第三章 数据链路层

第4讲 滑动窗口协议

东南大学仪器科学与工程学院

主讲：陈熙源

滑动窗口的概念

在发送完一个数据帧后，不是停下来等待确认帧，而是可以连续再发送**若干个**数据帧。

但是，如果发送端一直没有收到对方的确认信息，那么实际上发送端**并不能无限制**的发送其数据。 这是因为？

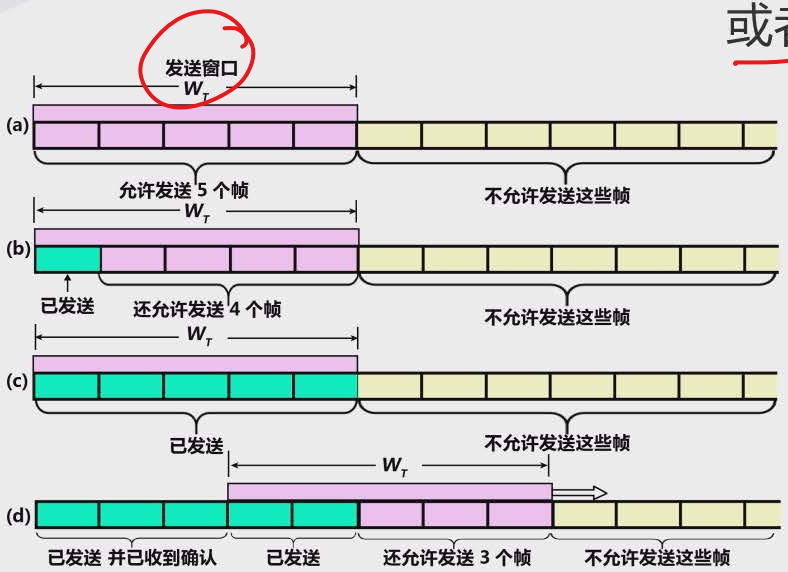
当未被确认的数据帧的数目太多时，只要有一帧出了差错，就要有很多的数据帧需要重传，这必然浪费很多时间。

因此，必须将已发送出去、但未被确认的数据帧的数目加以限制（**滑动窗口**）

滑动窗口协议在效率、复杂性和缓冲区需求等各个方面有所不同。

滑动窗口协议

发送方窗口内的序列号：代表已经被发送，但是还没有被确认的帧，或者是那些可以被发送的帧。

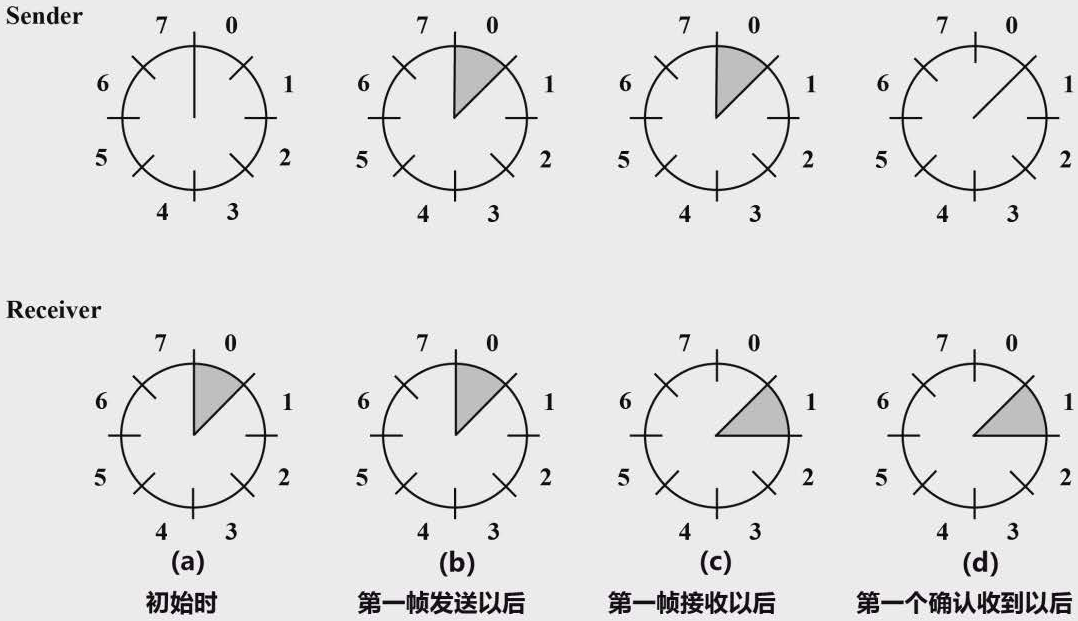


发送端的发送窗口：没有收到对方确认信息的情况下，可连续发送的数据帧的序号范围。

发送窗口的大小 W_T 表示：在还没有收到对方确认信息的情况下发送端最多可以发送多少个数据帧。

发送窗口用来对发送端进行流量控制。

滑动窗口协议



一个大小为1、有3位序列号的滑动窗口

1位滑动窗口协议

```
/* Protocol 4 (sliding window) is bidirectional. */
#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 0 or 1 only */
  seq_nr frame_expected;                            /* 0 or 1 only */
  frame r, s;                                       /* scratch variables */
  packet buffer;                                    /* current packet being sent */
  event_type event;
  next_frame_to_send = 0;                           /* next frame on the outbound stream */
  frame_expected = 0;                               /* frame expected next */
  from_network_layer(&buffer);                      /* fetch a packet from the network layer */
  s.info = buffer;                                  /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                       /* insert sequence number into frame */
  s.ack = 1 – frame_expected;                       /* piggybacked ack */
  to_physical_layer(&s);                            /* transmit the frame */
  start_timer(s.seq);                               /* start the timer running */
```

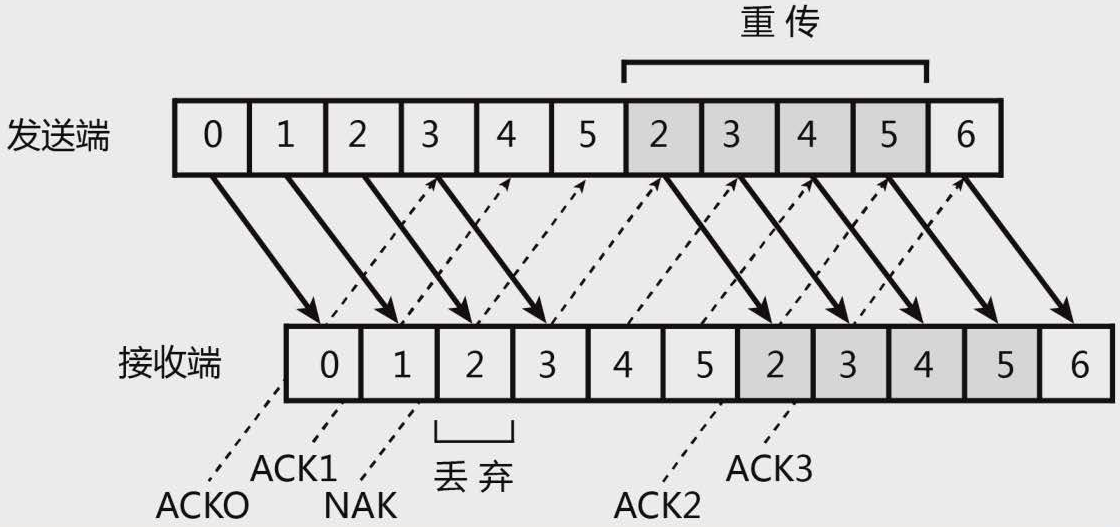
1位滑动窗口协议

```
while (true) {
    wait_for_event(&event);                  /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) {            /* a frame has arrived undamaged. */
        from_physical_layer(&r);             /* go get it */

        if (r.seq == frame_expected) {       /* handle inbound frame stream. */
            to_network_layer(&r.info);       /* pass packet to network layer */
            inc(frame_expected);             /* invert seq number expected next */
        }

        if (r.ack == next_frame_to_send) {   /* handle outbound frame stream. */
            stop_timer(r.ack);               /* turn the timer off */
            from_network_layer(&buffer);     /* fetch new pkt from network layer */
            inc(next_frame_to_send);         /* invert senderís sequence number */
        }
    }
    s.info = buffer;                         /* construct outbound frame */
    s.seq = next_frame_to_send;              /* insert sequence number into it */
    s.ack = 1 – frame_expected;              /* seq number of last received frame */
    to_physical_layer(&s);                   /* transmit a frame */
    start_timer(s.seq);                      /* start the timer running */
  }
}
```

使用回退n帧技术的协议

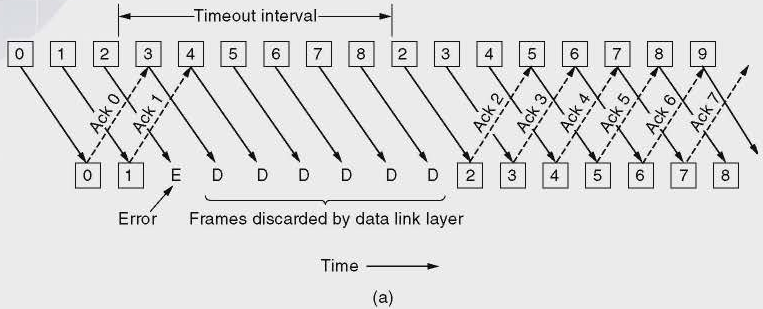


当出现差错必须重传时， 要向回走 N 个帧， 然后再开始重传。

在GBN协议中，接收窗口的大小 $WR = 1$

回退N协议

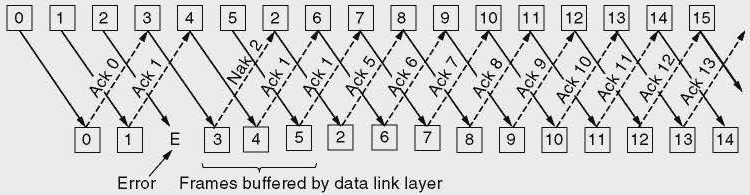
管道化技术与错误修复



(a)

在两种情况下错误的影响：

(a)接收方的窗口尺寸为1：接收方丢弃所有后续的帧，并且不为这些丢弃的帧发送确认

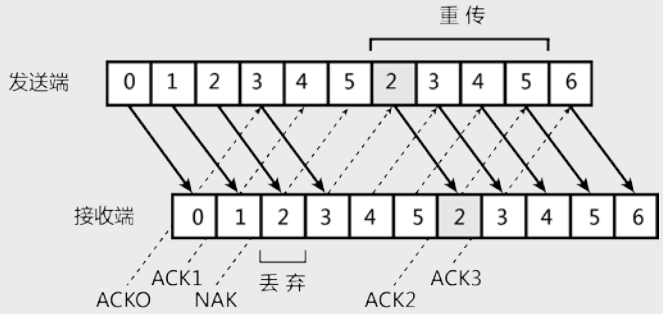


(b)

(b)接收方的窗口尺寸较大：窗口内的任何一帧都能够被接受，并被缓存起来，等到所有此前的帧都到达之后，再传递给网络层。

使用选择性重传的协议

处理错误策略二：选择性重传的协议



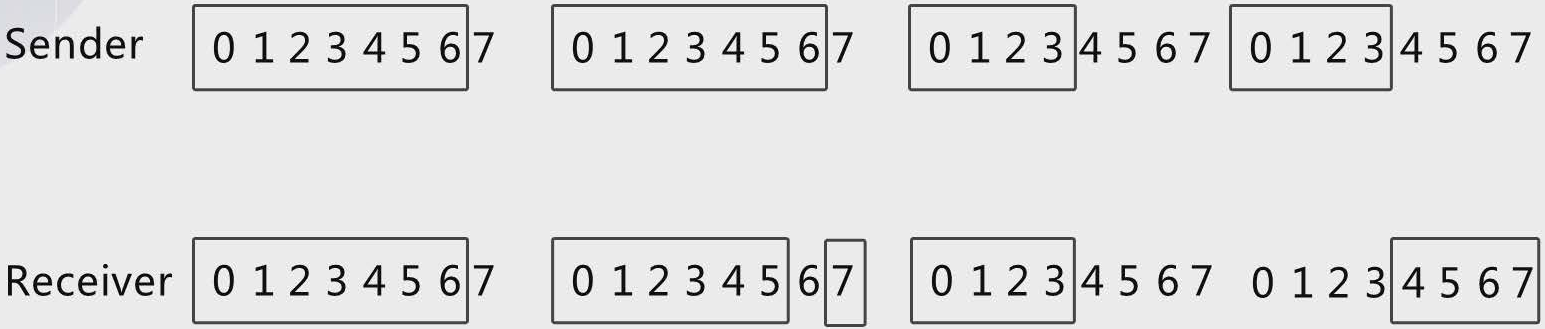
接收窗口>1

可加大接收窗口，先收下发送序号不连续但仍处在接收窗口中的那些数据帧。等到所缺序号的数据帧收到后再一并送交主机。

选择重传协议可避免重复传送那些本来已经正确到达接收端的数据帧。

但付出的代价是在接收端要设置具有相当容量的缓存空间。

使用选择性重传的协议



(a) 窗口大小为7的初始状态；

(b) 7帧都已送出并接受，但是均未被确认；

(c) 窗口大小为4的初始状态；

(d) 4帧都已送出并接受，但是均未被确认；

窗口尺寸=(接收方窗口大小+1)/2

作 业

想象一个滑动窗口协议，他的序号占用位数相当多，使得序号几乎永远不会回转。试问4个窗口边界和窗口大小之间必须满足什么样的关系？假设这里的窗口的大小固定不变，并且发送方和接收方的窗口大小相同。