Module 3: Evidence of Learning

Summary of Learning:
Throughout this module, I have deepened my understanding of the transport layer protocols, particularly focusing on TCP (Transmission Control Protocol), and practiced analysing TCP interactions using Wireshark. Here's how I've achieved each of the learning objectives:

Explain the role of transport layer protocols in computer networking:
Through discussions and role plays with my peers, I have learned that transport layer protocols serve as the interface for communication between processes over a network. These protocols define the rules and conventions for data exchange, ensuring compatibility and interoperability. We discussed examples such as TCP and UDP, understanding their importance in facilitating various internet applications.

Demonstrating my understanding of TCP:
By participating in role plays and analysing TCP interactions using Wireshark, I have gained a deeper understanding of how TCP operates. I learned that TCP is a client-server protocol used for establishing reliable connections between hosts. It operates over IP and follows a connection-oriented model, where hosts establish a connection before exchanging data. Analysing TCP segments in Wireshark provided insights into the structure of TCP segments, including details such as sequence numbers, acknowledgment numbers, and control flags.

Evidence of Learning:
I have attached screenshots showcasing our group discussions, and Wireshark analysis, demonstrating my active participation and engagement in the learning process. These screenshots serve as evidence of my understanding of transport layer protocols and TCP. I have also produced the notes I took for this module.

Overall, this module has been instrumental in enhancing my knowledge of computer networking concepts, particularly regarding the role of transport layer protocols and the operation of TCP. Through collaborative learning and practical analysis, I feel better equipped to apply these concepts in real-world scenarios

Notes

Transport vs. Network Layer:
The transport layer provides logical communication between processes, enhancing network layer services like logical communication between hosts.
It's compared to a household analogy where the transport protocol is like parents who multiplex/demultiplex messages to/from their children, and the network-layer protocol is like the postal service.

Internet Transport Protocols:
TCP (Transmission Control Protocol): Offers reliable, in-order delivery with congestion and flow control, and requires connection setup2.
UDP (User Datagram Protocol): Provides unreliable, unordered delivery without congestion control, acting as a no-frills extension of "best-effort" IP.

Multiplexing/Demultiplexing:
Involves handling data from multiple sockets and using header information to deliver received segments to the correct socket.
Connectionless demultiplexing uses the destination port number, while connection-oriented demultiplexing uses a 4-tuple (source/destination IP addresses and port numbers)3.

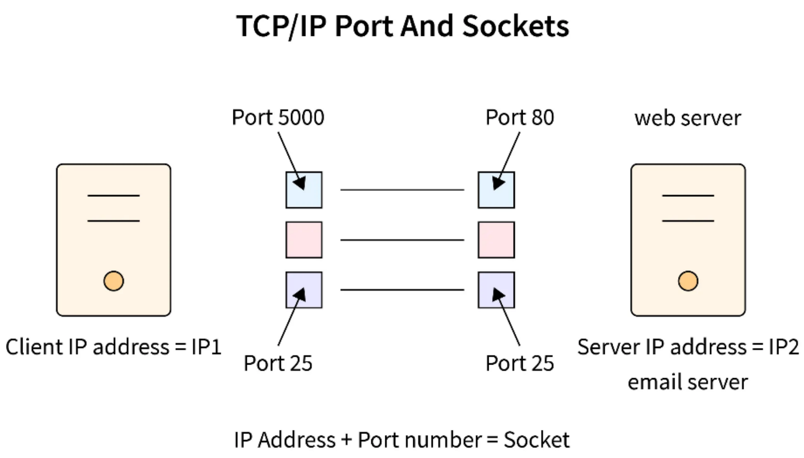
UDP Characteristics:
UDP is used for applications that are loss-tolerant and rate-sensitive, such as streaming multimedia, DNS, and SNMP.
It's a connectionless protocol, meaning there's no handshaking between sender and receiver, and each UDP segment is handled independently.

Sockets are an abstraction used in network programming to represent the endpoint of a communication channel between processes over a network. Sockets allow processes to send and receive data using the transport layer protocols like TCP and UDP.

- Endpoint for Communication: Sockets serve as the endpoint for sending and receiving data packets on a network.
- Process Communication: They enable processes on different hosts or the same host to communicate over a network.
- TCP/UDP Protocols: Sockets can use TCP for reliable, connection-oriented communication or UDP for connectionless communication.
- Multiplexing/Demultiplexing: They allow multiple network connections to be multiplexed over a single physical connection and demultiplex incoming data to the correct process.

Extra resources I used:

- *What is a socket? (The JavaTM Tutorials > Custom Networking > All about Sockets)*. (n.d.). https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html#:~:text=Definition:,another%20program%20on%20the%20network.



Evidence: Module 3 Exercises

You need to provide evidence of successful completion of **all module exercises.**

Exercise 1:

Quiz 03 - Module 3: Transport Layer

Total points 100/100

The respondent's email (**rnirosh134@cicracampus.net**) was recorded on submission of this form.

What are the two main transport layer protocols used in Internet applications? 10/10

◉ TCP & UDP

○ UDP & DNS

○ DNS & HTTP

○ TCP & HTTP

Exercise 2:

Transport layer protocols provide logical communication between different hosts in the network. 10/10

○ True

◉ False

Exercise 3:

Why could be the reason for an application developer to choose UDP over TCP for the network application that he/she is developing? 10/10

○ UDP provides fast and reliable data transfer

◉ Application does not require reliable data transfer and would like to avoid TCP congestion control

○ UDP is much better than TCP

○ Application developer prefers UDP over TCP

Exercise 4:

A network application can still implement reliable data transfer even when the application runs over UDP by introducing reliable data transfer into application layer protocol. Is the above statement true or false? 10/10

- (x) True
- ( ) False

Exercise 5:

Suppose we have a network application that runs over UDP. Assume that an application process in Host A has a UDP socket with port number 10541 and two other hosts (Hosts X and Y) send UDP segments to Host A with the destination port number 10541. What would be happening at Host A next? 10/10

- ( ) Segments that Host X and Y sent to Host A will be directed to the same socket at HostA and the application process will sort the messages later.
- (x) Segments that Host X and Y sent to Host A will be directed to the same socket at Host A and at the socket interface, operating system will use destination IP address to identify the origin of the segment.
- ( ) All the segments will direct to a buffer and the application process will pull the required data from the buffer.
- ( ) Segments that Host X and Y sent to Host A will be directed to two different sockets and hence two application process.

Exercise 6:

Which of the following statements are correct? 10/10

- [ ] Your browser is uploading a large file to a server over a TCP connection. The server just received a segment with sequence number x and y bytes. The acknowledgement number of the acknowledgement that sever will send to your browser is x+1.
- [ ] Your browser is uploading a large file to a server over a TCP connection. The server just received a segment with sequence number x and y bytes. The sequence number of the subsequent segment that server will be received is x+1, if there is no packet drop in the network.
- [ ] Once the TCP "receive window" size is decided at the connection establishment, the value of receive window will remain the same throughout the duration of the TCP connection.
- [ ] Your browser is uploading a large file to a server over a TCP connection. The server do not have any data to send to your browser. Therefore, server will not send acknowledgements to your browser because the server cannot piggyback the acknowledgments on data.
- [x] None

Exercise 7:

Host X sends two TCP segments to Host Y. The first segment has sequence number 1001 and the second segment has sequence number 1010. However, the first segment is lost during the transmission but the second segment arrives at Host Y. What will be the acknowledgement number that Host Y will be using when it sends the acknowledgement to Host X. 10/10

- ( ) 1010
- ( ) 1002
- ( ) 1011
- (•) 1001

Exercise 8:

Which of the following statements are correct? Please note that there could be more than one correct statement. 10/10

- [x] TCP flow control is implemented at the sender with the use of receive window variable set by the receiver.
- [x] Congestion window variable that is maintained by the sender is used for TCP congestion control.
- [ ] TCP flow control is implemented at the sender with the use of receive window variable set by the sender.
- [ ] TCP congestion control have 4 different phases and it starts with congestion avoidance.

Exercise 9:

Suppose Host A is sending a large file of X Bytes to Host B. What is the maximum value of X that we can have such that TCP sequence numbers are not exhausted? Assume TCP has MSS of 500 Bytes and receive window size of 1000 Bytes. Remeber TCP sequence number field has 4 bytes (32 bits). 10/10

- ( ) 8589934 Bytes
- (•) 4,294,967,296 Bytes
- ( ) 4,294,967 Bytes
- ( ) 4,294,967,000 Bytes

Exercise 10:

We use checksum in both TCP and UDP to detect errors. Assume you have the following 2 bytes: 11011100 and 01100111. What would be the 1s complement of the addition of the above two bytes? Make sure that you wraparound any carried over bits. 10/10

10111011

Evidence: Active Classes

Active class 4: UDP - Unreliable Data Protocol?

This activity was about the User Datagram Protocol (UDP) in the transport layer. There were group discussions, analysing UDP packets using Wireshark, and building a client-server application using python.

Activity 1:
We've learned the importance of the transport layer and the fundamental differences between UDP and TCP2. Through group discussions and practical exercises, I've grasped the scenarios where UDP's speed and efficiency outweigh TCP's reliability.

\# **active-class-group-activity** Active class 4: UDP - Unreliable Data Protocol? Search

H44mid Today at 8:43 PM
a. When there's a time crunch to pack everything in the kitchen swiftly before the moving truck arrives, it's akin to employing a quick packing method, reminiscent of UDP (User Datagram Protocol) in networking. The focus here is on speed over meticulousness, mirroring UDP's emphasis on rapid data transmission without ensuring perfect order or reliability. Items are grouped and packed swiftly, prioritizing efficiency over careful organization, much like how UDP prioritizes speed in transmitting data packets.

b. In the scenario where safety, organization, and maintaining the original order of items are crucial, it reflects the meticulous packing method akin to TCP (Transmission Control Protocol). Much like TCP ensures reliable data delivery by establishing connections, acknowledging received packets, and retransmitting lost ones, the meticulous packing method emphasizes careful handling and labeling to ensure items reach their destination safely and in their original order. Each item is packed with care, and boxes are labeled methodically, similar to TCP's focus on ensuring packets arrive reliably and in sequence.

c. When considering large file transfers and faster communication needs, TCP and UDP each have their roles. TCP is ideal for large file transfers due to its reliability, ensuring all parts of the file reach their destination intact. Conversely, UDP might be chosen for faster communication needs where speed is prioritized over reliability, such as real-time streaming or online gaming. UDP sacrifices reliability for speed, similar to how a quick packing method sacrifices meticulousness for speed, making it suitable for scenarios where occasional packet loss is acceptable, and minimizing latency is crucial. (edited)
💯 3

\# **active-class-group-activity** Active class 4: UDP - Unreliable Data Protocol? Search

Baxy Today at 8:35 PM
Hey there, fellow network warriors! ⚔️ Today's active class is all about UDP, the transport protocol 🚙 that's like your friend who packs for a move. You know, the one who throws everything in a box with reckless abandon, hoping for the best?

So, can you guess which scenario relates to the lightning-fast, "get it there quick" UDP and the reliable, "everything in its place" TCP?

Before we dive into the nitty-gritty of analyzing UDP packets 📦 and building a crazy client-server application 👫 (because who says network learning can't be fun?), let's chat 🔊 about these two transport protocols and see which one best suits different situations. Let the brainstorming begin! 🧠
🔥 2

Activity 1

Group discussion:
Assume that you are helping your friend to move his/her house and your responsibility is to take care of moving goods in the kitchen.

a. What method would you take if your friend needs to pack everything within an hour because the moving truck will be there soon?

b. How would your method change if you have more time and your friend would like to make sure each item in the kitchen is safely delivered, and your friend wants to keep those on the same order it was on the previous house?

c. Now, can you relate the above scenarios to the well-known two transport layer protocols? Consider different scenarios including large file transfer and faster communication. (edited)

active-class-group-activity | Active class 4: UDP - Unreliable Data Protocol?

ipiot Today at 8:44 PM

A. First i would pack the essential items first and i would get the fragile items first and wrap them for safety

B. I would first sort the kitchen items by category, and wrapping the fragile items individually. i would use secure containers to protect the liquid items.

C. Large File Transfer: A protocol like UDP (User Datagram Protocol) would be comparable if your friend had to pack everything in one hour. Although UDP allows for speedier communication, dependable delivery cannot be guaranteed. In a same vein, packaging that prioritises speed and efficiency mirrors the UDP's preference for speed above dependability.

Faster Communication: A protocol like TCP (Transmission Control Protocol) would be essential in the case when your friend wants to guarantee the safe delivery of each item in the kitchen while keeping things organised. Data delivery is dependable and well-organized thanks to TCP. In a similar vein, TCP's attention to detail and dependability are demonstrated by the time it takes to sift, bundle, and keep kitchen supplies organised.

💯 2 🔥 2

active-class-group-activity | Active class 4: UDP - Unreliable Data Protocol?

Baxy Today at 8:45 PM

A Part:

Your friend needs to be out in an hour? Forget carefully wrapping grandma's china! It's all about speed! Here's the UDP method:

--> Just like UDP doesn't care much about establishing a connection, we don't waste time finding the perfect box. We grab the first one we see.
--> Pots, pans, spatulas – everything goes in with reckless abandon. Similar to UDP packets, there's no guarantee everything will arrive, but hey, we got it in the box (hopefully)!
--> We don't bother labeling boxes or checking if everything made it. We just shove it on the moving truck and hope for the best. Just like UDP, there's no confirmation if things arrived or not.

B Part:

Now, if there's time and your friend wants their kitchen to arrive intact, it's TCP time! Here's how we roll:

--> Bubble wrap, packing peanuts, sturdy boxes – we've got it all. TCP establishes a connection first, just like we plan and gather everything we need.
--> Each pot and pan gets the royal treatment – careful wrapping to ensure safe delivery. TCP guarantees order and retransmission if something gets lost, just like each item gets its rightful protection.
--> Boxes are meticulously labeled with contents and placement in the new house. TCP ensures everything arrives in the order it was sent, just like we make sure boxes go in the right room.

active-class-group-activity | Active class 4: UDP - Unreliable Data Protocol?

C Part:

Relating it to Network Protocols:
Now, the connection becomes clear!

--> UDP: The "get it there fast" protocol, perfect for quick exchanges like real-time video or online gaming where a little lag might be tolerable.

--> TCP: The "reliable delivery" protocol, ideal for large file transfers or situations where order and data integrity are crucial. Imagine downloading a giant movie – you wouldn't want missing chunks that mess up the whole thing, right?

So, just like packing for a move, the choice between UDP and TCP depends on your priorities: speed or guaranteed delivery. (edited)

🔥 3 💯 3

Amjad Today at 8:50 PM

a. When there's a rush to pack everything quickly, it's like using a rapid packing method similar to UDP (User Datagram Protocol) in networking, prioritizing speed over meticulousness.

b. For safety and organization, it resembles TCP (Transmission Control Protocol), ensuring items are carefully packed and labeled to maintain their order, just like TCP ensures reliable data delivery.

c. TCP is preferred for large file transfers due to its reliability, while UDP might be chosen for faster communication needs, sacrificing reliability for speed, similar to how quick packing prioritizes speed over meticulousness.

🔥 3 💯 3

active-class-group-activity | Active class 4: UDP - Unreliable Data Protocol?

mabrook Today at 8:54 PM

a. In a rushed packing scenario, the focus is on getting everything packed within the hour before the moving truck arrives, without much attention to detail or organization.
UDP is all about getting data from one place to another quickly, even if there's a risk of some packets (or items) being lost or damaged in transit.

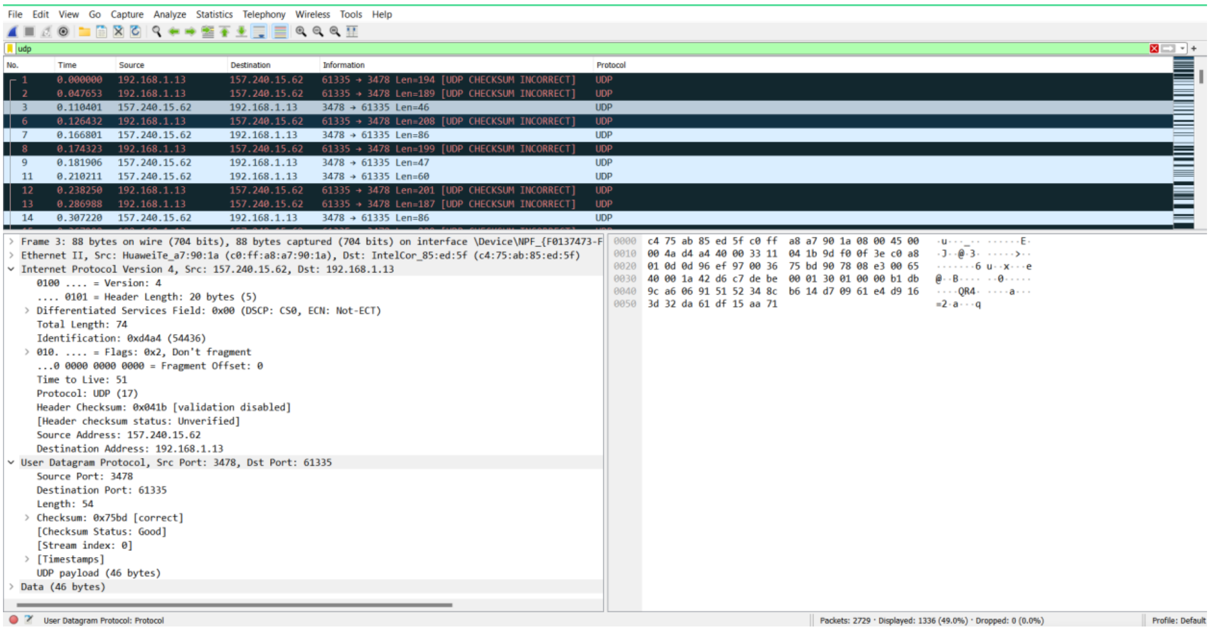
b. When there's more time and the goal is to ensure each item in the kitchen is safely delivered and maintains its order, it resembles TCP (Transmission Control Protocol).TCP ensures reliable data delivery by carefully organizing and labeling packets, just like carefully packing and labeling items in the kitchen.

c. UDP might be chosen for faster communication needs, sacrificing reliability for speed. Similarly, in a rushed packing scenario, a rapid UDP-style packing method might be employed, prioritizing speed over meticulousness. (edited)

👍 3 💯 3

Activity 2:
By using Wireshark to capture and analyse UDP packets, I've become familiar with the UDP packet structure. This hands-on activity helped me understand the significance of each field in the UDP header and the implications of the Length field.

This is the screenshot of the packet capture using Wireshark



a. Pick one UDP packet from the filtered packets. Examine the number of fields that are in the header of the selected UDP packet. You can take a screenshot of the packet and name and explain the fields in the UDP packet.

   The UDP header contains four fields. These fields are:

- **Source Port**: The port number of the application that sent the datagram.
- **Destination Port:** The port number of the application that will receive the datagram.
- **Length:** The length of the UDP header and the data in bytes.
- **Checksum**: A 16-bit checksum used for error checking.

b. Can you identify the length of each UDP header field from this UDP packet? You can indicate the length in bytes. You may want to examine the displayed information in Wireshark's packet content field.

   Lengths of each UDP header field in bytes:
   Source Port: 2 bytes
   Destination Port: 2 bytes
   Length: 2 bytes
   Checksum: 2 bytes

   The total length of the UDP header is therefore always fixed at **8 bytes**.

c. Can you explain the value in the Length field? What exactly this value indicates? You can verify your answer by examining captured UDP packets.

The value in the Length field indicates the total length of the UDP header and the UDP data, combined, in bytes. In my Wireshark screenshot, this field is indicated as "Len".

For example, in the first UDP packet listed, the value in the "Len" field is "54". This means that the total length of the UDP header (8 bytes) plus the length of the UDP data (UDP payload) in that packet is 54 bytes.

Let me break it down for easy understanding:

Total Length = UDP header + UDP data
Total Length = 54 bytes
UDP header length = 8 bytes (fixed value)
UDP data length = Total Length - UDP header length
UDP data length = 54 bytes - 8 bytes = 46 bytes
Therefore, in this specific packet, the UDP data itself is 46 bytes long.

d. Is there a maximum number of bytes that we can include in UDP payload?

Yes, there is a limit for that, UDP datagrams are encapsulated within IP packets for transmission on a network. The maximum theoretical size of an IP packet is 65,535 bytes, limited by the 16-bit field size in the IP header that specifies the total length of the IP packet. However, the largest possible UDP payload is further limited by the Maximum Transmission Unit (MTU) of the network path. The MTU is the largest size packet that can be transmitted on a specific network segment.

e. Can you identify the protocol number given for UDP?

Yes, the protocol number given for UDP in the image is 17.

This can be identified from the "Protocol" field in the first line of the capture. It shows "UDP" next to the decimal number "17".

f. Examine two consecutive UDP packets your host sends/receives. Explain the relationship between the port numbers in these two packets.

both packets actually do share the same source and destination ports:

Source Port: 3478
Destination Port: 61335

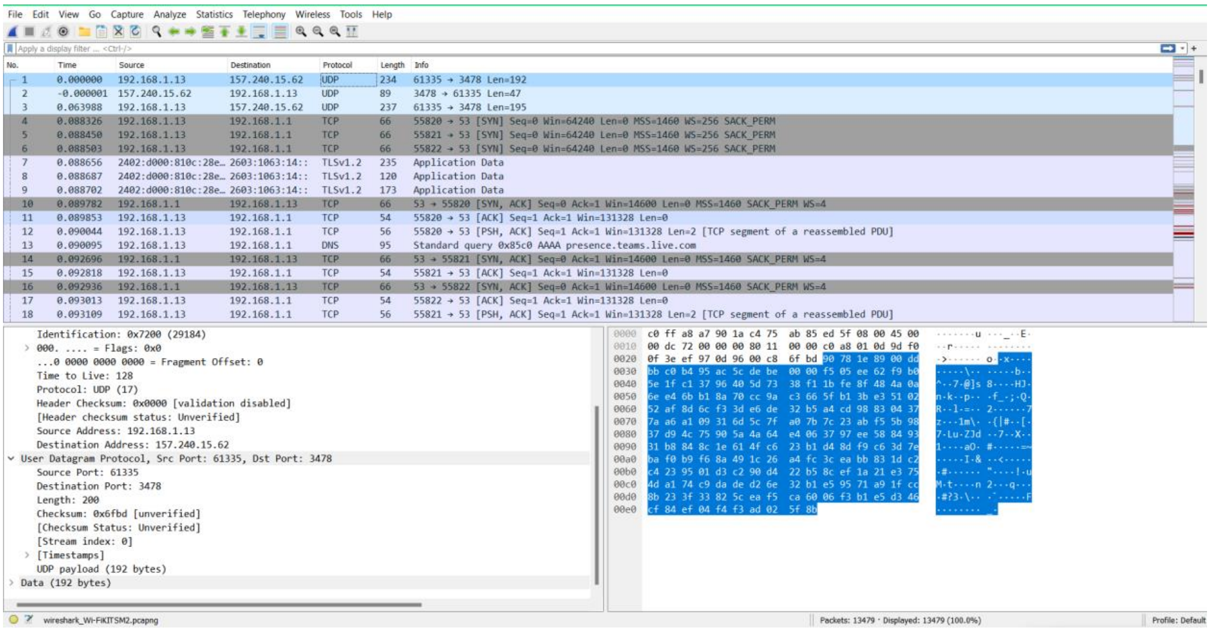
This indicates that these packets are likely part of the same communication flow, possibly between your host and another machine on the network.

Source Port (3478) is likely the port chosen by the host's application to initiate the communication with the other machine.

Destination Port (61335) is assigned to the application on the other machine that the host is communicating with. It's possible this specific port is well-known for a particular service, or it could also be a dynamically chosen port on the remote machine.

g. Clear the cache and run another application such as MSTeams while using Web browsing. Capture packets of these two applications. Now you can analyse the captured UDP packets again. Do all UDP packets captured in Wireshark use the same port number? Explain your answer.

Screenshot:



No, not all UDP packets captured in Wireshark will use the same port number, as shown when comparing the two images you sent.

Screenshot 1:

This screenshot shows packets captured during web browsing activity.

The displayed UDP packets use various source and destination port numbers. for instance:
source port: 3478
Destination port: 61335

Screenshot 2:

This screenshot shows packets captured during web browsing and Microsoft Teams usage.

Similar to the first screenshot, various source and destination port numbers are used in the displayed UDP packets. like:
source port: 61335 (likely MS teams)
destination port: 3478

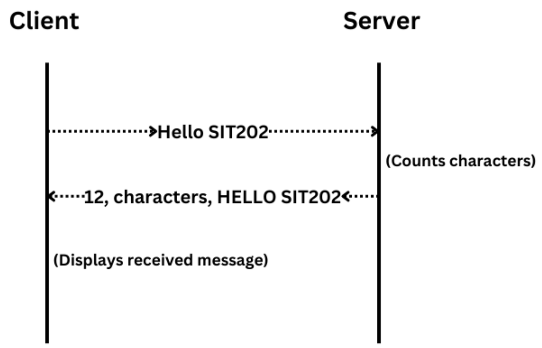
Explanation:

UDP is a connectionless protocol, meaning each datagram (packet) is treated independently. Each UDP communication session is identified by the combination of source and destination IP addresses and port numbers. Applications can choose any available port number when initiating UDP communication. This is why we see various port numbers used in both screenshots.

Even though both screenshots likely involve web browsing activity, the source and destination ports used can differ based on the specific web traffic and the servers involved. In addition, the introduction of MS Teams in the second screenshot adds another application using UDP communication, further contributing to the variation in port numbers observed.

Activity 3:
The creation of a simple client-server application using Python was a challenging yet rewarding task3. It not only solidified my programming skills but also provided a clear demonstration of UDP's operation in real-time communication.

1. First you need to draw a diagram to explain the operation of the client- server program that uses UDP.



- The client sends a message (Hello SIT202) to the server using UDP.
- The server receives the message and counts the number of characters in the message.
- The server sends a response back to the client, which includes the number of characters counted and the original message in uppercase letters (12, HELLO SIT202).
- The client receives the response from the server and displays the received message on its terminal.

2. Then one of you can develop the client side and the other can develop the server side of the program (you can also develop both end systems and demonstrate the outcome if you chose to do so)

Server code:

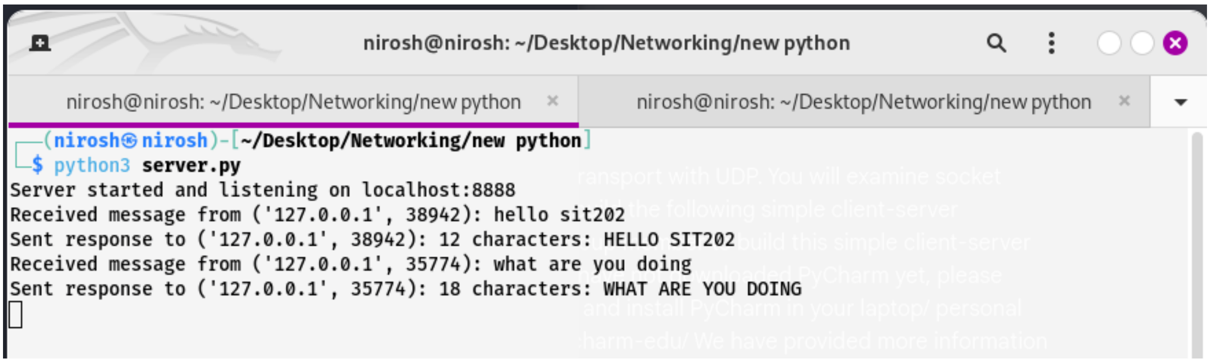
```
import socket

# Server IP address and port
HOST = "localhost"
PORT = 8888  # Using the same port as the provided code

# Create a UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the IP address and port
server_socket.bind((HOST, PORT))

print(f"Server started and listening on {HOST}:{PORT}")

while True:
    try:
        # Receive data from a client (message)
        data, client_address = server_socket.recvfrom(4096)  # Adjust buffer size if needed
        message_bytes = data.decode()

        # Count the number of characters in the message
        character_count = len(message_bytes)

        # Convert the message to uppercase
        uppercase_message = message_bytes.upper()

        # Create a response message (character count + uppercase message)
        response = f"{character_count} characters: {uppercase_message}"

        # Send the response back to the client's IP address and port (the same one it received from)
        server_socket.sendto(response.encode(), client_address)

        print(f"Received message from {client_address}: {message_bytes}")
        print(f"Sent response to {client_address}: {response}")

    except KeyboardInterrupt:  # Handle Ctrl+C to gracefully exit
        print("\nServer shutting down...")
        break

# Close the socket (not strictly necessary in a loop, but good practice)
server_socket.close()
```

Client Code:

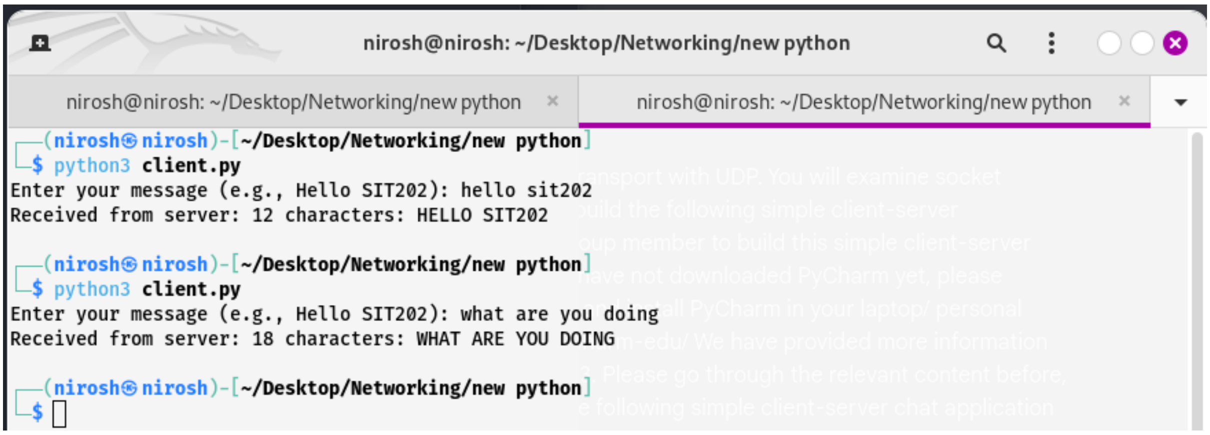
```
import socket

# Server IP address and port (replace with actual server IP if needed)
HOST = "localhost"
PORT = 8888  # Using the same port as the provided code

# Create a UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Get the message to send from the user
message = input("Enter your message (e.g., Hello SIT202): ")

# Encode the message as bytes
message_bytes = message.encode()

# Send the message to the server
client_socket.sendto(message_bytes, (HOST, PORT))

# Receive the server's response (character count and uppercase message)
data, server_address = client_socket.recvfrom(4096)  # Adjust buffer size if
needed
response_bytes = data.decode()

# Print the received message
print(f"Received from server: {response_bytes}")

# Close the socket
client_socket.close()
```

3. You need to show the output of the client- server program to demonstrate that your program works as expected. Please make sure to include the server and client codes and output (screenshots) in your Module 3 lesson review

Server-side output:



Client-side output:



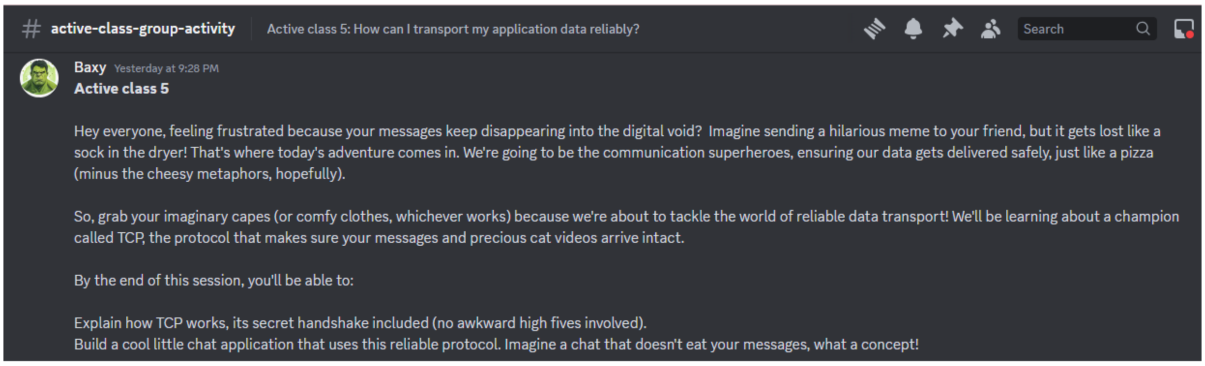
Throughout these activities, I encountered difficulties, such as initially grasping the UDP packet structure. However, by reviewing the module content and seeking guidance from the peers, I overcame these challenges and gained a comprehensive understanding of the concepts.

One particular mistake I made was in the socket programming activity, where I initially misunderstood the message flow between the client and server. This error taught me the importance of careful protocol design and the value of testing and debugging in software development. Overall, these experiences have significantly contributed to my learning journey in this module.

Active class 5: How can I transport my application data reliably?

This activity was about the importance of a reliable transport layer protocol, especially TCP, how it ensures reliable communication. In this activity I'm engaging in group discussions, analysing the UDP protocols with Wireshark and building client-server chat application using TCP.

Activity 1:
I actively participated in the group discussions and collaborated with my friends to understand the importance of a reliable transport layer protocol. I can reference the group activity where we developed a transport layer protocol to ensure reliable image transmission.



active-class-group-activity Active class 5: How can I transport my application data reliably? Search

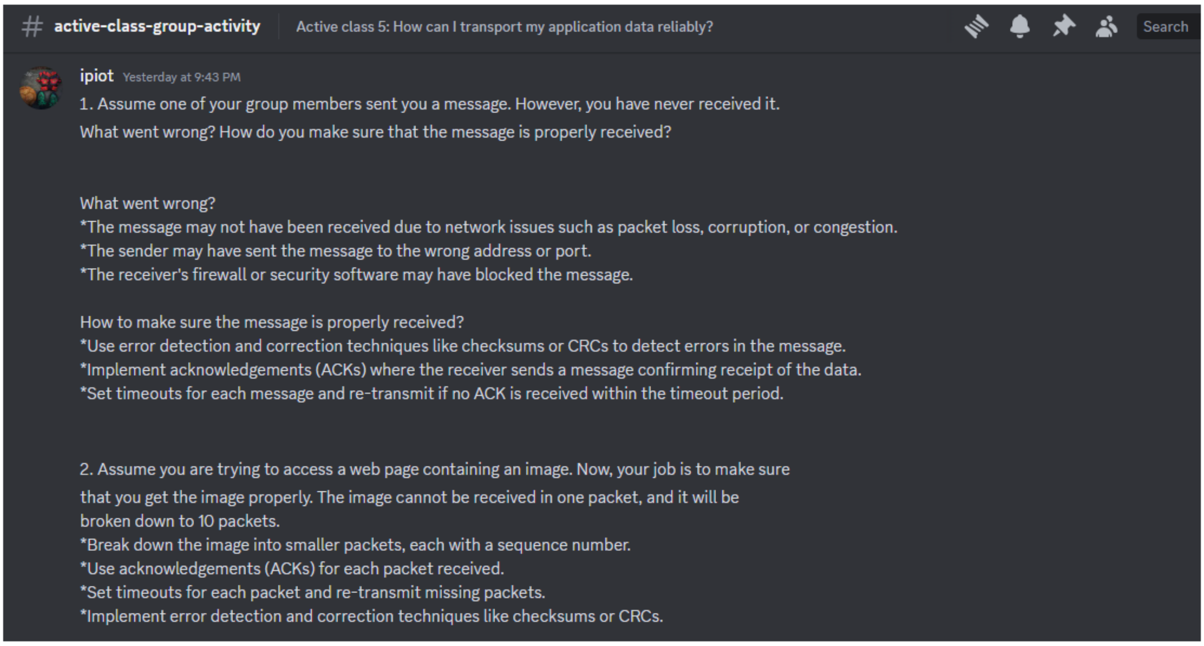
Baxy Yesterday at 9:28 PM
Active class 5

Hey everyone, feeling frustrated because your messages keep disappearing into the digital void? Imagine sending a hilarious meme to your friend, but it gets lost like a sock in the dryer! That's where today's adventure comes in. We're going to be the communication superheroes, ensuring our data gets delivered safely, just like a pizza (minus the cheesy metaphors, hopefully).

So, grab your imaginary capes (or comfy clothes, whichever works) because we're about to tackle the world of reliable data transport! We'll be learning about a champion called TCP, the protocol that makes sure your messages and precious cat videos arrive intact.

By the end of this session, you'll be able to:

Explain how TCP works, its secret handshake included (no awkward high fives involved).
Build a cool little chat application that uses this reliable protocol. Imagine a chat that doesn't eat your messages, what a concept!



active-class-group-activity Active class 5: How can I transport my application data reliably? Search

ipiot Yesterday at 9:43 PM
1. Assume one of your group members sent you a message. However, you have never received it.
What went wrong? How do you make sure that the message is properly received?

What went wrong?
*The message may not have been received due to network issues such as packet loss, corruption, or congestion.
*The sender may have sent the message to the wrong address or port.
*The receiver's firewall or security software may have blocked the message.

How to make sure the message is properly received?
*Use error detection and correction techniques like checksums or CRCs to detect errors in the message.
*Implement acknowledgements (ACKs) where the receiver sends a message confirming receipt of the data.
*Set timeouts for each message and re-transmit if no ACK is received within the timeout period.

2. Assume you are trying to access a web page containing an image. Now, your job is to make sure
that you get the image properly. The image cannot be received in one packet, and it will be
broken down to 10 packets.
*Break down the image into smaller packets, each with a sequence number.
*Use acknowledgements (ACKs) for each packet received.
*Set timeouts for each packet and re-transmit missing packets.
*Implement error detection and correction techniques like checksums or CRCs.

active-class-group-activity | Active class 5: How can I transport my application data reliably?

ipiot Yesterday at 9:57 PM
3. Partner up with one of your group members and develop a transport layer protocol to guarantee that the image is properly received by the client. One can act as the server where the webpage/image is stored and the other can act as the client who wants to receive the image (this means your group has 2 pairs of server-client networks).

I am acting as the **client** for this scenario

H44mid Yesterday at 9:58 PM
I'm acting as the **Server** for the Scenario

ipiot Yesterday at 9:58 PM
Client: "Sending SYN (Synchronize) to initiate connection."

H44mid Yesterday at 9:58 PM
Server: "Received SYN. Sending SYN-ACK (Synchronize-Acknowledgment) to acknowledge and synchronize." (edited)

ipiot Yesterday at 9:58 PM
Client: "Received SYN-ACK. Sending ACK (Acknowledgment) to finalize connection establishment."

H44mid Yesterday at 9:58 PM
Server: "Connection established. Ready to receive request." (edited)

ipiot Yesterday at 9:58 PM
Client: "Sending request for the image."

H44mid Yesterday at 9:58 PM
Server: "Received request. Preparing image for transmission." (edited)

active-class-group-activity | Active class 5: How can I transport my application data reliably?

ipiot Yesterday at 9:58 PM
Client: "Waiting for image transmission."

H44mid Yesterday at 9:59 PM
Server: "Dividing image into packets for sending." (edited)

ipiot Yesterday at 9:59 PM
Client: "Received packet 1. Sending ACK."

H44mid Yesterday at 9:59 PM
Server: "ACKnowledged. Transmitting packet 2." (edited)

ipiot Yesterday at 9:59 PM
Client: "Packet 2 received. Sending ACK."

H44mid Yesterday at 9:59 PM
Server: "ACKnowledged. Sending remaining packets."

ipiot Yesterday at 9:59 PM
Client: "All packets received. Sending ACK."

H44mid Yesterday at 9:59 PM
Server: "ACKnowledged. Image transmission complete."

ipiot Yesterday at 9:59 PM
Client: "Confirmed. Image received. Thank you."

active-class-group-activity | Active class 5: How can I transport my application data reliably?

H44mid Yesterday at 9:59 PM

Server: "You're welcome. Connection will be terminated."

4. Write down protocol that you would use, including the communication between the client and server and all the messages that you are passing.

The protocol involves the client requesting the image from the server, which divides it into 10 packets for transmission. The client acknowledges receipt of each packet, prompting re-transmission if necessary. Once all packets are received, the server confirms successful transmission, ensuring reliable image delivery. (edited)

ipiot Yesterday at 10:14 PM

5. How do you guarantee that your protocol does the job as you expected?

*By using **sequence numbers**and **timeouts**, the protocol can guarantee that the image is received properly, even if some packets are lost or delivered out of order.*
Double-Checking with Acknowledgements: Those confirmation messages you send back are like checking the pigeons made it to the right rooftop. The server knows each packet was delivered safely.**
*Rescuing Lost Packets**: If a packet goes missing, you can request a resend, just like calling for backup if a pigeon gets lost.
Keeping Things Orderly with Sequence Numbers: The numbered codes on the packets are like flight plans for the pigeons. They ensure everything arrives in the right order for reassembly. (edited)

6. Once you have finalized your protocol, check how other group members developed their protocols.

1. Is our protocol simpler or more complex than others? Does it involve fewer message exchanges?

active-class-group-activity | Active class 5: How can I transport my application data reliably?

Baxy Yesterday at 10:20 PM

Hey, our protocol seems pretty straightforward. It only uses four messages for the handshake and packet exchange. Maybe some groups have a more complex way of acknowledging packets? We could see if that affects the overall efficiency.

Amjad Yesterday at 10:20 PM

Our group's protocol uses a checksum for each packet to check for errors during transmission. That might be a bit more complex, but it could be helpful for catching corrupted data. (edited)

ipiot Yesterday at 10:21 PM

OK, thats great.

2. How do other protocols handle missing packets? Is there a more efficient way to request resends?

Amjad Yesterday at 10:21 PM

We included a timer in our protocol. If the client doesn't receive an acknowledgement within a certain time, it automatically requests a resend. Seems faster than waiting for a specific message. (edited)

Baxy Yesterday at 10:21 PM

For missing packets, we just ask for a resend directly. I wonder if there's a way to automatically detect missing packets without needing an extra message every time.

ipiot Yesterday at 10:22 PM

That's awesome to go on with discussions

3. Imagine this video was a giant movie with hundreds of packets. Can our protocol handle that much data? Would we need to add features to avoid duplicate packets (like having two pigeons arrive with the same message)?

Baxy Yesterday at 10:22 PM

A giant movie with hundreds of packets? Hmm, that's a good point. Our protocol might get bogged down with all those resends. Maybe we need a way to check for duplicates too, just in case some packets get sent twice.

active-class-group-activity | Active class 5: How can I transport my application data reliably?

Search

Amjad Yesterday at 10:22 PM
For a massive file, we were thinking of adding a sequence number range instead of individual numbers for each packet. Like, instead of 1, 2, 3... we could say packets 1-10 received, 11-20 received, and so on. That way, we can acknowledge a whole bunch at once.

Baxy Yesterday at 10:23 PM
7. With your group, discuss whether your protocol valid when you need to access a large file that might be broken down into 1000 packets. If not, what are the changes you would like to make in your protocol?

mabrook Yesterday at 10:23 PM
So, our protocol seems to work well for the 10-packet image, but what if we're downloading a giant movie with, like, 1000 packets?

Baxy Yesterday at 10:23 PM
That's a good point! With so many packets, it might be easy to miss one and not even realize it. Our protocol might not catch it.

mabrook Yesterday at 10:24 PM
Maybe we could add a double-check system. After receiving a packet, the client could send a small message back to the server saying, "Hey, I got packet number 23!"

Baxy Yesterday at 10:24 PM
Ooh, that's clever! Then the server can keep track of which packets have been acknowledged. If it doesn't hear back about a specific one, it knows to resend it.

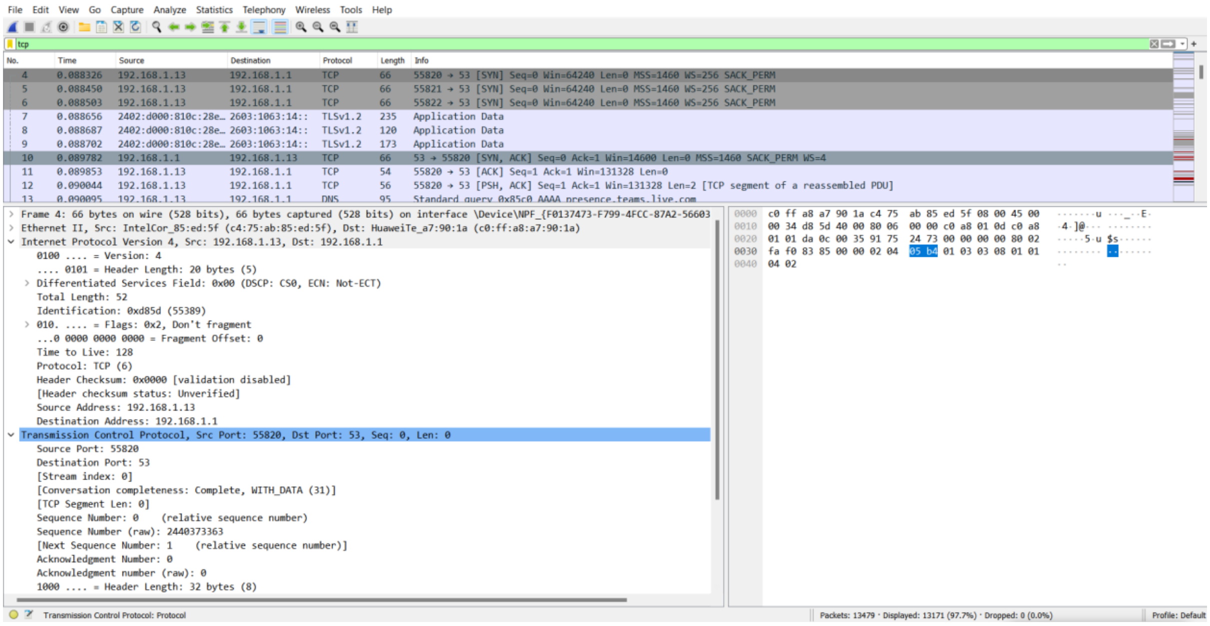
mabrook Yesterday at 10:24 PM
Exactly! We could call it a "packet confirmation" step. Also, with so many packets, what if we get duplicates? We don't want the movie to have two endings!

Baxy Yesterday at 10:24 PM
True! Maybe each packet could have a unique code besides the sequence number. That way, the client can recognize a duplicate and just ignore it.

Activity 2:
We used Wireshark to analyse TCP operations, including the three-way handshake. This hands-on experience solidified my understanding of TCP's reliable communication, which I have evidenced through screenshots of the TCP segments.

1. You can analyse the TCP three-way handshake in Wireshark packet capture. Analyse the order of segments sent/received by two end systems to establish a TCP connection.

Screenshot:



The order of the segments sent/received, based on the captured packets:

Frame 4: The first segment is sent from the host with IP address 192.168.1.13 (Source) to the host with IP address 192.168.1.1 (Destination). This segment has the following flags set:

SYN: This flag indicates that this is a synchronization segment used to initiate a new connection.
Seq=0: This is the initial sequence number chosen by the sending host.

Frame 7: The second segment is sent from the host with IP address 192.168.1.1 (Destination) in response to the first segment. This segment has the following flags set:

SYN: This flag is set to acknowledge the synchronization request from the first segment.
ACK: This flag indicates that the receiver is acknowledging the receipt of the first segment.
Seq=0: This is the initial sequence number chosen by the responding host.
Ack=1: This is the acknowledgment number for the first segment received.

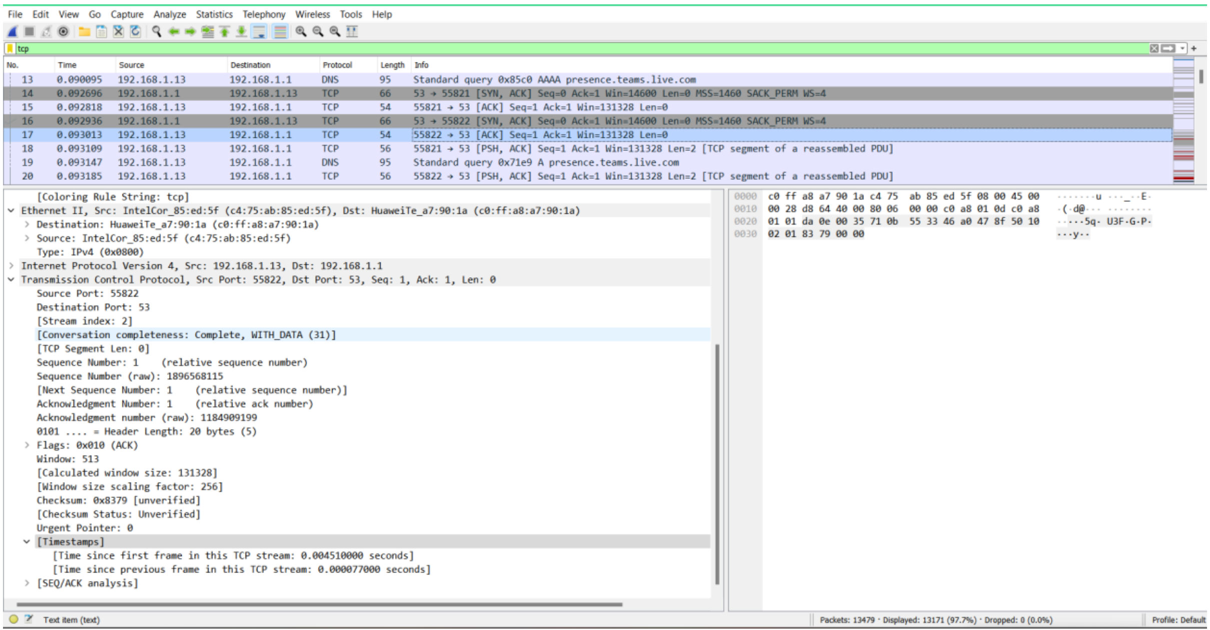
Frame 11: The third segment is sent from the host with IP address 192.168.1.13 (Source) in response to the second segment. This segment has the following flag set:

ACK: This flag acknowledges the receipt of the second segment.
Ack=1: This is the acknowledgment number for the second segment received.

After this three-way handshake, the TCP connection is established between the two end systems, and they can now exchange data segments.

2. Analyze the segment headers used, their purpose and sizes.

Screenshot:



A TCP segment header is located at the beginning of a TCP segment and contains information needed to control the transmission of data between two applications.

TCP Segment Header Fields:

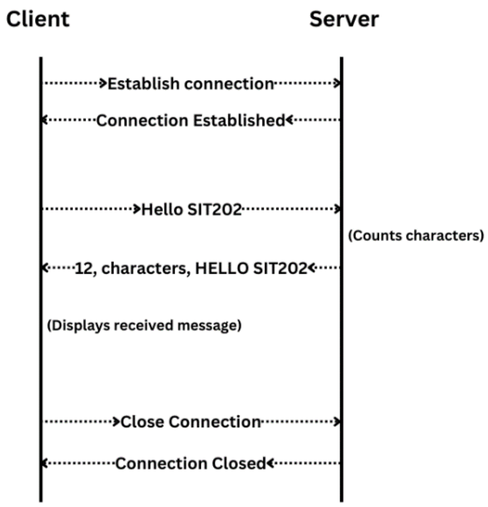
Source Port: The port number of the sending application.
Destination Port: The port number of the receiving application.
Sequence Number: A number used to order the data segments sent from an application. (Partially shown in Wireshark capture as "Seq")
Acknowledgment Number: A number used to acknowledge the receipt of data segments from the receiver. (Partially shown in Wireshark capture as "Ack")
Header Length: The length of the TCP segment header in 32-bit words (usually 5 words or 20 bytes).
Flags: Control flags that specify the current state of the TCP connection (e.g., SYN, ACK, FIN).
Window Size: The amount of data that the sender is willing to receive from the receiver before requiring an acknowledgment. (Shown in Wireshark capture as "Win")
Checksum: An error-checking value calculated over the TCP header and data.44

The sizes of these headers

Ethernet II Header: 14 bytes
IPv4 Header: 20 bytes
TCP Header: 20 bytes
DNS Query: variable length, depending on the domain name

Activity 3:
We built a simple client-server chat application using TCP, demonstrating the practical application of the concepts learned. I have provided the Python code and output screenshots as proof of my work.

1. First you need to draw a diagram to explain the operation and communication of the client- server program that uses TCP.



2. Then one of you can develop the client side and the other can develop the server side of the program (you can also develop both and demonstrate the outcome if you chose to do so)

Server code:

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 8888)
server_socket.bind(server_address)
server_socket.listen(1)
print(f'Server is listening on {server_address}')

while True:
   print('Waiting for a connection..')
   connection, client_address = server_socket.accept()
   try:
      print(f'Connection from {client_address}')

      while True:
          data = connection.recv(1024)
          if not data:
              break

          message = data.decode()
          char_count = len(message)
          uppercase_message = message.upper()

          response = f'Received {char_count} characters: {uppercase_message}'
          connection.sendall(response.encode())
          print(f'Sent response: {response}')


   finally:
       connection.close()

server_socket.close()
```

Client code:

```python
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 8888)
client_socket.connect(server_address)

try:
    message = 'How are you'
    client_socket.sendall(message.encode())
    print(f'Sent message: {message}')

    data = client_socket.recv(1024)
    response = data.decode()
    print(f'Received response: {response}')

finally:
    client_socket.close()
```

3. You need to show the output of the client- server program to demonstrate that your program works as expected. Please make sure to include the server and client codes and output (screenshots) in your lesson review.

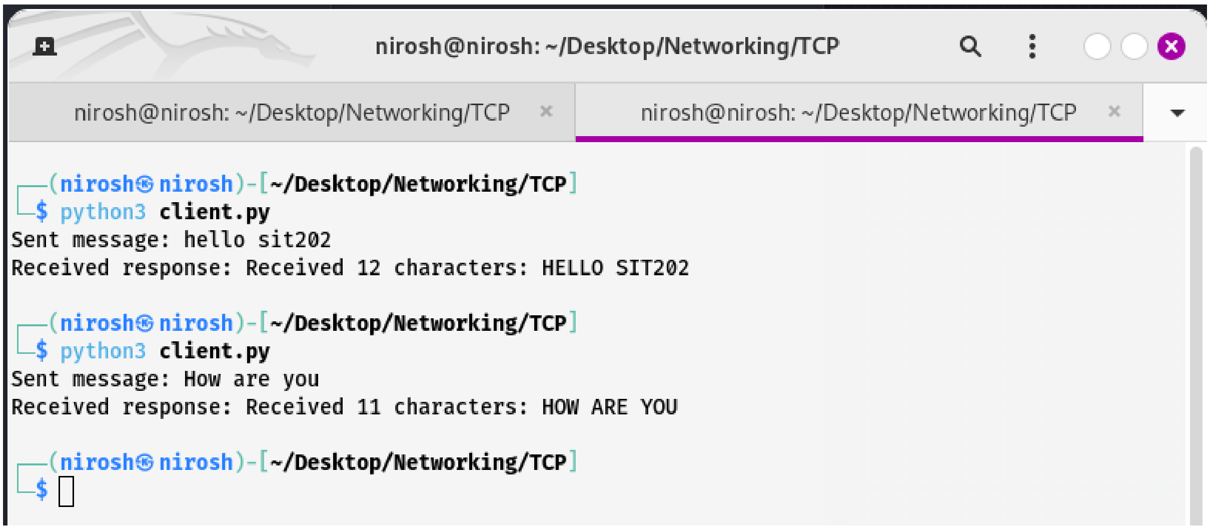
Server-side output:

nirosh@nirosh: ~/Desktop/Networking/TCP

nirosh@nirosh: ~/Desktop/Networking/TCP | nirosh@nirosh: ~/Desktop/Networking/TCP

```
┌──(nirosh㉿nirosh)-[~/Desktop/Networking/TCP]
└─$ python3 server.py
Server is listening on ('localhost', 8888)
Waiting for a connection..
Connection from ('127.0.0.1', 42148)
Sent response: Received 12 characters: HELLO SIT202
Waiting for a connection..
Connection from ('127.0.0.1', 38706)
Sent response: Received 11 characters: HOW ARE YOU
Waiting for a connection..
```

Client-side output:



I found the concept of TCP’s congestion control challenging. However, by reviewing additional resources and seeking guidance from the study materials, I was able to grasp the concept and apply it in the activities.

As for mistakes, I initially misunderstood the TCP segment structure, which led to errors in my Wireshark analysis. Through trial and error, and with feedback from my peers and friends, I learned the correct interpretation, which improved my protocol analysis skills.