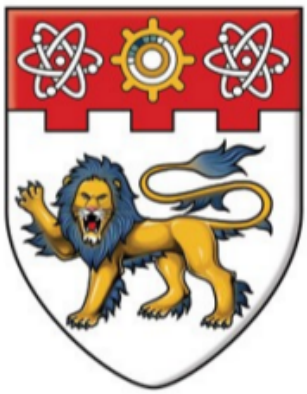
NANYANG TECHNOLOGICAL UNIVERSITY



NANYANG
TECHNOLOGICAL
UNIVERSITY

CZ3006 NET CENTRIC COMPUTING
ASSIGNMENT 1

ZHANG WANLU
U1420638G

SCHOOL OF COMPUTER ENGINEERING

1 Summary

The main purpose of this lab is to learn the network protocol hierarchy and to handle flow control as well as error control. The main content is to build a sliding window protocol in the provided communication system. The implemented protocol ensures all the required features listed below. The implementation is able to handle up to quality level 3 of the Network Simulator, which allows transmitting lose and damaged frames.

- Full-duplex data communication.
- In-order delivery of packets to the network-layer.
- Selective repeat retransmission strategy.
- Synchronization with the network-layer by granting credits.
- Negative acknowledgement.
- Separate acknowledgement when the reverse traffic is light or none.

2 Approaches

2.1 Full-duplex data communication

The full-duplex communication allows the data to be transmitted in two directions over one single channel, which means the receiver would also be a sender at the same time. The code below shows the implementation.

```
1    while(true) {
2
3      wait_for_event(event);
4
5      switch (event.type) {
6
7        case (PEvent.NETWORK_LAYER_READY):  // transmit frame
8          nbuffered++;
9          from_network_layer(out_buf[next_frame_to_send % NR_BUFS])
10         send_frame(PFrame.DATA, next_frame_to_send, frame_exp, out_buf);
11         // slide window
12         next_frame_to_send = (next_frame_to_send + 1) % (MAX_SEQ + 1);
13         break;
14
15       case (PEvent.FRAME_ARRIVAL):      // fetch frame
16         from_physical_layer(r);
17         ...
```

The technique of piggybacking is used to let the acknowledgement of the current received frame to piggybacked on to next outgoing frame. So that the data channel could have a better utilization. The code below shows the implementation.

```
1  public void send_frame(int frame_kind, int frame_nr, int frame_exp, Packet buffer
     []){
2    ...
3    s.ack = (frame_exp + MAX_SEQ) % (MAX_SEQ + 1);
4    if (frame_kind == PFrame.NAK)  {
5      no_nak = false;
6    }
7    to_physical_layer(s);        // transmit the frame
8    ...
9  }
```

2.2 In-order delivery of packets to the network-layer

SWP allows the the frames transmitted to the data link layer in a different order, while it still ensures that the packets sent to network layer are in order. This is realized by using sequence number. Those frames with higher sequence number cannot be delivered to the network layer unless all the frames with lower sequence number have been transmitted successfully. The code below shows the implementation.

```
    //if it's seq is btw the receiver's window
    // store the incoming frame into the buffer
    if (between(frame_exp, 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

        // up to the next not received frame
        while (arrived[frame_exp % NR_BUFS]){
            // pass frames and advance window
            to_network_layer(in_buf[frame_exp % NR_BUFS]);
            no_nak = true;  // allow the protocol to receive NAK
            arrived[frame_exp % NR_BUFS] = false;
            // advance lower edge of receiver's window
            frame_exp = (frame_exp + 1) % (MAX_SEQ + 1);
            // advance upper edge of receiver's window
            too_far = (too_far + 1) % (MAX_SEQ + 1);
            start_ack_timer();
        }
    }
```

2.3 Selective repeat retransmission strategy

Unlike *go back n*, the selective repeat retransmission strategy only requires for the retransmission of the damaged or lost frames, rather than discarding any other subsequent correct frames. The code below shows the implementation.

```
  case (PEvent.FRAME_ARRIVAL):
    from_physical_layer(r);     // fetch frame
    if (PFrame.KIND[r.kind].equals("DATA")){
        // send NAK if it's not the expected frame
        if ((r.seq != frame_exp) && no_nak)
            send_frame(PFrame.NAK, 0, frame_exp, out_buf);
        else
            start_ack_timer();
        // up to the next not received frame
        while (arrived[frame_exp % NR_BUFS]){
            // pass frames and advance window
                ...
          }
          ...
    case (PEvent.CKSUM_ERR):
        if (no_nak)
            // damaged frame, so send NAK
            send_frame(PFrame.NAK, 0, frame_exp, out_buf);
        break;
```

2.4 Synchronization with the network-layer by granting credit

At first, the credit granted to network layer is equal to the window size of the receiver. Once the sender is acknowledged by a successful send, one credit is grand to the network layer. Otherwise, the network layer is enabled for sending data. The code below shows the implementation.

```
  enable_network_layer(NR_BUFS);
  ...
  while (between(ack_exp, r.ack, next_frame_to_send)){
    nbuffered --;
    // frame arrive intact so stop the timers
    stop_timer(ack_exp % NR_BUFS);
    // advance the lower edge of the sender's window
    ack_exp = (ack_exp + 1) % (MAX_SEQ + 1);
    enable_network_layer(1);     // get credit
  }
```

2.5 Negative acknowledgement

When the receiver receives a damaged frame or an unexpected error, the receiver would send back a negative acknowledgement instead of the normal one, to inform the sender to resend the corresponding frame. The code below shows the implementation.

```
  // send NAK if it's not the expected frame
  if ((r.seq != frame_exp) && no_nak)
    send_frame(PFrame.NAK, 0, frame_exp, out_buf);

    ...

  case (PEvent.CKSUM_ERR):
    if (no_nak)
      // damaged frame, so send NAK
      send_frame(PFrame.NAK, 0, frame_exp, out_buf);
      break;
```

2.6 Separate acknowledgement when the reverse traffic is light or none

Sometimes, the sender might wait until timeout for the acknowledgement. This is pretty inefficient. A better way to resolve this problem is to introduce a timer for the acknowledgement. When a frame is received successfully, start the timer. When the timer times out, transmit the acknowledgement again. The code below shows the implementation.

```
  case (PEvent.FRAME_ARRIVAL):
    from_physical_layer(r);     // fetch frame
      if (PFrame.KIND[r.kind].equals("DATA")){
        // send NAK if it's not the expected frame
        if ((r.seq != frame_exp) && no_nak)
          send_frame(PFrame.NAK, 0, frame_exp, out_buf);
        else     // start the timer for acknowledgement
          start_ack_timer();
...
  case (PEvent.TIMEOUT):
    // the sender doesnt receive any ack for the data, so resend the data
    send_frame(PFrame.DATA, oldest_frame, frame_exp, out_buf);
  break;
```

3 Source Code

The following is the source code for the sliding window protocol implementation – SWP.java.

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

/*=============================================================*
 *   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                                     *
 *===============================================================*/

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);
```

```
72    }
73
74     private void from_physical_layer(PFrame fm) {
75        PFrame fm1 = swe.from_physical_layer();
76    fm.kind = fm1.kind;
77    fm.seq = fm1.seq;
78    fm.ack = fm1.ack;
79    fm.info = fm1.info;
80     }
81
82
83  /*=========================================================================*
84    implement your Protocol Variables and Methods below:
85   *=========================================================================*/
86
87    private boolean no_nak = true;
88    private Timer[] timer = new Timer[NR_BUFS];
89    private Timer ack_timer = new Timer();
90
91    public boolean between(int a, int b, int c){
92      return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
93    }
94
95    public void send_frame(int frame_kind, int frame_nr, int frame_exp, Packet buffer
        []){
96      // create a new frame for outbound frame
97      PFrame s = new PFrame();
98      // define the kind of this frame
99      s.kind = frame_kind;
100     if (frame_kind == PFrame.DATA){
101       s.info = buffer[frame_nr % NR_BUFS];
102     }
103     s.seq = frame_nr;
104     s.ack = (frame_exp + MAX_SEQ) % (MAX_SEQ + 1);
105     if (frame_kind == PFrame.NAK)  {
106       no_nak = false;
107     }
108     to_physical_layer(s);        // transmit the frame
109     if (frame_kind == PFrame.DATA)  {
110       start_timer(frame_nr);
111     }
112     stop_ack_timer();
113   }
114
115   public void protocol6() {
116     // outgoing frame's ack number from the inbound data
117     int ack_exp = 0;                // lower edge of the sender's window
118     // expected frame's seq from the inbound data
119     int frame_exp = 0;              // lower edge of the receier's window
120     int next_frame_to_send = 0;     // upper edge of the sender's window
121     int too_far = NR_BUFS;          // upper edge of the receiver's window
122     PFrame r = new PFrame();        // frame for receiving input
123     Packet in_buf[] = new Packet[NR_BUFS];     // buffer for inbound data
124     boolean arrived[] = new boolean[NR_BUFS];  // arrive or not
125     int nbuffered = 0;
126
127     enable_network_layer(NR_BUFS);
128
129     for (int i = 0; i < NR_BUFS; i++){
130       arrived[i] = false;           // nothing arrives at first
131       in_buf[i] = new Packet();     // initialization of in_buff
132     }
133
134     init();                         // initialization of out_buff
135
```

```
136     while(true) {
137
138       wait_for_event(event);
139
140       switch (event.type) {
141
142         case (PEvent.NETWORK_LAYER_READY):
143           nbuffered++;
144           from_network_layer(out_buf[next_frame_to_send % NR_BUFS]);         // fetch
      data
145           send_frame(PFrame.DATA, next_frame_to_send, frame_exp, out_buf);  // send
    data
146           next_frame_to_send = (next_frame_to_send + 1) % (MAX_SEQ + 1);     // slide
      window
147           break;
148
149         case (PEvent.FRAME_ARRIVAL):
150           from_physical_layer(r);     // fetch frame
151           if (PFrame.KIND[r.kind].equals("DATA")){
152             // send NAK if it's not the expected frame
153             if ((r.seq != frame_exp) && no_nak)
154               send_frame(PFrame.NAK, 0, frame_exp, out_buf);
155             // start the timer for acknowledgement, in case there is no outgoing
    frame that can be piggybacked
156             else
157               start_ack_timer();
158
159             // store the incoming frame into the buffer if it's seq is btw the
    receiver's window
160             if (between(frame_exp, r.seq, too_far) && (arrived[r.seq % NR_BUFS] ==
    false)){
161               // frames may be accepted in any order
162               arrived[r.seq % NR_BUFS] = true;   // mark buffer as full
163               in_buf[r.seq % NR_BUFS] = r.info;  // insert data into buffer
164
165               // up to the next not received frame
166               while (arrived[frame_exp % NR_BUFS]){
167                 // pass frames and advance window
168                 to_network_layer(in_buf[frame_exp % NR_BUFS]);
169                 no_nak = true;  // allow the protocol to receive NAK
170                 arrived[frame_exp % NR_BUFS] = false;
171                 frame_exp = (frame_exp + 1) % (MAX_SEQ + 1);  // advance lower edge
    of receiver's window
172                 too_far = (too_far + 1) % (MAX_SEQ + 1);      // advance upper edge
    of receiver's window
173                 start_ack_timer();
174               }
175             }
176
177           }
178
179           //  if receive a NAK signal
180           if (PFrame.KIND[r.kind].equals("NAK") && between(ack_exp, (r.ack+1)%(
    MAX_SEQ+1), next_frame_to_send)){
181             // resend the frame
182             send_frame(PFrame.DATA, (r.ack + 1) % (MAX_SEQ + 1), frame_exp, out_buf)
    ;
183           }
184           while (between(ack_exp, r.ack, next_frame_to_send)){
185             nbuffered --;
186             stop_timer(ack_exp % NR_BUFS);     // frame arrive intact so stop the
    timers
187             ack_exp = (ack_exp + 1) % (MAX_SEQ + 1);  // advance the lower edge of
    the sender's window
188             enable_network_layer(1);          // get credit
```

```
189             }
190             break;
191
192         case (PEvent.CKSUM_ERR):
193             if (no_nak)
194               // damaged frame, so send NAK
195               send_frame(PFrame.NAK, 0, frame_exp, out_buf);
196             break;
197
198         case (PEvent.TIMEOUT):
199             // the sender doesnt receive any ack for the data, so resend the data
200             send_frame(PFrame.DATA, oldest_frame, frame_exp, out_buf);
201             break;
202
203         case (PEvent.ACK_TIMEOUT):
204             // ack timer expired, send ack again
205             send_frame(PFrame.ACK, 0, frame_exp, out_buf);
206             break;
207
208         default:
209             System.out.println("SWP: undefined event type = " + event.type);
210             System.out.flush();
211       } // end of switch
212     }
213   }
214
215 /* Note: when start_timer() and stop_timer() are called,
216    the "seq" parameter must be the sequence number, rather
217    than the index of the timer array,
218    of the frame associated with this timer,
219    */
220
221    private void start_timer(int seq) {
222       // stop the previous indicated timer
223       stop_timer(seq);
224       // create new timer for sending frames
225       timer[seq % NR_BUFS] = new Timer();
226       // schedule the task for execution after 500 ms
227       timer[seq % NR_BUFS].schedule(new FrameTask(seq), 500);
228    }
229
230    private void stop_timer(int seq) {
231       try{
232         timer[seq % NR_BUFS].cancel();
233       } catch(Exception e) {}
234    }
235
236    private void start_ack_timer( ) {
237       stop_ack_timer();
238       ack_timer = new Timer();
239       ack_timer.schedule(new AckTask(), 300);
240    }
241
242    private void stop_ack_timer() {
243       try{
244         ack_timer.cancel();
245       } catch(Exception e) {}
246    }
247
248    class AckTask extends TimerTask {
249       @Override
250       public void run(){
251         swe.generate_acktimeout_event();
252       }
253    }
```

```
254
255     class FrameTask extends TimerTask {
256         private int seq;
257         public FrameTask(int seq){
258           super();
259           this.seq = seq;
260         }
261         @Override
262         public void run(){
263           swe.generate_timeout_event(this.seq);
264         }
265     }
266 }//End of class
267
268 /* Note: In class SWE, the following two public methods are available:
269     . generate_acktimeout_event() and
270     . generate_timeout_event(seqnr).
271
272     To call these two methods (for implementing timers),
273     the "swe" object should be referred as follows:
274       swe.generate_acktimeout_event(), or
275       swe.generate_timeout_event(seqnr).
276 */
```

4 Testing

4.1 Running Progress

The protocol was tested on all the three quality levels and all the testings were passed. The following images shows the running process for the two virtual machines under quality level 3.

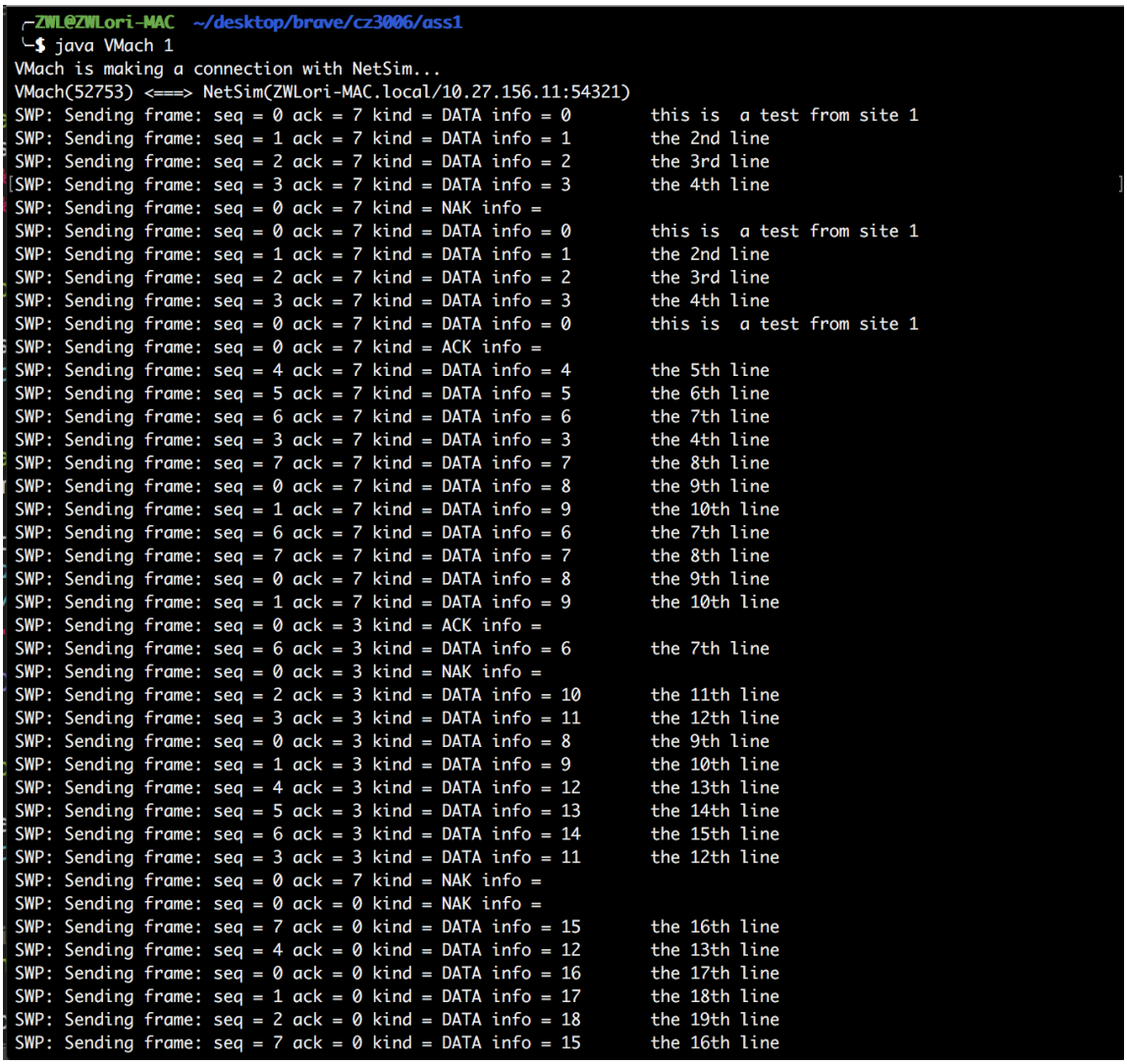


Figure 1: Virtual Machine 1 Part 1

```
SWP: Sending frame: seq = 6 ack = 3 kind = DATA info = 14       the 15th line
SWP: Sending frame: seq = 3 ack = 3 kind = DATA info = 11       the 12th line
SWP: Sending frame: seq = 0 ack = 7 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 0 kind = NAK info =
SWP: Sending frame: seq = 7 ack = 0 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 4 ack = 0 kind = DATA info = 12       the 13th line
SWP: Sending frame: seq = 0 ack = 0 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 1 ack = 0 kind = DATA info = 17       the 18th line
SWP: Sending frame: seq = 2 ack = 0 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 7 ack = 0 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 1 ack = 0 kind = DATA info = 17       the 18th line
SWP: Sending frame: seq = 0 ack = 0 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 2 ack = 0 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 7 ack = 0 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 0 ack = 2 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 4 kind = NAK info =
SWP: Sending frame: seq = 1 ack = 4 kind = DATA info = 17       the 18th line
SWP: Sending frame: seq = 2 ack = 4 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 0 ack = 4 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 7 ack = 4 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 0 ack = 4 kind = ACK info =
SWP: Sending frame: seq = 2 ack = 4 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 1 ack = 4 kind = DATA info = 17       the 18th line
SWP: Sending frame: seq = 0 ack = 4 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 7 ack = 4 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 0 ack = 4 kind = ACK info =
SWP: Sending frame: seq = 2 ack = 4 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 1 ack = 4 kind = DATA info = 17       the 18th line
SWP: Sending frame: seq = 0 ack = 4 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 7 ack = 0 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 0 ack = 0 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 0 kind = ACK info =
SWP: Sending frame: seq = 2 ack = 0 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 1 ack = 0 kind = DATA info = 17       the 18th line
SWP: Sending frame: seq = 0 ack = 0 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 7 ack = 0 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 0 ack = 0 kind = ACK info =
SWP: Sending frame: seq = 2 ack = 0 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 1 ack = 0 kind = DATA info = 17       the 18th line
SWP: Sending frame: seq = 0 ack = 0 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 7 ack = 0 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 0 ack = 4 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 4 kind = ACK info =
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
```

Figure 2: Virtual Machine 1 Part 2

```
SWP: Sending frame: seq = 0 ack = 4 kind = ACK info =
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 = 6 kind = DATA info = 22       the 23th line
SWP: Sending frame: seq = 0 ack = 6 kind = NAK info =
SWP: Sending frame: seq = 3 ack = 6 kind = DATA info = 19       the 20th line
SWP: Sending frame: seq = 0 ack = 6 kind = ACK info =
SWP: Sending frame: seq = 7 ack = 6 kind = DATA info = 23       the 24th line
SWP: Sending frame: seq = 0 ack = 6 kind = DATA info = 24       the 25th line
SWP: Sending frame: seq = 5 ack = 6 kind = DATA info = 21       the 22th line
SWP: Sending frame: seq = 1 ack = 2 kind = DATA info = 25       the 26th line
SWP: Sending frame: seq = 0 ack = 2 kind = NAK info =
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 22       the 23th line
SWP: Sending frame: seq = 0 ack = 3 kind = ACK info =
SWP: Sending frame: seq = 7 ack = 3 kind = DATA info = 23       the 24th line
SWP: Sending frame: seq = 0 ack = 3 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 3 kind = DATA info = 24       the 25th line
SWP: Sending frame: seq = 2 ack = 3 kind = DATA info = 26       the 27th line
SWP: Sending frame: seq = 3 ack = 3 kind = DATA info = 27       the 28th line
SWP: Sending frame: seq = 1 ack = 3 kind = DATA info = 25       the 26th line
SWP: Sending frame: seq = 4 ack = 7 kind = DATA info = 28       the 29th line
SWP: Sending frame: seq = 5 ack = 7 kind = DATA info = 29       the 30th line
SWP: Sending frame: seq = 6 ack = 7 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 0 ack = 2 kind = ACK info =
SWP: Sending frame: seq = 3 ack = 2 kind = DATA info = 27       the 28th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 4 ack = 2 kind = DATA info = 28       the 29th line
SWP: Sending frame: seq = 5 ack = 2 kind = DATA info = 29       the 30th line
SWP: Sending frame: seq = 3 ack = 2 kind = DATA info = 27       the 28th line
SWP: Sending frame: seq = 5 ack = 2 kind = DATA info = 29       the 30th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 4 ack = 2 kind = DATA info = 28       the 29th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 7 ack = 2 kind = DATA info = 31       the 32th line
SWP: Sending frame: seq = 0 ack = 2 kind = DATA info = 32       the 33th line
SWP: Sending frame: seq = 1 ack = 2 kind = DATA info = 33       the 34th line
SWP: Sending frame: seq = 0 ack = 2 kind = NAK info =
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 7 ack = 2 kind = DATA info = 31       the 32th line
SWP: Sending frame: seq = 0 ack = 2 kind = DATA info = 32       the 33th line
SWP: Sending frame: seq = 1 ack = 2 kind = DATA info = 33       the 34th line
SWP: Sending frame: seq = 7 ack = 2 kind = DATA info = 31       the 32th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 0 ack = 2 kind = DATA info = 32       the 33th line
```

Figure 3: Virtual Machine 1 Part 3

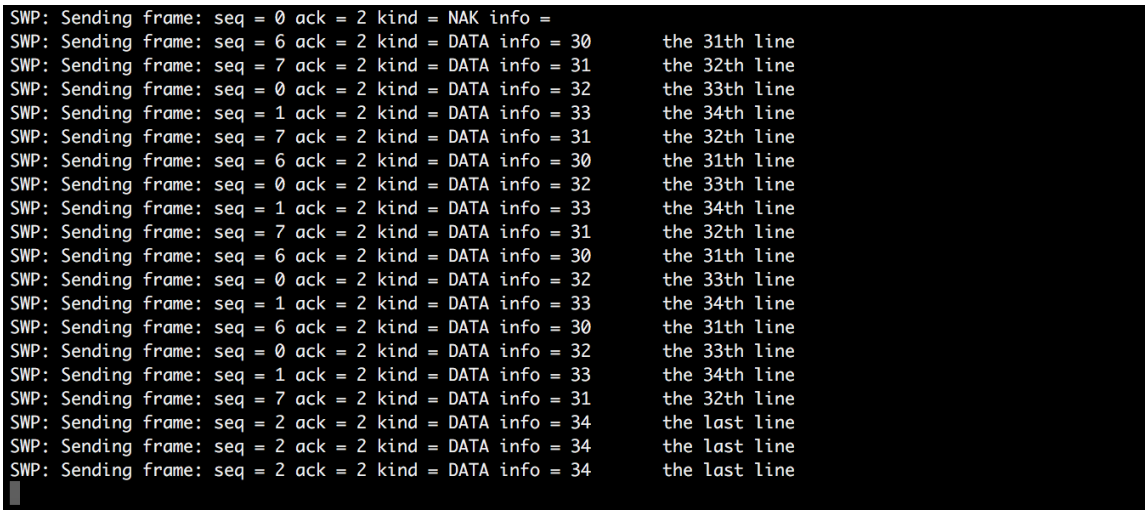


Figure 4: Virtual Machine 1 Part 4

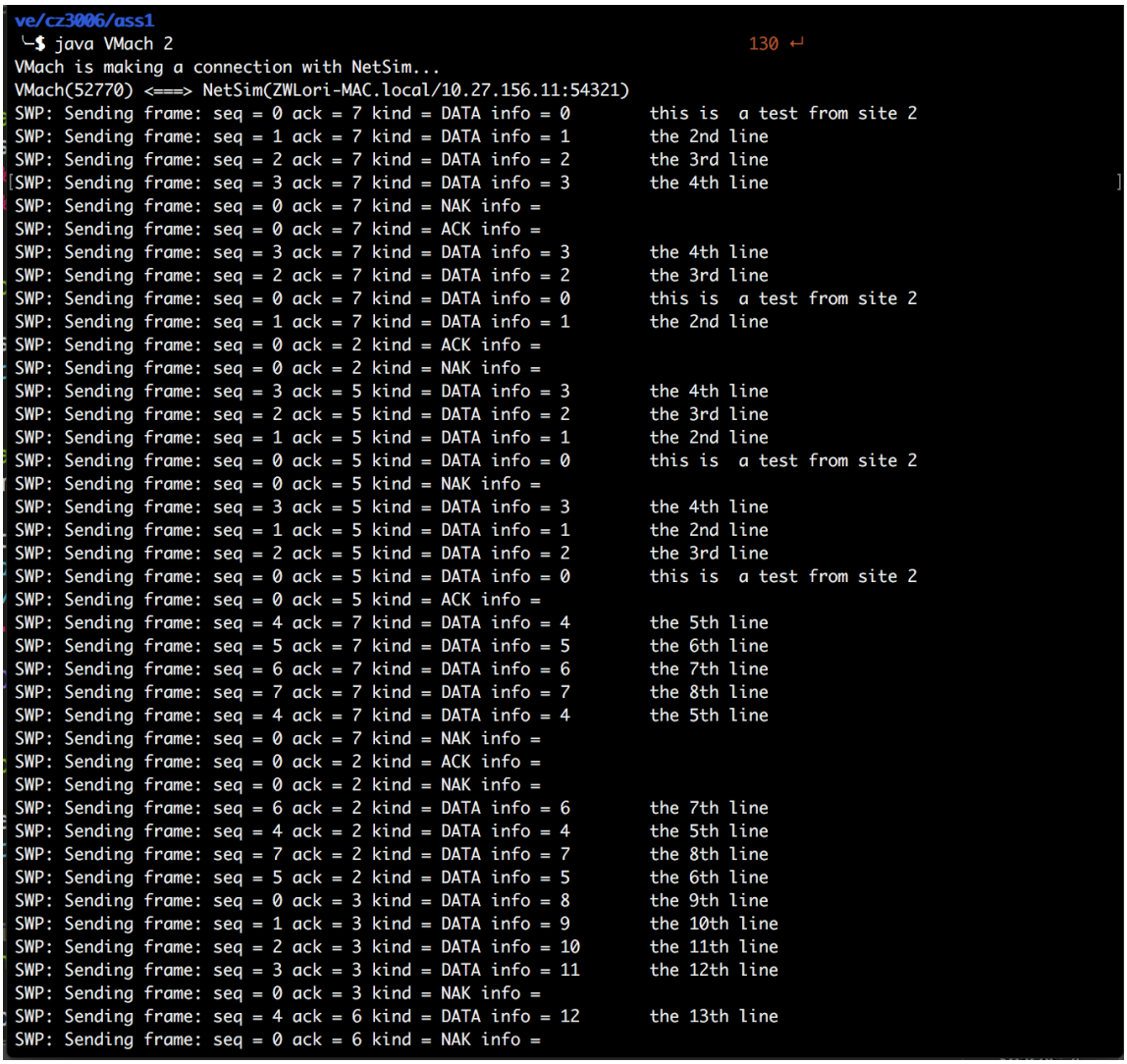


Figure 5: Virtual Machine 2 Part 1

```
SWP: Sending frame: seq = 1 ack = 3 kind = DATA info = 9        the 10th line
SWP: Sending frame: seq = 2 ack = 3 kind = DATA info = 10       the 11th line
SWP: Sending frame: seq = 3 ack = 3 kind = DATA info = 11       the 12th line
SWP: Sending frame: seq = 0 ack = 3 kind = NAK info =
SWP: Sending frame: seq = 4 ack = 6 kind = DATA info = 12       the 13th line
SWP: Sending frame: seq = 0 ack = 6 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 6 kind = ACK info =
SWP: Sending frame: seq = 1 ack = 6 kind = DATA info = 9        the 10th line
SWP: Sending frame: seq = 4 ack = 6 kind = DATA info = 12       the 13th line
SWP: Sending frame: seq = 3 ack = 6 kind = DATA info = 11       the 12th line
SWP: Sending frame: seq = 2 ack = 6 kind = DATA info = 10       the 11th line
SWP: Sending frame: seq = 0 ack = 6 kind = ACK info =
SWP: Sending frame: seq = 3 ack = 6 kind = DATA info = 11       the 12th line
SWP: Sending frame: seq = 1 ack = 6 kind = DATA info = 9        the 10th line
SWP: Sending frame: seq = 4 ack = 6 kind = DATA info = 12       the 13th line
SWP: Sending frame: seq = 2 ack = 6 kind = DATA info = 10       the 11th line
SWP: Sending frame: seq = 5 ack = 6 kind = DATA info = 13       the 14th line
SWP: Sending frame: seq = 6 ack = 6 kind = DATA info = 14       the 15th line
SWP: Sending frame: seq = 7 ack = 6 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 0 ack = 6 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 7 ack = 6 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 5 ack = 6 kind = DATA info = 13       the 14th line
SWP: Sending frame: seq = 6 ack = 6 kind = DATA info = 14       the 15th line
SWP: Sending frame: seq = 0 ack = 6 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 7 ack = 6 kind = DATA info = 15       the 16th line
SWP: Sending frame: seq = 5 ack = 6 kind = DATA info = 13       the 14th line
SWP: Sending frame: seq = 0 ack = 6 kind = DATA info = 16       the 17th line
SWP: Sending frame: seq = 6 ack = 6 kind = DATA info = 14       the 15th line
SWP: Sending frame: seq = 1 ack = 6 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 = 1 ack = 6 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 = 1 ack = 6 kind = DATA info = 17       the 18th 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 = 2 ack = 6 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 0 ack = 2 kind = ACK info =
SWP: Sending frame: seq = 5 ack = 2 kind = DATA info = 21       the 22th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 22       the 23th line
SWP: Sending frame: seq = 7 ack = 2 kind = DATA info = 23       the 24th line
SWP: Sending frame: seq = 0 ack = 2 kind = DATA info = 24       the 25th line
```

Figure 6: Virtual Machine 2 Part 2

```
SWP: Sending frame: seq = 1 ack = 6 kind = DATA info = 17       the 18th 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 = 2 ack = 6 kind = DATA info = 18       the 19th line
SWP: Sending frame: seq = 0 ack = 2 kind = ACK info =
SWP: Sending frame: seq = 5 ack = 2 kind = DATA info = 21       the 22th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 22       the 23th line
SWP: Sending frame: seq = 7 ack = 2 kind = DATA info = 23       the 24th line
SWP: Sending frame: seq = 0 ack = 2 kind = DATA info = 24       the 25th line
SWP: Sending frame: seq = 0 ack = 2 kind = NAK info =
SWP: Sending frame: seq = 1 ack = 2 kind = DATA info = 25       the 26th line
SWP: Sending frame: seq = 2 ack = 2 kind = DATA info = 26       the 27th line
SWP: Sending frame: seq = 0 ack = 4 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 4 kind = NAK info =
SWP: Sending frame: seq = 7 ack = 5 kind = DATA info = 23       the 24th line
SWP: Sending frame: seq = 0 ack = 5 kind = DATA info = 24       the 25th line
SWP: Sending frame: seq = 0 ack = 5 kind = NAK info =
SWP: Sending frame: seq = 3 ack = 5 kind = DATA info = 27       the 28th line
SWP: Sending frame: seq = 4 ack = 5 kind = DATA info = 28       the 29th line
SWP: Sending frame: seq = 5 ack = 5 kind = DATA info = 29       the 30th line
SWP: Sending frame: seq = 6 ack = 5 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 3 ack = 5 kind = DATA info = 27       the 28th line
SWP: Sending frame: seq = 0 ack = 7 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 7 kind = NAK info =
SWP: Sending frame: seq = 7 ack = 7 kind = DATA info = 31       the 32th line
SWP: Sending frame: seq = 0 ack = 2 kind = NAK info =
SWP: Sending frame: seq = 4 ack = 2 kind = DATA info = 28       the 29th line
SWP: Sending frame: seq = 5 ack = 2 kind = DATA info = 29       the 30th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 0 ack = 2 kind = DATA info = 32       the 33th line
SWP: Sending frame: seq = 1 ack = 2 kind = DATA info = 33       the 34th line
SWP: Sending frame: seq = 2 ack = 2 kind = DATA info = 34       the last line
SWP: Sending frame: seq = 0 ack = 2 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 5 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 6 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 0 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 0 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 0 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 1 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 1 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 1 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 2 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 2 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 2 kind = ACK info =
```

Figure 7: Virtual Machine 2 Part 3

4.2 Final Result

The following images show the contents of the receiving files after the transmission progress.

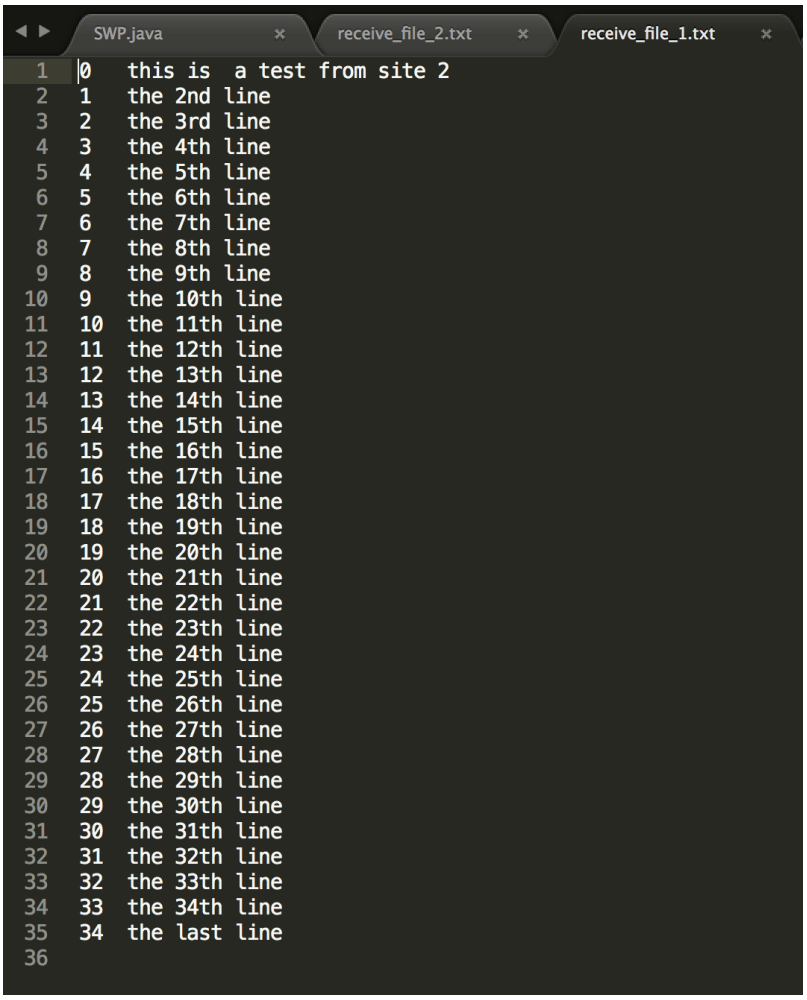


Figure 8: receive_file_1txt

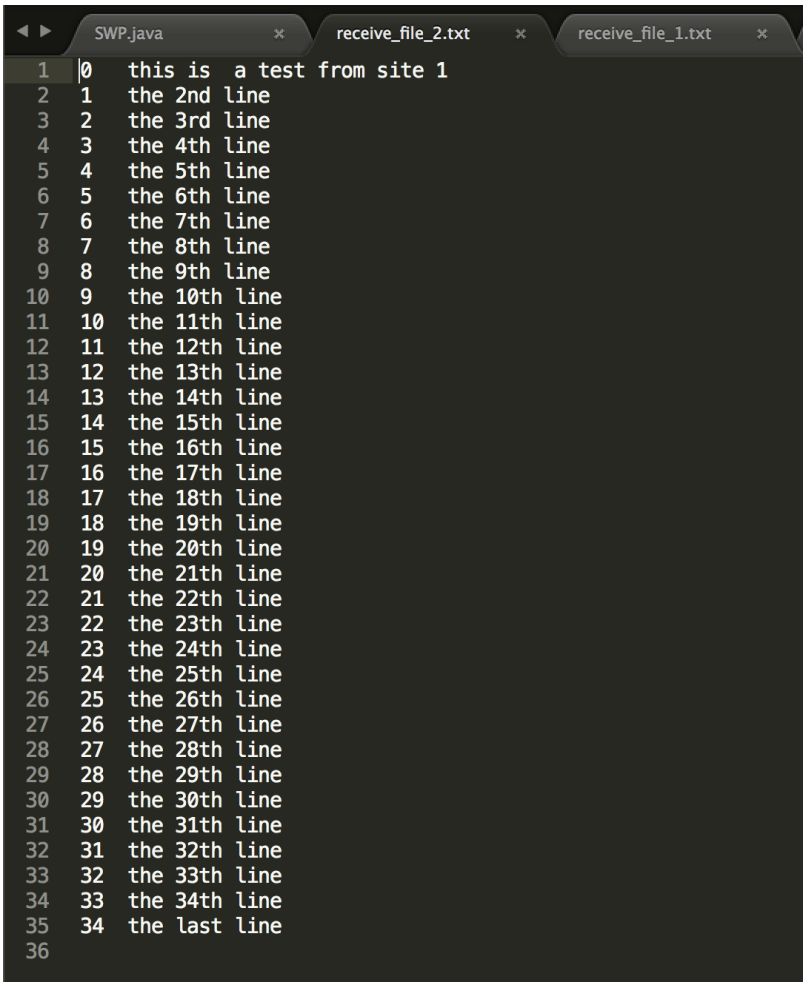


Figure 9: receive_file_2.txt

Since the text in receive_file_1.txt represents the content received by Virtual Machine 1, it should be the same as the content in the sending file from Virtual Machine 2. Similarly, what Virtual Machine 2 receives should be the same as the one stored in send_file_1.txt. As shown in the screenshots, the final result for the transmission is exactly as expected.