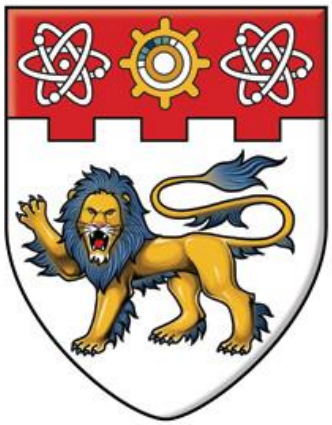

NANYANG TECHNOLOGICAL UNIVERSITY

COURSE CODE	**:**	**CZ3006**
COURSE NAME	**:**	**NET-CENTRIC COMPUTING**
NAME	**:**	**HTET NAING**
MATRICULATION NO.	**:**	**U1620683D**
LAB GROUP	**:**	**SSP4**

Contents

Assigned tasks and their status

1. Full-duplex data communication.
   Status : **Completed**

2. In-order delivery of packets to the network-layer.
   Status : **Completed**

3. Selective repeat retransmission strategy.
   Status : **Completed**

4. Synchronization with the network-layer by granting credits.
   Status : **Completed**

5. Negative acknowledgement.
   Status : **Completed**

6. Separate acknowledgment when the reverse traffic is light or none.
   Status : **Completed**

Implementation of assigned tasks

Full-duplex data transmission

Motivation to implement full-duplex data transmission arises from the necessity to transmit data bi-directionally. It can be achieved by having two separate simplex data channels. However, two separate physical circuits are required to implement them. It is not the ideal case since it leads to waste of bandwidth. The answer to overcome this bottleneck is to utilize a single circuit for transmitting data in both directions. In this assignment, the above idea is realized by having both the sender and receiver in a Single Protocol 6 function as shown below:

```
//when it is ready to transmit a new frame
case (PEvent.NETWORK_LAYER_READY):
    from_network_layer(out_buf[next_frame_to_send % NR_BUFS]);  // fetch the 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 edge of window
    break;

//when a data or control frame has arrived
case (PEvent.FRAME_ARRIVAL):
    from_physical_layer(r);  // fetch incoming frame from physical layer
```

Figure 1. Sender and Receiver in Protocol 6 (Code Snippet)

In-order delivery of packets to the network-layer

Data packets are required to be transmitted in reliable and sequential manner between two network layers. In-order delivery of packets to the network-layer can be achieved by implementing "sliding window" protocols. A sequence number is associated with all sliding window protocols and it ranges circularly from 0 up to some maximum 2^n -1. The word "window" infers "a list of consecutive sequence numbers" and its size is determined by the lower edge and the upper edge of the window.

The sender's window has sequence numbers that represent frames that have already been transmitted but are still waiting for the acknowledgement of their successful arrival. Correspondingly, the receiver's window has sequence numbers that represent frames that are expected to receive. Thus, if the sequence numbers from both the sender's window and the receiver's window are matched, it means that the data packet is successfully delivered in order and an acknowledgement (**Ack**) is generated. On the contrary, a negative acknowledgement (**Nak**) will be generated since the data packet fails to arrive in order. In this assignment, it is implemented as follows:

```
//Check if frames arrived are in order and advance window
while (arrived[frame_expected % NR_BUFS]) {
    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
}
```

Figure 2. In-order delivery of packets implementation code

Selective repeat retransmission strategy

In this assignment, selective repeat retransmission strategy is implemented in Protocol 6 method. The motivation for its implementation stems from the following circumstances. Consider a scenario where there is very long round-trip propagation delay. In such scenario, sender will be blocked for a long period of time during transmission and processing time.

The problem can be solved by a technique known as "pipelining" and it can be achieved by setting the maximum sender's window size so large that the sender is permitted to transmit multiple frames until the acknowledgement for the first frame returns. Since now multiple frames are being sent and it is very difficult to detect or recover from errors in a situation where a frame in the middle of long stream is damaged or lost. In this case, the receiver will discard all subsequent frames, sending no acknowledgements for the discarded frames. The timeout interval of the sender will expire eventually and all unacknowledged frames will be retransmitted in order. Given a high error rate, it can result in waste of a lot bandwidth.

Selective repeat retransmission strategy is introduced to address the above issue. Whenever data frames sent by the sender arrive to the receiver, it will buffer them. In other words, the receiver has a buffer reserved for each sequence number within its fixed window. It permits out-of-order arrival of data frames. In the meantime, a new acknowledgement technique is also added to the strategy. It is to send a negative acknowledgement (**Nak**) back to the sender if any data frame that is supposed to be received is missing. The purpose is to speed up the retransmission process. Therefore, the sender will always be prepared for the retransmission instead of waiting for the entire duration of the frame's timer to retransmit.

Negative acknowledgement (**Nak**) will be piggybacked with an actual acknowledgement. For instance, if arrival of **frame 0** and **frame 1** have been acknowledged but **frame 2** was lost during transmission. Then the subsequent acknowledgment will be of **frame 1** together with the piggybacked **Nak** of **frame 2** (implying only until frame 1 has been received in order)**.** Although **frame 3** or **frame 4** may have arrived and been stored in the receiver's buffers, they are not yet delivered to the network layer and hence, after their arrival, the acknowledgment sent back is still of **frame 1**. If a specific data frame is missing, only that data frame will be retransmitted. All the data frames that has successfully arrived are stored in the receiver's buffers first. Finally, despite their out-of-order arrival, they are sequentially delivered to the network layer in order. The following diagram is taken from the lecture slide and can be referred to gain more clarity of the above explanation.

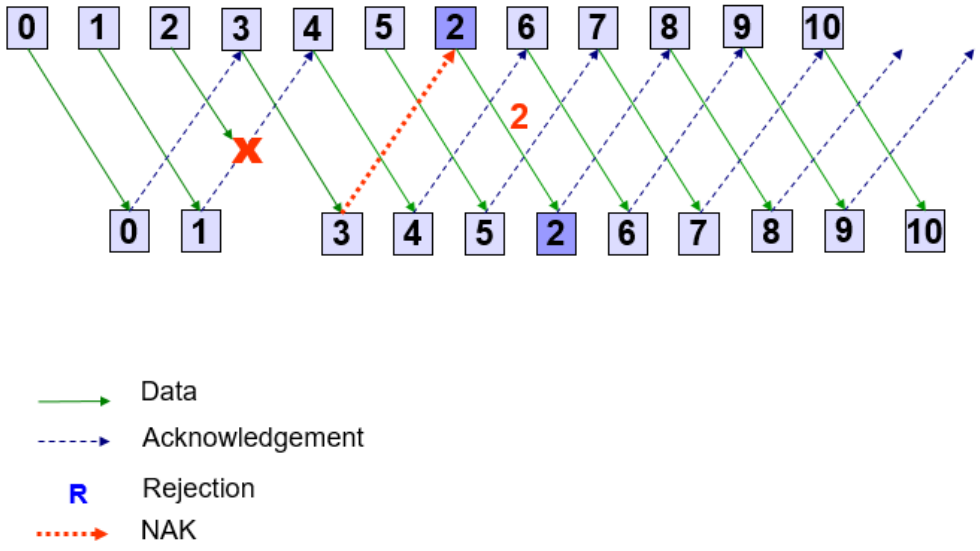

Figure 3. Selective Repeat Retransmission Strategy

Synchronization with the network-layer by granting credits

The network layer is to be synchronized with the sender as there may not always be data frames available for transmission. It can be implemented by granting credits to the network layer. The "credit" is the number of available buffers in the data link layer. The amount of credits to be granted is the same as the receiver's window size. The network layer is permitted to send if credit is available. In this assignment, it is implemented as follows:

```
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. Implementation of **enable_network_layer** method

```
//handle ack
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);                // credit incremented
}
```

Figure 5. After successful arrival of a frame, allocate one credit to the network layer, meaning that now the network layer is now permitted to send another frame.

Negative acknowledgement

The role of Negative acknowledgement (**Nak**) is already explained in detail in the **"Selective repeat retransmission strategy"** section. The purpose is to expedite the retransmission process so that the sender will always be ready for the retransmission in case of lost or damaged frames. In this assignment, it is implemented in the Protocol 6 function as follows:

```
//if the it is data frame, check whether if it necessitates to send a nak or start ack timer
   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 6. If the received frame is not the same as the expected frame, **Nak** will be sent.

```
// if it is nak frame, it means that there is negative acknowledgement hence, no_nak is no longer true
// Note only one nak per frame
if (frame_kind == PFrame.NAK) {
    no_nak = false;  //
}
```

Figure 7. If the frame received is **Nak** frame, **no_nak** should be set to false implying that there is negative acknowledgement.

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

Figure 8. If **no_nak** is true, it means that there is no negative acknowledgement yet. Hence, in case of damaged or lost frame, a negative acknowledgement can be sent.

Separate acknowledgment when the reverse traffic is light or none

Piggybacking is partially explained in **"Selective repeat retransmission strategy"** section. If a new data frame arrives fast, the acknowledgement can be piggybacked onto it. However, consider a case when there is no reverse traffic where acknowledgements can be piggybacked onto outgoing data frames. The lack of such reverse traffic can be solved with the assistance of an auxiliary timer.

Whenever a data frame arrives, an auxiliary timer is started by **start_ack_timer**. An interrupt due to the auxiliary timer is called an **ack_timeout** event. When there is no reverse traffic, a separate acknowledgement frame will be sent if no new packet has arrived by the end of this timeout period. Therefore, now the absence of reverse data frames to transmit piggybacked acknowledgements has been rectified and it also has made one-directional traffic flow possible. In this assignment, it is implemented as follows:

```
//if the it is data frame, check whether if it necessitates to send a nak or start ack timer
   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 9. Activate start_ack_timer whenever a frame arrives.

```
/* only one auxiliary timer exists,
if start_ack_timer is called while the timer is running, it will be reset to full timeout interval
hence it necessitates to stop the running timer first */
private void start_ack_timer( ) {
    stop_ack_timer();
    ack_timer = new Timer();
    /* The ack timeout interval is set to 200
    the interval has to be shorter than the frame timer interval
    so that frame retransmission timer does not expire and retransmit the frame */
    ack_timer.schedule(new TempAckTimerTask(), 200);
}
```

Figure 10. Only one auxiliary timer should exist and the timeout period should be shorter than the frame timer interval as explained in the comments.

```
//send separate ack if there is no reverse traffic
case (PEvent.ACK_TIMEOUT):
    send_frame(PFrame.ACK, 0, frame_expected, out_buf);
    break;
```

Figure 11. A separate acknowledgement will be sent when there is no reverse traffic by the end of the ack timeout period.

Output Screenshots

(to prove withstandability to quality level 3 in the Network Simulator component)

```
Last login: Mon Sep 18 20:11:34 on ttys003
htetnaing@HTETs-MacBook-Pro ~ $ cd ~/Documents/workspace/CZ3006_Assign_1/src
htetnaing@HTETs-MacBook-Pro src $ javac SWP.java
htetnaing@HTETs-MacBook-Pro src $ java NetSim 3
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: 192.168.1.89 : 53378
NetSim(Port= 54321) is waiting for connection ...
NetSim accepted connection from: 192.168.1.89 : 53380
VMach 2 loose frame seq = 0 error counter = 1
VMach 2 Check sum error for seq = 1 error counter = 2
VMach 1 loose frame seq = 1 error counter = 1
VMach 2 Check sum error for seq = 2 error counter = 3
VMach 1 Check sum error for seq = 0 error counter = 2
VMach 2 Check sum error for seq = 0 error counter = 4
VMach 2 Check sum error for seq = 0 error counter = 5
VMach 1 loose frame seq = 5 error counter = 3
VMach 1 Check sum error for seq = 7 error counter = 4
VMach 2 Check sum error for seq = 5 error counter = 6
VMach 2 Check sum error for seq = 6 error counter = 7
VMach 1 loose frame seq = 0 error counter = 5
VMach 2 loose frame seq = 0 error counter = 8
VMach 1 loose frame seq = 0 error counter = 6
VMach 1 loose frame seq = 7 error counter = 7
VMach 2 Check sum error for seq = 0 error counter = 9
VMach 2 loose frame seq = 1 error counter = 10
VMach 2 loose frame seq = 0 error counter = 11
VMach 2 Check sum error for seq = 6 error counter = 12
VMach 1 loose frame seq = 7 error counter = 8
VMach 1 Check sum error for seq = 0 error counter = 9
VMach 1 Check sum error for seq = 1 error counter = 10
VMach 1 loose frame seq = 2 error counter = 11
VMach 1 loose frame seq = 0 error counter = 12
VMach 1 loose frame seq = 7 error counter = 13
VMach 2 loose frame seq = 0 error counter = 13
VMach 1 Check sum error for seq = 0 error counter = 14
VMach 2 loose frame seq = 7 error counter = 14
VMach 1 Check sum error for seq = 2 error counter = 15
VMach 2 Check sum error for seq = 3 error counter = 15
VMach 2 Check sum error for seq = 5 error counter = 16
VMach 2 loose frame seq = 3 error counter = 17
VMach 1 Check sum error for seq = 0 error counter = 16
VMach 1 Check sum error for seq = 7 error counter = 17
VMach 2 Check sum error for seq = 4 error counter = 18
VMach 2 loose frame seq = 5 error counter = 19
VMach 1 Check sum error for seq = 0 error counter = 18
VMach 2 Check sum error for seq = 0 error counter = 20
VMach 2 Check sum error for seq = 0 error counter = 21
VMach 1 loose frame seq = 3 error counter = 19
VMach 1 Check sum error for seq = 5 error counter = 20
VMach 1 Check sum error for seq = 6 error counter = 21
VMach 2 loose frame seq = 1 error counter = 22
VMach 2 loose frame seq = 3 error counter = 23
VMach 2 loose frame seq = 4 error counter = 24
VMach 1 Check sum error for seq = 0 error counter = 22
VMach 1 Check sum error for seq = 5 error counter = 23
VMach 1 Check sum error for seq = 6 error counter = 24
```

```
VMach 2 Check sum error for seq = 6 error counter = 25
VMach 1 Check sum error for seq = 7 error counter = 25
VMach 1 Check sum error for seq = 5 error counter = 26
VMach 1 loose frame seq = 0 error counter = 27
VMach 2 loose frame seq = 4 error counter = 26
VMach 2 Check sum error for seq = 5 error counter = 27
VMach 1 Check sum error for seq = 7 error counter = 28
VMach 1 loose frame seq = 5 error counter = 29
VMach 2 Check sum error for seq = 1 error counter = 28
VMach 2 loose frame seq = 2 error counter = 29
VMach 2 loose frame seq = 3 error counter = 30
VMach 2 Check sum error for seq = 1 error counter = 31
VMach 2 loose frame seq = 4 error counter = 32
VMach 1 loose frame seq = 7 error counter = 30
VMach 2 Check sum error for seq = 0 error counter = 33
VMach 2 loose frame seq = 2 error counter = 34
VMach 2 Check sum error for seq = 3 error counter = 35
VMach 2 Check sum error for seq = 4 error counter = 36
VMach 2 Check sum error for seq = 1 error counter = 37
VMach 1 Check sum error for seq = 0 error counter = 31
VMach 2 loose frame seq = 0 error counter = 38
VMach 1 loose frame seq = 5 error counter = 32
VMach 1 Check sum error for seq = 1 error counter = 33
VMach 1 loose frame seq = 2 error counter = 34
VMach 1 loose frame seq = 3 error counter = 35
VMach 1 Check sum error for seq = 4 error counter = 36
VMach 2 Check sum error for seq = 2 error counter = 39
VMach 2 loose frame seq = 0 error counter = 40
VMach 2 loose frame seq = 7 error counter = 41
VMach 2 loose frame seq = 1 error counter = 42
VMach 2 loose frame seq = 7 error counter = 43
VMach 1 loose frame seq = 2 error counter = 37
VMach 1 Check sum error for seq = 5 error counter = 38
VMach 2 loose frame seq = 0 error counter = 44
VMach 2 Check sum error for seq = 1 error counter = 45
VMach 2 loose frame seq = 7 error counter = 46
VMach 1 Check sum error for seq = 3 error counter = 39
VMach 2 Check sum error for seq = 0 error counter = 47
VMach 2 loose frame seq = 1 error counter = 48
VMach 2 loose frame seq = 0 error counter = 49
VMach 1 Check sum error for seq = 6 error counter = 40
VMach 1 Check sum error for seq = 0 error counter = 41
VMach 1 Check sum error for seq = 0 error counter = 42
VMach 1 Check sum error for seq = 0 error counter = 43
VMach 1 loose frame seq = 0 error counter = 44
VMach 1 loose frame seq = 2 error counter = 45
VMach 1 Check sum error for seq = 3 error counter = 46
VMach 1 Check sum error for seq = 1 error counter = 47
VMach 1 loose frame seq = 0 error counter = 48
VMach 1 loose frame seq = 2 error counter = 49
VMach 1 loose frame seq = 3 error counter = 50
VMach 1 Check sum error for seq = 1 error counter = 51
VMach 1 loose frame seq = 3 error counter = 52
VMach 1 Check sum error for seq = 0 error counter = 53
VMach 1 loose frame seq = 3 error counter = 54
```

```
VMach 1 Check sum error for seq = 0 error counter = 55
VMach 1 Check sum error for seq = 1 error counter = 56
VMach 2 loose frame seq = 0 error counter = 50
VMach 2 loose frame seq = 0 error counter = 51
VMach 2 loose frame seq = 0 error counter = 52
VMach 1 Check sum error for seq = 3 error counter = 57
VMach 1 Check sum error for seq = 2 error counter = 58
VMach 2 loose frame seq = 0 error counter = 53
VMach 1 loose frame seq = 0 error counter = 59
VMach 2 Check sum error for seq = 0 error counter = 54
VMach 1 loose frame seq = 2 error counter = 60
VMach 1 Check sum error for seq = 3 error counter = 61
VMach 1 loose frame seq = 0 error counter = 62
VMach 2 loose frame seq = 0 error counter = 55
VMach 1 loose frame seq = 3 error counter = 63
VMach 1 Check sum error for seq = 1 error counter = 64
VMach 1 loose frame seq = 6 error counter = 65
VMach 1 Check sum error for seq = 1 error counter = 66
VMach 2 Check sum error for seq = 0 error counter = 56
VMach 1 Check sum error for seq = 6 error counter = 67
VMach 1 loose frame seq = 0 error counter = 68
VMach 1 loose frame seq = 1 error counter = 69
VMach 1 loose frame seq = 2 error counter = 70
VMach 1 loose frame seq = 1 error counter = 71
VMach 1 loose frame seq = 2 error counter = 72
VMach 2 Check sum error for seq = 0 error counter = 57
VMach 1 Check sum error for seq = 2 error counter = 73
VMach 2 loose frame seq = 0 error counter = 58
VMach 1 Check sum error for seq = 1 error counter = 74
VMach 1 loose frame seq = 2 error counter = 75
```

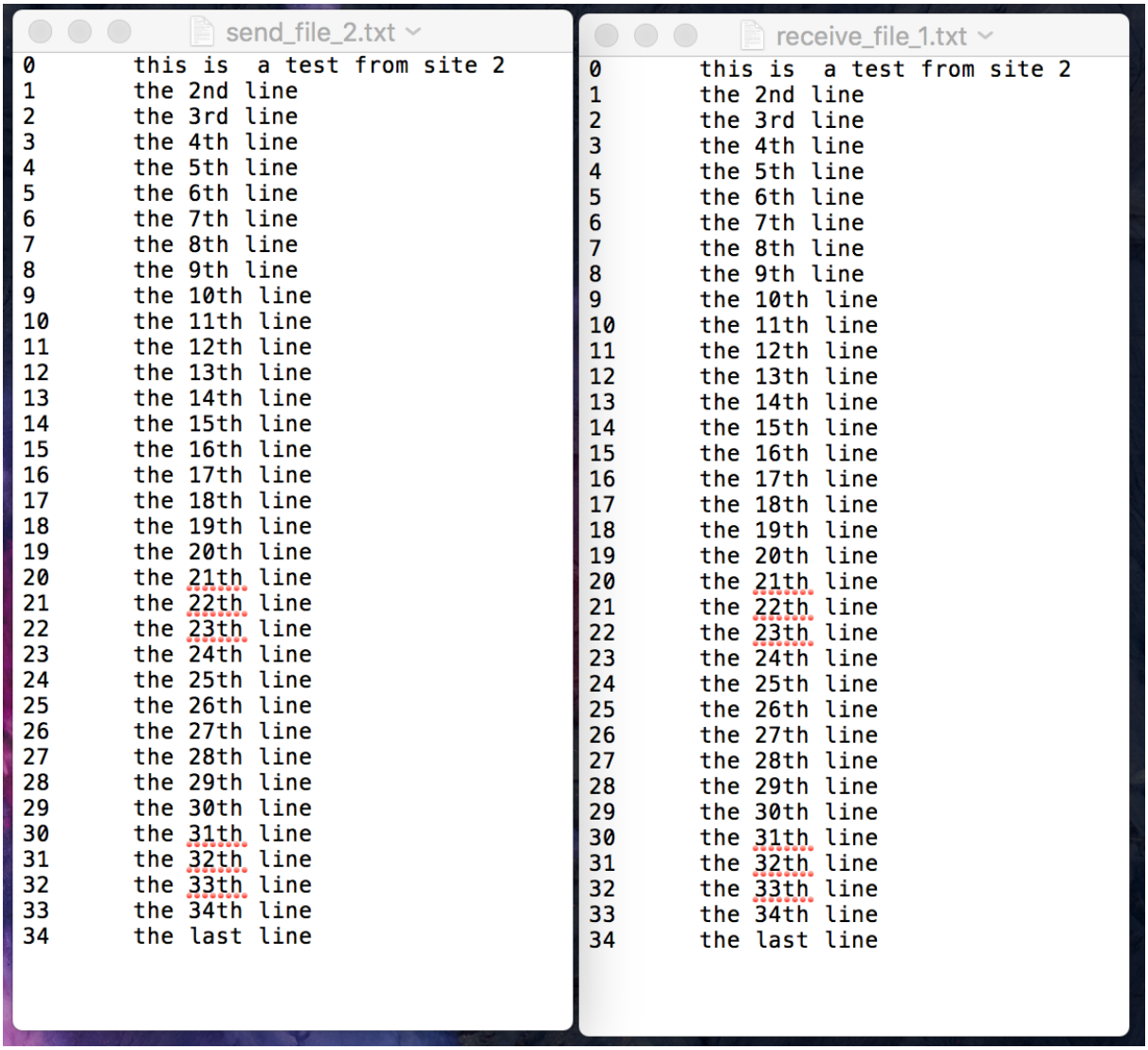

Figure 12. Tallied results between send_file_2.txt and receive_file_1.txt after running

NetSim 3

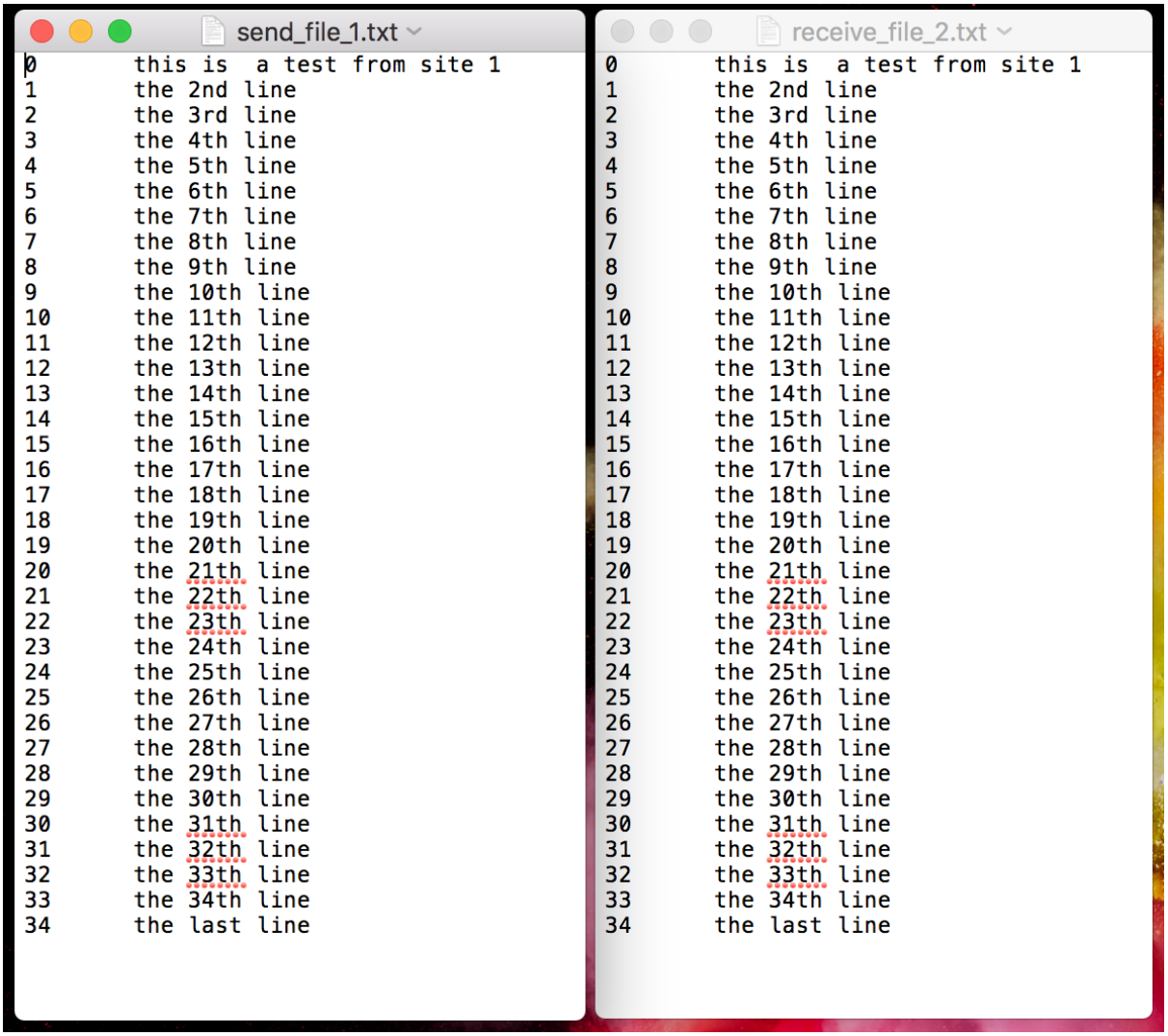

Figure 13. Tallied results between send_file_1.txt and receive_file_2.txt after running

NetSim 3

Sliding Window Protocol Java source code

```
/* Course code          :   CZ3006
 * Course name          :   Net-centric computing
 * Student Name         :   Htet Naing
 * Matriculation no.    :   U1620683D
 * Lab Group            :   SSP4
 */

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

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

   //Default state of no_nak should be true to indicate that there is no negative acknowledgement yet.
   private boolean no_nak = true;

   //Timer variables
   private Timer frame_timer[] = new Timer[NR_BUFS];
   private Timer ack_timer;

   //buffers for the inbound stream
   private Packet in_buf[] = new Packet[NR_BUFS];

   //The method below return true if a <=b < c circularly; false otherwise.
   //Essentially, it's checking if sequence numbers are above lower edge and below upper edge of the window
   static boolean between(int a, int b, int c)
   {

    return (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)));

   }

   // The method below constructs and sends a data, ack or nak frame.
   private void send_frame(int frame_kind, int frame_nr, int frame_expected, Packet buffer[]) {

     // scratch variable
    PFrame s = new PFrame();

    // frame_kind can be data, ack or nak
       s.kind = frame_kind;

       // if it is data frame, prepare to send the packet
       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 it is nak frame, it means that there is negative acknowledgement hence, no_nak is no longer true
       // Note only one nak per frame
       if (frame_kind == PFrame.NAK) {
           no_nak = false;  //
       }

       // transmit frame
       to_physical_layer(s);

       //if it is data frame, start frame timer for transmitting data
       if (frame_kind == PFrame.DATA) {
           start_timer(frame_nr);
       }
       // no need for separate ack frame
       stop_ack_timer();
   }

   //the method below increments the frame number circularly
   public static int inc(int num) {
       num = ((num + 1) % (MAX_SEQ + 1));
       return num;
   }

   public void protocol6() {

       int ack_expected = 0 ;          // lower edge of sender's window
       int next_frame_to_send  = 0;    // upper edge of sender's window + 1
       int frame_expected = 0;         // lower edge of receiver's window
       int too_far = NR_BUFS;          // upper edge of receiver's window + 1
       int index = 0;                  // index of the buffer

       boolean arrived[] = new boolean[NR_BUFS];  // keeps track of frames arrived
       PFrame r = new PFrame();  // scratch variable

       // Initialize Network Layer:
       // Allocate the credit according to number of buffers present at the data link layer
       enable_network_layer(NR_BUFS);
       for(index = 0; index < NR_BUFS; index++){
           arrived[index] = false;
           in_buf[index] = new Packet();
       }


       init(); //initially no packets are buffered
```

```
while(true) {
    wait_for_event(event);

    //there are five types of events
    switch(event.type) {

        //when it is ready to transmit a new frame
        case (PEvent.NETWORK_LAYER_READY):
            from_network_layer(out_buf[next_frame_to_send % NR_BUFS]);  // fetch the 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 edge of window
            break;

        //when a data or control frame has arrived
        case (PEvent.FRAME_ARRIVAL):
            from_physical_layer(r);  // fetch incoming frame from physical layer

         //if the it is data frame, check whether if it necessitates to send a nak or start ack timer
            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();
                }

                // Check if the frame received is within the acceptable range of the sliding window
                // And also check if it is a new frame
                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 frames arrived are in order, pass frames to network layer and advance window
                    while (arrived[frame_expected % NR_BUFS]) {
                        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
                    }
                }
            }

            // Check if the nak frame received is within the acceptable range of the sliding window
            // And resend the data of the frame if the range is valid.
            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);
            }

            //handle ack
            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);                // credit incremented
            }

            break;

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

        //if frame timer has timed out
        case (PEvent.TIMEOUT):

            send_frame(PFrame.DATA, oldest_frame, frame_expected, out_buf);
            break;

        //send separate ack if there is no reverse traffic
        case (PEvent.ACK_TIMEOUT):
            send_frame(PFrame.ACK, 0, frame_expected, out_buf);
            break;

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

```
        }

    /*
        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);
        frame_timer[seq % NR_BUFS] = new Timer();
        // The frame timer timeout interval is set to 500
        frame_timer[seq % NR_BUFS].schedule(new TempFrameTimerTask(seq), 500);
    }


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

    /* only one auxiliary timer exists,
    if start_ack_timer is called while the timer is running, it will be reset to full timeout interval
    hence it necessitates to stop the running timer first */
    private void start_ack_timer( ) {
        stop_ack_timer();
        ack_timer = new Timer();
        /* The ack timeout interval is set to 200
        the interval has to be shorter than the frame timer interval
        so that frame retransmission timer does not expire and retransmit the frame */
        ack_timer.schedule(new TempAckTimerTask(), 200);
    }

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

    public class TempAckTimerTask extends TimerTask {

        public void run(){
            swe.generate_acktimeout_event();
        }
    }

    public class TempFrameTimerTask extends TimerTask {
        public int seq;
        public TempFrameTimerTask(int seq){
            this.seq = seq;
        }
        public void run(){
            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).
*/
```

A listing of java source files

---

1. EventQueue.class
2. Forwarder.class
3. FrameHandler.class
4. NetSim.class
5. NetworkReceiver.class
6. NetworkSender.class
7. **Packet.class**
8. PacketQueue.class
9. **PEvent.class**
10. **PFrame.class**
11. **PFrame.java**
12. PFrameMsg.class
13. **SWE.class**
14. SWP$AckTask.class
15. SWP$ReTask.class
16. SWP$TempAckTimerTask.class
17. SWP$TempFrameTimerTask.class
18. SWP.class
19. **SWP.java**
20. **VMach.class**

Note: Those that are highlighted in bold are very important source files that are used in the implementation of the sliding window protocol.

Appendix

The contents from the following websites help reinforce my understanding in doing this assignment.

(1) https://www.techopedia.com/definition/24204/network-layer

(2) https://en.wikipedia.org/wiki/Sliding_window_protocol

(3) http://techdifferences.com/difference-between-go-back-n-and-selective-repeat-protocol.html

(4) https://en.wikipedia.org/wiki/Data_link_layer

(5) http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/