LAB GROUP: SSP4

CZ3006:

NET-CENTRIC COMPUTING

Assignment 1:

Implementation of a Sliding Window Protocol

by

Alvin Lee Yong Teck (U1620768F)

Professor Sun Chengzheng

N4-02b-56

czsun@ntu.edu.sg

Date of Submission: ____________

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Lab Group **SSP4**
Name **ALVIN LEE YONG TECK**

Table of Contents

Lab Group **SSP4**
Name **ALVIN LEE YONG TECK**

SUMMARY OF ASSIGNMENT TASKS STATUS

S/N	Tasks	Status
1	Full-Duplex Data Communication	Completed
2	In-Order Delivery of Packets to the Network-Layer	Completed
3	Selective Repeat Retransmission Strategy	Completed
4	Synchronisation with the Network-Layer by Granting Credits	Completed
5	Negative Acknowledgement	Completed
6	Separate Acknowledgement when the Reverse Traffic is light or none	Completed
7	Ability to withstand quality Level 3 of the Network Simulator component	Completed

I. Introduction

This assignment aims to enhance understanding of the **network protocol hierarchy and flow control** and **error control techniques** by implementing a **sliding window protocol** in a simulated communication network system.

The simulated communication system consists of the following two major components:

a. Network Simulator

This component simulates the physical transmission media which connects two communicating virtual machines. The component may be set to operate in one of the four different quality levels of service:

- Level 0: an error-free transmission media.
- Level 1: a transmission media which may lose frames.
- Level 2: a transmission media which may damage frames (i.e., generating checksum-errors).
- Level 3: a transmission media which may lose and damage frames.

b. Virtual Machine

This component simulates a communicating virtual machine. Internally, it is divided into two sub-components:

b.1. Sliding Window Protocol

This component implements the sliding window protocol (i.e., the data link layer). In this simulated system, this component cannot work alone, and must interact with the Sliding Window Environment component in order to fetch/deliver packets from/to the upper network layer, and to fetch/deliver frames from/to the lower physical layer.

b.2. Sliding Window Protocol

This component provides the environment in which the sliding window protocol component is working. Basically, this component implements the following interfaces:

- The interface between the **data link** layer and the **network** layer.
- The interface between the **data link** layer and the **physical** layer.
- The interface between the **data link** layer and the **underlying event queue**.

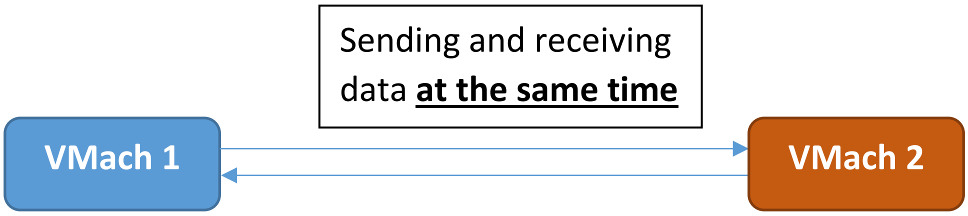
It should be pointed out that the Network Simulator component is running in one process, and the Virtual Machine component (including both the Sliding Window Protocol and the Sliding Window Environment) is running in another process. To simulate the communication between two virtual machines, two Virtual Machine processes must be executed.

II. Tasks

To implement the Sliding Window Protocol component (i.e., the data link-layer) of the simulated communication system. This component must implement all the features in the sliding window protocol, including:

a. Full-Duplex Data Communication

Full-Duplex Data Communication means that it is a bi-directional communication whereby data can be transmitted in both directions simultaneously in a single circuit.



To implement bi-directional transmission, it can be achieve by placing the receiver and sender into a single **Protocol 6** function, instead of two separate functions of receiver and sender. In addition, the **send_frame()** function is implemented separately so as to re-invoke in other parts of the code.

b. In-Order Delivery of Packets to the Network-Layer

A sequence number associated with each transmitted frame is introduced so that it can ensure that the packet delivered are in-order. The sequence numbers are **consecutive**, ranging from 0 to (2^n -1) **circularly**. The sender will have to maintain the sending window opened, and the receiver will have to maintain the receiving windows with the expected sequence number of frames to be received. After the receiver received the sequence number (which is the **same** as the one of the frame), it will send an acknowledgement back to the sender, causing the current window to close, and hence open the next sending window.

```
case (PEvent.FRAME_ARRIVAL):
    from_physical_layer(r);
    // check if frame kind is data
    if (r.kind == PFrame.DATA) {
        if ((r.seq != frame_expected) && no_nak) {
            send_frame(PFrame.NAK, 0, frame_expected, out_buf);
        }
        else {
            start_ack_timer();
        }
```

Figure 1: If data frame arrived is out-of-order, send NAK

```
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;
    in_buf[r.seq % NR_BUFS] = r.info;
    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);
        too_far = inc(too_far);
        start_ack_timer();
    }
}
```

Figure 2: If data frame arrived is in-order, transmit buffered frames to network layer

c. Selective Repeat Retransmission Strategy

To prevent the sender from being blocked during the period of waiting for transmission and processing, pipelining solution was hence introduced. It can be done by having to set a large maximum sender's window size, hence allowing the sender to transmit multiple frames until the acknowledgement for the first frame comes back, which will then make sure that the sender will always get the permission to transmit. However, the introduction of the pipelining solution raised concerns with regards to damaged/lost frame in the middle of the long stream. Thus, **selective repeat retransmission strategy** was introduced for error recovery of lost/damaged frames without the need to discard any frames.

When a frame is lost/damaged, the receiver will ***buffer(in_buf[])*** the correct subsequent frames. After receiving NAK (or timeout), the sender will retransmit the lost/damaged frame and the acknowledgement is done for the last frame buffered. The **selective repeat transmission strategy** is implemented in
Protocol 6 method.

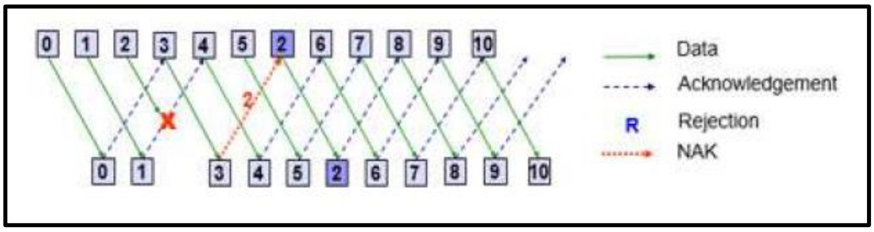


Figure 3: Selective Repeat strategy as given in CZ3006 lecture slides

d. Synchronisation with the Network-Layer by Granting Credits

The sender will have to synchronise with the network layer as it does not always have data to transmit. The sender will inform the data-link layer that it have to send frames, hence an event is generated. The data-link layer controls the network layer by preventing the network layer from sending more events until the network layer have the **credit** (the number of available buffer in the data-link layer). Additionally, the number of credits granted will be the same size as the size of the receiver's window. To implement synchronization, we will use **enable_network_layer(NR_BUFS)** to give credits to the network layer. Credit will be incremented and granted to the network layer upon receiving **complete** and **undamaged** frame.

```
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 4: Giving credits to network layer

```
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);
}
```

Figure 5: Credit incremented upon receiving complete and undamaged frame

e. Negative Acknowledgement

In cases where there are any lost/damaged frames that needs to be retransmitted, a **negative acknowledgement (NAK)** is sent by the receiver to the sender. Boolean variable **no_nak** was initially set to ***true***, indicating that no NAK has been sent yet. Variable **no_nak** will be set to ***false*** when a NAK has to be sent. NAK is sent when the frame arrives out-of-order or when the frame is lost/damaged.

```
case (PEvent.FRAME_ARRIVAL):
    from_physical_layer(r);                                  // fetch incoming frame from physical layer
    // check if frame kind is data
    if (r.kind == PFrame.DATA) {                             // An undamaged frame has arrived
        if ((r.seq != frame_expected) && no_nak) {
            send_frame(PFrame.NAK, 0, frame_expected, out_buf); // frame arrives out of order - send NAK
        }
        else {
            start_ack_timer();                               // see if separate ack is needed or not
        }
```

Figure 6: Send NAK if frame arrives lost/damaged (or out-of-order)

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

Figure 7: Frame is lost/damaged

f. Separate Acknowledgement when the Reverse Traffic is light or none

When the last frame is received successfully by the receiver after a delay, a separate acknowledgement packet is sent. In order to implement this, an acknowledgement timer will be used to keep track of the waiting time for an acknowledgement to be sent. When acknowledgement timeout, it will send a separate acknowledgement packet.

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

Figure 8: It will send a separate acknowledgement packet when acknowledgement timeout

g. Ability to withstand quality Level 3 of the Network Simulator component

Example of **NetSim 3** cmd output:

Command Prompt - java NetSim 3

```
C:\Users\Vin\Desktop\ass1A\CZ3006_ass1\src>java NetSim 3
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: 10.27.68.222 : 18094
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: 10.27.68.222 : 18095
VMach 1 loose frame seq = 1 error counter = 1
VMach 1 Check sum error for seq = 2 error counter = 2
VMach 2 Check sum error for seq = 0 error counter = 1
VMach 1 Check sum error for seq = 0 error counter = 3
VMach 1 loose frame seq = 4 error counter = 4
VMach 2 loose frame seq = 3 error counter = 2
VMach 1 Check sum error for seq = 0 error counter = 5
VMach 2 Check sum error for seq = 6 error counter = 3
VMach 2 Check sum error for seq = 0 error counter = 4
VMach 1 loose frame seq = 4 error counter = 6
VMach 1 loose frame seq = 5 error counter = 7
VMach 2 Check sum error for seq = 7 error counter = 5
VMach 1 loose frame seq = 0 error counter = 8
VMach 2 Check sum error for seq = 1 error counter = 6
VMach 2 Check sum error for seq = 6 error counter = 7
VMach 1 loose frame seq = 4 error counter = 9
VMach 1 Check sum error for seq = 5 error counter = 10
VMach 2 Check sum error for seq = 7 error counter = 8
VMach 2 Check sum error for seq = 6 error counter = 9
VMach 1 loose frame seq = 5 error counter = 11
VMach 1 Check sum error for seq = 6 error counter = 12
VMach 2 loose frame seq = 0 error counter = 10
VMach 1 Check sum error for seq = 7 error counter = 13
VMach 1 Check sum error for seq = 0 error counter = 14
VMach 1 Check sum error for seq = 6 error counter = 15
VMach 2 Check sum error for seq = 4 error counter = 11
VMach 1 loose frame seq = 0 error counter = 16
VMach 2 Check sum error for seq = 6 error counter = 12
VMach 1 Check sum error for seq = 0 error counter = 17
VMach 1 Check sum error for seq = 2 error counter = 18
```

```
VMach 1 loose frame seq = 3 error counter = 19
VMach 2 loose frame seq = 0 error counter = 13
VMach 1 Check sum error for seq = 0 error counter = 20
VMach 2 loose frame seq = 4 error counter = 14
VMach 2 Check sum error for seq = 7 error counter = 15
VMach 1 Check sum error for seq = 0 error counter = 21
VMach 1 Check sum error for seq = 1 error counter = 22
VMach 1 loose frame seq = 2 error counter = 23
VMach 2 Check sum error for seq = 0 error counter = 16
VMach 2 Check sum error for seq = 5 error counter = 17
VMach 2 loose frame seq = 0 error counter = 18
VMach 2 Check sum error for seq = 1 error counter = 19
VMach 2 loose frame seq = 2 error counter = 20
VMach 1 Check sum error for seq = 0 error counter = 24
VMach 2 loose frame seq = 3 error counter = 21
VMach 1 loose frame seq = 2 error counter = 25
VMach 1 loose frame seq = 3 error counter = 26
VMach 2 loose frame seq = 0 error counter = 22
VMach 2 loose frame seq = 0 error counter = 23
VMach 1 loose frame seq = 0 error counter = 27
VMach 1 Check sum error for seq = 2 error counter = 28
VMach 2 Check sum error for seq = 0 error counter = 24
VMach 2 Check sum error for seq = 2 error counter = 25
VMach 2 loose frame seq = 5 error counter = 26
VMach 2 loose frame seq = 5 error counter = 27
VMach 1 loose frame seq = 3 error counter = 29
VMach 1 Check sum error for seq = 4 error counter = 30
VMach 1 Check sum error for seq = 4 error counter = 31
VMach 1 loose frame seq = 6 error counter = 32
VMach 1 loose frame seq = 7 error counter = 33
VMach 2 Check sum error for seq = 0 error counter = 28
VMach 2 Check sum error for seq = 5 error counter = 29
VMach 2 loose frame seq = 0 error counter = 30
```

```
VMach 1 loose frame seq = 5 error counter = 34
VMach 1 loose frame seq = 4 error counter = 35
VMach 1 loose frame seq = 7 error counter = 36
VMach 2 Check sum error for seq = 7 error counter = 31
VMach 2 loose frame seq = 5 error counter = 32
VMach 2 loose frame seq = 0 error counter = 33
VMach 1 loose frame seq = 7 error counter = 37
VMach 1 loose frame seq = 0 error counter = 38
VMach 2 Check sum error for seq = 0 error counter = 34
VMach 2 loose frame seq = 0 error counter = 35
VMach 2 Check sum error for seq = 6 error counter = 36
VMach 2 Check sum error for seq = 7 error counter = 37
VMach 2 Check sum error for seq = 5 error counter = 38
VMach 2 Check sum error for seq = 0 error counter = 39
VMach 1 Check sum error for seq = 7 error counter = 39
VMach 1 Check sum error for seq = 1 error counter = 40
VMach 2 loose frame seq = 0 error counter = 40
VMach 1 Check sum error for seq = 5 error counter = 41
VMach 2 loose frame seq = 0 error counter = 41
VMach 2 Check sum error for seq = 6 error counter = 42
VMach 2 Check sum error for seq = 7 error counter = 43
VMach 2 loose frame seq = 5 error counter = 44
VMach 1 Check sum error for seq = 7 error counter = 42
VMach 2 loose frame seq = 0 error counter = 45
VMach 1 loose frame seq = 1 error counter = 43
VMach 1 Check sum error for seq = 4 error counter = 44
VMach 2 loose frame seq = 0 error counter = 46
VMach 1 loose frame seq = 5 error counter = 45
VMach 1 Check sum error for seq = 6 error counter = 46
VMach 2 loose frame seq = 7 error counter = 47
VMach 2 Check sum error for seq = 5 error counter = 48
VMach 1 loose frame seq = 0 error counter = 47
VMach 1 loose frame seq = 4 error counter = 48
VMach 2 Check sum error for seq = 6 error counter = 49
```

```
VMach 1 loose frame seq = 5 error counter = 49
VMach 2 loose frame seq = 7 error counter = 50
VMach 2 loose frame seq = 0 error counter = 51
VMach 2 loose frame seq = 3 error counter = 52
VMach 2 loose frame seq = 4 error counter = 53
VMach 1 loose frame seq = 4 error counter = 50
VMach 1 loose frame seq = 5 error counter = 51
VMach 1 loose frame seq = 6 error counter = 52
VMach 1 loose frame seq = 0 error counter = 53
VMach 2 loose frame seq = 0 error counter = 54
VMach 2 loose frame seq = 1 error counter = 55
VMach 1 loose frame seq = 0 error counter = 54
VMach 2 loose frame seq = 0 error counter = 56
VMach 2 Check sum error for seq = 3 error counter = 57
VMach 1 loose frame seq = 5 error counter = 55
VMach 1 Check sum error for seq = 7 error counter = 56
VMach 1 loose frame seq = 5 error counter = 57
VMach 1 loose frame seq = 0 error counter = 58
VMach 2 Check sum error for seq = 3 error counter = 58
VMach 1 Check sum error for seq = 0 error counter = 59
VMach 1 Check sum error for seq = 6 error counter = 60
VMach 2 Check sum error for seq = 0 error counter = 59
VMach 1 Check sum error for seq = 0 error counter = 61
VMach 2 loose frame seq = 0 error counter = 60
VMach 2 Check sum error for seq = 0 error counter = 61
VMach 1 loose frame seq = 0 error counter = 62
VMach 2 Check sum error for seq = 0 error counter = 62
VMach 2 loose frame seq = 0 error counter = 63
VMach 1 loose frame seq = 0 error counter = 63
VMach 1 loose frame seq = 1 error counter = 64
VMach 1 Check sum error for seq = 2 error counter = 65
VMach 2 Check sum error for seq = 0 error counter = 64
VMach 1 Check sum error for seq = 0 error counter = 66
VMach 1 Check sum error for seq = 1 error counter = 67
VMach 1 Check sum error for seq = 3 error counter = 68
```

III. Java Source Code – Sliding Window Protocol

```java
/*===============================================================================*
 * Author:         ALVIN LEE YONG TECK                                           *
 * Matric No:      U1620768F                                                     *
 * Tutorial Group: SSP4                                                          *
 * Lab Assignment: Assignment 1 - Implementation of a Sliding Window Protocol   *
 * Course Code:    CZ3006 NET CENTRIC COMPUTING                                  *
 *===============================================================================*/

/*================================================================*
 * 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.*;

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();
	}
}

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:
       * =========================================================================== */

      // No nak has been sent yet
      private boolean no_nak = true;

      // To check if the frame number is in the window (e.g. circular window range)
      private static boolean between(int a, int b, int c)
      {
            // Same as between in protocol5, but shorter and more obscure
            // the logic to check for Circular Window Range
            return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
      }

      // Function is separated to re-invoke in other parts of the code
      private void send_frame(int frame_kind, int frame_nr, int frame_expected, Packet buffer[])
      {
            // Construct and send a data, ack, or nak frame
            PFrame s = new PFrame();                              // create a new frame for the sender to send
            s.kind = frame_kind;                                  // frame_kind can be equal to data, ack, or nak

            // If the frame_kind is data, take data from one of the buffer
            if (frame_kind == PFrame.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 the frame_kind is nak, i.e. nak has been sent out (one nak per frame)
        if (frame_kind == PFrame.NAK) {
                no_nak = false;                                     // one nak per frame, please
        }

        to_physical_layer(s);                                       // sending the frame to receiver

        if (frame_kind == PFrame.DATA) {
                start_timer(frame_nr);                              // start timer for sending of data
        }

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

// To increase the frame number in a circular manner
private static int inc(int num){
        num = ((num + 1) % (MAX_SEQ + 1));
        return num;
}

public void protocol6() {
        init();

        int ack_expected = 0;                                       // lower edge of sender's window
                                                                    // next ack expected on the inbound stream

        int next_frame_to_send = 0;                                 // upper edge of sender's window + 1
                                                                    // number of next outgoing frame

        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 out_buf[] = new Packet[NR_BUFS];                   // 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 = 0;                                      // how many output buffers currently used
                                                          // initially no packets are buffered

enable_network_layer(NR_BUFS);                            // initialize

for (i = 0; i < NR_BUFS; i++) {
      arrived[i] = false;
}

while (true) {
      wait_for_event(event);                              // five possibilities: see event.type above
      switch (event.type) {
      /* ======================================
       * PEvent.NETWORK_LAYER_READY
       * - Network layer have a packet to send
       * ======================================
       */
      case (PEvent.NETWORK_LAYER_READY):                  // accept, save, and transmit a new frame
            //nbuffered = nbuffered + 1;                  // 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;

      /* ======================================
       * PEvent.FRAME_ARRIVAL
       * - A data or control frame has arrived
       * ======================================
       */
      case (PEvent.FRAME_ARRIVAL):
            from_physical_layer(r);                       // fetch incoming frame from physical layer
            // check if frame kind is data
            if (r.kind == PFrame.DATA) {                  // An undamaged frame has arrived
                  if ((r.seq != frame_expected) && no_nak) {
```

```
                    // frame arrives out of order - send NAK
                    send_frame(PFrame.NAK, 0, frame_expected, out_buf);
    }
                else {
                    start_ack_timer(); // see if separate ack is needed or not
                }
                // buffer not occupied
                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
                    // check if buffered frames are in-order
                    while (arrived[frame_expected % NR_BUFS]) {
                        // Pass frames and advance window
                        // pass to network layer if in-order
                        to_network_layer(in_buf[frame_expected % NR_BUFS]);
                        no_nak = true;                        // no nak has been sent
                        arrived[frame_expected % NR_BUFS] = false;
                        // advance lower edge of receiver's window
                        frame_expected = inc(frame_expected);
                        too_far = inc(too_far);         // advance upper edge of receiver's window
                        start_ack_timer();              // see if a separate ack is needed or not
                    }
                }
            }

            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 = nbuffered - 1;            // 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);
            }
            break;
```

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

            /* ======================================
             * PEvent.TIMEOUT
             * ======================================
             */
            case (PEvent.TIMEOUT):
                  send_frame(PFrame.DATA, oldest_frame, frame_expected, out_buf);   // timed out
                  break;

            /* ======================================
             * PEvent.ACK_TIMEOUT
             * ======================================
             */
            case (PEvent.ACK_TIMEOUT):
                  send_frame(PFrame.ACK,0,frame_expected, out_buf); // ack timer expired, send a separate 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 Timer[] clk_timer = new Timer[NR_BUFS];
private Timer ack_timer = new Timer();
private static final int ack_delay = 250;          // delay for ACK waiting for outgoing frame
private static final int delay = 500;              // delay for ACK

private void start_timer(int seq) {
	stop_timer(seq);
	clk_timer[seq % NR_BUFS] = new Timer();
	clk_timer[seq % NR_BUFS].schedule(new TempTimerTask(seq), delay);
}

private void stop_timer(int seq) {
	try{
		clk_timer[seq % NR_BUFS].cancel();
	}
	catch (Exception e){

	}
}

private void start_ack_timer() {
	stop_ack_timer();
	ack_timer = new Timer();
	ack_timer.schedule(new TimerTask() {
		public void run() {
			swe.generate_acktimeout_event();
		}
	}, ack_delay);
}

private void stop_ack_timer() {
	try {
		ack_timer.cancel();
	}
	catch (Exception e){

	}
}
```

```java
	private class TempTimerTask extends TimerTask{

		public int seq;						// keeps track of sequence number
		public TempTimerTask (int seq){
			super();
			this.seq = seq;
		}
		public void run() {
			swe.generate_timeout_event(this.seq);	// timeout event
		}

	}

}// 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).
 */
```