第三章 数据链路层

刘 轶
北京航空航天大学 计算机学院

本章内容

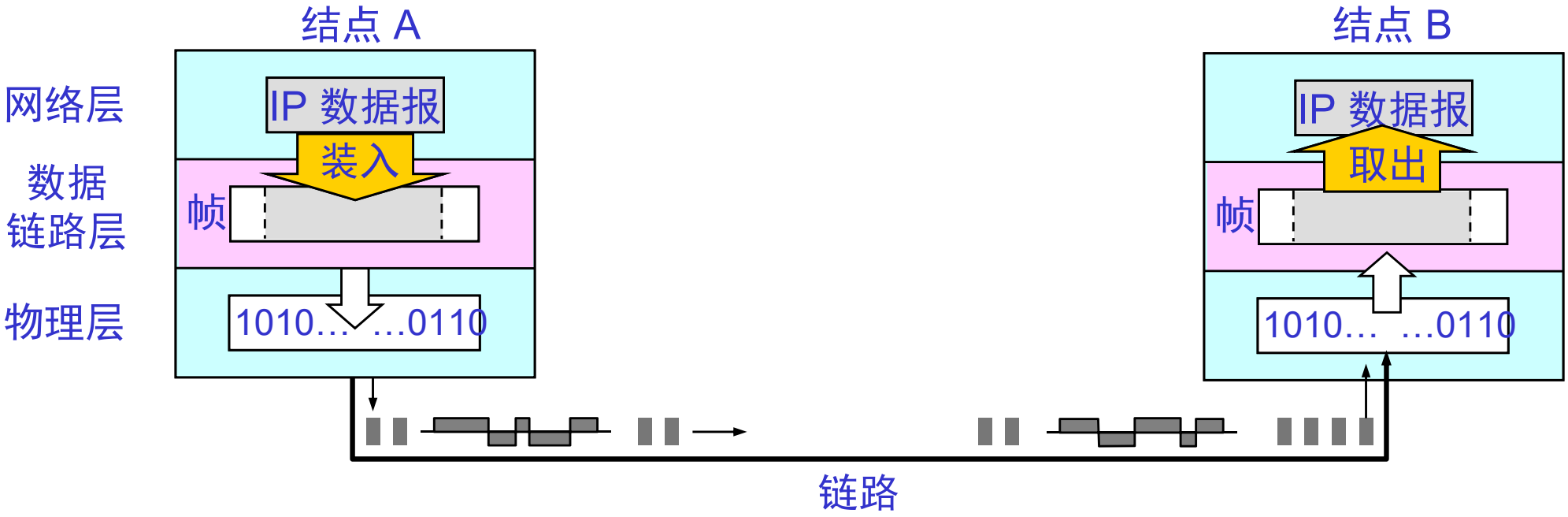
3.1 数据链路层设计要点
3.2 错误检测和纠正
3.3 基本数据链路协议
3.4 滑动窗口协议
3.5 点对点协议 PPP
3.6 介质访问控制
3.7 以太网
3.8 局域网互连

3.1 数据链路层设计要点

3.1 数据链路层设计要点

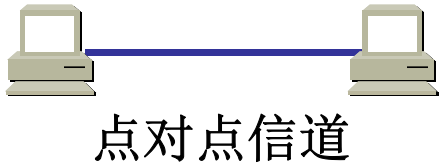
一、数据链路层概述(1/2)

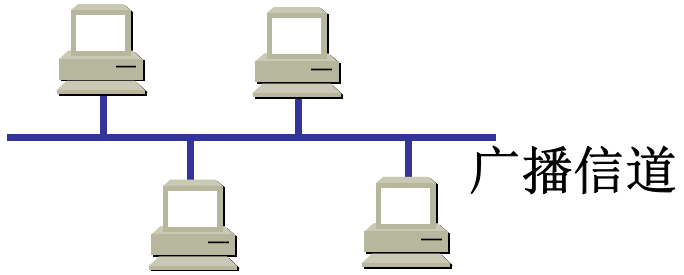
- 物理层实现了比特流的传输，数据链路层在其基础上实现帧(frame)的传输
  - 数据链路层传输的协议数据单元(PDU)是帧



3.1 数据链路层设计要点

一、数据链路层概述(2/2)





- 数据链路层使用的信道类型
  - 点对点信道
    - 使用一对一的点对点通信方式。
  - 广播信道
    - 使用一对多的广播通信方式
    - 广播信道上连接多个主机，必须采用专门的共享信道协议来协调数据发送
- 数据链路层涉及的问题
  - ① 成帧(**framing**)：怎样组成帧、怎样使接收方识别帧
  - ② 差错控制：帧在传输过程中出错的检测
  - ③ 流量控制及可靠传输：仅是数据链路层的选项
  - ④ 广播信道中的介质访问控制

3.1 数据链路层设计要点

二、成帧方法

- 成帧要考虑的问题：接收方如何识别帧的边界？
- 常用的成帧方法

(1) 字符计数法

在帧头部字段中指明本帧的字节数，接收方通过该字段得知该接收多少字节

(2) 字符填充的首尾定界法

定义专门的字符作为帧的起始/结束标志，并使用字符填充方式将标志字符与数据区分开来

(3) 比特填充的首尾定界法

定义专门的比特序列作为帧的起始/结束标志，并使用比特填充方式将标志序列与数据区分开来

本章PPP协议中可看到实例

(4) 物理编码违例法

使用无效的物理编码作为帧的开始/结束标志，供接收方识别

3.2 错误检测和纠正

3.2 错误检测和纠正

一、检错编码(Error detecting code)

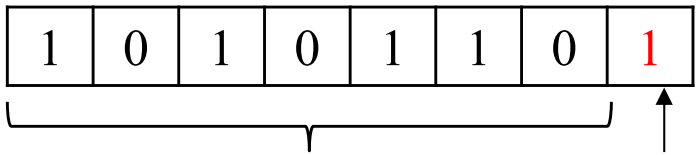
- 任何通信链路在传输数据时都可能出错
- 一般用误码率BER(Bit Error Rate)表示链路可靠性

$$\text{误码率} = \frac{\text{出错的比特数}}{\text{传送的总比特数}}$$

例如：误码率为10^{-10}表示平均每传送10^{10}个比特会出现一个比特出错

- 处理方法
  - 使用可检测并纠正错误的编码：纠错码
  - 使用可检测错误的编码 + 重传：检错码
- 常用检错编码方法
  - 奇偶校验
  - 简单累加和(校验和)
  - 循环冗余校验CRC(Cyclic Redundancy Check)

奇校验举例：



3.2 错误检测和纠正

一、检错编码(Error detecting code)

- 循环冗余校验CRC原理
  - 发送方把数据划分为组，设每组 k 个比特，在其后添加供差错检测用的n 位冗余码，(k+n)比特一起发送
  - 对数据M计算n位冗余码的过程：
    ① 用二进制的模 2 运算进行 2^n乘 M 的运算，这相当于在 M 后面添加 n 个 0
    ② 得到的 (k + n) 位的数除以事先选定好的长度为 (n + 1) 位的除数 P，得出商是 Q 而余数是 R，余数 R 比除数 P 少1 位，即 R 是 n 位
    ③ R作为冗余码，添加在数据M后面，最终发送数据：2^nM+R

    注：除数P为双方事先商定
  - 接收方对收到的(k+n)比特计算冗余码，结果为0表示传输正确，否则表示传输错误

3.2 错误检测和纠正

例：计算101001的3位CRC冗余码

已知：M=101001

k=6, n=3

设：除数P=1101

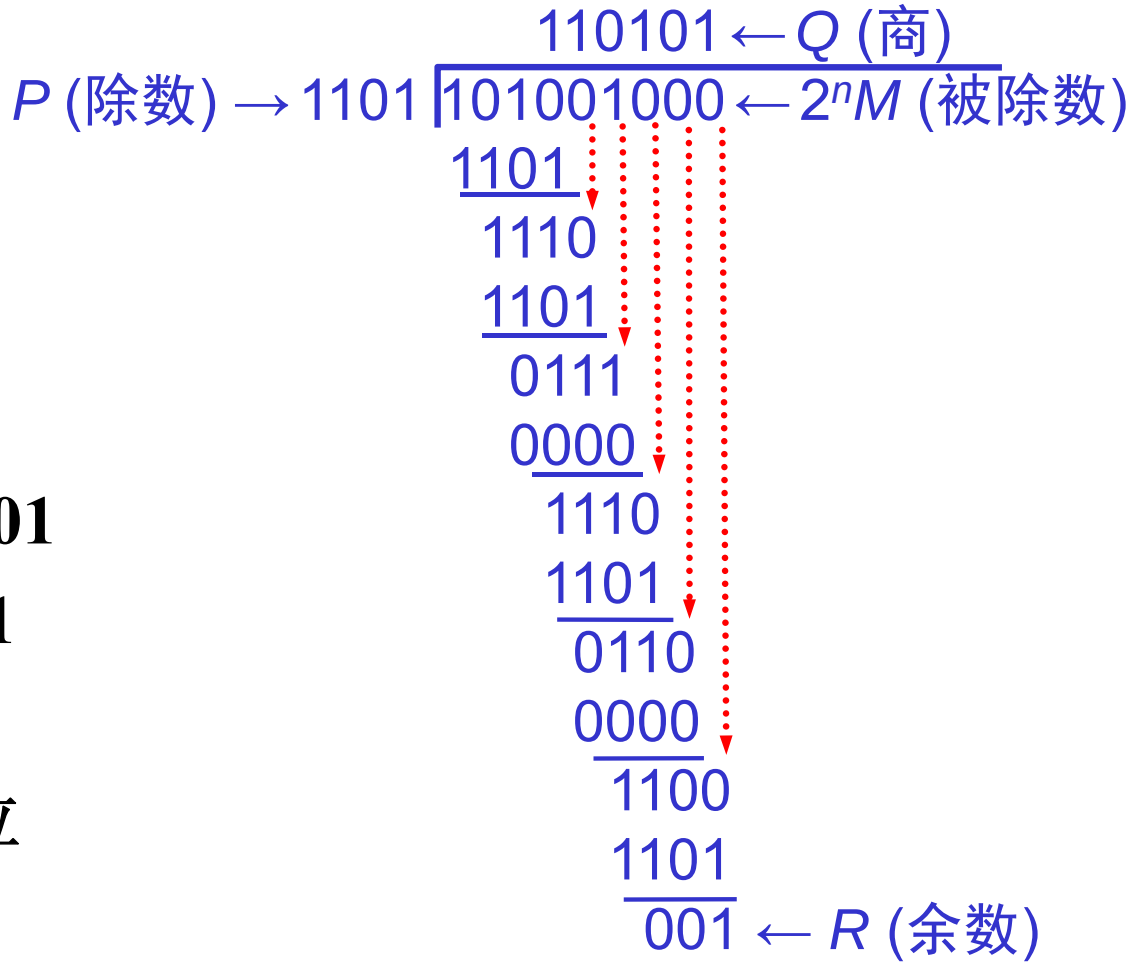
被除数：2^nM = 101001000

模 2 运算的结果：商 Q = 110101

余数 R = 001

发送的数据：$2^nM + R$

即：101001001，共 (k + n) 位



3.2 错误检测和纠正

- 通常用生成多项式$P(x)$表示除数P

  例：除数P=1101的生成多项式$P(X) = X^3 + X^2 + 1$

- 目前广泛使用的生成多项式

  $CRC\text{-}16 = X^{16} + X^{15} + X^2 + 1$

  $CRC\text{-}CCITT = X^{16} + X^{12} + X^5 + 1$

  $CRC\text{-}32 = X^{32}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$

- 在网络中具体实现时，通常采用硬件电路生成**CRC校验和(checksum)**

3.2 错误检测和纠正

二、纠错编码(Error correcting code)

- 检错码只能发现数据出现了错误，无法得知哪个比特出错
- 纠错编码通过增加冗余信息使得能够检测错误发生所在，以便于纠正，又称为前向纠错(forward error correcting)
  - 海明编码

- 关于数据链路层检错/纠错的讨论
  - 通过检错码/纠错码可以做到帧的无差错接收，或者说“无比特差错”
  - 并不意味着可靠传输，其他的错误包括：
    - 帧丢失、帧重复、帧失序

3.3 基本数据链路协议

3.3 基本数据链路协议

未考虑接收方的处理速度
设想：发送方发送>接收方处理速度

一、无限制的单工协议

- 假设前提
  - 单向传输
  - 理想信道
  - 发送方总有数据发送
  - 接收方总能及时处理所收到的数据

发送方

```
void sender1(void)
{
   frame  s;                  //待发送帧
   packet buffer;             //待发送数据包
   while ( true )
   {
      from_network_layer( &buffer ); //从网络层取包
      s.info = buffer;                //拷贝到s中发送
      to_physical_layer( &s );        //通过物理层发送
   }
}
```

接收方

```
void receiver1(void)
{
   frame  r;
   event_type event;   //等待到的事件
   while ( true )
   {
      wait_for_event( &event ); //等待帧到达
      from_physical_layer( &r );//接收帧
      to_network_layer( &r.info );//上交给网络层
   }
}
```

3.3 基本数据链路协议

考虑了接收方的处理速度
但未考虑传输出错

二、单工停-等(stop-and-wait)协议

- 按照“无限制的单工协议”，如果接收方处理帧的速度不及发送方，则帧可能丢失
- 解决方法：增加流量控制(flow control)机制，得到单工停等协议
  - 接收方每收到一帧，都向发送方返回一个应答帧
  - 发送方每发送一帧，都等待来自接收方的应答帧，之后才发送下一帧

发送方

```
void sender2(void)
{  frame  s;                    //待发送帧
   packet buffer;               //待发送数据包
   event_type event;            //等待到的事件
   while ( true )
   {
      from_network_layer( &buffer ); //从网络层取包
      s.info = buffer;                //拷贝到s中发送
      to_physical_layer( &s );        //通过物理层发送
      wait_for_event( &event );       //等待应答帧
   }
}
```

接收方

```
void receiver2(void)
{
   frame  r, s;
   event_type event;   //等待到的事件
   while ( true )
   {
      wait_for_event( &event ); //等待帧到达
      from_physical_layer( &r );//接收帧
      to_network_layer( &r.info );//上交给网络层
      to_physical_layer( &s );     //发送应答帧
   }
}
```

3.3 基本数据链路协议

三、有噪声信道的单工协议

- 在有噪声信道中，帧在传输过程中可能出错
- 解决方法　**---ARQ(Automatic Repeat reQuest)协议**
  - 校验和：使接收方能够检测帧是否出错
  - 确认帧：使发送方知道帧已被正确接收
  - 超时重发：发送方在规定时间内未收到确认帧，则重发帧
  - 帧序号：保证接收方不会重复接收帧
- 协议设计要考虑的三种情形

① 数据帧被正确接收

接收方返回确认帧，发送方收到后继续发送下一帧

② 数据帧出错或丢失

接收方未收到帧或校验出错丢弃该帧，发送方等待确认帧超时后，重发数据帧

③ 确认帧出错或丢失

发送方未收到有效的确认帧，重发数据帧，接收方收到后，检查帧序号重复，不上交该帧，只返回确认帧

发送方

```
void sender3(void)
{  …
   next_frame_to_send = 0;            //帧序号清0
   from_network_layer( &buffer ); //从网络层取包
   while ( true )
   {
      s.info = buffer;                    //拷贝到s中发送
      s.seq = next_frame_to_send;     //帧序号
      to_physical_layer( &s );          //通过物理层发送
      start_timer( s.seq );               //启动计时器
      wait_for_event( &event );         //等待应答帧
      if  ( event == 应答帧到达 )
      {
         from_physical_layer( &s );    //接收应答帧
         if  ( s.ack == next_frame_to_send ) //检查帧序号
         {
            stop_timer( s.ack );          //停止计时器
            from_network_layer( &buffer ); //取下一帧
            inc( next_frame_to_send ); //0,1切换
          }
       }
   }
}
```

接收方

```
void receiver3(void)
{
   frame_expected = 0;   //期待的帧序号
   while ( true )
   {
      //可能的事件：帧到达、校验错
      wait_for_event( &event );
      if ( event == 帧到达 )  //收到有效帧
      {
         from_physical_layer( &r );//接收帧
         if ( r.seq == frame_expected )//帧序号
         {
            to_network_layer( &r.info );//上交
            inc( frame_expected );        //0,1切换
          }
         s.ack = 1 – frame_expected;//应答序号
         to_physical_layer( &s );     //发送应答帧
       }
    }
}
```

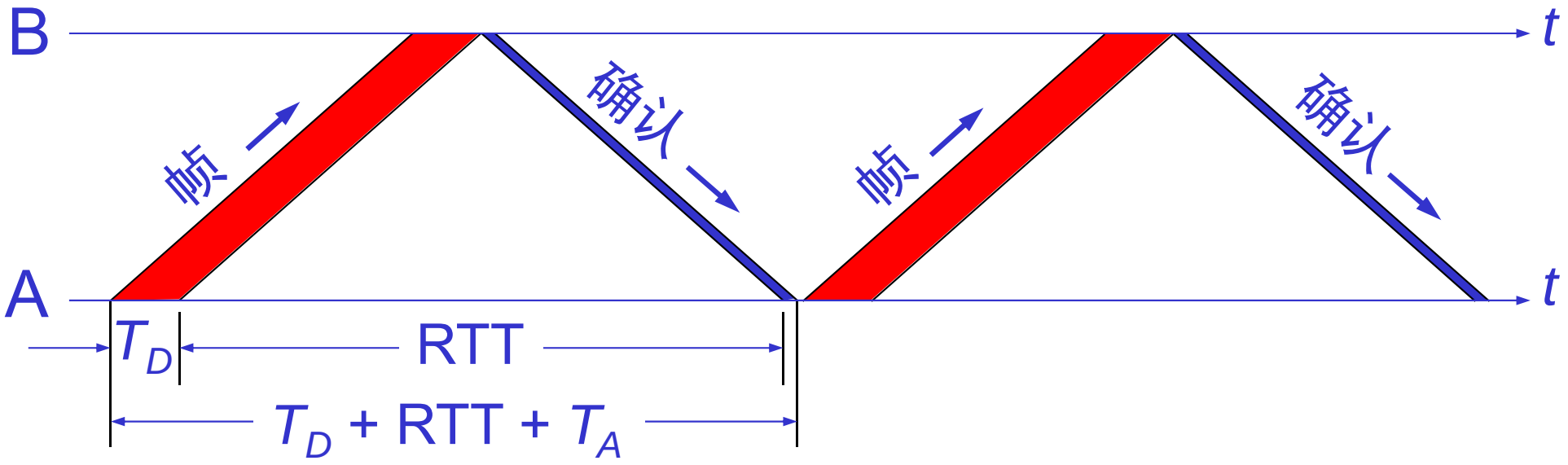
3.4 滑动窗口协议

一、简介

- 传输链路存在时延，而**ARQ**协议在同一时刻仅有一个帧在链路上传输(数据帧或确认帧)，其对信道的利用率较低
  - 请回忆第**1**章介绍的时延带宽积概念
- **ARQ**协议的信道利用率过低(尤其当传输时延较长时)

$$U = \frac{T_D}{T_D + RTT + T_A}$$

T_D：发送数据帧的时间
RTT：往返时延/环路时延
T_A：发送应答帧的时间



3.4 滑动窗口协议

2020年的一道考研题：

假设主机甲采用停-等协议向主机乙发送数据帧，数据帧和确认帧长度均为1000B，传输速率是10kbps，单向传播时延是200ms。则主机甲的最大信道利用率：

A. 80%　　B. 66.7%　　C. 44.4%　　D. 40%

解：

$T_D = T_A = (1000Bx8)/10000bps = 800ms$

$RTT = 200x2 = 400ms$

$U = T_D/(T_D + RTT + T_A) = 800/(800+400+800) = 800/2000 = 0.4$

3.4 滑动窗口协议

二、滑动窗口协议原理(sliding window protocol)

- 滑动窗口协议的基本思想
  - 允许发送方连续发送多个帧
  - 通过滑动窗口实现流量控制
    - 每个待发送的帧都有一个序列号
    - 发送方维护一个发送窗口，它包含一组序列号，对应允许它发送的帧
    - 接收方维护一个接收窗口，对应允许它接收的帧
- 问题：发送窗口和接收窗口的宽度如何确定？
  - 接收窗口宽度：
  - 发送窗口宽度：

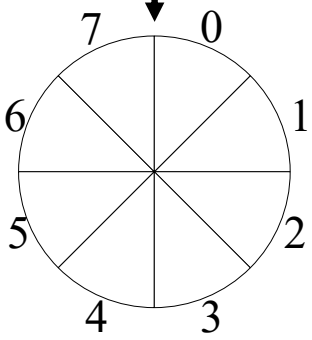
3.4 滑动窗口协议

二、滑动窗口协议原理

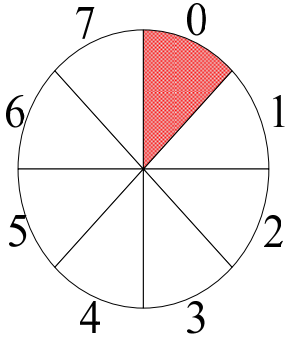
- 发送方
  - 发送窗口内的序列号代表允许它发送的帧
    - 窗口内最大的序列号称为窗口上边界，或窗口上沿、前沿
    - 窗口内最小的序列号称为窗口下边界，或窗口下沿、后沿
  - 每当从网络层得到一个数据包，将其组成帧发出后，发送窗口的上边界+1
  - 发送窗口下边界的帧被接收方确认后，下边界+1
- 接收方
  - 接收窗口内的序列号代表它可以接收的帧
  - 收到的帧序列号等于窗口下边界时，将该帧上交网络层，并返回确认帧，同时整个窗口向前移动1个位置
  - 如果收到帧序列号落在接收窗口之外，则将其丢弃
  - 注意：接收窗口总是保持固定大小

3.4 滑动窗口协议

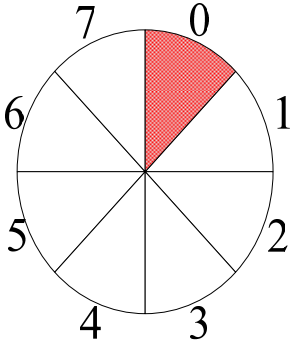
发送方

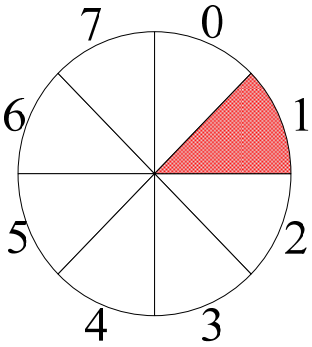


初始状态

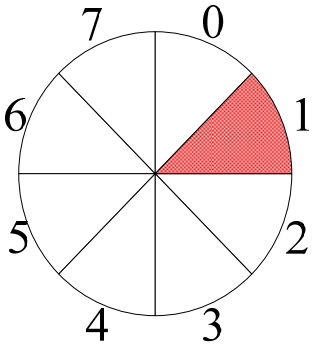


发出0号帧

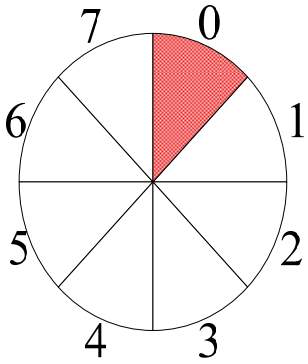




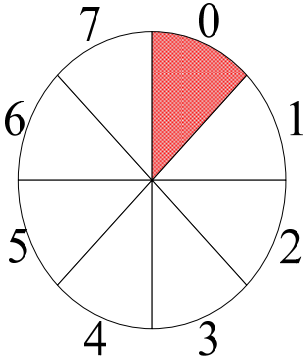
收到ACK0
发出1号帧



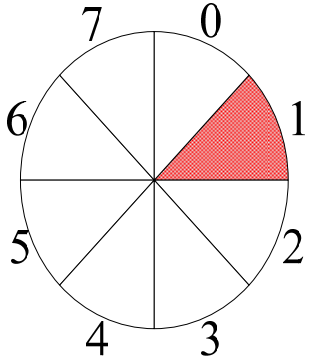
接收方



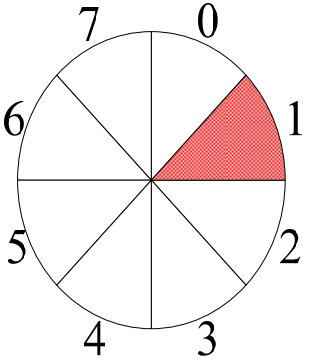
初始状态

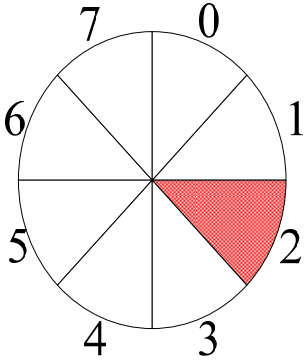


可收0号帧



收到0号帧
返回ACK0





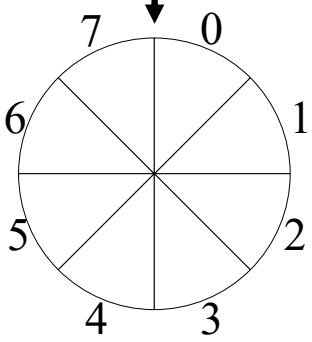
收到1号帧
返回ACK1

假设：发送窗口$W_T = 1$，接收窗口$W_r = 1$

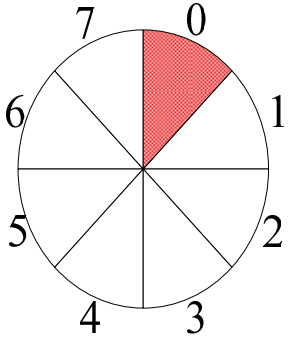
窗口最大尺寸为1的滑动窗口协议称为1位滑动窗口协议，即为ARQ协议

3.4 滑动窗口协议

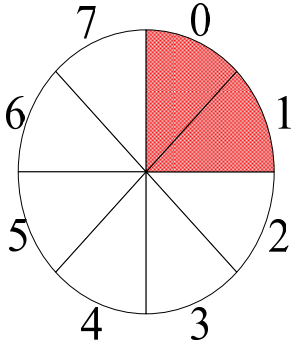
发送方



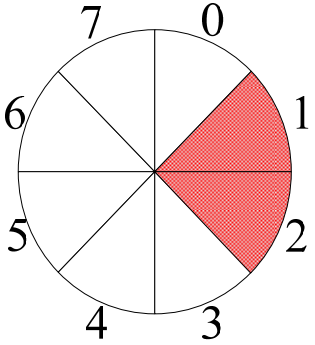
初始状态



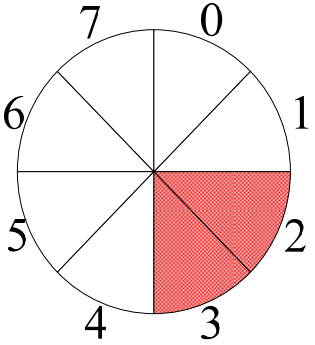
发出0号帧



发出1号帧

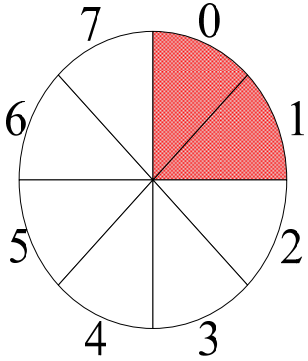


收到ACK0
发出2号帧

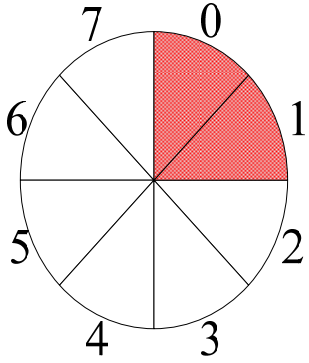


收到ACK1
发出3号帧

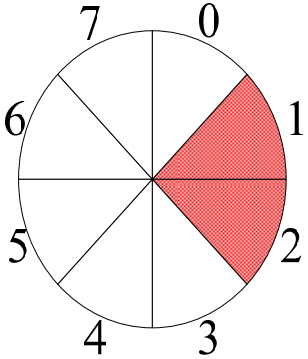
接收方



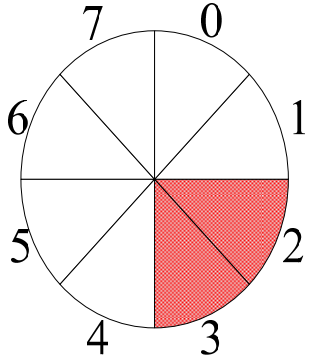
初始状态



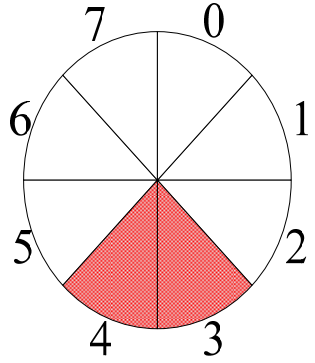
可收0,1号帧



收到0号帧
返回ACK0



收到1号帧
返回ACK1



收到2号帧
返回ACK2

假设：发送窗口$W_T = 2$，接收窗口$W_r = 2$