西南交通大學

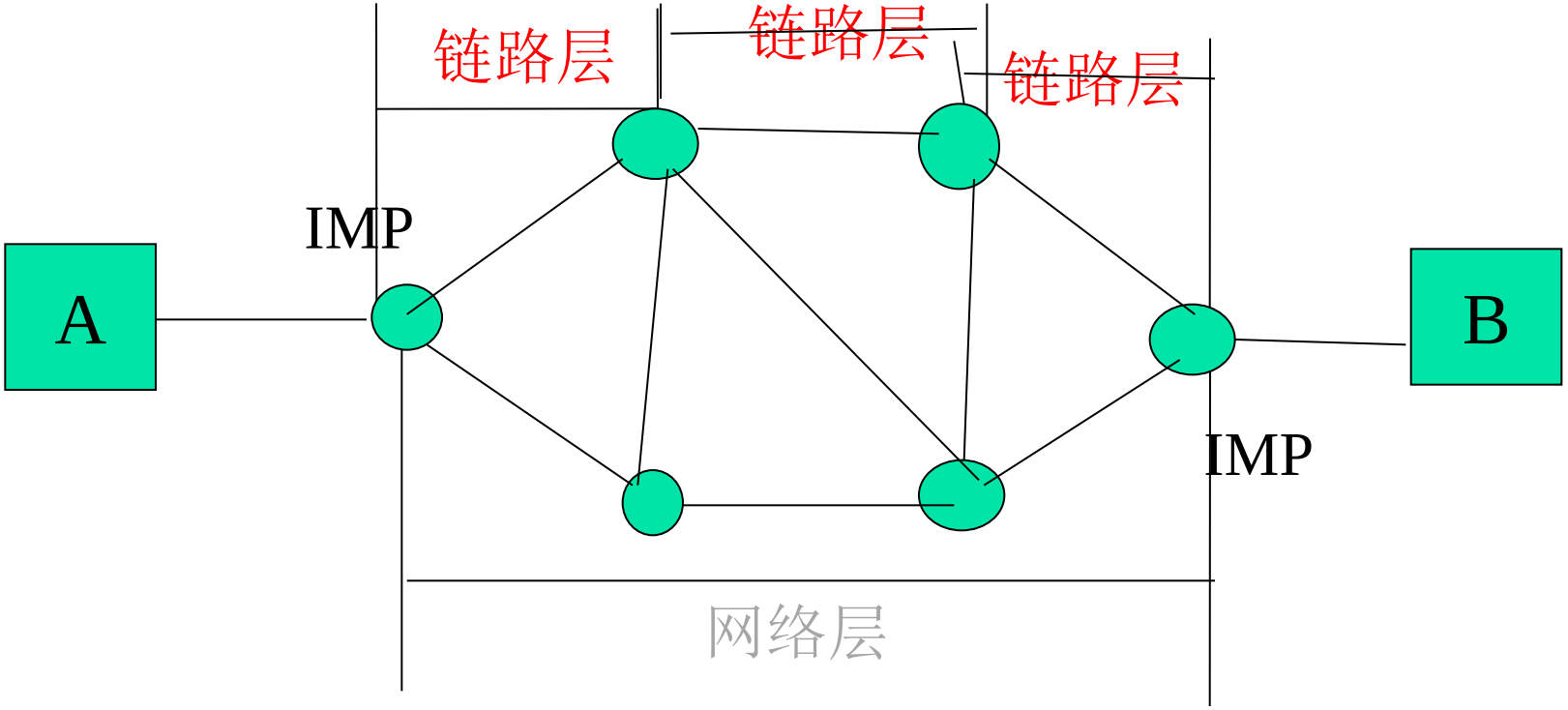
第四章

数据链路层

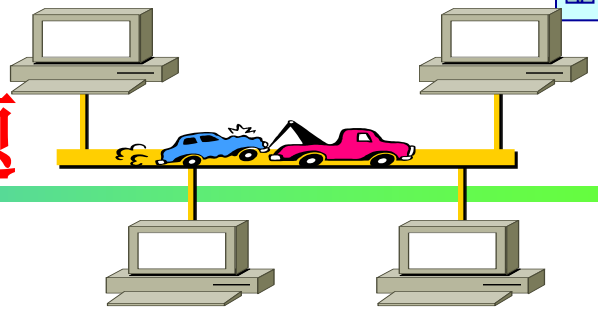
设计、制作、讲授：谭献海

EMAIL：xhtan@swjtu.cn

数据链路层的职责：确保数据（帧）在链路上的可靠传输



4.1 数据链路层主要设计问题



① 所传输信息含义的定义？----将比特串组合成帧（帧格式定义）、帧同步、帧定界

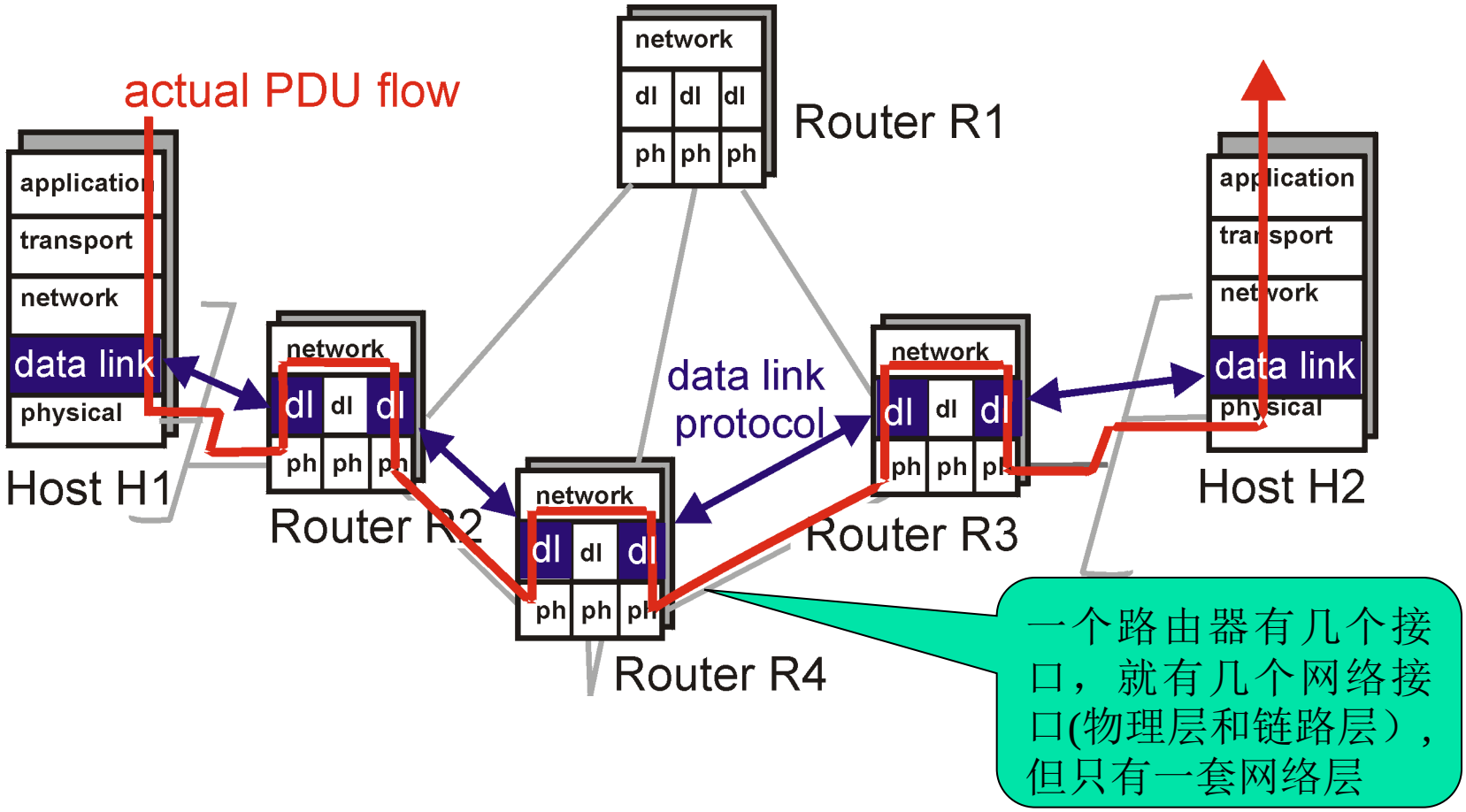
② 如何解决收发速率匹配，避免快的发送淹没慢的接收的问题？----流量控制

③ 如何确保数据帧的可靠传输？----差错控制

④ LAN or wireless等 ----共享信道分配（信道访问控制(MAC)协议）

⑤ 链路管理（链路的建立、维护、释放）

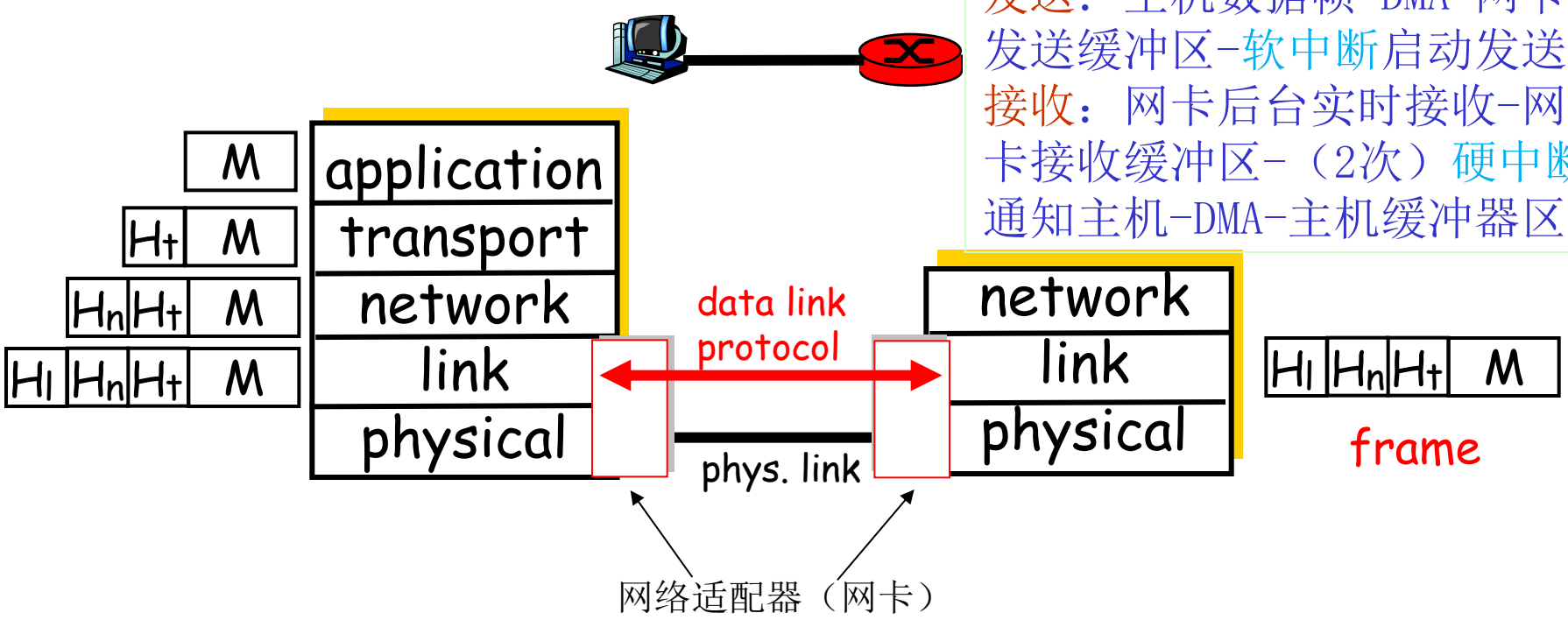
数据链路层模型


actual PDU flow
application
transport
network
data link
physical
Host H1
network
dl dl dl
ph ph ph
Router R2
network
dl dl dl
ph ph ph
Router R1
network
dl dl dl
ph ph ph
Router R4
data link
protocol
network
dl dl dl
ph ph ph
Router R3
application
transport
network
data link
physical
Host H2
一个路由器有几个接口，就有几个网络接口(物理层和链路层），但只有一套网络层

网络中数据链路层的实现

- 由网络适配器（网卡）实现物理层和数据链路层
  - Ethernet 网卡，X.25网卡
  - 网卡(可编程接口)组成: RAM，DSP 芯片，主机总线接口和网络接口(BNC,RJ45)

网卡完成数据帧的发送与接收：
发送：主机数据帧-DMA-网卡发送缓冲区-软中断启动发送
接收：网卡后台实时接收-网卡接收缓冲区-（2次）硬中断通知主机-DMA-主机缓冲器区



4.2

同步序列
(8字节)

同步结束,数
据开始标志

以太帧头
(14字节)

IP报头
(20字节)

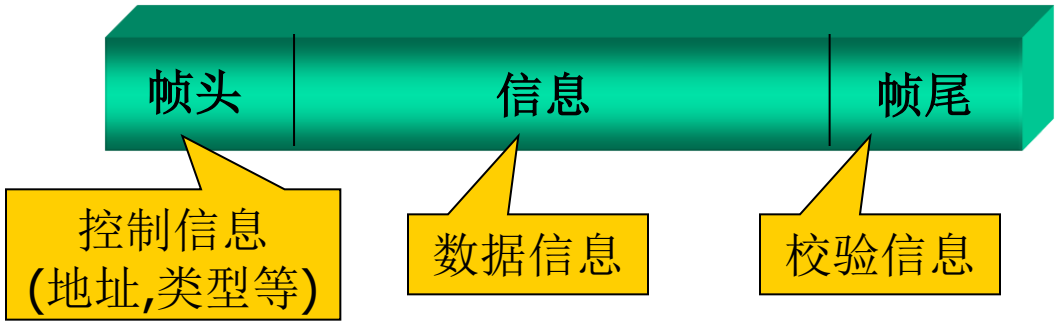
TCP报头
(20字节)

应用数据

上述比特流的含义

4.2 组帧

✓ 组帧：比特流分段，封装成数据帧



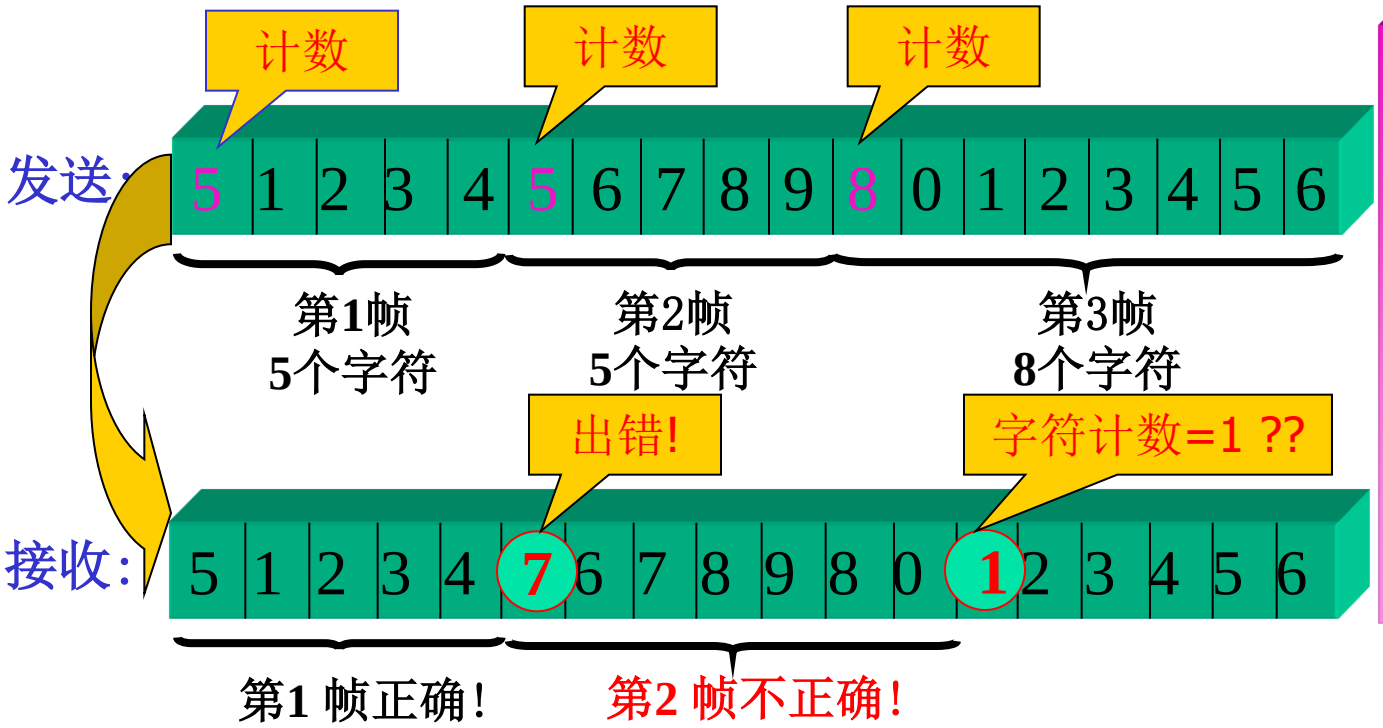
✓组帧的目的：将数据按块进行错误检测/控制

✓组帧方法：

- Timing：危险，无保障
- **Character count**(字符计数)**:** 简单，但错误会引起误解
- **Character delimiter and stuffing:** 用特殊的**控制字符**进行帧定界(面向字符协议)
- **Bit delimiter and stuffing:** 用特殊的**比特序列(01111110)**作帧定界(面向比特协议)
- 物理层编码违例（ ***coding violations***）

字符计数法

•在帧头中用一个长度域来表示整个帧的字符个数

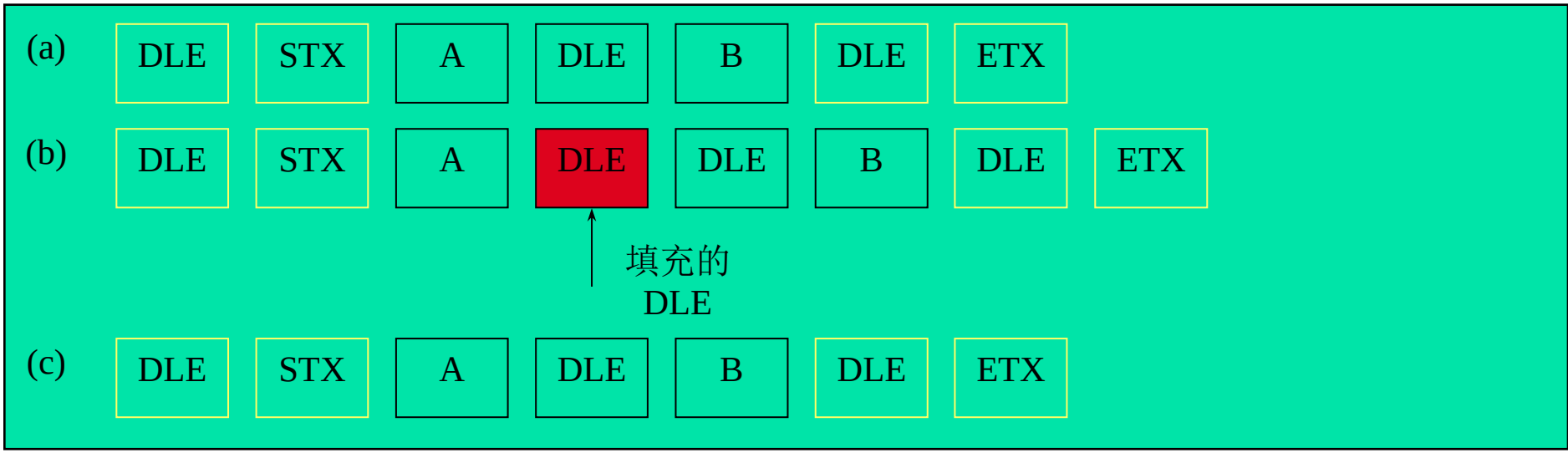


问题:
出错之后,接收方与发送方不同步!无法确定下一帧的开始位置. 请求重传也没有用处,因为发送方不知道应该回跳多少字符开始重传.

一般地,在数据链路协议中使用字符计数与其它方法相结合来提高可靠性.

面向字符协议--字符填充

- ✓ 用特殊控制字符（DLE STX 或 DLE ETX）进行帧定界
- ✓ 数据中出现控制字符‘DLE’等时转义----插入’DLE’
- ✓ 接收时由目的站自动判断并去掉(转义的)填充字符

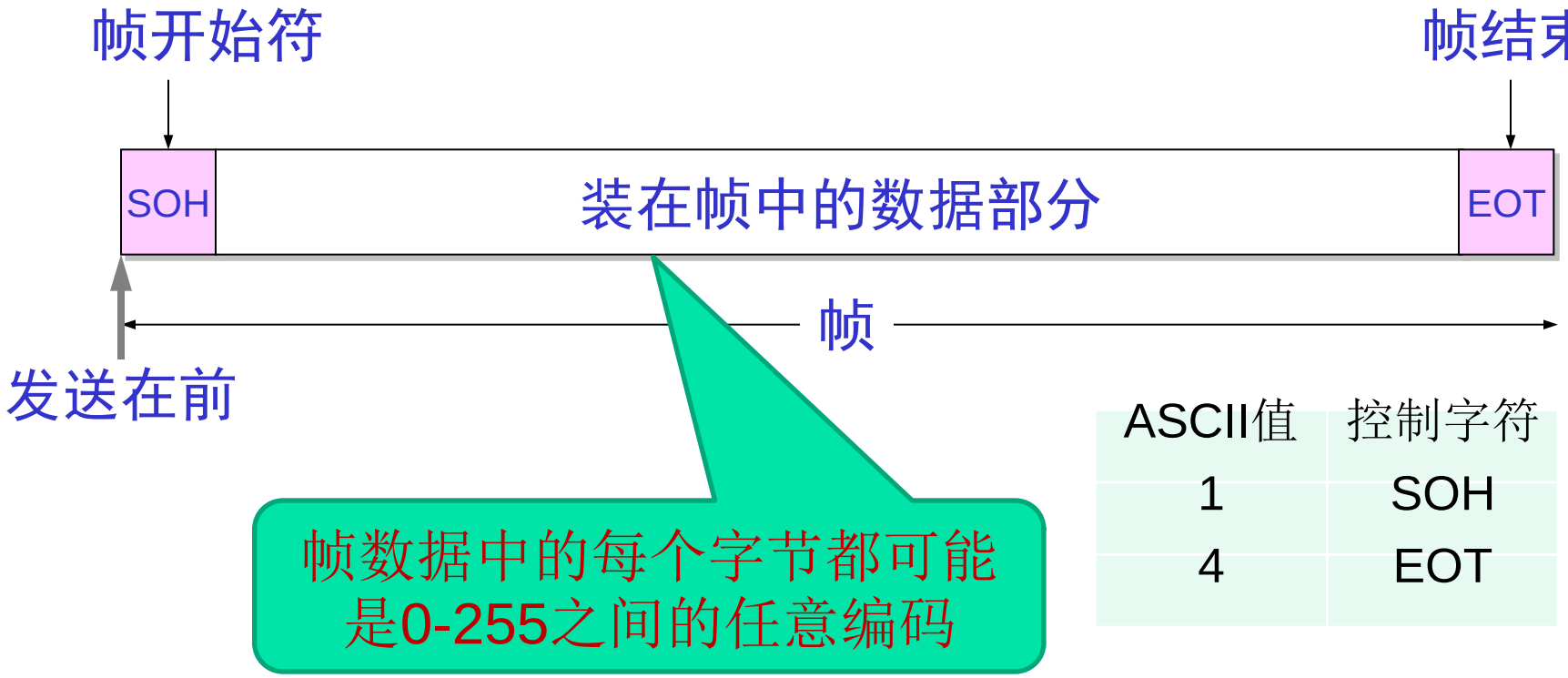


（a）网络层发出的数据；
（b）经数据链路层填充后的数据；
（c）接收方识别并去掉转义符后的数据

DLE-------Data Link Escape
STX-------Start of TeXt
ETX-------End of TeXt

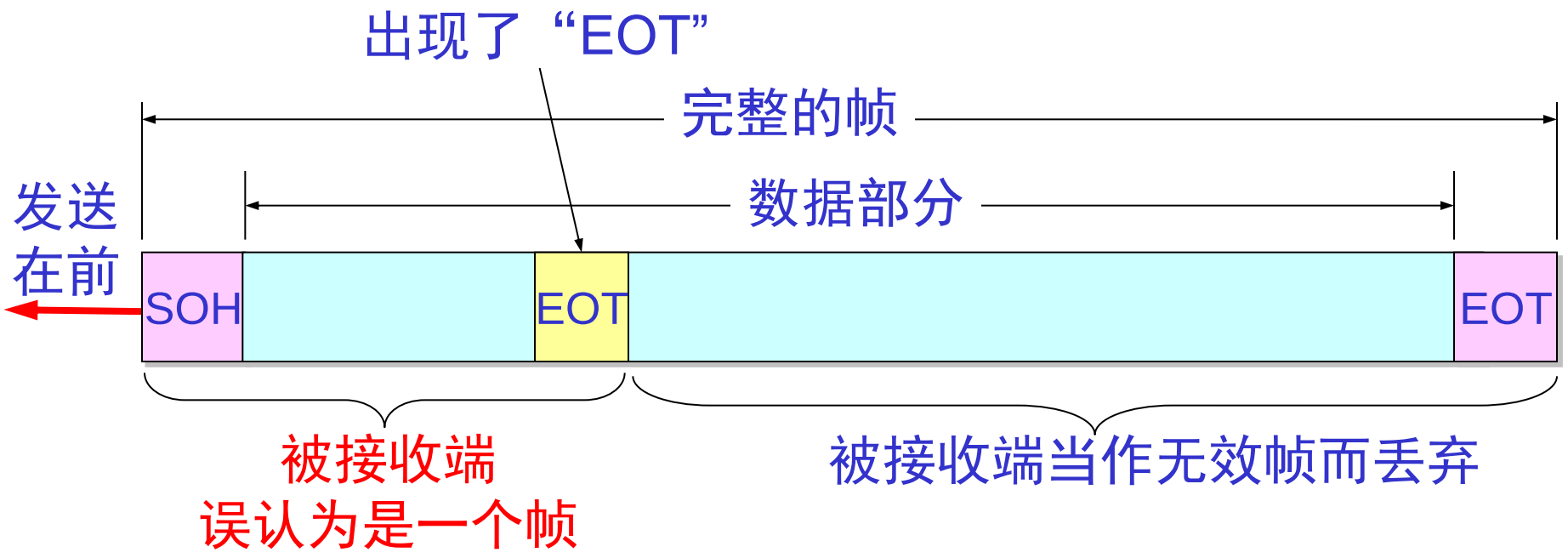
缺点：局限于8位字符和***ASCII***字符传送

用控制字符进行帧定界的方法举例



ASCII值	控制字符
1	SOH
4	EOT

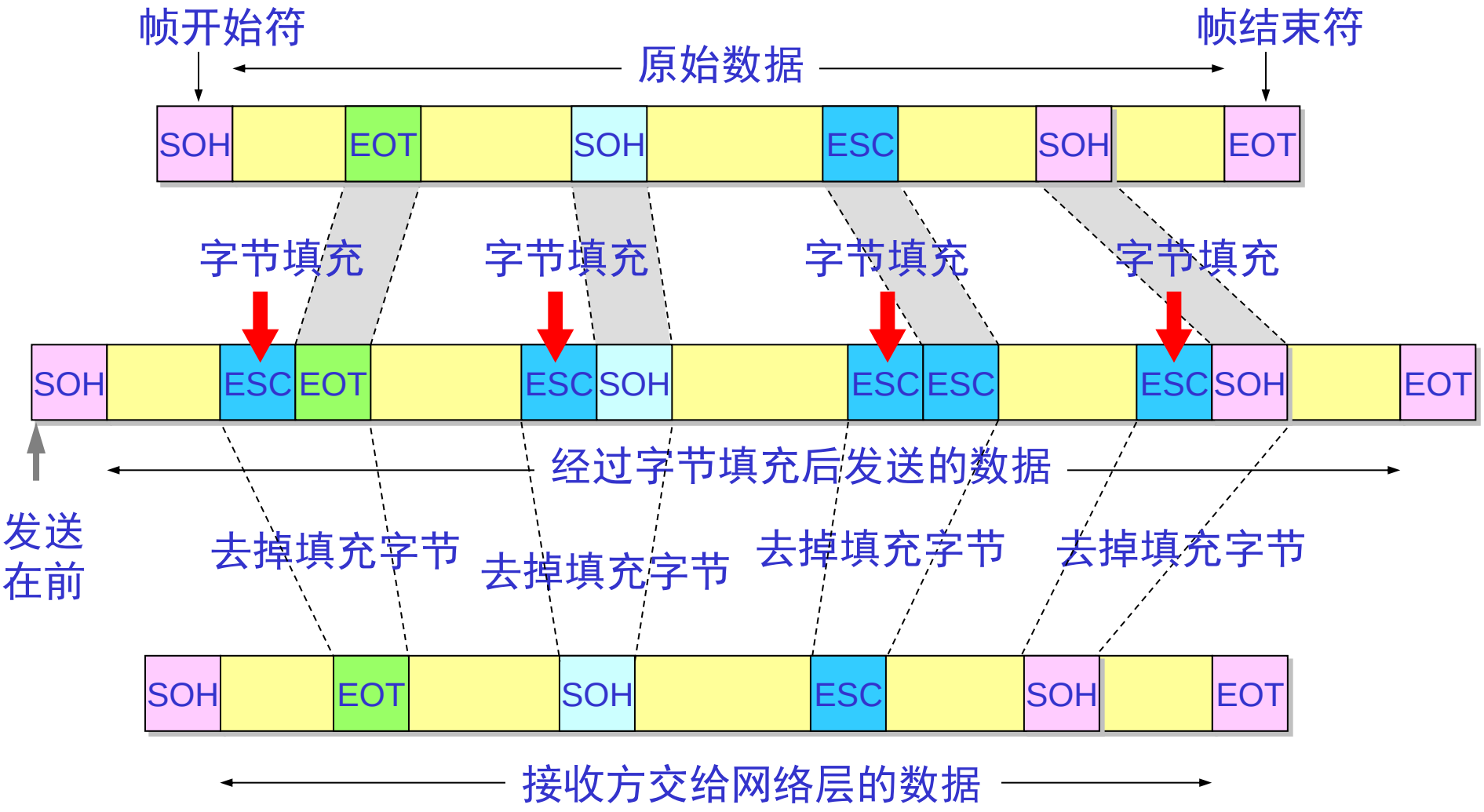
面向字符协议--透明传输


出现了“EOT”
完整的帧
数据部分
发送
在前
SOH
EOT
EOT
被接收端
误认为是一个帧
被接收端当作无效帧而丢弃

解决透明传输问题—转义

- 字节填充(byte stuffing)或字符填充(character stuffing)——
- 发送端的数据链路层在数据中出现控制字符“SOH”（帧开始）或“EOT”（帧结束）的前面插入一个转义字符“ESC”(其十六进制编码是 1B)。
- 转义字符也属于控制字符，如果它也出现在数据当中，那么也应在转义字符前面插入一个转义字符。
- 接收端的数据链路层在将数据送往网络层之前判断并删除插入的转义字符----删除第一个转义符ESC

用字节填充法解决透明传输的问题



面向比特协议--比特填充

- 用特殊的比特序列 (比特标志)进行帧定界 (01111110)
- 当数据中出现帧定界标志时需转义（0比特插入技术：连续5个1的后面自动插入一个0）
- 接收时判断并去掉填充比特

数据中某一段比特组合恰好出现和定界符一样的情况

0 1 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0

会被误认为是帧定界符

发送端在 5 个连 1 之后插入比特0再发送出去

0 1 0 0 1 1 1 1 1 0 1 0 0 0 1 0 1 0

插入 比特0

在接收端将 5 个连 1 之后的插入的比特0删除，恢复原样

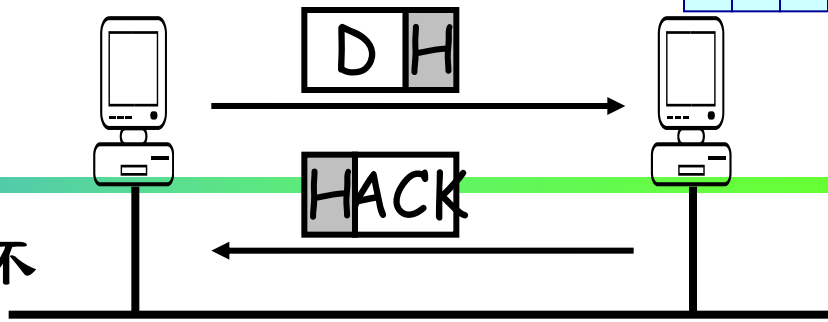
0 1 0 0 1 1 1 1 1 0 1 0 0 0 1 0 1 0

在此位置删除填入的 比特0

物理层编码违例

- 只适用于物理层编码有冗余的网络
- 802 LAN: Manchester encoding or Differential Manchester encoding 用high-low pair/low-high pair表示1/0, high-high/low-low不表示数据，每个周期均有一个高/低电平转换,非常有规律,可以用来判断是否是有效数据。
- 优点：不需要填充
- 缺点：编码效率低(50%)

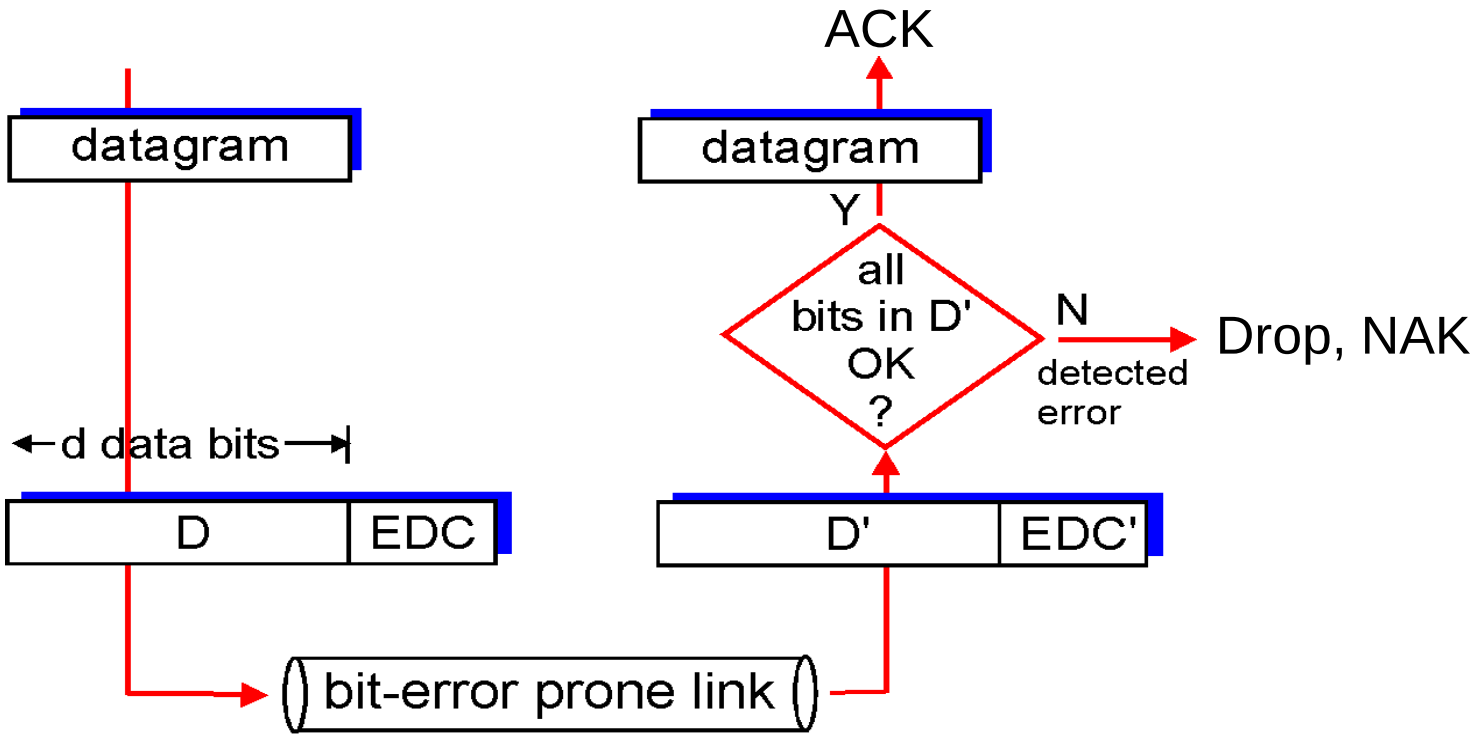
4.3 错误控制



- 目的：避免数据丢失或损坏
- 错误检测
  - 发送方在报文中插入错误检测位
  - 接收方根据这些检测位进行错误检测
  - 如果正确，则接受，并应答ACK
  - 如果错误，则丢弃，或应答NAK
- 重传
  - 如果收到NAK，或在规定时间内没有应答（超时-报文丢失），则重传
- 错误纠正
  - 发送方在报文中插入错误纠正码
  - 接收方根据纠错码进行错误纠正

错误检测

- EDC= 错误检测（纠正）码（冗余位）
- EDC越长，检错和纠错的性能越好



错误检测与纠正

- 单比特错与突发错

  单比特错

  突发错（多比特错）

  110101 → 100101 vs 100001

- N比特码字 = m 信息比特 + r 检查比特
- Hamming码距 = 相异比特的数目

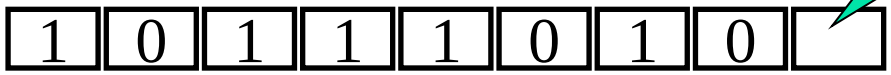
  1010101

  1001010

  0011111 ⇒ Hamming 码距 = 5
- Distanced code = 任意两个码字之间的最小Hamming 码距
- 要检测 d 比特错误， 需要 d+1 码距
- 要纠正 d错误， 需要 2d+1 码距

单字符 奇偶校验

奇偶校验位



单比特奇偶校验:

能检测奇数位错误

Odd Parity

(奇数个1)

√

1-bit error

√

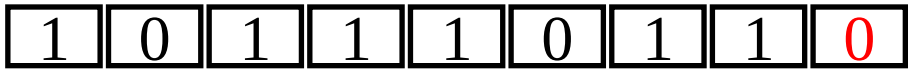
3-bit error

2-bit error

×

Even Parity

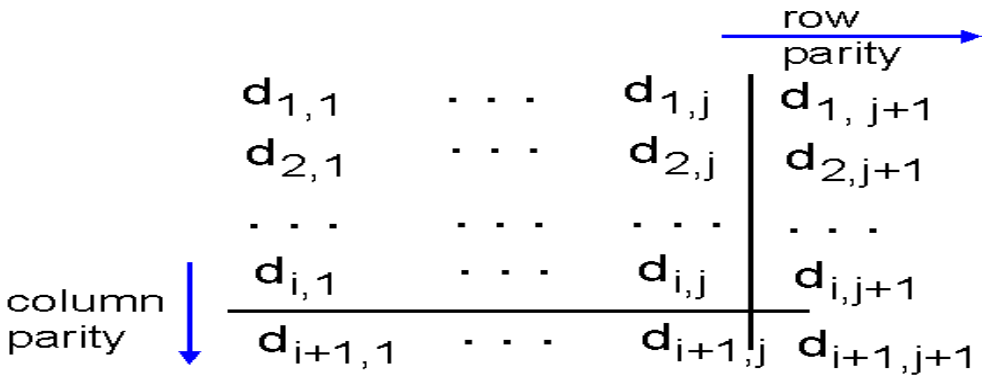
(偶数个1)

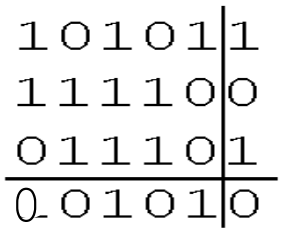


数据块 奇偶校验

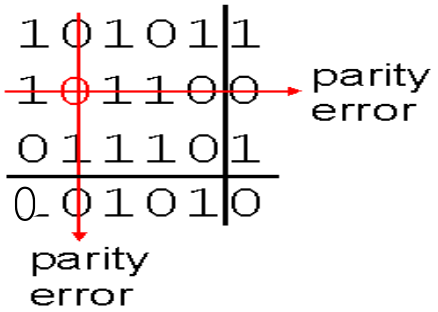
二维奇偶校验:*行列奇偶校验*

能纠正单比特错误;检测？比特错



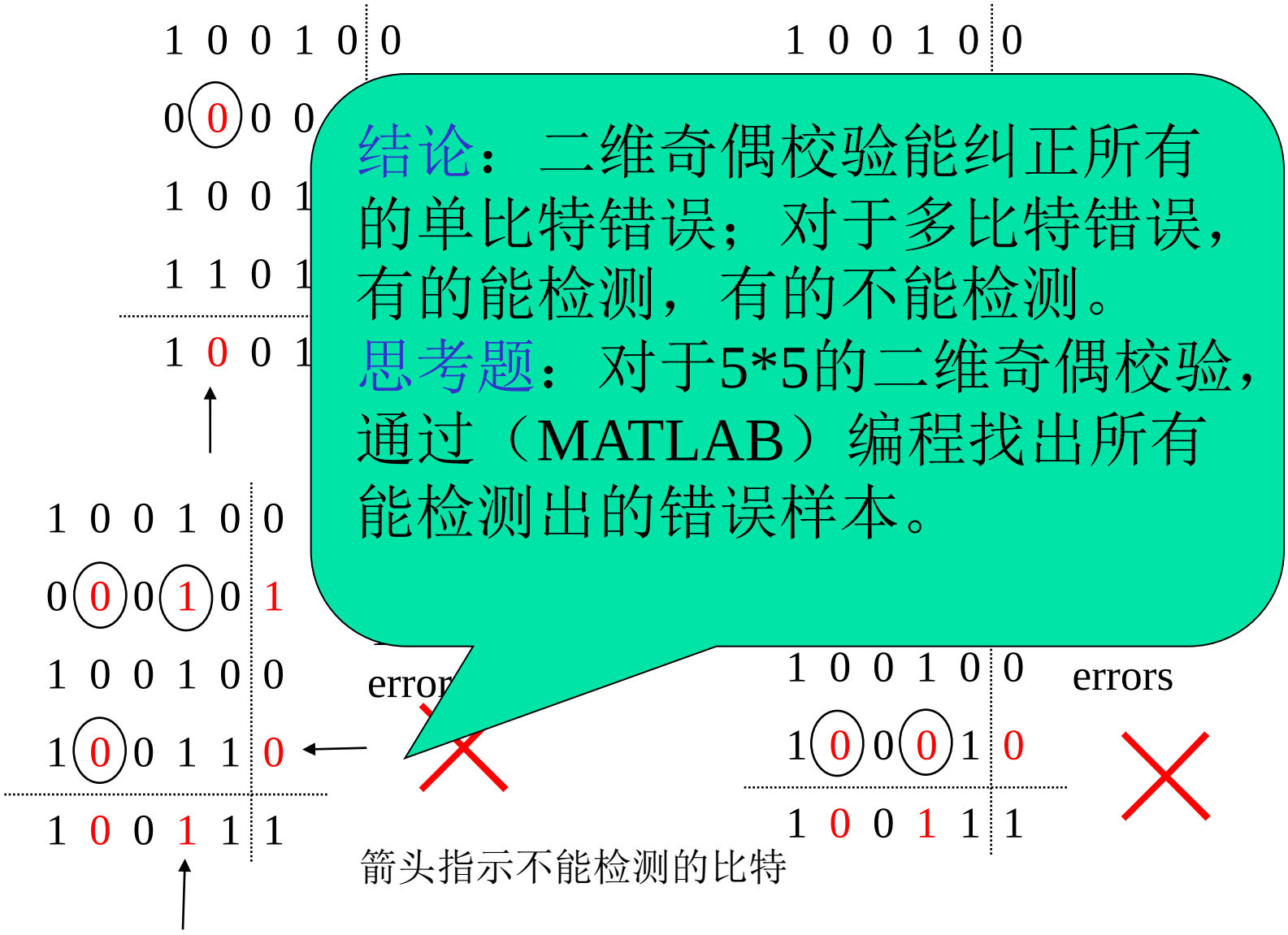


no errors



correctable single bit error

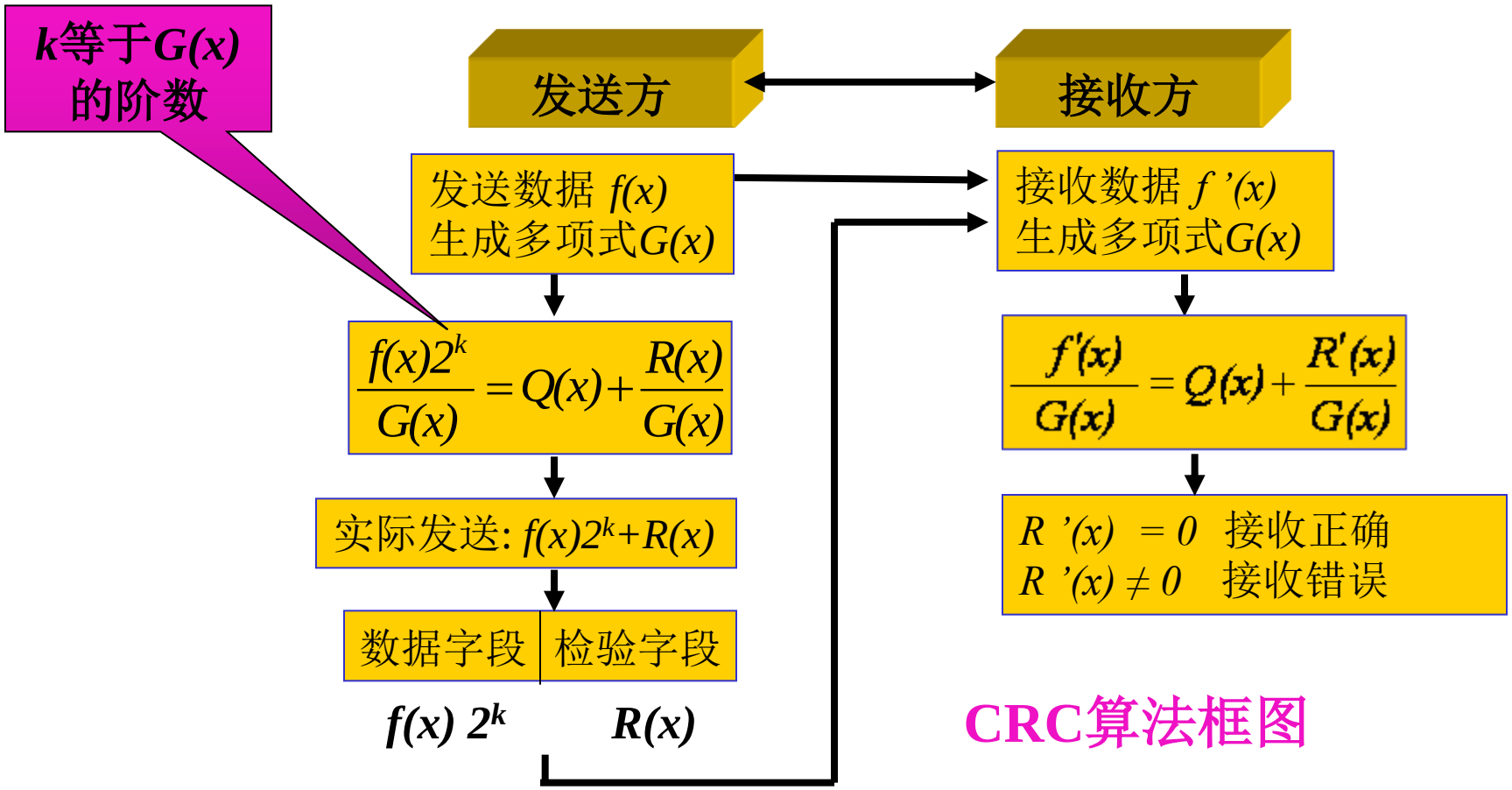
可以纠正所有的单比特错误



CRC--循环冗余校验

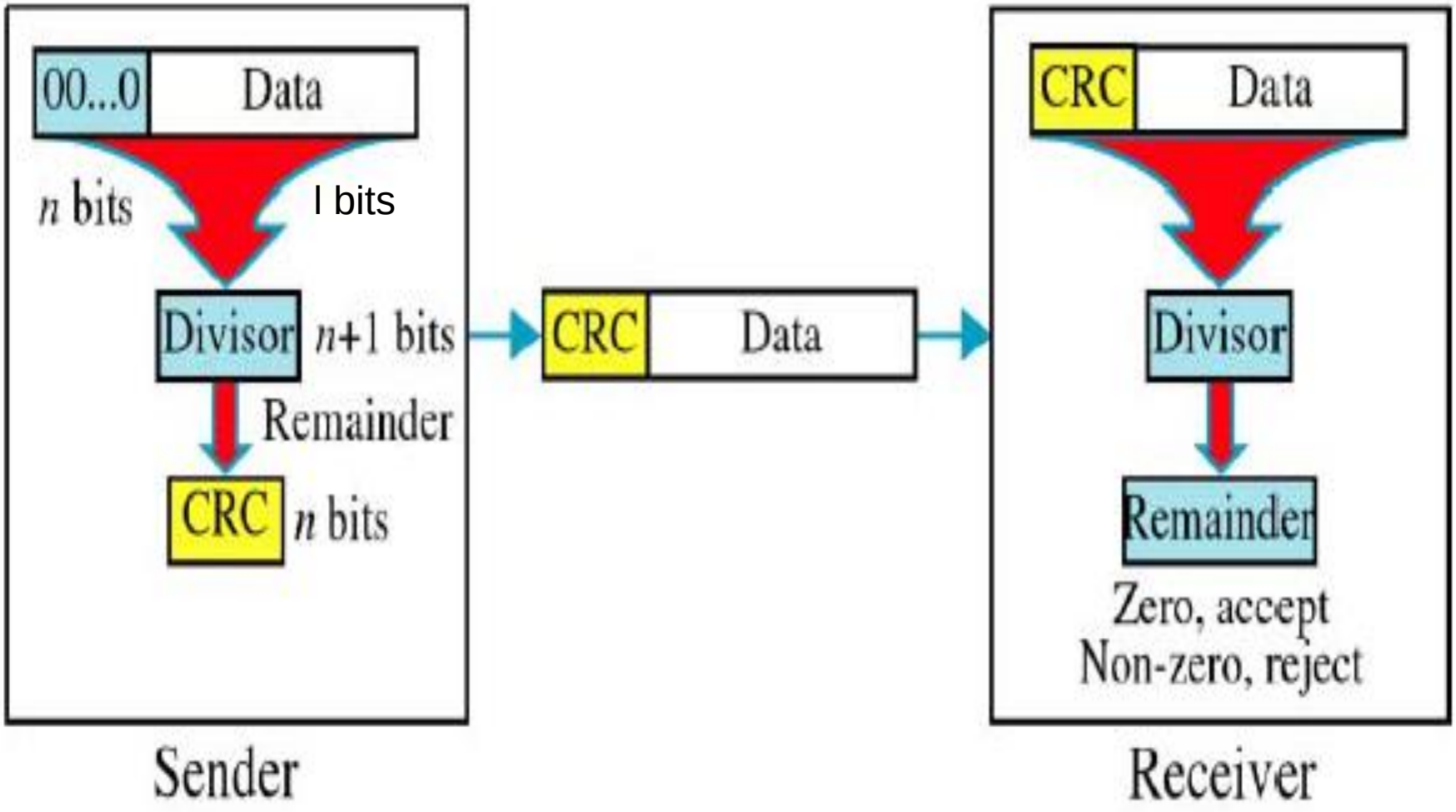
- 循环冗余码（CRC（Cyclic Redundancy Check））
- 多项式: 110001，表示为： $x^5 + x^4 + 1$
- CRC中的生成多项式G(x)
  - 发方、收方事前商定，双方需相同；
  - 生成多项式的最高位和最低位必须为1
  - 生成多项式必须比传输信息对应的多项式短。
- CRC基本思想：把整个数据块（数据+校验码）当作一个报文码多项式*f(x)*的系数，发送时用一个标准的生成多项式*g(x)*来除*f(x)*，将所得余数*r(x)*作为CRC校验码附加在报文码之后发出；接收时用同一生成多项式*g(x)*来除收到的码字多项式（数据+校验码），能除尽说明传输正确，否则说明有错。

CRC--循环冗余校验



CRC算法框图

CRC--循环冗余校验


00...0
Data
n bits
l bits
Divisor
n+1 bits
Remainder
CRC
n bits
Sender
CRC
Data
CRC
Data
Divisor
Remainder
Zero, accept
Non-zero, reject
Receiver

CRC--循环冗余校验

- 校验和计算算法
  - 设G(x)为 r 阶，在(m比特）数据帧的末尾添加 r 个0(校验位），使帧为m + r位，相应多项式为$2^rK(x)$;
  - 按模2除法用对应于G(x)的位串去除对应于$2^rK(x)$的位串;
  - 余数就是校验位
  -

模2的加、减法定义：

0＋0＝0　　0－0＝0

1＋0＝1　　1－0＝1

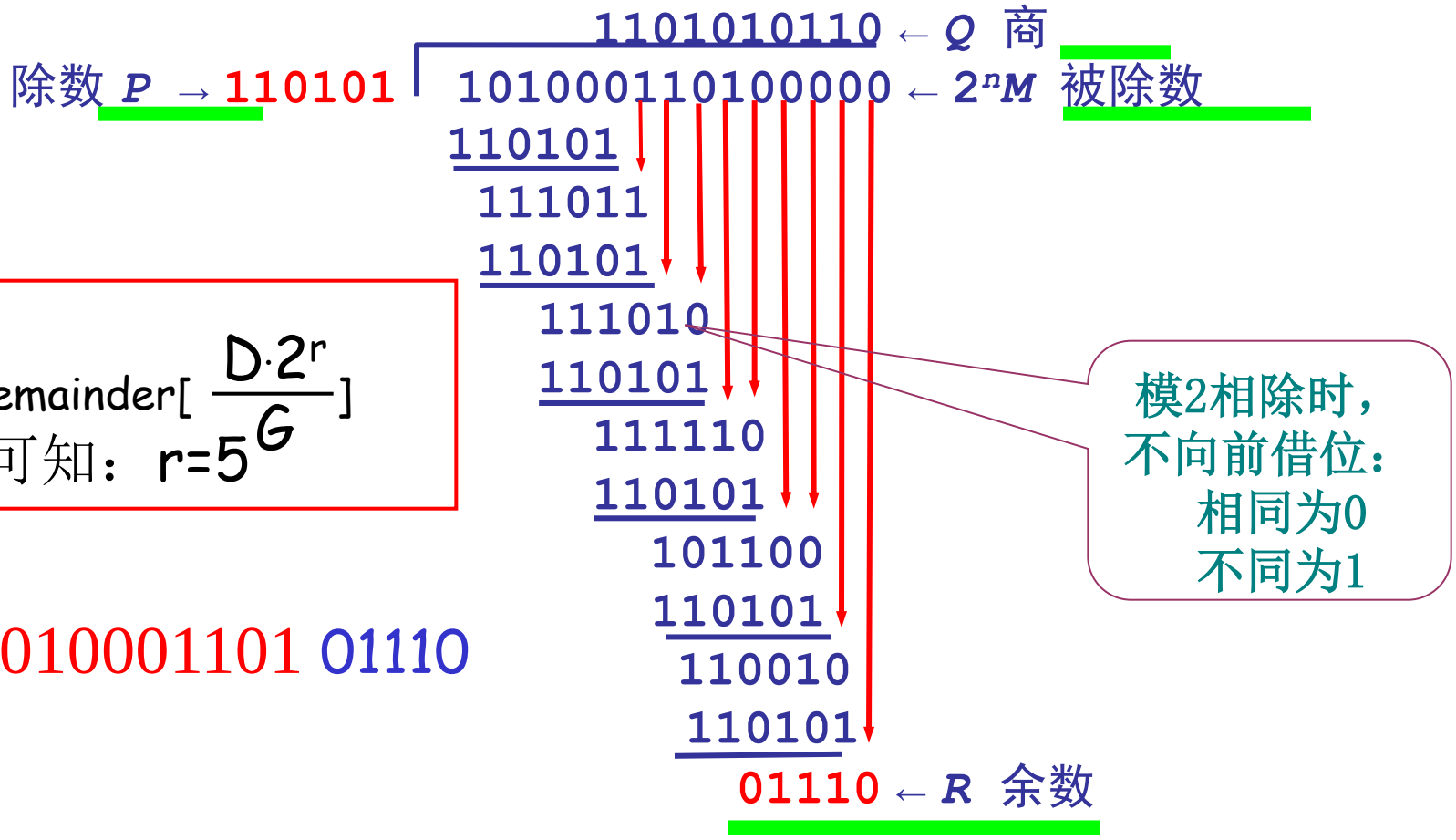
0＋1＝1　　0－1＝1

1＋1＝0　　1－1＝0

模2的加、减事实上与异或运算一样。

循环冗余检验的原理说明

Example: D= 1010001101, G = 110101



R = remainder[$\frac{D \cdot 2^r}{G}$]

由G可知： r=5

模2相除时，
不向前借位：
相同为0
不同为1

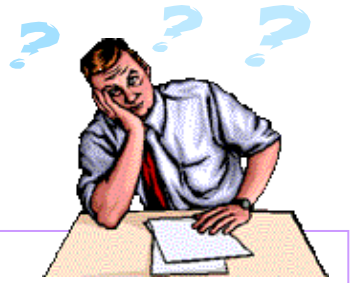
T = 1010001101 01110

循环冗余码的检错能力

- （n, k）循环码是码长为n，有k个数据码元的线性分组码
- 理论证明，CRC码能查出：
  - 全部的奇数个错误；
  - 所有的双位错（不相邻）；
  - 全部≤ n-k冗余位数的突发性错误；
  - 对于n-k+1位的突发性错误；检出率为$1-2^{-(n-k-1)}$；
  - 对于多于n-k+1位的突发性错误；检出率为$1-2^{-(n-k)}$；
- 实验表明，如果使用16位CRC码
  - 可以检出所有奇数位的差错及所有双位错，以及长度小于16位的突发错误，还能查出99.997%的17位和99.998% 18位或更长位的突发性错误。
  - 传输速率为9600bps时，传输3000年才会有一个错误。

CRC的实现

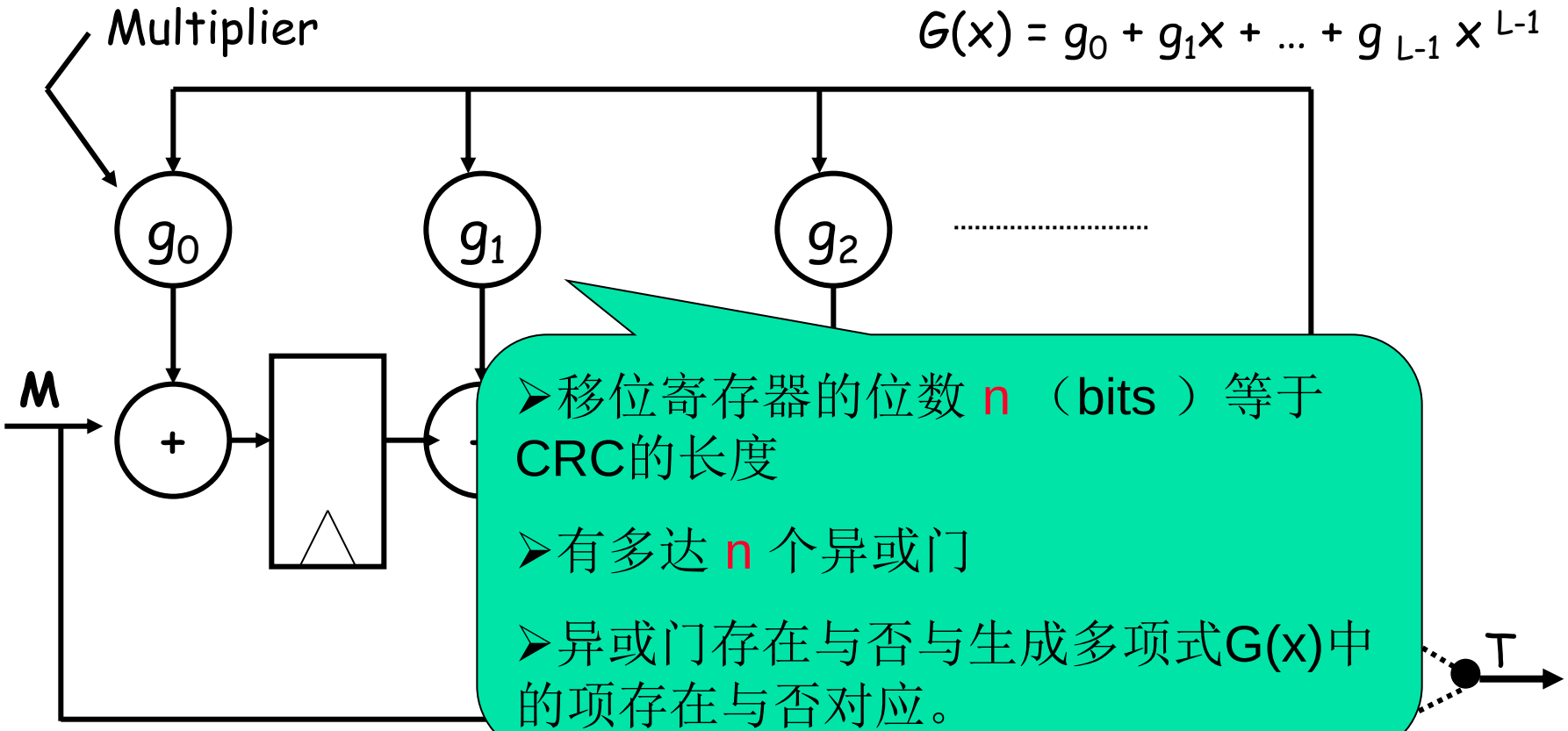
- ✓硬件实现
- ✓软件实现



思考题

自行编写或上网查找实现CRC的程序，输入具体的校验实例，给出程序的运行结果。

硬件:移位寄存器实现



- 移位寄存器的位数 n （bits ）等于CRC的长度
- 有多达 n 个异或门
- 异或门存在与否与生成多项式G(x)中的项存在与否对应。

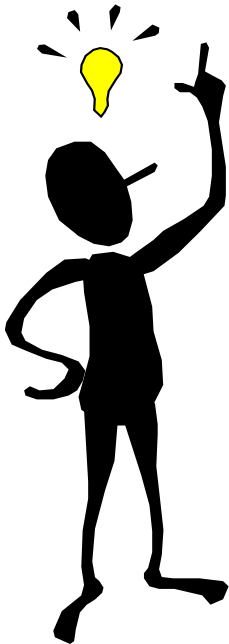
常用的CRC生成多项式

- CRC-12 用于传送6比特字符的流(streams)数据，产生12比特的FCS（帧校验码）----城域网
  - ✓ CRC-12: $X^{12}+X^{11}+X^3+X^2+X+1$
- CRC-16 和 CRC-CCITT 用于传送8比特字符的流数据，产生16比特的FCS----广域网
  - ✓ CRC-16: $X^{16}+X^{15}+X^2+1$ (USA)
  - ✓ CRC-CCITT: $X^{16}+X^{12}+X^5+1$ (Europe)
- CRC-32 使用 32 bit FCS。用于LAN和某些 DOD 应用。
  - ✓ CRC-32:

$$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$$

Internet 校验和算法

- 基本思路：将待校验的数据以16比特为单位，累加，取补，即为该数据块的校验和。
- 适用于IP/TCP等因特网协议



Error Correction

一般使用: Bose-Chaudhuri-Hocquenghem (BCH)

BCH (R + M, M, t)

校验位长度

信息位长度

码长

能检测所有 “t” 比特错误

e.g. BCH (1023, 923, 15)
R=100; M=923; t=15

如果 t=1 则该码称为： Reed-Solomon
一般用于： CD players

Detect or Correct?

Advantages of Error Detection（错误检测的优点）

- Requires smaller number of bits/overhead.
  （需要的错误检测位较短）
- Requires less/simpler processing（实现简单）

Advantages of Error Correction（纠错的优势）

- Reduces number of retransmissions（降低重传次数）.

Most data networks today use error detection, not error correction.（目前大多数网络均采用检错，而不是纠错）

4.4 流量控制

- 流量控制：**目的**是控制业务源发送数据的速率，使其与接收端的接收速率相匹配，也就是说业务源的数据传输率不能太快，以免淹没慢的接收端。
- 实现原理：**发送方节流**（throttled）

流量控制协议

开环控制：**漏桶、令牌桶技术**

闭环控制：

- **停等（Stop and Wait ）协议**
- **滑动窗口（Sliding Window）协议**
- **ON/OFF、端-端流量控制等**

漏桶、令牌桶流量控制

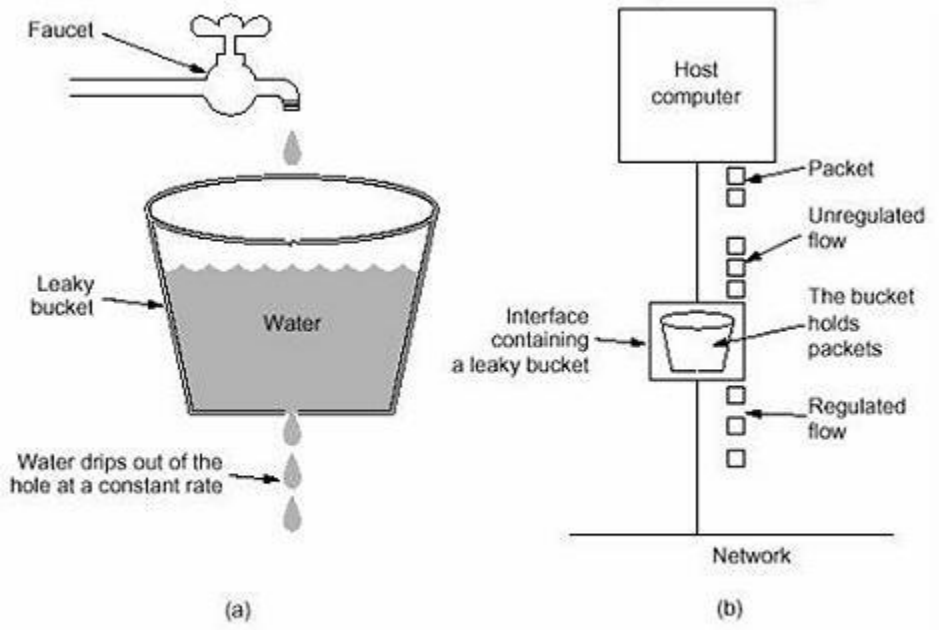
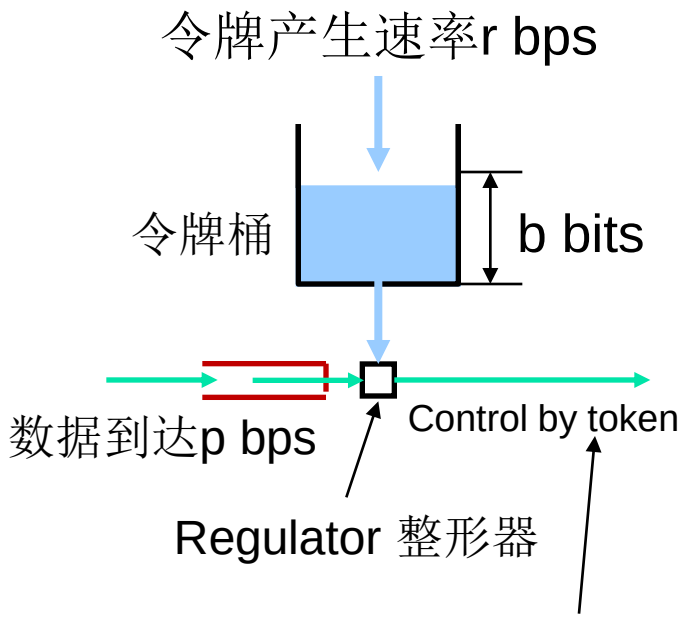


Fig. (a) A leaky bucket with water. (b) A leaky bucket with packets.

输入：突发、随机、不均匀；输出：恒定、均匀

图1 基于漏桶的流量控制

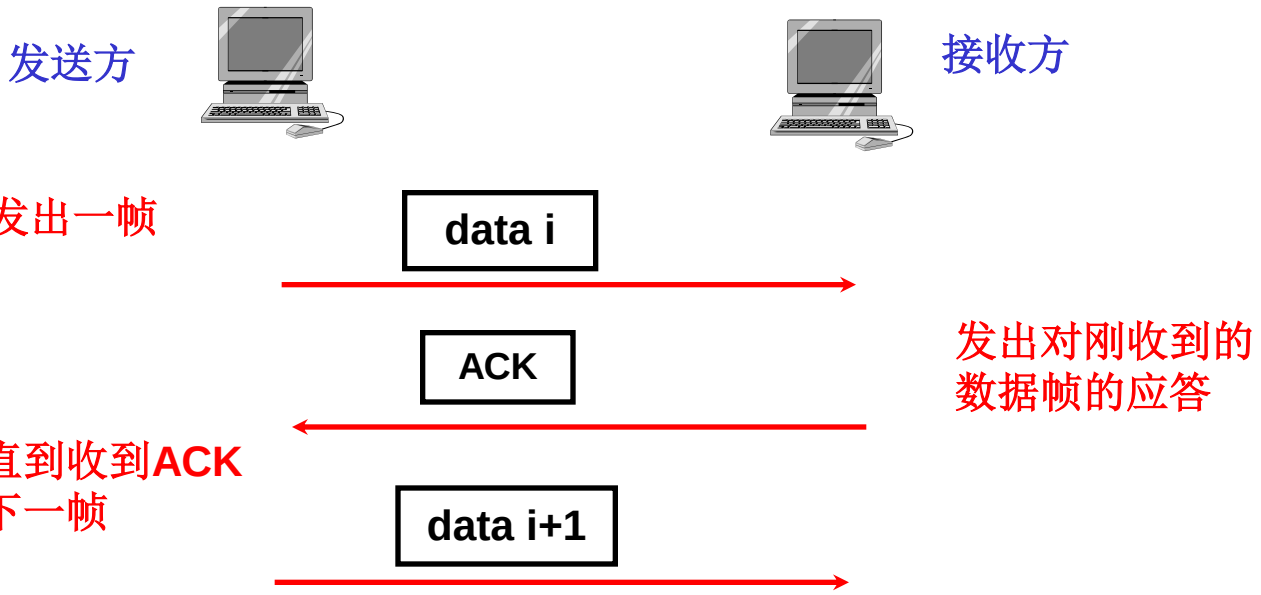


有多少令牌就能发送多少数据

图2 基于令牌桶的流量控制

停等协议

- 发送方发送一帧
- 停下来等待应答
- 如果正确，发送下一个帧
- 如果接收有错或报文丢失， 重传



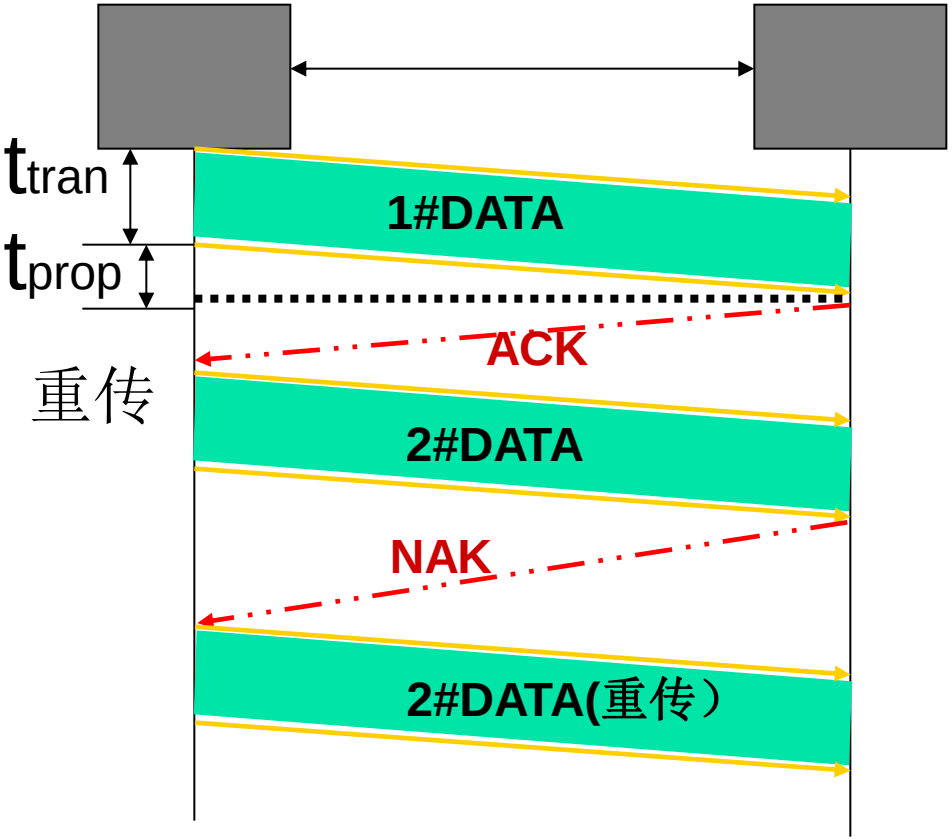
报文错误控制

- 几种出错可能
  - 数据帧**出错**
  - 数据帧丢失：
    - ✓ 数据帧**丢失**
    - ✓ 应答帧**丢失**
  - **重复**帧（收到两个相同的数据帧）
- ➢ 错误控制机制
  - ✓ ***ACK/NAK:*** 给发送方反馈发送结果—应答
  - ✓ ***Time-out:*** 报文（数据帧或应答帧）丢失
  - ✓ ***Sequence numbers:*** 区分重复报文，明确acked/nacked的报文序号

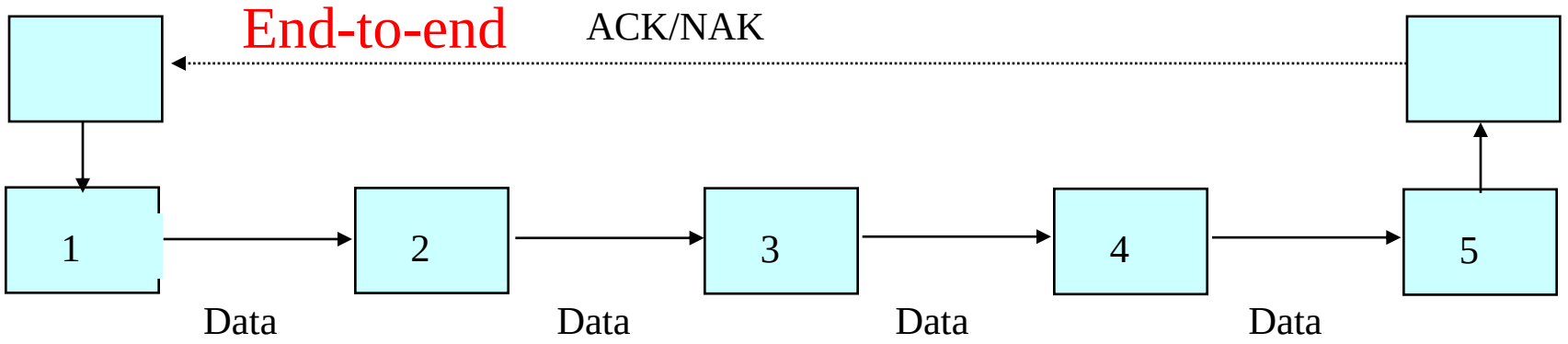
报文错误控制

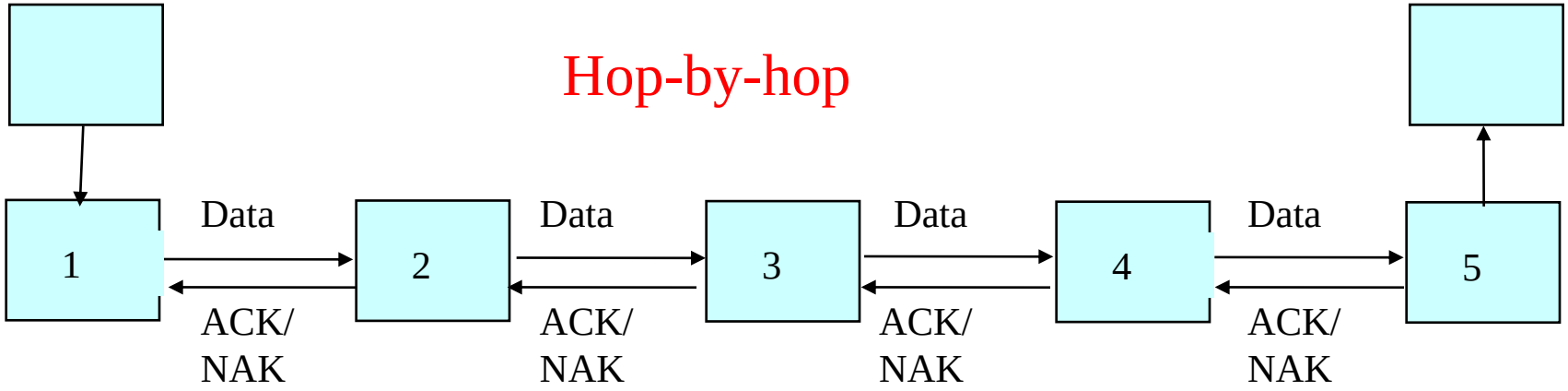
- 发送方在发送下一个帧之前等待肯定确认的协议叫做ARQ（Automatic Repeat reQuest：自动重传请求）

- 自动重传请求 (ARQ)
  - 发送端：发送后启动定时器
  - 接收端：错误检测
  - 应答(ACK/NAK)
  - 发送端：出错或丢失（超时）重传

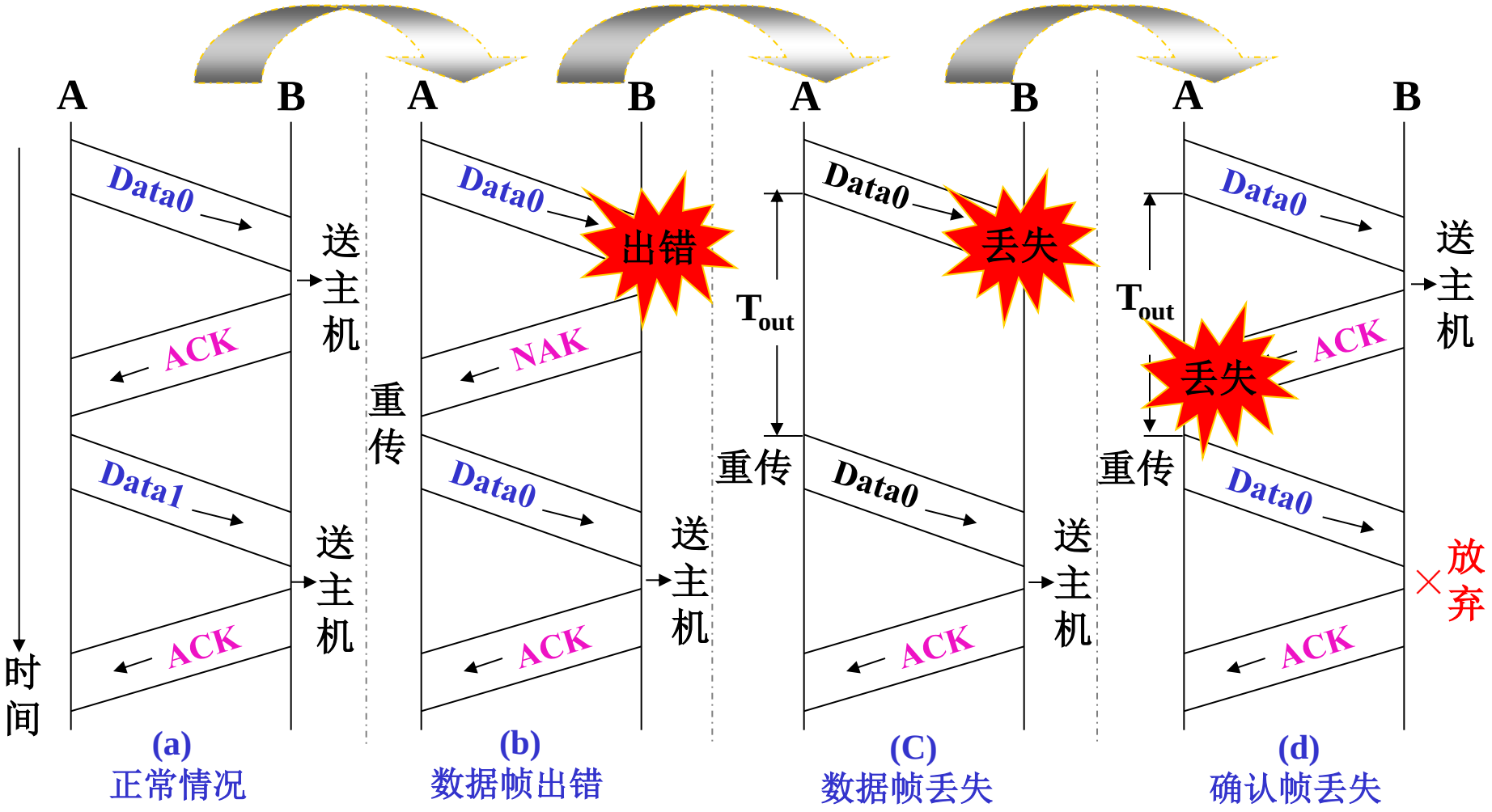


端--端应答与站--站应答

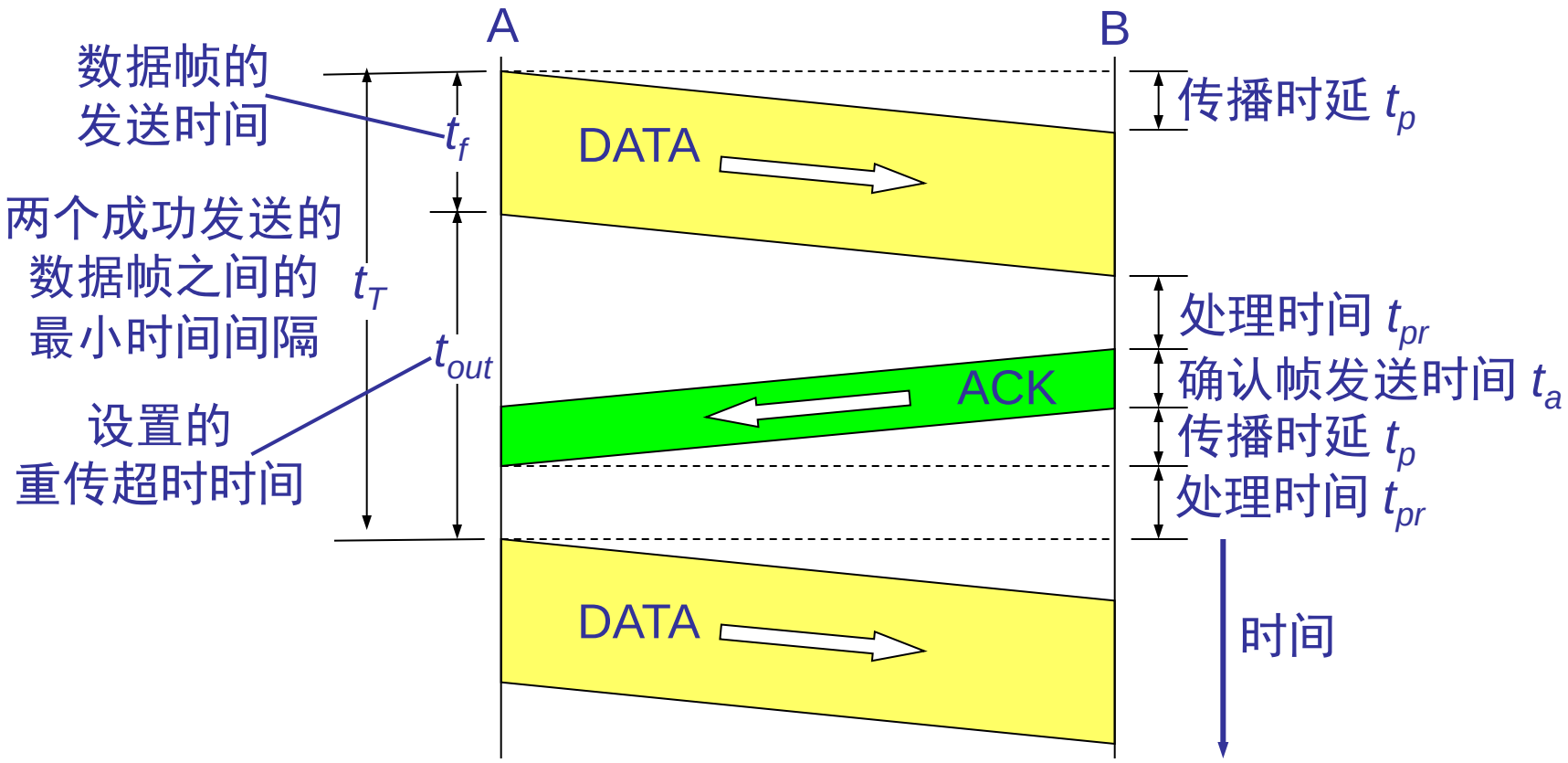

End-to-end
ACK/NAK
1
2
3
4
5
Data
Data
Data
Data


Hop-by-hop
1
2
3
4
5
Data
Data
Data
Data
ACK/
NAK
ACK/
NAK
ACK/
NAK
ACK/
NAK

停等 ARQ


A
B
Data0
ACK
送主机
Data1
ACK
送主机
时间
(a)
正常情况
A
B
Data0
出错
NAK
重传
Data0
ACK
送主机
(b)
数据帧出错
A
B
Data0
丢失
Tout
重传
Data0
ACK
送主机
(C)
数据帧丢失
A
B
Data0
送主机
Tout
丢失
ACK
重传
Data0
ACK
放弃
(d)
确认帧丢失

停止—等待协议的定量时延分析



$$t_T = t_p + t_f + t_{pr} + t_a + t_p + t_{pr} = t_f + \underbrace{t_p + t_{pr} + t_a + t_p + t_{pr}}_{t_{out}}$$

停止—等待协议的定量分析

令 $t_{out} = t_p + t_{pr} + t_\alpha + t_p + t_{pr}$

一般有： t_{pr} 和 $t_\alpha \ll t_p$ 则：$t_{out} = 2t_p$

所以 $t_T = t_f + t_{out} = t_f + 2t_p$

设数据帧出现差错(包括帧丢失)的概率为 p，则正确传输一个数据帧所需的平均时间为：

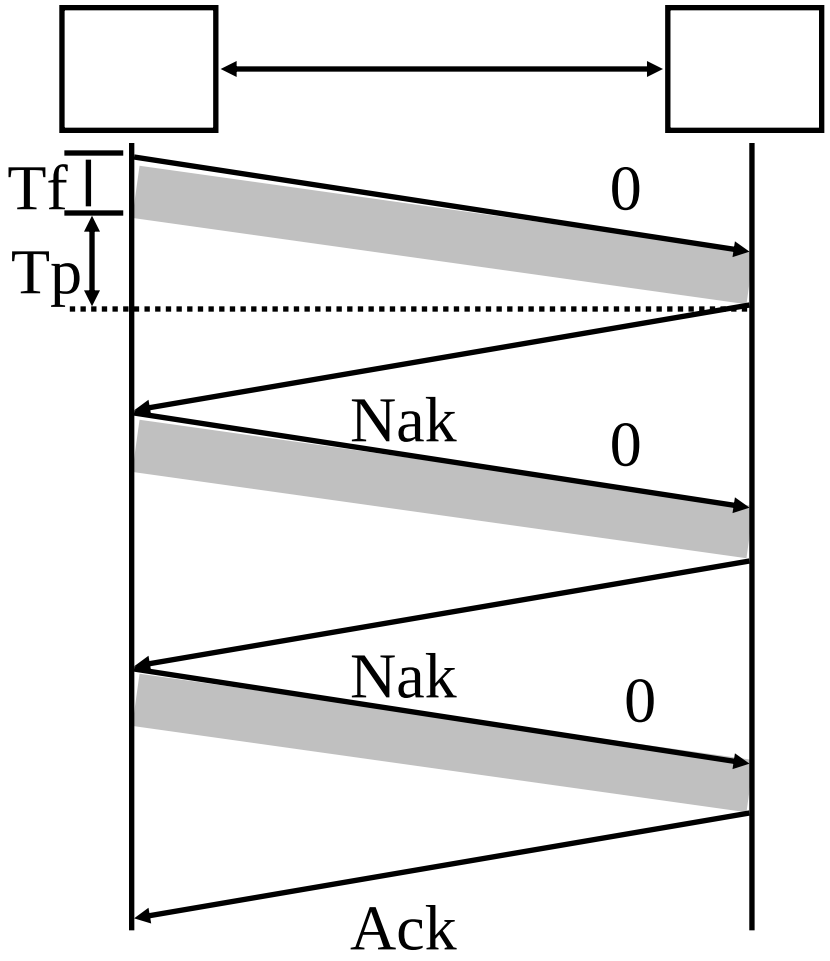
$$t_{avg} = t_T + (1-p)\sum_{i=1}^{\infty} i\, p^i t_T = \frac{t_T}{1-p}$$

由于链路的最大吞吐量λ_{max} 就是每秒成功发送的最大帧数

故有： $\lambda_{max} = \dfrac{1}{t_{avg}} = \dfrac{1-p}{t_T}$

停等 ARQ的性能

Tp=距离 / 信号传播速率
Tf=帧长 / 报文发送速率
真空中的光信号 = 300 m/μs
一般近似认为电缆中的信号传播速度为2/3光速



- P= 数据帧错误的概率
- $\alpha = T_p/T_f$
- $U=T_f/[N_r(T_f+2T_p)]$
  $= 1/[N_r(1+2\alpha)]$
- $N_r=\Sigma\ i\ P^{i-1}(1-P)$
  $=1/(1-P)$
- **$U=(1-P)/(1+2\alpha)$**

停等协议信道利用率分析

- ✓ *假定重传概率P=0*
- ✓ *卫星链路*: 传播时延 t_{prop} = 270 ms

  Frame Size = 4000 bits = 500 bytes

  Data rate = 56 kbps $\Rightarrow$ t_{frame} = 4/56 = 71 ms

  $\alpha = t_{prop}/t_{frame}$ = 270/71 = 3.8

  $U = 1/(2\alpha+1)$ = ***0.12*** (太低 !!)
- ✓ *短程链路*: 局域网：距离1 km = 5 μs,

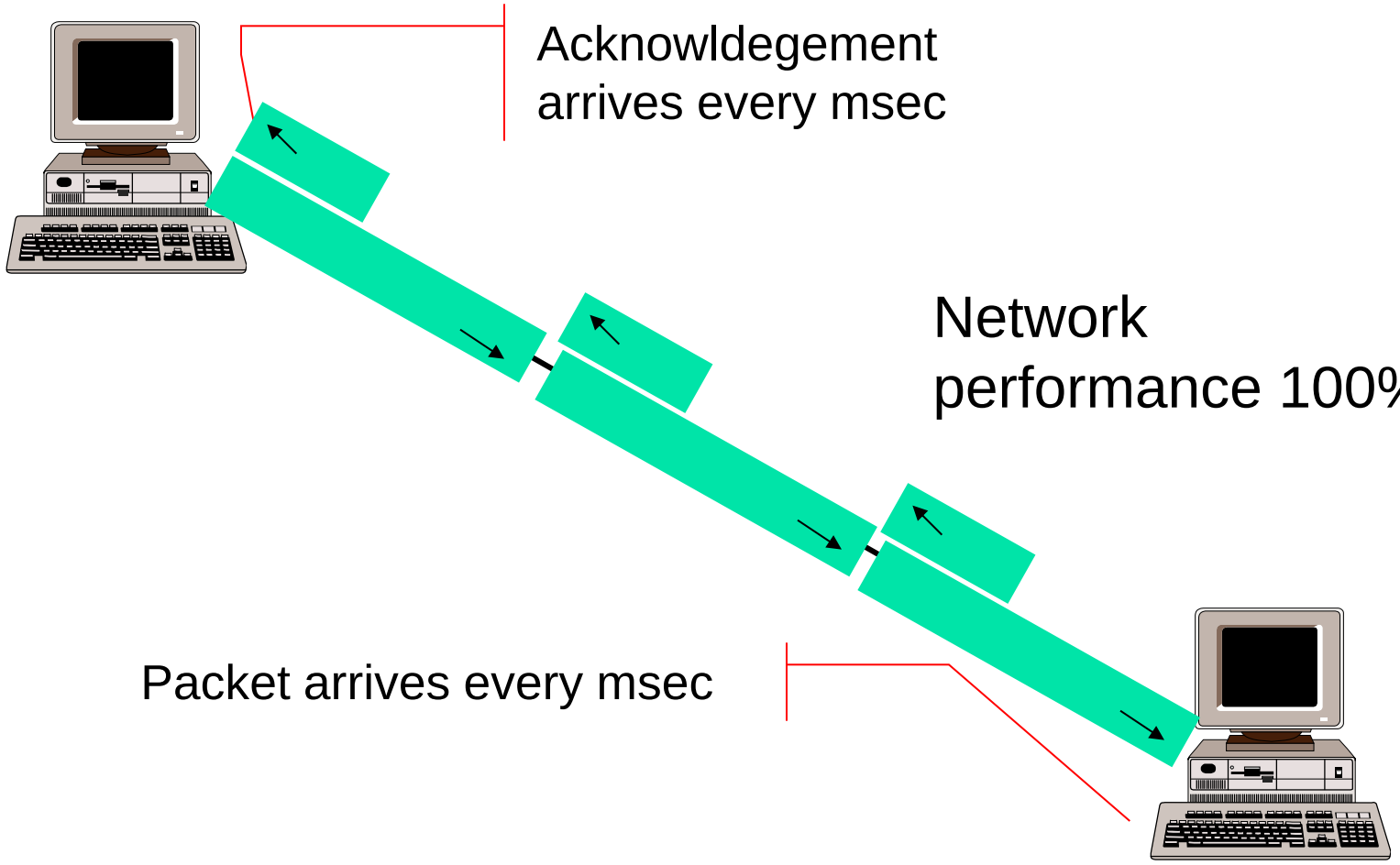
  Rate=10 Mbps,

  Frame=500 bytes $\Rightarrow$ t_{frame}= 4k/10M= 400 μs

  $\alpha=t_{prop}/t_{frame}$=5/400=0.012

  $\Rightarrow$ $U=1/(2\alpha+1)$= ***0.98*** (理想!)

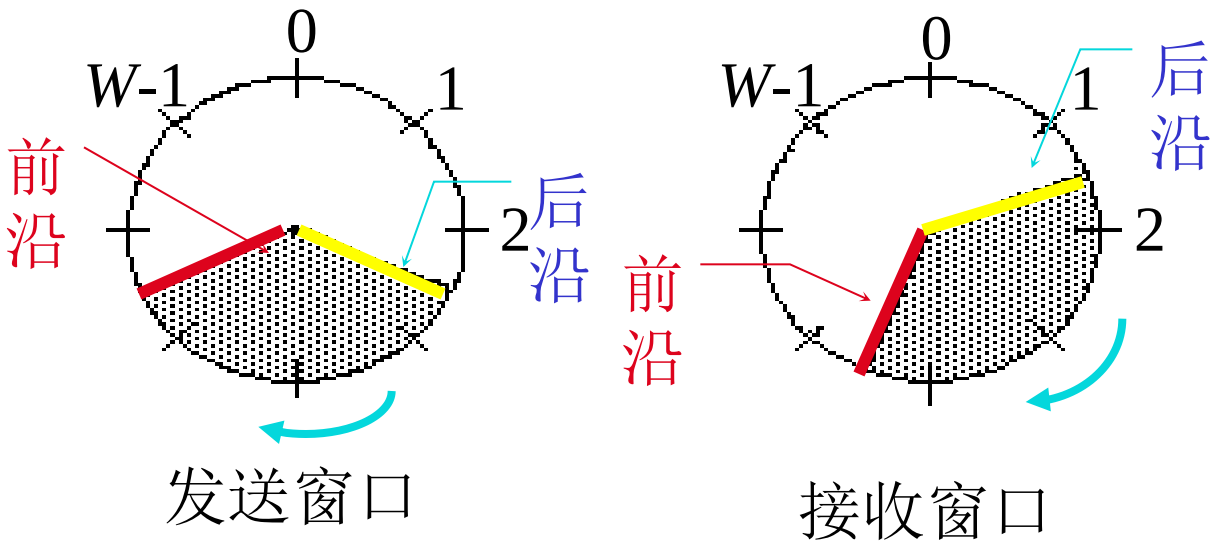
Pipelining



Pipelining— Sliding Window Flow Control

滑动窗口协议

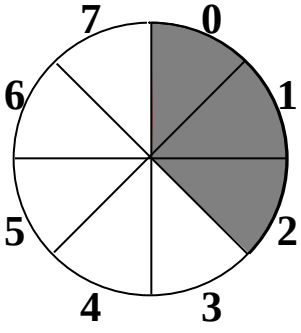
- ✓ Window = 允许发送/接收的一组报文的序号
- 发送窗口：允许发送的报文的序号
- 接收窗口：允许接收的报文的序号



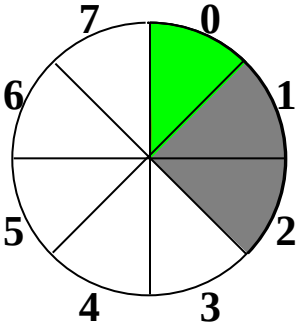
窗口名称	窗口的作用	最大窗口尺寸（允许发送的帧）	窗口实际张开的大小	窗口的改变方式
发送窗口	（1）指定允许发送的帧的序号 （2）存放已发出，但还没有收应答的帧的序号	2^n-1（GO-BACK-N协议） 或 $2^{(n-1)}$（select repeat协议） 表示序号在最大发送窗口内的帧到达时均可以发送	（实际发送出去的帧的序号） （0 ~最大窗口）可变	*初始时*：允许发送最大窗口等量的数据帧；但由于没有发出任何帧，所以（实际）发送窗口闭合； *发送一个帧*：发送窗口的前指针前进一格； *收到已发出的最小序号帧的应答*：发送窗口的后指针前进一格；
接收窗口	存放允许接收的帧的序号 （包括允许接收的范围和数量）	$2^{(n-1)}$ 表示允许接收的帧的序号	$2^{(n-1)}$ 固定	*初始时*：接收窗口张开 $2^{(n-1)}$ 格，表示准备接收 $2^{(n-1)}$ 个帧； *收到一个帧*：如果该帧的序号在接收窗口内，则允许接收并保存下来，否则丢弃； *发送应答*：当位于接收窗口内的序号最小的帧（或连续多个帧）收妥后，发送该（或连续多个）帧的应答，接收窗口的前后沿同时前进一格（或多格）

滑动窗口概念

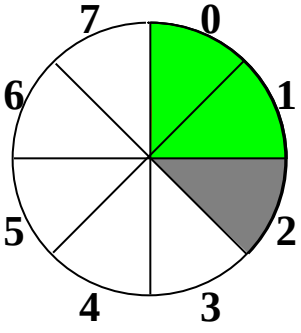
帧的序号由3比特组成（报文序号0~7），发送窗口和接收窗口的最大尺寸都为3，表示允许连续发送或接收3个帧。



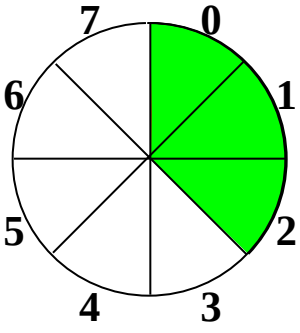
允许发送3个帧，但都没有发出



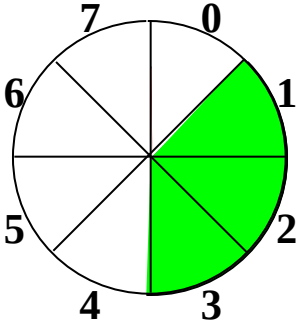
发出1#帧



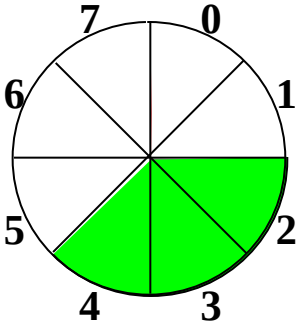
发出2#帧，应答都没有达到



发出3#帧，应答都没有达到

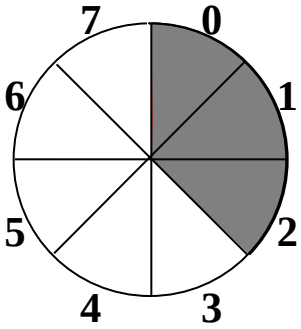


收到1#帧的应答，发窗前移一格，发出4#帧

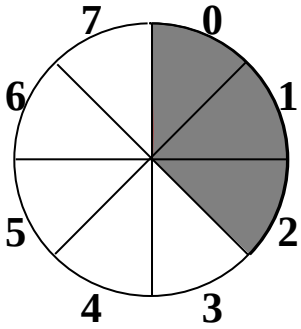


收到2#帧的应答，发窗前移一格，发出5#帧

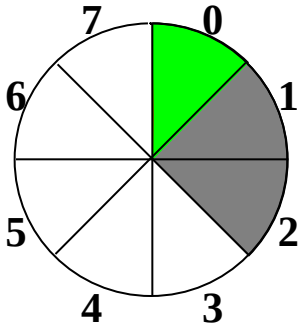
发送端发送窗口



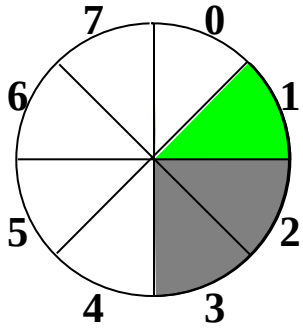
允许接收3个帧，但帧都没有到达



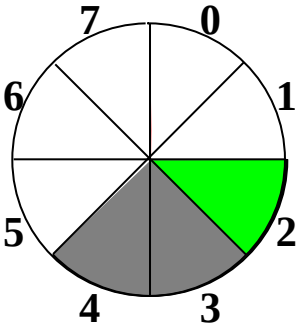
允许接收3个帧，但帧都没有到达



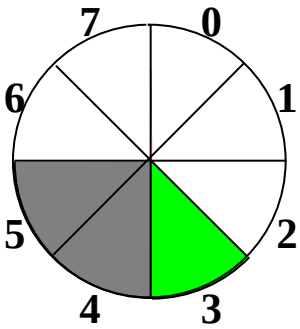
1#帧到达，接收



发出1#帧应答，收窗前移一个，允许接收2-4#帧；2#到达，接收



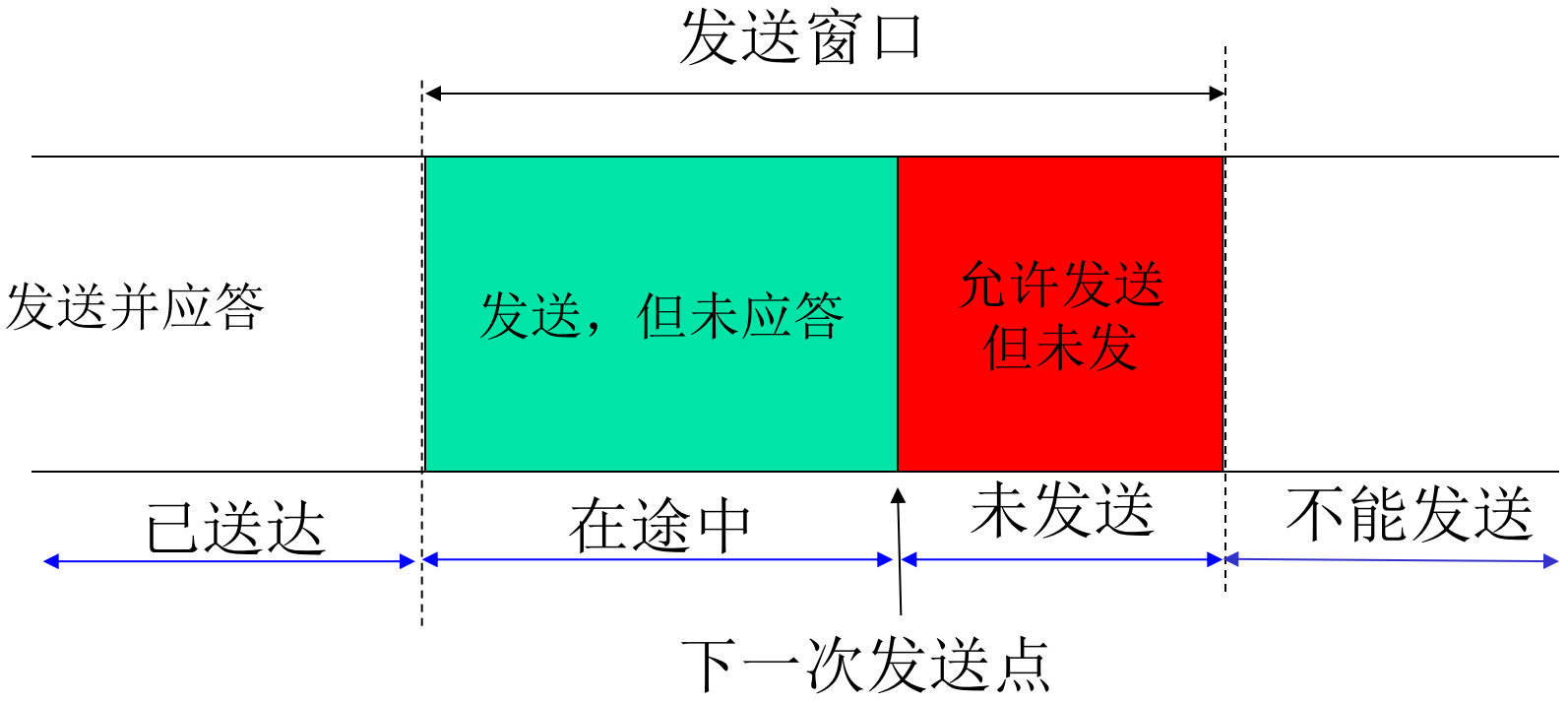
发出2#帧应答，收窗前移一个，允许接收3-5#帧；3#到达，接收



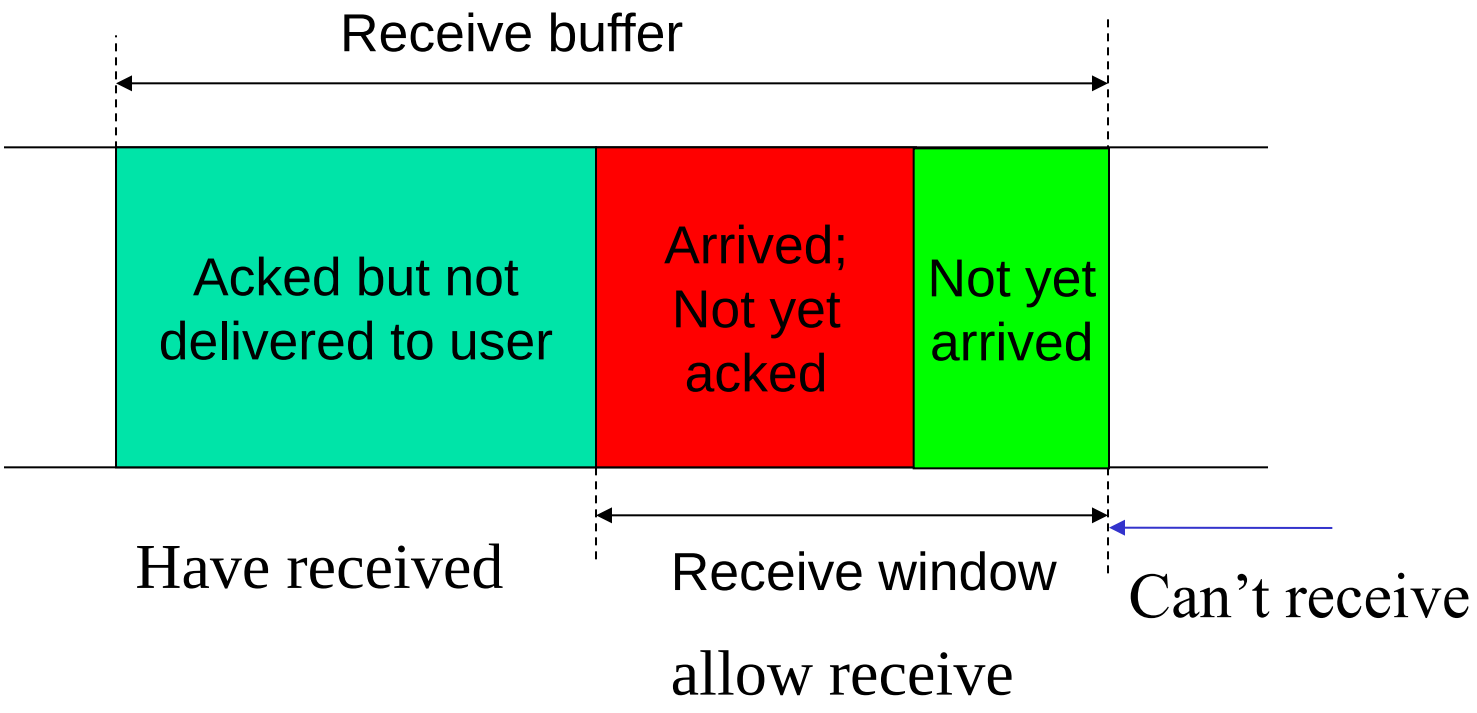
发出3#帧应答，收窗前移一个，允许接收4-6#帧；4#到达，接收

接收端接收窗口

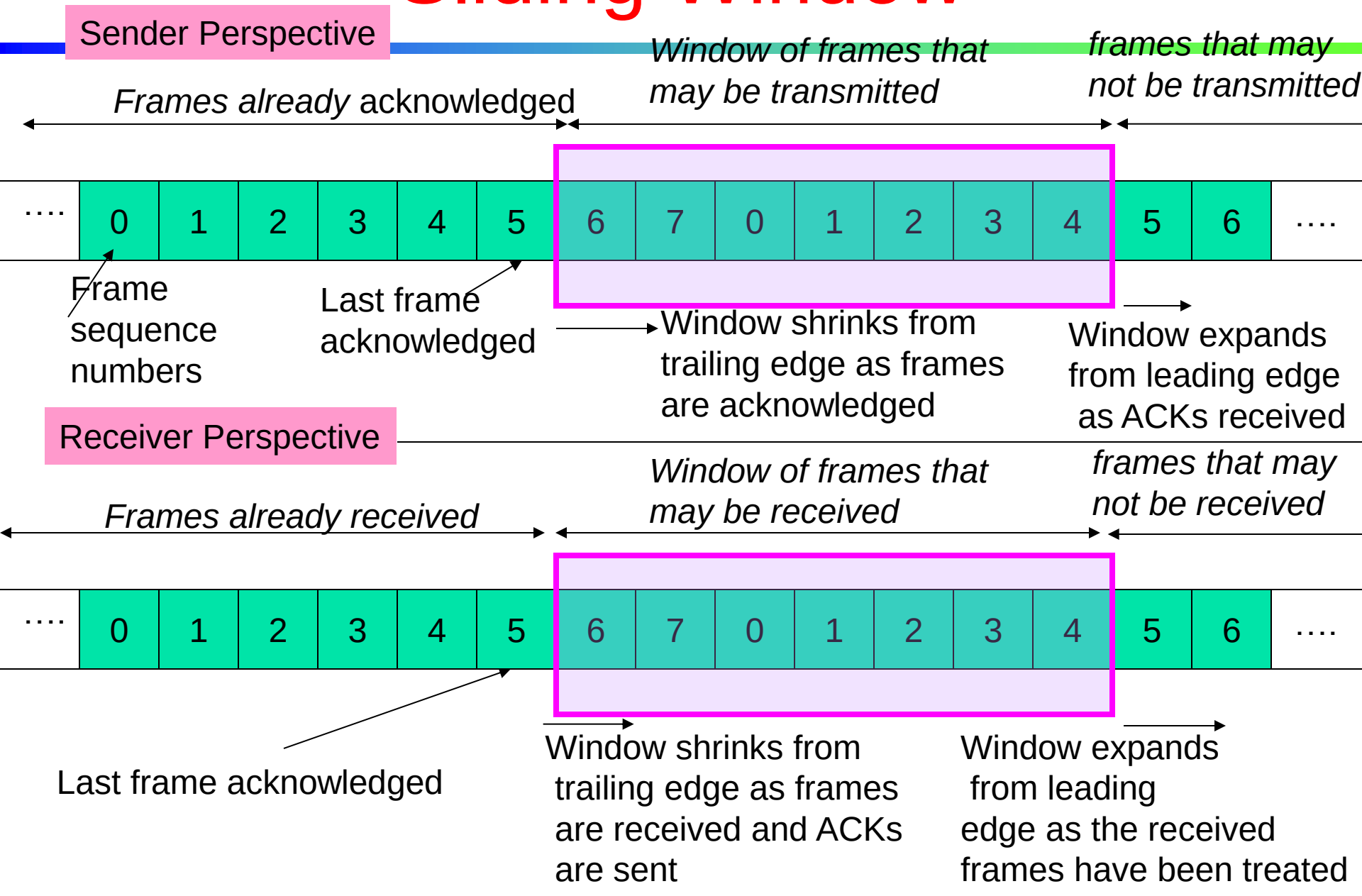
发送窗口



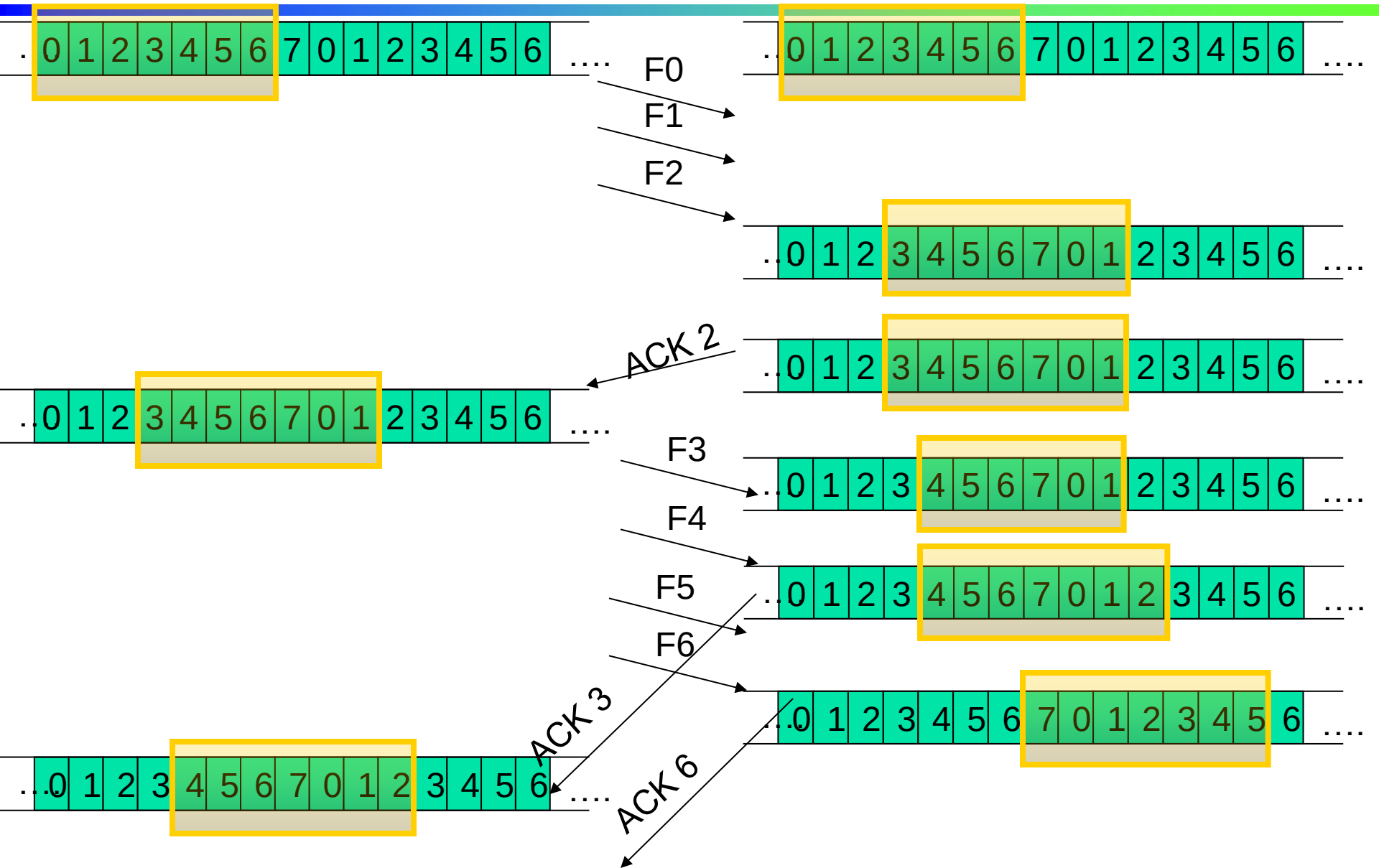
接收窗口


Receive buffer
Acked but not delivered to user
Arrived; Not yet acked
Not yet arrived
Have received
Receive window
allow receive
Can't receive

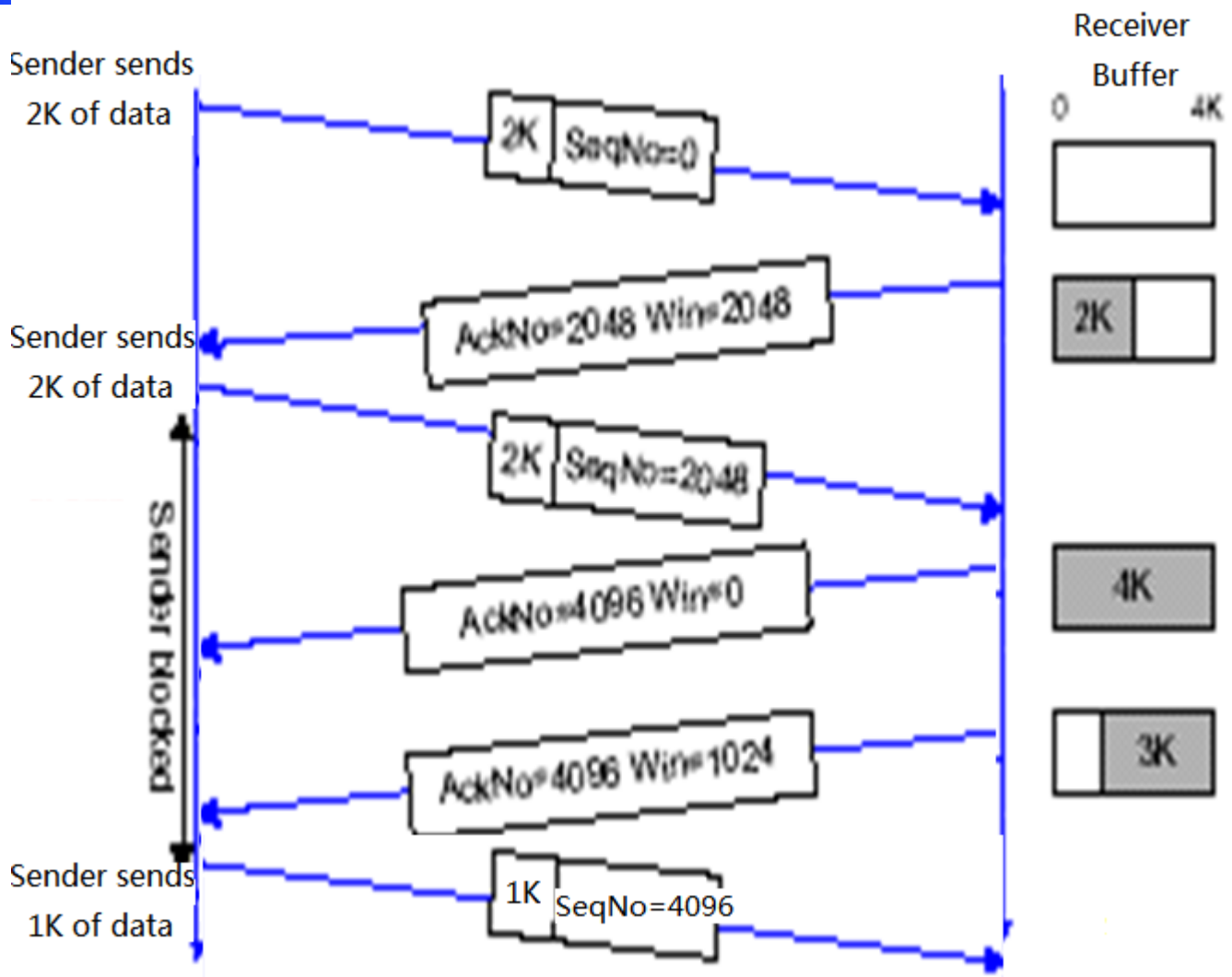
Sliding Window



Sliding Window: Example



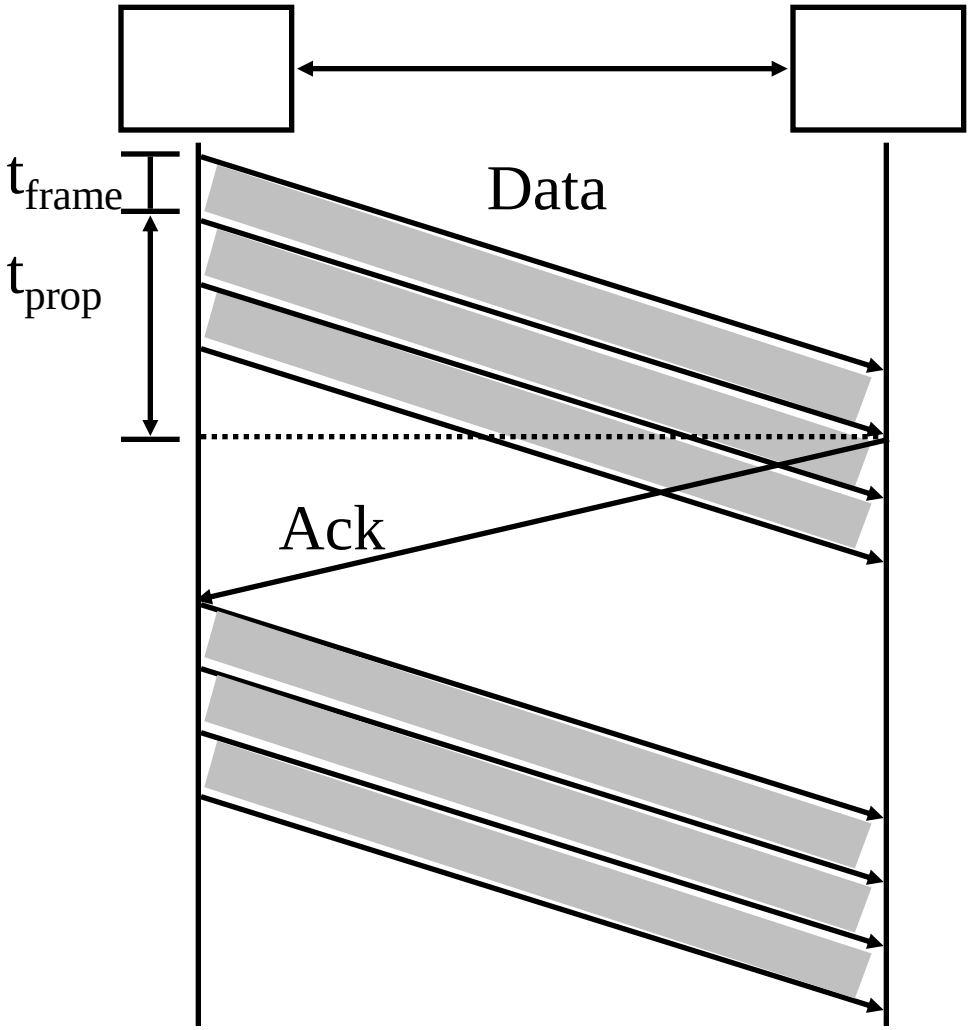
TCP中基于滑动窗口的流量控制



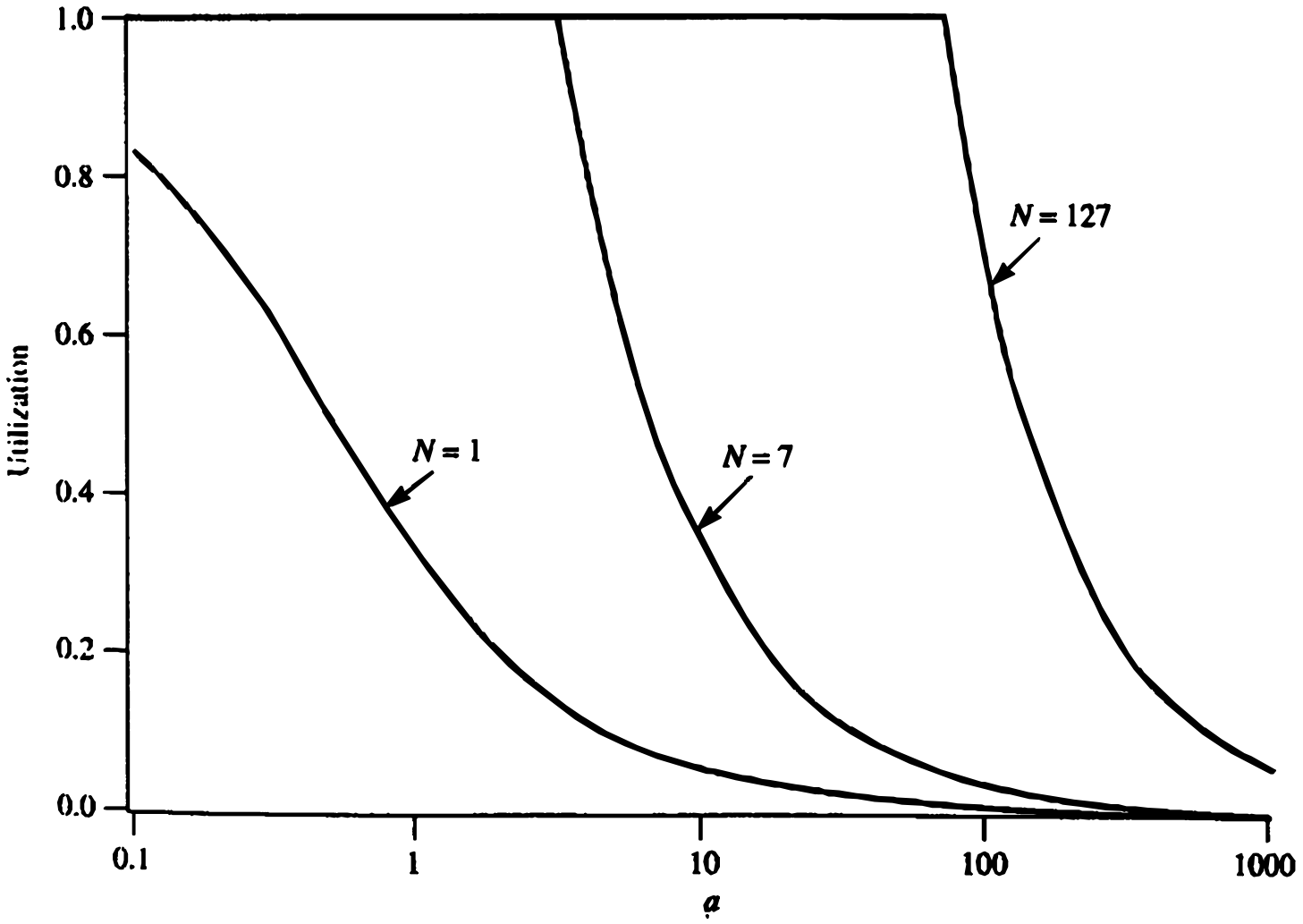
滑动窗口协议效率

$$U = \frac{Nt_{frame}}{2t_{prop}+t_{frame}}$$

$$= \begin{cases} \dfrac{N}{2\alpha+1} \\ 1 \text{ if } N>2\alpha+1 \end{cases}$$



窗口尺寸的影响



滑动窗口协议的重传控制

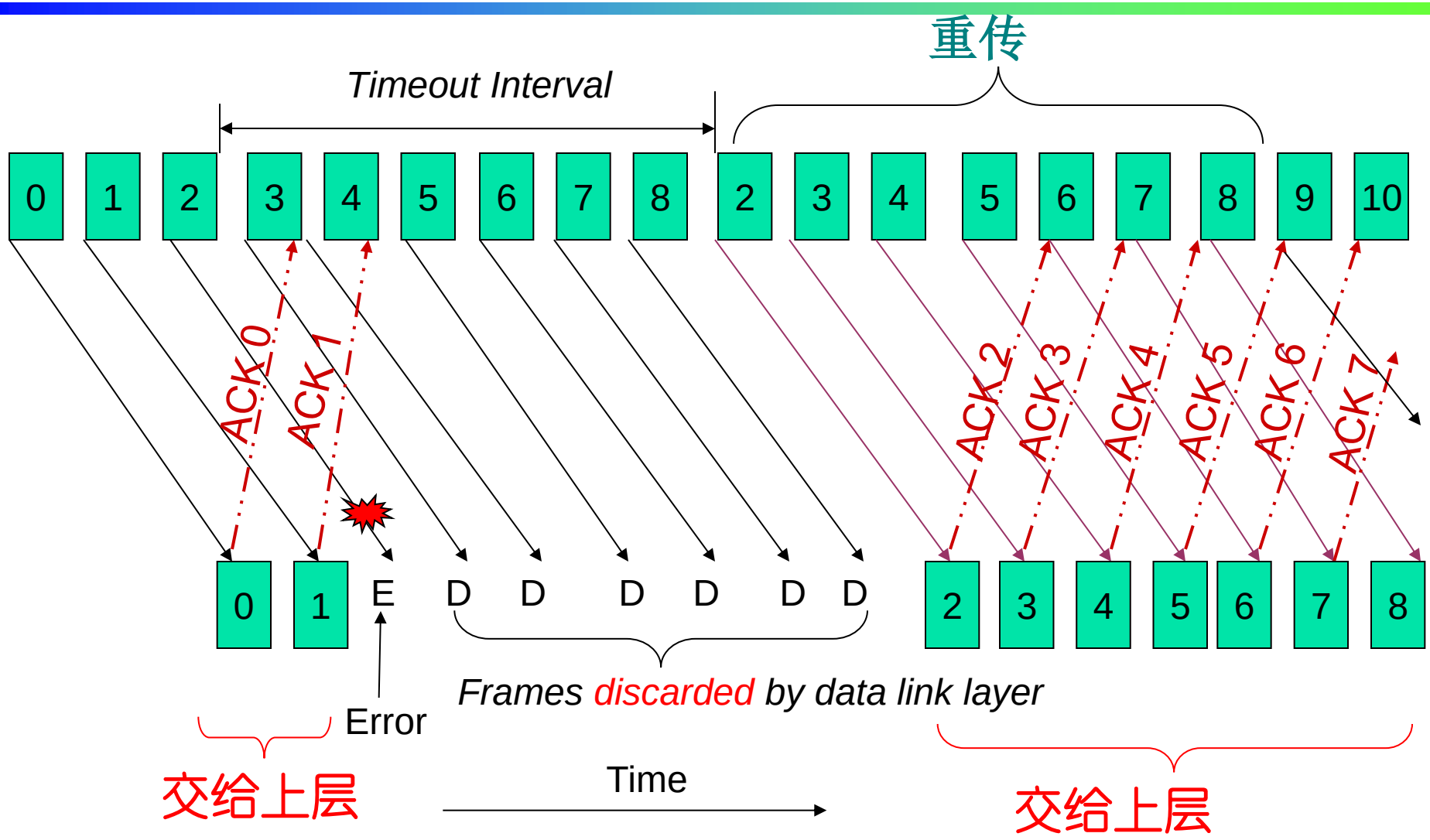
Go-back-N ARQ

那里出错，就退回到那里，重传从出错处开始的所有后续帧

Selective Repeat ARQ

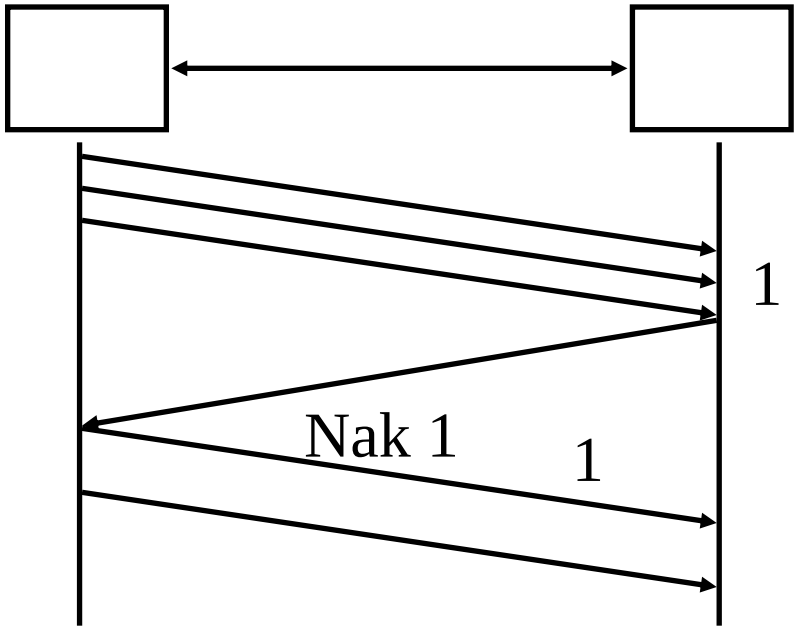
那个帧出错，就重传那个帧

Go-Back-N: Example



优点：不需要缓存； 缺点：重传的报文较多

Go-back-N的性能



GO-BACK-N的窗口大小

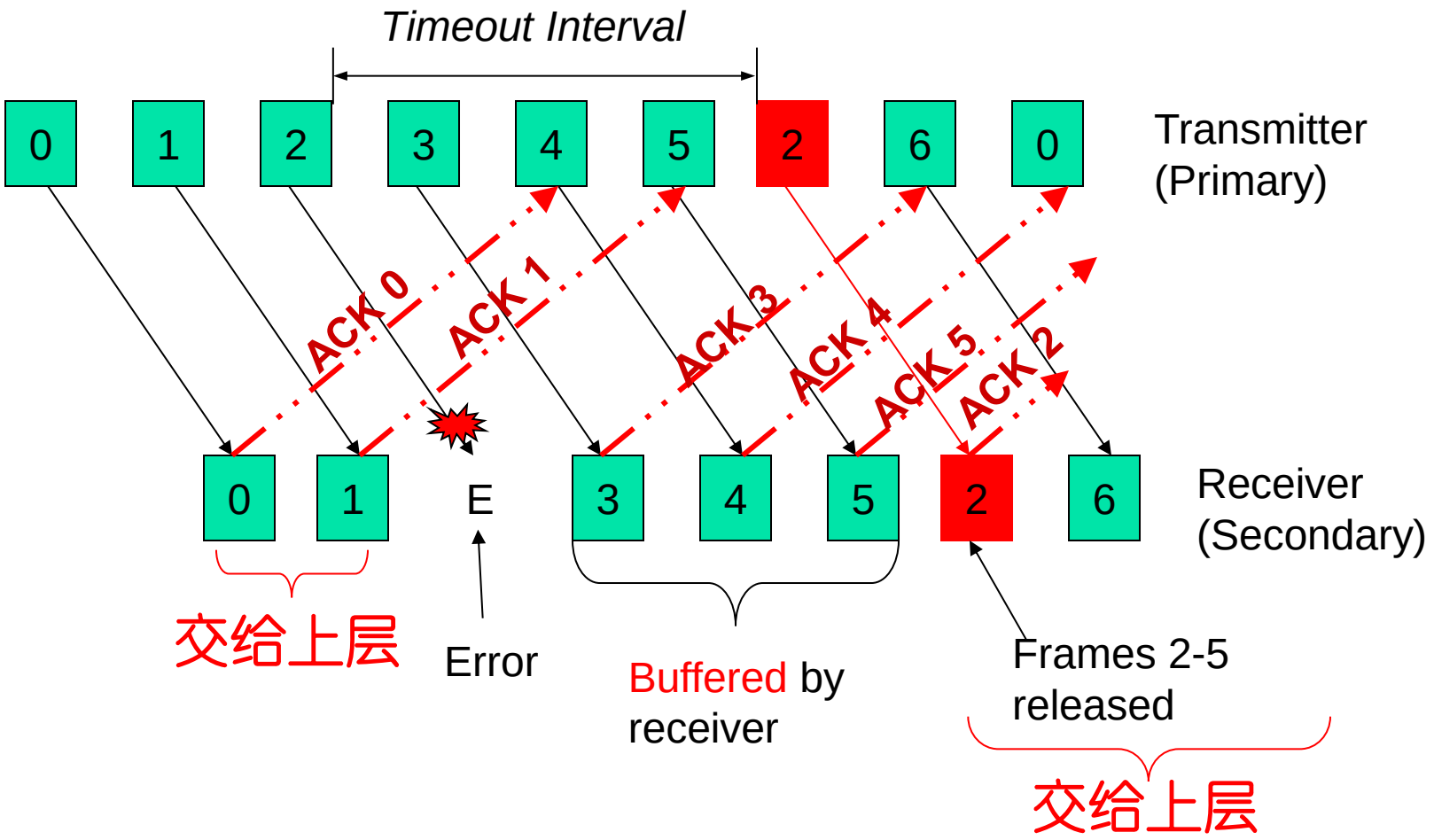
当帧的顺序号用n 比特编号时，要使滑动窗口协议能正确工作，必须满足下式：

$$W_T + W_R \leq 2^n$$

对于GO-BACK-N协议，因为必须向主机按序递交帧中携带的数据，即不能接收乱序帧，可见其$W_R = 1$，所以 $W_T \leqslant 2^n - 1$。

- Frames Retransmitted = $\begin{cases} 2\alpha+1 & \text{if } N>2\alpha+1 \\ N & \text{otherwise} \end{cases}$
- U = $\begin{cases} (1-P)/(1+2\alpha P) & \text{if } N>2\alpha+1 \\ N(1-P)/[(2\alpha+1)(1-P+NP)] & \text{otherwise} \end{cases}$

Selective Repeat Example



优点：重传的报文较多； 缺点：不需要缓存

选择重传 ARQ的性能

思考题：最大发送窗口大于 $2^{(n-1)}$ 时将会怎样？

选择重传ARQ的窗口大小

最大发送窗口： $2^{(n-1)}$

接收窗口： $2^{(n-1)}$

- 无错：

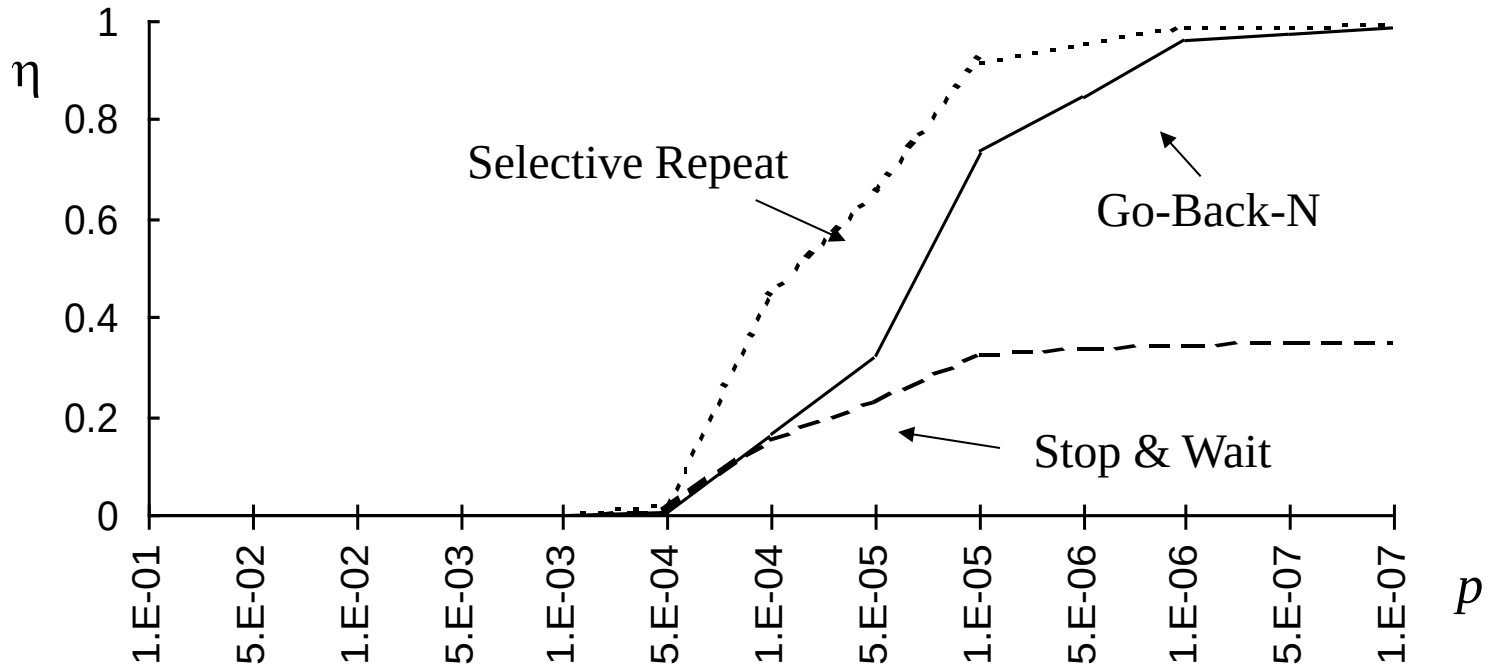
$$U=\begin{cases}1 & \text{if } N>2\alpha+1\\ N/(2\alpha+1) & \text{otherwise}\end{cases}$$

- 有错：

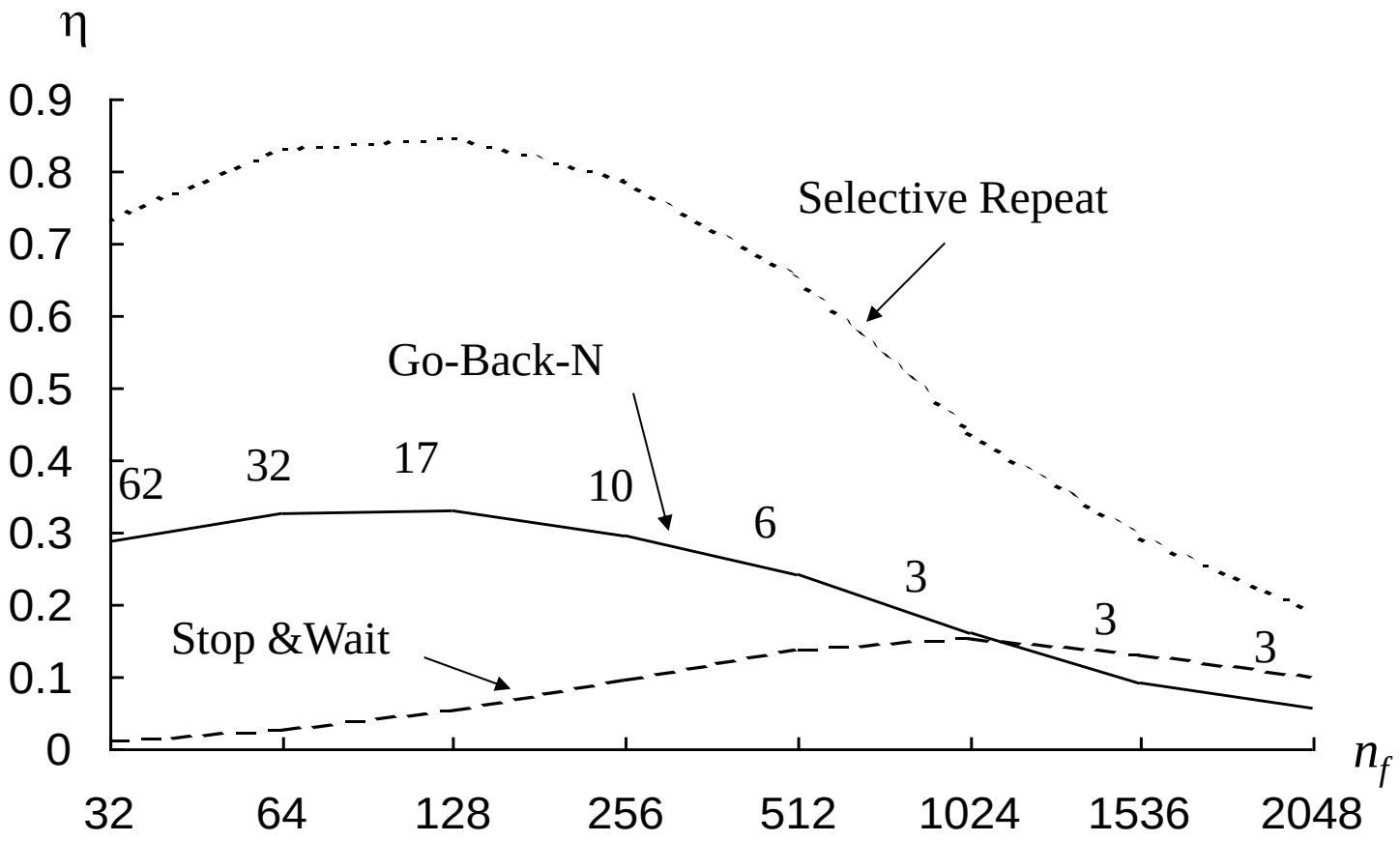
$$N_r=\Sigma\, i\, P^{i-1}(1-P) = 1/(1-P)$$

- $$\mathbf{U=}\begin{cases}\mathbf{(1-P)} & \mathbf{if\ N>(1+2\alpha)}\\ \mathbf{N(1-P)/(1+2\alpha)} & \text{otherwise}\end{cases}$$

ARQ协议的传输效率


η
1
0.8
0.6
0.4
0.2
0
Selective Repeat
Go-Back-N
Stop & Wait
1.E-01
5.E-02
1.E-02
5.E-03
1.E-03
5.E-04
1.E-04
5.E-05
1.E-05
5.E-06
1.E-06
5.E-07
1.E-07
p

帧长优化


η
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Selective Repeat
Go-Back-N
Stop &Wait
62
32
17
10
6
3
3
3
32
64
128
256
512
1024
1536
2048
n_f

Thanks !