# Department of Computer Science

**EE353: Computer Networks**

**Name:** M. Hasnain naeem

**Reg #:** 212728

**Class:** BSCS-7B

# Lab 12: *Analysis of UDP and TCP packets in Wireshark*

# Date: 9th DEC 2019

**Lab Engineer: Kaleem Ullah**

# Instructor: Dr. Zeeshan

***Lab Title:*** *Analysis of UDP and TCP packets in Wireshark*

***Objective of this lab:***

*In this lab, we will analyze the behavior of UDP in detail, determining the number of fields in UDP header, the value in the UDP header fields, and maximum number of bytes in UDP payload, source & destination port numbers etc.*

*we’ll also investigate the behavior of TCP in detail. We’ll do so by analyzing TCP segments sent and received from your computer to a remote server. We’ll study TCP’s use of sequence and acknowledgement numbers for providing reliable data transfer.*

***Instructions:***

* *Read carefully before starting the lab.*
* *These exercises are to be done individually.*
* *You are supposed to provide the answers to the questions listed at the end of this document and upload the completed report to your course’s LMS site.*
* *Avoid plagiarism by copying from the Internet or from your peers. You may refer to source/ text but you must paraphrase the original work.*

***Background:***

1. ***Introduction to UDP:***

UDP (User Datagram Protocol) is a simple transport layer protocol for client/server network applications based on [Internet Protocol (IP)](http://compnetworking.about.com/od/networkprotocolsip/g/ip_protocol.htm). UDP is the main alternative to TCP and one of the oldest network protocols in existence, introduced in 1980. UDP is often used in videoconferencing applications or computer games specially tuned for real-time performance. To achieve higher performance, the protocol allows individual packets to be dropped (with no retries) and UDP packets to be received in a different order than they were sent as dictated by the application.

1. ***UDP Datagrams:***

UDP network traffic is organized in the form of datagrams. A datagram comprises one message unit. The first eight (8) bytes of a datagram contain header information and the remaining bytes contain message data.

A UDP datagram header consists of four (4) fields of two bytes each: Source port number, Destination port number, Datagram size and checksum

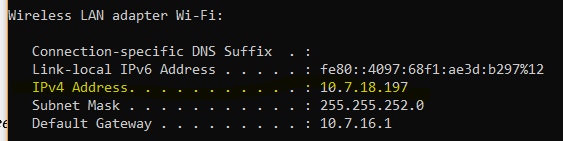
* 1. **UDP port number:**UDP [port numbers](http://compnetworking.about.com/od/networkprotocols/f/port-numbers.htm) allow different applications to maintain their own channels for data similar to TCP. UDP port headers are two bytes long.
  2. **Datagram size:**The UDP datagram size is a count of the total number of bytes contained in header and data sections. As the header length is a fixed size, this field effectively tracks the length of the variable-sized data portion (sometimes called payload). The size of datagrams varies depending on the operating environment but has a maximum of 65535 bytes.
  3. **Checksum**: UDP checksums protect message data from tampering. The checksum value represents an encoding of the datagram data calculated first by the sender and later by the receiver. Should an individual datagram be tampered with or get corrupted during transmission, the UDP protocol detects a checksum calculation mismatch. In UDP, check-summing is optional as opposed to TCP where checksums are mandatory.

***Steps for performing this lab:***

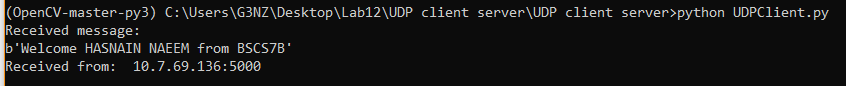
*Do the following:*

1. ***Download*** *files UDPCient.py and UDPServer.py from your LMS site.*
2. ***Edit*** *these files. In UDPClient.py TheserverIP address; use one of your neighbor and the message; as your name. In UDPServer.py use your own IP address*
3. ***Start up the Wireshark software.***
4. ***Begin packet capture,*** *select the Capture pull down menu and select Options.*
5. ***Selecting the network interface on which packets would be captured:*** *You can use most of the default values in this window. The network interfaces (i.e., the physical connections) that your computer has to the network will be shown in the Interface pull down menu at the top of the Capture Options window. Click Start. Packet capture will now begin*
6. ***Run your UDPServer and UDPClient.***
7. ***Stopping the capture and inspecting captured packets:*** *After you have received a welcome message, stop Wireshark packet capture*
8. ***Filtering:*** *Filter the UDP packets.*
9. ***Details of a packet:*** *Select theUDP messages shown in the packet-listing window and analyze by looking into the detail of packets pane and answer the questions given at the end of this document.*
10. ***Obtaining credit for this lab:*** *Now, please proceed to the questions section to answer the questions. You must note down your answers,along with screen shots in this file itself. Please note that you must upload this file (after duly filling in the answers) through the appropriate link at your LMS to obtain credit. Please clarify with your instructor/ lab engineer if you have any queries.*

**IP Information:**

**

***Task 1 Questions:***



1. *Select one UDP packet and determine the* ***Source IP, Source port No, Destination IP and Destination port No*** *of that UDP packet.*

**Server IP & Port:** 10.7.69.136:5000

**Client IP & Port:** 10.7.18.197:53093

1. *Select one UDP packet and determine how many* ***fields*** *are there in the UDP header. List* ***the name of these fields****.*

There are 4 fields. Namely:

* Source port
* Destination port
* Length
* Checksum

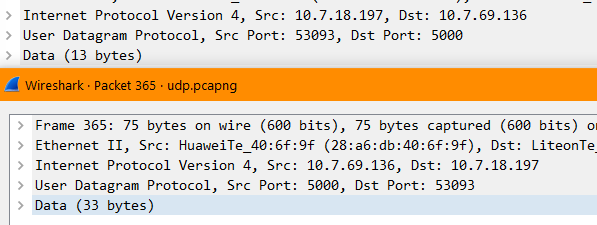
1. *From the packet content field, determine* ***the length (in bytes) of each of the UDP header*** *fields.*

* Source port: 2 bytes
* Destination port: 2 bytes
* Length: 2 bytes
* Checksum: 2 bytes

1. ***Examine the pair of UDP packets*** *in which your host sends the first packet and the second packet is a reply to the first packet. Describe the relationship between the port numbers in the two packets.*

In first packet, source port was that of client and destination port was that of server.

In second packet, positions were switched – server’s port was in the source field and client’s port was in the destination field.

**

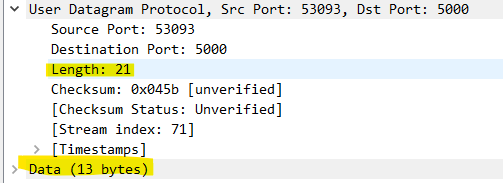
1. *Analyze the UDP packet and answer that the* ***value in the Length field*** *is the length of what? Verify your claim with your captured UDP packet.*

Value in the length field = length of the data + header size (8 bytes)

**Verification of claim:**

In the client’s packet, data length = 13.

So, the length field should 13 + 8 = 21.

****

1. *What is the* ***maximum number of bytes*** *that can be included in a UDP payload? Why this is the maximum?*

65,535 bytes is the theoretical limit of the UDP payload which includes the 8 bytes of header.

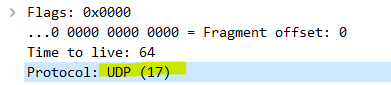
This theoretical limit is because the “length” field in the header can contain only 2 bytes. So, maximum value which it can hold is 216 -1 = 65,535.

1. *What is the* ***largest possible source port number****?*

Largest possible port number = 216 - 1 = 65,535. Because, “source port” field can hold only 16 bits.

1. *What is the* ***protocol number for UDP****? Give your answer in both hexadecimal and decimal notation.*

**Protocol number:** 11d OR 0x11.



1. *Which fields are included in calculating the UDP checksum?*

Following are included the calculation of UDP checksum:

* Header fields
  1. Source port
  2. Destination port
  3. Length
* Data in the packet

1. ***The Transmission Control Protocol (TCP) Introduction:***

TCP provides connections between clients and servers. A TCP client establishes a connection with a given server, exchanges data with that server across the connection, and then terminates the connection.

TCP also provides **reliability**. When TCP sends data to the other end, it requires an acknowledgment in return. If an acknowledgment is not received, TCP automatically retransmits the data and waits a longer amount of time. After some number of retransmissions, TCP will give up, with the total amount of time spent trying to send data typically between 4 and 10 minutes (depending on the implementation).

TCP contains algorithms to estimate the **round-trip time** (RTT) between a client and server dynamically so that it knows how long to wait for an acknowledgment. For example, the RTT on a LAN can be milliseconds while across a WAN it can be in seconds. Furthermore, TCP continuously estimates the RTT of a given connection, because the RTT is affected by variations in the network traffic.

TCP also sequences the data by associating a **sequence number** with every byte that it sends. For example, assume an application writes 2,048 bytes to a TCP socket, causing TCP to send two segments, the first containing the data with sequence numbers 1–1,024 and the second containing the data with sequence numbers 1,025–2,048. (A segment is the unit of data that TCP passes to IP.) If the segments arrive out of order, the receiving TCP will reorder the two segments based on their sequence numbers before passing the data to the receiving application. If TCP receives duplicate data from its peer (say the peer thought a segment was lost and retransmitted it, when it wasn't really lost, the network was just overloaded), it can detect that the data has been duplicated (from the sequence numbers), and discard the duplicate data.

In contrast to TCP, there is no reliability provided by UDP. UDP itself does not provide anything like acknowledgments, sequence numbers, RTT estimation, timeouts, or retransmissions. If a UDP datagram is duplicated in the network, two copies can be delivered to the receiving host. Also, if a UDP client sends two datagrams to the same destination, they can be reordered by the network and arrive out of order.

TCP provides **flow control**. TCP always tells its peer exactly how many bytes of data it is willing to accept from the peer at any one time. This is called the advertised window. At any time, the window is the amount of room currently available in the receive buffer, guaranteeing that the sender cannot overflow the receive buffer. The window changes dynamically over time: As data is received from the sender, the window size decreases, but as the receiving application reads data from the buffer, the window size increases. It is possible for the window to reach 0: when TCP's receive buffer for a socket is full and it must wait for the application to read data from the buffer before it can take any more data from the peer.

Finally, a TCP connection is **full-duplex**. This means that an application can send and receive data in both directions on a given connection at any time. This means that TCP must keep track of state information such as sequence numbers and window sizes for each direction of data flow: sending and receiving. After a full-duplex connection is established, it can be turned into a simplex connection if desired.

1. ***TCP Connection Establishment and Termination***

#### *2.1 Three-Way Handshake:*

Following scenario occurs when a TCP connection is established:

1. The server must be prepared to accept an incoming connection. This is normally done by calling socket, bind, and listen and is called a passive open.
2. The client issues an active open by calling connect. This causes the client TCP to send a "synchronize" (SYN) segment, which tells the server the client's initial sequence number for the data that the client will send on the connection. Normally, there is no data sent with the SYN; it just contains an IP header, a TCP header, and possible TCP options (which we will talk about shortly).
3. The server must acknowledge (ACK) the client's SYN and the server must also send its own SYN containing the initial sequence number for the data that the server will send on the connection. The server sends its SYN and the ACK of the client's SYN in a single segment.
4. The client must acknowledge the server's SYN.

The minimum number of packets required for this exchange is three; hence, this is called TCP's three-way handshake. We show the three segments in Figure 1

##### *Figure 1 TCP three-way handshake.*

We show the client's initial sequence number as J and the server's initial sequence number as K. The acknowledgment number in an ACK is the next expected sequence number for the end sending the ACK. Since a SYN occupies one byte of the sequence number space, the acknowledgment number in the ACK of each SYN is the initial sequence number plus one. Similarly, the ACK of each FIN is the sequence number of the FIN plus one.

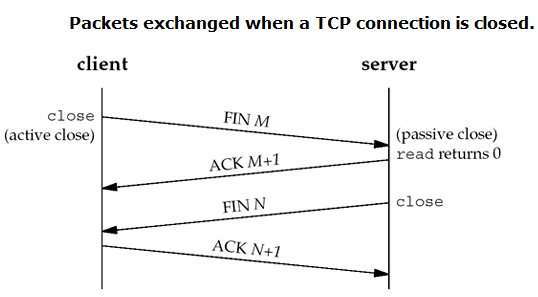
An everyday analogy for establishing a TCP connection is the telephone system. The **socket** function is the equivalent of having a telephone to use. **bind** is telling other people your telephone number so that they can call you. **listen** is turning on the ringer so that you will hear when an incoming call arrives. **connect** requires that we know the other person's phone number and dial it. **accept** is when the person being called answers the phone. Having the client's identity returned by accept (where the identify is the client's IP address and port number) is similar to having the caller ID feature show the caller's phone number. One difference, however, is that accept returns the client's identity only after the connection has been established, whereas the caller ID feature shows the caller's phone number before we choose whether to answer the phone or not.

#### *2.2. TCP Connection Termination*

While it takes three segments to establish a connection, it takes four to terminate a connection.

1. One application calls close first, and we say that this end performs the active close. This end's TCP sends a FIN segment, which means it is finished sending data.
2. The other end that receives the FIN performs the passive close. The received FIN is acknowledged by TCP. The receipt of the FIN is also passed to the application as an end-of-file (after any data that may have already been queued for the application to receive), since the receipt of the FIN means the application will not receive any additional data on the connection.
3. Sometime later, the application that received the end-of-file will close its socket. This causes its TCP to send a FIN.
4. The TCP on the system that receives this final FIN (the end that did the active close) acknowledges the FIN.

Since a FIN and an ACK are required in each direction, four segments are normally required. We use the qualifier "normally" because in some scenarios, the FIN in Step 1 is sent with data. Also, the segments in Steps 2 and 3 are both from the end performing the passive close and could be combined into one segment. We show these packets in Figure 2.

**

##### *Figure 2 Packets exchanged when a TCP connection is closed.*

A FIN occupies one byte of sequence number space just like a SYN. Therefore, the ACK of each FIN is the sequence number of the FIN plus one.

Between Steps 2 and 3 it is possible for data to flow from the end doing the passive close to the end doing the active close. This is called a half open connection.

The sending of each FIN occurs when a socket is closed. We indicated that the application calls close for this to happen, but realize that when a Unix process terminates, either voluntarily (calling exit or having the main function return) or involuntarily (receiving a signal that terminates the process), all open descriptors are closed, which will also cause a FIN to be sent on any TCP connection that is still open.

Although we show the client in Figure 2 performing the active close, either end—the client or the server—can perform the active close. Often the client performs the active close, but with some protocols (notably HTTP), the server performs the active close.

*Steps for obtaining credit for this lab.*

*Do the following:*

1. ***Download*** *files TCPCient.py and TCPServer.py from your LMS site.*
2. ***Edit*** *the client file to change two things. The serverIP address and the message; as your name.*
3. ***Start up the Wireshark software.***
4. ***Begin packet capture,*** *select the Capture pull down menu and select Options.*
5. ***Selecting the network interface on which packets would be captured:*** *You can use most of the default values in this window. The network interfaces (i.e., the physical connections) that your computer has to the network will be shown in the Interface pull down menu at the top of the Capture Options window. Click Start. Packet capture will now begin*
6. ***Run your TCPServer and your edited TCPClient.***
7. ***Stopping the capture and inspecting captured packets:*** *After you have received a message, stop Wireshark packet capture*
8. ***Filtering:*** *Filter the TCP packets.*
9. ***Details of a packet:*** *Select the TCP messages shown in the packet-listing window and analyze by looking into the detail of packets pane and answer the questions given at the end of this document.*

**Task 2 Questions:**



1. **What are the IP addresses and TCP port numbers used by the client and the server?**

**Server IP & Port:** 10.7.69.136:6000

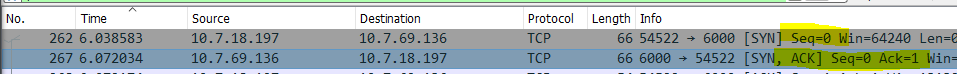
**Client IP & Port:** 10.7.18.197:54522



1. **What are the SEQ and ACK Nos of the TCP SYN segment that is used to initiate the TCP connection between the client and server?**

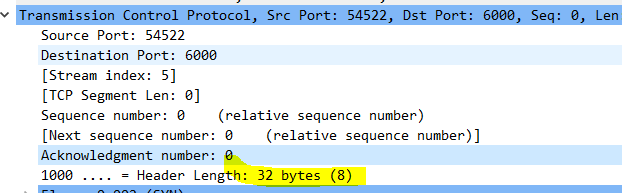
Seq #: 0

Ack #: 1



1. **What is the header length of TCP used for this connection?**

32 bytes.



1. **What is the minimum amount of available buffer space advertised by the client and the server?**

64240 bytes.

****

1. **What is the sequence number of the SYNACK segment sent by server to the client computer in reply to the SYN? What is the value of the ACK field in the SYNACK segment? How did the server determine that value?**

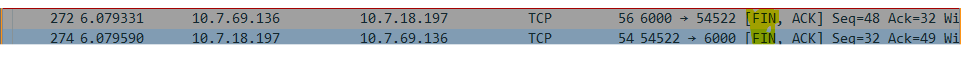
SYNACK segment ACK number: 1

It is determined by adding 1 to the received sequence number, which was 0.



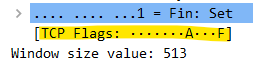
1. **Who has done the active close? Client or the server? How you have determined this?**

Initially client closed the connection.



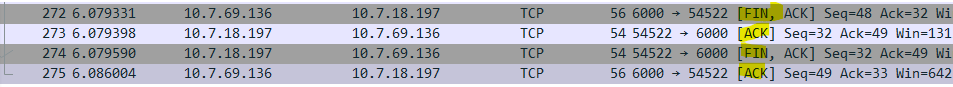


Then, server also closed its tunnel.



1. **What type of closure has been performed? 3 segment (FIN/FIN-ACK/ACK) or four segments (FIN/ACK/FIN/ACK) or simultaneous close?**

Four segment closure (FIN/ACK/FIN/ACK).



1. **What are SEQ and ACK Nos for all the segments used for the connection closure?**

**Segment # 1 (client to server)**

Seq #: 48

Ack #: 32

**Segment # 2 (server to client)**

Seq #: 32

Ack #: 49

**Segment # 3 (server to client)**

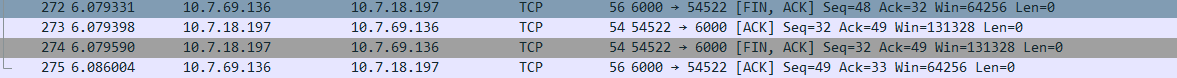
Seq #: 32

Ack #: 49

**Segment # 4 (server to client)**

Seq #: 49

Ack #: 33



1. **How many bytes in total have been transferred from the client to the server and from the server to the client during the whole connection?**

As the sequence number started from 0. So, it is safe to determine number of bytes transferred from the Ack numbers in the closing segments.

Number of byte transferred = Ack number – 1

**Bytes transferred are:**

* + **Client to server:** 32 (excluding the header bytes)
  + **Server to client:** 48 (excluding the header bytes)

**Conclusion:**

UDP and TCP protocols are used in the protocol layer. UDP has limited header size and functionality. TCP provides reliable data transferred but it involves more header size and time overhead. Moreover, Wireshark can be used to analyze both TCP & UDP packets.