Peyman Shobeiri

Computer networks- Assignment 2

McGill university

ID: 261286438

Winter 2025

Table of Contents

Part 1:

Problem 1: **Suppose P, Q, and R are network service providers with respective CIDR address allocations C1.0.0.0/8, C2.0.0.0/8, and C3.0.0.0/8. Each provider's customers initially receive address allocations that are a subset of the provider's. P has the following customers:**

PA, with allocation C1.A3.0.0/16

PB, with allocation C1.B0.0.0/12.

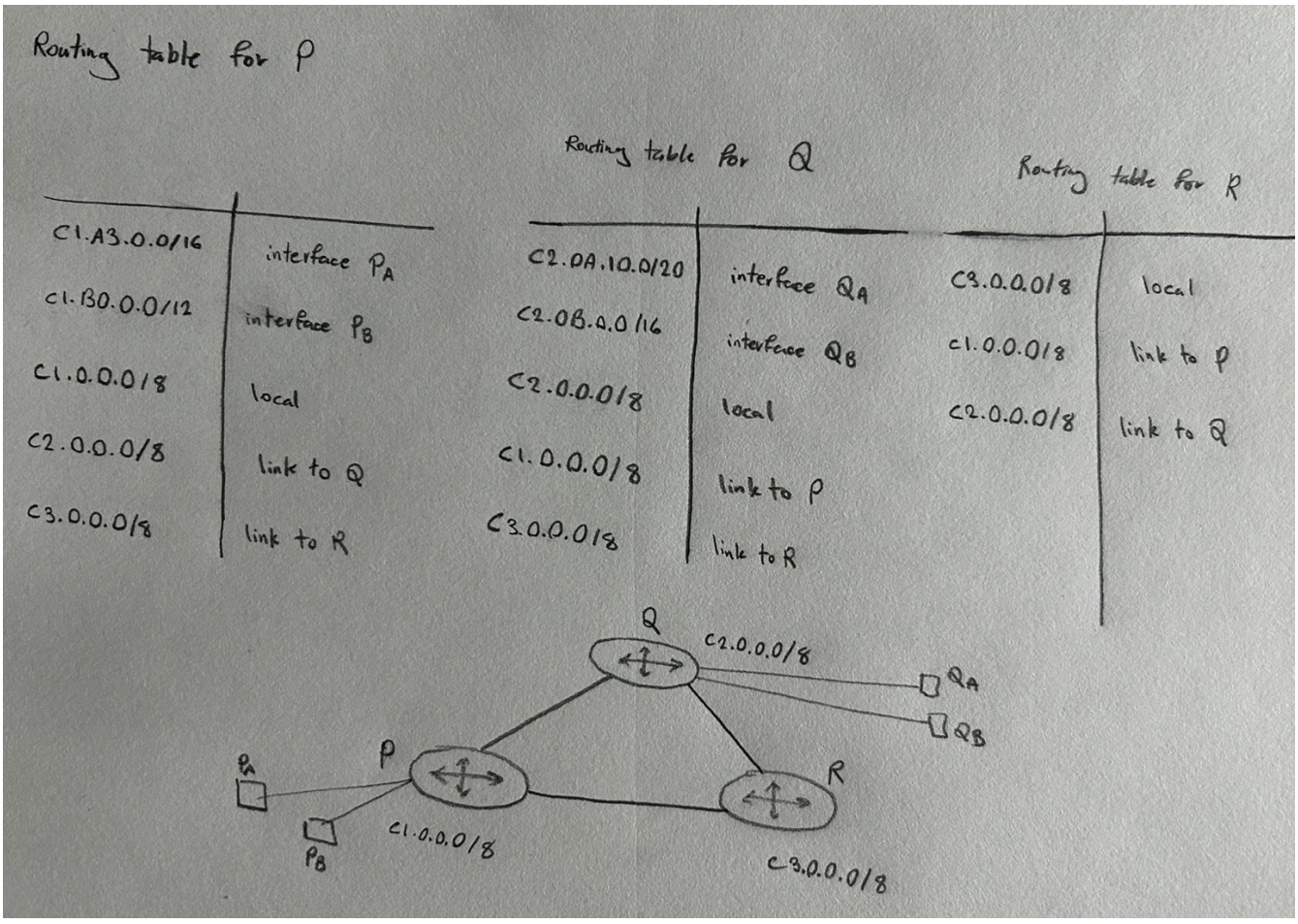
Q has the following customers:

QA, with allocation C2.0A.10.0/20

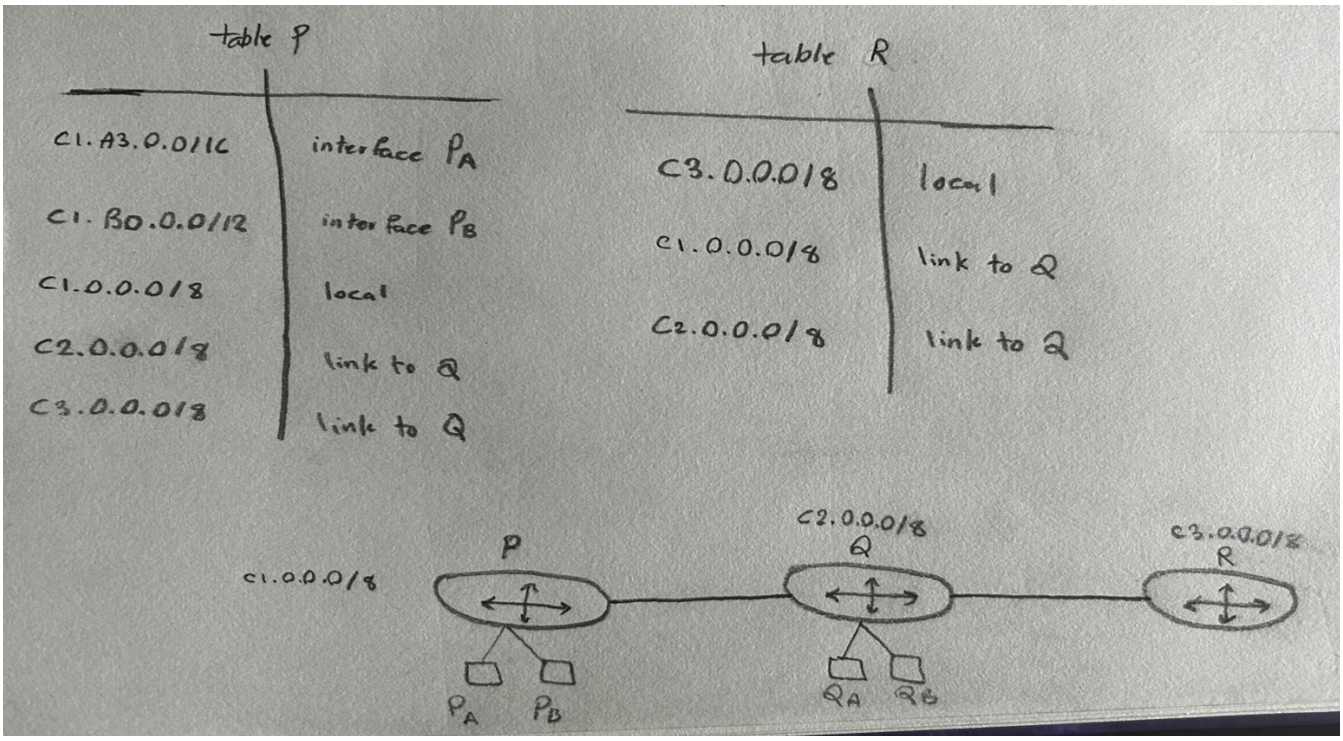
QB, with allocation C2.0B.0.0/16.

Assume there are no other providers or customers.

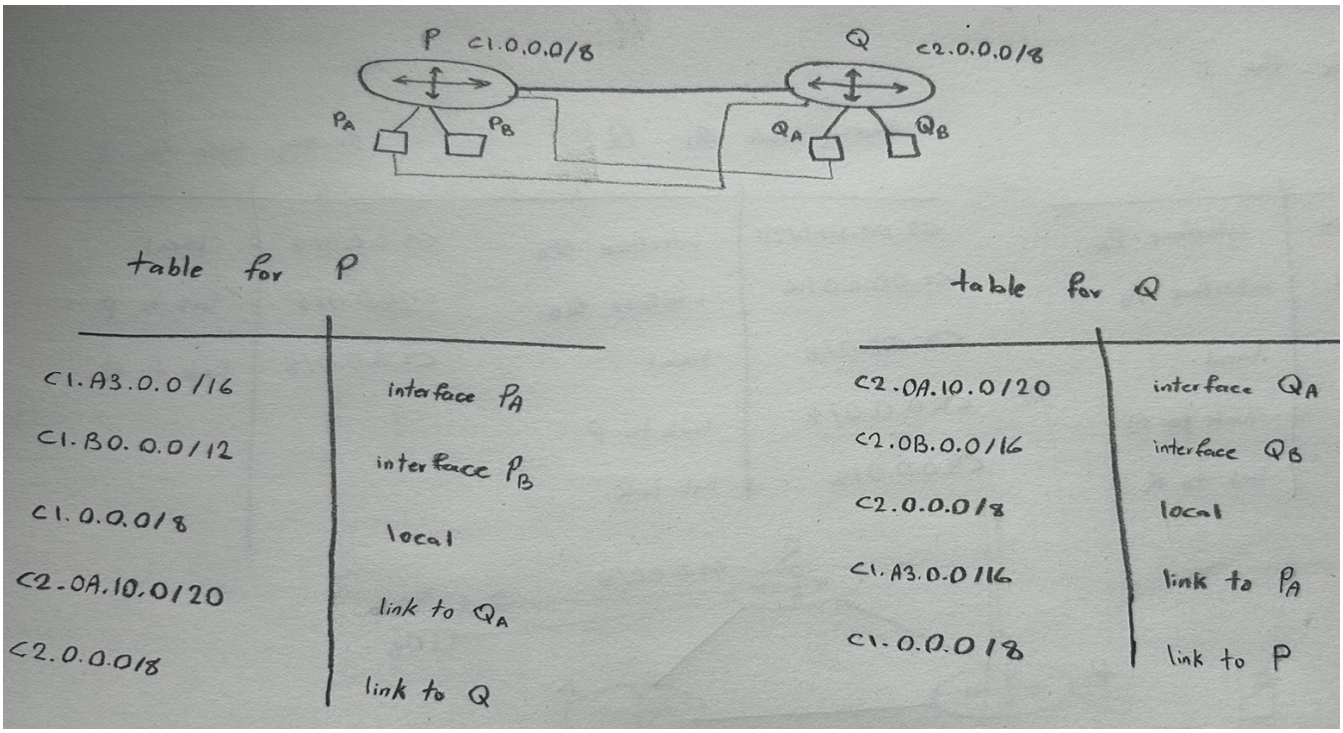
(a) **Give routing tables for P, Q, and R assuming each provider connects to both of the others.**



(b) **Now assume P is connected to Q and Q is connected to R, but P and R are not directly connected. Give tables for P and R.**



(c) **Suppose customer PA acquires a direct link to Q, and QA acquires a direct link to P, in addition to existing links. Give tables for P and Q, ignoring R.**



Problem 2: **Show the distance vector at each router after the distance-vector routing algorithm converges for the following network. Assume the link between router B and D failed, and show the updated distance vector at each router after the convergence of the algorithm following this link failure.**

1) Distance vector before the link failure:

	A	B	C	D	E
A	0	5	14	7	12
B	5	0	9	2	7
C	14	9	0	9	14
D	7	2	9	0	5
E	12	7	14	5	0

2) Distance vector after the link failure:

	A	B	C	D	E
A	0	5	14	23	28
B	5	0	9	18	23
C	14	9	0	9	14
D	23	18	9	0	5
E	28	23	14	5	0

Problem 3**: Provide a simple but complete example network where poison reverse would not solve the loop creation problem if the distance-vector algorithm is used for computing the forwarding tables.**

In the figure, Router A is the destination that all other routers must reach. Now, let's consider the scenario where the link between A and B fails. Under a basic distance vector protocol, a classic count to infinity loop can occur among routers B, C, and D. Router B notices that its connection to A is lost, so it attempts to learn a route to A from either C or D. Meanwhile, Router C may believe that B has a valid route to A, and the cycle continues. Even with the poison-reverse technique which tells that once a router selects a neighbor N as the next hop to reach a destination X, it should advertise the distance to X as infinity (∞) back to N, this method fails to prevent loops if each router learns about the route to A from different neighbors in a ring configuration.

Specifically, no single router is using a neighbor to reach A and simultaneously sending an ∞ advertisement back to that same neighbor. As a result, they continue to point to one another in a cycle. In other words, poison reverse is effective only when there are at most two routers involved. In a ring of three or more routers, each router can mistakenly believe that the other

neighbor has a valid route, which means the ∞ advertisement never reaches its intended destination to break the loop.

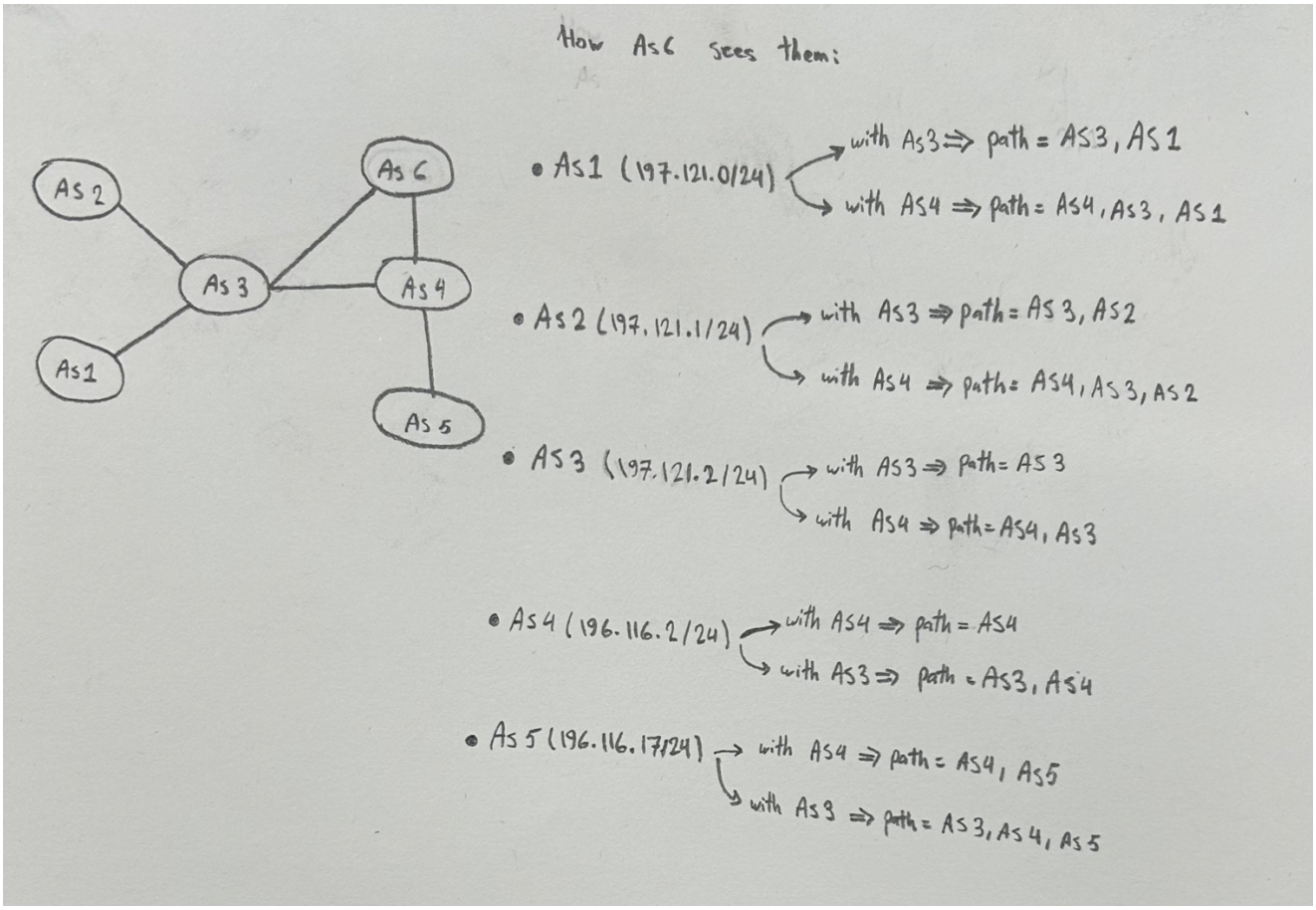
Problem 4: **For the following simple AS topology, give the BGP updates that can be received at AS 6.**

Because each Autonomous System (AS) re-advertises the routes it learns, AS 6 will generally receive two advertisements for each prefix:

1. One advertisement learned directly from the AS that is closest to the origin.

2. Another advertisement learned indirectly, through one additional AS hop.

So, for AS6 we have:



So, at AS 6, each route can be acquired in two ways: one short path from the nearer neighbor and one longer path that has crossed the other hop (AS3 or AS4). The actual best path choice for each prefix depends on local policies. Therefore, all of these updates could arrive at AS6.

Problem 5: **A host with private IP address 192.168.2.25 is communicating with an internet host with public IP 197.156.25.39. The local host is opening a TCP socket with local port number 9003 and communicating with a service running at 993. Show the data stored by the NAT in its table.**

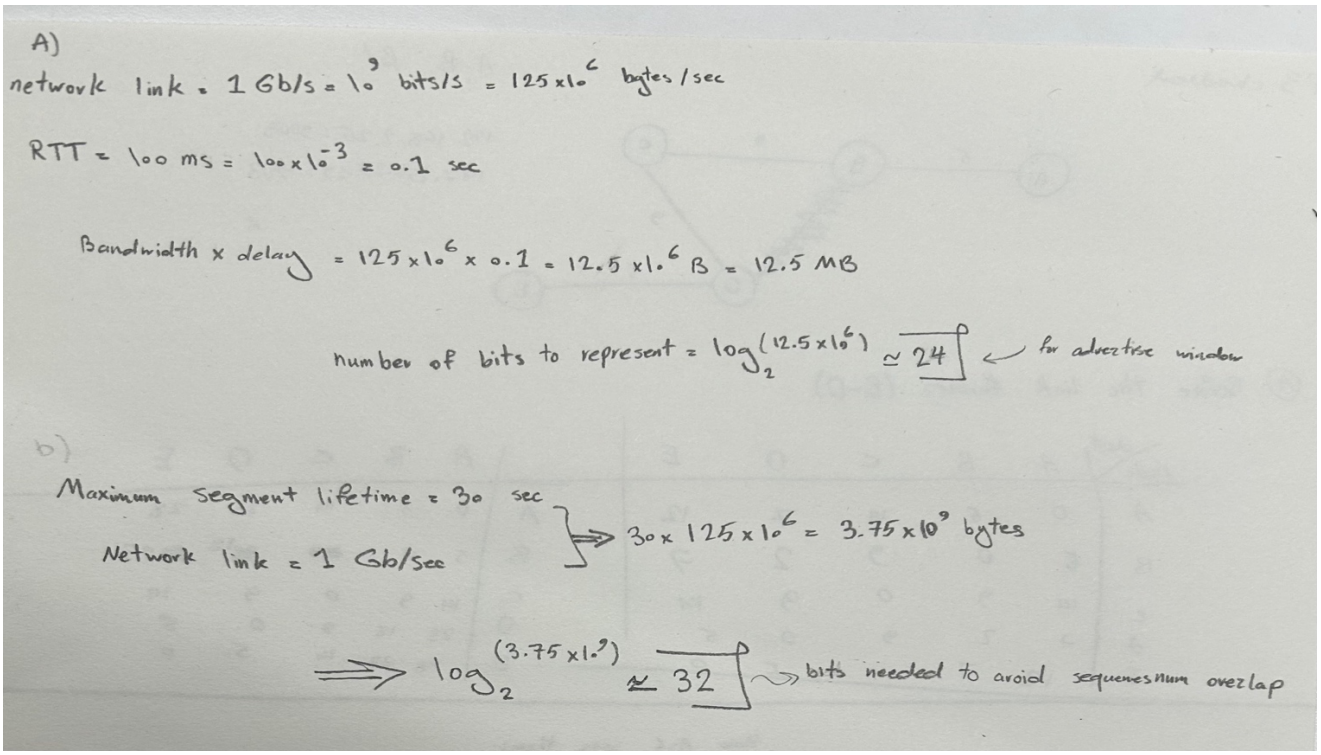
So, we know that the local (LAN) socket is 192.168.2.25:9003 and the destination (public) host is 197.156.25.39 and its port is 993. The Network Address Translation (NAT) has its own public IP address, such as 197.0.113.5, and it must select a port like 8001 to map the internal connection. NAT Translation Table is something like this:

WAN side (NAT public IP, port)	LAN side (private IP, local port)
(197.156.25.39, 993)	(192.168.2.25, 9003)

The NAT keeps track of its connections by associating each table entry with specific details. For instance, it records that an entry is for a connection heading to 197.156.25.39 on port 993. When packets return with the destination address to 197.0.113.5 the NAT knows to forward them back to the internal address (192.168.2.25, 9003).

Problem 6: **You are hired to design a reliable byte-stream protocol that uses a sliding window (like TCP). This protocol will run over a 1-Gbps network. The RTT of the network is 100 ms, and the maximum segment lifetime is 30 seconds.**

(a) **How many bits would you include in the Advertised Window and SequenceNum fields of your protocol header?**



Advertised Window must cover at least the full bandwidth-delay product. Additionally, the Sequence Number must be large enough to avoid wrapping around during the Maximum Segment Lifetime which is 30 seconds in this example.

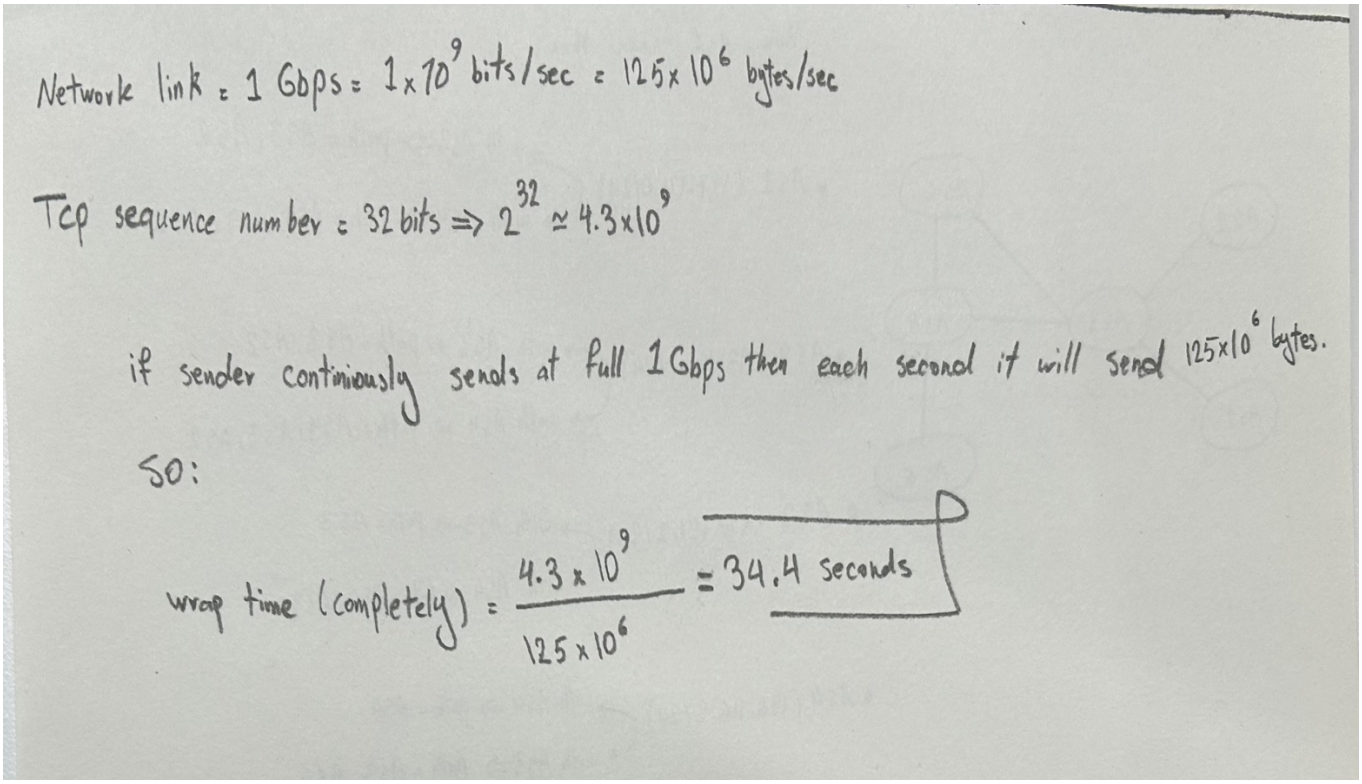
(b) **How would you determine the numbers given above, and which values might be less certain?**

I used the link's raw bit rate, round-trip time (RTT), and maximum segment lifetime to determine the appropriate sizes. The window size must be at least as large as the link's bandwidth-delay product to allow the sender to fully use the channel without waiting for ACK packets.

The sequence number field must be sufficiently large to ensure that during the maximum segment lifetime the same sequence numbers are not reused. This helps prevent confusion between late packets and new ones. In practice, both RTT and actual throughput can vary significantly, and the maximum segment lifetime is more of a design assumption than a fixed parameter.

Problem 7: **Suppose TCP operates over a 1-Gbps link.**

(a) **Assuming TCP could utilize the full bandwidth continuously, how long would it take the sequence numbers to wrap around completely?**



(b) **Suppose an added 32-bit timestamp field increments 1000 times during the wraparound time you found above. How long would it take for the timestamp to wrap around?**

We now add a 32-bit timestamp that increments 1000 times for every single wrap of the sequence space. This means that every 34.36 seconds, the timestamp increases by 1000 counts. The timestamp itself can hold approximately 4.3 billion distinct values ($2^{32} \approx 4.3 \times 10^9$). Therefore, the condition for overflowing the timestamp is:

$$\text{Number of wraps needed} = 2^{32}/1000 \approx 4.3 \times 10^6$$

Each wrap is 34.36 s, so total time is:

$$4.3 \times 10^6 \times 34.4 = 147{,}920{,}000 \approx 1.48 \times 10^8 \text{ sec}$$

Problem 8**: UDP is a transport protocol that does not add much to the best effort service model provided by the network layer. Discuss the reasons for the existence of UDP as a distinct tranport layer protocol (that is, why do we need UDP as a tranport layer protocol despite its minimal functionality beyond the services provided by the network layer)?**

User Datagram Protocol (UDP) is a lightweight protocol that serves several key purposes:

- Multiplexing/Demultiplexing: UDP uses port numbers to allow multiple applications on the same host to receive data independently, enabling proper delivery of packets.
- Low Overhead: Unlike TCP, UDP does not require connection setup or teardown making it ideal for time-sensitive applications like live audio and video that can have some data loss.
- Optional Checksum: UDP includes an optional checksum for error detection, providing a basic level of integrity for the transmitted data.
- Speed: UDP is stateless and simpler to implement, which makes it suitable for tasks like DNS queries where low latency is crucial.
-

Problem 9**: What is the problem that is solved by the Nagle algorithm? Why would we want to disable Nagle under some usage situations? Can you give an example situation where Nagle algorithm is disabled?**

1) The Nagle algorithm was created to prevent the sending of a large number of very small TCP segments. This situation typically occurs when an application generates data in small bursts, such as one byte per send call. If each byte is sent in its own TCP segment, network performance can suffer due to excessive overhead.

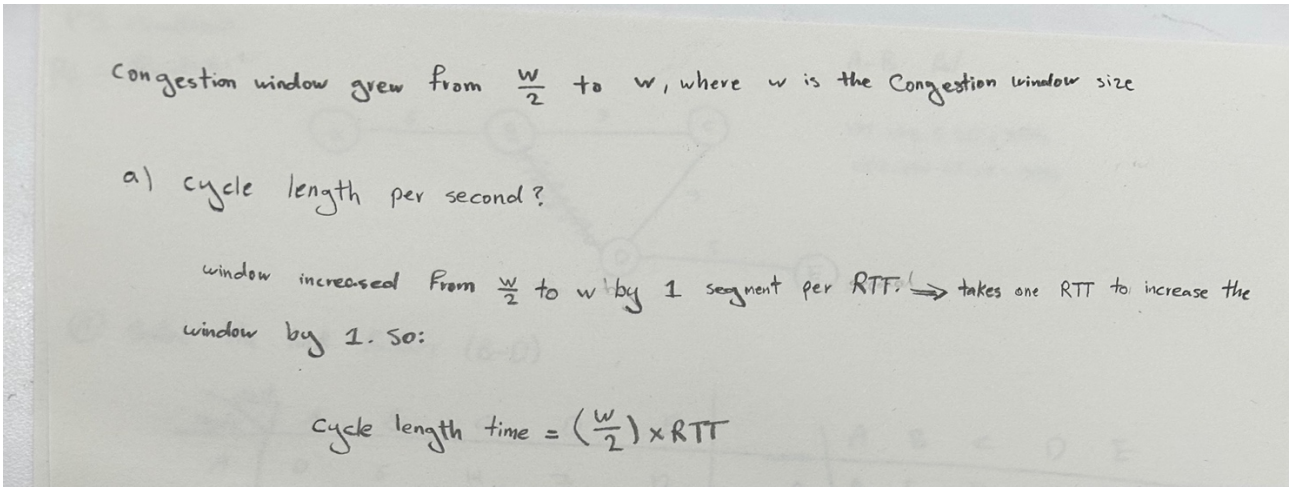
Nagle's solution is to buffer small amounts of outgoing data until either enough data has accumulated to fill a full segment (MSS) or an ACK for previously sent data is received. By combining small writes into fewer, larger TCP segments, this method reduces overhead and can improve throughput.

2) Although Nagle's algorithm helps to reduce overhead, it can introduce significant delays for applications that frequently send small messages and require immediate delivery. In cases where an application needs to transmit small messages quickly such as remote-control commands, low-latency streaming, etc. the delays caused by Nagle's algorithm can negatively affect responsiveness. Disabling Nagle allows each small message to be sent immediately without waiting for additional data or ACKs.

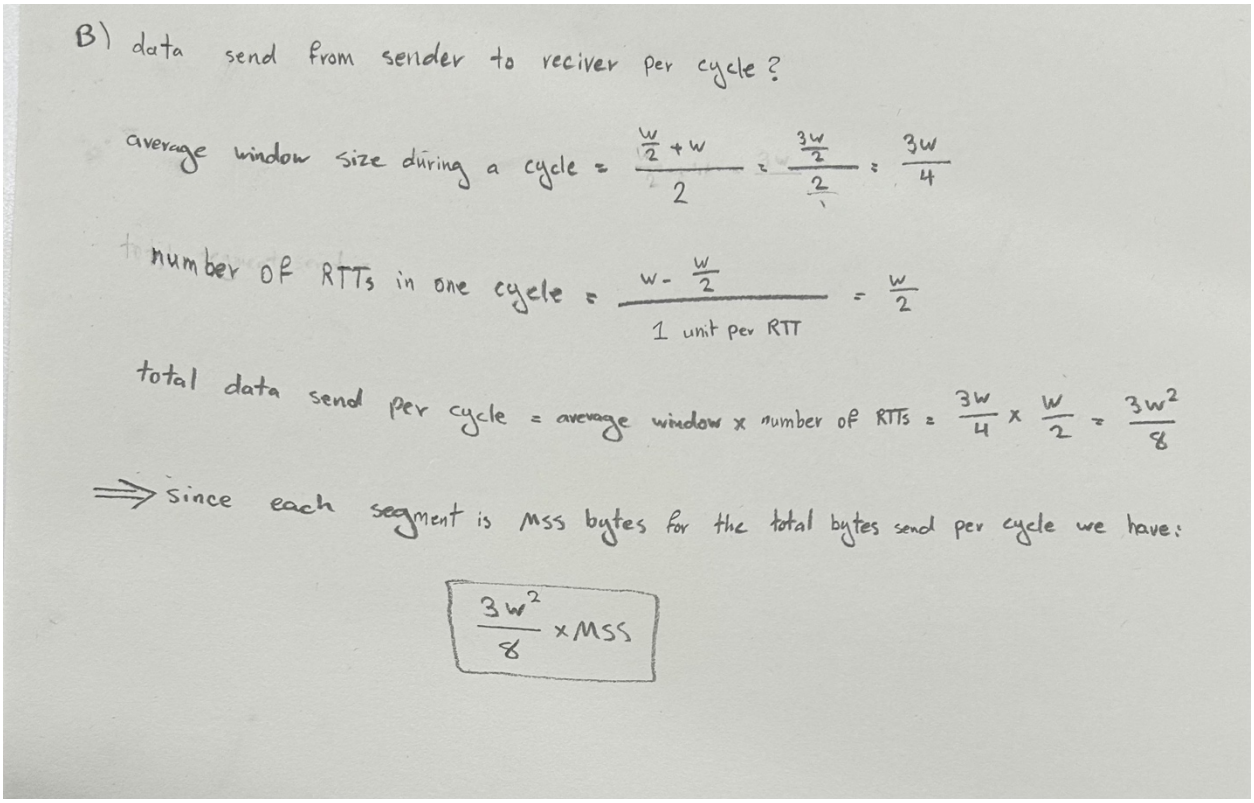
3) A common real-world example is SSH or Telnet sessions for remote consoles some implementations or user preferences disable Nagle to ensure that typed keystrokes are transmitted immediately rather than waiting for potential batching.

Problem 10**: Consider the Additive Increase and Multiplicative Decrease (AIMD) algorithm in TCP. For a TCP session that is ideal (the channel capacity that is available for the session is constant), derive an expression for the following:**

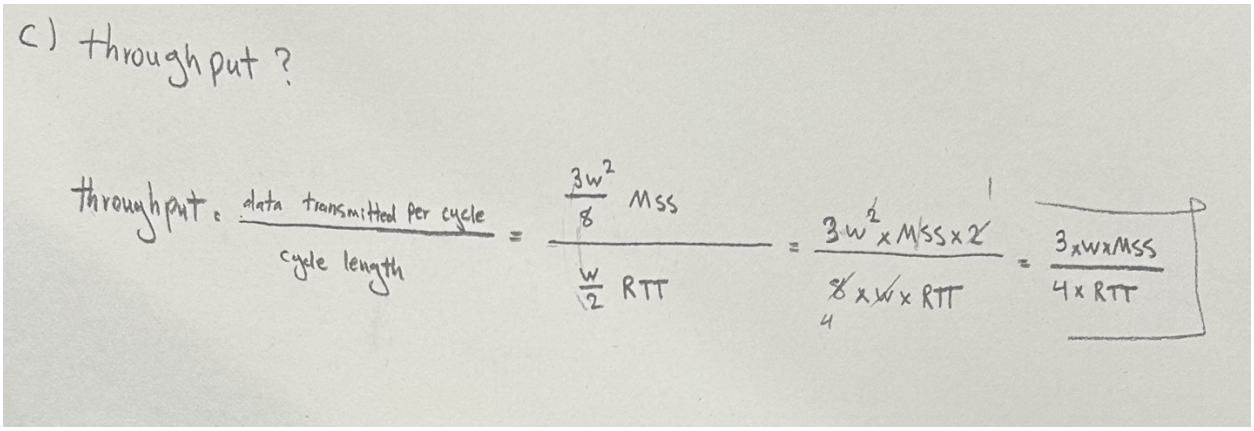
(a) **cycle length in seconds (here a cycle is the duration in the sawtooth waveform between two packet (segment) drops)**



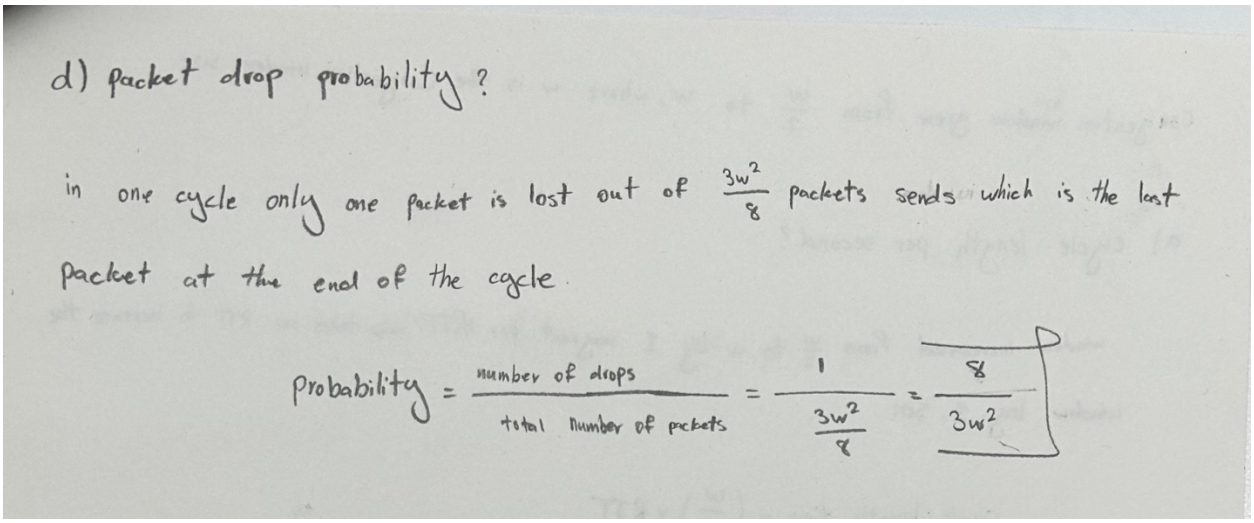
(b) **data transmitted from the sender to the receiver in a cycle in bytes**



(c) **throughput which is equal to the data transmitted per cycle / cycle length**



(d) **packet drop probability. In your derivation use the following notation: W - congestion window at the peak (in segments), MSS - maximum segment size (in bytes), RTT - round-trip time (in seconds), p - probability of packet loss.**

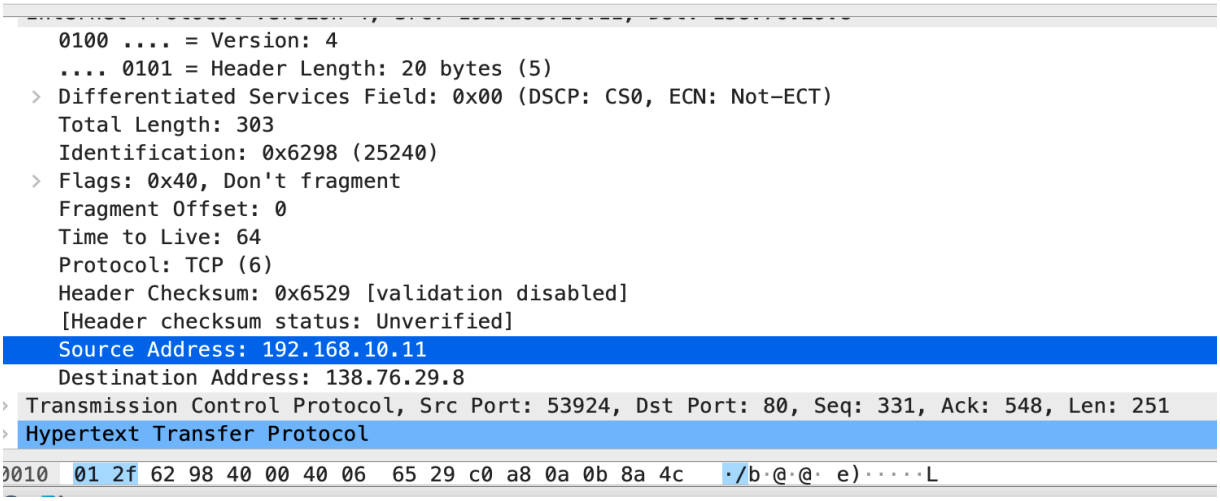


Part 2.1:

1. **What is the IP address of the client that sends the HTTP GET request in the nat-inside-wireshark-trace1-1.pcapng trace? What is the source port number of the TCP segment in this datagram containing the HTTP GET request? What is the destination IP address of this HTTP GET request? What is the destination port number of the TCP segment in this datagram containing the HTTP GET request?**

First, we apply the http.request filter. Then locate the packet where the info column shows GET. After that as shown in the figures, you will find the following information:

- Source IP : 192.168.10.11
- Destination IP : 138.76.29.8
- Source Port : 53924
- Destination Port: 80



```
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 303
    Identification: 0x6298 (25240)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x6529 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.10.11
    Destination Address: 138.76.29.8
> Transmission Control Protocol, Src Port: 53924, Dst Port: 80, Seq: 331, Ack: 548, Len: 251
> Hypertext Transfer Protocol

0010  01 2f 62 98 40 00 40 06  65 29 c0 a8 0a 0b 8a 4c   ·/b·@·@· e)·····L
```

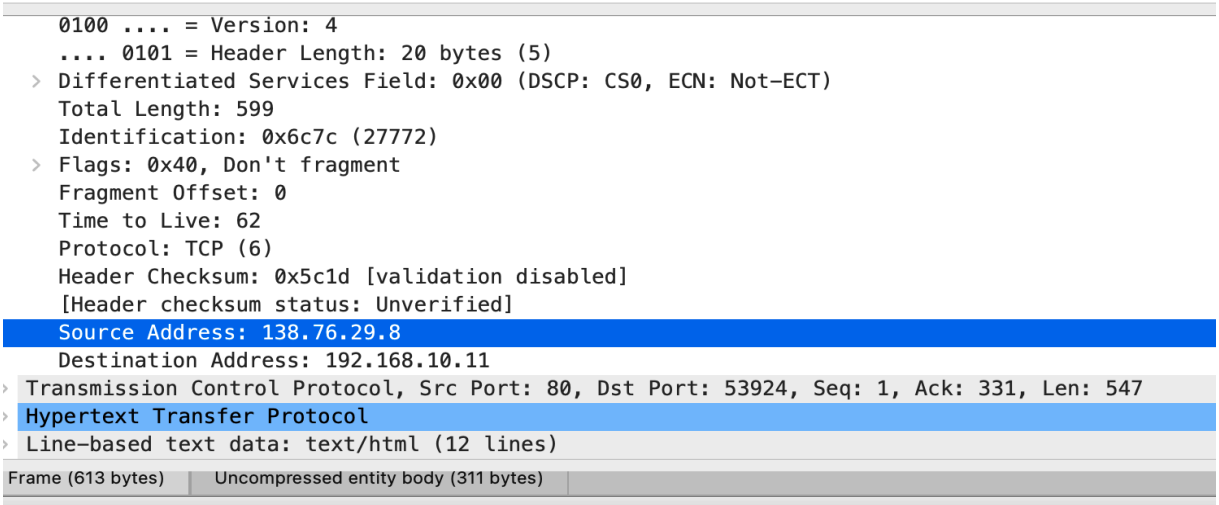
```
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x6529 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.10.11
    Destination Address: 138.76.29.8
> Transmission Control Protocol, Src Port: 53924, Dst Port: 80, Seq: 331, Ack: 548, Len: 251
    Source Port: 53924
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 251]
    Sequence Number: 331    (relative sequence number)
    Sequence Number (raw): 2729790325

0020  1d 08 d2 a4 00 50 a2 b5  4b 75 99 71 bf 31 80 18   ·····P·· Ku·q·1··
```

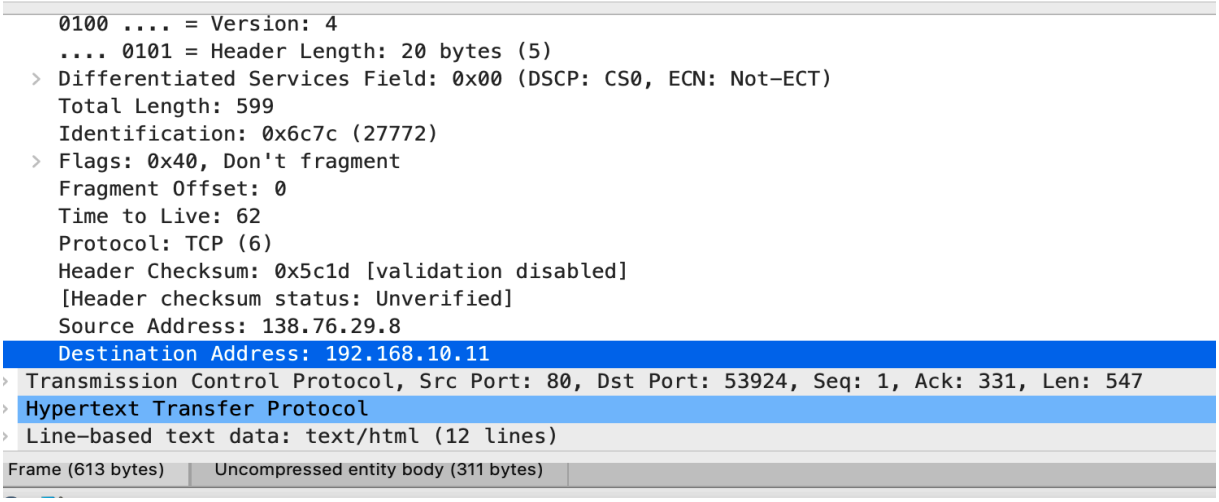
```
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x6529 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.10.11
    Destination Address: 138.76.29.8
> Transmission Control Protocol, Src Port: 53924, Dst Port: 80, Seq: 331, Ack: 548, Len: 251
    Source Port: 53924
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 251]
    Sequence Number: 331    (relative sequence number)
    Sequence Number (raw): 2729790325

0020  1d 08 d2 a4 00 50 a2 b5  4b 75 99 71 bf 31 80 18   ·····P·· Ku·q·1··
```

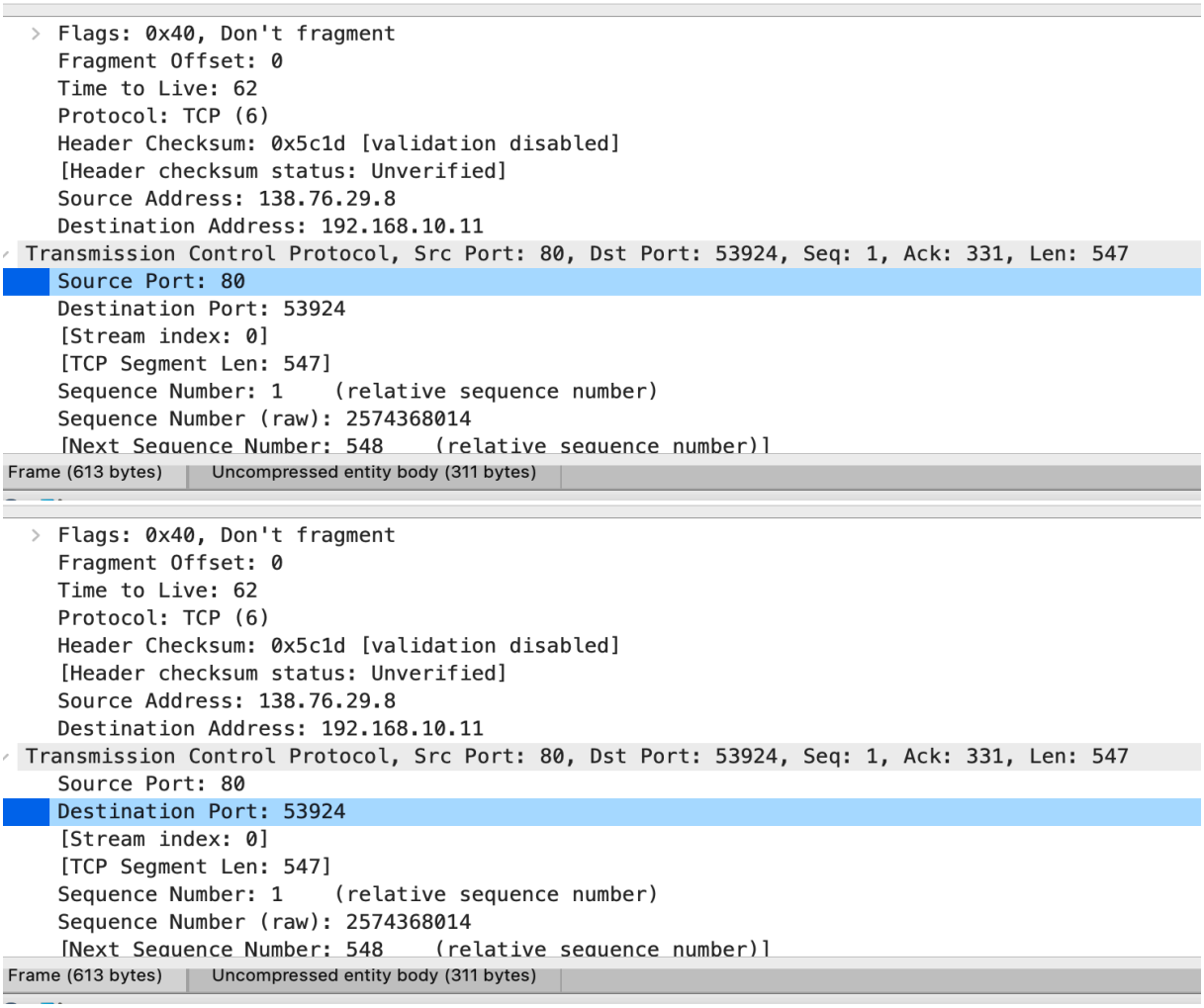
2. **What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying the corresponding HTTP 200 OK message?**

First, we apply the http.response filter. Then locate the packet where the info column shows 200 OK. After that as shown in the figures, you will find the following information:

- Source IP : 138.76.29.8
- Destination IP : 192.168.10.11
- Source Port : 80
- Destination Port : 53924







3. **In the nat-outside-wireshark-trace1-1.pcapng trace file, find the HTTP GET mes- sage that corresponds to the HTTP GET message that was sent from the client to the 138.76.29.8 server at time t=0.27362245, where t=0.27362245 is the time at which this message was sent, as recorded in the nat-inside-wireshark-trace1-1.pcapng trace file.**

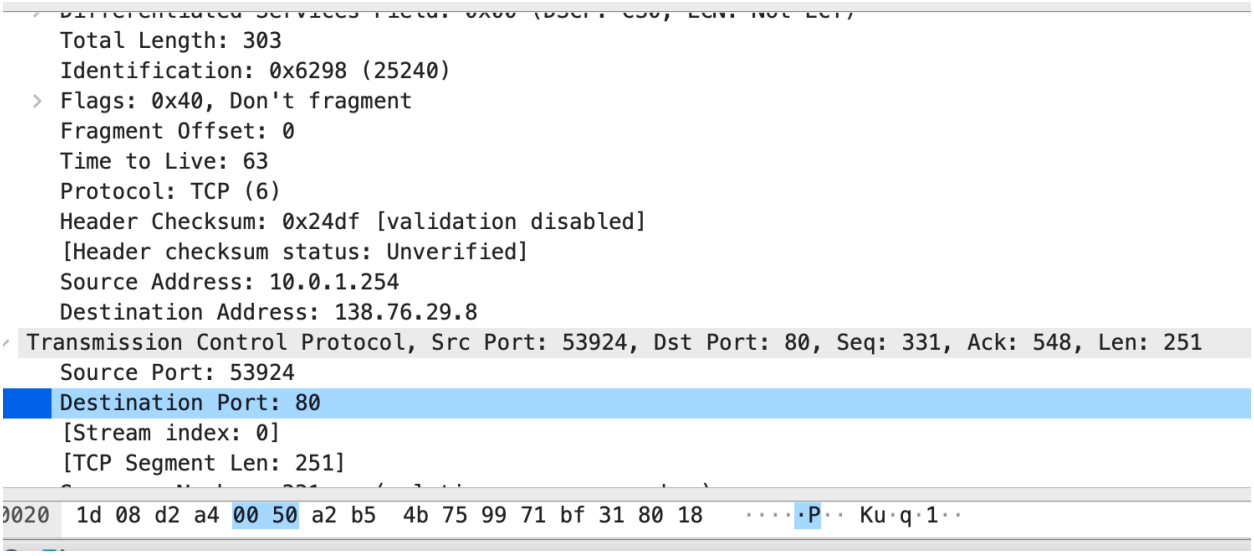
(a) **What are the source and destination IP addresses and TCP source and destination port numbers on the IP datagram carrying this HTTP GET (as recorded in the nat-outside-wireshark-trace1-1.pcapng trace file)?**

- Source IP : 10.0.1.254
- Destination IP : 138.76.29.8
- Source Port : 53924
- Destination Port : 80

```
Internet Protocol Version 4, Src: 10.0.1.254, Dst: 138.76.29.8
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 303
    Identification: 0x6298 (25240)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 63
    Protocol: TCP (6)
    Header Checksum: 0x24df [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.1.254
    Destination Address: 138.76.29.8
Transmission Control Protocol, Src Port: 53924, Dst Port: 80, Seq: 331, Ack: 548, Len: 251
    Source Port: 53924
0010  01 2f 62 98 40 00 3f 06  24 df 0a 00 01 fe 8a 4c   ·/b·@·?· $·····L
```

```
    Total Length: 303
    Identification: 0x6298 (25240)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 63
    Protocol: TCP (6)
    Header Checksum: 0x24df [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.1.254
    Destination Address: 138.76.29.8
Transmission Control Protocol, Src Port: 53924, Dst Port: 80, Seq: 331, Ack: 548, Len: 251
    Source Port: 53924
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 251]
0020  1d 08 d2 a4 00 50 a2 b5  4b 75 99 71 bf 31 80 18   ·····P·· Ku·q·1··
```

```
Internet Protocol Version 4, Src: 10.0.1.254, Dst: 138.76.29.8
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 303
    Identification: 0x6298 (25240)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 63
    Protocol: TCP (6)
    Header Checksum: 0x24df [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.1.254
    Destination Address: 138.76.29.8
Transmission Control Protocol, Src Port: 53924, Dst Port: 80, Seq: 331, Ack: 548, Len: 251
    Source Port: 53924
0010  01 2f 62 98 40 00 3f 06  24 df 0a 00 01 fe 8a 4c   ·/b·@·?· $·····L
```



(b) **Which of these four fields are different than in your answer to Question 1 above?**

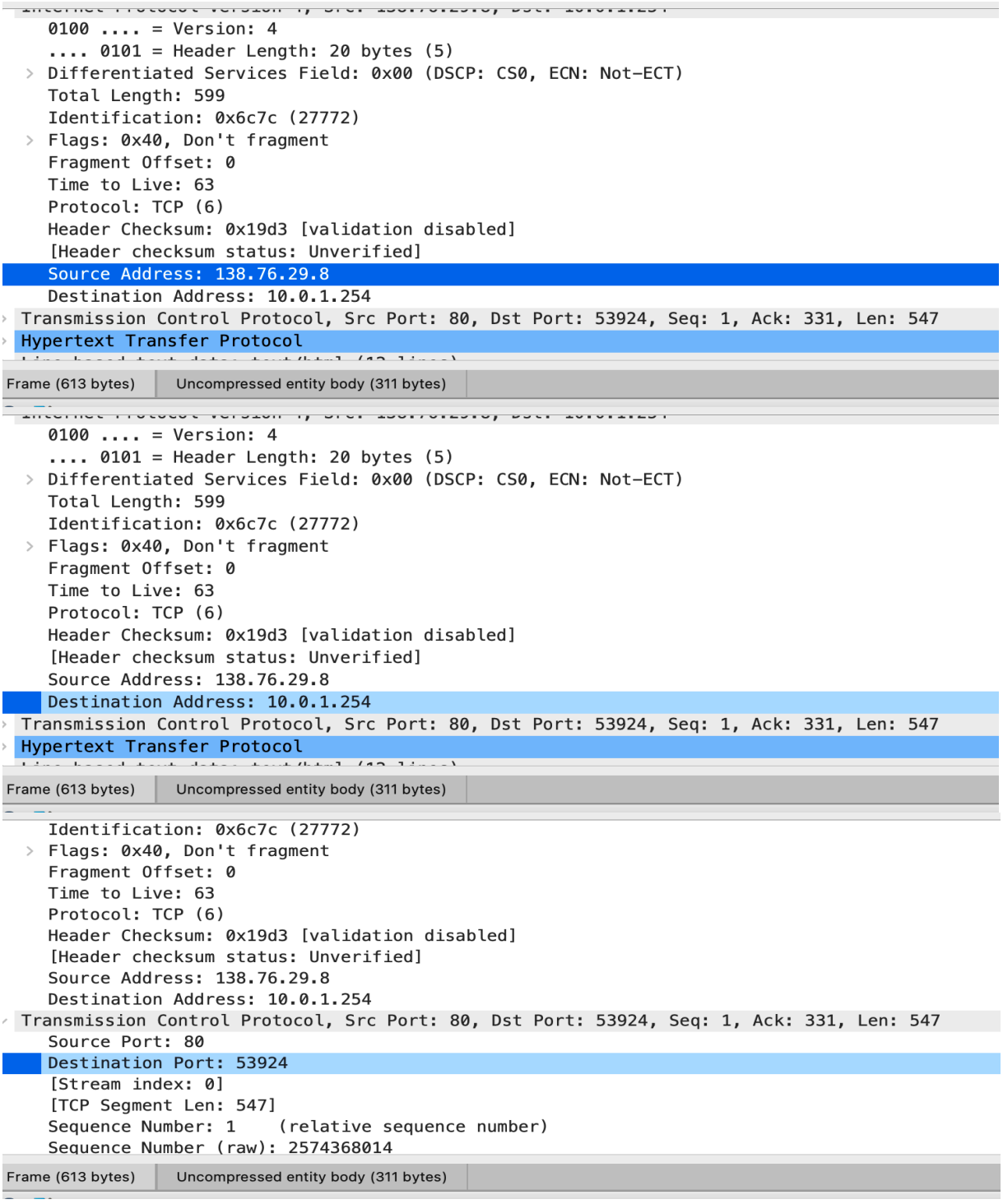
In this example only the Source IP address is different, but in other cases, the source port is also different.

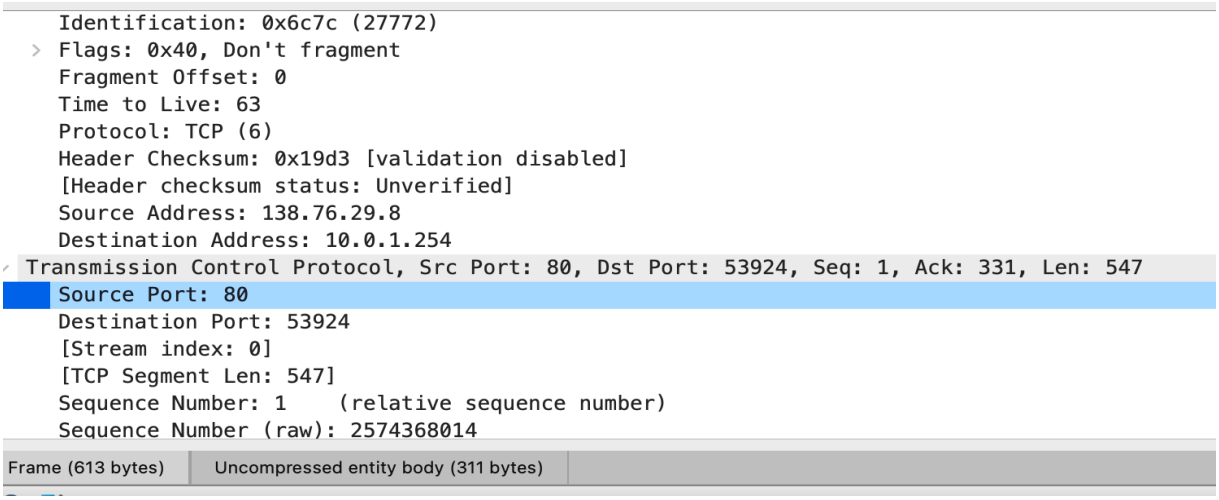
(c) **Which of the following fields in the IP datagram carrying the HTTP GET are changed from the datagram received on the local area network (inside) to the corresponding datagram forwarded on the Internet side (outside) of the NAT router: Version, Header Length, Flags, Checksum?**

Among these fields that are listed in the question only the Checksum varies between the inside and outside IP datagram, while the version, header length, and flags remain the same.

4. **What are the source and destination IP addresses and TCP source and destination port numbers on the IP datagram carrying the "200 OK" message that was received in response to the HTTP GET request you just examined in Question 3 (as recorded in the nat-outside-wireshark-trace1-1.pcapng trace file)?**

- Source IP : 138.76.29.8
- Destination IP : 10.0.1.254
- Source Port : 80
- Destination Port : 53924


0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 599
Identification: 0x6c7c (27772)
Flags: 0x40, Don't fragment
Fragment Offset: 0
Time to Live: 63
Protocol: TCP (6)
Header Checksum: 0x19d3 [validation disabled]
[Header checksum status: Unverified]
Source Address: 138.76.29.8
Destination Address: 10.0.1.254
Transmission Control Protocol, Src Port: 80, Dst Port: 53924, Seq: 1, Ack: 331, Len: 547
Hypertext Transfer Protocol
Frame (613 bytes)
Uncompressed entity body (311 bytes)
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 599
Identification: 0x6c7c (27772)
Flags: 0x40, Don't fragment
Fragment Offset: 0
Time to Live: 63
Protocol: TCP (6)
Header Checksum: 0x19d3 [validation disabled]
[Header checksum status: Unverified]
Source Address: 138.76.29.8
Destination Address: 10.0.1.254
Transmission Control Protocol, Src Port: 80, Dst Port: 53924, Seq: 1, Ack: 331, Len: 547
Hypertext Transfer Protocol
Frame (613 bytes)
Uncompressed entity body (311 bytes)
Identification: 0x6c7c (27772)
Flags: 0x40, Don't fragment
Fragment Offset: 0
Time to Live: 63
Protocol: TCP (6)
Header Checksum: 0x19d3 [validation disabled]
[Header checksum status: Unverified]
Source Address: 138.76.29.8
Destination Address: 10.0.1.254
Transmission Control Protocol, Src Port: 80, Dst Port: 53924, Seq: 1, Ack: 331, Len: 547
Source Port: 80
Destination Port: 53924
[Stream index: 0]
[TCP Segment Len: 547]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 2574368014
Frame (613 bytes)
Uncompressed entity body (311 bytes)

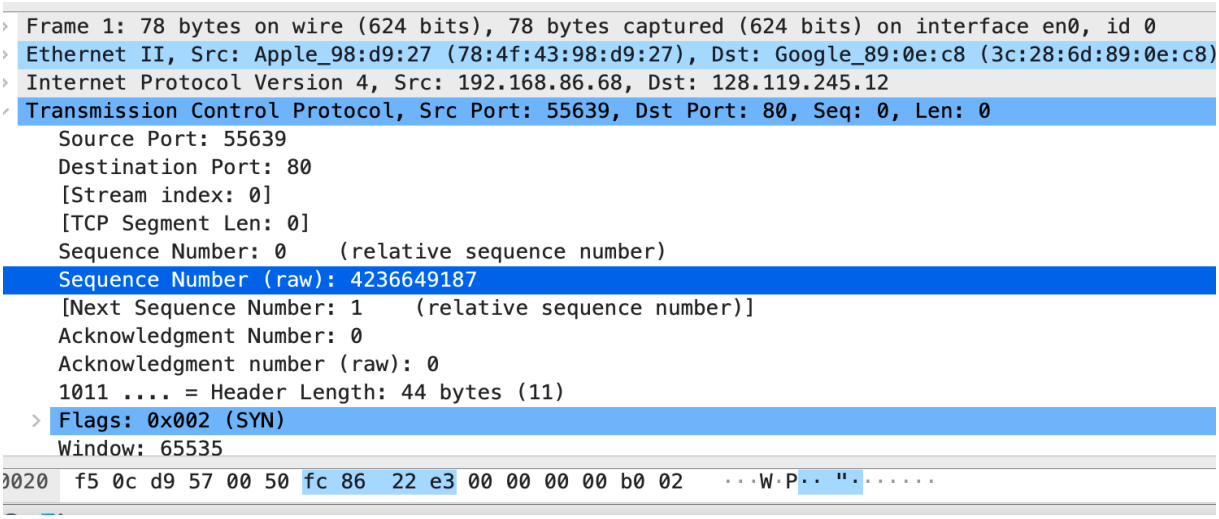


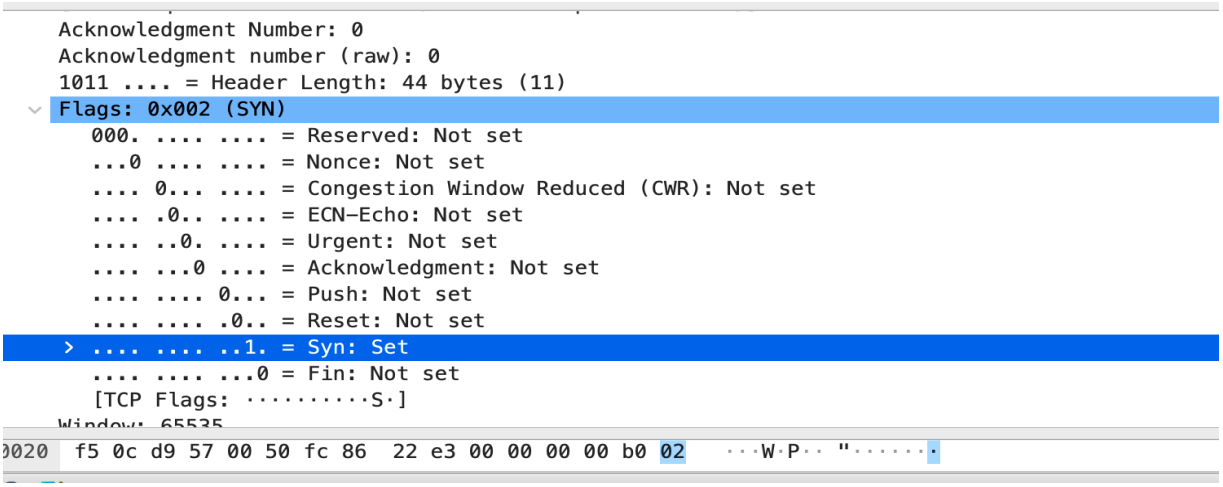
Part 2.2:

1. **What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and the web server? (Note: this is the "raw" sequence number carried in the TCP segment itself; it is NOT the packet # in the "No." column in the Wireshark window. Remember, there is no such thing as a "packet number" in TCP or UDP; as you know, there are sequence numbers in TCP, and that's what we're after here. What is it in this TCP segment that identifies the segment as a SYN segment?**

- The sequence number of the TCP SYN segment: 4236649187

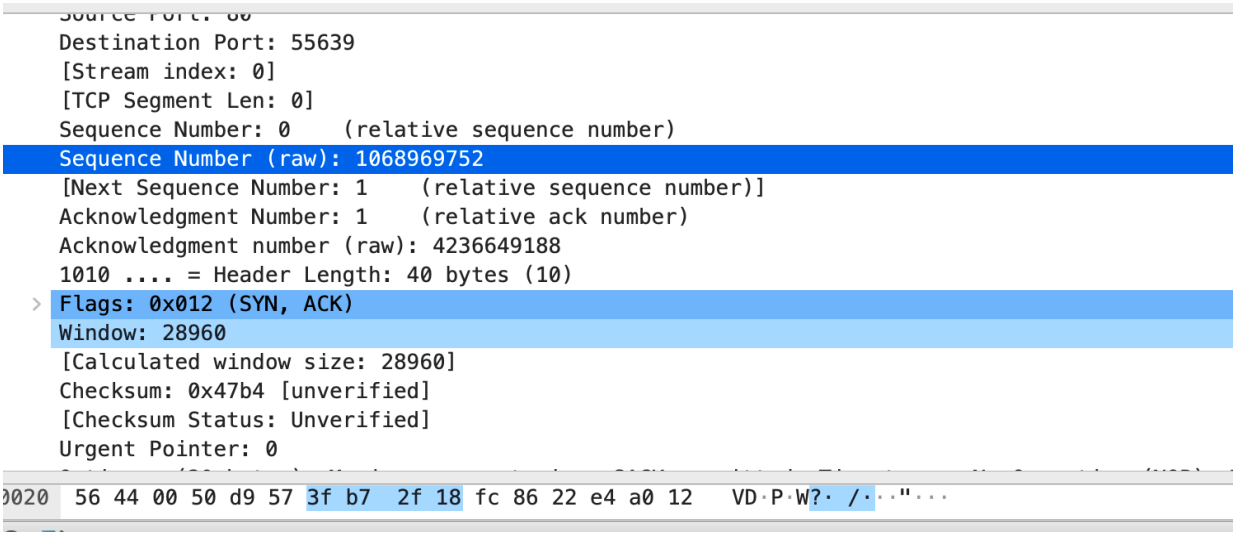
- In the Transmission Control Protocol, under the flag section, the SYN flag is set to 1, which identifies this segment as a SYN segment.

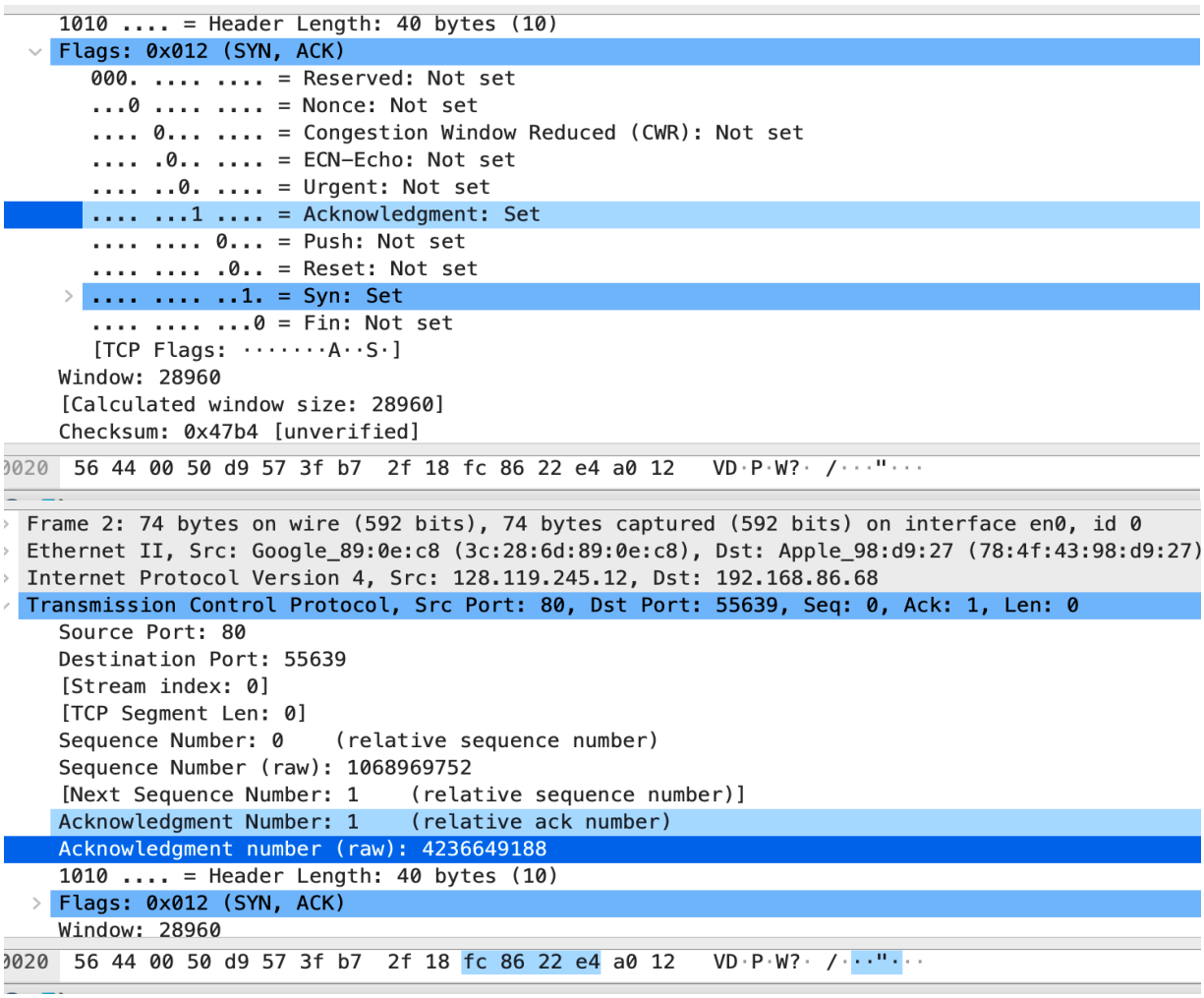




2. **What is the sequence number of the SYNACK segment sent by the web server to the client computer in reply to the SYN? What is it in the segment that identifies the segment as a SYNACK segment? What is the value of the Acknowledgement field in the SYNACK segment? How did the webserver determine that value?**

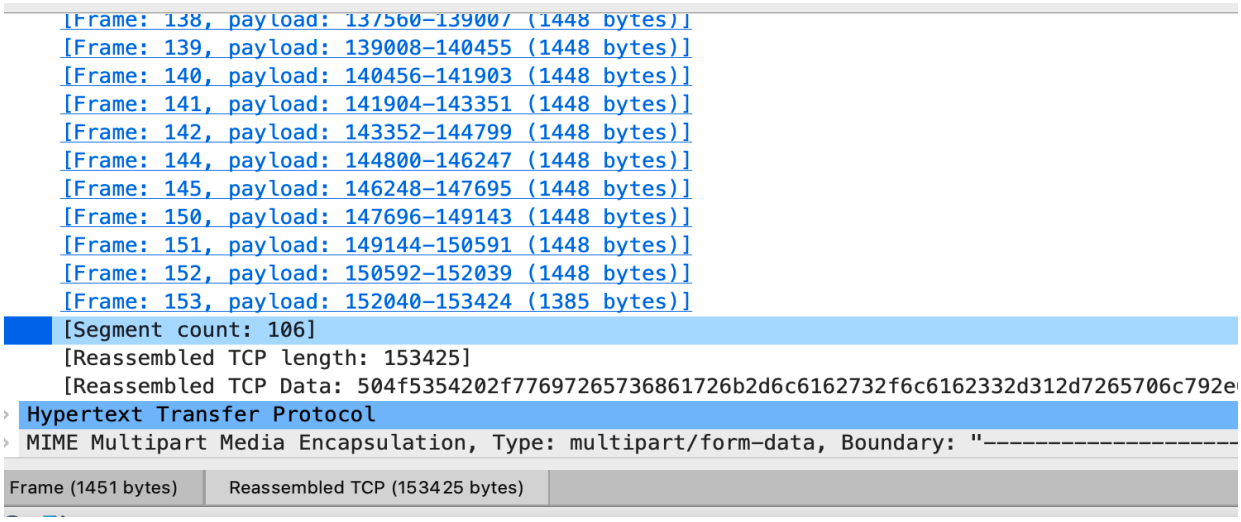
- sequence number of the SYNACK segment: 1068969752
- The combination of bits set in the Flags (SYN + acknowledgment) field indicates that this is a SYNACK segment.
- Acknowledgement field in the SYNACK: 4236649188
- In TCP each side acknowledges the other's sequence number by adding 1 to it. The server realizes the client's initial sequence number in the SYN packet. Since the client has used that sequence number by sending the SYN, the next sequence number the server expects is the client's initial sequence number plus 1 (4236649187 + 1).





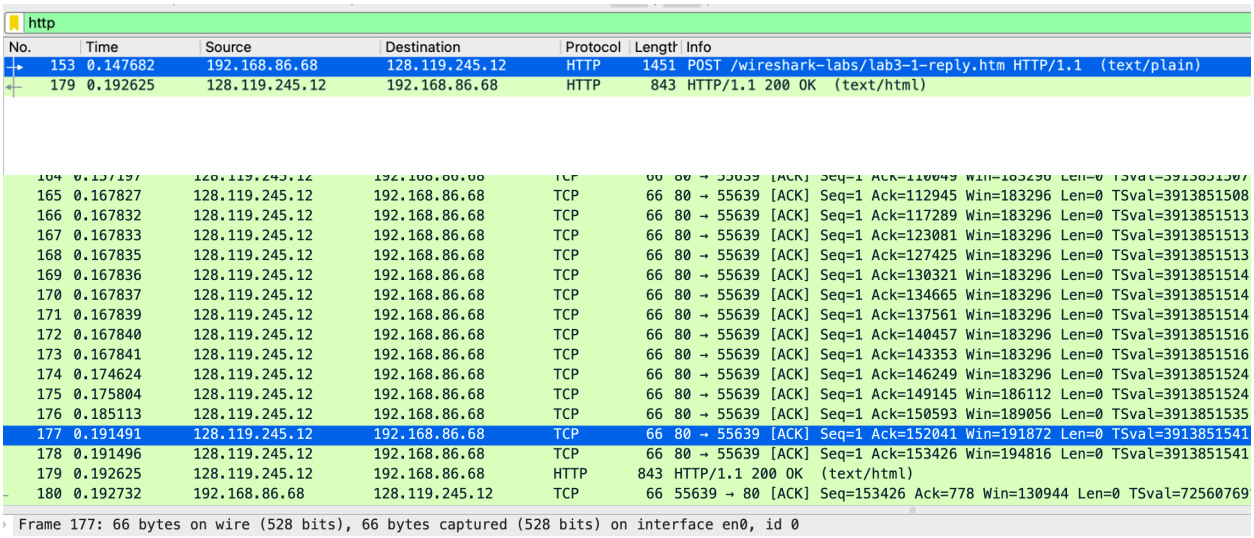
3. **How many TCP segments are used to carry the HTTP post message?**

- TCP segment count for carrying the HTTP Post: 106



4. **At what time was the first TCP segment of the HTTP POST message in the data- transfer part of the TCP connection sent? At what time was the ACK for this first data-containing segment received? What is the RTT for this first data-containing segment?**

- Time for first data segment: 0.147682
- Time for its ACK: 0.191491
- RTT: $T_{ACK} - T_{send} = 0.191491 - 0.147682 = 0.043809$



5. **What is the amount of available buffer space advertised to the client by the web server in the first TCP segment? Hint: Give the Wireshark-reported value for "Window Size Value" which must then be multiplied by the Window Scaling Factor to give the actual number of buffer bytes available at webserver for this connection.**

- Window size: 28960
- Window scaling factor: 7

buffer space advertised to the client by the web server = 28960 × 7 = 202,720 bytes

References:

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