Undergraduate Lab Report of Jinan University

Course Title Computer Networks Lab Evaluation

Lab Name UDP: User Datagram Protocol

Lab Address N117 Instructor Dr. CUI Lin (崔林)

Student Name 蒋云翔 Student No 2022102330

College International School

Department Major CST

Date 2024 / 11 / 23

1. Introduction

1) Objective

- To see how UDP works.
- Know the format of UDP datagram.
- Know the calculation of UDP checksum.
- Understand the advantages and disadvantages of UDP protocol.

2) Experiment Principle

1. Process-To-Process Delivery

The data link layer ensures frame delivery between two adjacent nodes on a link, known as node-to-node delivery. In contrast, the network layer manages the transport of datagrams between two hosts, termed host-to-host delivery. However, Internet communication transcends mere data exchange between nodes or hosts. It fundamentally occurs between two processes (application programs), necessitating process-to-process delivery. On both the source and destination hosts, multiple processes may be running simultaneously. Therefore, a system is needed to accurately channel data from a specific process on the source host to its corresponding process on the destination host.

This role is fulfilled by the transport layer, which oversees process-to-process delivery, ensuring that a packet or a message segment is accurately transmitted from one process to another. Typically, this communication follows a client/server model. The following illustrative figure can demonstrate these three types of deliveries and their respective scopes, emphasizing the layered nature of network communication.

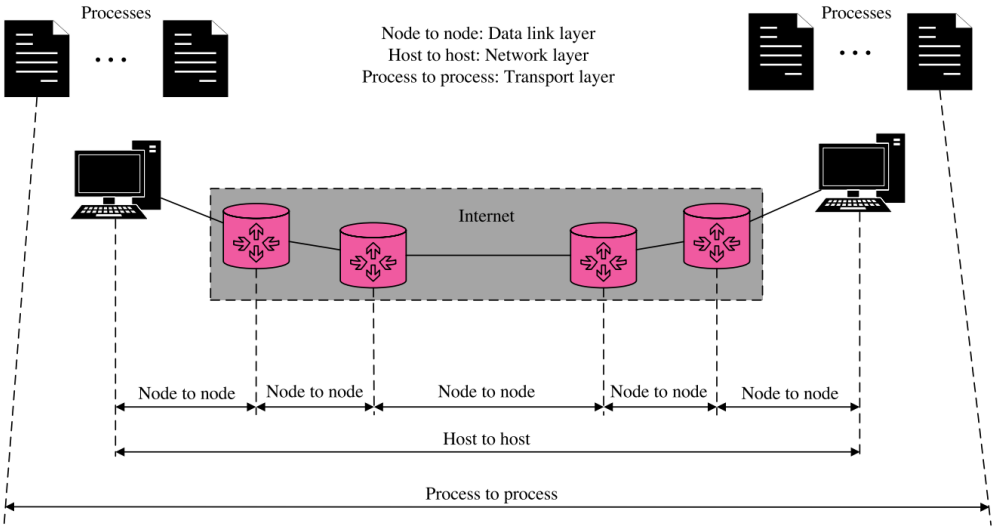

Figure 1: Types of data deliveries

1.1. Port Number

In the network layer, an IP address is essential to select a specific host from millions available. A network layer datagram requires a destination IP address for its delivery and a source IP address for responses from the destination.

At the transport layer, a different kind of address, known as a port number, is used to differentiate between multiple processes on the destination host. Here, the destination port number directs the delivery of the data, while the source port number is used for replies.

In the Internet model, port numbers are 16-bit integers ranging from 0 to 65,535. Client programs use an ephemeral port number, typically assigned randomly by the client host's transport layer software, for their temporary communication sessions.

Conversely, server processes use specific port numbers that aren't chosen randomly. If servers were to use random port numbers, client processes would struggle to locate them for service access, as the port number would be unknown. To avoid this confusion and the need for additional communication to discover service ports, the Internet uses universally recognized port numbers for servers, known as well-known port numbers. While there are some exceptions (like certain clients assigned well-known port numbers), generally, every client process is aware of the well-known port numbers of the server processes it needs to communicate with.

The following list or reference to well-known port numbers can illustrate this concept, highlighting the standardized port assignments used for common services and applications.

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
8	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Figure 2: Well-known port numbers of UDP

1.2. Socket Addresses

For process-to-process delivery, a unique combination of an IP address and a port number at each end is required to establish a connection. This combination, known as a socket address, distinctly identifies each participating process. The client socket address uniquely defines the client process, while the server socket address uniquely defines the server process.

In this context, a transport layer protocol relies on a pair of socket addresses: one for the client and one for the server. These four critical elements —two IP addresses and two port numbers —are encapsulated within the headers of the network communication. Specifically, the IP header carries the IP addresses of the source and destination, and the transport layer protocol header (either UDP or TCP) includes the corresponding port numbers. This arrangement ensures that each data packet is accurately directed from a specific process on the source to the appropriate process on the destination.

2. Connectionless Versus Connection-Oriented Service

Flow the view of communication, in the OSI-model, an upper layer can provide two kinds of servers: Connection-Oriented Service and Connectionless Service.

2.1. Connection-Oriented Service

Connection-oriented service involves combining two equal entities for communication and typically follows three key phases:

[1] **Establish the connection:** In this initial phase, the addresses of both the source and destination need to be specified in the service primitive and protocol data unit. This stage also allows for negotiation regarding the quality of service and other options.

[2] **Transfer the data:** Once the connection is established, complete source and destination addresses are not required in each packet. Instead, a connection identification is used for packet recognition. This results in control messages being significantly smaller than the data packets, reducing system overhead and enhancing communication channel efficiency. Furthermore, data packets are sent and received in a fixed sequence, ensuring orderly communication.

[3] **Release the connection:** After data transfer is complete, specific service primitives are used to terminate the connection.

In the context of connection-oriented service, the connection acts like a pipe where senders dispatch packets from one end and receivers collect them in the same order at the other end. This form of connection, often referred to as a Virtual Circuit, ensures the integrity of the data packets by preventing loss, duplication, and disorder.

For frequent communication between two users, permanent virtual circuits can be established. These circuits circumvent the need for repeated establishment and termination of connections, functioning similarly to a private telephone line.

2.2. Connectionless Service

In connectionless service, establishing a connection between two entities is not required, making this service more flexible and convenient. However, it does not guarantee protection against packet loss, duplication, or disorder. Additionally, the necessity to include complete source and destination addresses in each packet increases overall costs.

Connectionless services come in three varieties:

- **Datagram:** This service is characterized by its simplicity and cost-effectiveness, as it involves sending packets without awaiting responses. While it is quick and straightforward, it lacks reliability. Datagram service is well-suited for communications where high real-time performance and redundancy are crucial.
- **Confirmed delivery:** Often referred to as a reliable datagram, this service involves sending a confirmation message for each packet. These confirmations are generated by the service layer, not the end user. This ensures successful delivery to the receiver but does not confirm the accurate reception of the packets.
- **Request-reply:** Upon receiving a packet, the receiver sends back a reply message. However, there's a possibility of packet loss from both sender and receiver. If the receiver detects an issue with the packet, it responds with a message indicating the error.

These types of connectionless services offer different levels of reliability and efficiency, catering to various communication needs and scenarios.

3. Introduction of UDP

The User Datagram Protocol (UDP) is a fundamental part of the Internet protocol suite. It employs a straightforward connectionless transmission model, prioritizing minimal protocol complexity. Lacking handshaking dialogues, UDP allows the

unreliability of the underlying network protocol to be evident to the user's program. It does not provide guarantees for delivery, ordering, or protection against duplicates. However, UDP does offer checksums to ensure data integrity and uses port numbers to address different functions at the source and destination.

UDP enables computer applications to send messages, known as datagrams, to other hosts on an IP network without needing pre-established communication channels or data paths. This protocol is particularly useful in scenarios where error checking and correction are unnecessary or are handled by the application itself, thereby reducing network interface overhead. UDP is favored by time-sensitive applications where the timely arrival of packets is more critical than reliability, as it prioritizes the dropping of packets over delayed delivery, which is crucial in real-time systems. For applications requiring error correction at the network level, alternatives like the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) are more appropriate, as they are specifically designed for this purpose

UDP is characterized by several key attributes that make it suitable for a variety of applications:

- Its transaction-oriented nature is ideal for simple query-response protocols, as seen in the Domain Name System (DNS) and the Network Time Protocol (NTP).
- UDP's use of datagrams makes it a good fit for modeling other protocols, such as those used in IP tunneling or Remote Procedure Call and the Network File System.
- The protocol's simplicity is advantageous for bootstrapping and applications that do not require a full protocol stack, evident in protocols like DHCP and Trivial File Transfer Protocol.
- Being stateless, UDP can efficiently handle a vast number of clients, making it suitable for streaming media applications like IPTV.
- The absence of retransmission delays in UDP renders it appropriate for real-time applications, including Voice over IP, online gaming, and various protocols based on the Real Time Streaming Protocol.
- UDP works effectively in unidirectional communication, ideal for broadcasting information as utilized in many service discovery protocols and for sharing information like broadcast time or Routing Information Protocol.

These aspects of UDP highlight its versatility across different network scenarios, from lightweight communication to real-time data transmission.

4. Packet format of UDP

UDP packets, called user datagrams, have a fixed-size header of 8 Bytes. Figure 3 shows the format of a user datagram.

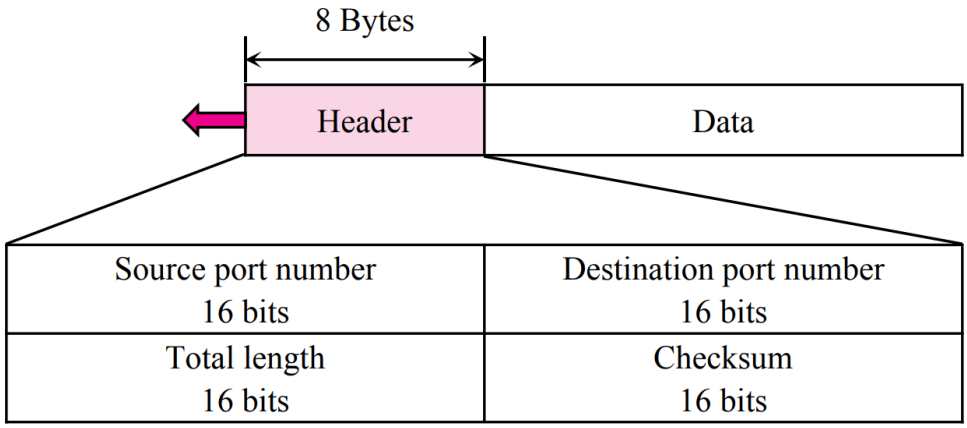

Figure 3: Packet format of UDP

UDP utilizes several fields in its datagram structure, each playing a critical role in the protocol's functioning:

- **Source port number:** This 16-bit field represents the port number used by the process on the source host. For a client (sending a request), this is typically an ephemeral port number, dynamically chosen by the UDP software on the source host. For a server (sending a response), it's usually a well-known port number.
- **Destination port number:** Also 16 bits in length, this field denotes the port number used by the process on the destination host. When the destination is a server (receiving a request), it's commonly a well-known port number. If it's a client (receiving a response), the port number is usually the ephemeral port number, which the server replicates from the received request packet.
- **Length:** This 16-bit field specifies the total length of the UDP datagram, including both header and data. While the theoretical maximum length is 65,535 Bytes, practical limitations dictate a shorter length due to UDP datagrams being encapsulated within IP datagrams, which also have a 65,535-byte limit. Although the IP header contains fields for total length and header length, from which the UDP datagram length could be inferred, UDP includes its own length field for efficiency, allowing the UDP layer to calculate data length directly.
- **Checksum:** This field covers the entire user datagram (header plus data) and is instrumental in detecting transmission errors.

These fields collectively ensure that UDP datagrams are correctly formatted, addressed, and validated for error-free transmission and reception.

5. UDP Encapsulation

When a process needs to send datagrams via UDP, it appends a pair of socket addresses (source and destination) and the data length to the UDP payload. UDP then prepends its header to this data, creating a UDP datagram. This datagram, along with the socket addresses, is handed over to the IP layer. The IP layer adds its own header to the datagram and sets the Protocol field to 17, signaling that the contained

data originates from UDP. Following this, the IP datagram is passed down to the Data Link Layer, where it may be further encapsulated with additional headers and potentially a tail for error checking or other purposes. Finally, the Physical Layer takes over, converting each bit of the datagram into electrical or optical signals for transmission across the network to the intended remote host. This layered approach ensures organized, efficient data handling and transmission from one host to another over the network.

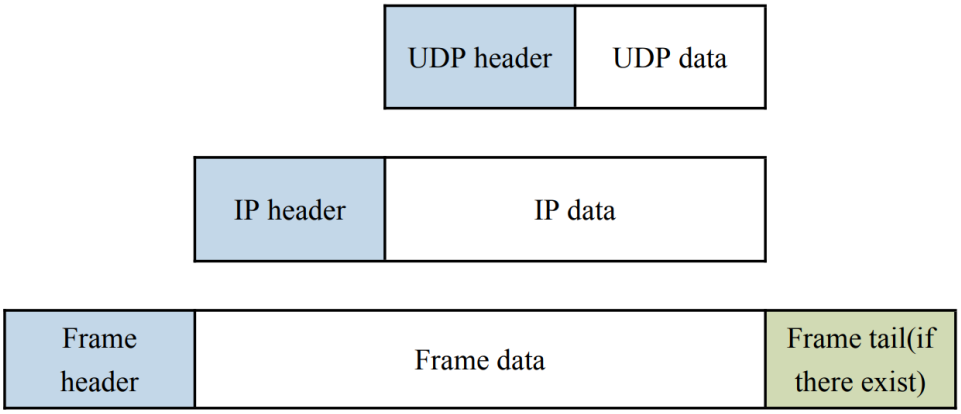

Figure 4: UDP Encapsulation

6. UDP Checksum

The calculation of the UDP checksum differs from that of IP and ICMP, as it encompasses three components: a pseudo header, the UDP header, and the data from the application layer.

The pseudo header, derived from the IP packet's header in which the UDP datagram is encapsulated, includes certain fields set to zeros. This pseudo header is crucial because, without it, a user datagram might reach its destination intact, but a corrupted IP header could result in the datagram being delivered to an incorrect host.

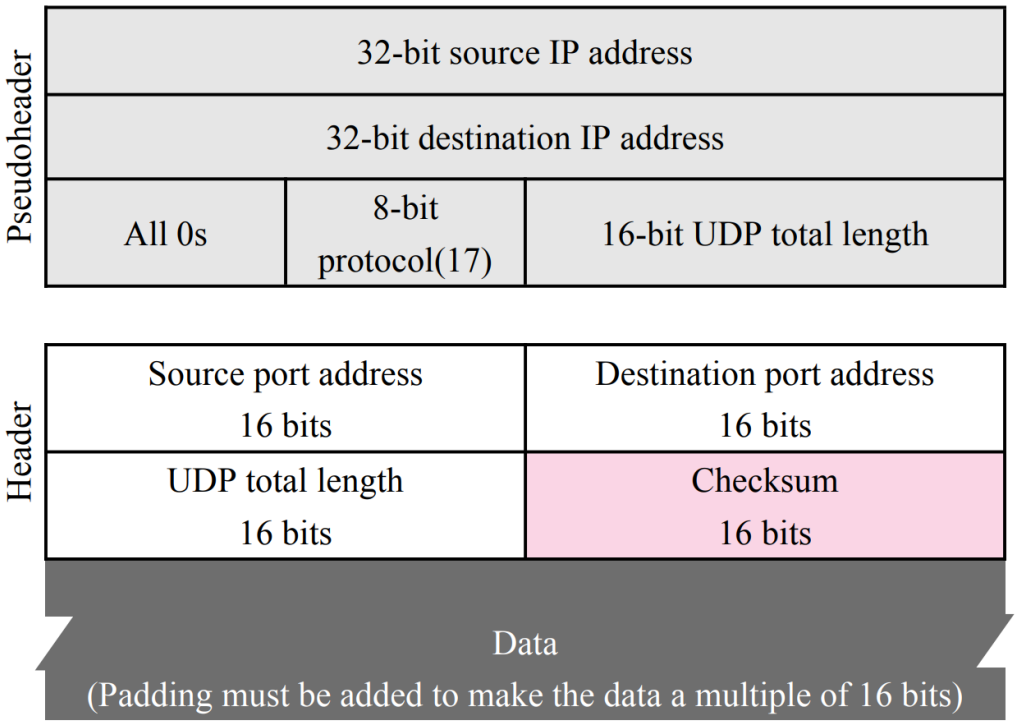

Figure 5: Add pseudo header to UDP datagram

Including the pseudo header in the checksum calculation addresses this issue. It incorporates the protocol field to confirm the packet's association with UDP, differentiating it from other transport-layer protocols. This is particularly important in situations where a process might use either UDP or TCP, as the destination port number could be the same for both. For UDP, the protocol field value is 17. Any alteration of this value during transmission would be detected by the checksum calculation at the receiver, prompting UDP to discard the packet, thereby preventing it from being delivered to an incorrect protocol.

It's noteworthy to mention the similarity between the fields of the pseudo header and the last 12 Bytes of the IP header. This design helps in further ensuring the accuracy and integrity of the transmitted data.

6.1. Checksum calculation of the sender

To calculate the UDP checksum, the sender performs the following steps:

(1) The pseudo header is appended to the UDP datagram. This pseudo header is part of the IP packet's header where the UDP datagram is to be encapsulated.

(2) The Checksum field in the UDP header is initially filled with all zeros.

(3) The datagram, including the pseudo header, is divided into 16-bit segments.

(4) If there is an odd number of segments, an additional 16-bit segment filled with all zeros is added. This extra segment is only for checksum calculation and is removed afterward.

(5) All these 16-bit segments are summed together using Radix-minus-one complement arithmetic.

(6) The final sum is then inverted (flipped) and placed into the UDP checksum field.

(7) The pseudo header is removed from the datagram.
(8) Finally, the UDP datagram, now with the calculated checksum, is handed over to the IP layer for transmission.

It's important to note that the arrangement of lines in the pseudo header and the addition of extra zeros do not impact the checksum calculation process.

An example of this process can be illustrated where, if the length of the UDP datagram is 15 Bytes, an additional byte filled with zeros is added to make the length even for checksum calculation.

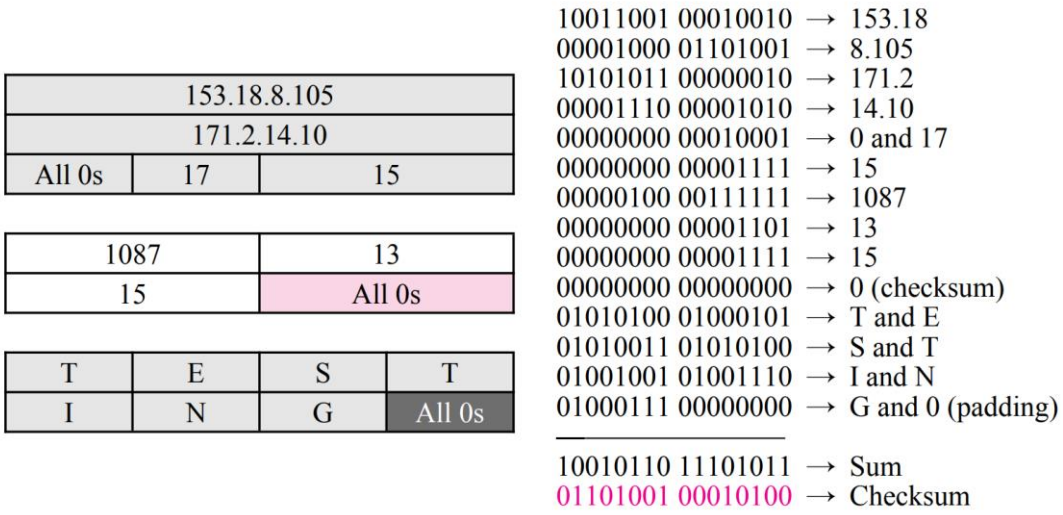

Figure 6: The process of UDP checksum

6.2. Checksum calculation of the receiver

To calculate and verify the UDP checksum, the receiver follows these steps:

[1] The pseudo header, which is part of the IP packet's header, is added to the UDP datagram.
[2] The datagram is padded with extra Bytes if necessary to ensure that its total length is an even number of Bytes.
[3] The datagram, including the pseudo header and any padding, is then divided into segments, each 16 bits long.
[4] These 16-bit segments are summed together using Radix-minus-one complement arithmetic.
[5] The sum obtained from the previous step is then inverted (flipped).
[6] The resulting value is checked: if it consists of all zeros, the checksum is considered valid. In this case, the pseudo header and any padding are removed, and the datagram is accepted. If the result is not all zeros, the datagram is discarded as it indicates a checksum error.

It is important to note that the checksum in UDP is optional. If a checksum is not used, the checksum field in the UDP header must be filled with zeros. This flexibility allows for different levels of error checking based on the application's needs and network reliability.

7. Use of UDP

UDP protocol finds its applications in various processes, each leveraging its unique characteristics:

- UDP is ideal for processes that need straightforward request-response communication without a heavy emphasis on flow and error control. It's generally not the go-to choice for processes like FTP, which require the transmission of large volumes of data.
- Processes that have built-in mechanisms for flow and error control can efficiently utilize UDP. An example is the Trivial File Transfer Protocol (TFTP), which, with its own flow and error control, pairs well with UDP.
- For multicasting, UDP is particularly well-suited. Unlike TCP, UDP software inherently supports multicasting, making it a preferred choice for sending data to multiple destinations simultaneously.
- Management processes, such as SNMP (Simple Network Management Protocol), often rely on UDP for their communication needs due to its simplicity and efficiency.
- UDP is also commonly used in certain routing update protocols like the Routing Information Protocol (RIP), where rapid dissemination of routing information is more critical than the guaranteed delivery of every single message.

These varied uses of UDP highlight its versatility, especially in scenarios where simplicity, speed, and efficiency are more desirable than strict reliability and sequence control.

2. Lab Environment

I accomplish this Lab at home using my own laptop with Wireless network (WLAN).

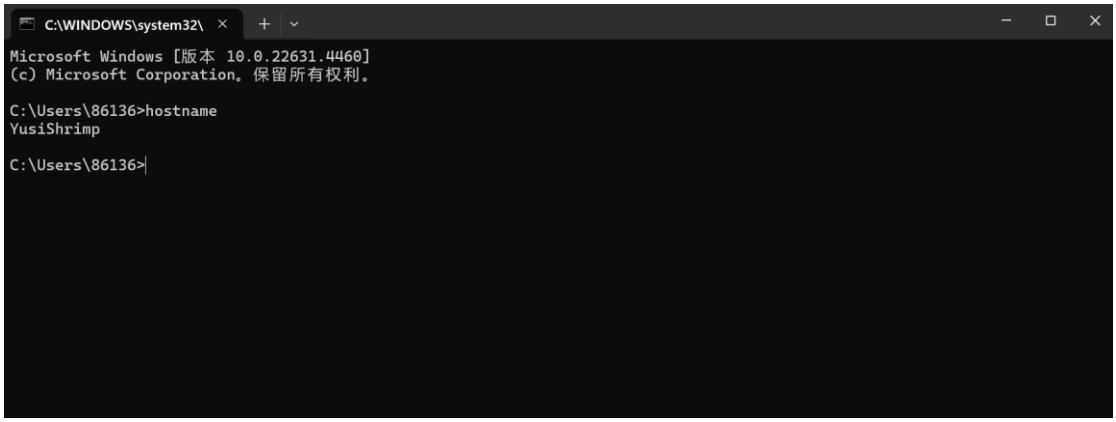

Figure 7: Computer name

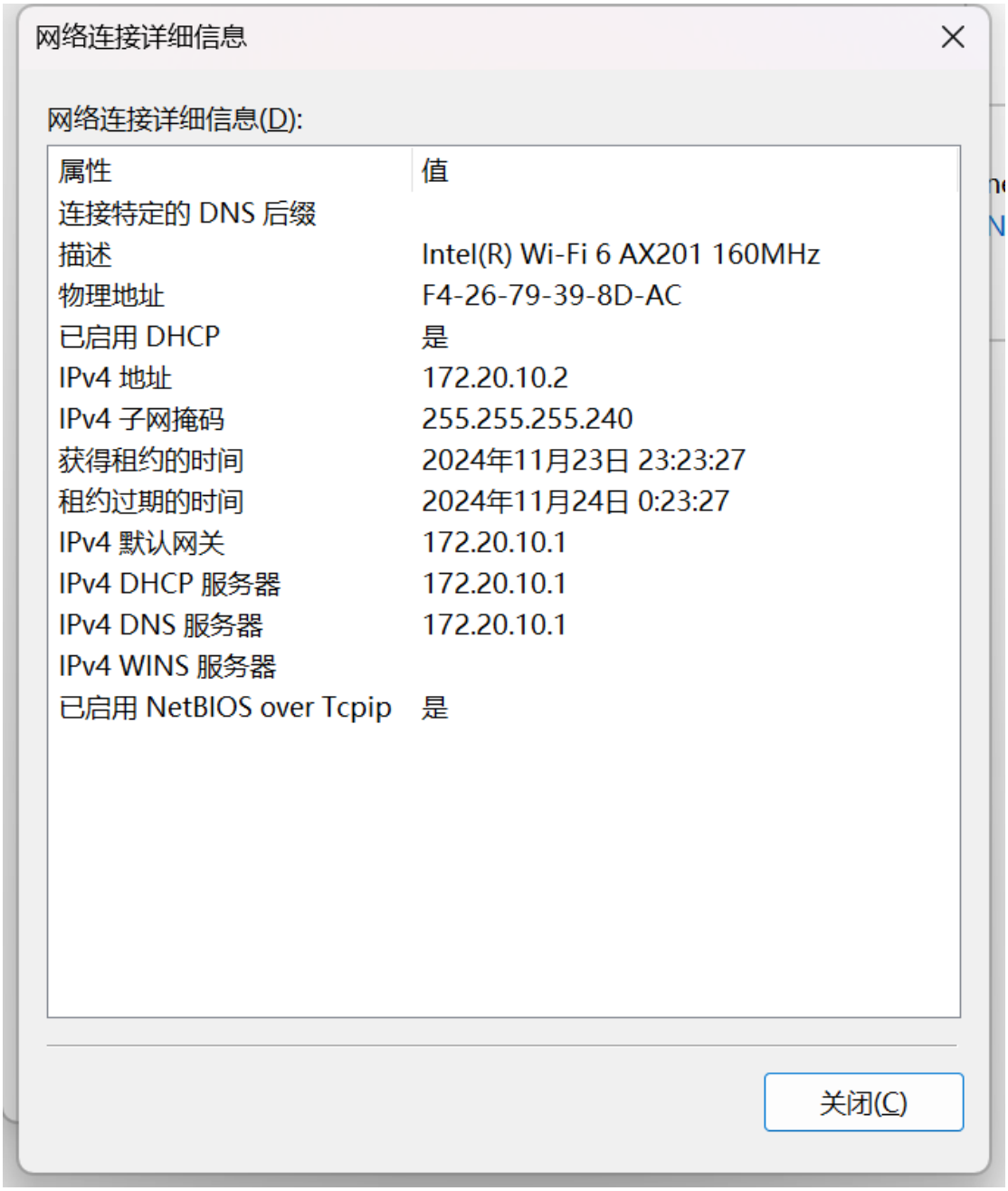

Figure 8: Detailed information of wireless network

3. Experiment Steps and Results

1) Task 1: Capture and Explore Trace of UDP

1. Experiment steps:

[1] Step 1: Launch Wireshark and start a capture with a filter of “udp”. (Remember to choose correct interface)

[2] Step 2: Perform some activities that will generate UDP traffic as described above, e.g., browse website or start a short VoIP call.

[3] Step 3: Stop Wireshark and inspect the trace you captured.

[4] Step 4: Check whether there are UDP messages without your computer’s IP

address as either the source or destination IP address. Examine these UDP messages and give the destination IP addresses that are used when your computer is neither the source IP address nor the destination IP address.

2. Results:

Select the “WLAN” interface and set the udp filter:

Figure 9: Select interface and set filters

Play some videos on “www.bilibili.com”:

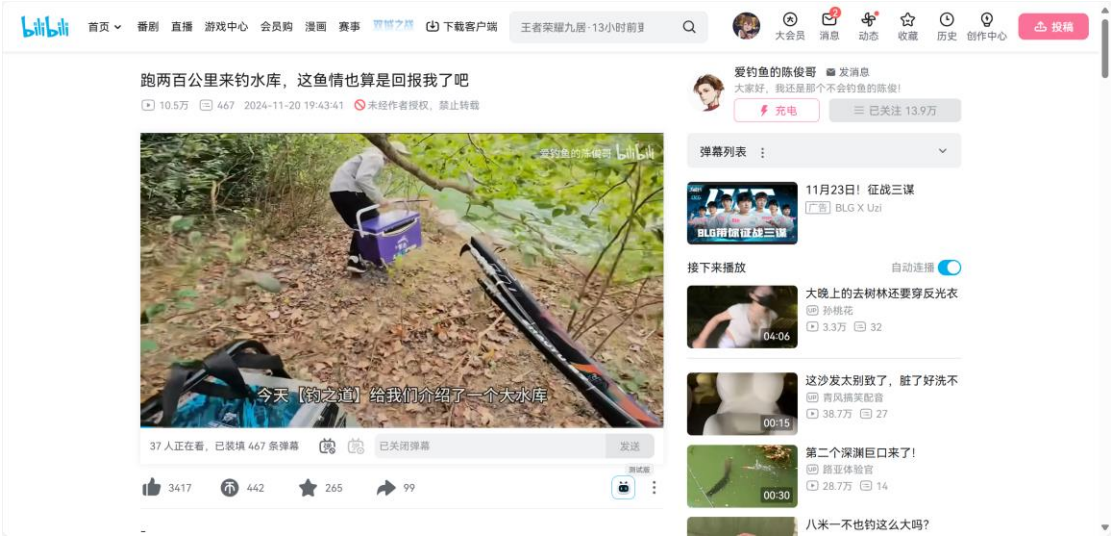

Figure 10: Play videos that will generate UDP traffic

Observe the UDP packets captured by Wireshark during the video viewing process. Since only the web page is opened to watch the video, and at the same time, I learned that video streaming usually uses the QUIC protocol, it can be basically determined that the large number of captured UDP packets come from the video stream being watched. Check the contents of each field in the UDP header:

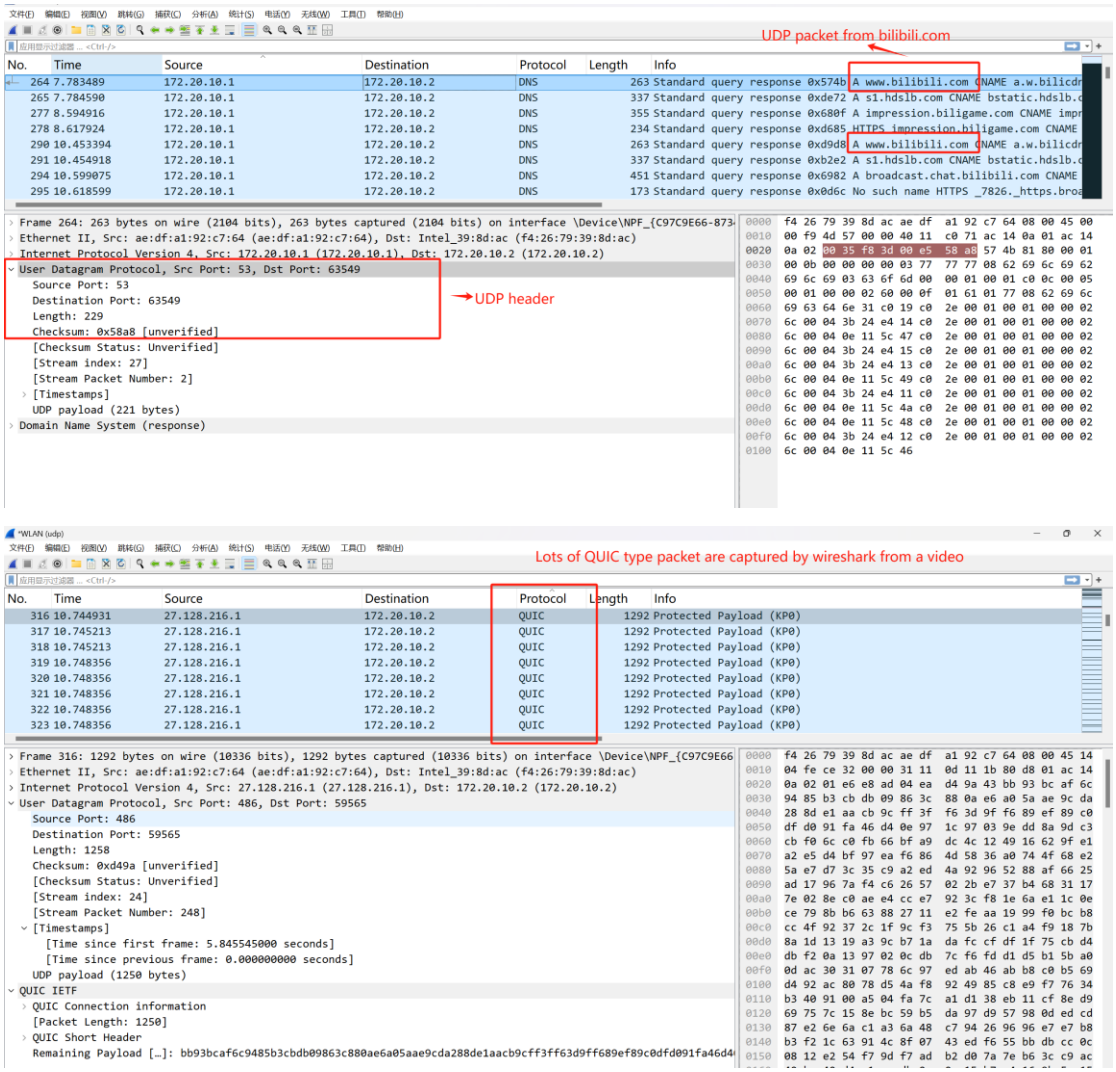

Figure 11: UDP packet capture trace of video stream

The QUIC protocol defines a layer on top of UDP that brings error handling, reliability, flow control, and built-in security to UDP. In addition, I also observed some UDP packets that did not use my computer's IP address as the source address and destination address. Such packets have different upper-layer protocols. Some packet contents are checked as follows:

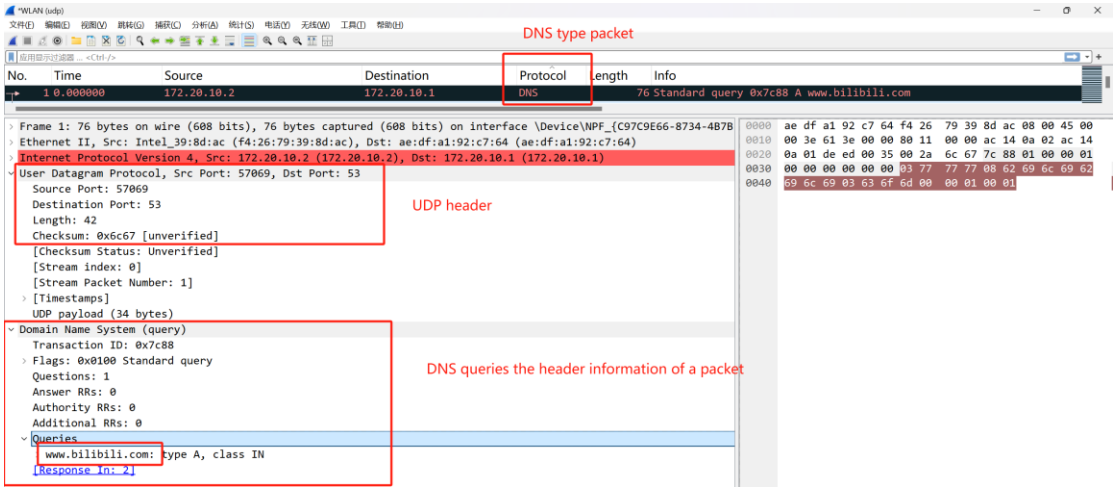

Figure 12: Captured DNS packet from the video

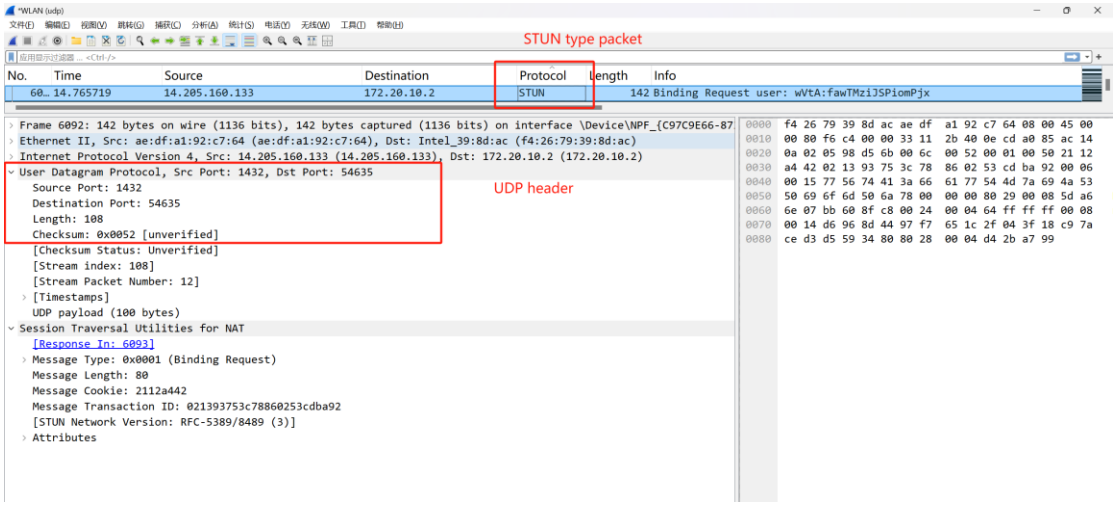

Figure 13: Captured STUN packet from the video

3. Answer the questions:

Q1: How does IP know that the next higher protocol layer is UDP?

A1: It primarily determines this by examining the protocol field in the IP packet. In an IP packet, there is a specific field called the "Protocol" field (in IPv4) or the "Next Header" field (in IPv6), which is used to specify the type of upper-layer protocol carried by the IP packet. For UDP, the value of this field is 17. Therefore, when the IP layer receives a packet, it looks at the value of the "Protocol" field. If the value is 17, the IP layer knows that the next higher protocol layer is UDP and then passes the packet to the UDP for processing.

Q2: What does the Length field of UDP include? And how long in bytes is the entire UDP header?

A2: The length field in the UDP header includes the total length of the UDP datagram, which is the header plus the data payload. The UDP header is always 8 Bytes long.

Q3: What is the typical size of UDP messages in your trace?

A3: The typical size of UDP messages in my trace is 1292 Bytes.

Q4: How long in bits is the UDP checksum? What should be included when calculating UDP the checksum?

A4: The UDP checksum is 16 bits long. When calculating the UDP checksum, the pseudo-header, the UDP header, and the UDP data should be included. The pseudo-header contains the source and destination IP addresses, the protocol number for UDP, and the UDP length field.

Q5: Explain why the UDP "Checksum" should include the pseudo header?

A5: The UDP "Checksum" includes the pseudo header to provide additional error checking beyond the UDP datagram itself. The pseudo header contains the source and destination IP addresses, the Protocol field set to UDP, and the length of the UDP payload. Including the pseudo header in the checksum calculation helps ensure that the datagram arrives at the correct destination and is being processed by the correct protocol. It helps detect if a datagram has been misrouted or corrupted at the IP layer, thus providing a mechanism for higher reliability in data transmission.

2) Task 2: UDP Server and Client

1. Experiment steps:

[1] **Step 1:** Download and compile the "udpserver.c" and "udpclient.c" in MinGW. Or you can use the "udpserver.exe" and "udpclient.exe".

[2] **Step 2:** Prepare two computers, one for UDP server, another for UDP client. Launch Wireshark and start a capture with a filter of "udp.port==60001" (port number is the UDP server port) on both computers.

[3] **Step 3:** Run "udpserver.exe" and "udpclient.exe" using the following command:
On the Server side:

udpserver.exe [Port of server, e.g., 60001]

On the Client side:

udpclient.exe [IP of server] [Port of server]

[4] **Step 4:** Send any random message from UDP client.

[5] **Step 5:** Stop Wireshark and inspect the trace you captured. If you can not capture any traffic, use the Wireshark traces and program output that we provided for you.

2. Results:

```
选择 管理员: 命令提示符
Microsoft Windows [版本 10.0.22631.4460]
(c) Microsoft Corporation。保留所有权利。

C:\Windows\System32>netstat -a -n -p UDP

活动连接

  协议   本地地址          外部地址        状态
  UDP    0.0.0.0:123            *:*
  UDP    0.0.0.0:500            *:*
  UDP    0.0.0.0:4500           *:*
  UDP    0.0.0.0:5050           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5355           *:*
  UDP    0.0.0.0:56376          *:*
  UDP    127.0.0.1:1900         *:*
  UDP    127.0.0.1:7897         *:*
  UDP    127.0.0.1:10140        *:*
  UDP    127.0.0.1:10141        *:*
  UDP    127.0.0.1:60148        *:*
  UDP    127.0.0.1:60149        *:*
  UDP    127.0.0.1:60666        *:*
  UDP    127.0.0.1:63843        *:*
  UDP    127.0.0.1:64898        127.0.0.1:64898
  UDP    172.20.10.2:137        *:*
  UDP    172.20.10.2:138        *:*
  UDP    172.20.10.2:1900       *:*
  UDP    172.20.10.2:5353       *:*
  UDP    172.20.10.2:63842      *:*
  UDP    192.168.137.1:137      *:*
  UDP    192.168.137.1:138      *:*
  UDP    192.168.137.1:1900     *:*
  UDP    192.168.137.1:5353     *:*
  UDP    192.168.137.1:63841    *:*
  UDP    192.168.214.1:137      *:*
  UDP    192.168.214.1:138      *:*
  UDP    192.168.214.1:1900     *:*
  UDP    192.168.214.1:5353     *:*
  UDP    192.168.214.1:63840    *:*

C:\Windows\System32>_
```

Figure 14: check all existing UDP services on local computer

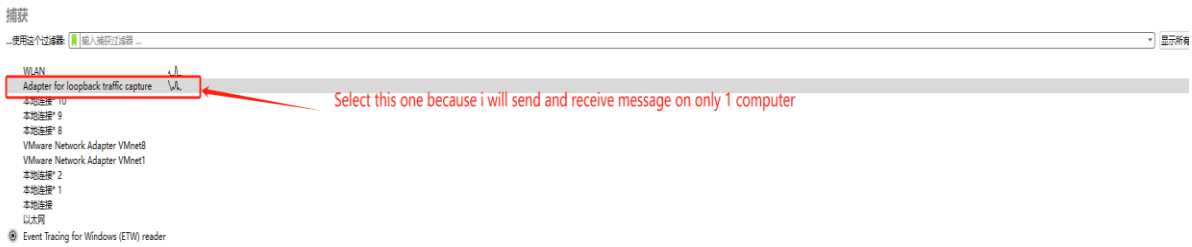

Figure 15: Choose adapter for loopback

Use 2 cmd windows to execute two exe files in VScode.

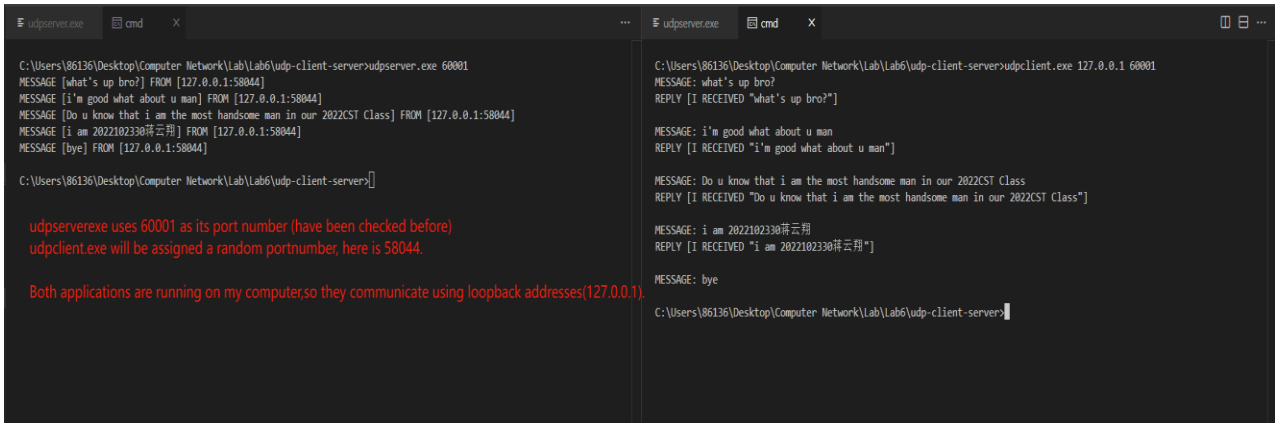

Figure 16: Run the udpserver.exe and udpclient.exe to communicate

Use Wireshark to capture the UDP packet on the “Adapter for loopback traffic capture” interface and check its content:

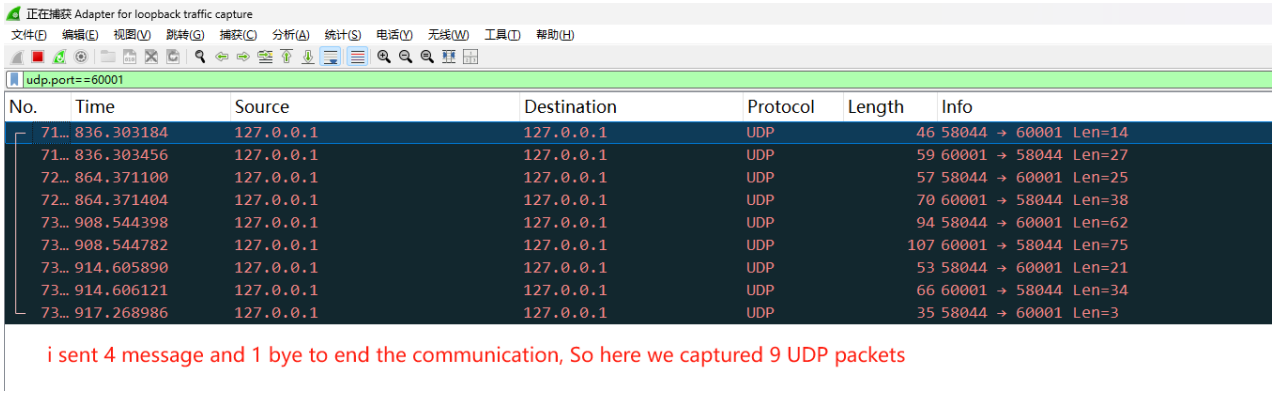

Figure 17: All the UDP packet captured by wireshark

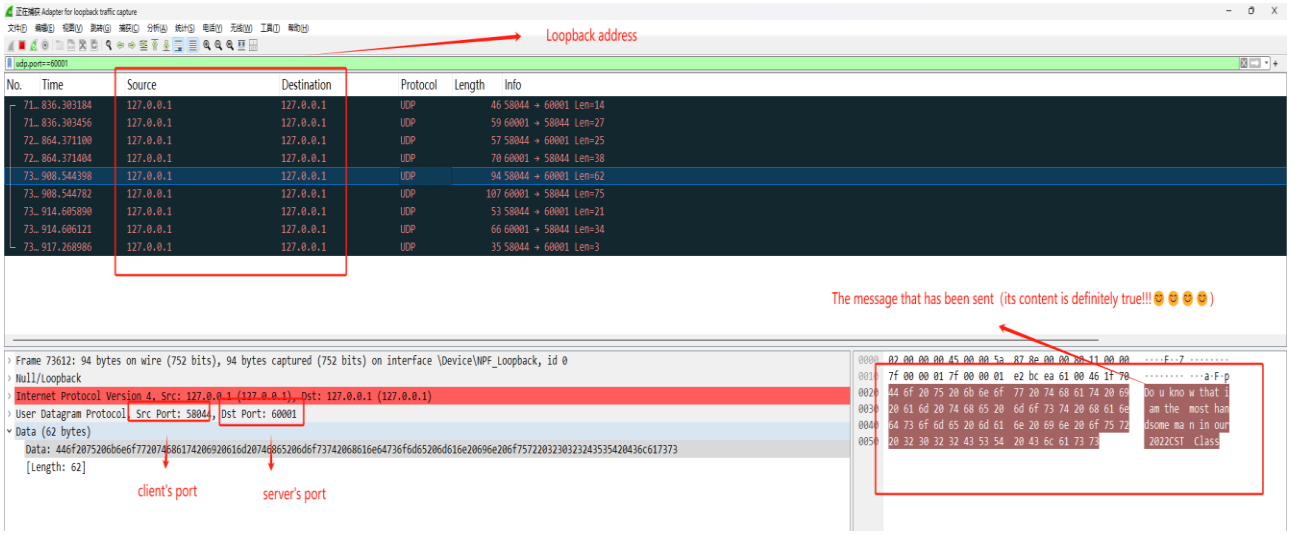

Figure 18: One of the Captured message sent by udpclient.exe

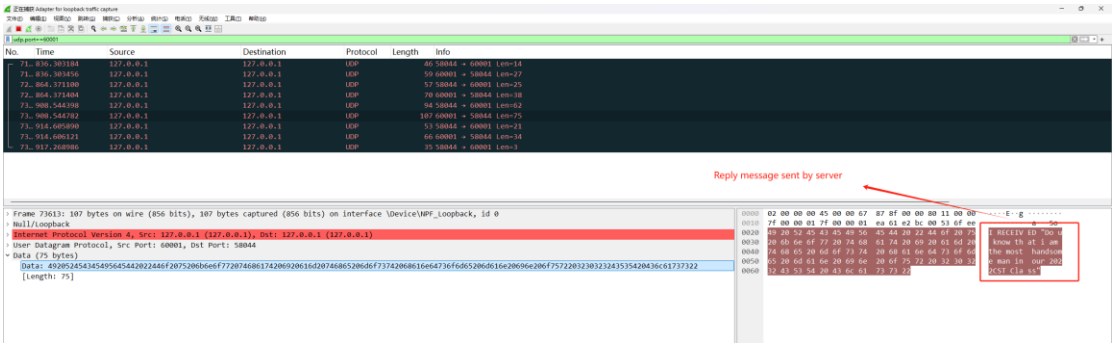

Figure 19: One of the Captured message replied by udpserver.exe

3. Answer the questions:

Q1: Is UDP a connection-oriented protocol? Explain the advantages and disadvantages of this feature.

A1: UDP is a connectionless protocol, which results in lower transmission overhead and reduced latency, making it suitable for applications that require speed and simplicity, such as live video or voice streaming. However, it lacks reliability features; there's no guarantee for message delivery, order, or protection against duplicates. Applications using UDP must handle error detection and correction themselves and

Q2: Does the conversation of UDP protocol include acknowledgement datagram? Explain the advantages and disadvantages of this feature.

A2: UDP does not use acknowledgment datagrams, which allows for faster data transmission since there's no need to wait for confirmation of receipt. This is advantageous for real-time applications where speed is more critical than reliability. However, this also means that UDP does not inherently provide reliability; lost packets are not retransmitted, and it's up to the application to handle any required error checking and correction.

Q3: What happened if the client sends a UDP message to a port that is not available on the server?

A3: If a client sends a UDP message to a port that is not available or not listening on the server, the server typically responds with an ICMP Destination Unreachable message, specifically with a port unreachable error. However, if ICMP messages are blocked or the server is configured not to send them, the client will not receive any indication of the issue and may remain unaware that the message was not received. UDP itself does not provide any mechanism for detecting such conditions..

4. Others

4.1 Answer to the questions

Q1: Can UDP use all "0" as the checksum?

A1: Yes, UDP can utilize a checksum of all zeros. When a UDP packet has a checksum value of zero, it signifies that the sender opted not to calculate the checksum. In this case, the receiver has the discretion to verify the packet's integrity. If the receiver decides to conduct an integrity check, it should ignore the checksum field as if it were not present.

Q2: In a LAN, a computer sends out a UDP message. If the destination MAC address is the broadcast MAC address and the destination IP address is the IP address of a host within the LAN, what would happen?

A2: In a LAN, if a computer sends out a UDP message with the destination MAC address set to the broadcast MAC address (FF:FF:FF:FF:FF:FF) and the destination IP address set to the IP address of a host within the LAN, the following would occur:
The frame would be delivered to every host in the LAN because the broadcast MAC address directs the network devices to send the frame to all devices on the local network. Each host on the LAN would then process the frame and check the IP address in the packet. Only the host with a matching IP address would process the UDP message further by checking the destination port number in the UDP header. If an application on the host is listening on that destination port, the UDP message would be delivered to that application. If not, the host might respond with an ICMP Destination Unreachable message, although this response is not guaranteed and depends on the host's configuration.

Q3: Briefly explain the ability of UDP on handling errors, e.g., bit flips, packet loss, duplication, and out-of-order?

A3: UDP employs a basic approach to error handling. It utilizes a checksum to detect data corruption, such as bit flips, within a packet. If the checksum validation fails, the packet is typically discarded.

However, UDP does not inherently address packet loss, duplication, or out-of-order delivery. It lacks mechanisms for retransmission, acknowledgment, or sequence control. These aspects must be managed by the application layer, which is responsible for implementing any necessary error recovery or data reordering processes. This design keeps UDP lightweight and fast, but it also means that higher-layer protocols or applications need to handle transmission reliability.

Q4: Use your own words to compare the reliability of UDP and IP?

A3: IP, a fundamental protocol in the internet layer, primarily handles routing packets across networks. Its unreliability arises from the absence of guarantees for packet delivery, lacking mechanisms to ensure packets reach their destination. IP does not track packet sequences or perform error correction beyond basic header checksums. Consequently, packets may be dropped due to network congestion or faulty routing paths without notifying the sender.

UDP, operating in the transport layer, builds upon IP's basic functionality but similarly lacks reliability features. UDP transmits data as datagrams without

establishing connections, thus missing mechanisms for packet delivery acknowledgment, retransmissions of lost packets, sequencing to reorder out-of-order packets, and handling of duplicated packets. While UDP includes a checksum for basic error detection in the header and payload, this is optional and does not provide comprehensive error correction.

Both protocols prioritize speed and efficiency over reliability. IP is concerned with routing packets between devices across multiple networks, whereas UDP facilitates packet transmission to specific applications within devices. Neither protocol ensures successful or orderly delivery, leaving such tasks to higher-layer protocols or specific application implementations.

4.2 Some personal thoughts after this lab

In this lab, I explored the structure of the UDP header and gained a more comprehensive understanding of the UDP protocol by capturing various packets and developing an application to send and receive UDP data packets. This experience has enhanced my grasp of the protocols within the computer network transport layer.