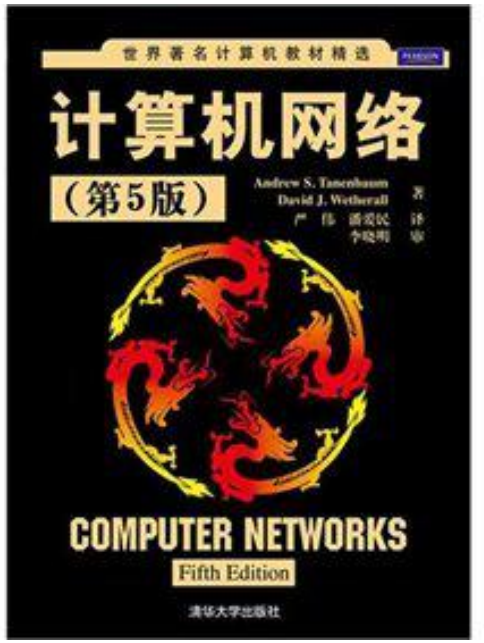
计算机网络

Andrew S. Tanenbaum（5 Edition）



安徽大學 互联网学院
School of Internet Anhui University

安徽大学 Anhui University

计算机网络

第1章 引言

第2章 物理层

第3章 数据链路层

第4章 介质访问控制子层

第5章 网络层

第6章 传输层

第7章 应用层

第8章 网络安全

第二章内容回顾

- **数据通信基础**

  傅里叶分析、奈奎斯特定理、香农定理

- **常用传输介质**

  导向性介质

  非导向性介质

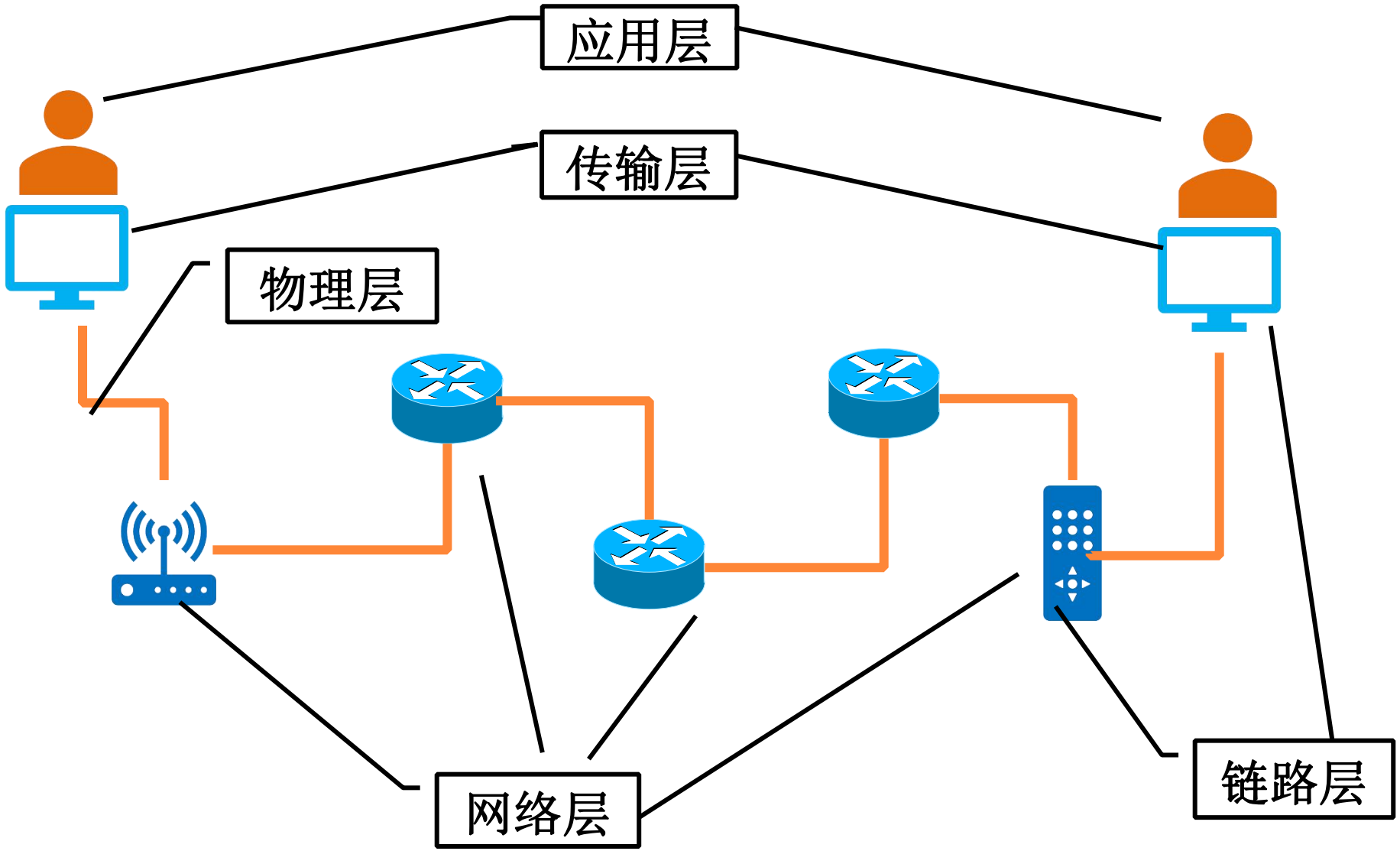
- **信道复用技术**

- **数字传输宽带接入技术**

  信息编码、宽带接入技术、移动接入技术

第3章　数据链路层

第3章　数据链路层



第3章 数据链路层

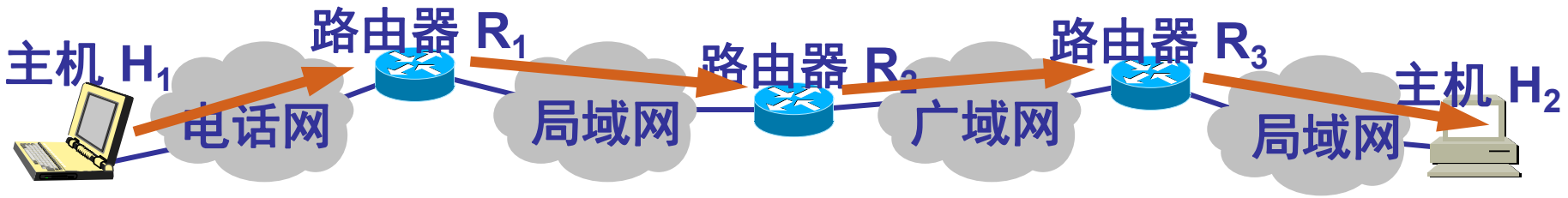
3.1 链路层设计问题
3.2 差错检测与纠正
3.3 基本数据链路层协议
3.4 滑动窗口协议
3.5 数据链路协议实例

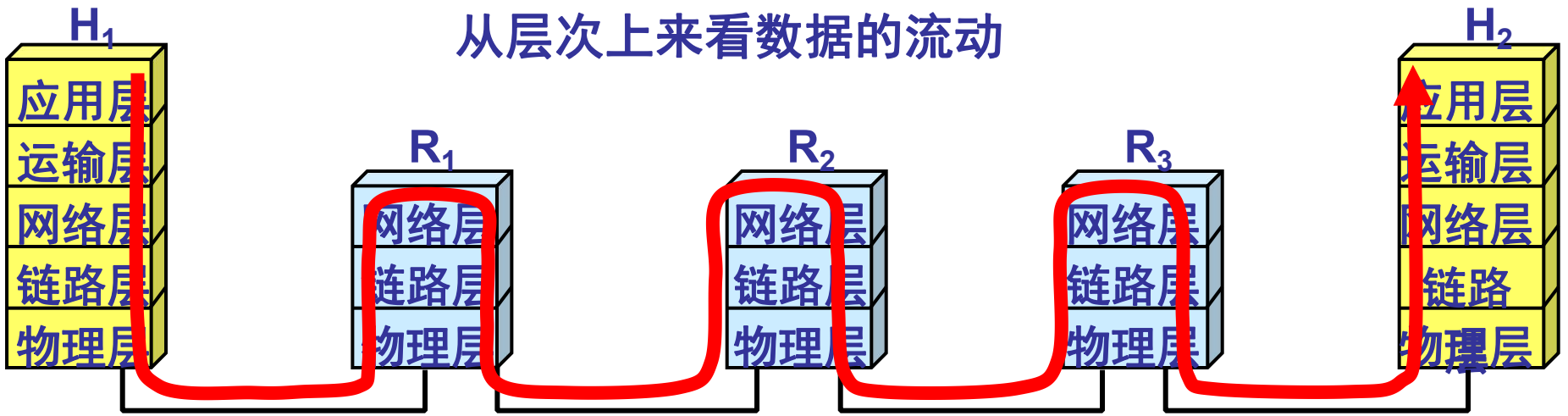
3.1 数据链路层的设计问题

- 虽然通信链路质量已经很好了，但还是会出错，且存在通信时延（非零延迟），会对数据传输效率有重要影响。需要设计协议解决这些问题。
- 数据链路层的功能
  - 为网络层提供一个良好的服务接口
  - 处理传输错误
  - 调节数据流，确保收发双方速度匹配
- 数据链路层从网络层获取数据包，封装成帧（frame），包括帧头，有效载荷和帧尾

3.1 数据链路层的设计问题

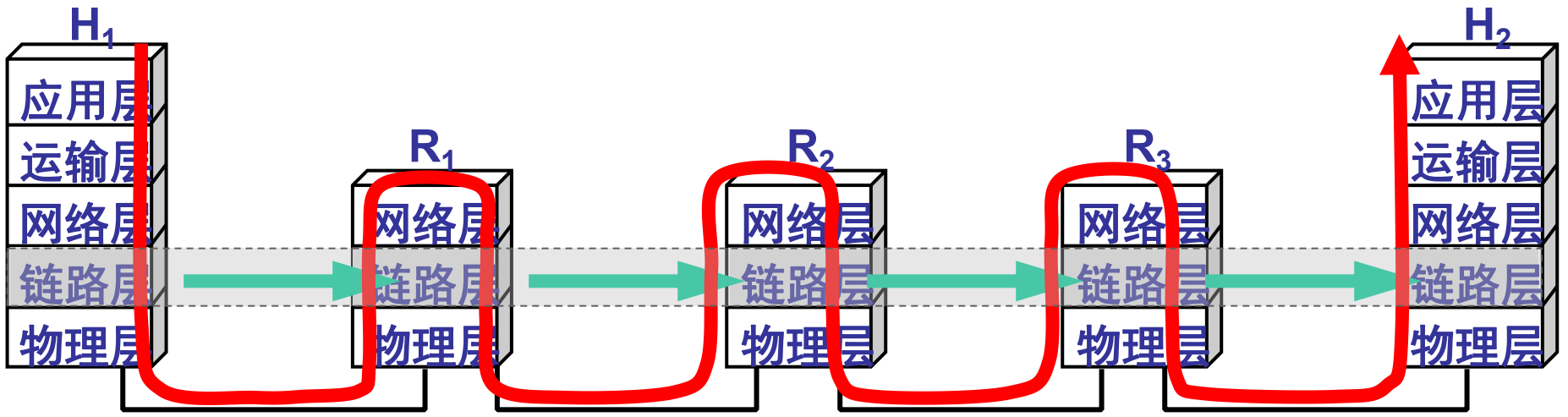
- 提供给网络层的服务





3.1 数据链路层的设计问题

仅从数据链路层观察帧的流动



3.1 数据链路层的设计问题

- 数据链路层提供给网络层的服务
  - 无确认无连接的服务
  - 有确认无连接的服务
  - 有确认有连接的服务
- 确认：接收方在收到数据帧后，必须给发送方发回一个确认
- 面向连接：发送方和接收方在传输数据之前必须建立一条数据链路，传输结束后必须释放该链路

3.1 数据链路层的设计问题

- 无确认无连接的服务
  - 无确认是指接收方在收到数据帧后，无需发回一个确认
  - 无连接的服务是指在数据传输前无需建立数据链路逻辑线路的连接。
  - 无确认并非不可靠，其可靠性可由上层负责

    例如：局域网

    共享信道不需要、也不允许建立连接

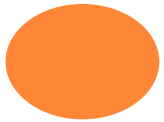
    信道较为理想，数据传输的误码率很低

    即使出错或丢失由上层负责恢复

3.1 数据链路层的设计问题

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

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

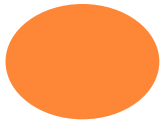


3.1 数据链路层的设计问题

- **有确认的有连接服务**
  - 使用前先建立连接，即先建立数据链路，并且每帧的传输必须得到确认
  - 有连接的服务必须在使用前先建立连接（即建立数据链路）消耗资源进行配置，然后使用连接，最后释放连接，并释放所有资源

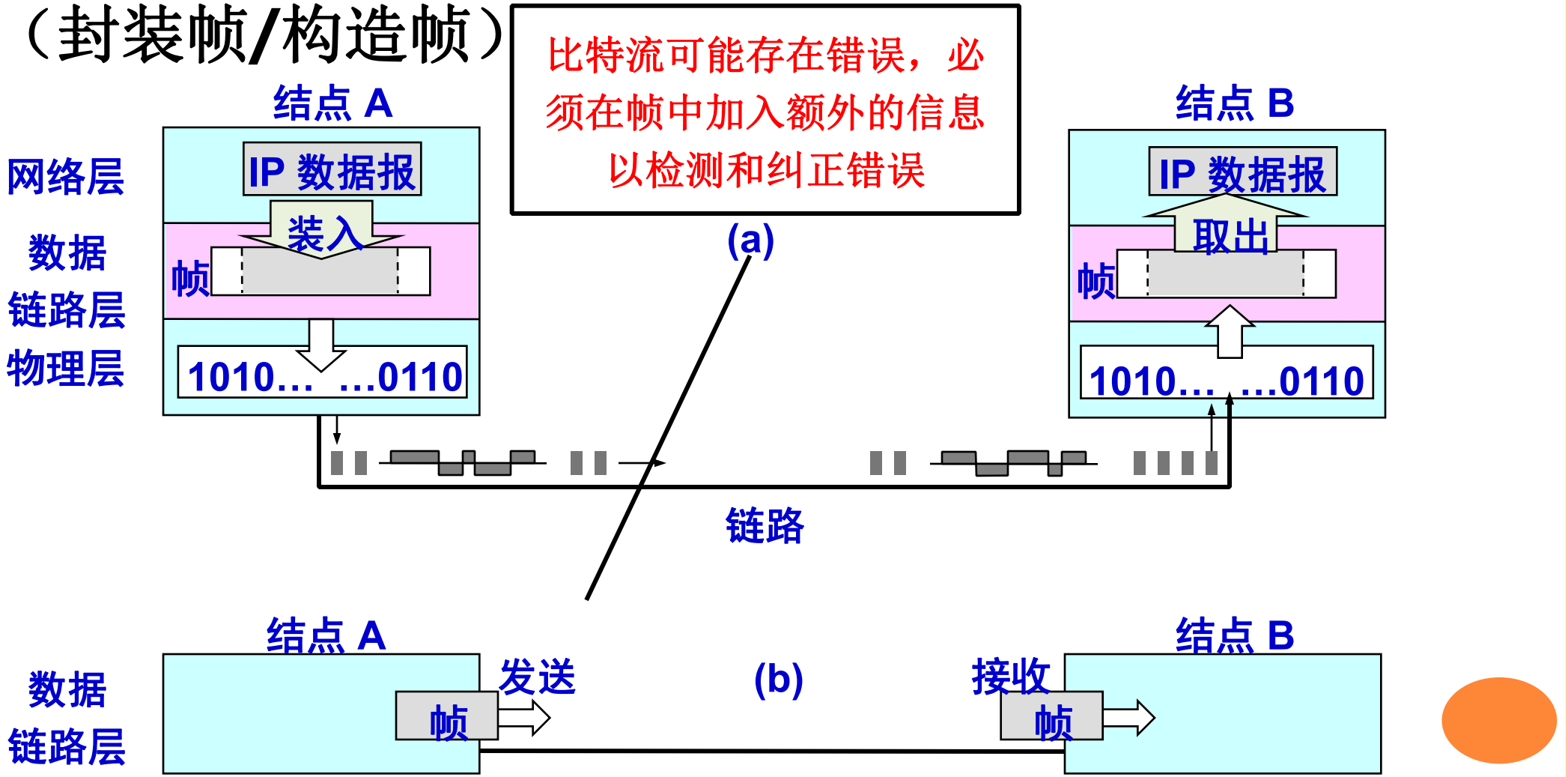
3.1 数据链路层的设计问题

- 成帧（封装数据帧）
- 差错控制
- 流量控制



3.1 数据链路层的设计问题

- 成帧（封装帧/构造帧）

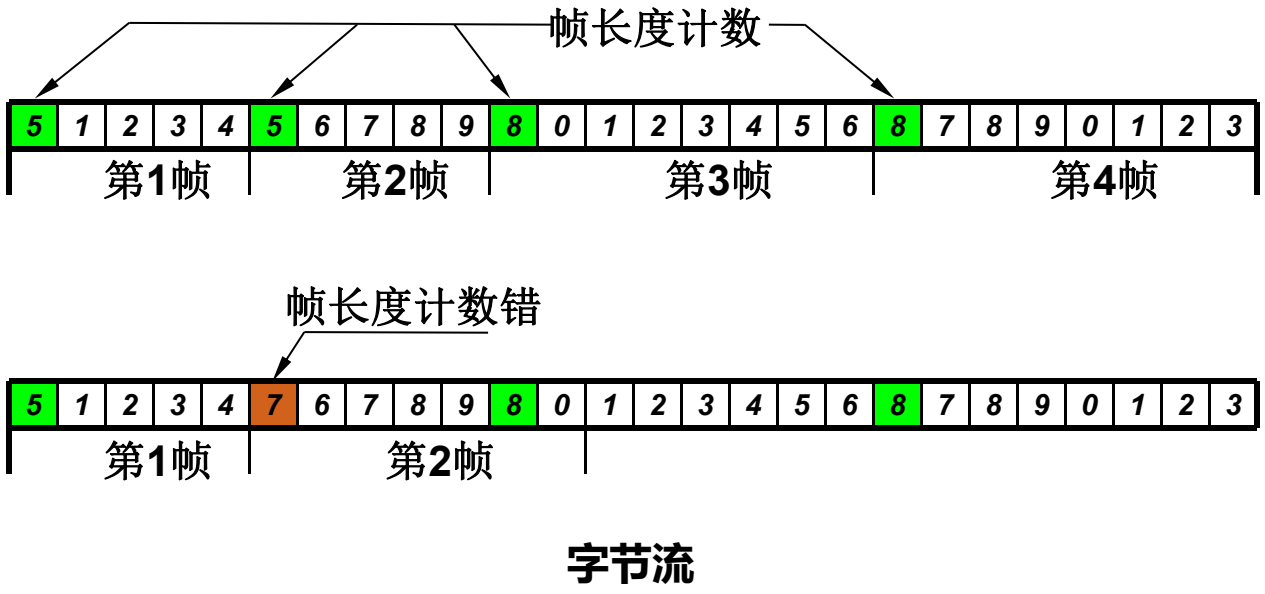


3.1 数据链路层的设计问题

- **成帧**
- 基本过程是将比特流拆分为多个离散的帧，为每个帧计算校验和，然后将校验和放在帧中一起传输，到达目标时重新计算校验和，并进行错误处理（丢弃或报告）
- 构造帧必须容易检测到帧开始，且尽可能少占用带宽
  - 字节计数法
  - 字节填充的标志字节法
  - 比特填充的标志比特法
  - 物理层编码违禁法

3.1 数据链路层的设计问题

- 成帧
- 字节计数法：假设帧的长度用一个字节表示，并作为帧的头部
- 缺点：一旦帧长度计数值被误读，将无法再同步



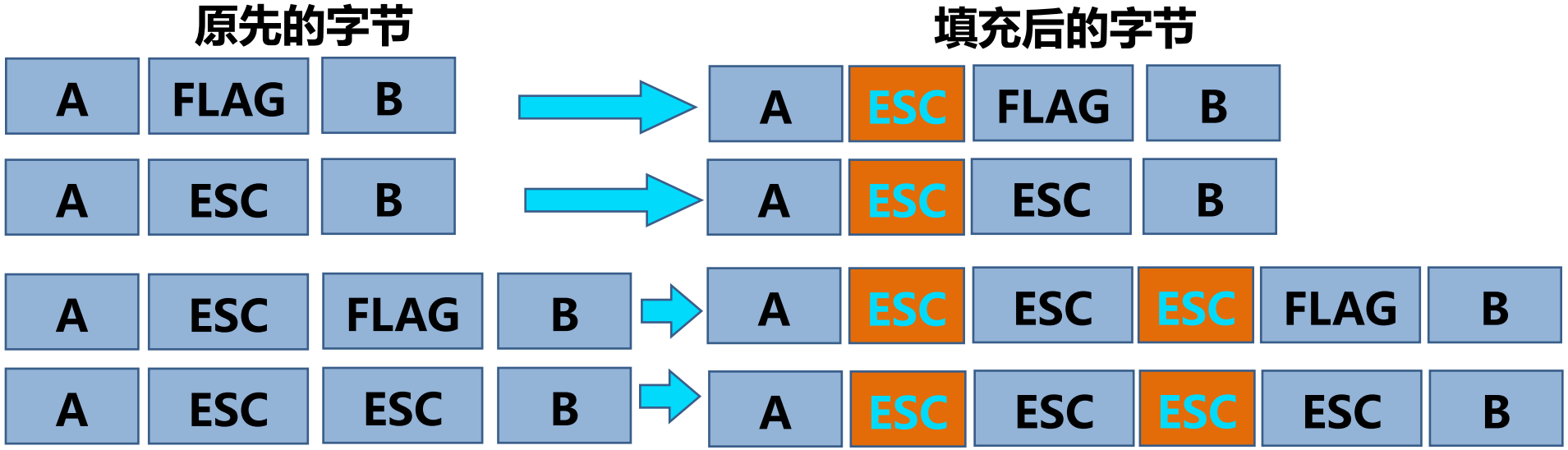
3.1 数据链路层的设计问题

- 成帧
- **字节填充的标志字节法**：用特殊的字节（标志字节）作为帧头和帧尾界符

- 传输数据中包含二进制数据时，很可能出现与FLAG相同的bit序列，导致帧的分界错误。

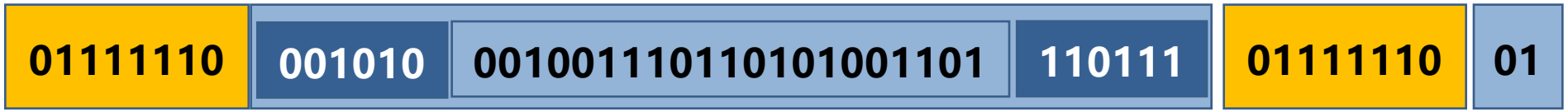
- 解决方法是在和FLAG相同的数据前再插入一个ESC （ASCII字符1BH），和实际的帧边界做区分，接收方再删除掉。这种技术称为“字节填充”(byte stuffing)

- 成帧
- 字节填充的标志字节法



3.1 数据链路层的设计问题

- 成帧
- **比特填充的标志比特法**：字节填充方法只能使用固定的8bit的字节，但帧里面的数据不仅仅是以8bit为最小单元的，比如汉字是16bit。因此考虑从比特级别上加入标志。
- 将01111110作为帧标志，即一个帧的开始（同时标志前一个帧的结束）



3.1 数据链路层的设计问题

- 成帧
- 比特填充的标志比特法：当发送方发现数据中存在一个与帧标志相同的位串01111110，则在连续5个1后自动插入一个0，即变成01111101。接收方将自动删除第5 个1后的0。

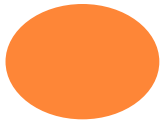
原数据	011011111 11111 11111 10010
填充后	0110111110111110111110010010
接收方	011011111 11111 11111 10010

3.1 数据链路层的设计问题

- 成帧
- **物理层编码违禁法：**由于比特编码中存在一些冗余比特，利用这些保留的信号来指示帧的开始和结束。
- 例如4B/5B编码中，从4位到5位的编码映射只用了16个，另16个（违禁的编码）可用于分帧。曼彻斯特编码的高-高，低-低电平等也可用于界定帧。

3.1 数据链路层的设计问题

- **成帧（封装数据帧）**
- **差错控制：**链路中可能存在错误包括数据帧本身错误，帧的丢失，确认帧的丢失，重复发送等
  - 对帧错误的处理：帧的校验
  - 对帧丢失的处理：超时和重发
  - 对帧重复的处理：帧的序号
- **流量控制：**避免一方速度太快，导致另一方无法接收，采用基于**反馈的流量控制**和**基于速率的流量控制**两种方式

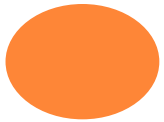


3.2 差错检测和纠正

- 差错的产生主要是在传输时，数据中的一位或几位因噪声干扰而出错，在无线信道或老化的线路中，出错是常态
- 两类利用冗余数据处理错误的方式
  - 检测到发生了错误，并能纠正错误-**纠错**(error-correcting code)(FEC,forward error correction)，用于质量较差的信道上
  - 只能检测到发生了错误，但不知错在哪里-**检错**(error-detection code)用于质量较好的信道如光纤上

3.2 差错检测和纠正

- **纠错码：**
  - 海明码（Hamming code）
  - 二进制卷积码(convolutional code)
  - 里德所罗门码(reed-Solomon code)
  - 低密度奇偶校验码(LDPC)
- 所有编码都将冗余信息加入到待发送信息中，n位帧信息=m个数据位+r个冗余码组成，描述为（n，m）码，n位单元称为码字(codeword)，m/n称为码率（code rate）



3.2 差错检测和纠正

- **海明码：**

给定两个码字 10001001 和 10110001，如果要分析两者之间多少位不同，通过异或XOR计算两个数据，得到1的个数即为不同的位数。两个码字中不同位的个数称为**海明距离**。

1 0 0 0 1 0 0 1

1 0 1 1 0 0 0 1

0 0 1 1 1 0 0 0

3.2 差错检测和纠正

- 海明码：方法是对数据位进行编码，通过计算校验位得出编码的具体数值，并填充到将2的幂次方的位上，其余用来填充数据。
- ✓ 利用异或计算最终值取0的方式得出校验码，如果为1表明有错误。
- ✓ 比如4个校验位，0000表示无错误，0001表示错误在第1位，0010表示第二位，…1011表示错误在11位

3.2 差错检测和纠正

- 海明码-确定校验位
- **校验位数r：$m+r+1 \leq 2^r$**
  - m 数据位，r 冗余码
  - 校验码的2^r种编码方式，可用于表示每一位可能的错误（包括自身在内），以及正确的状态。
- 确定校验位：2^k，2^{k+1}，2^{k+2}，2^{k+3}（k=0，1，2…r）

3.2 差错检测和纠正

- **海明码-计算校验位**
  - ✓ 将每个数据位的编号看成是2的幂之和，校验时检查对应的校验位即可。例如11=1+2+8，那么检查这几位可知道11位是否出错。
  - ✓ 反过来看校验位的计算，凡是数据编号对应位上校验码编号为1 的，都要参与该校验码计算

校验位	编号	二进制编码
x1	1	0001
x2	2	0010
x3	4	0100
x4	8	1000

3.2 差错检测和纠正

- 海明码-计算校验位
- 例如：为了求出x2,要使所有位置的第二位是1的数据（即形如× × 1 ×的位置的数据）的异或值为0。即x2XOR1XOR1XOR0XOR1XOR0 = 0。因此x2 = 1。同理可得x1 = 0, x3 = 1, x4 = 0。

编号	1	2	3	4	5	6	7	8	9	10	11
位置	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
内容	x1	x2	1	x3	0	1	0	x4	1	1	0

编号	1	2	3	4	5	6	7	8	9	10	11
位置	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
内容	0	1	1	1	0	1	0	0	1	1	0

3.2 差错检测和纠正

- **海明码-检错纠错**
- 计算数据位对应的校验码和自身的异或值是否为0，如为1则错误。
- 对最终数据的校验如下：将所有位置形如× × × 1, × × 1 ×, × 1 × ×, 1 × × ×的数据分别异或。

  × × × 1: 0xor1xor0xor0xor1xor1 = 1

  × × 1 × : 1xor1xor1xor0xor1xor1 = 1

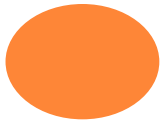
  × 1 × × : 1xor0xor1xor0 = 0

  1 × × × : 0xor1xor1xor1 = 1

只要不全为0，则表示有错误，1011表示错误在1011位（11位），纠错时只要取反即可

3.2 差错检测和纠正

- **检错码：**只检测，不纠错，发生错误就重传，适合高质量链路（光纤等）
  - 奇偶
  - 校验和
  - 循环冗余校验（CRC）



3.2 差错检测和纠正

- **检错码：**
  - **奇偶校验：** 计算比特流中1的数目，如果是偶数，奇校验法则在数据末尾添加1，偶校验法则在数据末尾添加0

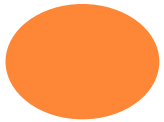
    例：原数据1011010

    偶校验1011010 0

    奇校验1011010 1

    如果出现1位错误，则校验码与接收方的计算结果不符。

    缺点是无法检测多位错误



3.2 差错检测和纠正

- **校验和（checksum）：**

✓ **算法简单、实现容易，但检错强度较弱**

✓ **将发送的数据看成是二进制整数序列，并划分成一段段规定的长度（如8位、16位、32位等），计算它们的和，如计算和时有进位，则将进位加到最后的校验和中，并将校验和与数据一起发送；在接收端，重新计算校验和，并与接收到的原校验和比较。**

例如：要传输“ Hello world.”

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

以16位为例：**4865H+6C6CH+6F20H+776FH+726CH+642EH+进位=71FCH**

3.2 差错检测和纠正

- **循环冗余检错码 CRC**
- 任何一个k位的帧都可看成为一个k-1次的多项式M(x)的系数列表

  如：1011001看成是多项式$x^6+x^4+x^3+x^0$的系数列表
- 预先设定一个生成多项式G(x)，G(x)为r阶，k>r，且最高/低位均为1
- 在M(x)低位加上r个0，然后除以G(x)，得到Q(x)为商、R(x)为余数，R(x)即为M(x)的CRC码，将CRC码接在帧后一起发送，即发送数据为$x^rM(x) + R(x)$
- 二进制运算中，减法和加法都做异或运算：0+1=1,1+1=0
- 因为($x^rM(x) - R(x)$)一定能被G(x)整除，即余数为0，则接收方只要计算CRC，并所得余数为0即为正确

CRC码计算举例

帧数据为11010011011

即：$M(x) = x^9+x^8+x^6+x^4+x^3+x+1$

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

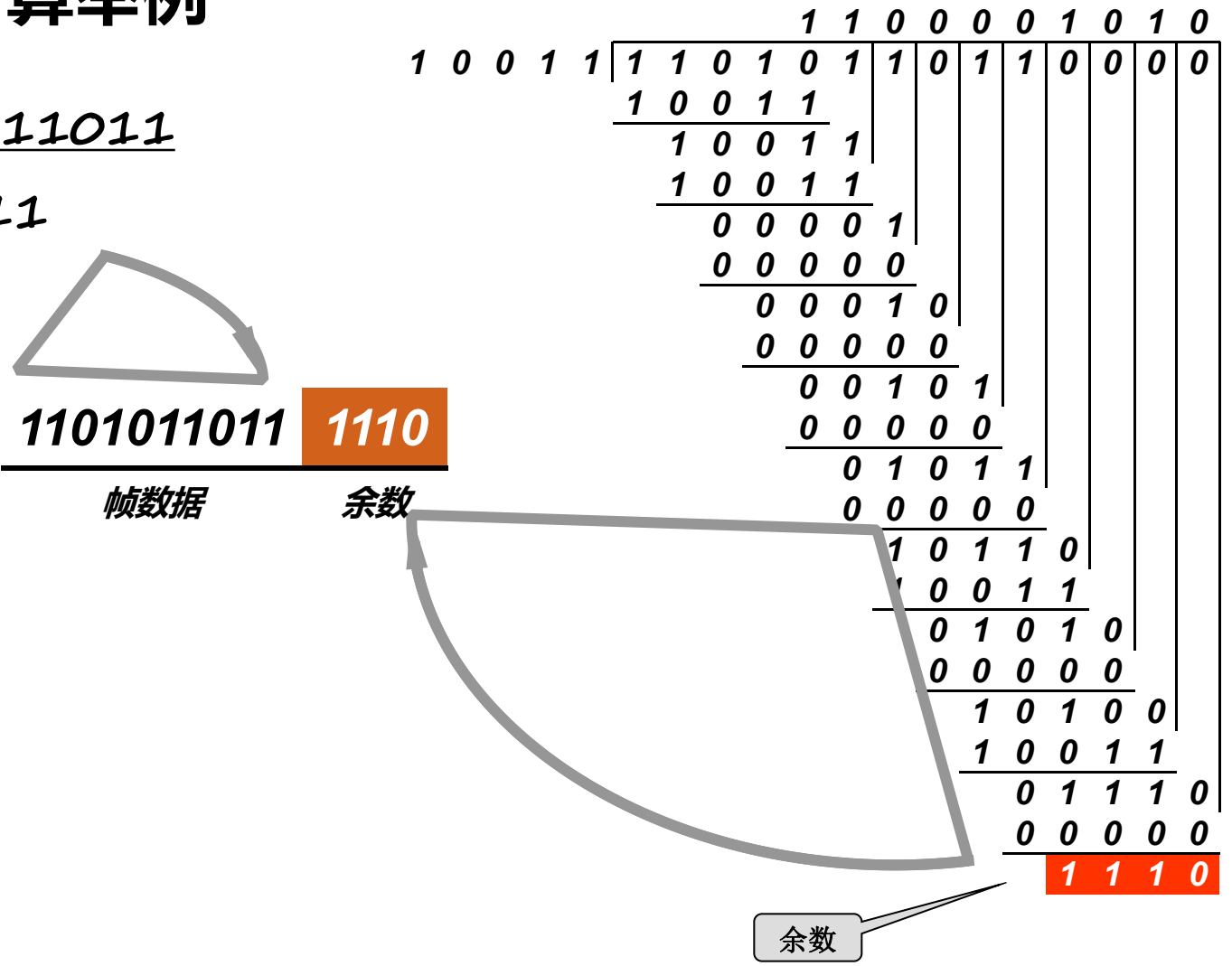
$T(x) = x^4M(x)$

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

$=x^{13}+x^{12}+x^{10}+x^8+x^7+x^5+x^4$

CRC码计算举例

- 帧：1101011011
- 除数：10011



- **实际传输帧：** 1101011011 1110

帧数据　　余数

3.2 差错检测和纠正

三个生成多项式国际标准

CRC-12：$X^{12}+X^{11}+X^3+X^2+X^1+1$　用于字符长度为6位

CRC-16：$X^{16}+X^{15}+X^2+1$　　用于字符长度为8位

CRC-CCITT：$X^{16}+X^{12}+X^5+1$　用于字符长度为8位

IEEE 802：$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X^1+1$

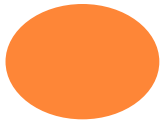
3.3 基本数据链路层协议

- 1、理想条件下的单工协议
- 2、无错信道上的单工停等协议
- 3、有错信道上的单工停等协议

3.3 基本数据链路层协议

通信过程中可能出现的问题：

- **数据帧错误**
- **数据帧丢失**
- **数据帧重复**
- **收发双方速度不协调**

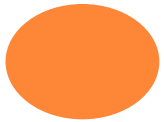


3.3 基本数据链路层协议

通信过程中可能出现的问题：

- **数据帧错误**
- **数据帧丢失**
- **数据帧重复**
- **收发双方速度不协调**

检错机制：校验码
纠错机制：纠错码
确认机制：向发送方发送一个特殊格式的帧，表明帧已收到/未收到。



3.3 基本数据链路层协议

通信过程中可能出现的问题：

- **数据帧错误**
- **数据帧丢失**
- **数据帧重复**
- **收发双方速度不协调**

通过发送方的重发定时器（超时）解决。
超时(TimeOut)：在传输过程中，如果所发送的帧丢失，接收方根本没有收到，无法发送确认帧，发送方也等待不到确认帧。为使通信正常继续，发送方每发送一帧，就启动一个重发定时器，在所设定的时间内，一般都应该收到确认，如收不到确认，则在重发定时器溢出后，再重发此帧

3.3 基本数据链路层协议

通信过程中可能出现的问题：

- **数据帧错误**
- **数据帧丢失**
- **数据帧重复**
- **收发双方速度不协调**

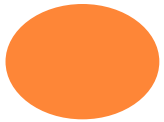
由于接收方确认帧的丢失，导致发送方多次发送同一帧，接收方也将多次收到同一帧，为能识别是否为相同的帧，应该在帧格式中增加一个帧的编号（序号）

3.3 基本数据链路层协议

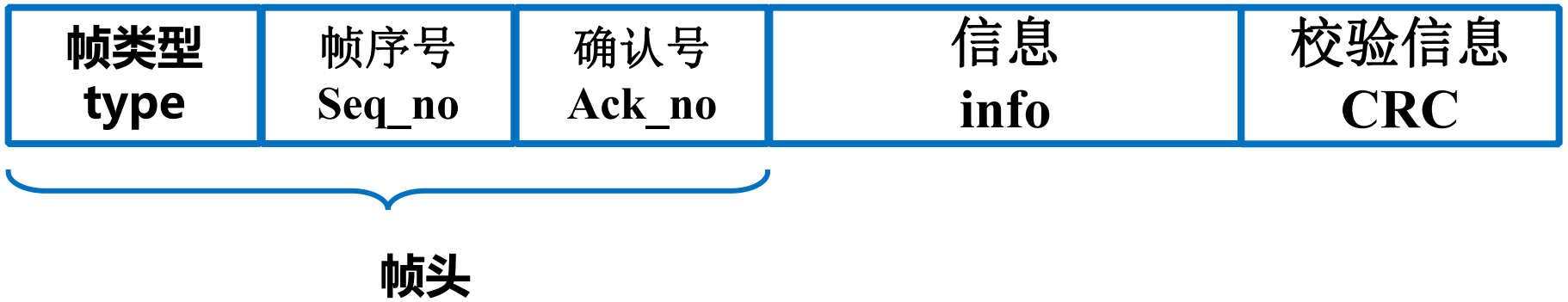
通信过程中可能出现的问题：

- **数据帧错误**
- **数据帧丢失**
- **数据帧重复**
- **收发双方速度不协调**

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



基于上述讨论，一个数据链路层的帧至少应该包括下列内容：



3.3 基本数据链路层协议

- 模型假设在一台主机中，物理层、数据链路层、网络层等，都有各自的进程在运行；
- 从简单到复杂，逐步设计实现可用的协议

3.3 基本数据链路层协议

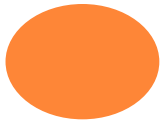
1、理想条件下的单工协议

- 链路是理想的传输通道，所传输的任何数据既不会损坏帧也不会丢失帧

  即：不需校验，也不可能出现重发，无需差错控制

- 不管发送方以怎样的速率发送数据，接收方都能及时接收并处理

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

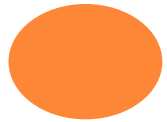


协议2：SENDER

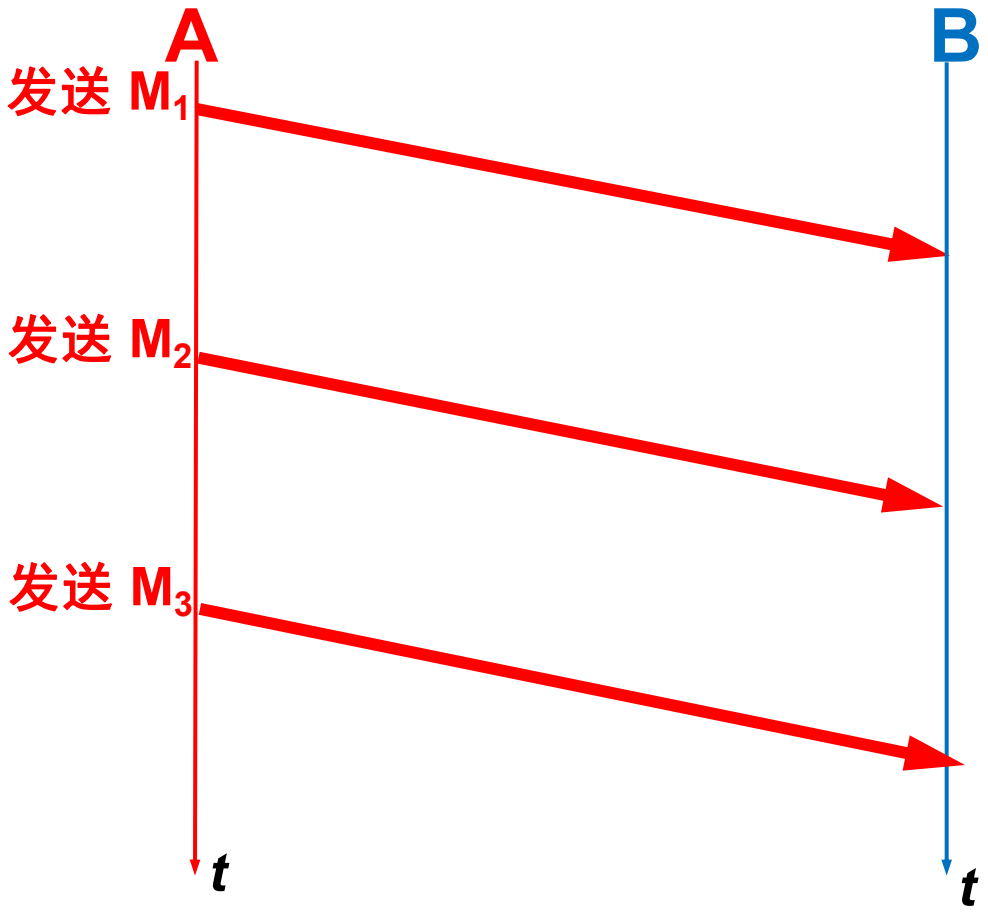
```
void sender1(void)
{
  frame s;
  packet buffer;

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

   }
}
```

协议1：RECEIVER

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

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



- 1、理想条件下的单工协议



2、无错信道上的单工停等协议

- **协议1中第一个假设保留，第二个假设撤消，假定：**
- 链路是理想的传输通道，所传输的任何数据既不会出错也不会丢失

- 考虑实际情况，接收方不可能具有足够高的CPU处理能力来及时处理所有的接收帧，也不可能具有永不溢出的缓冲区

  即：无需差错控制，但必须进行流量控制
- 这里的单工，其实是半双工，所谓流量控制是发送方必须收到前一帧的确认后才允许发送下一帧，接收方发出确认，意味着前一帧已接收并交网络层，准备接收下一帧

协议2:
SENDER

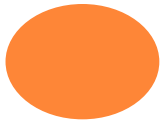
```
void sender2(void)
{  frame s;
   packet buffer;
   event_type event;

   while (true)
    {

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

协议2：RECEIVER

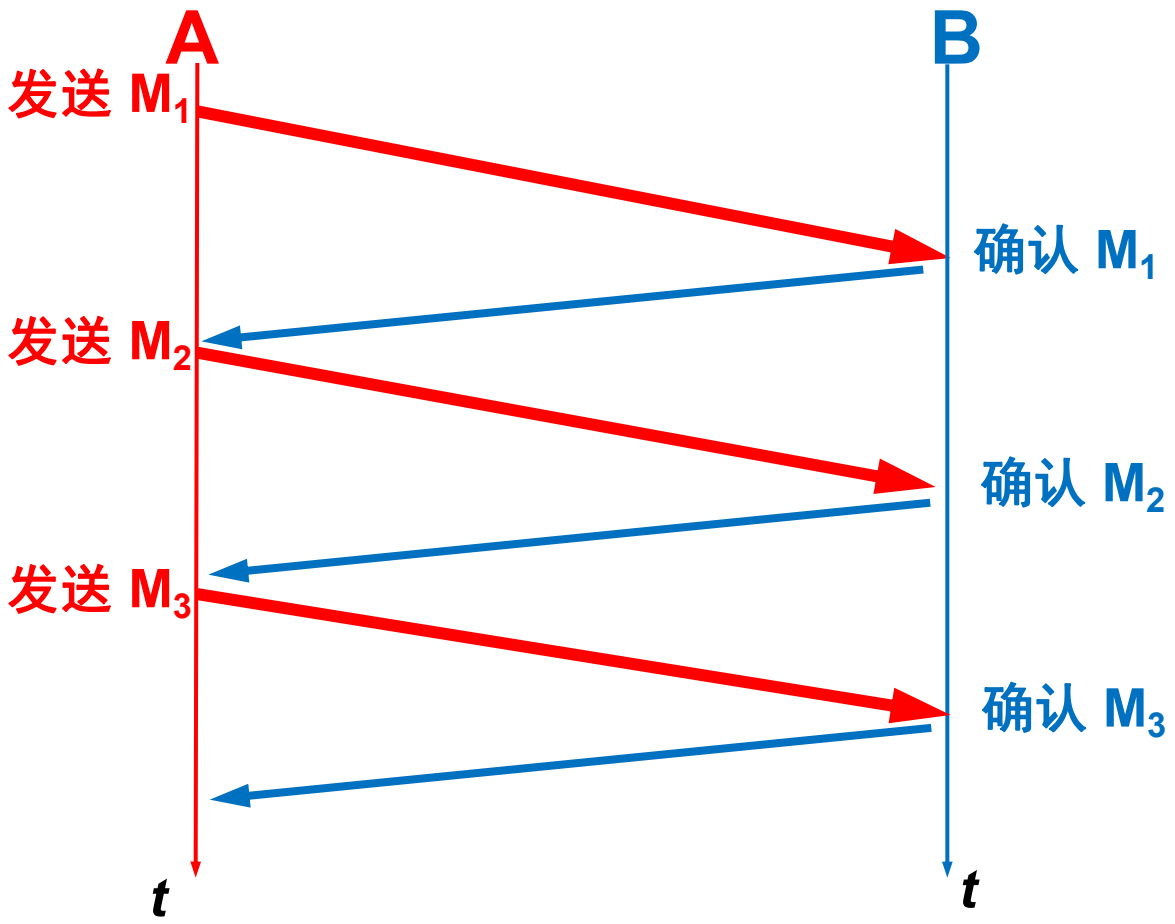
```
void receiver2(void)
{
  frame r,s;
  event_type event;

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

2、无错信道上的单工停等协议

- 协议2是一个半双工协议，即发送方和接收方使用同一信道，但发送方发送数据帧的过程和接收方发送确认帧的过程是严格交替的
- 协议2中，发送方发送的数据帧和接收方发送的确认帧采用相同的帧格式，发送方发送的数据帧中仅包含数据，接收方发送的确认帧数据为空，仅为一个信号
- 由于协议2定义的是一个理想的传输通道，所以不考虑传输差错问题



2、无错信道上的单工停等协议



3、有错信道上的单工停等协议

- 真实通信的信道存在诸多问题，或导致以下问题
- 1、帧错误；
- 2、帧丢失；
- 3、帧超时；

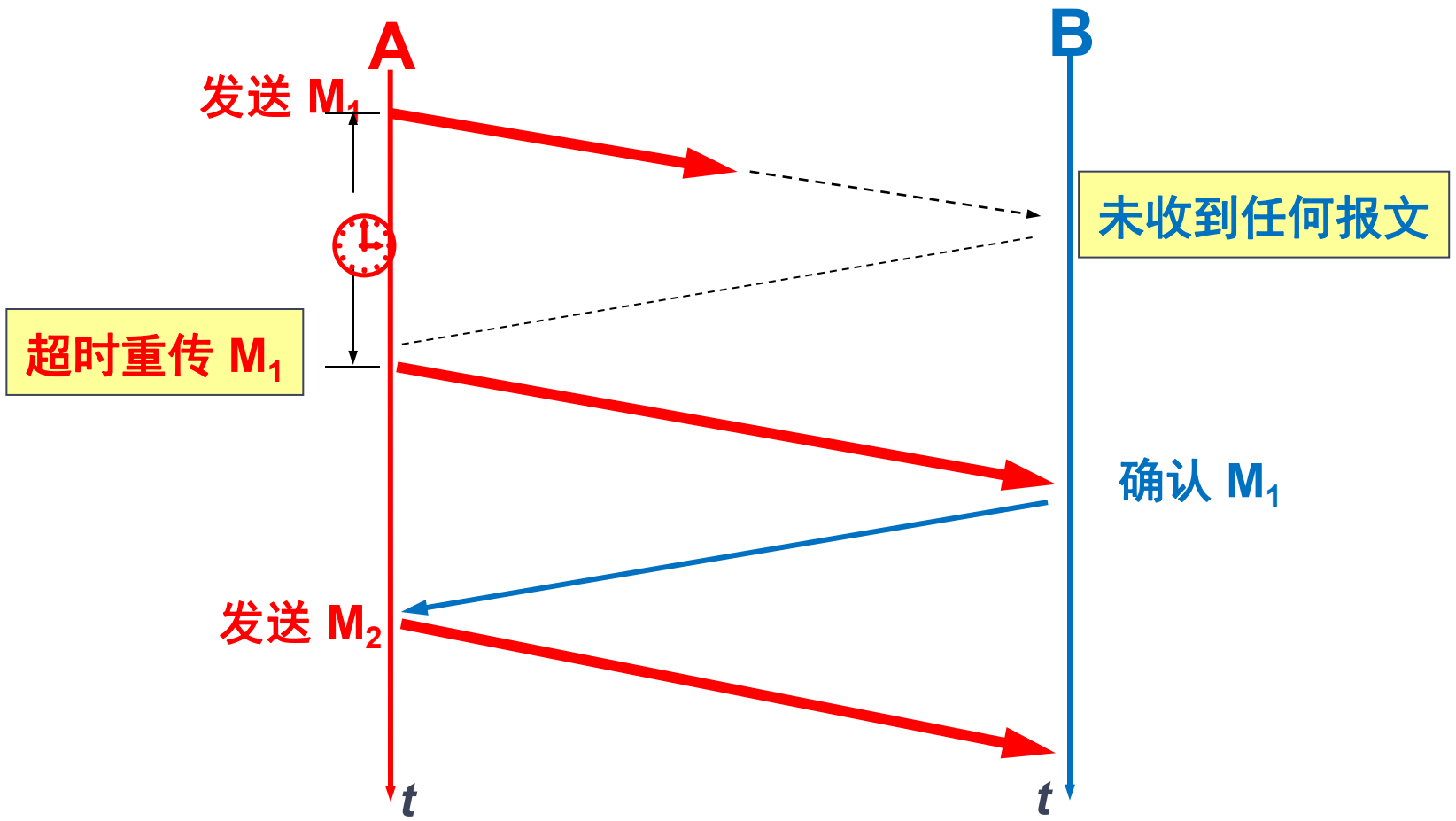
利用校验纠错机制，如果仍然出错怎么办？

如在协议2的基础上增加一个重发定时器，当发送方发送完一帧后启动一个重发定时器并等待确认

- 接收方收到一个错帧而不发ACK（无否定性确认机制）；
- 帧的丢失/超时，接收方不可能发送ACK；
- 接收方所发送的ACK丢失；

发送方的重发定时器都将超时，重发定时器一旦超时则立即重发

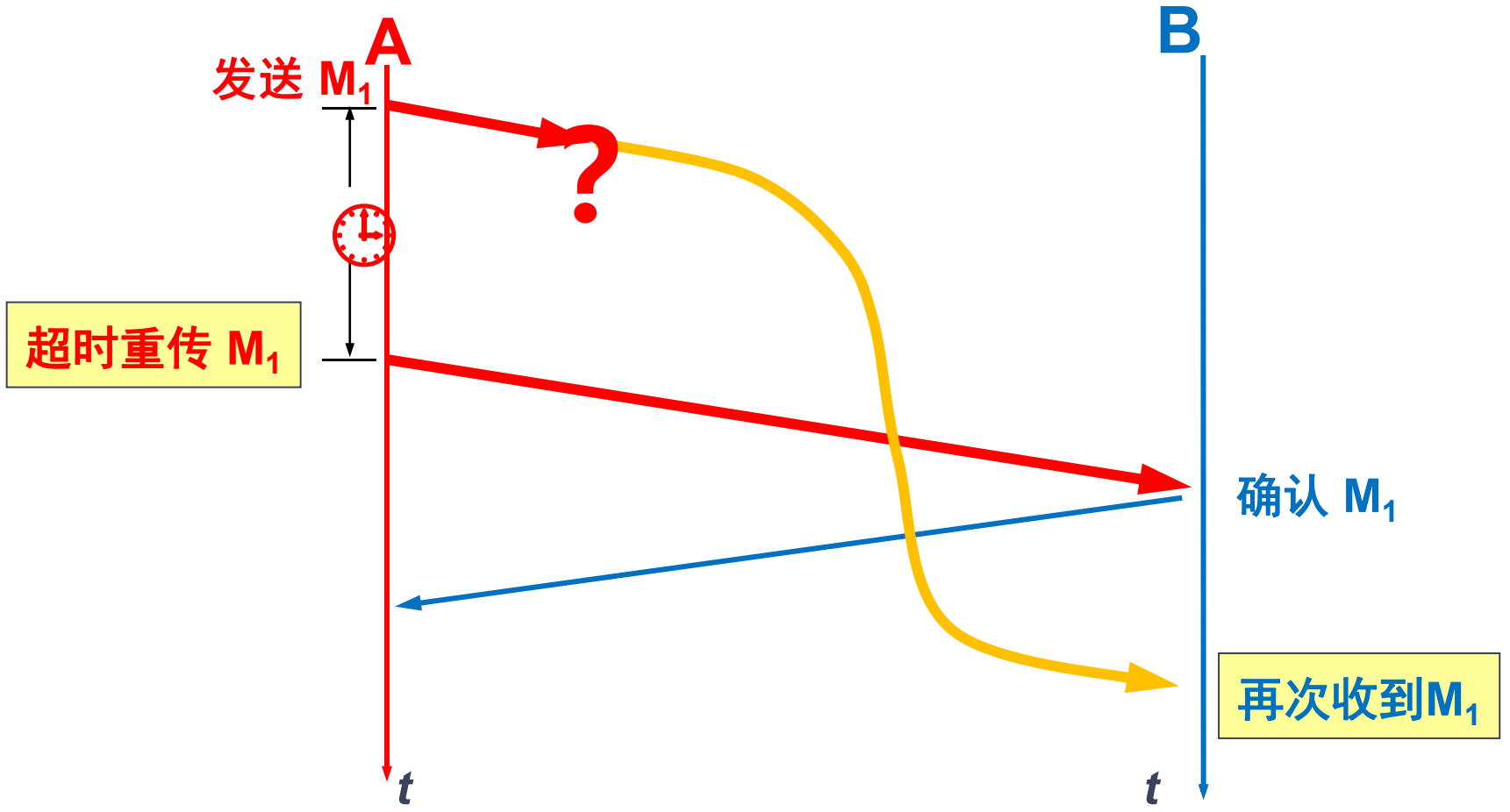
- 3、有错信道上的单工停等协议



3、有错信道上的单工停等协议

- 网络层只管接收链路层提交的数据，认为肯定是正确的。所以链路层必须考虑所有可能的错误，譬如：
- 由于通信线路的问题，发送方超时计数器*timeout*，重新发送帧，但之前发送的帧经过较长时间后也到达了接收方。此时两个重复的帧都是正确的，都会被上交到网络层，从而导致错误。

3、有错信道上的单工停等协议

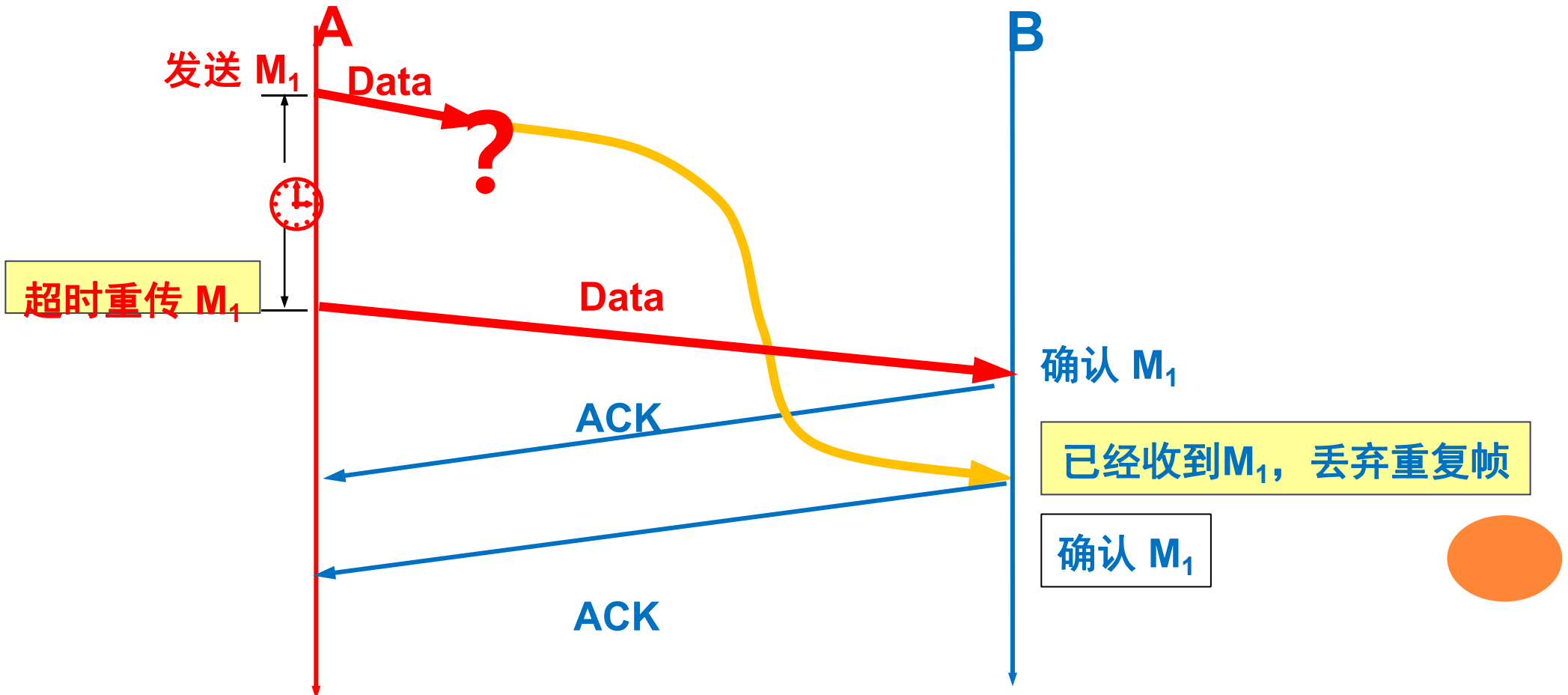


3.3 基本数据链路层协议

- 协议3的帧格式中定义一个帧序号字段，发送方要记录下一个准备发送的顺序号，接收方要记录下一个期待接收的顺序号
- 帧类型：有数据帧DATA 和确认帧ACK（肯定性确认）两种类型
  - 帧序号：在发送方的数据帧中是该帧的序号，在接收方的确认帧中是接收方期待接收的下一帧的序号
  - 协议3只定义了肯定性确认帧ACK，而没有定义否定性确认NAK
- 接收方收到一个正确（CRC正确）的帧，即便不是所期待的帧（即重复帧），都必须发送一个确认帧ACK
- 所谓单工协议，其实是半双工协议，发送和接收过程将严格交替
- **也称为ARQ协议：Automatic Repeat reQuest**

3、有错信道上的单工停等协议

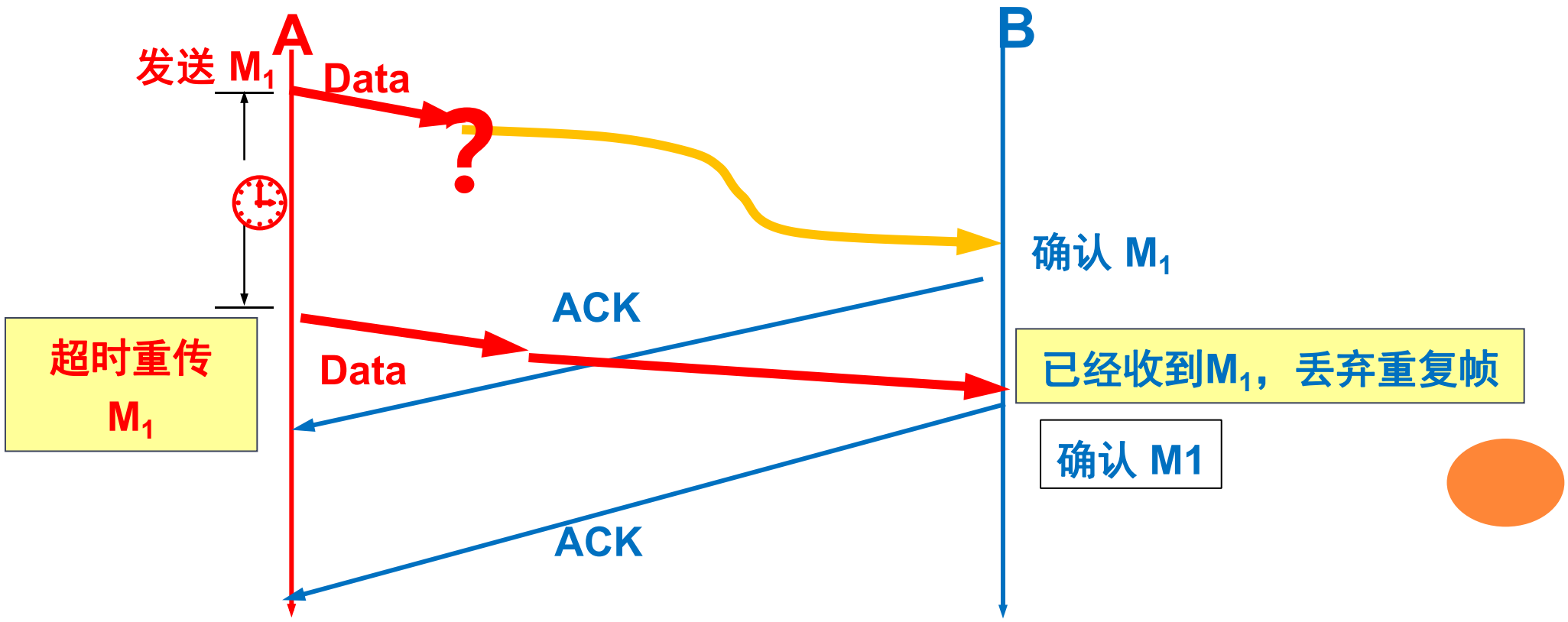
- 重复只能的问题解决方法：对所有帧（包括确认帧）编序号。



3、有错信道上的单工停等协议

- 重复帧的问题解决方法：对所有帧（包括确认帧）编序号。

  （重发未到达之前收到之前的M1）



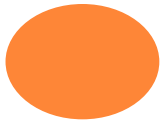
3.3 基本数据链路层协议

协议3: SENDER

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

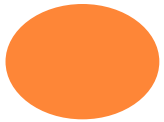
协议3: RECEIVER

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

           to_physical_layer(&s);
        }
    }
}
```



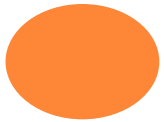
3、有错信道上的单工停等协议

- **发送方**：发送一帧后，启动计时器，并等待。如果：
  - 确认帧正常收到，发送下一帧，并递增序号
  - 确认帧损坏/收不到或计时器超时，重发帧
- **接收方**：接收到一个帧之后，如果：
  - 如果是正常帧，则交给网络层，并发送确认帧，递增序号
  - 是之前收到的帧（序号重复）或是错误的帧（校验不通过），丢弃，重复发送之前的确认帧，并等待下一轮传输。



之前协议存在的问题

- 由于数据是单向传输的，数据传输和应答不会同时传输，因此可使用半双工信道（通常半双工仅需单信道），但不能实现同时双向传输。也可以用两根信道，但传输应答的信道效率较低
- 如发送方的重发定时器设定的初始值较小，可能出现系统死锁

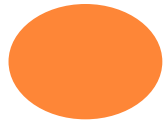


3.4 滑动窗口协议

双向传输解决方案

- 但在实际情况下，一般需要的是双向传输
- 双向传输的解决方案：
  - 用四条信道：两条数据，两条应答

    但信道利用率很低
  - 用两条信道：一条A到B，另一条B到A
    - ✓ **用不同的帧类型标志区分数据帧和确认帧**
    - ✓ **采用捎带确认（*piggybacking*）进一步提高信道效率**



3.4 滑动窗口协议

流量控制

- 发送速率和接收速率的匹配即流量控制

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

  - 基本数据链路协议
  - 滑动窗口协议

3.4 滑动窗口协议

- 捎带确认问题是数据帧和确认帧的发送时机不同步，可能想发送确认的时候，并没有数据帧可用于捎带。
- 一个捎带确认不一定只确认一个帧，而能确认多个帧
- **例如，A连续发送0、1、2、3、4号帧给B，而B一直没有数据要发送。当收到4以后，B要发送数据，此时捎带确认4，表示0到4号帧都收到了**

3.4 滑动窗口协议

双向传输总结

- 收、发使用两条信道

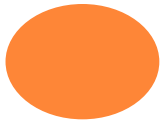
  发送方可连续发送多帧，接收方接收到一帧后就从另一个信道发回一个ACK，为提高信道使用效率，接收方可使用捎带确认

- 帧是有序号的

  即使过早超时而导致的重发也可根据帧的序号来避免帧的重复

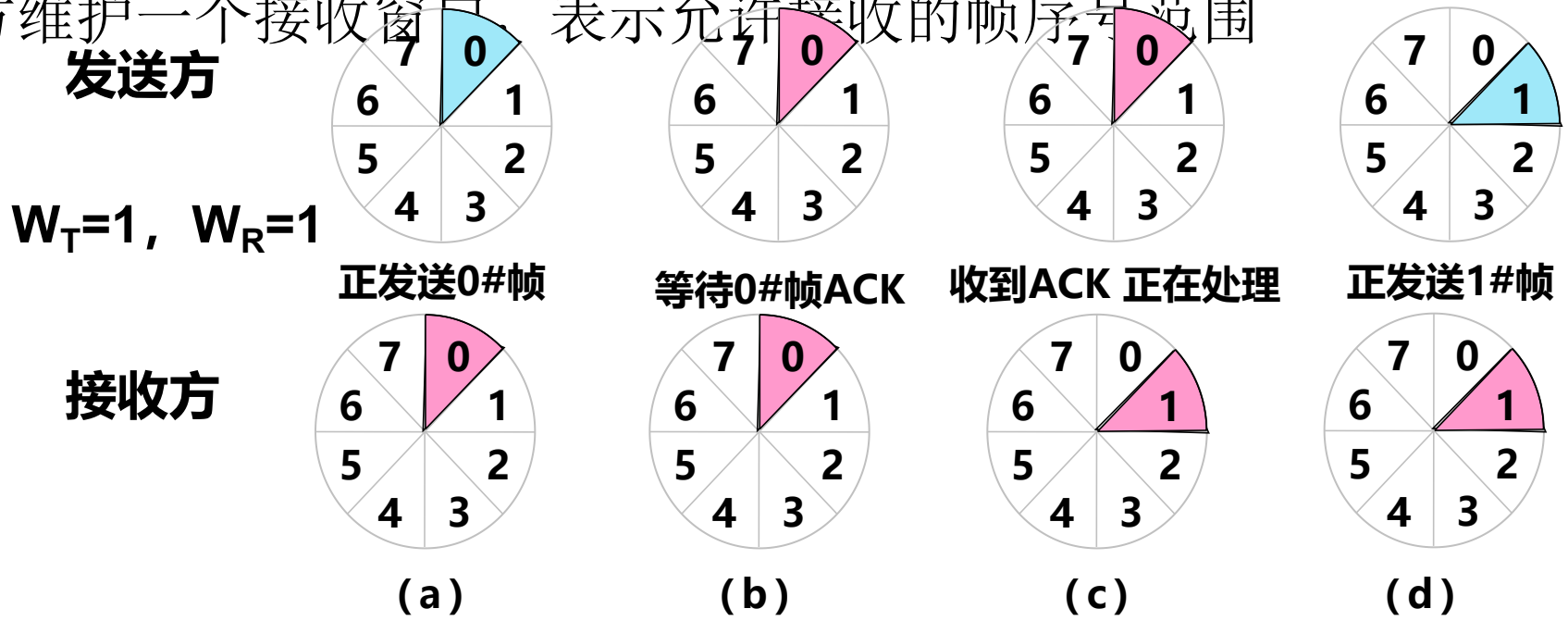
3.4 滑动窗口协议

- 滑动窗口是控制流量的一种方法
- 发送方维护一个发送窗口，包含允许发送的帧序号范围
- 接收方维护一个接收窗口，表示允许接受的帧序号范围



3.4 滑动窗口协议

- 使用滑动窗口协议进行控制流量，确保双方发送能够同步
- 发送方维护一个发送窗口，包含允许发送的帧序号范围
- 接收方维护一个接收窗口，表示允许接收的帧序号范围



动画演示

3.4 滑动窗口协议

三个滑动窗口协议

- **1位滑动窗口协议** (协议4)
  - 窗口$W_T=1$，接收窗口$W_R=1$
- **后退n帧的滑动窗口协议** (协议5)
  - 发送窗口$W_T=2^n-1$，接收窗口$W_R=1$
- **选择重传协议** (协议6)
  - 发送窗口$W_T=2^{n-1}$，接收窗口$W_R=2^{n-1}$

3.4 滑动窗口协议

1位滑动窗口协议

- A和B之间的通信是双向的，A和B都正运行一个滑动窗口协议，其中包含了发送和接收的功能。帧序号仅用1bit表示，A和B的发送窗口W_T=1、接收窗口W_R=1，窗口大小即缓冲区的个数。
- 发送方发送下一个数据帧必须在收到接收方的确认之后
- 接收方采用捎带确认，并且接收到帧后，必须对已接收到的帧发送确认（有可能是重复确认）
- 假设B收到A0，则回应确认帧号0，如果收到下一帧为A1，确认帧是1。但如果重复收到A0，则确认帧号仍旧为0，直至收到A为止

3.4 滑动窗口协议

1位滑动窗口协议

- 如果A超时间设置过短，协议能否正常运行？
- A发送一帧A0之后，在接收B的确认之前将会触发超时计时器，不断重发同一帧A0。
- 而B只会接受第一个A0，其他接收到的重复的帧会被B丢弃，并对之前收到的同一帧发送确认，直至A收到确认发送下一帧。
- 在捎带确认机制下，B发送的帧可能是

1（A0的确认，B0帧） 2（A0的确认，B1帧）、

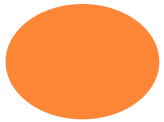
3（A0的确认，B2帧） …

3.4 滑动窗口协议

1位滑动窗口协议

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

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



3.4 滑动窗口协议

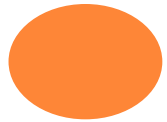
分析两种情况

- 情况1：

  A端首先发送，B端等待发送，无过早超时，其运行过程正常

- 情况2：

  A端和B端同时发送，无过早超时，其运行过程异常



3.4 滑动窗口协议

情况1：主机A发送,主机B等待发送

主机A

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

主机B

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

情况1：运行过程正常 （0-序号，1-确认号，A0-包）

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

协议4，运行过程正常

情况1：运行过程正常 （0-序号，1-确认号，A0-包）

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

协议4，运行过程正常

情况1：运行过程正常 （0-序号，1-确认号，A0-包）

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

协议4，运行过程正常

情况1：运行过程正常 （0-序号，1-确认号，A0-包）

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

协议4，运行过程正常

3.4 滑动窗口协议

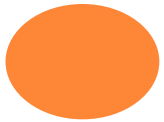
情况2：主机A、主机B同时发送

主机A

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

主机B

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



情况2：运行异常 （0-序号，1-确认号，A0-包）

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

协议4，运行过程不正常

情况2：运行异常 （0-序号，1-确认号，A0-包）

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

协议4，运行过程不正常

情况2：运行异常 （0-序号，1-确认号，A0-包）

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

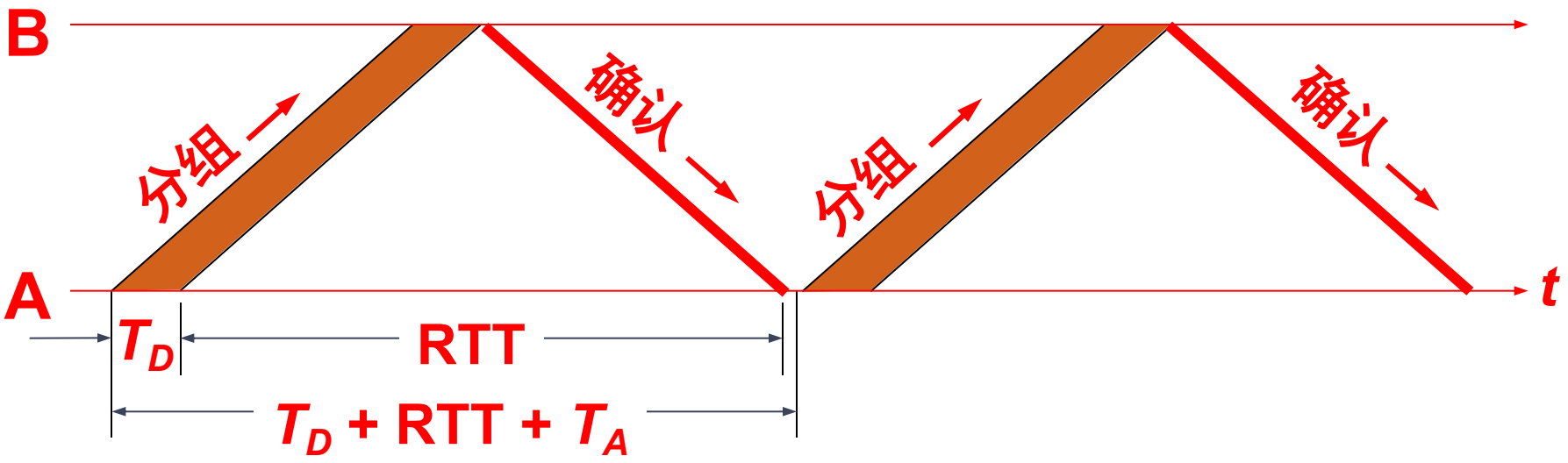
协议4，运行过程不正常

3.4 滑动窗口协议--回退N协议

- 协议4的主要问题：信道利用率太低

  例：在卫星信道上，往返时延500ms，发送时延20ms，从发出一帧到接收到确认，需要等待520ms，信道利用率为20/520=4%，被阻塞了500ms。

  信道利用率 $U = \frac{T_D}{T_D + RTT + T_A}$



3.4 滑动窗口协议--回退N协议

- 解决方法是在等待确认帧的时间内连续发送w个数据帧
- 利用带宽-延迟乘积计算最大的w值

W=2BD+1 B是带宽，D是单向传输时延

例：带宽50kbps，250ms传输时延

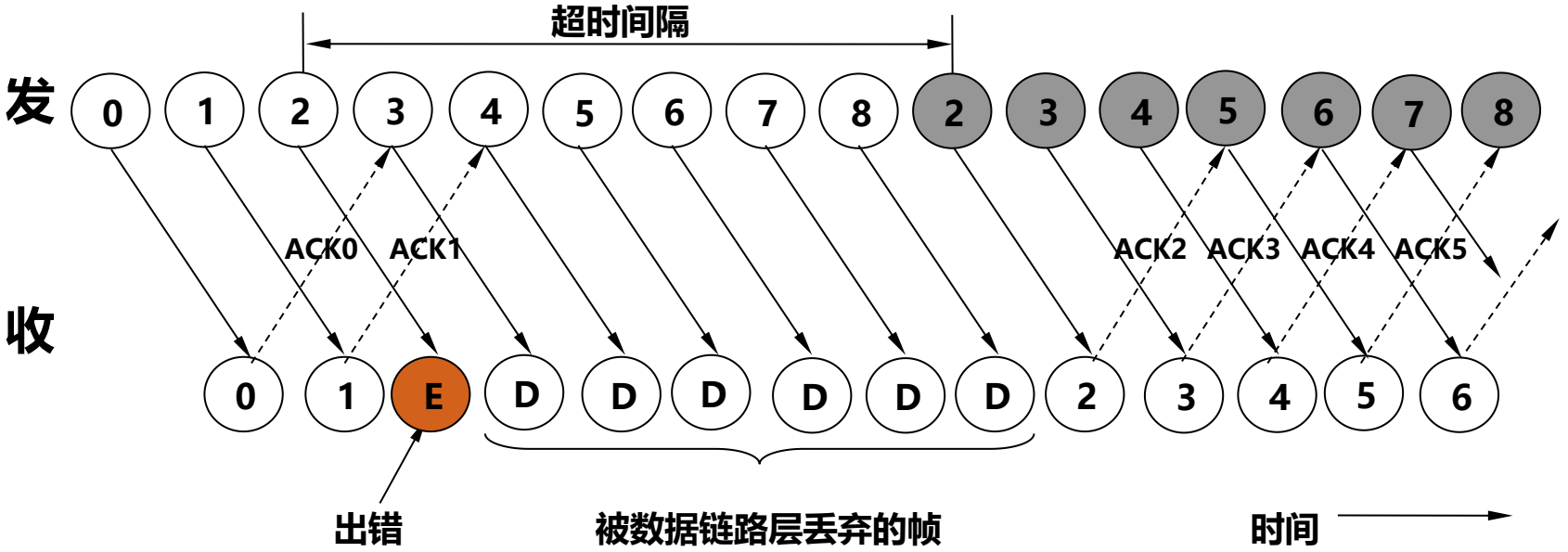
W=(2*50k*0. 25)+1帧=25k+1帧

如果1帧为1000bit（1k），则W=26帧

所以链路最大利用率 = $\frac{W}{2BD+1}$

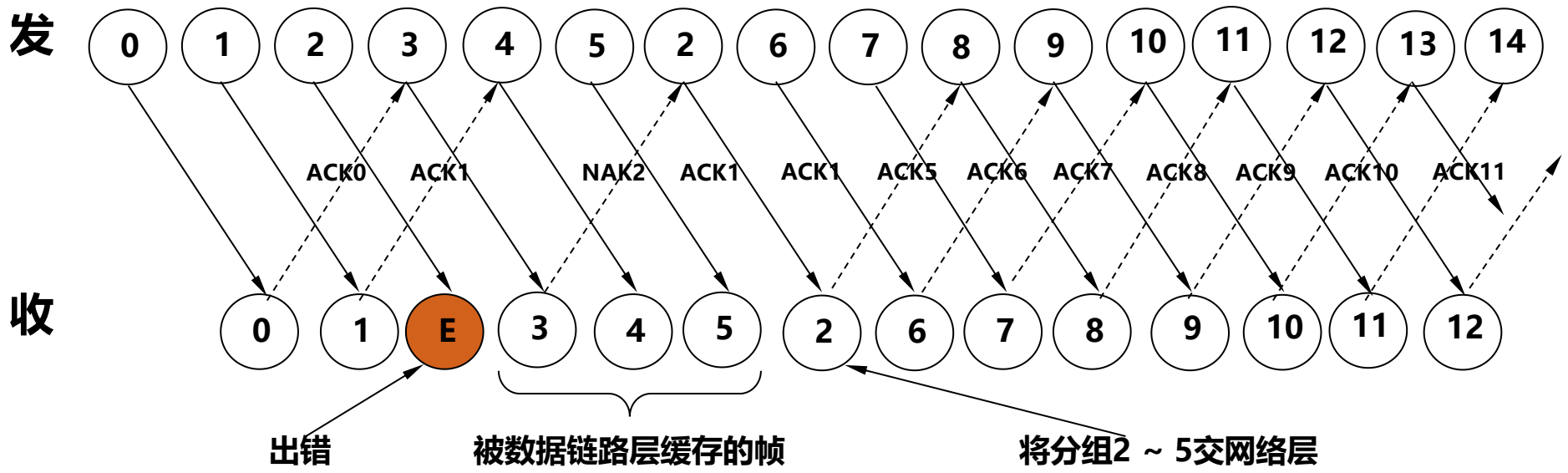
3.4 滑动窗口协议--回退N协议

- 协议5是一个发送管道化（pipelining）的协议，连续发送帧时，如果其中某一帧发生错误，该怎样处理后续正确的帧？此时接收帧的顺序已经不再按照原顺序。
- **回退n帧（后退n帧协议）**：发生错误后，接收方丢弃所有后续到达的帧，且不确认。发送方会在超时计时器*timeout*后，发送所有超时的帧。这种方式不适用于错误率高的信道。



3.4 滑动窗口协议--回退N协议

- **选择重传协议：接受并缓存错误帧后面所有正确的帧，发送方只超时重发错误的那一帧。通常需要使用否定确认（NAK），可以在超时之前就重发。不过需要考虑缓存空间的大小。**



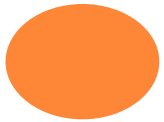
3.4 滑动窗口协议--回退N协议

（1）发送窗口的大小不能等同于序号空间的大小。

例如：序号0-7，发送窗口为7。第一次发送0-7共8个帧，接收方发送对7的确认ACK7。发送方再次发送一批新的0-7编号帧。

- **如果这批帧全部收到，接收方发送ACK7（对新的7号帧确认）**
- **如果这批帧全部丢失，接收方发送ACK7（对旧的7号帧确认）**
- **所以设置发送窗口小于序号空间，这样相邻两次确认的序号肯定不同。**

（2）回退N协议在发送方也需要缓存尚未被确认的帧



3.4 滑动窗口协议--选择重传协议

- 通过接收方的缓存机制，可不按照顺序接受帧，但也带来了新问题

- 相邻两批新旧序号范围重叠造成的问题

3.4 滑动窗口协议--选择重传协议

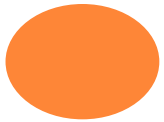
- 采用较小的窗口，避免序号重叠

3.4 滑动窗口协议--选择重传协议

- 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



3.5 数据链路层协议实例

- PPP协议
- 对称数字用户线

3.5 数据链路层协议实例

PPP协议组成

- 数据链路层早期使用的是HDLC协议，可实现可靠传输，但较复杂。
- 现在广泛使用的是更简单的点对点协议PPP（Point-to-Point Protocol）（RFC1661 1662 1663）。
- 包括以下三个部分：
  - （1）封装的方法：将IP数据封装到串行链路
  - （2）链路控制协议LCP：用于协商通信过程启动、测试、协商、关闭线路
  - （3）网络控制协议NCP，用于支持不同网络层协议

3.5 数据链路层协议实例

PPP的帧格式

- PPP的帧格式类似于HDLC，但是面向字符的协议（以字节为单位）

14	1	1	1/2	可变	2/4	1
标志 01111110	地址 11111111	控制 00000011	协议	有效载荷	校验和	标志 01111110

标志域：固定为01111110，与HDLC类同

地址域：固定为11111111

控制域：缺省为00000011，即无序号帧

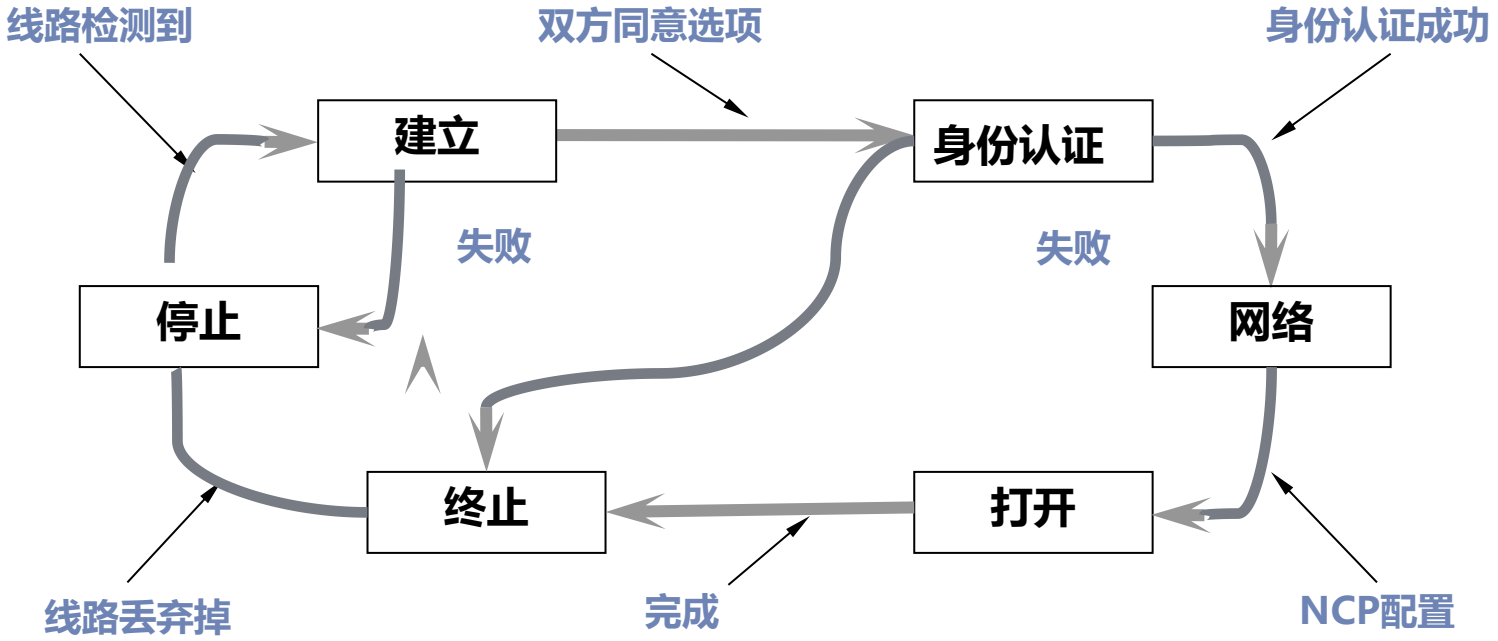
协议域：不同的协议不同的代码

载荷域：可变长，缺省最长1500字节

校验和：缺省为2字节，也可定义为4字节，仅是头部的校验和

3.5 数据链路层协议实例

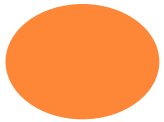
一次使用PPP协议的状态图



PPP链路建立到释放的状态转换图

小结

- 数据链路层的设计：三种不同质量的服务
- 成帧：帧定界方法（字节计数、字节填充、比特填充、编码违禁）
- 差错控制：纠错（海明、卷积、所罗门、奇偶）、检错（奇偶、校验和、*CRC*）
- 流量控制：停等协议，滑动窗口协议
- 实例介绍



第三章作业

3，6，20，27

16，22，34