CSCI4211: Introduction to Computer Networks

Homework Assignment II

Due 11:55 PM October 21st, 2015

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

Name: Sihan Chen   Student ID: 4452332

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 6^{th} 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

Logical communication means that from an application's perspective, it is as if the hosts running the processes were directly connected; in reality, the hosts may be on opposite sides of the planet, connected via numerous routers and a wide range of link types. Application processes use the logical communication provided by the transport layer to send messages to each other, free from the worry of the details of the physical infrastructure used to carry these messages.

- Multiplexing and Demultiplexing

The job of delivering the data in a transport-layer segment to the correct socket is called demultiplexing. The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called multiplexing. And they are concerns whenever a single protocol at one layer (at the transport layer or elsewhere) is used by multiple protocols at the next higher layer.

- Little Endian

Big-endian and little-endian are terms that describe the order in which a sequence of bytes are stored in computer memory. Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address). Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first.

- Handshaking

Handshaking is the cliente and server exchange transport-layer control information with each other before the application-layer messages begin to flow. The handshaking procedure alerts the cliente and server, allowing them to prepare for an onslaught of packets.

- Flow Control

Flow control is a speed-matching service- matching the rate at which the sender is sending against the rate at which the receiving application is Reading.

- Congestion Control

Congestion control is concerned with allocating the resources in a network such that the network can operate at an acceptable performance level when the demand exceeds or is near the capacity of the network resources. These resources include bandwidths of links, buffer space (memory), and processing capacity at intermediate nodes.

- Go-Back-N algorithm

In Go-Back-N algorithm, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline. And if the receiver find out the next received sequence number is wrong, it will discard all the packets after the expected sequence within the window size.

- Selective Repeat algorithm

Like Go-Back-N algorithm, the sender and receiver also keep a window size of N, Also ,as the name suggests, unlike Go-Back-N, selective-repeat protocols avoid unnecessary retransmissions by having the sender retransmit only those packets that it suspects were received in error (that is, were lost or corrupted) at the receiver.

- Fairness

Fairness measures or metrics are used in network engineering to determine whether users or applications are receiving a fair share of system resources. A congestion control mechanism is said to be fair if the average transmission rate of each connection is approximately R/K; that is, each connection gets an equal share of the link bandwidth.

- Slow Start

Slow-start is part of the congestion control strategy used by TCP, the data transmission protocol used by many Internet applications. Slow-start is used in conjunction with other algorithms to avoid sending more data than the network is capable of transmitting, that is, to avoid causing network congestion.

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**)

For the Stop-and-wait, the maximum possible rate of transmission is

$$MTR = \frac{8\ Kbits}{0.05s + \frac{8Kbits}{8Mbps}} = 156862.7451\ bps$$

$0.05s + \frac{8Kbits}{8Mbps}$ is the total time that we need, to transmit a packet. And because of stop-and-wait mechanism, we have can only transmit on packet on this time interval, so MTR is calculated as the equation shown above. And the maximum transmission rate for both GBN and SR are both 8Mbps, since if the window size is large enough then sender can continuously send packets without waiting for the ACK from a receiver. So the maximum possible rate is 8Mbps.

(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**)

Since the window will slide when it receive an ACK, the minimum window size in terms of the number of packets will be the time client takes to receive an ACK divide by the time it take to transmit a single packet. So

$$MSW = \frac{0.05s + \frac{8Kbits}{8Mbps}}{\frac{8Kbits}{8Mbps}} = 51$$

(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 the stop-and-wait, only the packet with sequence number 16 will be retransmitted. For GBN, any packets with a sequence number larger than 16 or equal to 16 and within the window will be retransmitted. For the SR, it is the same as stop-and-wait, only the packet with sequence number 16 will be retransmitted.

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**)

The sequence number is 188; The source port number is 303 and the destination port number is 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**)

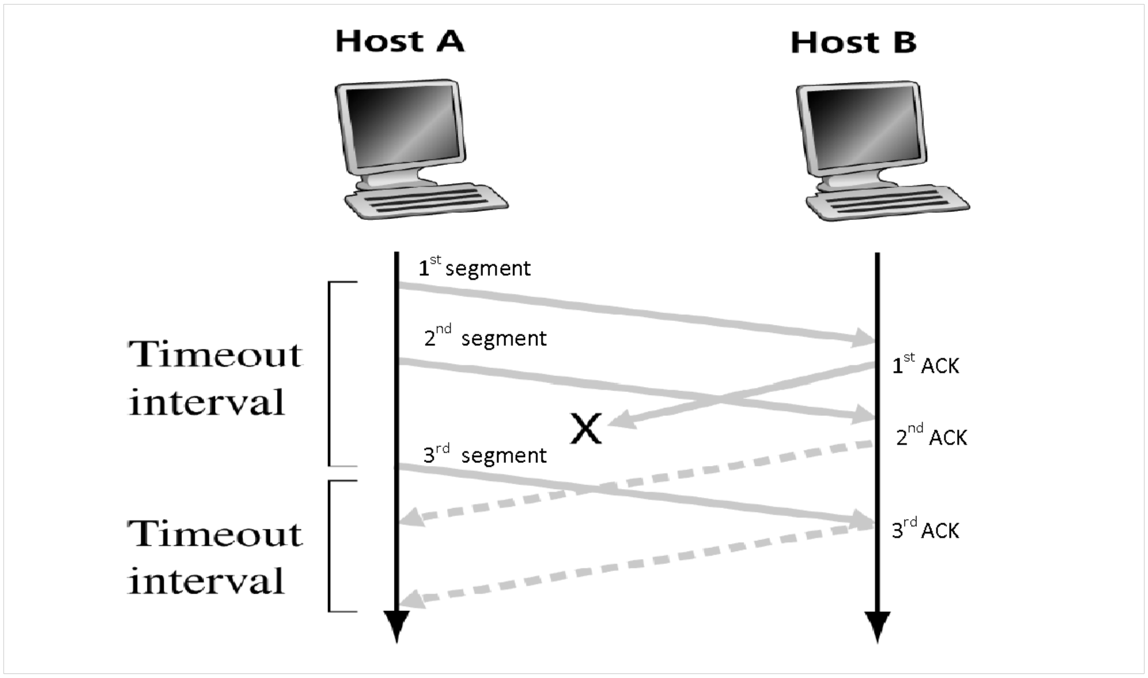
The acknowledgement number is 188; The source port number is 80, and the destination port number is 303

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**)

The acknowledgement number is 169

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**)

The sequence number for the third data segment will be 169, since the first acknowledgement is lost. The acknowledgement for the 2nd acknowledgement is 229 since both segment 1 and segment 2 are received successfully. The acknowledgement number for the 3rd acknowledgement is also 229, since third data segment is a retransmitted data segment.



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)

There are 4 fields in the UDP header, which are source port, destination port, length and checksum.

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

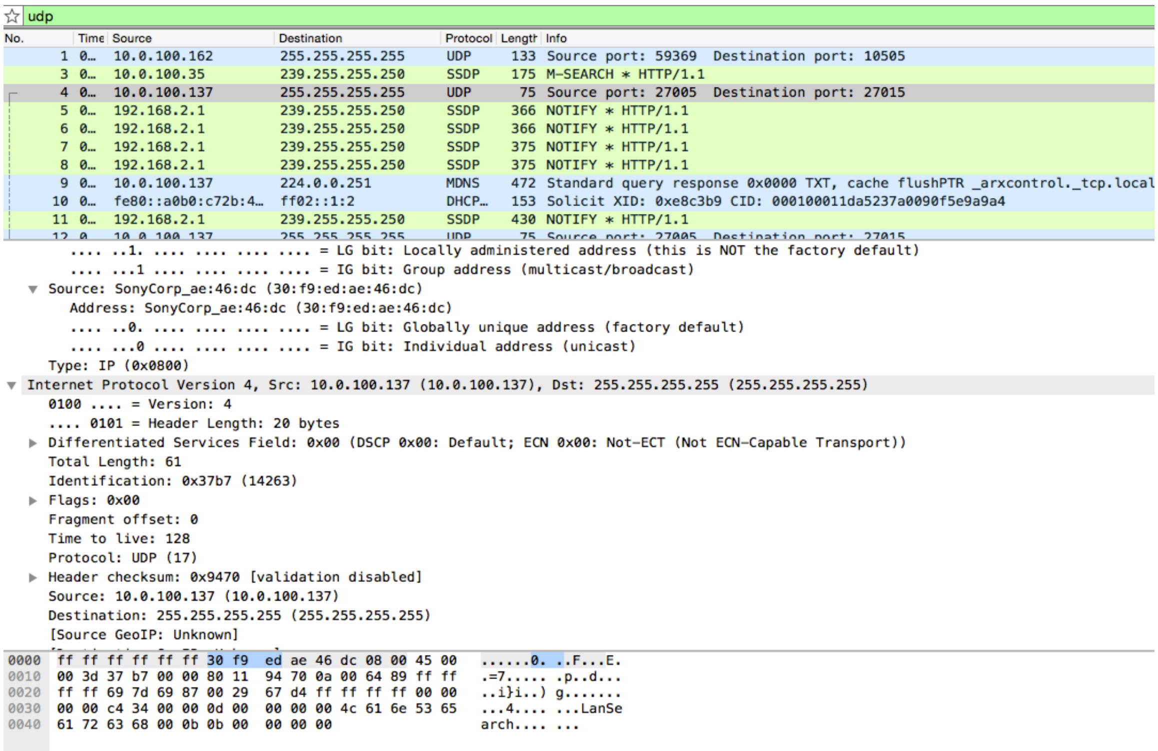
Since the maximum UDP message size is 65535 bytes, and head file will take 8 bytes. So the maximum number of bytes that can be included in a UDP payload is

$$65535\ bytes - 8\ bytes = 65527\ bytes$$

(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)

The protocol number for UDP is 17 in decimal and 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.



First, add up source and destination IP address.
Source IP address: 0x0a00 + 0x 6489 = 0x6e89
Destination IP address: 0xffff + 0xffff = 0x1fffe
Protocol number: 0x11
UDP length: 0x29
UDP header except checksum:
0x697d + 0x6987 + 0x0029
Then, Data,
0xffff + 0xffff + 0xc434 + 0x 0d00 + 0x004c + 0x616e + 0x5365+ 0x6172 + 0x6368 + 0x000b + 0x0b00.

Adding all together: 79824
Then split into 2 16 bits.
0x7 + 0x9824 = 0x982b
Last, do the one's complement, -0x982b = 0x67d4. Which is the checksum. So verified.

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

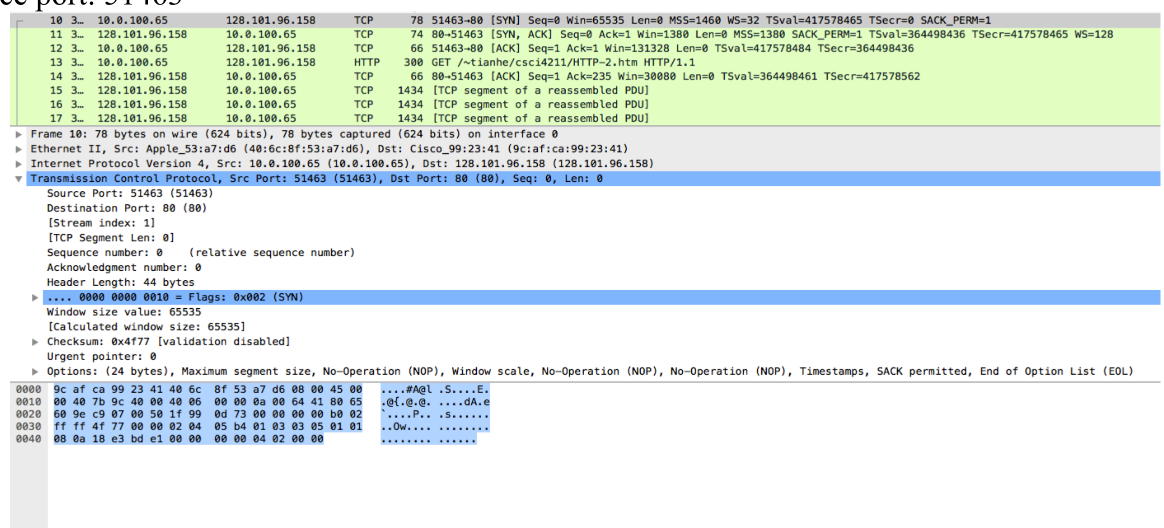
First, filter the packets displayed in the Wireshark window by entering "tcp" into the display filter window towards the top of the Wireshark window. What you should see is series of TCP and HTTP messages between your computer and

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

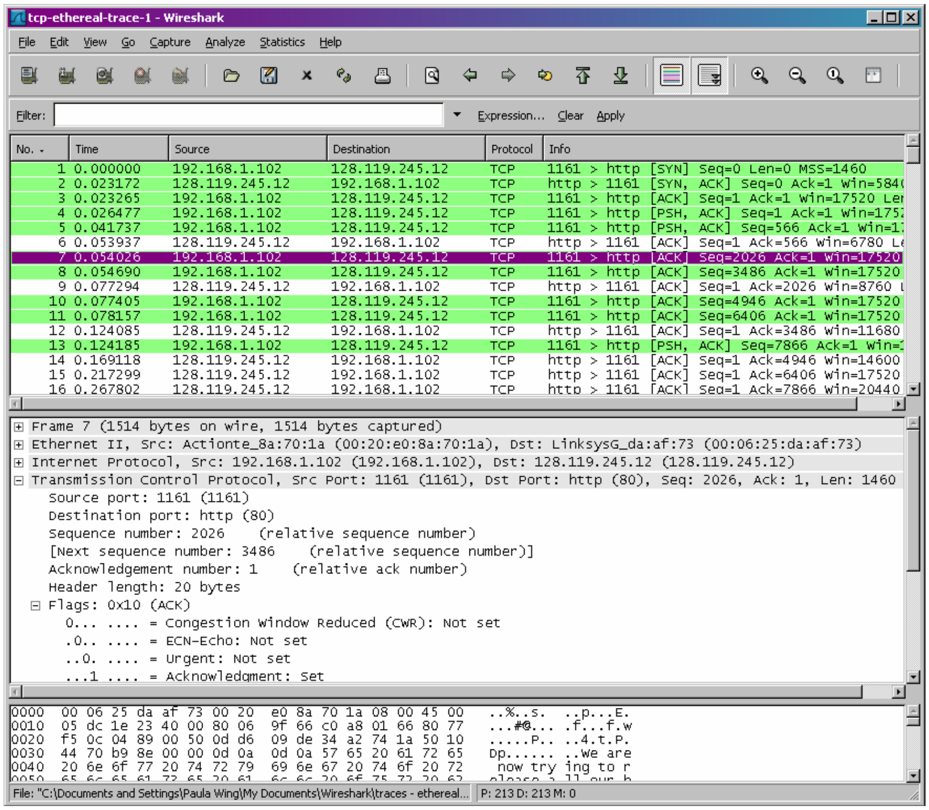
(a) What is the IP address and TCP port number used by the client computer (source) that downloads the bill of rights from www-users.cselabs.umn.edu? You need to paste an appropriate screenshot as the evidence. (2pts)

IP address: 10.0.100.65

source port: 51463



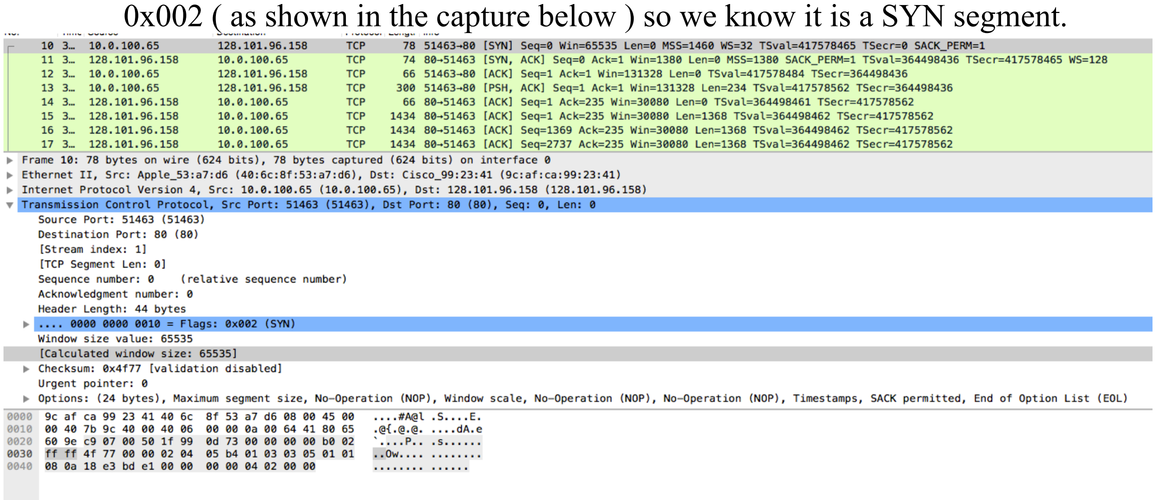
Since this lab is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select *Analyze->Enabled Protocols*. Then uncheck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



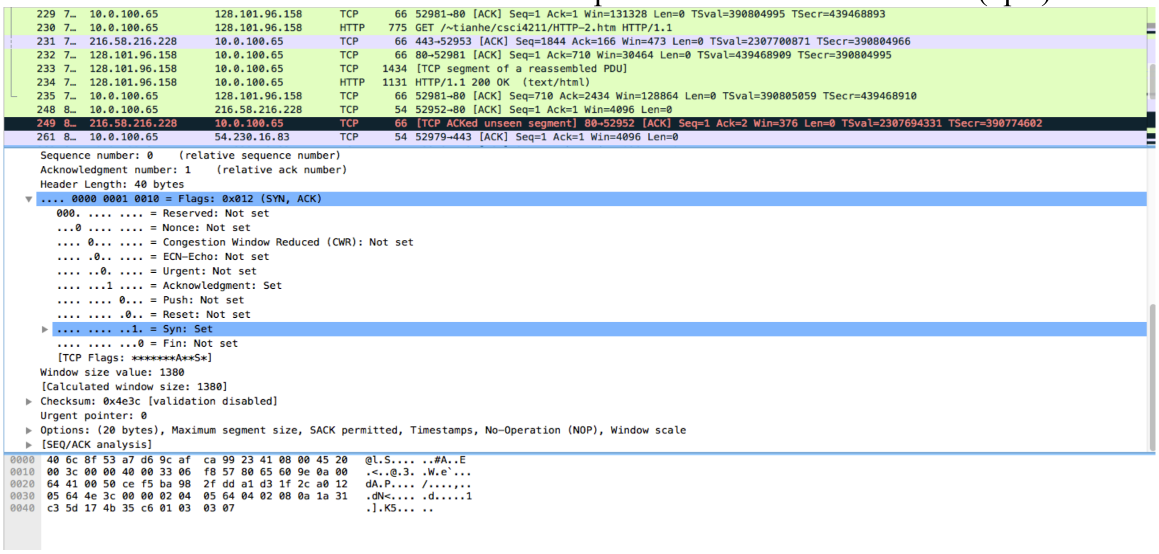
Please answer the following question (along with screenshots as the back-up evidence of your answer, if applicable)

(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)

The sequence number of the TCP SYN segment is 0. The flag was set to 0x002 (as shown in the capture below) so we know it is a SYN segment.



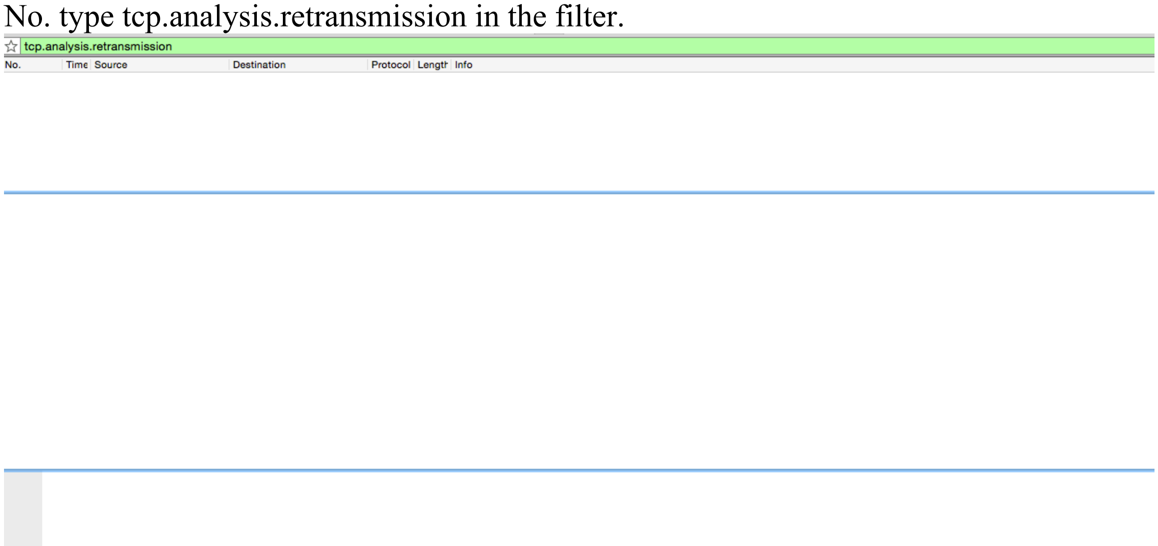
(c) What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender? (2pts)



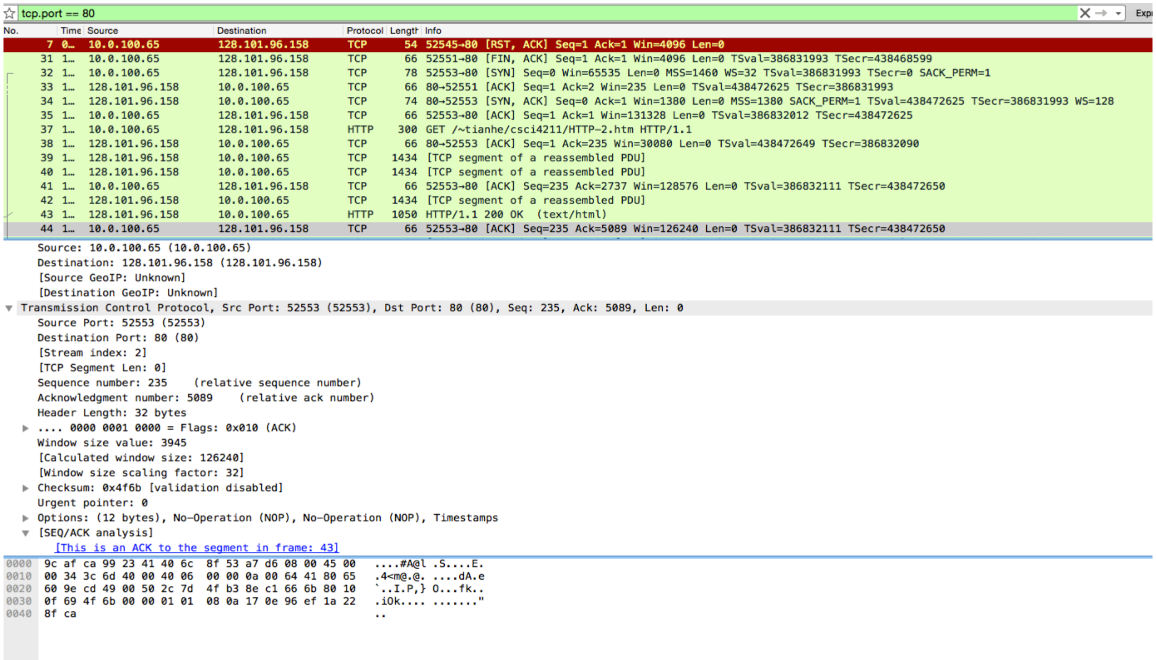
The minimum amount of available buffer space advertised at the received for the entire trace is 1380 bytes. And It will not throttle the sender.

(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.

No. type tcp.analysis.retransmission in the filter.



(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)



From the graph above, I know 5088 bytes have been transferred through this tcp connection. And the number of total bytes downloaded through this TCP connection does not equal to the html file size of the bill of rights. This is because before transmission, data were compressed.

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.

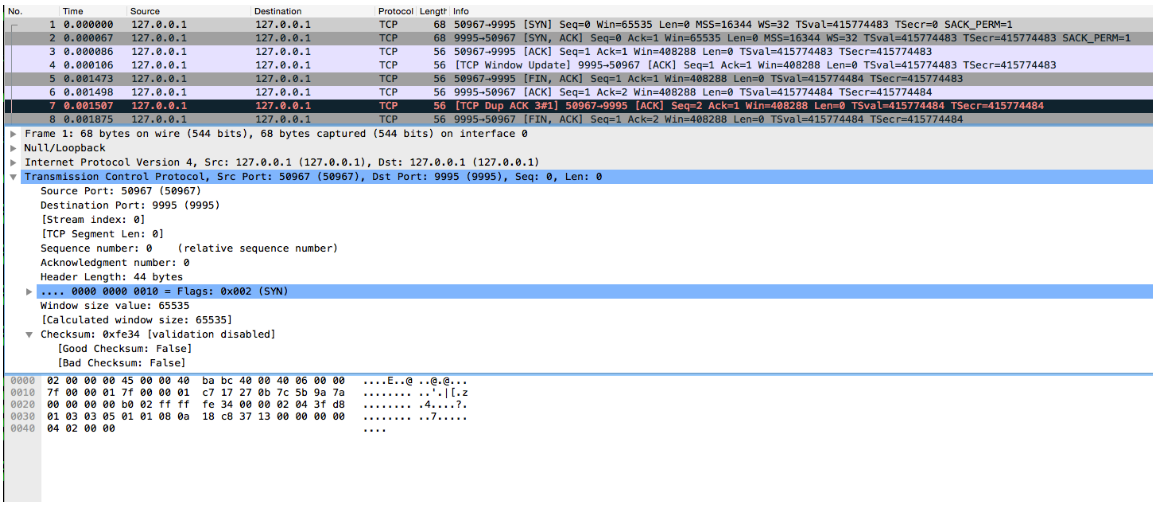
Run both the client and the server on your home LAN for parts (a) & (b).

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**)

1. 0.000086s
2. 0.000082s
3. 0.000093s
4. 0.000091s
5. 0.000081s

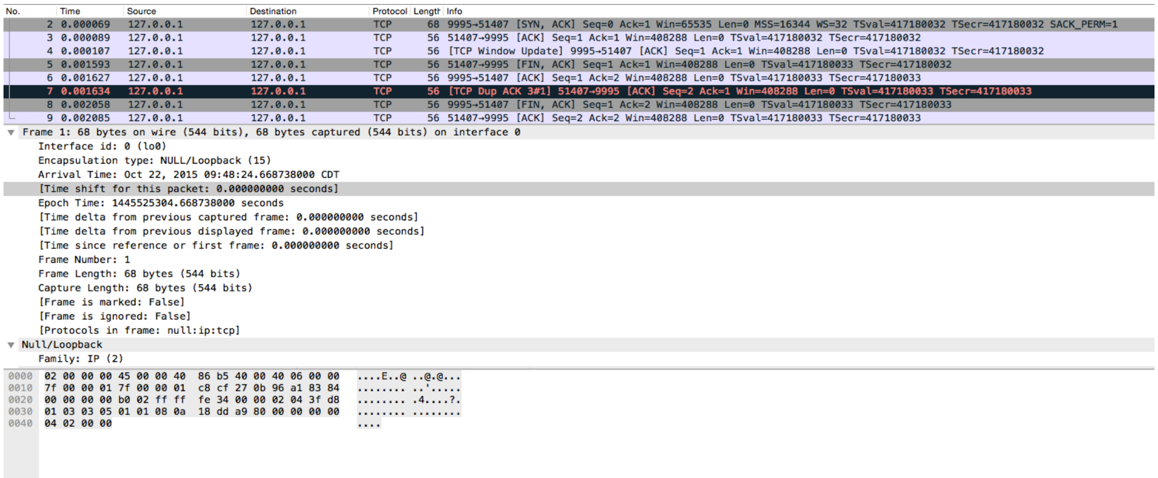
Average: 0.0000866 s



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**)

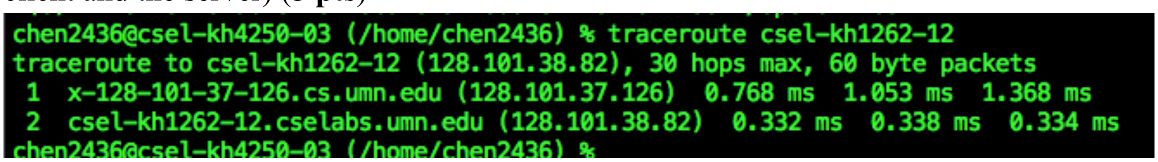
1. 0.000492s
2. 0.000503s
3. 0.000487s
4. 0.000477s
5. 0.000496s

Average Time: 0.000491 s



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**)



Path (hostname)	IP address
csel-kh4250-03	128.101.37.3
x-128-101-37-126.cs.umn.edu	128.101.37.126
csel.kh1262-12.cselabs.umn.edu	128.101.38.82

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	5×10^{-6} s	8×10^{-6} s	5×10^{-6} s	5×10^{-6} s
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	0.91459 s	0.48853 s	0.276172 s	0.204071 s
No. of messages	3907	1954	977	652
Total bytes transferred	1000000	1000000	1000000	1000000

Filename	input_large.txt			
buffer size / parameter	**256B**	**512B**	**1024B**	**1536B**
Download time	8.874445 s	4.976674 s	2.576187 s	1.966866 s
No. of messages	39063	19532	9766	6511
Total bytes transferred	10000000	10000000	10000000	10000000

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**)

```
chen2436@csel-kh4250-03 (/home/chen2436/csci4211/client) %
client: RX get_ack 1 10 0
client: RX get_ack 2 10 0
client: RX get_ack 3 10 0
client: RX get_ack 4 10 0
client: RX get_ack 5 10 0
client: RX get_ack 6 10 0
client: RX get_ack 7 10 0
client: RX get_ack 8 10 0
client: RX get_ack 9 10 0
client: RX get_ack 10 10 0
The average of RRTS is: 351.100000
The minimum of RRTS is: 229.000000
The standard deviation of RRTS is: 78.083865
chen2436@csel-kh4250-03 (/home/chen2436/csci4211/client) %
```

The average, minimum and standard deviation of RRTs shown above are all in microsecond. Code for this part is commented out in the code for server_udp and client_udp.

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	3×10^{-6} s	3×10^{-6} s	4×10^{-6} s	3×10^{-6} s
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	0.90943 s	0.479722 s	0.262709 s	0.192632 s
No. of messages	3907	1954	977	652
Total bytes transferred	1000000	1000000	1000000	1000000

Filename	input_large.txt			
buffer size / parameter	**256B**	**512B**	**1024B**	**1536B**
Download time	8.690199 s	4.763688 s	2.5999985 s	1.989571 s
No. of messages	39063	19532	9766	6511
Total bytes transferred	10000000	10000000	10000000	10000000

By comparing the results from UDP and TCP, UDP have smaller delay than TCP. This is because UDP do not have three-time handshake.

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**