CSCI4211: Introduction to Computer Networks

Homework Assignment II

Due 11:55 PM October 21st, 2015

Help-hot-line: csci4211-help@cs.umn.edu

Name: Yanbang Liu   Student ID:4446044

Important Notes:

Please submit your solutions as a single archive file (.zip or .tar or .tar.gz) on moodle. You may download the MS Word version of this assignment from moodle and edit it directly. Only online submissions are accepted for this assignment - do not attempt to submit hard copies. All textbook references pertain to the 6th edition.

You may discuss ideas and ask for clarifications freely with others on or off the class forum, and with Professor He or with the TAs. You must not provide or accept other assistance on the assignments. Feel free to post any queries you might have on the moodle discussion forum for this assignment.

Please do not write anything other than name and student ID on this cover page

For TA only:

Problem	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	50	
Total	**100**	

1. Definitions (10 pt. 1 pt. each)

Please read Chapter 3 and define following terminologies briefly.

- Logical Communication

From view of application, two processes on two hosts are connected directly, while in fact they are connected by the complext networks.

- Multiplexing and Demultiplexing

Multiplexing: the process that the transport layer at the sender host collects message from different processes (applications), add headers and wrap them into segments, and send them into network layer.
Demultiplexing: the process that the transport layer at the reciever host receives the segments from network layer, extract the data from the segments, and then sent the data to corresponding processes.

- Little Endian

An integer format that stores the least significant bytess first.

- Handshaking

The process that connetion oriented protocols use to establish connections between client and server.

- Flow Control

The recieving application might be busy when a data packet arrives. These data packets are saved in the receiver buffer and the buffer could overflow sometimes. Flow control is a machanism to prevent the sender data overflow the reciever buffer when the application is busy.

- Congestion Control

Congestion is caused when there are too many sources sending data at a high transmission rate at the same time. Congestion control is a machanism to control the network congestion (change window size, etc.) hence prevent packet loss in the network.

- Go-Back-N algorithm

This algorithm allows the GBN protocal to transmit N packets at the same time without receiving acknowledgement. When a seccesive packet is acknowledged, the size-N window slides forward by 1. However, if any packet in the window is lost, the algorithm retransmit all N packtes in the window.

- Selective Repeat algorithm

An algorithm that also allows N packets (size-N window) to be transmitted at the same time. However, instead of retransmiting all the packets in the window, the SR algorithm only retransmit the packets that are suspected missing or corrupted.

- Fairness

To ensure that all connections passing trough a bottleneck link are not congested and have enough transmition rate to send and recieve data.

- Slow Start

A TCP connection usually start at a small initial sending rate of MSS/RTT.

2. TCP Reliable Data Transfer (10 pts)

Consider two nodes which are connected by an 8Mbps link (assume 1Mbps = 1000Kbps) and RTT is 0.05 sec. Assume the size of each packet is 8K bits.

Answer the following questions for ARQ schemes:

(a) Assume that the link is error-free: what is the maximum possible rate of transmission for Stop-and-wait, GBN, and SR respectively? Why? (**6 pts**)

Stop-and-wait:

The packet is pushed into the network, the transmission time is 8K/8Mbps = 0.001 sec. The total propagation time is 0.05 sec. Thus, 8K is transmitted over 0.051 sec. Therefore rate of transmission = 8Kb/0.051s ~= 156.9Kbps.

GBN & SR:

Assume the sender keeps busy sending data all the time (assume this for maximum rate of transmission). Since there is no error, the pipe is always filled. Therefore the maximum possible rate of transmission would be 8Mbps as the entire link is filled with packets.

(b) For GBN, in order to allow sender to continuously send packets without any waiting, what is the minimum window size in terms of the number of packets? (**1 pt**)

The transmission time of one packet as calculated above is 0.001 sec, and the RTT is 0.05 sec, thus in order to fill the pipe, there should be 0.05/0.001 = 50 packets being sent out while the sender waiting for ACK from the receiver. Thus the minimum window size is 50.

(c) Suppose that we transmit 20 packets with sequence number from 1 to 20. The packet with sequence number 16 is lost and all other packets are received correctly. Assuming there is no ACK lost, for stop-and-wait, GBN, and SR, which packets have been retransmitted? (**3 pt**)

For stop and wait, and SR, only the packet with sequence number 16 is retransmitted. However for GBN, all packets after sequence number 16 (i.e. 16-20) are re transmitted

3. TCP Sequence Numbers (10 pts)

Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 168. Suppose that Host A then sends two segments to Host B back-to-back. The first and second segments contain 20 and 40 bytes of data, respectively. In the first segment, the sequence number is 169, source port number is 303, and the destination port number is 80. Host B sends an acknowledgement whenever it receives a segment from Host A.

a) In the second segment sent from Host A to B, what is the sequence number, source port number, and destination port number? (**3pts**)

170, 303, and 80

b) If the first segment arrives before the second segment, in the acknowledgement of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number? (**3pts**)

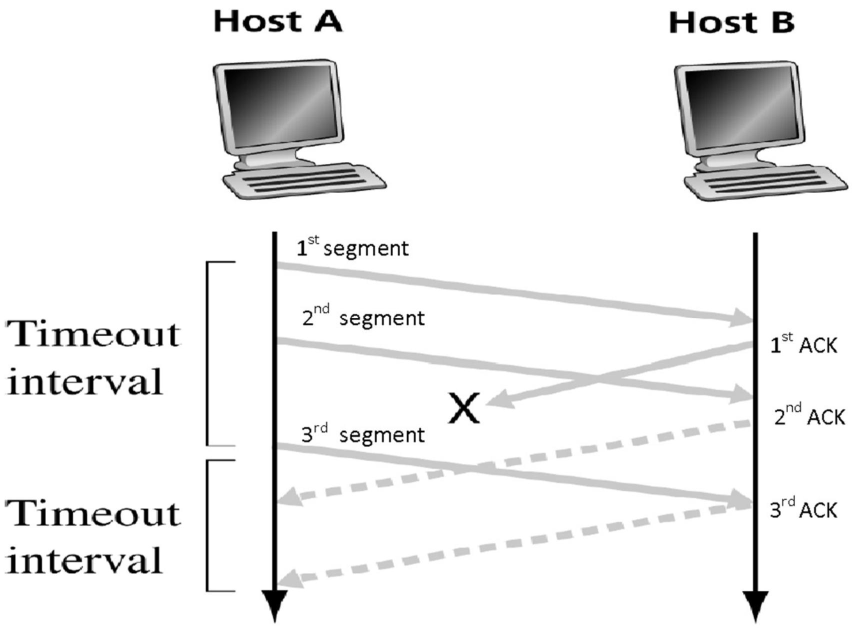
169, 80, and 303

c) If the second segment arrives before the first segment (out of order arrival), in the acknowledgement of the first arriving segment, what is the acknowledgment number? (**1pt**)

170

d) Suppose the two segments sent by A arrive in order at B. The first acknowledgement is lost and the second acknowledgement arrives after the first timeout interval, as shown in the figure below. Please provide the sequence number for the third (retransmitted) data segment (**1 pt**) and provide the acknowledgement number for the 2nd and 3rd acknowledgement. (**2pts**)

169; 170, and 169.


Host A
Host B
1st segment
2nd segment
3rd segment
1st ACK
2nd ACK
3rd ACK
X
Timeout interval
Timeout interval

4. Hands-on Practice I: UDP (10 pts)

In this practice, we'll take a quick look at the UDP transport protocol. As we saw in Chapter 3, UDP is a connectionless non-thrills protocol. Start capturing packets in Wireshark and then do something that will cause your host to send and receive several UDP packets. For example, use telnet [domain name].

Please answer the following question

(a) Select one packet. From this packet, determine how many fields there are in the UDP header. (Do not look in the textbook! Answer these questions directly from what you observe in the packet trace.) Name these fields. (2 pts)

4 fields are displayed:
Source Port,
Destination Port,
Length, and
Checksum
(The one in a bracket " [ ] ": Stream index is ignored as it is not in the datagram)

(b) What is the maximum number of bytes that can be included in a UDP payload? (1pt)

Max buffer length = 65535 bytes
IP Header length = 20 bytes
UDP Header length = 8 bytes
Thus payload length = 65535 - 20 - 8 = 65507

(c) What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation. (To answer this question, you'll need to look into the IP header.) (2pts)

17 in decimal
0x11 in hexadecimal

(d) Search "UDP" in Google and determine the fields over which the UDP checksum is calculated. Capture a VERY SMALL UDP packet. Manually verify the checksum in this packet. Show all work and explain all steps. (5 pts) You need to paste a screenshot of the UDP packet content as the evidence.

What we need (in 16-bit hexadecimal):
Source address: 0xc0a8, 0x0168
Destination address: 0xc0a8, 0x0101
Protocol: 0x0011
UDP length: 12 -> 0x000c

Add source and destination address together:
0xc0a8 + 0x0168 + 0xc0a8 + 0x0101 = 0x183b9

a) Add the result with Protocol number and UDP length:
0x183b9 + 0x0011 + 0x000c = 0x183d6

b) Add the UDP datagram (skip checksum):
0xc94e + 0x00c0 + 0x000c + 0x0801 + 0x0310 = 0xd52b

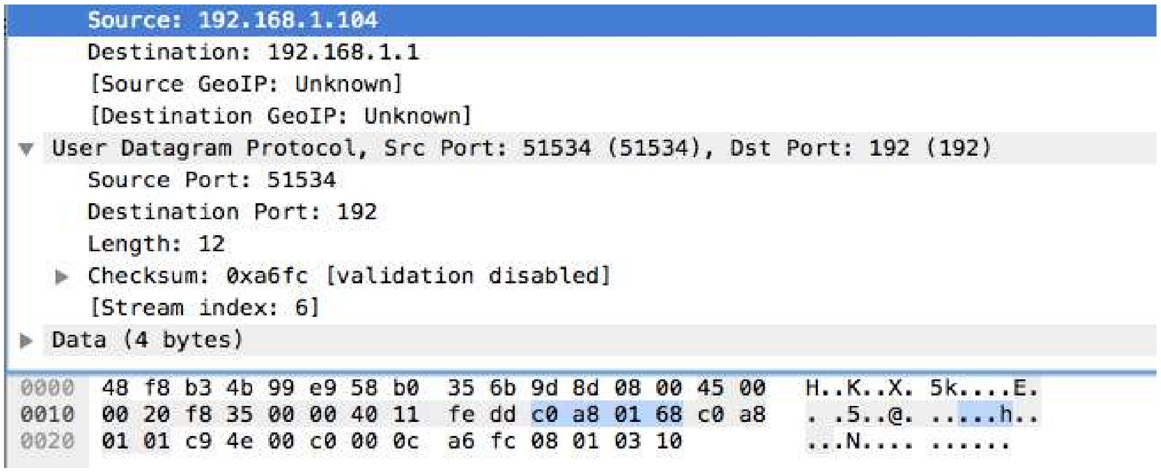
Add the results from a) and b)
0x183d6 + 0xd52b = 0x25901

Add the higher half to the lower half:
0x0002 + 0x5901 = 0x5903

Compliment the result:
~0x5903 = 0xa6fc

And this is correct.

Screenshot:



```
    Source: 192.168.1.104
    Destination: 192.168.1.1
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▼ User Datagram Protocol, Src Port: 51534 (51534), Dst Port: 192 (192)
    Source Port: 51534
    Destination Port: 192
    Length: 12
  ▶ Checksum: 0xa6fc [validation disabled]
    [Stream index: 6]
▶ Data (4 bytes)

0000  48 f8 b3 4b 99 e9 58 b0  35 6b 9d 8d 08 00 45 00   H..K..X. 5k....E.
0010  00 20 f8 35 00 00 40 11  fe dd c0 a8 01 68 c0 a8   . .5..@. .....h..
0020  01 01 c9 4e 00 c0 00 0c  a6 fc 08 01 03 10         ...N.... ......
```

5. Hands-on Practice II: TCP (10 pts)

In this practice, we'll investigate the behavior of TCP in detail. Before beginning our exploration of TCP, we'll need to use Wireshark to obtain a packet trace of the TCP. You'll do so by accessing:

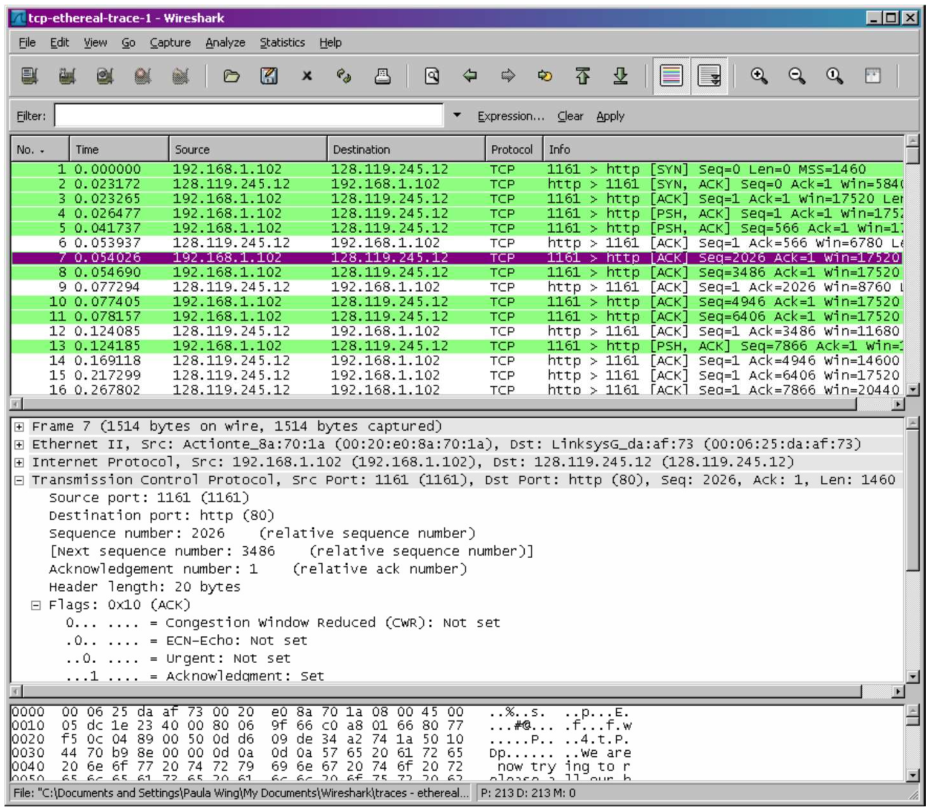
http://www-users.cs.umn.edu/~tianhe/csci4211/HTTP-2.htm

First, filter the packets displayed in the Wireshark window by entering "tcp" into the display filter window towards the top of the Wireshark window. What you should see is series of TCP and HTTP messages between your computer and

Please answer the following question (along with screenshots as the evidence of your answer)

(a) What is the IP address and TCP port number used by the client computer (source) that downloads the bill of rights from www-users.cselabs.umn.edu? You need to paste an appropriate screenshot as the evidence. (2pts)
192.168.1.101:50172

Since this lab is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select *Analyze->Enabled Protocols.* Then uncheck the HTTP box and select *OK*. You should now see a Wireshark window that looks like:


tcp-ethereal-trace-1 - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. -  Time       Source           Destination      Protocol  Info
     1 0.000000   192.168.1.102    128.119.245.12   TCP       1161 > http [SYN] Seq=0 Len=0 MSS=1460
     2 0.023172   128.119.245.12   192.168.1.102    TCP       http > 1161 [SYN, ACK] Seq=0 Ack=1 Win=584
     3 0.023265   192.168.1.102    128.119.245.12   TCP       1161 > http [ACK] Seq=1 Ack=1 Win=17520 Le
     4 0.026477   192.168.1.102    128.119.245.12   TCP       1161 > http [PSH, ACK] Seq=1 Ack=1 Win=175
     5 0.041737   192.168.1.102    128.119.245.12   TCP       1161 > http [PSH, ACK] Seq=566 Ack=1 Win=1
     6 0.053937   128.119.245.12   192.168.1.102    TCP       http > 1161 [ACK] Seq=1 Ack=566 Win=6780 L
     7 0.054026   192.168.1.102    128.119.245.12   TCP       1161 > http [ACK] Seq=2026 Ack=1 Win=17520
     8 0.054690   192.168.1.102    128.119.245.12   TCP       1161 > http [ACK] Seq=3486 Ack=1 Win=17520
     9 0.077294   128.119.245.12   192.168.1.102    TCP       http > 1161 [ACK] Seq=1 Ack=2026 Win=8760
    10 0.077405   192.168.1.102    128.119.245.12   TCP       1161 > http [ACK] Seq=4946 Ack=1 Win=17520
    11 0.078157   192.168.1.102    128.119.245.12   TCP       1161 > http [ACK] Seq=6406 Ack=1 Win=17520
    12 0.124085   128.119.245.12   192.168.1.102    TCP       http > 1161 [ACK] Seq=1 Ack=3486 Win=11680
    13 0.124185   192.168.1.102    128.119.245.12   TCP       1161 > http [PSH, ACK] Seq=7866 Ack=1 Win=
    14 0.169118   128.119.245.12   192.168.1.102    TCP       http > 1161 [ACK] Seq=1 Ack=4946 Win=14600
    15 0.217299   128.119.245.12   192.168.1.102    TCP       http > 1161 [ACK] Seq=1 Ack=6406 Win=17520
    16 0.267802   128.119.245.12   192.168.1.102    TCP       http > 1161 [ACK] Seq=1 Ack=7866 Win=20440
```

```
⊞ Frame 7 (1514 bytes on wire, 1514 bytes captured)
⊞ Ethernet II, Src: Actionte_8a:70:1a (00:20:e0:8a:70:1a), Dst: LinksysG_da:af:73 (00:06:25:da:af:73)
⊞ Internet Protocol, Src: 192.168.1.102 (192.168.1.102), Dst: 128.119.245.12 (128.119.245.12)
⊟ Transmission Control Protocol, Src Port: 1161 (1161), Dst Port: http (80), Seq: 2026, Ack: 1, Len: 1460
    Source port: 1161 (1161)
    Destination port: http (80)
    Sequence number: 2026    (relative sequence number)
    [Next sequence number: 3486    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
    Header length: 20 bytes
  ⊟ Flags: 0x10 (ACK)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...1 .... = Acknowledgment: Set
```

```
0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s.  ..p...E.
0010  05 dc 1e 23 40 00 80 06  9f 66 c0 a8 01 66 80 77   ...#@... .f...f.w
0020  f5 0c 04 89 00 50 0d d6  09 de 34 a2 74 1a 50 10   .....P.. ..4.t.P.
0030  44 70 b9 8e 00 00 0d 0a  0d 0a 57 65 20 61 72 65   Dp...... ..We are
0040  20 6e 6f 77 20 74 72 79  69 6e 67 20 74 6f 20 72    now try ing to r
```

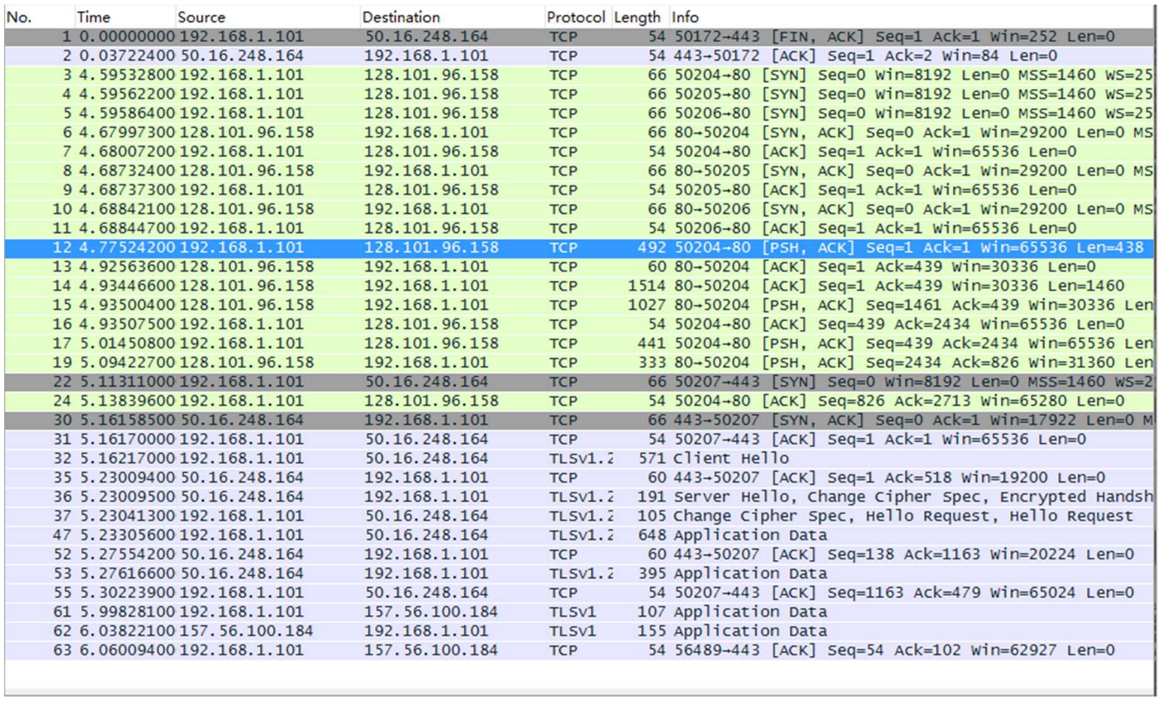
File: "C:\Documents and Settings\Paula Wing\My Documents\Wireshark\traces - ethereal... P: 213 D: 213 M: 0

Please answer the following question (along with screenshots as the back-up evidence of your answer, if applicable)

(b) What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and the webserver? What is it in the segment that identifies the segment as a SYN segment? (2pts)
There are three such segments and all their sequence numbers are 0 (relative sequence number). They have flags indicating that they are the SYN segments.

(c) What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender? (2pts)
As evidenced in the screenshot, the minimum window size is 8192 which is at the beginning of the connection establishment. The window size then grow to 65536 and then shrinks to 30336, which means throttle did happen.

(d) Are there any retransmitted segments in the trace file? What did you check for in order to answer this question? (2pts) You need to paste a screenshot as the evidence.\
There is no retransmitted segments, because when TCP retransmit packet, the packet would show in WireShark as a black field saying it is retransmitted segment. And clearly there is no such packets.
Evidence for a), b), c), and d):



No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	192.168.1.101	50.16.248.164	TCP	54	50172→443 [FIN, ACK] Seq=1 Ack=1 Win=252 Len=0
2	0.03722400	50.16.248.164	192.168.1.101	TCP	54	443→50172 [ACK] Seq=1 Ack=2 Win=84 Len=0
3	4.59532800	192.168.1.101	128.101.96.158	TCP	66	50204→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=25
4	4.59562200	192.168.1.101	128.101.96.158	TCP	66	50205→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=25
5	4.59586400	192.168.1.101	128.101.96.158	TCP	66	50206→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=25
6	4.67997300	128.101.96.158	192.168.1.101	TCP	66	80→50204 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MS
7	4.68007200	192.168.1.101	128.101.96.158	TCP	54	50204→80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
8	4.68732400	128.101.96.158	192.168.1.101	TCP	66	80→50205 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MS
9	4.68737300	192.168.1.101	128.101.96.158	TCP	54	50205→80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
10	4.68842100	128.101.96.158	192.168.1.101	TCP	66	80→50206 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MS
11	4.68844700	192.168.1.101	128.101.96.158	TCP	54	50206→80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
12	4.77524200	192.168.1.101	128.101.96.158	TCP	492	50204→80 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=438
13	4.92563600	128.101.96.158	192.168.1.101	TCP	60	80→50204 [ACK] Seq=1 Ack=439 Win=30336 Len=0
14	4.93446600	128.101.96.158	192.168.1.101	TCP	1514	80→50204 [ACK] Seq=1 Ack=439 Win=30336 Len=1460
15	4.93500400	128.101.96.158	192.168.1.101	TCP	1027	80→50204 [PSH, ACK] Seq=1461 Ack=439 Win=30336 Len
16	4.93507500	192.168.1.101	128.101.96.158	TCP	54	50204→80 [ACK] Seq=439 Ack=2434 Win=65536 Len=0
17	5.01450800	192.168.1.101	128.101.96.158	TCP	441	50204→80 [PSH, ACK] Seq=439 Ack=2434 Win=65536 Len
19	5.09422700	128.101.96.158	192.168.1.101	TCP	333	80→50204 [PSH, ACK] Seq=2434 Ack=826 Win=31360 Len
22	5.11311000	192.168.1.101	50.16.248.164	TCP	66	50207→443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
24	5.13839600	192.168.1.101	128.101.96.158	TCP	54	50204→80 [ACK] Seq=826 Ack=2713 Win=65280 Len=0
30	5.16158500	50.16.248.164	192.168.1.101	TCP	66	443→50207 [SYN, ACK] Seq=0 Ack=1 Win=17922 Len=0 M
31	5.16170000	192.168.1.101	50.16.248.164	TCP	54	50207→443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
32	5.16217000	192.168.1.101	50.16.248.164	TLSv1.2	571	Client Hello
35	5.23009400	50.16.248.164	192.168.1.101	TCP	60	443→50207 [ACK] Seq=1 Ack=518 Win=19200 Len=0
36	5.23009500	50.16.248.164	192.168.1.101	TLSv1.2	191	Server Hello, Change Cipher Spec, Encrypted Handsh
37	5.23041300	192.168.1.101	50.16.248.164	TLSv1.2	105	Change Cipher Spec, Hello Request, Hello Request
47	5.23305600	192.168.1.101	50.16.248.164	TLSv1.2	648	Application Data
52	5.27554200	50.16.248.164	192.168.1.101	TCP	60	443→50207 [ACK] Seq=138 Ack=1163 Win=20224 Len=0
53	5.27616600	50.16.248.164	192.168.1.101	TLSv1.2	395	Application Data
55	5.30223900	192.168.1.101	50.16.248.164	TCP	54	50207→443 [ACK] Seq=1163 Ack=479 Win=65024 Len=0
61	5.99828100	192.168.1.101	157.56.100.184	TLSv1	107	Application Data
62	6.03822100	157.56.100.184	192.168.1.101	TLSv1	155	Application Data
63	6.06009400	192.168.1.101	157.56.100.184	TCP	54	56489→443 [ACK] Seq=54 Ack=102 Win=62927 Len=0

(e) Based on the sequence number, you can calculate how many bytes (TCP payload) have been transferred through this TCP connection. Does the number of total bytes downloaded through this TCP connection equal to the html file size of the bill of rights? Explain why? (2 pts)
The acknowledge segment sent after the file chunk has the sequence number equals the total length of the data segment, so we can just look at the largest (and last) sequence number, in this case it is 2434 bytes.
The actual html file downloaded from chrome is 5187 bytes.
I cannot explain this.

6. Programming assignment: Network performance measurement via socket programming (50 pts)

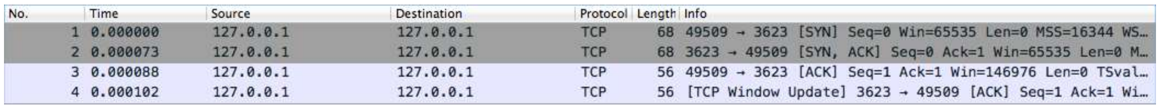
This problem requires building server and client applications in C which exchange packets over UDP and TCP. Using these applications, you will measure the performance of the network path between two hosts.

TCP performance measurement:
Build server and client applications that communicate over TCP sockets. All measurement must be performed on the client. Read and understand the questions prior to implementation.

Run both the client and the server on your home LAN for parts (a) & (b).

a. Using Wireshark, observe the TCP connection establishment process (namely, three way handshake). Repeat 5 times and provide the average time taken for connection establishment. Ignore the time for the last ACK in three way handshaking to reach the server, as this cannot be measured from the client side. Simply put, the establishment time is defined as the duration from when "synchronize" (SYN) segment (first segment) was sent out from the client until when ACK (third segment) was sent out from the client (in response to SYN & ACK segment (second segment) from the server. Also please provide a screenshot of Wireshark showing TCP connection establishment. (**5pts**)

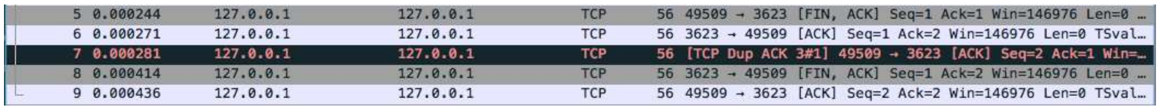
Average connection establish time = (88 + 87 + 84 + 93 + 87) / 5 = 87.8 microseconds



```
No.     Time       Source       Destination   Protocol Length Info
      1 0.000000   127.0.0.1    127.0.0.1     TCP      68 49509 → 3623 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS…
      2 0.000073   127.0.0.1    127.0.0.1     TCP      68 3623 → 49509 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 M…
      3 0.000088   127.0.0.1    127.0.0.1     TCP      56 49509 → 3623 [ACK] Seq=1 Ack=1 Win=146976 Len=0 TSval…
      4 0.000102   127.0.0.1    127.0.0.1     TCP      56 [TCP Window Update] 3623 → 49509 [ACK] Seq=1 Ack=1 Wi…
```

b. Using Wireshark, observe the TCP connection termination process, initiated by the client. Repeat 5 times and provide the average time taken for connection termination. Ignore the time for the last ACK to reach the server, as this cannot be measured from the client side. Simply put, the establishment time is defined as the duration from when FIN segment (First segment) was sent out from the client until when ACK (fourth segment) was sent out from the client in response to FIN segment (third segment) from the server. Also please provide a screenshot of Wireshark showing TCP connection termination. (**5pts**)

Average connection termination time = ((436 – 224) + (2060 – 268) + (578 – 191) + (239 – 159) + (1598 – 268)) / 5 = 760.2 microseconds

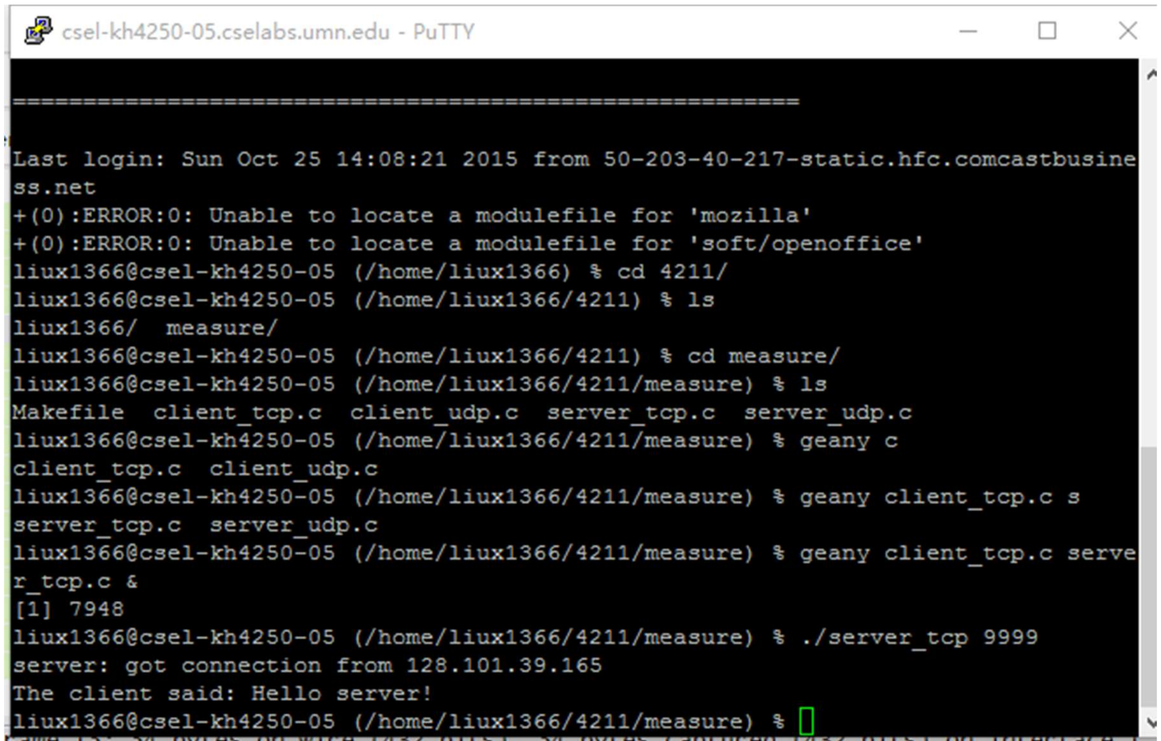


```
      5 0.000244   127.0.0.1    127.0.0.1     TCP      56 49509 → 3623 [FIN, ACK] Seq=1 Ack=1 Win=146976 Len=0 …
      6 0.000271   127.0.0.1    127.0.0.1     TCP      56 3623 → 49509 [ACK] Seq=1 Ack=2 Win=146976 Len=0 TSval…
      7 0.000281   127.0.0.1    127.0.0.1     TCP      56 [TCP Dup ACK 3#1] 49509 → 3623 [ACK] Seq=2 Ack=1 Win=…
      8 0.000414   127.0.0.1    127.0.0.1     TCP      56 3623 → 49509 [FIN, ACK] Seq=1 Ack=2 Win=146976 Len=0 …
      9 0.000436   127.0.0.1    127.0.0.1     TCP      56 49509 → 3623 [ACK] Seq=2 Ack=2 Win=146976 Len=0 TSval…
```

Now, run the client and the server on the CSELabs UNIX machines: The server can be on one of the machines in Keller Hall 4-250 and the client can be on one of the machines in Lind Hall 40.

c. Show the network path between the server and the client. Note that they need to be at least 2 hops apart. Please provide a screenshot of the traceroute command on Linux as well as a table including all the IP addresses on the path (including the client and the server) (**5 pts**)

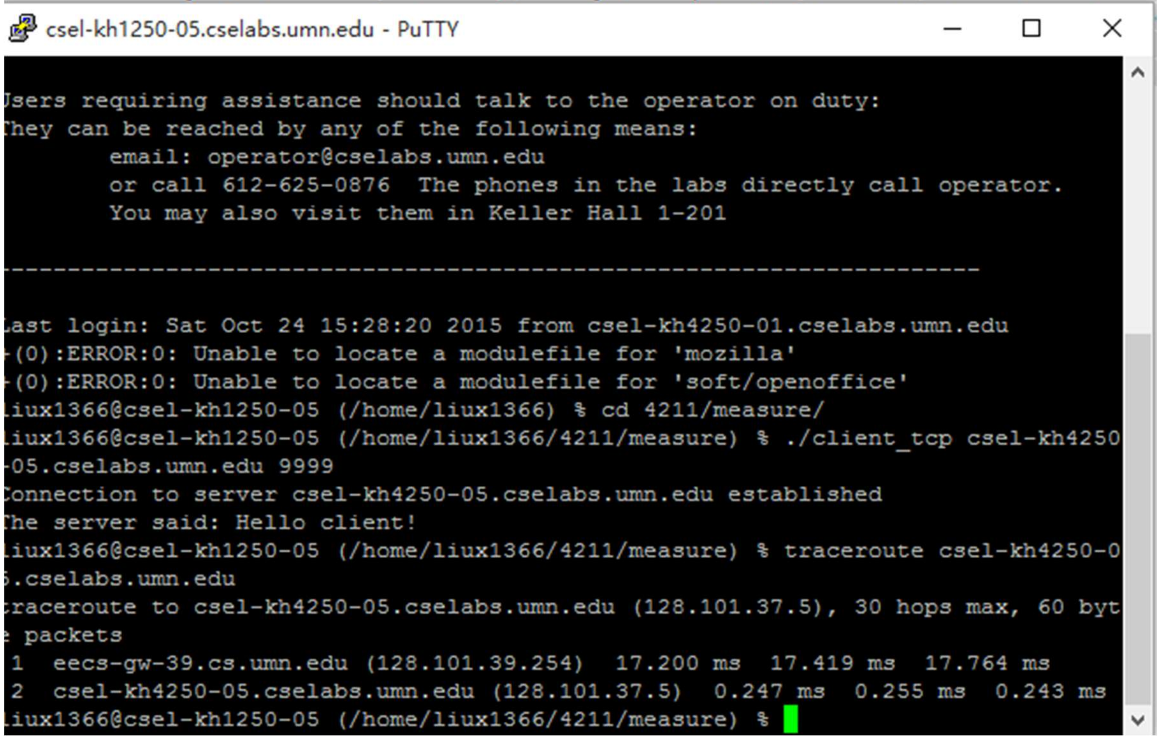
Host	IP address
Csel-kh1250-05.cselabs.umn.edu	128.101.39.165
Eecs-gw-39.cs.umn.edu	128.101.39.254
Csel-kh4250-05.cselabs.umn.edu	128.101.37.5



csel-kh4250-05.cselabs.umn.edu - PuTTY

```
========================================================

Last login: Sun Oct 25 14:08:21 2015 from 50-203-40-217-static.hfc.comcastbusine
ss.net
+(0):ERROR:0: Unable to locate a modulefile for 'mozilla'
+(0):ERROR:0: Unable to locate a modulefile for 'soft/openoffice'
liux1366@csel-kh4250-05 (/home/liux1366) % cd 4211/
liux1366@csel-kh4250-05 (/home/liux1366/4211) % ls
liux1366/  measure/
liux1366@csel-kh4250-05 (/home/liux1366/4211) % cd measure/
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % ls
Makefile  client_tcp.c  client_udp.c  server_tcp.c  server_udp.c
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % geany c
client_tcp.c  client_udp.c
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % geany client_tcp.c s
server_tcp.c  server_udp.c
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % geany client_tcp.c serve
r_tcp.c &
[1] 7948
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % ./server_tcp 9999
server: got connection from 128.101.39.165
The client said: Hello server!
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) %
```



csel-kh1250-05.cselabs.umn.edu - PuTTY

```
Users requiring assistance should talk to the operator on duty:
They can be reached by any of the following means:
        email: operator@cselabs.umn.edu
        or call 612-625-0876  The phones in the labs directly call operator.
        You may also visit them in Keller Hall 1-201

----------------------------------------------------------------------

Last login: Sat Oct 24 15:28:20 2015 from csel-kh4250-01.cselabs.umn.edu
+(0):ERROR:0: Unable to locate a modulefile for 'mozilla'
+(0):ERROR:0: Unable to locate a modulefile for 'soft/openoffice'
liux1366@csel-kh1250-05 (/home/liux1366) % cd 4211/measure/
liux1366@csel-kh1250-05 (/home/liux1366/4211/measure) % ./client_tcp csel-kh4250
-05.cselabs.umn.edu 9999
Connection to server csel-kh4250-05.cselabs.umn.edu established
The server said: Hello client!
liux1366@csel-kh1250-05 (/home/liux1366/4211/measure) % traceroute csel-kh4250-0
5.cselabs.umn.edu
traceroute to csel-kh4250-05.cselabs.umn.edu (128.101.37.5), 30 hops max, 60 byt
e packets
 1  eecs-gw-39.cs.umn.edu (128.101.39.254)  17.200 ms  17.419 ms  17.764 ms
 2  csel-kh4250-05.cselabs.umn.edu (128.101.37.5)  0.247 ms  0.255 ms  0.243 ms
liux1366@csel-kh1250-05 (/home/liux1366/4211/measure) %
```

Implement the message communication protocol (which is described in the last portion of the handout) on the TCP based server and client. The client should download a file from the server using the message protocol. **You must submit the code for this part of the assignment**.

d. The client should measure and report the time taken to download the file from the server.
   Repeat this step for different values of BUF_SZ (256B, 512B, 1024B & 1536B) and the three input files.
   NOTE: The time measurement should only include the time taken for recv() and should not include any other system/function calls (like time for I/O). You may use gettimeofday() to measure the time.

   Fill in the values in the table for each input file. You may consider only messages of type MSG_TYPE_RESP_GET for computation. (**10 pts**)

Filename	input_small.txt			
buffer size / parameter	**256B**	**512B**	**1024B**	**1536B**
Download time (microseconds)	2,995	12	10	8
No. of messages	1	1	1	1
Total bytes transferred	254	254	254	254

Filename	input_medium.txt			
buffer size / parameter	**256B**	**512B**	**1024B**	**1536B**
Download time (microseconds)	987,763	543,706	341,433	231,243
No. of messages	3,907	1,954	977	652
Total bytes transferred	1,000,000	1,000,000	1,000,000	1,000,000

Filename	Input_large.txt

buffer size / parameter	**256B**	**512B**	**1024B**	**1536B**
Download time (microseconds)	9,826,606	5,093,934	3,042,555	2,419,404
No. of messages	39,063	19,532	9,755	6,511
Total bytes transferred	10,000,000	10,000,000	10,000,000	10,000,000

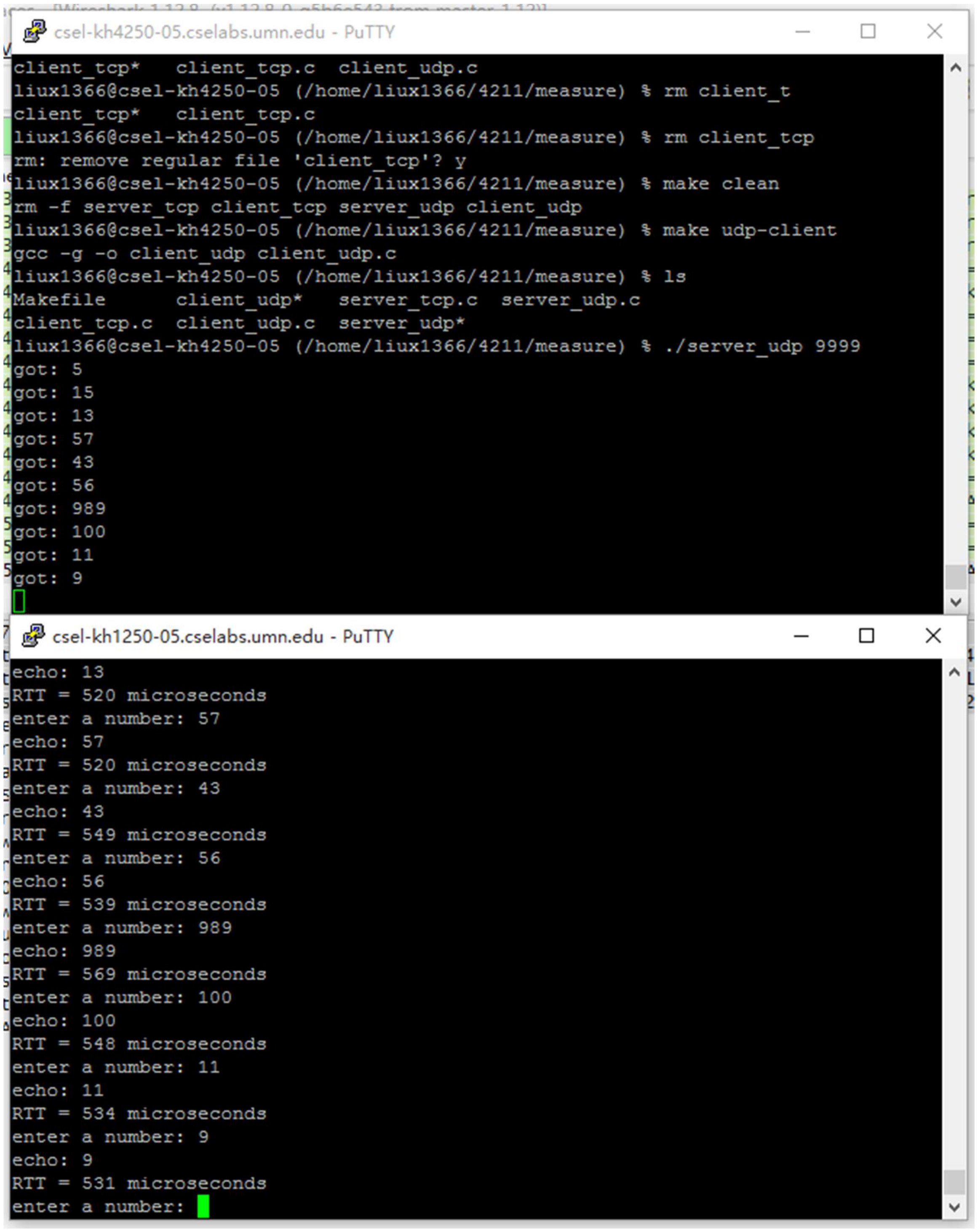
UDP performance measurement:
Build server and client applications that communicate over UDP sockets. All measurement must be performed on the client. Read and understand the questions prior to implementation.

Run the client and the server on the CSELabs UNIX machines: The server can be on one of the machines in Keller Hall 4-250 and the client can be on one of the machines in Lind Hall 40.

e. Use your own program to measure RTT (Round Trip Time) of UDP. Send 10 UDP datagrams from the client to server. The server should reply with a UDP datagram whenever it receives a datagram from the client. RTT can be obtained by computing the difference in time from when a datagram was sent out from client until the time when the corresponding reply was received by the client. Please provide the average, minimum and standard deviation of RTT and a screenshot of 10 UDP exchanges. (**5pts**)

Message (int)	**RTT (microseconds)**
5	617
15	554
13	520
57	520
43	549
56	539
989	569
100	548
11	534
9	531

Average RTT = 548.1 microseconds
Minimum RTT = 520 microseconds
Standard deviation = 27.15 microseconds


csel-kh4250-05.cselabs.umn.edu - PuTTY
csel-kh1250-05.cselabs.umn.edu - PuTTY

```
client_tcp*   client_tcp.c  client_udp.c
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % rm client_t
client_tcp*   client_tcp.c
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % rm client_tcp
rm: remove regular file 'client_tcp'? y
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % make clean
rm -f server_tcp client_tcp server_udp client_udp
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % make udp-client
gcc -g -o client_udp client_udp.c
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % ls
Makefile      client_udp*   server_tcp.c  server_udp.c
client_tcp.c  client_udp.c  server_udp*
liux1366@csel-kh4250-05 (/home/liux1366/4211/measure) % ./server_udp 9999
got: 5
got: 15
got: 13
got: 57
got: 43
got: 56
got: 989
got: 100
got: 11
got: 9
```

```
echo: 13
RTT = 520 microseconds
enter a number: 57
echo: 57
RTT = 520 microseconds
enter a number: 43
echo: 43
RTT = 549 microseconds
enter a number: 56
echo: 56
RTT = 539 microseconds
enter a number: 989
echo: 989
RTT = 569 microseconds
enter a number: 100
echo: 100
RTT = 548 microseconds
enter a number: 11
echo: 11
RTT = 534 microseconds
enter a number: 9
echo: 9
RTT = 531 microseconds
enter a number:
```

Implement the message communication protocol (which is described in the last portion of the handout) on the UDP based server and client. The client should download a file from the server using the message protocol. **You must submit the code for this part of the assignment**.

f. The client should measure and report the time taken to download the file from the server. Repeat this step for different values of BUF_SZ (256B, 512B, 1024B & 1536B) and the three input files.

NOTE: The time measurement should only include the time taken for recv() and should not include any other system/function calls (like time for I/O). You may use gettimeofday() to measure the time.

Present your results in a tabular form, like you did for part (d) above. Compare the timing results with the results you obtained for the TCP case in part (d). Explain the difference in delay. (**10 pts**)

Filename	input_small.txt			
buffer size / parameter	**256B**	**512B**	**1024B**	**1536B**
Download time (microseconds)	8	9	1,045	10
No. of messages	1	1	1	1
Total bytes transferred	254	254	254	254

Filename	input_medium.txt			
buffer size / parameter	**256B**	**512B**	**1024B**	**1536B**
Download time (microseconds)	1,003,539	537,234	295,340	241,555
No. of messages	3,907	1,954	977	652
Total bytes transferred	1,000,000	1,000,000	1,000,000	1,000,000

Filename	Input_large.txt			
buffer size / parameter	**256B**	**512B**	**1024B**	**1536B**
Download time(microseconds)	9,642,763	5,340,967	3,103,666	2,410,726
No. of messages	39,063	19,532	9,766	6,511

Total bytes transferred	10,000,000	10,000,000	10,000,000	10,000,000

Message communication protocol:

- Servers and clients using this protocol can communicate with each other by sending messages. Each message is a C structure of type `struct msg_t`. The servers and clients can only exchange messages in this format. (This means that the buffer passed in the `send()` and `recv()` system calls is always a `struct msg_t`).

```
#define BUF_SZ 1024
struct msg_t {
    enum msg_type_t msg_type;      /* message type */
    int cur_seq;                            /* current seq
number */
    int max_seq;                         /* max seq number
*/
    int payload_len;                          /* length of
payload */
    unsigned char payload[BUF_SZ]; /* buffer for data
*/
};
```

- Some details about `struct msg_t`:
  a. `msg_type`: This indicates the type of the message. Five message types have been predefined.
  b. `cur_seq, max_seq`: If a set of related messages have to be transmitted, then `cur_seq` indicates the sequence number of the current message and `max_seq` indicates the total number of messages in this sequence
  c. `payload`: This is a buffer of size `BUF_SZ` (1024B) which you can use to fill in any kind of data.
  d. `payload_len`: This will indicate the size/length of valid data in the buffer.

- Message types explained:
  a. `MSG_TYPE_GET`: Use this message to request a file for download
  b. `MSG_TYPE_GET_ERR`: Use this message to indicate errors in obtaining the file (if any)
  c. `MSG_TYPE_GET_RESP`: Use this message to send the file across the network

   d. `MSG_TYPE_GET_ACK`: Use this message to acknowledge a `MSG_TYPE_GET_RESP` message.
   e. `MSG_TYPE_FINISH`: Use this message to indicate end of session.

Requirements:

1. The TCP server executable must be named `server_tcp and` the UDP server executable must be named `server_udp`.
2. The TCP client executable must be named `client_tcp` and the UDP client executable must be named `client_udp`.
3. Provide two Makefiles, one in the server directory and one in the client directory, which can build these executables.
4. Please note that we will try to download files of different sizes while grading - it is your job to ensure that files of any size can be downloaded. You may test your programs against the provided input files (input_small.txt, input_medium.txt, input_large.txt)
5. The submission must include a README file which clearly contains:
   a. Name of the student, student ID and x500
   b. A brief description of how files are downloaded.

Server requirements:

The server program will be executed as follows:

```
$ ./server_tcp <port>
$ ./server_udp <port>
```

1. The server must listen for clients on a socket bound to the specified port.
2. Print the following message on the screen whenever a message is received:
   `server: RX <msg_type> <cur_seq> <max_seq> <payload_len>`
3. The server must send a file to the client when the client requests it.
4. If the requested file is not found in the current working directory, then the server must respond to the client with an appropriate error message
   (hint: message type - `MSG_TYPE_GET_ERR`)

Client requirements:

The client program will be executed as follows:

```
$ ./client_tcp <server-ip> <port> <filename>
$ ./client_udp <server-ip> <port> <filename>
```

1. The client must first connect to the specified server on the specified port.
2. The client must then attempt to download the specified file from the server and save it in the current working directory.
3. File integrity must be preserved when downloading files. This means that downloaded file must exactly match the file on the server. (Hint: Use the diff utility to compare the two files). You will lose significant points if the downloaded file differs from the file on the server.

4. Print the following message on the screen whenever a message is received:
   `client: RX <msg_type> <cur_seq> <max_seq> <payload_len>`

Sample server:
A sample TCP server application has been provided. The server will allow a client to connect and download a file. You may test your client with this server initially. However, you are required to develop and submit your own servers and clients for this assignment. The sample server works as follows:

1. Start the TCP server
2. Start the TCP client & connect to the server.
3. Send a `MSG_TYPE_GET` from the client to the server, with the name of the file to be download in the payload.
4. If the server cannot find the file, it will respond with a `MSG_TYPE_GET_ERR` message. If the file is found, the server will break the file into chunks and transmit each chunk in a `MSG_TYPE_GET_RESP` message. Depending on the file size, there will be multiple such messages. The actual file contents will be stored in the payload in each message.
5. The client must send a `MSG_TYPE_GET_ACK` message for each `MSG_TYPE_GET_RESP` it receives - or else the server will not respond with the next message.

You can use the same mechanism or a different mechanism to download the files. The README file should clearly explain the mechanism you use to download the file.

Execution environment:

1. Please ensure that your code compiles and executes on the CSELabs UNIX machines. You will lose significant points if your code cannot be compiled/executed on these machines.
2. While developing/implementing your solution, the server and client can run on the same machine - You can use the IP address as localhost or 127.0.0.1.
3. When it comes to actual measurement, the server and client should run on different machines, ideally multiple hops away. Our suggestion is to use two CSE Lab machines as described below:
   - Run the server on one of the machines in Keller Hall 4-250.
   - Run the client on one of the machines in Lind Hall 40.
   - Please use only port numbers between **9000** and **10000**. (These ports are currently allowed by the system staff).
   - You can use '**ip addr show**' or '**ipconfig**' to get the IP address of the machine on which the server is running.

   If your code does not execute in this scenario, you will lose significant points on your submission.

Deliverables:

1. You must upload a single archive file (.zip or .tar or .tar.gz) on moodle.
   When extracted, the archive file must be a single folder. The name of the folder should be your **student ID**. The folder should contain the following files:
   - Readme
   - server source files & Makefile
   - client source files & Makefile
   - message.h
   - MS Word/PDF document
2. DO NOT include the test files (input_small.txt, input_medium.txt & input_large.txt)
3. DO NOT include any executable files - we will build the executables using your Makefiles.

For example, here is a sample submission:

```
1234567/
    Readme
    PA2.doc
    message.h
    server/
        server_tcp.c
       server_udp.c
       Makefile
    client/
        client_tcp.c
       client_udp.c
        Makefile
```

You can create an archive file from the contents of the above directory as follows:

```
$ tar cvf 1234567.tar.gz 1234567/
```

Grading:
README, Makefiles, comments, readability: **4 points**
Packaging the submission as specified: **3 points**
Logging messages on the screen in the correct format: **1 points**
File download: **2 points**