NANYANG
TECHNOLOGICAL
UNIVERSITY

Implementation of Sliding Window Protocol

CZ3006 – Net Centric Computing

Instructor: Prof. Sun Chengzheng

Zhang Danyang
Matric. No. U1122983C
Solo Project

Lab Group: SSP4

TABLE OF CONTENTS

1 OBJECTIVE

The objective of the CZ3006 Net Centric Computing laboratory, Implementation of Sliding Window Protocol, is to understand protocols in Data Link Layer in a fine details. Follow control, piggybagging, buffering, circular window, and error control are appreciated and examined.

In this laboratory, the protocol 6, Sliding Window Protocol, is implemented. It is simulated over communication network environment of NetSim connecting two virtual machines VMach 1 and VMach 2.

2 SUMMARY OF COMPLETENESS

Features	Completeness	Author
Full-duplex data communication (Sender is the Receiver and vice versa over single communication channel)	Completed	Zhang Danyang
In-order delivery of packets to the network-layer (Out-of-order receive from Physical Layer, and in-order deliver to Network Layer)	Completed	Zhang Danyang
Selective repeat retransmission strategy	Completed	Zhang Danyang
Synchronization with the network-layer (Granting Credit)	Completed	Zhang Danyang
Negative acknowledgement (NACK)	Completed	Zhang Danyang
Separate acknowledgment when the reverse traffic is light or none (individual ACK)	Completed	Zhang Danyang
Ability to withstand quality level 0, 1, 2, and 3 of the simulator component.	Completed	Zhang Danyang

3 APPROACHES

3.1 FULL-DUPLEX DATA COMMUNICATION

Full-duplex data communication allows data to transmit in two directions simultaneously with single channel, instead of having two separate communication. The sender is the receiver at the same time and vice versa.

```
while(true) {
   wait_for_event(event);
    switch(event.type) {
        case (PEvent.NETWORK_LAYER_READY): // sending out
          …

        case (PEvent.FRAME_ARRIVAL ): // receiving

          …
```

Piggybacking is developed so that when a data frame arrives, the acknowledgement would not be sent immediately. Instead, the acknowledgement would be piggybacked onto the next outgoing data frame to better utilize the available channel.

```
private void send_frame(int frame_kind, int frame_number, int frame_expected,
Packet out_buffer[]) {
     ...
     frame.ack = (frame_expected+MAX_SEQ)%(MAX_SEQ+1);
     ...

 }
```

3.2 IN-ORDER DELIVERY OF PACKETS TO THE NETWORK-LAYER

Sliding Window Protocol allows the frames to be received out of order while the packets would be passed to the network layer in order, provided that the sequence number of incoming frame is within the window. The frame which has higher sequence number will not be delivered to the network layer until the lower sequence number frame has been delivered.

```
if(this.between(frame_expected, frame_received.seq,
too_far)&&!arrived[frame_received.seq%NR_BUFS]) {
    // frames received may be in any order
    this.arrived[frame_received.seq%NR_BUFS] = true;
    this.in_buf[frame_received.seq%NR_BUFS] = frame_received.info;

    while(this.arrived[frame_expected%NR_BUFS]){
        /*
        Notice: Network Layer must receive the packet in order
         */
        this.to_network_layer(this.in_buf[frame_expected%NR_BUFS]);
        this.no_nak = true;
        this.arrived[frame_expected%NR_BUFS] = false;
        frame_expected = this.inc(frame_expected);
        too_far = this.inc(too_far);
        this.start_ack_timer();
    }

}
```

The *between* method is used to check if the sequence number of the frame falls into the receiver expected frames.

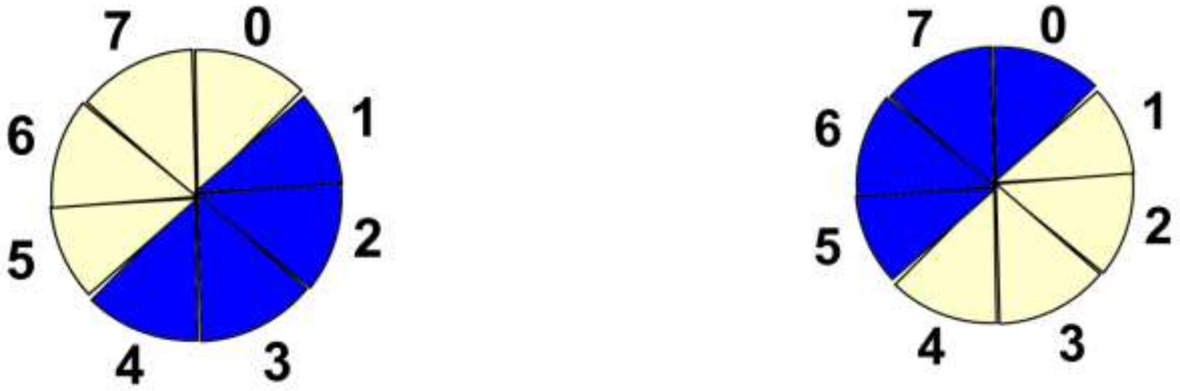


Figure 3-1 Two situtaions of *between*: a<c or a>c

```
    private boolean between(int a, int b, int c) {
        // normal situation
        if(a<c) {
            return a<=b&&b<c;
        }
        else if (c<a) {
            return a<=b || b<c;
        }
        return false;

    }
```

3.3 SYNCHRONIZATION WITH THE NETWORK-LAYER BY GRANTING CREDITS

In initialization, the credits granted to the Network Layer equals to the receiver’s window size.

```
        this.enable_network_layer(NR_BUFS);
```

When sending one frame out, one credit is deducted. When the frame in the sending window is confirmed by acknowledgment, one credit is restored.

```
 while(between(ack_expected, frame_received.ack, next_frame_to_send)) {
     stop_timer(ack_expected%NR_BUFS);
     ack_expected = this.inc(ack_expected);
     this.enable_network_layer(1); // grant credit to network_layer

 }
```

3.4 SELECTIVE REPEAT RETRANSMISSION STRATEGY

Selective repeat retransmission strategy only required the lost/damaged frames to be retransmitted instead of *go back n*, and the subsequent frames will be accepted and buffered in the receiving buffer in the receiver, provided the frame numbers are within the window.

```
if(this.between(frame_expected, frame_received.seq,
too_far)&&!arrived[frame_received.seq%NR_BUFS]) {
    // frames received may be in any order
    this.arrived[frame_received.seq%NR_BUFS] = true;
    this.in_buf[frame_received.seq%NR_BUFS] = frame_received.info;
    …

}
```

For requesting for retransmission of lost/damaged frame, a negative acknowledgment for that frame is sent as discussed in the following section.

3.5 NEGATIVE ACKNOWLEDGEMENT

For the receiver to notify the sender the incoming frame is received as an error or unexpectedly, the receiver will send a negative acknowledgement to the sender.

Out-of-order receiving is allowed but negative acknowledgment is sent, because selective repeat is designed and implemented. In the case of unexpected frame

```
if(frame_received.seq!=frame_expected&&this.no_nak) {
    this.send_frame(PFrame.NAK, 0, frame_expected, this.out_buf);

}
```

In the case of check sum error where the data in the frame is damaged:

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

3.6 SEPARATE ACKNOWLEDGMENT WHEN REVERSE TRAFFIC IS LIGHT OR NONE

It is not uncommon that the acknowledgement will be waiting for an extended period of time. To resolve this issue, an acknowledgement timer is introduced where a separate acknowledgement will be sent indicating the last frame which is received successfully.

Start timer when passing the received frame into the Network Layer (last line of the code):

```
if(this.between(frame_expected, frame_received.seq,
too_far)&&!arrived[frame_received.seq%NR_BUFS]) {
    // frames received may be in any order
    this.arrived[frame_received.seq%NR_BUFS] = true;
    this.in_buf[frame_received.seq%NR_BUFS] = frame_received.info;

    while(this.arrived[frame_expected%NR_BUFS]){
        /*
        Notice: Network Layer must receive the packet in order
         */
        this.to_network_layer(this.in_buf[frame_expected%NR_BUFS]);
        this.no_nak = true;
        this.arrived[frame_expected%NR_BUFS] = false;
        frame_expected = this.inc(frame_expected);
        too_far = this.inc(too_far);
        this.start_ack_timer();
    }

}
```

In case of time out, transmit the acknowledgment in a separate frame:

```
case (PEvent.TIMEOUT):
    this.send_frame(PFrame.DATA, this.oldest_frame, frame_expected, this.out_buf);
    break;
```

4 TESTING

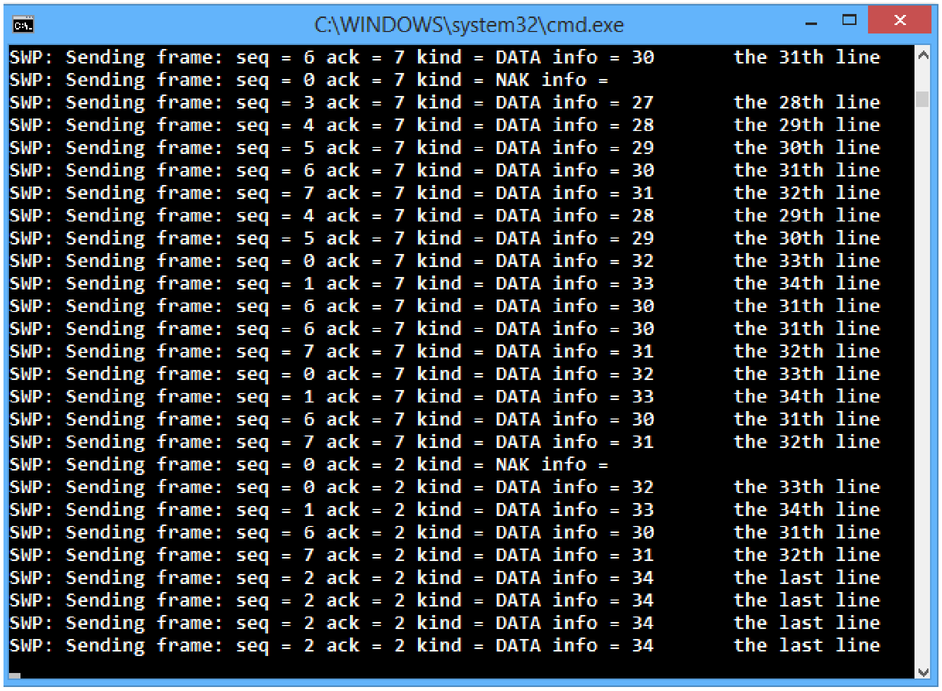
4.1 GENERAL

According to different simulated network environment, different levels of intensity were used.

- java NetSim 0: 5 times
- java NetSim 1: 5 times
- java NetSim 2: 5 times
- java NetSim 3: 25+ times

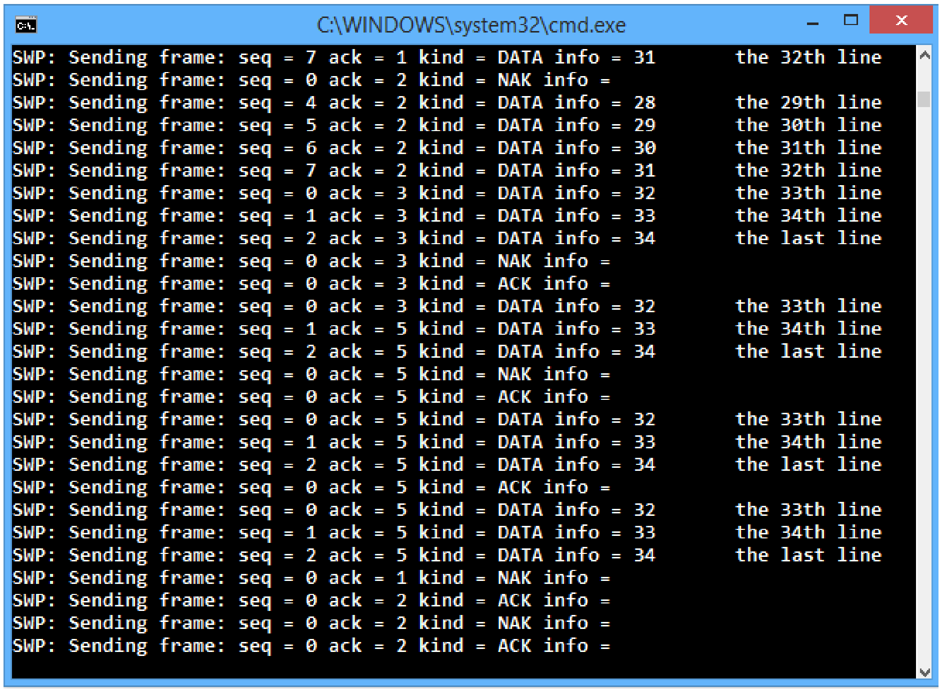
In all tests, the receiver's files were identical with the sender's files.

4.2 SAMPLE TESTING RESULTS AT LEVEL 3



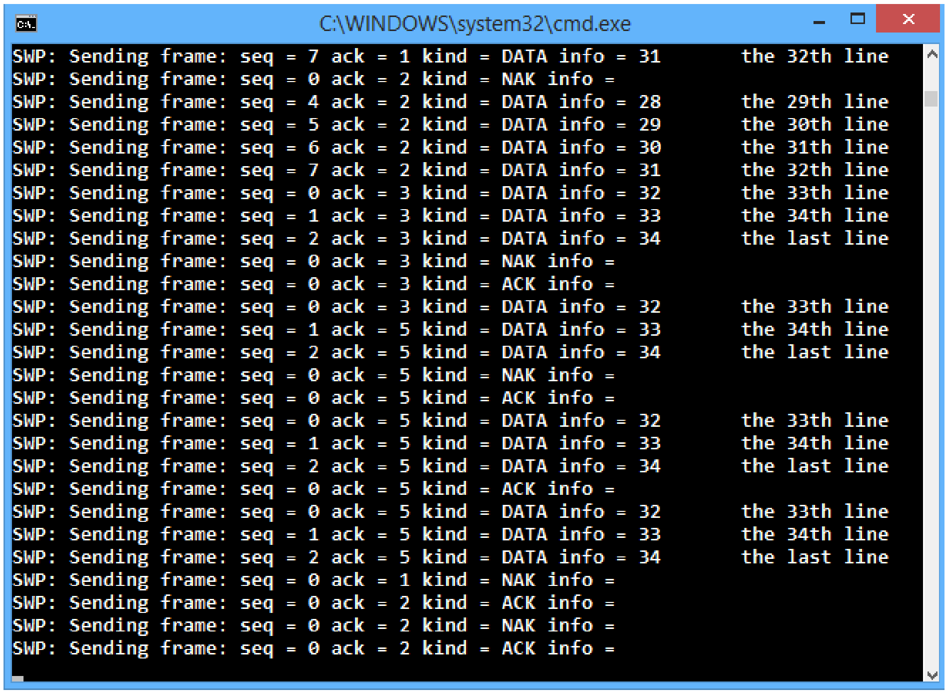
```
SWP: Sending frame: seq = 6 ack = 7 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 0 ack = 7 kind = NAK info =
SWP: Sending frame: seq = 3 ack = 7 kind = DATA info = 27       the 28th line
SWP: Sending frame: seq = 4 ack = 7 kind = DATA info = 28       the 29th line
SWP: Sending 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 = 7 ack = 7 kind = DATA info = 31       the 32th 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 = 0 ack = 7 kind = DATA info = 32       the 33th line
SWP: Sending frame: seq = 1 ack = 7 kind = DATA info = 33       the 34th line
SWP: Sending frame: seq = 6 ack = 7 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 6 ack = 7 kind = DATA info = 30       the 31th line
SWP: Sending frame: seq = 7 ack = 7 kind = DATA info = 31       the 32th line
SWP: Sending frame: seq = 0 ack = 7 kind = DATA info = 32       the 33th line
SWP: Sending frame: seq = 1 ack = 7 kind = DATA info = 33       the 34th line
SWP: Sending frame: seq = 6 ack = 7 kind = DATA info = 30       the 31th line
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 = 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 = 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 = 2 ack = 2 kind = DATA info = 34       the last line
SWP: Sending frame: seq = 2 ack = 2 kind = DATA info = 34       the last line
SWP: Sending frame: seq = 2 ack = 2 kind = DATA info = 34       the last line
SWP: Sending frame: seq = 2 ack = 2 kind = DATA info = 34       the last line
```

Figure 4-1 VMach1



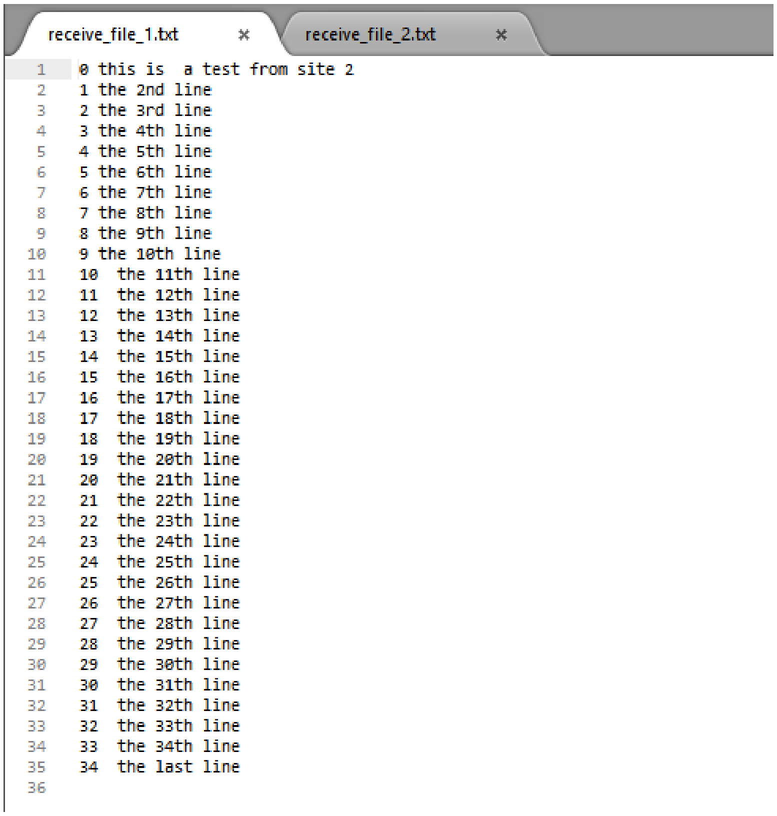
```
SWP: Sending frame: seq = 7 ack = 1 kind = DATA info = 31      the 32th line
SWP: Sending frame: seq = 0 ack = 2 kind = NAK info =
SWP: Sending frame: seq = 4 ack = 2 kind = DATA info = 28      the 29th line
SWP: Sending frame: seq = 5 ack = 2 kind = DATA info = 29      the 30th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 30      the 31th line
SWP: Sending frame: seq = 7 ack = 2 kind = DATA info = 31      the 32th line
SWP: Sending frame: seq = 0 ack = 3 kind = DATA info = 32      the 33th line
SWP: Sending frame: seq = 1 ack = 3 kind = DATA info = 33      the 34th line
SWP: Sending frame: seq = 2 ack = 3 kind = DATA info = 34      the last line
SWP: Sending frame: seq = 0 ack = 3 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 3 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 3 kind = DATA info = 32      the 33th line
SWP: Sending frame: seq = 1 ack = 5 kind = DATA info = 33      the 34th line
SWP: Sending frame: seq = 2 ack = 5 kind = DATA info = 34      the last line
SWP: Sending frame: seq = 0 ack = 5 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 5 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 5 kind = DATA info = 32      the 33th line
SWP: Sending frame: seq = 1 ack = 5 kind = DATA info = 33      the 34th line
SWP: Sending frame: seq = 2 ack = 5 kind = DATA info = 34      the last line
SWP: Sending frame: seq = 0 ack = 5 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 5 kind = DATA info = 32      the 33th line
SWP: Sending frame: seq = 1 ack = 5 kind = DATA info = 33      the 34th line
SWP: Sending frame: seq = 2 ack = 5 kind = DATA info = 34      the last line
SWP: Sending frame: seq = 0 ack = 1 kind = NAK 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 4-2 VMach 2



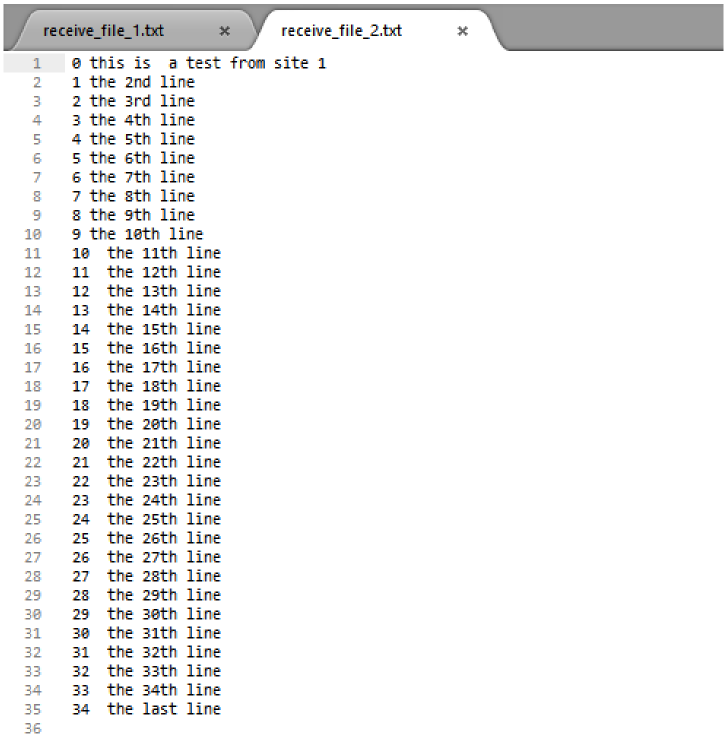
```
SWP: Sending frame: seq = 7 ack = 1 kind = DATA info = 31      the 32th line
SWP: Sending frame: seq = 0 ack = 2 kind = NAK info =
SWP: Sending frame: seq = 4 ack = 2 kind = DATA info = 28      the 29th line
SWP: Sending frame: seq = 5 ack = 2 kind = DATA info = 29      the 30th line
SWP: Sending frame: seq = 6 ack = 2 kind = DATA info = 30      the 31th line
SWP: Sending frame: seq = 7 ack = 2 kind = DATA info = 31      the 32th line
SWP: Sending frame: seq = 0 ack = 3 kind = DATA info = 32      the 33th line
SWP: Sending frame: seq = 1 ack = 3 kind = DATA info = 33      the 34th line
SWP: Sending frame: seq = 2 ack = 3 kind = DATA info = 34      the last line
SWP: Sending frame: seq = 0 ack = 3 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 3 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 3 kind = DATA info = 32      the 33th line
SWP: Sending frame: seq = 1 ack = 5 kind = DATA info = 33      the 34th line
SWP: Sending frame: seq = 2 ack = 5 kind = DATA info = 34      the last line
SWP: Sending frame: seq = 0 ack = 5 kind = NAK info =
SWP: Sending frame: seq = 0 ack = 5 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 5 kind = DATA info = 32      the 33th line
SWP: Sending frame: seq = 1 ack = 5 kind = DATA info = 33      the 34th line
SWP: Sending frame: seq = 2 ack = 5 kind = DATA info = 34      the last line
SWP: Sending frame: seq = 0 ack = 5 kind = ACK info =
SWP: Sending frame: seq = 0 ack = 5 kind = DATA info = 32      the 33th line
SWP: Sending frame: seq = 1 ack = 5 kind = DATA info = 33      the 34th line
SWP: Sending frame: seq = 2 ack = 5 kind = DATA info = 34      the last line
SWP: Sending frame: seq = 0 ack = 1 kind = NAK 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 4-3 NetSim



```
0 this is  a test from site 2
1 the 2nd line
2 the 3rd line
3 the 4th line
4 the 5th line
5 the 6th line
6 the 7th line
7 the 8th line
8 the 9th line
9 the 10th line
10  the 11th line
11  the 12th line
12  the 13th line
13  the 14th line
14  the 15th line
15  the 16th line
16  the 17th line
17  the 18th line
18  the 19th line
19  the 20th line
20  the 21th line
21  the 22th line
22  the 23th line
23  the 24th line
24  the 25th line
25  the 26th line
26  the 27th line
27  the 28th line
28  the 29th line
29  the 30th line
30  the 31th line
31  the 32th line
32  the 33th line
33  the 34th line
34  the last line
```

Figure 4-4 What VMach 1 has received

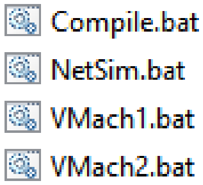


```
0 this is  a test from site 1
1 the 2nd line
2 the 3rd line
3 the 4th line
4 the 5th line
5 the 6th line
6 the 7th line
7 the 8th line
8 the 9th line
9 the 10th line
10  the 11th line
11  the 12th line
12  the 13th line
13  the 14th line
14  the 15th line
15  the 16th line
16  the 17th line
17  the 18th line
18  the 19th line
19  the 20th line
20  the 21th line
21  the 22th line
22  the 23th line
23  the 24th line
24  the 25th line
25  the 26th line
26  the 27th line
27  the 28th line
28  the 29th line
29  the 30th line
30  the 31th line
31  the 32th line
32  the 33th line
33  the 34th line
34  the last line
```

Figure 4-5 What VMach 2 has received

5 SUGGESTIONS

Students should write batch files in Windows OS or bash files in UNIX-like OS to automatically execute commands of “javac SWP.java”, “java NetSim 3”, “java VMach 1”, “java VMach 2”:



Compile.bat
NetSim.bat
VMach1.bat
VMach2.bat

Such batch/ bash files will avoid manual input of instructions in command line.

6 SOURCE CODE LISTING

6.1 INDEX

Important source files and class files:

- SWP.java: Sliding Window Protocol file.
- SWP.class: Complied class file of SWP.java.
- SWP$NormalTimerTask.class: complied class file for SWP’s internal class NormalTimerTask.
- SWP$AckTimerTask.class: complied class file for internal class AckTimerTask.

6.2 SOURCE CODE

All codes implemented are below. Please kindly note that some comments are out of line due to the width limit of MS Office Word.

```
/*===========================================================================*
    implement your Protocol Variables and Methods below:
 *==========================================================================*/
// Sending Receiving
private boolean no_nak = true;
private Packet in_buf[] = new Packet[NR_BUFS];

// Timers
private static final int NORMAL_TIMEOUT = 600;
private static final int ACK_TIMEOUT = 300;
private Timer normal_timers[] = new Timer[NR_BUFS]; // one-time timer; new Timer() every time
private Timer ack_timer; // one-time timer; new Timer() every time
```

```
/**
 * Initilize inf_buf[]
 */
private void init_in_buff() {
    for (int i=0; i<NR_BUFS; i++) {
        this.in_buf[i] = new Packet();
    }
}

private boolean arrived[] = new boolean[NR_BUFS];

/**
 * Initialize arrived[]
 */
private void init_arrived() {
    for (int i=0; i<NR_BUFS; i++) {
       this.arrived[i] = false;
    }
}

/**
 * Return true if circularly; false otherwise.
 * see if it falls within the window
 * following convention of b \in [a, c)
 * @param a int
 * @param b int
 * @param c int
 * @return boolean
 */
private boolean between(int a, int b, int c) {
    // normal situation
    if(a<c) {
        return a<=b&&b<c;
    }
    else if (c<a) {
        return a<=b || b<c;
    }
    return false;
}

/**
 * Circular increment of a sequence number over the sequence number space
 * @param seq sequence number
 * @return seuqnce number + 1
 */
private int inc(int seq) {
    return (seq+1)%(MAX_SEQ+1);
}

/**
 * Send the frame from output buffer to the physical layer
 * Piggybacking the ACK
 * @param frame_kind DATA, ACK, NAK
 * @param frame_number sequence number, ACK NAK frame by default frame_number = 0;
 * @param frame_expected to calculate the acknowledgement number (the one before it)
 * @param out_buffer from which the output frame is extracted
 */
private void send_frame(int frame_kind, int frame_number, int frame_expected, Packet
out_buffer[]) {
    PFrame frame = new PFrame(); // scratch
    frame.kind = frame_kind; // DATA, ACK, NAK
    if (frame_kind==PFrame.DATA)  {
        frame.info = out_buffer[frame_number%NR_BUFS];
    }
    frame.seq = frame_number;
    frame.ack = (frame_expected+MAX_SEQ)%(MAX_SEQ+1); // one before the frame_expected //
piggybacking the ack
    if (frame_kind==PFrame.NAK) {
        this.no_nak = false;
    }
    this.to_physical_layer(frame);
    if (frame_kind==PFrame.DATA) { // start timer only after sending
        this.start_timer(frame_number); // frame number correspond to the sequence number
    }
    this.stop_ack_timer(); // piggybagged
}

/**
 * Sliding Window Protocol (i.e. protocol 6)
 */
public void protocol6() {
    this.init(); // initialize the out buffer
    this.init_in_buff(); // initialize the input buffer
    this.init_arrived(); // initialize the arrived array
    // Send
    // e.g. 0 1 2 | 3 4 5 6 | 7 8
    int ack_expected = 0; // lower edge to the sender's window
    int next_frame_to_send = 0; // upper edge of sender's windows + 1 // sequence number
    // Receive
    int frame_expected = 0; // lower edge of receiver's window
    int too_far = NR_BUFS; // upper edge of receiver's window + 1

    PFrame frame_received = new PFrame();  // Scratch
    // int nr_output_buffered = 0; // how many output buffers currently used // it is written
but never read

    this.enable_network_layer(NR_BUFS);
    while(true) {
        wait_for_event(event);
        switch(event.type) {
            case (PEvent.NETWORK_LAYER_READY): // sending out
                /*
                Whenever a new packet arrives from the network layer, it is given the next
highest sequence number, and
                the upper edge of the window is advanced by one
                 */
                // nr_output_buffered++;
                this.from_network_layer(out_buf[next_frame_to_send%NR_BUFS]);
                this.send_frame(PFrame.DATA, next_frame_to_send, frame_expected,
this.out_buf);
                next_frame_to_send = inc(next_frame_to_send);
                break;
            case (PEvent.FRAME_ARRIVAL ): // receiving
                this.from_physical_layer(frame_received);
                if(frame_received.kind==PFrame.DATA) {
                    // An undamanged frame has arrived
                             if(frame_received.seq!=frame_expected&&this.no_nak) {
                                 this.send_frame(PFrame.NAK, 0, frame_expected, this.out_buf);
// seq number not expected
                             }
                    else {
                        this.start_ack_timer();
                    }

                    if(this.between(frame_expected, frame_received.seq, 
too_far)&&!arrived[frame_received.seq%NR_BUFS]) {
                        // frames received may be in any order
                        this.arrived[frame_received.seq%NR_BUFS] = true;
                        this.in_buf[frame_received.seq%NR_BUFS] = frame_received.info;

                        while(this.arrived[frame_expected%NR_BUFS]){
                            /*
                            Pass frames and advance window.
                            [expected, .., frame_received, .., too far]

                            When a frame whose sequence number is equal to the lower edge of 
the window
                            is received, it is passed to the network layer, an acknowledgement 
is generated (timer start),
                            and the window is rotated by one.

                            Notice: Network Layer must receive the packet in order
                             */
                            this.to_network_layer(this.in_buf[frame_expected%NR_BUFS]);
                            this.no_nak = true;
                            this.arrived[frame_expected%NR_BUFS] = false;
                            // frame_expected++; // should do circular increment
                            frame_expected = this.inc(frame_expected);
                            too_far = this.inc(too_far);
                            this.start_ack_timer();
                        }
                    }
                }
                if(frame_received.kind==PFrame.NAK&&between(ack_expected, 
(frame_received.ack+1)%(MAX_SEQ+1), next_frame_to_send)) {
                    /*
                    frame_received.ack is all frames received correctly received and 
acknowledged
                    nak for the
                     */
                    this.send_frame(PFrame.DATA, (frame_received.ack+1)%(MAX_SEQ+1), 
frame_expected, this.out_buf); // retransmit the lost DATA
                }

                while(between(ack_expected, frame_received.ack, next_frame_to_send)) {
                    /*
                    When an acknowledgement comes in, the lower edge is advanced by one.
                    Acknowledgement received for the series of frames [lower, .., ack] (ACK 
not for single frame)
                     */
                    // nr_output_buffered--;
                    stop_timer(ack_expected%NR_BUFS);
                    ack_expected = this.inc(ack_expected); // looping until the ack_expected 
== frame_received.ack + 1
                    this.enable_network_layer(1); // grant credit to network_layer // suspect 
losing frame if the frame is not in order
                }
                break;
            case (PEvent.CKSUM_ERR):
                if(this.no_nak) {
                    this.send_frame(PFrame.NAK, 0, frame_expected, this.out_buf); // frame 
DATA damaged
                }
                break;
            case (PEvent.TIMEOUT):
                this.send_frame(PFrame.DATA, this.oldest_frame, frame_expected, this.out_buf);
                break;
            case (PEvent.ACK_TIMEOUT):
                this.send_frame(PFrame.ACK, 0, frame_expected, this.out_buf); // retransmit
ACK if have waited too long for piggybacking
                break;
            default:
                System.out.println("SWP: undefined event type = "  + event.type);
                System.out.flush();
        }
        // Enable Disable buffer
    }
}

/* 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,
*/

/**
 * Start a normal timer for frame
 * @param seq sequence number
 */
private void start_timer(int seq) {
    this.stop_timer(seq);
    this.normal_timers[seq%NR_BUFS] = new Timer();
    this.normal_timers[seq%NR_BUFS].schedule(new NormalTimerTask(seq), NORMAL_TIMEOUT);
}

/**
 * Stop a normal timer for frame
 * @param seq sequence number
 */
private void stop_timer(int seq) {
    if(this.normal_timers[seq%NR_BUFS]!=null) {
        this.normal_timers[seq%NR_BUFS].cancel();
        // this.normal_timers[seq%NR_BUFS] = null;
    }
}

/**
 * Start a ack timer for a frame
 */
private void start_ack_timer() {
    this.stop_ack_timer();
    this.ack_timer = new Timer();
    this.ack_timer.schedule(new AckTimerTask(), ACK_TIMEOUT);
}

/**
 * Stop a ack timber for a frame
 */
private void stop_ack_timer() {
    if(this.ack_timer!=null) {
        this.ack_timer.cancel();
        // this.ack_timer = null;
    }
}
```

```java
// Internal Classes
private class NormalTimerTask extends TimerTask {
    private int seq;

    private NormalTimerTask(int seq) {
        super(); // following python convention
        this.seq = seq;
    }

    @Override
    public void run() {
        SWP.this.stop_timer(this.seq);
        SWP.this.swe.generate_timeout_event(seq);
    }
}

private class AckTimerTask extends TimerTask {
    @Override
    public void run() {
        SWP.this.stop_ack_timer();
        SWP.this.swe.generate_acktimeout_event();
    }

}
```