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

Lab 4 Report

##### *Your name: Bingkun Guo Due 11/5/2013 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.

/\*\*

\* Name: Bingkun Guo

\* Date: Nov/11/2013

\*

\* Reliable Data Transport class.

\*

\* This class implements a reliable data transport service.

\* It uses a sliding window protocol, on a packet basis,

\* with selective repeat.

\*

\* An application layer thread provides new packet payloads to be

\* sent using the provided send() method, and retrieves newly arrived

\* payloads with the receive() method. Each application layer payload

\* is sent as a separate UDP packet, along with a sequence number and

\* a type flag that identifies a packet as a data packet or an

\* acknowledgment. The sequence numbers are 15 bits.

\*/

import java.io.\*;

import java.net.\*;

import java.util.\*;

import java.util.concurrent.\*;

public class Rdt implements Runnable {

private int wSize; // protocol window size

private long timeout; // retransmission timeout in ns

private Substrate sub; // Substrate object for packet IO

// queues for communicating with source/sink

private ArrayBlockingQueue<String> fromSrc;

private ArrayBlockingQueue<String> toSnk;

private Packet[] sendBuffer;

private Packet[] rcvBuffer;

private long[] resendTimes;

private List<Integer> resendList;

private short sendBase, nextSeqNum, rcvBase;

private long oldestTime;

private Thread myThread; // local thread for this object

private boolean quit; // used to signal quitting time

/\*\* Initialize a new Rdt object.

\* @param wSize is the window size used by protocol; the sequence #

\* space is twice the window size

\* @param timeout is the time to wait before retransmitting

\* @param sub is a reference to the Substrate object that this object

\* uses to handle the socket IO

\*/

Rdt(int wSize, double timeout, Substrate sub) {

this.wSize = Math.min(wSize,(1 << 14) - 1);

this.timeout = ((long) (timeout \* 1000000000)); // sec to ns

this.sub = sub;

// create queues for application layer interface

fromSrc = new ArrayBlockingQueue<String>(1000,true);

toSnk = new ArrayBlockingQueue<String>(1000,true);

quit = false;

sendBuffer = new Packet[2\*wSize];

resendTimes = new long[2\*wSize];

resendList = new LinkedList<Integer>();

rcvBuffer = new Packet[2\*wSize];

sendBase = nextSeqNum = rcvBase = 0;

}

/\*\* Start the Rdt running. \*/

public void start() throws Exception {

myThread = new Thread(this); myThread.start();

}

/\*\* Stop the Rdt. \*/

public void stop() throws Exception { quit = true; myThread.join(); }

/\*\* Increment sequence number, handling wrap-around.

\* @param x is a sequence number

\* @return next sequence number after x

\*/

private short incr(short x) {

x++; return (x < 2\*wSize ? x : 0);

}

/\*\* Compute the difference between two sequence numbers,

\* accounting for "wrap-around"

\* @param x is a sequence number

\* @param y is another sequence number

\* @return difference, assuming x is "clockwise" from y

\*/

private int diff(short x, short y) {

return (x >= y ? x-y : (x + 2\*wSize) - y);

}

**/\*\* 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 selective repeat feature.**

**\*/**

**public void run() {**

**long t0 = System.nanoTime();**

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

**while (!quit || resendList.size() > 0/\* we still have un-acked packets \*/ ) {**

**now = System.nanoTime() - t0;**

**if (rcvBuffer[rcvBase] != null) {**

**// if receive buffer has a packet that can be**

**// delivered, deliver it to sink**

**Packet p = rcvBuffer[rcvBase];**

**try {**

**toSnk.put(p.payload);**

**} catch (Exception e) {**

**System.err.println("toSnk: put exception" + e);**

**System.exit(1);**

**}**

**rcvBuffer[rcvBase] = null;**

**rcvBase = incr(rcvBase);**

**} else if (sub.incoming()) {**

**// else if the substrate has an incoming packet**

**// get the packet from the substrate and process it**

**Packet p = sub.receive();**

**if (p.type == 0) {**

**// if it's a data packet, ack it and add it**

**// to receive buffer as appropriate**

**if (diff(p.seqNum, rcvBase) < wSize)**

**if (rcvBuffer[p.seqNum] == null)**

**rcvBuffer[p.seqNum] = p;**

**Packet ackPkt = new Packet();**

**ackPkt.type = 1;**

**ackPkt.seqNum = p.seqNum;**

**sub.send(ackPkt);**

**} else if (diff(p.seqNum, sendBase) < diff(nextSeqNum, sendBase)) {**

**// if it's an ack, update the send buffer and**

**// related data as appropriate**

**// reset the timer if necessary**

**if (sendBuffer[p.seqNum] != null) {**

**sendBuffer[p.seqNum] = null;**

**resendList.remove(new Integer(p.seqNum));**

**while (sendBase != nextSeqNum**

**&& sendBuffer[sendBase] == null) {**

**sendBase = incr(sendBase);**

**}**

**if (sendBase != nextSeqNum) {**

**oldestTime = resendTimes[resendList.get(0)];**

**}**

**}**

**}**

**} else if (now > oldestTime && sub.ready() && sendBase != nextSeqNum) {**

**// if the resend timer has expired, re-send the**

**// oldest un-acked packet and reset timer**

**int seqNum = resendList.remove(0).intValue();**

**Packet resendPkt = sendBuffer[seqNum];**

**sub.send(resendPkt);**

**resendTimes[seqNum] = now + timeout;**

**resendList.add(new Integer(seqNum));**

**int oldestSeqNum = resendList.get(0).intValue();**

**oldestTime = resendTimes[oldestSeqNum];**

**} else if (fromSrc.size() > 0 && sub.ready()**

**&& (diff(nextSeqNum, sendBase) < wSize)) {**

**// 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**

**// create a packet containing the message,**

**// and send it, after updating the send buffer**

**// and related data**

**Packet p = new Packet();**

**p.type = 0;**

**p.payload = fromSrc.poll();**

**p.seqNum = nextSeqNum;**

**sendBuffer[nextSeqNum] = p;**

**resendTimes[nextSeqNum] = now + timeout;**

**resendList.add(new Integer(nextSeqNum));**

**if (sendBase == nextSeqNum) {**

**oldestTime = resendTimes[sendBase];**

**}**

**nextSeqNum = incr(nextSeqNum);**

**sub.send(p);**

**} else {**

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

**try {**

**Thread.sleep(1);**

**} catch(Exception e) {**

**System.exit(1);**

**}**

**}**

**}**

**}**

/\*\* Send a message to peer.

\* @param message is a string to be sent to the peer

\*/

public void send(String message) {

try {

fromSrc.put(message);

} catch(Exception e) {

System.err.println("Rdt:send: put exception" + e);

System.exit(1);

}

}

/\*\* Test if Rdt is ready to send a message.

\* @return true if Rdt is ready

\*/

public boolean ready() { return fromSrc.remainingCapacity() > 0; }

/\*\* Get an incoming message.

\* @return next message

\*/

public String receive() {

String s = null;

try {

s = toSnk.take();

} catch(Exception e) {

System.err.println("Rdt:send: take exception" + e);

System.exit(1);

}

return s;

}

/\*\* Test for the presence of an incoming message.

\* @return true if there is an incoming message

\*/

public boolean incoming() { return toSnk.size() > 0; }

}

*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.

your output here

wSize= 5 timeout= .5 dropProb= .25

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

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[0] testing 0

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[1] testing 1

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[2] testing 2

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[3] testing 3

/127.0.0.1:51881 received from /127.0.0.1:30000 data[0] testing 0

/127.0.0.1:51881 received from /127.0.0.1:30000 data[1] testing 1

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[2]

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[3]

/127.0.0.1:51881 received from /127.0.0.1:30000 data[2] testing 2

discarding ack[0]

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 ack[1]

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 ack[2]

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[4] testing 4

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[4]

/127.0.0.1:51881 received from /127.0.0.1:30000 data[3] testing 3

discarding ack[3]

/127.0.0.1:51881 received from /127.0.0.1:30000 data[4] testing 4

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 ack[4]

/127.0.0.1:51881 received from /127.0.0.1:30000 data[0] testing 0

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 ack[0]

discarding data[0] testing 0

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[1] testing 1

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[1]

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[0] testing 0

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[0] testing 0

/127.0.0.1:51881 received from /127.0.0.1:30000 data[5] testing 5

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 ack[5]

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[0]

discarding data[5] testing 5

discarding data[6] testing 6

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[7] testing 7

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[8] testing 8

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[9] testing 9

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[7]

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[8]

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[9]

/127.0.0.1:51881 received from /127.0.0.1:30000 data[3] testing 3

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 ack[3]

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[5] testing 5

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[6] testing 6

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[5]

/127.0.0.1:51881 sending to localhost/127.0.0.1:30000 data[6] testing 6

/127.0.0.1:51881 received from /127.0.0.1:30000 ack[6]

Sender: sent 17 data packets, 8 acks

discarded 3 data packets, 2 acks

runLength 2.503978679

Receiver: received 8 data packets, 10 acks

discarded 0 arrivals

runLength 2.406154446

SrcSnk: sent 10, received 6

runLength 0.3

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

/127.0.0.1:30000 received from /127.0.0.1:51881 data[0] testing 0

/127.0.0.1:30000 received from /127.0.0.1:51881 data[1] testing 1

/127.0.0.1:30000 received from /127.0.0.1:51881 data[2] testing 2

/127.0.0.1:30000 received from /127.0.0.1:51881 data[3] testing 3

/127.0.0.1:30000 sending to /127.0.0.1:51881 data[0] testing 0

discarding ack[0]

discarding ack[1]

/127.0.0.1:30000 sending to /127.0.0.1:51881 data[1] testing 1

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[2]

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[3]

/127.0.0.1:30000 sending to /127.0.0.1:51881 data[2] testing 2

/127.0.0.1:30000 received from /127.0.0.1:51881 ack[1]

/127.0.0.1:30000 received from /127.0.0.1:51881 ack[2]

/127.0.0.1:30000 received from /127.0.0.1:51881 data[4] testing 4

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[4]

/127.0.0.1:30000 sending to /127.0.0.1:51881 data[3] testing 3

/127.0.0.1:30000 sending to /127.0.0.1:51881 data[4] testing 4

/127.0.0.1:30000 received from /127.0.0.1:51881 ack[4]

/127.0.0.1:30000 sending to /127.0.0.1:51881 data[0] testing 0

/127.0.0.1:30000 received from /127.0.0.1:51881 ack[0]

discarding data[5] testing 5

/127.0.0.1:30000 received from /127.0.0.1:51881 data[1] testing 1

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[1]

discarding data[3] testing 3

/127.0.0.1:30000 received from /127.0.0.1:51881 data[0] testing 0

discarding data[5] testing 5

discarding ack[0]

discarding data[3] testing 3

/127.0.0.1:30000 sending to /127.0.0.1:51881 data[5] testing 5

/127.0.0.1:30000 received from /127.0.0.1:51881 data[0] testing 0

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[0]

/127.0.0.1:30000 received from /127.0.0.1:51881 ack[5]

/127.0.0.1:30000 received from /127.0.0.1:51881 data[7] testing 7

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[7]

/127.0.0.1:30000 received from /127.0.0.1:51881 data[8] testing 8

/127.0.0.1:30000 received from /127.0.0.1:51881 data[9] testing 9

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[8]

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[9]

/127.0.0.1:30000 sending to /127.0.0.1:51881 data[3] testing 3

/127.0.0.1:30000 received from /127.0.0.1:51881 ack[3]

/127.0.0.1:30000 received from /127.0.0.1:51881 data[5] testing 5

/127.0.0.1:30000 received from /127.0.0.1:51881 data[6] testing 6

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[5]

discarding ack[6]

/127.0.0.1:30000 received from /127.0.0.1:51881 data[6] testing 6

/127.0.0.1:30000 sending to /127.0.0.1:51881 ack[6]

Sender: sent 12 data packets, 14 acks

discarded 4 data packets, 4 acks

runLength 2.308341825

Receiver: received 14 data packets, 6 acks

discarded 0 arrivals

runLength 2.406546515

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?

*There are 7 packets sent by the client were retransmissions. Because the sender sent 17 packets while the SrcSnk only sent 10 packets.*

*The client discarded 3 data packets, so 3 of these were caused by the discarding of the data packets.*

*The server reported 4 discarded acknowledgements, so 4 of these were caused by the discarding of acknowledgements.*

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 specific run length for this test was 0.3 seconds, the actual time it took to transfer all the packets was about 2.4 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.

your output here

wSize= 5 timeout= .5 dropProb= .25

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

discarding data[0] testing 0

discarding data[1] testing 1

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[2] testing 2

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[3] testing 3

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[4] testing 4

discarding data[0] testing 0

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[1] testing 1

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[2] testing 2

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[2]

discarding data[3] testing 3

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[2]

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[4] testing 4

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[4]

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[0] testing 0

discarding data[1] testing 1

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[0]

discarding data[5] testing 5

/192.168.4.2:53649 received from /192.168.7.1:50000 data[2] testing 2

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 ack[2]

discarding data[3] testing 3

/192.168.4.2:53649 received from /192.168.7.1:50000 data[4] testing 4

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 ack[4]

/192.168.4.2:53649 received from /192.168.7.1:50000 data[1] testing 1

discarding ack[1]

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[1] testing 1

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[5] testing 5

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[3] testing 3

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[5]

/192.168.4.2:53649 received from /192.168.7.1:50000 data[3] testing 3

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 ack[3]

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[3]

/192.168.4.2:53649 received from /192.168.7.1:50000 data[0] testing 0

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 ack[0]

/192.168.4.2:53649 received from /192.168.7.1:50000 data[1] testing 1

discarding ack[1]

/192.168.4.2:53649 received from /192.168.7.1:50000 data[5] testing 5

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 ack[5]

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[1] testing 1

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[1]

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[6] testing 6

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[7] testing 7

discarding data[8] testing 8

discarding data[9] testing 9

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[6]

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[7]

/192.168.4.2:53649 received from /192.168.7.1:50000 data[1] testing 1

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 ack[1]

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[8] testing 8

/192.168.4.2:53649 sending to h7x1/192.168.7.1:50000 data[9] testing 9

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[8]

/192.168.4.2:53649 received from /192.168.7.1:50000 ack[9]

Sender: sent 24 data packets, 8 acks

discarded 9 data packets, 2 acks

runLength 2.588804614

Receiver: received 8 data packets, 11 acks

discarded 0 arrivals

runLength 2.150030577

SrcSnk: sent 10, received 6

runLength 0.3

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

/192.168.7.1:50000 received from /192.168.4.2:53649 data[2] testing 2

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[2]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[1] testing 1

discarding ack[1]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[2] testing 2

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[2]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[4] testing 4

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[4]

discarding data[0] testing 0

discarding data[1] testing 1

/192.168.7.1:50000 received from /192.168.4.2:53649 data[0] testing 0

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[0]

/192.168.7.1:50000 sending to /192.168.4.2:53649 data[2] testing 2

discarding data[3] testing 3

/192.168.7.1:50000 received from /192.168.4.2:53649 ack[2]

/192.168.7.1:50000 sending to /192.168.4.2:53649 data[4] testing 4

/192.168.7.1:50000 received from /192.168.4.2:53649 ack[4]

discarding data[0] testing 0

/192.168.7.1:50000 sending to /192.168.4.2:53649 data[1] testing 1

/192.168.7.1:50000 received from /192.168.4.2:53649 data[1] testing 1

discarding ack[1]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[5] testing 5

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[5]

/192.168.7.1:50000 sending to /192.168.4.2:53649 data[3] testing 3

/192.168.7.1:50000 received from /192.168.4.2:53649 data[3] testing 3

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[3]

/192.168.7.1:50000 received from /192.168.4.2:53649 ack[3]

/192.168.7.1:50000 sending to /192.168.4.2:53649 data[0] testing 0

/192.168.7.1:50000 sending to /192.168.4.2:53649 data[1] testing 1

/192.168.7.1:50000 received from /192.168.4.2:53649 ack[0]

/192.168.7.1:50000 sending to /192.168.4.2:53649 data[5] testing 5

/192.168.7.1:50000 received from /192.168.4.2:53649 ack[5]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[1] testing 1

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[1]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[6] testing 6

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[6]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[7] testing 7

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[7]

/192.168.7.1:50000 sending to /192.168.4.2:53649 data[1] testing 1

/192.168.7.1:50000 received from /192.168.4.2:53649 ack[1]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[8] testing 8

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[8]

/192.168.7.1:50000 received from /192.168.4.2:53649 data[9] testing 9

/192.168.7.1:50000 sending to /192.168.4.2:53649 ack[9]

Sender: sent 12 data packets, 13 acks

discarded 4 data packets, 2 acks

runLength 2.081257109

Receiver: received 13 data packets, 6 acks

discarded 0 arrivals

runLength 2.17625564

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).

your output here

wSize= 5 timeout= .5 dropProb= .2

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

Sender: sent 287 data packets, 124 acks

discarded 55 data packets, 24 acks

runLength 25.919273208

Receiver: received 124 data packets, 200 acks

discarded 0 arrivals

runLength 25.968917537

SrcSnk: sent 200, received 100

runLength 20.0

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

Sender: sent 163 data packets, 232 acks

discarded 39 data packets, 32 acks

runLength 25.860156012

Receiver: received 232 data packets, 100 acks

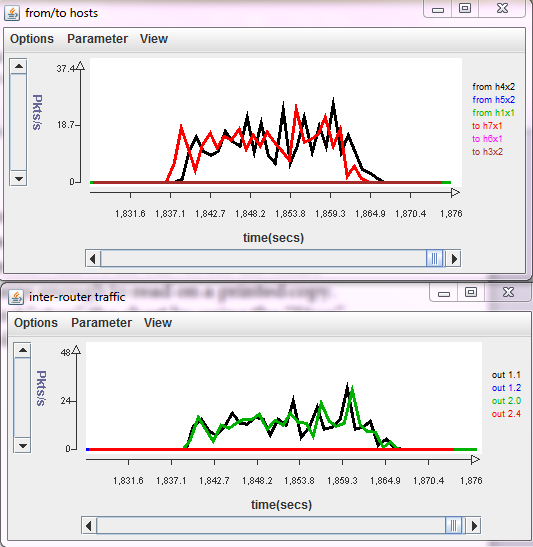
discarded 0 arrivals

runLength 25.955185489

SrcSnk: sent 100, received 200

runLength 20.0

screenshot here

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.)

*The specified run length for script2 is 20 seconds.*

*Since the server delta is 0.2 seconds, thus it sends 5 packets / second.*

*Since the client delta is 0.1 seconds, thus it sends 10 packets / second.*

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?

*Based on the outputs, it took approximately 26 seconds to deliver all the packets.*

*The effective packet delivery rate from the client to the server should be 200/26 = 7 packets / second*

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?

*The Rdt at the client sent 287 data packets, 87 of them were retransmissions.*

*The average sending rate for the client is (287 + 124) / 26 = 15 packets / second*

*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.

*The RTT between h4x2 and h7x1 is about 50ms.*

*If window size is 1 packet, you can send 1 packet per RTT, so the maximum sending rate would be 1 / 0.05 = 20 packets / second.*

*If the sending rate is 1000 packets / second, it means (window size / 0.05s) is at least 1000, which means window size is at least 1000 \* 0.05 = 50 packets*

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?

*Since the delta is 0.004, it means the client will send a packet every 0.004s, which means the sending rate would be 1/0.004 = 250 packets / second.*

*To satisfy this sending rate, window size / 0.05s >= 250, which means window size >= 250 \* 0.05, thus the smallest window size is 13 packets.*

your output here

guobingkun@onlusr:~/473/lab4$ ./script3 13 0.6 0 0.004

wSize= 13 timeout= 0.6 dropProb= 0 delta= 0.004

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

Sender: sent 2500 data packets, 0 acks

discarded 0 data packets, 0 acks

runLength 9.996056355

Receiver: received 0 data packets, 2500 acks

discarded 0 arrivals

runLength 9.954911151

SrcSnk: sent 2500, received 0

runLength 10.0

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

Sender: sent 0 data packets, 2500 acks

discarded 0 data packets, 0 acks

runLength 9.945286427

Receiver: received 2500 data packets, 0 acks

discarded 0 arrivals

runLength 10.041664363

SrcSnk: sent 0, received 2500

runLength 10.0

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.

your output here

guobingkun@onlusr:~/473/lab4$ ./script3 295 0.6 0 0.00017

wSize= 295 timeout= 0.6 dropProb= 0 delta= 0.00017

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

Sender: sent 54625 data packets, 0 acks

discarded 0 data packets, 0 acks

runLength 10.178857002

Receiver: received 0 data packets, 54625 acks

discarded 0 arrivals

runLength 10.230942408

SrcSnk: sent 54625, received 0

runLength 10.0

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

Sender: sent 0 data packets, 54625 acks

discarded 0 data packets, 0 acks

runLength 10.169566621

Receiver: received 54625 data packets, 0 acks

discarded 0 arrivals

runLength 10.264326903

SrcSnk: sent 0, received 54625

runLength 10.0

screenshot here



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 I were able to achieve was approximately 5500 packets / second.*

*The specified sending rate was 5883 packets / second, which is about 7% higher.*

*Yes. There was a small amount of queueing with peak of nearly 500 bytes.*

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?

*Window size of 300: the maximum length of the queues was about 600 bytes.*

*Window size of 400: the maximum length of the queues was about 4900 bytes.*

*Window size of 450: the maximum length of the queues was about 6900 bytes.*

*Window size of 500: the maximum length of the queues was about 9000 bytes.*

How does the throughput compare for these five cases?

*Window sizes of 300,400,450 all have a very close throughput to each other. They all sent approximately 55000 packets in 10 seconds, which derives a throughput of 5500 packets / second.*

*In the case of window size of 500, the client sent about 25000 packets in 11 seconds, which derives a throughput of 2300 packets / second.*

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

*From the queue table at port 1 of router 1, I know the queue capacity at port 1 is 10000 bytes. Thus when the window size is very large, it is very likely to cause the queue overflow. If queue overflow happens, packets will be dropped and timeout events will happen and the hosts have to do retransmission, which slows down the speed for advancing the window. It also possible that since there are so many packets to send, the ArrayBlockingQueue in the sender will be filled up, which means when you want to put new packet in the queue, you have to wait until it is not full, thus the hosts will end up having less time to send packets.*