

**CZ 3006 Net-Centric Computing  
Lab Assignment 1**

Lab Group : TS2  
Submitted by : John Michael Estrella Vega  
 U1721312J  
 &  
 Kwek Dan Yi, Germaine  
 U1722265L

Submission Deadline: 15 March 19

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING  
NANYANG TECHNOLOGICAL UNIVERSITY

Table of Content s

[Task Summary 3](#_Toc2725441)

[Implementation 4](#_Toc2725442)

[1. Full Duplex Data Communication 4](#_Toc2725443)

[2. In-order delivery of packets to the network-layer 5](#_Toc2725444)

[3. Selective repeat retransmission strategy. 5](#_Toc2725445)

[4. Synchronization with the network-layer by granting credits 5](#_Toc2725446)

[5. Negative acknowledgement 6](#_Toc2725447)

[6. Separate acknowledgment when the reverse traffic is light or none 6](#_Toc2725448)

[Approach 6](#_Toc2725449)

[John’s Summary 6](#_Toc2725450)

[Germaine’s Summary 7](#_Toc2725451)

[Work Distribution 7](#_Toc2725452)

[Program Outputs 7](#_Toc2725453)

[Java Source code for Sliding Window Protocol 13](#_Toc2725454)

[Java source Files Listing 20](#_Toc2725455)

# **Task Summary**

1. Full-duplex data communication.

Status: **Completed**

1. In-order delivery of packets to the network-layer.

Status: **Completed**

1. Selective repeat retransmission strategy.

Status: **Completed**

1. Synchronization with the network-layer by granting credits.

Status: **Completed**

1. Negative acknowledgement.

Status: **Completed**

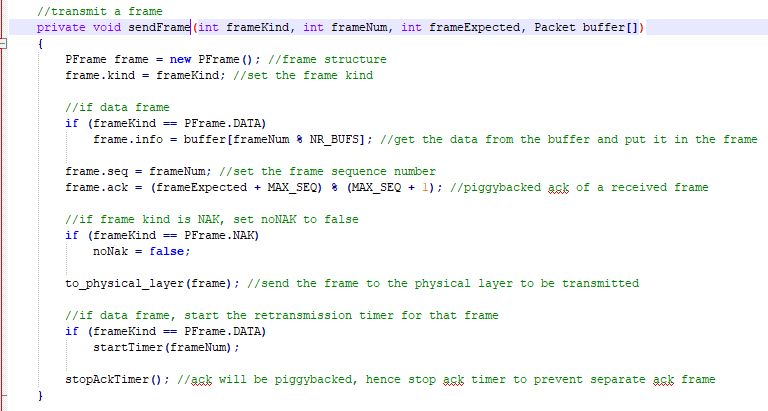
1. Separate acknowledgment when the reverse traffic is light or none.

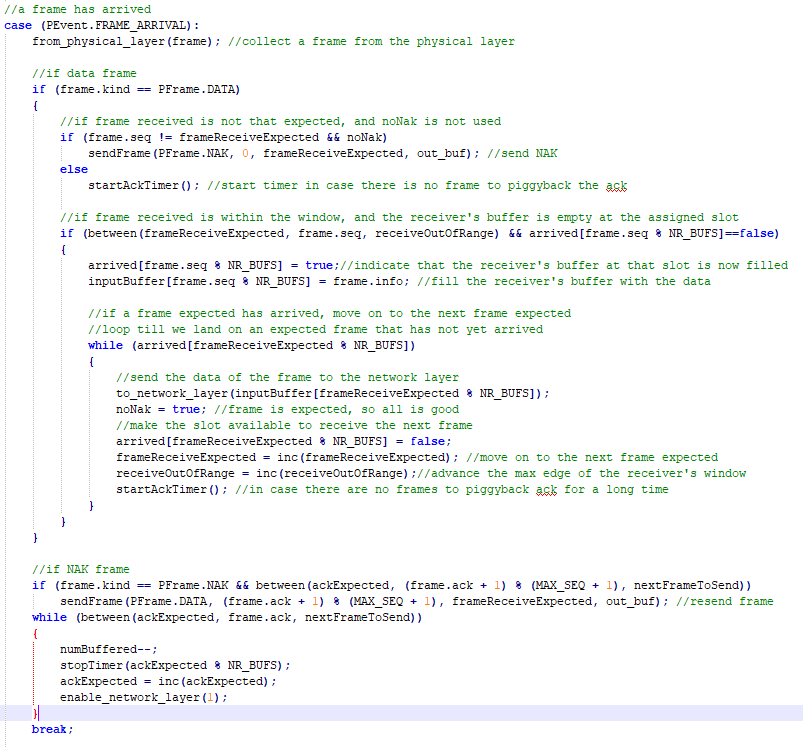
Status: **Completed**

# **Implementation**

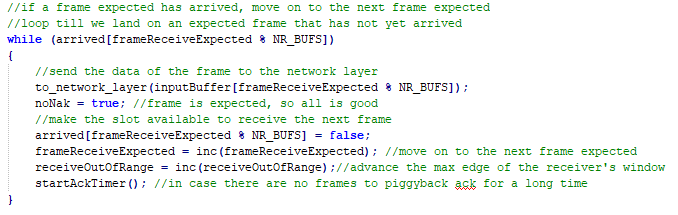
## 1. Full Duplex Data Communication

Our code makes sure that each virtual machine sends frames using the sendFrame() method, and receives frames using the case PEvent.FRAME\_ARRIVAL.  
  
Below is a snippet of the code showing the sendFrame() method.



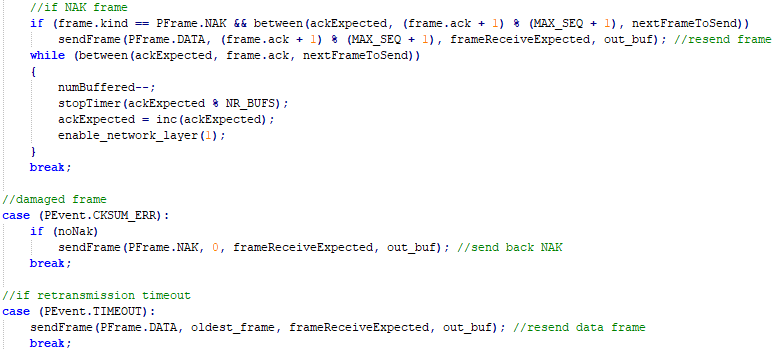
A snippet of the case PEvent.FRAME\_ARRIVAL is shown below.  


## 2. In-order delivery of packets to the network-layer

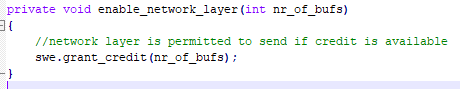
In the snippet below, we can see that inputBuffer can accept the frames out of order. Each frame depending on its sequence number will have an assigned slot in the buffer, which ensures that even if the frames arrive out of order, they will be held in order in the buffer, and thus sent to the network layer in order.  
  


## 3. Selective repeat retransmission strategy.

Instead of protocol 5 where all frames that arrive out of order are rejected, protocol 6 only requests for retransmission for that NAK or damaged frame or in the case of a timeout event.



## 4. Synchronization with the network-layer by granting credits

The method below grants the number of credits to a network layer allowing it to generate the same number of packets.  
  


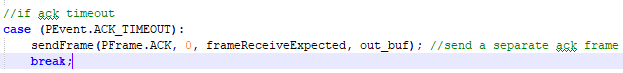
## 5. Negative acknowledgement

Should a frame be received as a negative acknowledgement, it will be retransmitted under the selective repeat strategy.  
  
  
  
Upon receiving a frame, should it not be the one expected, an NAK should be sent to the sender, as seen below.



## 6. Separate acknowledgment when the reverse traffic is light or none

When ack timeout occurs before a frame can be found to piggyback the acknowledgement, then a separate ack frame will be sent to prevent retransmission which will happen when the retransmission timer expires on the sender’s side.



# **Approach**

## John’s Summary

Our early meetings revolved around discussions on how to go about approaching the project. Realising that most of the information could be found in the lecture slides, we both identified the various methods that need to be used in protocol6() to achieve our selective repeat retransmission strategy.   
  
Methods include inc(), used to increment sequence numbers, sendFrame() , as the name suggests to send frames, between() which is used to check if sequence numbers is within window range, startTimer() and stopTimer() which are used to handle retransmission, and startAckTimer() and stop AckTimer() which are used to handle sending separate acknowledgement frames.  
  
Jointly we proceeded to implement the method protocol6(). We started off declaring and initializing the variables and arrays to be used in the method. We decided that by writing some of the code for protocol6() together, it would be less daunting. Among the switch cases in the protocol, we also implemented case(PEvent.Frame\_Arrival) together as it was a big case and complicated to understand at first. At the end we also commented the code together.

For my part, I implemented methods inc(), sendFrame(),beween() and case(PEvent.NETWORK\_LAYER\_READY) and case(PEvent.ACK\_TIMEOUT).

## Germaine’s Summary

We first read the code given in the lecture slides and identified the methods needed to implement the protocol6() method. We identified these methods as inc() which is used to increment sequence numbers, sendFrame() which is used to send frames, between() which is used to check if a sequence number is within the window range, startTimer() and stopTimer() which are used to handle retransmission, and startAckTimer() and stopAckTimer() which are used to handle sending a separate acknowledgement frame.

We then proceeded to implement the method protocol6(). We first declared and initialised the variables and arrays to be used in the protocol. We then proceeded to implement the switch statement which switches between handling different event types. We implemented each of the case statements.

Now we will describe how we split the work.

Jointly: Read and understand the code given in the lecture slides, and identify methods to be used in protocol6(). We also declared and initialised variables and arrays in protocol6() together, and implemented the case(PEvent.FRAME\_ARRIVAL) together. Commented on the code(inline documentation) together.

John: Implemented methods inc(), sendFrame(), between(). Also implemented case(PEvent.NETWORK\_LAYER\_READY) and case(PEvent.ACK\_TIMEOUT).

Germaine: Implemented methods startTimer(), stopTimer(), startAckTimer(), stopAckTimer(). Also implemented case(PEvent.CKSUM\_ERR) and case(PEvent.TIMEOUT).

# **Work Distribution**

Joint Work

Understand code from the lecture slides, method identification for protocol 6(). Variable and array initialization and naming. Implementation of case(PEvent.FRAME\_ARRIVAL). Comments and inline documentation.

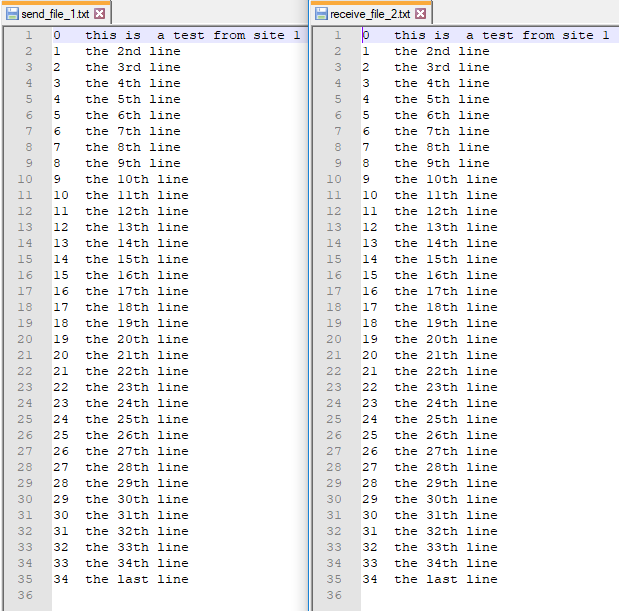
John’s Work  
  
Implementation of methods inc(), sendFrame(), between(), case(PEvent.NETWORK\_LAYER\_READY) and case( PEVENT.ACK\_TIMEOUT).  
  
Germaine’s Work

Implementation of methods startTImer(), stopTimer(), startAckTimer(), stopAckTimer(),case(PEvent.CKSUM\_ERR) and case(PEvent.TIMEOUT).

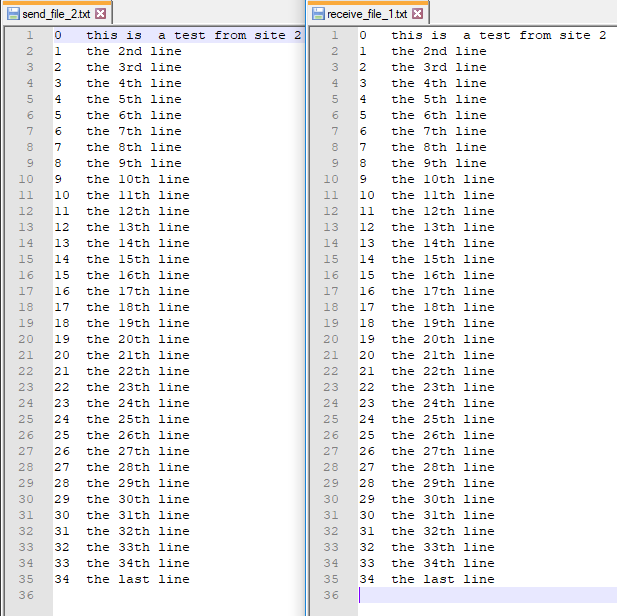
# **Program Outputs**

The implementation withstands quality level 3. The output of the cmd.exe is shown below.  
  
C:\Users\John\Desktop\ass1>java NetSim 3  
NetSim(Port= 54321) is waiting for connection ...  
NetSim accepted connection from: 10.27.130.250 : 57080  
NetSim(Port= 54321) is waiting for connection ...  
NetSim accepted connection from: 10.27.130.250 : 57081  
VMach 1 loose frame seq = 1 error counter = 1  
VMach 2 loose frame seq = 2 error counter = 1  
VMach 2 Check sum error for seq = 3 error counter = 2  
VMach 2 loose frame seq = 0 error counter = 3  
VMach 1 loose frame seq = 4 error counter = 2  
VMach 1 Check sum error for seq = 0 error counter = 3  
VMach 1 loose frame seq = 3 error counter = 4  
VMach 2 Check sum error for seq = 2 error counter = 4  
VMach 2 Check sum error for seq = 2 error counter = 5  
VMach 2 Check sum error for seq = 3 error counter = 6  
VMach 1 Check sum error for seq = 5 error counter = 5  
VMach 1 Check sum error for seq = 6 error counter = 6  
VMach 1 loose frame seq = 7 error counter = 7  
VMach 1 Check sum error for seq = 4 error counter = 8  
VMach 2 Check sum error for seq = 5 error counter = 7  
VMach 1 Check sum error for seq = 6 error counter = 9  
VMach 2 loose frame seq = 2 error counter = 8  
VMach 1 Check sum error for seq = 7 error counter = 10  
VMach 2 Check sum error for seq = 0 error counter = 9  
VMach 1 loose frame seq = 0 error counter = 11  
VMach 2 loose frame seq = 2 error counter = 10  
VMach 1 Check sum error for seq = 1 error counter = 12  
VMach 2 Check sum error for seq = 5 error counter = 11  
VMach 1 loose frame seq = 0 error counter = 13  
VMach 1 Check sum error for seq = 3 error counter = 14  
VMach 2 loose frame seq = 0 error counter = 12  
VMach 2 loose frame seq = 4 error counter = 13  
VMach 2 Check sum error for seq = 2 error counter = 14  
VMach 2 Check sum error for seq = 3 error counter = 15  
VMach 1 loose frame seq = 5 error counter = 15  
VMach 1 loose frame seq = 3 error counter = 16  
VMach 1 loose frame seq = 4 error counter = 17  
VMach 2 loose frame seq = 4 error counter = 16  
VMach 2 Check sum error for seq = 3 error counter = 17  
VMach 1 loose frame seq = 5 error counter = 18  
VMach 2 loose frame seq = 1 error counter = 18  
VMach 1 loose frame seq = 3 error counter = 19  
VMach 1 Check sum error for seq = 0 error counter = 20  
VMach 2 loose frame seq = 2 error counter = 19  
VMach 2 loose frame seq = 4 error counter = 20  
VMach 1 Check sum error for seq = 0 error counter = 21  
VMach 2 loose frame seq = 1 error counter = 21  
VMach 1 Check sum error for seq = 4 error counter = 22  
VMach 2 loose frame seq = 0 error counter = 22  
VMach 1 Check sum error for seq = 1 error counter = 23  
VMach 2 Check sum error for seq = 4 error counter = 23  
VMach 1 Check sum error for seq = 1 error counter = 24  
VMach 2 Check sum error for seq = 0 error counter = 24  
VMach 2 Check sum error for seq = 1 error counter = 25  
VMach 1 loose frame seq = 2 error counter = 25  
VMach 1 Check sum error for seq = 4 error counter = 26  
VMach 1 Check sum error for seq = 1 error counter = 27  
VMach 2 loose frame seq = 0 error counter = 26  
VMach 2 Check sum error for seq = 1 error counter = 27  
VMach 1 Check sum error for seq = 3 error counter = 28  
VMach 1 loose frame seq = 1 error counter = 29  
VMach 2 loose frame seq = 0 error counter = 28  
VMach 2 Check sum error for seq = 1 error counter = 29  
VMach 2 loose frame seq = 3 error counter = 30  
VMach 1 Check sum error for seq = 5 error counter = 30  
VMach 1 loose frame seq = 0 error counter = 31  
VMach 2 loose frame seq = 1 error counter = 31  
VMach 2 Check sum error for seq = 2 error counter = 32  
VMach 2 Check sum error for seq = 3 error counter = 33  
VMach 1 loose frame seq = 0 error counter = 32  
VMach 2 Check sum error for seq = 0 error counter = 34  
VMach 2 loose frame seq = 1 error counter = 35  
VMach 2 Check sum error for seq = 2 error counter = 36  
VMach 2 Check sum error for seq = 4 error counter = 37  
VMach 1 Check sum error for seq = 2 error counter = 33  
VMach 2 Check sum error for seq = 0 error counter = 38  
VMach 2 Check sum error for seq = 1 error counter = 39  
VMach 2 Check sum error for seq = 2 error counter = 40  
VMach 1 loose frame seq = 4 error counter = 34  
VMach 1 loose frame seq = 5 error counter = 35  
VMach 2 Check sum error for seq = 3 error counter = 41  
VMach 1 Check sum error for seq = 2 error counter = 36  
VMach 2 loose frame seq = 0 error counter = 42  
VMach 2 Check sum error for seq = 1 error counter = 43  
VMach 2 Check sum error for seq = 2 error counter = 44  
VMach 1 loose frame seq = 0 error counter = 37  
VMach 1 Check sum error for seq = 1 error counter = 38  
VMach 2 Check sum error for seq = 0 error counter = 45  
VMach 2 Check sum error for seq = 5 error counter = 46  
VMach 2 Check sum error for seq = 7 error counter = 47  
VMach 1 loose frame seq = 0 error counter = 39  
VMach 2 Check sum error for seq = 5 error counter = 48  
VMach 2 loose frame seq = 0 error counter = 49  
VMach 1 loose frame seq = 0 error counter = 40  
VMach 1 Check sum error for seq = 2 error counter = 41  
VMach 1 loose frame seq = 1 error counter = 42  
VMach 2 loose frame seq = 0 error counter = 50  
VMach 2 loose frame seq = 5 error counter = 51  
VMach 2 Check sum error for seq = 7 error counter = 52  
VMach 1 Check sum error for seq = 2 error counter = 43  
VMach 1 loose frame seq = 1 error counter = 44  
VMach 2 loose frame seq = 0 error counter = 53  
VMach 2 loose frame seq = 5 error counter = 54  
VMach 2 Check sum error for seq = 6 error counter = 55  
VMach 1 loose frame seq = 0 error counter = 45  
VMach 2 Check sum error for seq = 7 error counter = 56  
VMach 1 loose frame seq = 0 error counter = 46  
VMach 2 loose frame seq = 5 error counter = 57  
VMach 2 loose frame seq = 7 error counter = 58  
VMach 1 Check sum error for seq = 0 error counter = 47  
VMach 2 Check sum error for seq = 5 error counter = 59  
VMach 2 Check sum error for seq = 0 error counter = 60  
VMach 2 Check sum error for seq = 7 error counter = 61  
VMach 2 Check sum error for seq = 6 error counter = 62  
VMach 2 loose frame seq = 0 error counter = 63  
VMach 2 Check sum error for seq = 6 error counter = 64  
VMach 1 loose frame seq = 0 error counter = 48  
VMach 2 Check sum error for seq = 5 error counter = 65  
VMach 2 Check sum error for seq = 7 error counter = 66  
VMach 2 loose frame seq = 6 error counter = 67  
VMach 1 loose frame seq = 0 error counter = 49  
VMach 2 Check sum error for seq = 5 error counter = 68  
VMach 2 loose frame seq = 0 error counter = 69  
VMach 2 Check sum error for seq = 6 error counter = 70  
VMach 1 Check sum error for seq = 0 error counter = 50  
VMach 2 Check sum error for seq = 0 error counter = 71  
VMach 2 Check sum error for seq = 2 error counter = 72  
VMach 2 loose frame seq = 4 error counter = 73  
VMach 2 loose frame seq = 2 error counter = 74  
VMach 2 loose frame seq = 5 error counter = 75  
VMach 2 Check sum error for seq = 3 error counter = 76  
VMach 2 Check sum error for seq = 2 error counter = 77  
VMach 2 Check sum error for seq = 3 error counter = 78  
VMach 2 Check sum error for seq = 2 error counter = 79  
VMach 2 Check sum error for seq = 4 error counter = 80  
VMach 2 Check sum error for seq = 5 error counter = 81  
VMach 2 Check sum error for seq = 2 error counter = 82  
VMach 2 Check sum error for seq = 3 error counter = 83  
VMach 2 loose frame seq = 4 error counter = 84  
VMach 2 Check sum error for seq = 5 error counter = 85  
VMach 1 loose frame seq = 0 error counter = 51  
VMach 2 Check sum error for seq = 4 error counter = 86  
VMach 2 loose frame seq = 2 error counter = 87  
VMach 2 Check sum error for seq = 4 error counter = 88  
VMach 2 loose frame seq = 0 error counter = 89  
VMach 2 loose frame seq = 1 error counter = 90  
VMach 2 loose frame seq = 2 error counter = 91  
VMach 2 loose frame seq = 1 error counter = 92  
VMach 1 Check sum error for seq = 0 error counter = 52  
VMach 2 Check sum error for seq = 2 error counter = 93

The results of running NetSim 3 between send\_file\_1.txt and receive\_file\_2.txt are shown below.



The results of running NetSim 3 between send\_file\_2.txt and receive\_file\_1.txt are shown below.



# 

# **Java Source code for Sliding Window Protocol**

//Germaine and John's code, edited from the file provided.

/\*===============================================================\*

\* 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:

\*==========================================================================\*/

boolean noNak **=** **true;** //keep track if an NAK has to be sent

private Timer normalTimer**[]** **=** **new** Timer**[**NR\_BUFS**];** //each packet sent has a retransmission timer

private Timer ackTimer**;** //to indicate that a separate ack frame needs to be sent

private static final int retransmissionTimeout **=** 1000**;** //retransmit after 1s

private static final int separateAckTimeout **=** 500**;** //send a separate ACK frame after 0.5s

//increment the sequence number

private int inc**(**int seq**)**

**{**

**return** **(**seq **+** 1**)** **%** **(**MAX\_SEQ **+** 1**);** //seq cannot exceed MAX\_SEQ

**}**

//transmit a frame

private void sendFrame**(**int frameKind**,** int frameNum**,** int frameExpected**,** Packet buffer**[])**

**{**

PFrame frame **=** **new** PFrame**();** //frame structure

frame**.**kind **=** frameKind**;** //set the frame kind

//if data frame

**if** **(**frameKind **==** PFrame**.**DATA**)**

frame**.**info **=** buffer**[**frameNum **%** NR\_BUFS**];** //get the data from the buffer and put it in the frame

frame**.**seq **=** frameNum**;** //set the frame sequence number

frame**.**ack **=** **(**frameExpected **+** MAX\_SEQ**)** **%** **(**MAX\_SEQ **+** 1**);** //piggybacked ack of a received frame

//if frame kind is NAK, set noNAK to false

**if** **(**frameKind **==** PFrame**.**NAK**)**

noNak **=** **false;**

to\_physical\_layer**(**frame**);** //send the frame to the physical layer to be transmitted

//if data frame, start the retransmission timer for that frame

**if** **(**frameKind **==** PFrame**.**DATA**)**

startTimer**(**frameNum**);**

stopAckTimer**();** //ack will be piggybacked, hence stop ack timer to prevent separate ack frame

**}**

//check if b is in the range of a and c

private boolean between**(**int a**,** int b**,** int c**)**

**{**

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

**}**

//start the retransmission timer

private void startTimer**(**int seq**)**

**{**

stopTimer**(**seq**);** //stop the previous timer

normalTimer**[**seq **%** NR\_BUFS**]** **=** **new** Timer**();** //create a new timer for that frame

normalTimer**[**seq **%** NR\_BUFS**].**schedule**(new** NormalTimerTask**(**seq**),** retransmissionTimeout**);** //after timeout, execute the timer task

**}**

//stop the retransmission timer

private void stopTimer**(**int seq**)**

**{**

//if a timer is running for that frame

**if** **(**normalTimer**[**seq **%** NR\_BUFS**]** **!=** **null)**

normalTimer**[**seq **%** NR\_BUFS**].**cancel**();** //cancel the timer

**}**

//start the separate ack frame timer

private void startAckTimer**(** **)**

**{**

stopAckTimer**();** //stop the currently running timer

ackTimer **=** **new** Timer**();** //create a new timer object

ackTimer**.**schedule**(new** ACKTimerTask**(),** separateAckTimeout**);** //after timeout, execute the timer task

**}**

//stop separate ack frame timer

private void stopAckTimer**()**

**{**

//if timer is running

**if** **(**ackTimer **!=** **null)**

ackTimer**.**cancel**();** //cancel it

**}**

//TimerTask is an abstract class that needs to be implemented

class NormalTimerTask **extends** TimerTask

**{**

int seq**;**

public NormalTimerTask**(**int seq**)**

**{**

**this.**seq **=** seq**;** //will be used below

**}**

//run method is abstract and needs to be implemented

@Override

public void run**()**

**{**

stopTimer**(this.**seq**);** //stop retransmission timer for this frame after timeout

swe**.**generate\_timeout\_event**(**seq**);**

**}**

**}**

//TimerTask is an abstract class that needs to be implemented

private class ACKTimerTask **extends** TimerTask

**{**

//implement run

@Override

public void run**()**

**{**

stopAckTimer**();** //stop separate ack frame timer for this frame after timeout

swe**.**generate\_acktimeout\_event**();**

**}**

**}**

//protocol 6

public void protocol6**()**

**{**

oldest\_frame **=** MAX\_SEQ **+** 1**;** //oldest frame in the window

int ackExpected **=** 0**;** //minimum edge of the sender's window

int nextFrameToSend **=** 0**;** //maximum edge of the sender's window + 1

int frameReceiveExpected **=** 0**;** //minimum edge of the receiver's window

int receiveOutOfRange **=** NR\_BUFS**;** //maximum edge of the receiver's window + 1

PFrame frame **=** **new** PFrame**();** //new frame structure

private Packet inputBuffer**[]** **=** **new** Packet**[**NR\_BUFS**];** //the receiving buffer

private boolean arrived**[]** **=** **new** boolean**[**NR\_BUFS**];** //to keep track if a frame has arrived

init**();** //initialise out\_buf, see method declared above

//initialise arrive and inputBuffer arrays

**for** **(**int i **=** 0**;** i **<** NR\_BUFS**;** i**++)**

**{**

arrived**[**i**]** **=** **false;**

inputBuffer**[**i**]** **=** **new** Packet**();**

**}**

enable\_network\_layer**(**NR\_BUFS**);** //enable network layer

int numBuffered **=** 0**;** //number of buffer slots filled

**while(true)**

**{**

wait\_for\_event**(**event**);** //wait for event

//depending on event type

**switch(**event**.**type**)**

**{**

//network layer ready

**case** **(**PEvent**.**NETWORK\_LAYER\_READY**):**

numBuffered**++;** //a slot in out\_buf will be filled

from\_network\_layer**(**out\_buf**[**nextFrameToSend **%** NR\_BUFS**]);** //collect data from network layer and place it in out\_buf

sendFrame**(**PFrame**.**DATA**,** nextFrameToSend**,** frameReceiveExpected**,** out\_buf**);** //send the frame

nextFrameToSend **=** inc**(**nextFrameToSend**);** //move on to the next frame to send

**break;**

//a frame has arrived

**case** **(**PEvent**.**FRAME\_ARRIVAL**):**

from\_physical\_layer**(**frame**);** //collect a frame from the physical layer

//if data frame

**if** **(**frame**.**kind **==** PFrame**.**DATA**)**

**{**

//if frame received is not that expected, and noNak is not used

**if** **(**frame**.**seq **!=** frameReceiveExpected **&&** noNak**)**

sendFrame**(**PFrame**.**NAK**,** 0**,** frameReceiveExpected**,** out\_buf**);** //send NAK

**else**

startAckTimer**();** //start timer in case there is no frame to piggyback the ack

//if frame received is within the window, and the receiver's buffer is empty at the assigned slot

**if** **(**between**(**frameReceiveExpected**,** frame**.**seq**,** receiveOutOfRange**)** **&&** arrived**[**frame**.**seq **%** NR\_BUFS**]==false)**

**{**

arrived**[**frame**.**seq **%** NR\_BUFS**]** **=** **true;**//indicate that the receiver's buffer at that slot is now filled

inputBuffer**[**frame**.**seq **%** NR\_BUFS**]** **=** frame**.**info**;** //fill the receiver's buffer with the data

//if a frame expected has arrived, move on to the next frame expected

//loop till we land on an expected frame that has not yet arrived

**while** **(**arrived**[**frameReceiveExpected **%** NR\_BUFS**])**

**{**

//send the data of the frame to the network layer

to\_network\_layer**(**inputBuffer**[**frameReceiveExpected **%** NR\_BUFS**]);**

noNak **=** **true;** //frame is expected, so all is good

//make the slot available to receive the next frame

arrived**[**frameReceiveExpected **%** NR\_BUFS**]** **=** **false;**

frameReceiveExpected **=** inc**(**frameReceiveExpected**);** //move on to the next frame expected

receiveOutOfRange **=** inc**(**receiveOutOfRange**);**//advance the max edge of the receiver's window

startAckTimer**();** //in case there are no frames to piggyback ack for a long time

**}**

**}**

**}**

//if NAK frame

**if** **(**frame**.**kind **==** PFrame**.**NAK **&&** between**(**ackExpected**,** **(**frame**.**ack **+** 1**)** **%** **(**MAX\_SEQ **+** 1**),** nextFrameToSend**))**

sendFrame**(**PFrame**.**DATA**,** **(**frame**.**ack **+** 1**)** **%** **(**MAX\_SEQ **+** 1**),** frameReceiveExpected**,** out\_buf**);** //resend frame

**while** **(**between**(**ackExpected**,** frame**.**ack**,** nextFrameToSend**))**

**{**

numBuffered**--;**

stopTimer**(**ackExpected **%** NR\_BUFS**);**

ackExpected **=** inc**(**ackExpected**);**

enable\_network\_layer**(**1**);**

**}**

**break;**

//damaged frame

**case** **(**PEvent**.**CKSUM\_ERR**):**

**if** **(**noNak**)**

sendFrame**(**PFrame**.**NAK**,** 0**,** frameReceiveExpected**,** out\_buf**);** //send back NAK

**break;**

//if retransmission timeout

**case** **(**PEvent**.**TIMEOUT**):**

sendFrame**(**PFrame**.**DATA**,** oldest\_frame**,** frameReceiveExpected**,** out\_buf**);** //resend data frame

**break;**

//if ack timeout

**case** **(**PEvent**.**ACK\_TIMEOUT**):**

sendFrame**(**PFrame**.**ACK**,** 0**,** frameReceiveExpected**,** out\_buf**);** //send a separate ack frame

**break;**

**default:**

System**.**out**.**println**(**"SWP: undefined event type = " **+** event**.**type**);**

System**.**out**.**flush**();**

**}**

**}**

**}**

**}**//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).

\*/

# **Java Source Files Listing**

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

Bolded files are important to SWP.java, for implementing the system.