Couse: CZ3006 Net-Centri Computing
Name: Xavier Tan
Matric Number: U1720697B

Summary:

Implemented protocol 6; for the 5 different event cases, send the frame when network layer is ready, when the frame arrives, checksum error, timeout, acknowledgment timeout. The function to send the frame to physical layer also have to be implemented. The timer and acknowledgement timer also have to be implemented.

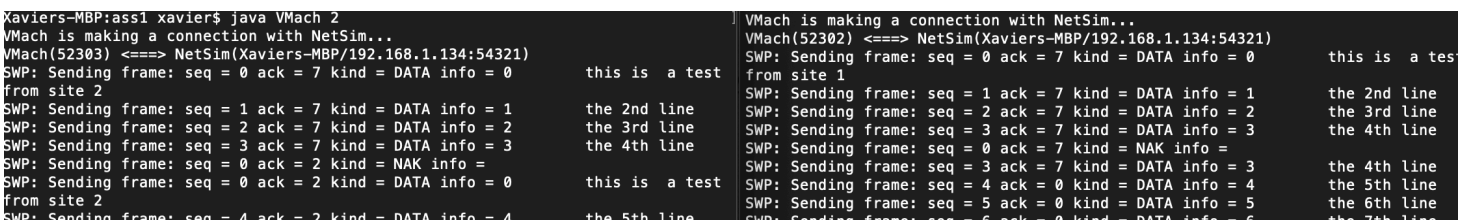
Java Source files used:

import java.util.Timer; //to create the timer object
import java.util.TimerTask; //to create the timerTask object
Both Objects needed for the timer functions.
Two packages used to build the timer.
Another 2 classes is created to run the timer using the swe functions given.

Approaches:

1. Full-duplex data communication.

   The SWP file have both send and receive function from network layer, and both send and receive function from physical later. Hence creating a full duplex data communication, since in the network_layer_ready case, it fetches a new packet from network layer and sends a frame, and in frame_arrival case is again send frames after it gets frame from physical layer.
   The protocol uses same circuit for data in both directions. Since the reverse channel has the same capacity as the forward channel. In this model the data frames from VMach1 to VMach2 are intermixed with the acknowledgement frames from VMach1 to VMach2. By looking at the type of file(Kind) field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgement. 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 can be piggybacked.



```
Xaviers-MBP:ass1 xavier$ java NetSim 3
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: xaviers-mbp : 52302
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: xaviers-mbp : 52303
VMach 2 Check sum error for seq = 0 error counter = 1
VMach 2 loose frame seq = 1 error counter = 2
VMach 1 Check sum error for seq = 3 error counter = 1
VMach 1 loose frame seq = 3 error counter = 2
```

2. In-order delivery of packets to the network-layer.

   The receiving data link layer's window corresponds to the frames it may accept. Any frame falling outside the window is discarded. When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer, an acknowledgement is generated, and the window size increased by one. Unlike the sender's window, the receiver's window always remains at its initial size. The network layer is always fed data in the proper order, regardless of the data link layer's window size. This is done through the between function which will choose the correct sequence number of the packets.

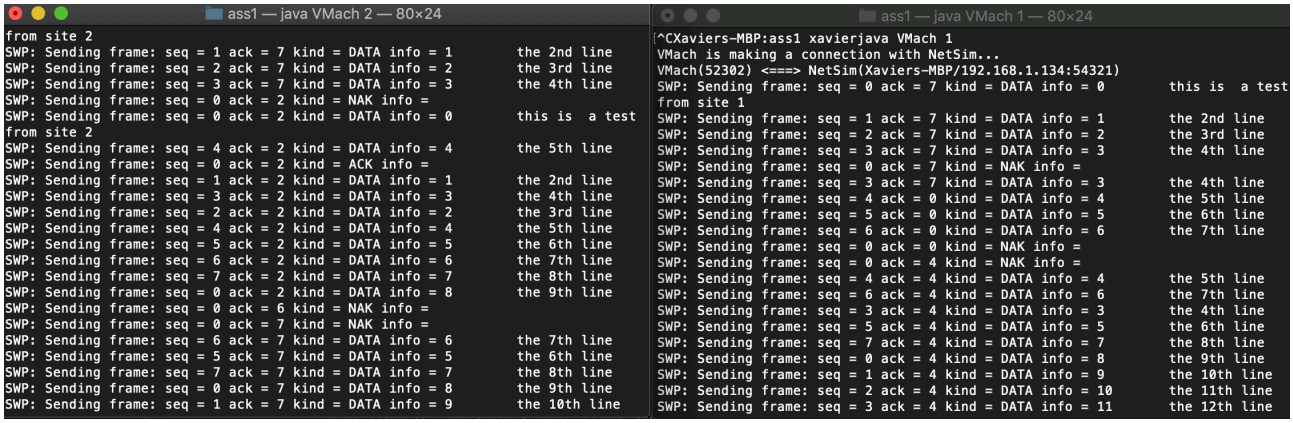
```
if (between(frame_expected, r.seq, too_far) && (arrived[r.seq % NR_BUFS]==false)){
  /* Frames may be accepted in any order. */
    arrived[r.seq % NR_BUFS] = true; /* mark buffer as full */
    in_buf[r.seq  % NR_BUFS] = r.info; /* insert data into buffer */
while(arrived[frame_expected % NR_BUFS]){
  /* Pass frames and advance window. */
  to_network_layer(in_buf[frame_expected % NR_BUFS]);
  no_nak = true;
  arrived[frame_expected % NR_BUFS] = false;
  frame_expected = inc(frame_expected); //advance lower edge of receiver's window
  too_far = inc(too_far); /* advance upper edge of receiver's window */
  start_ack_timer(); /* to see if a separate ack is needed */
}
```

3. Selective repeat retransmission strategy.

   If the VMach 1 realises it received a damaged/lost frame(no nak is true in the CKSUM_ERR case), it will send a NAK to get the other VMach 'B' . So when VMach 2 receives another frame and the frame kind is NAK, the machine will resend the frame expected by VMach 1.

```
case (PEvent.CKSUM_ERR): {
  if(no_nak)
    send_frame(PFrame.NAK, 0, frame_expected, out_buf); //damaged frame
} break;
```

```
//resend frame if NAK recieved
if((r.kind == PFrame.NAK) && between(ack_expected, (r.ack + 1)%(MAX_SEQ + 1),next_frame_to_send))
      send_frame(PFrame.DATA, (r.ack+1)%(MAX_SEQ+1), frame_expected, out_buf);
```



4. Synchronization with the network-layer by granting credits.

```
private void enable_network_layer(int nr_of_bufs){
    swe.grant_credit(nr_of_bufs);
}
```

At the start of the protocol, it will enable the network layer and tell network layer the number of buffer datalink layer has. So that the buffer size of datalink is always larger or equal to network layer, to ensure no overflowing of data from datalink layer to network layer. So when an frame arrives to the datalink layer, a buffer is used up by both side. Thus there is a need a credit to network layer, so that it knows that datalink is going to send one frame.

```
while(between(ack_expected, r.ack, next_frame_to_send)){
//  nbuffered--; /* handle piggybacked ack */
  stop_timer(ack_expected % NR_BUFS); /* frame arrived intact */
  ack_expected = inc(ack_expected);  /* advance lower edge of sender's window */
  enable_network_layer(1); // 1 buffer enabled
}
```

5. Negative acknowledgement.

   In the CKSUM_ERR case the protocol will get the machine to send a NAK frame to its counterpart. Whenever the receiver suspects that it has received a damaged frame or a frame that it did not expects it sends a negative acknowledgement (NAK) frame back to the sender.

```
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: xaviers-mbp : 52664
VMach 1 loose frame seq = 0 error counter = 1
VMach 1 loose frame seq = 1 error counter = 2
VMach 1 Check sum error for seq = 0 error counter = 3
VMach 2 Check sum error for seq = 0 error counter = 1
VMach 2 Check sum error for seq = 6 error counter = 2
VMach 2 Check sum error for seq = 7 error counter = 3
VMach 1 loose frame seq = 0 error counter = 4
VMach 1 Check sum error for seq = 1 error counter = 5
VMach 1 loose frame seq = 2 error counter = 6
VMach 2 Check sum error for seq = 1 error counter = 4
VMach 1 loose frame seq = 0 error counter = 7
VMach 2 loose frame seq = 7 error counter = 5
VMach 2 loose frame seq = 2 error counter = 6
VMach 2 Check sum error for seq = 0 error counter = 7
VMach 1 Check sum error for seq = 2 error counter = 8
VMach 1 Check sum error for seq = 0 error counter = 9
VMach 2 Check sum error for seq = 7 error counter = 8
VMach 1 loose frame seq = 0 error counter = 10
VMach 1 loose frame seq = 2 error counter = 11
VMach 1 Check sum error for seq = 3 error counter = 12
VMach 2 loose frame seq = 7 error counter = 9
```

```
case (PEvent.CKSUM_ERR): {
  if(no_nak)
    send_frame(PFrame.NAK, 0, frame_expected, out_buf); //damaged frame
} break;

case (PEvent.TIMEOUT): {
  send_frame(PFrame.DATA, oldest_frame, frame_expected, out_buf); //timed out, resend again
} break;

case (PEvent.ACK_TIMEOUT): {
  send_frame(PFrame.ACK, 0, frame_expected, out_buf); /* ack timer expired; send ack */
}
```

6. Separate acknowledgment when the reverse traffic is light or none.

After an data frame arrives, an auxiliary timer is started by *startacktimer*. If there is no reverse traffic before this timer expires, a separate acknowledgement frame is sent. An interrupt due to the auxiliary timer is called an *acktimeout* event. This is done by swe.generate_acktimeout_event(); Now, one-directional traffic flow is possible because the lack of reverse data frames thus acknowledgments don't have to wait for piggybacks.Only one auxiliary timer exists, and if *startacktimer* is called while the timer is running, it is reset to a full acknowledgement timeout interval.

The auxiliary timer is shorter than the timer used for timing out data frames. Auxiliary timer is 50 while sequence timer is 200.

This is so that the correct frame is acknowledged before frame's retransmission timer expire and retransmit the frame.

```
/*===============================================================*
 *  File: SWP.java                                               *
 *                                                               *
 *  This class implements the sliding window protocol            *
 *  Used by VMach class                                          *
 *  Uses the following classes: SWE, Packet, PFrame, PEvent,     *
 *                                                               *
 *  Author: Professor SUN Chengzheng                             *
 *          School of Computer Engineering                       *
 *          Nanyang Technological University                     *
 *          Singapore 639798                                     *
 *===============================================================*/

import java.util.Timer;
import java.util.TimerTask;

public class SWP {

/*========================================================================*
 the following are provided, do not change them!!
 *========================================================================*/
   //the following are protocol constants.
   public static final int MAX_SEQ = 7;
   public static final int NR_BUFS = (MAX_SEQ + 1)/2; /* number of buffers should be 2^n – 1 for selective
repeat protocol*/

   // the following are protocol variables
   private int oldest_frame = 0; /* initial value is only for the simulator */
   private PEvent event = new PEvent();
   private Packet out_buf[] = new Packet[NR_BUFS]; //output buffer size same as the number of input buffer

   //the following are used for simulation purpose only
   private SWE swe = null;
   private String sid = null;

   //Constructor
   public SWP(SWE sw, String s){
      swe = sw;
      sid = s;
   }

   //the following methods are all protocol related

   //create and initialise our out buffer
   private void init(){
      for (int i = 0; i < NR_BUFS; i++){
               out_buf[i] = new Packet();
      }
   }

   private void wait_for_event(PEvent e){
      swe.wait_for_event(e); //may be blocked
      oldest_frame = e.seq;  //set timeout frame seq
   }

   private void enable_network_layer(int nr_of_bufs) {
   //network layer is permitted to send if credit is available
      swe.grant_credit(nr_of_bufs);
   }
```

```
 private void from_network_layer(Packet p) {
   swe.from_network_layer(p);
 }

 private void to_network_layer(Packet packet) {
          swe.to_network_layer(packet);
  }

 private void to_physical_layer(PFrame fm)  {
   System.out.println("SWP: Sending frame: seq = " + fm.seq +
                      " ack = " + fm.ack + " kind = " +
                      PFrame.KIND[fm.kind] + " info = " + fm.info.data );
   System.out.flush();
   swe.to_physical_layer(fm);
 }

//get physcial frame from the physcial layer
private void from_physical_layer(PFrame fm) {
   PFrame fm1 = swe.from_physical_layer();
          fm.kind = fm1.kind;
          fm.seq = fm1.seq;
          fm.ack = fm1.ack;
          fm.info = fm1.info;
 }


/*=============================================================================*
        implement your Protocol Variables and Methods below:
 *=============================================================================*/

 boolean no_nak = true; /* no nak has been sent yet */

 //Timer initialization
 Timer[] f_timer = new Timer[NR_BUFS];
 Timer ack_timer = null;

//make sure the frame to send does not exceed MAX_SEQ.
        public int inc(int next_frame_to_send) {
                return ((next_frame_to_send + 1) % (MAX_SEQ + 1));
        }

//choose frame
        public boolean between(int frame_expected, int frame_seq, int too_far){
                return ((frame_expected <= frame_seq)&&(frame_seq < too_far))
                                || ((too_far < frame_expected)&&(frame_expected<=frame_seq))
                                || ((frame_seq<too_far)&&(too_far<frame_expected));
        }
/* Construct and send a data, ack, or nak frame. */
 public void send_frame(int fk, int frame_nr, int frame_expected, Packet[] buffer){

          PFrame fm = new PFrame();
          fm.kind = fk;/* kind == data, ack, or nak */

  //if physcial frame is the correct type,
          if(fm.kind == PFrame.DATA){
    fm.info = buffer[frame_nr%NR_BUFS];
  }

          fm.seq = frame_nr; /* only meaningful for data frames */
          fm.ack = (frame_expected + MAX_SEQ)%(MAX_SEQ+1);
```

```
        if(fm.kind == PFrame.NAK) /* one nak per frame, please */
                no_nak = false;

        to_physical_layer(fm); /* transmit the frame */

        if(fm.kind == PFrame.DATA)
                start_timer(frame_nr);

        stop_ack_timer(); // no need for separate ack frame, since piggybacking
 }

 public void protocol6() {

                int ack_expected = 0; /* lower edge of sender's window , initialize to 0*/
                int next_frame_to_send = 0; // upper edge of sender's window + 1 , next ack expected on
                the inbound stream is 0
                int frame_expected = 0; /* lower edge of receiver's window */
                int too_far = NR_BUFS; /* upper edge of receiver's window + 1 */
                 int i; // index into buffer pool
                PFrame r = new PFrame(); /* scratch variable */
                Packet in_buf[] = new Packet[NR_BUFS]; /* buffers for the inbound stream */
                init(); /* buffers for the outbound stream */
                boolean[] arrived = new boolean[NR_BUFS];
                 int nbuffered = 0; // initially no packets buffered


                 enable_network_layer(NR_BUFS); //initialize

        while(true) {
                wait_for_event(event); // five possibilities:
                        switch(event.type) {

                          case (PEvent.NETWORK_LAYER_READY): { // accept,save, and transmit new frame.
                                from_network_layer(out_buf[next_frame_to_send%NR_BUFS]); //fetch new packet
                                send_frame(PFrame.DATA, next_frame_to_send, frame_expected, out_buf);
                        //transmit frame
                                next_frame_to_send = inc(next_frame_to_send); //advance  upper window edge
                          } break;

                          case (PEvent.FRAME_ARRIVAL ): {
                /* a data or control frame has arrived */
                                from_physical_layer(r);/* fetch incoming frame from physical layer */
                                if (r.kind == PFrame.DATA) { //if frame is undamaged and data is same
                                        if ((r.seq != frame_expected) && no_nak)
                                                send_frame(PFrame.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;
                                        frame_expected = inc(frame_expected); //advance lower edge of receiver's
window
                                        too_far = inc(too_far); /* advance upper edge of receiver's window */
```

```
                                        start_ack_timer(); /* to see if a separate ack is needed */
                                                }
                                        }
                                }
        //resend frame if NAK recieved
                                if((r.kind == PFrame.NAK) && between(ack_expected, (r.ack + 1)%(MAX_SEQ +
1),next_frame_to_send))
                                send_frame(PFrame.DATA, (r.ack+1)%(MAX_SEQ+1), frame_expected, out_buf);
                                        while(between(ack_expected, r.ack, next_frame_to_send)){
                                                stop_timer(ack_expected % NR_BUFS); /* frame arrived intact */
                                                ack_expected = inc(ack_expected);  /* advance lower edge of
sender's window */
                                                enable_network_layer(1); // 1 buffer enabled
                                                }
                        } break;

case (PEvent.CKSUM_ERR): {
            if(no_nak)
                        send_frame(PFrame.NAK, 0, frame_expected, out_buf);
                        //damaged frame, send a NAK to the other machine
} break;

case (PEvent.TIMEOUT): {
                         send_frame(PFrame.DATA, oldest_frame, frame_expected, out_buf);
                        //timed out, resend again
} break;

case (PEvent.ACK_TIMEOUT): {
                          send_frame(PFrame.ACK, 0, frame_expected, out_buf); /* ack timer expired; send ack */
 } break;

default:
             System.out.println("SWP: undefined event type = " + event.type);
            System.out.flush();
}
  }
    }

 /* Note: when start_timer() and stop_timer() are called,
    the "seq" parameter must be the sequence number, rather
    than the index of the timer array,
    of the frame associated with this timer,
   */

   private void start_timer(int seq) {
              stop_timer(seq);
      //create new timer and new timertask
      f_timer[seq % NR_BUFS] = new Timer();
      //schedule the  task for execution after 200ms
      f_timer[seq % NR_BUFS].schedule(new f_task(seq), 200);
   }

   private void stop_timer(int seq) {
              if (f_timer[seq % NR_BUFS] != null) {
         f_timer[seq % NR_BUFS].cancel();
         f_timer[seq % NR_BUFS] = null;
      }
   }

   private void start_ack_timer() {
```

```
            stop_ack_timer();

        //starts another timer for sending separate ack
        ack_timer = new Timer();
        ack_timer.schedule(new ack_task(), 50);
    }

    private void stop_ack_timer() {
            if (ack_timer != null) {
          ack_timer.cancel();
          ack_timer = null;
        }
    }

    class ack_task extends TimerTask {

        public void run() {
          //stop timer
          stop_ack_timer();
          swe.generate_acktimeout_event();
        }
     }

    class f_task extends TimerTask {

        private int seq;

        public f_task(int seq) {
          this.seq = seq;
        }

        public void run() {
          //stops this timer, discarding any scheduled tasks for the current seq
          stop_timer(seq);
          swe.generate_timeout_event(seq);
        }
     }
}//End of class


/* Note: In class SWE, the following two public methods are available:
    . generate_acktimeout_event() and
    . generate_timeout_event(seqnr).

    To call these two methods (for implementing timers),
    the "swe" object should be referred as follows:
      swe.generate_acktimeout_event(), or
      swe.generate_timeout_event(seqnr).
*/
```