第四章 点到点无差错传输和数据链路层

“101计划”
计算机网络

回顾

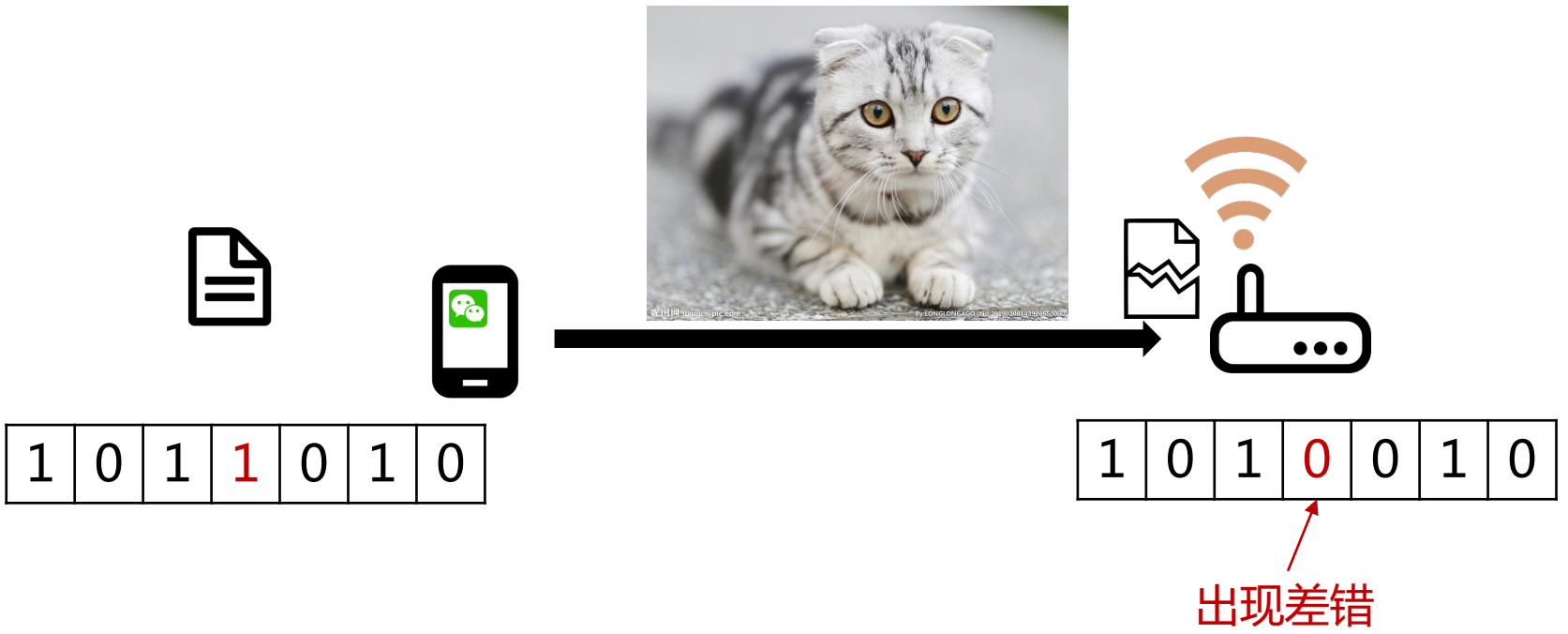
Xiamen University

❖ 数据链路层（Data Link Layer）

- 实现**相邻**节点间的数据传输
- 封装成帧：从物理层的比特流中提取完整的帧
- 透明传输：传输内容对帧定界透明
- 差错检验：检测和纠正比特传输错误
- 共享信道访问控制：同一信道同时传输信号，如同一间教室内，多人同时发言，需要纪律来控制
- 物理地址（MAC address）：48位，理论上唯一网络标识，烧录在网卡，不便更改
- ……

应用层 Application Layer
表示层 Presentation Layer
会话层 Session Layer
传输层 Transport Layer
网络层 Network Layer
数据链路层 Data Link Layer
物理层 Physical Layer

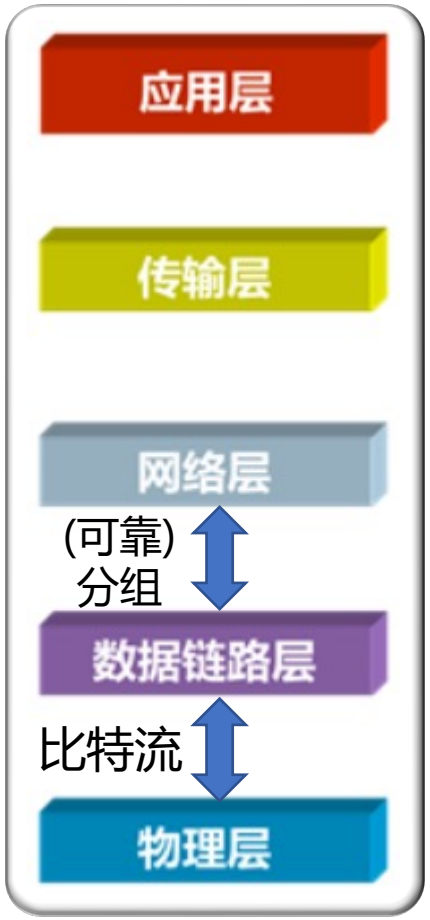
网络猫

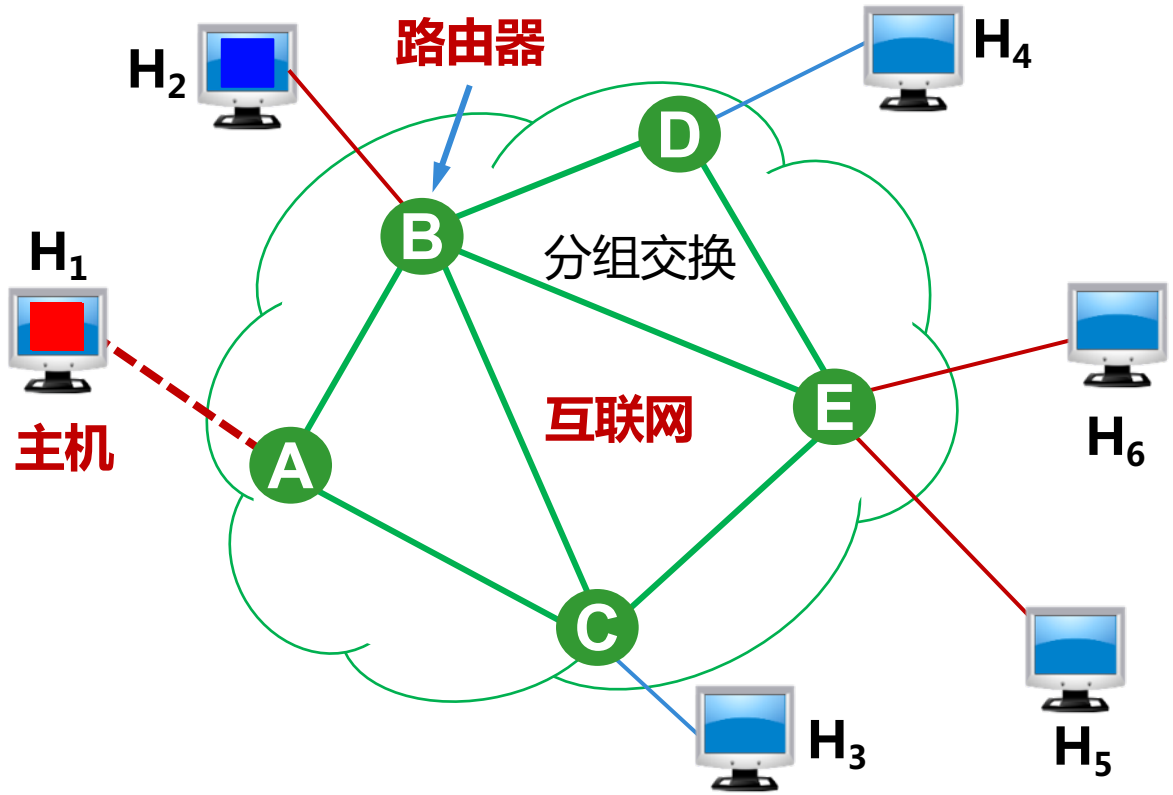


传输过程中出现差错，怎么办？

如何能检测发现错误？是否能够纠正错误？

数据链路层在协议栈中的位置





链路层
在哪里？

数据链路层（DLL）

逻辑链路控制 (LLC)

介质访问控制 (MAC)

点到点通信的
数据链路层

共享介质
数据链路层

数据链路层的功能

- 成帧（Framing）
  - 将比特流划分成“帧”的主要目的是为了检测和纠正物理层在比特传输中可能出现的错误，数据链路层功能需借助“帧”的各个域来实现
- 差错控制（Error Control）
  - 处理传输中出现的差错，如位错误、丢失等
- 基本数据链路层协议
  - 通信双方交互的协议规定（可靠性）
  - 流量控制（Flow Control）：确保发送方的发送速率，不大于接收方的处理速率，避免接收缓冲区溢出

本节内容

4.1 数据链路层的定义和功能

1. 成帧
2. 差错检测与纠正

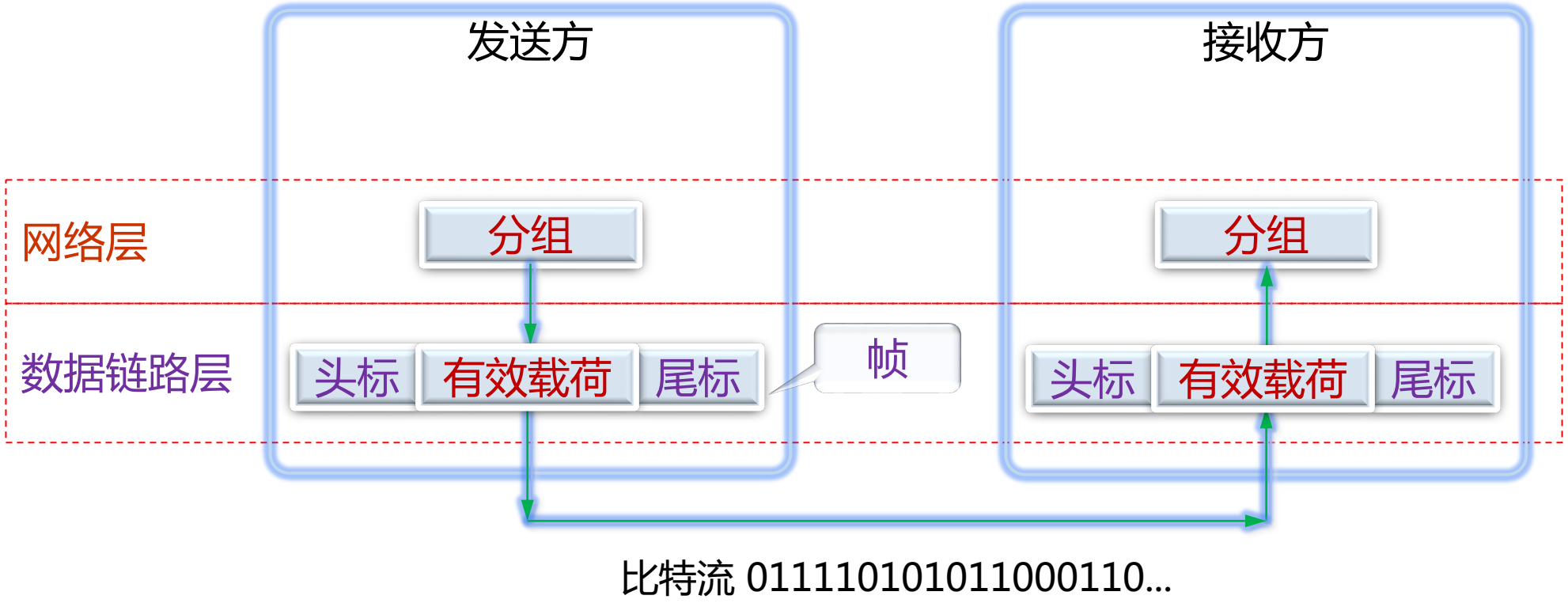
4.2 基本的数据链路层协议

4.3 滑动窗口协议

4.4 典型链路层协议

成帧（Framing）

- 分组 (packet) 与 帧(frame)的关系



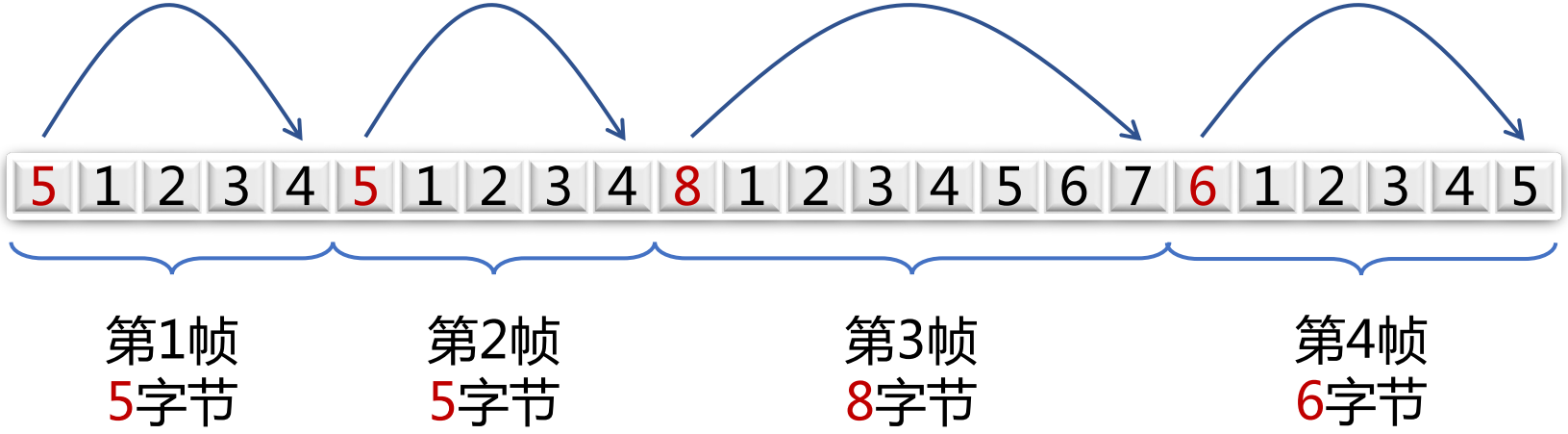
成帧（Framing）

- 关键问题：从源源不绝的比特流，到帧？
  - 接收方必须能从物理层接收的比特流中明确区分出一帧的开始和结束
  - 这个问题被称为帧同步或帧定界

- 成帧（framing）的方式
  - 字节计数法（ Byte count ）
  - 带字节填充的定界符法（ Flag bytes with byte stuffing ）
  - 带比特填充的定界符法（ Flag bits with bit stuffing ）
  - 物理层编码违例（ Physical layer coding violations ）

成帧（Framing）

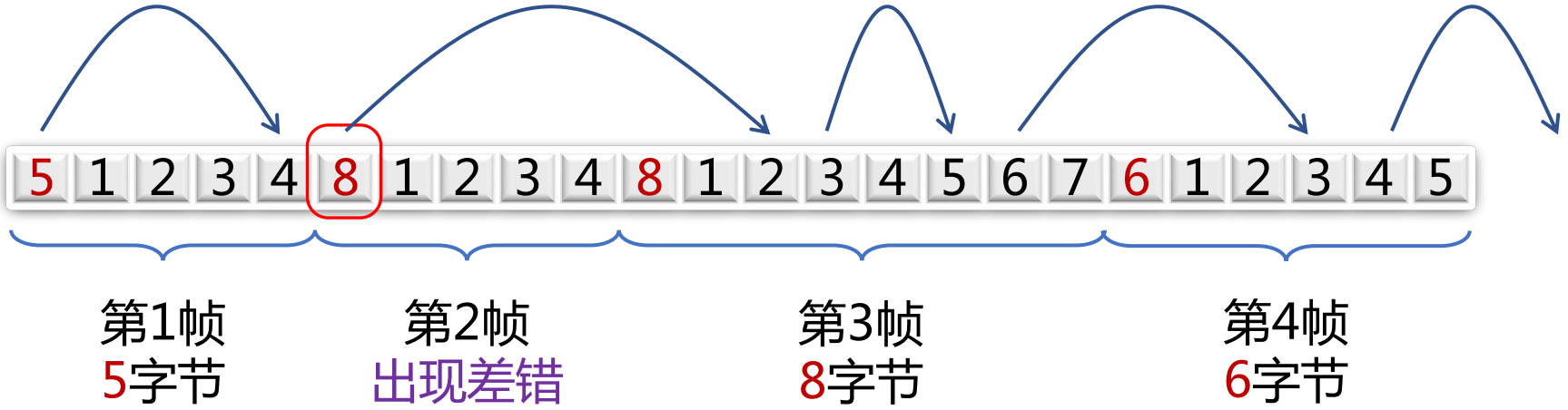
- 字节计数法：在帧首部使用计数字段来表明帧内字符数
- 无差错传输的情形



问题：如果某个计数字节出错会发生什么情况？

成帧（Framing）

- 字节计数法：出现了一个字节差错的情形



破坏了帧的边界，导致一连串帧的错误！怎么办？

透过现象看本质，连问三个为什么

成帧（Framing）

➢ 带字节填充的定界符法

- 定界符（FLAG）：一个特殊的字节，比如 01111110，即 0x7E，用于区分前后两个不同的帧

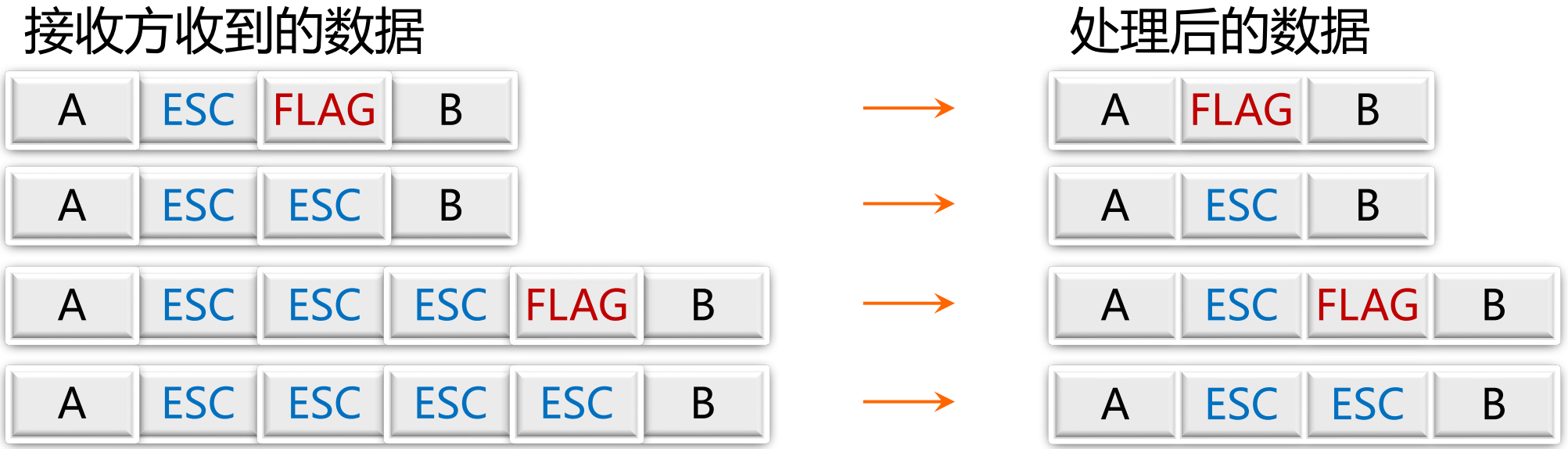
问题：如果有效载荷部分包含与“定界符”相同的字节会有什么问题？

成帧（Framing）

- 带字节填充的定界符法：发送方检查有效载荷，进行字节填充

成帧（Framing）

- 接收方的处理：逐个检查收到的每一个字节
  - 收到ESC：后一字节无条件成为有效载荷，不予检查
  - 收到FLAG：则为帧的边界

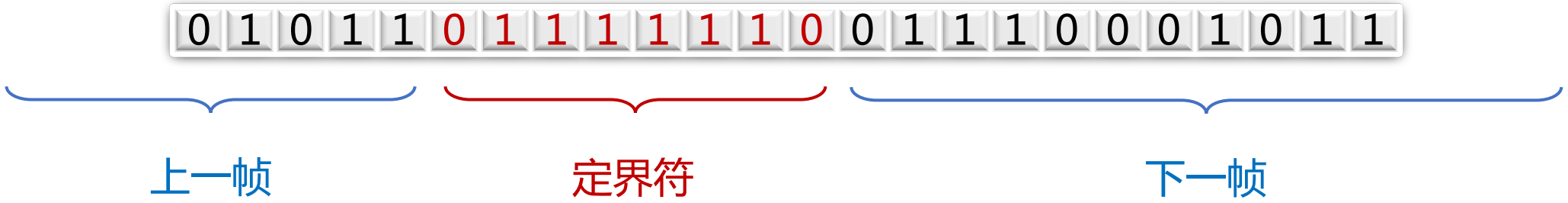


问题：带字节填充的定界符法不够灵活怎么办？

成帧（Framing）

➢ 带**比特**填充的定界符法

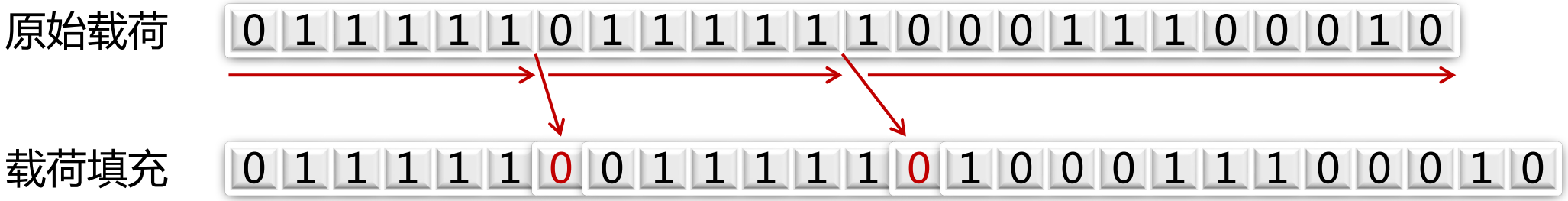
- 定界符：两个0比特之间，连续6个1比特，即01111110，0x7E



问题：如果有效载荷部分包含与“定界符”相同的位组合如何解决？

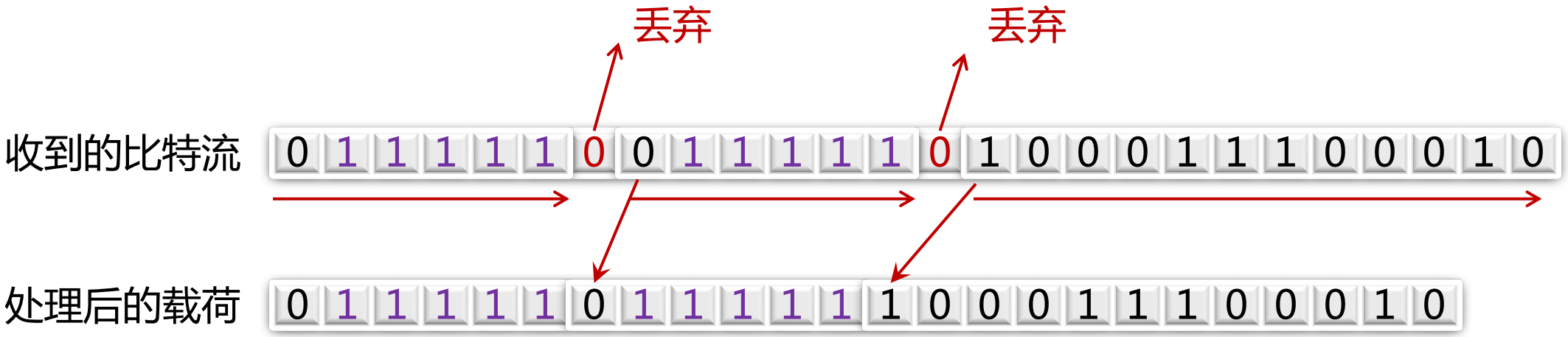
成帧（Framing）

- 带**比特**填充的定界符法：发送方检查有效载荷
  - 若在有效载荷中出现连续5个1比特，则直接插入1个0比特



成帧（Framing）

- 带**比特**填充的定界符法：接收方的处理
  - 若出现连续5个1比特
    - 若下一比特为0，则为有效载荷，直接丢弃0比特
    - 若下一比特为1，则连同后一比特的0，构成定界符，一帧结束



成帧（Framing）

➢ 物理层编码违例

- 核心思想：选择的定界符不会在数据部分出现
- 4B/5B编码方案
  - 4比特数据映射成5比特编码，剩余的一半码字（16个码字）未使用，可以用做帧定界符
  - 例如： 00110组合不包含在4B/5B编码中，可做帧定界符
- 曼切斯特编码 / 差分曼切斯特编码
  - 正常的信号在周期中间有跳变，持续的高电平（或低电平）为违例码，可以用作定界符
  - 例如：802.5令牌环网
- 前导码
  - 存在很长的 前导码（preamble），可以用作定界符（并同步时钟）
  - 例如：传统以太网、802.11

小结：数据链路层的设计问题

- 数据链路层的功能
  - 成帧，差错控制，流量控制
- 帧的定界方法
  - 字节计数法
  - 带字节填充的定界符法：借助转义字节 ESC 和定界符 FLAG
  - 带比特填充的定界符法：借助连续出现的 1 的个数
  - 物理层编码违例

差错控制

- 大胆思考，都会有什么样的差错？不仅仅是1比特错误
  - 差错（ incorrect ）：数据发生错误
  - 丢失（ lost ）：接收方未收到
  - 乱序（out of order）：先发后到，后发先到
  - 重复（repeatedly delivery）：一次发送，多次接收

交谈中会有什么问题？

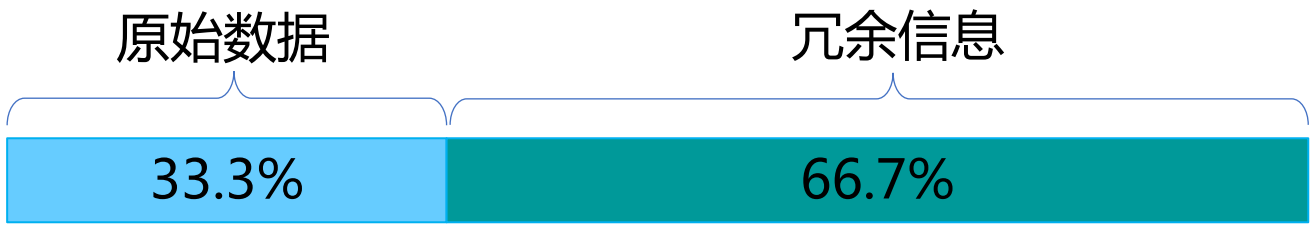
- 解决方案：差错检测与纠正、确认重传
  - 编码：检错与纠错编码
  - 确认：接收方校验数据（差错校验），并给发送方应答，防止差错
  - 定时器：发送方启动定时器，防止丢失
  - 顺序号：接收方检查序号，防止乱序递交、重复递交

差错检测和纠正概述

- 如何解决信道传输差错问题
  - 通常采用增加冗余信息（或称校验信息）的策略
  - 简单示例：每个比特传三份，如果每比特的三份中有一位出错，可以纠正

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

设计具体编码前来点理论基础？



携带2/3的冗余信息，冗余太多怎么办？

差错检测和纠正概述

- 码字 (code word)
  - 一个n=m+r位单元（含m个数据位和r个校验位），描述为 (n, m) 码
- 码率 (code rate)
  - 码字中不含冗余部分所占的比例，可以用m/n表示
- 海明距离 (Hamming distance)
  - 两个码字之间对应不同比特的数目
  - 如果两个码字的海明距离为d，则需要d个单比特错误就可以把一个码字转换成另一个码字
  - 海明距离为 **d** 的编码，可以检测 **d-1** 个单比特错误
  - 海明距离为 **d** 的编码，可以纠正 **(d-1)/2** 个单比特错误

码字A: 00000
码字B: 11111
A和B的海明距离为5

收到码字
00001
00011
00111
01111

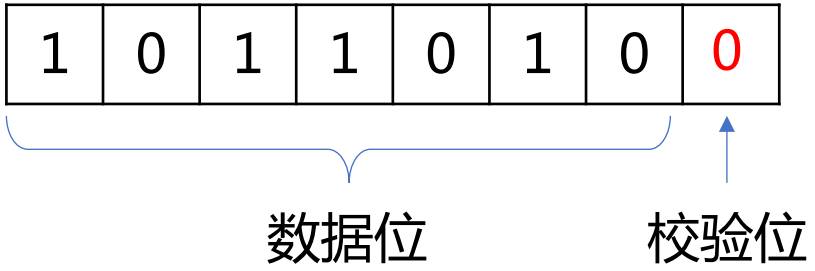
如果是3位错误呢？

典型检错码

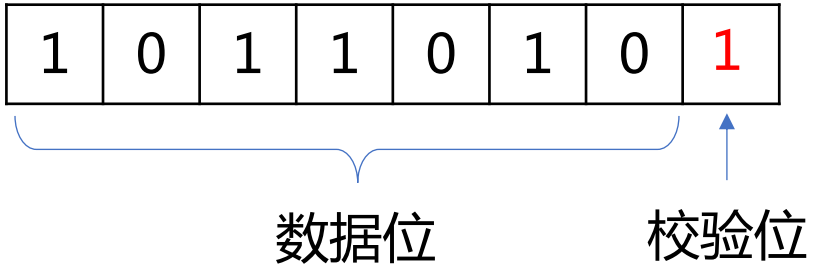
- 检错码（error-detecting code）
  - 在数据块中包含一些冗余信息，使接收方能够推断是否发生错误，但不能推断哪位发生错误
  - 接收方可以请求发送方重传数据
  - 主要用在高可靠、误码率较低的信道上，例如光纤链路
  - 偶尔发生的差错，可以通过重传解决差错问题
- 常用检错码
  - 奇偶检验 (Parity Check)：1位奇偶校验是最简单、最基础的检错码
  - 校验和 (Checksum)：主要用于TCP/IP体系中的网络层和传输层
  - 循环冗余校验 (Cyclic Redundancy Check，CRC)：数据链路层广泛使用的校验方法

典型检错码—奇偶校验

- 1位奇偶校验：增加1位校验位，可以检查奇数位错误
  - 偶校验：保证1的个数为偶数个



  - 奇校验：保证1的个数为奇数个



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

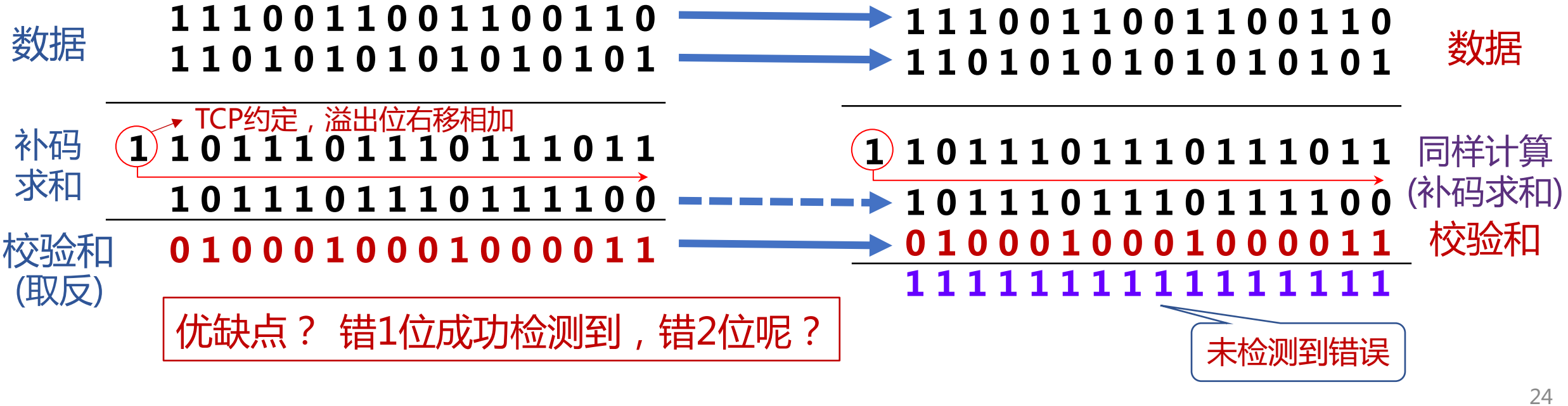
会有什么问题？
连续突发错误？

典型检错码—校验和

- 以TCP/IP体系中主要采用的校验方法为例

发送方：进行 16 位二进制补码求和运算，计算结果取反，随数据一同发送

接收方：进行 16 位二进制补码求和运算（包含校验和）。若结果非全1，则检测到错误



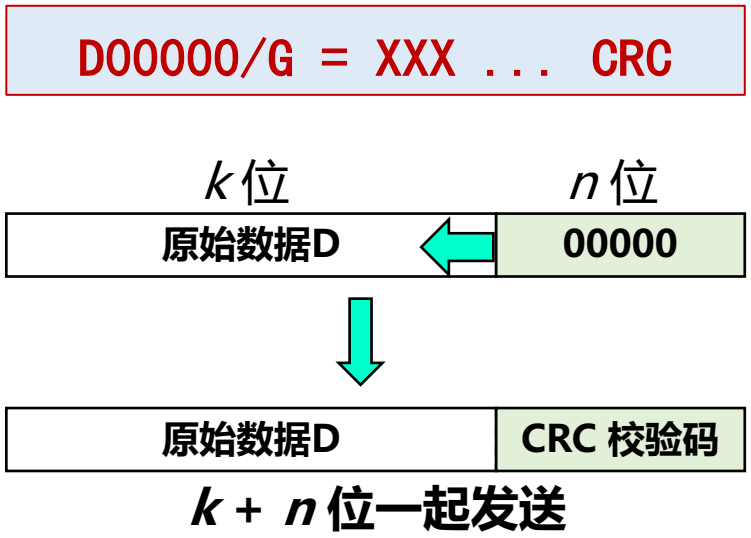
优缺点？ 错1位成功检测到，错2位呢？

典型检错码—循环冗余校验CRC

➢ CRC校验码计算方法

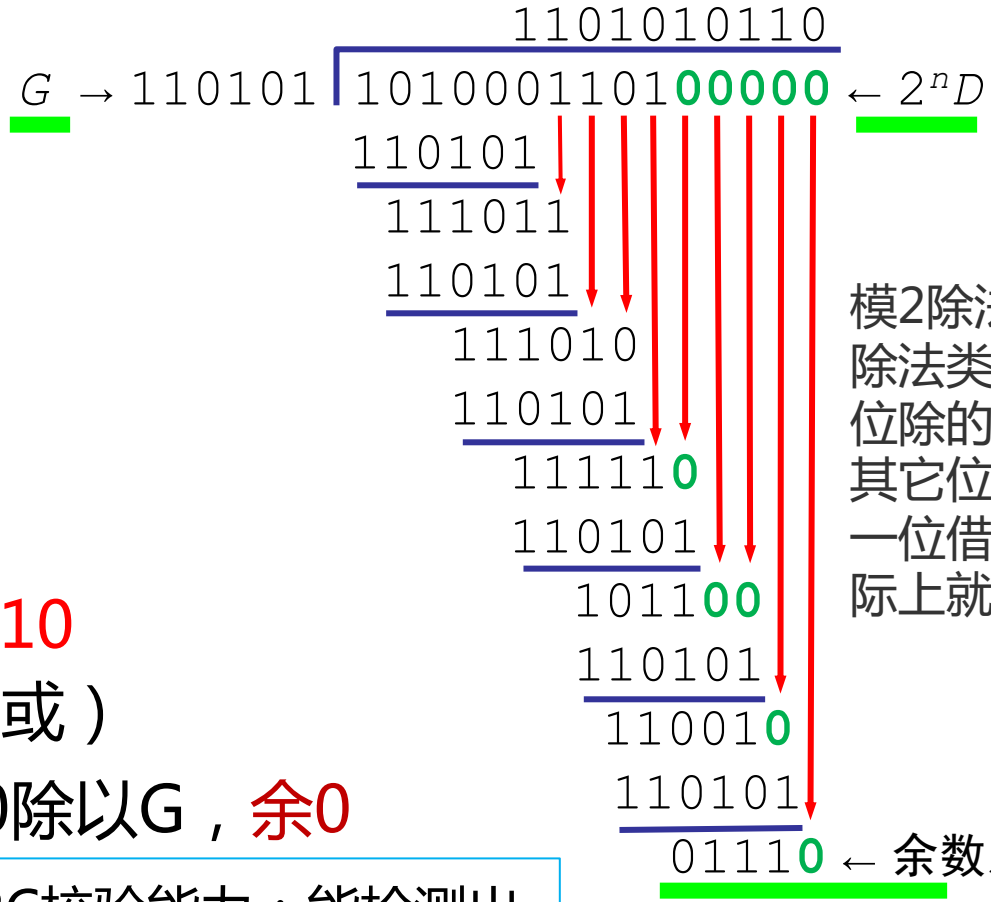
- 设原始数据D为k位二进制（如D = 1010001101）
- 生成多项式G：如果要产生n位CRC校验码，事先选定一个n+1位二进制G (收发双方提前商定)

  G最高位为1（如G=110101，即$G=x^5+x^4+x^2+1$）
- CRC校验码：将原始数据D乘以2^n（如1010001101000000），除以G（模2除），得到n位余数R即为CRC校验码（不足n位前面用0补齐）



检错码—循环冗余校验CRC

- CRC[1]校验码计算示例
  - 数据：D = 1010001101
  - n = 5
  - 生成多项式：G = 110101，即$G = x^5 + x^4 + x^2 + 1$
  - 余数：R = 01110
  - 发送数据：T = 10100011010111 0（模2运算：加减法不进位，即异或）
  - 接收方：T = 10100011010111 0除以G，余0
  - 有差错情况：(T+E)/G余0？



模2除法：与算术除法类似，但每一位除的结果不影响其它位，即不向上一位借位，所以实际上就是异或

CRC校验能力：能检测出少于n+1位的突发错误

[1] Peterson, W. W.; Brown, D. T. (January 1961). "Cyclic Codes for Error Detection". *Proceedings of the IRE*. **49** (1): 228–235.

检错码—循环冗余校验CRC

- 四个国际标准生成多项式
  - CRC-12 = $x^{12}+x^{11}+x^3+x^2+x+1$
  - CRC-16 = $x^{16}+x^{15}+x^2+1$
  - CRC-CCITT = $x^{16}+x^{12}+x^5+1$
  - CRC-32 = $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

例如，以太网、无线局域网使用CRC-32生成多项式

典型纠错码

- 纠错码（error-correcting code)
  - 主要用于错误发生比较频繁的信道上，如无线链路
  - 也经常用于物理层，以及更高层（例如，实时流媒体应用和内容分发）
  - 使用纠错码的技术通常称为前向纠错 FEC（Forward Error Correction)
  - 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（定位出错的位置）

1	0	1	1	0	1	0	0

考虑二维奇偶校验，是否可以进行纠错？

以二维偶校验为例：

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

无错情况

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

有错情况

典型纠错码—海明码

- 纠错码设计目标
  - 找到出错位置，从而提供纠错能力
  - 如何缩小并定位错误的范围？
  - 每个校验位对数据位的子集做校验？
  - 问题：如何设置和计算校验位？
- 海明码编码过程
  - 校验位：2的幂次方位（记为p1, p2, p4, p8）
  - 例：11比特的数据01011001101按顺序放入数据位
  - 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

校验位*：*1*、*2*、*4*、*8

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

	1_1	0_2	0_3
1_4	1_5	0_6	1_7
1_8	1_9	0_{10}	0_{11}
1_{12}	1_{13}	0_{14}	1_{15}

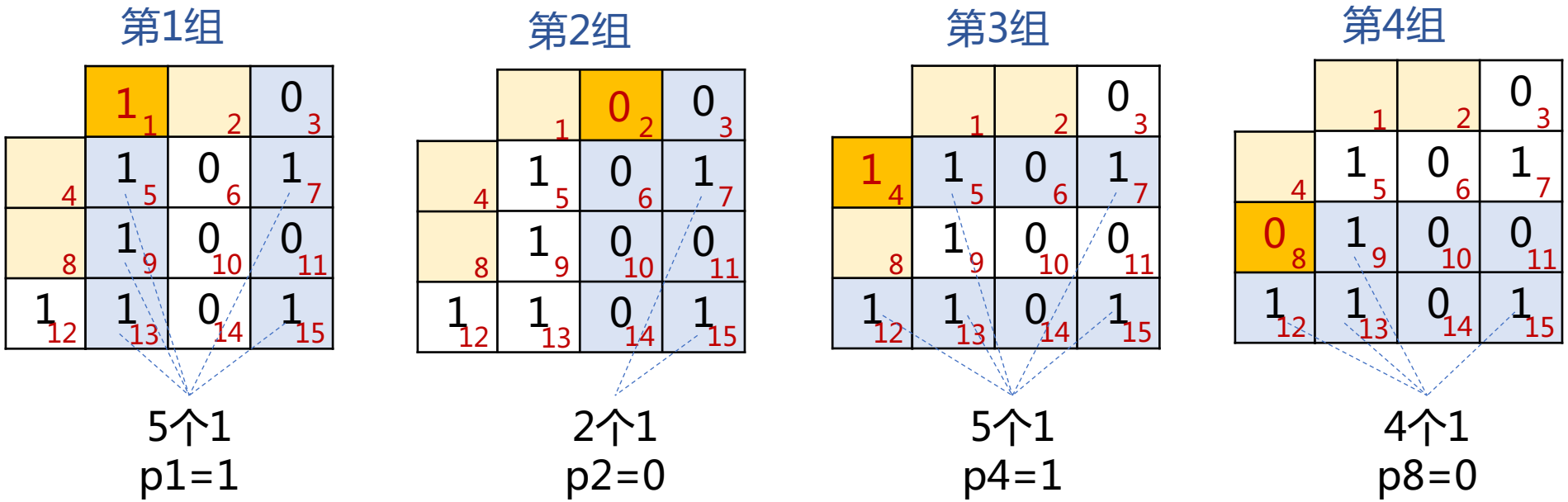
校验位　数据位

以 (15, 11)海明码为例
右下角数字为码字中位序号

典型纠错码—海明码

- 子集的选择与校验位计算
  - 海明码缺省为偶校验（也可以使用奇校验）

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

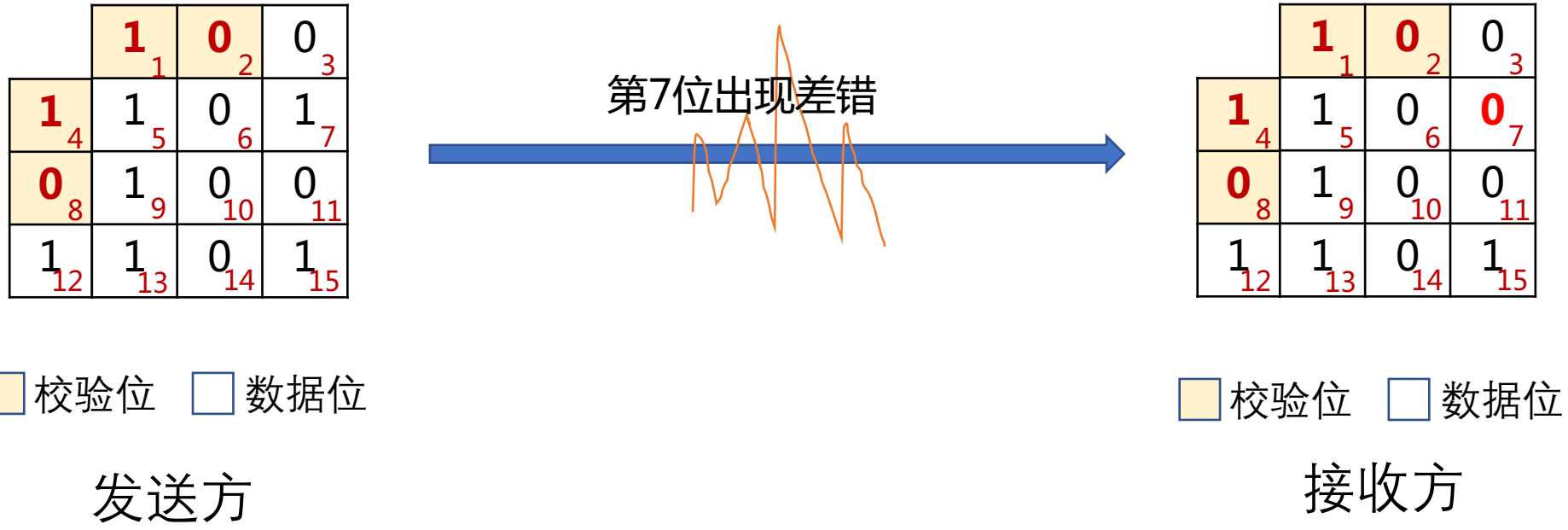


附：另一维度理解海明码

•		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
编码后数据位置		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15
奇偶校验位覆盖率	p1	X		X		X		X		X		X		X		X		X		X	
	p2		X	X			X	X			X	X			X	X			X	X	
	p4				X	X	X	X					X	X	X	X					X
	p8								X	X	X	X	X	X	X	X					
	p16																X	X	X	X	X

典型纠错码—海明码

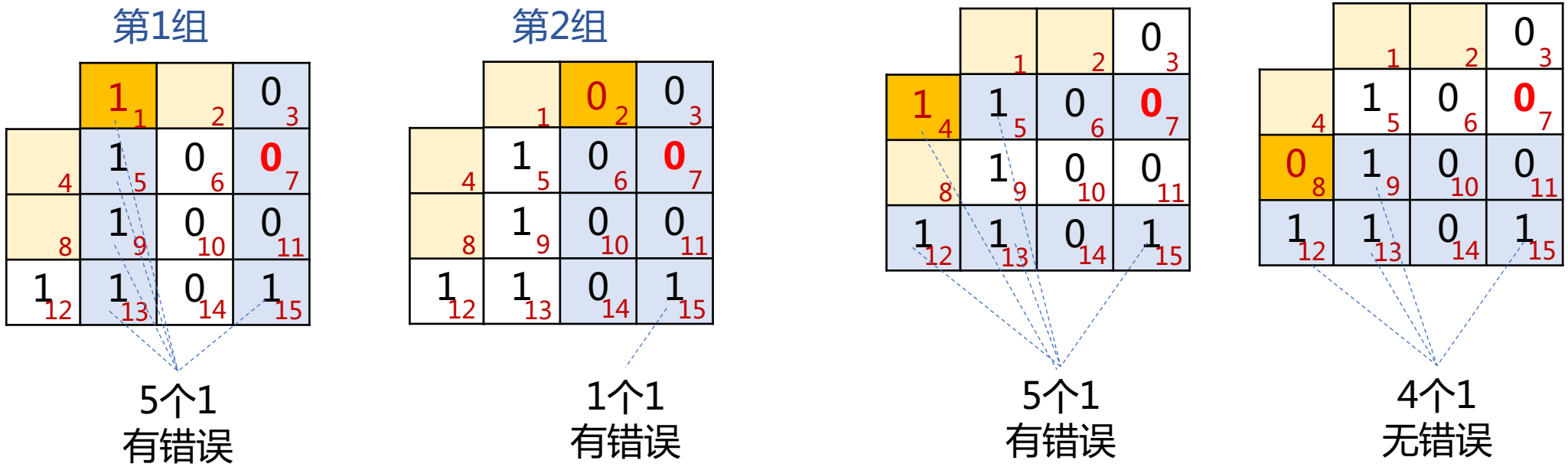
- 码字的发送与接收
  - 如果发送过程中第7位出现差错，如何定位错误？



典型纠错码—海明码

- 定位错误与纠正
  - 组1和组2的校验结果，定位错误位于第4列
  - 组3校验结果指明存在错误，组4校验结果指明无错误，可定位错误位于第2行
  - 结论：定位第2行、第4列（第7比特错误）

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



典型纠错码—海明码

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

- 定位错误与纠正
  - 定位：1+2+4 = 7
- 海明码纠正的实现过程
  - 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（每组运算）
  - 若 p_k 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 j，则第 j 位出错

练习：海明码

- 问题
  1. ***ASCII：10011010 --》？*** 构造纠一位错误的海明码
  2. 假定实际接收到的数据是011100101110. 则接收方可以计算出哪一位出错并对其进行更正。 ***How？***
  3. *补充: 假定收到的数据是001100101010，接收方该如何反应？*

练习答案

1. 首先，确定校验位个数，设计一位纠错码
   - 要求：m个信息位，r个校验位，纠正单比特错；
   - 对2^m个有效信息中任何一个，有n 个与其距离为1的无效码字，因此有：$(n + 1)\ 2^m \le 2^n$
   - 利用 n = m + r，得到 $(m + r + 1) \le 2^r$。给定m，利用该式可以得出校正单比特误码的校验位数目的下界

   其次，计算每位校验位，此码字为: 011100101010

2. 记下所有出错的校验位，可以发现校验位2和8的数据不正确. 错误校验位 2 + 8 = 10, 则位置10的数据出错。

位置1检查1,3,5,7,9,11:
? _ 1 _ 0 0 1 _ 1 0 1 0. 偶数个1，因此位置1设为0，

位置2检查2,3,6,7,10,11:
0 ? 1 _ 0 0 1 _ 1 0 1 0. 奇数个1，因此位置2设为1，

位置4检查4,5,6,7,12:
0 1 1 ? 0 0 1 1 0 1 0. 奇数个1，因此位置4设为1，

位置8检查8,9,10,11,12:
0 1 1 1 0 0 1 ? 1 0 1 0. 偶数个1，因此位置8设为0，

5G新标准——编码的力量

- 华为主导的5G标准：极化码—Polar码
  - Polar码最早由土耳其教授E. Arikan在2007年提出
  - 华为慧眼识珠，积极发展并应用Polar码
  - 5G标准的师徒之争：华为推崇极化码；
    美国高通推崇低密度奇偶校验（LDPC）码
  - LDPC码：数据信道编码方案；极化码：控制信道编码方案

“国王般的礼遇”

- 极化码的基本思路
  - 信道极化理论是Polar编码理论的核心，包括信道组合和信道分解部分
  - 信道极化过程本质上是一种信道等效变换的过程，当信道的数目趋于无穷大时，出现极化现象：一部分信道将趋于无噪信道，另外一部分则趋于全噪信道
  - 无噪信道的传输速率将会达到信道容量I (W)，而全噪信道的传输速率趋于零
  - Polar码的编码策略正是应用了这种现象的特性，利用无噪信道传输用户有用的信息，全噪信道传输约定的信息或者不传信息

小结：差错检测和纠正

- 检错和纠错
  - 通过增加冗余信息，实现检错码和纠错码
  - 海明距离：两个码字之间对应比特的不同数目
- 检错码
  - 奇偶检验：增加1位校验位，可以检查奇数位错误
  - 校验和：如TCP/IP 校验和，使用 16 位二进制补码求和，而后取反
  - 循环冗余校验CRC
- 纠错码
  - 海明码：以奇偶校验为基础，提供1位纠错能力

本节内容

4.1 数据链路层的定义和功能

4.2 基本的数据链路层协议

4.3 滑动窗口协议

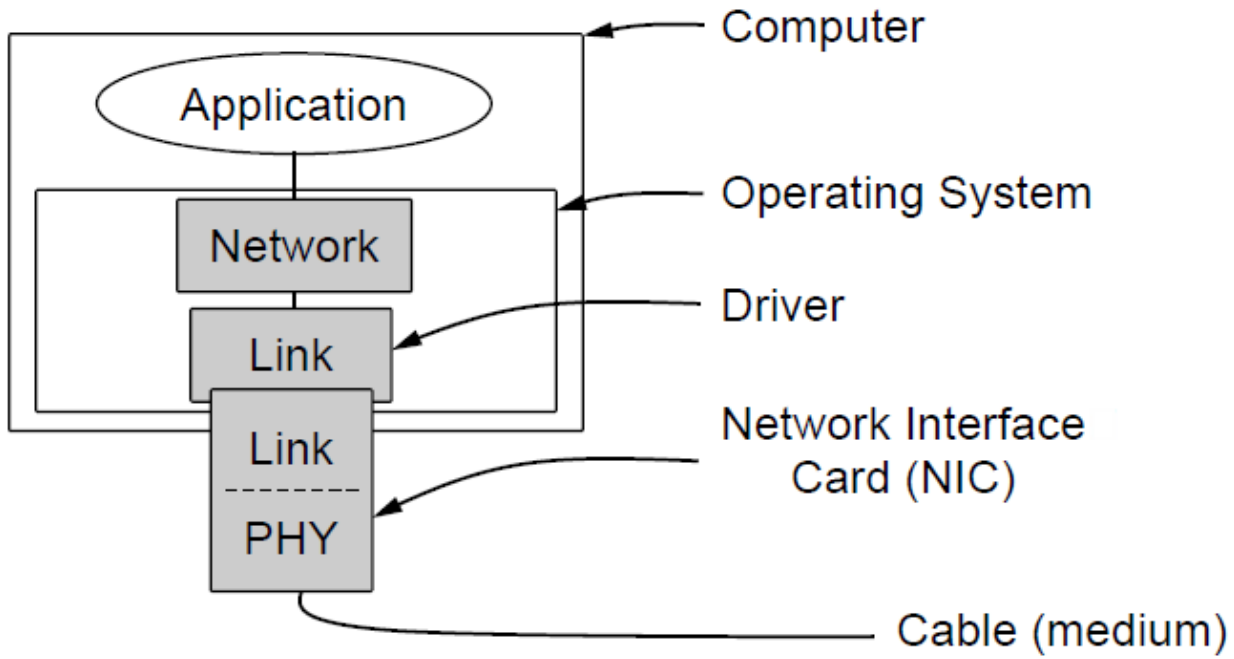
4.4 典型链路层协议

步步深入，越来越现实

1. 乌托邦式单工协议
2. 无错信道单工停止-等待协议
3. 有错信道单工停止-等待协议

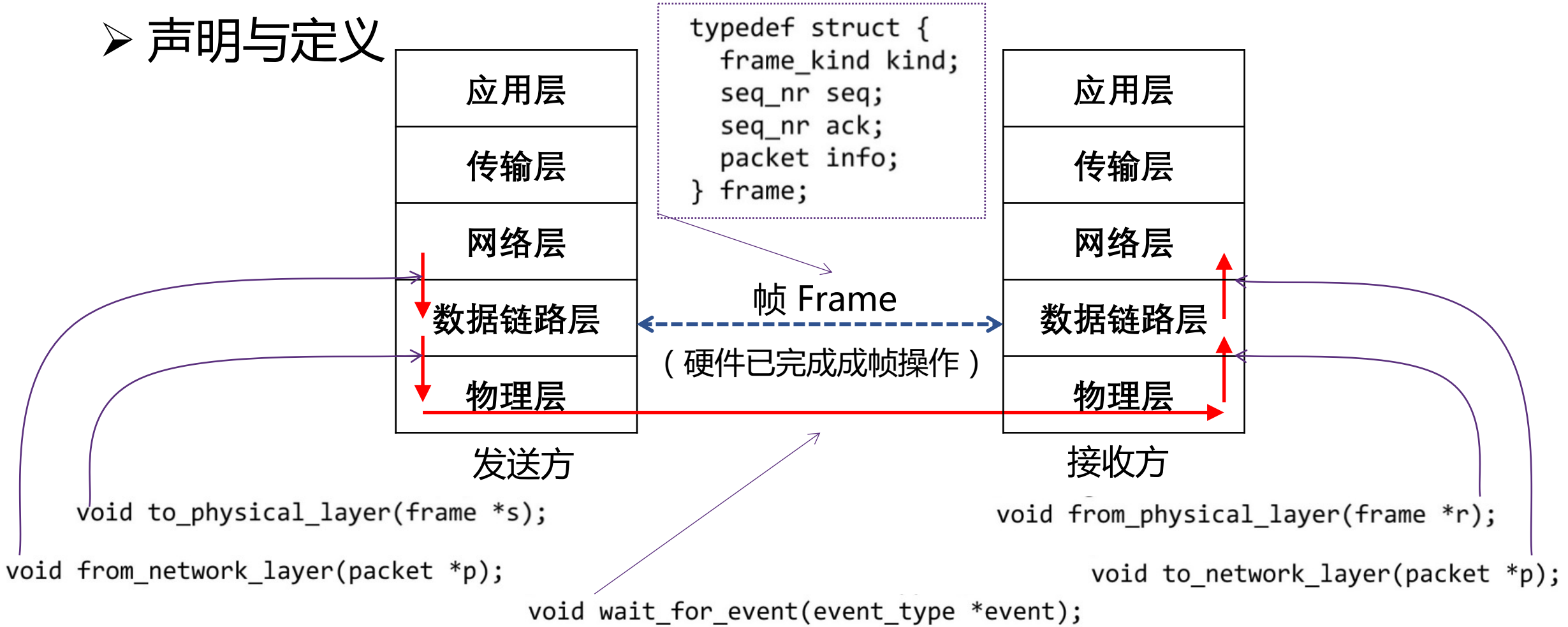
物理层、数据链路层和网络层的实现

- NIC
  - NIC，Network Interface Card
  - 物理层进程和某些数据链路层进程，运行在专用硬件上
- 协议栈与驱动
  - 数据链路层进程的其他部分和网络层进程，作为操作系统的一部分（协议栈），运行在CPU上
  - 数据链路层进程的软件通常以设备驱动的形式存在



基本的协议定义

- 声明与定义



event_type: 帧到达frame_arrival、帧出错cksum_err、超时timeout等

乌托邦式单工协议P1

- 从理想而简单的假设开始设计
  - 单工（Simplex）协议：数据单向传输
  - 完美信道：帧不会丢失或受损
  - 始终就绪：发送方/接收方的网络层始终处于就绪状态
  - 瞬间完成：发送方/接收方能够生成/处理无穷多的数据
- 乌托邦：完美但不现实的协议
  - 不处理任何流量控制或纠错工作
  - 接近于无确认的无连接服务，必须依赖更高层次解决上述问题

乌托邦式单工协议P1

➢发送方

- 在循环中不停发送
- 从网络层获得数据
- 封装成帧
- 交给物理层
- 完成一次发送

缺点？
比比看谁快？！

```
frame s;
packet buffer;

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

发送程序：取数据，构成帧，发送帧

➢ 接收方

- 在循环中持续接收
- 等待帧到达（frame_arrival）
- 从物理层获得帧
- 解封装，将帧中的数据传递给网络层
- 完成一次接收

```
frame r;
event_type event;

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

帧到达frame_arrival

接收程序：等待，接收帧，送数据给高层

流量控制

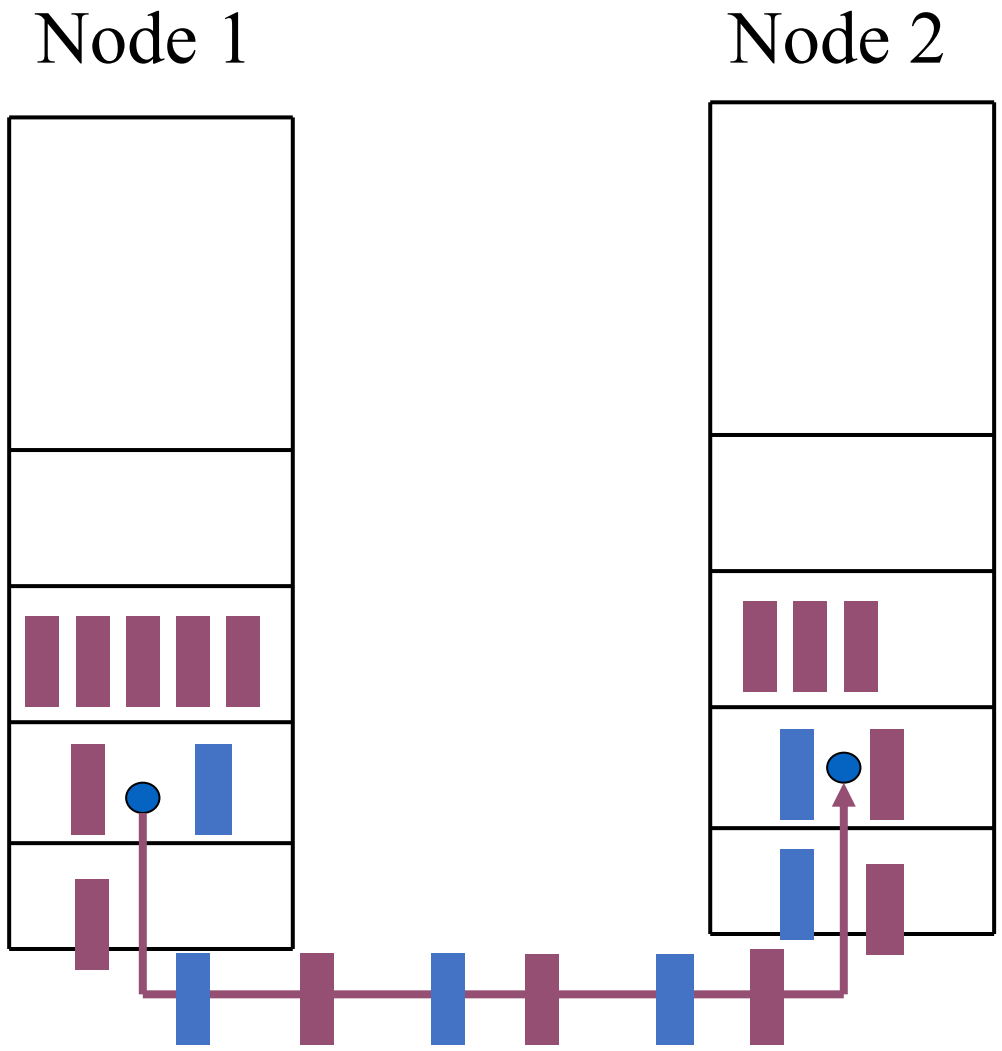
- 链路层存在的问题：接收方的处理速率
  - 发送方发送帧的速度过快（大于接收方处理速度），将导致接收方被“淹没”(overwhelming)
  - 基于速率 (rate-based) 的流量控制？
    - 发送方根据内建机制，自行限速
  - 基于反馈 (feedback-based) 的流量控制√
    - 接收方反馈，发送方调整发送速率

如何能让接收方不被淹没？

- 仍然假设
  - 通信信道不会出错（Error-Free）
  - 数据传输保持单向, 但是需要双向传输链路（半双工物理信道）

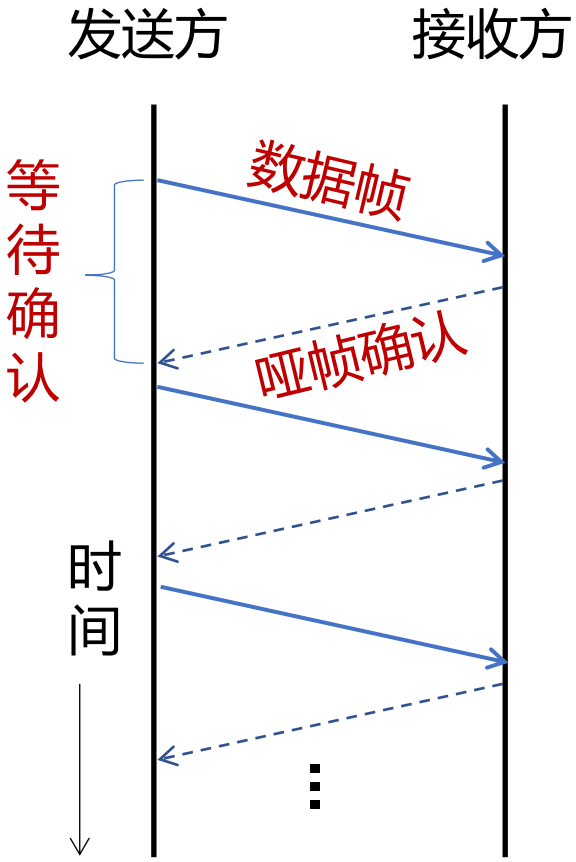
无错信道上的停等式协议P2

- 停-等式协议（stop-and-wait）
  - 增加确认机制：**Ack**nowledgment
  - 发送方发送一帧后暂停，等待确认到达后发送下一帧
  - 接收方完成接收后，回复确认接收
- 哑帧确认（dummy frame）
  - 确认帧的内容是不重要的



无错信道上的停等式协议P2

- 停-等式协议（stop-and-wait）
  - 增加确认机制：**Ack**nowledgment
  - 发送方发送一帧后暂停，等待确认到达后发送下一帧
  - 接收方完成接收后，回复确认接收
- 哑帧确认（dummy frame）
  - 确认帧的内容是不重要的



时序图

无错信道上的停等式协议P2

- 发送方
  - 完成一帧发送后
  - 等待确认到达
  - 确认到达后，发送下一帧

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

发送程序：取数据，组帧，发送帧，等待确认帧

- 接收方
  - 完成一帧接收后
  - 交给物理层一个哑帧
  - 作为成功接收上一帧的确认

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

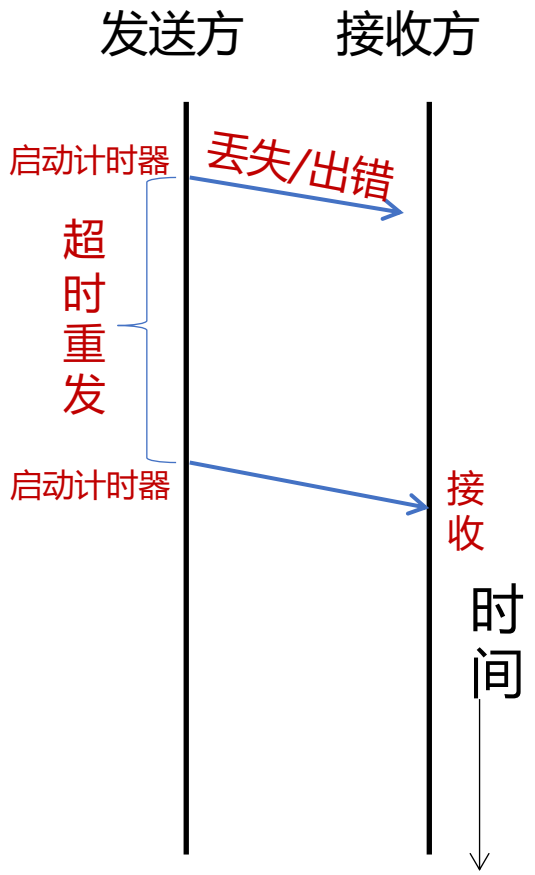
接收程序：
等待，接收帧，送数据给高层，回送确认帧

发现缺点？通信信道会出错吗？

有错信道上的单工停等式协议P3

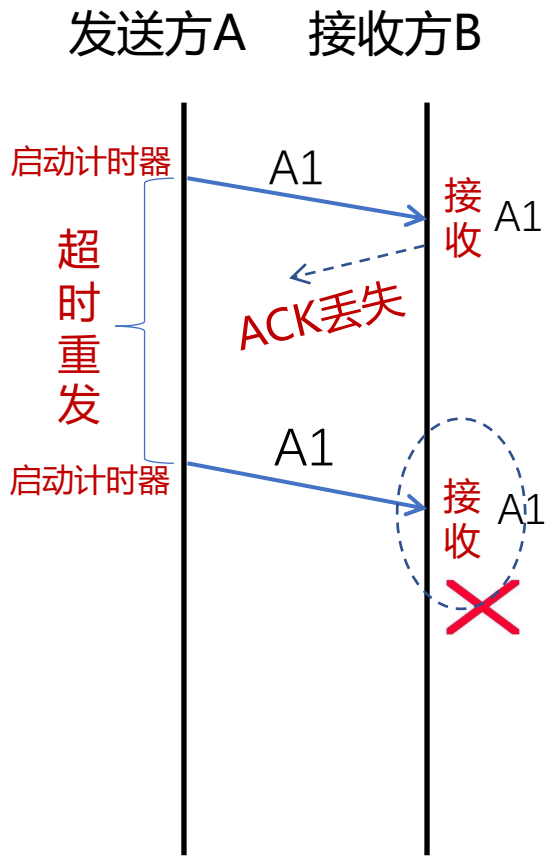
- 可能有什么出错方式？
  - 帧在传输过程中可能会被损坏，接收方能够检测出来
  - 帧在传输过程中可能会丢失 ，永远不可能到达接收方
- 发明解决方案？
  - 发送方增加一个计时器(timer)，如果经过一段时间没有收到确认，发送方将超时，于是再次发送该帧
- 除了数据会丢失，还有其他什么问题？

反向信道出错会什么结果？



有错信道上的单工停等式协议P3

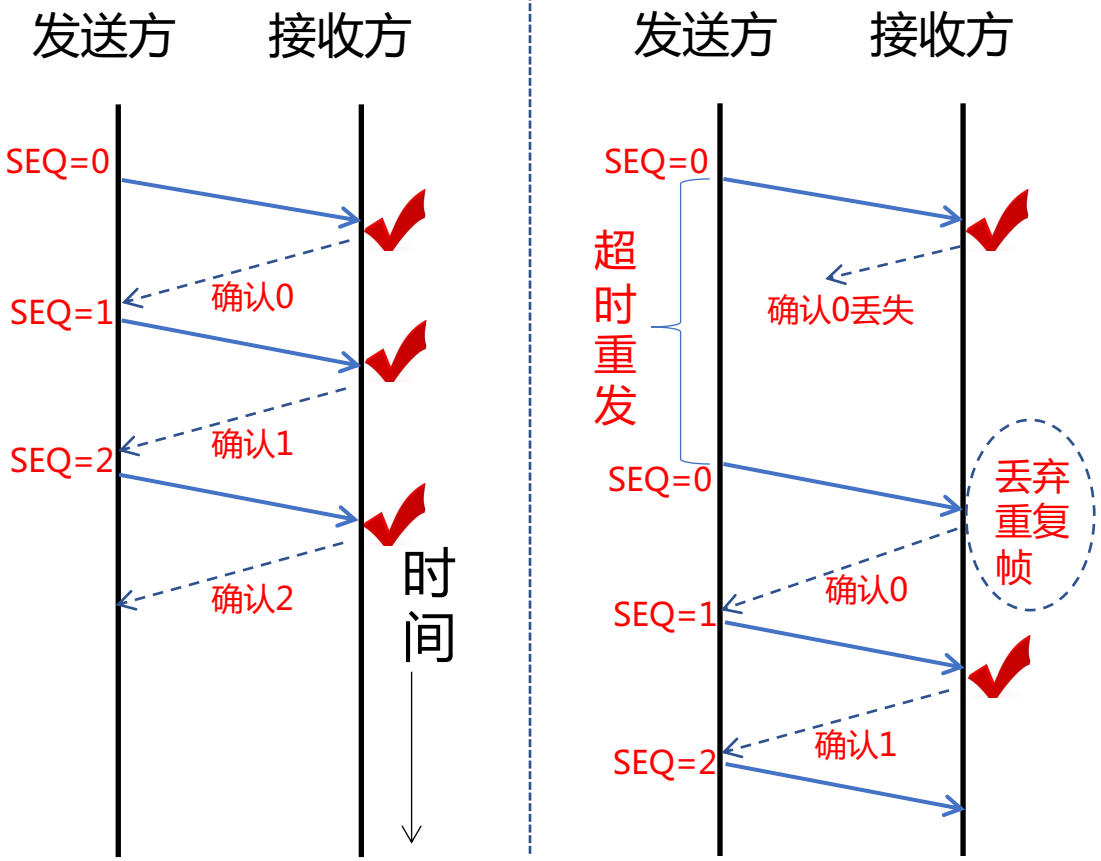
- 考虑一个特别场景
  - A发送帧A1
  - B收到了A1
  - B生成确认ACK
  - ACK在传输中丢失
  - A超时，重发A1
  - B收到A1的另一个副本（并把它交给网络层）
- 其他的场景
  - 另一个导致副本产生的场景是过长的延时 (long delay)



接收端如何识别重复的报文？

有错信道上的单工停等式协议P3

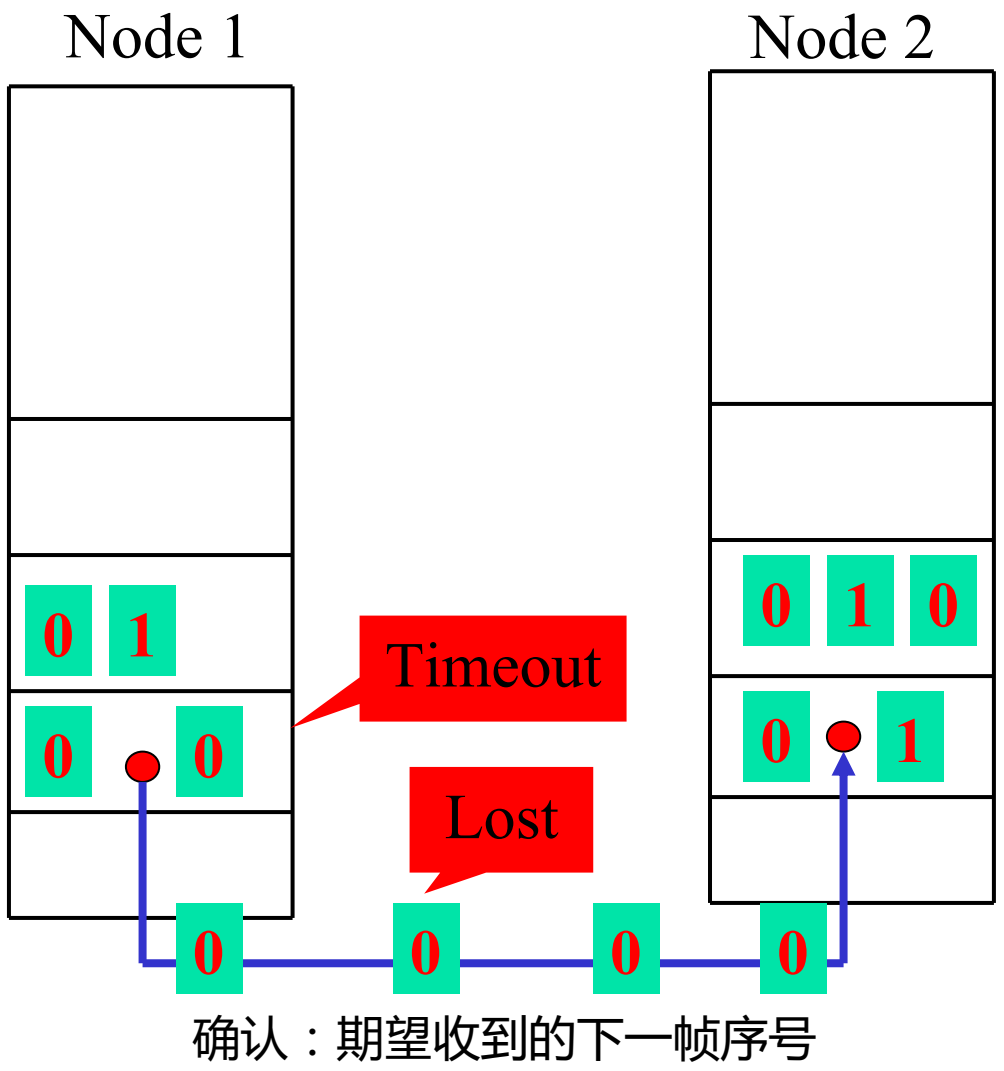
- 序号（SEQ：sequence number）
  - 接收方需区分到达的帧是否为第一次发来的新帧
  - 让发送方在发送的帧的头部放一个序号，接收方可以检查它所收到的帧序号
  - 由此判断这是新帧还是应该被丢弃的重复帧
- 为精简帧头，序号最小多少位？
  - 1 bit序号(0或1)可满足要求
- 带有重传的肯定确认
  - 发送方在发送下一个数据前，需要得到确认



SEQ=2改为SEQ=0

有错信道上的单工停等式协议P3

- 序号（SEQ：sequence number）
  - 接收方需区分到达的帧是否为第一次发来的新帧
  - 让发送方在发送的帧的头部放一个序号，接收方可以检查它所收到的帧序号
  - 由此判断这是新帧还是应该被丢弃的重复帧
- 为精简帧头，序号最小多少位？
  - 1 bit序号(0或1)可满足要求
- 带有重传的肯定确认
  - 发送方在发送下一个数据前，需要得到确认



有错信道上的单工停等式协议P3

- 发送方
  - 初始化帧序号0，发送帧
  - 等待：正确的确认/错误的确认/超时
  - 正确确认：发送下一帧
  - 超时/错误确认：重发

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

正确确认，读取下一帧

- 接收方
  - 初始化期待0号帧
  - 等待帧达到
  - 正确帧：交给网络层，并发送该帧确认
  - 错误帧：发送上一个成功接收帧的确认

```
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);
        }
}
```

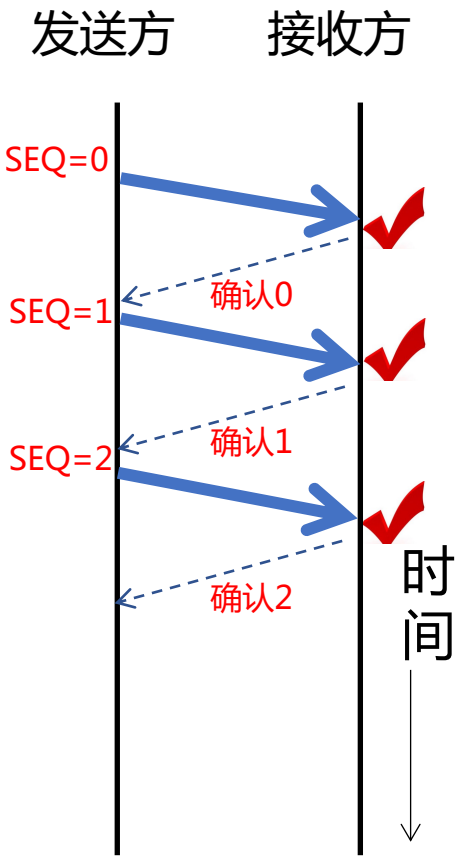
交给网络层

发送确认帧

有错信道上的单工停等式协议P3

- 信道利用率
  - F = 帧大小 (bits)
  - R= 链路带宽 (Bandwidth in bits/second)
  - 每帧发送时间：F/R，即停等协议的发送工作时间
  - 发送空闲时间：RTT
  - 信道利用率 (line utilization)=F/(F+R · RTT)
  - 当 F<R · RTT 时，信道利用率 < 50%

R·RTT=2倍时延带宽积



有错信道上的单工停等式协议P3

- 停止等待协议的效率问题
  - 举例:
    - Short frame length: 1000 bit frames
    - High bandwidth: 1 Mbps channel (卫星信道)
    - 传播延迟propagation delay: 270 ms
  - 每一帧的发送时间是 1毫秒（ 1000 bits/(1,000,000 bits/sec ））
  - 由于传播延迟较长，发送者在541毫秒之后才能收到确认
  - 信道利用率1/541
  - 停止等待协议的问题是只能有一个没有被确认的帧在发送中

有错信道上的单工停等式协议P3

- 信道利用率=F/(F+R·RTT)，帧大小F，R·RTT为时延带宽积的2倍
- 信道利用率很低

> 长肥网络 (LFN，Long Fat Network)：如果一个网络的带宽-时延积 (bandwidth-delay product) 很明显的大于 10^5 bits (~12 kB)，则可以被认为是长肥网络

- 提高效率的方法？
  - 尝试使用更大的帧？
  - 但帧最大长度受信道比特错误率的限制
  - 帧越大，传输出错概率越高，导致更多重传

停等协议，效率太低！
如何优化？下回分解☺

小结

- 数据链路层基本协议
  - 乌托邦式单工协议P1：完美信道，双方始终就绪，收发瞬间完成
  - 无错信道上的停等式协议P2：发送后停下等待ACK，接收后回复ACK
  - 有错信道上的单工停等式协议P3：引入计时器和帧序号，处理帧和ACK丢失
- 信道利用率计算
  - F/(F + R*RTT)，F是帧大小，RTT是发送空闲时间，R是信道容量

总结：链路层点到点传输三大功能

- 成帧的方式
  - 字节计数法，带字节填充的定界符法，带比特填充的定界符法
- 差错检测和纠正
  - 海明距离
  - 检错码：奇偶校验，校验和，循环冗余校验
  - 纠错码：海明码
- 协议：可靠性、淹没对端
  - 乌托邦式单工协议P1
  - 无错信道上的停等式协议P2
  - 有错信道上的单工停等式协议P3
  - 信道利用率计算

步步深入，越来越现实

提升传输效率

本节内容

4.1 数据链路层的定义和功能

4.2 基本的数据链路层协议

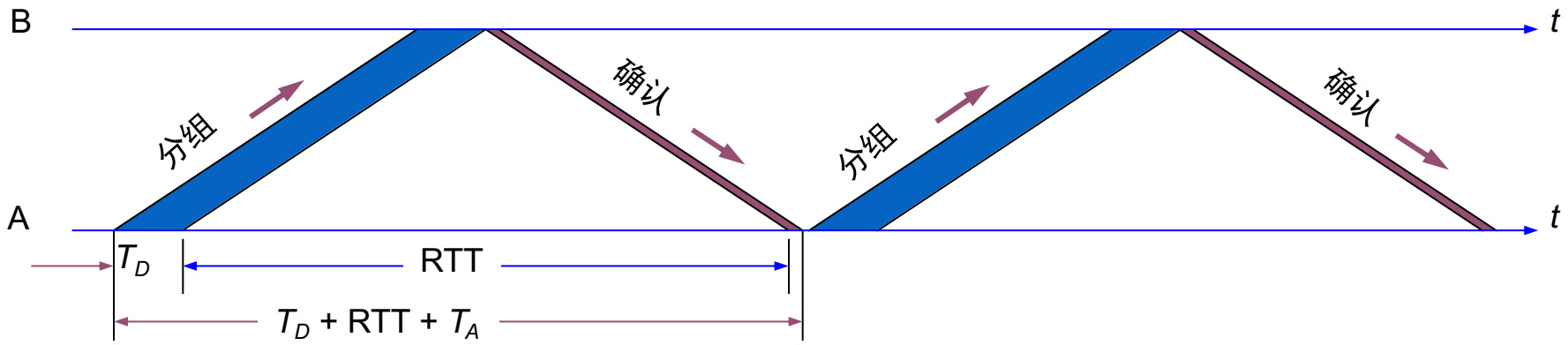
4.3 滑动窗口协议

4.4 典型链路层协议

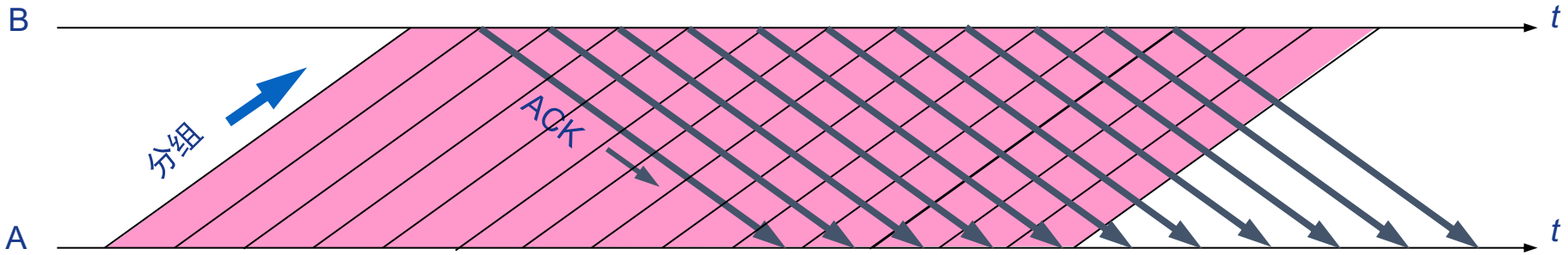
1. 滑动窗口协议P4
2. 回退N协议P5
3. 选择重传协议P6

思考：如何提升传输效率？

停止等待协议的优点是简单，但缺点是信道利用率太低



解决方案：Pipelining,流水线技术: 连续发送多分组后再等待确认



滑动窗口协议—协议基本思想

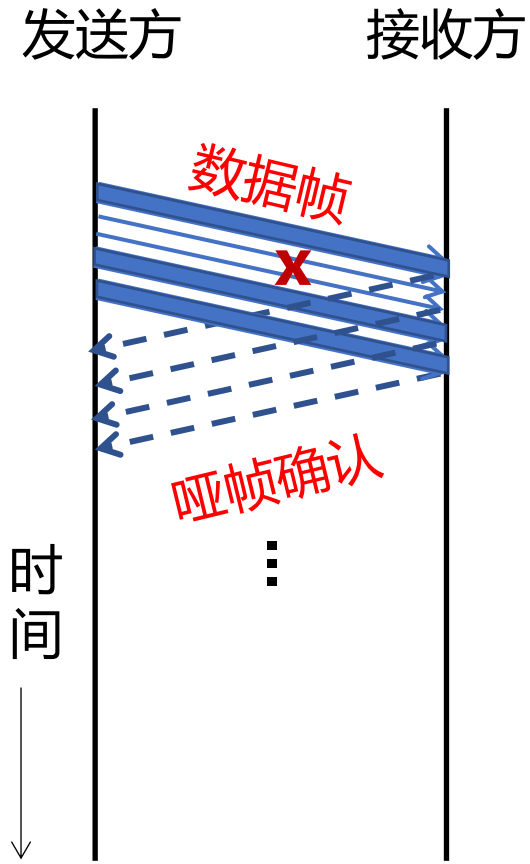
- 流水线技术的需求

  - 帧丢失？ACK丢失？
  - 重传哪个帧？
  - 流控：未确认的帧数？

  - 发送方：要暂存哪些帧以便可能的重传
  - 接收方：如何能向网络层按序提交数据
  - 双方：允许发送方发多少帧以不淹没接收方

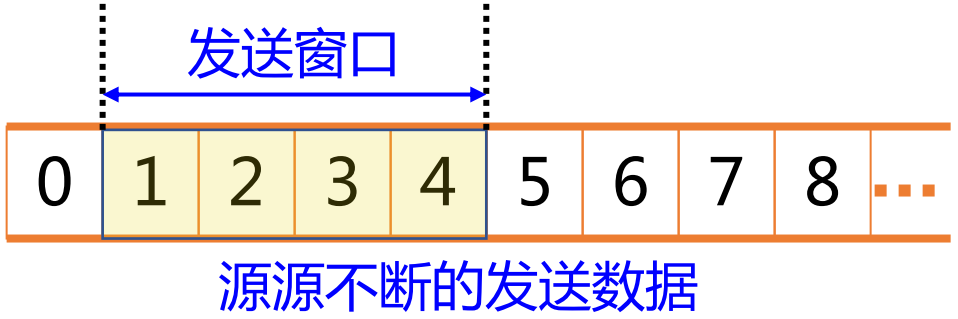
- 流水线技术的实现方式

  - 滑动窗口协议（本节的三个协议P4/P5/P6）
  - 都能在实际（非理想）环境下正常工作
  - 区别仅在于效率、复杂性和对缓冲区的要求



滑动窗口协议—协议基本思想

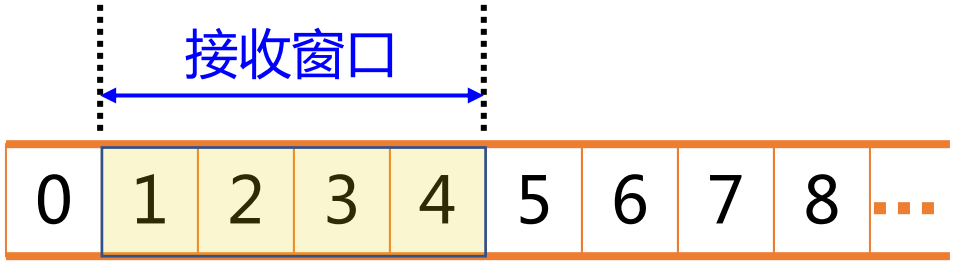
- 发送者面临的问题
  - 发了哪些帧，需要重传时怎么办？
- 发送窗口
  - 发送端始终保持一个已发送但尚未确认的帧的序号表，称为发送窗口
  - 发送窗口的上界表示要发送的下一个帧的序号，下界表示未得到确认的帧的最小序号
  - 发送窗口大小 = 上界 - 下界，大小可变
  - 发送端每发送一个新帧，帧序号取上界值，上界加1
  - 每接收到一个确认序号 = 发送窗口下界的正确响应帧，下界加1



滑动窗口

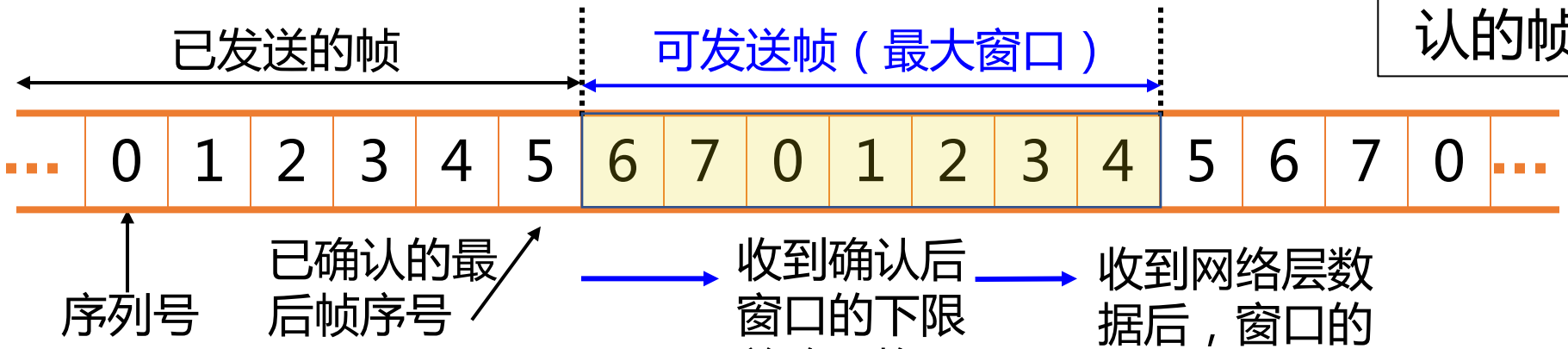
滑动窗口协议—协议基本思想

- 接收者面临的问题
  - 保序：接收方需按照发送方网络层发送的顺序，将数据提交给上层
  - 避免被发送方淹没：需要告诉发送方，能够发送多少数据
- 接收窗口
  - 接收端有一个接收窗口，但不一定与发送窗口相同
  - 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
  - 接收窗口的上界表示允许接收的最大序号，下界表示希望接收的最小序号
  - 序号等于下界的帧被正确接收，并产生一个确认帧，上界、下界都加1
  - 接收窗口大小取决于接收方能力（通常保持不变）



滑动窗口协议—协议基本思想

- 发送窗口 (例如大小7)



- 接收窗口(如大小7)

单选题 1分

设置

在滑动窗口协议中，若收到接收窗口以外的帧，接收方会如何处理？

A 将这些帧暂存

B 将这些帧丢弃

C 不确定

提交

回顾：有错信道单工停等式协议P3

- 发送方
  - 初始化帧序号0，发送帧
  - 等待：正确的确认/错误的确认/超时
  - 正确确认：发送下一帧
  - 超时/错误确认：重发

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

正确确认，读取下一帧

- 接收方
  - 初始化期待0号帧
  - 等待帧达到
  - 正确帧：交给网络层，并发送该帧确认
  - 错误帧：发送上一个成功接收帧的确认

```
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);
        }
}
```

交给网络层

发送确认帧

哑帧确认
（数据单向发送）

一比特滑动窗口协议P4

- 设计目标
  - 通信双方互发数据
  - 信道条件：全双工
  - 最最简单的窗口机制（即窗口大小为1比特）

呼唤
循序渐进的协议设计
（先别搞流水线）

- 哑帧确认->捎带确认（piggybacking）
  - 将确认帧与反向数据帧合并，可以暂时延迟待发确认，以便附加到下一个待发数据帧
  - 优点：充分利用信道带宽，减少帧数目，减少“帧到达”中断

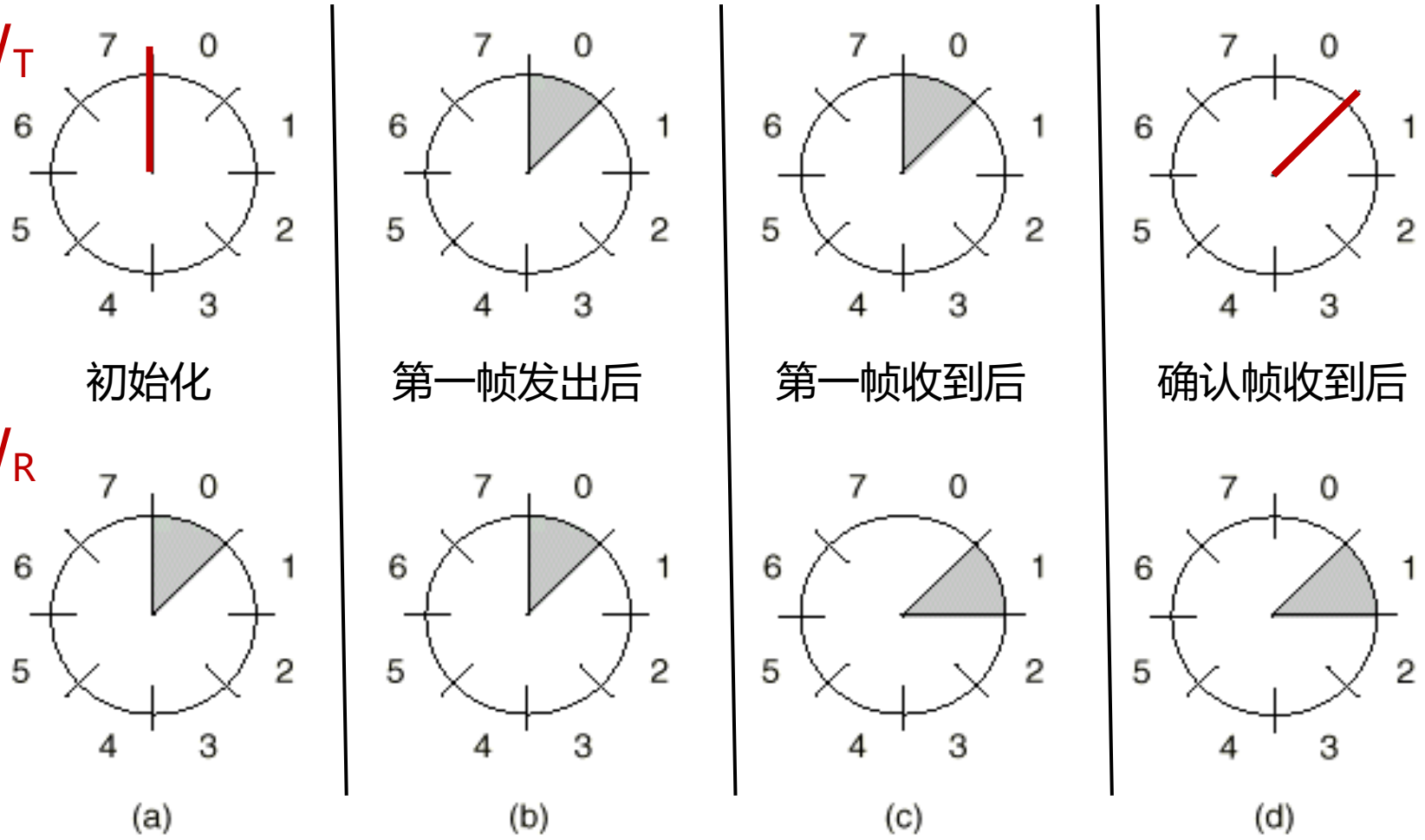
发送方和接收方
代码二合一

一比特滑动窗口协议P4

- 发送窗口与接收窗口(窗口大小1)

发送窗口W_T

接收窗口W_R



一比特滑动窗口协议P4

发送方（ $W_T=W_R=1$，序号空间0和1）

1. 初始化：ack_expected = frame_expected = next_frame_to_send = 0
2. 从网络层接收分组，放入相应的缓冲区，构造帧，物理层发送，开启计时。
3. 等待确认帧到达，从物理层接收一个帧，判断确认号正确，则准备发送新帧（从网络层接收新分组、计算新序号）。
4. 发送该帧，跳转至3

（发送方）

捎带确认

- 两个独立的序号序列
  - 发送方：r.ack确认后，增s.seq本地序号
  - 接收方：接收r.seq对端序号，s.ack确认

```
while(1){
   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) {
         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_timer(s.seq);
}
```

（确认号正确）

分析：1、超时或 2、接收到数据

一比特滑动窗口协议P4

接收方（ $W_T=W_R=1$，序号空间0和1）

1. 初始化：ack_expected = frame_expected = next_frame_to_send = 0
2. 等待帧到达，从物理层接收一个帧，校验和计算，并判断收到的帧序号是否正确，正确则交给网络层处理，期待帧号增加。
3. 返回确认帧，跳转至2。

接收方

- 窗口大小：N = 1，序号取值范围：0，1
- 可进行数据双向传输，信息帧中可含有确认信息（piggybacking技术）
- 信息帧中包括两个序号域：发送序号和确认序号（已经正确收到的帧的序号）

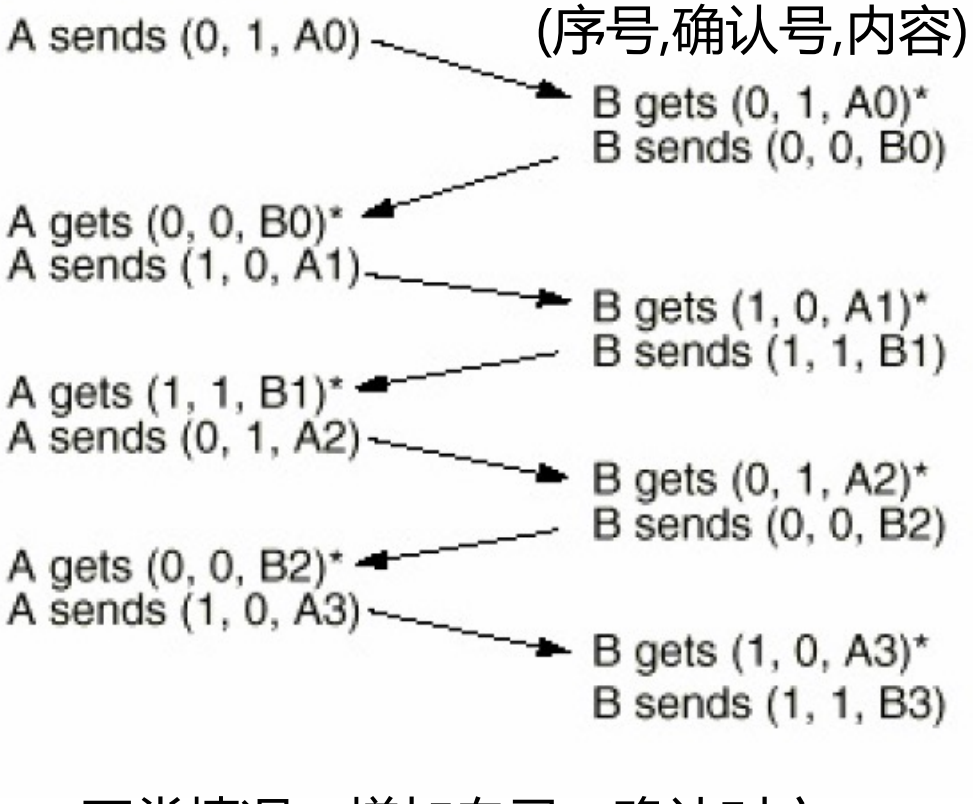
```
while(1){
   wait_for_event(&event);
   if(event==frame_arrival){
      from_physical_layer(&r);
      if(r.seq==frame_expected){
         to_network_layer(&r.info);
         inc(frame_expected);
         }
      if(r.ack==next_frame_to_send) {
         from_network_layer(&buffer);
         inc(next_frame_to_send);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

发送方：r.ack确认后，增s.seq
接收方：接收r.seq，s.ack确认

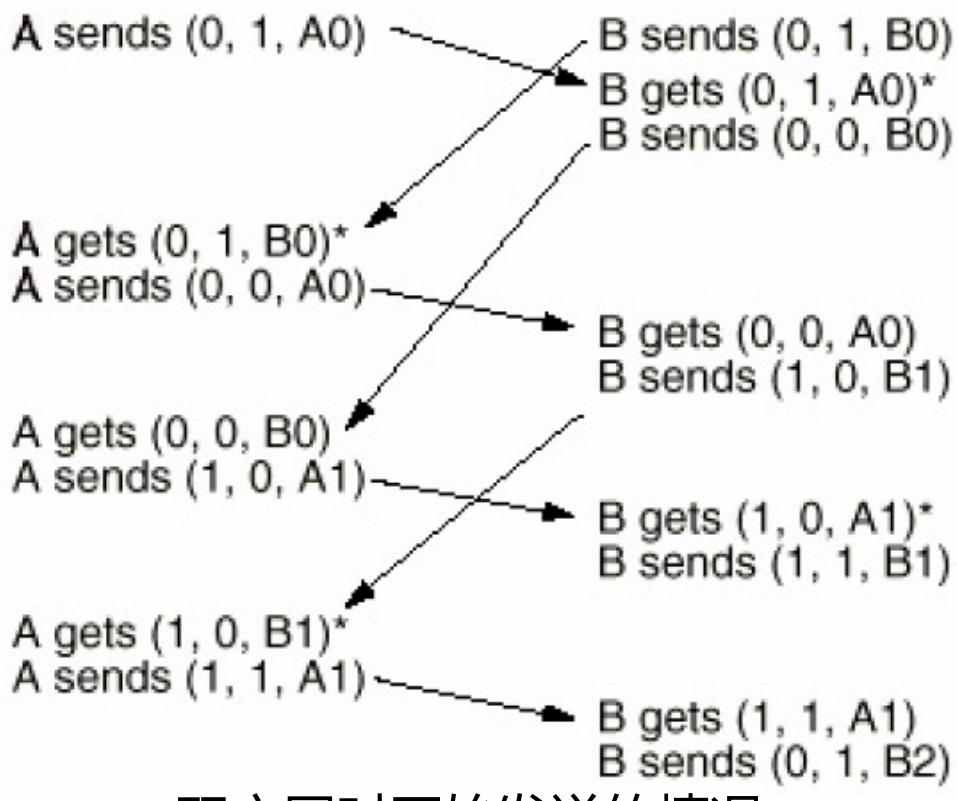
一比特滑动窗口协议P4

- 存在的问题
  - 若双方同时开始发送，则会有一半重复帧

根本问题：停等！



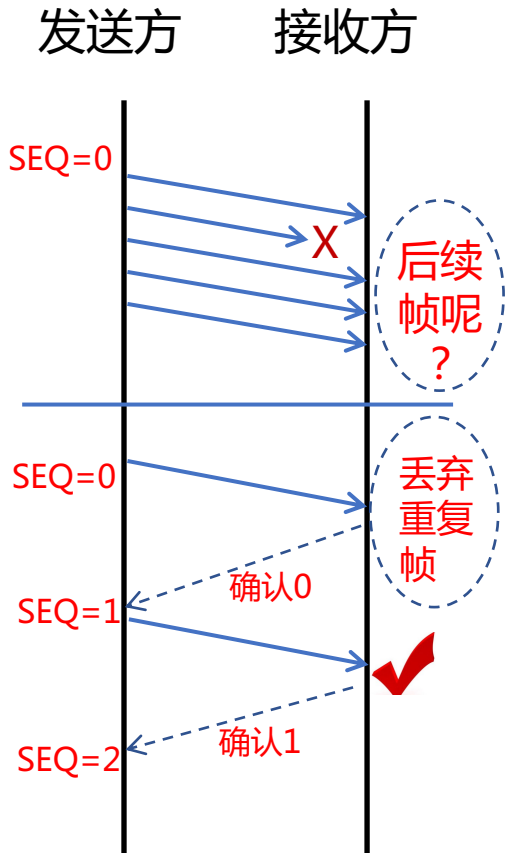
正常情况：增加自己、确认对方



双方同时开始发送的情况

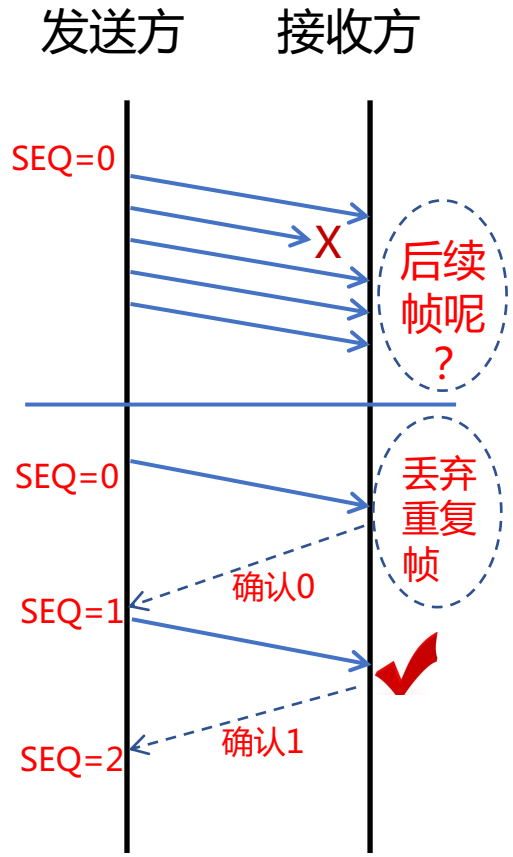
一比特滑动窗口协议P4

- 停止-等待机制降低了信道利用率
  - 假如将链路看成是一根管道，数据是管道中流动的水
  - 能否将水充满管道？在传输延迟较长的信道上，停-等协议无法使数据充满管道，因而信道利用率很低
- 发明新协议？
  - 流水线协议：连续发送多帧后再等待确认
  - 窗口机制：允许发送方在没收到确认前连续发送多个帧？2倍时延带宽积的数据量？
  - 出错怎么办：接收方 or 发送方？



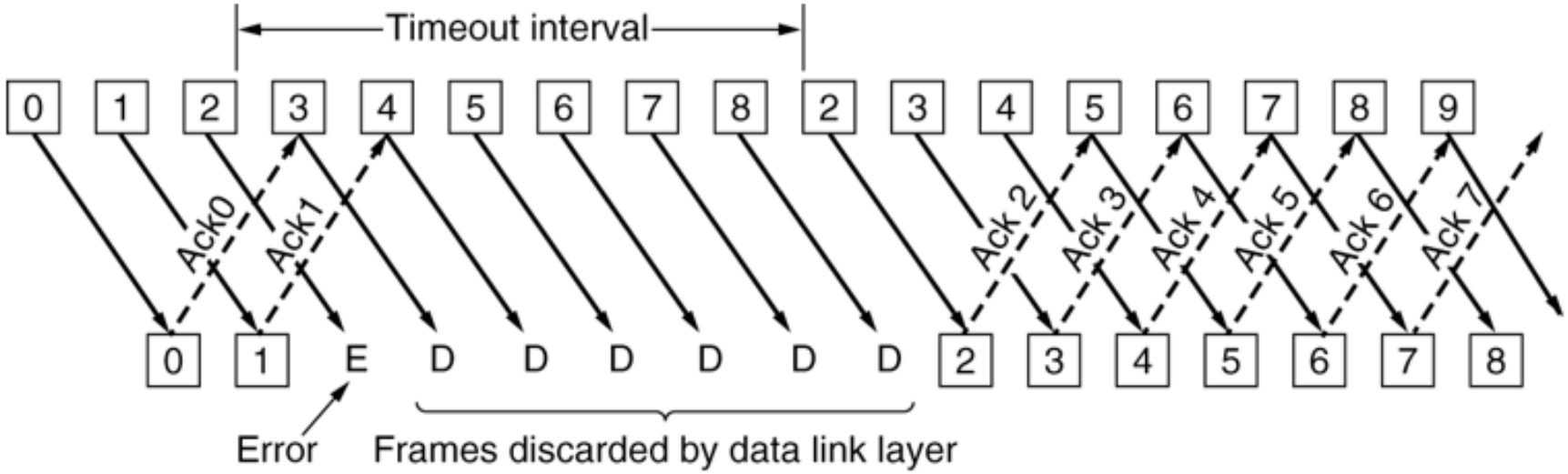
回退N协议P5—设计目标与基本思路

- 设计目标
  - 目标1：向上层按序提交（不能提交乱序或错误内容）
  - 目标2：实现流水线的发送机制，提高信道利用率
  - 简化设计：接收窗口为1
- 出错全部重发
  - 由于接收窗口为1，只能按顺序接收帧
  - 当接收端收到出错帧或乱序帧时，丢弃所有的后继帧，并且不为这些帧发送确认
  - 发送端超时后，重传所有未被确认的帧
- 优缺点
  - 缺点：按序接收，出错后即便有正确帧到达也丢弃重传



回退N协议P5—协议原理分析

- 基本原理
  - 当发送方发送了N个帧后，若发现该N帧的前一个帧在计时器超时后仍未返回其确认信息，则该帧被判为出错或丢失，此时发送方就重新发送出错帧及其后的N帧
- 滑动窗口长度
  - 出错全部重发时，若帧序号为n位，接收窗口$W_R=1$，发送窗口 $W_T \leq 2^n - 1$

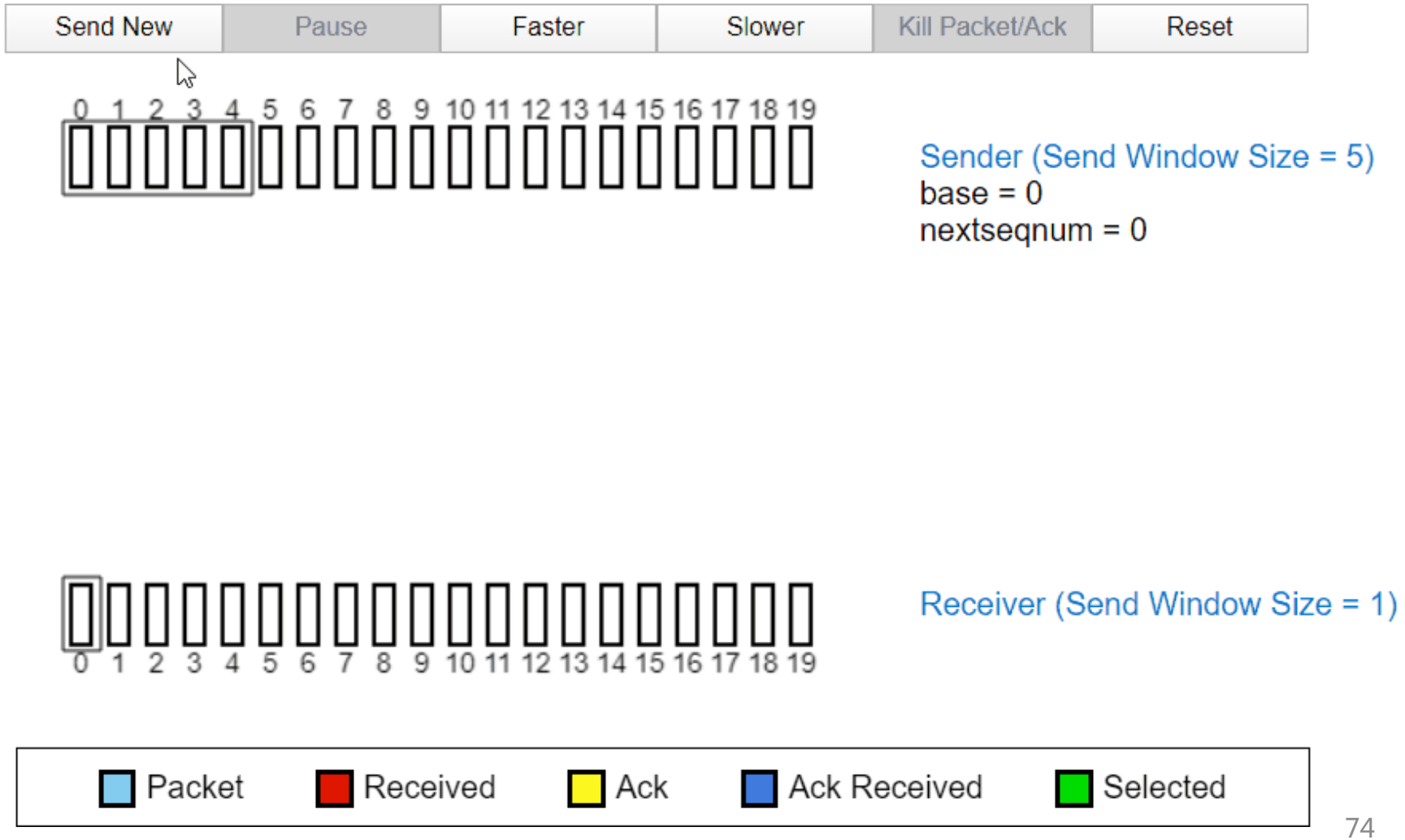


回退N协议P5—协议原理分析

- 正常情况下的GBN

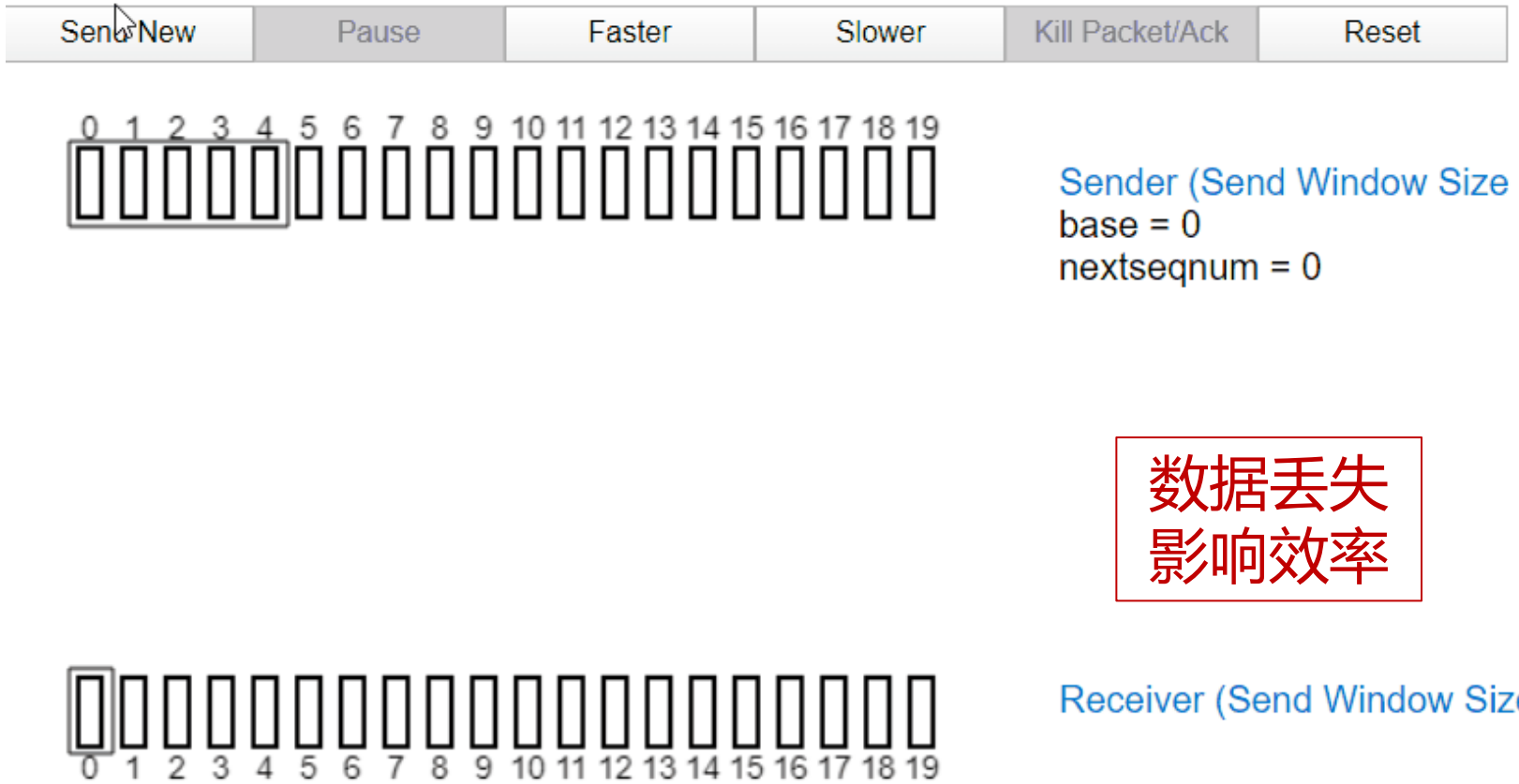
在回退N步协议中，允许发送方发送多个分组而不用等待确认

正常情况下，发送、接收窗口会随着帧的确认状态随包右移



回退N协议P5—协议原理分析

- 发送过程中数据包丢失



2号包发生丢失后的数据？

- 因为接收窗口为1，所以分组3和分组4都被丢弃

2号包发生丢失如何ack？

- 不发送ACK？
- 全部返回ACK1

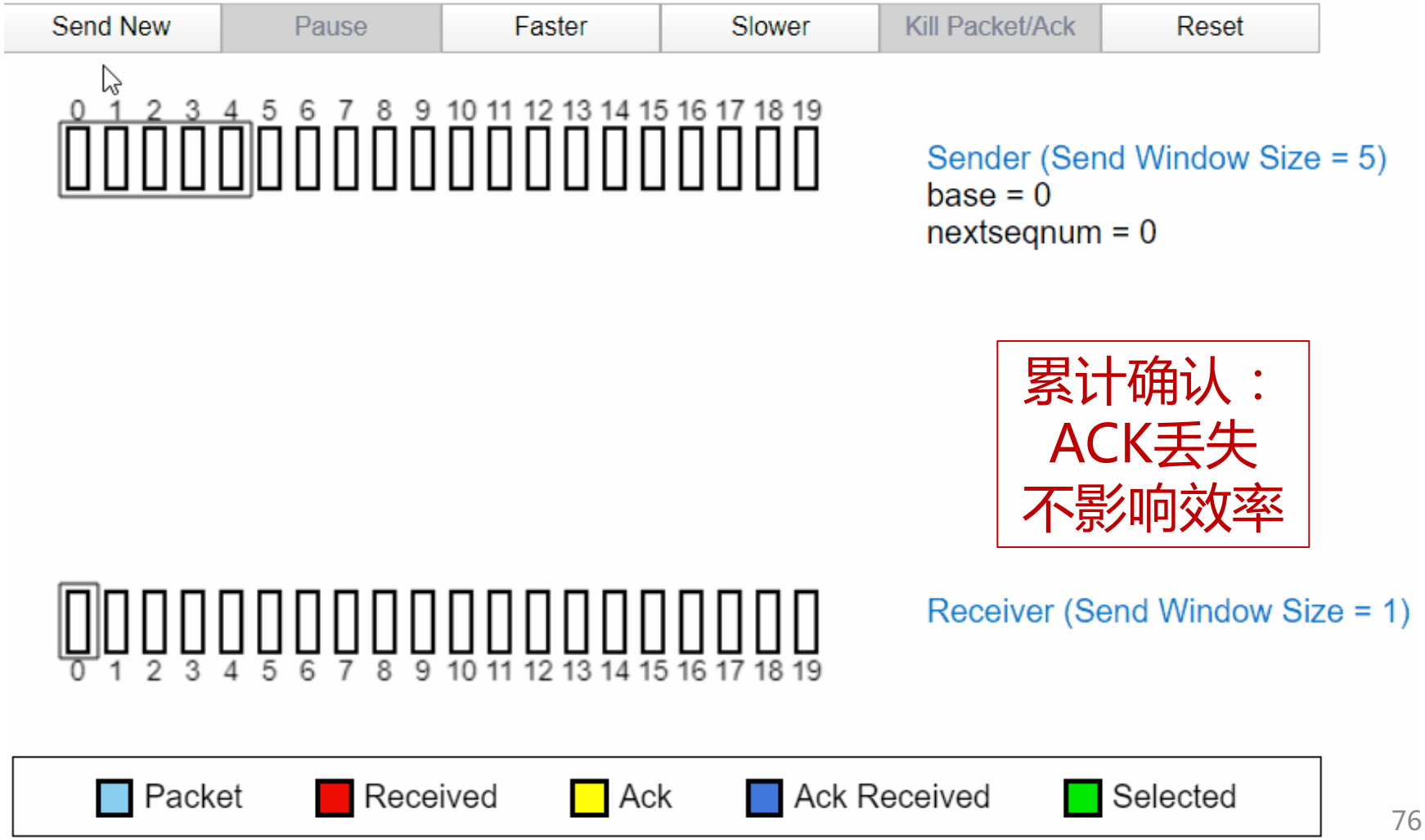
数据丢失
影响效率

回退N协议P5—协议原理分析

- 接收方返回的ACK丢失

累计确认：

- 确认后续序号，代表已收到前面所有序号
- 对收到的分组，可不必逐个发送确认
- 可仅对最后一个分组发送确认



回退N协议P5—协议的实现分析

- 实现要点
  - 发送方为支持重传，发送方需要缓存多个分组，即增加序号范围
- 两个窗口
  - 发送窗口：发送方维持一组连续允许发送的帧序号，不断向前滑动
  - 接收窗口：接收方维持一个允许接收的帧序号，不断向前滑动
- 发送方的三个功能
  - 上层发送数据：检测有没有可以使用的序号，如果有就发送
  - 收到ACK：对n号帧的确认，采用累积确认
  - 超时事件：如果出现超时，就重传已发送未确认的所有分组

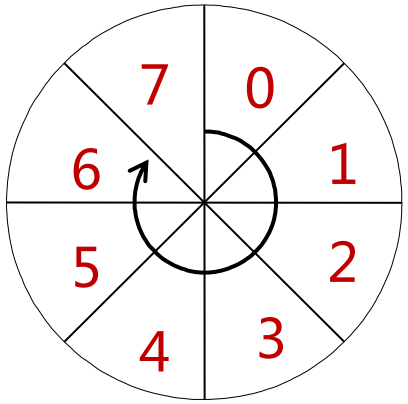
回退N协议P5—协议的实现分析

发送方：

1. 窗口尺寸：$1<W_T\leq 2^n-1$，最多连续发送窗口中的W_T个PDU
2. 窗口滑动：收到期望的ACK(k)：窗口底部移到PDU(k)，最大窗口顶部向前移动
3. 窗口滑动后，发送新进入窗口的PDU，始终保持窗口里最多有W_T个PDU未确认
4. 超时重发： 超过T未收到期望的ACK，重发窗口中的PDU（回退整个窗口）
5. 超次数失败：超过最大重发次数N_{max}仍无正确应答

$W_T=2^n$时面临的问题

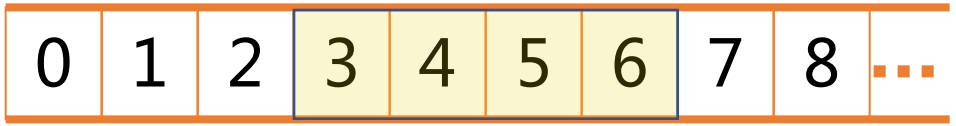
- 发送方无法判断Ack是哪一次的
- 接收方无法判断数据是否是重传（因为ack可能丢失）



回退N协议P5—协议的实现分析

接收方：

1. 窗口尺寸：$W_R=1$
2. 按序接收：按照PDU编号依序接收，出错、乱序PDU一律丢弃
3. 确认含义：ACK(k)表示对k-1及以前各编号的PDU的确认，同时期望接收第k号PDU
4. 确认策略：按序到达的PDU可立即确认，也可延迟确认(收到多帧后一起确认)
5. 但出错或乱序的PDU，只能反向确认NACK(k)（期望接收k号PDU）或不应答，在正确接收到PDU(k)前不能发送ACK(k+1)等



回退N协议P5—协议的实现分析

实现基本过程如下：

1. 初始化。ack_expected = 0 (此时处于发送窗口的下沿)；next_frame_to_send = 0，frame_expected = 0 (初始化正在发送的帧和期待的帧序号)；nbuffered = 0 (进行发送大小初始化)
2. 等待事件发生（网络层准备好，帧到达，收到坏帧，超时）
3. 如果事件为网络层准备好，则执行以下步骤。从网络层接收一个分组，放入相应的缓冲区；发送窗口大小加1；使用缓冲区中的数据分组、next_frame_to_send和frame_expected构造帧，继续发送；next_frame_to_send加1；跳转（7）

```
while(1) {
   wait_for_event(&event);
   switch(event){
       case network_layer_ready:
           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);
          break;
```

回退N协议P5—协议的实现分析

4. 如果事件为帧到达，则从物理层接收一个帧，则执行以下步骤。

首先检查帧的seq域，若正是期待接收的帧（seq = frame_expected），将帧中携带的分组交给网络层，frame_expected加1；然后检查帧的ack域，若ack落于发送窗口内，表明该序号及其之前所有序号的帧均已正确收到，因此终止这些帧的计时器，修改发送窗口大小及发送窗口下沿值将这些帧去掉，继续执行步骤（7）

```
case frame_arrival:
    from_physical_layer(&r);
    if(r.seq==frame_expected) {
        to_network_layer(&r.info);
        inc(frame_expected);
        }
    while(between(ack_expected, r.ack, net_frame_to_send)){
        nbuffered=nbuffered-1;
        stop_timer(ack_expected);
        inc(ack_expected);
        }
    break;
```

回退N协议P5—协议的实现分析

5. 如果事件是收到坏帧，继续执行步骤（7）
6. 如果事件是超时。使next_frame_to_send = ack_expected，从发生超时的帧开始重发发送窗口内的所有帧，然后继续执行步骤（7）
7. 若发送窗口大小小于所允许的最大值（MAX-SEQ），则可继续允许网络层发送，否则则暂停网络层发送

```
  case cksum_err:  ;
      break;
  case timeout:
      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);
          }
  }
  if(nbuffered<MAX_SEQ)
      enable_network_layer();
  else
      disable_network_layer();
}
```

回退N协议的缺点：
重传所有已发送未确认的分组？

选择重传协议P6—协议设计思想

- 设计目标
  - 目标：发送方能否仅重传出错的帧，不重传后续可能正确的帧
- 设计思想
  - 乱序接收：接收端收到的数据包的顺序可能和发送的数据包顺序不一样，因此在数据包里必须含有序号来帮助接收端进行排序
  - 按序交付：暂存落在接收窗口内的帧，等比它序列号小的所有帧都正确接收后，按次序交付给网络层
  - 接收方需要暂存出错帧之后的数据帧，即接收窗口需要大于1
- 优缺点
  - 缺点：在接收端需要占用一定容量的缓存（即接收窗口大于1 ）

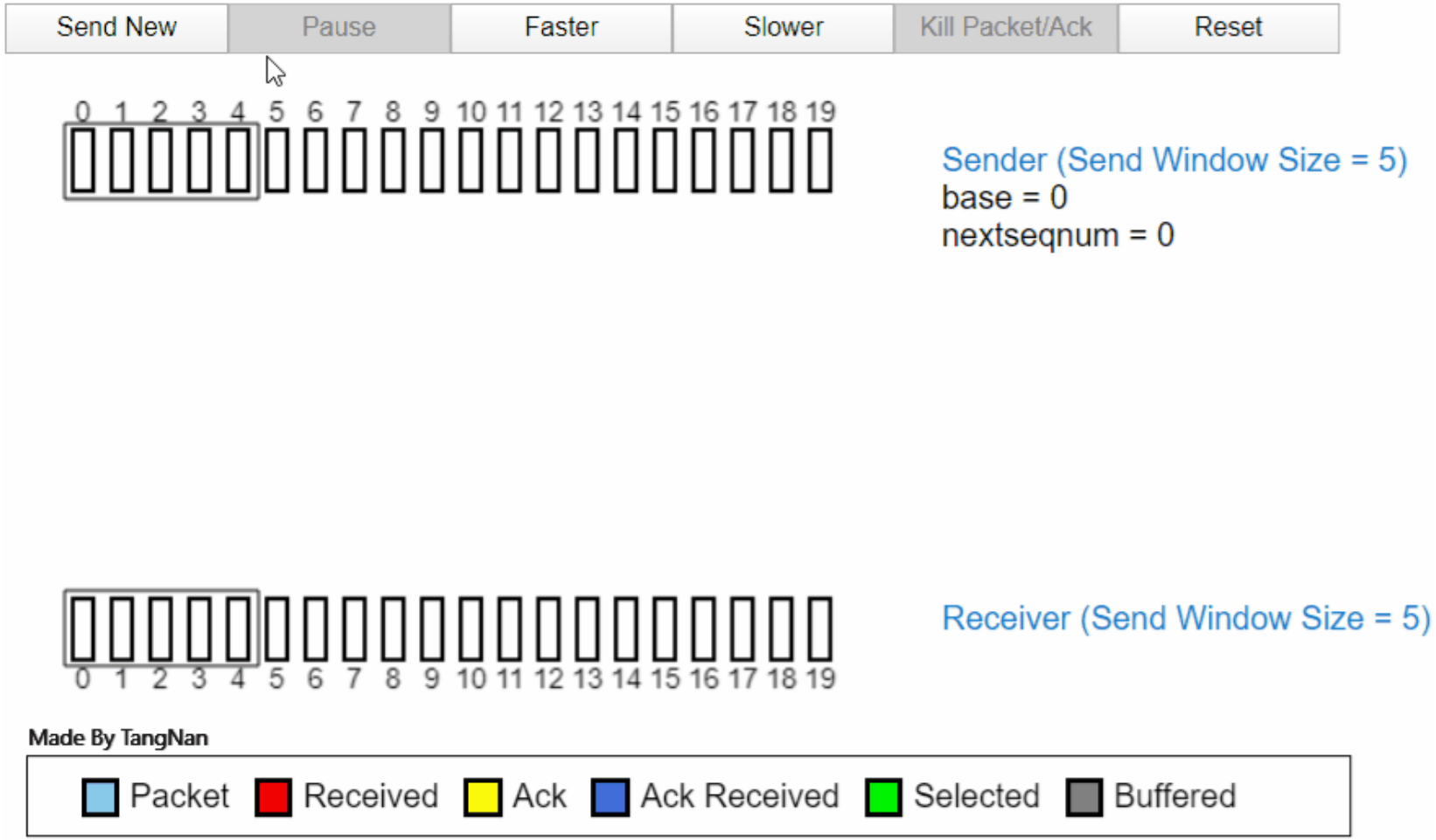
选择重传协议P6——协议原理分析

- 正常情况下SR

允许发送方发送多个分组而不用等待确认

（接收窗口大于1）

两个窗口都随包右移



选择重传协议P6—协议原理分析

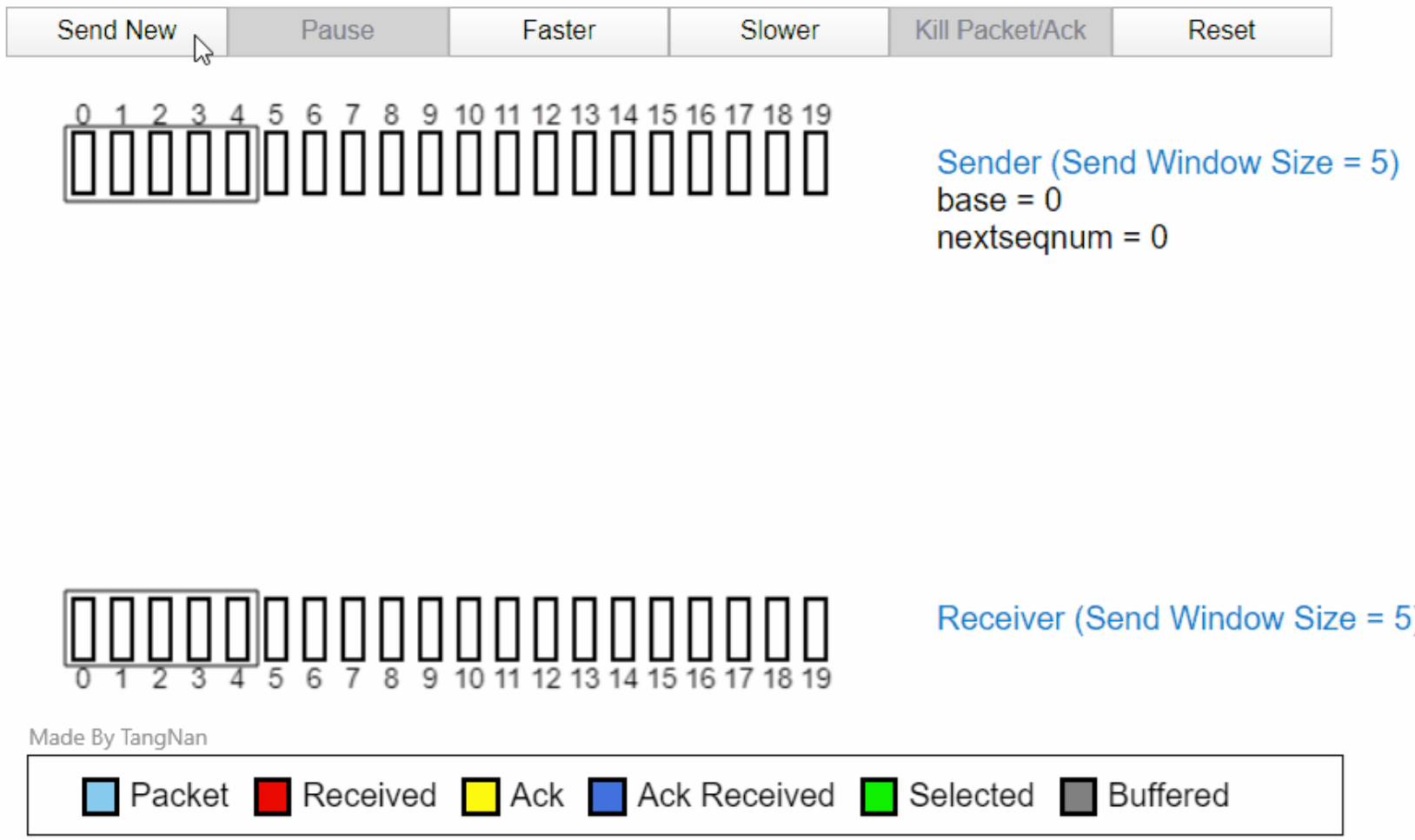
- 发送过程丢包

选择重传必须要知道丢失了哪个分组

2号包丢失：

- 接收方缓存了分组3,4...
- 接收方返回ACK3, ACK4
- 发送方：分组2计时器超时后，重新发送分组2

选择重传不能使用累计确认

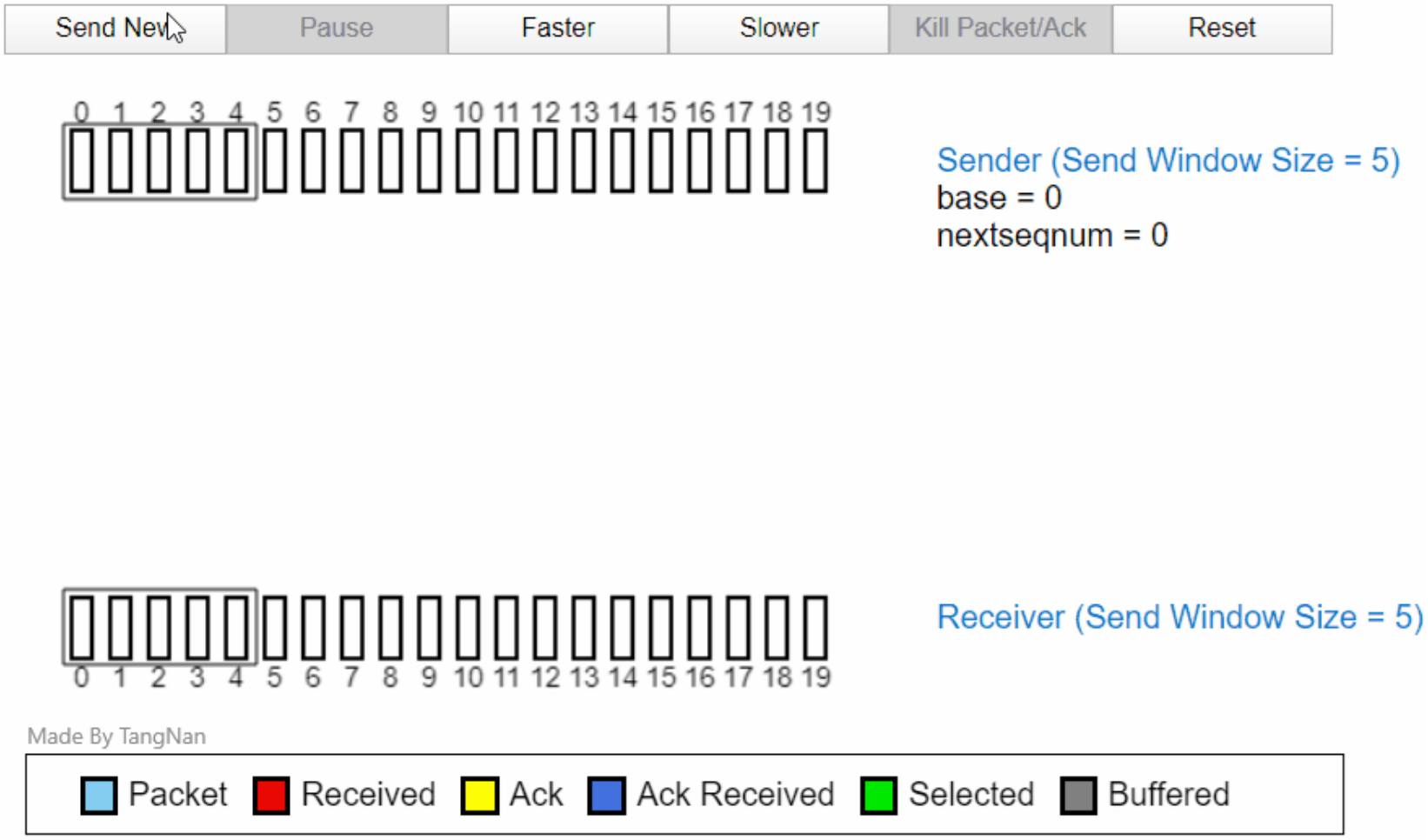


选择重传协议P6——协议原理分析

- 确认过程发生丢包

ACK2丢失：

- 发送方窗口无法滑动，导致发送停滞
- 发送方：分组2计时器超时后会被重新发送



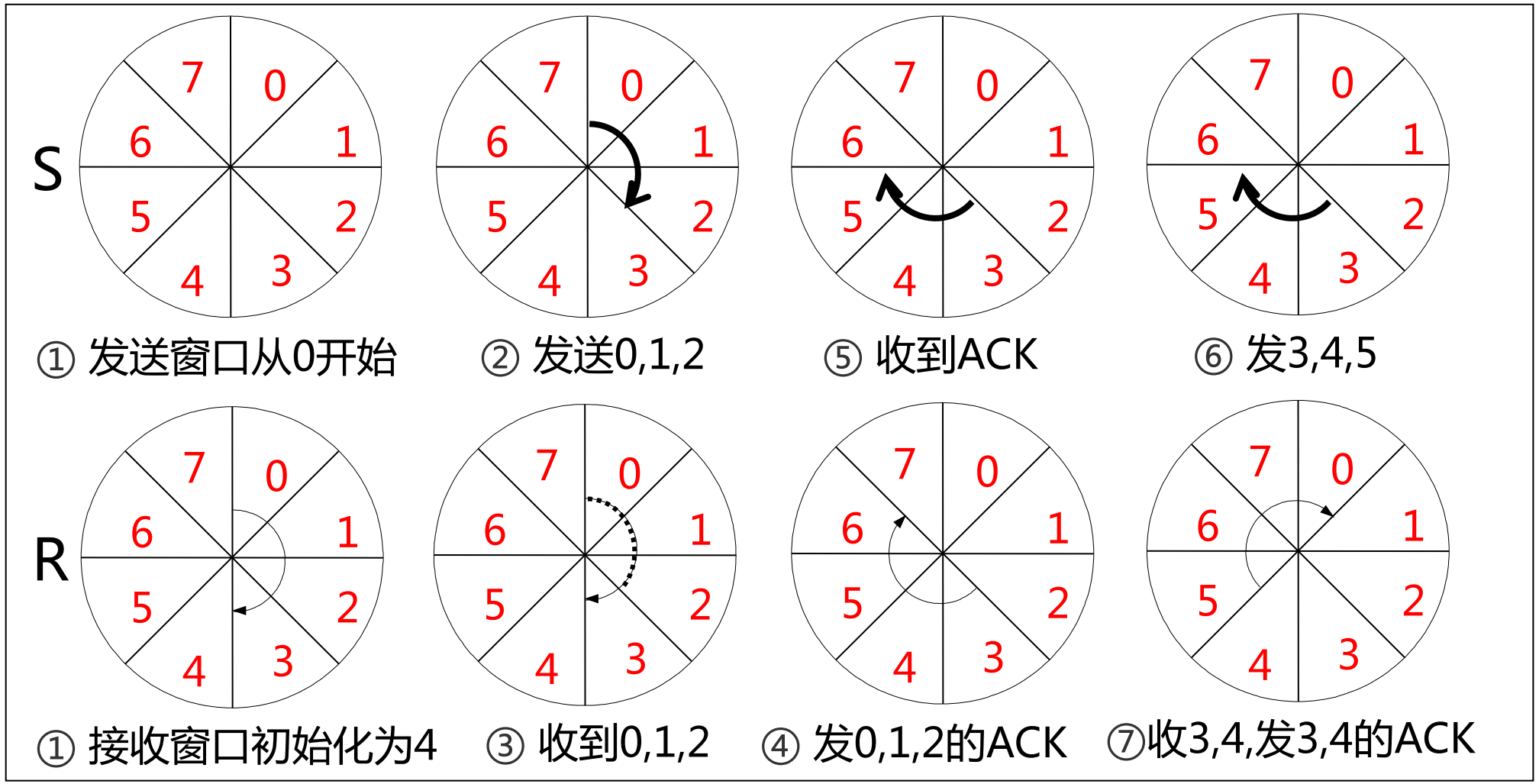
选择重传协议P6—协议原理分析

- 实现要点
  - 与P5 GBN不同，P6是给每个帧设置定时器，发送端只重传出错帧/超时
  - 接收端缓存所收到的乱序帧，当前面帧到达后一起按序提交上层
- 两个窗口
  - 发送窗口：发送方维持一组连续帧序号（以便重传）
  - 接收窗口：接收方维持一组连续的允许接收帧序号（以不被淹没）
- 发送方必须响应的三件事
  - 上层的调用：检测有没有可以使用的序号，如果有就发送
  - 收到ACK：收到最小序号的ACK，则窗口滑动；收到其他序号ACK则标记
  - 超时事件：每个PDU都有定时器，哪个超时重传哪个

选择重传协议P6—协议原理分析

- 发送窗口(大小为3)与接收窗口(大小为4)

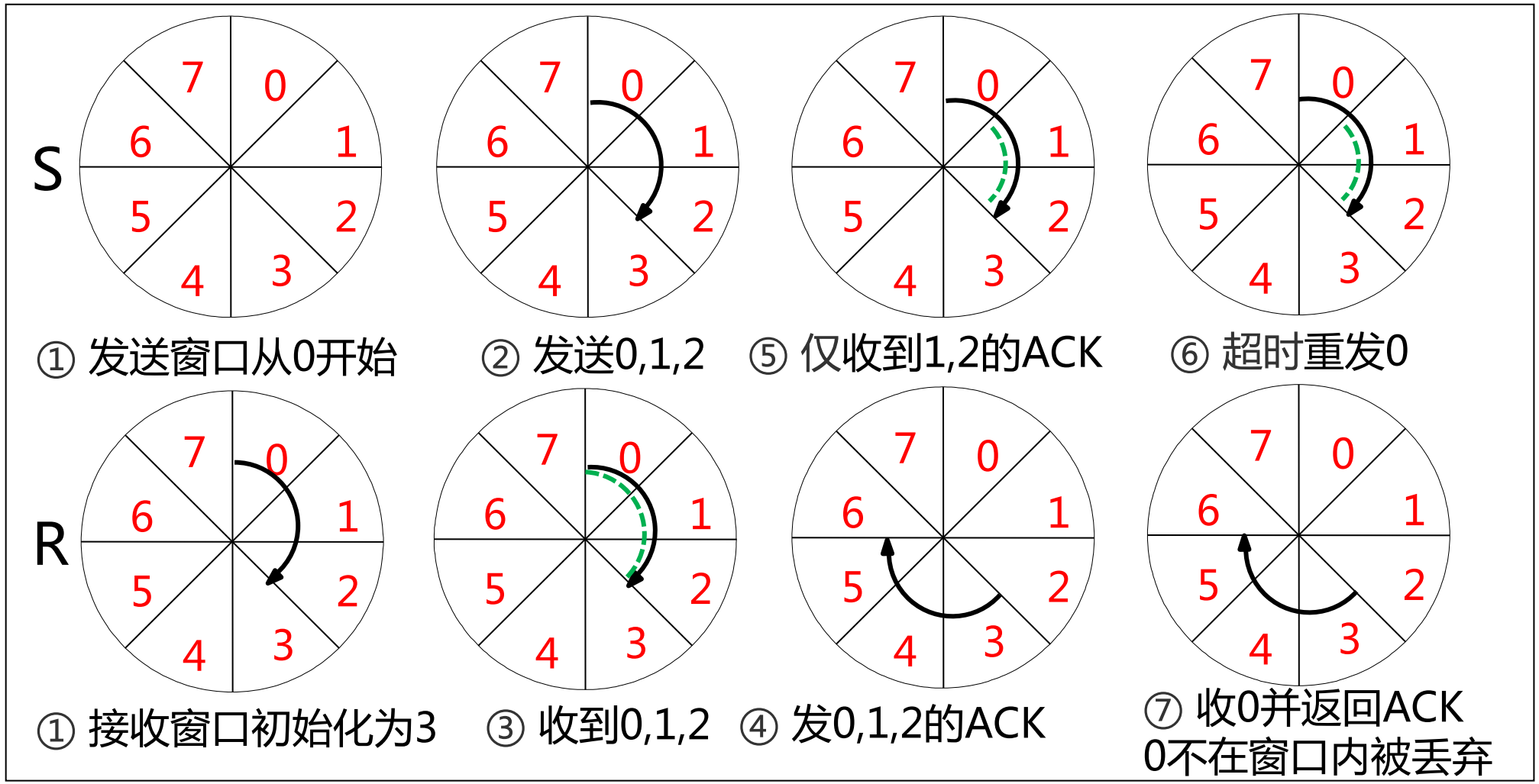
P6 要求窗口大小满足 $1<W \leq 2^{n-1}$，本例中 n 为 3，故窗口最大为 4



选择重传协议P6—协议原理分析

- 发送窗口(大小为3)与接收窗口(大小为3)

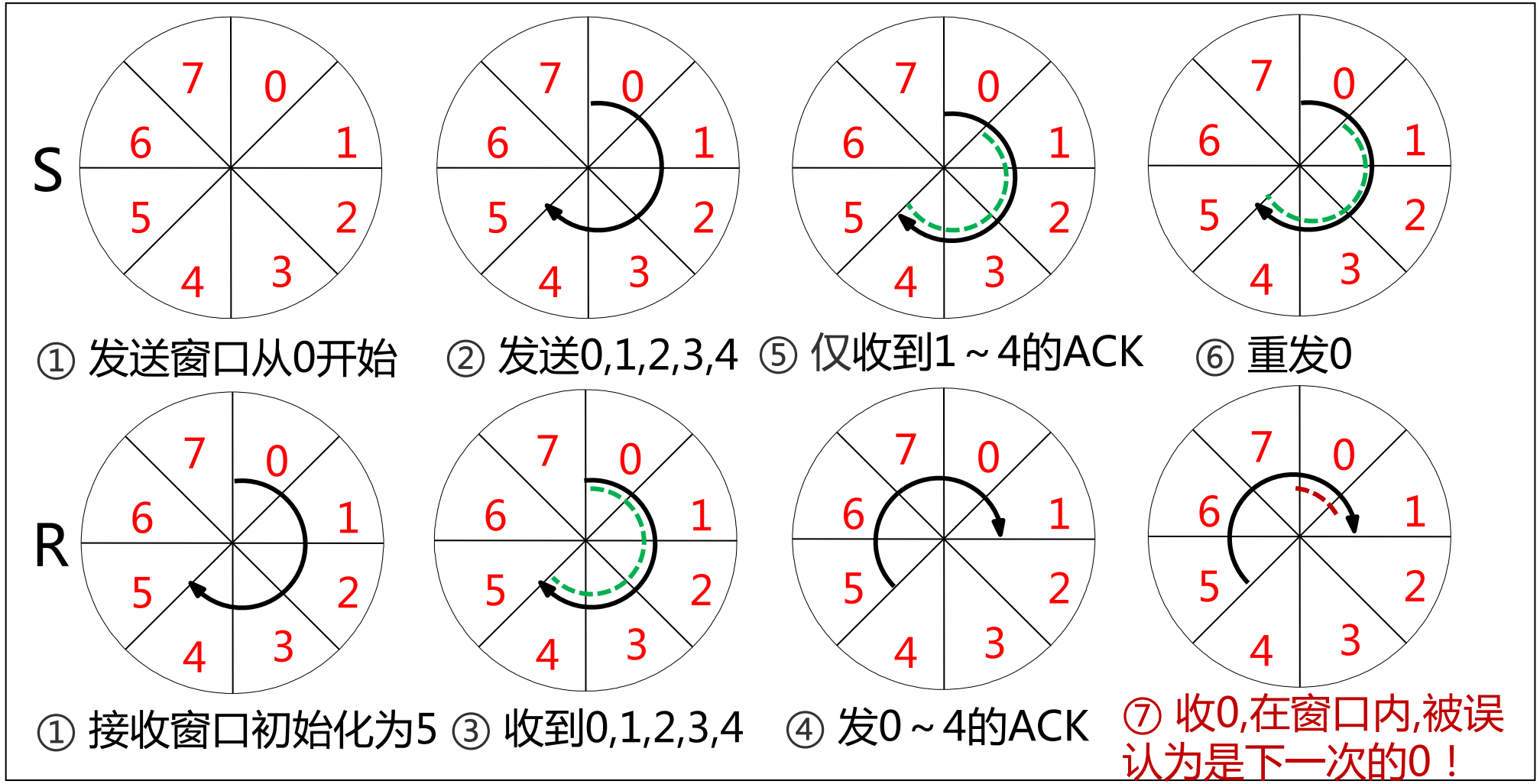
P6 要求窗口大小满足 $1<W \leq 2^{n-1}$，本例中 n 为 3，故窗口最大为 4



选择重传协议P6—协议原理分析

- 发送窗口(大小为5)与接收窗口(大小为5)

P6 要求窗口大小满足 $1<W \leq 2^{n-1}$，本例中 n 为 3，故窗口最大为 4



单选题 1分

设置

分别使用累计确认机制与选择重传机制传输帧。在 ACK 丢失的情况下，请选出效率更高的机制。

A 累计确认

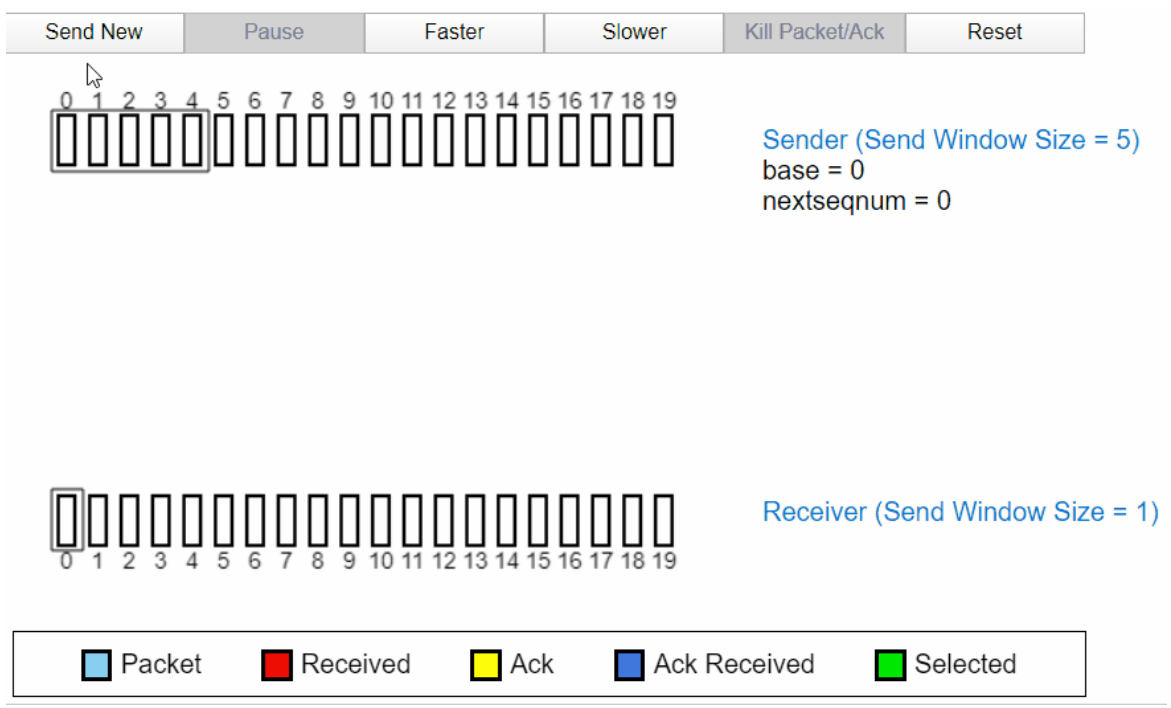
B 选择重传

C 不确定

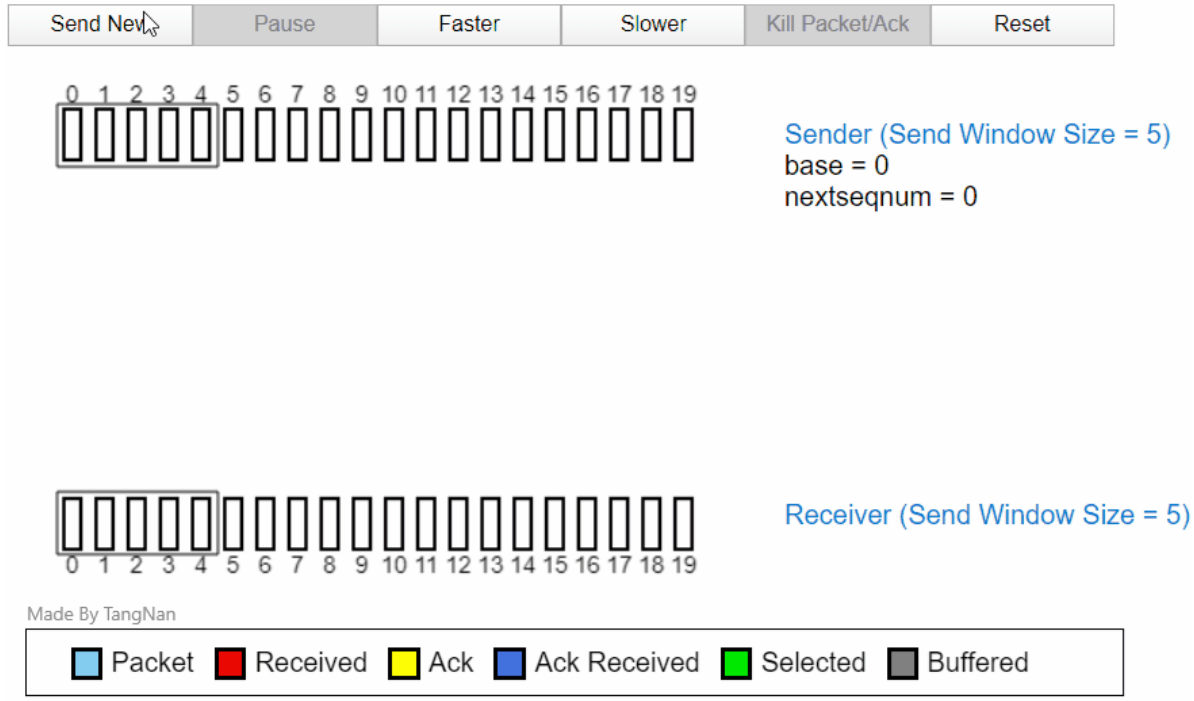
提交

选择重传协议P6—协议原理分析

- 累计确认和选择重传的比较



累计确认在ack丢失时效率高



选择重传在ack丢失时效率低

选择重传协议P6—协议的实现分析

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

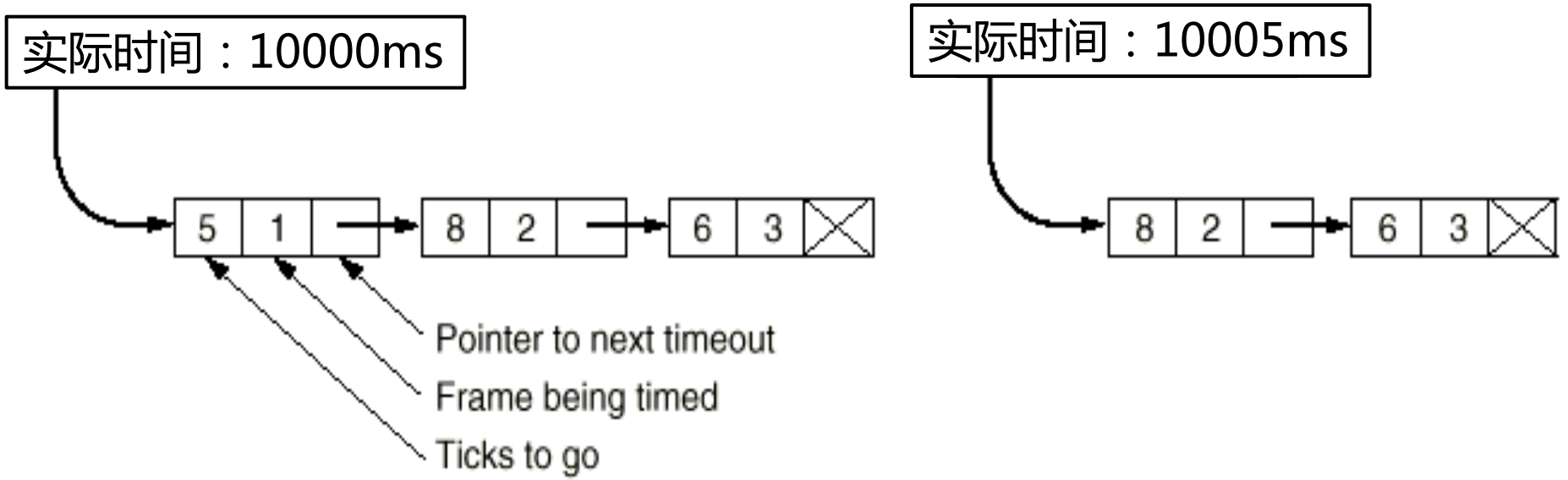
选择重传协议P6—协议的实现分析

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

大量计时器
谁来管理？

选择重传协议P6—协议的实现分析

- 计时器链表
  - 由于有大量未确认帧，如何避免设置大量计时器？
  - 例如：需要设置3个计时器，5ms、13ms和19ms



选择重传协议P6—协议实现分析

实现基本过程如下：

1、初始化。ack_expected = 0 (此时处于发送窗口的下沿)；next_frame_to_send = 0，frame_expected = 0（初始化正在发送的帧和期待的帧序号）；nbuffered = 0（进行发送窗口大小初始化）；

2、等待事件发生（网络层准备好，帧到达，收到坏帧，超时，确认超时）。

3、如果事件为网络层准备好，则执行以下步骤。从网络层接收一个分组，放入相应的缓冲区；发送窗口大小加1；使用缓冲区中的数据分组、next_frame_to_send和frame_expected构造帧，继续发送；next_frame_to_send加1；跳转（8）；

4、如果事件为帧到达，则从物理层接收一个帧，则执行以下步骤。首先检查帧的kind域，若是数据包，再检查seq域，若不是期待接收的帧（seq ！= frame_expected）并且不是nak，则发送nak，否则开启定时器；如果seq落入接收窗口之内并且没有被接收，则接收帧，将帧中携带的分组交给网络层，frame_expected、too_far加1,开启确认定时器；若kind为nak则重新发送数据。最后检查帧的ack域，若ack落于发送窗口内，表明该序号及其之前所有序号的帧均已正确收到，因此终止这些帧的计时器，修改发送窗口大小及发送窗口下沿值将这些帧去掉，继续执行步骤（8）；

5、如果事件是收到坏帧，如果no_nak为真，则发送nak帧，然后继续执行步骤（8）。

6、如果事件是发送超时，即：next_frame_to_send = oldest_frame，则重发超时帧，然后继续执行步骤（8）。

7、如果事件是确认超时，则重发超时的确认帧，然后继续执行步骤（8）。

8、若发送窗口大小小于所允许的最大值（MAX_SEQ），则可继续向网络层发送，否则则暂停继续向网络层发送，同时返回互步骤（2）等待。

选择重传协议P6—协议的实现分析

核心代码

```
while(1) {
   wait_for_event(&event);   // 包括5种情况
   switch(event) {
      case network_layer_ready:  // 从网络层接收数据，传输新帧
         from_network_layer(&out_buffer[next_frame_to_send%NR_BUFS]);
         nbuffered=nbuffered+1;
         send_frame(data,next_frame_to_send, frame_expected, out_buffer);
         inc(next_frame_to_send);
         break;
      case frame_arrival:  // 数据帧或控制帧到达
         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)) {  // 乱序收帧
               arrived[r.seq%NR_BUFS]=true;
               in_buf[r.seq%NR_BUFS]=r.info;
               while(arrived[frame_expected%NR_BUFS]){  // 收到收端窗口下界的帧，接收窗口右移
                  to_network_layer(&in_buf[frame_expected%NR_BUFS]);    // 按序将帧上传到网络层
                  no_nak=true;
                  arrived[frame_expected%NR_BUFS]=false;
```

若收到的帧不是收端窗口下界 frame_expected 的帧，则发送对应 nak，请求重传收端窗口下界的帧（这里 nak 是累计确认），不发送 ack；否则发送 ack

选择重传协议P6—协议的实现分析

核心代码

```
                inc(frame_expected);
                inc(too_far);
                start_ack_timer();   // 准备为当前收端窗口下界的帧发 ack（ack 左侧的帧均已收到，
                }                    这里 ack 也是累计确认的）
            }
        }
    if((r.kind==nak) && between(ack_expected ,(r.ack+1)%(MAX_SEQ+1), next_frame_to_send))
        send_frame(data,(r.ack+1)%(MAX_SEQ+1),frame_expected, out_buf);
    while(between(ack_expected, r.ack, next_frame_to_send)) {
        nbuffered=nbuffered-1;
        stop_timer(ack_expected%NR_BUFS);
        inc(ack_expected);
        }
    break;
  case cksum_err: if(no_nak) send_frame(nak, 0, frame_expected, out_buf); break;  // 帧出错，发送 nak
  case timeout: send_frame(data, oldest_frame,frame_expected,out_buf); break;  // 超时
  case ack_timeout: send_frame(ack, 0, frame_expected, out_buf); break;  // ack 定时器超时，发送 ack
  }
if (nbuffered<MAX_BUFS)  enable_network_layer();  else  disable_network_layer();
}
```

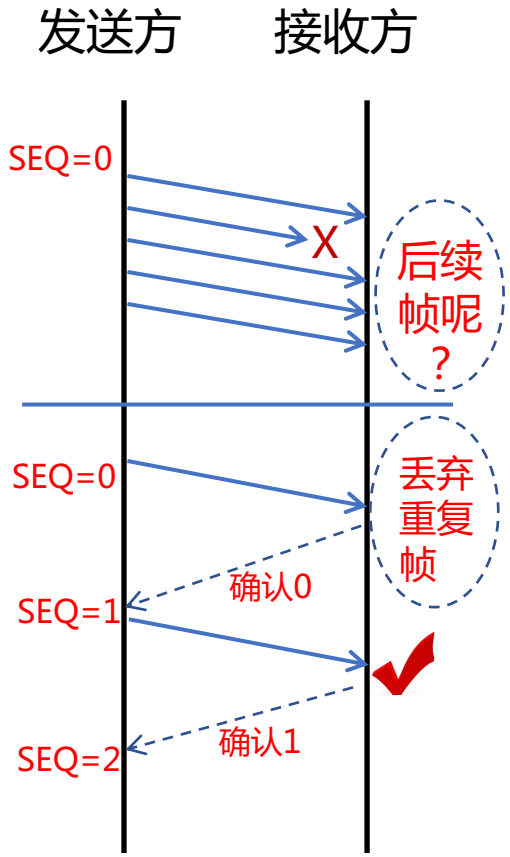
引入 NAK 后，选择重传协议 P6 的 ACK 和 NAK 都是累计确认的

滑动窗口协议—小结

- 基本的数据链路层协议
  - 乌托邦式单工协议P1
  - 交互：无错信道上的停等协议P2
  - 差错：有错信道上的停等协议P3
- 一比特滑动窗口协议P4
  - 捎带确认：发送方与接收方二合一
  - 仍为停等：发送窗口等于1，接收窗口等于1
- GBN/回退N协议P5
  - 流水线：可连续发多帧增加在途数据
  - 发送窗口大于1，接收窗口等于1
  - 接收方从坏帧起丢弃所有后继帧，发送方从坏帧开始重传
- 选择重传协议P6
  - 减少重传：接收方可暂存坏帧的后继帧，发送方只重传坏帧
  - 发送窗口大于1，接收窗口大于1
  - 接收窗口较大时，需较大缓冲区

学到大招了吗？
从最简单开始
逐步深入
越来越实际

窗口大小是核心
优缺点？



思考

- 如何实现可靠传输？
  - 纠错编码 v.s. 检错码、确认和重传机制
- 适用性分析？
  - 链路层的可靠传输服务通常用于高误码率的链路，如无线链路
  - 对于误码率低的链路，链路层协议可以不实现可靠传输功能
- 高层协议呢？
  - 传输层是否有必要再实现丢失重传的可靠传输服务？

优化与折衷
链路利用率、复杂程度和协议开销
（与网络环境和上层需求相关）

本节内容

4.1 数据链路层的定义和功能

4.2 基本的数据链路层协议

4.3 滑动窗口协议

4.4 典型链路层协议

1. 高级数据链路控制协议
2. 点对点协议

高级数据链路控制协议

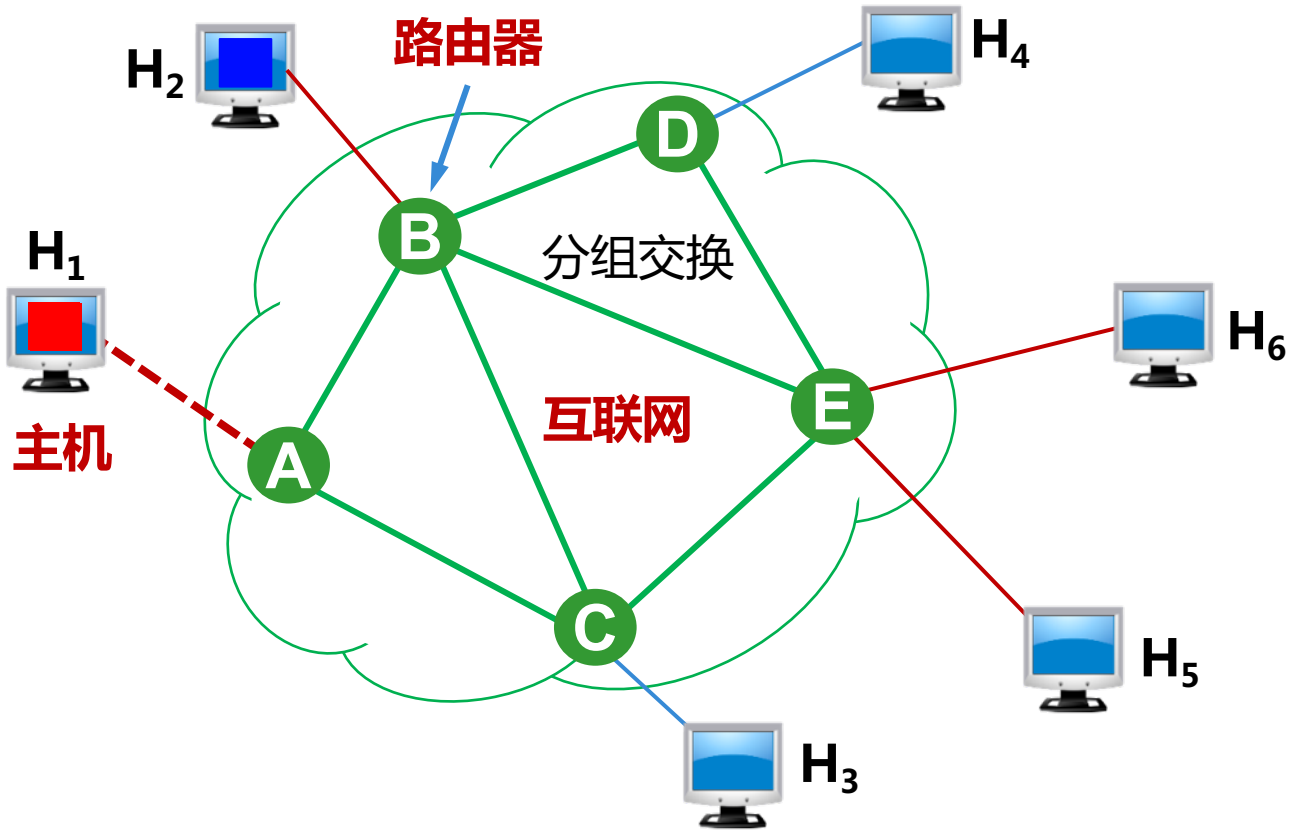
➢ HDLC协议介绍

- 高级数据链路控制HDLC (High-level Data Link Control)协议
- 帧头和帧尾都是特定的二进制序列，即帧标志：01111110作为帧的边界
- 可以采用多种编码方式实现高效、可靠的透明传输
- 校验字段：使用16bit的CRC-CCITT标准，或32bit的CRC-32校验
- 超时断连，递增序号，流量控制和差错控制
- 由国际标准化组织（ISO）颁布
- 被用于早期的X.25协议和帧中继网络

核心路由器的需求

核心路由器
高速网口100G
16~64个
核心诉求是路由转发
重传会有什么问题？

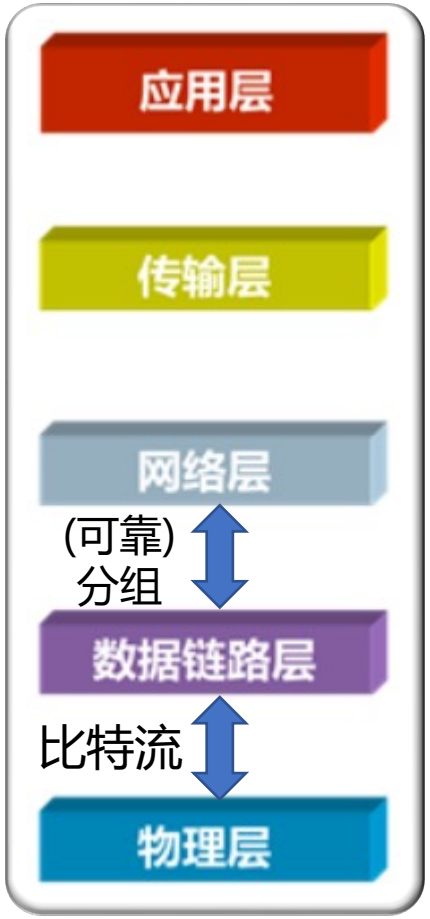
互联网设计原则
核心路由器
只做不得不做的任务
避免了重传怎么办？



世界上最为复杂的人造系统之一

从DHLC到更为简单的PPP

- 链路层的简化
  - 随着通信技术进步，信道可靠性大幅提升（路由器带宽不断增大）
  - 没有必要（难以）在链路层使用复杂协议（序号、检错、重传等）来实现数据的可靠传输
  - 不可靠传输协议PPP已成为数据链路层主流协议
  - 可靠传输责任落到传输层TCP协议上

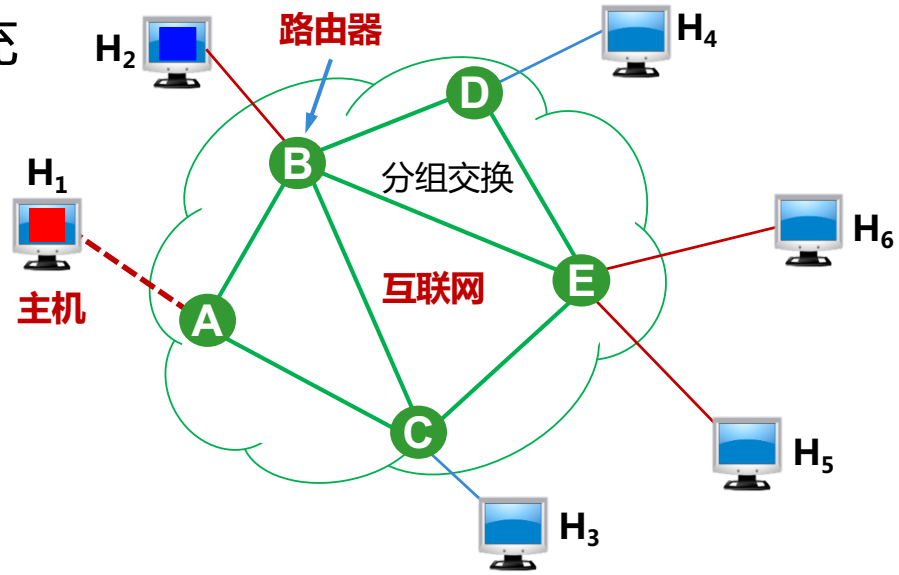


PPP协议简介

- PPP协议
  - PPP(Point-to-Point Protocol)协议由IETF制定，1994年成为RFC1661
  - PPP协议是目前使用最多的数据链路层协议之一
  - 能在不同链路上运行，能承载不同的网络层
- 主要功能特点
  - 利用帧定界符封装成帧：字节填充、零比特填充
  - 帧的差错检测
  - 实时监测链路工作状态
  - 设置链路最大传输单元（MTU）
  - 网络层地址协商机制
  - 数据压缩协商机制

PPP是点到点，不是点到多点，更不是端到端。

简单、灵活



PPP协议未实现的功能

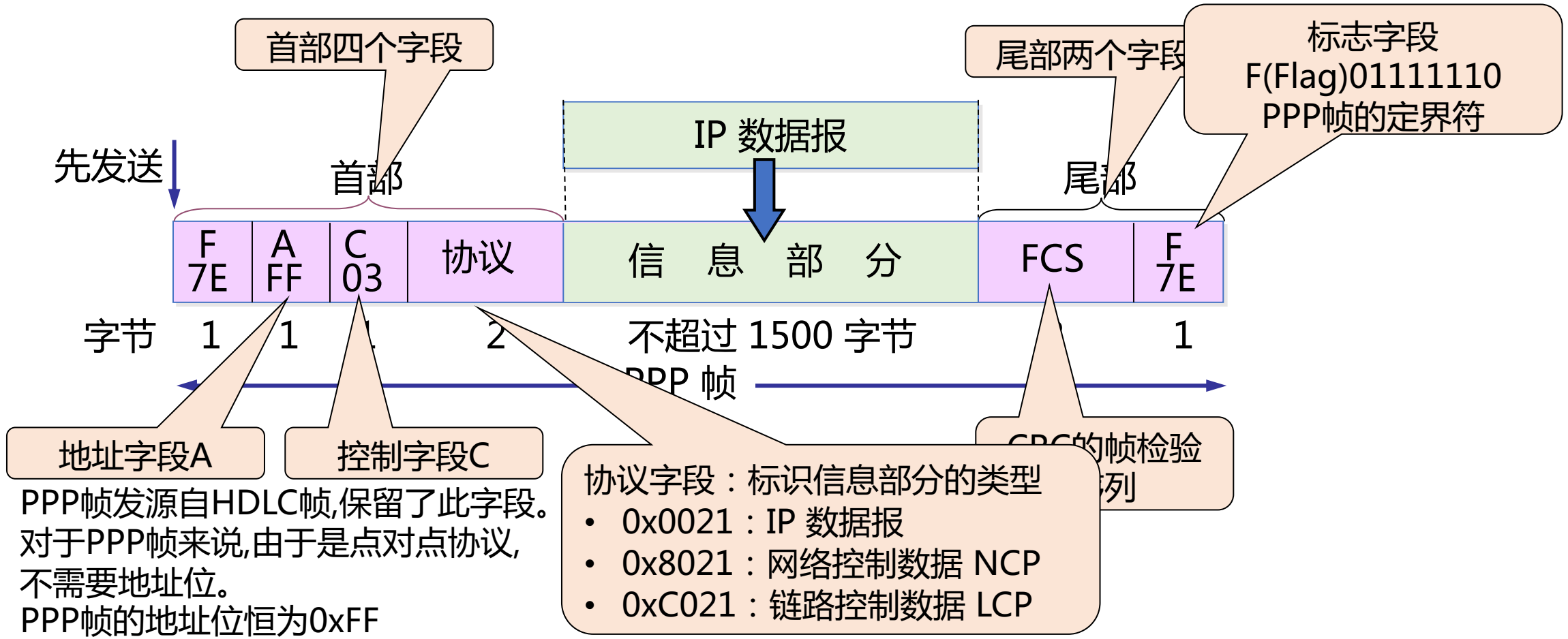
未实现的功能

- 帧数据的纠错功能
  - 数据链路层的PPP协议只进行检错，PPP协议是不可靠传输协议
- 流量控制功能
  - PPP协议未实现点到点的流量控制
- 单工和半双工链路
  - PPP协议支持全双工链路
- 可靠传输功能
  - PPP为不可靠协议
  - 不使用帧的序号（不可靠网络中可能使用有序号的工作方式）
- 多点连接功能
  - PPP协议不支持多点线路，只支持点对点的链路通信

PPP协议的构成

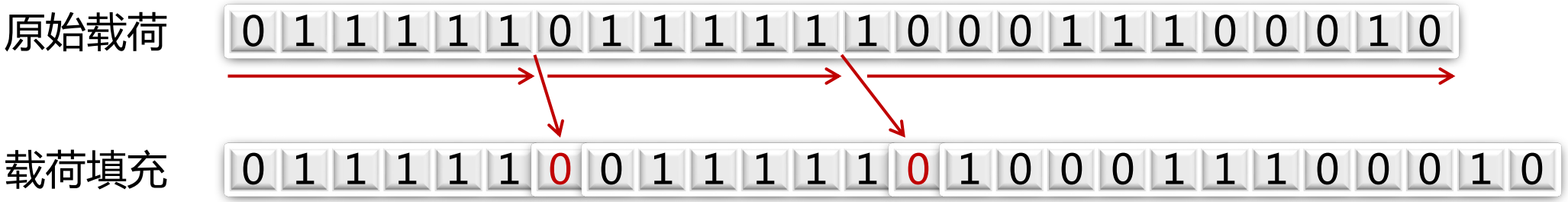
- 封装 (Encapsulation)
  - 提供在同一链路上支持不同的网络层协议
  - IP数据包在PPP帧中是其信息部分，其长度受到MTU的限制
  - PPP既支持异步链路（无奇偶检验的8比特数据），也支持面向比特的同步链路
- 链路控制协议 LCP (Link Control Protocol)
  - 用来建立、配置和测试数据链路的链路控制协议，通信双方可协商一些选项
- 网络控制协议 NCP (Network Control Protocol)
  - 其中每个协议支持一种不同的网络层协议，如IP、OSI的网络层、DECnet、AppleTalk等

PPP协议的帧格式


首部四个字段
尾部两个字段
标志字段
F(Flag)01111110
PPP帧的定界符
IP 数据报
先发送
首部
尾部
F
7E
A
FF
C
03
协议
信 息 部 分
FCS
F
7E
字节 1 1 1 2 不超过 1500 字节 2 1
PPP 帧
地址字段A
控制字段C
协议字段：标识信息部分的类型
• 0x0021：IP 数据报
• 0x8021：网络控制数据 NCP
• 0xC021：链路控制数据 LCP
PPP帧发源自HDLC帧,保留了此字段。
对于PPP帧来说,由于是点对点协议,
不需要地址位。
PPP帧的地址位恒为0xFF

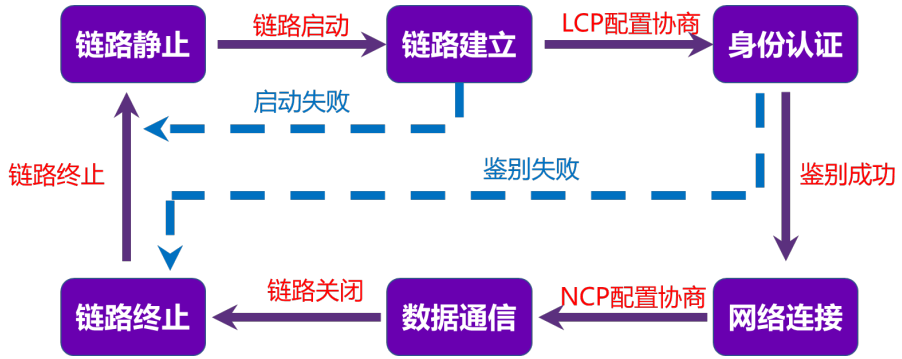
PPP协议的成帧方式

- PPP通常使用的字符填充法
  - 避免在信息字段中出现和标志字段一样的比特组合（0X7E）
  - 当PPP使用异步传输时，定义转义字符0X7D，并使用字节填充
  - 发送端进行字节填充，链路上的字节数超过上层发送的字节数
- PPP支持带**比特**填充的定界符法
  - PPP协议用在SONET/SDH链路时，采用标志字段0x7E
  - 若在有效载荷中出现连续5个1比特，则直接插入1个0比特



PPP协议的工作状态及转换

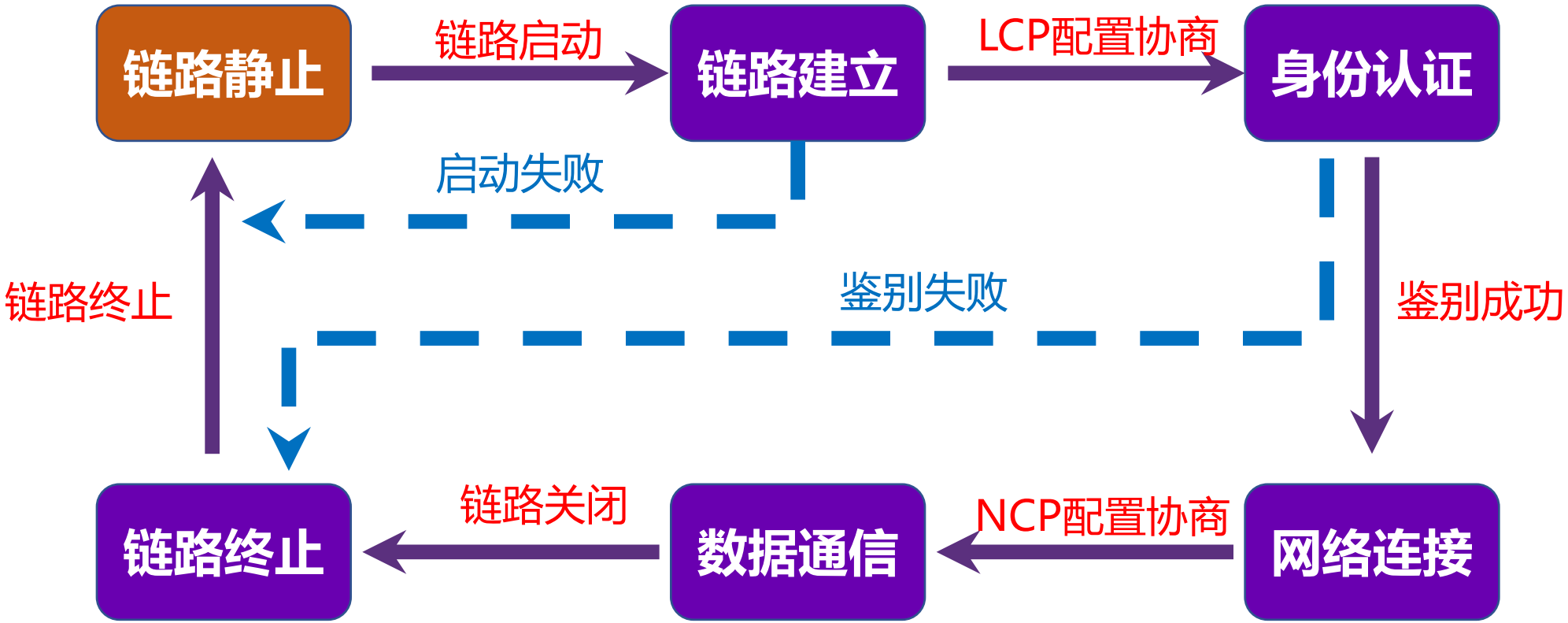
- 状态转换图(State Transform Diagram)：状态机
  - 通过描述系统的状态和引起系统状态转换的事件，指出作为特定事件的结果将执行哪些动作，从而描述系统的行为
  - 状态转换图中以节点表示状态，有向边来表示内外部事件和响应（变迁）
- 状态
  - PPP协议的状态转换图中共设置了六个状态



- 状态的变迁
  - PPP协议中通过一些外部事件来触发状态的变迁，如链路启动、LCP配置协商、鉴别成功、NCP配置协商、链路关闭、链路终止等

PPP协议的工作状态及转换

- 采用状态转换图来表示PPP协议工作的行为模型



本章总结

- 成帧的方式
  - 字节计数法，带字节填充的定界符法，带比特填充的定界符法
- 差错检测和纠正
  - 海明距离
  - 检错码：奇偶校验，校验和，循环冗余校验
  - 纠错码：海明码
- 基本的数据链路层协议
  - 乌托邦式单工协议P1
  - 无错信道上的停等协议P2
  - 有错信道上的停等协议P3
- 提升效率的滑动窗口协议
  - 一比特滑动窗口协议P4
  - 回退N协议P5
  - 选择重传协议P6
- 数据链路协议实例（了解）
  - PPP协议
  - 协议状态机的描述方式

思考与展望

- 数据链路层
  - 终于搞定了点到点信道的传输难题
  - 接下来呢？
- 共享信道访问？
  - 局限在一跳范围内
  - 多个设备难以协同
  - 互相冲突怎么办？
  - 集中控制v.s.分布式？
  - 发明：交换机、WLAN