Network and Security Project 1

sebha15@student.sdu.dk
antof15@student.sdu.dk
jocoh15@student.sdu.dk

October 19, 2017

1 Introduction

The purpose of this project is to design and implement a reliable data transfer between multiple hosts using selective repeat. The handed out skeleton provides a starting point on the selective repeat algorithm, and the program already provides a reliable data transfer between two host. Thus, the work lies in expanding this network to 4 host, using the already implemented selective repeat algorithm.

2 Design

In designing the software, we for the better part of the project strictly adhered to the 'recommended' section of the description. We retained all previously created global values; adding only a few to ease changing of local ones. We also settled with using the events already included in rdt.c. As a final note, we initially assumed station-numbers to be zero-indexed, but after various segmentation faults discovered they were not, leading to other variables being created to keep an overview.

3 Implementation

```
case timeout: /* Ack timeout or regular timeout.*/
                        // Check if it is the ack_timer
                        // Get station for which timer ran out
                        timer_id = event.timer_id;
                        sender_index = -1;
                        for (int x = 0; x < nrOfStations; x++) {
                                if (ack_timer_id[x] == timer_id){
                                        sender_index = x;
                                        break;
                                } else {
                                        for (int t = 0; t < NR_BUFS; t
                                            ++){
                                                if (timer_ids[x][t] ==
                                                    timer_id){
                                                        sender_index =
                                                            x;
                                                        break;
                                                }
                                        }
                                        if (sender_index != -1){
                                                break;
                                        }
                                }
                        }
                        logLine(info, "Message from timer: '%s'\n", (
                            char *) event.msg );

                        if( timer_id == ack_timer_id[sender_index] ) {
                            // Ack timer timer out
                                logLine(trace, "Timeout with id: %d -
                                    acktimer_id is %d\n", timer_id,
                                    ack_timer_id[sender_index]);
                                free(event.msg);
                                ack_timer_id[sender_index] = -1;
                                printf("Explicit Ack sending for frame:
                                     %d to %d\n", frame_expected[
                                    sender_index], sender_index+1 );
                                send_frame(ACK,0,frame_expected[
                                    sender_index], out_buf[sender_index
                                    ], sender_index+1);
                        } else {
                                int timed_out_seq_nr = atoi( (char *)
                                    event.msg );
                                logLine(debug, "Timeout for frame -
                                    need to resend frame %d\n",
                                    timed_out_seq_nr);
                                printf("Timer %d timed out. Sending
                                    frame %d to %d\n", timer_id,
                                    timed_out_seq_nr ,sender_index+1 );
                                send_frame(DATA, timed_out_seq_nr,
                                    frame_expected[sender_index],
                                    out_buf[sender_index], sender_index
                                    +1);
                        }
                        break;
```

Probably the most edited piece of code compared to the original rdt.c. While not very efficient,

we decided on just searching our timer-arrays for the correct ID. As will be mentioned in testing, this might be part of the source, if not the source itself, of the unstable behaviour.
since all other changes to the code revolves around increasing variables a dimension, we chose not to include other parts.

4 Testing

In designing the scenario for testing, we ended upon 4 stations; each sending a fixed number of messages to the next i.e. station 1 -> station 2 -> station 3 -> station 4. We tested this (picture 1) scenario several times with different error frequencies; working correct, as in the messages were received and outputted. However, during testing we also found very unstable behaviour when the error frequency was set at 0.200 or more. We were not able to correct this, but narrowed it down to timers not being found/existing when needed; a thing to correct.

Figure 1: Testing with error freq= 0.200

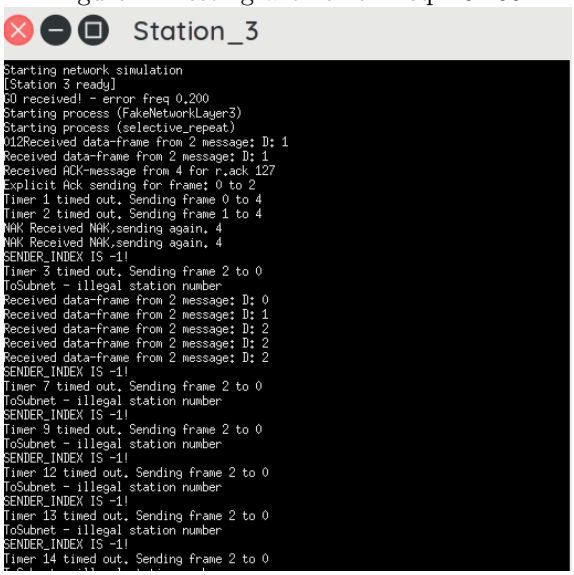
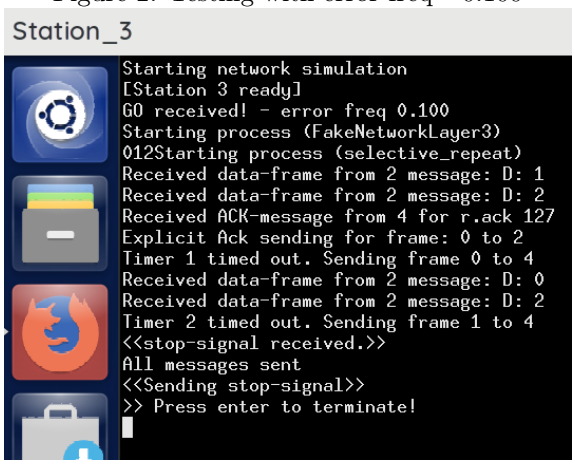


Figure 2: Testing with error freq= 0.100



5 Conclusion

The final program succesfully implements a reliable data transfer between multiple hosts, using the selective repeat algorithm, thought it works correctly only under narrow circumstances.
We found the subnet provided, more difficult to work with than what was likely intended. In hindsight we also probably would have worked in the reverse order of the 'recommended' section, given that we really only felt an understanding after having edited the function 'selective repeat'.

6 Appendix

```
/*
 * Reliable data transfer between two stations
 *
 * Author: Jacob Aae Mikkelsen.
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "rdt.h"
#include "subnetsupport.h"
#include "subnet.h"
#include "fifoqueue.h"
#include "debug.h"

/* En macro for at lette overfoerslen af korrekt navn til Activate */
#define ACTIVATE(n, f) Activate(n, f, #f)

#define MAX_SEQ 127        /* should be 2^n - 1 */
#define NR_BUFS 4
#define nrOfStations 4

/* Globale variable */

char *StationName;         /* Globalvariabel til at overfoere programnavn
*/
int ThisStation;           /* Globalvariabel der identificerer denne station. */
int nrOfMessagesToSend = 3;
log_type LogStyle;         /* Hvilken slags log skal systemet foere
*/
boolean network_layer_enabled[nrOfStations];

LogBuf mylog;                /* logbufferen
*/

FifoQueue from_network_layer_queue;                        /* Queue for data from networ
FifoQueue for_network_layer_queue;    /* Queue for data for the network layer */

mlock_t *network_layer_lock;
mlock_t *write_lock;

packet ugly_buffer; // TODO Make this a queue

int ack_timer_id[4] = {-1,-1,-1,-1};
int timer_ids[nrOfStations][NR_BUFS];
boolean nak_possible[nrOfStations]; /* no nak has been sent yet */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
        boolean x = (a <= b) && (b < c);
        boolean y = (c < a) && (a <= b);
        boolean z = (b < c) && (c < a);
        // TODO Omskriv saa det er til at fatte!
        logLine(debug, "a=%d, b=%d, c=%d, x=%d, y=%d, z=%d\n", a,b,c,x,y,z);
```

```
    return x || y || z;
}

/* Copies package content to buffer, ensuring it has a string end character. */
void packet_to_string(packet* data, char* buffer)
{
        strncpy ( buffer, (char*) data->data, MAX_PKT );
        buffer[MAX_PKT] = '\0';

}
// Send_frame takes REAL station-number
static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet
{
    /* 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)
    {
        nak_possible[dest] = false;        /* one nak per frame, please */
    }
    to_physical_layer(&s, dest);        /* transmit the frame */
    if (fk == DATA)
    {
        start_timer(frame_nr, dest);
    }
    stop_ack_timer(dest);        /* no need for separate ack frame */
}

/* Fake network/upper layers for station 1
 *
 * Send 20 packets and receive 10 before we stop
 * */
void FakeNetworkLayer1()
{
        //(i%nrOfStations)+1
        int target = 2;
        char *buffer;
        packet *pack;
        int i,j;
    long int events_we_handle;
    event_t event;
        FifoQueueEntry e;

    from_network_layer_queue = InitializeFQ();
    for_network_layer_queue = InitializeFQ();

    // Setup some messages
    for( i = 0; i < nrOfMessagesToSend; i++ ) {
        pack = (packet *) malloc(sizeof(packet));
        buffer = (char *) malloc(sizeof(char *) * (MAX_PKT-8));
        pack->dest = target;
        pack->src = ThisStation;
```

```
        sprintf(buffer, "D: %d", i);
        strcpy(pack->data, buffer);
        printf(  "%d", i );
            EnqueueFQ( NewFQE( (void *) pack ), from_network_layer_queue );
    }

    events_we_handle = network_layer_allowed_to_send | data_for_network_layer;

    // Give selective repeat a chance to start
    sleep(2);

    i = 0;
    j = 0;
    while( true ) {
        // Wait until we are allowed to transmit
        Wait(&event, events_we_handle);

        switch(event.type) {
                case network_layer_allowed_to_send:
                                Lock( network_layer_lock );
                        if( i < nrOfMessagesToSend && network_layer_enabled[target-1]
                                // Signal element is ready
                                logLine(info, "Sending signal for message #%d\n", i);
                                network_layer_enabled[target-1] = false;
                                Signal(network_layer_ready, NULL);
                                i++;
                        }
                                Unlock( network_layer_lock );
                        break;
                case data_for_network_layer:
                                Lock( network_layer_lock );

                                e = DequeueFQ( for_network_layer_queue );
                        logLine(succes, "Received message: %s\n" ,( (char *) e->val)

                                Unlock( network_layer_lock );

                        j++;
                        break;
        }

                if( i >= nrOfMessagesToSend) {

                    logLine(info, "Station %d done. - ('sleep(5)')\n", ThisStation);
                    /* A small break, so all stations can be ready */
                    sleep(5);
                    printf("All messages sent\n");
                    Stop();
                }
    }
}

void FakeNetworkLayer2()
{
        int target = 3;
        char *buffer;
        packet *pack;
        int i,j;
```

```
long int events_we_handle;
event_t event;
    FifoQueueEntry e;

from_network_layer_queue = InitializeFQ();
for_network_layer_queue = InitializeFQ();

// Setup some messages
for( i = 0; i < nrOfMessagesToSend; i++ ) {
    pack = (packet *) malloc(sizeof(packet));
    buffer = (char *) malloc(sizeof(char *) * (MAX_PKT-8));
    pack->dest = target;
    pack->src = ThisStation;
    sprintf(buffer, "D: %d", i);
    strcpy(pack->data, buffer);
    printf(  "%d", i );
        EnqueueFQ( NewFQE( (void *) pack ), from_network_layer_queue );
}

events_we_handle = network_layer_allowed_to_send | data_for_network_layer;

// Give selective repeat a chance to start
sleep(2);

i = 0;
j = 0;
while( true ) {
    // Wait until we are allowed to transmit
    Wait(&event, events_we_handle);

    switch(event.type) {
            case network_layer_allowed_to_send:
                            Lock( network_layer_lock );
                    if( i < nrOfMessagesToSend && network_layer_enabled[target-1]
                            // Signal element is ready
                            logLine(info, "Sending signal for message #%d\n", i);
                            network_layer_enabled[target-1] = false;
                            Signal(network_layer_ready, NULL);
                            i++;
                    }
                            Unlock( network_layer_lock );
                    break;
            case data_for_network_layer:
                            Lock( network_layer_lock );

                            e = DequeueFQ( for_network_layer_queue );
                    logLine(succes, "Received message: %s\n" ,( (char *) e->val)

                            Unlock( network_layer_lock );

                    j++;
                    break;
    }

            if( i >= nrOfMessagesToSend) {

                logLine(info, "Station %d done. - (\'sleep(5)\')\n", ThisStation);
                /* A small break, so all stations can be ready */
```

```
                    sleep(5);
                    printf("All messages sent\n");
                    Stop();
                }
    }
}

void FakeNetworkLayer3()
{
        int target = 4;
        char *buffer;
        packet *pack;
        int i,j;
    long int events_we_handle;
    event_t event;
        FifoQueueEntry e;

    from_network_layer_queue = InitializeFQ();
    for_network_layer_queue = InitializeFQ();

    // Setup some messages
    for( i = 0; i < nrOfMessagesToSend; i++ ) {
        pack = (packet *) malloc(sizeof(packet));
        buffer = (char *) malloc(sizeof(char *) * (MAX_PKT-8));
        pack->dest = target;
        pack->src = ThisStation;
        sprintf(buffer, "D: %d", i);
        strcpy(pack->data, buffer);
        printf(  "%d", i );
            EnqueueFQ( NewFQE( (void *) pack ), from_network_layer_queue );
    }

    events_we_handle = network_layer_allowed_to_send | data_for_network_layer;

    // Give selective repeat a chance to start
    sleep(2);

    i = 0;
    j = 0;
    while( true ) {
        // Wait until we are allowed to transmit
        Wait(&event, events_we_handle);

        switch(event.type) {
                case network_layer_allowed_to_send:
                                Lock( network_layer_lock );
                        if( i < nrOfMessagesToSend && network_layer_enabled[target-1]
                                // Signal element is ready
                                logLine(info, "Sending signal for message #%d\n", i);
                                network_layer_enabled[target-1] = false;
                                Signal(network_layer_ready, NULL);
                                i++;
                        }
                                Unlock( network_layer_lock );
                        break;
                case data_for_network_layer:
                                Lock( network_layer_lock );
```

```
                                e = DequeueFQ( for_network_layer_queue );
                            logLine(succes, "Received message: %s\n" ,( (char *) e->val)

                                Unlock( network_layer_lock );

                            j++;
                            break;
        }

                if( i >= nrOfMessagesToSend) {

                    logLine(info, "Station %d done. - ('sleep(5)')\n", ThisStation);
                    /* A small break, so all stations can be ready */
                    sleep(5);
                    printf("All messages sent\n");
                    Stop();
                }
    }
}


void log_event_received(long int event) {
        char *event_name;
        switch(event) {
                case 1:
                        event_name = "frame_arrival";
                        break;
                case 2:
                        event_name = "timeout";
                        break;
                case 4:
                        event_name = "network_layer_allowed_to_send";
                        break;
                case 8:
                        event_name = "network_layer_ready";
                        break;
                case 16:
                        event_name = "data_for_network_layer";
                        break;
                default:
                        event_name = "unknown";
                        break;
        }
        logLine(trace, "Event received %s\n", event_name);
}

void selective_repeat() {
    seq_nr ack_expected[nrOfStations];              /* lower edge of sender's window
    seq_nr next_frame_to_send[nrOfStations];        /* upper edge of sender's window
    seq_nr frame_expected[nrOfStations];            /* lower edge of receiver's windo
    seq_nr too_far[nrOfStations];                   /* upper edge of receiver's windo
    int i, packet_dest, sender_index, frame_sender;
/* index into buffer pools */
    frame r;                            /* scratch variable */
    packet pck;                                     /* scratch variable*/
    packet out_buf[nrOfStations][NR_BUFS];          /* buffers for the outbound strea
    packet in_buf[nrOfStations][NR_BUFS];           /* buffers for the inbound stream
    boolean arrived[nrOfStations][NR_BUFS];         /* inbound bit map */
```

```
seq_nr nbuffered[nrOfStations];                    /* how many output buffers curren
event_t event;
long int events_we_handle;
unsigned int timer_id;

write_lock = malloc(sizeof(mlock_t));
network_layer_lock = (mlock_t *)malloc(sizeof(mlock_t));

Init_lock(write_lock);
Init_lock( network_layer_lock );

logLine(trace, "Starting selective repeat %d\n", ThisStation);

packet_dest = 0;
for (i = 0; i < nrOfStations; i++) {
    for (int j = 0; j<NR_BUFS; j++){
            arrived[i][j] = false;
            timer_ids[i][j] = -1;
    }
    nak_possible[i] = false;
    ack_timer_id[i] = -1;
    enable_network_layer(i);     /*initialize*/
    ack_expected[i] = 0;         /* next ack expected on the inbound stream */
            next_frame_to_send[i] = 0;         /* number of next outgoing frame */
    frame_expected[i] = 0;         /* frame number expected */
    too_far[i] = NR_BUFS;         /* receiver's upper window + 1 */
    nbuffered[i] = 0;         /* initially no packets are buffered */
}

events_we_handle = frame_arrival | timeout | network_layer_ready;


/*
// If you are in doubt how the event numbers should be, comment in this, and you
printf("%#010x\n", 1);
printf("%#010x\n", 2);
printf("%#010x\n", 4);
printf("%#010x\n", 8);
printf("%#010x\n", 16);
printf("%#010x\n", 32);
printf("%#010x\n", 64);
printf("%#010x\n", 128);
printf("%#010x\n", 256);
printf("%#010x\n", 512);
printf("%#010x\n", 1024);
    */

while (true) {
    // Wait for any of these events
    Wait(&event, events_we_handle);
    log_event_received(event.type);

    switch(event.type) {
        case network_layer_ready:         /* accept, save, and transmit a new frame
            //printf("Station: %d Case: network_layer_ready\n", ThisStation );
            logLine(trace, "Network layer delivers frame - lets send it\n");
                from_network_layer(&pck);
                packet_dest = pck.dest-1;
```

```
                    nbuffered[packet_dest] = nbuffered[packet_dest]+1;
/* expand the window */
                    memcpy(&out_buf[packet_dest][next_frame_to_send[packet_dest] % NR
                    //from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]); /*
                    send_frame(DATA, next_frame_to_send[packet_dest], frame_expected[
/* transmit the frame */
                    inc(next_frame_to_send[packet_dest]);        /* advance upper wind
                    //printf("Next frame to send is now: %d\n", next_frame_to_send[pa
                    break;

                case frame_arrival:         /* a data or control frame has arrived */

                                from_physical_layer(&r);         /* fetch incoming fram
                                //printf("Station: %d Case: frame_arrival from station
                                frame_sender = r.fromStation;
                    sender_index = r.fromStation-1; /* I'm not very smart*/
                    packet_dest = sender_index;
                    if (r.kind == ACK){
                        printf("Received ACK-message from %d for r.ack %d\n", r.fromS
                    }
                    if (ThisStation == 1 ){
                        printf("frame_sender = %d, sender_index = %d \n", frame_sender
                    }

                                if (r.kind == DATA) {
                                        printf("Received data-frame from %d message: %
                                        /* An undamaged frame has arrived. */
                                        if ((r.seq != frame_expected[sender_index]) &&
                                                send_frame(NAK, 0, frame_expected[send
                                        } else {
                                                start_ack_timer(frame_sender);
                                        }
                                        if (between(frame_expected[sender_index], r.se
                                                /* Frames may be accepted in any order
                                                arrived[sender_index][r.seq % NR_BUFS]
/* mark buffer as full */
                                                in_buf[sender_index][r.seq % NR_BUFS]
/* insert data into buffer */
                                                while (arrived[sender_index][frame_ex
                                                        /* Pass frames and advance win
                                                        to_network_layer(&in_buf[sende
                                                        nak_possible[sender_index]= tr
                                                        arrived[sender_index][frame_e
                                                        inc(frame_expected[sender_inde
/* advance lower edge of receiver's window */
                                                        inc(too_far[sender_index]);
/* advance upper edge of receiver's window */
                                                        start_ack_timer(frame_sender)
/* to see if (a separate ack is needed */
                                                }
                                        }
                                }
                                if((r.kind==NAK) && between(ack_expected[sender_index]
                                        printf("NAK %s %d\n", "Received NAK, sending ag
                                        send_frame(DATA, (r.ack+1) % (MAX_SEQ + 1), fr
                                }
                                //printf("Expecting ack -> %d frame_ack -> %d next_fra
                                logLine(info, "Are we between so we can advance window
```

```
                                while (between(ack_expected[sender_index], r.ack, next
                                        logLine(debug, "Advancing window %d\n", ack_e
                                        nbuffered[sender_index] = nbuffered[sender_inc
/* handle piggybacked ack */
                                        stop_timer(ack_expected[sender_index]% NR_BUF
/* frame arrived intact */
                                        inc(ack_expected[sender_index]);
/* advance lower edge of sender's window */
                                }
                                break;

                        case timeout: /* Ack timeout or regular timeout. Muligvis fejl her.*/
                                // Check if it is the ack_timer
                                // Get station for which timer ran out
                                timer_id = event.timer_id;
                                //printf("Looking for %d! \n",timer_id );
                                sender_index = -1;
                                for (int x = 0; x < nrOfStations; x++) {
                                        //printf("Looking at ACK_timers[%d] -> %d \n", x,ack_
                                        if (ack_timer_id[x] == timer_id){
                                                sender_index = x;
                                                break;
                                        } else {
                                                for (int t = 0; t < NR_BUFS; t++){
                                                        //printf("Looking at timer_ids[%d][%d
                                                        if (timer_ids[x][t] == timer_id){
                                                                sender_index = x;
                                                                break;
                                                        }
                                                }
                                                if (sender_index != -1){
                                                        break;
                                                }
                                        }
                                }
                                if (sender_index == -1){
                                                printf("SENDER_INDEX IS -1!\n");
                                }
                                //printf("Timer subject -> %d timer_id -> %d ack_timer_id -> %
                                logLine(info, "Message from timer: '%s'\n", (char *) event.msg

                                if( timer_id == ack_timer_id[sender_index] ) { // Ack timer ti
                                        logLine(trace, "Timeout with id: %d - acktimer_id is %
                                        free(event.msg);
                                        ack_timer_id[sender_index] = -1;
                                        printf("Explicit Ack sending for frame: %d to %d\n", f
                                        send_frame(ACK,0,frame_expected[sender_index], out_bu
                                } else {
                                        int timed_out_seq_nr = atoi( (char *) event.msg );
                                        logLine(debug, "Timeout for frame - need to resend fra
                                        printf("Timer %d timed out. Sending frame %d to %d\n"
                                        send_frame(DATA, timed_out_seq_nr, frame_expected[send
                                }
                                break;
                }

                if (nbuffered[packet_dest] < NR_BUFS) {
                    //printf("Station: %d Enabling network_layer for %d \n",ThisStation,
```

```
                enable_network_layer(packet_dest);
            } else {
                //printf("station: %d Disabling network_layer for %d \n",ThisStation
                disable_network_layer(packet_dest);
            }
        }
}

void enable_network_layer(int NeighbourID) {

        Lock( network_layer_lock );
        logLine(trace, "enabling network layer\n");
        network_layer_enabled[NeighbourID] = true;
        //printf("Layer enabled for %d \n",NeighbourID+1 );
        Signal( network_layer_allowed_to_send, NULL );
        Unlock( network_layer_lock );
}

void disable_network_layer(int NeighbourID){
        Lock( network_layer_lock );
        logLine(trace, "disabling network layer\n");
        network_layer_enabled[NeighbourID] = false;
        Unlock( network_layer_lock );
}

void from_network_layer(packet *p) {
    FifoQueueEntry e;

        Lock( network_layer_lock );
        e = DequeueFQ( from_network_layer_queue );
    Unlock( network_layer_lock );

        if(!e) {
                logLine(error, "ERROR: We did not receive anything from the queue, lik
        } else {
                memcpy(p, (packet *)ValueOfFQE( e ), sizeof(packet));
            free( (void *)ValueOfFQE( e ) );
                DeleteFQE( e );
        }

}


void to_network_layer(packet *p) {
        char * buffer;
    Lock( network_layer_lock );

    buffer = (char *) malloc ( sizeof(packet) * (MAX_PKT-8));
    packet_to_string(p, buffer);

    EnqueueFQ( NewFQE( (void *) p ), for_network_layer_queue );

    Unlock( network_layer_lock );

    Signal( data_for_network_layer, NULL);
}
```

```
void print_frame(frame* s, char *direction) {
        char temp[MAX_PKT+1];

        switch( s->kind ) {
                case ACK:
                        printf( "%s: ACK frame. Ack seq_nr=%d\n", direction, s->ack);
                        logLine(info, "%s: ACK frame. Ack seq_nr=%d\n", direction, s->
                        break;
                case NAK:
                        printf("%s: NAK frame. Nak seq_nr=%d\n", direction, s->ack);
                        logLine(info, "%s: NAK frame. Nak seq_nr=%d\n", direction, s->
                        break;
                case DATA:
                        printf("%s: DATA frame. Frame seq_nr=%d\n", direction, s->ack
                        packet_to_string(&(s->info), temp);
                        logLine(info, "%s: DATA frame [seq=%d, ack=%d, kind=%d, (%s)]
                        break;
        }
}

int from_physical_layer(frame *r) {
        r->seq = 0;
        r->kind = DATA;
        r->ack = 0;

        int source, dest, length;

        logLine(trace, "Receiving from subnet in station %d\n", ThisStation);
        FromSubnet(&source, &dest, (char *) r, &length);
        r->fromStation = source;

        return 0;
}


void to_physical_layer(frame *s, int reciever)
{
        /*
        int send_to;

        if (ThisStation == 1){
                send_to = 2;
        } else {
                send_to = 1;
        }
        */
        //print_frame(s, "sending");

        s->fromStation = ThisStation;
        s->sendTime = GetTime();

        //printf("%d ToSubnet(%d, %d, (char *) s, sizeof(frame)\n",s->kind, ThisStatio
        ToSubnet(ThisStation, reciever, (char *) s, sizeof(frame));
}


void start_timer(seq_nr k, int NeighbourID) {
```

```
        int index = NeighbourID-1;
        char *msg;
        msg = (char *) malloc(100*sizeof(char));
        sprintf(msg, "%d", k); // Save seq_nr in message

        timer_ids[index][k % NR_BUFS] = SetTimer( frame_timer_timeout_millis, (void *)
        //printf("timer_ids index %d \n", index);
        //printf("Timer set at timer_ids[%d][%d] \n", index, k % NR_BUFS);
        logLine(trace, "start_timer for seq_nr=%d timer_ids=[%d, %d, %d, %d] %s\n", k,
}


void stop_timer(seq_nr k, int NeighbourID) {
        int timer_id;
        char *msg;
        int index = NeighbourID-1;

        timer_id = timer_ids[index][k];
        logLine(trace, "stop_timer for seq_nr %d med id=%d\n", k, timer_id);

    if (StopTimer(timer_id, (void *)&msg)) {
        logLine(trace, "timer %d stoppet. msg: %s \n", timer_id, msg);
        free(msg);
    } else {
        logLine(trace, "timer %d kunne ikke stoppes. Maaske er den timet ud?timer_ids=
    }
}


void start_ack_timer(int NeighbourID)
{
        int index = NeighbourID-1;
        if( ack_timer_id[index] == -1 ) {
                logLine(trace, "Starting ack-timer\n");
                char *msg;
                msg = (char *) malloc(100*sizeof(char));
                strcpy(msg, "Ack-timer");
                ack_timer_id[index] = SetTimer( act_timer_timeout_millis, (void *)msg
                logLine(debug, "Ack-timer startet med id %d\n", ack_timer_id[index]);
        }
}

void stop_ack_timer(int NeighbourID)
{
        char *msg;
        int index = NeighbourID-1;

        logLine(trace, "stop_ack_timer\n");
    if (StopTimer(ack_timer_id[index], (void *)&msg)) {
            logLine(trace, "timer %d stoppet. msg: %s \n", ack_timer_id, msg);
        free(msg);
    }
    ack_timer_id[index] = -1;
}


int main(int argc, char *argv[])
{
```

```
  StationName = argv[0];
  ThisStation = atoi( argv[1] );

  if (argc == 3)
    printf("Station %d: arg2 = %s\n", ThisStation, argv[2]);

  mylog = InitializeLB("mytest");

    LogStyle = synchronized;

    printf( "Starting network simulation\n" );

  /* processerne aktiveres */
  ACTIVATE(1, FakeNetworkLayer1);
  ACTIVATE(2, FakeNetworkLayer2);
  ACTIVATE(3, FakeNetworkLayer3);
  //ACTIVATE(4, FakeNetworkLayer4);
  ACTIVATE(1, selective_repeat);
  ACTIVATE(2, selective_repeat);
  ACTIVATE(3, selective_repeat);
  ACTIVATE(4, selective_repeat);

  /* simuleringen starter */
  Start();
  exit(0);
}


/*
void FakeNetworkLayer2()
{
    long int events_we_handle;
    event_t event;
        int i,j;
        FifoQueueEntry e;
        char *buffer;
        packet *pack;


    from_network_layer_queue = InitializeFQ();
    5 = InitializeFQ();

    events_we_handle = network_layer_allowed_to_send | data_for_network_layer;

    j = 0;
    while( true ) {
        // Wait until we are allowed to transmit
        //printf ("THIS IS LINE: %d.\n", __LINE__);
        printf("FakeNetworkLayer2 Waiting\n");
        Wait(&event, events_we_handle);

        switch(event.type) {
                case network_layer_allowed_to_send:
                        printf("Station: %d Case: network_layer_allowed_to_send : Fak
                                Lock( network_layer_lock );
                        if( network_layer_enabled[0] && !EmptyFQ( from_network_layer_
) {
                                logLine(info, "Signal from network layer for message\
```

```
                        network_layer_enabled[0] = false;
                        ClearEvent( network_layer_ready ); // Don't want to s
                        printf("Station: %d Signal: network_layer_ready\n", T
                        Signal(network_layer_ready, NULL);
                }
                        Unlock( network_layer_lock );
                break;
        case data_for_network_layer:
                printf("Station: %d Case: data_for_network_layer\n", ThisStat
                        Lock( network_layer_lock );

                        e = DequeueFQ( for_network_layer_queue );
                logLine(succes, "Received message: %s\n" ,( (char *) e->val)

                if( j < 2) {
                                ( (char *) e->val)[0] = 'd';
                                EnqueueFQ( e, from_network_layer_queue );
                }

                        Unlock( network_layer_lock );

                        j++;
                logLine(info, "j: %d\n" ,j );
                break;
        }
        if( EmptyFQ( from_network_layer_queue ) && j >= 2) {
                logLine(succes, "Stopping - received 20 messages and sent 10\n
);
                printf("Stopping - received 20 messages and sent 10\n");
            sleep(5);
            Stop();
        }
    }*/
```