Chap 3, THE DATA LINK LAYER

3. The following data fragment occurs in the middle of a data stream for which the byte-stuffing algorithm described in the text is used: A B ESC C ESC FLAG FLAG D. What is the output after stuffing?

答: A B ESC ESC C ESC ESC ESC FLAG ESC FLAG D.

7. Can you think of any circumstances under which an open-loop protocol (e.g., a Hamming code) might be preferable to the feedback-type protocols discussed throughout this chapter?

答：

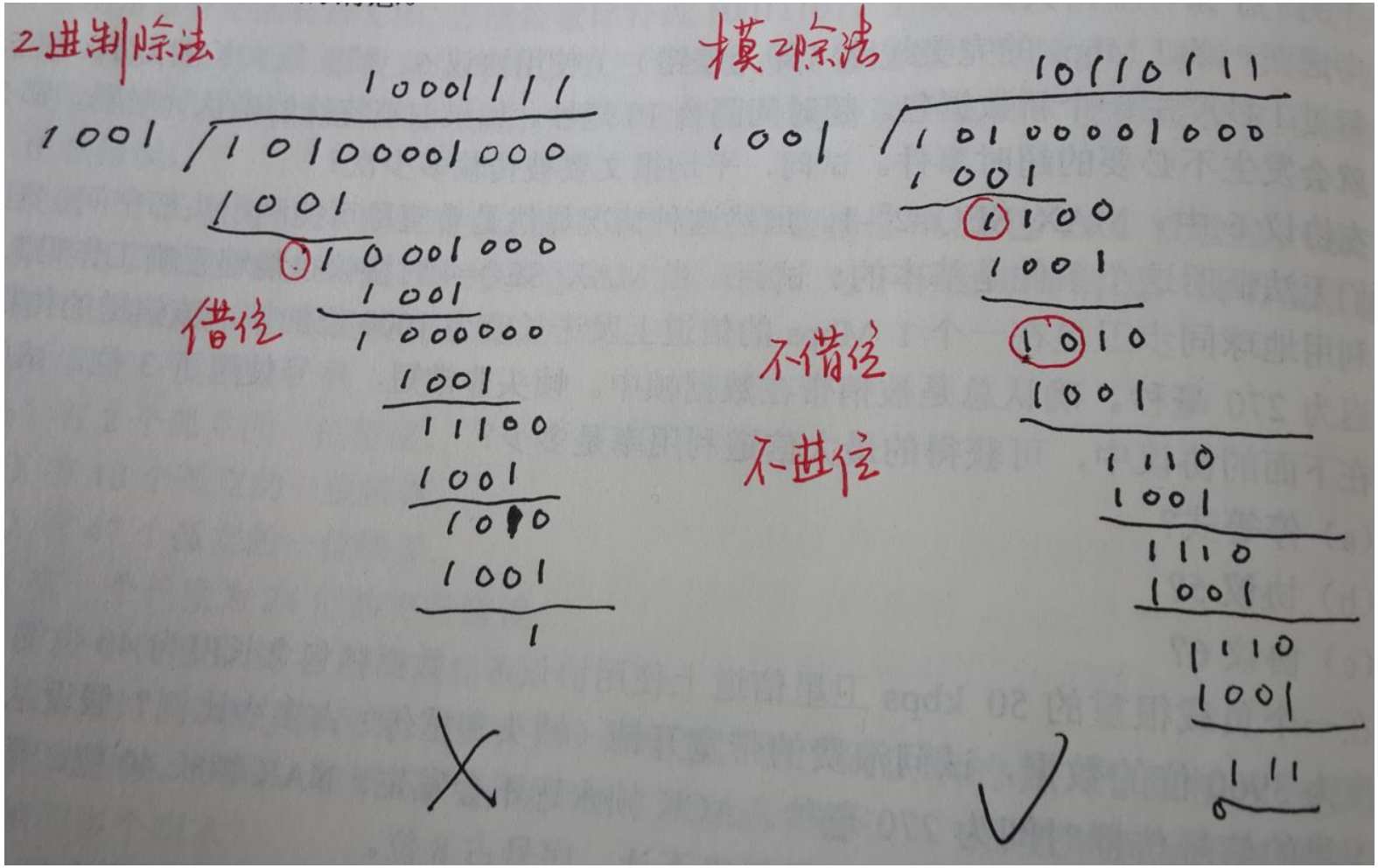
1）对于出错率较高但是每次只有很少码元（比如一个bit）出错的场合，如果使用检错重传的方法，重传的概率很高；此外重传的数据发生错误的概率依然很大，可能会导致反复的重传，这些过多的重传使信道的吞吐率下降。此时使用纠错码较好。

2）对于实时通信场合，使用纠错码还可减小出错重传造成的数据到达时延过高带来的服务质量的影响。

3）对于超远距离通信场合，如宇宙探测，往返时间过大，使用纠错码可以避免重传时延。

4）对于不希望重传以免因为交互而暴露发信方身份或者位置的场合，如军事和安全领域，使用纠错码更好。

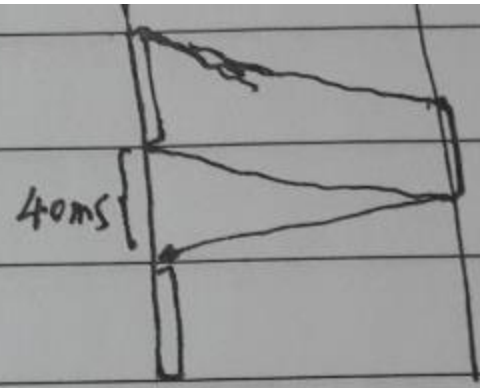
16. What is the remainder obtained by dividing $x^7 + x^5 + 1$ by the generator polynomial $x^3 + 1$?



$x^2 + x + 1$

20. A channel has a bit rate of 4 kbps and a propagation delay of 20 msec. For what range of frame sizes does stop-and-wait give an efficiency of at least 50%?

带宽：$4kbps = 4 \times 10^3$ bit/s



传输延时：20msec

50%的效率意味着传完一个帧后需要休息另一半的时间

采用停等协议，传输往返时间为40ms，说明帧的发送时间也是40ms，可知一个帧至少160 bits.

25. In protocol 6, when a data frame arrives, a check is made to see if the sequence number differs from the one expected and *no_nak* is true. If both conditions hold, a NAK is sent. Otherwise, the auxiliary timer is started. Suppose that the else clause were omitted. Would this change affect the protocol's correctness?

解答：

原程序是下面这一段：

```
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)) {
   ….
   start ack_timer(); /* to see if a separate ack is needed */
   }
```

这里else start ack timer() 对应(r.seq == frame expected) 或者no_nak = false；由于下面对于良序到达且非重复帧也会设置ack_timer，如果删除else，则有两种情况没有设置ack_timer：一是乱序但是no_nak是false，二是良序且重复。

情况一（乱序但是no_nak是false时不设ack_timer ）：
1.假设迄今一切正常（因此初始条件no_nak是true，ack_timer不设）；
2.接下来有一批数量等于滑窗大小的帧被接受，接收方接收时设的ack_timer到期后将发送ack帧，但是它们对应的ack丢失，timer到期后，发送方timeout事件将促使重发前面这批帧；
3.接受方发现这些帧不是期待的（确认后滑窗就转动过去了），因为步骤2中接收方接收时将no_nak已经设成true，所以接收方会send frame(nak, 0, frame expected, out buf)；
4.假设这个帧也丢失，发送方收不到确认timeout后只能再发；
5.由于no_nak已经变成false，接收方即便发现是旧帧，也不会再有任何其他动作**[注意如果没有删除else那句，这时会设ack_timer]**，而只是忽视它们（因为帧到达事件里头data帧对应的两种情况入口都不满足了）。
6.由于接受方滑窗已经完全过去，所以发送方发送窗口里头的帧都是旧帧，所以就死锁了。

```
  expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                                   /* should be 2^n – 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                              /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ + 1;                  /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol 5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;                                          /* scratch variable */

  s.kind = fk;                                      /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                                 /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;                    /* one nak per frame, please */
  to_physical_layer(&s);                            /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                                 /* no need for separate ack frame */
}
```

```
void protocol6(void)
{
  seq_nr ack_expected;                          /* lower edge of sender's window */
  seq_nr next_frame_to_send;                    /* upper edge of sender's window + 1 */
  seq_nr frame_expected;                        /* lower edge of receiver's window */
  seq_nr too_far;                               /* upper edge of receiver's window + 1 */
  int i;                                        /* index into buffer pool */
  frame r;                                      /* scratch variable */
  packet out_buf[NR_BUFS];                      /* buffers for the outbound stream */
  packet in_buf[NR_BUFS];                       /* buffers for the inbound stream */
  boolean arrived[NR_BUFS];                     /* inbound bit map */
  seq_nr nbuffered;                             /* how many output buffers currently used */
  event_type event;

  enable_network_layer();                       /* initialize */
  ack_expected = 0;                             /* next ack expected on the inbound stream */
  next_frame_to_send = 0;                       /* number of next outgoing frame */
  frame_expected = 0;
  too_far = NR_BUFS;
  nbuffered = 0;                                /* initially no packets are buffered */
  for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
  while (true) {
    wait_for_event(&event);                     /* five possibilities: see event_type above */
    switch(event) {
      case network_layer_ready:                 /* accept, save, and transmit a new frame */
           nbuffered = nbuffered + 1;           /* expand the window */
           from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
           send_frame(data, next_frame_to_send, frame_expected, out_buf);/* transmit the frame */
           inc(next_frame_to_send);             /* advance upper window edge */
           break;
```

```
case frame_arrival:                    /* a data or control frame has arrived */
    from_physical_layer(&r);           /* fetch incoming frame from physical layer */
    if (r.kind == data) {
        /* An undamaged frame has arrived. */
        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)) {
            /* Frames may be accepted in any order. */
            arrived[r.seq % NR_BUFS] = true;      /* mark buffer as full */
            in_buf[r.seq % NR_BUFS] = r.info;     /* insert data into buffer */
            while (arrived[frame_expected % NR_BUFS]) {
                /* Pass frames and advance window. */
                to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                no_nak = true;
                arrived[frame_expected % NR_BUFS] = false;
                inc(frame_expected);      /* advance lower edge of receiver's window */
                inc(too_far);             /* advance upper edge of receiver's window */
                start_ack_timer();        /* to see if a separate ack is needed */
            }
        }
    }
    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);

ack: while (between(ack_expected, r.ack, next_frame_to_send)) {
        nbuffered = nbuffered – 1;            /* handle piggybacked ack */
        stop_timer(ack_expected % NR_BUFS);   /* frame arrived intact */
        inc(ack_expected);                    /* advance lower edge of sender's window */
    }
    break;
```

```
        case cksum_err:
             if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* damaged frame */
             break;
        case timeout:
             send_frame(data, oldest_frame, frame_expected, out_buf); /* we timed out */
             break;
        case ack_timeout:
             send_frame(ack,0,frame_expected, out_buf);     /* ack timer expired; send ack */
    }
    if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
  }
}
```

Figure 3-21. A sliding window protocol using selective repeat.

26. Suppose that the three-statement while loop near the end of protocol 6 was removed from the code. Would this affect the correctness of the protocol or just the performance? Explain your answer.

解答：

原程序是下面这一段：

```
while (between(ack_expected, r.ack, next_frame_to_send)) {
        nbuffered = nbuffered – 1;              /* handle piggybacked ack */
        stop_timer(ack_expected % NR_BUFS);     /* frame arrived intact */
        inc(ack_expected);                      /* advance lower edge of sender's window */
}
```

这个while循环主要对到达的ack帧进行相应处理，如果删除，发送方就不知道下一个发送哪个帧，只会不断等timeout然后选择发送窗口里头的帧重复发送。

从而死锁停滞且不断浪费带宽资源。

32. Frames of 1000 bits are sent over a 1-Mbps channel using a geostationary satellite whose propagation time from the earth is 270 msec. Acknowledgements are always piggybacked onto data frames. The headers are very short. Three-bit sequence numbers are used. What is the maximum achievable channel utilization for

(a) Stop-and-wait?
(b) Protocol 5?
(c) Protocol 6?