



## 第三章 数据链路层（二）

---

袁华, [hyuan@scut.edu.cn](mailto:hyuan@scut.edu.cn)

华南理工大学计算机科学与工程学院  
广东省计算机网络重点实验室

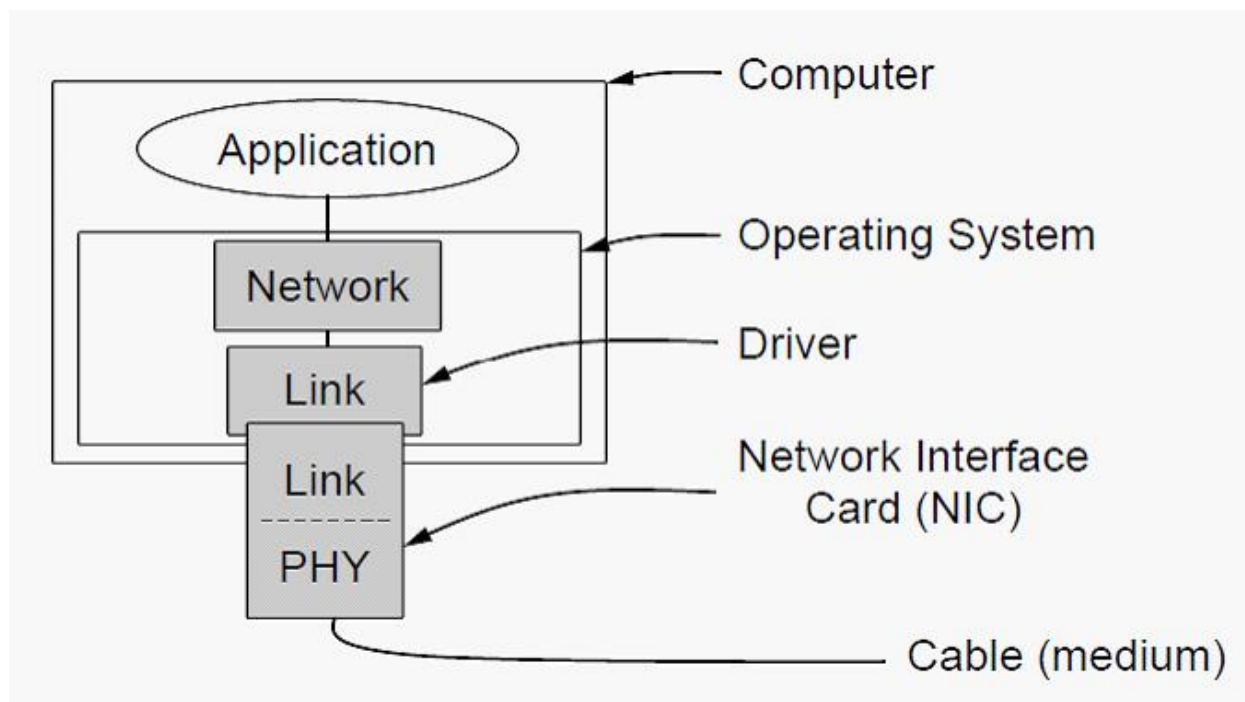


# 本节目的（3.3、3.4）

---

- 掌握**6**种基本的**DLL**协议
- 掌握滑动窗口技术
- 掌握**PAR/ARQ**（肯定确认重传/自动重复请求）技术
- 掌握捎带确认技术

# 链路层通信模型P168





# 几个假设 P168

---

- 物理层、数据链路层和网络层各自是独立的处理进程
- 机器**A**希望向**B**发送的是一个可靠的、面向连接的长数据流
- 假设机器不会崩溃
- 从网络层拿到的数据是纯数据

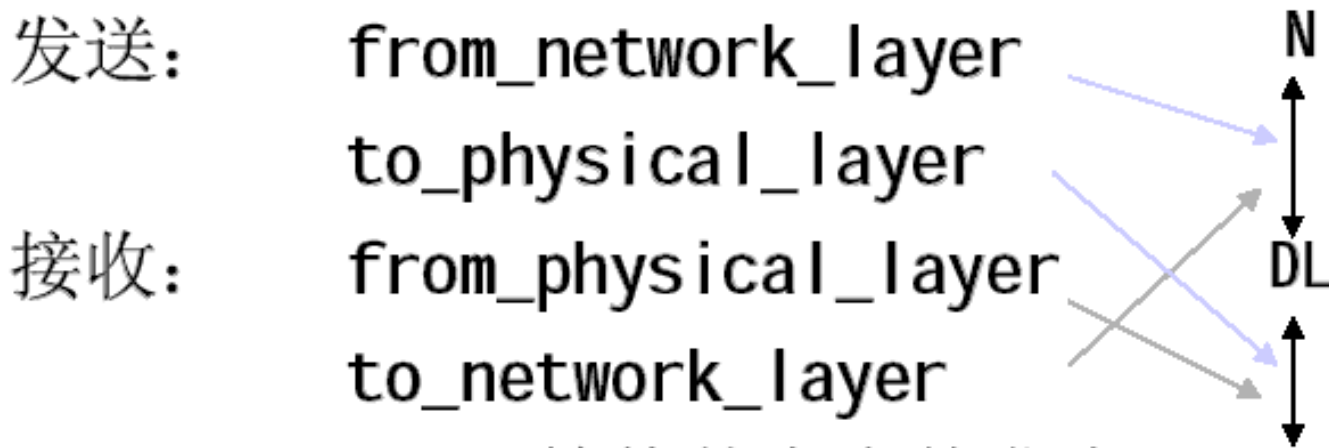
# 基本数据链路协议

- 由浅入深地学习六个协议，了解数据链路层的主要工作，这些工作原理和概念是网络通信的基础，其运行机理在网络层和传输层也会采用。
- 首先介绍最简单的三个协议：
  - 无限制的**单工**协议
  - **单工**停一等协议
  - 有噪声信道的**单工**协议

数据的传输在某时是单向的

## 几点说明 (1/2)

- 六个协议共同使用的数据类型、调用的函数过程定义见图3.9 (**protocol.h**)。
- 与网络层、物理层之间的数据传送接口





## 几点说明 (2/2)

---

- **Wait\_for\_event** 等待某个事件发生  
event: frame\_arrival, cksum\_err, timeout
- **Timer** 计时器的作用
  - start\_timer, stop\_timer, start\_ack\_timer, stop\_ack\_timer

重传定时器

捎带确认定时器



# 帧结构: P169

---

- **typedef struct{  
    frame\_kind kind;  
    seq\_nr seq;  
    seq\_nr ack;  
    packet info;  
}frame;**



# 无限制的**单工**协议(协议1) P171

- 数据**单向**传送
- 收发双方的网络层都处于就绪状态（**随时待命**）
- 处理时间忽略不计（**瞬间完成**）
- 可用的缓存空间无穷大（**无限空间**）
- 假设**DLL**之间的信道永远不会损坏或者丢失帧（**完美通道**）
- “乌托邦”

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"
```

```
void sender(void)
```

```
{
    frame s;                /* buffer for an outbound frame */
    packet buffer;          /* buffer for an outbound packet */

    while (true) {
        from_network_layer(&buffer);    /* go get something to send */
        s.info = buffer;                /* copy it into s for transmission */
        to_physical_layer(&s);          /* send it on its way */
    }
    /* Tomorrow, and tomorrow, and tomorrow,
       Creeps in this petty pace from day to day
       To the last syllable of recorded time.
       - Macbeth, V, v */
}
```

封装, 发送

```
void receiver(void)
```

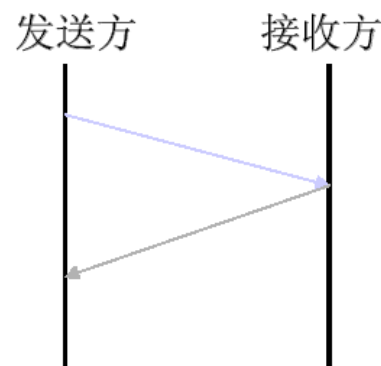
```
{
    frame r;
    event_type event;        /* filled in by wait, but not used here */


    while (true) {
        wait_for_event(&event);    /* only possibility is frame_arrival */
        from_physical_layer(&r);    /* go get the inbound frame */
        to_network_layer(&r.info); /* pass the data to the network layer */
    }
}
```

解封装, 发送

# 单工的停—等协议 (协议2) P172

- 解决如何避免收方被涌入的数据淹没，即取消“接收方允许无限量接收”的假设
- 解决方法：收方回发一个**哑帧**，接收方收到哑帧，表明收方允许接收数据，此时再次发送下一帧数据。
- 实际上是半双工协议





```

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);
    }
}

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);
    }
}

```

/\* buffer for an outbound frame \*/  
 /\* buffer for an outbound packet \*/  
 /\* frame\_arrival is the only possibility \*/

/\* go get something to send \*/  
 /\* copy it into s for transmission \*/  
 /\* bye bye little frame \*/  
 /\* do not proceed until given the go ahead \*/

/\* buffers for frames \*/  
 /\* frame\_arrival is the only possibility \*/

/\* only possibility is frame\_arrival \*/  
 /\* go get the inbound frame \*/  
 /\* pass the data to the network layer \*/  
 /\* send a dummy frame to awaken sender \*/

# 有错误信道的单工协议 (协议3) P173

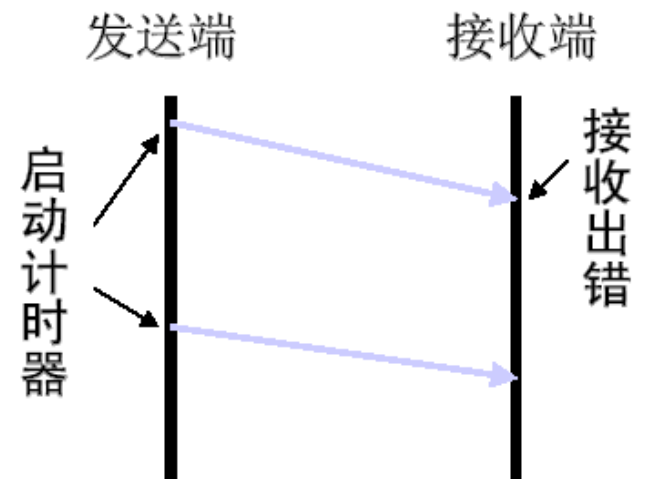
- 有噪声就会产生差错，有差错就可能会引起以下这些问题：
  - 接收方检测到错误帧，如何通知发送方？如何恢复正确帧？
  - 数据帧或确认帧在途中丢失将如何解决？
  - 有可能收到重复帧，如何解决？

# 解决以上问题的方法 (1/2)

- 确认；重发；计时器；帧序号。
- 确认帧
  - 接收端进行差错检查后，发确认帧送回发送端，告诉接收是否正确。
  - 为了减少网络内的流量，只在接收无差错时才发确认帧，出错时不发确认帧。发送端等候计时器给定的时间后仍未收到确认帧，则知道接收出错，此时发送端重发原来的帧。

# 主动确认重传P175

- 接收方无误，回发确认帧。
- **ARQ:automatic repeat request P175**
- **PAR:positive acknowledgement with retransmission**



# 解决以上问题的方法（2/2）

## ■ 重发

- 网络中采用检错码，无法纠正错误，由重发原来帧的方式来恢复正确的帧。

## ■ 重传计时器

- 控制何时重发，防止无限期等待（死锁）。

## ■ 帧序号

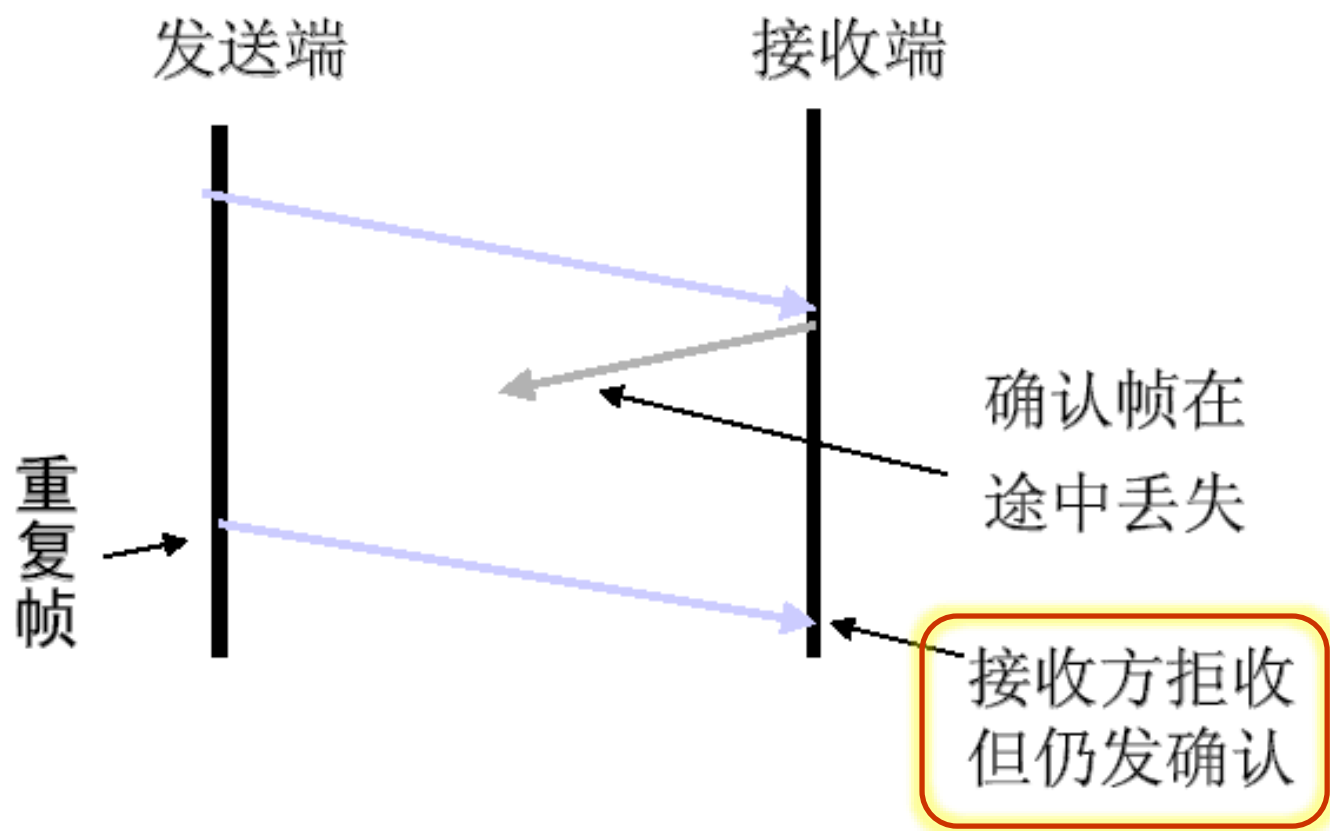
- 防止重发时接收端收到重复的帧，序号还用于接收时排序。
- 保证送给网络层的都是按序无重复的分组



# 计时器的作用

- 通过发送方计时器（超时）解决。
  - 超时（**TimeOut**）：在传输过程中，所发送的帧丢失，接收方根本没有收到，不可能发送确认帧（包括否定性确认），而发送方正在等待接收方的确认帧，当然也不可能等到接收方的确认。所以，发送方一旦发送一帧，就启动一计时器，在规定的时间内，一般都应收到确认，如收不到确认，则在计时器溢出时，再重发此帧。
  - 确认超时：捎带确认超时，单发确认帧

# 接收端为什么会收到重复帧？





# 区分重复帧

---

- 重发机制也包括当接收方发送的确认帧丢失而导致发送方的重发
- 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，要能区别是否是同一帧，则每帧都应有一个**帧的编号**
- 支持重传的肯定确认协议

```

void sender3(void)
{
    seq_nr next_frame_to_send;    /* seq number of next outgoing frame */
    frame s;                      /* scratch variable */
    packet buffer;                /* buffer for an outbound packet */
    event_type event;

    next_frame_to_send = 0;       /* initialize outbound sequence numbers */
    from_network_layer(&buffer);  /* fetch first packet */
    while (true) {
        s.info = buffer;          /* construct a frame for transmission */
        s.seq = next_frame_to_send; /* insert sequence number in frame */
        to_physical_layer(&s);    /* send it on its way */
        start_timer(s.seq);       /* if answer takes too long, time out */
        wait_for_event(&event);   /* frame_arrival, cksum_err, timeout */
        if (event == frame_arrival) {
            from_physical_layer(&s); /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                stop_timer(s.ack); /* turn the timer off */
                from_network_layer(&buffer); /* get the next one to send */
                inc(next_frame_to_send); /* increment next_frame_to_send */
            }
        }
    }
}

```

```

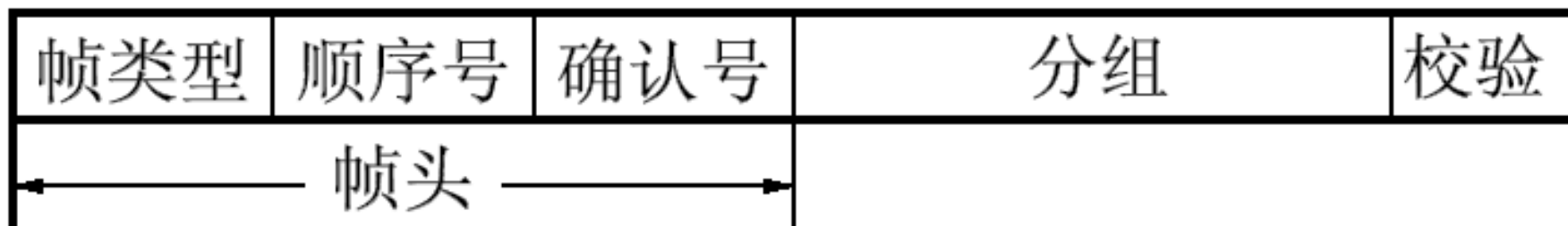
void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;

    frame_expected = 0;
    while (true) {
        wait_for_event(&event);    /* possibilities: frame_arrival, cksum_err */
        if (event == frame_arrival) { /* a valid frame has arrived. */
            from_physical_layer(&r); /* go get the newly arrived frame */
            if (r.seq == frame_expected) { /* this is what we have been waiting for. */
                to_network_layer(&r.info); /* pass the data to the network layer */
                inc(frame_expected); /* next time expect the other sequence nr */
            }
            s.ack = 1 - frame_expected; /* tell which frame is being acked */
            to_physical_layer(&s); /* send acknowledgement */
        }
    }
}

```

# 帧格式

一般意义上的帧的格式



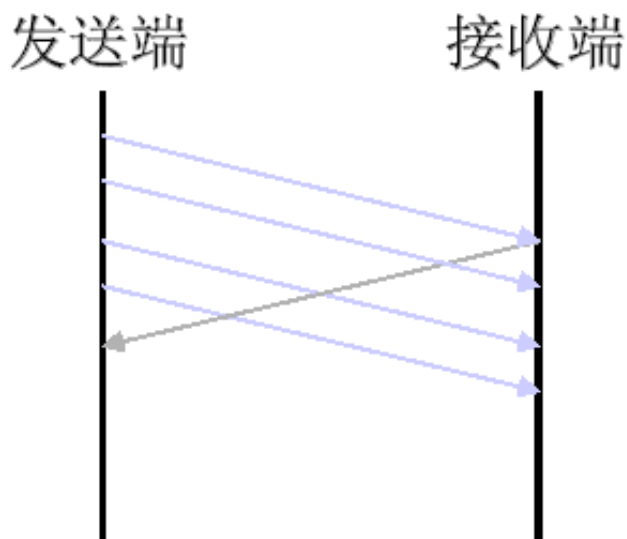


# 捎带确认 (piggyBack P176)

- 捎带确认的作用
  - 进一步减少确认帧。
- 捎带确认：将确认暂时延迟以便可以钩到一个外发的数据帧 (**s.ack**)
- 如无法“捎带”，当一个控制捎带确认的计时器超时后，单独发确认帧。

# 批量发送数据

- 停一等协议是每收到一个确认才能发送下一帧，发送端等待时间太长，网络通信效率不高。为了提高效率，可以在等待的时间发送数据帧，这样大大减少了浪费的时间。





# 问题:引出滑窗技术

- 如果发送端可以连续发送一批数据帧，必须考虑接收端是否来得及接纳与处理这么多的帧，这里就提出了网络流量控制问题。
- 发送窗口**P177**
- 接收窗口**P177**



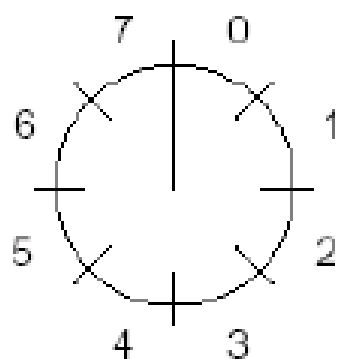


# 一位的滑动窗口协议

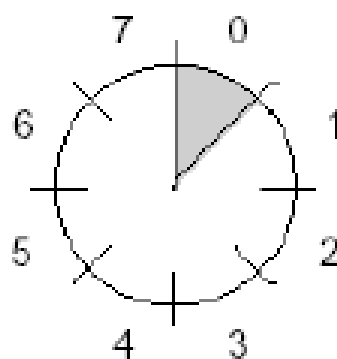
- 滑动窗口用于控制网络中的流量。（P177）
- 发送方和接收方接收能力的匹配即流量控制。
- 如接收方的处理能力低于发送方，接收方的缓冲区可能被“占满”，所以通常在接方收的缓冲区到达一定量时，应及时通知发送方，暂停发送，等候通知，这就是流量控制机制。

# 大小为**1**，有**3**位序列号的滑动窗口

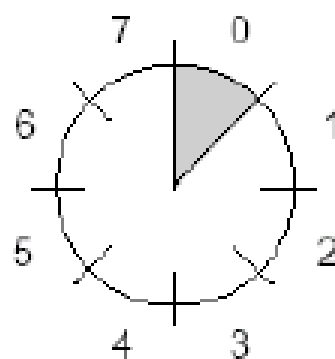
Sender



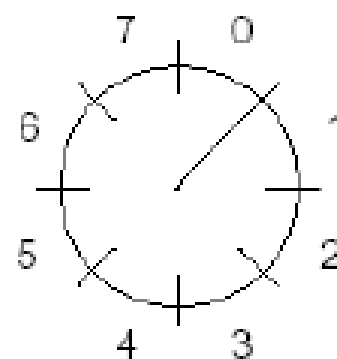
初始时



发送了第一帧

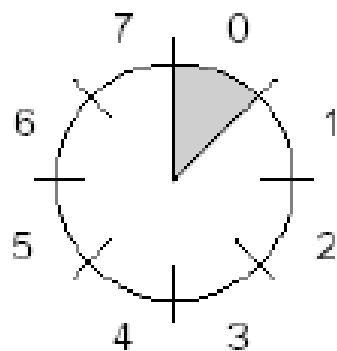


接收了第一帧

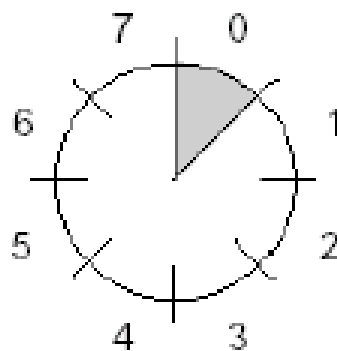


收到第一个确认帧

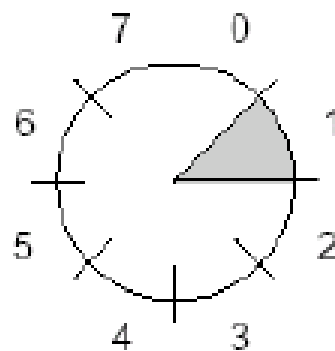
Receiver



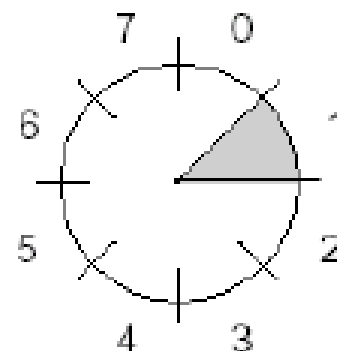
(a)



(b)



(c)



(d)

# 原理

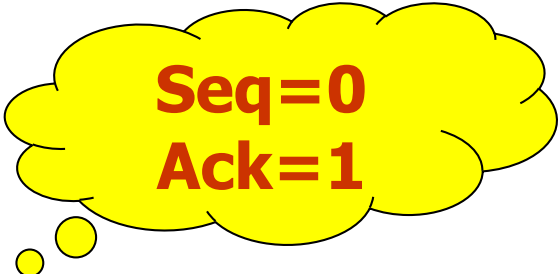
- 接收方收到帧后，首先核对是否为预期帧号 (**frame\_expected**)，如果是的，则接收并 **frame\_expected+1**，即移动接收窗口。
- 发送端收到应答帧，核对响应帧号 **next\_frame\_to\_send**，核对无误后，从网络层取新的帧，并执行 **next\_frame\_to\_send+1**，即移动发送窗口。如核对帧号不正确，则不移动窗口



```
void protocol4(void)
{
```

```
    seq_nr next_frame_to_send;
    seq_nr frame_expected;
    frame r, s;
    packet buffer;
    event_type event;
```

```
    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);
```



**Seq=0**  
**Ack=1**

```

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);
}

```

移动  
接收窗口

收到对方的  
捎带确认

移动  
发送窗口

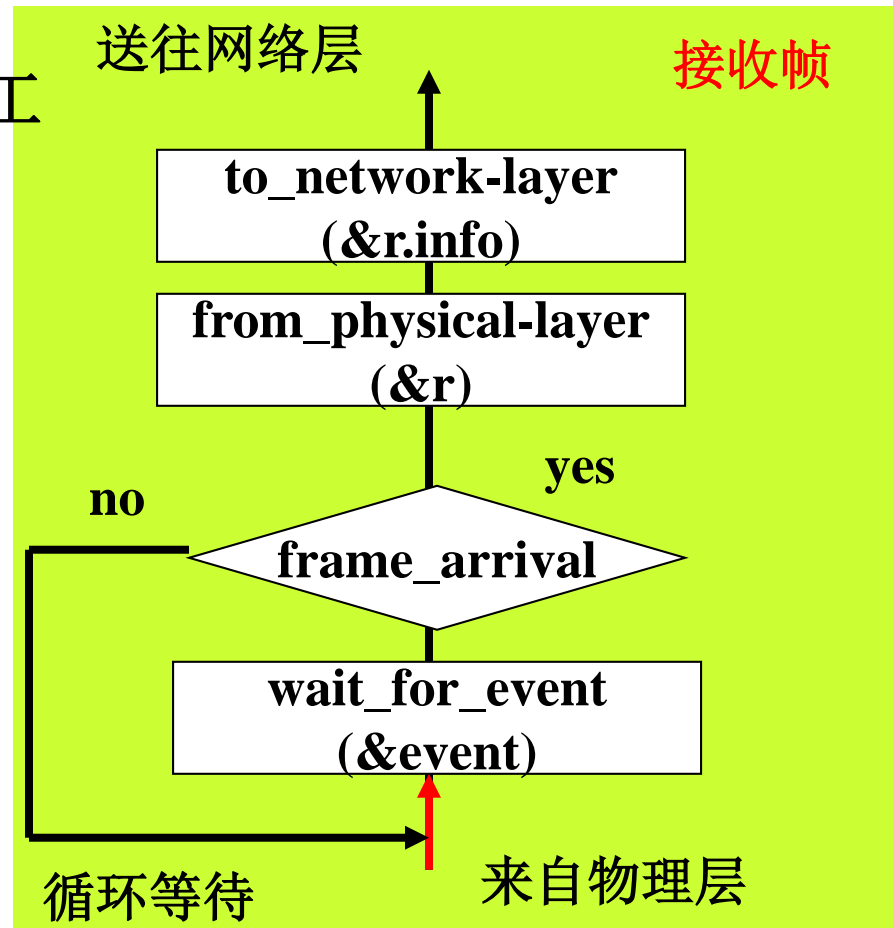
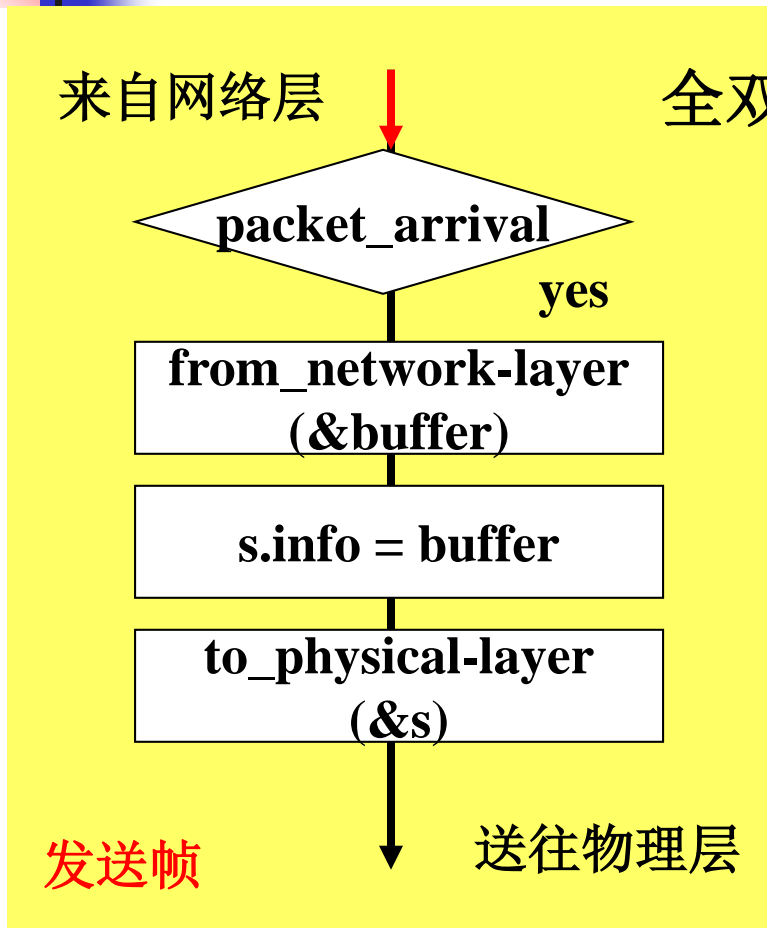
捎带确认



# 滑动窗口协议

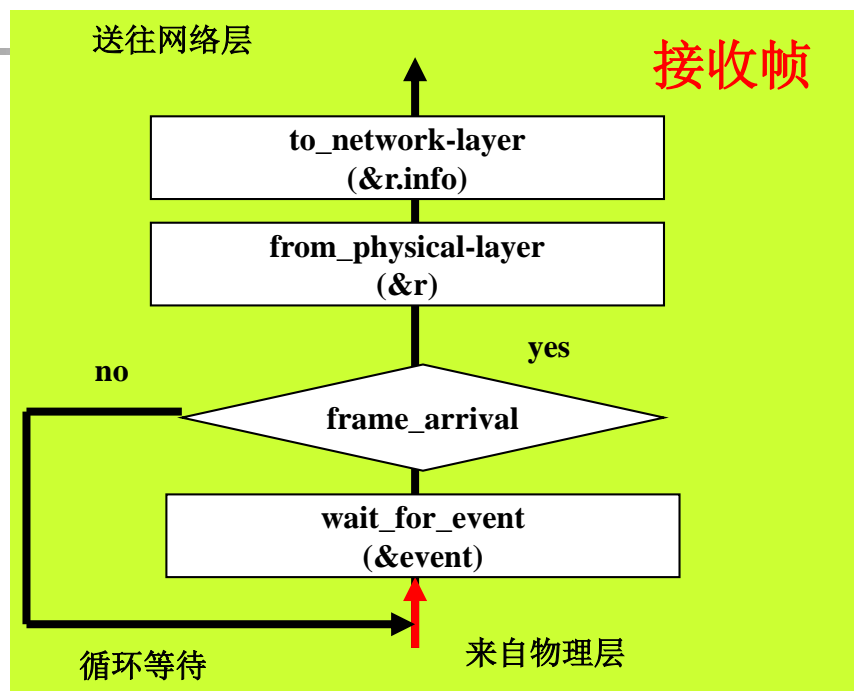
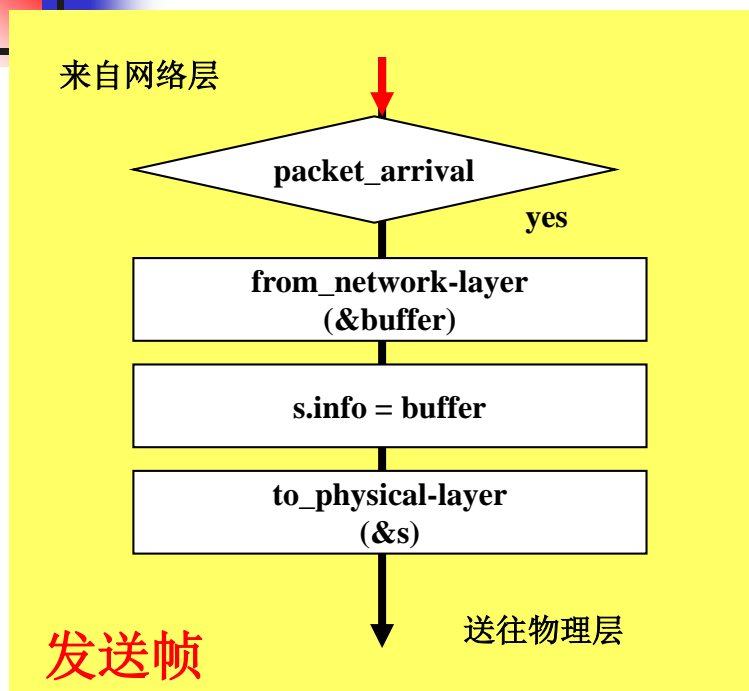
- 提高效率的方法
  - 传输方式：全双工（**full-duplex**）
  - 确认方式：捎带（**piggyback**）
  - 发送/接收方式：滑动窗口（**sliding window**）
- 滑动窗口协议
  - 协议4： **n=1**——引出滑动窗口的基本概念
  - 协议5： 回退**n**帧（**Go Back n**）
  - 协议6： 选择重传（**Select Repeat**）

# 传输方式：同时运行发/收进程





# 确认方式：在数据帧中捎带确认



A send s to B

A get r from B

类型

序列号

确认号

数据

data	seq fromA	ack toB	packet from A L3
------	-----------	---------	------------------

# w=1滑动窗口概念

## A的发送窗口

	A7	窗口外的 发送帧
	A6	
1	A5	待确认的 发送帧
0	A4	
1	A3	被确认的 发送帧
0	A2	
1	A1	
0	A0	
Seq	Data	

## B的接收窗口

Seq	Data	提交网络 层的正确 接收帧
0	A0	
1	A1	
0	A2	
1	A3	待提交的 接收帧
0	A4	
1	A5	窗口外的 接收帧
	A6	

# 滑动窗口的基本概念

- 每个待发送帧被赋予一个序列号**seq**
  - **seq**的取值范围是  $0 \sim 2^n - 1$  ( $n$ 位字段)
- 建立缓冲区
  - 发送窗口：缓存已发送、待确认的帧
    - 顺序接收来自网络层的分组，成帧，赋予序列号
    - 最多保存**W**个已经发送、等待确认的帧
    - 窗口达到最大值**W**时强制关闭网络层
  - 接收窗口：缓存期待接收的帧（序号）
    - 对进入窗口的帧顺序提交网络层，产生确认
    - 落在窗口外的帧被丢弃

# 发送窗口的滑动 ( $w=3$ )

顺序接收来自网络  
层的分组，成帧

10	
01	A1
00	A0
Seq	Data

赋予序列号，经物理层  
发送并在缓冲区保存最  
多W个待确认帧。

收到确认帧后向后  
滑动，至窗口满。

	A4
11	A3
10	A2
01	A1
00	A0
Seq	Data

	A7
	A6
01	A5
00	A4
11	A3
10	A2
01	A1
00	A0
Seq	Data

# 接收窗口的滑动 ( $w=3$ )

对进入窗口的帧顺序提交网络层，产生确认。

Seq	Data
00	A0
01	
10	

接收来自物理层的帧

接收窗口向后滑动，最多保存待上交帧 $W$ 个。

Seq	Data
00	A0
01	A1
10	A2
11	

接收窗口满，落在窗口外的帧被丢弃。

Seq	Data
00	A0
01	A1
10	A2
11	A3
00	A4
01	A5
	A6

# 协议4的滑动窗口基本工作原理

## ■ 窗口设置

- 滑动窗口最大值: **MAX\_SEQ = 1**
- 通信双方初始值: **seq = 0, ack = 1** (期待接收 **seq = 0**)

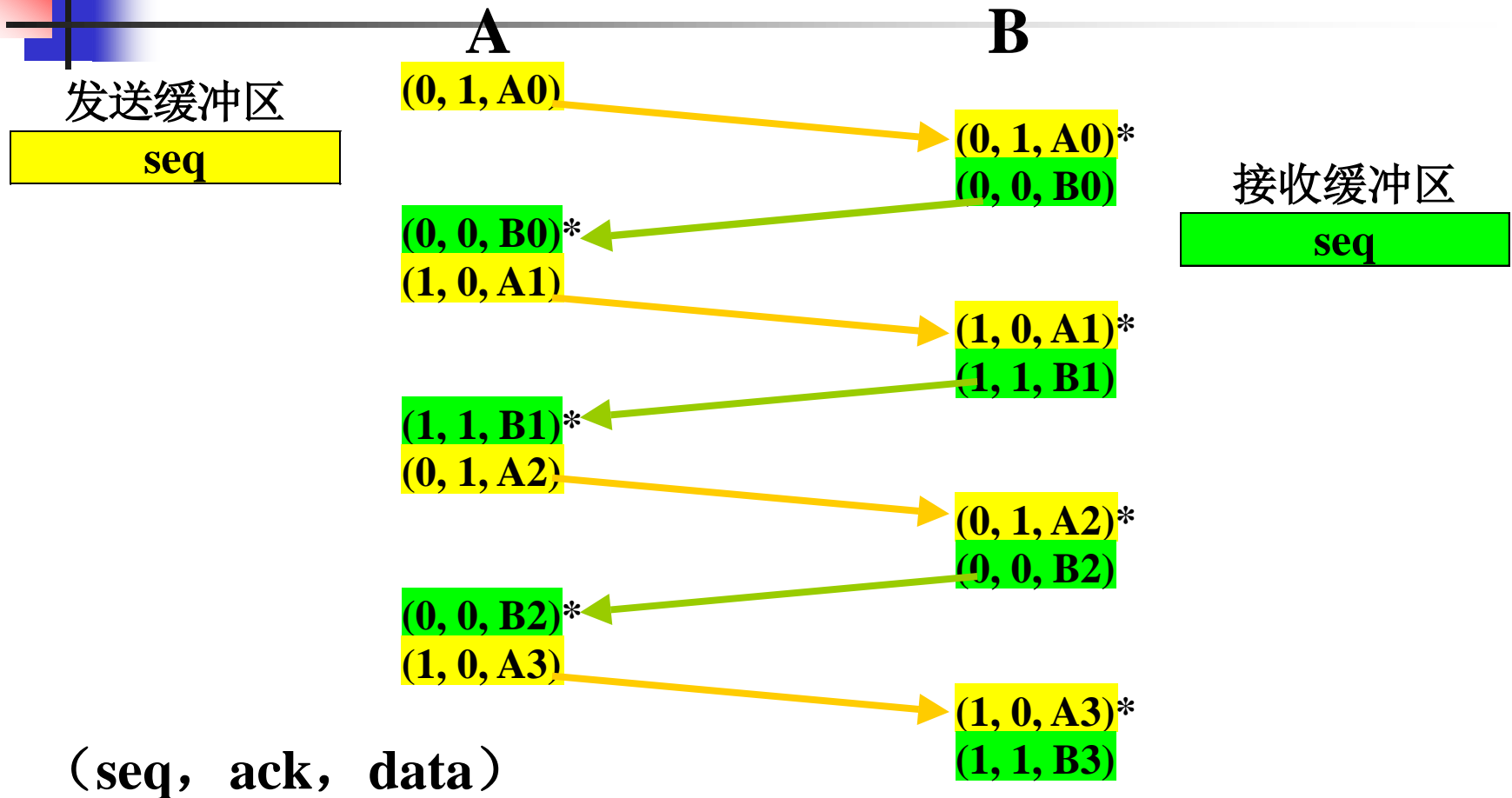
## ■ 窗口滑动机制

- **A**首先发送数据帧 (**seq = 0, ack = 1, A0**)
- **B**收到**A0**, 发送捎带确认帧 (**seq = 0, ack = 0, B0**)
- **A**收到对**A0**的确认, 滑动窗口, 发送帧 (**seq = 1, ack = 0, A1**)

## ■ 特点

- 序列号**seq**和确认值**ack**“0”“1”交替
- 滑动窗口长度**W = 1**, 收到确认才移动窗口
- 保证按顺序将接收到的正确帧只一次上交网络层

# 正常情况下发送窗口滑动机制



# 异常情况一：对重复帧的差错控制

定时器  
设置短了

超时重传

超时重传

seq ✓ acq ✓

期望 seq<sub>B</sub>=1

seq ✗ acq ✗

期望 seq<sub>B</sub>=1

eq<sub>B</sub>=0

(0, 1, A0)

(0, 1, A0)

(0, 1, A0)

(0, 0, B0)\*

(1, 0, A1)

(0, 0, B0)

(1, 0, A1)

B

(0, 1, A0) \*

(0, 0, B0)

(0, 1, A0)

(0, 0, B0)

(0, 1, A0)

(0, 0, B0)

(1, 0, A1)\*

(1, 1, B 1)

期望 seq<sub>A</sub>=0

交网络层

期望 seq<sub>A</sub>=1

期望 acq<sub>A</sub>=0

seq ✗ acq ✗

期望 seq<sub>A</sub>=1

seq ✗ acq ✗

期望 seq<sub>A</sub>=1

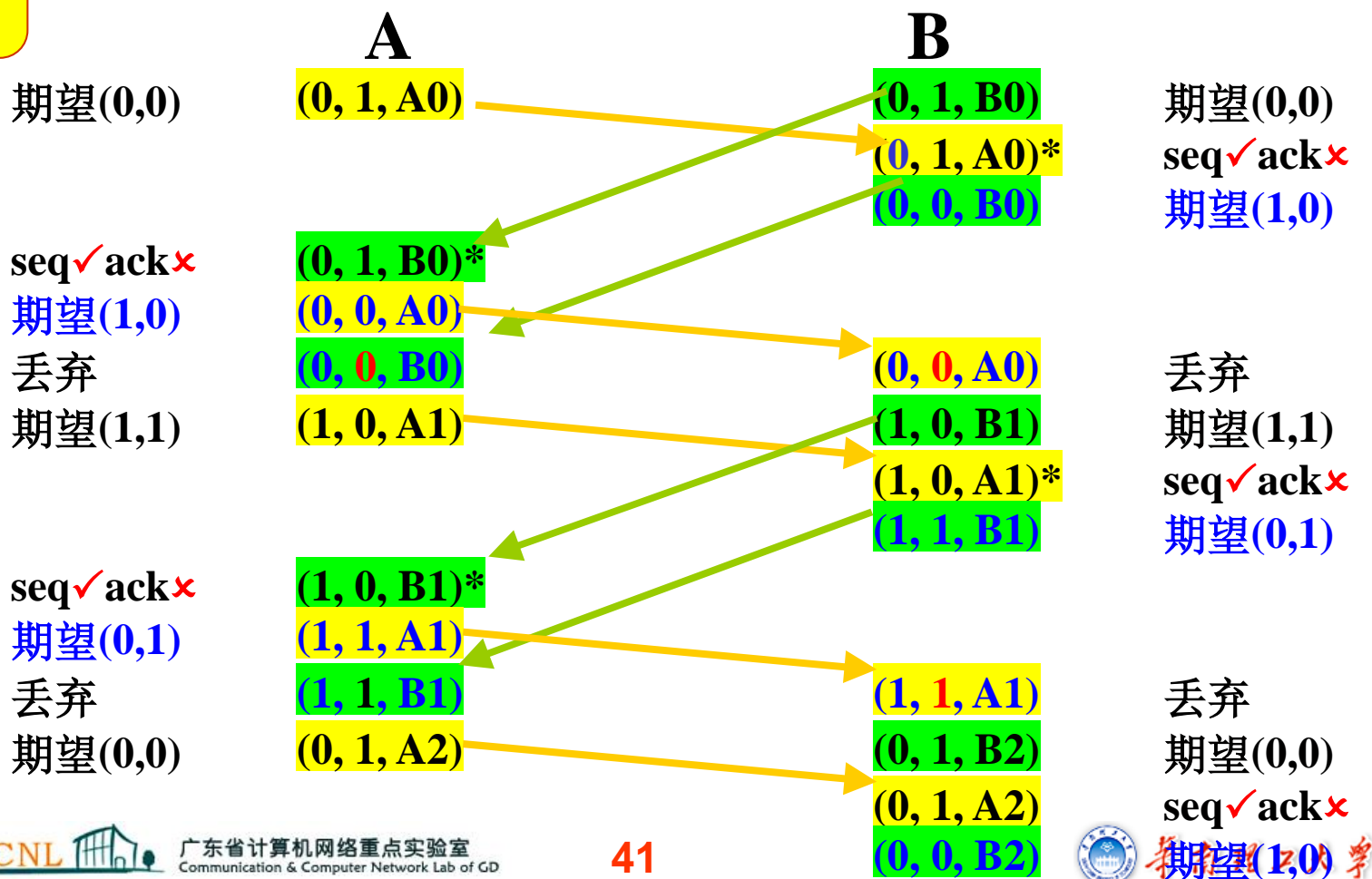
seq ✓ acq ✓

期望 seq<sub>A</sub>=0



# 异常情况二： 同步开始发送过程的差错控制

同时开始  
传输





## 协议4的信道利用率<sub>P180~181</sub>

---

通过例题分析协议4（双向停-等协议）存在的信道利用率低问题，其解决办法为管道化技术（**pipelining**），并提出差错控制面临的新问题。

# 协议4的信道利用率

- 在协议4中假设:以下时间是可以忽略的:
    - 接收方处理到达帧的时间
  - 事实上, 在低速信道上, 来回时间 (**RTT: the round-trip time**) 可能非常大, 发送方在这段时间处于**blocked**状态
  - 如果:
    - 信道传输速率是: **b** bps
    - 每帧的大小是: **k** bits
    - 来回时间是: **R** sec
- 则信道的利用率是:

$$\text{Line Utilization Rate} = k / (k + bR)$$



## 例题P181

---

- 已知：
  - 信道容量  $b = 50 \text{ kbps}$
  - 传输延迟  $R = 500 \text{ ms}$ （双程）
  - 数据帧的长度  $k = 1000 \text{ bit}$
  - 设接收方收到数据帧后马上回送确认短帧，没有延时
- 求：信道利用率

$$\text{Or: } k/k+bR=1000/1000+50\text{kbps}\times 500\text{ms}=3.85\%$$

## 例题解答

- 在源端发送数据帧过程需要的时间

$$T_f = k/b = 20 \text{ ms}$$

- 从发送完毕到确认帧返回需要的时间（双程延迟）

$$R = 500 \text{ ms}$$

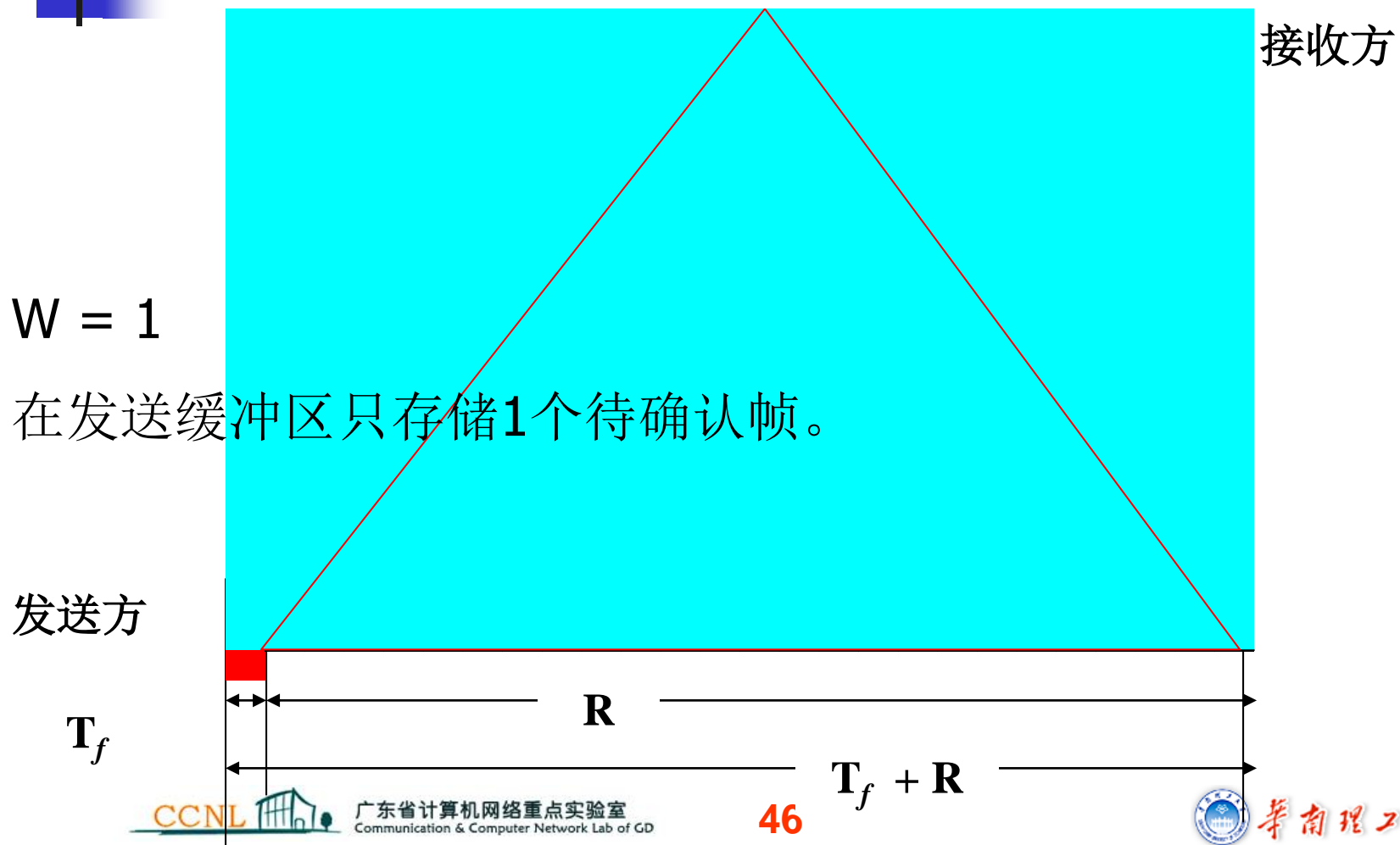
- 从开始发送到确认返回总共需要的时间

$$(T_f + R) = 20 + 500 = 520 \text{ ms}$$

- 线路的利用率

$$T_f / (T_f + R) = 20 / 520 = 3.85 \%$$

# 信道利用率不足4%



# 提高信道利用率的方法

- 增加滑动窗口最大长度**W**

$$\begin{aligned}\text{信道利用率} &= W * T_f / (T_f + R) \\ &= W * k / (k + bR)\end{aligned}$$

- 理想情况下，使例题信道利用率达到**100%**，  
则滑动窗口最大长度为：

$$W = (T_f + R) / T_f = 520 / 20 = 26$$

# 使信道利用率达到100%

接收方

**$W = 26$**

连续发送、并在发送缓冲区存储**26**个待确认帧。

发送方

$T_f$

$T_f + R$



# 怎样找到一个合适的w值？ P180

- 信道上的容量：一帧从发送方传输到接收期间可容纳的帧数量
- 带宽-延迟积：**BD**
- 窗口值： **$w=2*BD+1$**
- 上述的例子：
  - **$BD=50\text{kbps}*0.250=12.5\text{kb}$**
  - **$W=2*12.5\text{kb}+1=26\text{kb}=26\text{帧}$**

实际上：

$$w \leq 2*BD+1$$

# 一个例子：2014考研题

- 主机甲和主机乙之间使用后退N帧协议（GBN）传输数据，甲的发送窗口尺寸为**1000**，数据帧长为**1000**字节，信道为**100Mbps**，乙每收到一个数据帧立即利用一个短帧（忽略其传输延迟）进行确认。若甲乙之间的单向传播延迟是**50ms**，则甲可以达到的最大平均传输速率约为：
  - A. 10Mnps                      B. 20Mbps
  - C. 80Mbps                      D. 100Mbps



# 解答

---

- 设可达到的最大传输率为 **x**，于是
  - **$1000f * 1000Bpf * 8 = xbps * 2 * 50ms / 1000ms$**
  - **$8000000 = 8000 + 2 * 0.05x$**
  - **$x = 80Mbps$**
  - 或者： **$w = 1251$** ，现在只有 **1000**，于是
  - **$X = 100M * (1000 / 1251) = 80M$**



# 用管道化技术发送帧面临的新问题

- 出错情况

- 连续发送 $W$ 个数据帧，其中有一帧出错，但其后续帧被成功发送

- 接收方的接收策略选择

- 丢弃错帧及后续帧，其后续帧因不是期望接收帧也被丢弃
- 丢弃错帧，缓存后续正确接收帧

- 对应的发送方的重传策略选择

- 缓存在发送窗口中的出错帧以及其后续帧全部重发——协议5
- 只重发出错帧——协议6



## 协议5：回退n帧

---

### 接收方的接收策略选择：

丢弃错帧，其后续帧因不是期望接收帧也被丢弃（接收窗口为**1**）。

### 发送方的重传策略选择：

缓存在发送窗口中的出错帧以及其后续帧全部重发。

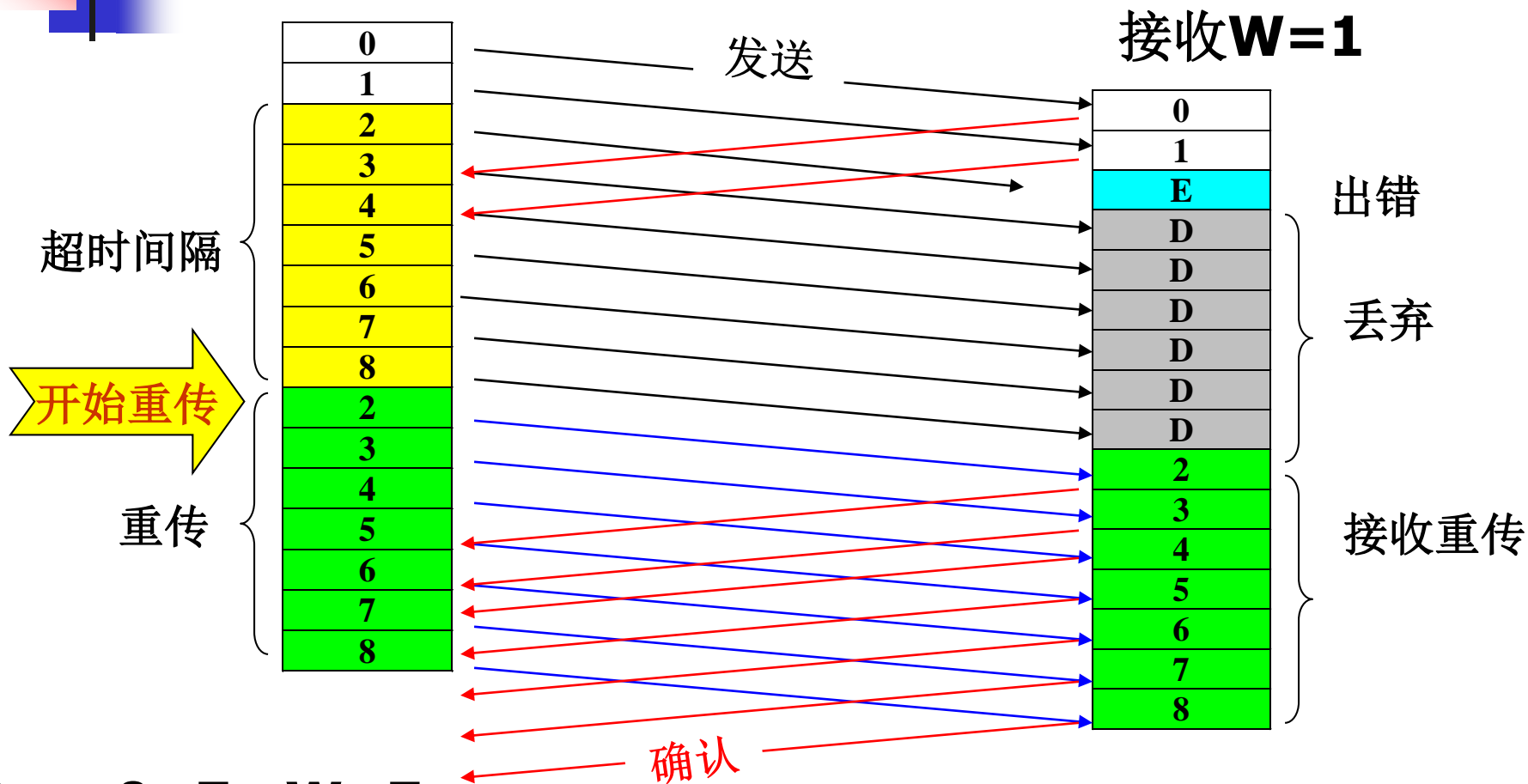


# 回退n帧协议的基本概念

- 定义序列号**seq**的取值范围和滑动窗口长度**W**
- 发送方连续发送至发送窗口满
- 接收窗口为**1**，对出错帧不确认（引发超时）
- 发送方超时重传，从未被确认帧开始

举例（**MAX\_SEQ = 7**）

# 回退n帧协议举例



Seq=0~7; W=7

# 回退n帧协议的工作原理分析

## 发送方

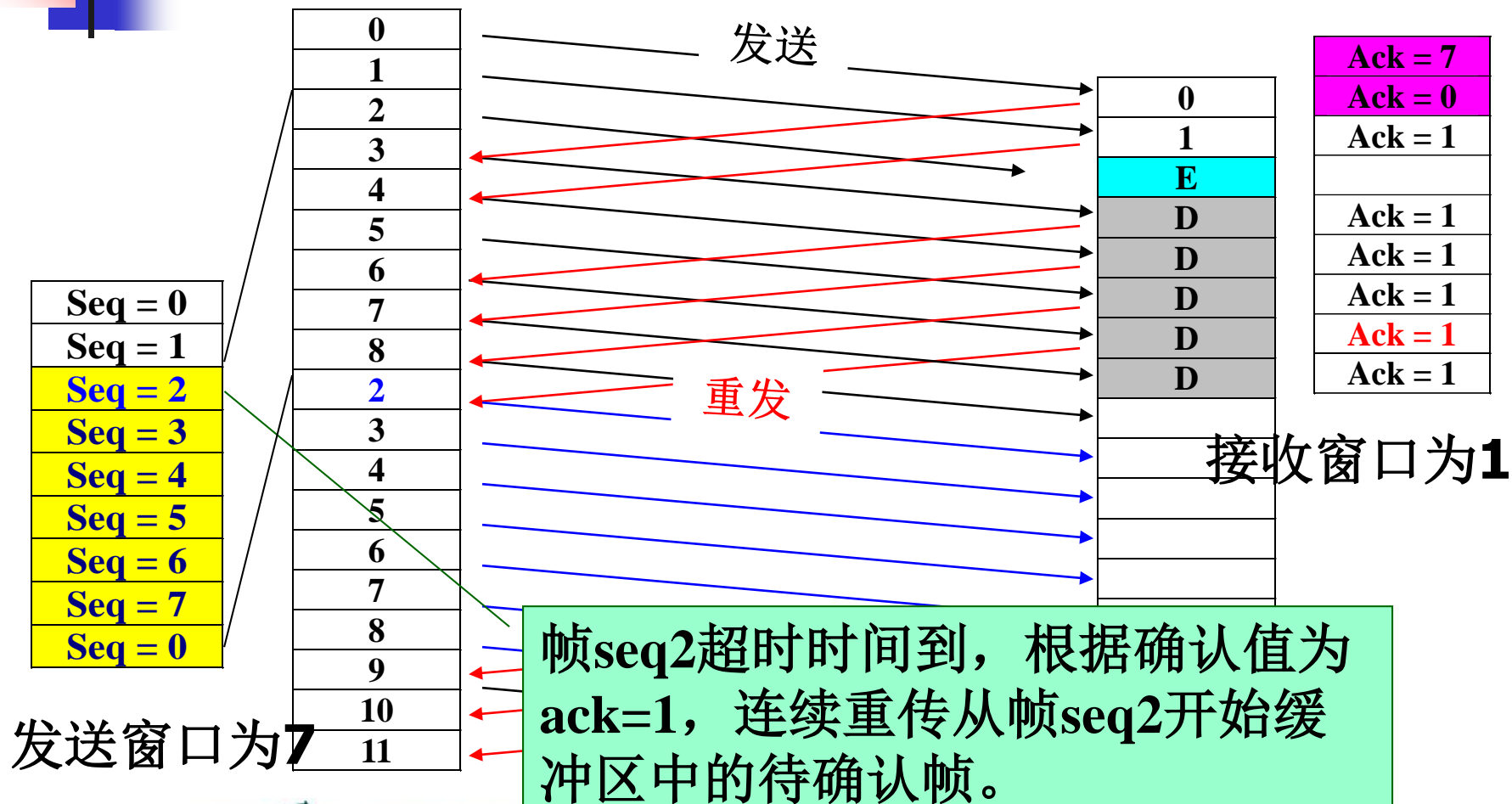
- 正常发送
  - 对帧编号，待确认帧缓存
- 收到确认
  - 释放确认帧所占缓冲区，滑动发送窗口
- 差错帧超时时间到
  - 回退到超时帧（出错帧），顺序重传最后被确认帧以后的缓冲区中缓存的帧

## 接收方

- 收到每一个期望的正确帧
  - 上交网络层、回送确认
- 收到出错帧或非期望帧
  - 丢弃，回送对接收的最后正确帧的确认
- 收到重传帧（即为一个期望的正确帧）



# 关键步骤：帧seq2超时，回退7帧重传



# 构造帧 (P183)

**s.kind=data**

**s.info=buffer[frame\_nr]**

**s.seq=frame\_nr**

**s.ack=(frame\_expected+Max\_seq)%(Max\_seq+1)**

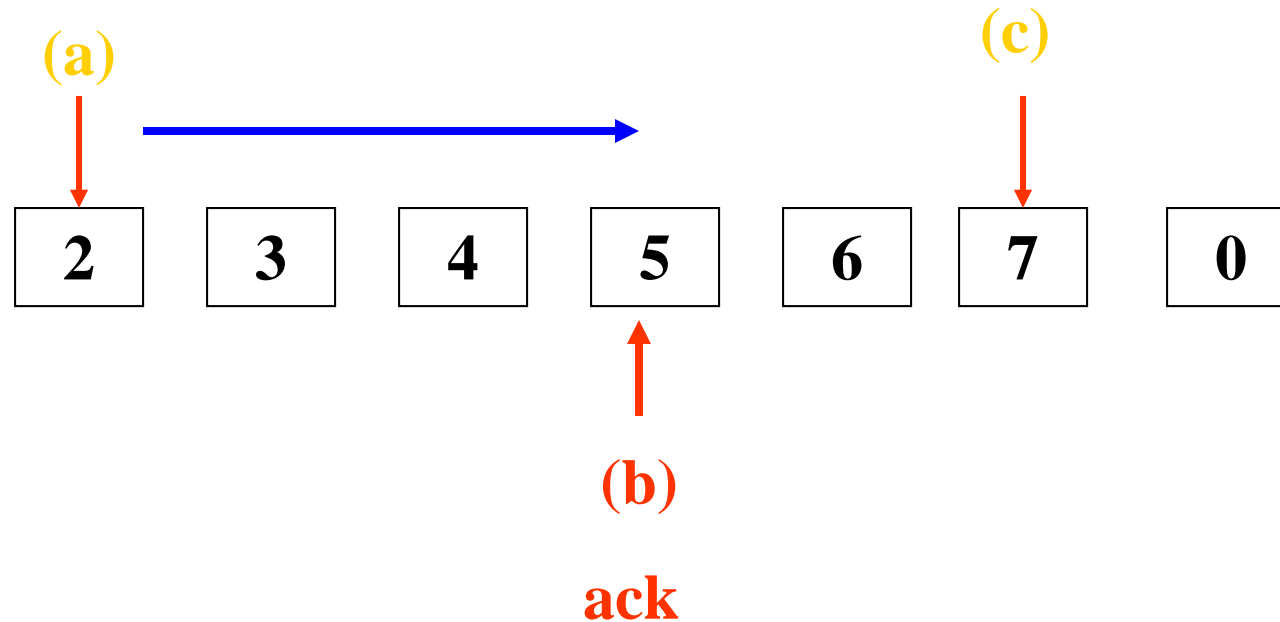


<b>F</b>	<b>A</b>	<b>C</b>	<b>N(S)</b>	<b>P/F</b>	<b>N(R)</b>	<b>Info.</b>	<b>FCS</b>	<b>F</b>
----------	----------	----------	-------------	------------	-------------	--------------	------------	----------

between (seq\_nr a, seq\_nr b, seq\_nr c)

ack\_expected

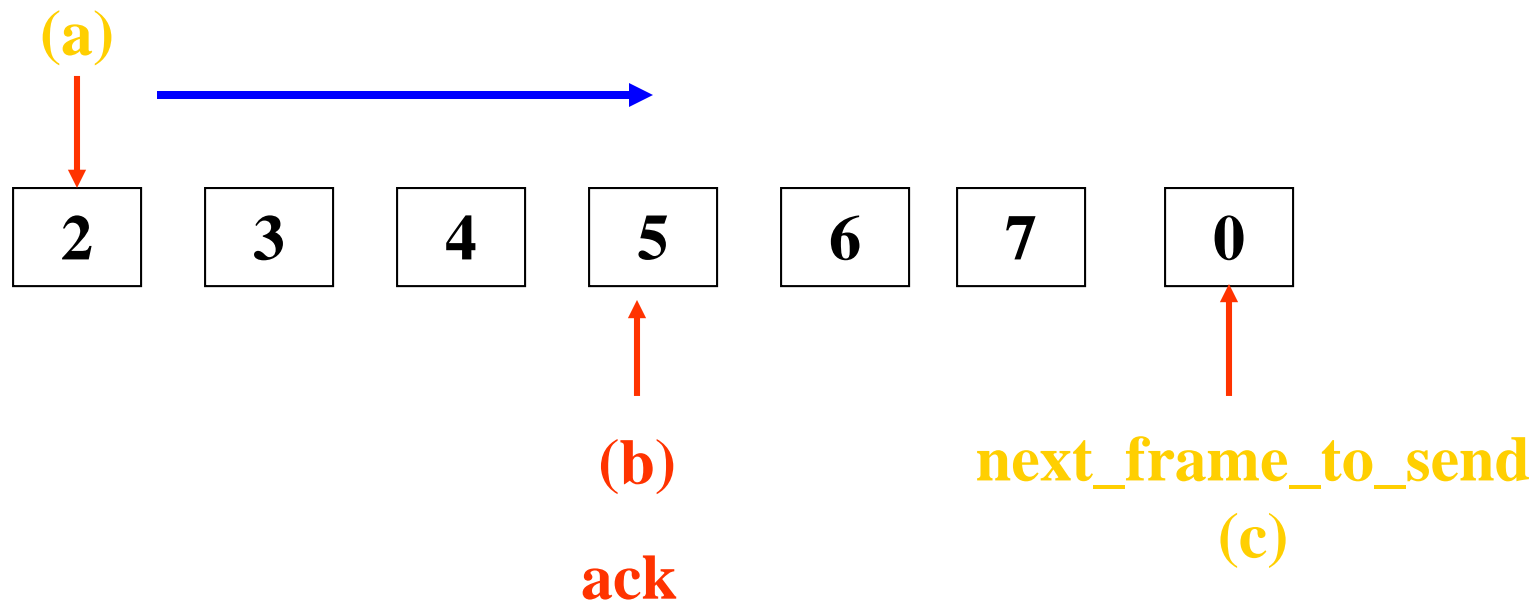
next\_frame\_to\_send



条件: ①  $(a \leq b) \ \&\& \ (b < c)$

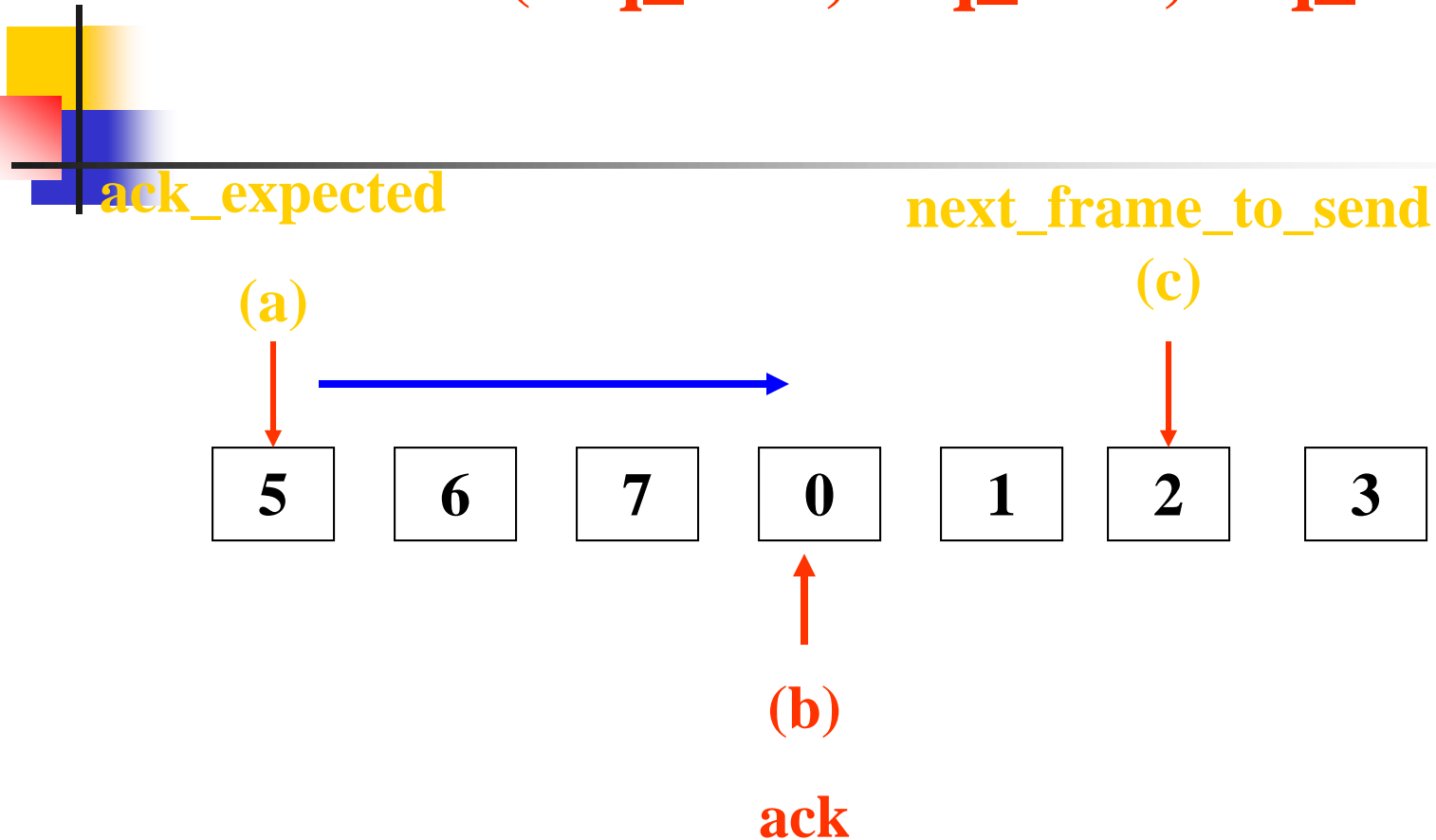
between (seq\_nr , seq\_nr , seq\_nr )

ack\_expected



条件: ②  $(a \leq b) \ \&\& \ (c < a)$

between (seq\_nr , seq\_nr , seq\_nr )



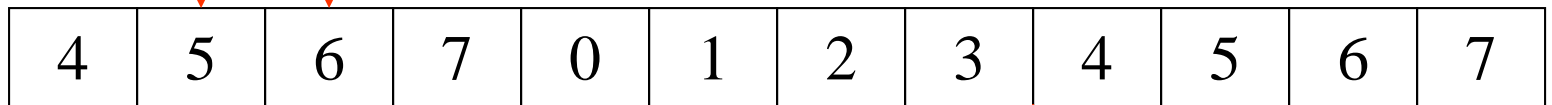
条件: ③  $(b < c) \ \&\& \ (c < a)$

## 图 3.19 - --P183

```
While (between(ack_expected ,r.ack, next_frame_to_send ))  
{  
    nbuffered= nbuffered-1    (发送窗口上边界)  
    stop_timer(ack_expected )  
    inc(ack_expected )        (发送窗口下边界)  
}
```

ack\_expected

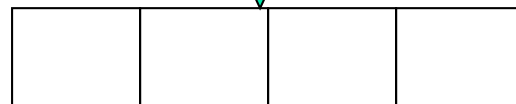
ack



对方已正常收到

已发送对方还未确认

入重发缓冲区

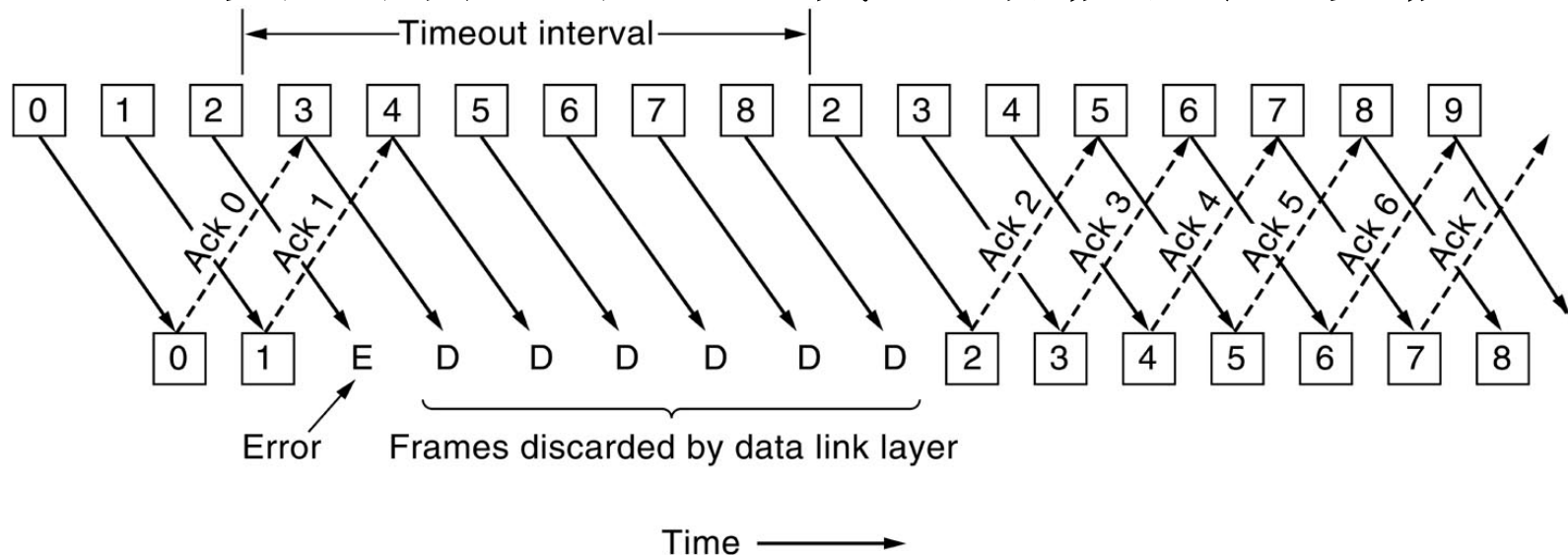


MAX\_SEQ

窗口大小

# 协议5回退n帧小结

- 接收窗口  $w=1$
- 接收方遇到出错帧，怎么办？（D&ack）
- 发送方超时，重传出错帧及后续帧







## 协议6：选择重传

---

接收方的接收策略选择：

丢弃错帧，缓存后续正确接收帧；

发送方的重传策略选择：

只重发出错帧。



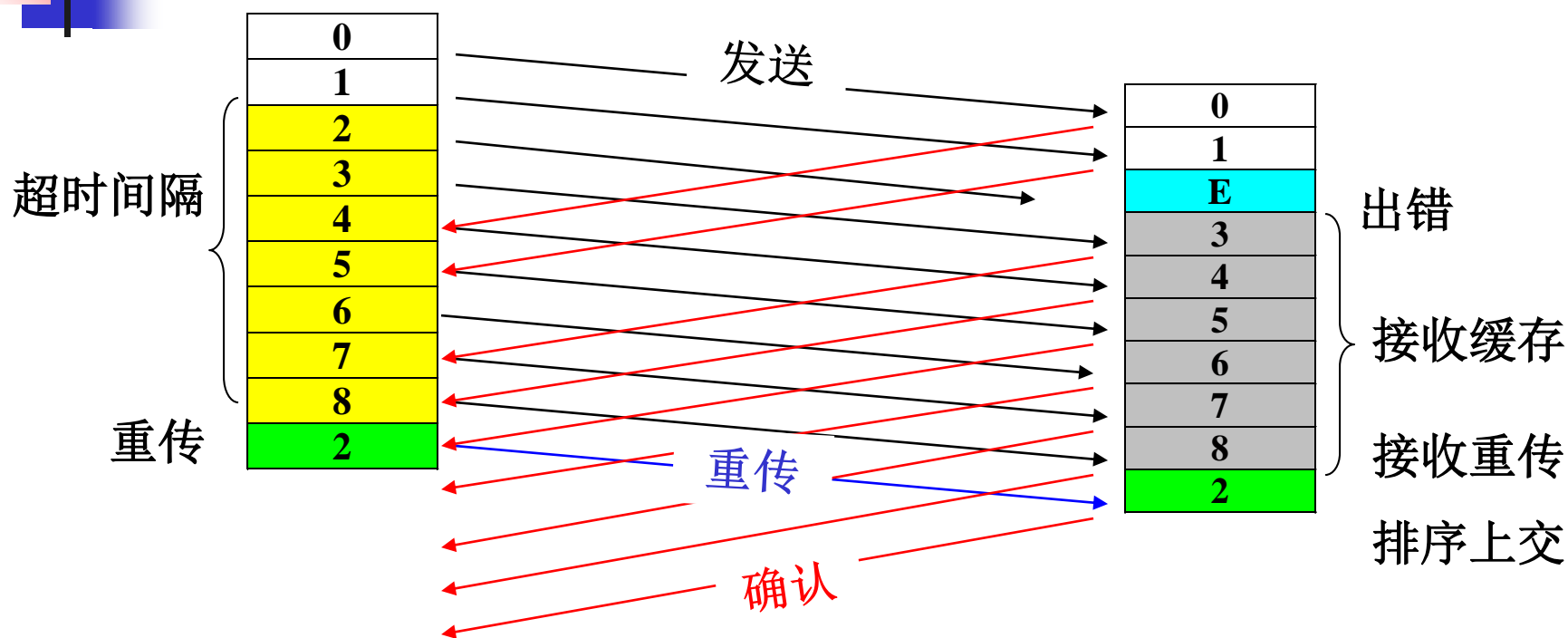
# 选择重传协议的基本概念

---

- 接收窗口存储差错帧后继的所有正确帧
- 发送方只重传差错帧
- 接收方接收重传帧，按正确顺序将分组提交网络层

举例（**MAX\_SEQ = 15**）

# 选择重传协议举例



Seq=0~15; W=8

# 选择重传协议的工作原理分析

## 发送方

- 正常发送
  - 对帧编号，待确认帧缓存
- 收到确认
  - 释放确认帧所占缓冲区，滑动发送窗口
- 差错帧超时时间到
  - 重传缓存的最后被确认帧以后的那一帧

## 接收方

- 正常接收
  - 上交网络层、回送确认，滑动接收窗口
- 收到非期望的正确帧
  - 缓存，回送对接收的最后正确帧的确认
- 收到重传帧
  - 将缓存帧排序上交，回送确认，滑动接收窗口

## 关键步骤:

## 接收方收到非期望的正确帧—缓存



发送

**Ack = 15**

## Ack = 0

**Ack = 1**

**Ack = 1**

## Ack = 1

**Ack = 1**

**Ack = 1**

**Ack = 1**

## 接收方缓冲区

2

3

4

5

6

---

**7**

8

0

南

D3

D4

D5

D6

D7

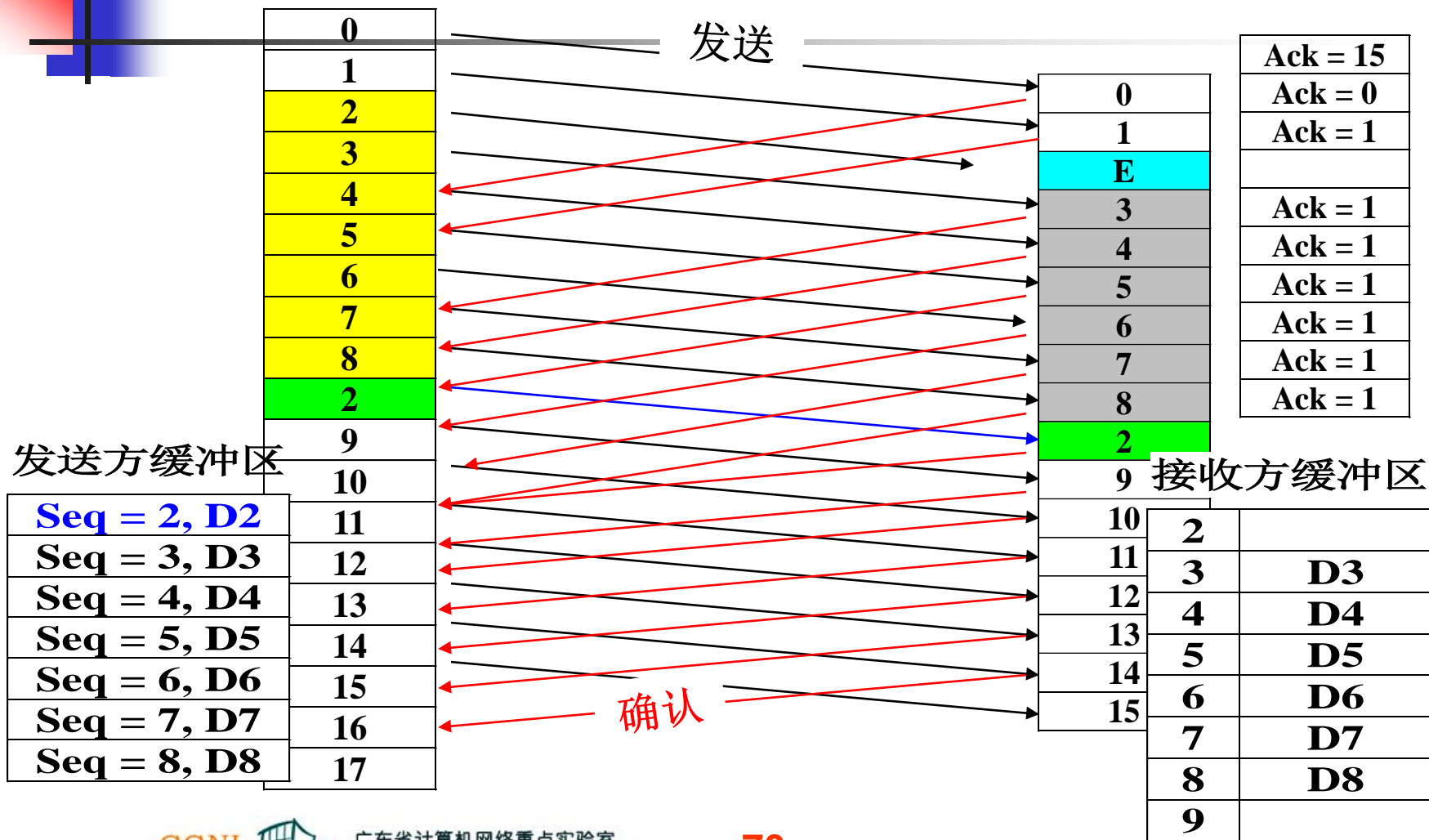
---

---

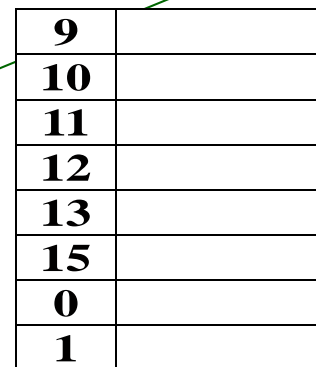
大雪

接收方期望接收seq2, 缓存其后的正确帧。

# 关键步骤： 发送方选择帧seq2重传



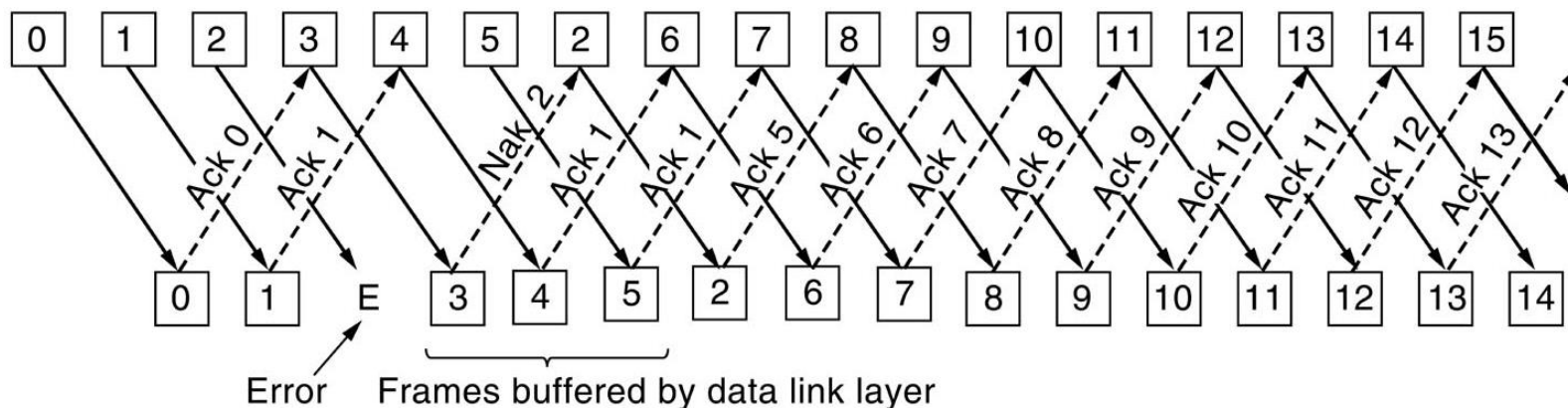
## 接收方收到重传帧seq2—排序上交



将帧seq2和接收方缓冲区中的帧正确排序，提交网络层，回送ack=8，滑动接收窗口。

# 协议6选择重传小结

- 接收方缓存
- 乱序帧，需正确排序





# 注意：NAK/ 累计确认的作用

P182

## ■ 否定确认NAK

- 加快出错帧的重传
- 对出错帧回送否定确认，使发方不再等到超时再重传

## ■ 累计确认 P184

- $n$ 号帧的确认到达时，暗含一个意思： $n-1$ 、 $n-2$ .....等 $n$ 号帧之前的帧也被确认

# 差错控制策略比较 和滑动窗口大小的选择





# 差错控制策略比较

---

## ■ 回退n帧

- 发送方需要较大的缓冲区，以便重传
- 重传帧数多，适于信道出错率较少的情况

## ■ 选择重传

- 接收方需要较大的缓冲区，以便按正确顺序将分组提交网络层
- 重传帧数少，适于信道质量不好的情况

# 滑动窗口长度 $w$ 的选择

- 协议5（回退 $n$ 帧）

- $\text{MAX\_SEQ} = 7$ （ $\text{Seq}=0 \sim \text{MAX\_SEQ}$ ）

- $W = 7$

发送窗口： $W = \text{MAX\_SEQ}$

- 协议6（选择重传）

- 教材举例

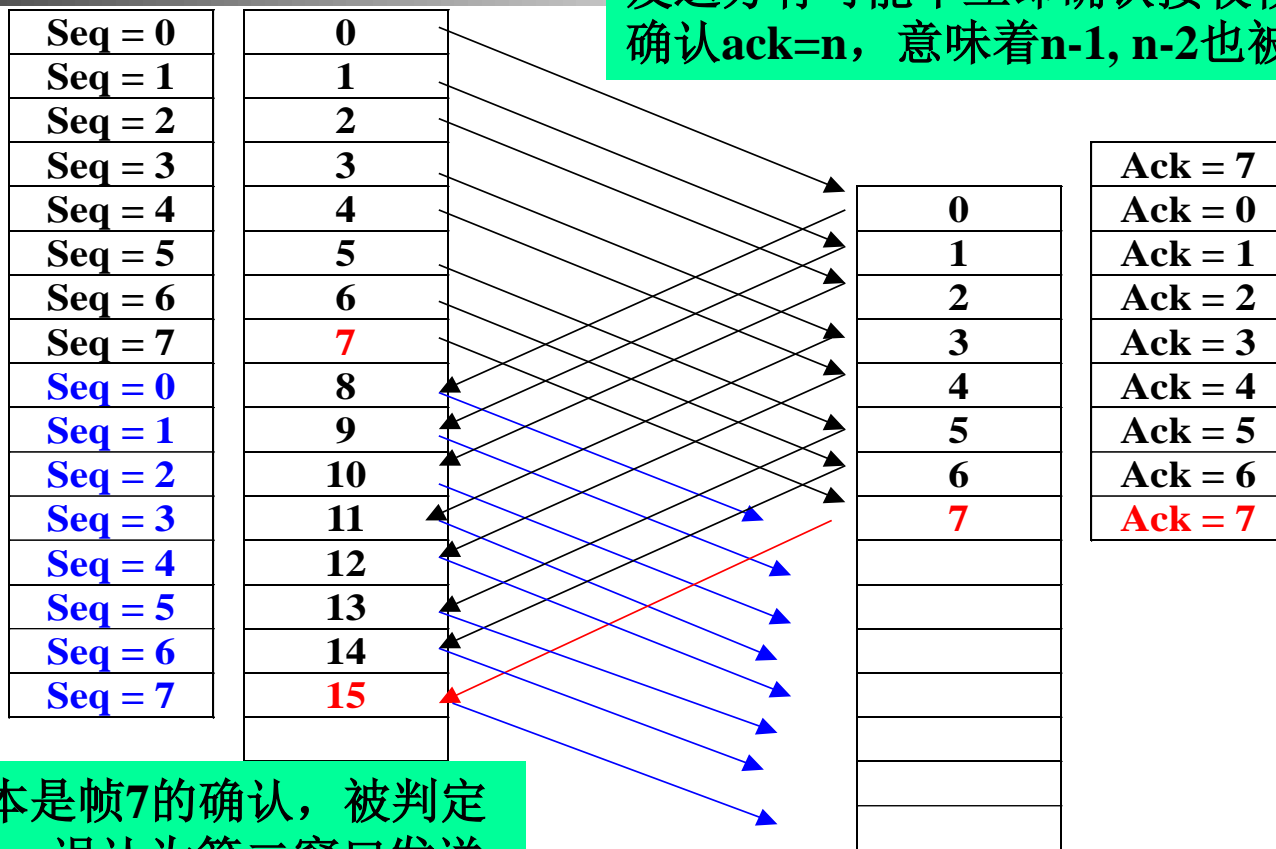
- $\text{MAX\_SEQ} = 7$ （ $\text{Seq}=0 \sim \text{MAX\_SEQ}$ ）

- $W = 4$

接收窗口： $W = (\text{MAX\_SEQ} + 1) / 2$ ； 发送窗口小于接收窗口

# 协议5: $W = 8$ , 异常情况P184

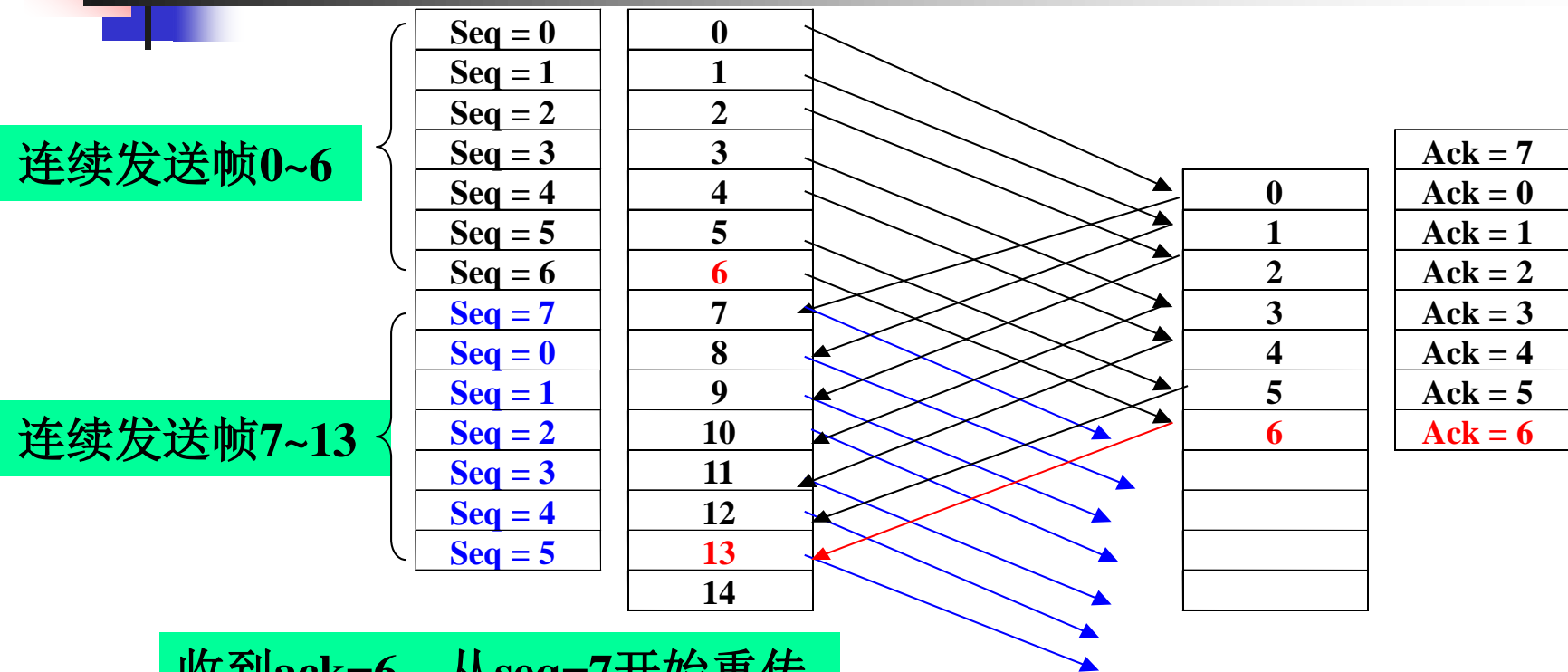
发送方有可能不立即确认接收帧, 捎带确认ack=n, 意味着n-1, n-2也被确认。



收到ack=7, 本是帧7的确认, 被判定为帧15的确认, 误认为第二窗口发送成功, 开始发送后续帧。

第二个窗口发送的数据帧全部出错丢弃。

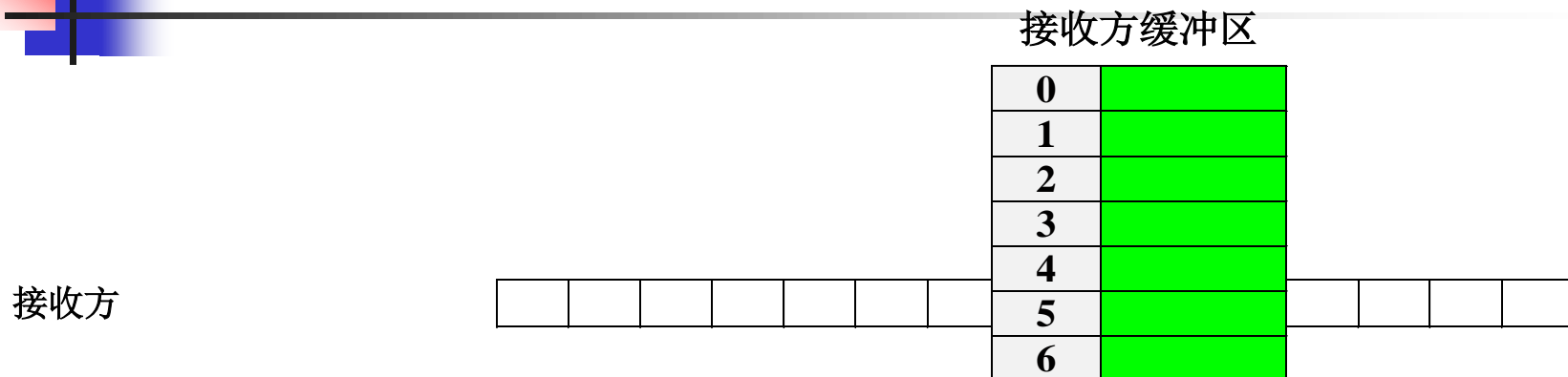
# 协议5: $W = 7$ , 异常情况



收到ack=6, 从seq=7开始重传第二窗口的数据帧, 不会误认为第二窗口发送成功。

第二个窗口发送的数据帧全部重发。

# 协议6: $W=7$ , 初始缓冲区空



<b>7</b>	
<b>0</b>	
<b>1</b>	
<b>2</b>	
<b>3</b>	
<b>4</b>	
<b>5</b>	

帧0~6上交网络层，回送确认，接收窗口滑动。

[illegible]

## 帧0~6的确认丢失!

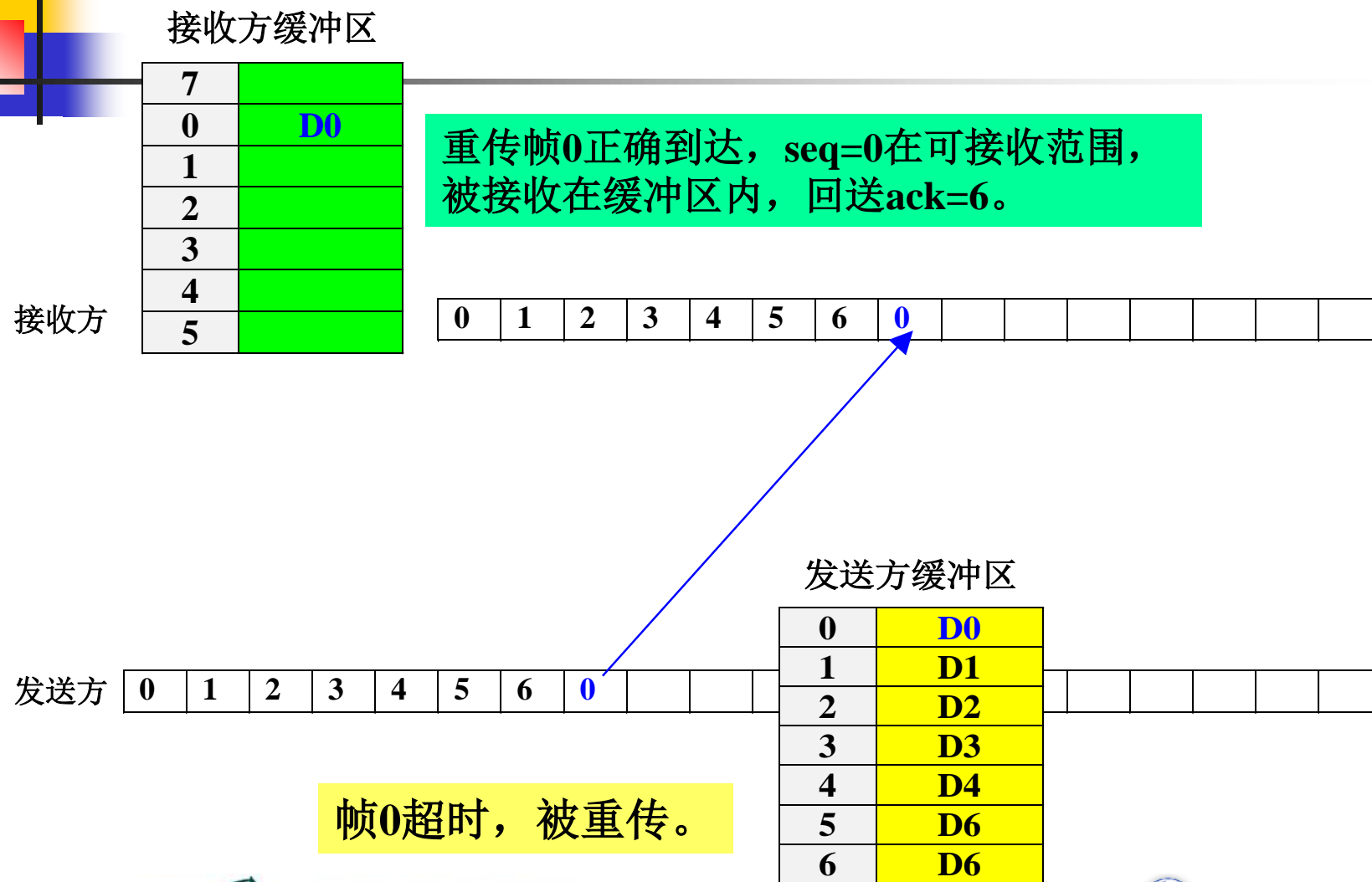
<b>0</b>	<b>D0</b>
<b>1</b>	<b>D1</b>
<b>2</b>	<b>D2</b>
<b>3</b>	<b>D3</b>
<b>4</b>	<b>D4</b>
<b>5</b>	<b>D6</b>
<b>6</b>	<b>D6</b>

0	1	2	3	4	5	6			
---	---	---	---	---	---	---	--	--	--

发送帧0~6等待确认。



# 协议6： 帧0超时重传并被正确接收



# 协议6： 帧0被重复提交

接收方缓冲区

7	D7
0	D0
1	
2	
3	
4	
5	

接收方

第7帧正确提交，重传帧0被认为是正确帧提交，出现重复提交错误。回发对第6帧的确认

0	1	2	3	4	5	6	0	7	8	9	10	11	12	13
---	---	---	---	---	---	---	---	---	---	---	----	----	----	----

发送方

0	1	2	3	4	5	6	0	7	8	9	10	11	12	13			
---	---	---	---	---	---	---	---	---	---	---	----	----	----	----	--	--	--

发送方缓冲区

7	D7
0	D8
1	D9
2	D10
3	D11
4	D12
5	D13

收到第6帧的确认，认为0~6被正确接收，发送第7~13帧。

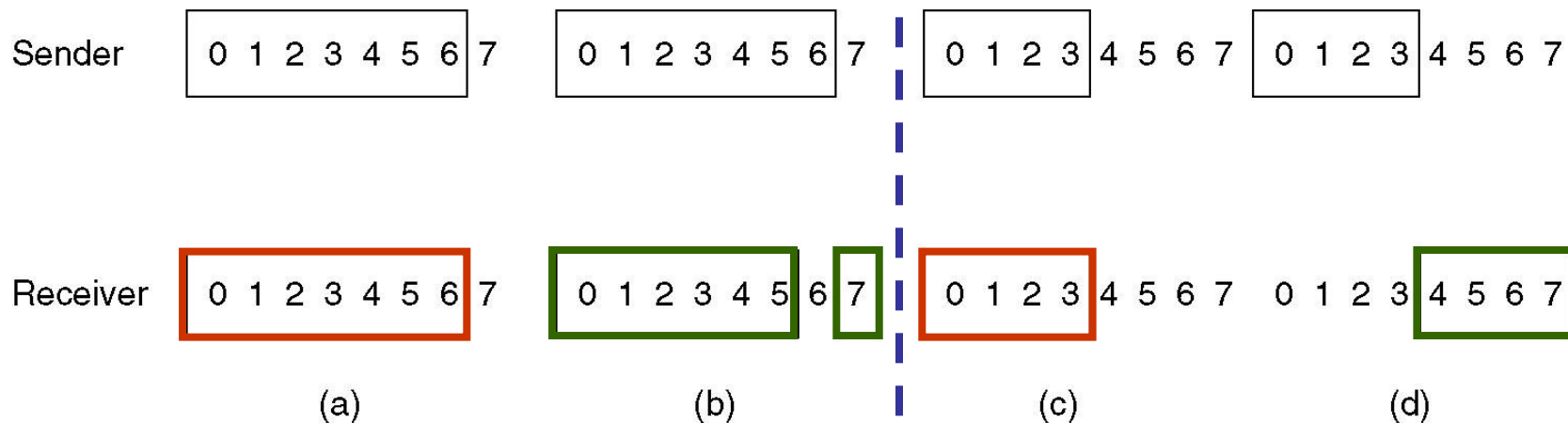
# 解决办法：保证新老窗口不重叠

P188

MAX\_SEQ=7

$W = 7$

$W = 4$



新老窗口重叠

新老窗口不重叠

# 协议6: $W=(MAX\_SEQ+1)/2$

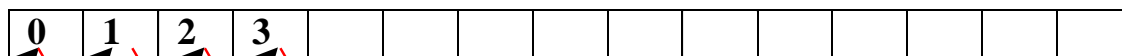
帧0~3的重传帧落在接收窗口外，被拒绝，不会出现重复提交错误。

接收方缓冲区

4	
5	
6	
7	

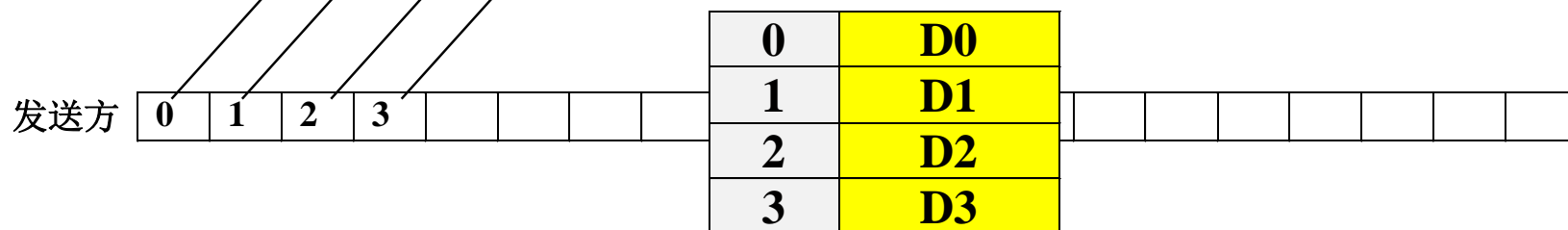
帧0~3上交网络层，回送确认，接收窗口滑动。

接收方



帧0~3的确认丢失!

发送方缓冲区



发送帧0~3等待确认。

GD



# 3个协议的窗口大小

---

- **One-Bit sliding window (协议4):**
  - $0 \leq \text{size of Sending window} \leq 1$
  - $\text{size of receiving window} = 1$
- **Go-back-N (协议5):**
  - $0 \leq \text{size of Sending window} \leq \text{MAX\_SEQ}$
  - $\text{size of receiving window} = 1$
- **Selective Repeat (协议6):**
  - $0 \leq \text{size of Sending window} \leq (\text{MAX\_SEQ} + 1) / 2$
  - $\text{size of receiving window} = (\text{MAX\_SEQ} + 1) / 2$



# 本节小结

---

- 理解掌握**6**种基本的**DLL**协议
- 理解涉及到的相关问题，例如肯定确认重传技术、捎带确认、滑动窗口技术、管道技术



# 课堂练习1

- **1** 数据链路层最重要的作用就是：通过一些协议，在不太可靠的物理链路上实现（ ）数据传输。
- **2** 在数据链路层，数据的传送单位是（ ）。
- **3** 在计算机网络通信中，常采用（ ）方式进行差错控制。
- **4** 所谓（ ）就是不管所传数据是什么样的比特组合，都应当能够在链路上传送。
- **5** 物理层要解决（ ）同步的问题；数据链路层要解决（ ）同步的问题。
- **6** 所谓（ ）就是从收到的比特流中正确无误地判断出一个帧从哪个比特开始以及到哪个比特结束。



# 参考答案

---

- **1 可靠的**
- **2 帧**
- **3 检错重发**
- **4 透明传输**
- **5 比特、 帧**
- **6 帧同步**





## 课堂练习2

---

- 在停-等待协议中，应答帧为什么不需要序号？
- 由停止等待协议的工作原理可知：收方每收到一个正确的数据后，都立即向发方发送一个应答帧，发方只有收到上一个数据的确认帧后，才能继续发送下一帧。所以，在停止等待协议中，无须对应答帧进行编号。



## 课堂练习3

---

- 解释零比特填充法。
- 在**HDLC**的帧结构中，若在两个标志字段之间的比特串中，碰巧出现了和标志字段**7E**（为**6**个连续**1**加上两边各一个**0**）一样的比特组合，那么就会误认为是帧的边界。为了避免出现这种情况，**HDLC**采用零比特填充法使一帧中不会出现**6**个连续**1**。

# 课堂练习4

- 数据链路(逻辑链路)与链路(物理链路)有何区别？
- 物理链路:就是一条无源的点到点的物理线路段，中间没有任何其他的交换结点。在进行数据通信时，两个计算机之间的通路往往是由许多的链路串接而成的。

逻辑链路:在物理线路之外，加上一些必要的规程来控制这些数据的传输。实现这些规程的硬件和软件加到链路上，就构成了逻辑链路。

## 课堂练习5

- 两台主机之间的数据链路层采用了回退n帧协议（**GBN**）传输数据，数据传输速率为**16kbps**，单向传播延迟是**270ms**，数据帧长度为**128~512B**，接收方总是以数据帧等长的帧进行确认，为使得信道利用率达到最高，帧序号的比特数至少为多少位？（**2012**考研真题）



# 参考答案

---

■ **A. 5**

■ **B. 4**

■ **C. 3**

■ **D. 2**

# 解答1

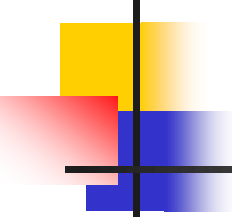
- 发送 **128B**长度的帧，传输时间是 $128 \times 8 / 16 = 64\text{ms}$ ，传输应答确认帧的时间也是**64ms**，那么从发送一帧到接收到确认帧所需要的总时间是 $64 + 270 + 64 + 270 = 668\text{ms}$ ，于是为使信道利用率最高， $= W \times 64 / 668 = 100\%$ ，so  **$W = 10.4$** . ( **$W$**  是窗口数)
- 同样地，传输**512B**帧长帧时， **$W = 4.1$** 。
- 窗口数跟所需序列号有关，是为了区分不同的帧， **$W = 10.4$**  需要**4bit**来表达，而 **$W = 4.1$** ，仅需**3bit**来表示。所以，答案应该是最坏的情形**4bit**。



## 解答2

---

- 根据教材的解释，使信道利用率最大的窗口数应该这样计算： **$w=2BD+2$**
- 所以，当帧长为**128B**时，  
 **$w=2*270ms*16kbps/（128B*8）+2=10.4$**
- 当帧长为**512B**时， **$w=2*270ms*16kbps/（512B*8）+2=4.1$**



---

# 谢谢!

