Computer Networks

Chapter 3
Data Link Layer

主要内容

3.1 定义和功能

1. 定义
2. 为网络层提供服务
3. 成帧
4. 差错控制
5. 流量控制

3.2 错误检测和纠正

1. 纠错码
2. 检错码

3.3 基本的数据链路层协议

1. 无约束单工协议
2. 单工停等协议
3. 有噪声信道的单工协议

3.4 滑动窗口协议

1. 一比特滑动窗口协议

2. 退后n帧协议

3. 选择重传协议

3.5常用的数据链路层协议

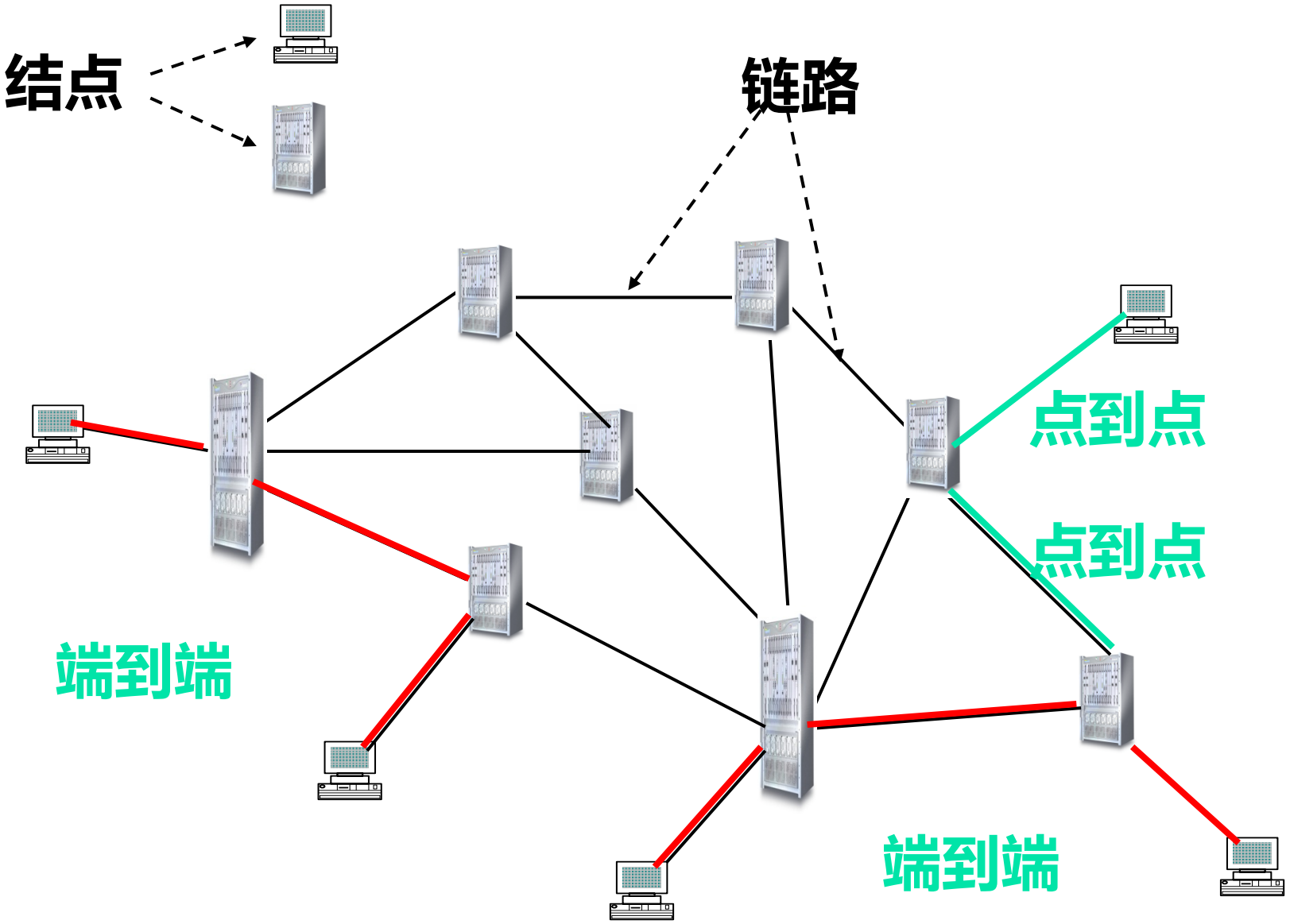
3.1 定义和功能（1）

1. 定义

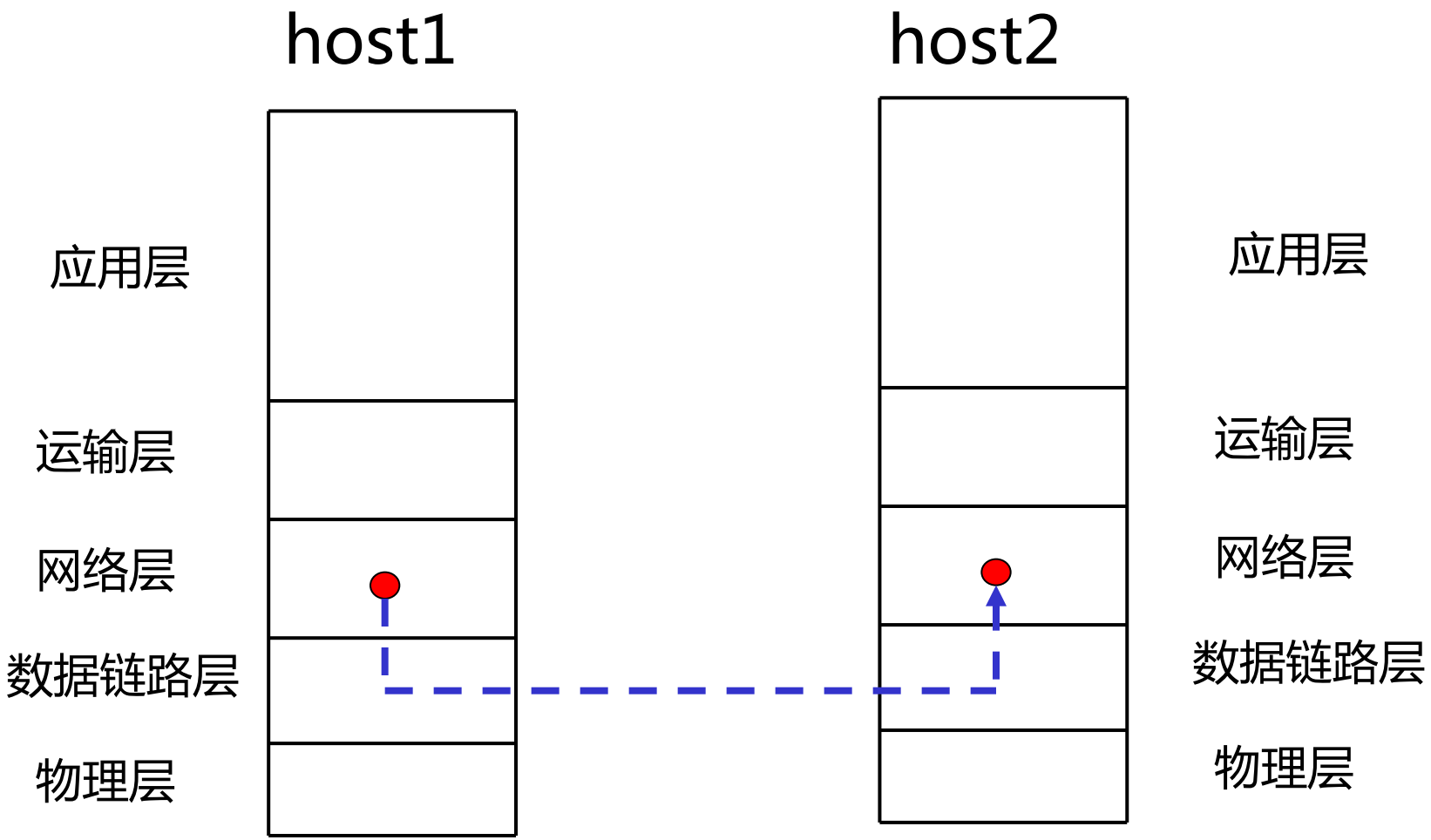
- 要解决的问题
  - 如何在有差错的线路上，进行无差错传输
- ISO关于数据链路层的定义
  - 数据链路层的目的是为了提供功能上和规程上的方法，以便建立、维护和释放网络实体间的数据链路。

3.1 定义和功能（2）

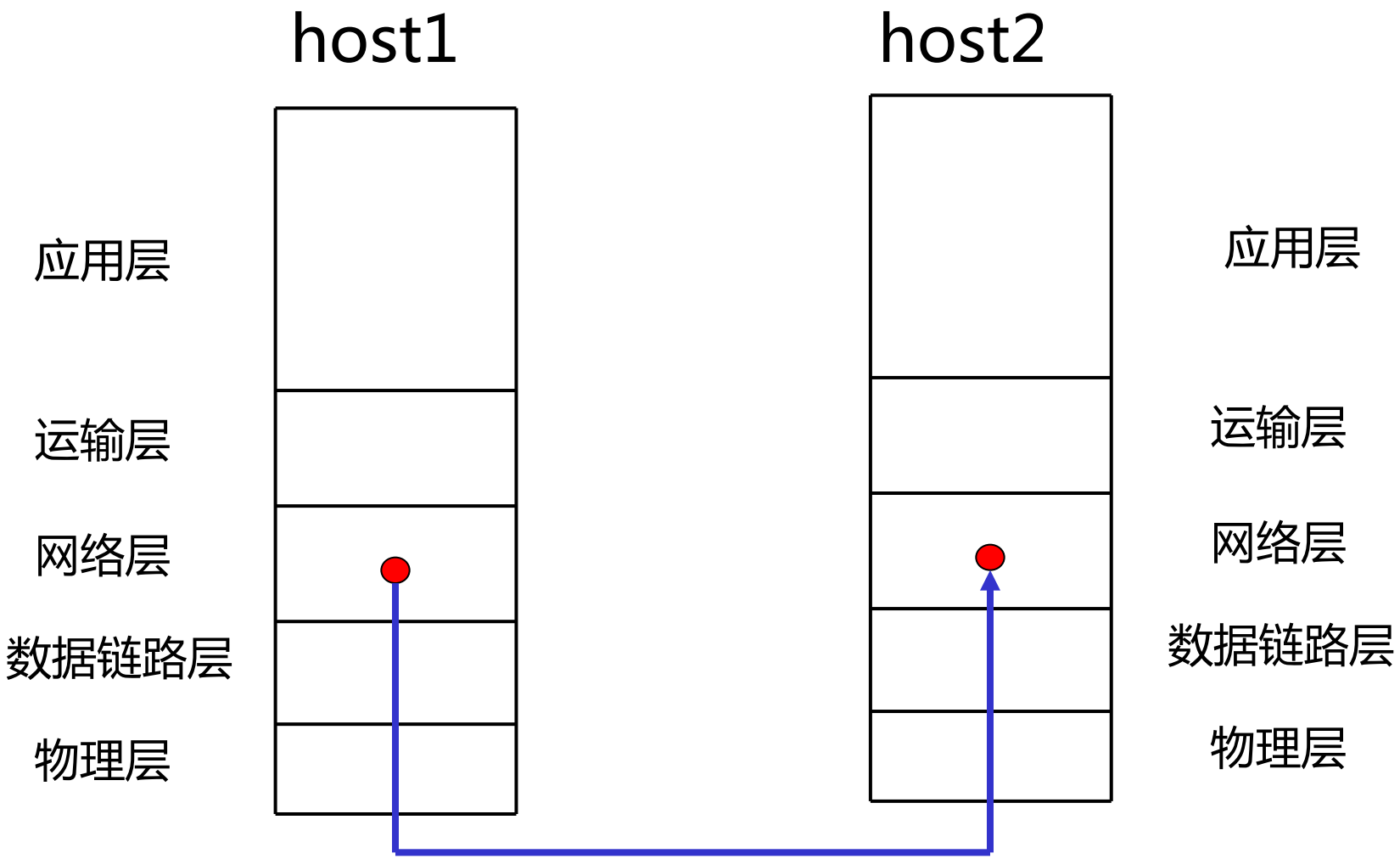
- 基本概念
  - **结点（node）**：网络中的主机（host）和路由器（router）称为结点
  - **链路（link）**：通信路径上连接相邻结点的通信信道称为链路
    - 数据链路层协议定义了一条链路的两个结点间交换的数据单元格式，以及结点发送和接收数据单元的动作。
  - **端到端（end to end）**：从源结点（source node）到目的结点（destination node）的通信称为端到端通信，通信路径（path）可能由多个链路组成。
  - **点到点（point to point）**：在相邻结点间的一条链路上的通信称为点到点通信。
  - **虚拟数据通路**
  - **实际数据通路**


结点
链路
点到点
点到点
端到端
端到端

虚拟数据通路 (host1 to host2)



实际数据通路(host1 to host2)



3.1 定义和功能（3）

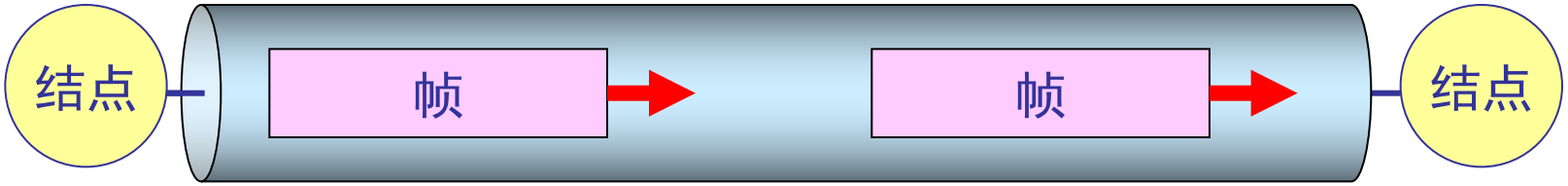
- 数据链路控制规程
  - 为使数据能迅速、正确、有效地从发送点到达接收点所采用的控制方式。
- 数据链路层协议应提供的最基本功能
  - 数据在数据链路上的正常传输(建立、维护和释放)
  - 定界与同步，也处理透明性问题
  - 差错控制
  - 顺序控制
  - 流量控制

数据链路层基本概念说明

- 链路(link)是一条无源的点到点的物理线路段，中间没有任何其他的交换结点。
  - 一条链路只是一条通路的一个组成部分。
- 数据链路(data link) 除了物理线路外，还必须有通信协议来控制这些数据的传输。若把实现这些协议的硬件和软件加到链路上，就构成了数据链路。
  - 现在最常用的方法是使用适配器（即网卡）来实现这些协议的硬件和软件。
  - 一般的适配器都包括了数据链路层和物理层这两层的功能。

数据链路层像个数字管道

- 常常在两个对等的数据链路层之间画出一个数字管道，而在这条数字管道上传输的数据单位是帧。



- 早期的数据通信协议曾叫作通信规程(procedure)。因此在数据链路层，规程和协议是同义语。

数据链路层的主要功能

(1) 链路管理
(2) 帧定界
(3) 流量控制
(4) 差错控制
(5) 将数据和控制信息区分开
(6) 透明传输
(7) 寻址

3.1 定义和功能（4）

2. 设计目标：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

3.1 定义和功能（5）

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
  - 带字符填充的首尾字符定界法
  - 带位填充的首尾标记定界法
  - 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。

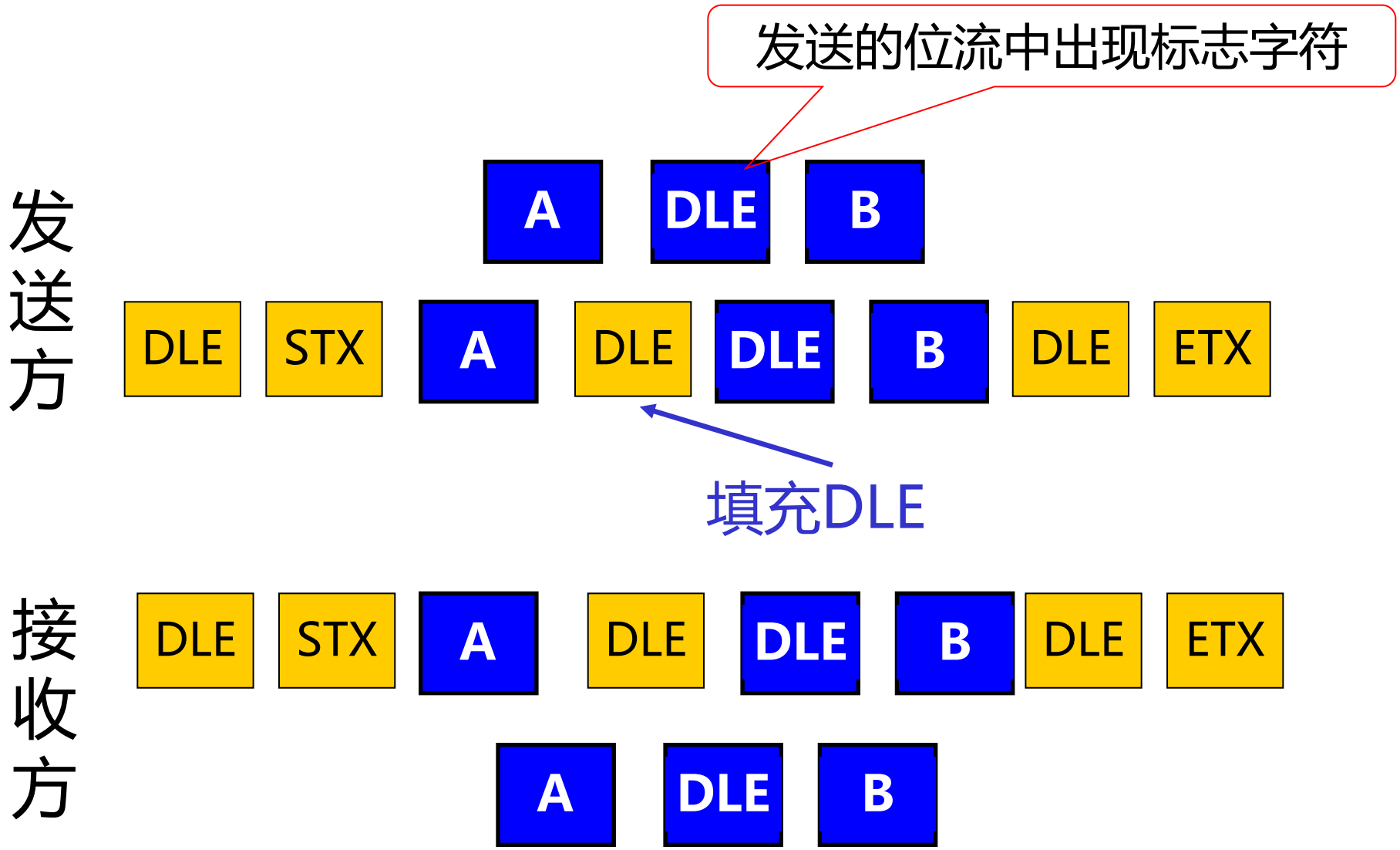
3.1 定义和功能（6）

- 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

3.1 定义和功能（7）

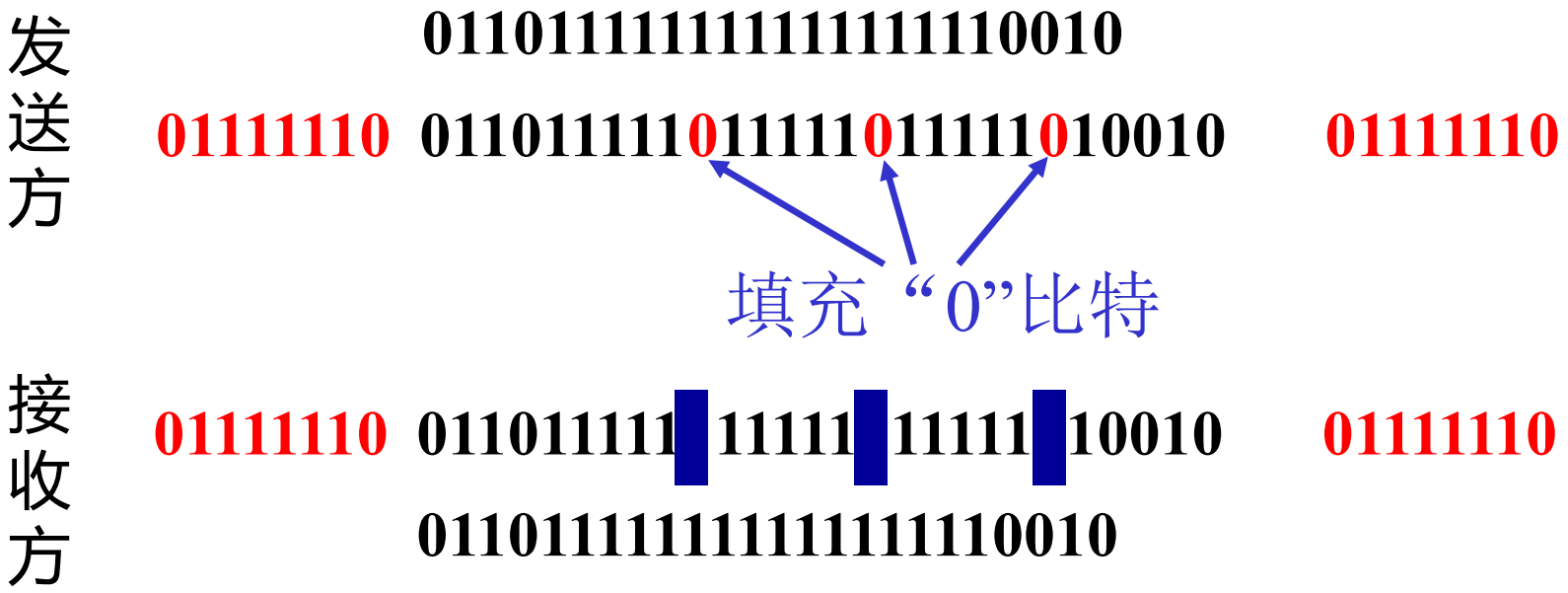
- 带字符填充的首尾字符定界法
  - 起始字符 DLE STX，结束字符DLE ETX
    - DLE:Data Link Escape
    - STX:Start of Text
    - ETX:End of Text
  - 字符填充：数据中出现标志字符时插入转义字符
  - 缺点：局限于8位字符和ASCII字符传送。

首尾字符定界法举例


发送的位流中出现标志字符
发送方
A
DLE
B
DLE
STX
A
DLE
DLE
B
DLE
ETX
填充DLE
接收方
DLE
STX
A
DLE
DLE
B
DLE
ETX
A
DLE
B

3.1 定义和功能（8）

- 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(flag)
  - “0”比特插入删除技术



3.1 定义和功能（9）

- 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：曼彻斯特编码或差分曼彻斯特编码用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。

3.1 定义和功能（10）

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. 流量控制

- 基于反馈机制
- 流量控制主要在传输层实现

3.2 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。

3.2 错误检测和纠正（2）

1. 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比特(对应)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；
- 对于n位码字的集合，只有2^m个码字是有效的。也就是说，通常并未使用所有2^n个码字。

检错和纠错

- 一种编码的检错和纠错能力取决于编码后码字海明距离的大小。
- 为了检测出d个比特的错，需要使用海明距离为d+1的编码
  例如：数据后加奇偶校验位，编码后的海明距离为2，能检测1比特错。
- 例如：
  000 0(前三个数据位，后一个是冗余的奇偶校验位)
  001 1
  010 1
  100 1
  011 0
  100 1
  101 0
  110 0
  111 1

  共有8个有效的码字，你会发现两位不同，即海明距离为2，也就是需要改变两位才能变成另一个有效的码字，而奇偶校验只能检测一位错。(此例为偶校验，码字为奇校验的为无效码字)

检错和纠错

- 在任意两个有效码字间找出具有最小海明距离的两个码字，该海明距离便定义为全部码字的海明距离。
- 为了纠正d个比特的错，必须用距离为2d+1的编码。

  例如有4个有效码字：它们是0000000000，0000011111，1111100000，1111111111，海明距离为5，能纠正2比特错。

3.2 错误检测和纠正（3）

- 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。
- 要求设计仅纠正单比特错的纠错码

设计纠错码

- 要求：m个信息位，r个校验位，当r满足什么条件时，能纠正所有单比特错；
- 对2^m个合法报文的任何一个而言，有n个与该报文距离为1的无效码字，所以2^m个合法报文的每一个都对应有n+1个各不相同的位图，n位码字的总的位图是2^n个：

$(n+1)2^m <= 2^n$，n=m+r代入

$(m+r+1)2^m <= 2^{m+r}$

$2^r >= m+r+1$ ⇨ 纠正单比特误码的校验位下界r

3.2 错误检测和纠正（4）

- 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

3.2 错误检测和纠正（5）

- 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。例：若校验位1、2、8出错，则第11位变反。

3.2 错误检测和纠正（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。

海明编码举例

字符	ASCII	校验位
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

位传输的顺序

1	2	3	4	5	6	7	8	9	10	11
		1		1		1		1		1
		2			2	2			2	2
				4	4	4				
								8	8	8

K取不同值时，所对应的校验位。如，位置3由1、2位置的校验位校验，5由1、4校验…

上例中，m=7，r=4，n=11，显然2^4>=11+1，采用偶校验

3=1+2，5=1+4

6=2+4，7=1+2+4

9=1+8，10=2+8

11=1+2+8

校验位：1∈（3，5，7，9，11）

2 ∈（3，6，7，10，11）

4 ∈（5，6，7）

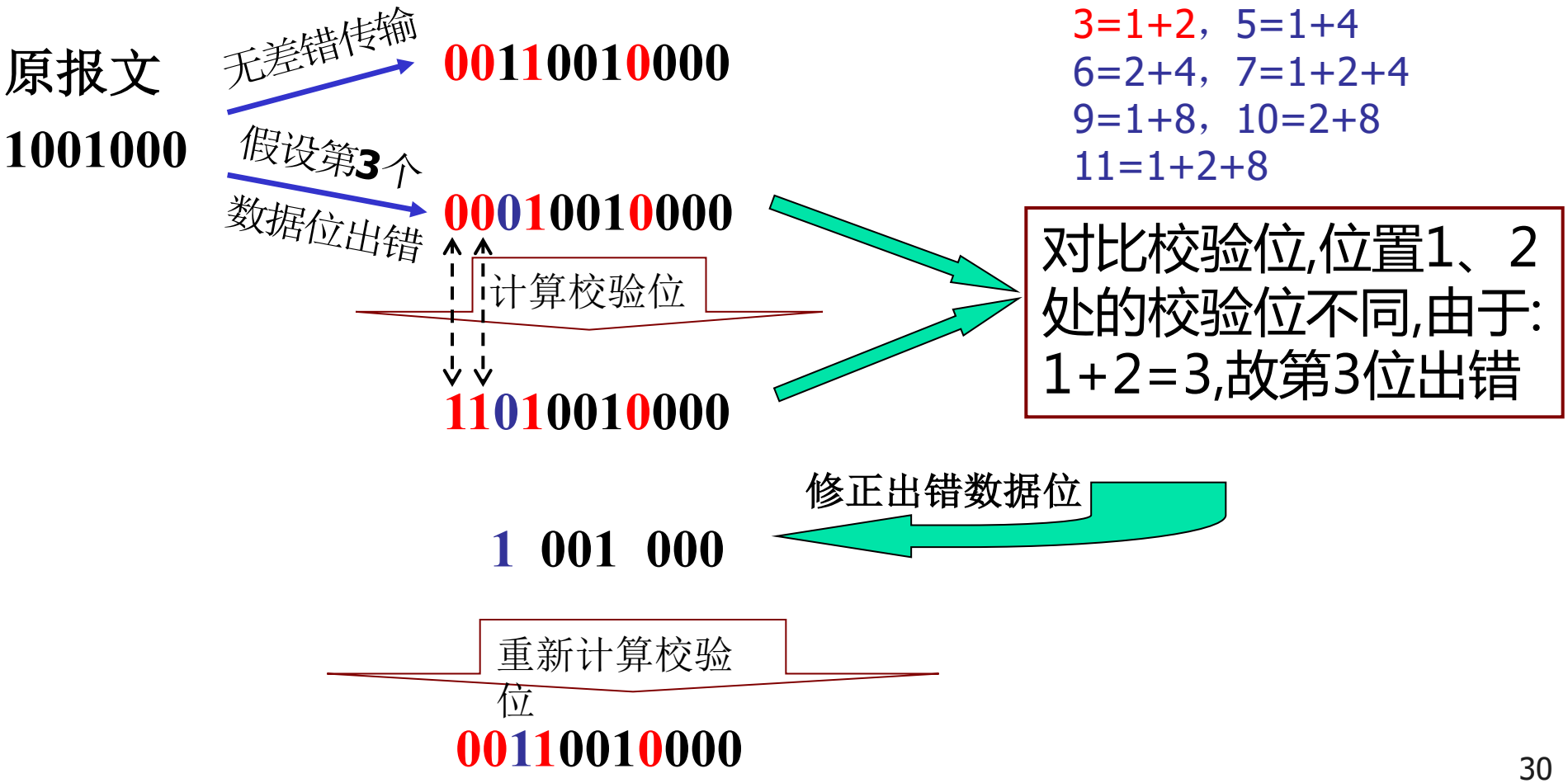
8 ∈（9，10，11）

能纠正单比特错！

例如，在接收方，如果校验位1不满足偶校验，而其他校验位都满足，则第1位出错，…

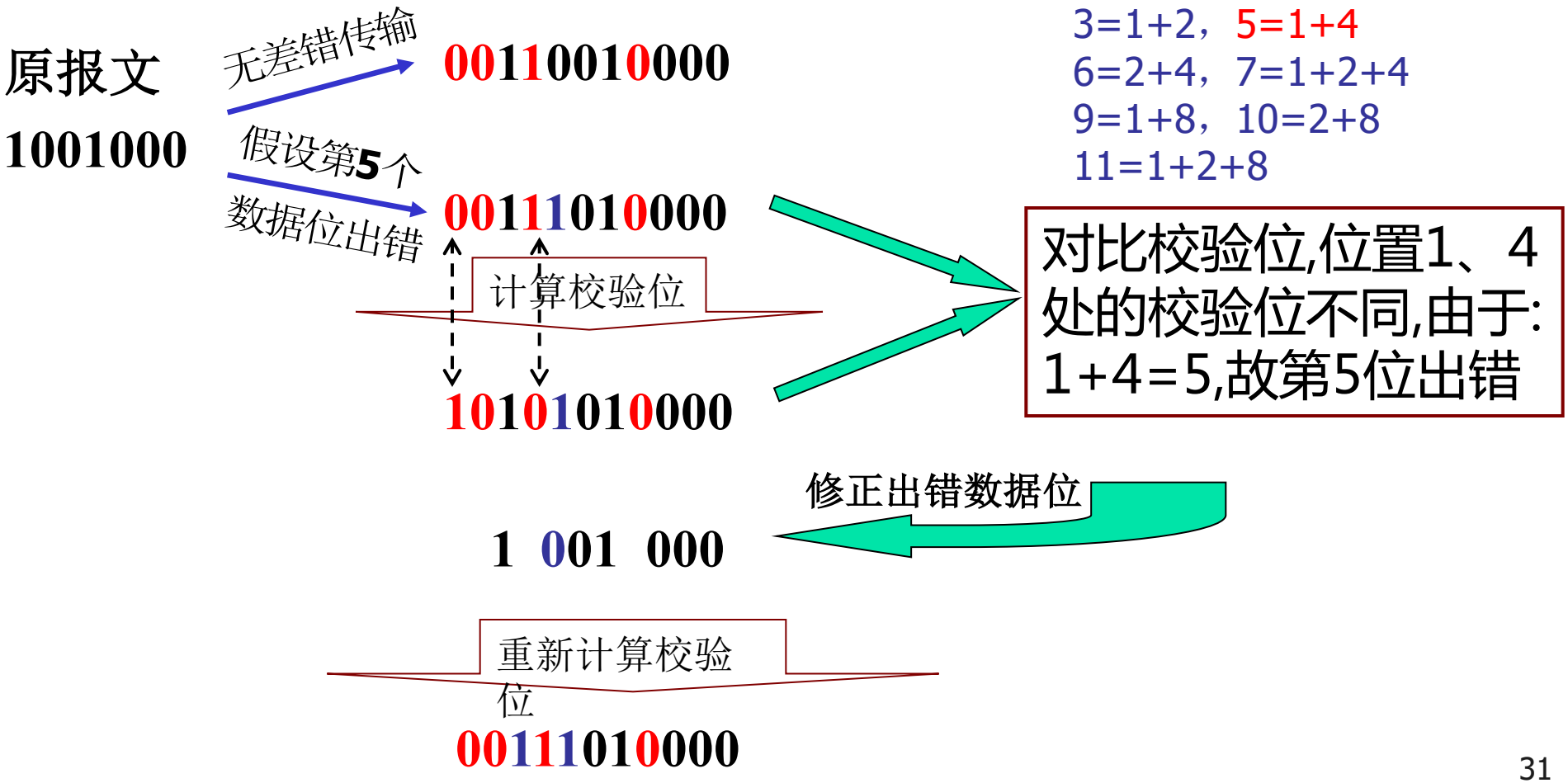
纠正码字中的单个数据位错误

1 ∈（3，5，7，9，11）
2 ∈（3，6，7，10，11）
4 ∈（5，6，7）
8 ∈（9，10，11）
3=1+2，5=1+4
6=2+4，7=1+2+4
9=1+8，10=2+8
11=1+2+8

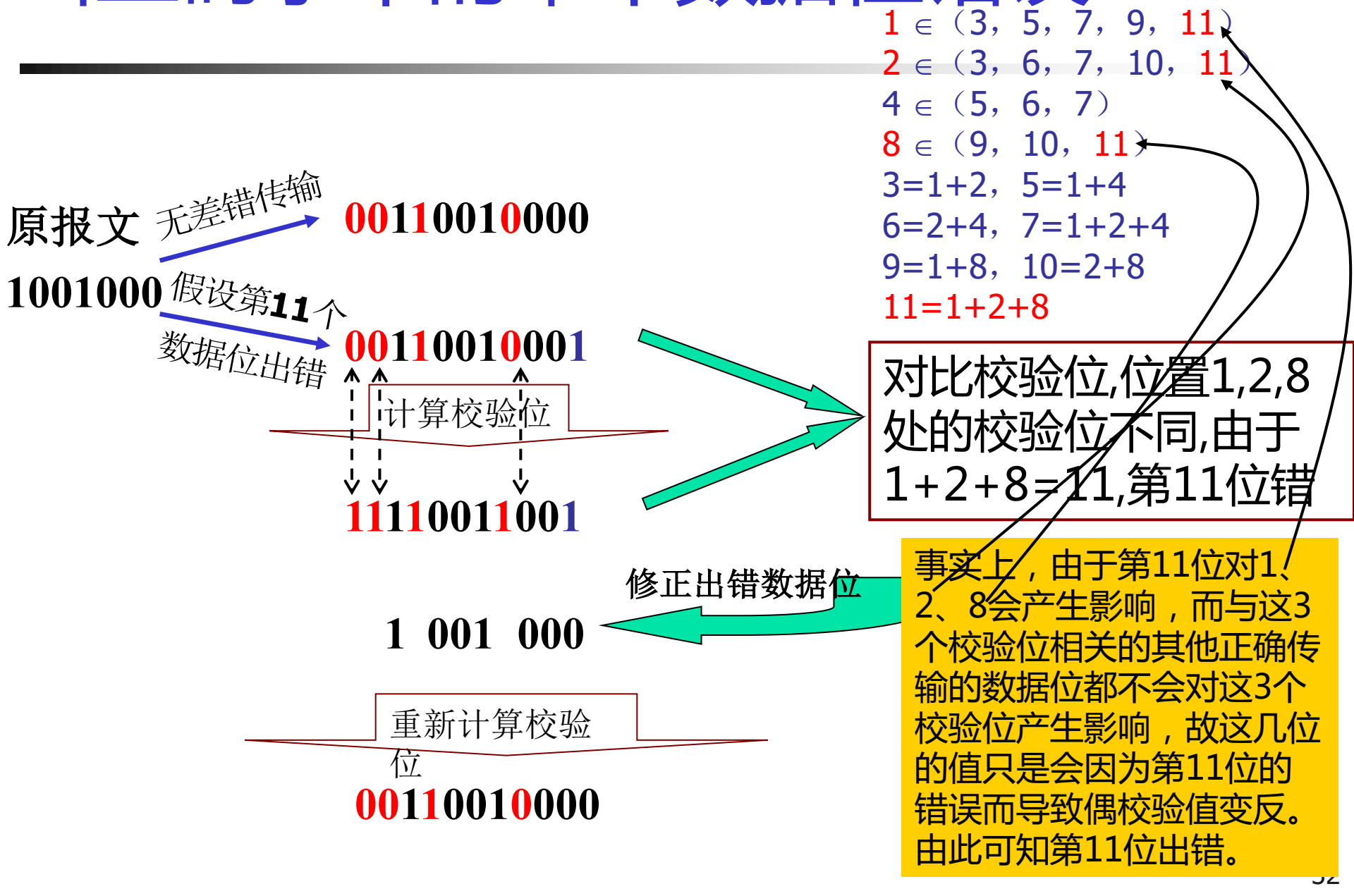


纠正码字中的单个数据位错误

1 ∈（3，5，7，9，11）
2 ∈（3，6，7，10，11）
4 ∈（5，6，7）
8 ∈（9，10，11）
3=1+2，5=1+4
6=2+4，7=1+2+4
9=1+8，10=2+8
11=1+2+8

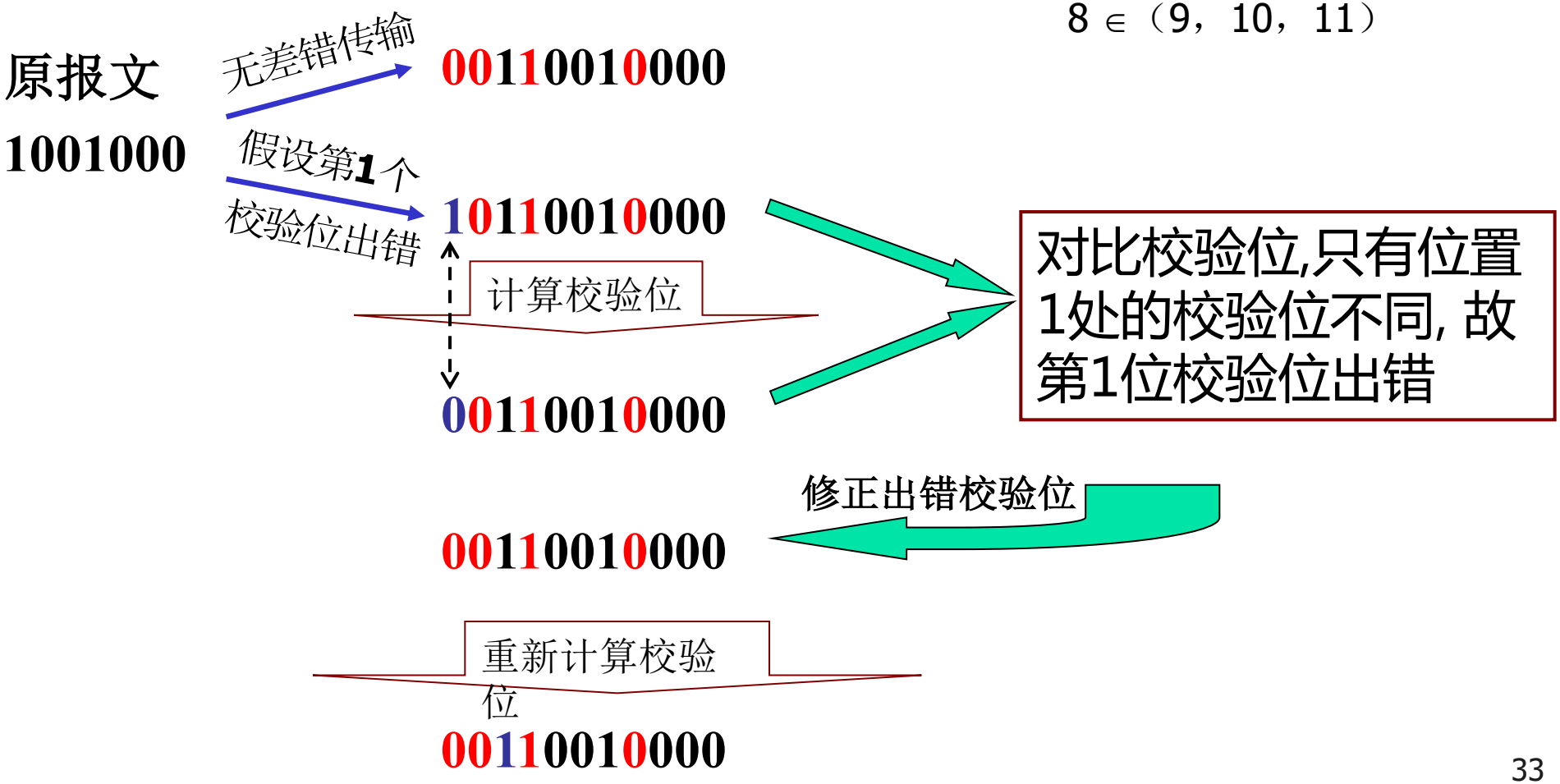


纠正码字中的单个数据位错误



纠正码字中的单个校验位错误

1 ∈（3，5，7，9，11）
2 ∈（3，6，7，10，11）
4 ∈（5，6，7）
8 ∈（9，10，11）

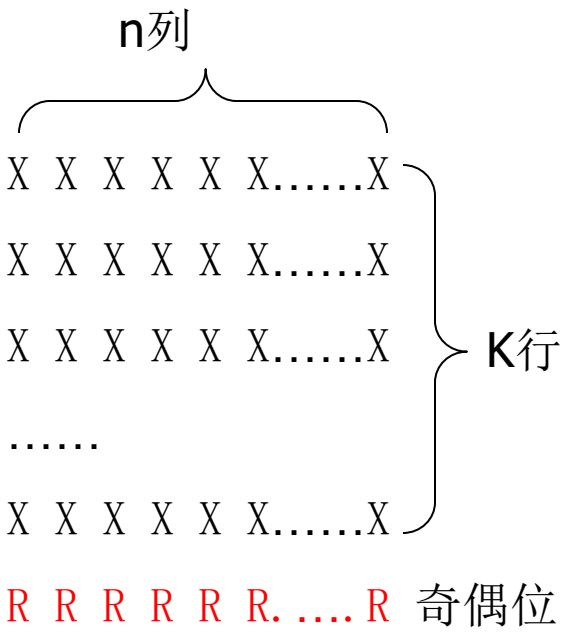


纠错码与检错码

- 在实际通信中使用纠错码好还是检错码好呢？
- 例如：假设一个信道误码率是10^{-6}，且出错是孤立产生的（即只有单比特错），数据块长度为1000比特，如果采用纠错编码，需要10个校验位（$2^{10}>1011$），传送1M数据需要10000个校验位；如果采用检错编码，每个数据块只需一个奇偶校验位，传送1M数据只需1000个校验位和一个重传的数据1001位，共需要2001比特的额外开销。
- 在多数通信中采用检错编码，但在单工信道中需要纠错编码。

改进的奇偶校验

- 将数据位组成一个n位宽，K位高的长方形距阵来发送，然后对每一列单独计算奇偶位，并附在最后一行作为冗余位。



检错率：

1.该方法可以检测长度为n的突发性错误，但不能检测长度为n+1的突发性错误。

2.假设n列中任意一列检测出错的概率为1/2，那么，整个数据块的错判率为（1/2）^{n}。

该方法用在ICMP报头检验中。

3.2 错误检测和纠正（7）

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。

3.2 错误检测和纠正（8）

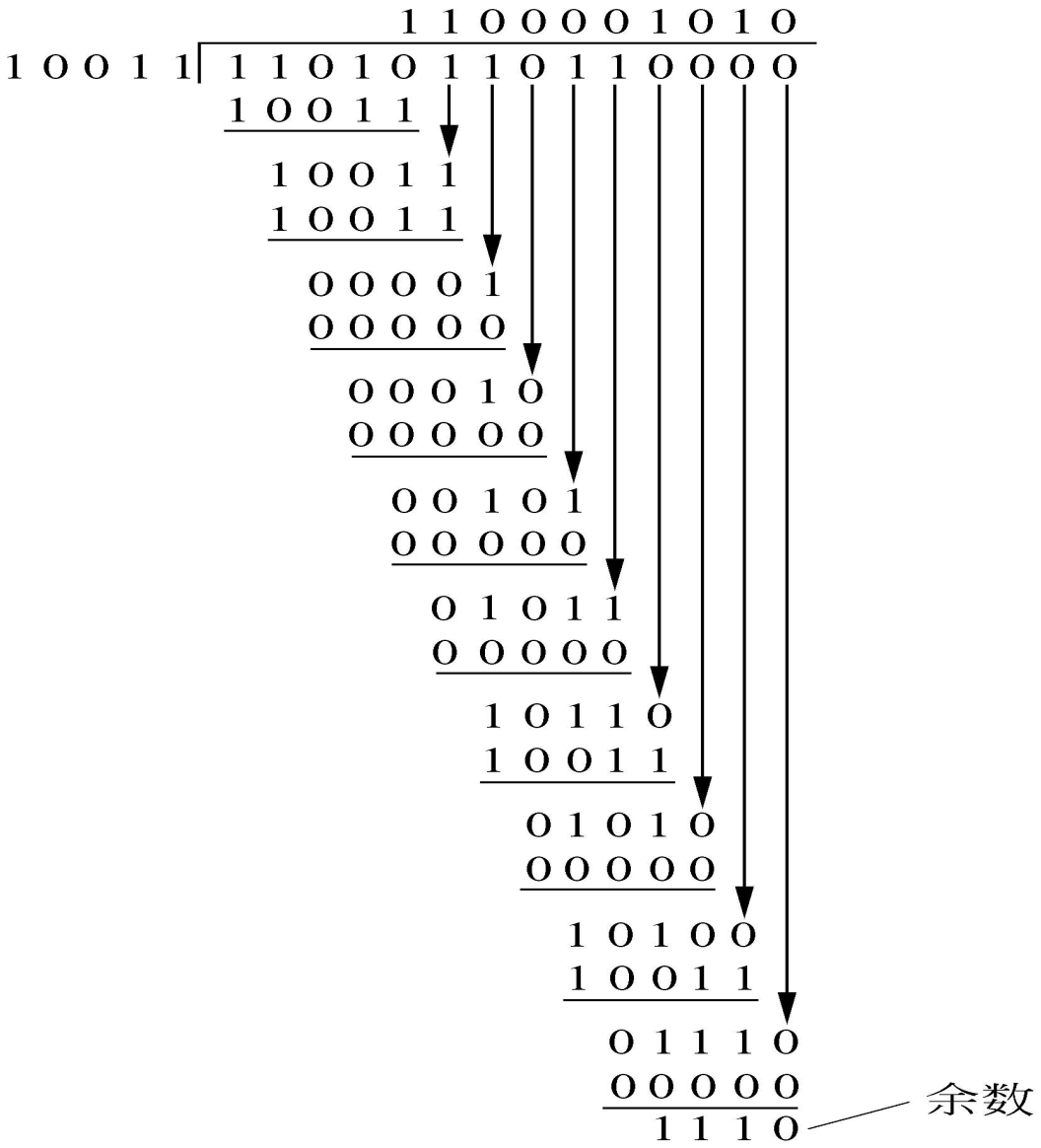
- CRC码基本思想
  - 校验和（checksum）加在帧尾，使带校验和的帧的多项式能被G(x)除尽；收方接收时，用G(x)去除它，若有余数，则传输出错。
- 校验和计算算法
  - 设G(x)为 r 阶，在帧的末尾加 r 个0，使帧为m+r位，相应多项式为$x^rM(x)$；
  - 按模2除法用对应于G(x)的位串去除对应于$x^rM(x)$的位串；
  - 按模2减法从对应于$x^rM(x)$的位串中减去余数(等于或小于r位)，结果就是要传送的带校验和的多项式T(x)。

多项式编码校验和计算示例

帧 ：1 1 0 1 0 1 1 0 1 1 (**M(X)**)

除数 ：1 0 0 1 1 (**G(X)**)

附加 4 个零后形成的串 ：1 1 0 1 0 1 1 0 1 1 0 0 0 0 (**M(X)*X^r**)



传输的帧 ：1 1 0 1 0 1 1 0 1 1 1 1 1 0 (**T(X)**)

3.2 错误检测和纠正（9）

- CRC的检错能力
  - 发送：T(x)；接收：T(x) + E(x), E(x) ≠ 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。

3.2 错误检测和纠正（10）

- CRC检错能力的几种情况分析
  - 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$E(x) = x^i + x^j = x^j(x^{i-j} + 1)$，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$(k \leq i - j)$；
  - 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；

3.2 错误检测和纠正（11）

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

帧检验序列 FCS

- 在数据后面添加上的冗余码称为帧检验序列 FCS (Frame Check Sequence)。
- 循环冗余检验 CRC 和帧检验序列 FCS并不等同。
  - CRC 是一种常用的检错方法，而 FCS 是添加在数据后面的冗余码。
  - FCS 可以用 CRC 这种方法得出，但 CRC 并非用来获得 FCS 的惟一方法。

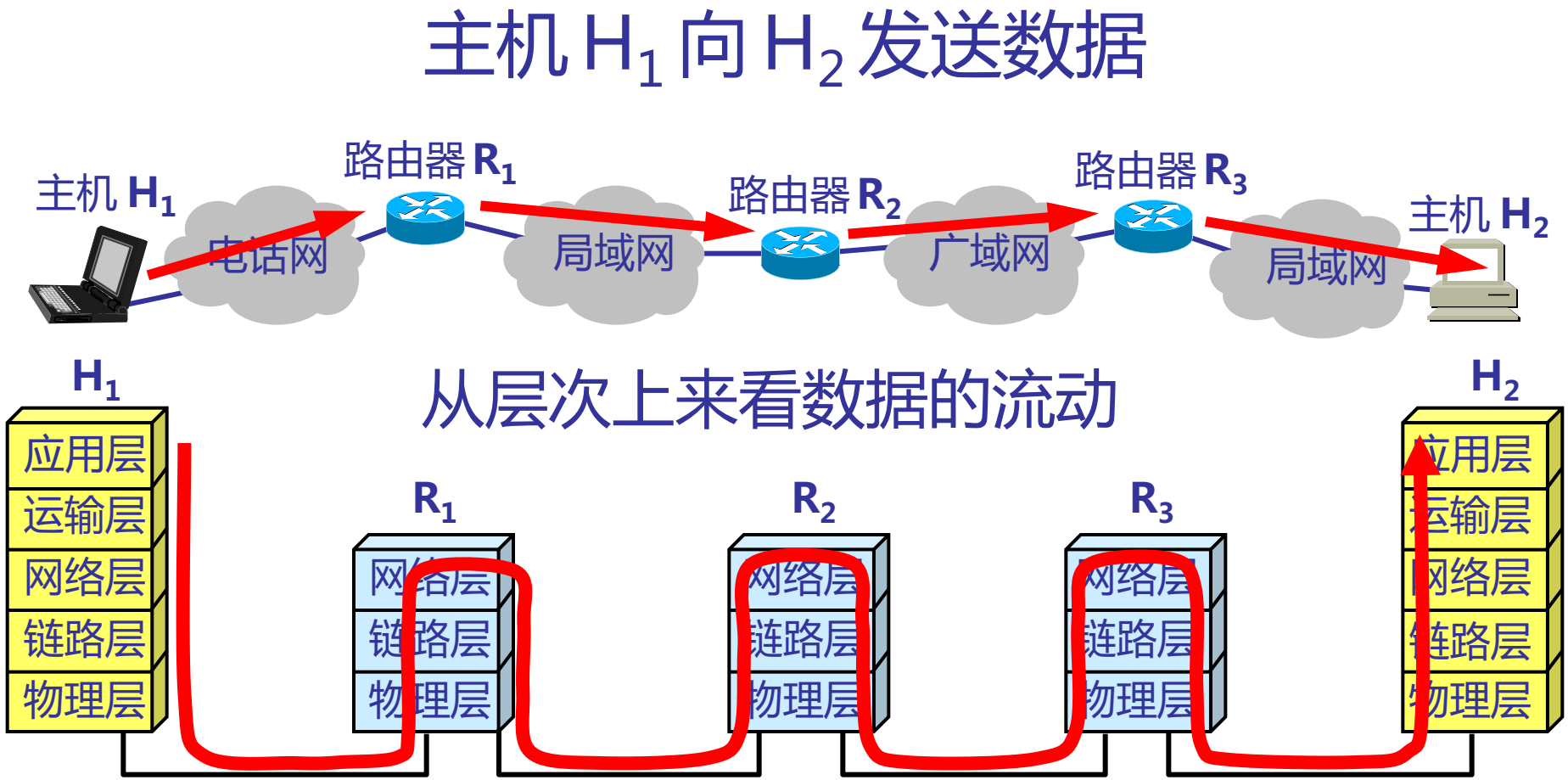
检测出差错

- 只要得出的余数R 不为0，就表示检测到了差错。
- 但这种检测方法并不能确定究竟是哪一个或哪几个比特出现了差错。
- 一旦检测出差错，就丢弃这个出现差错的帧。
- 只要经过严格的挑选，并使用位数足够多的除数P，那么出现检测不到的差错的概率就很小很小。

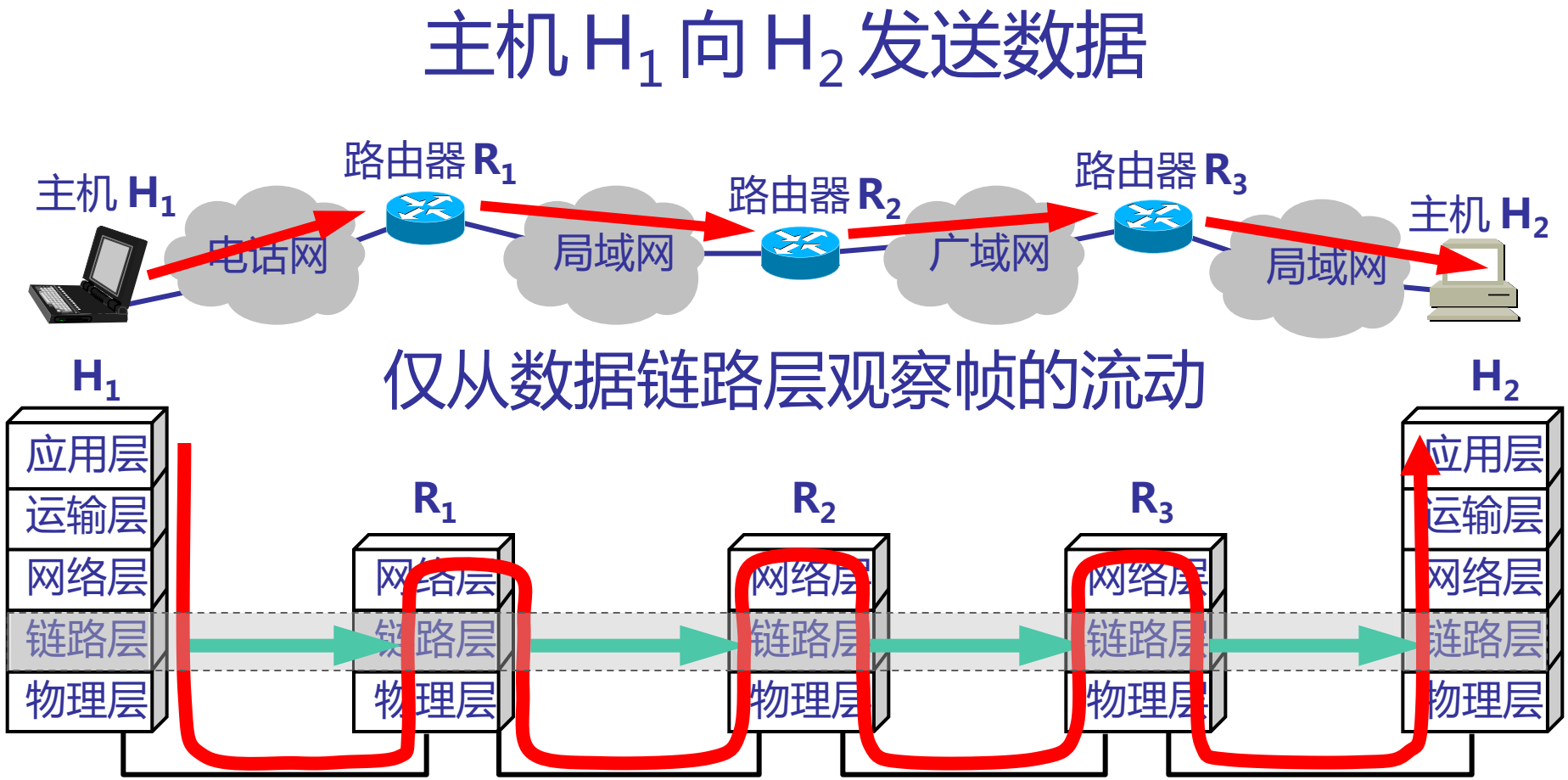
应当注意

- 仅用循环冗余检验 CRC 差错检测技术只能做到无差错接受(accept)。
- “无差错接受”是指：“凡是接受的帧（即不包括丢弃的帧），我们都能以非常接近于 1 的概率认为这些帧在传输过程中没有产生差错”。
- 也就是说：“凡是接受的帧都没有传输差错”（有差错的帧就丢弃而不接受）。
- 要做到“可靠传输”（即发送什么就收到什么）就必须再加上确认和重传机制。

数据链路层的简单模型


主机 H1 向 H2 发送数据
主机 H1
路由器 R1
路由器 R2
路由器 R3
主机 H2
电话网
局域网
广域网
局域网
从层次上来看数据的流动
H1
R1
R2
R3
H2
应用层
运输层
网络层
链路层
物理层

数据链路层的简单模型(续)

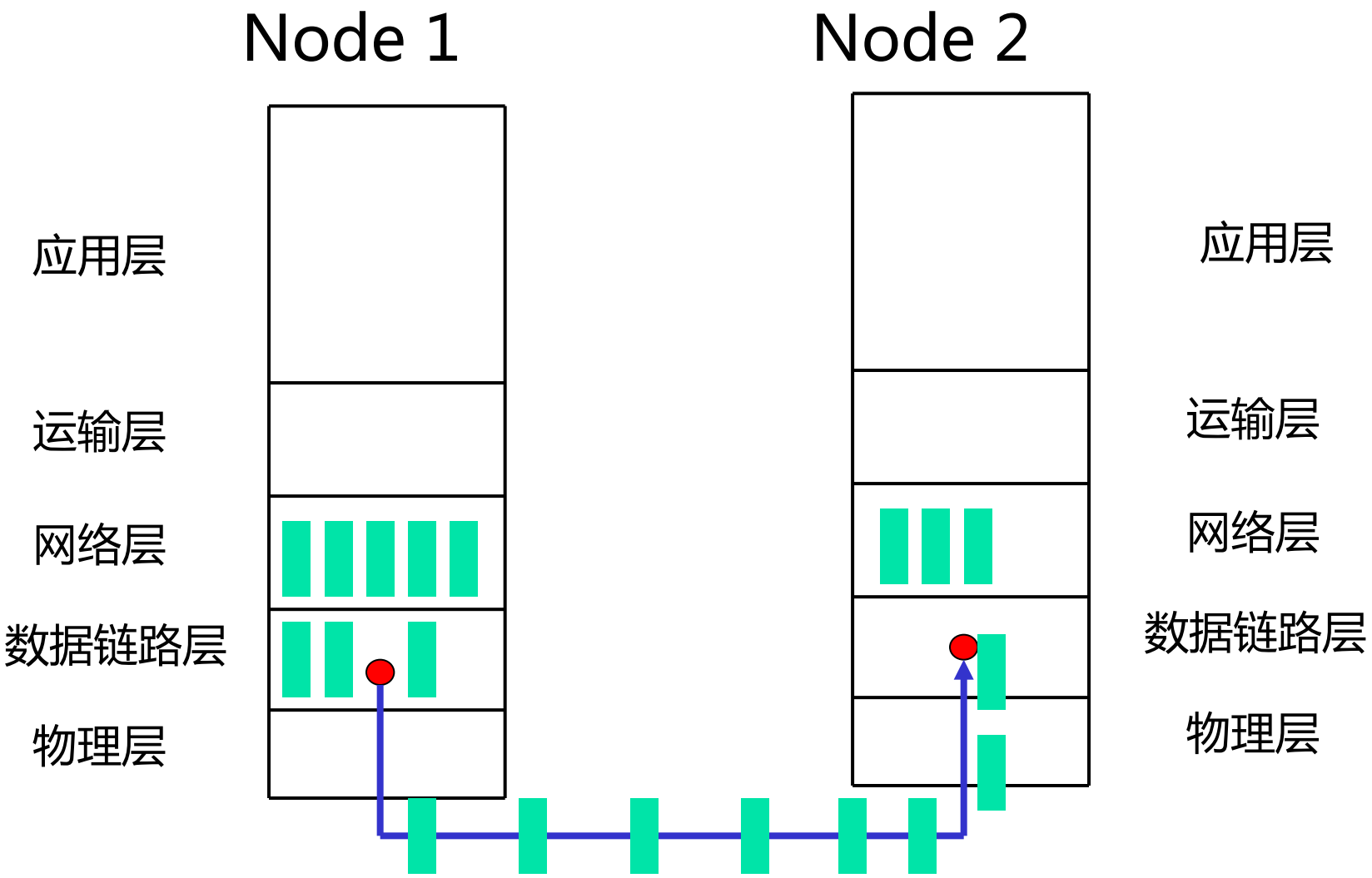


3.3 基本的数据链路层协议（1）

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

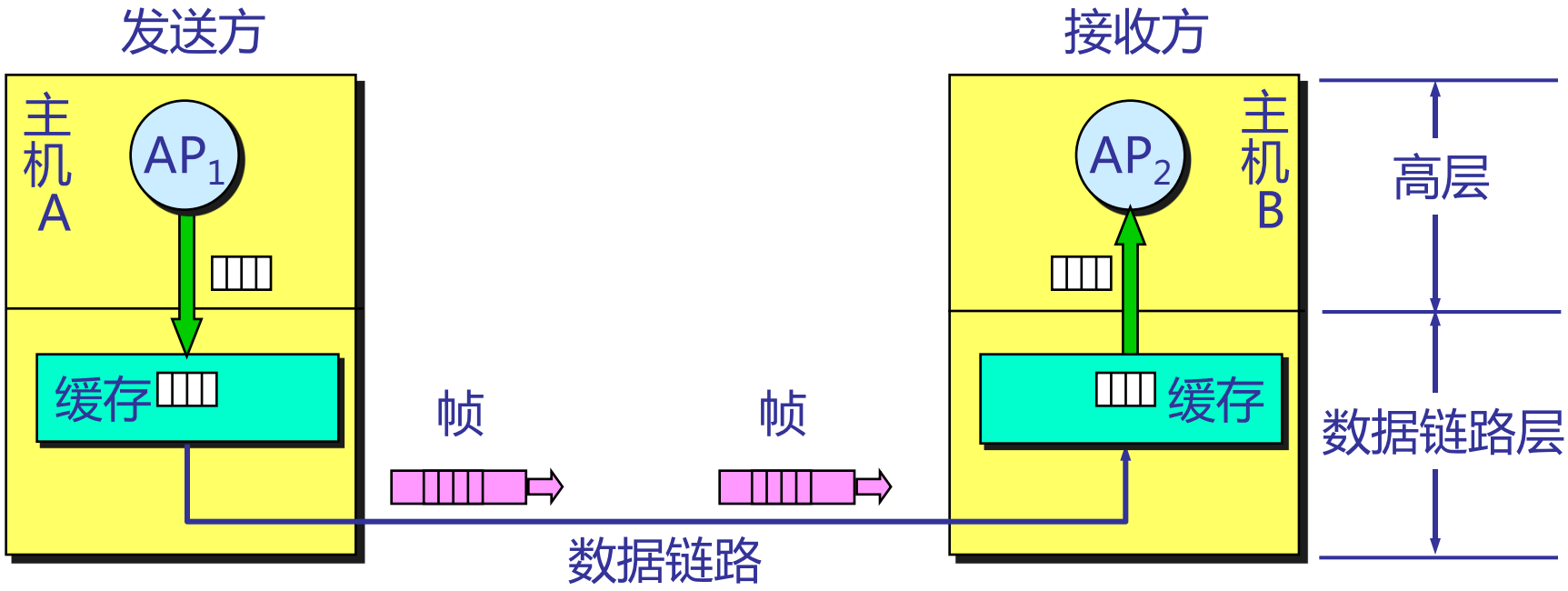
无约束单工协议



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

图3-12 A utopian simplex protocol.

无约束单工协议是一种完全理想化的数据传输



完全理想化的数据传输:基于两个假定

- 假定 1： 链路是理想的传输信道，所传送的任何数据既不会出差错也不会丢失。
- 假定 2： 不管发方以多快的速率发送数据，收方总是来得及收下，并及时上交主机。
  - 这个假定就相当于认为：接收端向主机交付数据的速率永远不会低于发送端发送数据的速率。

具有最简单流量控制的数据链路层协议

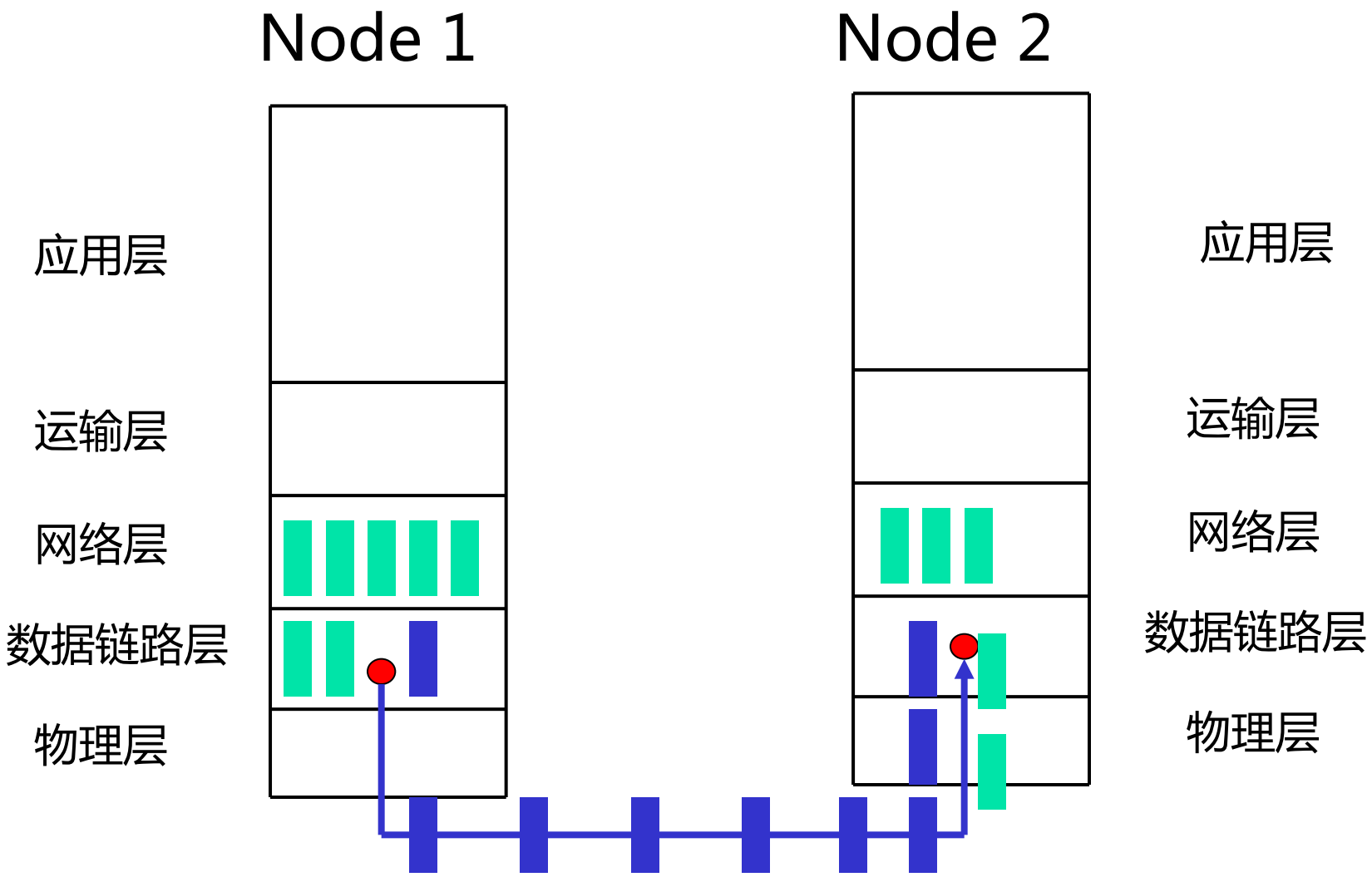
- 现在去掉上述的第二个假定。但是，仍然保留第一个假定，即主机 A 向主机 B传输数据的信道仍然是无差错的理想信道。然而现在不能保证接收端向主机交付数据的速率永远不低于发送端发送数据的速率。
- 由收方控制发方的数据流，乃是计算机网络中流量控制的一个基本方法。

3.3 基本的数据链路层协议（2）

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。

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 */
 }
}
```

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

具有最简单流量控制的数据链路层协议算法

在发送结点：

(1) 从主机取一个数据帧。

(2) 将数据帧送到数据链路层的发送缓存。

(3) 将发送缓存中的数据帧发送出去。

(4) 等待。

(5) 若收到由接收结点发过来的信息(此信息的格式与内容可由双方事先商定好)，则从主机取一个新的数据帧，然后转到(2)。

具有最简单流量控制的数据链路层协议算法（续）

在接收结点：

(1) 等待。

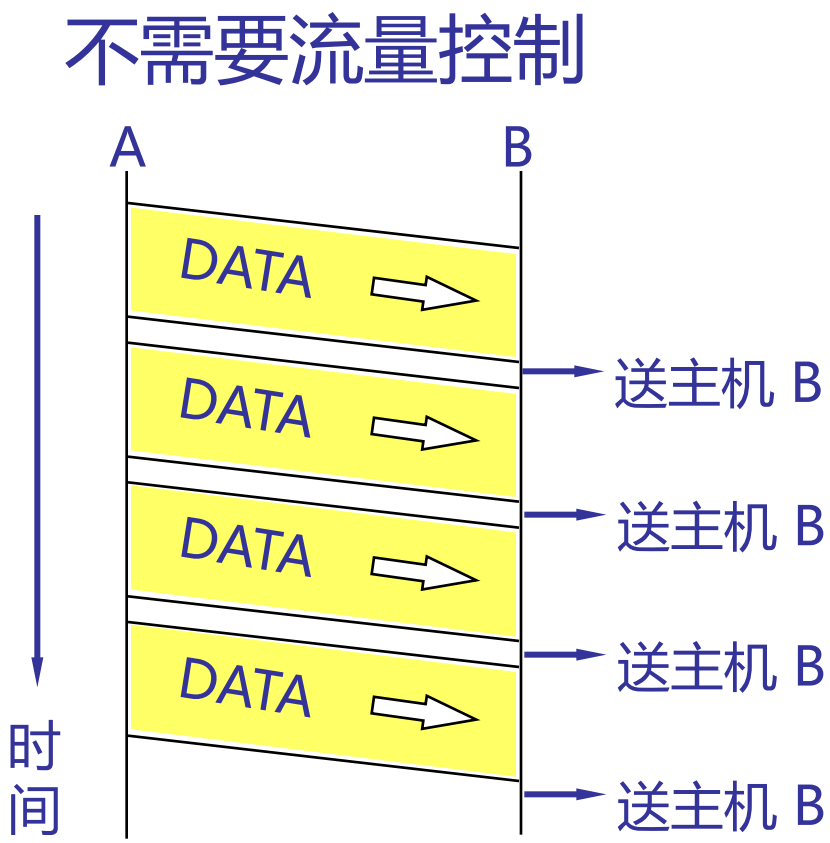
(2) 若收到由发送结点发过来的数据帧，则将其放入数据链路层的接收缓存。

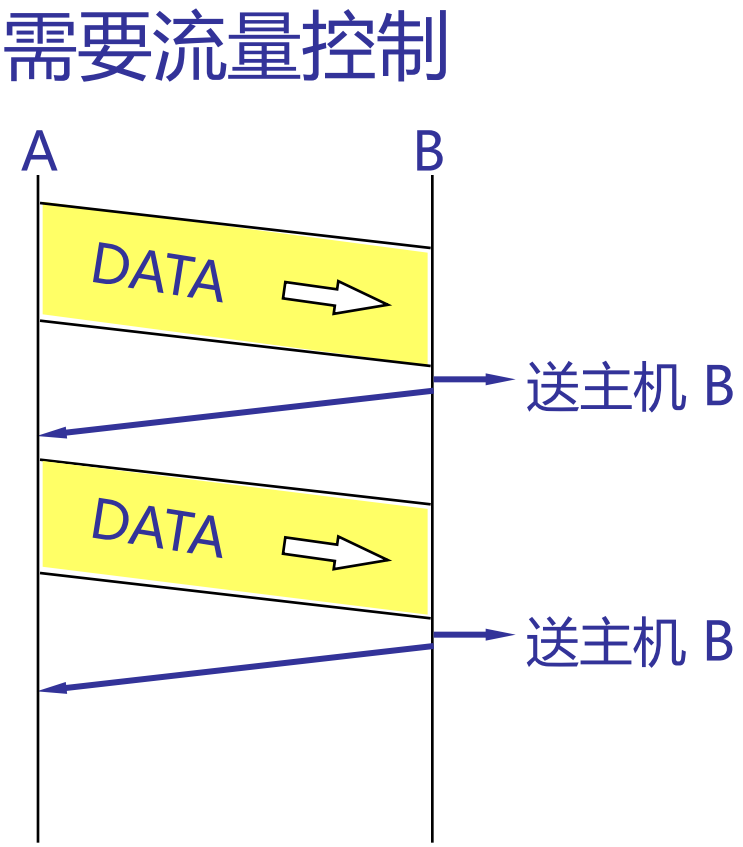
(3) 将接收缓存中的数据帧上交主机。

(4) 向发送结点发一信息，表示数据帧已经上交给主机。

(5) 转到(1)。

两种情况的对比（传输均无差错）





3.3 基本的数据链路层协议（3）

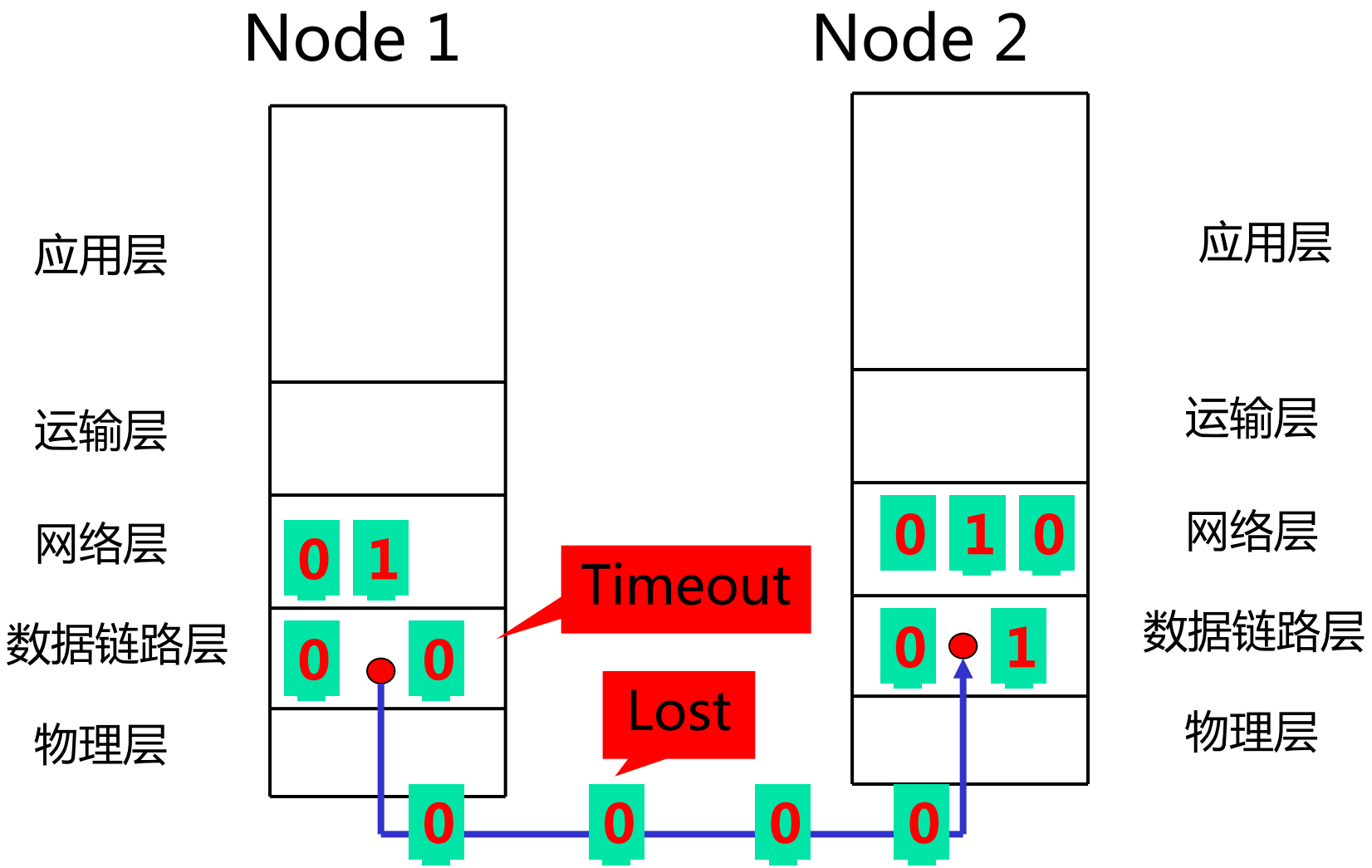
3. 有噪声信道的单工协议(A Simplex Protocol for a Noisy Channel)

- 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

3.3 基本的数据链路层协议（4）

- 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）
- 工作过程

有噪声信道的单工协议

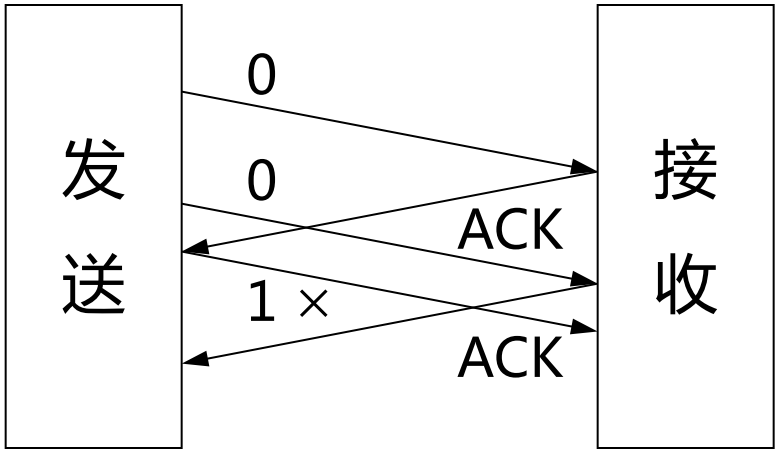

Node 1
Node 2
应用层
运输层
网络层
数据链路层
物理层
应用层
运输层
网络层
数据链路层
物理层
0
1
0
0
0
1
0
0
Timeout
Lost
0
0
0
0

```
/* 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);        /* none of the fields are used */
      }
  }
}
```

图3-14 A positive acknowledgement with retransmission protocol

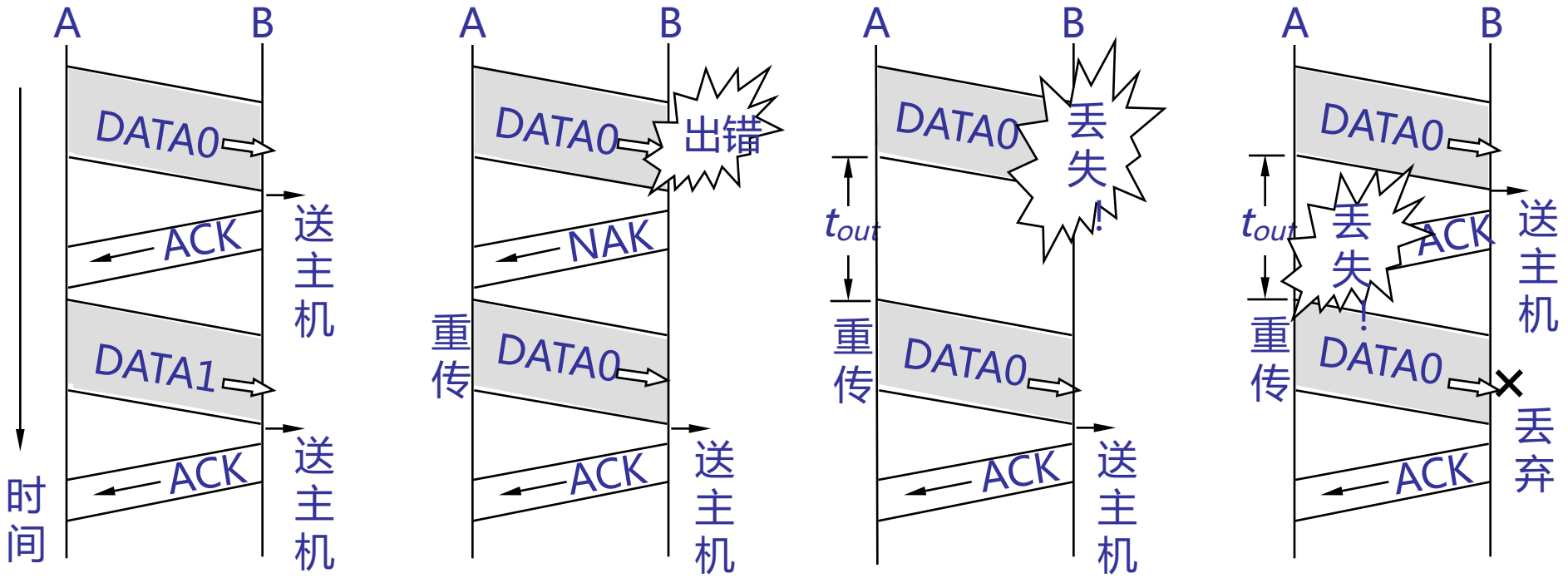
3.3 基本的数据链路层协议（5）

- 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亦称为实用的停止等待协议

有四种情况



(a) 正常情况　(b) 数据帧出错　(c) 数据帧丢失　(d) 确认帧丢失

超时计时器的作用

- 结点A发送完一个数据帧时，就启动一个超时计时器(timeout timer)。
  - 计时器又称为定时器。
- 若到了超时计时器所设置的重传时间 t_{out}而仍收不到结点 B 的任何确认帧，则结点 A 就重传前面所发送的这一数据帧。
- 一般可将重传时间选为略大于“从发完数据帧到收到确认帧所需的平均时间”。

解决重复帧的问题

- 使每一个数据帧带上不同的发送序号。每发送一个新的数据帧就把它的发送序号加 1。
- 若结点 B 收到发送序号相同的数据帧，就表明出现了重复帧。这时应丢弃重复帧，因为已经收到过同样的数据帧并且也交给了主机 B。
- 但此时结点 B 还必须向 A 发送确认帧 ACK，因为 B 已经知道 A 还没有收到上一次发过去的确认帧 ACK。

帧的编号问题

- 任何一个编号系统的序号所占用的比特数一定是有限的。因此，经过一段时间后，发送序号就会重复。
- 序号占用的比特数越少，数据传输的额外开销就越小。
- 对于停止等待协议，由于每发送一个数据帧就停止等待，因此用一个比特来编号就够了。
  - 一个比特可表示 0 和 1 两种不同的序号。

帧的发送序号

- 数据帧中的发送序号 seq以 0 和 1 交替的方式出现在数据帧中。
- 每发一个新的数据帧，发送序号就和上次发送的不一样。用这样的方法就可以使收方能够区分开新的数据帧和重传的数据帧了。

可靠传输

- 虽然物理层在传输比特时会出现差错，但由于数据链路层的停止等待协议采用了有效的检错重传机制，数据链路层对上面的网络层就可以提供可靠传输的服务。

停止等待协议的算法

- 通常不使用否认帧（实用的数据链路层协议大都是这样的），而且确认帧带有序号 *n*。
- 按照习惯的表示法，ACK*n* 表示“第 *n* – 1 号帧已经收到，现在期望接收第 *n* 号帧”。
  - ACK1 表示“0 号帧已收到，现在期望接收的下一帧是 1 号帧”；
  - ACK0 表示“1 号帧已收到，现在期望接收的下一帧是 0 号帧”。

在发送结点

(1) 从主机取一个数据帧，送交发送缓存。

(2) next_frame_to_send←0。

(3) seq← next_frame_to_send 。

next_frame_to_send：发送状态变量

seq: 发送帧的序号, ack: 确认帧序号

(4) 将发送缓存中的数据帧发送出去。

(5) 设置超时计时器。

(6) 等待。 {等待以下(7)和(8)这两个事件中最先出现的一个}

(7) 收到确认帧 ACK，

若 *ack* = next_frame_to_send，则：

从主机取一个新的数据帧，放入发送缓存；

inc(next_frame_to_send)，转到 (3)。

否则，丢弃这个确认帧，转到(4)。

(8) 若超时计时器时间到，则转到(4)。

在接收结点

Frame_expected:接收状态变量

seq:接收帧的序号

(1) frame_expected←0。

(2) 等待。

(3) 收到一个数据帧，用CRC检验有无错误；

若 seq = frame_expected ，则执行(4)；

否则(传输有错误或重复帧)丢弃此数据帧,然后转到(6)。

(4) 将收到的数据帧中的数据部分送交上层软件

（也就是数据链路层模型中的主机）。

(5) inc(frame_expected)。

(6) *ack* ← 1 – frame_expected ；

发送确认帧 ACK，转到(2)。

停止等待协议的要点

- 只有收到序号正确的确认帧 ACK后，才更新发送状态变量 next_frame_to_send一次，并发送新的数据帧。
- 接收端接收到数据帧时，就要将接收序号seq与本地的接收状态变量frame_expected相比较。
  - 若二者相等就表明是新的数据帧，就收下，并发送确认。
  - 否则为重复帧，就必须丢弃。但这时仍须向发送端发送确认帧 ACK，而接收状态变量frame_expected和确认序号 *ack* 都不变。

停止等待协议的要点（续）

- 连续出现相同发送序号的数据帧，表明发送端进行了超时重传。连续出现相同序号的确认帧，表明接收端收到了重复帧。
- 发送端在发送完数据帧时，必须在其发送缓存中暂时保留这个数据帧的副本。这样才能在出差错时进行重传。只有确认对方已经收到这个数据帧时，才可以清除这个副本。

停止等待协议的要点（续）

- 实用的CRC 检验器都是用硬件完成的。
- CRC 检验器能够自动丢弃检测到的出错帧。因此所谓的“丢弃出错帧”，对上层软件或用户来说都是感觉不到的。
- 发送端对出错的数据帧进行重传是自动进行的，因而这种差错控制体制常简称为 ARQ (Automatic Repeat reQuest)，直译是自动重传请求，但意思是自动请求重传。

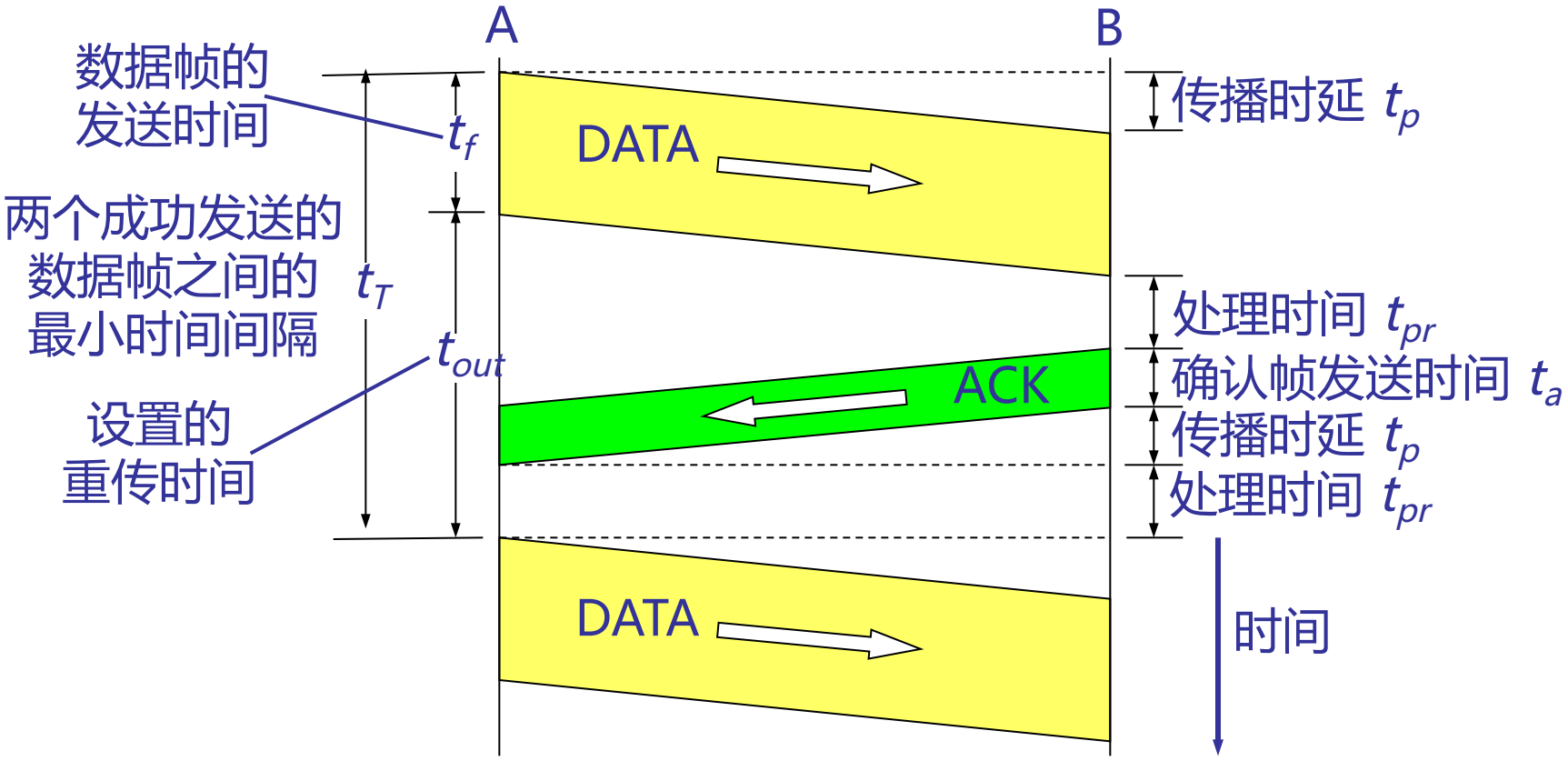
停止等待协议的定量分析

- 设 t_f 是一个数据帧的发送时间，且数据帧的长度是固定不变的。显然，数据帧的发送时间 t_f 是数据帧的长度 l_f (bit) 与数据的发送速率 C (bit/s)之比，即

$$t_f = l_f / C = l_f / C \text{ (s)} \qquad (3\text{-}1)$$

- 发送时间 t_f 也就是数据帧的发送时延。
- 数据帧沿链路传到结点B还要经历一个传播时延 t_p。
- 结点 B 收到数据帧要花费时间进行处理，此时间称为处理时间 t_{pr}，发送确认帧 ACK 的发送时间为 t_a。

停止等待协议中
数据帧和确认帧的发送时间关系



重传时间

- 重传时间的作用是：数据帧发送完毕后若经过了这样长的时间还没有收到确认帧，就重传这个数据帧。
- 为方便起见，我们设重传时间为

$$t_{out} = t_p + t_{pr} + t_a + t_p + t_{pr}$$

- 设上式右端的处理时间 t_{pr} 和确认帧的发送时间 t_a 都远小于传播时延 t_p，因此可将重传时间取为两倍的传播时延，即

$$t_{out} = 2t_p$$

简单的数学分析

- 两个发送成功的数据帧之间的最小时间间隔是

$$t_T = t_f + t_{out} = t_f + 2t_p$$

- 设数据帧出现差错(包括帧丢失)的概率为 p，但假设确认帧不会出现差错。
- 设正确传送一个数据帧所需的平均时间 t_{av}

$$t_{av} = t_T(1 + \text{一个帧的平均重传次数})$$

简单的数学分析（续）

一帧的平均重传次数

= {1 × **P**[重传次数为 1] + 2 × **P**[重传次数为 2] + 3 × **P**[重传次数为 3] +...}

= {1 × **P**[第 1 次发送出错] × **P**[第 2 次发送成功]

+ 2 × **P**[第 1, 2 次发送出错] × **P**[第 3 次发送成功]

+ 3 × **P**[第 1, 2, 3 次发送出错] × **P**[第 4 次发送成功] +...}

$= p(1-p) + 2p^2(1-p) + 3p^3(1-p) + \ldots$

这里 **P[X]** 是出现事件 **X** 的概率。

简单的数学分析（续）

得出正确传送一个数据帧所需的平均时间：

$$t_{av} = t_T + (1-p)\sum_{i=1}^{\infty} ip^i t_T = t_T/(1-p)$$

当传输差错率增大时，t_{av}也随之增大。当无差错时，

$p = 0$, $t_{av} = t_T$。

简单的数学分析（续）

每秒成功发送的最大帧数就是链路的最大吞吐量 λ_{max}。显然，

$$\lambda_{max} = 1/t_{av} = (1 - p) / t_T$$

在发送端，设数据帧的实际到达率为λ，则λ不应超过最大吞吐量 λ_{max}，即

$$\lambda \le (1 - p) / t_T$$

用时间 t_f 进行归一化，得出归一化的吞吐量 ρ 为

$$\rho \equiv \lambda t_f \le (1 - p) / \alpha < 1$$

其中参数 α 是 t_T 的归一化时间：

$$\alpha \equiv t_T / t_f \ge 1$$

当重传时间远小于发送时间时，$\alpha \approx 1$，此时的归一化吞吐量

$$\rho \le 1 - p$$

停止等待协议 ARQ 的优缺点

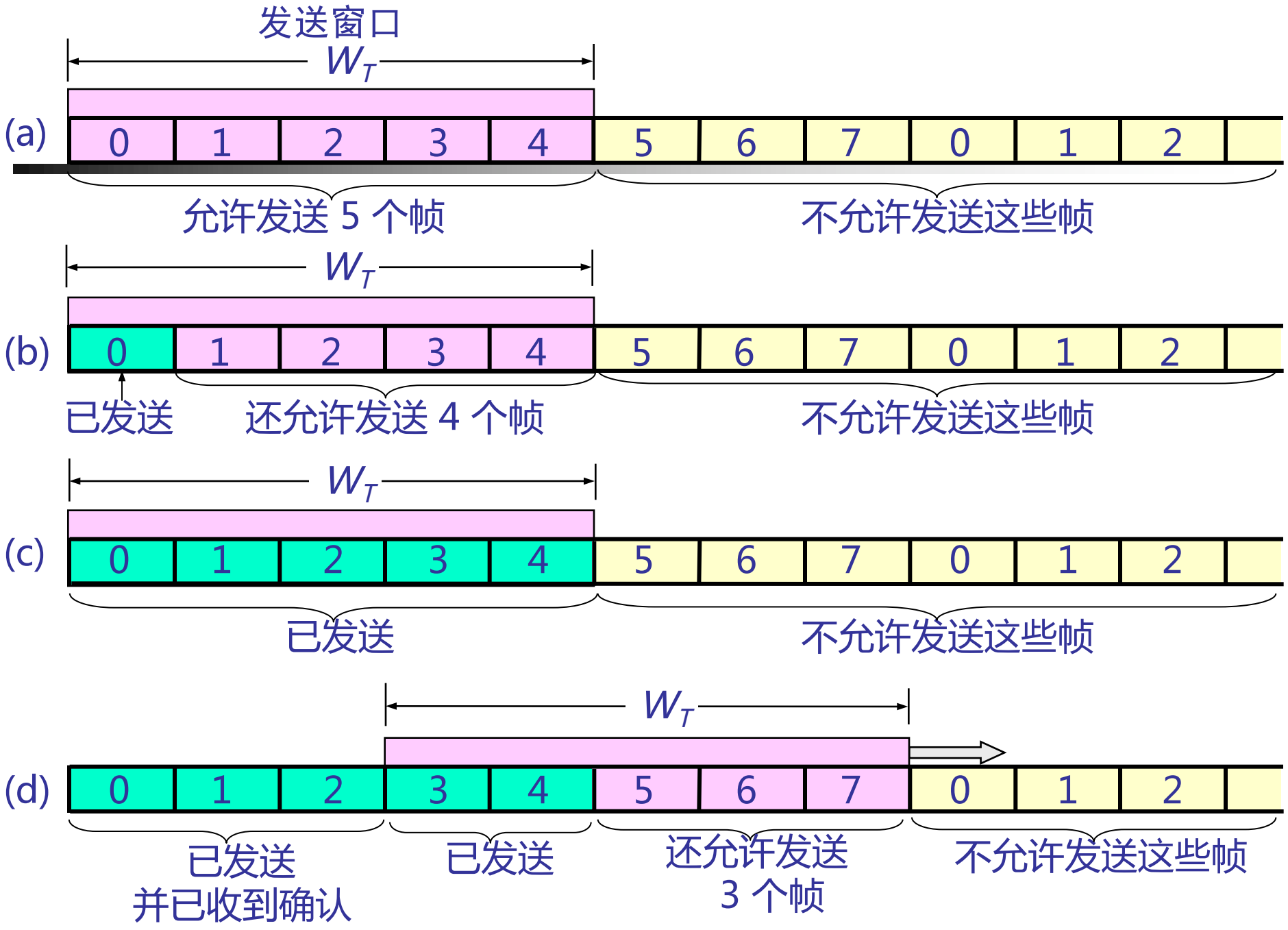
- 优点：比较简单 。
- 缺点：通信信道的利用率不高，也就是说，信道还远远没有被数据比特填满。
- 为了克服这一缺点，就产生了另外两种协议，即连续 ARQ(亦称退后n帧协议)和选择重传 ARQ (即选择重传协议)。这将在后面进一步讨论。

连续 ARQ 协议的工作原理

- 在发送完一个数据帧后，不是停下来等待确认帧，而是可以连续再发送若干个数据帧。
- 如果这时收到了接收端发来的确认帧，那么还可以接着发送数据帧。
- 由于减少了等待时间，整个通信的吞吐量就提高了。

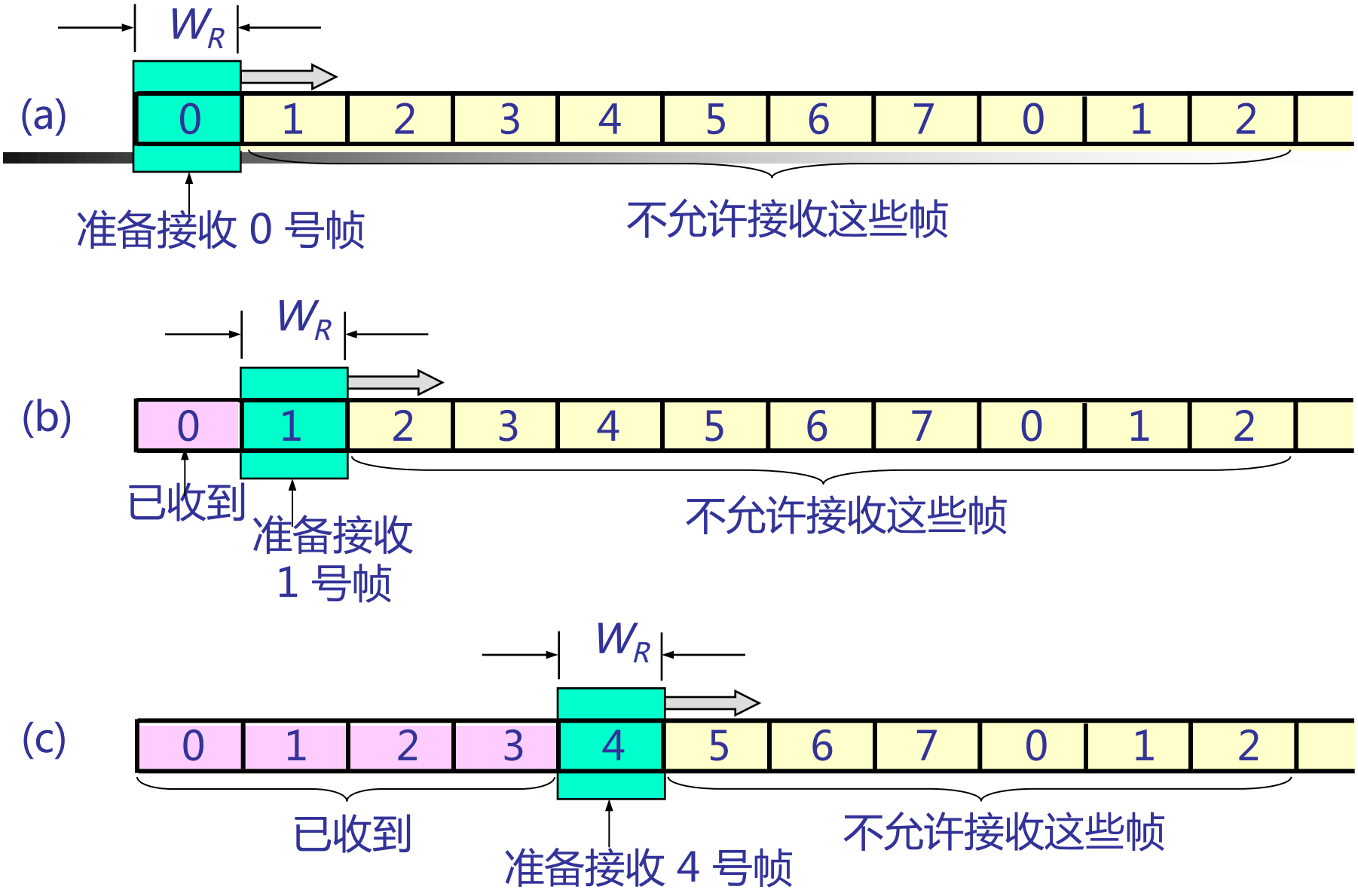
滑动窗口的引入

- 发送端和接收端分别设定发送窗口和接收窗口 。
- 发送窗口用来对发送端进行流量控制。
- 发送窗口的大小 W_T 代表在还没有收到对方确认信息的情况下发送端最多可以发送多少个数据帧。


发送窗口
W_T
(a)
0 1 2 3 4 5 6 7 0 1 2
允许发送 5 个帧
不允许发送这些帧
W_T
(b)
0 1 2 3 4 5 6 7 0 1 2
已发送
还允许发送 4 个帧
不允许发送这些帧
W_T
(c)
0 1 2 3 4 5 6 7 0 1 2
已发送
不允许发送这些帧
W_T
(d)
0 1 2 3 4 5 6 7 0 1 2
已发送
并已收到确认
已发送
还允许发送
3 个帧
不允许发送这些帧

接收端设置接收窗口

- 在接收端只有当收到的数据帧的发送序号落入接收窗口内才允许将该数据帧收下。
- 若接收到的数据帧落在接收窗口之外，则一律将其丢弃。
- 在连续 ARQ 协议中，接收窗口的大小 $W_R = 1$。
  - 只有当收到的帧的序号与接收窗口一致时才能接收该帧。否则，就丢弃它。
  - 每收到一个序号正确的帧，接收窗口就向前（即向右方）滑动一个帧的位置。同时发送对该帧的确认。


W_R
(a)
0 1 2 3 4 5 6 7 0 1 2
准备接收 0 号帧
不允许接收这些帧
W_R
(b)
0 1 2 3 4 5 6 7 0 1 2
已收到
准备接收
1 号帧
不允许接收这些帧
W_R
(c)
0 1 2 3 4 5 6 7 0 1 2
已收到
准备接收 4 号帧
不允许接收这些帧

滑动窗口的重要特性

- 只有在接收窗口向前滑动时（与此同时也发送了确认），发送窗口才有可能向前滑动。
- 收发两端的窗口按照以上规律不断地向前滑动，因此这种协议又称为滑动窗口协议。
- 当发送窗口和接收窗口的大小都等于 1时，就是停止等待协议。

发送窗口的最大值

- 当用 n 个比特进行编号时，若接收窗口的大小为 1，则只有在发送窗口的大小 $W_T \leq 2^n - 1$时，连续 ARQ 协议才能正确运行。
- 例如，当采用 3 bit 编码时，发送窗口的最大值是 7 而不是 8。
- Why?

3.4 滑动窗口协议（1）

- 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。

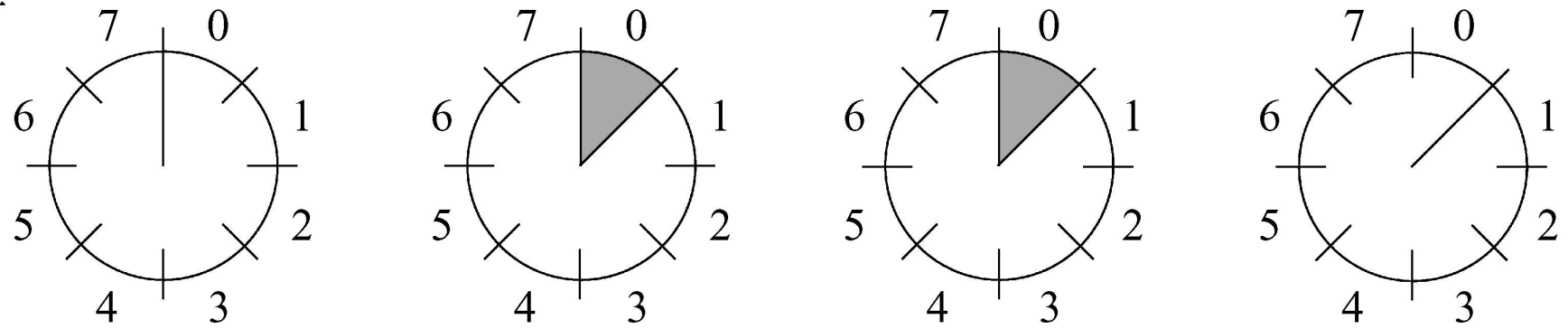
3.4 滑动窗口协议（2）

- 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。发送窗口大小 = 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；

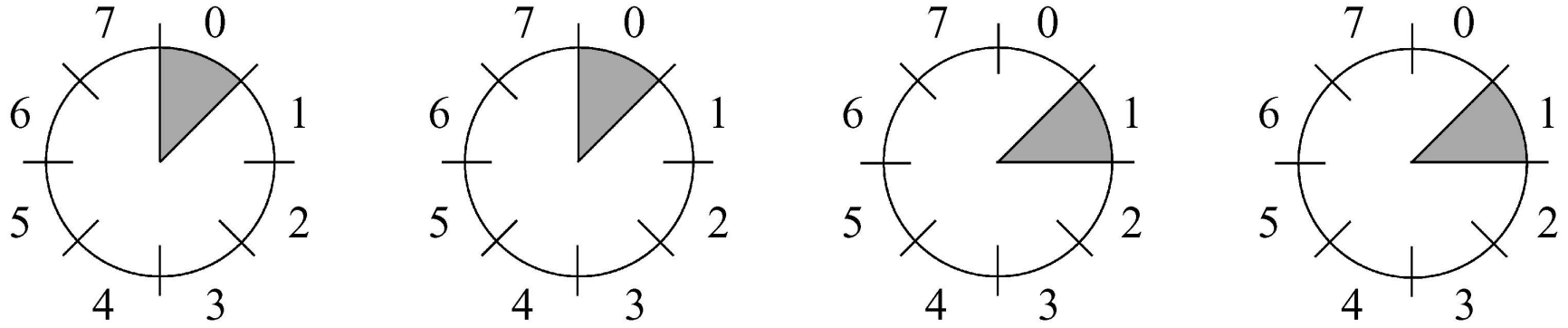
3.4 滑动窗口协议（3）

- 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序号最大的帧，下界表示希望接收的帧；
- 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。

发送方



接收方



(a) (b) (c) (d)

大小为1、有3位序号的滑动窗口

3.4 滑动窗口协议（4）

1.一比特滑动窗口协议（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

```
/* 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 */
```

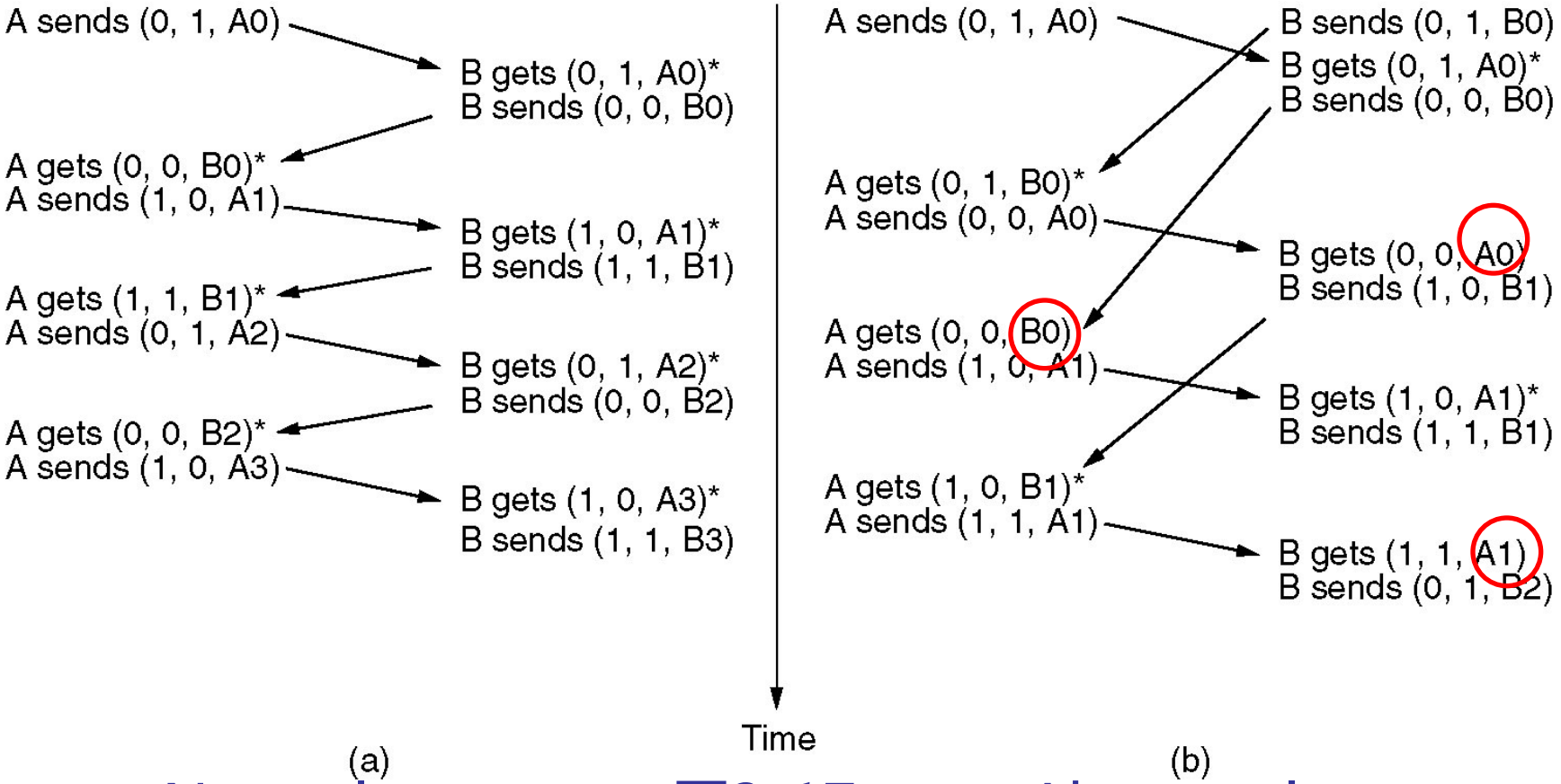
```
while (true) {
    wait_for_event(&event);              /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) { /* a frame has arrived undamaged. */
        from_physical_layer(&r);         /* go get it */

        if (r.seq == frame_expected) {
            /* Handle inbound frame stream. */
            to_network_layer(&r.info);   /* pass packet to network layer */
            inc(frame_expected);         /* invert sequence number expected next */
        }

        if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
            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 */
  }
}
```

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

3.4 滑动窗口协议（5）

- 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；
    - Fig. 3-17（第三版、第四版、第五版翻译书上该图均有误）
  - 效率低，传输时间长。

滑动窗口协议的两种情况



Normal case. 图3-17 Abnormal case.

The notation is (序号, 确认号, 分组号). An asterisk indicates where a network layer accepts a packet.

3.4 滑动窗口协议（6）

2. 退后n帧协议(A Protocol Using Go Back n)

- 为提高传输效率而设计
  - 例：
    - 卫星信道传输速率50kbps，往返传播延迟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%

3.4 滑动窗口协议（7）

- 一般情况，信道带宽*b* 比特/秒，帧长度*L*比特，往返传播延迟*R* 秒，则信道利用率为:

$$(L/b) / (L/b + R) = L / (L + Rb)$$

- 结论
  - 传播延迟大、信道带宽高、帧短时，信道利用率低。
- 解决办法
  - 连续发送多帧后再等待确认，称为流水线技术（pipelining）。
- 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

3.4 滑动窗口协议（8）

- 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。

后退n帧的滑动窗口协议

- 协议4的主要问题是信道利用率太低

  发送端等待发送下一帧的时间至少是发送端到接收端信号传播时间的两倍，没有充分利用两条信道的传输能力

- 信道的利用率定义为

  数据发送时间/从数据开始发送到ACK返回的总耗时

- 协议5是一个发送管道化（pipelining）的协议

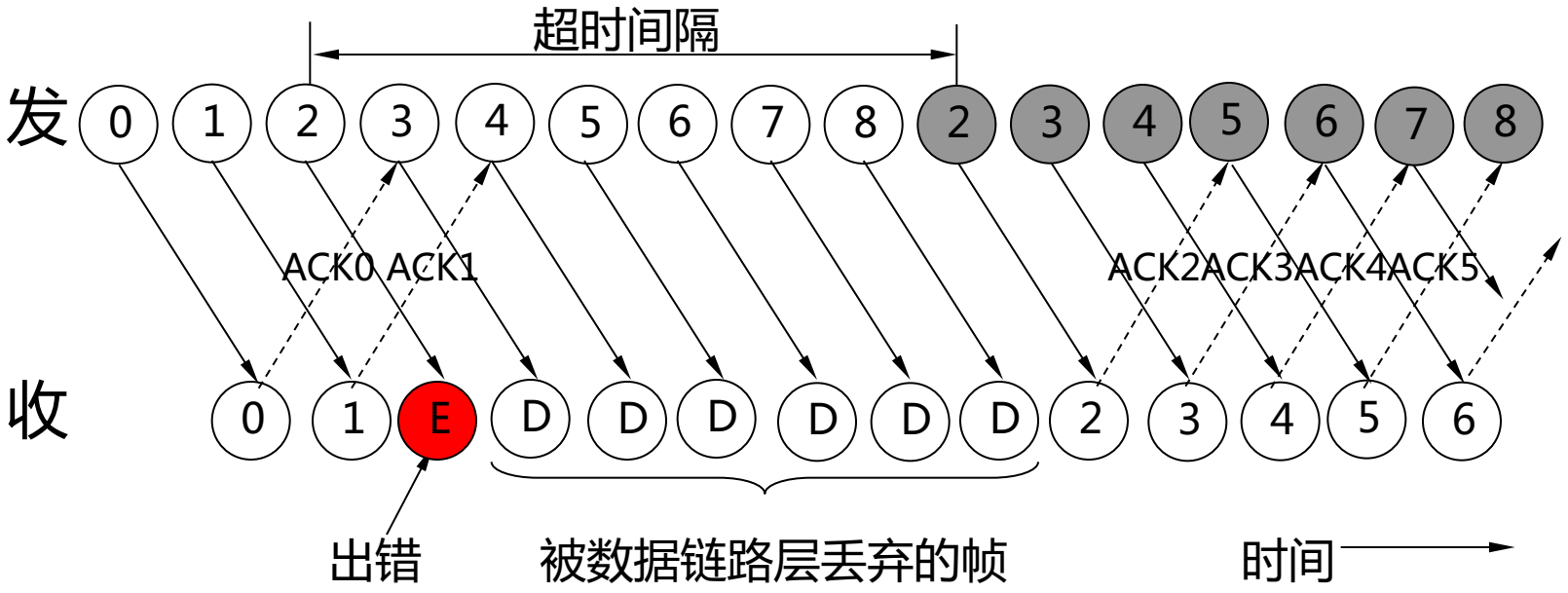
  在等待ACK的时间内连续发送

- 出错后重发必须后退n帧

  一旦重发定时器超时，必须将已发的n帧全部重发

后退n帧的滑动窗口协议图例

- 有一个差错时后退n帧（ $W_T=7$, $W_R=1$ ）



Tnbm P181 Fig. 3-18 (a)

退后n 帧协议（go back n）

- 设帧序号由3个bit表示，即0 ~ 7，并且$W_T = 7$，$W_R = 1$
- 设发送方有大量数据待发送给对方，由于$W_T = 7$，即有7个发送缓冲区，所以可连续发送7帧，并每发送一帧将启动一个重发定时器，但缓冲区的覆盖（窗口的旋转）必须在收到ACK之后，因为一旦该帧的重发定时器超时，必须将原缓冲区内的帧重发

退后n 帧协议（go back n）续

- 接收方采用捎带确认，并假设需确认时，接收方总能从网络层取到待发送给发送方的的数据，以便捎带确认(piggybacking)
- 由于接收方的$W_R = 1$，如期待接收的帧出错，则丢弃此帧及以后所有收到的帧，不发确认（无NAK机制）
- 当发送方出现超时（错帧的TimeOut）后，重发自该帧起的所有已发送帧（在当前的缓冲区中），如发送方连续发送了7帧（2 ~ 8），而2#帧无确认，超时后，必须从2#帧起全部重发

后退n帧的滑动窗口协议总结

（$W_T = 2^n - 1$，$W_R = 1$）

- 协议5，即管道化（pipelining）协议是一个很实用的点对点可靠传输的协议，特别适用于差错率较低的信道，此时，信道利用率很高
- 如果帧序号由n位组成，则发送窗口$W_T = 2^n - 1$，接收窗口$W_R = 1$

选择性重发的滑动窗口协议

- 在差错率较高的信道上，使用协议5可能导致大量的重发，从而使信道利用率大幅度降低

  设帧长为1500 Byte，连续发送7帧共84000 bits，当信道的误码率高于$1.2x10^{-5}$ ，信道的利用率将非常低

- 协议6是一个选择性重发的滑动窗口协议

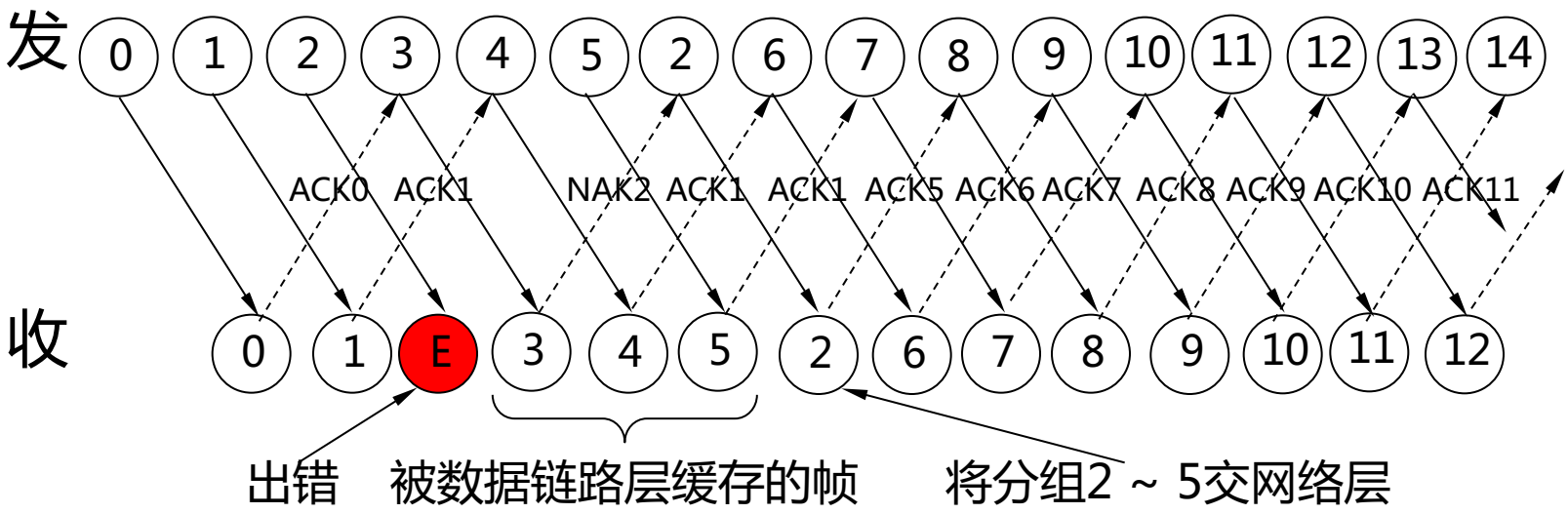
  发送方在某帧的重发定时器超时（没有收到该帧的ACK）后，只要重发该帧即可，而不必重发所有已发送的帧

- 发送方可根据所定义的发送窗口大小，连续发送

  通常发送方将当前缓冲区内的帧连续发送完后，等待确认，发送缓冲区的覆盖（窗口的旋转）将依据收到的ACK的序号，该序号的帧及其以前的所有帧都可被覆盖

选择性重发窗口协议图例

- 有一个差错时仅重发一帧（ W_T =4, W_R = 4 ）



P181 Fig. 3-18 (b)

选择性重发窗口协议说明

- 设帧序号由3个bit表示，即0 ~ 7，并且假设$W_T = 4$， $W_R = 4$
- 设发送方有大量数据等待发送给对方，由于$W_T = 4$，即有4个发送缓冲区，所以可连续发送4帧，并每发送一帧将启动一个（带帧序号的）重发定时器，但缓冲区的覆盖必须在收到确认之后，因为一旦该帧的重发定时器超时，必须将原缓冲区内的帧重发
- 协议6中，由于接收方的$W_R = 4$，如期待接收的帧n#丢失，然而对后继到达的正确的 (n+1)#帧可照常接收，由于(n+1)#帧正确接收，也需要确认，但此确认应有别于对正期待帧的确认，这就是肯定性确认ACK和否定性确认NAK的区别，但无论是ACK或NAK，其中的确认号都必须为n，即期待接收的第n帧

辅助定时器和否定性确认

- 协议6中接收方定义了一个辅助定时器

  接收方采用捎带确认的前提是有数据帧发送给发送方，然而并非需要捎带确认时，接收方上层总是有数据需发送的，所以接收方定义了一个辅助定时器，凡收到一个正确的数据帧并需发送确认时，立即启动辅助定时器，在辅助定时器溢出前，上层有数据帧发送，则可捎带确认，如辅助定时器溢出，则立即发送一个单独的确认，以免发送方的重发定时器超时而导致的数据帧的重发

- 协议6中定义了一个否定性确认的帧格式

  当接收方接收到一个有问题的帧时（包括两种情况：1. CRC校验错；2. CRC校验正确但帧序号错），发送一个否定性确认NAK

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;
```

P185 Fig. 3-21（协议6）

```
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()；如窗口未满，则允许网络层事件；
  else
     disable_network_layer()；如窗口满，则不允许网络层事件；
  }
}
```

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

如序号错则发nak，否则启动辅助定时器

如序号在接收窗口范围内，接受该帧

如顺序正确，则交网络层，调整参数，启动辅助定时器

r.ack+1正是接收方期待接收的帧

```
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);
```

处理 ack

```
while (between(ack_expected, r.ack, ext_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

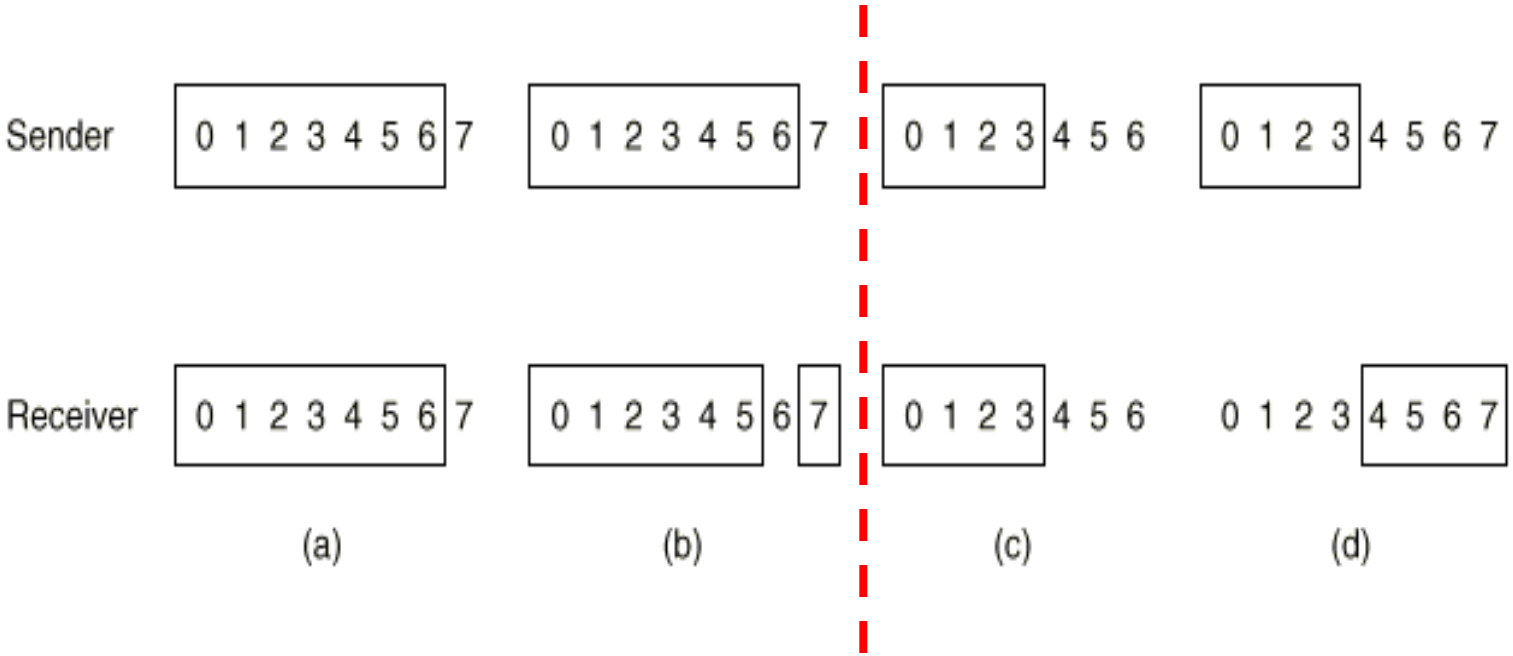
协议6中极端情况的分析

当 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

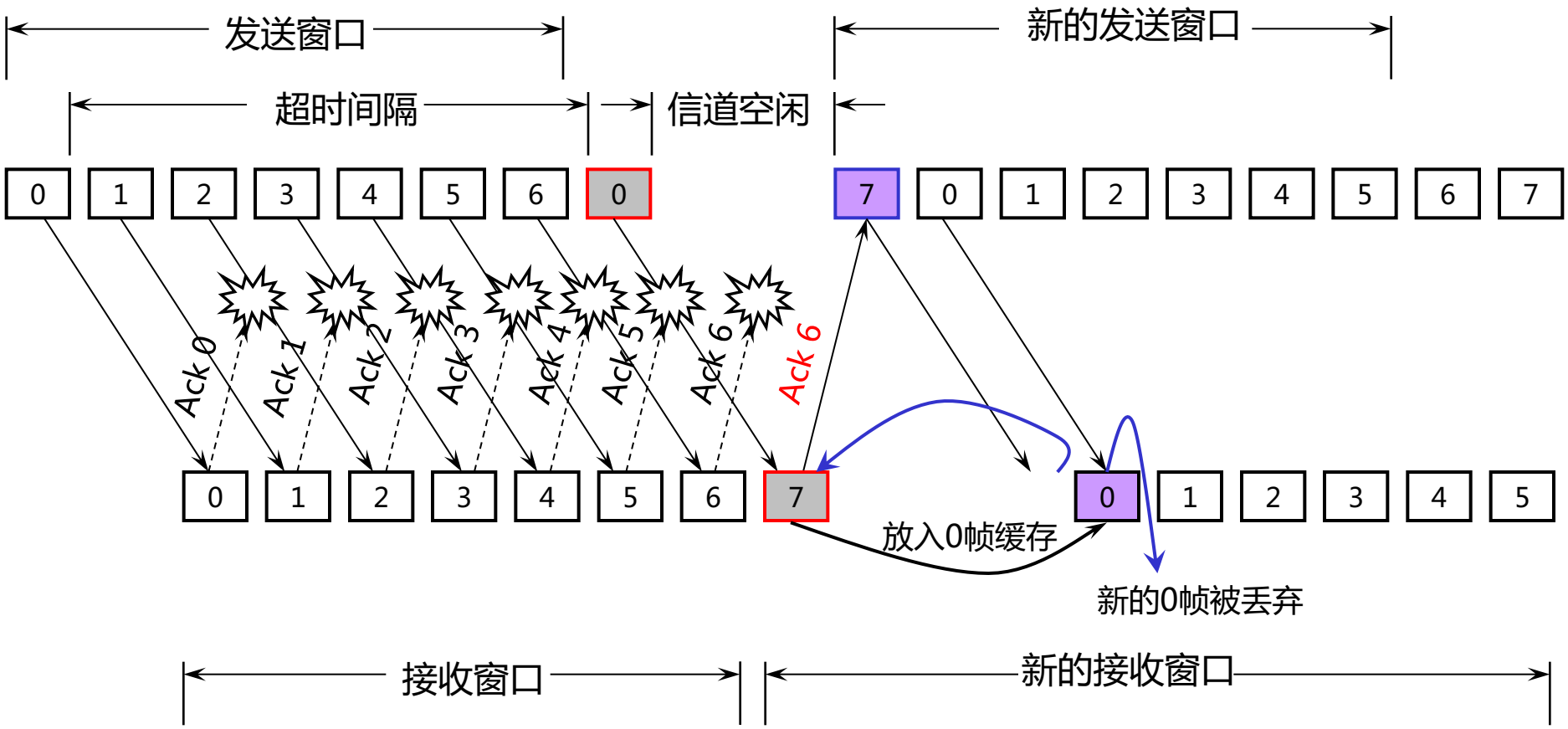
- 发送方当前的发送窗口为0 1 2 3 4 5 6，连续发送了7帧，帧号为0 1 2 3 4 5 6，然后等待确认
- 接收方在初始化后，接收窗口为0 1 2 3 4 5 6，在正确收到0#帧后，由于捎带确认，所以立即启动辅助定时器，并且在辅助定时器没有超时之前，发送方发送的7帧都正确地收到后，然后辅助定时器超时，接收方立即发送了ACK7，意即0 ~ 6帧全部收到并期待接收第7帧，然后取出分组交网络层、清缓冲区并调整窗口为7 0 1 2 3 4 5
- 发送方一直在等待确认，但接收方发送的ACK7由于某种原因丢失了，在重发定时器陆续超时的过程中，发送方又陆续重发了0 1 2 3 4 5 6帧，并继续等待确认

图示

- 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（7，0 ~ 5），确认帧全部丢失，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 7，0 ~ 5，收方已收过帧 0，丢弃新帧 0，协议出错。



协议的进一步图示说明


发送窗口
新的发送窗口
超时间隔
信道空闲
0 1 2 3 4 5 6 0
7 0 1 2 3 4 5 6 7
Ack 0
Ack 1
Ack 2
Ack 3
Ack 4
Ack 5
Ack 6
Ack 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5
放入0帧缓存
新的0帧被丢弃
接收窗口
新的接收窗口

协议6中极端情况的分析（续）

- 接收方收到0 1 2 3 4 5 6帧，认为是第二批来的帧，按正常处理，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，但由于期待接收的第7帧未到，所以只能仍发ACK7，意即再次确认上次收到的0 ~ 6，但由于第7帧未到，所以，已收到的0 1 2 3 4 5帧不能上交网络层
- 但在发送方来看，在收到了ACK7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，又从网络层取分组，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帧已在缓冲区中，是重复的，应丢弃，第7帧接收，发送ACK6，然后将7 0 1 2 3 4 5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，协议失败

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，窗口中的有效顺序号在调整前和调整后有重叠
- 所以，通常：发送窗口+ 接收窗口 <= 2^n

  且：发送窗口 = 接收窗口
- 发送窗口 > 接收窗口 或者 发送窗口 < 接收窗口 都是不合适的

协议6更具实用性

- 协议6中：如果帧序号由n位组成，即0 ~ $2^n - 1$
  - 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，其中的确认号为当前所期待接收的帧的序号
  - 当收到一个CRC校验错的帧，则发一个NAK
  - 当首次收到一个CRC校验正确、但序号错的帧，则发送一个NAK
- 协议6中：增加了一个辅助定时器，当辅助定时器超时，则立即发送一个ACK，其中的确认号为当前所期待接收的帧的序号
  - 当收到一个CRC校验正确、序号也正确的帧，即正是所期待的帧，将启动一个辅助定时器（为等待捎带）
  - 当再次收到一个CRC校验正确、但序号错的帧，因为已发送过NAK ，所以不再发NAK，此时也将启动一个辅助定时器（为等待捎带）

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

小结

- 可靠传输
  - 通过确认和重传机制
  - 传输层协议，如TCP，也提供可靠传输服务
  - 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。

3.5 常用的数据链路层协议（1）

- 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
  - 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

3.5 常用的数据链路层协议（2）

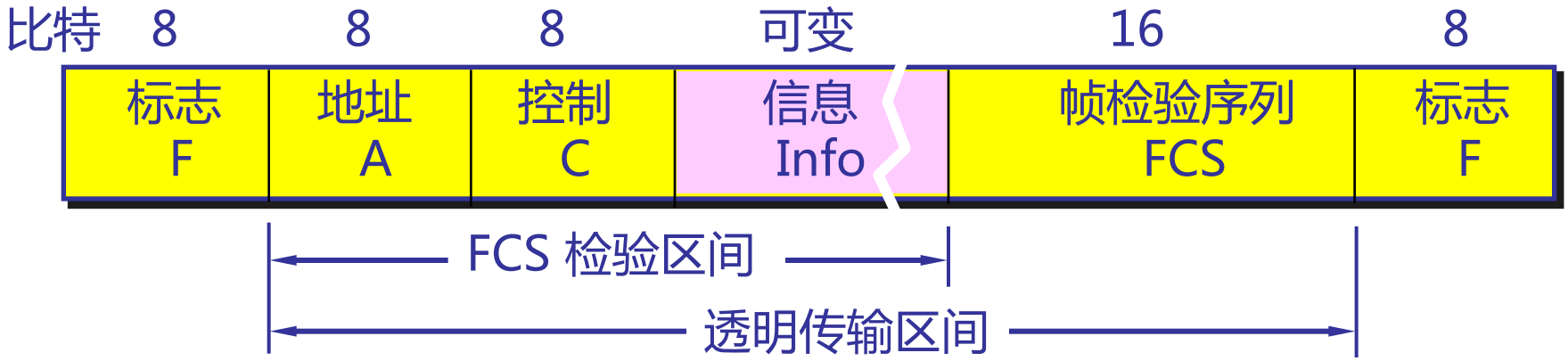
- 面向比特的链路层协议
  - IBM的SNA使用的数据链路协议SDLC（Synchronous Data Link Control protocol）；
  - ANSI修改SDLC，提出ADCCP（Advanced Data Communication Control Procedure）；
  - ISO修改SDLC，提出HDLC（High-level Data Link Control）；
  - CCITT修改HDLC，提出LAP（Link Access Procedure）作为X.25网络接口标准的一部分，后来改为LAPB。

3.5 常用的数据链路层协议（3）

1. 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 } 语法
  - 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 的帧结构（4）



- 标志字段 F (Flag) 为 6 个连续 1 加上两边各一个 0 共 8 bit。在接收端只要找到标志字段就可确定一个帧的位置。

透明传输（5）

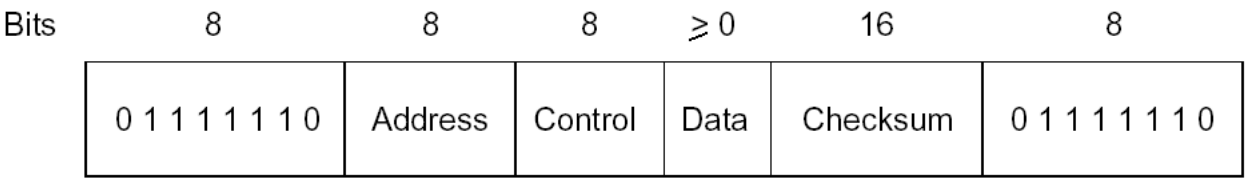
- HDLC 采用零比特填充法使一帧中两个 F 字段之间不会出现 6 个连续 1。
- 在发送端，当一串比特流数据中有 5 个连续 1 时，就立即填入一个 0。
- 在接收帧时，先找到 F 字段以确定帧的边界。接着再对比特流进行扫描。每当发现 5 个连续 1 时，就将其后的一个 0 删除，以还原成原来的比特流。
- 采用零比特填充法就可传送任意组合的比特流，或者说，就可实现数据链路层的透明传输。
- 当连续传输两个帧时，前一个帧的结束标志字段 F 可以兼作后一帧的起始标志字段。
- 当暂时没有信息传送时，可以连续发送标志字段，使收端可以一直和发端保持同步。

其他字段（6）

- 地址字段 A 是 8 bit。
- 帧检验序列 FCS 字段共 16 bit。所检验的范围是从地址字段的第一个比特起，到信息字段的最末一个比特为止。
- 控制字段 C 共 8 bit，是最复杂的字段。HDLC 的许多重要功能都靠控制字段来实现。

3.5 常用的数据链路层协议（7）

- 帧结构



  - 定界符
    - 01111110
    - 空闲的点到点线路上连续传定界符
  - 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。
  - 控制域（Control）
    - 序号
      - 使用滑动窗口技术，3位序号，发送窗口大小为7

3.5 常用的数据链路层协议（8）

- 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
- 帧类型
  - 信息帧（Information）
  - 监控帧（Supervisory）
  - 无序号帧（Unnumbered）

3.5 常用的数据链路层协议（9）

- 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
  - 捎带确认（Next）
    - 捎带第一个未收到的帧序号，而不是最后一个已收到的帧序号

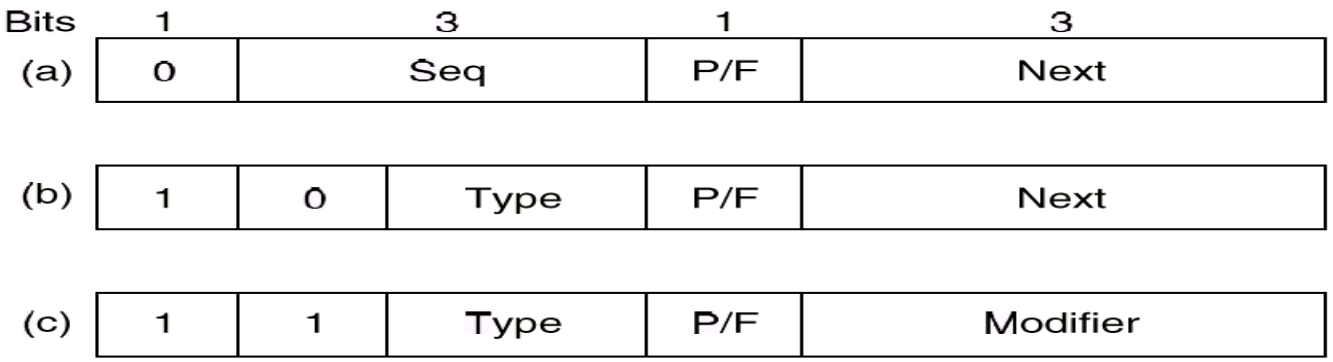


Fig. 3-25. Control field of (a) an information frame, (b) a supervisory frame, (c) an unnumbered frame.

3.5 常用的数据链路层协议（10）

- 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”。
- 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)

3.5 常用的数据链路层协议（11）

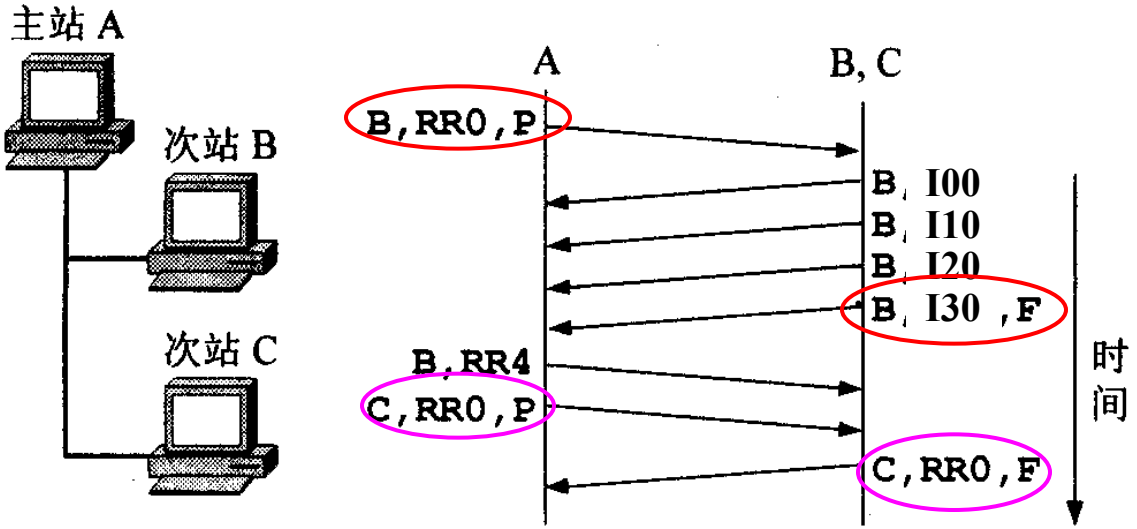
第3-4比特	帧名	功能
0 0	RR(Receive Ready) 接收准备就绪	准备收下一帧 确认序号为Next-1及其以前的各帧
1 0	RNR(Receive Not Ready) 接收未就绪	暂停接收下一帧 确认序号为Next-1及其以前的各帧
0 1	REJ(Reject) 拒绝	从Next起的所有帧都否认, 但 确认序号为Next-1及其以前的各帧
1 1	SREJ(Selective Reject) 选择拒绝	只否认序号为Next的帧, 但 确认序号为Next-1及其以前的各帧

- 无序号帧
  - 可以用来传控制信息，也可在不可靠无连接服务中传数据。

3.5 常用的数据链路层协议（12）

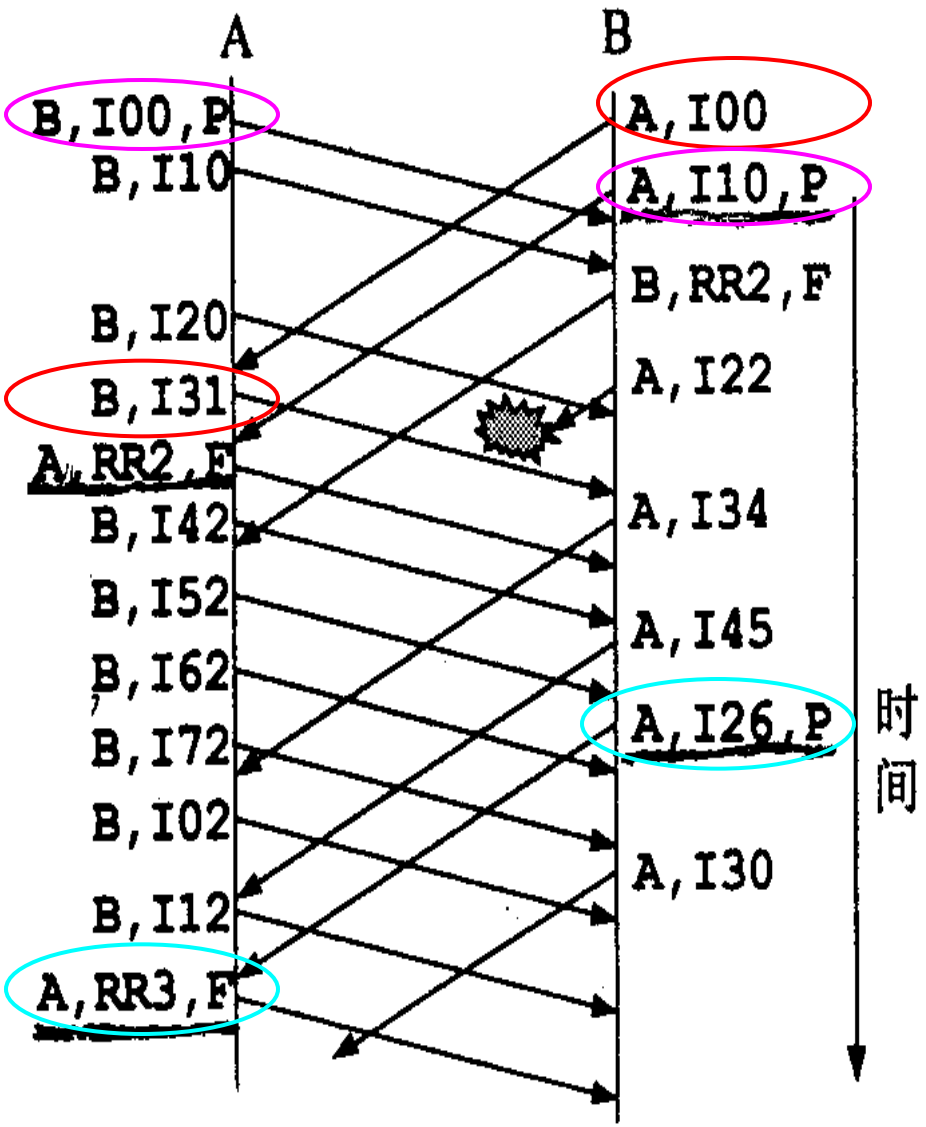
- 监控帧的第5比特也是P/F比特。
- 若P/F值为零，则P/F比特并没有任何意义。只有当P/F比特的值为1时才具有意义。
- 需要注意：在不同的数据传送方式中，P/F比特的用法是不一样的。在说明帧的传送过程中，为了更清楚地表示P/F比特的作用，往往将它写为P比特或F比特。
- 在非平衡配置的正常响应方式NRM中，次站不能主动向主站发送信息。次站只有收到主站发出的P比特为1的命令帧(S帧或I帧)以后才能发送响应帧。若次站有数据发送，则在最后一个数据帧中将F比特置1。若无数据发送，则应在回答的S帧中将F比特置1。
- 在非平衡配置的异步响应方式ARM或在平衡配置的异步平衡方式ABM中，任何一个站都可以在主动发送的S帧和I帧中将P比特置1。对方站收到P = 1的帧后，应尽早地回答本站的状态并将P比特置1，不过此时并不表示数据已发完和不再发送数据了。

3.5 常用的数据链路层协议（13）



- 图中主站A与次站B和C连成多点链路。
- 我们将所传送的帧的一些主要参数按照“地址，帧名和序号，P/F”的先后顺序标注在图中。
- 这里的“地址”是指地址字段中应填入的站地址，“帧名”是指帧的名称，如RR或I， 序号则是指监控帧中的Next或信息帧中的SeqNext(R)。 P/F是在当P/F为1时才写上P或F，表明此时控制字段的第5比特为1。

3.5 常用的数据链路层协议（14）

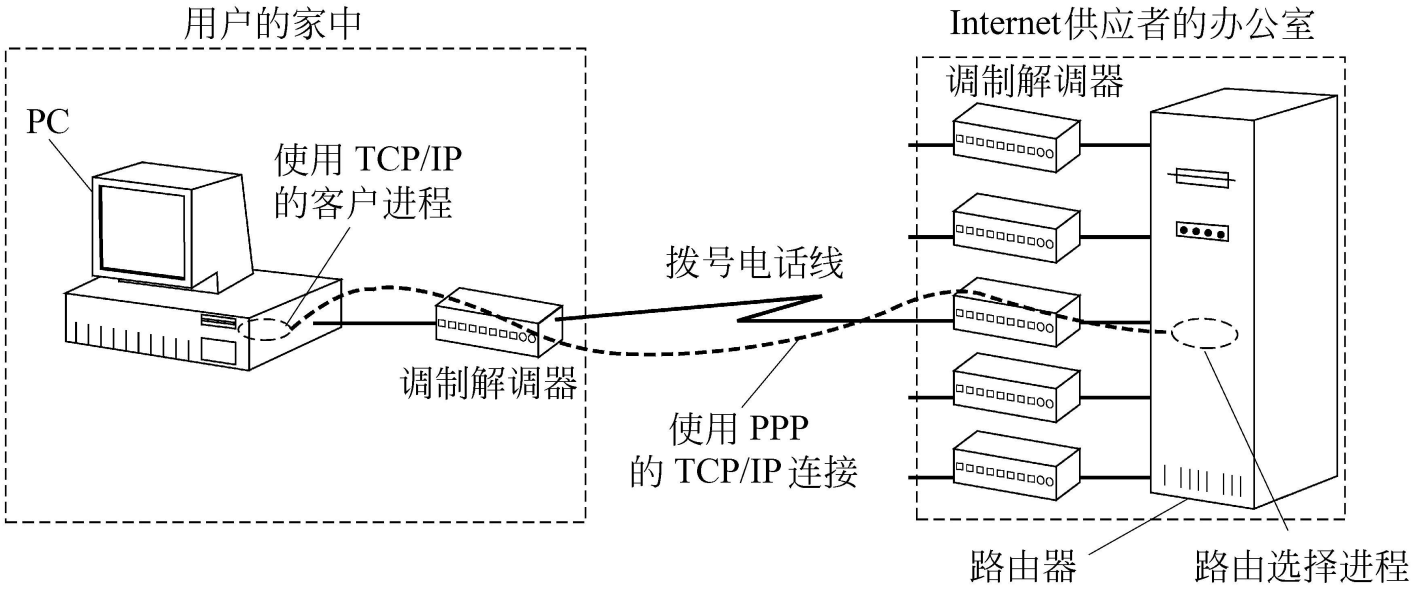


- 有了P/F比特，使HDLC规程使用起来更加灵活。
- 在两个复合站全双工通信时，任何一方都可随时使P = 1。这时对方就要立即回答RR帧，并置F = 1。这样做可以更早地收到对方的确认。
- 如果不使用P/F比特，则收方不一定马上发出确认帧。收方可以在发送自己的信息帧时，在某一个信息帧中捎带把确认信息发出(利用Next)。

3.5 常用的数据链路层协议（15）

2. Internet数据链路层协议

- 点到点通信的两种主要情形
  - 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（Access Server）（dial-up host-router connection）



3.5 常用的数据链路层协议（16）

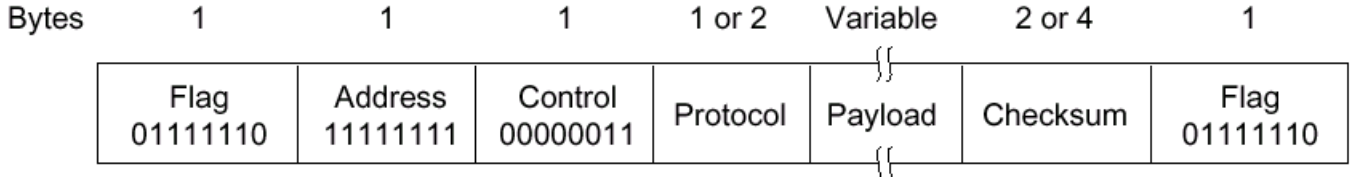
- SLIP —— Serial Line IP
  - 1984年，Rick Adams提出，RFC1055，发送原始IP包，用一个标记字节来定界，采用字符填充技术；
  - 新版本提供TCP和IP头压缩技术，RFC 1144
  - 存在的问题
    - 不提供差错校验
    - 只支持IP
    - IP地址不能动态分配
    - 不提供认证
    - 多种版本并存，互连困难

3.5 常用的数据链路层协议（17）

- 点到点协议 PPP——Point-to-Point Protocol
  - RFC 1661，RFC 1662，RFC 1663
  - 与SLIP相比，PPP有很大的提高，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；
  - 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

3.5 常用的数据链路层协议（18）

- 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



  - 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：指示净负荷中是何种包，缺省大小为2个字节。
  - 净负荷域：变长，缺省为1500字节；
  - 校验和域：2或4个字节

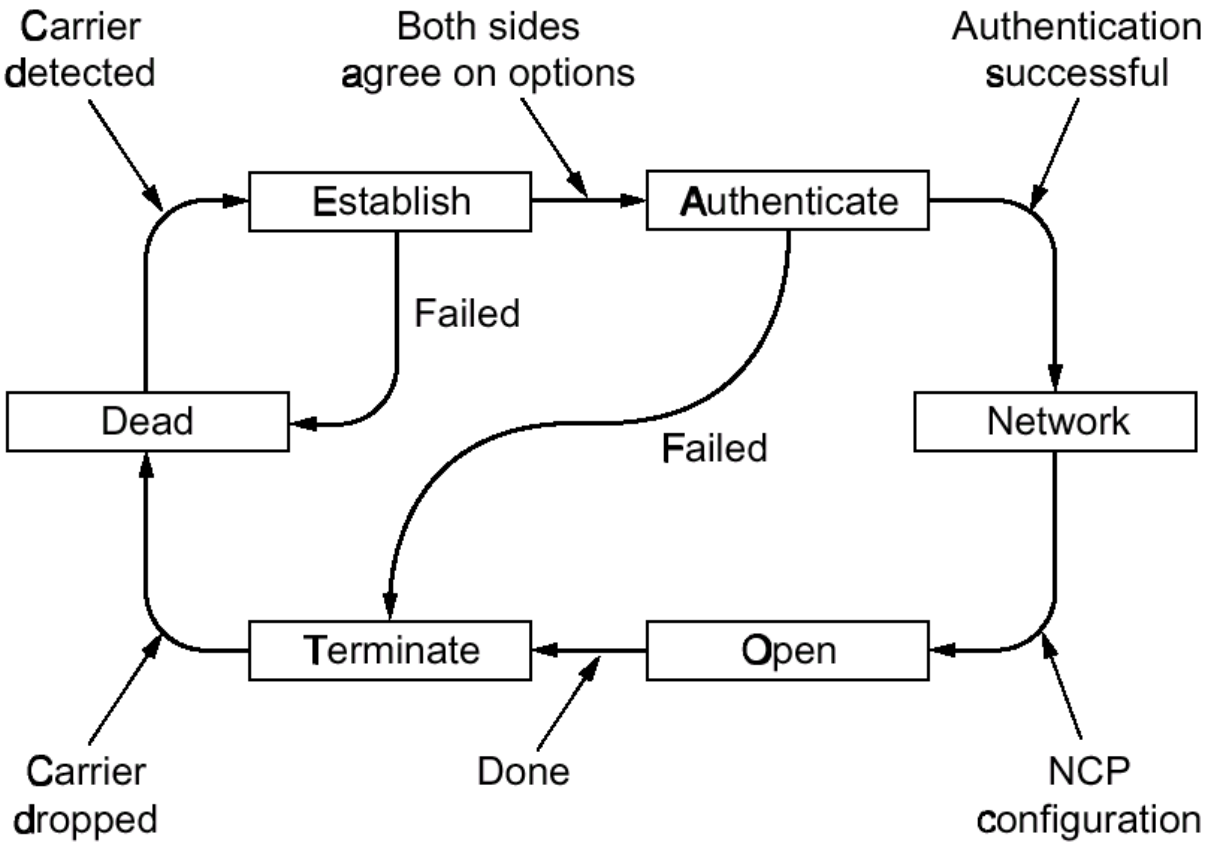
3.5 常用的数据链路层协议（19）

PPP总结：

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、选项协商和包头压缩功能，并具有利用HDLC帧进行可靠传输的可选功能。

3.5 常用的数据链路层协议（20）

- PPP链路 up / down 过程（简单状态图）



3.5 常用的数据链路层协议（21）

- LCP用来在ESTABLISH状态协商数据链路协议选项，并不关心选项内容，而是提供一种协商机制，并且提供检测链路质量的方法。RFC 1661 定义了11种LCP帧类型。

Name	Direction	Description
Configure-request	I → R	List of proposed options and values
Configure-ack	I ← R	All options are accepted
Configure-nak	I ← R	Some options are not accepted
Configure-reject	I ← R	Some options are not negotiable
Terminate-request	I → R	Request to shut the line down
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request received
Protocol-reject	I ← R	Unknown protocol requested
Echo-request	I → R	Please send this frame back
Echo-reply	I ← R	Here is the frame back
Discard-request	I → R	Just discard this frame (for testing)

本章小结

- 数据链路层的功能
- 组帧
- 差错控制
  - 检错编码
  - 纠错编码
- 流量控制与可靠传输机制
  - 流量控制、可靠传输与滑轮窗口机制
  - 单帧滑动窗口与停止-等待协议
  - 多帧滑动窗口与后退N帧协议（GBN）
  - 多帧滑动窗口与选择重传协议（SR）
- 常见数据链路层协议
  - PPP协议、HDLC协议