Wesleyan University, Spring 2023, COMP 332

Homework 6: TCP, UDP, and IP addressing

Due by 11:59pm on April 12, 2023

---

1. Written problems (10 points)

Problem 1. *Hosts A and B are communicating over a TCP connection. Host B has already received from A all bytes up through byte 248. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 40 and 60 bytes of data respectively. In the first segment, the sequence number is 249, the source port number is 503, 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 Host B, what are the sequence number, source port number, and destination port number?*

b: *Suppose the first segment arrives before the second segment. Host B sends an acknowledgement in response to the first segment: what is the acknowledgment number, the source port number, and the destination port number?*

c: *Suppose the second segment arrives before the first segment. Host B sends an acknowledgement in response to the first segment. What is the acknowledgement number?*

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. Draw a timing diagram, showing these segments and all other segments and acknowledgements sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data. For each acknowledgement that you add, provide the acknowledgement number.*

Solution:

a: The sequence number is 289, the source port number is 503 and the destination port number is 80.

b: The acknowledgement number is 289, the source port number is 80 and the destination port number is 503.

c: The acknowledgement number is 249, indicating that Host B is still waiting for bytes 249 and onwards.

d: See Figure 1.

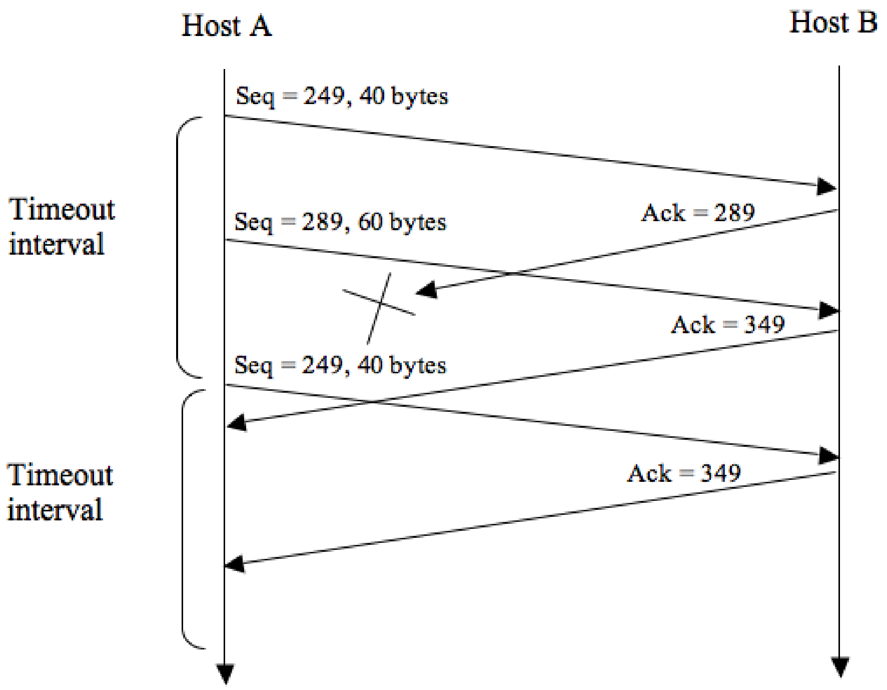


FIGURE 1. Solution to question 1(d).

PROBLEM 2. *Figure 2 plots the TCP window size over time for TCP reno. Using Figure 2, answer the following questions.*

a: *Identify the transmission rounds when TCP slow start is operating.*

b: *Identify the transmission rounds when TCP congestion avoidance is operating.*

c: *Identify the transmission rounds when TCP fast retransmit is operating.*

d: *After round 15, is segment loss detected by a triple duplicate ACK or a timeout?*

e: *After round 27, is segment loss detected by a triple duplicate ACK or a timeout?*

f: *What is the value of ssthresh at the first transmission round?*

g: *What is the value of ssthresh at the 16th transmission round?*

h: *What is the value of ssthresh at the 19th transmission round?*

i: *What is the value of ssthresh at the 28th transmission round?*

j: *During what transmission round is the 61st segment transmitted?*

Solution:

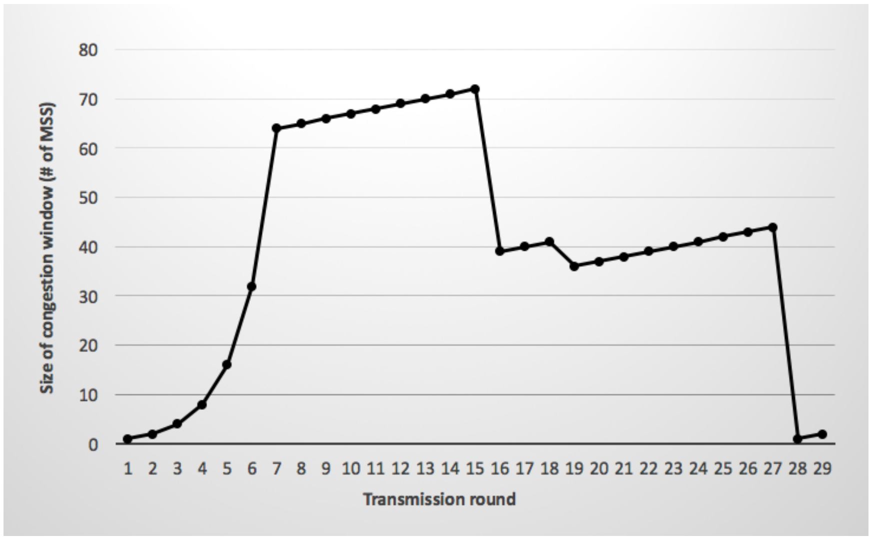


Figure 2. TCP congestion window size over time for TCP Reno.

a: Rounds 1 through 7, Rounds 28 through 29.

b: Rounds 8 through 15, Rounds 19 through 27.

c: Rounds 16 through 18.

d: Triple duplicate ack.

e: Timeout.

f: ssthresh = 64.

g: ssthresh = 36.

h: ssthresh = 36.

i: ssthresh = 44/2 = 22.

j: Sent during round 6.

```
Round 1: Segment 1 sent.
Round 2: Segments 2, 3 sent.
Round 3: Segments 4, 5, 6, 7  sent.
Round 4: Segments 8, 9, 10, 11, 12, 13, 14, 15 sent.
Round 5: Segments 16 through 31 sent.
Round 6: Segments 32 through 63 sent.
```

Problem 3. *Consider a router that interconnects three subnets: Subnet 1, Subnet 2, and Subnet 3. Suppose all of the interfaces in each of these three subnets are required to have the prefix*

223.1.17/24. Also suppose that Subnet 1 is required to support up to 70 interfaces, and Subnet 2 and 3 are each required to support up to 60 interfaces. Provide three network addresses (of the form a.b.c.d/x) that satisfy these constraints.

Solution:

```
223.1.17.0/25
223.1.17.128/26
223.1.17.192/26
```

PROBLEM 4. *Open wireshark. While recording traffic, open* `www.nytimes.com`*. Once the webpage has loaded, stop recording traffic and enter the filter* `ip.addr == 151.101.117.164`*, which is the IP address for* `www.nytimes.com`*.*

a: *Choose a packet sourced from* `www.nytimes.com` *that contains actual data sent by the nytimes server. Take a screenshot of the packet, with the IP header expanded and making sure to include the Transmission Control Protocol line.*

b: *How many bytes are in the IP header for the packet in part (a)? How many bytes are in IP packet? List all of the fields you see in the IP header and the number of bytes used by each field.*

c: *How many bytes are in the TCP header for the TCP segment contained in the packet in part (a)? How many bytes are in the TCP segment?*

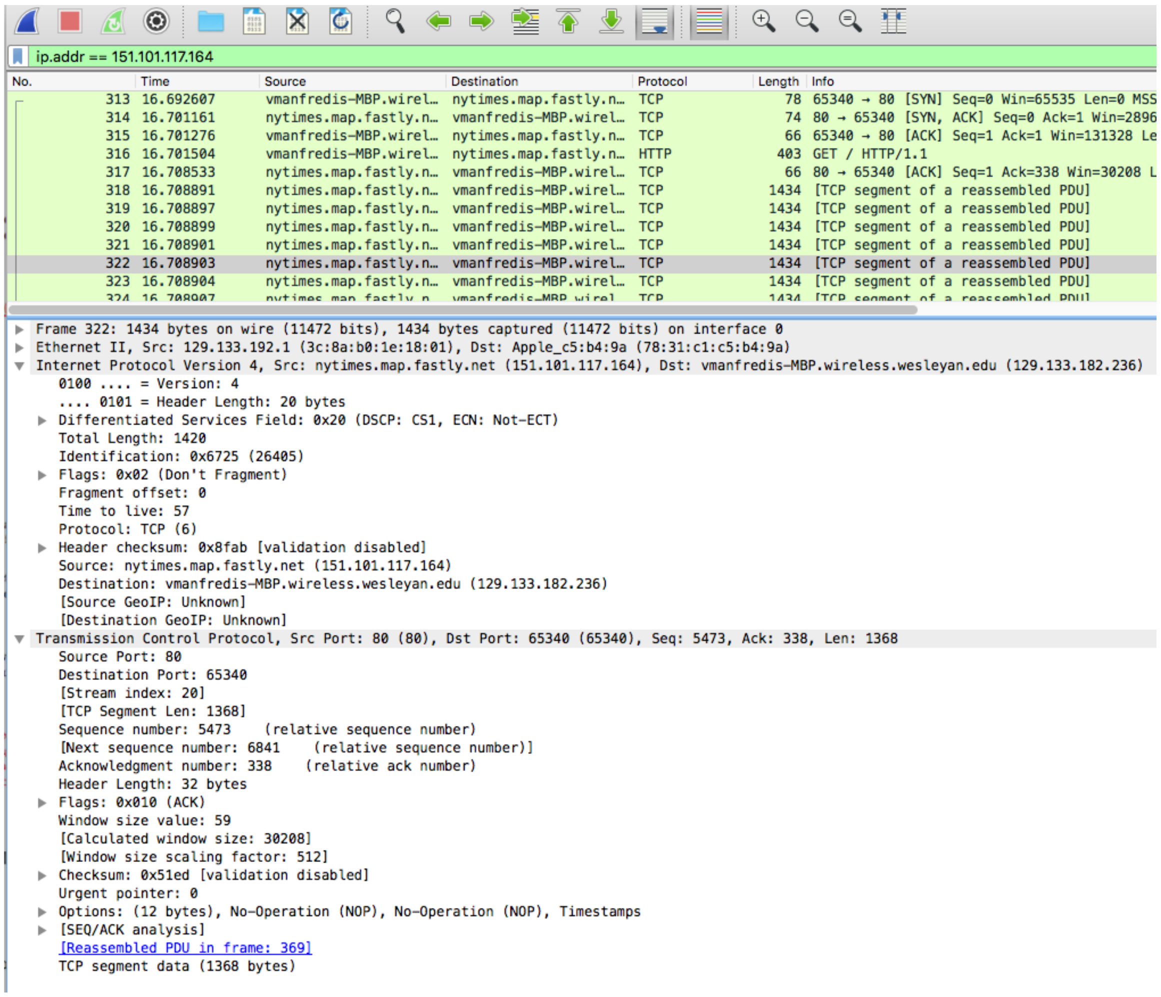
d: *For the traffic you have recorded, which IP header fields change from one packet to the next?*

Solution:

a: Packet is shown in Figure 3.

b: The IP header length is 20 bytes and the IP packet length is 1420 bytes. The fields in the IP header are as follows.

```
Version (4 bits = 0.5 bytes).
Header length (4 bits = 0.5 bytes).
Differentiated services (8 bits = 1 byte).
Total length (16 bits = 2 bytes).
Identification (16 bits = 2 bytes)
Flags (3 bits).
Fragment offset (13 bits).
Time-to-live (8 bits = 1 byte).
Protocol (8 bits = 1 byte).
Header checksum (16 bits = 2 bytes).
Source IP address (32 bits = 4 bytes).
Destination IP address (32 bits = 4 bytes).
```



ip.addr == 151.101.117.164

```
No.         Time        Source                 Destination            Protocol   Length  Info
        313 16.692607   vmanfredis-MBP.wirel…  nytimes.map.fastly.n…  TCP            78  65340 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS
        314 16.701161   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP            74  80 → 65340 [SYN, ACK] Seq=0 Ack=1 Win=2896
        315 16.701276   vmanfredis-MBP.wirel…  nytimes.map.fastly.n…  TCP            66  65340 → 80 [ACK] Seq=1 Ack=1 Win=131328 Le
        316 16.701504   vmanfredis-MBP.wirel…  nytimes.map.fastly.n…  HTTP          403  GET / HTTP/1.1
        317 16.708533   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP            66  80 → 65340 [ACK] Seq=1 Ack=338 Win=30208 L
        318 16.708891   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP          1434  [TCP segment of a reassembled PDU]
        319 16.708897   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP          1434  [TCP segment of a reassembled PDU]
        320 16.708899   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP          1434  [TCP segment of a reassembled PDU]
        321 16.708901   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP          1434  [TCP segment of a reassembled PDU]
        322 16.708903   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP          1434  [TCP segment of a reassembled PDU]
        323 16.708904   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP          1434  [TCP segment of a reassembled PDU]
        324 16.708907   nytimes.map.fastly.n…  vmanfredis-MBP.wirel…  TCP          1434  [TCP segment of a reassembled PDU]
```

```
▶ Frame 322: 1434 bytes on wire (11472 bits), 1434 bytes captured (11472 bits) on interface 0
▶ Ethernet II, Src: 129.133.192.1 (3c:8a:b0:1e:18:01), Dst: Apple_c5:b4:9a (78:31:c1:c5:b4:9a)
▼ Internet Protocol Version 4, Src: nytimes.map.fastly.net (151.101.117.164), Dst: vmanfredis-MBP.wireless.wesleyan.edu (129.133.182.236)
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes
   ▶ Differentiated Services Field: 0x20 (DSCP: CS1, ECN: Not-ECT)
     Total Length: 1420
     Identification: 0x6725 (26405)
   ▶ Flags: 0x02 (Don't Fragment)
     Fragment offset: 0
     Time to live: 57
     Protocol: TCP (6)
   ▶ Header checksum: 0x8fab [validation disabled]
     Source: nytimes.map.fastly.net (151.101.117.164)
     Destination: vmanfredis-MBP.wireless.wesleyan.edu (129.133.182.236)
     [Source GeoIP: Unknown]
     [Destination GeoIP: Unknown]
▼ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 65340 (65340), Seq: 5473, Ack: 338, Len: 1368
     Source Port: 80
     Destination Port: 65340
     [Stream index: 20]
     [TCP Segment Len: 1368]
     Sequence number: 5473    (relative sequence number)
     [Next sequence number: 6841    (relative sequence number)]
     Acknowledgment number: 338    (relative ack number)
     Header Length: 32 bytes
   ▶ Flags: 0x010 (ACK)
     Window size value: 59
     [Calculated window size: 30208]
     [Window size scaling factor: 512]
   ▶ Checksum: 0x51ed [validation disabled]
     Urgent pointer: 0
   ▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
   ▶ [SEQ/ACK analysis]
     [Reassembled PDU in frame: 369]
     TCP segment data (1368 bytes)
```

FIGURE 3. Packet from nytimes.

c: TCP header length is 32 bytes. TCP segment length is 32 + 1368 = 1400 bytes. The TCP segment length plus the IP header of 20 bytes should give the IP length of 1420 bytes.

d: Total length, Identification, TTL, Header checksum. If you have packets from different flows, you'll also see Source IP address and Destination IP address changing.

2. Coding and hands-on problems (10 points)

You have two options for coding on this homework: you should choose one of the options. Option 1 is an individual project implementing a UDP ping application. Option 2 is a group project for two people updating your chat application from the last homework to run over UDP. If you choose Option 2, please email me to let me know, and also let me know with whom you are working. More details below.

```
>python3 udp_client.py localhost 55555
['udp_client.py', 'localhost', '55555'] 3
Reply from 127.0.0.1: PING 0 WED APR  4 14:01:01 2018
RTT: 0.009649038314819336
Reply from 127.0.0.1: PING 1 WED APR  4 14:01:01 2018
RTT: 0.001322031021118164
Reply from 127.0.0.1: PING 2 WED APR  4 14:01:01 2018
RTT: 0.0014090538024902344
Request timed out.
Request timed out.
Request timed out.
Reply from 127.0.0.1: PING 6 WED APR  4 14:01:13 2018
RTT: 0.002029895782470703
Reply from 127.0.0.1: PING 7 WED APR  4 14:01:13 2018
RTT: 0.001703023910522461
Reply from 127.0.0.1: PING 8 WED APR  4 14:01:13 2018
RTT: 0.0016598701477050781
Request timed out.
```

FIGURE 4. Example output for UDP ping client.

Option 1: Individual project implementing a UDP ping application. In Option 1 you will create a UDP ping application. The ping server will create a UDP socket, bind it to port 50007, and then wait for client messages. The ping client will open a UDP socket and send a short message to the server that the server echoes back to the client. The ping client will record the time from when its message is generated until when it receives a message back, and will print this information out. The ping client will send 10 of these messages. Example client output is shown in Figure 4. There is no code distribution for this homework, however, please split your code into 2 files that you will submit: `udp_client.py` and `udp_server.py`. Because the channel between 2 processes on the same device isn't very noisy, to experiment with loss, you should add some code to the server that randomly does not respond to ping messages that it receives (i.e., "drops" the message).

You may wonder why we don't try the ping client out on a real website: I encourage you to do this and you'll soon realize that there is no process waiting for UDP connections on most servers on any ports. Instead, as we saw in class, in practice, ping and traceroute rely on the generation of ICMP packets in response to the query that was generated (whether the query was generated over UDP or TCP). However, because ICMP (and IP) headers belong to the network layer, there is no socket for these packets to go to: if we want to look at or generate IP or ICMP packets we must use what is called a raw socket. Homework 7 will use raw sockets to do exactly this.

Notes:

- For more on sockets, look here `https://docs.python.org/3/library/socket.html`. Recall that a UDP socket is of type SOCK_DGRAM.
- If you use the chrome browser, you may need to turn off the use of QUIC: to do so you will need to go to `chrome://flags/` in your browser. Otherwise you may see QUIC traffic instead of UDP traffic. QUIC is a transport layer protocol developed by Google.

- The command line tool, `nmap` is very powerful, and one of its capabilities is to send a UDP ping. Eg., try out the command `sudo nmap -P0 -sU -p55555 www.wesleyan.edu`. To find out more about `nmap`, type `man nmap` in a terminal.

Option 2: group project for two people updating your chat application from the last homework to run over UDP. In Option 2, you will update your chat application from homework 5 to run over UDP rather than TCP. This will require some thought to redesign the code. Because the channel between 2 processes on the same device isn't very noisy, to experiment with loss, you should add some code to the server that randomly does not respond to ping messages that it receives (i.e., "drops" the message): you should implement a simple timeout and retransmit on the client side to recover from these drops. You and your partner should test out your code with two clients running on different machines: to do this, you may need to check that the ports being used on each machine are accessible to external connecitons.

3. Submission

Submit your written work as `hw6.pdf` and your `*.py` files to the Google Drive directory I have created for you named `comp332-s23-USERNAME/hw6/`. You should replace `USERNAME` with your Wesleyan username.

Do not forget that your written work must be submitted as a PDF! And make sure that at the top of each file you have put your name! Do not, however, change the names of the files.

You should *not* submit `hw4.h` or any test or driver programs. When we test your code, we will add in our copy of `hw4.h` and our own testing program. In particular, if you change `hw4.h` in order to make your code compile, then your code will probably fail to compile with our `hw4.h`, and hence you will receive little to no credit for the coding portion of this assignment.