Sliding Window Protocols

- What happens if a frame in the middle of a long stream is damaged or lost?
- Large numbers of succeeding frames will arrive at the receiver before the sender even finds out that anything is wrong.
- When a damaged frames arrives at the receiver, it obviously should be discarded, but what should the receiver do with all the corrected frames following it?
- The receiver data link layer is obligated to hand packets to the network layer in order.
- Two basic approaches are available for dealing with errors in the presence of pipelining
  - Go-Back-N
  - Selective Repeat

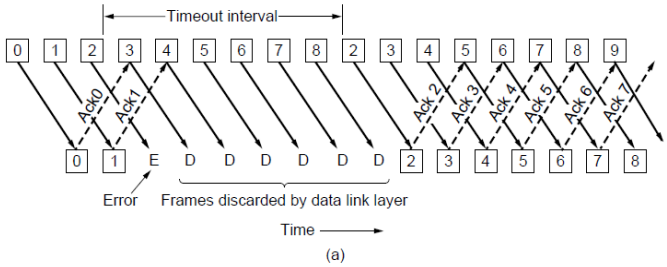
129

Go-back-n Sliding Window Protocols

- In Go-back-n, after receiving erroneous frame receiver simply discard all subsequent frames, and sending no acknowledgements for the discarded frames.
- This strategy corresponds to a receive window of size 1.
- In other words, the data link layer refuses to accept any frame except the next one it must give to the network layer.
- If the sender's window fills up before the timer runs out, the pipeline will begin to empty.
- Eventually, the sender will time out and retransmit all unacknowledged frames in order, starting with the damaged or lost one.
- This approach can waste a lot of bandwidth if the error rate is high.

130

Go-back-n Sliding Window Protocols



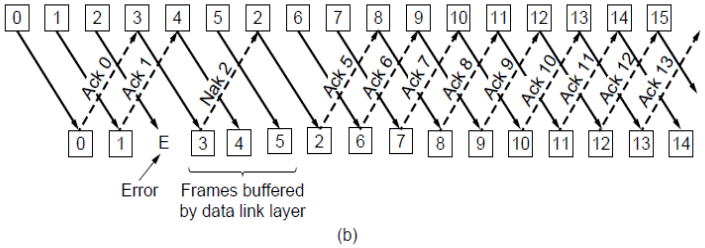
Pipelining and error recovery. Effect of an error when (a) receiver's window size is 1

- In Fig. (a), Frames 0 and 1 are correctly received and acknowledged. Frame 2, however, is damaged or lost.
- The sender, unaware of this problem, continues to send frames until the timer for frame 2 expires. Then it backs up to frame 2 and starts all over with it, sending 2, 3, 4, etc. all over again.

Selective repeat Sliding Window Protocols

- The other general strategy for handling errors when frames are pipelined is called selective repeat.
  - When it is used, a bad frame that is received is discarded, but good frames received after it are buffered.
  - When the sender times out, only the oldest unacknowledged frame is retransmitted. If that frame arrives correctly, the receiver can deliver to the network layer, in sequence, all the frames it has buffered.
  - Selective repeat corresponds to a receiver window larger than 1.
  - This approach can require large amounts of data link layer memory if the window is large.
- Selective repeat is often combined with having the receiver send a negative acknowledgement (NAK) when it detects an error, or a frame out of sequence.
- NAKs stimulate retransmission before the corresponding timer expires and thus improve performance.

Selective repeat Sliding Window Protocols



Pipelining and error recovery. Effect of an error when (b) receiver's window size is large.

- In Fig.(b), frames 0 and 1 are correctly received and acknowledged and frame 2 is lost.
- When frame 3 arrives at the receiver, the data link layer there notices that it has missed a frame, so it sends back a NAK for 2 but buffers 3.

133

Selective repeat Sliding Window Protocols

- When frames 4 and 5 arrive, they, too, are buffered by the data link layer instead of being passed to the network layer.
- Eventually, the NAK 2 gets back to the sender, which immediately resends frame 2.
- When that arrives, the data link layer now has 2, 3, 4, and 5 and can pass all of them to the network layer in the correct order.
- It can also acknowledge all frames up to and including 5, as shown in the figure (b).
- If the NAK get lost, eventually the sender will time out for frame 2 and send it (and only it) of its own accord, but that may be a quite a while later.
- In effect, the NAK speeds up the retransmission of one specific frame.

134

Sliding Window Protocols

- The go-back-n protocol works well if errors are rare.
- If the line is poor it wastes a lot of bandwidth on retransmitting frames.
- A Selective Repeat protocol allows the receiver to accept and buffer the frames following a damaged or lost one.
- These two alternative approaches are trade-offs between efficient use of bandwidth and data link layer buffer space.
- Depending on which resource is scarcer, one or the other can be used.

135

Protocol Using Go-Back-N (3)

```
/* Protocol 5 (Go-back-n) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike in the previous
   protocols, the network layer is not assumed to have a new packet all the time. Instead,
   the network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <= b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
       return(true);
   else
       return(false);
}
```

. . .

A sliding window protocol using go-back-n.

136

Protocol Using Go-Back-N (4)

```
static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Construct and send a data frame. */
  frame s;                                  /* scratch variable */

  s.info = buffer[frame_nr];                /* insert packet into frame */
  s.seq = frame_nr;                         /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);/* piggyback ack */
  to_physical_layer(&s);                    /* transmit the frame */
  start_timer(frame_nr);                    /* start the timer running */
}
```

. . .

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (5)

```
void protocol5(void)
{
  seq_nr next_frame_to_send;                /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;                      /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;                    /* next frame expected on inbound stream */
  frame r;                                  /* scratch variable */
  packet buffer[MAX_SEQ + 1];               /* buffers for the outbound stream */
  seq_nr nbuffered;                         /* number of output buffers currently in use */
  seq_nr i;                                 /* used to index into the buffer array */
  event_type event;
```

. . .

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (6)

```
enable_network_layer();                 /* allow network_layer_ready events */
ack_expected = 0;                       /* next ack expected inbound */
next_frame_to_send = 0;                 /* next frame going out */
frame_expected = 0;                     /* number of frame expected inbound */
nbuffered = 0;                          /* initially no packets are buffered */

while (true) {
  wait_for_event(&event);               /* four possibilities: see event_type above */

  . . .
```

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (7)

```
switch(event) {
  case network_layer_ready:             /* the network layer has a packet to send */
       /* Accept, save, and transmit a new frame. */
       from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
       nbuffered = nbuffered + 1;       /* expand the sender's window */
       send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
       inc(next_frame_to_send);         /* advance sender's upper window edge */
       break;

  case frame_arrival:                   /* a data or control frame has arrived */
       from_physical_layer(&r);         /* get incoming frame from physical layer */

       if (r.seq == frame_expected) {
            /* Frames are accepted only in order. */
            to_network_layer(&r.info);  /* pass packet to network layer */
            inc(frame_expected);        /* advance lower edge of receiver's window */
       }

    . . .
```

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (8)

```
            /* Ack n implies n – 1, n – 2, etc.  Check for this. */
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                  /* Handle piggybacked ack. */
                  nbuffered = nbuffered – 1;          /* one frame fewer buffered */
                  stop_timer(ack_expected);           /* frame arrived intact; stop timer */
                  inc(ack_expected);                  /* contract sender's window */
            }
            break;

      case cksum_err: break;                          /* just ignore bad frames */

      case timeout:                                   /* trouble; retransmit all outstanding frames */
            next_frame_to_send = ack_expected;        /* start retransmitting here */
            for (i = 1; i <= nbuffered; i++) {
                  send_data(next_frame_to_send, frame_expected, buffer);/* resend frame */
                  inc(next_frame_to_send);            /* prepare to send the next one */
            }

}
```

. . .

A sliding window protocol using go-back-n.

Protocol Using Go-Back-N (9)

```
            if (nbuffered < MAX_SEQ)
                  enable_network_layer();
            else
                  disable_network_layer();
      }
}
```

A sliding window protocol using go-back-n.

Maximum Outstanding Frames in Go-back-n

- For go-back-n, the maximum number of frames that may be outstanding at any instant is MAX_SEQ, even though there are MAX_SEQ + 1 distinct sequence numbers (which are 0, 1, . . . , MAX_SEQ).
- To see why this restriction is required, consider the following scenario with MAX_SEQ = 7:
  1. The sender sends frames 0 through 7.
  2. A piggybacked acknowledgement for 7 comes back to the sender.
  3. The sender sends another eight frames, again with sequence numbers 0 through 7.
  4. Now another piggybacked acknowledgement for frame 7 comes in.
- The question is this: did all eight frames belonging to the second batch arrive successfully, or did all eight get lost ?
- In both cases the receiver would be sending frame 7 as the acknowledgement.

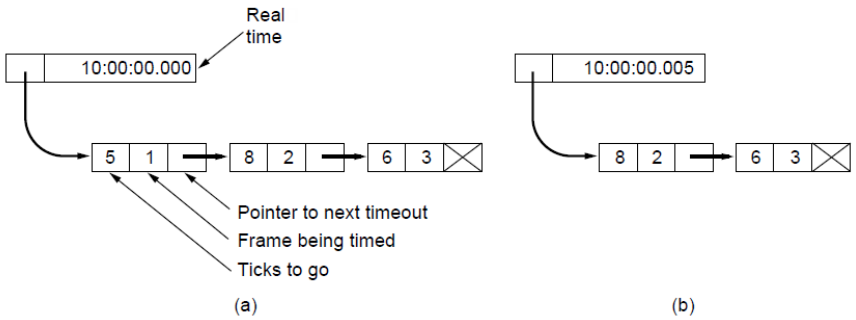
143

Cumulative Acknowledgment

- When an acknowledgment comes in for frame n, frames n-1, n-2, … are also automatically acknowledged.
- Because protocol 5 has multiple outstanding frames, it logically needs multiple timers, one per outstanding frame.
  - Each frame times out independently of all the other ones.
  - The pending timeouts form a linked list, with each node of the list containing the
    - Number of clock ticks until the timer expires,
    - Frame being timed,
    - A pointer to the next node

144

Protocol Using Go-Back-N (10)



Simulation of multiple timers in software. (a) The queued timeouts (b) The situation after the first timeout has expired.

Selective Repeat

- The go-back-n protocol works well if errors are rare.
- If the line is poor it wastes a lot of bandwidth on retransmitting frames.
- A Selective Repeat protocol allows the receiver to accept and buffer the frames following a damaged or lost one.

Selective Repeat

- Sender and Receiver maintain a window of outstanding and acceptable sequence numbers, respectively.
- Sender:
  - Window size starts a 0,
  - Grows to some predefined maximum,
- Receiver:
  - Is always fixed in size
  - Equal to the predetermined maximum.
  - Has a buffer reserved for each sequence number within its fixed window.
  - Associated with buffer is a bit (*arrived*) telling whether the buffer is full or empty.

147

Selective Repeat

- Whenever the frame arrives, the receiver does the following:
  - Check the frame sequence number if it does fall within its fixed window,
  - If this is the case and the sequence no. has not been received in previous transmissions it will be stored.
  - It does so without regard if the frame contains the next packet expected by the network layer.
  - This frame is going to be kept until all the lower-numbered frames have already been delivered to the network layer in the correct order.

148

Protocol Using Selective Repeat (1)

```
/* Protocol 6 (Selective repeat) accepts frames out of order but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   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));
}

  . . .
```

A sliding window protocol using selective repeat.

149

Protocol Using Selective Repeat (2)

```
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 */
}

  . . .
```

A sliding window protocol using selective repeat.

150

Protocol Using Selective Repeat (3)

```
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;

  . . .
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (4)

```
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;

. . .
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (5)

```
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;

  . . .
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (6)

```
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 */
  . . .
```

A sliding window protocol using selective repeat.

Protocol Using Selective Repeat (7)

```
            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 */
            }
        }
    }

    . . .
```

A sliding window protocol using selective repeat.

155

Protocol Using Selective Repeat (8)

```
    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);

    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;

    . . .
```

A sliding window protocol using selective repeat.

156

Protocol Using Selective Repeat (9)

```
        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();
  }
}
```

A sliding window protocol using selective repeat.

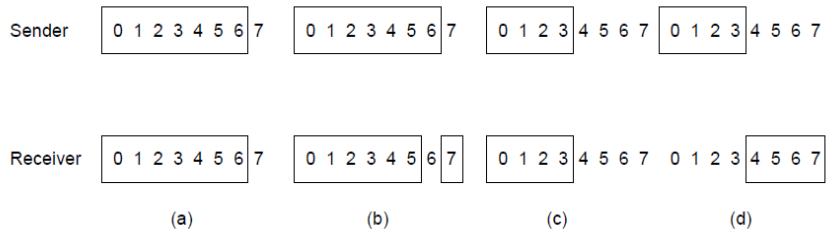
157

Selective Repeat

- Example:
  - 3-bit sequence number,
  - 7 frames are permitted to be send by transmitter, before it is required to wait (for acknowledgment).
- Window size restriction:
  - Must be at most half the range of the sequence numbers.
  - Window size for protocol 6 is (MAX_SEQ+1)/2.

158

Protocol Using Selective Repeat (10)



a) Initial situation with a window of size7
b) After 7 frames sent and received but not acknowledged.
c) Initial situation with a window size of 4.
d) After 4 frames sent and received but not acknowledged.

159

Selective Repeat

- If there is a lot of traffic in one direction and no traffic in the other direction, the protocol will block when the sender window reaches its maximum.
- Auxillary timer is started by start_ack_timer after an in-sequence data frame arrives.
  - If no reverse traffic has presented itself for piggybacking before the timer expires, a separate acknowledgement will be send.
  - Interrupt due to this timer is called ack_timeout event.

160

High Level Data Link Control (HDLC)

- HDLC is an important data link control protocol
- Specified as ISO 33009, ISO 4335.
- HDLC defines: three station types:
  - **Primary station:** Responsible for controlling the operation of the link. Frames issued by the primary are called commands.
  - **Secondary station:** Operates under the control of the primary station. Frames issued by a secondary are called responses. The primary maintains a separate logical link with each secondary station on the line.
  - **Combined station:** Combines the features of primary and secondary. A combined station may issue both commands and responses.

162

High Level Data Link Control (HDLC)

- It also defines two link configurations:
  - **Unbalanced configuration:** Consists of one primary and one or more secondary stations and supports both full-duplex and half-duplex transmission.
  - **Balanced configuration:** Consists of two combined stations and supports both full-duplex and half-duplex transmission.

163

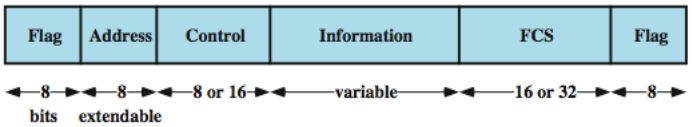
HDLC Transfer Modes

- HDLC defines three data transfer modes:
- Normal Response Mode (NRM)
  - Used with unbalanced configuration, primary initiates transfer
  - used on multi-drop lines, eg host + terminals
- Asynchronous Balanced Mode (ABM)
  - Used with balanced configuration, either station initiates transmission, widely used, more efficient use of a full-duplex point-to-point link because of no polling overhead,
- Asynchronous Response Mode (ARM)
  - unbalanced configuration, secondary may initiate transmit without permission from primary, rarely used

164

HDLC Frame Structure

- HDLC uses synchronous transmission of frames
- All transmissions are in the form of frames, and a single frame format suffices for all types of data and control exchanges.



(a) Frame format

165

Flag Fields and Bit Stuffing

- delimit frame at both ends with 01111110 seq
- receiver hunts for flag sequence to synchronize
- bit stuffing used to avoid confusion with data containing flag seq 01111110
  - 0 inserted after every sequence of five 1s
  - if receiver detects five 1s it checks next bit
  - if next bit is 0, it is deleted (was stuffed bit)
  - if next bit is 1 and seventh bit is 0, accept as flag
  - if sixth and seventh bits 1, sender is indicating abort

Original Pattern:

111111111111011111101111110

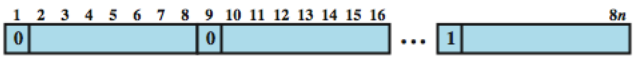
After bit-stuffing

111110**0**11111**0**11011111**0**1011111**0**10

166

Address Field

- identifies secondary station that sent or will receive frame
- usually 8 bits long
- may be extended to multiples of 7 bits
  - LSB indicates if is the last octet (1) or not (0)
- all ones address 11111111 is broadcast

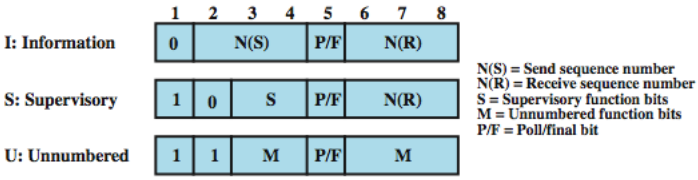


(b) Extended Address Field

167

Control Field

- different control field format for different frame type
  - Information - data transmitted to user (next layer up)
    - Flow and error control piggybacked on information frames
  - Supervisory - ARQ when piggyback not used
  - Unnumbered - supplementary link control functions
- first 1-2 bits of control field identify frame type

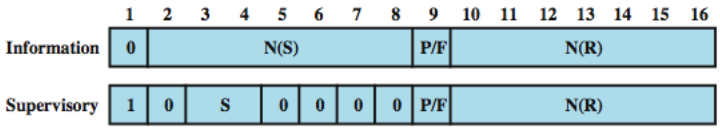


(c) 8-bit control field format

168

Control Field

- use of Poll/Final bit depends on context
- in command frame is P bit set to1 to solicit (poll) response from peer
- in response frame is F bit set to 1 to indicate response to soliciting command
- seq number usually 3 bits
  - can extend to 8 bits as shown below



(d) 16-bit control field format

169

Information & FCS Fields

- Information Field
  - in information and some unnumbered frames
  - must contain integral number of octets
  - variable length
- Frame Check Sequence Field (FCS)
  - used for error detection
  - either 16 bit CRC or 32 bit CRC

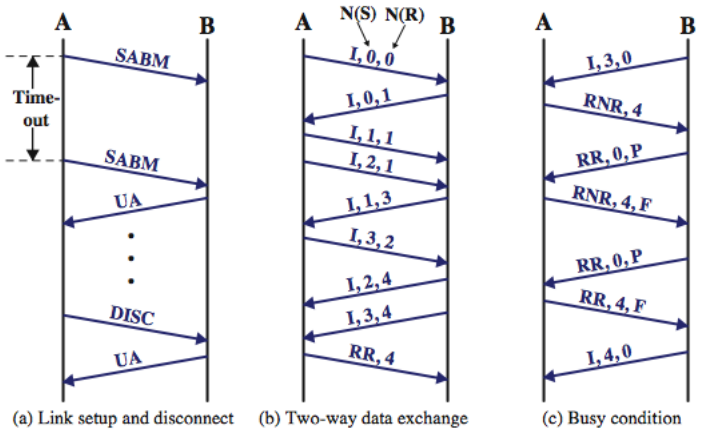
170

HDLC Operation

- consists of exchange of information, supervisory and unnumbered frames
- have three phases
  - initialization
    - by either side, set mode & seq
  - data transfer
    - with flow and error control
    - using both I & S-frames (RR, RNR, REJ, SREJ)
  - disconnect
    - when ready or fault noted

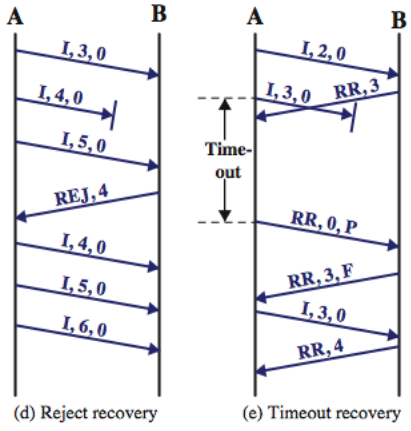
171

HDLC Operation Example



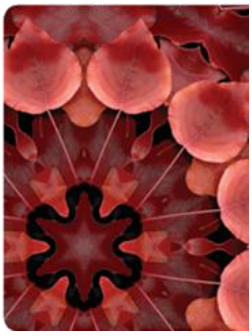
172

HDLC Operation Example



173

End



181