







第3章 数据链路层

- ❖ 定义，功能 
- ❖ 数据帧的组成 
- ❖ 可靠性传输 
- ❖ 数据链路层示例 



数据链路层的定义

网络层有个实体即进程，数据链路层将借助于物理层为网络层提供服务

- ❖ 为网络层提供一个较好定义的服务接口
- ❖ 定义一个合适的传输差错率
- ❖ 对传输的数据流进行管理，以免快速的发送淹没慢速的接收端

数据链路层的协议数据单元(**PDU**)是帧



数据链路层的功能

数据链路层的任务是把网络层的数据组合成帧，加上一定的校验，然后交物理层用某种信号表示二进制数据位送到目的计算机，并通过目的计算机的物理层和数据链路层送到网络层，也就是为网络层提供一条可靠的数据链路



相连、物理链路和数据链路

- ❖ 所谓相连，可以理解为物理的连接，但当采用多路复用技术时也可以是信道连接。其特征是传输的数据是顺序的
- ❖ 物理链路：一段无源的点到点的物理连接，中间没有任何交换节点
- ❖ 数据链路：一条物理连接加上为实现数据的传输，两端配置的硬件和相关的通信协议



数据链路层服务的区分规则

❖ 数据链路层的服务是通过有无连接、有无确认来区分的

- 无确认无连接
- 有确认无连接
- 有确认有连接



确认和连接

- ❖ 确认：接收方在收到数据帧后，必须给发送方发回一个确认
- ❖ 面向连接：发送方和接收方在传输数据之前必须建立逻辑连接，传输结束后必须释放连接



数据链路层的服务

❖ 无确认无连接



❖ 有确认无连接



❖ 有确认有连接





无确认的面向无连接服务

- ❖ 无确认是指接收方在收到数据帧后，毋需发回一个确认
- ❖ 无连接服务是指在数据传输前毋需建立逻辑链路
- ❖ 物理线路的连接并非意味着提供有连接的服务
- ❖ 无确认并非不可靠，其可靠性由上层负责

例如：局域网 共享信道毋需建立连接

信道较为理想，数据传输的误码率很低
即使出错或丢失由上层负责恢复



数据链路层的服务

❖ 无确认无连接



❖ 有确认无连接



❖ 有确认有连接





有确认的面向无连接服务

- ❖ 使用前不建立连接，即不建立数据链路，但每帧传输必须得到确认
- ❖ 这在信号传播延时较大、线路状态不一定很可靠的情况下是有效的

例如：无线通信 如建立连接，则信道使用率很低
但数据传输的误码率相对较高，
确认是必要的



数据链路层的服务

❖ 无确认无连接



❖ 有确认无连接



❖ 有确认有连接





有确认的面向连接服务

- ❖ 使用前先建立连接，即先建立数据链路，并且每帧的传输必须得到确认
- ❖ 有连接的服务必须在使用前先建立连接（即建立逻辑链路），然后使用，最后释放连接

例如：电话







数据的可靠传输

保证直接相连的两台主机的可靠性传输

- ❖ 将传输的信息组合成帧
- ❖ 校验和重发
- ❖ 流量控制







第3章 数据链路层

- ❖ 定义，功能 
- ❖ 数据帧的组成 
- ❖ 可靠性传输 
- ❖ 数据链路层示例 



数据帧的组成

帧的组成必须保证能识别一个完整的帧，并保证一旦出现传输差错导致前一个帧丢失，也必须能识别下一个帧，即再同步

- ❖ 字符计数法 
- ❖ 带字符填充的首尾界符法 
- ❖ 带位填充的首尾标志法 
- ❖ 物理层编码违例法 



字符计数法

❖ 帧的长度用一个字节表示，作为帧的头部







Tnbm P188 Fig. 3-4 字符计数成帧法

一旦帧长度计数被误读，将无法再同步，所以很少采用



数据帧的组成

帧的组成必须保证能识别一个完整的帧，并保证一旦出现传输差错导致前一个帧丢失，也必须能识别下一个帧，即再同步

- ❖ 字符计数法 
- ❖ 带字符填充的首尾界符法 
- ❖ 带位填充的首尾标志法 
- ❖ 物理层编码违例法 



带字符填充的首尾界符法

❖ 用特殊的字符作为帧头和帧尾



Tnbnm P189 Fig. 3-5 (a) 由Flag标志的一个帧

这是一种面向字符的帧格式，所传输的数据都是字符（**ASCII**或**EBCDIC**字符），但帧中不允许出现帧界符标志，在面向字符的串型通信中常使用这种格式(**PPP**)

接收方一旦丢失了一个**FLAG**，只要继续搜索下一个**FLAG**，就可重新确定帧边界，即具有再同步能力



面向字符的帧格式

- ❖ 面向字符的帧格式不适宜传输数据中包含二进制数的帧，因为在包含二进制数的帧中很可能出现**FLAG**的字符（通常**FLAG**用**ASCII**字符**7EH**定义）
- ❖ 一种方法是在二进制数中偶然出现的**FLAG**前再插入一个**ESC**（**ASCII**字符**1BH**），这就称为字符填充法

41 33 7E 9C 4B 0C	41 33 1B 7E 9C 4B 0C
41 33 1B 9C 4B 0C	41 33 1B 1B 9C 4B 0C
41 33 1B 7E 9C 4B 0C	41 33 1B 1B 1B 7E 9C 4B 0C
41 33 1B 1B 9C 4B 0C	41 33 1B 1B 1B 1B 9C 4B 0C



数据帧的组成

帧的组成必须保证能识别一个完整的帧，并保证一旦出现传输差错导致前一个帧丢失，也必须能识别下一个帧，即再同步

- ❖ 字符计数法
- ❖ 带字符填充的首尾界符法
- ❖ 带位填充的首尾标志法
- ❖ 物理层编码违例法





带位填充的首尾标志法

字符并非都是8位的，东方文字是16位的，**UNICODE**是16位的

❖ 在面向二进制位的同步串型通信中常使用带位填充的首尾标志格式，如**HDLC**

这是一种面向二进制位的帧格式，把所有需传输的数据（不论是字符或表示一个浮点数的二进制位串，还是一个**MP3**的文件）一字排开，并以特殊的位模式**01111110**作为帧标志，即一个帧的开始（同时标志前一个帧的结束）

如果由于干扰，一个帧标志没有正确接收，则继续扫描接收串，一旦扫描到**01111110**，即新的一帧从此开始，即具有再同步能力



面向bit的帧格式

- ❖ 当帧中出现一个与帧标志相同的位串**01111110**，则在**5个1**后插入一个**0**，即变成**01111101**，接收方将自动删除第**5个1**后的**0**
- ❖ 这称为位插入法，也称为透明传输

0110111111111111111110010
011011111 0 11111 0 11111 0 10010
01101111111111111111111110010

Tnbm P190 Fig. 3-6 (a) (b) (c) 位插入法示例



数据帧的组成

帧的组成必须保证能识别一个完整的帧，并保证一旦出现传输差错导致前一个帧丢失，也必须能识别下一个帧，即再同步

- ❖ 字符计数法
- ❖ 带字符填充的首尾界符法
- ❖ 带位填充的首尾标志法
- ❖ 物理层编码违例法





物理层编码违例法

- ❖ 在曼切斯特编码中，连续高电平或连续低电平可用作帧边界

采用冗余编码技术，如曼切斯特编码，即两个采样脉冲才可得到一个二进制位





数据0：低-高电平对 

数据1：高-低电平对 

高-高电平对和低-低电平对没有使用，如在二进制编码中出现称为编码违例，但这两种违例编码可用作帧边界，在令牌环网中使用编码违例格式



第3章 数据链路层

- ❖ 定义，功能 
- ❖ 数据帧的组成 
- ❖ 可靠性传输 
- ❖ 数据链路层示例 



可靠性传输

❖ 差错控制：校验、重发和序号

避免帧错误的保证：帧的校验

避免帧丢失的保证：超时和重发

避免帧重复的保证：帧有序号

❖ 流量控制：窗口协议

发送方和接收方之间的协调

❖ 协议描述和验证



差错控制

如何保证数据帧的正常传输：通过三种手段处理三种可能出现的情况：

❖ 确认



❖ 数据帧丢失



❖ 重复帧





确认

接收方在收到了正确的帧后向发送方发肯定性确认。如收到的帧有问题，则发否定性确认，此时发送方将重发此帧

确认的前提是必须经过差错检测



差错检测和校正

- ❖ 超时只是传输出错的一种情况，更多的传输出错是数据中的一位或几位因噪声干扰而出错
- ❖ 噪声分两种：
 - 信道所固有的、持续存在的随机热噪声
 - 外界突发原因而造成的冲击噪声
- ❖ 通常接收方能检错，甚至纠错
- ❖ 纠错码是除 **m** 个数据位外增加 **r** 个冗余位作为纠错位，整个长度为 **$n = m + r$**



纠错码和检错码

❖ 纠错码:

- 海明 (Hamming) 码



❖ 检错码:

- 校验和 (Check Sum)
- 块校验码 (Block Check Code)
- 循环冗余检错码 **CRC**



(Cyclic Redundancy Check)



海明 (Hamming) 纠错码

m 个数据位外加 r 个纠错位。在 2^i ($i=0,1,2,3\dots$) 的位置放的是纠错位。 m 个数据位的次序不变。如字符 m 的 7 位二进制码为 **1101101**，要加上 4 位纠错码 **0011** (偶校验)，共 **11** 个 bit。

11	10	9	8	7	6	5	4	3	2	1
A7	A6	A5	P4	A4	A3	A2	P3	A1	P2	P1
1	1	0	0	1	1	0	0	1	1	1



信息位影响的纠错位

一个信息位影响多个纠错位

信息位	信息组位	组位展开	影响的纠错位
A7	11	$=8+2+1$	对P4、P2、P1有影响
A6	10	$=8+2$	对P4、P2有影响
A5	9	$=8+1$	对P4、P1有影响
A4	7	$=4+2+1$	对P3、P2、P1有影响
A3	6	$=4+2$	对P3、P2有影响
A2	5	$=4+1$	对P3、P1有影响
A1	3	$=2+1$	对P2、P1有影响



纠错位的取值

一个纠错位由多个信息异或然后取偶/奇校验

纠错位	纠错位的取值（再取偶/奇校验）
P4	$=A7 \oplus A6 \oplus A5$
P3	$=A4 \oplus A3 \oplus A2$
P2	$=A7 \oplus A6 \oplus A4 \oplus A3 \oplus A1$
P1	$=A7 \oplus A5 \oplus A4 \oplus A2 \oplus A1$



海明码纠错位取值举例

❖ 采用偶校验

检错位	纠错位的取值（再取偶校验）
P4	$=A7 \oplus A6 \oplus A5 = 0$
P3	$=A4 \oplus A3 \oplus A2 = 0$
P2	$=A7 \oplus A6 \oplus A4 \oplus A3 \oplus A1 = 1$
P1	$=A7 \oplus A5 \oplus A4 \oplus A2 \oplus A1 = 1$

11	10	9	8	7	6	5	4	3	2	1
A7	A6	A5	P4	A4	A3	A2	P3	A1	P2	P1
1	1	0	0	1	1	0	0	1	1	1



接收方的校验

校验位	校验位计算表达式
S4	$=A7 \oplus A6 \oplus A5 \oplus P4$
S3	$=A4 \oplus A3 \oplus A2 \oplus P3$
S2	$=A7 \oplus A6 \oplus A4 \oplus A3 \oplus A1 \oplus P2$
S1	$=A7 \oplus A5 \oplus A4 \oplus A2 \oplus A1 \oplus P1$

计算结果**S4 S3 S2 S1**为**0 0 0 0**意味着接收正确

计算结果**S4 S3 S2 S1**为**0 1 0 1**意味着第**5**位出错



接收方的校验举例 (无错)

字符m采用海明码的发送序列为**11001100111**

如接收到的序列也为 **11001100111**

11	10	9	8	7	6	5	4	3	2	1
A7	A6	A5	P4	A4	A3	A2	P3	A1	P2	P1
1	1	0	0	1	1	0	0	1	1	1

校验位	校验位计算表达式的值	
S4	$=A7 \oplus A6 \oplus A5 \oplus P4 = 1 \oplus 1 \oplus 0 \oplus 0$	=0
S3	$=A4 \oplus A3 \oplus A2 \oplus P3 = 1 \oplus 1 \oplus 0 \oplus 0$	=0
S2	$=A7 \oplus A6 \oplus A4 \oplus A3 \oplus A1 \oplus P2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1$	=0
S1	$=A7 \oplus A5 \oplus A4 \oplus A2 \oplus A1 \oplus P1 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1$	=0

S4 S3 S2 S1 = 0 0 0 0 意味校验正确



接收方的校验举例 (有错)

字符m采用海明码的发送序列为**11001100111**

如接收到的序列为 **110011**1**0111**

11	10	9	8	7	6	5	4	3	2	1
A7	A6	A5	P4	A4	A3	A2	P3	A1	P2	P1
1	1	0	0	1	1	1	0	1	1	1

校验位	校验位计算表达式的值	
S4	$=A7 \oplus A6 \oplus A5 \oplus P4 = 1 \oplus 1 \oplus 0 \oplus 0$	=0
S3	$=A4 \oplus A3 \oplus A2 \oplus P3 = 1 \oplus 1 \oplus 1 \oplus 0$	=1
S2	$=A7 \oplus A6 \oplus A4 \oplus A3 \oplus A1 \oplus P2 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1$	=0
S1	$=A7 \oplus A5 \oplus A4 \oplus A2 \oplus A1 \oplus P1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1$	=1

S4 S3 S2 S1 = 0 1 0 1 意味第5位出错



纠错码和检错码

❖ 纠错码:

- 海明 (Hamming) 码



❖ 检错码:

- 校验和 (Check Sum)
- 块校验码 (Block Check Code)
- 循环冗余检错码 **CRC**



(Cyclic Redundancy Check)



校验和

❖ 算法简单、实现容易，但检错率不高

将发送的数据看成是二进制整数序列，并划分成一段段规定的长度（如**8位**、**16位**、**32位**等），计算他们的和。如校验和大于规定的长度，则将进位加到最后的校验和中。将校验和和数据一起发送。在接收端，重新计算校验和，与原校验和比较。如要传输“**Hello world.**”

H	e	l	l	o	␣	w	o	r	l	d	.		
48	65	6C	6C	6F	20	77	6F	72	6C	64	2E	71	FC

以**16位**为例：

4865H+6C6CH+6F20H+776FH+726CH+642EH+进位=71FCH



纠错码和检错码

❖ 纠错码:

- 海明 (Hamming) 码



❖ 检错码:

- 校验和 (Check Sum)
- 块校验码 (Block Check Code)
- 循环冗余检错码 **CRC**



(Cyclic Redundancy Check)



块校验码

- ❖ 块校验码**BCC** (**B**lock **C**heck **C**ode) 简单常用，但检错的强度较弱，如在同一列上有偶数位错，则不能检测

如传输的数据都是**ASCII**字符（即面向字符，这在应用中很多），每个字符进行奇偶校验，然后把所有的字符（连同奇偶位）进行异或运算，运算结果即为其块校验码。发送端一面发送一面计算，最后发送块校验码，接收端一面接收一面计算，最后与接收到的块校验码比较

如“ **Hello world.**”，采用偶校验，校验后的字符序列为：

H	e	l	l	o	␣	w	o	r	l	d	.	
48	65	6C	6C	6F	A0	77	6F	72	6C	E4	2E	2E

$$\begin{aligned} &48H \oplus 65H \oplus 6CH \oplus 6CH \oplus 6FH \oplus A0H \oplus 77H \oplus 6FH \oplus 72H \\ &\oplus 6CH \oplus E4H \oplus 2EH = 2EH \end{aligned}$$



纠错码和检错码

❖ 纠错码:

- 海明 (Hamming) 码



❖ 检错码:

- 校验和 (Check Sum)
- 块校验码 (Block Check Code)
- 循环冗余检错码 **CRC**



(Cyclic Redundancy Check)



循环冗余检错码 CRC

- ❖ 任何一个**k**位的帧看成为一个**k-1**次的多项式 **$M(x)$**
如: **1011001**看成 **$x^6+x^4+x^3+x^0$**
- ❖ 设定一个多项式编码生成多项式 **$G(x)$** , **$G(x)$** 为**r**阶, **$k > r$**
- ❖ 如 **$x^r M(x)/G(x) = Q(x) + R(x)/G(x)$** 其中 **$Q(x)$** 为商、 **$R(x)$** 为余数, **$R(x)$** 即为 **$M(x)$** 的**CRC**码
- ❖ 将**CRC**码接在帧后一起发送, 即发送数据为 **$x^r M(x) + R(x)$**
- ❖ 二进制运算中, 减法和加法都做异或运算: **$0+1=1, 1+1=0$**
- ❖ 因为 **$(x^r M(x) - R(x))$** 一定能被 **$G(x)$** 整除, 即余数为**0**, 则接收方只要计算的余数为**0**即为正确



CRC码计算举例

如一帧为1101011011

即: $M(x) = x^9 + x^8 + x^6 + x^4 + x^3 + x + 1$

$$G(x) = x^4 + x + 1$$

$$\begin{aligned} T(x) &= x^4 M(x) \\ &= x^4 (x^9 + x^8 + x^6 + x^4 + x^3 + x + 1) \\ &= x^{13} + x^{12} + x^{10} + x^8 + x^7 + x^5 + x^4 \end{aligned}$$

CRC码计算举例 (续1)

❖ 帧: 1101011011

❖ 除数: 10011

❖ 实际传输帧:

1101011011 1110

帧数据

余数

余数

1 1 1 0

Tnbm P198 Fig. 3-8 CRC码计算举例



CRC码计算举例 (续2)

11010110110000/10011

= 1100001010 1110

即**11010110110000 + 1110**能被**10011**整除

(模2运算的加、减和异或，其运算结果相同)

如发送方发送的**T(x)**，接收方收到的是

T(x)+E(x)，除非**E(x)**是**G(x)**的整倍数，否则不能被整除，即都能被检测到已出错



三个生成多项式国际标准

❖ **CRC-12:** $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$

用于字符长度为**6**位

❖ **CRC-16 :** $x^{16} + x^{15} + x^2 + 1$

用于字符长度为**8**位

❖ **CRC-CCITT :** $x^{16} + x^{12} + x^5 + 1$

用于字符长度为**8**位



差错控制

如何保证数据帧的正常传输：通过三种手段处理三种可能出现的情况：

❖ 确认



❖ 数据帧丢失



❖ 重复帧





数据帧丢失

- ❖ 通过发送方计时器（超时）解决
- ❖ 超时（**TimeOut**）：在传输过程中，所发送的帧丢失，接收方根本没有收到，不可能发送确认帧（包括否定性确认），而发送方正在等待接收方的确认帧，当然也不可能等到接收方的回音。所以，发送方一旦发送一帧，就启动一计时器，在规定的时间内，一般都应收到回音，如收不到回音，则在计时器溢出时，再重发此帧



差错控制

如何保证数据帧的正常传输：通过三种手段处理三种可能出现的情况：

❖ 确认



❖ 数据帧丢失



❖ 重复帧





重复帧

- ❖ 重发机制也包括当接收方发送的确认帧丢失而导致发送方的重发
- ❖ 由于接收方确认帧的丢失，导致发送方多次发送同一帧，接收方也将多次收到同一帧，要能区别是否是同一帧，则每帧都应有一个帧的编号



可靠性传输

❖ 差错控制：校验、重发和序号

避免帧错误的保证：帧的校验

避免帧丢失的保证：超时和重发

避免帧重复的保证：帧有序号

❖ 流量控制：窗口协议

发送方和接收方之间的协调

❖ 协议描述和验证



流量控制

❖ 发送方和接收方速率的匹配即流量控制

如接收方的处理能力低于发送方，即使传输中没有出错，也可能被“淹没”，所以通常在接收方的缓冲区到达一定量时，应及时通知发送方，暂停发送，等候通知，这就是流量控制机制

➤ 基本数据链路协议



➤ 滑动窗口协议





基本数据链路协议

可以这样理解：在一台主机中，物理层、数据链路层、网络层等等都有各自的进程在运行，并且假设：

A、B两台主机要求可靠的、面向连接的通信，在接收方的数据链路层，目前正运行的是**wait_for_event(&event)**，即等待某个事件发生



wait_for_event(&event)的参数

如发生了某个事件，此过程将返回参数，**event**有两个取值：

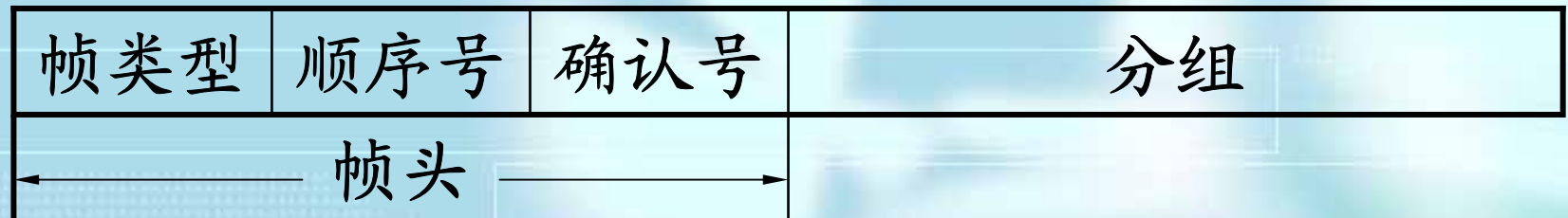
checksum_err和**frame_arrival**

- ❖ 如**event = checksum_err**，意即所接收的帧的校验和错，应考虑不发送确认帧，或发送否定性确认帧
- ❖ 如**event = frame_arrival**，即帧已到达并校验正确，则调用**from_physical_layer**，从物理层取得帧，检查帧头的控制信息，如一切正常，则仅把其中的分组交网络层



帧的基本格式

一般意义上的帧的格式



一系列过程和数据结构的定义:

Tnbnm P202 Fig. 3-9



协议1：一种无限制的单工协议

一种理想的环境，理想的协议，假定：

- ❖ 链路是理想的传输通道，所传输的任何数据既不会出错也不会丢失

即：不需校验，也不可能重发

- ❖ 不管发送方以怎样的速率发送数据，接收方都能及时接收

即接收端处理器的处理速度无限高，处理时间可忽略不计，缓冲区空间无限大，毋需流量控制

Tnbnm P205 Fig. 3-10 一种无限制的单工协议



协议1: SENDER

```
void sender1(void)
{
    frame s;
    packet buffer;

    while (true)
    {
        from_network_layer(&buffer);
        s.info=buffer;
        to_physical_layer(&s);
    }
}
```



协议1: RECEIVER

```
void receiver1(void)
{
    frame r;
    event_type event;

    while (true)
    {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
    }
}
```



协议2：一个单工的停 – 等协议

然而，接收方不可能具有足够高的**CPU**处理能力来及时处理所有的接收帧，也不可能具有足够大的缓冲区，但仍假定：

- ❖ 链路是理想的传输通道，所传输的任何数据既不会出错也不会丢失

即：不需校验，也不可能重发

所以，当它来不及处理时，应通知发送方暂缓发送，一旦可以继续接收，则通知发送方继续发送

Tnbn P207 Fig. 3-11 一个单工的停 – 等协议



协议2: SENDER

```
void sender2(void)
{ frame s;
  packet buffer;
  event_type event;

  while (true)
  {
    from_network_layer(&buffer);
    s.info=buffer;
    to_physical_layer(&s);
    wait_for_event(&event);
  }
}
```



协议2: RECEIVER

```
void receiver2(void)
{
    frame r,s;
    event_type event;

    while (true)
    {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
    }
}
```



协议3：有噪声信道的单工协议

在噪声信道中应考虑传输有差错的情况

所谓差错：

- ❖ 帧的损坏：如帧中若干位出错，通常**CRC**都能检测到
 - ❖ 帧的丢失：一旦帧头出错，接收方将检测下一个帧头，以同步，但该帧已丢失
 - ❖ 帧的重复：如确认帧丢失，接收方将接收到重复帧
- 所以，发送方应启动一个计时器，一旦超时立即重发

Tnbnm P210 Fig. 3-12 一个肯定性确认和超时重发协议



协议3的要点

- ❖ 发送方要记录下一个准备发送的序号
- ❖ 接收方要记录下一个期望接收的序号
- ❖ 发送过程和接收过程是严格交替的
- ❖ 接收方收到一个正确的帧，即便不是期望的帧，都将发送一个确认（即一个空的确认帧）

也称为**ARQ**协议：

Automatic Repeat reQuest



协议3: SENDER

```
void sender3(void)
{ next_frame_to_send=0;
  from_network_layer(&buffer);
  while (true)
  { s.info=buffer;
    s.seq=next_frame_to_send;
    to_physical_layer(&s);
    start_time(s.seq)
    wait_for_event(&event);
    if (event==frame_arrival)
    {from_physical_layer(&s);
     if (s.ack==next_frame_to_send) {
       stop_timer(s.ack);
       from_network_layer(&buffer);
       inc(next_frame_to_send);
     }
    }
  }
}
```



协议3: RECEIVER

```
void receiver3(void)
{ frame_expected=0;
  while (true)
  { wait_for_event(&event);
    if (event==frame_arrival)
    { from_physical_layer(&r);
      if (r.seq==frame_expected)
      { to_network_layer(&r.info);
        inc(frame_expected); }
      s.ack=1-frame_expected;
      to_physical_layer(&s);
    }
  }
}
```



协议3的重发机制存在的问题

❖ 效率较低

如接收方收到的帧出错或者整个数据帧丢失，则不发确认帧，发送方在超时后重发，直至正确（效率极低）

❖ 接收方会收到重复帧

如接收方收到了正确的数据帧并发送了确认，但此确认丢失，发送方在超时后重发此帧，这样，接收方的数据链路层收到了两个完全相同的帧，其网络层将收到两个完全相同的分组（这是不能允许的）



流量控制

❖ 发送方和接收方速率的匹配即流量控制

如接收方的处理能力低于发送方，即使传输中没有出错，也可能被“淹没”，所以通常在接收方的缓冲区到达一定量时，应及时通知发送方，暂停发送，等候通知，这就是流量控制机制

➤ 基本数据链路协议



➤ 滑动窗口协议





协议3的问题

- ❖ 不能双向传输，效率低
- ❖ 由于应答无序号，导致系统不可靠

其实，在协议**3**中，发送方应重发当前帧还是发下一帧，仅根据超时还是收到确认而定，亦即仅与前一帧和后一帧有关，所以，在协议**3**中，顺序号仅需一位，**0/1**



双向传输解决方案

- ❖ 用四根信道：两根数据，两根应答
但信道利用率很低
- ❖ 用两根信道：一根**A到B**，另一根**B到A**
问题是如何区分数据和应答
解决方法有两种：
 - 用不同的帧类型
 - 用捎带确认



滑动窗口协议

❖ 收发使用两条信道

发送方可连续发送多帧，接收方接受到一帧后就从另一个信道发一个确认，为提高信道使用效率，可使用捎带确认

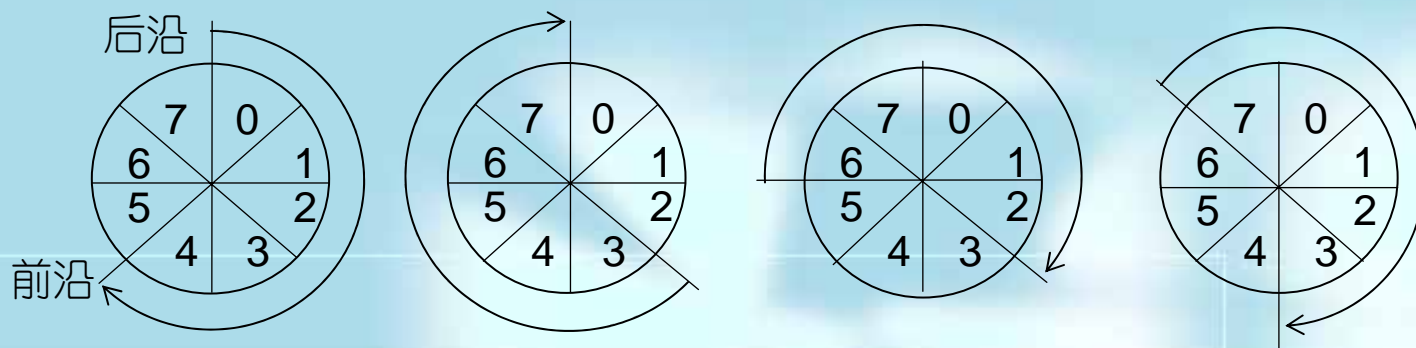
❖ 帧是有序号的

即使过早超时而导致的重发也可根据帧的序号来避免帧的重复

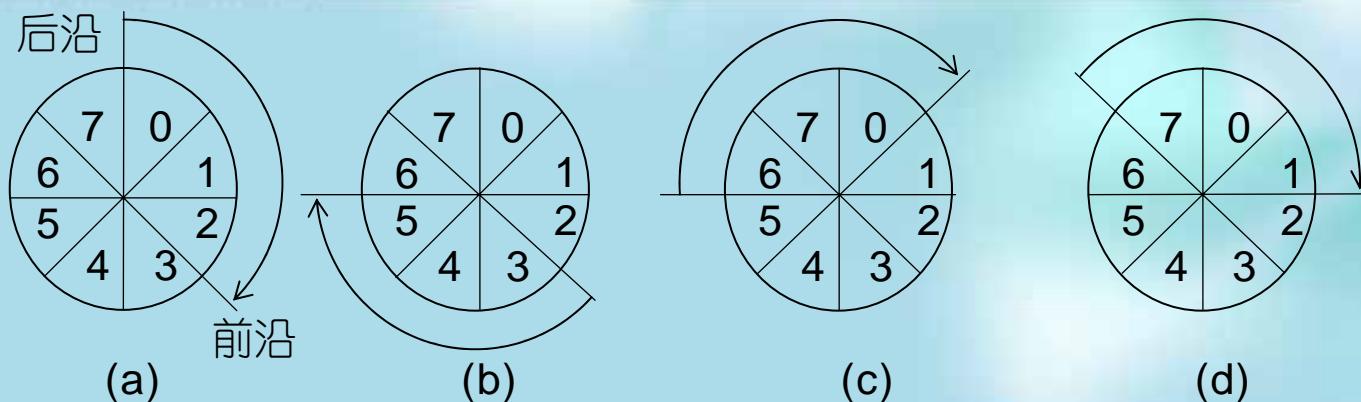


滑动窗口原理 (设 $W_T=5$, $W_R=3$)

❖ 发送方



❖ 接收方





三个窗口协议（协议4-6）

❖ 发送窗口 $W_T=1$ ，接收窗口 $W_R=1$ 

Tnbnm P215 Fig. 3-14 一位滑动窗口协议（协议4）

❖ 发送窗口 $W_T=7$ ，接收窗口 $W_R=1$ 

Tnbnm P220 Fig. 3-17 后退n帧的滑动窗口协议（协议5）

❖ 发送窗口 $W_T=4$ ，接收窗口 $W_R=4$ 

Tnbnm P224 Fig. 3-19 选择性重发滑动窗口协议（协议6）



一位滑动窗口协议

❖ 发送窗口 $W_T=1$ ，接收窗口 $W_R=1$

❖ 即停-等协议

Tnbm P215 Fig. 3-14 (协议4)



一位滑动窗口协议

```
void protocol4(void)
{ next_frame_to_send=0;
  frame_expected=0;
  from_network_layer(&buffer);
  s.info=buffer;
  s.seq=next_frame_to_send;
  s.ack=1-frame_expected;
  to_physical_layer(&s);
  start_time(s.seq) ;
```



```
while (true)
{ wait_for_event(&event);
  if (event==frame_arrival)
  { from_physical_layer(&r);
    if (r.seq==frame_expected)
    { to_network_layer(&r.info);
      inc(frame_expected); }
    if (r.ack==next_frame_to_send)
    { stop_timer(r.ack);
      from_network_layer(&buffer);
      inc(next_frame_to_send); }
    }
  s.info=buffer;
  s.seq=next_frame_to_send;
  s.ack=1-frame_expected
  to_physical_layer(&s);
  start_time(s.seq)
}
```



比较两种情况

❖ 情况1:

A端首先发送，**B**端等待发送，无过早超时，其运行过程正常

❖ 情况2:

A端和**B**端都同时发送，无过早超时，其运行过程不正常



情况1：主机A发送主机B等待

主机A

- ❖ 初始化后组成发送帧，其中：
 - 发送帧顺序号**seq=0**
 - 对接收到的帧的确认**ack=1**
 - 实际上此确认无意义
 - 通过物理层发送

主机B

- ❖ 初始化后尚未组成发送帧，在接收到**A0**帧后交网络层并组成发送帧，（捎带确认）其中：
 - 发送帧顺序号**seq=0**
 - 对接收到的帧的确认**ack=0**
 - 通知**A**，**0**帧收到



情况1：运行过程正常

A发送 (0, 1, A0)	Seq ack	A发送A0, seq = 0
	B收到 (0, 1, A0) ▲	B收到A0
	B发送 (0, 0, B0)	B发送B0, 并确认A0
A收到 (0, 0, B0) ▲		A收到B0及对A0的确认
A发送 (1, 0, A1)		A发送A1及对B0的确认
	B收到 (1, 0, A1) ▲	B收到A1及对B0的确认
	B发送 (1, 1, B1)	B发送B1及对A1的确认
A收到 (1, 1, B1) ▲		A收到B1及对A1的确认
A发送 (0, 1, A2)		A发送A2及对B1的确认
	B收到 (0, 1, A2) ▲	B收到A2及对B1的确认
	B发送 (0, 0, B2)	B发送B2及对A2的确认
A收到 (0, 0, B2) ▲		A收到B2及对A2的确认
A发送 (1, 0, A3)		A发送A3及对B2的确认
	B收到 (1, 0, A3) ▲	B收到A3及对B2的确认
	B发送 (1, 1, B3)	

Tnbnm P216 Fig. 3-15 (a) 协议4，运行过程正常



比较两种情况

❖ 情况1:

A端首先发送，**B**端等待发送，无过早超时，其运行过程正常

❖ 情况2:

A端和**B**端都同时发送，无过早超时，其运行过程不正常



情况2：主机A发送主机B也发送

主机A

- ❖ 初始化后组成发送帧，其中：
 - 发送帧顺序号**seq=0**
 - 对接收到的帧的确认**ack=1**
 - 实际上此确认无意义
 - 通过物理层发送

主机B

- ❖ 初始化后也组成发送帧，其中：
 - 发送帧顺序号**seq=0**
 - 对接收到的帧的确认**ack=1**
 - 实际上此确认无意义
 - 通过物理层发送



情况2：运行过程不正常

A发送 (0, 1, A0)	B发送 (0, 1, B0)	B也已从网络层得到B0
	B收到 (0, 1, A0) ▲	B以为B0分组A没有收到
	B发送 (0, 0, B0)	所以B又重发B0分组
A收到 (0, 1, B0) ▲		A以为A0分组B没有收到
A发送 (0, 0, A0)		所以A又重发A0分组
	B收到 (0, 0, A0) ►	B收到对B0的确认并丢弃A0
	B发送 (1, 0, B1)	B发送B1及对A0的确认
A收到 (0, 0, B0) ►		A收到对A0的确认并丢弃B0
A发送 (1, 0, A1)		A发送A1及对B0的确认
	B收到 (1, 0, A1) ▲	B以为B1分组A没有收到
	B发送 (1, 1, B1)	所以B又重发B1分组
A收到 (1, 0, B1) ▲		A以为A1分组B没有收到
A发送 (1, 1, A1)		所以A又重发A1分组
	B收到 (1, 1, A1) ►	B收到对B1的确认并丢弃A1
	B发送 (0, 1, B2)	

Tnbnm P216 Fig. 3-15 (b) 协议4，运行过程不正常



三个窗口协议（协议4-6）

❖ 发送窗口 $W_T=1$ ，接收窗口 $W_R=1$ 

Tnbnm P215 Fig. 3-14 一位滑动窗口协议（协议4）

❖ 发送窗口 $W_T=7$ ，接收窗口 $W_R=1$ 

Tnbnm P220 Fig. 3-17 后退n帧的滑动窗口协议（协议5）

❖ 发送窗口 $W_T=4$ ，接收窗口 $W_R=4$ 

Tnbnm P224 Fig. 3-19 选择性重发滑动窗口协议（协议6）



退后n帧的协议

- ❖ 协议4的主要问题是信道利用率太低：
发送端的等待时间至少是发送端到接收端信号传播时间的两倍
- ❖ 信道的利用率为：
数据发送时间/从数据发送开始到确认返回的总耗时
- ❖ 发送管道化（**pipelining**）：
在等待确认的时间内连续发送
- ❖ 存在的问题主要是信道不可靠：
一旦数据帧丢失，在发送端在意识到丢失（**TimeOut**）时，已有大量数据帧到达接收端



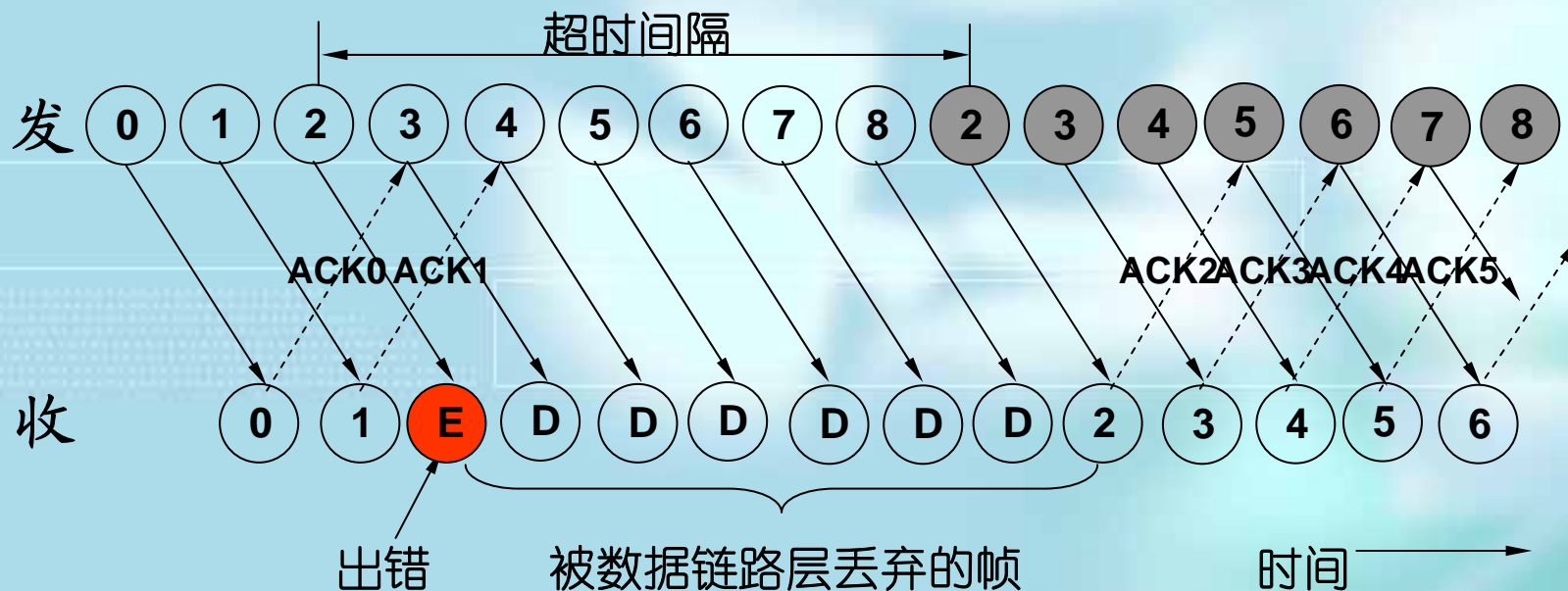
退后n 帧协议 (go back n)

- ❖ 当接收端发现一错帧后，抛弃此帧及以后所有收到的帧，不发确认。当发送端出现超时（错帧的**TimeOut**）后，重发自该帧起的所有已发送帧
- ❖ 问题：浪费带宽
发送方连续发送了7帧（2~8），但2#帧无确认，超时后，从2#帧起开始重发



后退n帧的滑动窗口协议图例

❖ 有一个差错时后退n帧（接收窗口为1、发送窗口为7）

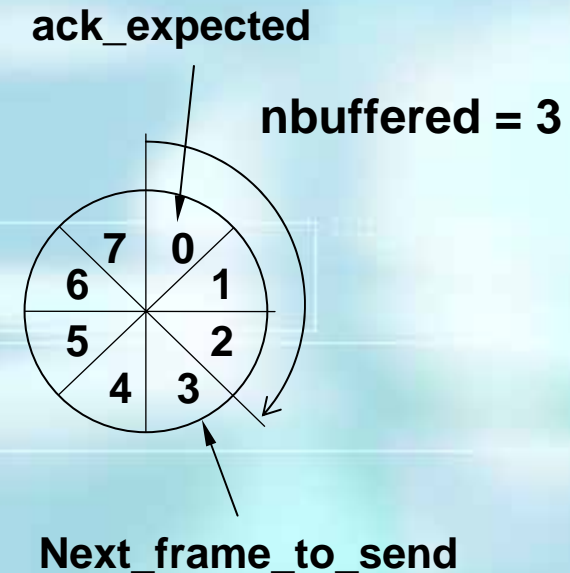


Tnbm P218 Fig. 3-16 (a)



后退n帧的滑动窗口协议程序

```
void protocol5(void)
{
    enable_network_layer();
    ack_expected = 0;
    next_frame_to_send = 0;
    frame_expected = 0;
    nbuffered = 0;
```





```
while (true)
{
    wait_for_event(&event);
    switch (event)
    {
        case network_layer_ready:
            network layer ready处理;
        case frame_arrival: frame arrival处理;
        case cksum_err: 暂且忽略;
        case timeout: timeout处理;
    }
    if nbuffered < MAX_SEQ)
        enable_network_layer();
    else disable_network_layer();
}
}
```



network_layer_ready处理:

from_network_layer

(**&buffer[next_frame_to_send]**);

nbuffered = nbuffered + 1;

send_data

**(next_frame_to_send,
frame_expected,buffer);**

inc(next_frame_to_send);

装配一个数据帧
并发送，发送缓
冲区数+1，准备
发送下一数据帧



frame_arrival处理:

```
from_physical_layer(&r);  
if (r.seq == frame_expected)  
{ to_network_layer(&r.info);  
  inc(frame_expected);  
}
```

如收到一个数据帧
则交网络层, 并期
望接收下一数据帧

```
while (between(ack_expected, r.ack,  
              next_frame_to_send))
```

```
{ nbuffered = nbuffered - 1;  
  stop_timer(ack_expected);  
  inc(ack_expected);  
}
```

如收到一个确认帧
则释放一个缓冲区
定时器复位, 并期
望接收下一确认帧



TimeOut处理:

从等待确认的帧开始全部重发

```
next_frame_to_send = ack_expected;  
for ( i = 1; i <= nbuffered; i++)  
{  
    send_data(next_frame_to_send,  
              frame_expected, buffer);  
    inc(next_frame_to_send);  
}
```



后退n帧的滑动窗口协议

$$(W_T = 2^n - 1, W_R = 1)$$

❖ 如果编号由n位组成，则发送窗口

$$W_T = 2^n - 1, \text{ 接收窗口 } W_R = 1$$

❖ 即管道化 (**pipelining**) 协议

Tnbn P220 Fig. 3-17 (协议5)



三个窗口协议（协议4-6）

❖ 发送窗口 $W_T=1$ ，接收窗口 $W_R=1$ 

Tnbnm P215 Fig. 3-14 一位滑动窗口协议（协议4）

❖ 发送窗口 $W_T=7$ ，接收窗口 $W_R=1$ 

Tnbnm P220 Fig. 3-17 后退n帧的滑动窗口协议（协议5）

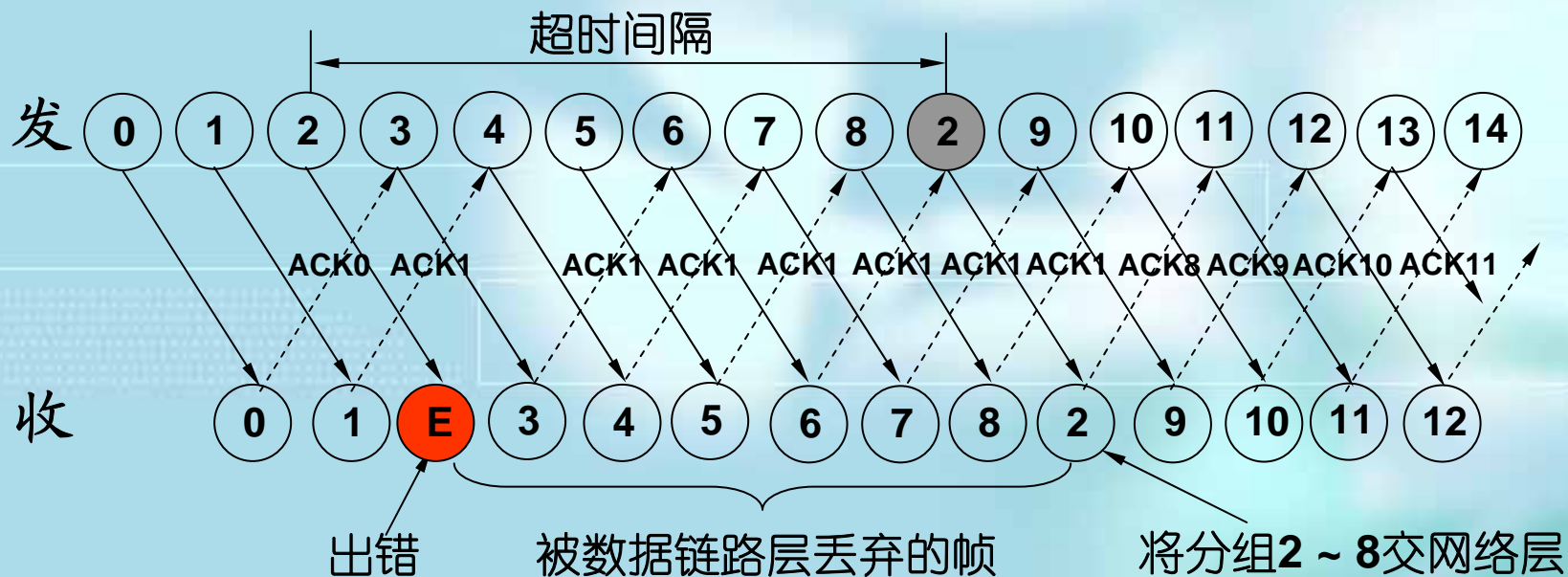
❖ 发送窗口 $W_T=4$ ，接收窗口 $W_R=4$ 

Tnbnm P224 Fig. 3-19 选择性重发滑动窗口协议（协议6）



选择性重发窗口协议图例

- ❖ 有一个差错时仅重发一帧（接收窗口和发送窗口都为7）



Tnbm P218 Fig. 3-16 (b)



选择性重发窗口协议

$$(W_T = 2^n / 2, W_R = 2^n / 2)$$

❖ 如果编号由n位组成:

发送窗口 $W_T = 2^n / 2$, 接收窗口 $W_R = 2^n / 2$

❖ 当接收到一个错帧（包括**CRC**错或帧序号错）时，发送一个否定性确认**NAK**

❖ 接收方定义了一个辅助计时器

Tnbnm P224 Fig. 3-19 （协议6）



选择性重发窗口协议程序

```
void protocol6(void)
{
    enable_network_layer();
    ack_expected = 0;
    next_frame_to_send = 0;
    frame_expected = 0;
    too_far = NR_BUFS;
    nbuffered = 0;
    for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```



```
while (true)
{ wait_for_event(&event);
  switch (event)
  { case network_layer_ready: 发送处理 ;
    case frame_arrival: 帧到达处理 ;
    case cksum_err: 收到一个坏帧且没有发过nak, 则发nak;
      if (no_nak) send_frame(nak,0, frame_expected, out_buf);
    case timeout: 等待确认帧的超时, 重发;
      send_frame(data, oldest_frame, frame_expected, out_buf);
    case ack_timeout: 辅助定时器超时, 发一单独的ack;
      send_frame(ack, 0, frame_expected, out_buf);
    }
  if (nbuffered < NR_BUFS)
    enable_network_layer();如窗口未滿, 則允許網絡層事件;
  else
    disable_network_layer();如窗口滿, 則不允許網絡層事件;
  }
}
```



发送数据处理

```
nbuffered = nbuffered + 1; /* 已用窗口数 + 1 */
```

```
from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]);  
/* 取新的分组 */
```

```
send_frame(数据帧data, 本发送帧序号next_frame_to_send, 捎带确认序号frame_expected, 帧的数据out_buf);  
/* 发送该帧 */
```

```
inc(next_frame_to_send); /* 发送窗口的前沿+1 */
```




帧到达处理

```
from_physical_layer(&r);
```

```
if (r.kind == data)
```

```
{ if ((r.seq != frame_expected) && no_nak)
    send_frame(nak, 0, frame_expected, out_buf);
```

```
    else start_ack_timer();
```

如序号错则发nak, 否则启动辅助定时器

```
    if (between(frame_expected, r.seq, too_far) &&
```

```
        (arrived[r.seq % NR_BUFS] == false))
```

```
    { arrived[r.seq % NR_BUFS] = true;
```

```
      in_buf[r.seq % NR_BUFS] = r.info;
```

如序号在接收窗口范围内,
接受该帧

```
      while (arrived[frame_expected % NR_BUFS])
```

```
      { to_network_layer(&in_buf[frame_expected % NR_BUFS]);
```

```
        no_nak = true; arrived[frame_expected % NR_BUFS] = false;
```

```
        inc(frame_expected); inc(too_far); start_ack_timer();
```

```
      } }
```

```
if (r.kind == nak) && (between
```

r.ack+1正是接收方期望接收的帧

```
(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);
```

处理ack while (between(ack_expected, r.ack, next_frame_to_send))

```
{ nbuffered = nbuffered - 1; stop_timer(ack_expected % NR_BUFS);
```

```
  inc(ack_expected); }
```

已用窗口数-1, 启动辅助定时器,
等待下一个ack

如顺序正确,
则交网络层,
调整参数, 启
动辅助定时器



协议6中 ($n=3$) : $W_T=W_R=7$

考虑一种很极端的情况:

$n=3$ 即 帧号为 **0 1 2 3 4 5 6 7**

且 发送窗口 W_T = 接收窗口 W_R = 7

- ❖ 发送方连续发送了**7**帧, 帧号为**0 1 2 3 4 5 6**, 然后等待确认
- ❖ 在尚未接收到帧前, 接收窗口为**0 1 2 3 4 5 6**。发送方发送的**7**帧正确地收到后, 接收方发出了确认**6**, 意即**0 ~ 6**帧全部收到, 然后取出分组交网络层、清缓冲区并调整窗口为**7 0 1 2 3 4 5**
- ❖ 发送方一直在等待确认, 但确认帧由于某种原因失踪了, 在超时后, 发送方又重发**0 1 2 3 4 5 6**帧, 并等待确认



协议6中 ($n=3$) : $W_T=W_R=7$

(续)

- ❖ 接收方收到**0 1 2 3 4 5 6**帧，认为是第二批来的帧，照正常处理，发现**0 1 2 3 4 5**均在其接收窗口内，当然接收并存入缓冲，**6**丢弃，但由于应首先到达的**7**未到，所以只能发**6 ack**，意即再次确认上次收到的**0 ~ 6**
- ❖ 但在发送方来看，收到了**6 ack**后才知道，重发的**0 ~ 6**总算收到了，于是，调整窗口为**7 0 1 2 3 4 5**，从网络层取分组，并发送第二批帧
- ❖ 接收方在收到**7 0 1 2 3 4 5**后，发现**0 1 2 3 4 5**帧已在缓存中，是重复的，应丢弃，**7**接收。然后交网络层，清缓冲区、调整接收窗口
- ❖ 此时，接收方的网络层发现：数据链路层交来的第二批分组中的**0 1 2 3 4 5**与原来的重复



当 $W_T=W_R=7$ ，时协议6失败的原因

- ❖ 原因在于接收窗口过大，新窗口与原窗口中的有效顺序号有重叠
- ❖ 所以，通常：发送窗口 + 接收窗口 $\leq 2^n$
且：发送窗口 \geq 接收窗口



协议6更具实用性

❖ 协议6中:

发送窗口 = 接收窗口 = $(MAX_SEQ + 1) / 2$

❖ 协议6中: 增加了否定性确认**NAK**, 当收到一个坏帧, 或收到一个非期望的帧, 则发一个**NAK**帧

❖ 协议6中: 增加了一个辅助定时器, 当收到一个正确的帧, 而没有可捎带确认的数据帧, 当辅助定时器超时, 则立即发送一个**ACK**帧



可靠性传输

❖ 差错控制：校验、重发和序号

避免帧错误的保证：帧的校验

避免帧丢失的保证：超时和重发

避免帧重复的保证：帧有序号

❖ 流量控制：窗口协议

发送方和接收方之间的协调

❖ 协议描述和验证



协议描述和验证

由于实际使用的协议非常复杂，需有形式化的和数学的方法来验证协议的正确性

- ❖ 有限状态机模型
(finite state machine)
- ❖ Petri网模型





有限状态机模型

- ❖ 协议机（**protocol machine**）：包括发送方和接收方
- ❖ 有限状态机把协议形式化为一个四元组（**S, M, I, T**）其中：
 - S**：进程和信道可能进入的状态集合
 - M**：能在信道上进行交换的帧的集合
 - I**：进程初始状态的集合
 - T**：两两状态之间转换的集合

每个系统状态都可分解为：发送方状态、接收方状态、信道状态

状态在发生某个事件时，可能会转换到另一个状态，把状态作为节点，转换用有向线段表示，则协议的状态图是一个有向图

有向图中的节点（状态）是一个三元组（发送方状态 接收方状态 信道状态）



协议3的有限状态机模型

- ❖ 协议3: 在非可靠信道上实现单向流量控制协议
- ❖ 协议3的要点:
 - ❖ 帧的顺序号为1 位
 - ❖ 发送方要记录下一个准备发送的顺序号
 - ❖ 接收方要记录下一个期望接收的顺序号
 - ❖ 接收方收到一个正确的帧, 即便不是期望的帧, 都将发送一个空的确认帧
 - ❖ 发送过程和接收过程是严格交替的



协议3的有限状态机模型 (续1)

❖ 协议3的发送方、接收方和信道的状态如下:

发送方状态	0	发送了 0 号帧
	1	发送了 1 号帧
接收方状态	0	期望接收 0 号帧
	1	期望接收 1 号帧
信道状态	0	信道上有 0 号帧
	1	信道上有 1 号帧
	A	信道上有 ACK 帧
	S	信道上没有任何帧



协议3的有限状态机模型 (续2)

《Computer Networks v4》cs.sjtu 2004-5-20

状态	动作	下一状态及说明
0 0 0	发送方发送0, 接收方期待0, 信道上为0	0 1 A, 正常
	0帧丢失	0 0 S, 发送方将重发0帧
0 0 1	发送方发送0, 接收方期待0, 信道上为1	不可能出现 发送0, 信道上不可能出现1
0 0 A	发送方发送0, 接收方期待0, 信道上为ACK	不可能出现 在发送ACK前, 接收方应期待1
0 0 S	发送方重发0, 接收方期待0, 信道上为空	0 0 0 0帧丢失, 将重发0帧
0 1 0	发送方发送0, 接收方期待1, 信道上为0	0 1 A 接收方正确接收0, 但拒收, 但仍发ACK
	0帧丢失	0 1 S, 发送方将重发0帧
0 1 1	发送方发送0, 接收方期待1, 信道上为1	不可能出现 发送0, 信道上不可能出现1
0 1 A	发送方发送0, 接收方期待1, 信道上为ACK	1 1 1, 正常 接收方发送ACK, 期待1
	ACK帧丢失	0 1 S, 发送方将重发0帧
0 1 S	发送方重发0, 接收方期待1, 信道上为空	0 1 0 0帧丢失, 发送方将重发0帧



协议3的有限状态机模型 (续3)

《计算机网络 (第4版)》cs.sjtu 2004-5-20

状态	动作	下一状态及说明
1 0 0	发送方发送1, 接收方期待0, 信道上为0	不可能出现 发送1, 信道上不可能出现0
1 0 1	发送方发送1, 接收方期待0, 信道上为1	1 0 A 接收方正确接收1, 但拒收, 但仍发ACK
	1帧丢失	1 0 S, 发送方将重发1帧
1 0 A	发送方发送1, 接收方期待0, 信道上为ACK	0 0 0, 正常 接收方发送ACK, 期待0
	ACK帧丢失	1 0 S, 发送方将重发1帧
1 0 S	发送方重发1, 接收方期待0, 信道上为空	1 0 1 1帧丢失, 发送方将重发1帧
1 1 0	发送方发送1, 接收方期待1, 信道上为0	不可能出现 发送1, 信道上不可能出现0
1 1 1	发送方发送1, 接收方期待1, 信道上为1	1 0 A, 正常 接收方正确接收1后, 期待0, 并发ACK
	1帧丢失	1 1 S, 发送方将重发1帧
1 1 A	发送方发送1, 接收方期待1, 信道上为ACK	不可能出现 在发送ACK前, 接收方应期待0
1 1 S	发送方发送1, 接收方期待1, 信道上为空	1 1 1 1帧丢失, 将重发1帧

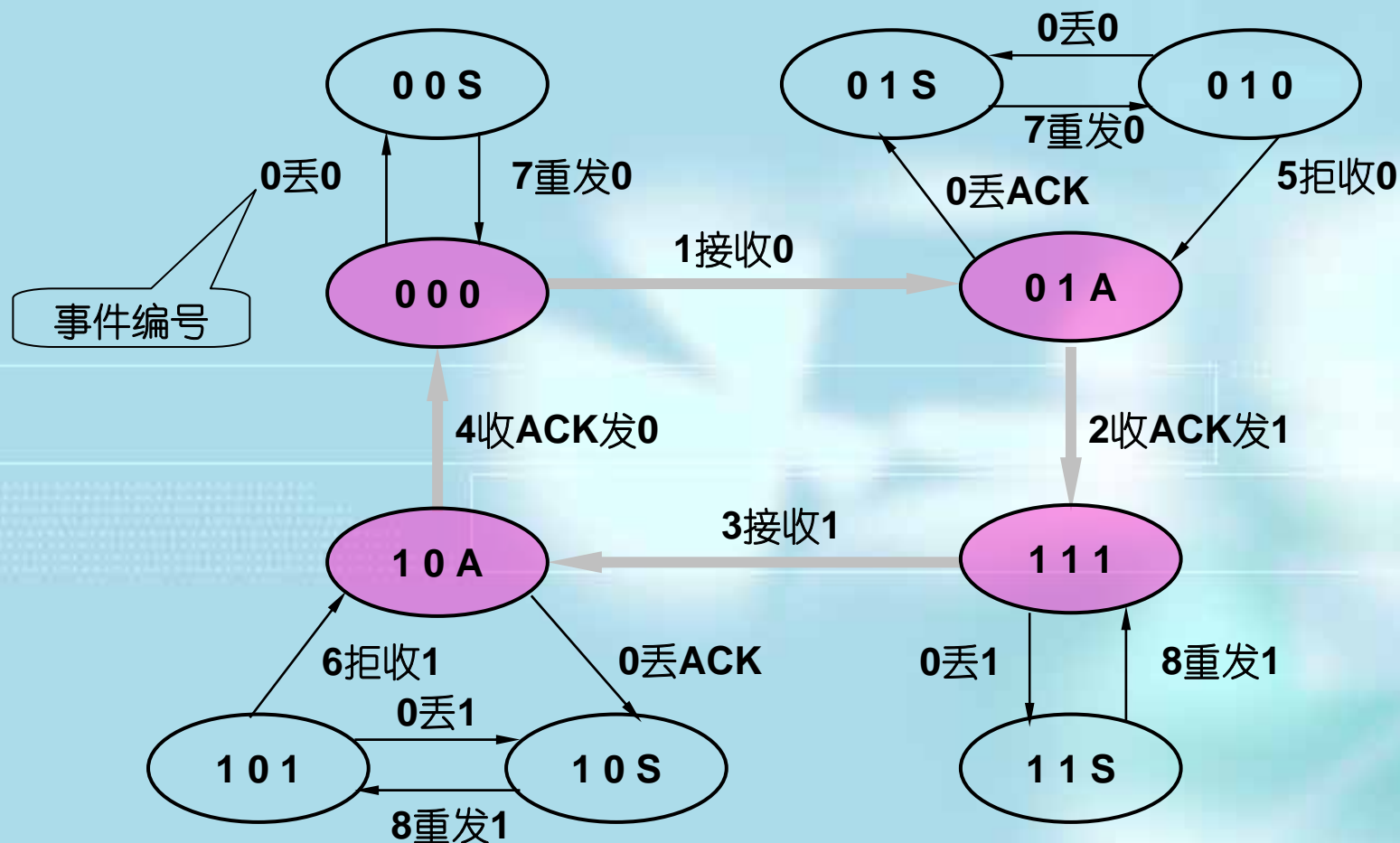


按事件编号排序

从	事件	事件编号	到
0 0 0	丢失 0 帧	0	0 0 S
0 1 0	丢失重发 0 帧	0	0 1 S
0 1 A	丢失 ACK 帧	0	0 1 S
1 0 1	丢失重发 1 帧	0	1 0 S
1 0 A	丢失 ACK 帧	0	1 0 S
1 1 1	丢失 1 帧	0	1 1 S
0 0 0	接收 0 号帧	1	0 1 A
0 1 A	发送 1 号帧	2	1 1 1
1 1 1	接收 1 号帧	3	1 0 A
1 0 A	发送 0 号帧	4	0 0 0
0 1 0	拒收 0 号帧	5	0 1 A
1 0 1	拒收 1 号帧	6	1 0 A
0 0 S	重发 0 号帧	7	0 0 0
0 1 S	重发 0 号帧	7	0 1 0
1 0 S	重发 1 号帧	8	1 0 1
1 1 S	重发 1 号帧	8	1 1 1



协议3有限状态机状态变迁图



Tnbnm P231 Fig. 3-21 协议3的状态图



事件和状态变迁表

《Computer Networks v4》cs.sjtu 2004-5-20

编号	事件简称	事件	动作	状态变迁
0	数据丢失	信道出错, 造成帧丢失	无	(X X X) → (X X S)
1	接收0号帧	0号帧到达正期待0号帧的接收方	接收0号帧, 改期待帧号为1, 并向信道发ACK	(0 0 0) → (0 1 A)
2	发送1号帧	ACK帧到达发送0号帧的发送方	接收ACK帧, 发送帧号改为1, 并从网络层取分组发送1号帧	(0 1 A) → (1 1 1)
3	接收1号帧	1号帧到达正期待1号帧的接收方	接收1号帧, 改期待帧号为0, 并向信道发ACK	(1 1 1) → (1 0 A)
4	发送0号帧	ACK帧到达发送1号帧的发送方	接收ACK帧, 发送帧号改为0, 并从网络层取分组发送0号帧	(1 0 A) → (0 0 0)
5	拒收0号帧	0号帧到达正期待1号帧的接收方	从信道取下0号帧, 拒收, 即不交网络层, 并向信道发ACK	(0 1 0) → (0 1 A)
6	拒收1号帧	1号帧到达正期待0号帧的接收方	从信道取下1号帧, 拒收, 即不交网络层, 并向信道发ACK	(1 0 1) → (1 0 A)
7	重发0号帧	发送0号帧的发送方等待ACK计时器超时	重发0号帧	(0 X S) → (0 X 0)
8	重发1号帧	发送1号帧的发送方等待ACK计时器超时	重发1号帧	(1 X S) → (1 X 1)



检查内容

- ❖ 检查有无死锁
- ❖ 不管事件如何发生，接收方不会连续收到两个**0**帧或两个**1**帧
- ❖ 每个状态都是可达的



协议描述和验证

由于实际使用的协议非常复杂，需有形式化的和数学的方法来验证协议的正确性

- ❖ 有限状态机模型
(finite state machine)
- ❖ Petri网模型





Petri网模型

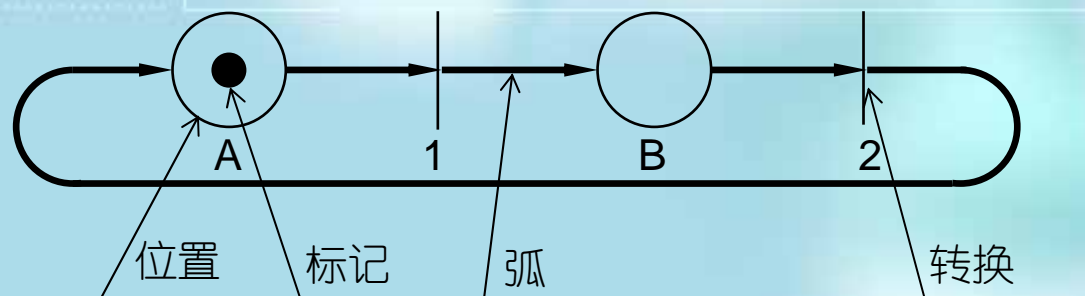
❖ Petri网有四个基本元素：

位置 (**place**)：系统可能处的状态，用圈表示

转换 (**transition**)：状态将作转换，用粗线段表示

弧 (**arc**)：包括输入弧和输出弧，用带箭头的弧线表示

标记 (**token**)：系统目前所处的位置，用园点表示



Tnbm P232 Fig. 3-22 具有2个位置和2个转换的Petri网



Petri网模型运算规则

- ❖ 当一个转换的输入弧中至少有一个输入标记时，则状态转换允许
- ❖ 转换允许的状态一旦被触发，则从每个输入位置上移走一个标记，并在每个输出位置上填上一个标记
- ❖ 每个转换或位置上，如果输入弧与输出弧的个数不相等，则标记不会守恒，即意味着可能会出错

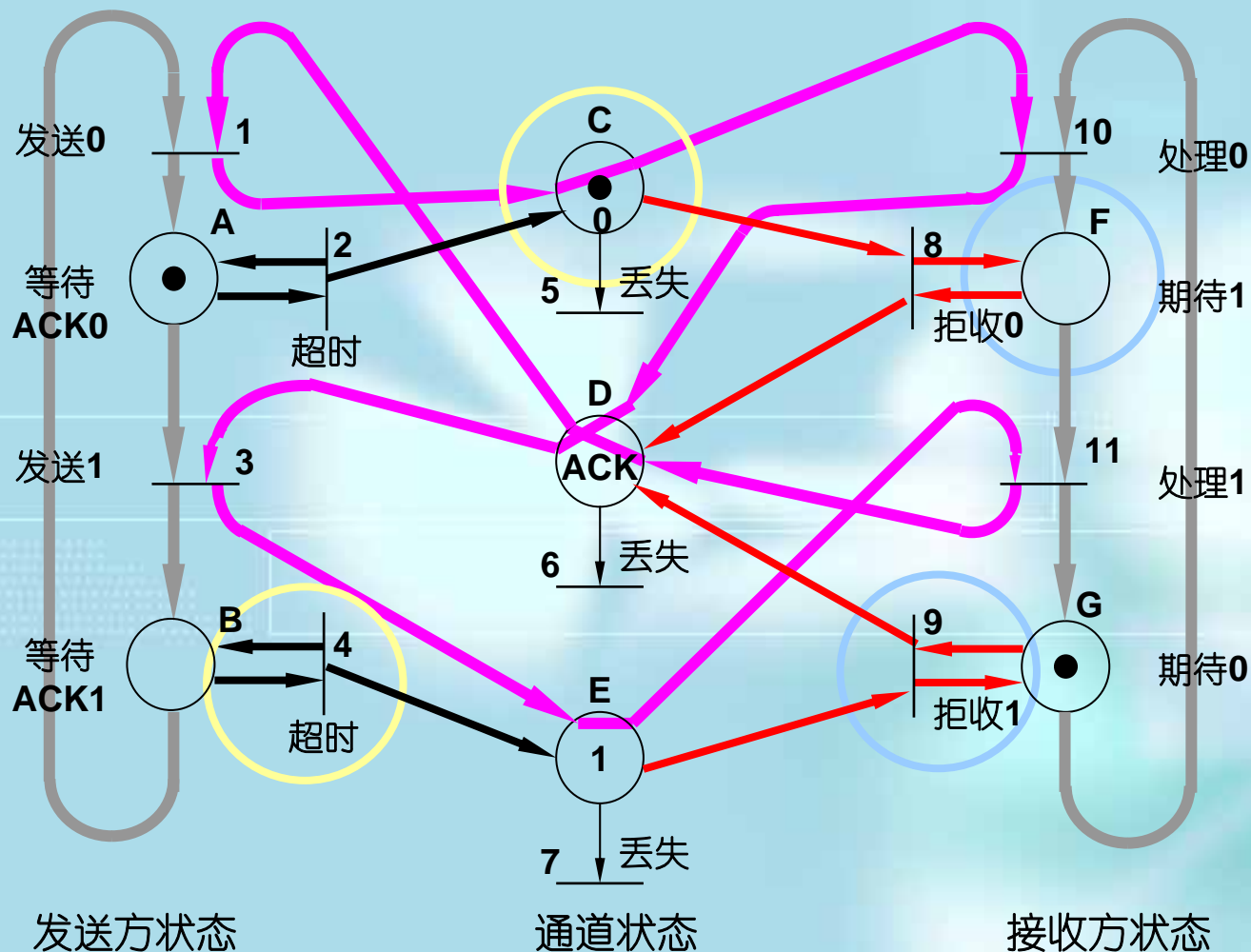


协议3的Petri模型

- ❖ 发送方、接收方和通道的状态都分别表示
- ❖ 发送方正常运行的状态变迁轨迹自成环路（灰线）
- ❖ 接收方正常运行的状态变迁轨迹自成环路（灰线）
- ❖ 通道正常运行的状态变迁轨迹自成环路（玫瑰红线）
- ❖ 某个位置和转换上，输入弧和输出弧相等，意味着正常，如蓝圈所示（位置F、转换9），否则意味着出错，如黄圈所示（位置C、转换4）
- ❖ 当出现0帧丢失或0帧的ACK帧丢失，将触发转换5或转换6，这对于发送方来说，意味着超时，所以将触发转换2，重发0帧（粗白箭头所示），并重新同步
- ❖ 如由于0帧的ACK帧丢失引起的重发（粗红箭头所示），因为接收方的期待帧号为1，所以将触发转换8，拒收0帧，但需发ACK帧（粗红箭头所示）
- ❖ 如出现1帧丢失或1帧的ACK帧丢失，情况同上







协议3的Petri网状态变迁图



Tnbm P233 Fig. 3-23 协议3的Petri网模型



第3章 数据链路层

- ❖ 定义，功能 
- ❖ 数据帧的组成 
- ❖ 可靠性传输 
- ❖ 数据链路层示例 



数据链路层示例

❖ **HDLC – 高级数据链路控制**



❖ **因特网中的数据链路层**



➤ **SLIP --- 串型线路IP**

➤ **PPP --- 点对点协议**



HDLC 的基本工作原理

- ❖ 最早由**IBM SNA**提出**SDLC**
(**Synchronous Data Link Control**)
- ❖ **ISO**根据**SDLC**，提出了**HDLC** (**High level Data Link Control**)
- ❖ 是面向**bit**的同步通信协议



HDLC的配置方式

- ❖ 非平衡配置：又分为点对点 and 点对多点两种。非平衡配置的特点是有有一个主站及一个或多个从站组成。主站发出的帧叫命令，从站发出的帧叫响应
- ❖ 平衡配置：两个站都是复合站，同时具有主站和从站的功能



HDLC的帧格式




8	8	8	≥ 0	16	8
01111110	地址	控制	数据	校验和	01111110

帧标志序列	即 01111110 ，作为帧的分隔标志，如线路空闲，则用标志序列填充，用位插入方法实现透明传输
地址域	在总线型多终端情况下，是终端的站号；在点对点的情况下，用来标志命令和响应
控制域	定义帧的类型、序号等和其它一些功能
数据域	用户数据，长度任意
校验和	CRC 码， ISO 和 CCITT 有相似的生成多项式



HDLC的帧类型

❖ **HDLC**的帧有三种类型，不同的类型其控制域的定义有些不同

- 信息帧（**I**）： **Information Frame** 
- 监控帧（**S**）： **Supervisory Frame** 
- 无序号帧（**U**）： **Unnumbered Frame** 



信息帧 (I)

0	当前发送帧号	P/F	捎带确认帧号
---	--------	-----	--------

P/F (Poll/Final) 位:

P 主机查询哪个终端要发送数据

F 终端发送数据的最后一帧用**F**

捎带确认帧号:

即下一个期望接收帧号，意味着以前的帧的接收确认



HDLC的帧类型

❖ **HDLC**的帧有三种类型，不同的类型其控制域的定义有些不同

➤ 信息帧（**I**）：**Information Frame**



➤ 监控帧（**S**）：**Supervisory Frame**




➤ 无序号帧（**U**）：**Unnumbered Frame**





监控帧 (S)

❖ 监控帧 有四种格式

RR		1	0	0	0	P/F	捎带确认号
RNR		1	0	0	1	P/F	捎带确认号
REJ		1	0	1	0	P/F	捎带确认号
SREJ		1	0	1	1	P/F	捎带确认号



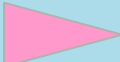



接收准备好

- ❖ **RR (Receive Ready)** : 接收准备好
- ❖ 即确认**Next-1**及**Next-1**帧以前的所有帧，并准备好接收**Next**帧



监控帧 (S)

❖ 监控帧 有四种格式

RR		1	0	0	0	P/F	捎带确认号
RNR		1	0	0	1	P/F	捎带确认号
REJ		1	0	1	0	P/F	捎带确认号
SREJ		1	0	1	1	P/F	捎带确认号



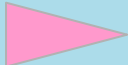



接收未准备好

- ❖ **RNR (Receive Not Ready)** : 接收未准备好
- ❖ 即确认**Next-1**及**Next-1**帧以前的帧，并要求发送方停止发送



监控帧 (S)

❖ 监控帧 有四种格式

RR		1	0	0	0	P/F	捎带确认号
RNR		1	0	0	1	P/F	捎带确认号
REJ		1	0	1	0	P/F	捎带确认号
SREJ		1	0	1	1	P/F	捎带确认号



拒绝接收

- ❖ **REJ (REJect)** : 拒绝接收
- ❖ 即否定性确认, 确认**Next-1**及**Next-1**帧以前的所有帧, 并要求重发**Next**及**Next**以后的所有帧



监控帧 (S)

❖ 监控帧 有四种格式

RR	▶	1	0	0	0	P/F	捎带确认号
RNR	▶	1	0	0	1	P/F	捎带确认号
REJ	▶	1	0	1	0	P/F	捎带确认号
SREJ	▶	1	0	1	1	P/F	捎带确认号



选择性拒收

- ❖ **SREJ (Selective REJect)** : 选择性拒收
- ❖ 即否定性确认，确认**Next-1**及**Next-1**帧以前的所有帧，并仅要求重发**Next**帧。（此类型在**SDLC**及**LAPB**中不允许）



HDLC的帧类型

❖ **HDLC**的帧有三种类型，不同的类型其控制域的定义有些不同

- 信息帧（**I**）： **Information Frame**
- 监控帧（**S**）： **Supervisory Frame**
- 无序号帧（**U**）： **Unnumbered Frame**





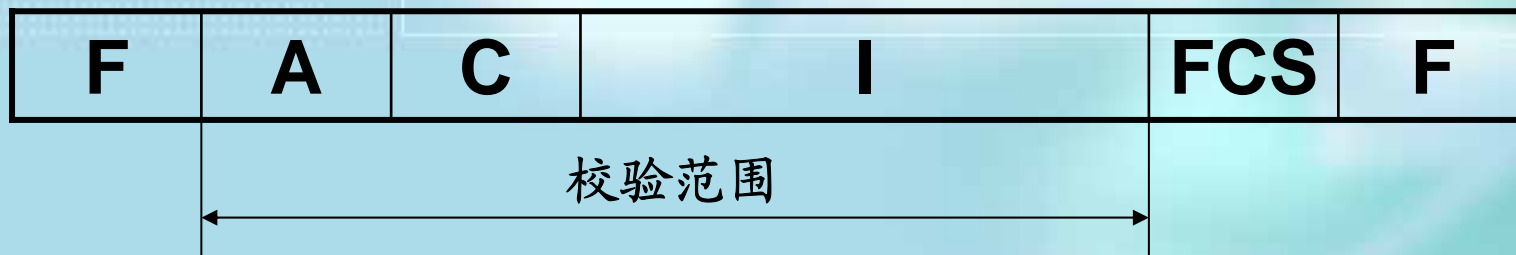
无序号帧 (U)

- ❖ 控制字段的**1、2 bit**都为**1**的帧为无序号帧
- ❖ 无序号帧用于控制链路本身。如呼叫、确认和断开连接等控制
- ❖ 在无序号帧中用**5位 (3, 4, 6, 7, 8)**来表示无序号帧的类型，但**32**种可能的类型中有些是保留的，并且，不同的协议中，无序号类型区别很大 (略)



HDLC帧的校验


- ❖ CRC校验
- ❖ 生成多项式为 $X^{16}+X^{12}+X^5+1$
- ❖ 校验范围





数据链路层示例

❖ **HDLC – 高级数据链路层控制** 

❖ **因特网中的数据链路层** 

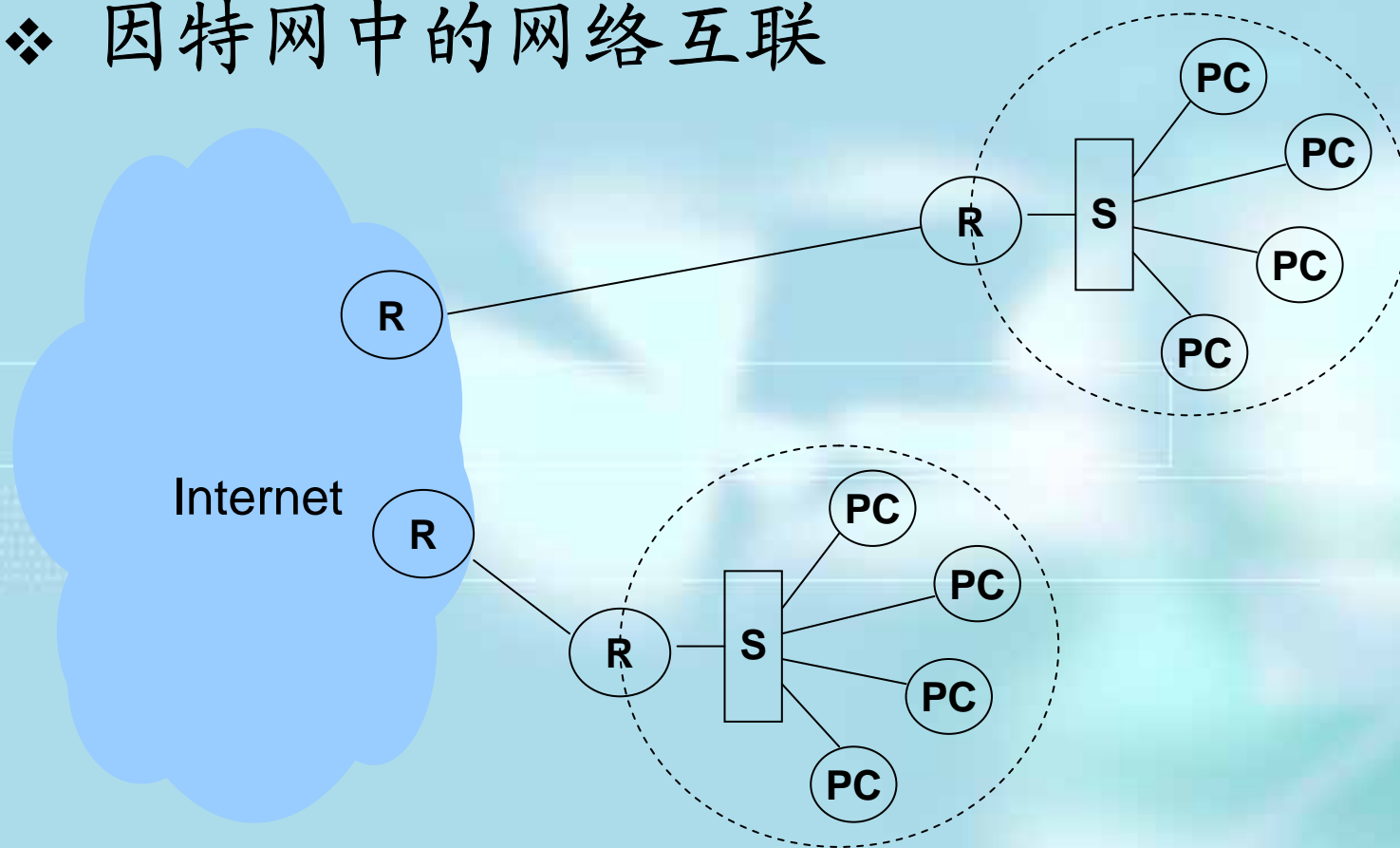
➤ **SLIP --- 串型线路IP**

➤ **PPP --- 点对点协议**



因特网模型 1

❖ 因特网中的网络互联

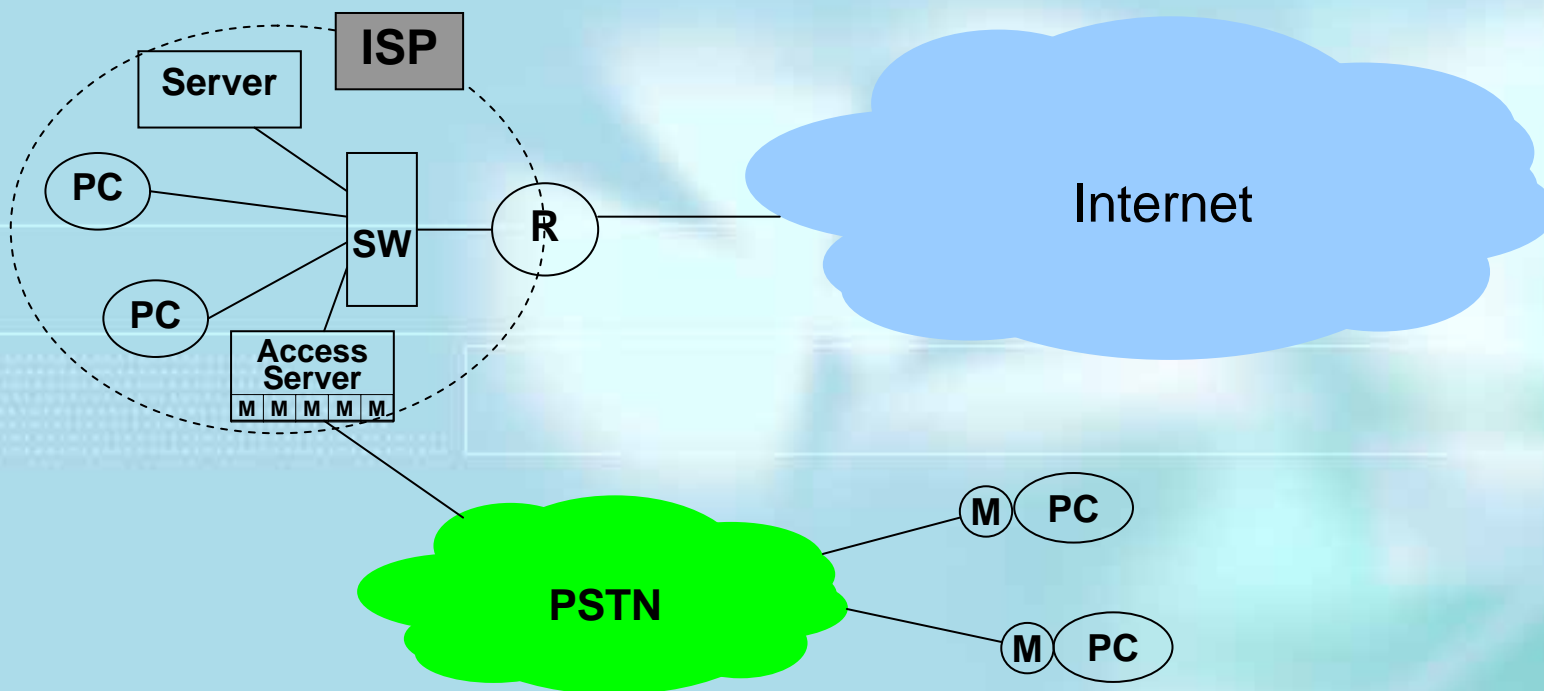


其中的路由器之间是通过点对点来连接的



因特网模型 2



❖ 因特网中家庭用户的入网



远程家庭用户通过拨号与**ISP**联接后，采用点对点方式连接



因特网中的数据链路层

- ❖ **SLIP --- 串型线路IP** 
(**Serial Line IP**)
- ❖ **PPP --- 点对点协议** 
(**Pointe – to – Point Protocol**)





SLIP --- 串型线路IP

❖ SLIP --- 串型线路IP (Serial Line IP)

协议简单，无差错控制，只适用于**IP**，
通信的每一方必须事先知道对方的**IP**地
址，只支持低速的业务 (**19.2K**)
未成为标准，不详细介绍



因特网中的数据链路层

- ❖ **SLIP --- 串型线路IP** 
(**Serial Line IP**)
- ❖ **PPP --- 点对点协议** 
(**Pointe – to – Point Protocol**)

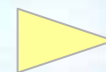


PPP --- 点对点协议

❖ PPP --- 点对点协议

(Pointe – to – Point Protocol)

- PPP的功能
- PPP的组成部分
- PPP的帧格式
- PPP的链路控制协议
- PPP的工作过程





PPP的功能

❖ PPP是Internet标准

(RFC1661 1662 1663)

- 处理错误监测
- 支持多种协议 (**IP**、**IPX**、**DECnet**等)
- 连接时允许协商**IP**地址
- 允许身份认证



PPP --- 点对点协议

❖ PPP --- 点对点协议

(Pointe – to – Point Protocol)

- PPP的功能
- PPP的组成部分
- PPP的帧格式
- PPP的链路控制协议
- PPP的工作过程





PPP的组成部分

- ❖ **PPP**提供了串行点对点链路上传输数据报的方法，包括以下三个部分：
 - 串行链路上封装数据报的方法。既支持异步链路，也支持面向**bit**的同步链路
 - 扩展的链路控制协议（**Link Control Protocol - LCP**），用于建立、配置和测试数据链路的连接
 - 网络控制协议（**NCP**）簇，支持各种网络层协议



PPP --- 点对点协议

❖ PPP --- 点对点协议

(Pointe – to – Point Protocol)

- PPP的功能
- PPP的组成部分
- PPP的帧格式
- PPP的链路控制协议
- PPP的工作过程





PPP的帧格式

❖ **PPP的帧格式类似于HDLC，但是面向字符的协议（以字节为单位）**

1	1	1	1/2	可变	2/4	1
标志 01111110	地址 11111111	控制 00000011	协议	有效载荷	校验和	标志 01111110

标志域：固定为**01111110**，与**HDLC**相同 协议域：不同的协议不同的代码

地址域：固定为**11111111**

载荷域：可变长，缺省最长**1500**字节

控制域：缺省为**00000011**，即无序号帧
(即毋需确认)

校验和：缺省为**2**字节，也可定义为**4**字节，仅是头部的校验和

由于地址域和控制域基本固定，所以在**LCP**中省略



PPP --- 点对点协议

❖ PPP --- 点对点协议

(Pointe – to – Point Protocol)

- PPP的功能
- PPP的组成部分
- PPP的帧格式
- PPP的链路控制协议
- PPP的工作过程





PPP的链路控制协议LCP

- ❖ **PPP的LCP (Link Control Protocol)** 提供了建立、配置、维护和终止点对点链接的方法
- ❖ **LCP的过程按以下四个阶段进行:**
 - 链路的建立和配置协调
 - 链路质量检测
 - 网络层协议配置阶段
 - 关闭链路



LCP帧的类型

❖ **LCP帧的类型有三种：**

- 链路建立帧：建立和配置链路
- 链路终止帧：终止链路
- 链路维护帧：管理、维护链路



PPP --- 点对点协议

❖ PPP --- 点对点协议

(Pointe – to – Point Protocol)

- PPP的功能
- PPP的组成部分
- PPP的帧格式
- PPP的链路控制协议
- PPP的工作过程





PPP的工作过程

- ❖ 发送端**PPP**首先发送**LCP**帧，以配置和测试数据链路
- ❖ 在**LCP**建立好数据链路并协调好所选设备之后，发送端**PPP**发送**NCP**帧，以选择和配置一个或多个网络协议
- ❖ 当所选的网络层协议配置好后，便可将各网络层协议的分组发送到数据链路上
- ❖ 配置好的链路将一直保持通信状态，直到**LCP**帧或**NCP**帧明确提示关闭链路，或有其它的外部事件发生（如用户干预等）

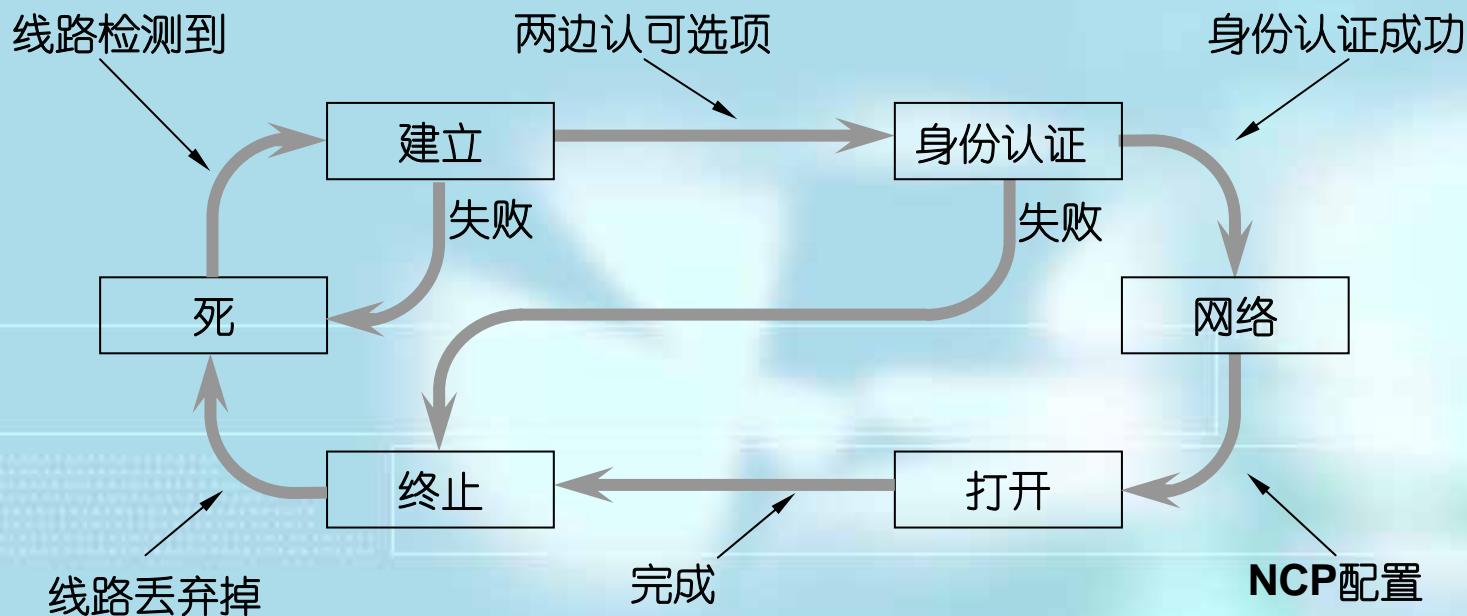


一次使用PPP协议的过程

1. 初始状态
2. 建立连接: 建立成功到**3)**, 否则到**1)**
3. 选项协商: 协商成功到**4)**, 否则到**7)**
4. 身份认证: 认证成功到**5)**, 否则到**7)**
5. 配置网络: 网络配置完后到**6)**
6. 数据传输: 数据传输完后到**7)**
7. 释放链路: 回到**1)**



一次使用PPP协议的状态图



Tnbm P241 Fig. 3-28

建立/取消线路的简化阶段流程图



第3章 习题

Tnbm P243

#5 、 #15、 #17 (#11)

#18 (#12) 、 #29 (#22)

#31 (#24)



v3中没有的习题

5. A bit string, 011110111110111110, needs to be transmitted at the data link layer. What is the string actually transmitted after bit stuffing?
15. A bit stream 10011101 is transmitted using the standard CRC method described in the text. The generator polynomial is x^3+1 . Show the actual bit string transmitted. Suppose the third bit from the left is inverted during transmission. Show that this error is detected at the receiver's end.