```
/* Go-back-n */
#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)
{
    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[ ])
{
    frame s;
    s.info = buffer[frame_nr];
    s.seq = frame_nr;
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
    to_physical_layer(&s);
    start_timer(frame_nr);
}

void protocol5(void)
{
    seq_nr next_frame_to_send;
    seq_nr ack_expected;
    seq_nr frame_expected;
    frame r;
    packet buffer[MAX_SEQ + 1];
    seq_nr nbuffered;
    seq_nr i;
    event_type event;

    enable_network_layer();
    ack_expected = 0;
    next_frame_to_send = 0;
    frame_expected = 0;
    nbuffered = 0;
    while (true) {
        wait_for_event(&event);
        switch(event) {
            case network_layer_ready:
                from_network_layer(&buffer[next_frame_to_send]);
                nbuffered = nbuffered + 1;
                send_data(next_frame_to_send, frame_expected, buffer);
                inc(next_frame_to_send);
                break;

            case frame_arrival:
                from_physical_layer(&r);
                if (r.seq == frame_expected) {
                    to_network_layer(&r.info);
                    inc(frame_expected);
                }
                while (between(ack_expected, r.ack, next_frame_to_send)) {
                    nbuffered = nbuffered - 1;
                    stop_timer(ack_expected);
                    inc(ack_expected);
                }
                break;

            case cksum_err:
                break;

            case timeout:
                next_frame_to_send = ack_expected;
                for (i = 1; i <= nbuffered; i++) {
                    send_data(next_frame_to_send, frame_expected, buffer);
                    inc(next_frame_to_send);
                }
        }
        if (nbuffered < MAX_SEQ)
            enable_network_layer();
        else
            disable_network_layer();
    }
}
```

```
/* Protocol 6 (Selective repeat) */
#define MAX_SEQ 7
#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;
seq_nr oldest_frame = MAX_SEQ + 1;

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
    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[ ]) {
    frame s;
    s.kind = fk;
    if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
    s.seq = frame_nr;
    s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
    if (fk == nak) no_nak = false;
    to_physical_layer(&s);
    if (fk == data) start_timer(frame_nr % NR_BUFS);
    stop_ack_timer();
}

void protocol6(void) {
    seq_nr ack_expected;
    seq_nr next_frame_to_send;
    seq_nr frame_expected;
    seq_nr too_far;
    int i;
    frame r;
    packet out_buf[NR_BUFS];
    packet in_buf[NR_BUFS];
    boolean arrived[NR_BUFS];
    seq_nr nbuffered;
```

```
event_type event;
enable_network_layer();
ack_expected = 0;
next_frame_to_send = 0;
frame_expected = 0;
too_far = NR_BUFS;
nbuffered = 0;
for (i = 0; i < NR_BUFS; i++) arrived[i] = false;

while (true) {
    wait_for_event(&event);
    switch(event) {
        case network_layer_ready:
            nbuffered = nbuffered + 1;
            from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]);
            send_frame(data, next_frame_to_send, frame_expected, out_buf);
            inc(next_frame_to_send);
            break;

        case frame_arrival:
            from_physical_layer(&r);
            if (r.kind == data) {
                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)) {
                    arrived[r.seq % NR_BUFS] = true;
                    in_buf[r.seq % NR_BUFS] = r.info;
                    while (arrived[frame_expected % NR_BUFS]) {
                        to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                        no_nak = true;
                        arrived[frame_expected % NR_BUFS] = false;
                        inc(frame_expected);
                        inc(too_far);
                        start_ack_timer();
                    }
                }
            }
```

```
                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;
                    stop_timer(ack_expected % NR_BUFS);
                    inc(ack_expected);
                }
                break;

            case cksum_err:
                if (no_nak) send_frame(nak, 0, frame_expected, out_buf);
                break;
            case timeout:
                send_frame(data, oldest_frame, frame_expected, out_buf);
                break;
            case ack_timeout:
                send_frame(ack,0,frame_expected, out_buf);
        }
        if (nbuffered < NR_BUFS)
            enable_network_layer();
        else
            disable_network_layer();
    }
}
```