Sliding Window Protocols

Sliding Window Protocols

- A One-Bit Sliding Window Protocol
- A Protocol Using Go Back N
- A Protocol Using Selective Repeat

Sliding Window Protocol

Full Duplex data transmission

Have two separate Communication channels and use each one for simplex data traffic (in different directions).

If this is done, we have two separate physical circuits, each with a "forward" channel (for data) and a "reverse" channel (for acknowledgements).

In both cases the bandwidth of the reverse channel is almost entirely wasted. In effect, the user is paying for two circuits but using only the capacity of one.

A better idea is to use the same circuit for data in both directions.
In this model the data frames from *A* to *B* are intermixed with the acknowledgement frames from *A* to *B*.

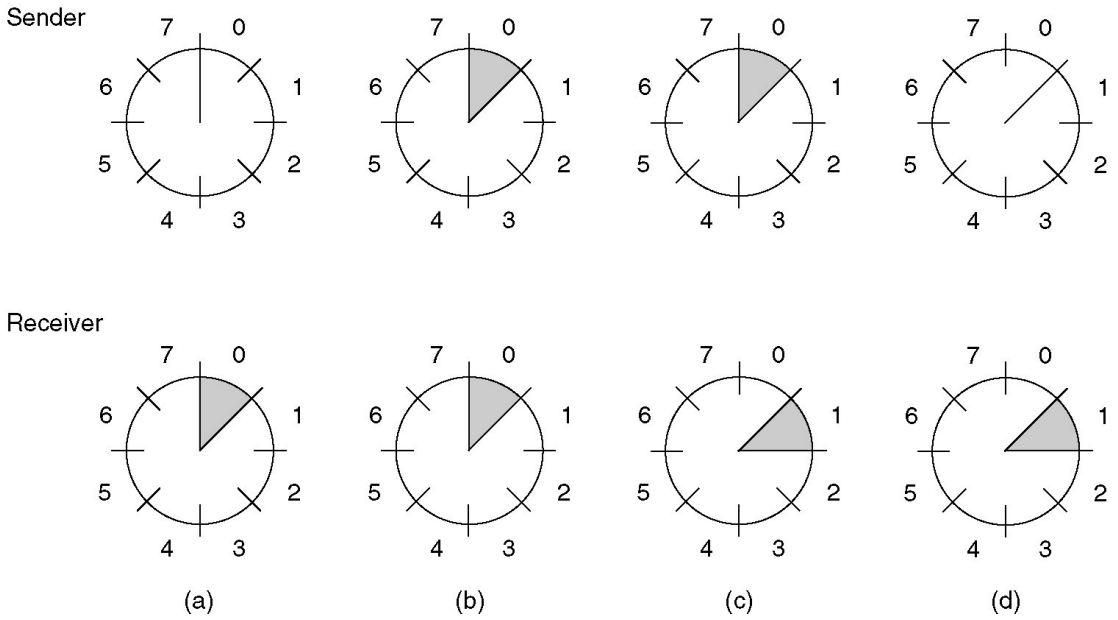
By looking at the *kind* field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgement.

Sliding Window Protocol

Piggybacking

When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame (using the *ack* field in the frame header). In effect, the acknowledgement gets a free ride on the next outgoing data frame. The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking**.

Sliding Window Protocols (2)



A sliding window of size 1, with a 3-bit sequence number.

(a) Initially.

(b) After the first frame has been sent.

(c) After the first frame has been received.

(d) After the first acknowledgement has been received.

A One-Bit Sliding Window Protocol

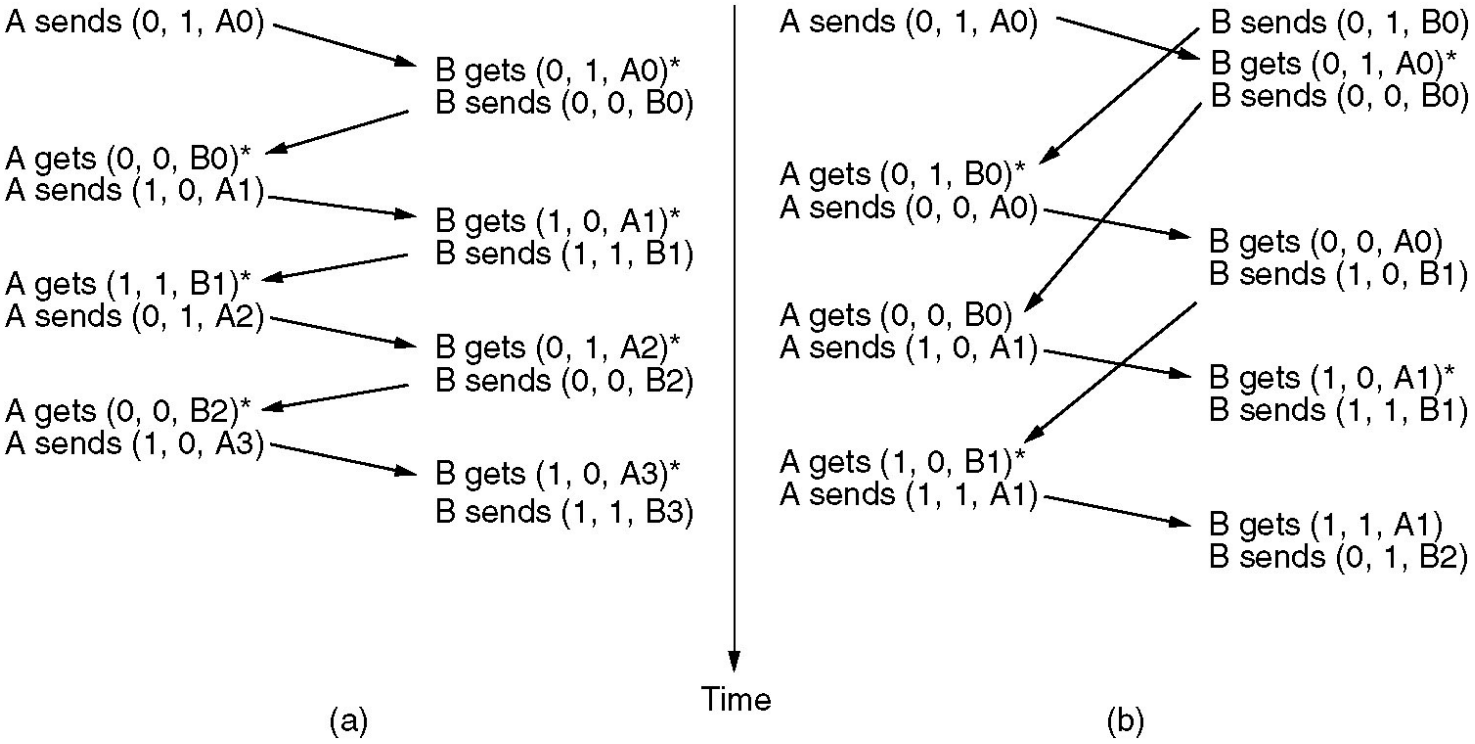
```
/* Protocol 4 (sliding window) is bidirectional. */
#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 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 */
```


Continued →

A One-Bit Sliding Window Protocol (ctd.)

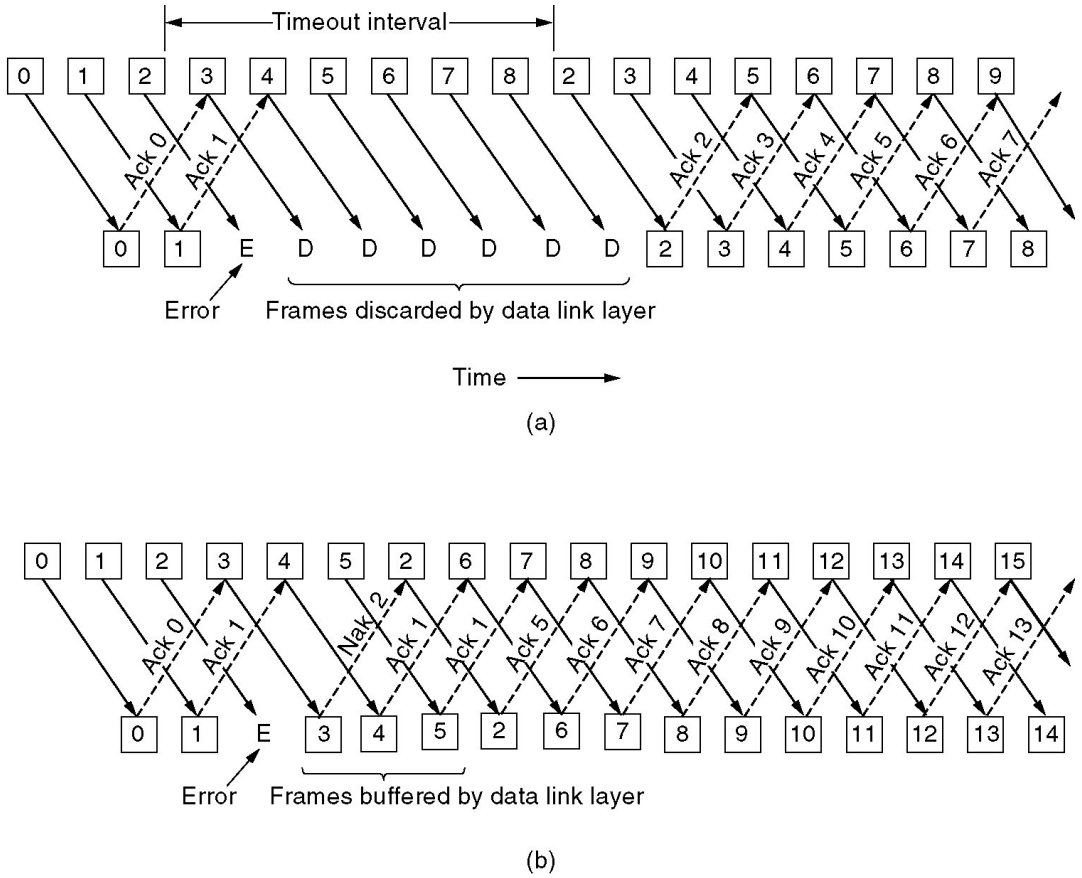
```
while (true) {
    wait_for_event(&event);                 /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) {           /* a frame has arrived undamaged. */
        from_physical_layer(&r);            /* go get it */

        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 */
  }
}
```

A One-Bit Sliding Window Protocol (2)



Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

A Protocol Using Go Back N



Pipelining and error recovery. Effect on an error when

(a) Receiver's window size is 1.

(b) Receiver's window size is large.

Sliding Window Protocol Using Go Back N

```
/* Protocol 5 (pipelining) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike 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                       /* should be 2^n – 1 */
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);
}

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 */
}
```


Continued →

Sliding Window Protocol Using Go Back N

```
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;                   /* # output buffers currently in use */
  seq_nr i;                           /* used to index into the buffer array */
  event_type event;

  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 */
```

Continued →

Sliding Window Protocol Using Go Back N

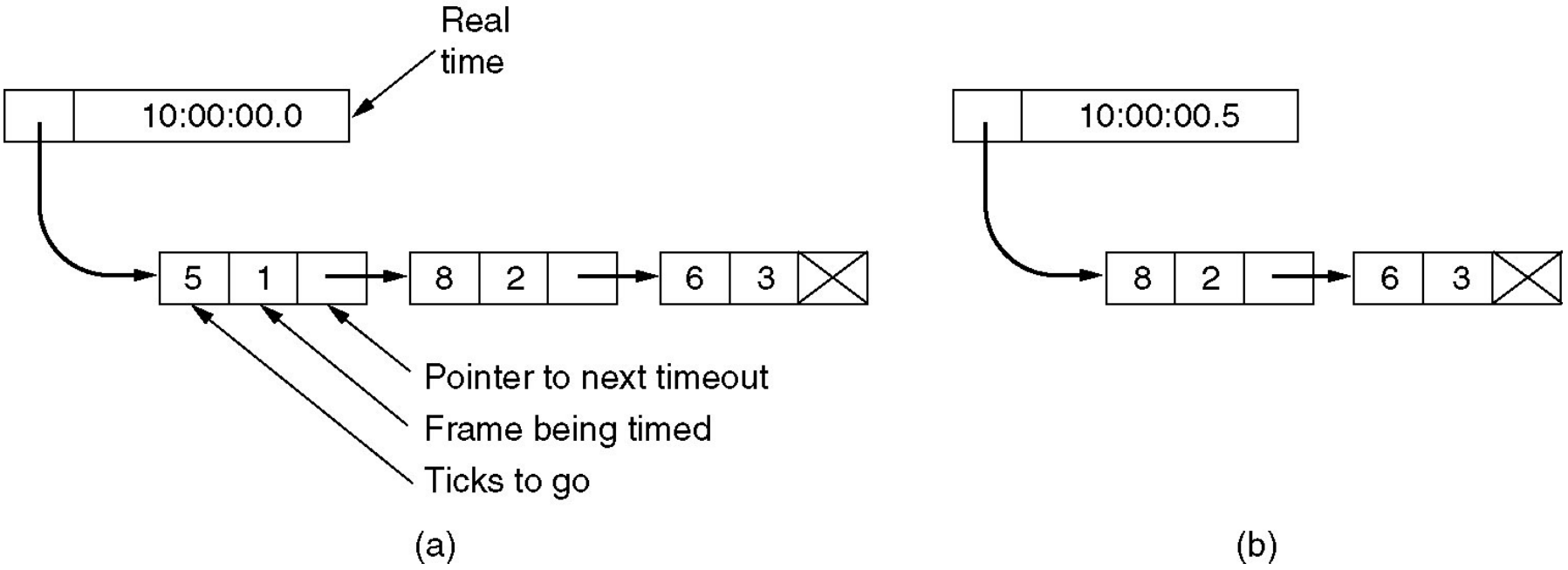
```
while (true) {
  wait_for_event(&event);              /* four possibilities: see event_type above */

  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 */
          }
```


Continued →

Sliding Window Protocol Using Go Back N

```
            /* 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 1 frame */
                inc(next_frame_to_send);  /* prepare to send the next one */
            }

    }

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

Sliding Window Protocol Using Go Back N (2)



Simulation of multiple timers in software.

A Sliding Window Protocol Using Selective Repeat

```
/* Protocol 6 (nonsequential receive) 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 protocol5, 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 */
}
```


Continued →

A Sliding Window Protocol Using Selective Repeat (2)

```
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;
```

Continued →

A Sliding Window Protocol Using Selective Repeat (3)

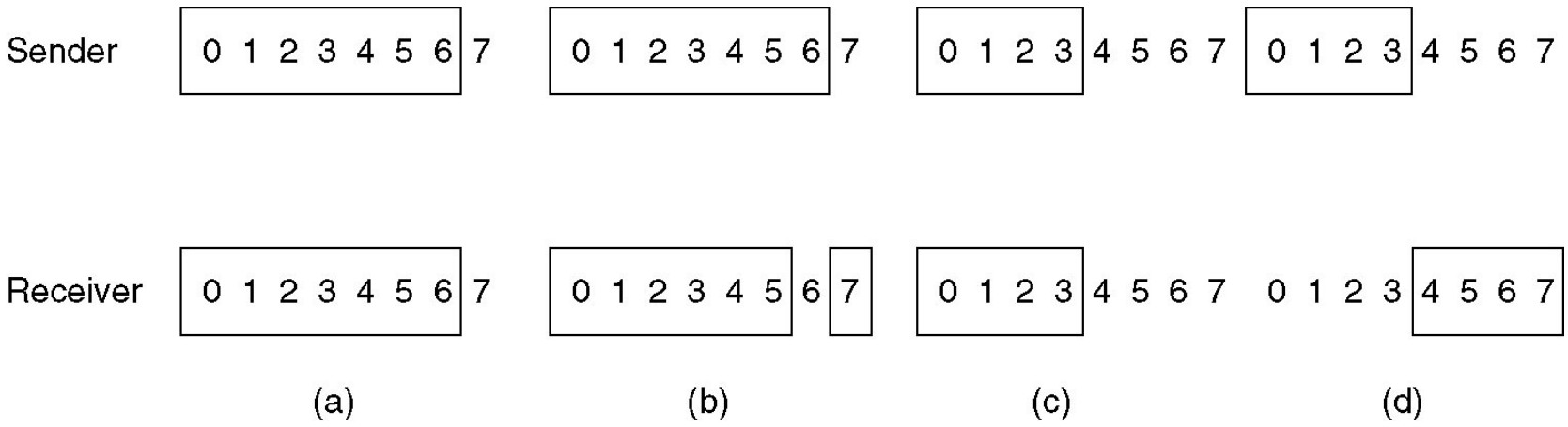
```
while (true) {
  wait_for_event(&event);                    /* five possibilities: see event_type above */
  switch(event) {
    case network_layer_ready:                /* accept, save, and transmit a new frame */
        nbuffered = nbuffered + 1;           /* expand the window */
        from_network_layer(&out  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 */
                }
            }
        }
```

Continued →

A Sliding Window Protocol Using Selective Repeat (4)

```
        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;
    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 (5)



(a) Initial situation with a window size seven.

(b) After seven frames sent and received, but not acknowledged.

(c) Initial situation with a window size of four.

(d) After four frames sent and received, but not acknowledged.