Three protocols of increasing complexity

Protocol 1
Assumes:
 Data transmission in one direction only (simplex).
 No errors take place on the physical channel.
 The sender/receiver can generate/consume an infinite amount of data.
 Always ready for sending/receiving.

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender1(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 */
    }
}
void receiver1(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 */
      }
}
```

Protocol 2
Assumes:
 No longer assume receiver can process incoming data infinitely fast.
 Sender ships one frame and then waits for acknowledgment (stop and wait.)
 The contents of the acknowledgment frame are unimportant.
 Data transmission is one directional, but must have bi-directional line.

```
/* Protocol 2 (stop-and-wait)
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender2(void)
{
      frame       s;                      /* buffer for an outbound frame */
      packet      buffer;                 /* buffer for an outbound packet */
      event_type  event;                  /* frame_arrival is the only possibility */
      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 */
            wait_for_event(event(&event); /* do not proceed until given go ahead*/
    }
void receiver2(void)
{
      frame       r, s;
      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 */
            to_physical_layers);          /* send a dummy frame to awaken sender */
```

```
      }
}
```

Protocol 3
Assumes:
 The channel is noisy and we can lose frames (they never arrive).
 Simple approach, add a time-out to the sender so if no ACK after a certain period, it retransmits the frame.
 Scenario of a bug that could happen if we're not careful:
      A transmits frame one
      B receives A1
      B generates ACK
      ACK is lost
      A times out, retransmits
      B gets duplicate copy of A1 (and sends it on to network layer.)
 Use a sequence number. How many bits? 1-bit is sufficient for this simple case because only concerned about two successive frames.
 Positive Acknowledgment w/ Retransmission (PAR): Sender waits for positive acknowledgment before going to next data item.

```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1                    /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout } event_type;
#include "protocol.h"

void sender3(void)
{
      seq_nr      next_frame_to_send;               /* Seq number of next outgoing frame */
      frame       s;                          /* buffer for an outbound frame */
      packet      buffer;                     /* buffer for an outbound packet */
      event_type  event;                      /* frame_arrival is the only possibility */
      next_frame_to_send = 0;
      from_network_layer(&buffer);            /* go get something to send */

      while (true) {
          s.info = buffer;                    /* copy it into s 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(&event);       /* frame arrival or cksum err, or timeout */
          if ( event == frame_arrival) {
              from_physical_layers(&s);       /* Get the ACK */
              if ( s.ack == next_frame_to_send ) {
                  from_network_layer( &buffer );      /* get the next one to send */
                  inc( next_frame_to_send );  /* invert 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);             /* only possibility is frame arrival */
          if ( frame == event_arrival ) {     /* A valid frame has arrived */
              from_physical_layer(&r);        /* go get the inbound frame */
              if ( r.seq == frame_expected ) {  /* This is what we've been waiting for */
                  to_network_layer(&r.info);  /* pass the data to the network layer */
                  inc(frame_expected);        /* next time expect the other seq # */
              }
             s.ack = 1 - frame_expected;
```

```
        to_physical_layer(&s);                  /* send a dummy frame to awaken sender */
        }
    }
}
```