

**Assignment 1 Report**

**CSC 302: Net Centric Computing**

**Academic Year 2011/2012**

**Semester 2**

**Student Name: Desmond Poh**

**Matriculation Number: U0922923K**

**Lab Group: TS1**

# Tasks Done

## Full-duplex data communication

Full-duplex data communication indicates that data is transmitting in 2 directions simultaneously.

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

It is very crucial that the network layer receives the frames in the same order as they were sent. This problem arises when the frames are lost or damaged in the process. These frames then have to be resent. Hence the frames are no longer in the correct sequence on the receiving end.

## Selective repeat retransmission strategy

Buffering all the arrived packets that fall within the window, selective repeat can be achieved. In addition, accompanied by the negative acknowledgment, frames that have error are identified, and a request for that particular frame is sent.

## Synchronization with the network-layer by granting credits

Synchronization can be achieved by utilizing the method ***enable\_network\_layer(int nr\_of\_bufs****)*. The amount of credits, required to send packets to the receiver, initially given to the network layer is equivalent to the size of the receiver’s window. During implementation, in the case *PEvent.FRAME\_ARRIVAL*, the integer *nr\_of\_bufs* is incremented in the while loop as shown:

**while (between(ack\_expected, r.ack, next\_frame\_to\_send)) {**

**stop\_timer(ack\_expected);**

**ack\_expected = inc(ack\_expected);**

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

**}**

This is done so when a successful transmission has been acknowledged by the receiver.

## Negative acknowledgement

Declaring a boolean *no\_nak* variable for the expected frame to be received allows negative acknowledgment to be achieved. An error detected in the received frame will call for a negative acknowledgement for that particular frame to be sent. In our implementation, this *no\_nak* variable will be set to false when a negative acknowledgement has to be sent, assuming that the network has high degree of correctness.

## Separate acknowledgement when the reverse traffic is light or more

Separate acknowledgement is achieved by utilizing a timer that sends an acknowledgement packet for the last frame acknowledged/received successfully by the receiver after a delay.

# Source Code

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

\* 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);

}

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:

\* ==========================================================================

\*/

private boolean no\_nak = true; // no nak has been sent yet

private Packet in\_buf[] = new Packet[NR\_BUFS]; // buffers for the inbound stream

private final int delay = 200; // timer delay for a frame ACK

private final int ack\_delay = 100; // timer delay for a ACK piggyback

TimerThread[] timers = new TimerThread[NR\_BUFS];

TimerThread ackTimer = null;

/\*

\* Return true if a<=b<c circularly; false otherwise

\*/

private static boolean between(int a, int b, int c) {

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

}

private void send\_frame(int framekind, int frame\_nr, int frame\_expected, Packet[] buffer) {

PFrame s = new PFrame();

s.kind = framekind;

if (s.kind == PFrame.DATA) {

s.info = buffer[frame\_nr % NR\_BUFS];

}

s.seq = frame\_nr;

s.ack = ((frame\_expected + MAX\_SEQ) % (MAX\_SEQ + 1));

if (s.kind == PFrame.NAK) {

no\_nak = false;

}

to\_physical\_layer(s);

if (s.kind == PFrame.DATA) {

start\_timer(frame\_nr);

}

stop\_ack\_timer();

}

/\*

\* function to increase circularly

\*/

private static int inc(int num) {

num = ((num + 1) % (MAX\_SEQ + 1));

return num;

}

public void protocol6() {

init();

int ack\_expected = 0; //lower edge of sender's window

int next\_frame\_to\_send = 0; //upper edge of sender's window+1

int frame\_expected = 0; //lower edge of receiver's window

int too\_far = NR\_BUFS; //upper edge of receiver's window+1

int i; //index into buffer pool

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

boolean arrived[] = new boolean[NR\_BUFS]; // inbound bit map

enable\_network\_layer(NR\_BUFS);

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

arrived[i] = false;

}

while (true) {

wait\_for\_event(event);

switch (event.type) {

case (PEvent.NETWORK\_LAYER\_READY):

from\_network\_layer(out\_buf[next\_frame\_to\_send % NR\_BUFS]);

send\_frame(0, next\_frame\_to\_send, frame\_expected, out\_buf);

next\_frame\_to\_send = inc(next\_frame\_to\_send);

break;

case (PEvent.FRAME\_ARRIVAL):

from\_physical\_layer(r);

if (r.kind == PFrame.DATA) {

if ((r.seq != frame\_expected) && no\_nak) {

send\_frame(PFrame.NAK, 0, frame\_expected, out\_buf);

} else {

start\_ack\_timer();

}

if (between(frame\_expected, r.seq, too\_far) && arrived[r.seq % NR\_BUFS] == false) {

arrived[r.seq % NR\_BUFS] = true;

in\_buf[r.seq % NR\_BUFS] = r.info;

while (arrived[frame\_expected % NR\_BUFS]) {

//pass frames and advance window

to\_network\_layer(in\_buf[frame\_expected % NR\_BUFS]);

no\_nak = true; //where is it??

arrived[frame\_expected % NR\_BUFS] = false;

frame\_expected = inc(frame\_expected);

too\_far = inc(too\_far);

start\_ack\_timer();

}

}

}

if (r.kind == PFrame.NAK && between(ack\_expected, (r.ack + 1) % (MAX\_SEQ + 1), next\_frame\_to\_send)) {

send\_frame(PFrame.DATA, (r.ack + 1) % (MAX\_SEQ + 1), frame\_expected, out\_buf);

}

while (between(ack\_expected, r.ack, next\_frame\_to\_send)) {

stop\_timer(ack\_expected);

ack\_expected = inc(ack\_expected);

enable\_network\_layer(1);

}

break;

case (PEvent.CKSUM\_ERR):

if (no\_nak) {

send\_frame(PFrame.NAK, 0, frame\_expected, out\_buf);

}

break;

case (PEvent.TIMEOUT):

send\_frame(PFrame.DATA, oldest\_frame, frame\_expected, out\_buf);

break;

case (PEvent.ACK\_TIMEOUT):

send\_frame(PFrame.ACK, 0, frame\_expected, out\_buf);

break;

default:

System.out.println("SWP: undefined event type = "

+ event.type);

System.out.flush();

}

}

}

/\*

\* Note: when start\_timer() and stop\_timer() are called, the "seq" parameter

\* must be the sequence number, rather than the index of the timer array, of

\* the frame associated with this timer,

\*/

private void start\_timer(int seq) {

// Create a new timer

TimerThread temp = new TimerThread(seq, delay, "frame\_timer", swe);

temp.start();

// Stop the timer if it is running

if (timers[seq % NR\_BUFS] != null) {

timers[seq % NR\_BUFS].timeout = true;

}

// Save a reference to our newly created timer

timers[seq % NR\_BUFS] = null;

timers[seq % NR\_BUFS] = temp;

}

private void stop\_timer(int seq) {

timers[seq % NR\_BUFS].stopTimerThread();

}

private void start\_ack\_timer() {

// Stop the timer if it is running

if (ackTimer != null && ackTimer.isAlive()) {

ackTimer.stopTimerThread();

}

// Create and start new ACK timer

ackTimer = new TimerThread(0, ack\_delay, "ack\_timer", swe);

ackTimer.start();

}

private void stop\_ack\_timer() {

if (ackTimer != null) {

ackTimer.stopTimerThread();

}

}

}//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).

\*/

public class TimerThread extends Thread {

boolean timeout = false;

int seq\_no;

int delay;

String type;

SWE swe = null;

TimerThread(int seq, int delay, String type, SWE swe) {

this.seq\_no = seq;

this.delay = delay;

this.type = type;

this.swe = swe;

}

public void run() {

int time\_lapsed = 0;

int sleep\_time = 10;

while (timeout == false && time\_lapsed < delay) {

try {

sleep(sleep\_time);

time\_lapsed += sleep\_time;

} catch (InterruptedException ie) {

}

}

if (timeout == false) {

if (type.equals("frame\_timer")) {

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

} else {

swe.generate\_acktimeout\_event();

}

}

timeout = true;

}

public void stopTimerThread() {

timeout = true;

}

}