CSE 473 – Introduction to Computer Networks

Lab 4 Report – 70 Points

Your name here: Richard Wu; Jiaming Qiu *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.

```
/** Main thread for the Rdt object.
 *
 *  Inserts payloads received from the application layer into
 *  packets, and sends them to the substrate. The packets include
 *  a type and a seq num fields. 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)
   /* run the loop if quit is false or we still have un-acked packets */
   while (!quit || diff(sendSeqNum, sendBase) != 0 ) {
      now = System.nanoTime() - t0;
      // TODO
      // if receive buffer has a packet that can be
      //    delivered, deliver it to sink
      if(recvBuf[recvBase] != null) {
         // Put the payload of the packet at the tail of toSnk
         try {
            toSnk.put(recvBuf[recvBase].payload);
         } catch(Exception e) {
            System.err.println("Rdt:run toSnk exception " + e);
            System.exit(1);
         }
         recvBuf[recvBase] = null;
         recvBase = incr(recvBase);
      }
      // else if the substrate has an incoming packet
      //      get the packet from the substrate and process it
      else if(sub.incoming()) {
         Packet p = sub.receive();
      // if it's a data packet, ack it and add it
      //    to receive buffer as appropriate
         if(p.type == 0) {
            // If this packet is as expected
            if(p.seqNum == expSeqNum) {
               recvBuf[expSeqNum] = p;
         lastRcvd = expSeqNum;
         expSeqNum = incr(expSeqNum);
      }
      // Send ACK
      Packet ack_p = new Packet();
      ack_p.type = 1;
      ack_p.seqNum = lastRcvd;
      sub.send(ack_p);

   }
// if it's an ack, update the send buffer and
//    related data as appropriate
//    reset the timer if necessary
   else {
      short seqNum = p.seqNum;
      // If it's not a duplicated ACK
      if(incr(seqNum) != sendBase) {
         sendBase = incr(seqNum);
         dupAcks = 0;
         // reset timer
         sendAgain = now + timeout;
      }
      else {
         dupAcks += 1;
      }
      // Three duplicated ACKs (plus one correctly received)
      // Retransmission
      if(dupAcks == 3) {
         // reset timer
         // that's the only way to do retransmission again
         // Only until ACK is received, dupAcks will be reset
         sendAgain = now + timeout;
         for(short i = sendBase; i != sendSeqNum; i = incr(i)) {
            sub.send(sendBuf[i]);
         }
      }
   }
}
// else if the resend timer has expired, re-send all
//      packets in the window and reset the timer
else if(now >= sendAgain) {
   sendAgain = now + timeout;
   for(short i = sendBase; i != sendSeqNum; i = incr(i)) {
      sub.send(sendBuf[i]);
   }
}
// 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
//      create a packet containing the message,
// and send it, after updating the send buffer
// and related data
else if(fromSrc.size() > 0 &&
      diff(sendSeqNum, sendBase) < wSize &&
            sub.ready()) {
         Packet p = new Packet();
         p.type = 0;
         p.seqNum = sendSeqNum;
         p.payload = fromSrc.poll();
         sub.send(p);
         sendBuf[sendSeqNum] = p;
         sendSeqNum = incr(sendSeqNum);
         // If this is the first packet, set timer
         if(diff(sendSeqNum, sendBase) == 1) {
            sendAgain = now + timeout;
         }
      }
      // 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);
         }
      }
   }
}
```

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 code. Paste a copy of the output below.

Note: if there are errors in your code, you might need to kill the process running the server in order to free up the port this script uses. To do this, use the command "pkill -9 java".

```
wuyuanpeidembp:lab4 wuyuanpei$ ./script0
wSize= 5  timeout= .5  dropProb= .25
************** client report ****************
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[0] testing 0
discarding data[1] testing 1
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[2] testing 2
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[3] testing 3
discarding data[4] testing 4
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[0]
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[0]
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[5] testing 5
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[0]
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[1] testing 1
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[2] testing 2
discarding data[3] testing 3
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[4] testing 4
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[5] testing 5
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[1]
discarding data[6] testing 6
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[2]
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[2]
discarding data[7] testing 7
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[3] testing 3
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[4] testing 4
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[5] testing 5
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[6] testing 6
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[7] testing 7
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[3]
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[8] testing 8
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[4]
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[5]
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[6]
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[7]
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 data[9] testing 9
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[8]
/127.0.0.1:65493 received from /127.0.0.1:11313 ack[9]
/127.0.0.1:65493 received from /127.0.0.1:11313 data[0] testing 0
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 ack[0]
/127.0.0.1:65493 received from /127.0.0.1:11313 data[1] testing 1
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 ack[1]
/127.0.0.1:65493 received from /127.0.0.1:11313 data[2] testing 2
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 ack[2]
```

```
/127.0.0.1:65493 received from /127.0.0.1:11313 data[3] testing 3
discarding ack[3]
/127.0.0.1:65493 received from /127.0.0.1:11313 data[4] testing 4
discarding ack[4]
/127.0.0.1:65493 received from /127.0.0.1:11313 data[5] testing 5
/127.0.0.1:65493 sending to localhost/127.0.0.1:11313 ack[5]
  Sender: sent 20 data packets, 6 acks
          discarded 5 data packets, 2 acks
          runLength 2.192538931
Receiver: received 6 data packets, 13 acks
          discarded 0 arrivals
          runLength 2.085731619
  SrcSnk: sent 10, received 6
          runLength 0.3
************** server report ****************
/127.0.0.1:11313 received from /127.0.0.1:65493 data[0] testing 0
/127.0.0.1:11313 received from /127.0.0.1:65493 data[2] testing 2
/127.0.0.1:11313 received from /127.0.0.1:65493 data[3] testing 3
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[0]
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[0]
discarding ack[0]
/127.0.0.1:11313 received from /127.0.0.1:65493 data[5] testing 5
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[0]
/127.0.0.1:11313 received from /127.0.0.1:65493 data[1] testing 1
/127.0.0.1:11313 received from /127.0.0.1:65493 data[2] testing 2
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[1]
/127.0.0.1:11313 received from /127.0.0.1:65493 data[4] testing 4
/127.0.0.1:11313 received from /127.0.0.1:65493 data[5] testing 5
discarding ack[2]
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[2]
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[2]
/127.0.0.1:11313 received from /127.0.0.1:65493 data[3] testing 3
/127.0.0.1:11313 received from /127.0.0.1:65493 data[4] testing 4
/127.0.0.1:11313 received from /127.0.0.1:65493 data[5] testing 5
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[3]
/127.0.0.1:11313 received from /127.0.0.1:65493 data[6] testing 6
/127.0.0.1:11313 received from /127.0.0.1:65493 data[7] testing 7
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[4]
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[5]
/127.0.0.1:11313 received from /127.0.0.1:65493 data[8] testing 8
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[6]
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[7]
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[8]
/127.0.0.1:11313 received from /127.0.0.1:65493 data[9] testing 9
/127.0.0.1:11313 sending to /127.0.0.1:65493 ack[9]
/127.0.0.1:11313 sending to /127.0.0.1:65493 data[0] testing 0
/127.0.0.1:11313 received from /127.0.0.1:65493 ack[0]
/127.0.0.1:11313 sending to /127.0.0.1:65493 data[1] testing 1
/127.0.0.1:11313 received from /127.0.0.1:65493 ack[1]
/127.0.0.1:11313 sending to /127.0.0.1:65493 data[2] testing 2
/127.0.0.1:11313 received from /127.0.0.1:65493 ack[2]
/127.0.0.1:11313 sending to /127.0.0.1:65493 data[3] testing 3
/127.0.0.1:11313 sending to /127.0.0.1:65493 data[4] testing 4
/127.0.0.1:11313 sending to /127.0.0.1:65493 data[5] testing 5
```

```
/127.0.0.1:11313 received from /127.0.0.1:65493 ack[5]
  Sender: sent 6 data packets, 15 acks
          discarded 0 data packets, 2 acks
          runLength 2.086664752
Receiver: received 15 data packets, 4 acks
          discarded 0 arrivals
          runLength 2.192236228
  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?

   The client side sent 20 data packets(including both transmitted and discarded). However, for the client in the application layer, it only sent 10 packets. Therefore, 20-10=10 packets are because of retransmission.

   In our experiment, all of the 10 retransmissions marked in yellow are due to discarding of data packets.

2. 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 run length for this test is 0.3 second in application layer. However, rdt in transport layer actually need more time to finish all the transmissions. In our example, the sender in client side took 2.1925 seconds; the receiver took 2.0857 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.

```
wuyuanpei@onlusr:~/473/lab4$ ./script1
wSize= 5  timeout= .5  dropProb= .25
************** client report ****************
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[0] testing 0
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[1] testing 1
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[2] testing 2
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[3] testing 3
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[4] testing 4
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[0] testing 0
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[1] testing 1
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[2] testing 2
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[3] testing 3
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[4] testing 4
/192.168.4.2:56605 received from /192.168.7.1:11313 data[0] testing 0
/192.168.4.2:56605 received from /192.168.7.1:11313 data[2] testing 2
/192.168.4.2:56605 received from /192.168.7.1:11313 data[3] testing 3
/192.168.4.2:56605 received from /192.168.7.1:11313 data[4] testing 4
/192.168.4.2:56605 received from /192.168.7.1:11313 data[0] testing 0
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 ack[0]
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 ack[0]
discarding ack[0]
discarding ack[0]
discarding ack[0]
/192.168.4.2:56605 received from /192.168.7.1:11313 data[1] testing 1
/192.168.4.2:56605 received from /192.168.7.1:11313 data[2] testing 2
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 ack[1]
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 ack[2]
/192.168.4.2:56605 received from /192.168.7.1:11313 data[3] testing 3
/192.168.4.2:56605 received from /192.168.7.1:11313 data[4] testing 4
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 ack[3]
discarding ack[4]
/192.168.4.2:56605 received from /192.168.7.1:11313 ack[0]
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[5] testing 5
/192.168.4.2:56605 received from /192.168.7.1:11313 ack[1]
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[6] testing 6
/192.168.4.2:56605 received from /192.168.7.1:11313 ack[4]
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[7] testing 7
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[8] testing 8
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 data[9] testing 9
/192.168.4.2:56605 received from /192.168.7.1:11313 data[5] testing 5
discarding ack[5]
/192.168.4.2:56605 received from /192.168.7.1:11313 ack[5]
/192.168.4.2:56605 received from /192.168.7.1:11313 ack[7]
/192.168.4.2:56605 received from /192.168.7.1:11313 ack[8]
/192.168.4.2:56605 received from /192.168.7.1:11313 ack[9]
/192.168.4.2:56605 received from /192.168.7.1:11313 data[4] testing 4
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 ack[5]
```

```
/192.168.4.2:56605 received from /192.168.7.1:11313 data[5] testing 5
/192.168.4.2:56605 sending to h7x1/192.168.7.1:11313 ack[5]
  Sender: sent 15 data packets, 12 acks
          discarded 0 data packets, 5 acks
          runLength 1.096745583
Receiver: received 12 data packets, 7 acks
          discarded 0 arrivals
          runLength 0.600155227
  SrcSnk: sent 10, received 6
          runLength 0.3
************** server report ***************
/192.168.7.1:11313 received from /192.168.4.2:56605 data[0] testing 0
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[0] testing 0
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[1] testing 1
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[2] testing 2
discarding data[3] testing 3
discarding data[4] testing 4
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[0] testing 0
discarding data[1] testing 1
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[2] testing 2
discarding data[3] testing 3
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[4] testing 4
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[0] testing 0
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[1] testing 1
discarding data[2] testing 2
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[3] testing 3
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[4] testing 4
discarding data[0] testing 0
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[1] testing 1
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[2] testing 2
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[3] testing 3
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[4] testing 4
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[0] testing 0
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[1] testing 1
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[2] testing 2
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[3] testing 3
discarding data[4] testing 4
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[0] testing 0
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[1] testing 1
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[2] testing 2
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[3] testing 3
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[4] testing 4
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[0] testing 0
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[1] testing 1
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[2] testing 2
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[3] testing 3
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[4] testing 4
discarding ack[0]
/192.168.7.1:11313 received from /192.168.4.2:56605 data[0] testing 0
/192.168.7.1:11313 received from /192.168.4.2:56605 data[1] testing 1
/192.168.7.1:11313 sending to /192.168.4.2:56605 ack[0]
/192.168.7.1:11313 sending to /192.168.4.2:56605 ack[1]
/192.168.7.1:11313 received from /192.168.4.2:56605 data[2] testing 2
/192.168.7.1:11313 received from /192.168.4.2:56605 data[3] testing 3
```

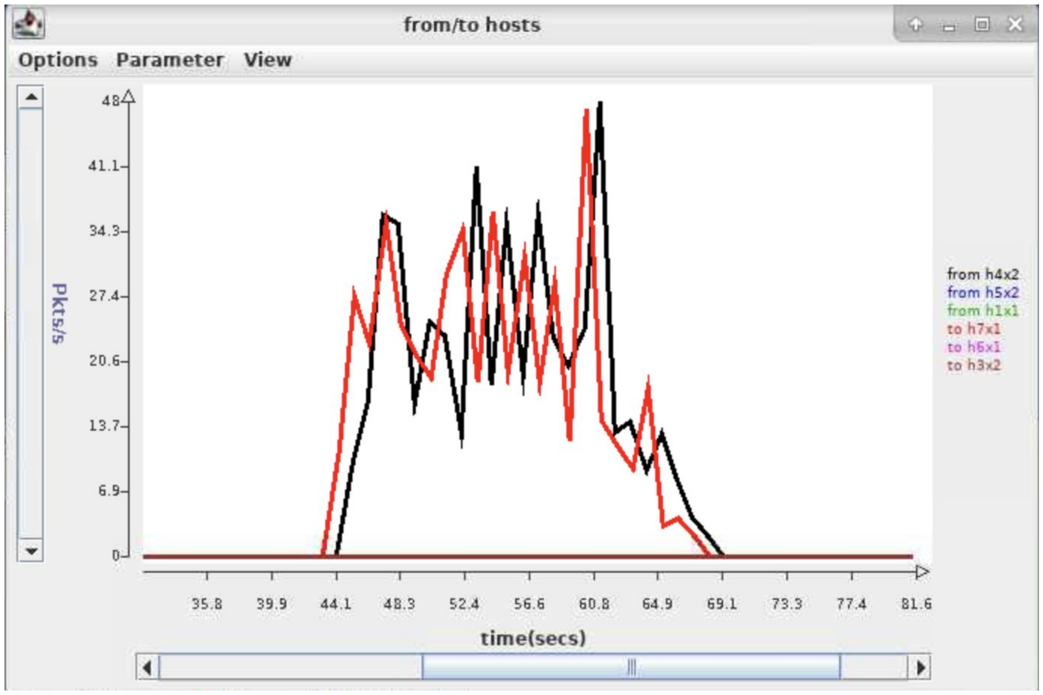
```
discarding ack[2]
discarding ack[3]
/192.168.7.1:11313 received from /192.168.4.2:56605 data[4] testing 4
/192.168.7.1:11313 sending to /192.168.4.2:56605 ack[4]
/192.168.7.1:11313 received from /192.168.4.2:56605 ack[0]
/192.168.7.1:11313 received from /192.168.4.2:56605 ack[0]
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[5] testing 5
/192.168.7.1:11313 received from /192.168.4.2:56605 ack[1]
/192.168.7.1:11313 received from /192.168.4.2:56605 ack[2]
/192.168.7.1:11313 received from /192.168.4.2:56605 ack[3]
/192.168.7.1:11313 received from /192.168.4.2:56605 data[5] testing 5
/192.168.7.1:11313 sending to /192.168.4.2:56605 ack[5]
/192.168.7.1:11313 received from /192.168.4.2:56605 data[6] testing 6
discarding ack[6]
/192.168.7.1:11313 received from /192.168.4.2:56605 data[7] testing 7
/192.168.7.1:11313 received from /192.168.4.2:56605 data[8] testing 8
/192.168.7.1:11313 sending to /192.168.4.2:56605 ack[7]
/192.168.7.1:11313 sending to /192.168.4.2:56605 ack[8]
/192.168.7.1:11313 received from /192.168.4.2:56605 data[9] testing 9
/192.168.7.1:11313 sending to /192.168.4.2:56605 ack[9]
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[4] testing 4
/192.168.7.1:11313 sending to /192.168.4.2:56605 data[5] testing 5
/192.168.7.1:11313 received from /192.168.4.2:56605 ack[5]
/192.168.7.1:11313 received from /192.168.4.2:56605 ack[5]
  Sender: sent 38 data packets, 11 acks
          discarded 7 data packets, 4 acks
          runLength 0.612288749
Receiver: received 11 data packets, 7 acks
          discarded 0 arrivals
          runLength 0.855701554
  SrcSnk: sent 6, received 10
          runLength 0.3
```

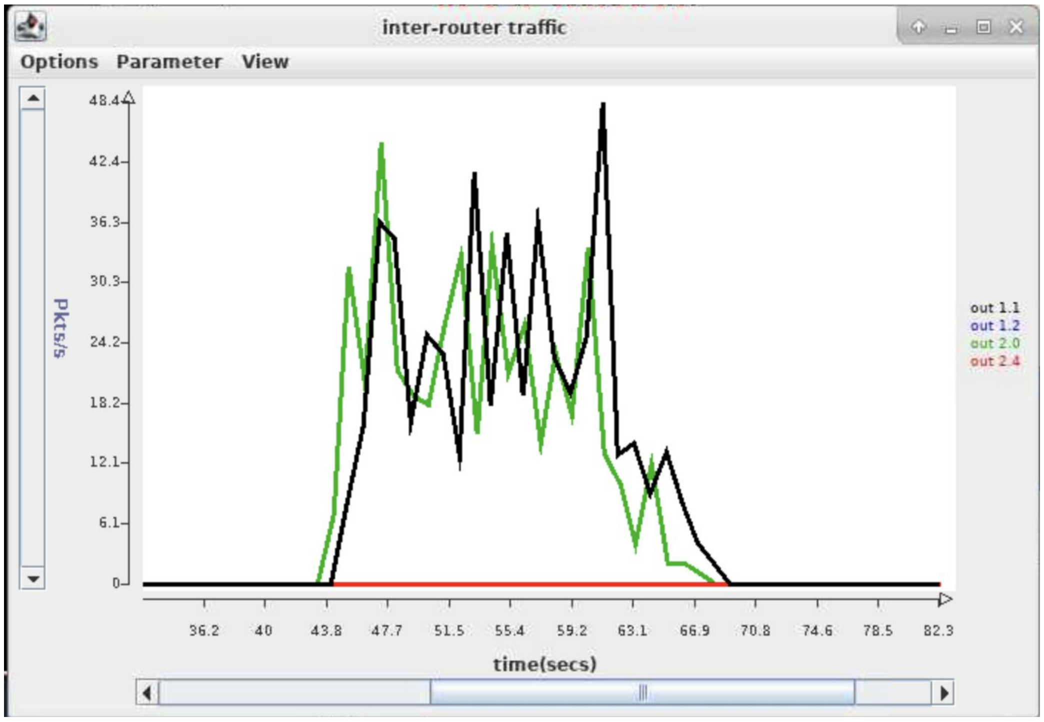
Part D. **[15 points]** Use the provided *script2* to run this next test. Paste a copy of the output below. Also, paste a screen capture showing the two monitoring windows labeled "from/to hosts" and "inter-router traffic". Make sure your screenshot shows the curves for the entire duration of the script 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

wuyuanpei@onlusr:~/473/lab4$ ./script2
wSize= 5  timeout= .5  dropProb= .2
************** client report ****************
  Sender: sent 464 data packets, 164 acks
          discarded 93 data packets, 35 acks
          runLength 22.871336645
Receiver: received 164 data packets, 284 acks
          discarded 0 arrivals
          runLength 22.478451894
  SrcSnk: sent 200, received 100
          runLength 20.0
************** server report ****************
  Sender: sent 249 data packets, 369 acks
          discarded 55 data packets, 84 acks
          runLength 22.485232059
Receiver: received 369 data packets, 129 acks
          discarded 0 arrivals
          runLength 22.57708297
  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.)

   Specified run length is 20 seconds.

   Specified delta time for server is 0.2 second per packet. i.e. rate is 5 packets/second

   Specified delta time for client is 0.1 second per packet. i.e. rate is 10 packets/second

2. 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 sender at client side took 22.87 seconds to deliver all the packets.

   Effective packet delivery date is 200 packets/22.87 seconds = 8.745 packets/second

3. 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 464 data packets in total. However, only 200 are effective. Therefore 464-200=264 are retransmissions.

   The total number of packets sent is 464+164=628 packets

   628/22.87=27.460 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.

   Approximately 50ms (50.131ms on average).

   Since at most 1 packet will be sent successfully within each RTT, the maximum rate will be 1 packet/ 50 ms = 20 packets/second.

   In order to send 1000 packets/second, for each RTT = 50ms, 50 packets need to be sent successfully. Therefore the window size should be at least 50.

2. 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 sending rate is 1 packet/ 0.004 second = 250 packets/second.

   The smallest window size will be 12.5 that rounds to 13.

```
jim@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 9.995794697
Receiver: received 0 data packets, 2500 acks
          discarded 0 arrivals
          runLength 9.94926963
  SrcSnk: sent 2500, received 0
          runLength 10.0
```

```
************** server report ****************
  Sender: sent 0 data packets, 2500 acks
          discarded 0 data packets, 0 acks
          runLength 9.955607265
Receiver: received 2500 data packets, 0 acks
          discarded 0 arrivals
          runLength 10.049720058
  SrcSnk: sent 0, received 2500
          runLength 10.0
```

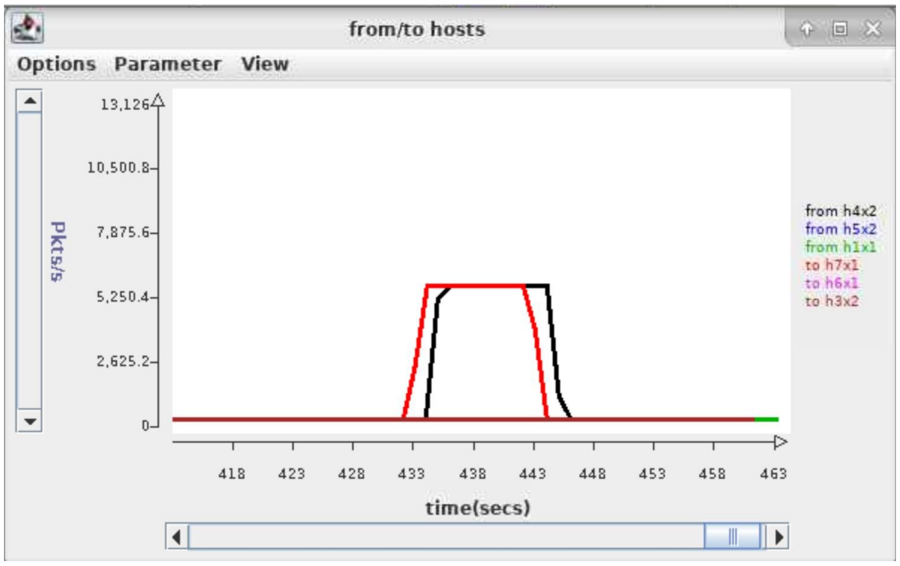
Yes, because it took approximately 10 seconds to transmit 2500 packets successfully. Therefore the packets were delivered at the rate of 250 packets/second, which is the specified sending rate.

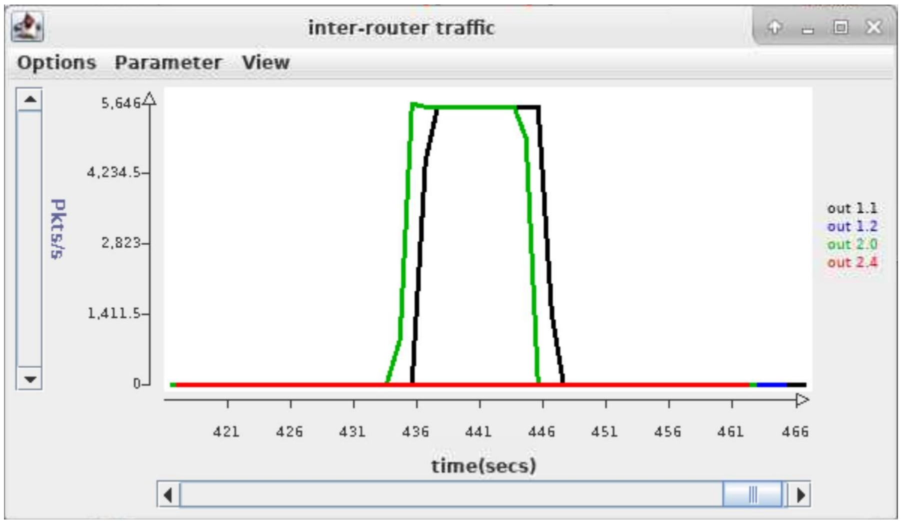
3. In this part, you are to determine the maximum rate at which you can send traffic between the two routers. Determine the maximum sending rate by decreasing the delta value, while increasing the window size to match (keep the timeout value at 0.6 and the discard probability at 0). Observe the packet 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.

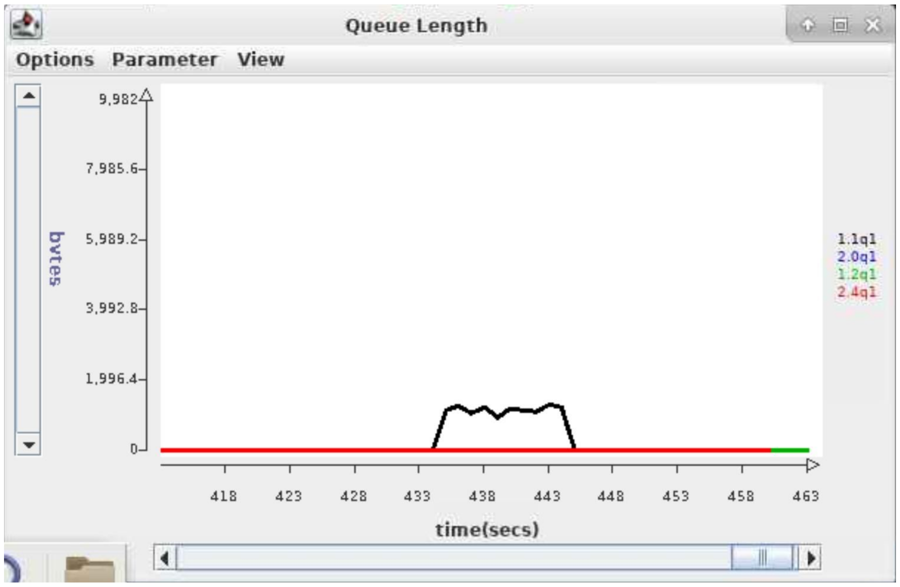
```
your output here

jim@onlusr:~/cse_473/lab4$ ./script3 313 0.6 0 0.00016
wSize= 313  timeout= 0.6  dropProb= 0  delta= 0.00016
************** client report ****************
  Sender: sent 56052 data packets, 0 acks
          discarded 0 data packets, 0 acks
          runLength 10.180591984
Receiver: received 0 data packets, 56052 acks
          discarded 0 arrivals
          runLength 10.137577821
  SrcSnk: sent 56052, received 0
          runLength 10.0
************** server report ****************
  Sender: sent 0 data packets, 56052 acks
          discarded 0 data packets, 0 acks
          runLength 10.120814817
Receiver: received 56052 data packets, 0 acks
          discarded 0 arrivals
          runLength 10.215437636
  SrcSnk: sent 0, received 56052
          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?

About 56052 packets /10.18 second = 5506 packets/second.

The specified sending rate is 1 packet / 0.00016 second = 6250 packets/second.

Yes. As we can observe from the monitor window, the queueing is about 1600 bytes long.

4. 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?

for wSize = 300: 736 Bytes

for wSize = 400: 5198 Bytes

for wSize = 450: 7682 Bytes

for wSize = 500: 9982 Bytes

How does the throughput compare for these four cases?

The throughputs are approximately the same, which is about 5500 packets/second.

For the first three cases, the packets were sent at this same rate. But in the last case where wSize = 500, packets were actually sent at approximately 7600 packets/second, although throughput (i.e. rate of packets successfully sent) still remains the same.

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

In both four cases, the sending rate reaches the maximum rate the link can forward packets. Therefore the throughputs are approximately the same.

But in the last case where wSize = 500, the length of queue reaches 9982 Bytes, which is very close to the threshold of 10000 bytes of the queue table at port 1 of router 1. Therefore many packets are lost (since the queue table is full and incoming packets are discarded). So client has to send packets at a higher rate to compensate for the packets loss described above.