|  |
| --- |
|  |
| Implementation of Sliding Window Protocol |
| Net Centric Computing |
|  |
|  |
| **Instructor: Prof. Sun Chengzheng**  **Zhang Danyang**  **Matric. No. XXXXX**  **Solo Project**  **Lab Group: SSP4** |

Table of Contents

[1 Objective 2](#_Toc381107103)

[2 Summary of Completeness 2](#_Toc381107104)

[3 Approaches 3](#_Toc381107105)

[3.1 Full-duplex data communication 3](#_Toc381107106)

[3.2 In-order delivery of packets to the network-layer 3](#_Toc381107107)

[3.3 Synchronization with the network-layer by granting credits 5](#_Toc381107108)

[3.4 Selective repeat retransmission strategy 5](#_Toc381107109)

[3.5 Negative acknowledgement 5](#_Toc381107110)

[3.6 Separate acknowledgment when reverse traffic is light or none 6](#_Toc381107111)

[4 Testing 7](#_Toc381107112)

[4.1 General 7](#_Toc381107113)

[4.2 Sample testing results at level 3 7](#_Toc381107114)

[5 Suggestions 10](#_Toc381107115)

[6 Source Code Listing 10](#_Toc381107116)

[6.1 Index 10](#_Toc381107117)

[6.2 Source code 10](#_Toc381107118)

# 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.

# 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 |

# Approaches

## 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.



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.

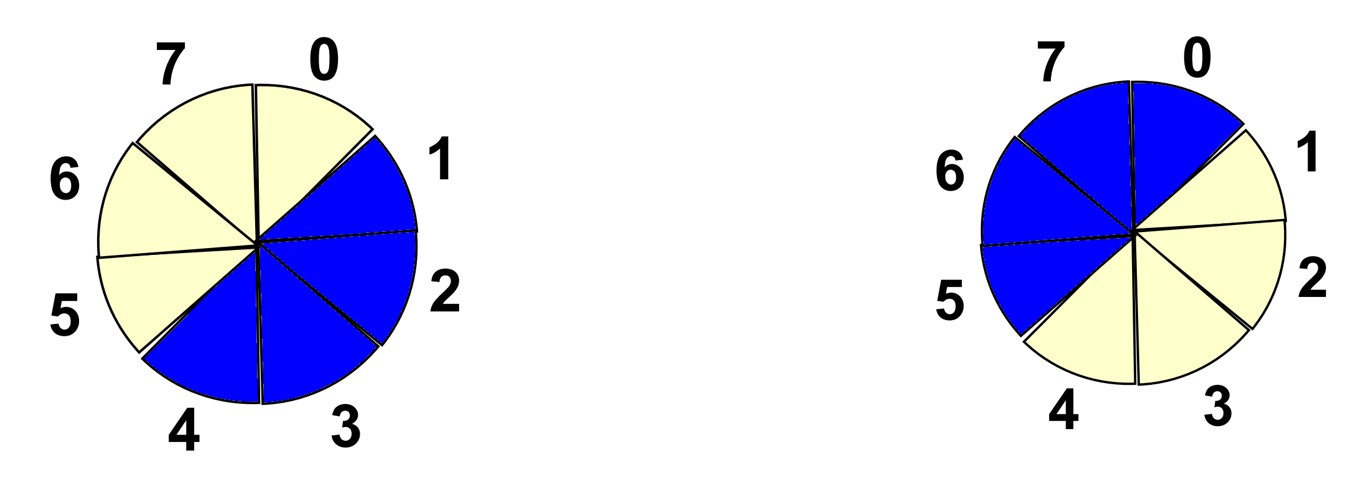


## In-order delivery of packets to the network-layer

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.



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**



## Synchronization with the network-layer by granting credits

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



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



## 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.



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

## 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



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



## 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):



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



# Testing

## 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.

## Sample testing results at level 3

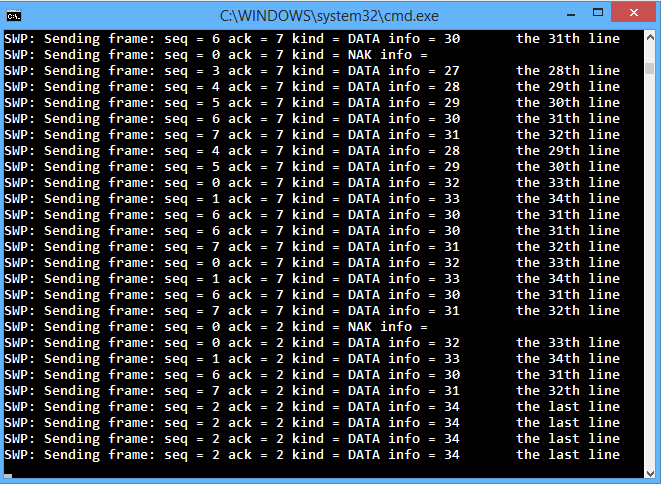


Figure 4‑1 VMach1

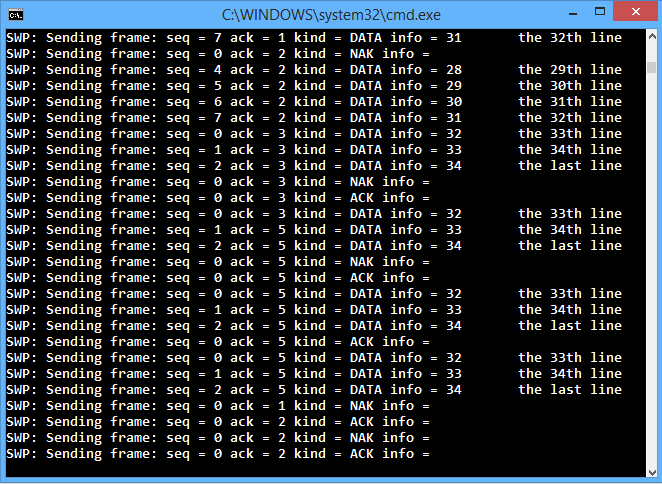


Figure 4‑2 VMach 2

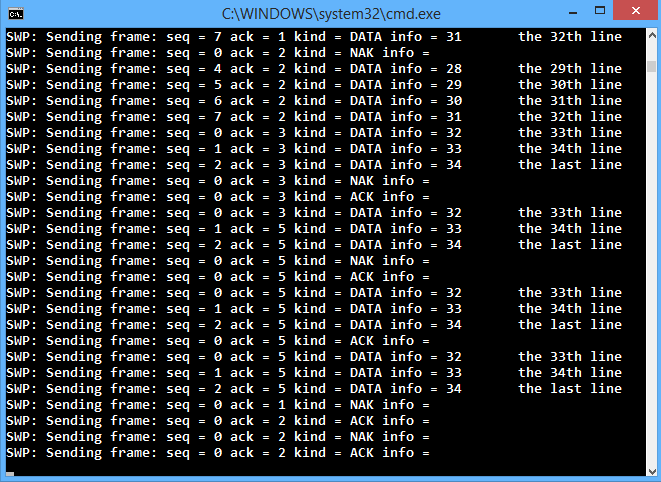


Figure 4‑3 NetSim

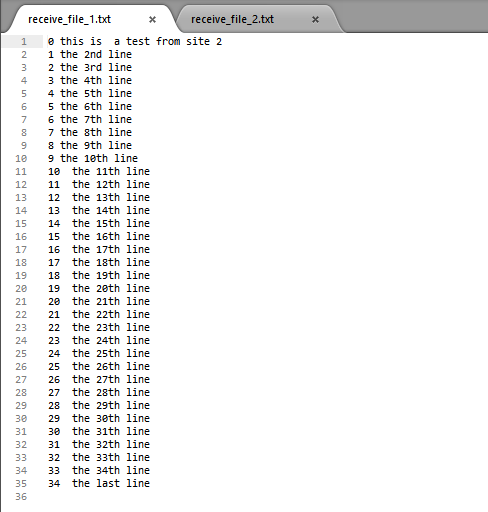


Figure 4‑4 What VMach 1 has received

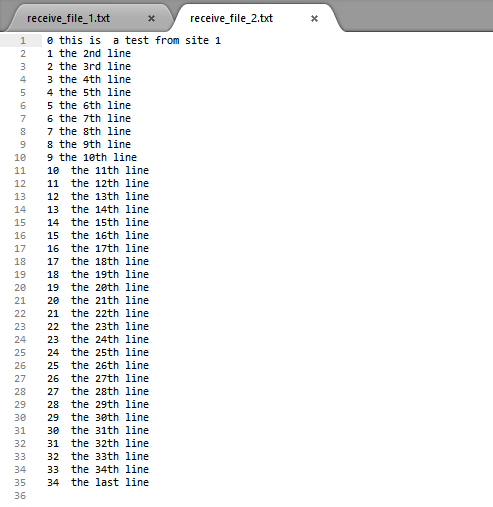
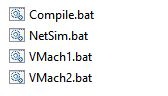


Figure 4‑5 What VMach 2 has received

# 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”:



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

# Source Code Listing

## 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.

## Source code

All codes written by the student are below excluding the part written by instructor. 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;

}

}

// 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();

}

}