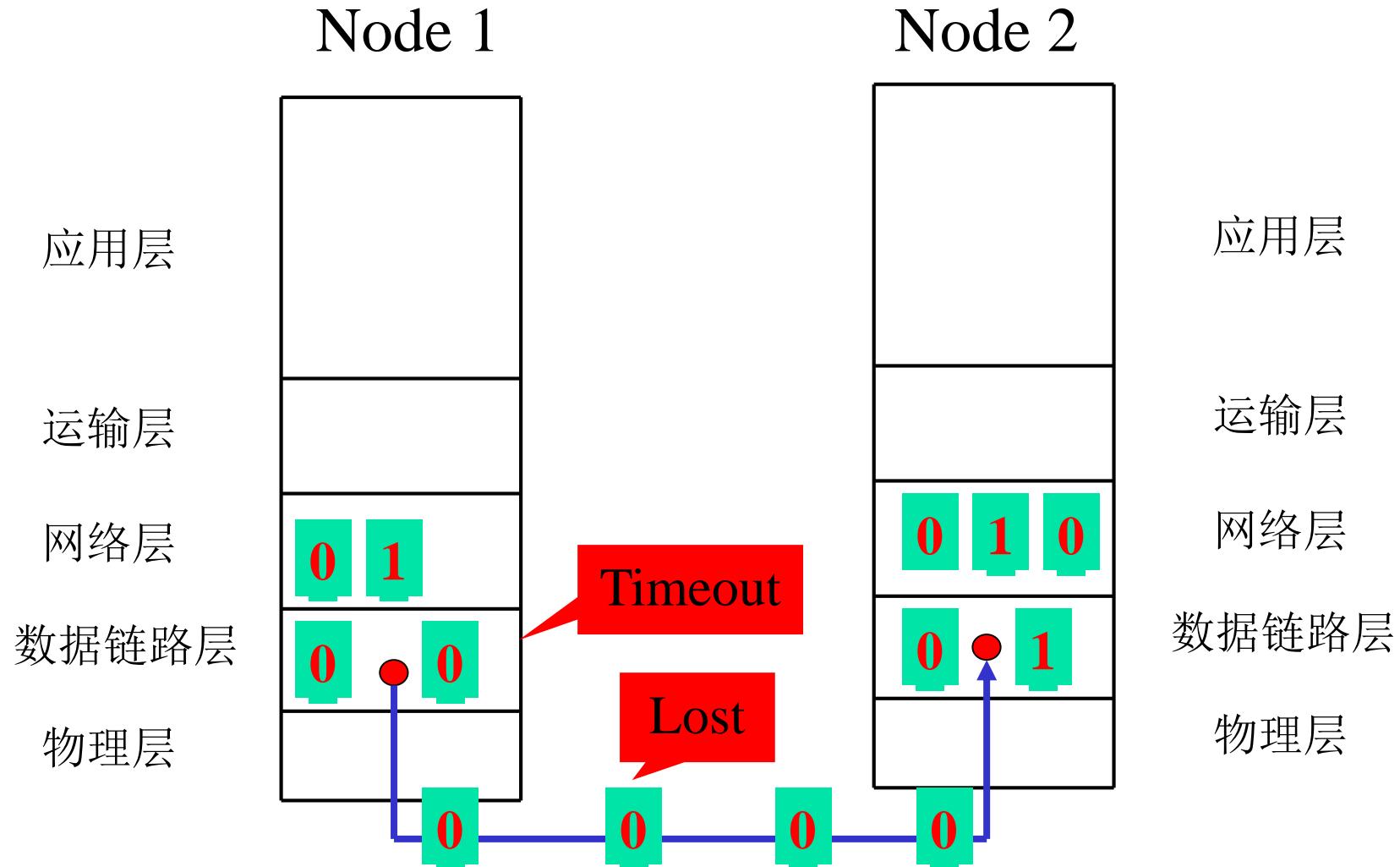


有噪声信道的单工协议（续）

- **发方在发下一个帧之前等待一个肯定确认的协议叫做PAR或ARQ**
 - PAR: Positive Acknowledgement with Retransmission
 - ARQ: Automatic Repeat reQuest
- **工作过程**



有噪声信道的单工协议 (续)





```
/* 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) {
                from_network_layer(&buffer); /* get the next one to send */
                inc(next_frame_to_send); /* invert next_frame_to_send */
            }
        }
    }
}

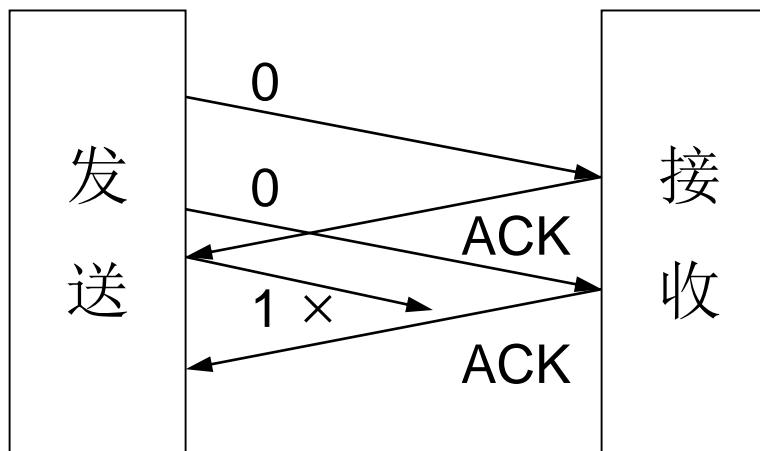
void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;

    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); /* only the ack field is use */
        }
    }
}
```



有噪声信道的单工协议（续）

- 如果确认帧没有序号，则协议3有漏洞
 - 由于确认帧中没有序号，超时时间不能太短，否则协议失败





主要内容

- **数据链路层概述**
 - 定义和功能
 - 为网络层提供的服务
- **组帧**
- **错误检测和纠正**
 - 纠错码
 - 检错码
- **基本的数据链路层协议**
 - 无约束单工协议
 - 单工停等协议
 - 有噪声信道的单工协议
- **滑动窗口协议**
 - 一比特滑动窗口协议
 - 退后n帧重传协议
 - 选择重传协议
- **协议说明与验证**
 - 协议形式化描述技术
 - 有限状态机模型
 - Petri网模型
- **常用的数据链路层协议**
 - 高级数据链路控制规程 HDLC
 - X.25的链路层协议 LAPB
 - PPP协议



滑动窗口协议

- 单工 ——> 全双工
- 捎带/载答 (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**



滑动窗口协议（续）

■ 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
 - 发送窗口的下界表示未得到确认的帧的最小序号。
 - 发送窗口大小 = 上界 - 下界，大小可变

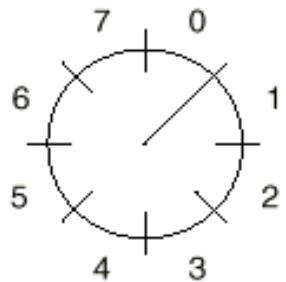
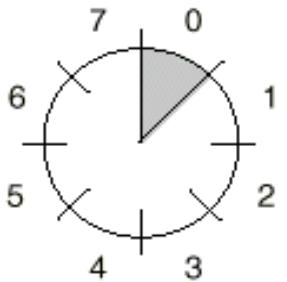
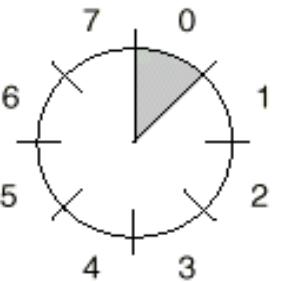
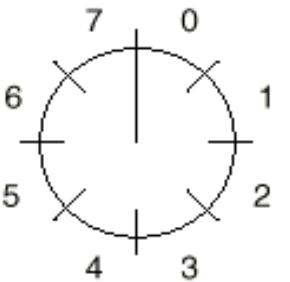


滑动窗口协议（续）

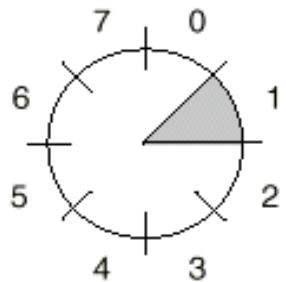
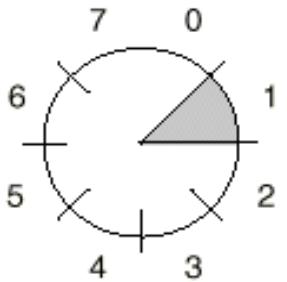
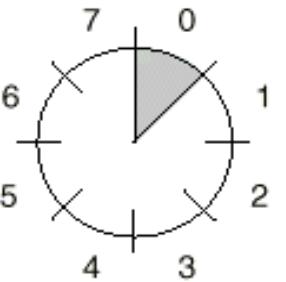
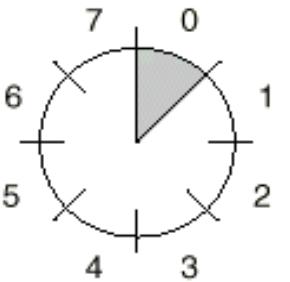
- 发送端每发送一个帧，序号取上界值，上界加1；每接收到一个确认序号 = 发送窗口下界的正确响应帧，下界加1；若确认序号落在发送窗口之内，则发送窗口下界连续加1，直到发送窗口下界 = 确认序号 + 1（累计确认）
- 接收端有一个接收窗口，大小固定，但不一定与发送窗口相同。接收窗口的上界表示允许接收的最大序号，下界表示希望接收的序号
- 接收窗口容纳允许接收的信息帧，落在窗口外的帧均被丢弃。序号等于下界的帧被正确接收，并产生一个响应帧，上界、下界都加1。接收窗口大小不变



Sender



Receiver



(a)

(b)

(c)

(d)

Fig. 3-12. 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.



一比特滑动窗口协议

- 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: 发送序号和确认序号 (已经正确收到的帧的序号)
- 工作过程

```
/* Protocol 4 (sliding window) is bidirectional and is more robust than protocol 3. */
```

```
#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;            /* number of frame arriving frame expected */
    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); /* could be: frame_arrival, cksum_err, 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 sequence number expected next */
        }

        if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
            from_network_layer(&buffer);  /* fetch new packet from network layer */
            inc(next_frame_to_send);      /* invert sender's sequence number */
        }
    }

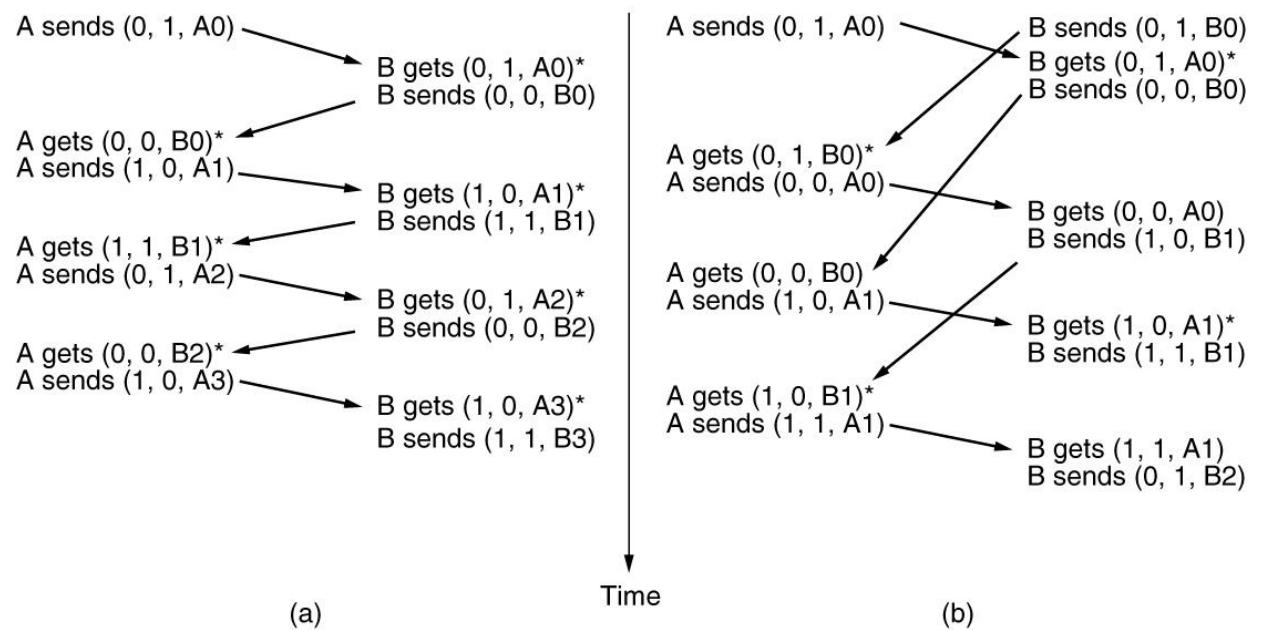
    s.info = buffer;           /* construct outbound frame */
    s.seq = next_frame_to_send; /* insert sequence number into it */
    s.ack = 1 - frame_expected; /* seq number of last received frame */
    to_physical_layer(&s);    /* transmit a frame */
    start_timer(s.seq);       /* start the timer running */
}
```



一比特滑动窗口协议（续）

■ 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





滑动窗口协议（续）

■ 一比特滑动窗口协议传输效率低

- 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 = 发送时间 + 信息信道延迟 + 确认信道延迟
 - 其中：确认帧很短，忽略发送时间
 - 传输一个帧所需时间 = $1000\text{bit} / 50\text{kbps} + 500\text{ms} = 520\text{ms}$
 - 信道利用率 = $20 / 520 \approx 4\%$



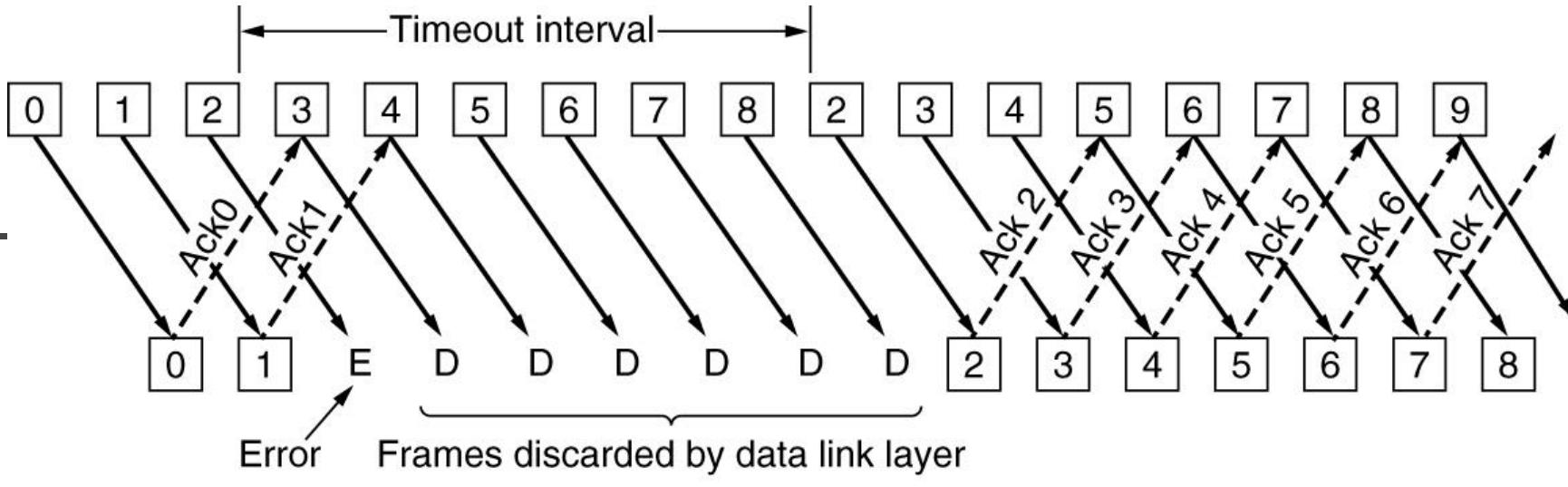
滑动窗口协议（续）

- 一般情况，信道带宽 b 比特/秒，帧长度 L 比特，往返传输延迟 R 秒
 - 则信道利用率为：
$$\frac{\frac{L}{b}}{\frac{L}{b} + R} = \frac{L}{L + Rb}$$
- 结论
 - 传输延迟大，信道带宽高，帧短时：信道利用率低
- 解决办法
 - 流水线技术：连续发送多帧后再等待确认
- 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

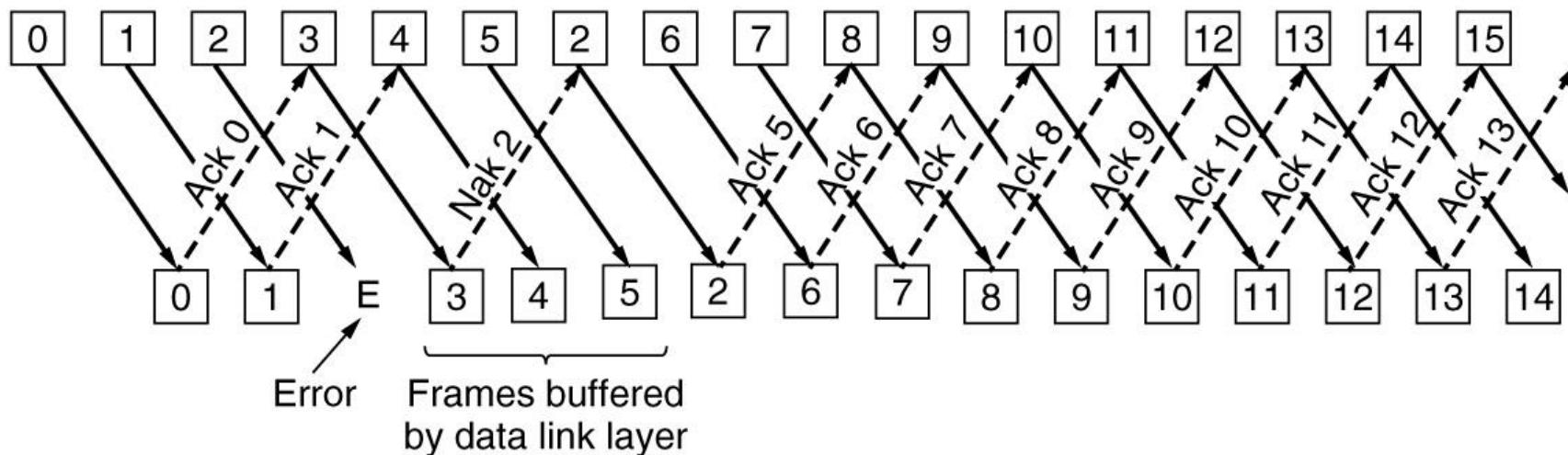


滑动窗口协议（续）

- **两种改进方法**
 - 退后n帧重传 (go back n)
 - 发送窗口大于1，接收窗口为1
 - 接收方从坏帧起丢弃所有后继帧，发送方从坏帧开始重传
 - 对于出错率较高的信道，浪费带宽
 - 选择重传 (selective repeat)
 - 发送窗口大于1，接收窗口大于1
 - 接收方可暂存坏帧的后继帧，发送方只重传坏帧
 - 接收窗口较大时，需较大缓冲区



(a)



(b)



退后n帧重传协议

- **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
- **发送窗口尺寸 < 序号个数 (MaxSeq + 1)**
 - 考虑 MaxSeq = 7 的情况
 - 发送方发送帧 0 ~ 7
 - 序号为 7 的帧的确认被捎带回发送方
 - 发送方发送另外 8 个帧，序号为 0 ~ 7
 - 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？（累计确认）



```
/* Protocol 5 (pipelining) 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 */
}
```



```
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 */

    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 */
            }

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

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();
}
```

From: *Computer Networks*, 3rd ed. by Andrew S. Tanenbaum, © 1996 Prentice Hall

Fig. 3-16. A sliding window protocol using go back n.



计时器

- 由于有多个未确认帧，需要设多个计时器

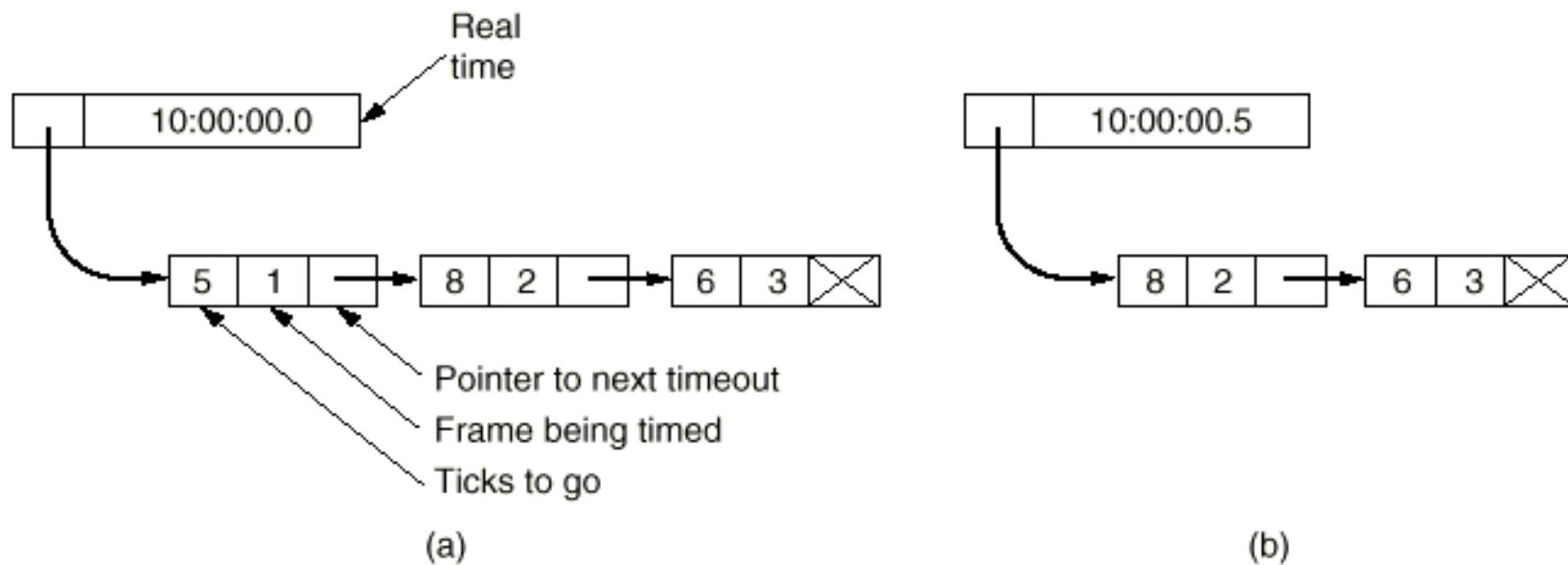


Fig. 3-17. Simulation of multiple timers in software.



退后n帧重传协议协议实现分析

■ 事件驱动

- Network_layer_ready (内部事件)
 - 发送帧 (帧序号, 确认序号, 数据)
- Frame_arrival (外部事件)
 - 检查帧序号, 落在接收窗口内则接收, 否则丢弃
 - 检查确认序号, 落在发送窗口内则移动发送窗口, 否则不做处理
- Cksum_err (外部事件)
 - 丢弃
- timeout (内部事件)
 - 退后n帧重传



退后n帧重传协议协议实现分析（续）

■ 计时器处理

- 启动：发送帧时启动
- 停止：收到正确确认时停止
- 超时：产生 timeout 事件



选择重传协议

■ 目的

- 采用选择重传技术在不可靠信道上传输时，不会因不必要的重传而浪费信道资源

■ 窗口设置

- 要求：发送窗口尺寸+接收窗口尺寸 \leq 序号个数
- 设 MaxSeq = 7, 若发送窗口和接收窗口尺寸 = 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
- 设 MaxSeq = 7, 若发送窗口=7, 接收窗口尺寸 = 4, 发方发帧 0 ~ 6, 收方全部收到, 接收窗口前移 (7 ~ 2), 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



Sender	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table>	0	1	2	3	4	5	6	7	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table>	0	1	2	3	4	5	6	7	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr></table>	0	1	2	3	4	5	6	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table>	0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7																												
0	1	2	3	4	5	6	7																												
0	1	2	3	4	5	6																													
0	1	2	3	4	5	6	7																												
Receiver	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table>	0	1	2	3	4	5	6	7	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table>	0	1	2	3	4	5	6	7	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr></table>	0	1	2	3	4	5	6	<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr></table>	0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7																												
0	1	2	3	4	5	6	7																												
0	1	2	3	4	5	6																													
0	1	2	3	4	5	6	7																												

Fig. 3-19. (a) Initial situation with a window of size seven. (b) After seven frames have been sent and received but not acknowledged. (c) Initial situation with a window size of four. (d) After four frames have been sent and received but not acknowledged.



选择重传协议（续）

- 窗口参数
 - 发送窗口下界: AckExpected
 - 发送窗口上界: NextFrameToSend
 - 接收窗口下界: FrameExpected
 - 接收窗口上界: TooFar
- 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



```

/* 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;             /* initial value is only 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 */
}

```



```
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;
    too_far = NR_BUFS;
    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;

        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);/* damaged frame */
    break;

case timeout:
    send_frame(data, oldest_frame, frame_expected, out_buf);/* we timed out */
    break;

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();
}
```



选择重传协议（续）

■ 协议实现分析

- 事件驱动（1）
 - Network_layer_ready（内部事件）
 - 发送帧（帧类型，帧序号，确认序号，数据）
 - Frame_arrival（外部事件）
 - 若是数据帧，则检查帧序号，落在接收窗口内则接收，否则丢弃；不等于接收窗口下界还要发NAK
 - 若是NAK，则选择重传
 - 检查确认序号，落在发送窗口内则移动发送窗口，否则不做处理



选择重传协议（续）

■ 协议实现分析

- 事件驱动（2）
 - Cksum_err（外部事件）：发送NAK
 - timeout（内部事件）：选择重传
 - Ack_timeout（内部事件）：发送确认帧ACK



选择重传协议（续）

- 计时器处理
 - 启动：发送数据帧时启动
 - 停止：收到正确确认时停止
 - 超时：产生timeout事件
- Ack计时器处理
 - 启动：收到帧的序号等于接收窗口下界或已经发过 NAK 时启动
 - 停止：发送帧时停止
 - 超时：产生 ack_timeout 事件



小结

-
- **可靠传输**
 - 纠错编码
 - 检错码、确认和重传机制
 - **传送层协议TCP主要负责可靠传输服务**
 - **链路层的可靠传输服务通常用于高误码率的连路上，如无线链路**
 - **对于误码率低的链路，链路层协议可以不实现可靠传输功能**



主要内容

- **数据链路层概述**
 - 定义和功能
 - 为网络层提供的服务
- **组帧**
- **错误检测和纠正**
 - 纠错码
 - 检错码
- **基本的数据链路层协议**
 - 无约束单工协议
 - 单工停等协议
 - 有噪声信道的单工协议
- **滑动窗口协议**
 - 一比特滑动窗口协议
 - 退后n帧重传协议
 - 选择重传协议
- **协议说明与验证**
 - 协议形式化描述技术
 - 有限状态机模型
 - Petri网模型
- **常用的数据链路层协议**
 - 高级数据链路控制规程 HDLC
 - X.25的链路层协议 LAPB
 - PPP协议



协议说明与验证

- 协议工程与协议的形式化描述技术
- 协议工程
 - 协议说明 (Protocol Specification)
 - 协议验证 (Protocol Verification)
 - 协议实现 (Protocol Implementation)
 - 协议测试 (Protocol Testing)
 - 一致性测试 (Conformance Testing)
 - 互操作性测试 (Interoperability Testing)
 - 性能测试 (Performance Testing)



协议工程

■ 协议说明

- 既定义一个协议实体提供给它的用户的服务，又定义该协议实体的内部操作

■ 协议验证

- 验证协议说明是否完整正确
- 用于系统实现前的设计阶段，避免可能出现的设计错误
- 以协议说明为基础，涉及逻辑证明，原则上验证协议所有可能的状态
- 可达性分析是一种常用的验证方法
 - 利用图论知识可以解决状态的可达性问题
 - 可达性分析能够用来解决协议的不完整性、死锁和无关变迁等问题



协议工程（续）

- 协议实现
 - 用硬件和/或软件实现协议说明中规定的功能
- 协议测试
 - 用测试的方法来检查协议实现是否满足要求，包括
 - 协议实现是否与协议说明一致（一致性测试）
 - 协议实现之间的互操作能力（互操作性测试）
 - 协议实现的性能（性能测试）等



形式化描述技术

- 在协议的说明、验证、实现和测试过程中使用**形式化描述技术**，不仅可以比较容易地理解协议，而且可以使协议描述更加精确，大大简化了协议的研究工作
- **形式化描述的意义**
 - 实际使用的协议非常复杂，给协议的理解、验证、实现和测试等工作带来困难，需要采用形式化的、数学的描述方法来描述协议
 - 但是目前大多数协议还是采用自然语言描述
- **自然语言描述协议的缺点**
 - 冗余
 - 多义性
 - 结构性不好
 - 不便于自动验证、测试、实现



形式化描述技术（续）

- **形式化描述技术FDT (Formal Description Technique) /形式化方法FM (Formal Method)** 广泛应用于协议工程研究中
- 一种形式化方法总是以一种**形式体系**为基础，只是在具体应用时，大都做了便于描述的改进和扩充



形式化描述技术（续）

■ 常用的形式化方法

- 有限状态机FSM (Finite State Machine)
 - 扩展: EFSM
- 形式化语言模型
 - LOTOS, Estelle, SDL
 - 都有相应扩展
- Petri网
 - 扩展: 时间Petri网, 随机Petri网, 高级Petri网
- 进程代数 (Process Algebra)
 - 扩展: 随机进程代数



有限状态机模型

■ 定义

- 一个有限状态机是一个四元组(S, M, I, T)，其中 S 是状态的集合， M 是标号的集合， I 是初始状态的集合， T 是变迁的集合

■ 网络协议建模

- 基本出发点：认为通信协议主要是由响应多个“事件”的处理过程组成
- 事件
 - 命令（来自用户）
 - 信息到达（来自底层）
 - 内部超时

■ 优点：简单明了，比较精确

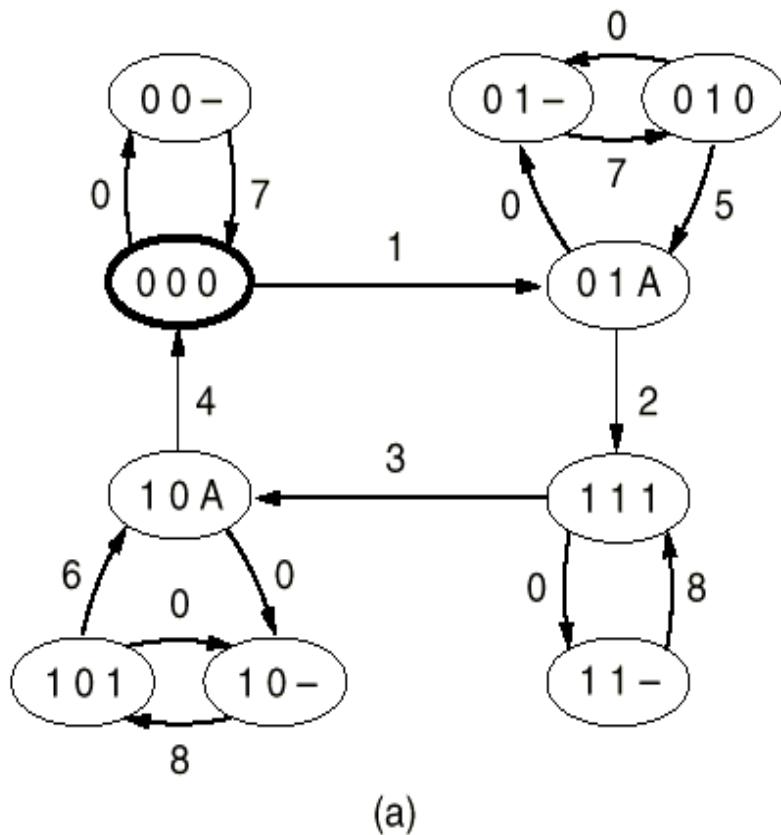
■ 缺点：复杂协议的事件数和状态数会剧增，状态爆炸



有限状态机模型（续）

■ 例，协议3

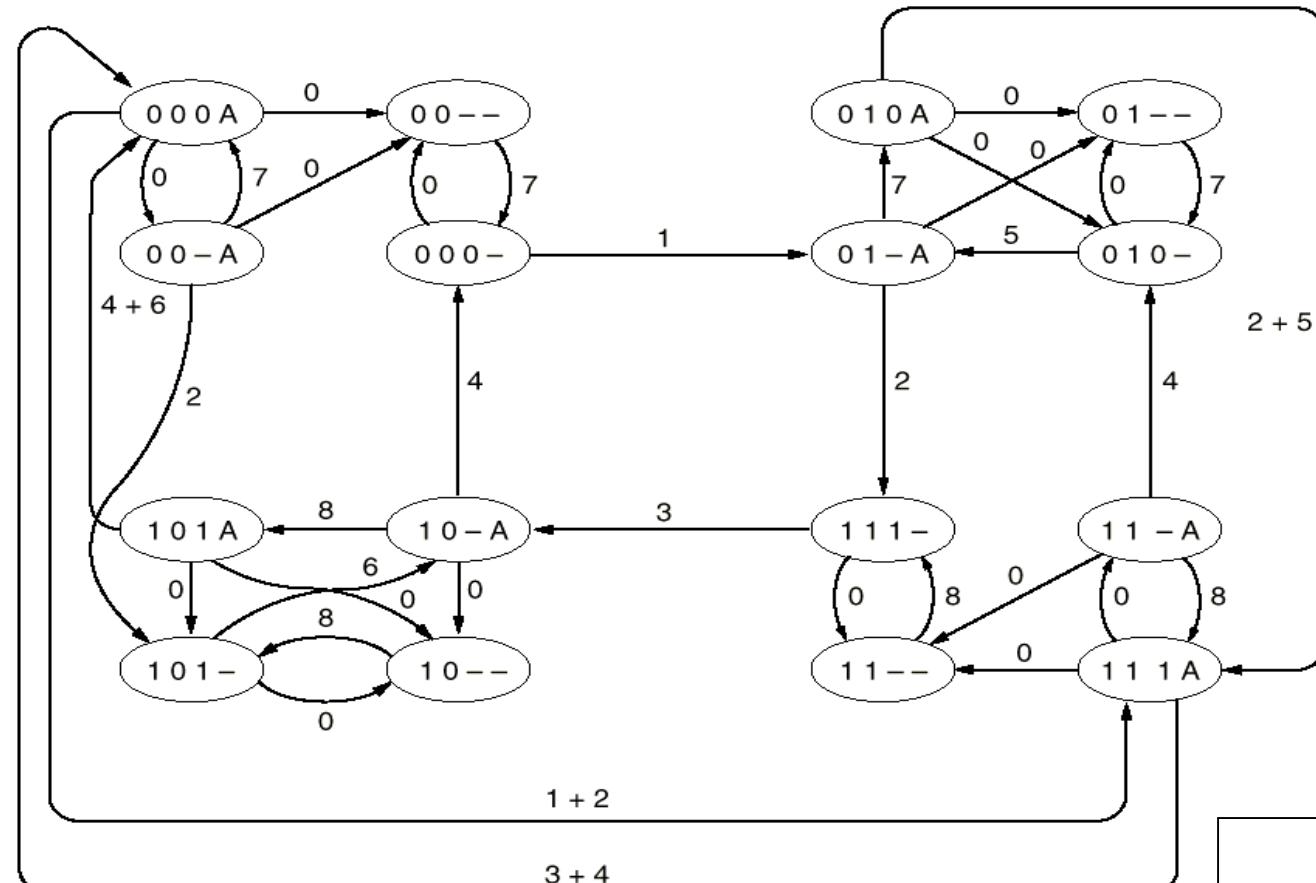
- 每个状态用三个字母表示：XYZ
 - X：发送方发送的帧序号，为 0 或 1
 - Y：接收方准备接收的帧序号，为 0 或 1
 - Z：信道状态，为 0, 1, A 或 - (空)
- 初始状态为 (000)
- 半双工信道
- 全双工信道



(b)

Transition	Who runs?	Frame accepted (frame lost)	Frame emitted	To network layer
0	-			-
1	R	0	A	Yes
2	S	A	1	-
3	R	1	A	Yes
4	S	A	0	-
5	R	0	A	No
6	R	1	A	No
7	S	(timeout)	0	-
8	S	(timeout)	1	-

Fig. 3-20. (a) State diagram for protocol 3. (b) Transitions.



(a)

(0 0 0 -), (0 1 - A), (0 1 0 A), (1 1 1 A), (1 1 - A), (0 1 0 -), (0 1 - A), (1 1 1 -)

(b)

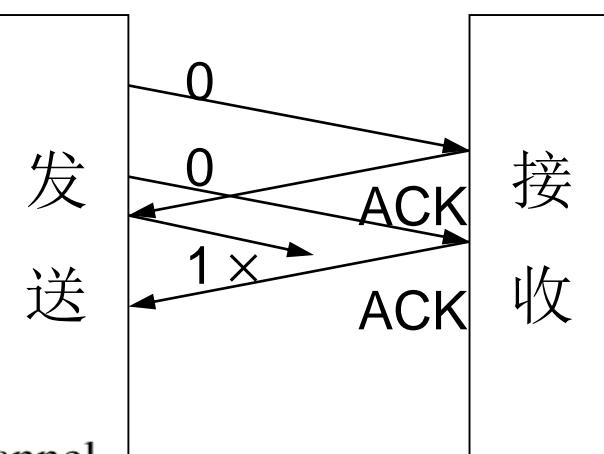


Fig. 3-21. (a) State graph for protocol 3 and a full-duplex channel.
(b) Sequence of states causing the protocol to fail.



Petri网模型

- Petri网模型最早在1962年 Carl Adam Petri的博士论文中提出来，**主要特性**：是具有较强的对**并行、不确定性、异步和分布**的描述能力和分析能力
- **Petri网研究的系统模型行为特性包括**
 - 状态的可达 (reachability)
 - 位置的限界 (boundedness)
 - 变迁的活性 (liveness)
 - 初始状态的可逆达 (reversibility)
 - 标识 (marking) 之间的可达 (reachability)
 - 事件之间的同步距离 (synchronic distance)
 - 公平性 (fairness)



Petri网模型（续）

■ Petri网定义

■ 结构元素

- 位置 (place) : 描述系统状态, 用一个圆圈表示
- 变迁 (transition) : 描述修改系统状态的事件, 用一个长方形或线段表示
- 弧 (arc) : 描述状态与事件之间的关系, 包括输入弧和输出弧, 用有向弧表示

■ 活动元素 —— 标记 (token)

- 包含在位置中, 用点表示
- 用来表示处理的信息单元、资源单元和顾客、用户等对象
- 如果位置用来描述条件, 它可以包含一个标记或不包含标记, 当包含标记时, 条件为真, 否则, 为假
- 如果位置用来定义状态, 位置中的标记个数用于规定这个状态



Petri网模型（续）

■ 变迁实施规则 (firing rule)

- 如果一个变迁的所有输入位置至少包含一个标记，则这个变迁可能实施
- 一个可实施变迁的实施导致从它所有输入位置中都清除一个标记，在它所有输出位置中产生一个标记
- 当使用大于1的弧权 (weight) 时，在变迁的所有输入位置中都要包含至少等于连接弧权的标记个数它才可实施，并根据弧权，在每个输出位置中产生相应标记个数
- 变迁的实施是一个原子操作，输入位置清除标记和输出位置产生标记是一个不可分割的完整操作

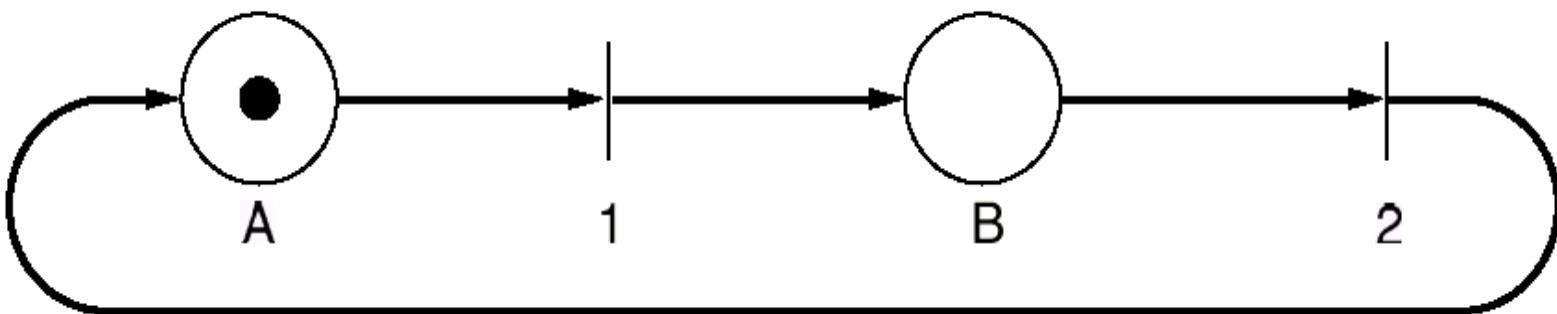


Fig. 3-22. A Petri net with two places and two transitions.



Petri网模型（续）

■ Petri网的组成

- 条件/事件 (C/E) 网
 - 每个位置最多一个标记，表示条件
- 位置/变迁 (P/T) 网
 - 每个位置中的标记可以有多个



Petri网模型（续）

■ 高级Petri网

- 包括谓词/变迁网、着色网等
- 给标记以属性，即标记有区别
- 从没有参数的网，发展到时间Petri网和随机Petri网
- 从一般有向弧发展到可变弧
- 从自然数标记个数发展到概率标记个数
- 从原子变迁发展到谓词变迁和子网变迁



Petri网模型（续）

■ 例：协议3（半双工信道）

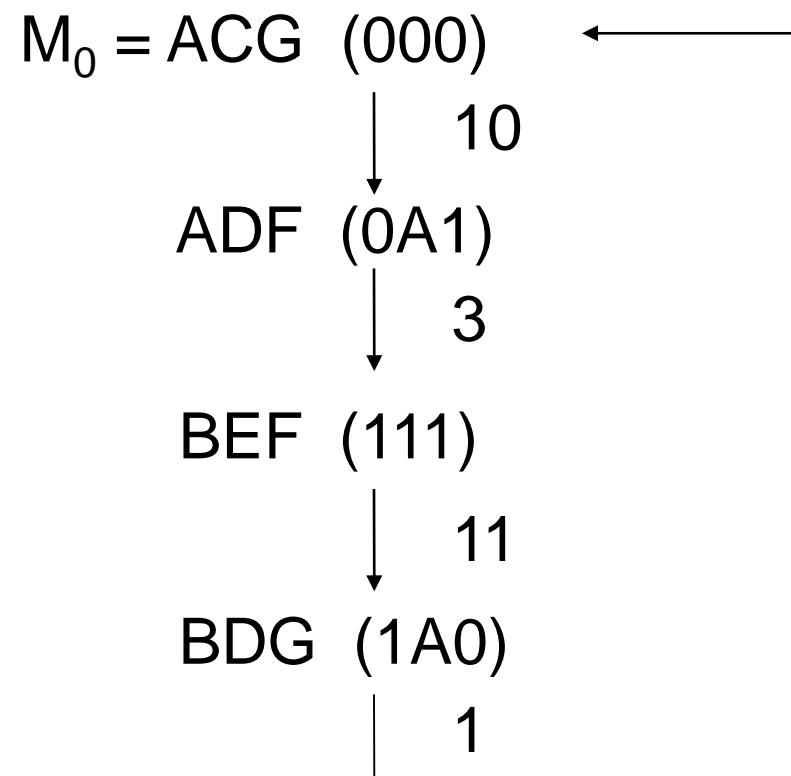
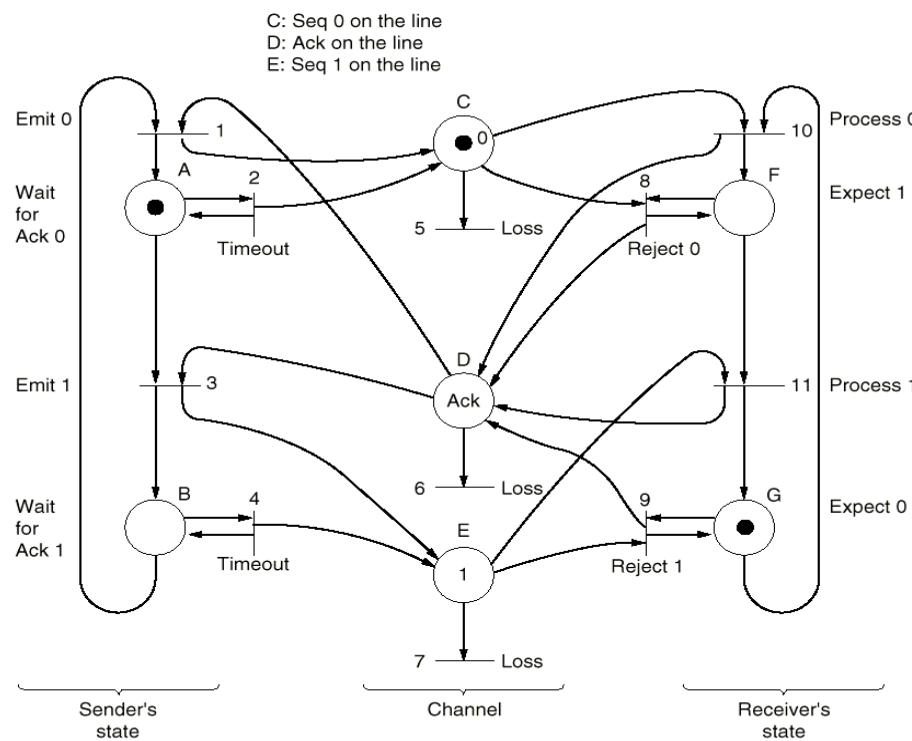


Fig. 3-23. A Petri net model for protocol 3.



形式化方法的补充说明

- 模型描述能力的增强会在某种程度上增加模型分析的难度
- **模型仅仅是手段，不是目的**
 - Modeling is a bridge.



主要内容

- **数据链路层概述**
 - 定义和功能
 - 为网络层提供的服务
- **组帧**
- **错误检测和纠正**
 - 纠错码
 - 检错码
- **基本的数据链路层协议**
 - 无约束单工协议
 - 单工停等协议
 - 有噪声信道的单工协议
- **滑动窗口协议**
 - 一比特滑动窗口协议
 - 退后n帧重传协议
 - 选择重传协议
- **协议说明与验证**
 - 协议形式化描述技术
 - 有限状态机模型
 - Petri网模型
- **常用的数据链路层协议**
 - 高级数据链路控制规程 HDLC
 - X.25的链路层协议 LAPB
 - PPP协议



常用的数据链路层协议

- ISO和CCITT在数据链路层协议的标准制定方面做了大量工作，各大公司也形成了自己的标准
- **数据链路层协议分类（1）**
 - 面向字符的链路层协议
 - ISO 的 IS1745, 基本型传输控制规程及其扩充部分 (BM 和 XBM)
 - IBM的二进制同步通信规程 (BSC)
 - DEC的数字数据通信报文协议 (DDCMP)
 - PPP



常用的数据链路层协议

■ 数据链路层协议分类 (2)

- 面向比特的链路层协议
 - IBM的SNA使用的数据链路协议SDLC
 - ANSI修改SDLC，提出ADCCP
 - ISO修改SDLC，提出HDLC
 - CCITT修改HDLC，提出LAP和LAPB



高级数据链路控制规程HDLC

- 1976年，ISO提出HDLC (High-level Data Link Control)
- HDLC的组成
 - 帧结构
 - 规程元素 }
 - 规程类型 语义
 - 使用 HDLC 的语法可以定义多种具有不同操作特点的链路层协议
- HDLC的适用范围
 - 计算机与计算机通信
 - 计算机与终端通信
 - 终端与终端通信



HDLC（续）

- **数据站（简称站 station），由计算机（路由器）和终端组成，负责发送和接收帧。HDLC涉及三种类型的站**
 - 主站（primary station）
 - 主要功能是发送命令（包括数据），接收响应，负责整个链路的控制（如系统的初始、流控、差错恢复等）
 - 次站（secondary station）
 - 主要功能是接收命令，发送响应，配合主站完成链路的控制
 - 组合站（combined station）
 - 同时具有主、次站功能，既发送又接收命令和响应，并负责整个链路的控制



HDLC（续）

■ HDLC适用的链路构型

- 非平衡型
 - 点到点式



- 点到多点式

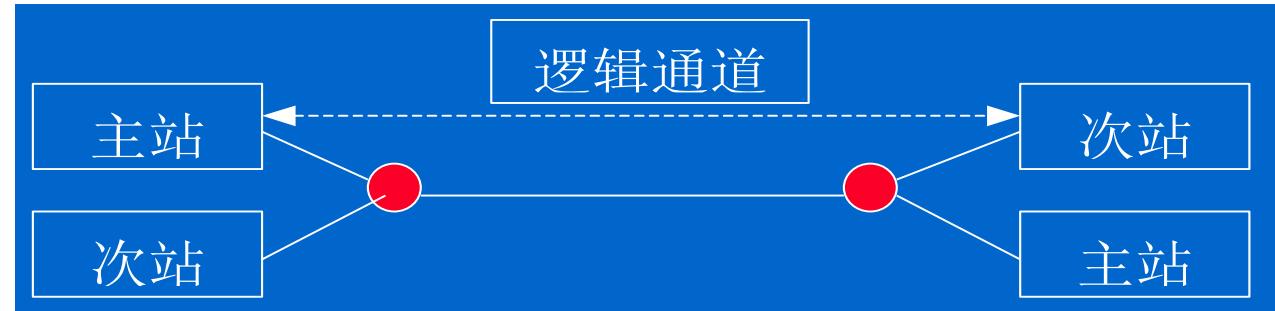


- 适合把智能和半智能的终端连接到计算机



HDLC (续)

- 平衡型
 - 主站 — 次站式



- 组合式
 - 适合于计算机和计算机之间的连接





HDLC的基本操作模式

■ 正规响应模式 NRM

- 适用于点到点式和多点式两种非平衡构型。只有当主站向次站发出探询后，次站才能获得传输帧的许可

■ 异步响应模式 ARM

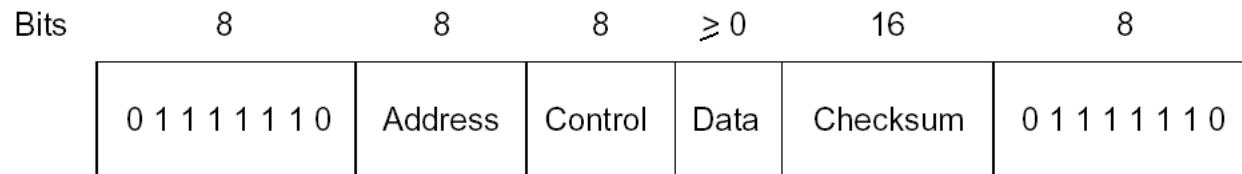
- 适用于点到点式非平衡构型和主站—次站式平衡构型。次站可以随时传输帧，不必等待主站的探询

■ 异步平衡模式 ABM

- 适用于通信双方都是组合站的平衡构型，也采用异步响应，双方具有同等能力



HDLC帧结构



- 定界符
 - 01111110
 - 空闲的点到点链路上连续传定界符
- 地址域 (Address)
 - 多终端线路，用来区分终端
 - 点到点线路，有时用来区分命令和响应
 - 若帧中的地址是接收该帧的站的地址，则该帧是命令帧
 - 若帧中的地址是发送该帧的站的地址，则该帧是响应帧



HDLC帧结构（续）

■ 控制域 (Control)

- 序号：使用滑动窗口技术
- 确认
- 其它

■ 数据域 (Data)

- 任意信息，任意长度（上层协议SDU有上限）

■ 校验和 (Checksum)

- CRC校验
- 生成多项式：CRC-CCITT



HDLC帧结构（续）

■ 帧类型

- 信息帧 (Information)
- 监控帧 (Supervisory)
- 无序号帧 (Unnumbered)
 - 可以用来传控制信息，也可在无连接不可靠服务中传数据

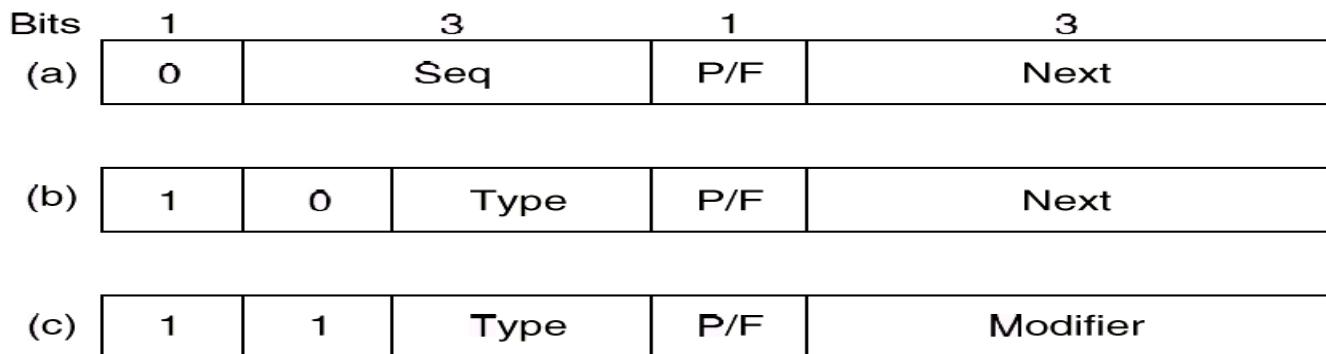


Fig. 3-25. Control field of (a) an information frame, (b) a supervisory frame, (c) an unnumbered frame.



HDLC帧结构—控制域

- **序号 (Seq)** : 使用滑动窗口技术，3位序号，发送窗口大小为7
- **确认 (Next)** : 指带第一个未收到的帧序号
- **探询/结束 P/F位 (Poll/Final)**
 - 命令帧置 “P”，响应帧置 “F”
 - 有些协议，P/F位用来强迫对方机器立刻发控制帧
 - 多终端系统中，计算机置 “P”，允许终端发送数据；终端发向计算机的帧中，最后一个帧置为 “F”，其它置为 “P”



HDLC帧结构—控制域（续）

■ 类型 (Type)

- “0” 表示确认帧 RR (RECEIVE READY)
- “1” 表示否定性确认帧 REJ (REJECT)
- “2” 表示接收未准备好 RNR (RECEIVE NOT READY)
- “3” 表示选择拒绝 SREJ (SELECTIVE REJECT)
 - HDLC和ADCCP允许选择拒绝，SDLC和LAPB不允许



HDLC命令

- **DISC (DISConnect)**
- **SNRM (Set Normal Response Mode)**
- **SARM (Set Asynchronous Response Mode)**
- **SABM (Set Asynchronous Balanced Mode)**
 - HDLC和LAPB使用
- **SABME(SABM的扩展)**
- **SNRME(SNRM的扩展)**
- **FRMR (FRaMe Reject)**
 - 校验和正确，语义错误
- **无序号确认UA (Unnumbered Acknowledgement)**
 - 对控制帧进行确认，用于确认模式建立和接受拆除命令
- **UI (Unnumbered Information)**



HDLC的功能组合

- 三种站，两种构型，三种操作模式，以及规程元素中定义的各种帧的各种组合产生多种链路层协议
- **构造链路层协议的过程**
 - 选择站构型 ——> 基本操作模式 ——> 基本帧种类 ——> 12种任选功能 ——> 得到协议



X. 25的链路层协议LAPB

- **X.25协议**
 - 分组级: PLP
 - 帧级: LAP (Link Access Procedure) , LAPB (Balanced)
 - 物理级: X.21
- **X.25协议规程使用HDLC规程的原理和术语**
- **X.25 LAP:** HDLC非平衡规程帧的基本清单 + 任选功能2、8、12，也可组成主站—次站式平衡规程
- **X.25 LAPB:** HDLC组合站平衡规程帧的基本清单 + 任选功能2、8、11、12
- **因此:** X.25 LAP、LAPB是HDLC的子集



LAPB的命令和响应

■ LAPB 的帧格式与 HDLC 完全相同

格式	命令	响应	控制域编码			
信息帧	I(信息)		0	N(S)	P	N(R)
监控帧	RR(接收准备好)	RR(接收准备好)	1	000	P/F	N(R)
	RNR(接收未准备好)	RNR(接收未准备好)	1	010	P/F	N(R)
	REJ(拒绝)	REJ(拒绝)	1	011	P/F	N(R)
无序号帧	SARM(置异步响应模式)	DM(拆除模式)	1	111	P/F	000
	SABM(置异步平衡模式)		1	111	P	100
	DISC(拆除)		1	100	P	010
		UA(无序号确认)	1	100	F	110
		CMDR(命令拒绝)	1	110	F	001
		FRMR(帧拒绝)				



LAPB的检错和纠错方法

- **帧格式**
 - 采用CRC校验，只检错，不纠错，丢弃出错帧
- **超时机制**
 - 计时器超时重传，重传N次，则向上层协议报告
 - 超时机制用来检错，重传用来纠错
- **帧序号**
 - 若接收方发现帧序号错，就发拒绝帧给发送方，发送方重传，既检错也纠错
- **采用P/F位来进行校验指示**
 - 发送置为 P 的命令帧，等待置为 F 的响应帧，能及时发现远程数据站是否收到命令帧
- **规程规定：**第一项必须使用；第二、三、四项组合使用



Internet数据链路层协议

■ 点到点通信的两种场景

- 路由器到路由器
- 通过modem拨号上网，连到路由器或接入服务器

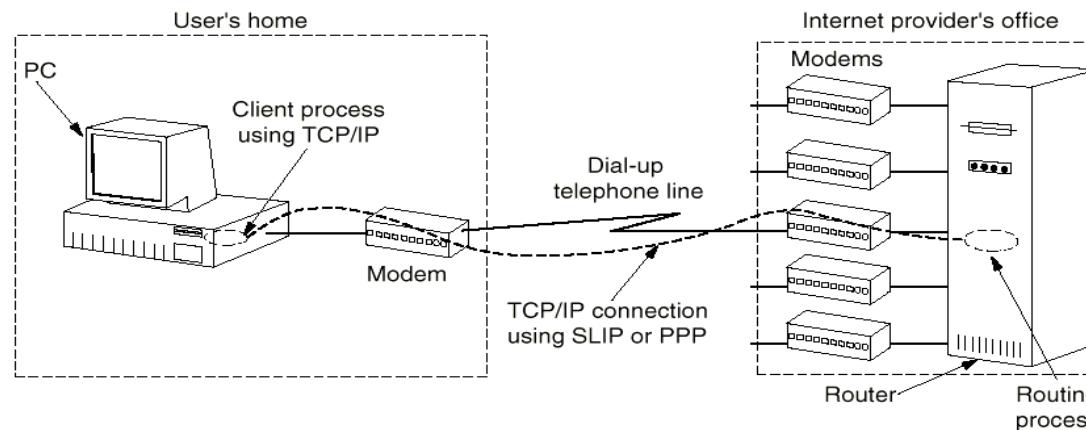


Fig. 3-26. A home personal computer acting as an Internet host.



SLIP: Serial Line IP

- 1984年，Rick Adams提出SLIP，RFC1055
- **基本思想：**发送原始IP包，用一个标记字节来定界，采用字符填充技术
- **新版本提供TCP和IP头压缩技术，RFC 1144**
- **存在的问题**
 - 不提供差错校验
 - 只支持IP
 - 不提供认证
 - 多种版本并存，互连困难



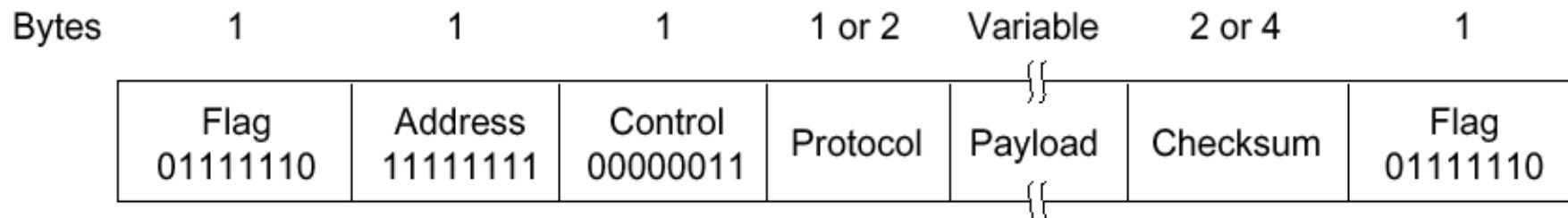
PPP: Point-to-Point Protocol

- RFC 1661, RFC 1662, RFC 1663
- PPP改进了SLIP，提供差错校验、支持多种协议、允许动态分配IP地址、支持认证等
- **以帧为单位发送，而不是原始IP包**
- **协议包括两部分**
 - 链路控制协议LCP (Link Control Protocol)
 - 可使用多种物理层服务：modem, HDLC串线, SDH/SONET等
 - 网络控制协议NCP (Network Control Protocol)
 - 可支持多种网络层协议



PPP帧格式

- 帧格式与HDLC相似，区别在于PPP是面向字符的，采用字符填充技术



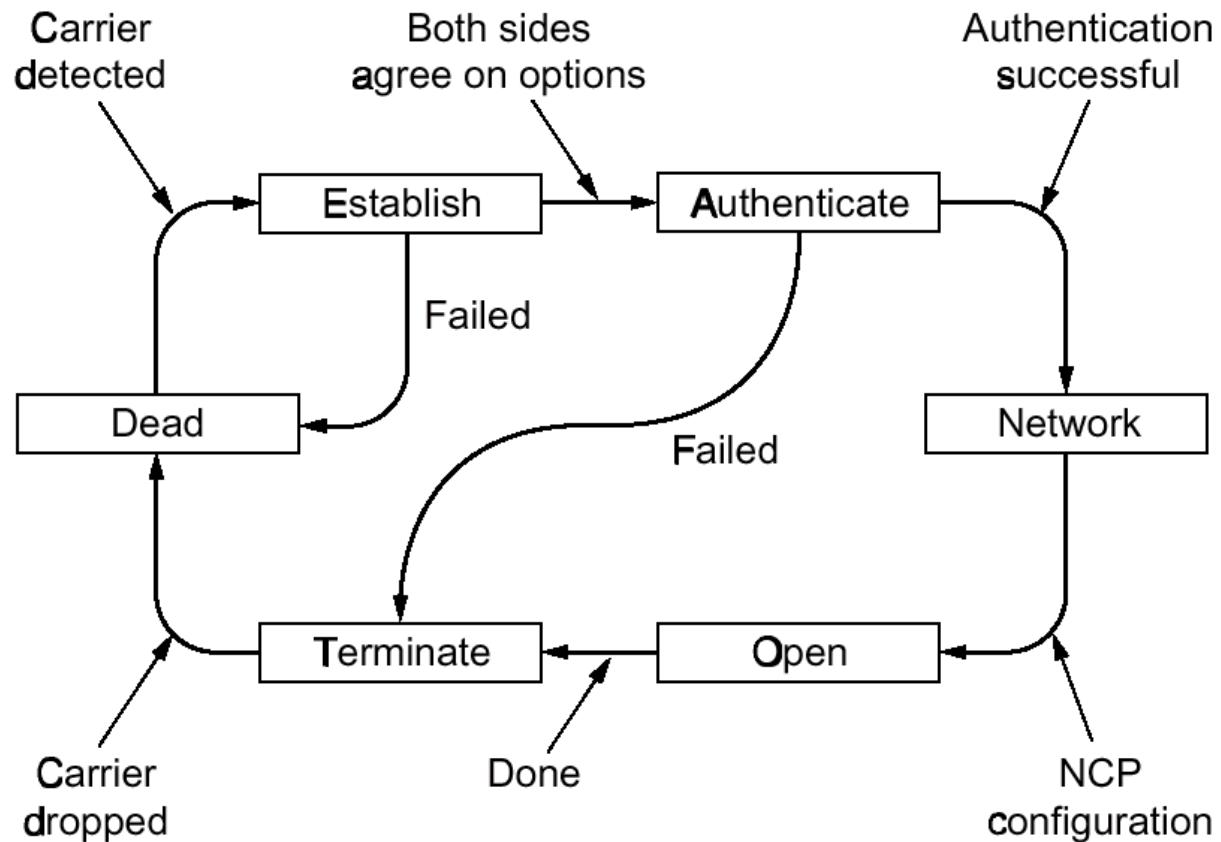


PPP帧格式（续）

- **标记域:** 01111110, 字符填充
- **地址域:** 11111111
- **控制域**
 - 缺省值为00000011, 表示无序号帧, 不提供使用序号和确认的可靠传输
 - 不可靠线路上, 也可使用有序号的可靠传输
- **协议域:** 指示净负荷中包的类型, 缺省为2字节
- **净负荷域:** 变长, 缺省为1500字节
- **校验和域:** 2或4个字节



PPP链路 up / down 过程



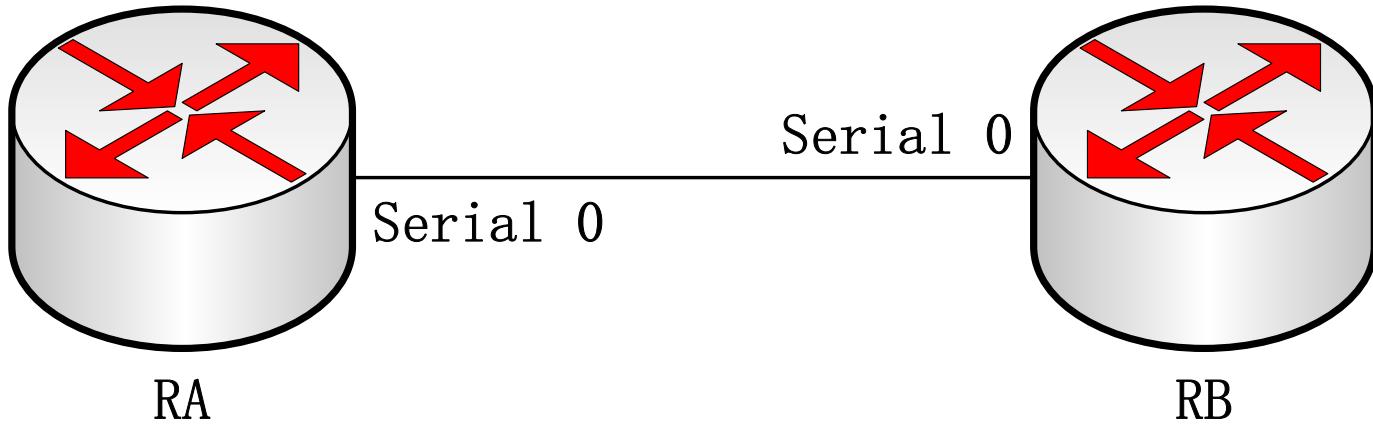


小结

- 三种数据链路层协议：**HDLC、LAPB（面向比特）和PPP（面向字符）**
- HDLC具有**三种站，两种构型，三种操作模式**
- X.25 LAPB是HDLC的子集
- PPP
 - 具有多协议成帧机制
 - 可以在modem, HDLC bit-serial lines, SDH/SONET等物理层上运行
 - 支持差错检测、选项协商、包头压缩、动态分配IP地址和认证等
 - 包括两部分协议：LCP和NCP
 - 通常不使用滑动窗口技术，但是也具有利用HDLC帧进行可靠传输的可选功能



在路由器上简单配置HDLC

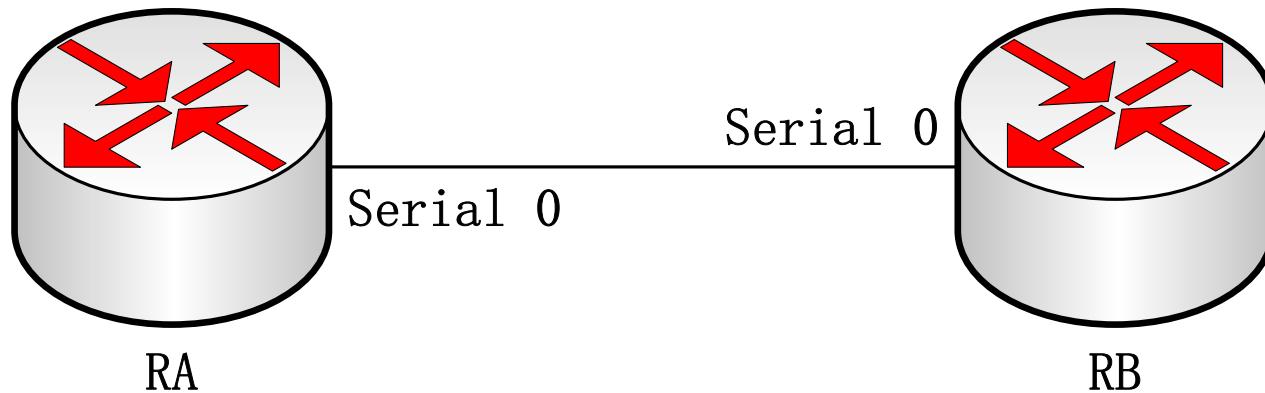


```
interface serial 0  
encapsulation hdlc
```

```
interface serial 0  
encapsulation hdlc
```



在路由器上简单配置LAPB



```
interface serial 0  
encapsulation lapb dce
```

```
interface serial 0  
encapsulation lapb dte
```



在路由器上简单配置PPP



```
interface serial 0  
encapsulation ppp  
ppp authentication chap
```

```
interface serial 0  
encapsulation ppp  
ppp authentication chap
```