Peyman Shobeiri

Computer networks- Assignment 1

McGill university

ID: 261286438

Winter 2025

Table of Contents

Part 1: ***4***

Problem 1 **4**
(a) 4
(b) 4
(c) 4
(d) 5

Problem 2: **5**

Problem 3 **6**
(a) 6
(b) 6

Problem 4 **7**
(a) 7
(b) 8

Problem 5 **8**

Problem 6: **9**

Problem 7: **9**
(a) 10
(b) 10
(c) 10
(d) 10
(e) 11

Problem 8 (old version): **11**

Problem 8 (new version): **12**

Problem 9 **14**

Part 2: ***15***

1. **15**

2. **16**

3. **16**

4. **17**

5. **17**

6. **18**

7. **19**

8. **19**

9. **20**

References: ***22***

Part 1:

Problem 1: **Suppose a 1-Gbps point-to-point link is being set up between the Earth and a new lunar colony. The distance from the moon to the Earth is approximately 385,000 km, and data travels over the link at the speed of light - 3×10^8 m/s.**

(a) **Calculate the minimum RTT for the link.**

In order to calculate the minimum round-trip time, we first need to find the one-way propagation delay. Based on the text book and lecture PDFs the one-way propagation delay could be calculated using the following:

$$\text{One way propagation delay} = \frac{\text{Distance}}{\text{Speed of light}} = 385 \times 10^6 / 3 \times 10^8 = 1.283$$

in the above equation the unit for the speed of light is m/s that's why I have converted the km to the m for the distance. Now the total RTT is:

$$\text{RTT} = 2 \times \text{One way propagation delay} = 2 \times 1.283 = 2.566 \text{ seconds}$$

(b) **Using the RTT as the delay, calculate the delay × bandwidth product for the link.**

"*The delay × bandwidth product gives the volume of the pipe—the maximum number of bits that could be in transit through the pipe at any given instant"* (Section 1.5.2, page 44). Therefor:

$$\text{Delay} \times \text{Bandwidth} = 2.566 \times (1 \times 10^9) = 2.566 \times 10^9 \text{ bits}$$

(c) **What is the significance of the delay × bandwidth product computed in (b)?**

As mentioned in part b, the delay × bandwidth represents the amount of data that can be in transit over the link before an acknowledgment is received. In other words, as it is mentioned in the book if we think of the channel as a pipe, the latency is similar to the length of the pipe, while the bandwidth corresponds to its diameter (Figure 1.21, page 45). Therefore, the product of delay × bandwidth indicates how many bits the sender needs to transmit before the first bit reaches the receiver. In our case, if we want to use the 1 Gbps of our link we have to send the window of at least 2.566×10^9 bits of data before receiving acknowledgments to keep the link fully used.

(d) **A camera on the lunar base takes pictures of the Earth and saves them in digital format to disk. Suppose Mission Control on Earth wishes to download the most current image, which is 25 MB. What is the minimum amount of time that will elapse between when the request for the data goes out and the transfer is finished?**

$$\text{Time to send the image} = \frac{\text{Size of image}}{Link\ size} = (25 \times 10^6 \times 8) / (1 \times 10^9) = 0.2 \text{ sec}$$

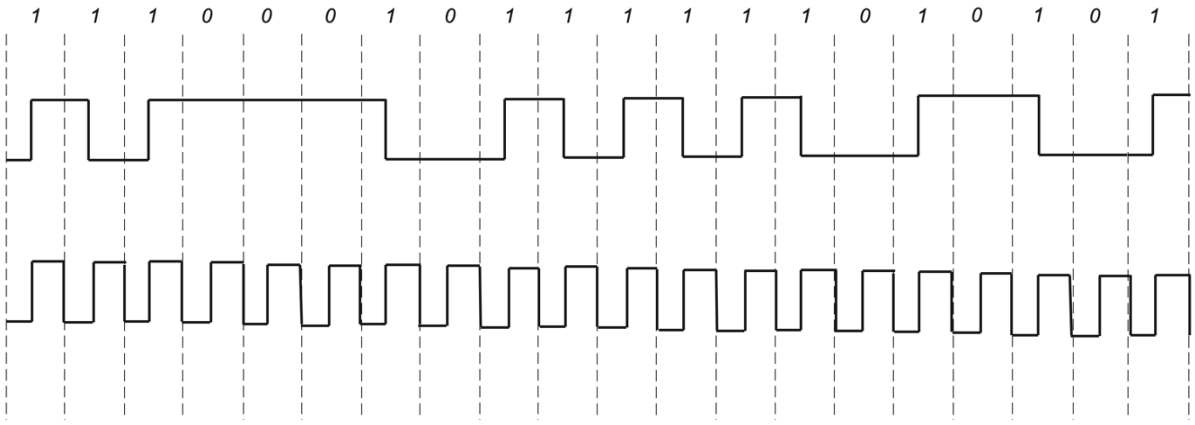
Note that the image size is 25 MB. Since the link unit is in bits per second, we need to convert the image size to bits ($25 \times 8 \times 10^6$). Additionally, there is a 1.283-second delay to send a request, which has been calculated in part a. There is also a 1.283-second delay for the last bit to propagate back from the Moon to Earth. Therefore, the minimum time is:

$$\text{Minimum time} = 0.2 + 1.283 + 1.283 = 2.766 \text{ seconds}$$

Problem 2: **Show the 4B/5B encoding, and the resulting NRZI signal, for the following bit sequence:**

1110 0101 0000 0011

4 Bit Code	5 Bit Code
1110	11100
0101	01011
0000	11110
0011	10101



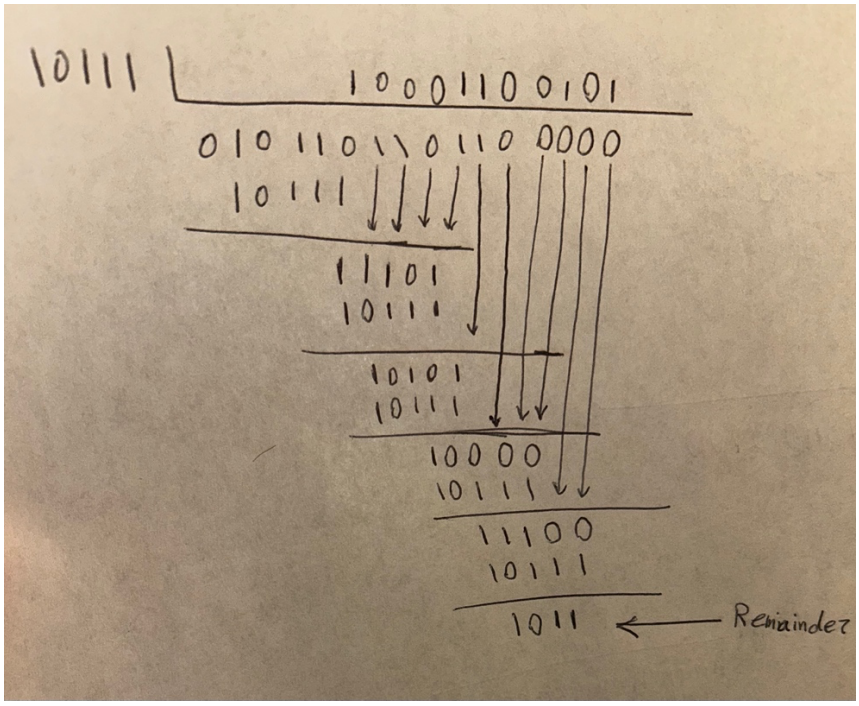
Problem 3**: Suppose we want to transmit the message 010110110110 and protect it from errors using the CRC polynomial $x^4 + X^2 + x + 1$.**

(a) **Use polynomial long division to determine the message that should be transmitted.**

M(x) = 010110110110

K = 4 => T(x) = 010110110110 0000

C(x) = 10111

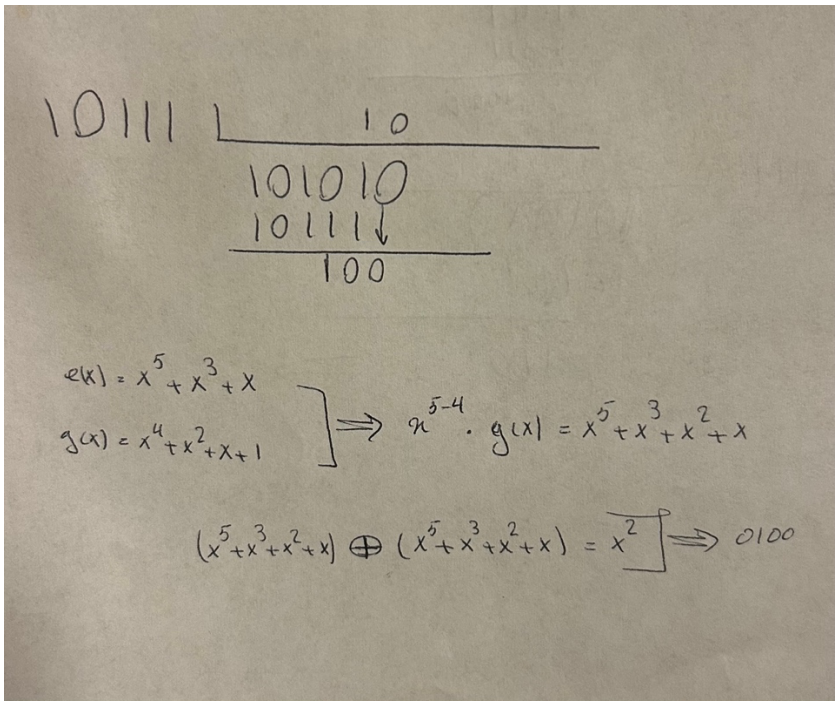


So, the final 16-bit frame that should be sent over the link is:

Final message = 0101101101101011

(b) **Suppose 101010 is the error that occurs in transmission of the code word to the remote host due to noise on the transmission link. What is the result of the receiver's CRC calculation? How does the receiver know that an error has occurred?**

The calculation in the receiver is as follows:



In the receiver, two scenarios can happen. First, if the remainder of the division is zero, it means that the received message is correct and there were no errors. Second, if there is a remainder that is not zero that means that an error has occurred. In our case, the remainder is x^2 (0100), which shows that an error has happened since e(x) divided by g(x) was not zero. This tells the receiver that one or more bits have been corrupted.

Problem 4: **Suppose that if a host detects a transmission while it is transmitting a frame, then: (i) if the host has already transmitted the 64 bit preamble, the host stops transmitting the frame and sends a 32 bit jamming sequence; (ii) else the host finishes transmitting the 64 bit preamble and then sends a 32 bit jamming sequence. For simplicity, assume a collision is detected as soon as an interfering signal first begins to reach a host. Suppose the packets are 512 bits long, which is the minimum length allowed. Hosts A and B are the only active hosts on a 10 Mbps Ethernet and the propagation time between them is 15 µS, or 150 bit durations. Suppose A begins transmitting a frame at time t = 0, and just before the frame reaches B, B begins sending a frame, and then almost immediately B detects a collision.**

(a) **Does A finish transmitting the frame before it detects that there was a collision? Explain.**

No. As mentioned in the question, it takes 15 µs, or 150-bit durations, for A's signal to arrive at B. Since A starts sending its preamble at time t = 0, by t = 15 µs, A's signal reaches B. However, at the same time, B starts sending its signals and at the same time, it detects a collision around t = 15 µs. Since B has not yet completed its preamble, it will finish transmitting the 64-bit preamble and then send a 32-bit jamming sequence. It will take another 15 µs for B's signal to reach A.

Thus, A detects the collision at around t = 30 µs. So, in order to find the bits that have passed we have:

$$30 / 0.1 = 300 \text{ bits}$$

The total message length is:

$$512 \text{ bits} + 64 \text{ bits} = 576 \text{ bits}$$

At the time A senses the collision, it has only transmitted 300 bits. Therefore, no, A does not finish transmitting its frame before it detects that there was a collision.

(b)**What time does A finish sending a jamming signal? What time does B finish sending a jamming signal**

As calculated in part a, host A detects the collision at t = 30 µs and since it has finished sending the preamble by then (64 / 0.1 = 6.4 µs), it will stop sending data as soon as it detects a collision and immediately sends 32 bits jamming signal. Therefore, sending the 32 bits jamming signal at 10 Mbps Ethernet will take 3.2 µs (32 / 0.1) so host A will complete the jamming signal at:

$$30 + 3.2 = 33.2 \text{ µs}$$

Host B starts sending its preamble around 15 µs and detects the collision at 15 µs and since the preamble is not done, it finishes the preamble and then transmits 32 bits of jamming. So, if we consider the worst-case scenario host B has sent only 1 bit of 64 bits of its preamble at t = 15 µs and has 63 more bits to go. Therefore, it takes around 6.3 µs (63 / 0.1) to finish its preamble and 3.2 µs for its jamming signal (32 / 0.1). In total, it takes around 9.5 µs for host B to finish sending its jamming signal, and since this host has started at t = 15 µs it would finish sending its jamming signal at:

$$15 + 9.5 = 24.5 \text{ µs}$$

Problem 5**: Consider a selective repeat Protocol with a sender window size of 4 and a sequence number range of 32. Suppose that at time t, the next in-order packet that the receiver is expecting has a sequence number of k. Assume that the medium does not reorder messages. What are the possible sets of sequence numbers inside the sender's window at time t? Justify your answer.**

In the selective repeat protocol, the smallest sequence number in the window moves when the earliest unacked packet is acknowledged. Since the next in-order packet that the receiver is expecting has the sequence number k then it means that all the packets with sequence numbers up to (k-1) mod 32 have been received by the receiver and the sender has received the acks for those packets. Therefore, the first unacked packet in the sender window can't be less

than k. Now let's consider that the sender has sent the k and received its acked then the receiver should have expected the sequence number of k+1, not k. Therefore, we could understand that the first packet in the sender window has the sequence number of exactly k.

Because the sender window is 4 the sender could send 4 packets at maximum but the thing is that the sender may not use all of its slots in the window. So to find the possible set in sequence numbers inside the sender window at time t we could start from k and imagine that k is the only unpacked packet and increase the numbers of packets in the sender window so we have these possibilities:

1. Sender's window = k
2. Sender's window = k, k+1
3. Sender's window = k, k+1, k+2
4. Sender's window = k, k+1, k+2, k+3

Since the window size is 4 these are all the possible options that we could have in our sender window.

Problem 6: **Suppose you are designing a sliding window protocol for a 1-Mbps point- to-point link to the moon, which has a one-way latency of 1.25 seconds. Assuming that each frame carries 1 KB of data, what is the minimum number of bits you need for the sequence number?**

In order to have the minimum number of bits for our sequence number we first need to calculate the delay × bandwidth product to find the volume of the pipe since the window size should at least cover the delay × bandwidth over the RTT to keep the pipeline full. As the question mentioned the one-way latency is 1.25 sec so the RTT is 2.5 (1.25 + 1.25).

$$\text{delay} \times \text{bandwidth} = 2.5 \times (1 \times 106) = 2.5 \times 106 \text{ bits}$$

Each frame is 1 KB in other words each frame is 8000 bits (1×103×8). So, to fill the pipeline we need the:

$$\text{number of frames} = 2.5 \times 106 / 8000 = 312.5$$

we need at least 313 frames and in a sliding window protocol, the sender's window must fit within the sequence number space. So, we have:

$$2n \geq 313 \text{ frames} \rightarrow n = 9$$

the minimum number of bits required for the sequence number is 9.

Problem 7: **Suppose a router has built up the routing table shown in following table. The router can deliver packets directly over interfaces 0 and 1, or it can forward packets to**

routers R2, R3, or R4. Describe what the router does with a packet addressed to each of the following destinations:

(a) **128.96.39.10**

Row	Network address	Broadcast address	Usable host address	
			From	to
1	128.96.39.0	128.96.39.127	128.96.39.1	128.96.39.126
2	128.96.39.128	128.96.39.255	128.96.39.129	128.96.39.254
3	128.96.40.0	128.96.40.127	128.96.40.1	128.96.40.126
4	192.4.153.0	192.4.153.63	192.4.153.1	192.4.153.62

Let's first find the useable host addresses of each network. In order to do that we should look at the subnet mask and find out the network and host size. For example, in the first row, we have a subnet mask for 255.255.255.128, which in binary is:

11111111.11111111.11111111.10000000

This shows that the first 25 bits of any IP address within this network are fixed and we have 7 bits for host addresses. Now with these 7 bits, we have 2^7 possible values for our host. So, we have 128 IP addresses but the address with all zeros is for the network address and the one with all ones is for the broadcasting. Therefore, this network covers addresses from 128.96.39.0 to 128.96.39.127 which are shown in the table above. We use the same logic to calculate the other addresses and fill out the table.

For 128.96.39.10 we could see that it fit in the range of the first row so the router will send this packet to the interface 0.

(b) **128.96.40.12**

Using the table above we could see that this address will fit in the third row so the router will send it to the R2.

(c) **128.96.40.151**

This packet doesn’t match any of the about ranges so the router will send it through the default. So, this packet will be sent to R4.

(d) **192.4.153.17**

This packet is in the range of the fourth row so the router will send it through R3.

(e) **192.4.153.90**

This packet is out of range again and does not fit in any of our ranges so the router will send it to the default and the packet will be sent to R4.

Problem 8 (old version): **Using Dijkstra's algorithm compute the forwarding table at x for the following network.**

To do this, we create a tree, and in each step, we find the unseen neighbors with the minimum cost from the root, which is x in our case. Since we are starting from x, the neighbors for x are y,v, and w, so we have :

- x to y -> cost: 6
- x to v -> cost: 3
- **x to w -> cost: 1**

we choose the node with the smallest cost among all the leaves. So, we choose w and check its neighbors therefore, we have:

- **w to v -> cost: 1 => x to v -> 2**
- w to u -> cost: 3 => x to u -> 4

selected node is v, then we have:

- **v to y -> cost: 1 => x to y -> 3**
- v to t -> cost: 9 => x to t -> 11
- v to u -> cost: 1 => x to u -> 3

the selected node is y. Its neighbors are:

- y to z -> cost: 14 => x to z -> 17
- y to t -> cost: 4 => x to t -> 7

selected node u from the last step with the cost of 3 since it has the lower cost (this has been shown in the tree by the green color).

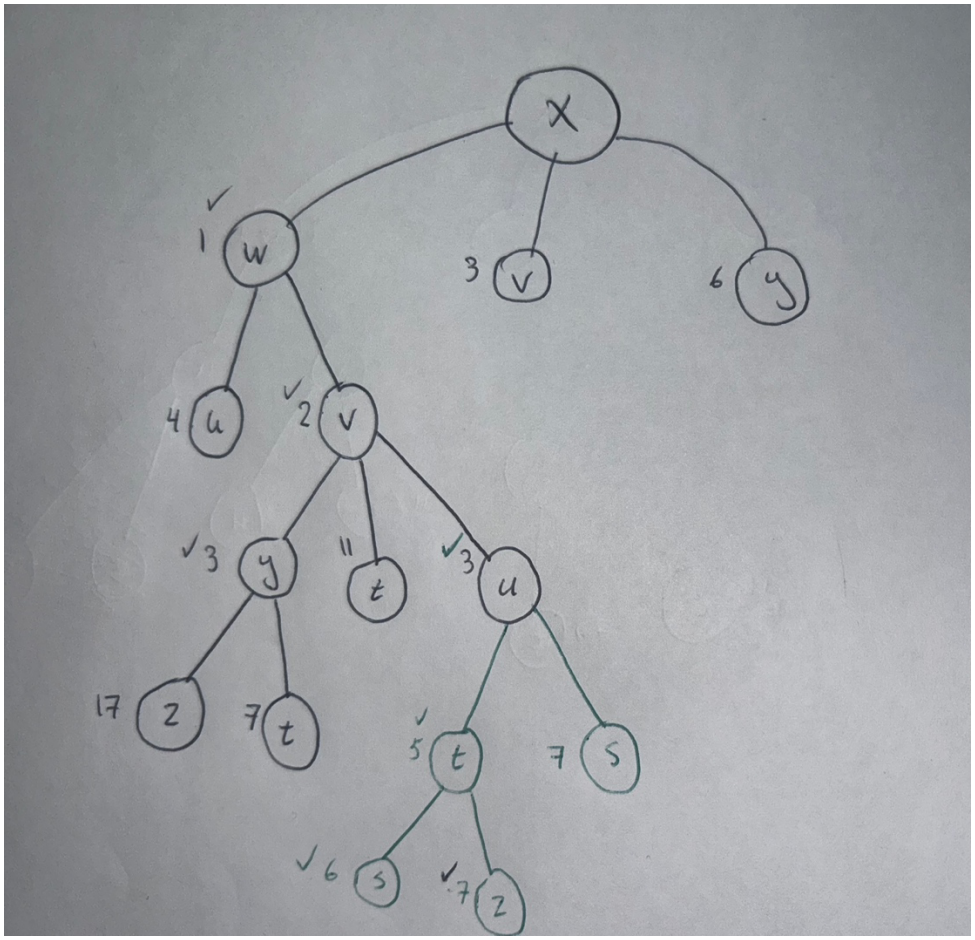
- **u to t -> cost: 2 => x to t -> 5**

- u to s -> cost: 4 => x to s -> 7

the selected node is t so:

- **t to s -> cost: 1 => x to s -> 6**
- t to z -> cost: 2 => x to z -> 7

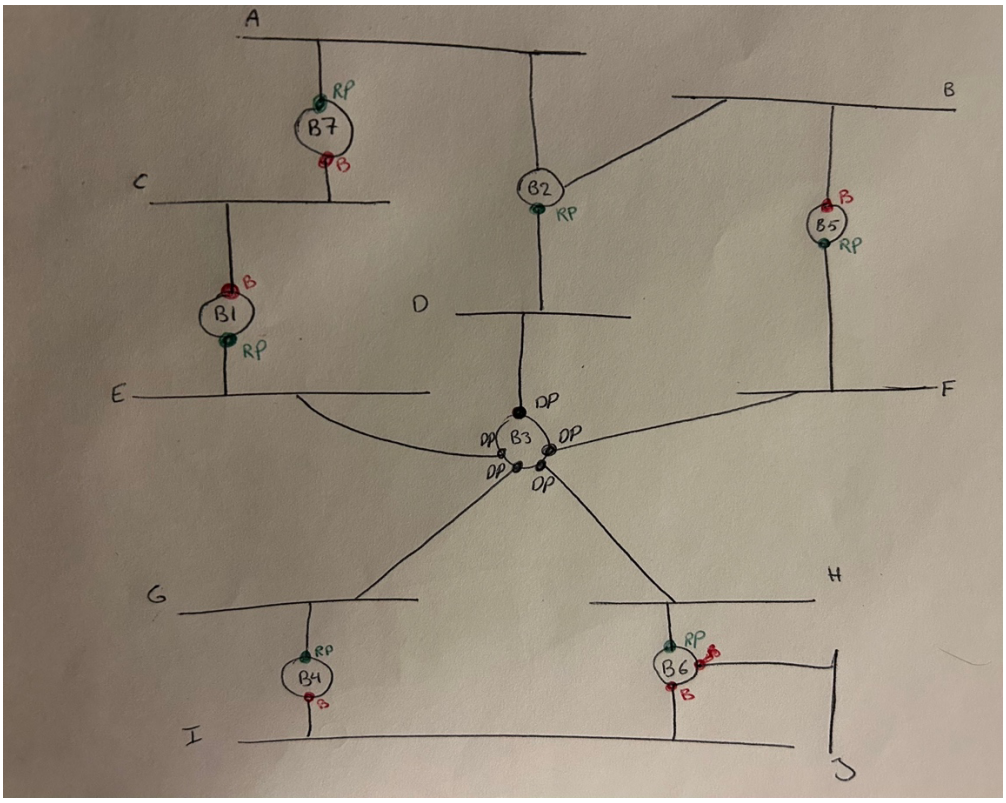
selected node = s. finally, the only unseen node is z with the minimum cost of 7 from the x.

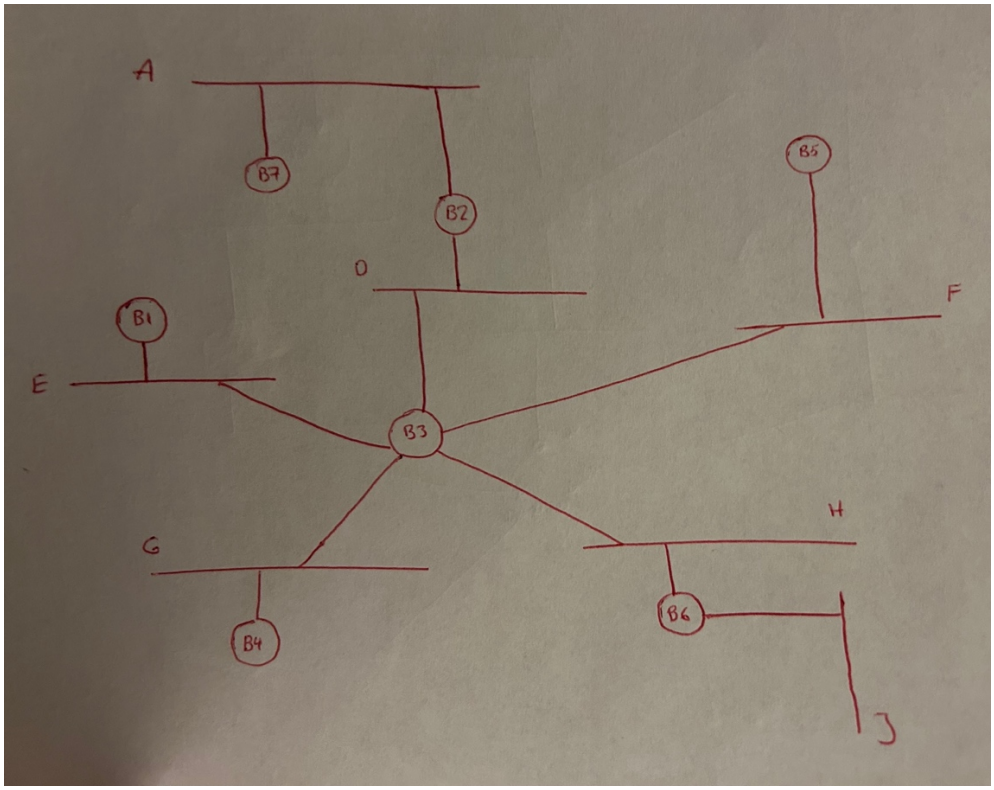


Problem 8 (new version): **Given the extended LAN shown in the figure below, indicate which ports are not selected by the spanning tree algorithm, and discuss how any ties are resolved.**

To create a spanning tree, we need to develop a tree that does not contain any loops. The first step is to identify the root of the tree, which is selected based on the MAC addresses and priority. Here, we will select B3 as the root since it is the most central node, and there are no specified MAC addresses or priorities. Next, we determine the root ports, which are the ports that provide the shortest path to the root. In the figure, these ports are marked green and tagged with RP. After determining the root ports, we move on to select the designated ports, which are marked

with black on the switches and tagged as DP. Finally, we determine the blocked ports, which are shown in red. The resulting spanning tree is shown in the second figure.





Problem 9**: Suppose a TCP message that contains 1024 bytes of data and 20 bytes of TCP header is passed to IP for delivery across two networks interconnected by a router (i.e., it travels from the source host to a router to the destination host). The first network has an MTU of 1024 bytes; the second has an MTU of 576 bytes. Each network's MTU gives the size of the largest IP datagram that can be carried in a link-layer frame. Give the sizes and offsets of the sequence of fragments delivered to the network layer at the destination host. Assume all IP headers are 20 bytes.**

First, we need to calculate the payload size from the first network to the router. The TCP message size is 1024 bytes, and the TCP header adds an additional 20 bytes, totaling 1044 bytes. However, our Maximum Transmission Unit (MTU) is 1024 bytes, which means we need to create fragments to send the data. Each fragment will have its own IP header, which is also 20 bytes. To determine the maximum data size for each fragment on this network, we subtract the MTU from the IP header which gives us 1004 bytes. However, since 1004 is not a multiple of 8 bytes, we must reduce it to 1000 bytes because the IP fragment offset field measures offset in units of 8 bytes. Thus, the first fragment will be structured as follows:

Header	Data	Total length	Offset
20	1000	1020	0

We have 44 bytes remaining (1044 – 1000). The second fragment can be calculated similarly, but here we also need to consider the offset. The IP header is 20 bytes, and the data is 44 bytes, making the total length 64 bytes. The offset is calculated based on the previous fragment, as explained below:

$$\text{Offset} = 1000 / 8 = 125$$

So, the second fragment looks like:

Header	Data	Total length	Offset
20	44	64	125

Now we are at the second phase which sends data from the router to the second network. In this phase, the MTU is 576 bytes. Since we first receive a fragment that is 1000 bytes in size, we need to fragment it again. Each fragment will have a data size of 556 bytes (MTU-IP header). Following the same principle as before, the closest multiple of 8 is 552 bytes. Thus, the first fragment will be:

Header	Data	Total length	Offset
20	552	572	0

For the second fragment we have the same logic with the offset calculation:

Offset 552 / 8 = 69

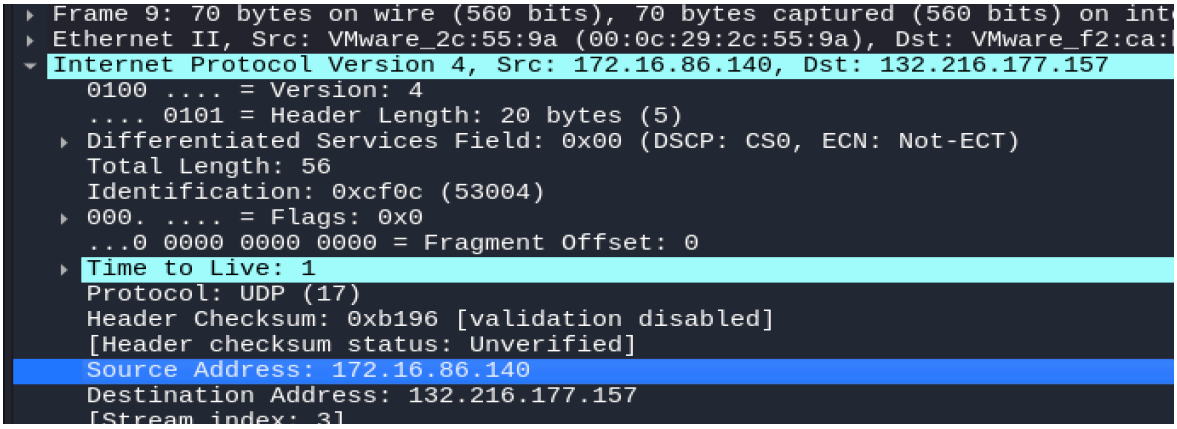
Header	Data	Total length	Offset
20	448	468	69

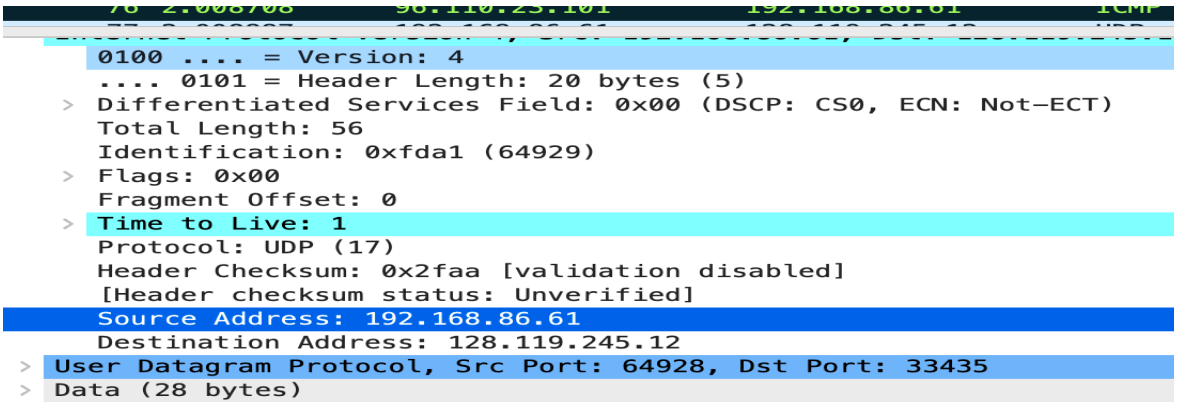
The last fragment is 64 bytes, which is smaller than our MTU, so we don't need to create any additional fragments. Therefore, we will simply send it to the network (20 bytes IP header + 44 bytes data, with the offset of 125). The second network will receive a total of 552 + 448 + 44 bytes of data, which is the same data that we sent from the first network.

Part 2:

1. **Select the first UDP segment sent by your computer via the traceroute command to www.mcgill.ca. (Hint: this is 44th packet in the trace file in the ip-wireshark- trace1-1.pcapng file in footnote 1). Expand the Internet Protocol part of the packet in the packet details window. What is the IP address of your computer?**

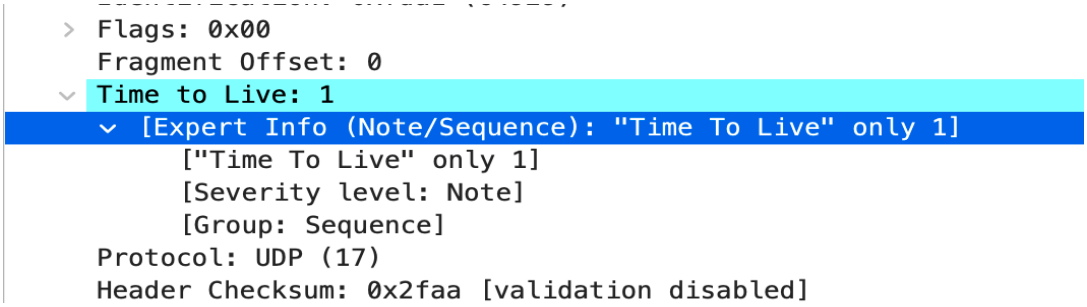
The IP address for my own computer is 172.16.86.140, and the IP address for the trace file that was provided to us is 192.168.86.61. The screenshots for these IP addresses are shown below:

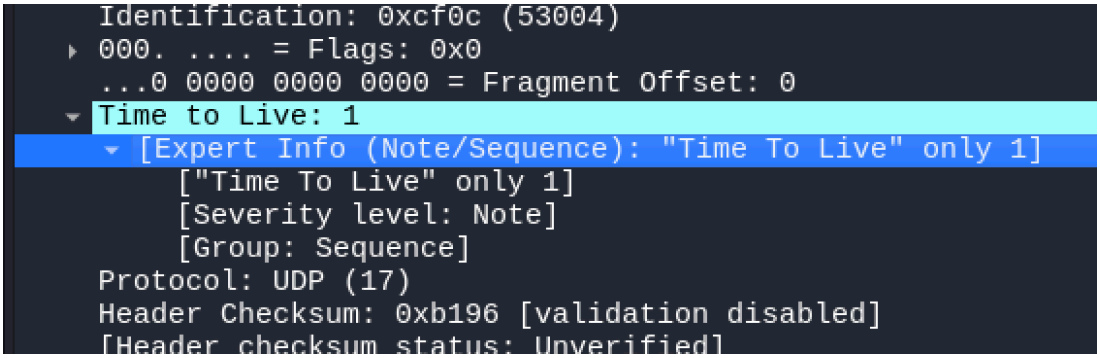




2. **What is the value in the time-to-live (TTL) field in this IPv4 datagram's header?**

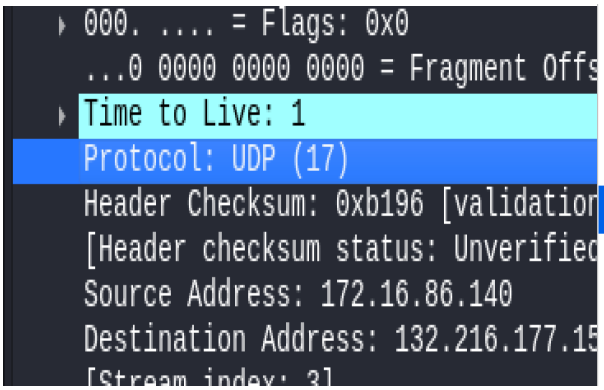
Time to live is 1 in both cases.

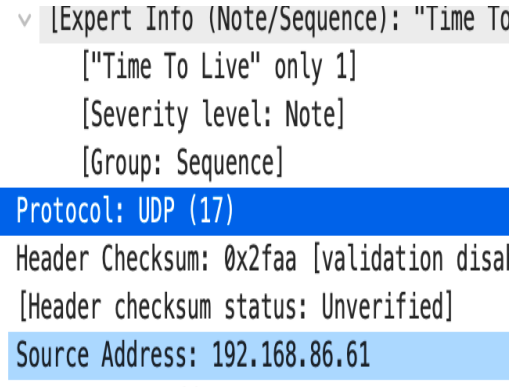




3. **What is the value in the upper layer protocol field in this IPv4 datagram's header? [Note: the answers for Linux/MacOS differ from Windows here].**

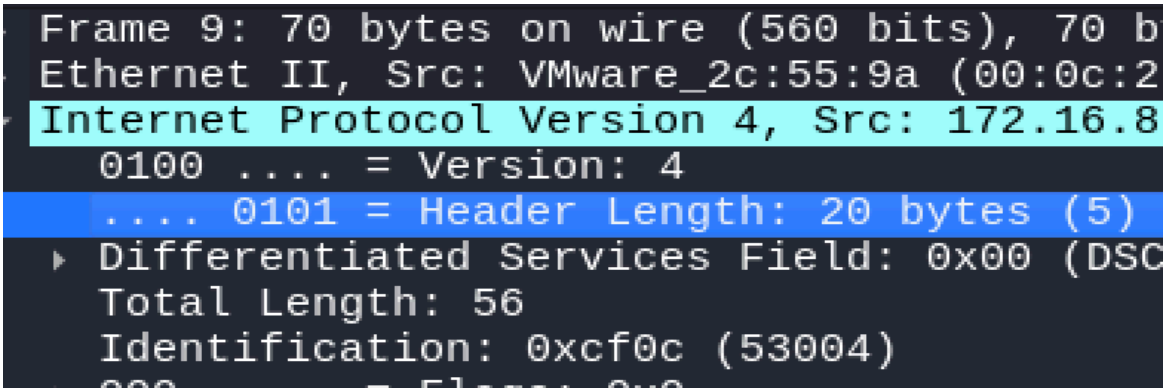
Here, the protocol is UDP (17) in both files.

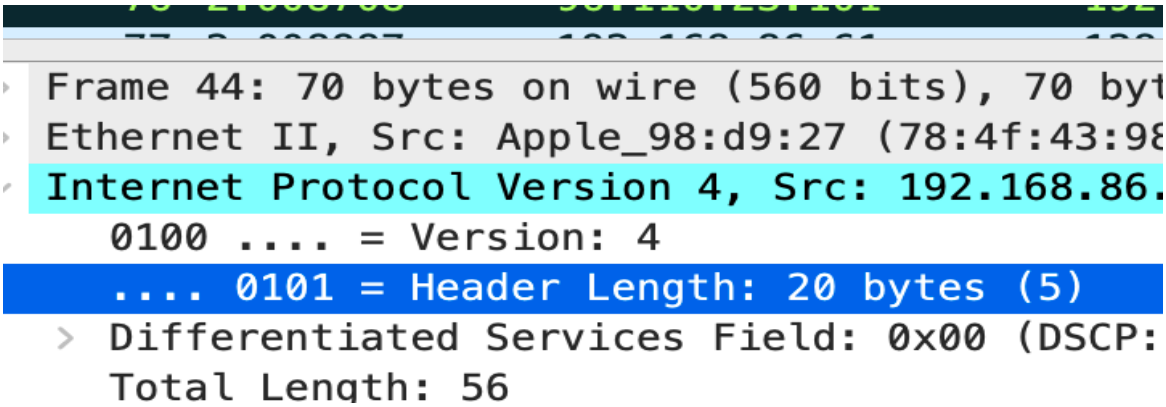




4. **How many bytes are in the IP header?**

The IP header size is 20 bytes for both files. You can find this in the header length part.





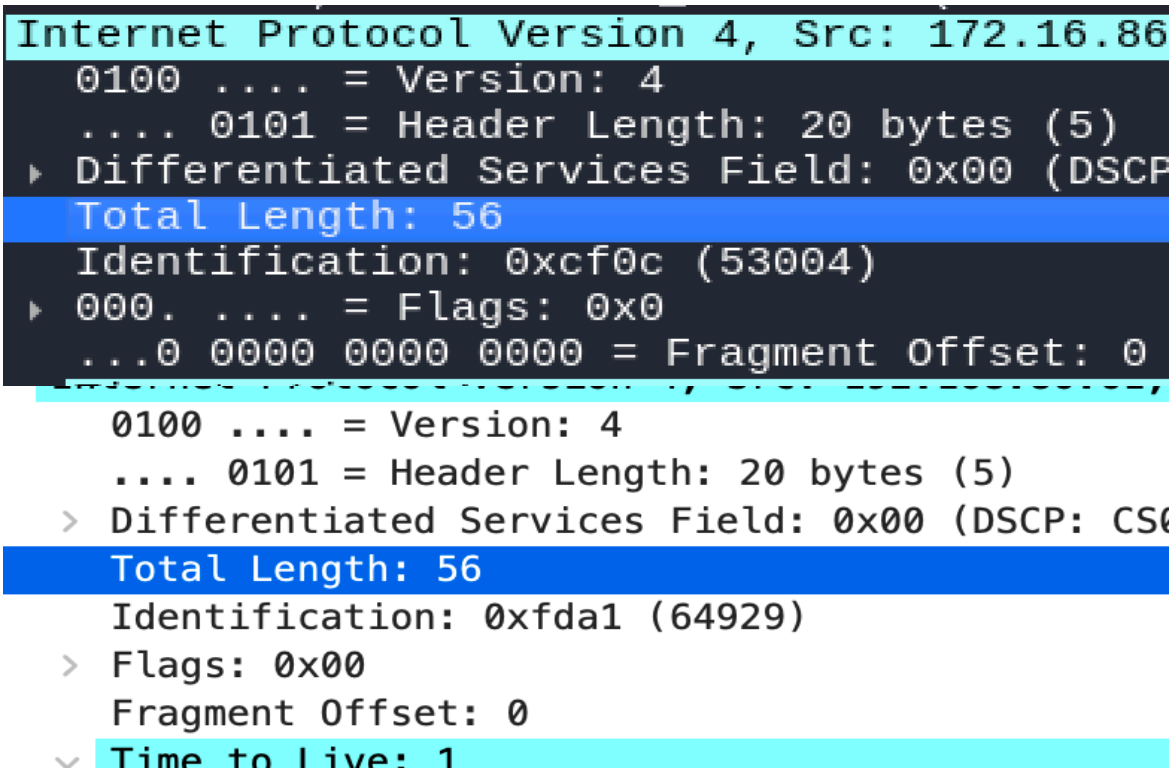
5. **How many bytes are in the payload of the IP datagram? Explain how you determined the number of payload bytes.**

As shown in the figures, we have the total length of the packet as well as the IP header size. So, we could calculate the payload size using the following formula:

Payload size = total length – IP header size

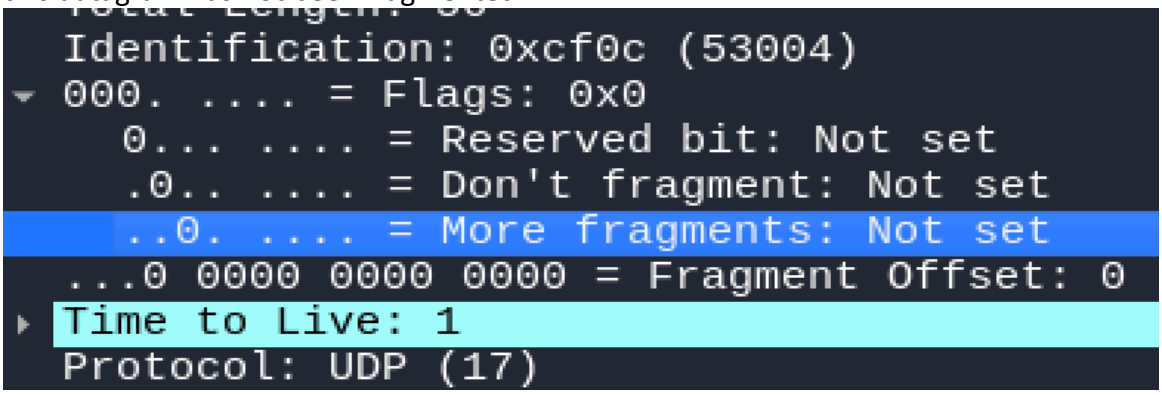
We can see that the total length is 56 bytes for both test cases, and we know that the IP header size is 20 bytes, so the size of the payload is:

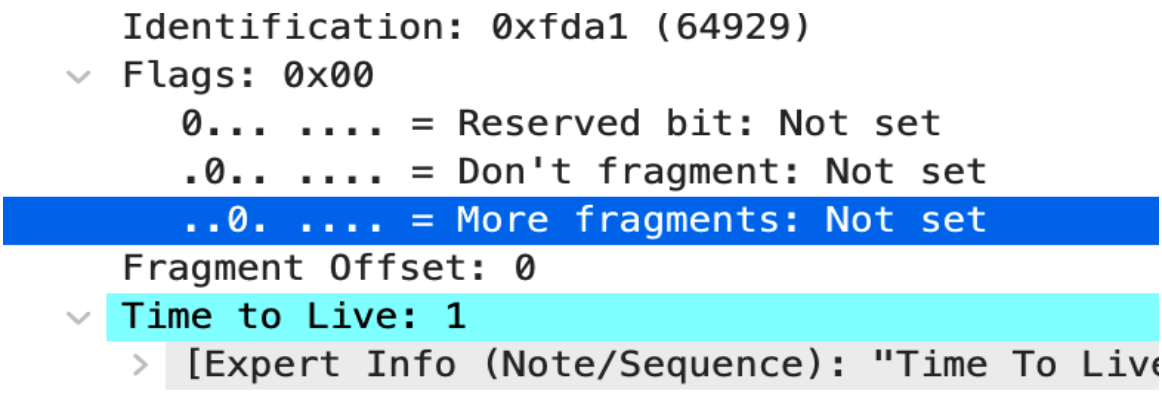
Bytes in payload = 56 – 20 = 36 bytes



6. **Has this IP datagram been fragmented? Explain how you determined whether or not the datagram has been fragmented**.

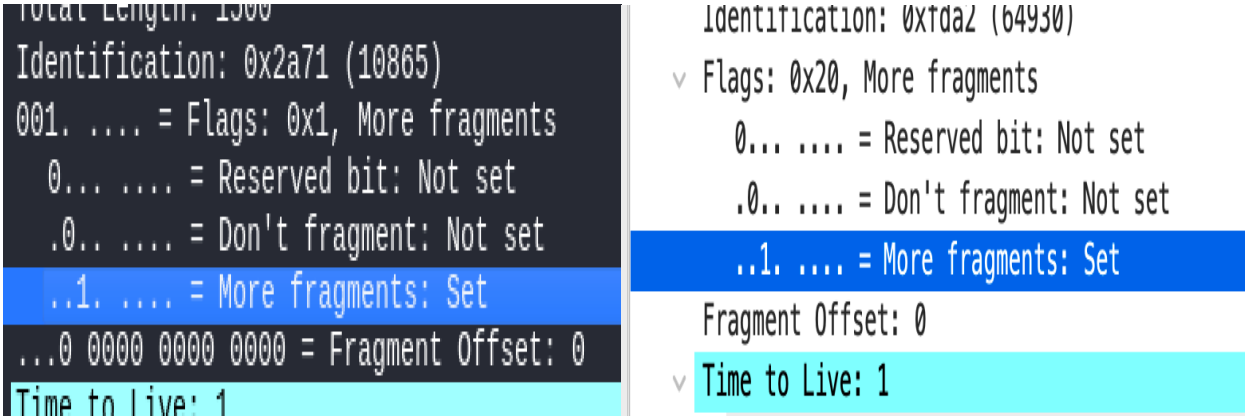
No, this datagram has not been fragmented. You can verify this by looking at the flags. Specifically, there is a flag called More Fragments, which is set to 0, indicating that the datagram is not fragmented. Additionally, as explained in Part 1, the offset is also 0, which further confirms that this datagram has not been fragmented.





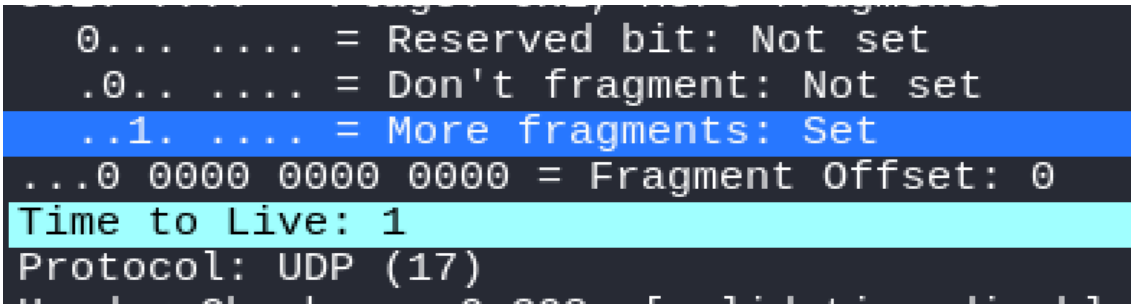
7. **Find the first IP datagram containing the first part of the segment sent to 132.216.177.157 by your computer via the traceroute command to www.mcgill.ca, after you specified that the traceroute packet length should be 3000. (Hint: This is packet 179 in the ip-wireshark-trace1-1.pcapng trace file which is designated to 128.119.145.12). What information in the IP header indicates that this datagram has been fragmented?**

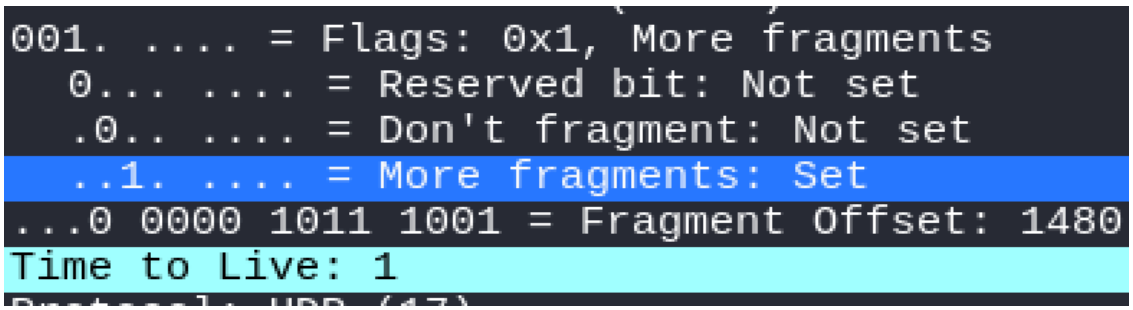
In the flags section, you can see that the more fragments are set to 1, which means that this datagram has been fragmented. Note that the offset is set to 0 because this is the first fragment that we are sending.

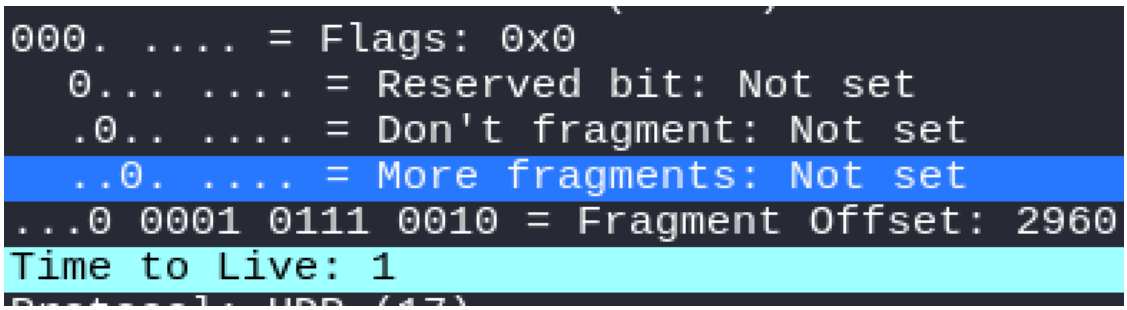


8. **How many fragments are holding of the data of this UDP segment? Explain how you determined the number of fragments.**

To find the number of fragments, we could look at the more fragments flag in the flags section. Combining this flag with the offset, we could determine the number of fragments. The first fragment has the more fragments flag set to 1 and offset zero, and the last one, which in this example is the third one, has the more fragments set to 0 and offset of 2960.

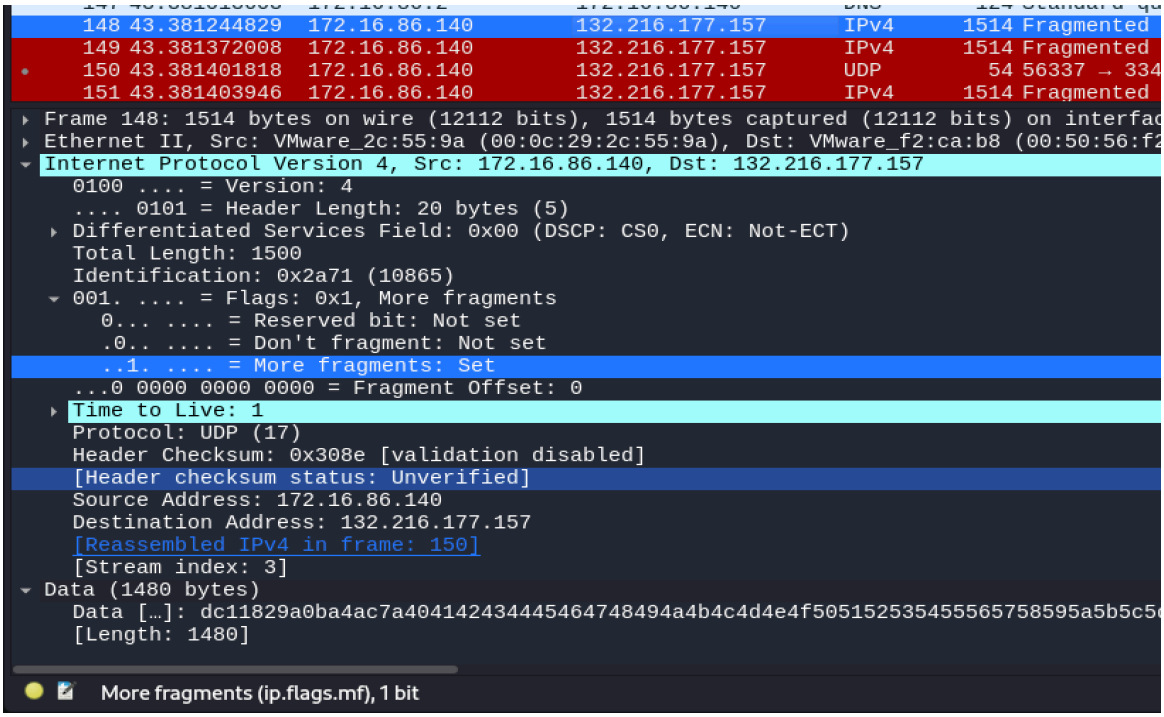


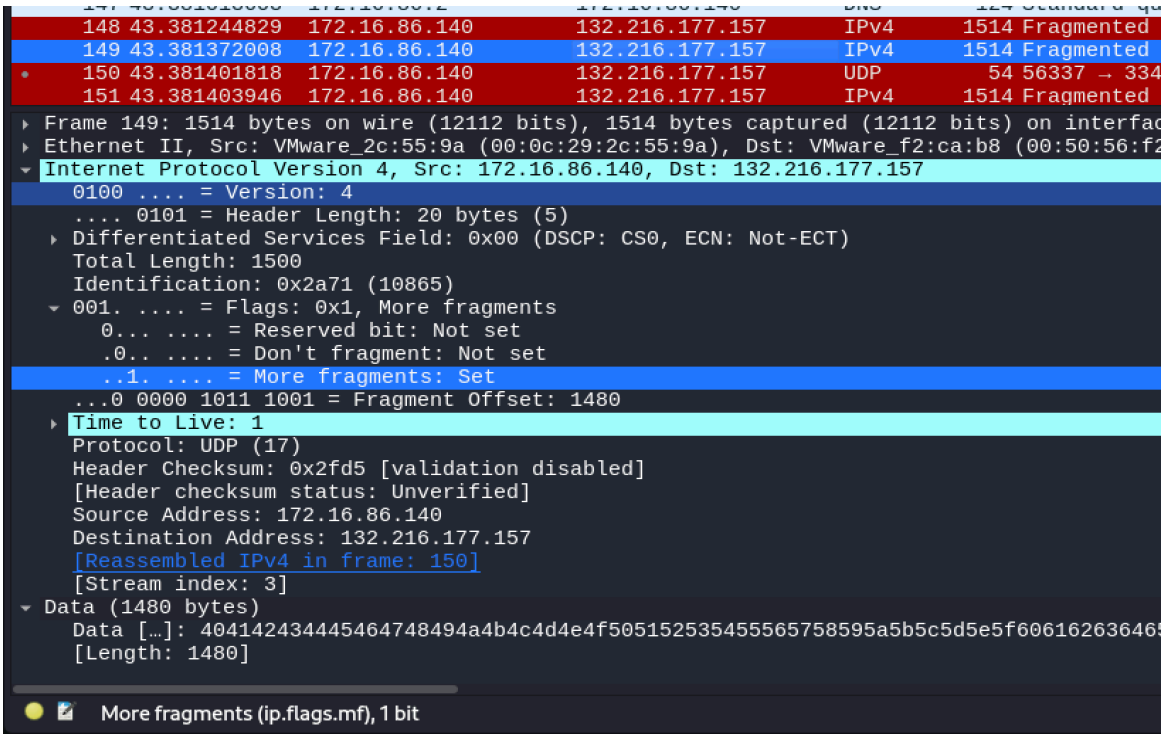


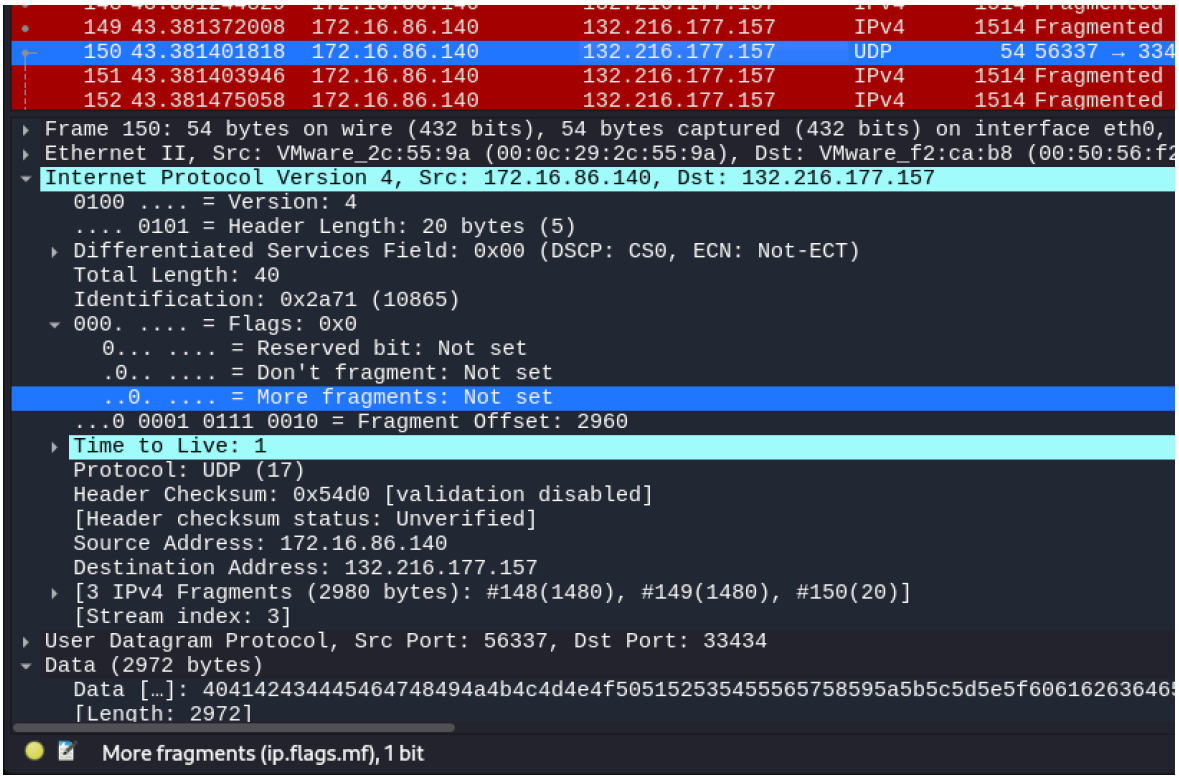


9. **Inspect the IP datagrams containing the fragments of the fragmented UDP segment and list the fields change in the IP header between each fragment datagram. What information in the IP header for this packet indicates whether this is the first fragment, a latter fragment, or the last fragment of that segment?**

As shown in the figures, the IP address, source address, destination address, protocol, and TTL are the same across all fragments. For the second part of the question, as mentioned in the previous question, you can identify which fragment is the first, middle, or last one by looking at the more fragments flag and the offset. Typically, the first fragment has the more fragments flag set to 1 and an offset of 0. The middle fragment also has the more fragments flag set to 1 but with an offset greater than 0 (in our example, it's 1480). Finally, the last fragment has the more fragments flag set to 0 and an offset greater than 0 (in our case, it is 2960).


148 43.381244829 172.16.86.140 132.216.177.157 IPv4 1514 Fragmented
149 43.381372008 172.16.86.140 132.216.177.157 IPv4 1514 Fragmented
150 43.381401818 172.16.86.140 132.216.177.157 UDP 54 56337 → 334
151 43.381403946 172.16.86.140 132.216.177.157 IPv4 1514 Fragmented
Frame 148: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interfac
Ethernet II, Src: VMware_2c:55:9a (00:0c:29:2c:55:9a), Dst: VMware_f2:ca:b8 (00:50:56:f2
Internet Protocol Version 4, Src: 172.16.86.140, Dst: 132.216.177.157
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1500
Identification: 0x2a71 (10865)
001. .... = Flags: 0x1, More fragments
0... .... = Reserved bit: Not set
.0.. .... = Don't fragment: Not set
..1. .... = More fragments: Set
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 1
Protocol: UDP (17)
Header Checksum: 0x308e [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.16.86.140
Destination Address: 132.216.177.157
[Reassembled IPv4 in frame: 150]
[Stream index: 3]
Data (1480 bytes)
Data […]: dc11829a0ba4ac7a4041424344454647484 94a4b4c4d4e4f505152535455565758595a5b5c5
[Length: 1480]
More fragments (ip.flags.mf), 1 bit


148 43.381244829 172.16.86.140 132.216.177.157 IPv4 1514 Fragmented
149 43.381372008 172.16.86.140 132.216.177.157 IPv4 1514 Fragmented
150 43.381401818 172.16.86.140 132.216.177.157 UDP 54 56337 → 334
151 43.381403946 172.16.86.140 132.216.177.157 IPv4 1514 Fragmented
Frame 149: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interfac
Ethernet II, Src: VMware_2c:55:9a (00:0c:29:2c:55:9a), Dst: VMware_f2:ca:b8 (00:50:56:f2
Internet Protocol Version 4, Src: 172.16.86.140, Dst: 132.216.177.157
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1500
Identification: 0x2a71 (10865)
001. .... = Flags: 0x1, More fragments
0... .... = Reserved bit: Not set
.0.. .... = Don't fragment: Not set
..1. .... = More fragments: Set
...0 0000 1011 1001 = Fragment Offset: 1480
Time to Live: 1
Protocol: UDP (17)
Header Checksum: 0x2fd5 [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.16.86.140
Destination Address: 132.216.177.157
[Reassembled IPv4 in frame: 150]
[Stream index: 3]
Data (1480 bytes)
Data […]: 4041424344454647484 94a4b4c4d4e4f505152535455565758595a5b5c5d5e5f60616263646
[Length: 1480]
More fragments (ip.flags.mf), 1 bit



References:

Larry Peterson & Bruce Davie, Computer Networks --a systems approach (5th Edition or Latest edition available from the book site)