###### *CSE 473 – Introduction to Computer Networks*

Lab 4 Report

##### *Katie Thomas, Danny Munro Due 11/6/2014 1*

*Part A*. (20 points)Paste a copy of the completed source code for *Rdt* below. Highlight your changes by making them **bold**(you may omit sections of the original program that contain no added code). Remember to also place a complete copy in the repository before you make your final commit.

Rdt.java

/\*\* Main thread for the Rdt object.

\*

\* Inserts payloads received from the application layer into

\* packets, and sends them to the substrate. The packets include

\* the number of packets and chars sent so far (including the

\* current packet). It also takes packets received from

\* the substrate and sends the extracted payloads

\* up to the application layer. To ensure that packets are

\* delivered reliably and in-order, using a sliding

\* window protocol with the go-back-N feature.

\*/

public void run() {

long t0 = System.nanoTime();

long now = 0; // current time (relative to t0)

int numUnacked = 0;

sendAgain = timeout;

while (!quit || numUnacked != 0) {

//If the timer is running, update the protocol's working

//understanding of "now" (i.e. run the timer), and update

//the duration of the timer (sendAgain) if "now" is being updated

//for the first time

if(timerOn){

if (now == 0) {

now = System.nanoTime() - t0;

sendAgain = now + timeout;

}

else {

now = System.nanoTime() - t0;

}

}

Packet p = new Packet();

// if receive buffer has a packet that can be

// delivered, deliver it to sink

if (recvBuf[recvBase] != null) {

p = recvBuf[recvBase];

toSnk.add(p.payload);

recvBuf[recvBase] = null;

recvBase = incr(recvBase);

}

//immediately resend packets before trying to process anything else

else if (dupAcks == 3) {

resend(now);

}

// else if the substrate has an incoming packet

// get the packet from the substrate and process it

else if (sub.incoming()) {

p = sub.receive();

//if it's a data packet

if (p.type == 0) {

//if expected packet, add to recv buffer and update info

if (p.seqNum == expSeqNum) {

recvBuf[recvBase] = p;

expSeqNum = incr(expSeqNum);

lastRcvd = p.seqNum;

}

//send ack back to sub only if rcvd >=0

if(lastRcvd >= 0) {

Packet ack = new Packet();

ack.type = 1;

ack.seqNum = lastRcvd;

sub.send(ack);

}

}

//if ack

else {

//if not timeout

if (now <= sendAgain) {

//if seq num == sendBase-1 (with handled wrap around)

if (p.seqNum == diff(sendBase, (short)1)) {

dupAcks++;

}

//if ack seq num within window

else if (diff(p.seqNum, sendBase) < wSize) {

int numUpdates = (diff(p.seqNum,sendBase)) + 1;

int lastSent = diff(sendSeqNum, (short)1);

if ((int) p.seqNum == lastSent) {

timerOn = false;

}

//process all packets from sendBase to ack received

//in window.

for (int x = 0; x < numUpdates; ++x) {

sendBuf[sendBase] = null;

sendBase = incr(sendBase);

dupAcks = 0;

--numUnacked;

}

}

}

}

}

// else if the resend timer has expired,

// re-send all un-acked packets and reset their timers

else if (now > sendAgain) {

resend(now);

}

// else if there is a message from the source waiting to be sent

// and the send window is not full

// and the substrate can accept a packet

else if ((fromSrc.size() !=0) &&

(diff(sendSeqNum,sendBase) < wSize) && sub.ready()) {

//create a packet containing the message and send it

Packet data = new Packet();

data.type = 0;

data.seqNum = sendSeqNum;

data.payload = fromSrc.poll();

sub.send(data);

//update send buffer and related data

++numUnacked;

sendBuf[data.seqNum] = data;

sendSeqNum = incr(sendSeqNum);

sendAgain = now + timeout; //reset timer

//start timer

timerOn = true;

}

// else nothing to do, so sleep for 1 ms

else {

try {

Thread.sleep(1);

} catch(Exception e) {

System.err.println("Rdt:run: sleep exception " + e);

System.exit(1);

}

}

}

}

/\*\* Resend all packets between the ones numbered with sendSeqNum and

\* sendBase

\* @param now is the current time

\*/

public void resend(long now) {

int numResend = diff(sendSeqNum, sendBase); //=num of packets to resend

dupAcks = 0;

while (!sub.readyX(numResend)) { //do nothing until ready

try {

Thread.sleep(0,1);

} catch(Exception e) {

System.err.println("Rdt:run: sleep exception " + e);

System.exit(1);

}

}

short base = sendBase; //send packets

for (int i = 0; i < numResend; ++i) {

sub.send(sendBuf[base]);

base = incr(base);

}

sendAgain = now + timeout; //reset timer

timerOn = true;

}

Sender.java

/\*\* Return true is ready to accept x packets/ \*/

public boolean readyX(int x) { return sendq.remainingCapacity() > x; }

Substrate.java

/\*\*Test is substrate is ready to receive x more packets.

\* @return true if substrate is ready

\*/

public boolean readyX(int x) { return sndr.readyX(x); }

*Part B.* (10 points) Use the provided *script0* to test your client and server on a single computer. You may do this testing on any Unix (including MacOS) or Linux computer (shell.cec.wustl.edu or onl.wustl.edu). All you need to do is type

./script0

in the folder that contains all your Python code. Paste a copy of the output below.

wSize= 5 timeout= .5 dropProb= .25

\*\*\*\*\*\*\*\*\*\*\*\*\*\* client report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[0] testing 0

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[0]

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[1] testing 1

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[1]

discarding data[2] testing 2

discarding data[3] testing 3

discarding data[4] testing 4

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[5] testing 5

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[6] testing 6

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[1]

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[2] testing 2

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[3] testing 3

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[4] testing 4

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[5] testing 5

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[6] testing 6

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[2]

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[3]

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[4]

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[5]

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[6]

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[7] testing 7

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[8] testing 8

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 data[9] testing 9

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[7]

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[8]

/127.0.0.1:56752 received from /127.0.0.1:11313 ack[9]

/127.0.0.1:56752 received from /127.0.0.1:11313 data[1] testing 1

/127.0.0.1:56752 received from /127.0.0.1:11313 data[2] testing 2

/127.0.0.1:56752 received from /127.0.0.1:11313 data[3] testing 3

/127.0.0.1:56752 received from /127.0.0.1:11313 data[4] testing 4

/127.0.0.1:56752 received from /127.0.0.1:11313 data[4] testing 4

/127.0.0.1:56752 received from /127.0.0.1:11313 data[0] testing 0

/127.0.0.1:56752 received from /127.0.0.1:11313 data[1] testing 1

/127.0.0.1:56752 received from /127.0.0.1:11313 data[2] testing 2

/127.0.0.1:56752 received from /127.0.0.1:11313 data[4] testing 4

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 ack[0]

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 ack[1]

discarding ack[2]

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 ack[2]

/127.0.0.1:56752 received from /127.0.0.1:11313 data[5] testing 5

discarding ack[2]

/127.0.0.1:56752 received from /127.0.0.1:11313 data[3] testing 3

/127.0.0.1:56752 received from /127.0.0.1:11313 data[5] testing 5

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 ack[3]

discarding ack[3]

/127.0.0.1:56752 received from /127.0.0.1:11313 data[5] testing 5

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 ack[3]

/127.0.0.1:56752 received from /127.0.0.1:11313 data[5] testing 5

discarding ack[3]

/127.0.0.1:56752 received from /127.0.0.1:11313 data[4] testing 4

/127.0.0.1:56752 received from /127.0.0.1:11313 data[5] testing 5

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 ack[4]

/127.0.0.1:56752 sending to localhost/127.0.0.1:11313 ack[5]

Sender: sent 15 data packets, 11 acks

discarded 3 data packets, 4 acks

runLength 5.802542

Receiver: received 16 data packets, 11 acks

discarded 0 arrivals

runLength 5.801191

SrcSnk: sent 10, received 6

runLength 0.3

\*\*\*\*\*\*\*\*\*\*\*\*\*\* server report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/127.0.0.1:11313 received from /127.0.0.1:56752 data[0] testing 0

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[0]

/127.0.0.1:11313 received from /127.0.0.1:56752 data[1] testing 1

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[1]

/127.0.0.1:11313 received from /127.0.0.1:56752 data[5] testing 5

discarding ack[1]

/127.0.0.1:11313 received from /127.0.0.1:56752 data[6] testing 6

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[1]

/127.0.0.1:11313 received from /127.0.0.1:56752 data[2] testing 2

/127.0.0.1:11313 received from /127.0.0.1:56752 data[3] testing 3

/127.0.0.1:11313 received from /127.0.0.1:56752 data[4] testing 4

/127.0.0.1:11313 received from /127.0.0.1:56752 data[5] testing 5

/127.0.0.1:11313 received from /127.0.0.1:56752 data[6] testing 6

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[2]

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[3]

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[4]

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[5]

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[6]

/127.0.0.1:11313 received from /127.0.0.1:56752 data[7] testing 7

/127.0.0.1:11313 received from /127.0.0.1:56752 data[8] testing 8

/127.0.0.1:11313 received from /127.0.0.1:56752 data[9] testing 9

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[7]

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[8]

/127.0.0.1:11313 sending to /127.0.0.1:56752 ack[9]

discarding data[0] testing 0

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[1] testing 1

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[2] testing 2

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[3] testing 3

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[4] testing 4

discarding data[0] testing 0

discarding data[1] testing 1

discarding data[2] testing 2

discarding data[3] testing 3

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[4] testing 4

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[0] testing 0

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[1] testing 1

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[2] testing 2

discarding data[3] testing 3

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[4] testing 4

/127.0.0.1:11313 received from /127.0.0.1:56752 ack[0]

/127.0.0.1:11313 received from /127.0.0.1:56752 ack[1]

/127.0.0.1:11313 received from /127.0.0.1:56752 ack[2]

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[5] testing 5

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[3] testing 3

discarding data[4] testing 4

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[5] testing 5

/127.0.0.1:11313 received from /127.0.0.1:56752 ack[3]

discarding data[4] testing 4

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[5] testing 5

/127.0.0.1:11313 received from /127.0.0.1:56752 ack[3]

discarding data[4] testing 4

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[5] testing 5

discarding data[4] testing 4

discarding data[5] testing 5

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[4] testing 4

/127.0.0.1:11313 sending to /127.0.0.1:56752 data[5] testing 5

/127.0.0.1:11313 received from /127.0.0.1:56752 ack[4]

/127.0.0.1:11313 received from /127.0.0.1:56752 ack[5]

Sender: sent 27 data packets, 12 acks

discarded 11 data packets, 1 acks

runLength 5.703068

Receiver: received 12 data packets, 7 acks

discarded 0 arrivals

runLength 5.802747

SrcSnk: sent 6, received 10

runLength 0.3

1. Based on the report output, how many of the packets sent by the client were retransmissions? How many of these were caused by the discarding of the data packets and how many were caused by the discarding of acknowledgments?

*Client sent 15 data packets, and since it only sent 10 unique packets, the remaining 5 were retransmissions of data packets. The client also sent 11 acks, and since it only sent 6 unique acks, the remaining 5 were retransmissions of acks – for a total of 10 retransmissions. The client discarded 2 data packets and the server discarded 1 ack. Where those discarded packets fell in the send window dictate how many packets must be resent.*

1. What was the specified run length for this test? How does that compare to the actual time it took to transfer all the packets?

*The specified runlength was .3 seconds, but the actual runlength for the server was about 5.8 seconds.*

*Part C*. (5 points) Use the provided *script1* to test your client and server on two computers in ONL, using the provided ONL configuration. To run the script, just type

./script1

in the folder ~/473/lab4 on your onl account. Your Java classes should be in this folder, along with the script. Paste a copy of the output below.

wSize= 5 timeout= .5 dropProb= .25

\*\*\*\*\*\*\*\*\*\*\*\*\*\* client report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[0] testing 0

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[1] testing 1

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[2] testing 2

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[3] testing 3

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[1]

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[2]

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[4] testing 4

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[3]

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[5] testing 5

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[4]

discarding data[6] testing 6

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[5]

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[7] testing 7

discarding data[8] testing 8

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[5]

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[9] testing 9

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[5]

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[6] testing 6

discarding data[7] testing 7

discarding data[8] testing 8

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[9] testing 9

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[6]

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[6]

/192.168.4.2:55572 received from /192.168.7.1:11313 data[0] testing 0

discarding ack[0]

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[7] testing 7

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[8] testing 8

/192.168.4.2:55572 received from /192.168.7.1:11313 data[1] testing 1

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 data[9] testing 9

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 ack[1]

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[7]

/192.168.4.2:55572 received from /192.168.7.1:11313 ack[9]

/192.168.4.2:55572 received from /192.168.7.1:11313 data[3] testing 3

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 ack[1]

/192.168.4.2:55572 received from /192.168.7.1:11313 data[4] testing 4

discarding ack[1]

/192.168.4.2:55572 received from /192.168.7.1:11313 data[5] testing 5

discarding ack[1]

/192.168.4.2:55572 received from /192.168.7.1:11313 data[2] testing 2

/192.168.4.2:55572 received from /192.168.7.1:11313 data[3] testing 3

/192.168.4.2:55572 received from /192.168.7.1:11313 data[4] testing 4

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 ack[2]

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 ack[3]

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 ack[4]

/192.168.4.2:55572 received from /192.168.7.1:11313 data[5] testing 5

/192.168.4.2:55572 sending to h7x1/192.168.7.1:11313 ack[5]

Sender: sent 17 data packets, 9 acks

discarded 4 data packets, 3 acks

runLength 1.983793903

Receiver: received 9 data packets, 11 acks

discarded 0 arrivals

runLength 1.880028822

SrcSnk: sent 10, received 6

runLength 0.3

\*\*\*\*\*\*\*\*\*\*\*\*\*\* server report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/192.168.7.1:11313 received from /192.168.4.2:55572 data[0] testing 0

/192.168.7.1:11313 received from /192.168.4.2:55572 data[1] testing 1

discarding ack[0]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[2] testing 2

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[1]

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[2]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[3] testing 3

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[3]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[4] testing 4

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[4]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[5] testing 5

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[5]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[7] testing 7

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[5]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[9] testing 9

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[5]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[6] testing 6

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[6]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[9] testing 9

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[6]

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[0] testing 0

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[1] testing 1

discarding data[2] testing 2

/192.168.7.1:11313 received from /192.168.4.2:55572 data[7] testing 7

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[7]

/192.168.7.1:11313 received from /192.168.4.2:55572 data[8] testing 8

/192.168.7.1:11313 received from /192.168.4.2:55572 data[9] testing 9

discarding ack[8]

/192.168.7.1:11313 sending to /192.168.4.2:55572 ack[9]

/192.168.7.1:11313 received from /192.168.4.2:55572 ack[1]

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[3] testing 3

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[4] testing 4

/192.168.7.1:11313 received from /192.168.4.2:55572 ack[1]

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[5] testing 5

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[2] testing 2

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[3] testing 3

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[4] testing 4

/192.168.7.1:11313 sending to /192.168.4.2:55572 data[5] testing 5

/192.168.7.1:11313 received from /192.168.4.2:55572 ack[2]

/192.168.7.1:11313 received from /192.168.4.2:55572 ack[3]

/192.168.7.1:11313 received from /192.168.4.2:55572 ack[4]

/192.168.7.1:11313 received from /192.168.4.2:55572 ack[5]

Sender: sent 10 data packets, 13 acks

discarded 1 data packets, 2 acks

runLength 1.862903314

Receiver: received 13 data packets, 6 acks

discarded 0 arrivals

runLength 2.013660028

SrcSnk: sent 6, received 10

runLength 0.3

*Part D*. (15 points) Use the provided *script2* to run this next test. Paste a copy of the output below. Also, paste a screen capture showing the two monitoring windows labeled “from/to hosts” and “inter-router traffic”. Make sure your screenshot shows the curves for the entire duration of the script run, and that the text labels are large enough to read on a printed copy. You will find it easier to do the screen capture if you first “stop” the chart by using the “Stop” menu item in the Options menu (to restart the chart, select this item again).

wSize= 5 timeout= .5 dropProb= .2

\*\*\*\*\*\*\*\*\*\*\*\*\*\* client report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sender: sent 465 data packets, 170 acks

discarded 82 data packets, 35 acks

runLength 20.611732041

Receiver: received 174 data packets, 298 acks

discarded 0 arrivals

runLength 20.208034618

SrcSnk: sent 200, received 100

runLength 20.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\* server report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sender: sent 216 data packets, 380 acks

discarded 42 data packets, 82 acks

runLength 20.406633533

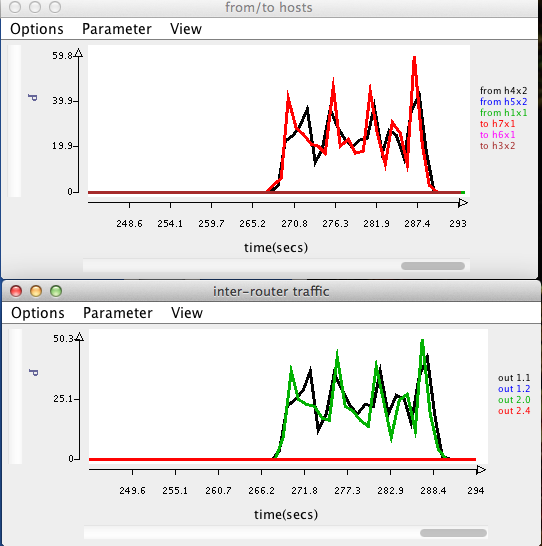
Receiver: received 383 data packets, 135 acks

discarded 0 arrivals

runLength 20.558908207

SrcSnk: sent 100, received 200

runLength 20.0

Answer the following questions, based on your the results of this test.

1. What was the specified run length for this script? What was the specified packet sending rate for the client and server? (You will need to examine the script in order to answer this question.)

*Specified run length was 20.0 seconds, and specified packet sending rates were 1 packet/.2 seconds (5 packets/sec) for the server and 1 packet/.1 seconds (10 packets/sec) for the client.*

1. How long did it take to deliver all the packets? What was the effective packet delivery rate from the client to the module at the server?

*\*\*\*The time taken to deliver all the packets equals max(time for client to deliver all packets to server, time for server to deliver all packets to client) =* 20.611732041. The effective packet data delivery was max

*It took ~20.5sec to send all of the packets, just barely over the specified 20s. The effective delivery rate is then 200/20.5 = 9.75pack/sec (roughly 10p/s).*

1. How many packets did the *Rdt* module at the client send (including retransmissions but excluding acks)? How many of these were retransmissions? What was the average sending rate for the client, including both retransmissions and acks?

*Client sent 465 total data packets (265 of which were retransmissions). Average sending rate for client was (465+170) packets/20.5 seconds = 30.9 packets/sec.*

*Part E.* (20 points) In this part you will be using the provided *script3* to answer some questions about the performance of your protocol when run from a client at *h4x2* to a server at *h7x1* (in this script, the server does not send any data packets). The script takes several arguments, whose values you will need to specify, when running the experiments needed to answer the questions below.

1. Determine the round-trip delay between *h4x2* and *h7x1* using *ping* (make sure you are using the correct addresses, so that your packets go through your experimental network, and not the ONL control network). What value did you get? Based on this, if your protocol is configured with a window size of 1 packet, what is the maximum rate at which it can send packets? What is the smallest window size that would allow it to send 1000 packets per second? Note that later answers in this section depend on your ability to answer this part correctly, so make sure you understand this.

*RTT (when pinging h7x1 from h4x2) is consistently 50.1ms. Given a window size of 1 packet, the RDT no longer uses pipelining, so it can send 1 packet/50.1 ms (about 19.96 packets/sec). The protocol would need to have a window of at least 51 packets to send 1000 packets/sec.*

1. In this part, you will run *script3* with a timeout value of 0.6 seconds, a drop probability of 0 and a delta value of .004. What sending rate does this correspond to? Choose the smallest window size that is consistent with this sending rate and paste a copy of the output of your run below. Were the packets actually delivered to the destination at the specified sending rate?

*The corresponding sending rate is 1 packet/.004 sec = 250 packets/sec. Multiplying this by an RTT of 50.1 ms (i.e. the time which we have to continually send packets before an ack is received for the first one) gets a minimum window size of 13 packets (rounded up from 12.53). In practice, the packets were delivered at a rate of 2500 packets/10.005062052 seconds, which is very close to the theoretical 250 packets/sec.*

1. In this part, you are to determine the maximum rate at which you can send traffic between the two routers. Determine the maximum sending rate by decreasing the delta value, while increasing the window size to match (keep the timeout value at 0.6 and the discard probability at 0). Observe the packet rate on the inter-router link using the monitoring window and stop decreasing delta when you no longer get any increase in the peak transfer rate observed. At this point, your sending rate is being constrained by the link’s ability to forward packets. Paste a copy of the script output from the run that achieves this maximum packet rate. Also, paste a screen shot showing all three of the monitoring windows from this run.

wSize= 334 timeout= .6 dropProb= 0 delta= .00015

\*\*\*\*\*\*\*\*\*\*\*\*\*\* client report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sender: sent 55293 data packets, 0 acks

discarded 0 data packets, 0 acks

runLength 10.182698588

Receiver: received 0 data packets, 55293 acks

discarded 0 arrivals

runLength 10.137125748

SrcSnk: sent 55293, received 0

runLength 10.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\* server report \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Sender: sent 0 data packets, 55293 acks

discarded 0 data packets, 0 acks

runLength 10.12143191

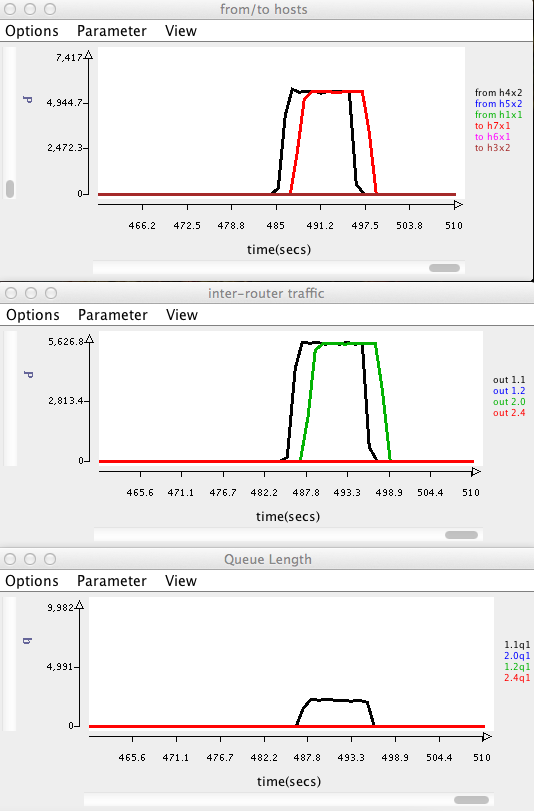
Receiver: received 55293 data packets, 0 acks

discarded 0 arrivals

runLength 10.216315696

SrcSnk: sent 0, received 55293

runLength 10.0



What was the maximum packet sending rate you were able to achieve? What was the specified sending rate? Did you observe any queueing at the inter-router link?

*The maximum packet sending rate is about 5700pack/sec (as seen by the peak from h4x2 at the beginning of the transfer). The specified sending rate was 1/0.00015 = 6666.67 packets/sec. Yes, we observed a queue that sustained at ~2200 bytes for approximately the length of the run (~10 sec).*

1. Run *script3* with a windows size of 300, timeout of 0.6, discard probability of 0 and a delta of .00015. Now run it again with window sizes of 400, 450 and 500. For each of these runs note the maximum length of the queues at the inter-router link. What are these maximum queue lengths?

*The runs of window size 300, 400, 450, and 500 had max queue lengths of roughly 950, 5000, 7500, and 10000 bytes, respectively.*

How does the throughput compare for these five cases?

*Each of the runs had similar throughputs of about 5500 packets/sec, assuming you are measuring correctly received packets. However, with a window size of 500, the sender sends about 20 times more packets (3.5 million) then the receiver receives (160000).*

Explain the observed results as best you can. Hint: you may want to examine the queue table at port 1 of router 1.

*The queue table shows that the inter-router queue holds 10000 bytes. Somewhere between a window size of 450 and 500, the window size is large enough that*