Implementation

Full-duplex data communication

A full-duplex data communication is used in this assignment where are two separate simplex data channel. Thus, data can be transmitted in both directions on a single circuit at the same time.

In this assignment, Virtual Machine 1 (VMach 1) and Virtual Machine 2 (VMach 2) can both send and receive data at the same time (shown in Figure 1a, 1b, and 1c). When VMach 1 send data in one channel, it can also simultaneously receive data in another channel. So does VMach 2. A field, known as ‘kind’, is in the frame header to differentiate data from acknowledgement (assignment shown in Figure 1d).

```
Command Prompt - java  NetSim 0
Z:\Ass1>java NetSim 0
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: 10.27.56.76 : 56619
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: 10.27.56.76 : 56620
```

Figure 1a

```
Command Prompt - java  VMach 1
z:\Ass1>java VMach 1
VMach is making a connection with NetSim...
VMach(56619) <===> NetSim(DESKTOP-93NO2OQ/10.27.56.76:54321)
SWP: Sending frame: seq = 0 ack = 7 kind = DATA info = 0       this is  a test from site 1
SWP: Sending frame: seq = 1 ack = 7 kind = DATA info = 1       the 2nd line
SWP: Sending frame: seq = 2 ack = 7 kind = DATA info = 2       the 3rd line
SWP: Sending frame: seq = 3 ack = 7 kind = DATA info = 3       the 4th line
SWP: Sending frame: seq = 4 ack = 0 kind = DATA info = 4       the 5th line
SWP: Sending frame: seq = 5 ack = 0 kind = DATA info = 5       the 6th line
SWP: Sending frame: seq = 6 ack = 0 kind = DATA info = 6       the 7th line
SWP: Sending frame: seq = 7 ack = 0 kind = DATA info = 7       the 8th line
SWP: Sending frame: seq = 0 ack = 4 kind = DATA info = 8       the 9th line
SWP: Sending frame: seq = 1 ack = 4 kind = DATA info = 9       the 10th line
SWP: Sending frame: seq = 2 ack = 5 kind = DATA info = 10      the 11th line
SWP: Sending frame: seq = 3 ack = 5 kind = DATA info = 11      the 12th line
SWP: Sending frame: seq = 4 ack = 5 kind = DATA info = 12      the 13th line
SWP: Sending frame: seq = 5 ack = 1 kind = DATA info = 13      the 14th line
SWP: Sending frame: seq = 6 ack = 1 kind = DATA info = 14      the 15th line
SWP: Sending frame: seq = 7 ack = 1 kind = DATA info = 15      the 16th line
SWP: Sending frame: seq = 0 ack = 1 kind = DATA info = 16      the 17th line
SWP: Sending frame: seq = 1 ack = 2 kind = DATA info = 17      the 18th line
SWP: Sending frame: seq = 2 ack = 6 kind = DATA info = 18      the 19th line
SWP: Sending frame: seq = 3 ack = 6 kind = DATA info = 19      the 20th line
SWP: Sending frame: seq = 4 ack = 6 kind = DATA info = 20      the 21th line
SWP: Sending frame: seq = 5 ack = 6 kind = DATA info = 21      the 22th line
SWP: Sending frame: seq = 6 ack = 0 kind = DATA info = 22      the 23th line
SWP: Sending frame: seq = 7 ack = 3 kind = DATA info = 23      the 24th line
SWP: Sending frame: seq = 0 ack = 3 kind = DATA info = 24      the 25th line
SWP: Sending frame: seq = 1 ack = 3 kind = DATA info = 25      the 26th line
SWP: Sending frame: seq = 2 ack = 3 kind = DATA info = 26      the 27th line
```

Figure 1b

```
Command Prompt - java  VMach 2
z:\Ass1>java VMach 2
VMach is making a connection with NetSim...
VMach(56620) <===> NetSim(DESKTOP-93NO2OQ/10.27.56.76:54321)
SWP: Sending frame: seq = 0 ack = 3 kind = DATA info = 0       this is  a test from site 2
SWP: Sending frame: seq = 1 ack = 3 kind = DATA info = 1       the 2nd line
SWP: Sending frame: seq = 2 ack = 3 kind = DATA info = 2       the 3rd line
SWP: Sending frame: seq = 3 ack = 3 kind = DATA info = 3       the 4th line
SWP: Sending frame: seq = 4 ack = 5 kind = DATA info = 4       the 5th line
SWP: Sending frame: seq = 5 ack = 0 kind = DATA info = 5       the 6th line
SWP: Sending frame: seq = 6 ack = 0 kind = DATA info = 6       the 7th line
SWP: Sending frame: seq = 7 ack = 0 kind = DATA info = 7       the 8th line
SWP: Sending frame: seq = 0 ack = 0 kind = DATA info = 8       the 9th line
SWP: Sending frame: seq = 1 ack = 4 kind = DATA info = 9       the 10th line
SWP: Sending frame: seq = 2 ack = 5 kind = DATA info = 10      the 11th line
SWP: Sending frame: seq = 3 ack = 5 kind = DATA info = 11      the 12th line
SWP: Sending frame: seq = 4 ack = 5 kind = DATA info = 12      the 13th line
SWP: Sending frame: seq = 5 ack = 5 kind = DATA info = 13      the 14th line
SWP: Sending frame: seq = 6 ack = 1 kind = DATA info = 14      the 15th line
SWP: Sending frame: seq = 7 ack = 2 kind = DATA info = 15      the 16th line
SWP: Sending frame: seq = 0 ack = 2 kind = DATA info = 16      the 17th line
SWP: Sending frame: seq = 1 ack = 2 kind = DATA info = 17      the 18th line
SWP: Sending frame: seq = 2 ack = 2 kind = DATA info = 18      the 19th line
SWP: Sending frame: seq = 3 ack = 6 kind = DATA info = 19      the 20th line
SWP: Sending frame: seq = 4 ack = 6 kind = DATA info = 20      the 21th line
SWP: Sending frame: seq = 5 ack = 7 kind = DATA info = 21      the 22th line
SWP: Sending frame: seq = 6 ack = 7 kind = DATA info = 22      the 23th line
SWP: Sending frame: seq = 7 ack = 7 kind = DATA info = 23      the 24th line
SWP: Sending frame: seq = 0 ack = 3 kind = DATA info = 24      the 25th line
SWP: Sending frame: seq = 1 ack = 3 kind = DATA info = 25      the 26th line
SWP: Sending frame: seq = 2 ack = 3 kind = DATA info = 26      the 27th line
```

Figure 1c

```java
void send_frame(int frameKind, int frame_nr, int frame_expected, Packet buffer[]) {
  //initialize a new frame
  PFrame tempFrame = new PFrame();

  //fill up the content of the frame
  tempFrame.kind = frameKind; //kind == data, ack, or nak

  if(frameKind == PFrame.DATA) {
     tempFrame.info = buffer[frame_nr % NR_BUFS];
  }
  tempFrame.seq = frame_nr; //only meaningful for data frames
  tempFrame.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);

  if (frameKind == PFrame.NAK) {
    no_nak = false; //one nak per frame, please
  }

  to_physical_layer(tempFrame); // transmit the frame

  if (frameKind == PFrame.DATA) {
    start_timer(frame_nr);  //start timer after DATA frame is sent
  }
  stop_ack_timer(); //due to piggyback, no need ack timer to send a seperate ack frame.
}
```

Figure 1d

In-order delivery of packets to the network-layer

In the event of FRAME_ARRIVAL, if it is a data frame, the data will be stored in a buffer. The frames can arrive the physical-layer in random order as they will all be stored in the in_buf[] buffer in-order. To deliver packets to the network-layer in-order, it is ensured by having a WHILE loop with a condition that is only true when the frame for lower edge of the sliding window has arrived undamaged (in Figure 2). In the WHILE loop, packets stored in in_buf[] buffer will be delivered to the network layer in order.

```
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
  //only pass frames and advance window once the frame for the lower edge of the receiver's sliding window has arrived
  while (arrived[frame_expected % NR_BUFS]) {
    //Pass frames and advance window.
    to_network_layer(in_buf[frame_expected % NR_BUFS]);
    no_nak = true;  //reset NaK to true since expected frame is received
    arrived[frame_expected % NR_BUFS] = false;  //reset inbound bit map for wrap around
    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 if no DATA frame are sent by the receiver to piggyback onto
  }
}
```

Figure 2

Selective repeat retransmission strategy

The difference between Go-Back-N strategy and selective repeat retransmission strategy is selective repeat retransmission strategy allows receiver to accept frames in random order or even accept frames that are not the expected frame, lower edge of the sliding window. This is accomplished by having an incoming buffer that stores the packets in-order before sending them to the network-layer (in Figure 3). When the expected frame, at the lower edge of the sliding window, is not received, a Negative Acknowledgement will be sent to the sender for retransmission of the frame.

```
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) {
    //An undamaged frame has arrived.
    //check if frame is lost and if NaK frame has been sent yet
    //need to check no_nak to prevent sending multiple no_nak frame incase the right seq data frame haven't reach yet
    if ((r.seq != frame_expected) && no_nak) {
      send_frame(PFrame.NAK, 0, frame_expected, out_buf); //since sending one NaK frame only, it's frame seq is 0
    } else {
      //useful when:
      //1. ACK frames from receiver are all lost, receiver shifted it's sliding window.
      //2. Sender's start_timer() time out for the frames that were sent since no ACK was received for them, thus resend those DATA frames.
      //3. Since the DATA frames received is not expected by the receiver, NaK is sent to the sender
      //4. Nak sent by receiver got lost.
      //5. Sender may keep sending those DATA frames while receiver keeps rejecting them.
      //6. start_ack_timer helps to prevent deadlock.
      start_ack_timer();  //sends correct ACK to sender to resynchronize
    }
    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
      //only pass frames and advance window once the frame for the lower edge of the receiver's sliding window has arrived
      while (arrived[frame_expected % NR_BUFS]) {
        //Pass frames and advance window.
        to_network_layer(in_buf[frame_expected % NR_BUFS]);
        no_nak = true;  //reset NaK to true since expected frame is received
        arrived[frame_expected % NR_BUFS] = false;  //reset inbound bit map for wrap around
        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 if no DATA frame are sent by the receiver to piggyback onto
      }
    }
  }
```

Figure 3

Synchronization with the network-layer by granting credits

During enabling of network-layer, the sliding window size is passed to enable_network_layer() (in Figure 4a). In the enable_network_layer(), the sliding window size sets the number of credits granted in the network-layer so that when the number of frames sent exceeds window size, the network-layer will be disabled (in Figure 4b).

```
public void protocol6() {
  int ack_expected; //lower edge of sender's window
  int next_frame_to_send; //upper edge of sender's window + 1
  int frame_expected; //lower edge of receiver's window
  int too_far; //upper edge of receiver's window + 1
  int i; //index into buffer pool
  PFrame r = new PFrame(); //scratch variable
  init(); //buffers for the outbound stream
  Packet in_buf[] = new Packet[NR_BUFS]; //buffers for the inbound stream
  boolean arrived[] = new boolean[NR_BUFS]; //inbound bit map
  int nbuffered; //how many output buffers currently used
  //PEvent event; //this.event

  enable_network_layer(NR_BUFS); //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

  //set all inbound but map to false
  for (i = 0; i < NR_BUFS; i++) {
    arrived[i] = false;
  }

  while (true) {
```

Figure 4a

```
 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);
 }
```

Figure 4b

Negative acknowledgement

During transmission of data frames from sender to receiver, frames may be lost or damaged. When a frame is lost, an IF statement will check if the frame received is expected and if no recent Negative Acknowledgement NaK frame has been sent (in Figure 5a and 5c). If both conditions are true, a NaK will be sent by the receiver.

If a frame is damaged, it will be detected by checking the checksum and thus resulting in a checksum error event. When a checksum error event occurs, NaK will be sent by the receiver (in Figure 5b and 5c).

```
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) {
    //An undamaged frame has arrived.
    //check if frame is lost and if NaK frame has been sent yet
    //need to check no_nak to prevent sending multiple no_nak frame incase the right seq data frame haven't reach yet
    if ((r.seq != frame_expected) && no_nak) {
      send_frame(PFrame.NAK, 0, frame_expected, out_buf); //since sending one NaK frame only, it's frame seq is 0
    } else {
```

Figure 5a

```
case (PEvent.CKSUM_ERR):
   if (no_nak)
     send_frame(PFrame.NAK, 0, frame_expected, out_buf); //send NaK for resend of frame when there is checksum error (damaged frame)
      break;
```

Figure 5b

```
SWP: Sending frame: seq = 4 ack = 3 kind = DATA info = 4        the 5th line
SWP: Sending frame: seq = 5 ack = 3 kind = DATA info = 5        the 6th line
SWP: Sending frame: seq = 3 ack = 3 kind = DATA info = 3        the 4th line
SWP: Sending frame: seq = 0 ack = 7 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 7 kind = ACK info =
SWP: Sending frame: seq = 3 ack = 7 kind = DATA info = 3        the 4th line
```

Figure 5c

Separate acknowledgment when the reverse traffic is light or none

Acknowledgment are usually piggybacked on reverse outgoing frames to reduce wasted bandwidth. However, there are scenarios when the reverse traffic is light or none. If the wait is too long, the sender's data frame timer may time out due to not receiving acknowledgement from the receiver. This results in retransmission of data frame to the receiver.

Therefore, the solution is to implement an acknowledgement timer. The acknowledgement timer will be begin when an undamaged data packet is sent to the network-layer (in Figure 6a). The acknowledgement timer will be terminated if the acknowledgement piggybacks on a reverse outgoing frame (in Figure 6b). If not, once time is up, the acknowledgement timer will trigger an ACK_TIMEOUT event, thus a separate acknowledgement is sent (in Figure 6c and 6e). The acknowledgement timer has to be shorter than the data frame timer (in Figure 6d).

```
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
  //only pass frames and advance window once the frame for the lower edge of the receiver's sliding window has arrived
  while (arrived[frame_expected % NR_BUFS]) {
    //Pass frames and advance window.
    to_network_layer(in_buf[frame_expected % NR_BUFS]);
    no_nak = true;  //reset NaK to true since expected frame is received
    arrived[frame_expected % NR_BUFS] = false;  //reset inbound bit map for wrap around
    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 if no DATA frame are sent by the receiver to piggyback onto
  }
}
```

Figure 6a

```
void send_frame(int frameKind, int frame_nr, int frame_expected, Packet buffer[]) {
  //initialize a new frame
  PFrame tempFrame = new PFrame();

  //fill up the content of the frame
  tempFrame.kind = frameKind; //kind == data, ack, or nak

  if(frameKind == PFrame.DATA) {
    tempFrame.info = buffer[frame_nr % NR_BUFS];
  }
  tempFrame.seq = frame_nr; //only meaningful for data frames
  tempFrame.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);

  if (frameKind == PFrame.NAK) {
    no_nak = false; //one nak per frame, please
  }

  to_physical_layer(tempFrame); // transmit the frame

  if (frameKind == PFrame.DATA) {
    start_timer(frame_nr);  //start timer after DATA frame is sent
  }
  stop_ack_timer(); //due to piggyback, no need ack timer to send a seperate ack frame.
}
```

Figure 6b

```
case (PEvent.ACK_TIMEOUT):
  send_frame(PFrame.ACK, 0, frame_expected, out_buf); //ack timer expired; Send a seperate ACK frame
  break;
```

Figure 6c

```
private void start_timer(int seqnr) {
  stop_timer(seqnr);
  timers[seqnr % NR_BUFS] = new Timer();
  timers[seqnr % NR_BUFS].schedule(new TimerTask() {
    @Override
    public void run() {
      swe.generate_timeout_event(seqnr);
    }
  }, 500);
}

private void stop_timer(int seqnr) {
  if(timers[seqnr % NR_BUFS] != null) {
    timers[seqnr % NR_BUFS].cancel();
    timers[seqnr % NR_BUFS].purge();
    timers[seqnr % NR_BUFS] = null;
  }
}

private void start_ack_timer() {
  stop_ack_timer();
  ackTimer = new Timer();
  ackTimer.schedule(new TimerTask() {
    @Override
    public void run() {
      swe.generate_acktimeout_event();
    }
  }, 150);
}
```

Figure 6d

```
SWP: Sending frame: seq = 3 ack = 3 kind = DATA info = 3        the 4th line
SWP: Sending frame: seq = 5 ack = 3 kind = DATA info = 5        the 6th line
SWP: Sending frame: seq = 6 ack = 3 kind = DATA info = 6        the 7th line
SWP: Sending frame: seq = 0 ack = 3 kind = ACK info =
SWP: Sending frame: seq = 6 ack = 3 kind = DATA info = 6        the 7th line
SWP: Sending frame: seq = 4 ack = 3 kind = DATA info = 4        the 5th line
```

Figure 6e

Java Source File (SWP.java)

```java
/*===============================================================*
 *  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.io.*;
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;

   // the following are protocol variables
   private int oldest_frame = 0;
   private PEvent event = new PEvent();
   private Packet out_buf[] = new Packet[NR_BUFS];

   //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
   private void init(){
      for (int i = 0; i < NR_BUFS; i++){
     out_buf[i] = new Packet();
      }
   }

```

```java
    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);
    }

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

   private Timer[] timers = new Timer[NR_BUFS]; //timer
   private Timer ackTimer = new Timer(); //acknoledgement timer

   static boolean between(int a, int b, int c) {
     //returns true if a <= b < c circularly; false otherwise.
     //Same as between in protocol5, but shorter and more obscure.
     return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
   }
```

```
//own written methods
//create frame and copy content of packet to it
void send_frame(int frameKind, int frame_nr, int frame_expected, Packet buffer[]) {
  //initialize a new frame
  PFrame tempFrame = new PFrame();

  //fill up the content of the frame
  tempFrame.kind = frameKind; //kind == data, ack, or nak

  if(frameKind == PFrame.DATA) {
     tempFrame.info = buffer[frame_nr % NR_BUFS];
  }
  tempFrame.seq = frame_nr; //only meaningful for data frames
  tempFrame.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);

  if (frameKind == PFrame.NAK) {
    no_nak = false; //one nak per frame, please
  }

  to_physical_layer(tempFrame); // transmit the frame

  if (frameKind == PFrame.DATA) {
    start_timer(frame_nr);  //start timer after DATA frame is sent
  }
  stop_ack_timer(); //due to piggyback, no need ack timer to send a seperate ack frame.
}

int inc(int seq) {
  //circular increment
  return (seq + 1) % (MAX_SEQ + 1);
}

public void protocol6() {
  int ack_expected; //lower edge of sender's window
  int next_frame_to_send; //upper edge of sender's window + 1
  int frame_expected; //lower edge of receiver's window
  int too_far; //upper edge of receiver's window + 1
  int i; //index into buffer pool
  PFrame r = new PFrame(); //scratch variable
  init(); //buffers for the outbound stream
  Packet in_buf[] = new Packet[NR_BUFS]; //buffers for the inbound stream
  boolean arrived[] = new boolean[NR_BUFS]; //inbound bit map
  int nbuffered; //how many output buffers currently used
  //PEvent event; //this.event

```

```java
    enable_network_layer(NR_BUFS); //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

    //set all inbound but map to false
    for (i = 0; i < NR_BUFS; i++) {
      arrived[i] = false;
    }

    while (true) {
      wait_for_event(event); //five possibilities: NETWORK_LAYER_READY, FRAME_ARRIVAL, CHECKSUM ERROR, TIMEOUT, ACK_TIMEOUT
      switch (event.type) {
        //to send
        case (PEvent.NETWORK_LAYER_READY): //accept, save, and transmit a new frame
          nbuffered++; //expand the window
          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 the 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) {
            //An undamaged frame has arrived.
            //check if frame is lost and if NaK frame has been sent yet
            //need to check no_nak to prevent sending multiple no_nak frame incase the right seq data frame haven't reach yet
            if ((r.seq != frame_expected) && no_nak) {
              send_frame(PFrame.NAK, 0, frame_expected, out_buf); //since sending one NaK frame only, it's frame seq is 0
            } else {
              //useful when:
              //1. ACK frames from receiver are all lost, receiver shifted it's sliding window.
              //2. Sender's start_timer() time out for the frames that were sent since no ACK was received for them, thus resend those DATA frames.
              //3. Since the DATA frames received is not expected by the receiver, NaK is sent to the sender
              //4. Nak sent by receiver got lost.
              //5. Sender may keep sending those DATA frames while receiver keeps rejecting them.
              //6. start_ack_timer helps to prevent deadlock.
              start_ack_timer();  //sends correct ACK to sender to resynchronize
            }
```

```java
            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
              //only pass frames and advance window once the frame for the lower edge of the receiver's sliding window has arrived
              while (arrived[frame_expected % NR_BUFS]) {
                //Pass frames and advance window.
                to_network_layer(in_buf[frame_expected % NR_BUFS]);
                no_nak = true;  //reset NaK to true since expected frame is received
                arrived[frame_expected % NR_BUFS] = false;  //reset inbound bit map for wrap around
                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 if no DATA frame are sent by the receiver to piggyback onto
              }
            }
          }

          //send DATA frame that was damaged or did not arrived again receiving NaK frame from the receiver
          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)) {
            nbuffered--; //handle piggybacked ack
            stop_timer(ack_expected); //frame arrived intact
            ack_expected = inc(ack_expected); //advance lower edge of sender's window
            enable_network_layer(1);
          }
          break;

        case (PEvent.CKSUM_ERR):
          if (no_nak)
            send_frame(PFrame.NAK, 0, frame_expected, out_buf); //send NaK for resend of frame when there is checksum error (damaged frame)
              break;

        case (PEvent.TIMEOUT):
          send_frame(PFrame.DATA, oldest_frame, frame_expected, out_buf); //timer expired; retransmit DATA frame
          break;

        case (PEvent.ACK_TIMEOUT):
          send_frame(PFrame.ACK, 0, frame_expected, out_buf); //ack timer expired; Send a seperate ACK frame
          break;

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

```java

/* 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 seqnr) {
  stop_timer(seqnr);
  timers[seqnr % NR_BUFS] = new Timer();
  timers[seqnr % NR_BUFS].schedule(new TimerTask() {
    @Override
    public void run() {
      swe.generate_timeout_event(seqnr);
    }
  }, 500);
}

private void stop_timer(int seqnr) {
  if(timers[seqnr % NR_BUFS] != null) {
    timers[seqnr % NR_BUFS].cancel();
    timers[seqnr % NR_BUFS].purge();
    timers[seqnr % NR_BUFS] = null;
  }
}

private void start_ack_timer() {
  stop_ack_timer();
  ackTimer = new Timer();
  ackTimer.schedule(new TimerTask() {
    @Override
    public void run() {
      swe.generate_acktimeout_event();
    }
  }, 150);
}

```

```java
private void stop_ack_timer() {
  if(ackTimer != null) {
    ackTimer.cancel();
    ackTimer.purge();
    ackTimer = null;
  }
}

}//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).
*/
```