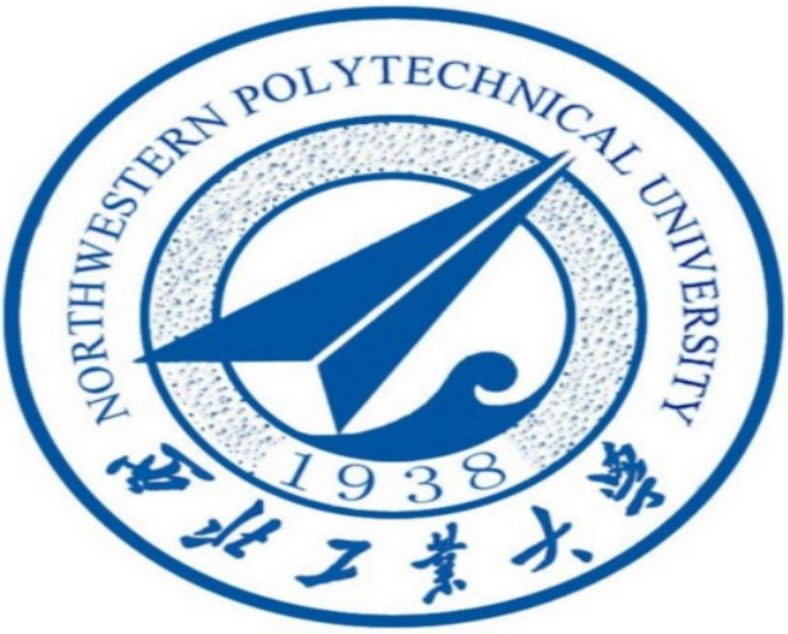

ComputerNetworks

Homework 1

Student Name : ABID ALI
Student ID : 2019380141
Student Email : abiduu354@gmail.com / **3616990795@qq.com**
Lecture : Professor Bo Yang
Submission : 10/25/22
Submitted Email : liu1218446147@mail.nwpu.edu.cn

1. Capture TCP 3-way handshake via Wireshark.

Solution:

Three-Way HandShake or a TCP 3-way handshake is a process which is used in a TCP/IP network to make a connection between the server and client. It is a three-step process that requires both the client and server to exchange synchronization and acknowledgment packets before the real data communication process starts.

TCP message types

Message	Description
Syn	Used to initiate and establish a connection. It also helps you to synchronize sequence numbers between devices.
ACK	Helps to confirm to the other side that it has received the SYN.
SYN-ACK	SYN message from local device and ACK of the earlier packet.
FIN	Used to terminate a connection.

TCP Three-Way Handshake Process

TCP traffic begins with a three-way handshake. In this TCP handshake process, a client needs to initiate the conversation by requesting a communication session with the Server:

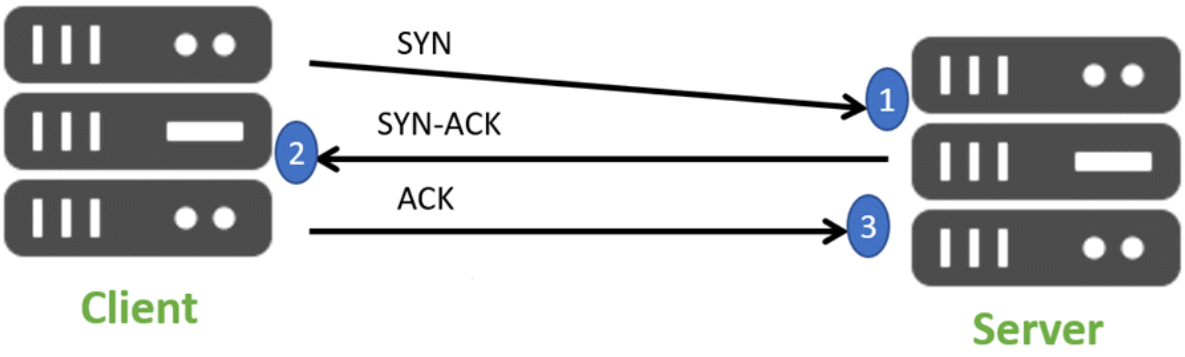

Figure: 3 way Handshake Diagram

- **Step 1:** In the first step, the client establishes a connection with a server. It sends a segment with SYN and informs the server about the client should start communication, and with what should be its sequence number.

- **Step 2:** In this step server responds to the client request with SYN-ACK signal set. ACK helps you to signify the response of segment that is received and SYN signifies what sequence number it should able to start with the segments.

- **Step 3:** In this final step, the client acknowledges the response of the Server, and they both create a stable connection will begin the actual data transfer process.

Process of capturing

Step 1: Start browser and wait few minutes.

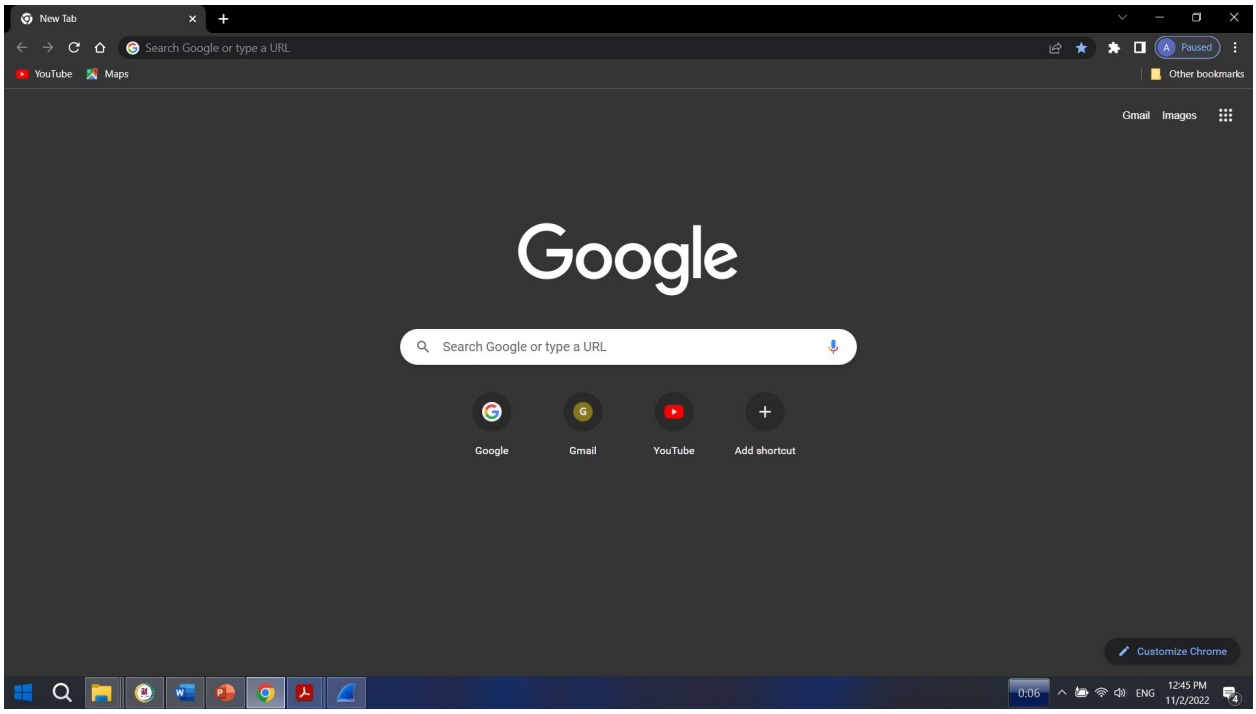

Figure: Opening The Browser

Step 2: Start Wireshark and wait wait few minutes .

Double-click the Wireshark icon 

, which is located on the desktop.

Step 3: Press wifi capturing packets

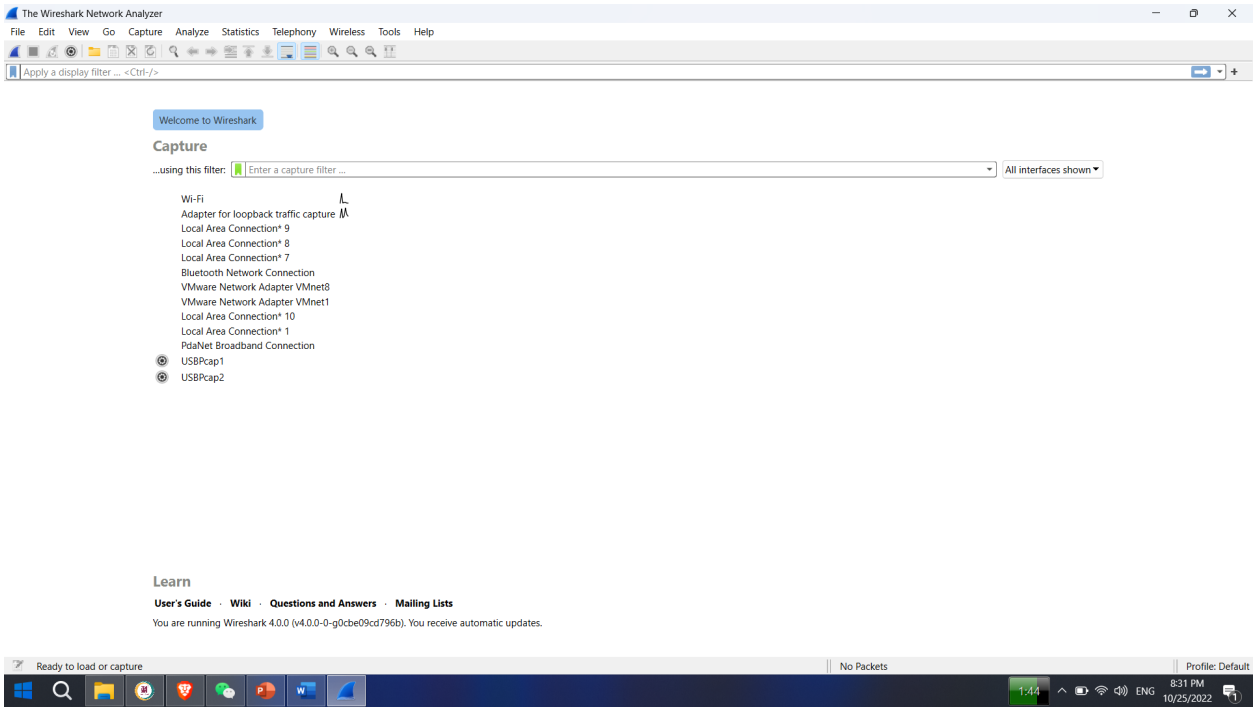

Figure: Opened Wireshark Application and pressing wifi

Step 4:

(1) Press this button  to capture network, pasted the link the browser http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html

(2) When we saw the GET http message along with TCP 3 way handshake ,we press stop the button  .

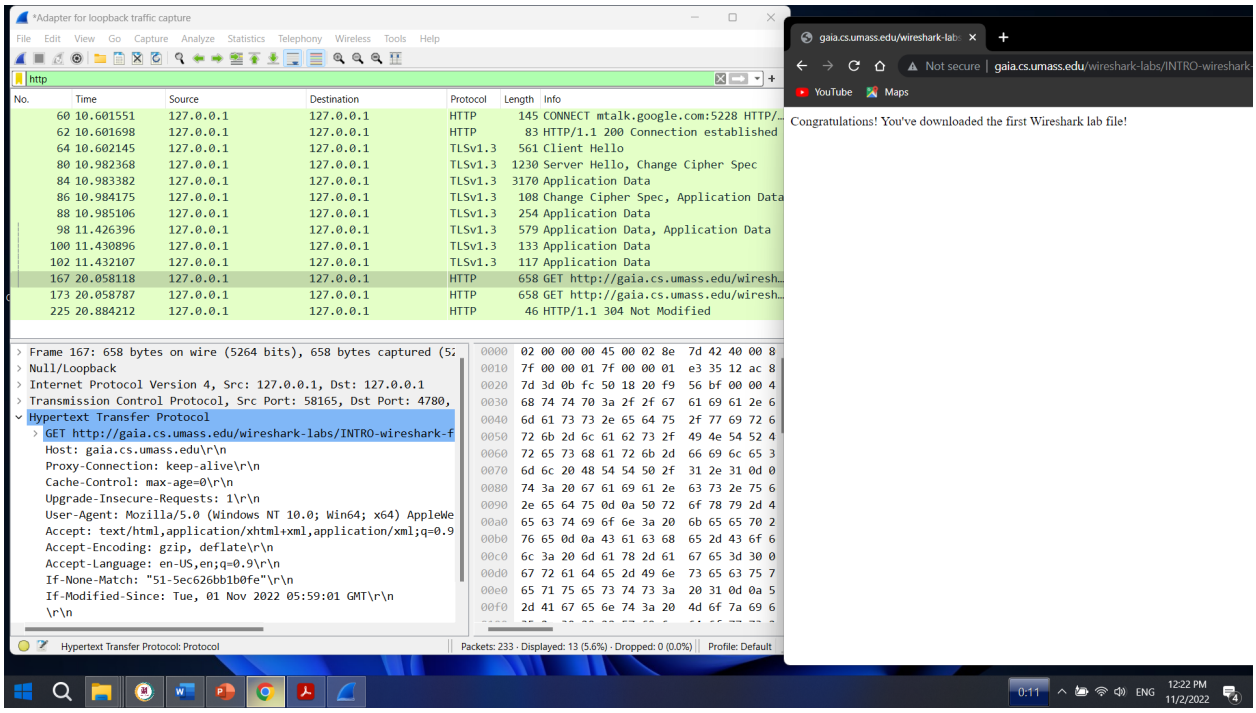

Figure 1: Http Get message

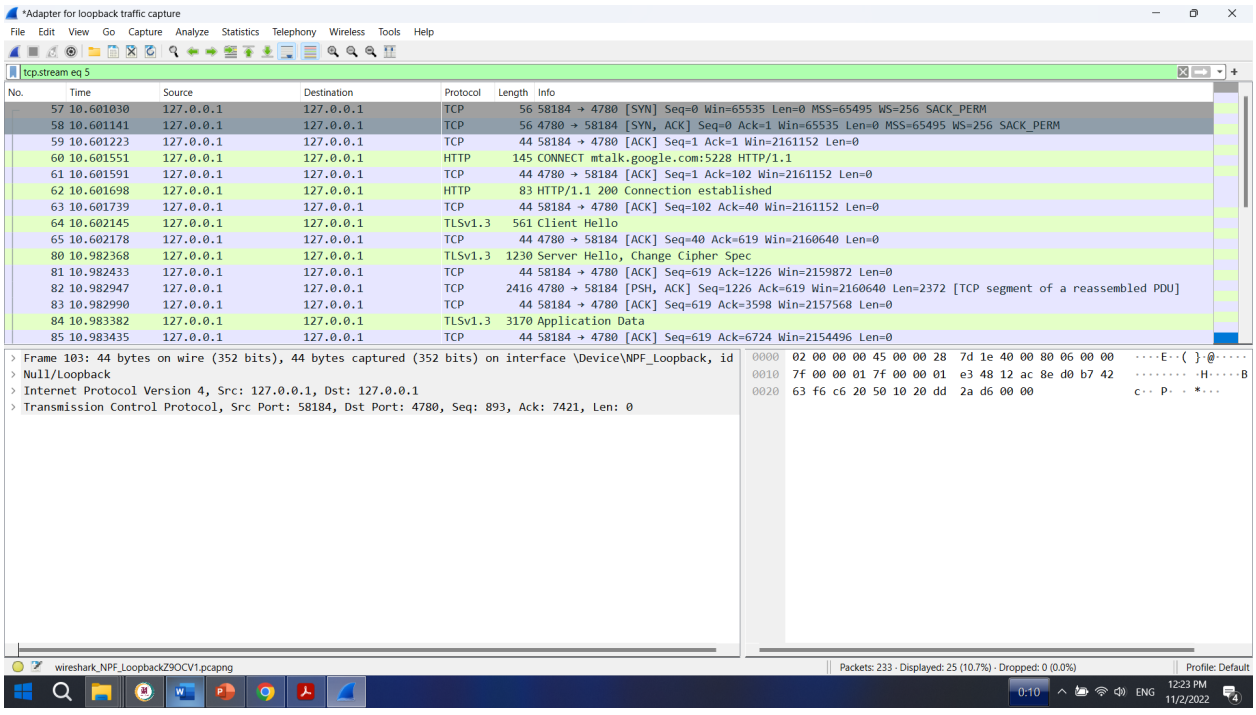

Figure 2: 3 way handshake

Step 5:

We are analyzing the packets

(1) (client) SYN

(2) (server) SYN ACK

(3) (client) ACK

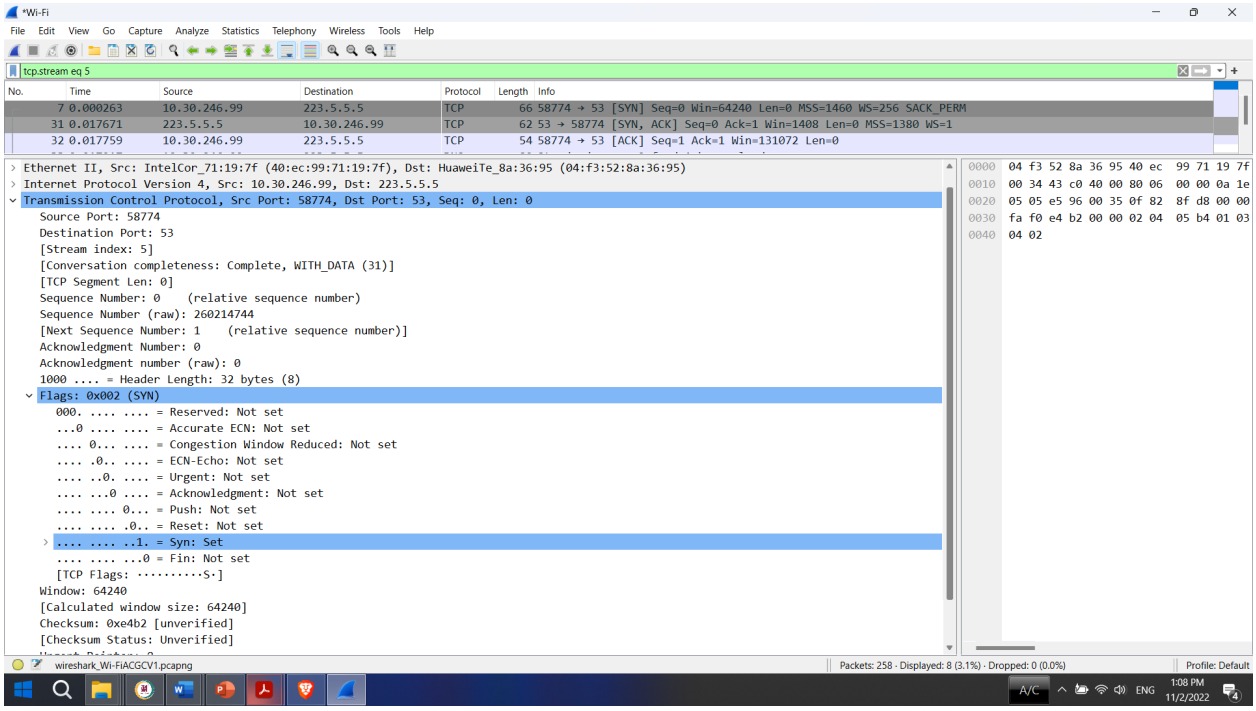

Figure 3: (client) SYN

- (client) SYN:
  - Flag = SYN
  - Seq. No = 0;

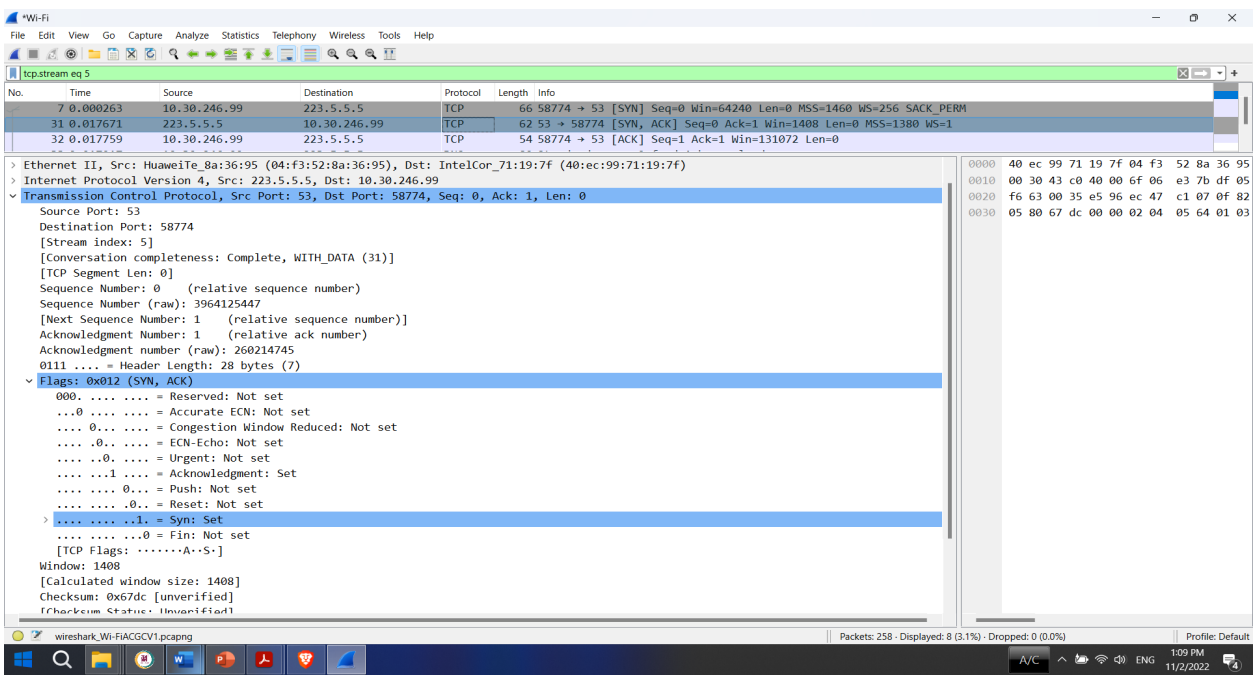

Figure 4: (server) SYN ACK

- (server) SYN ACK:
  - Flag = SYN ACK
  - Seq. No = 0;
  - ACK No = 1;

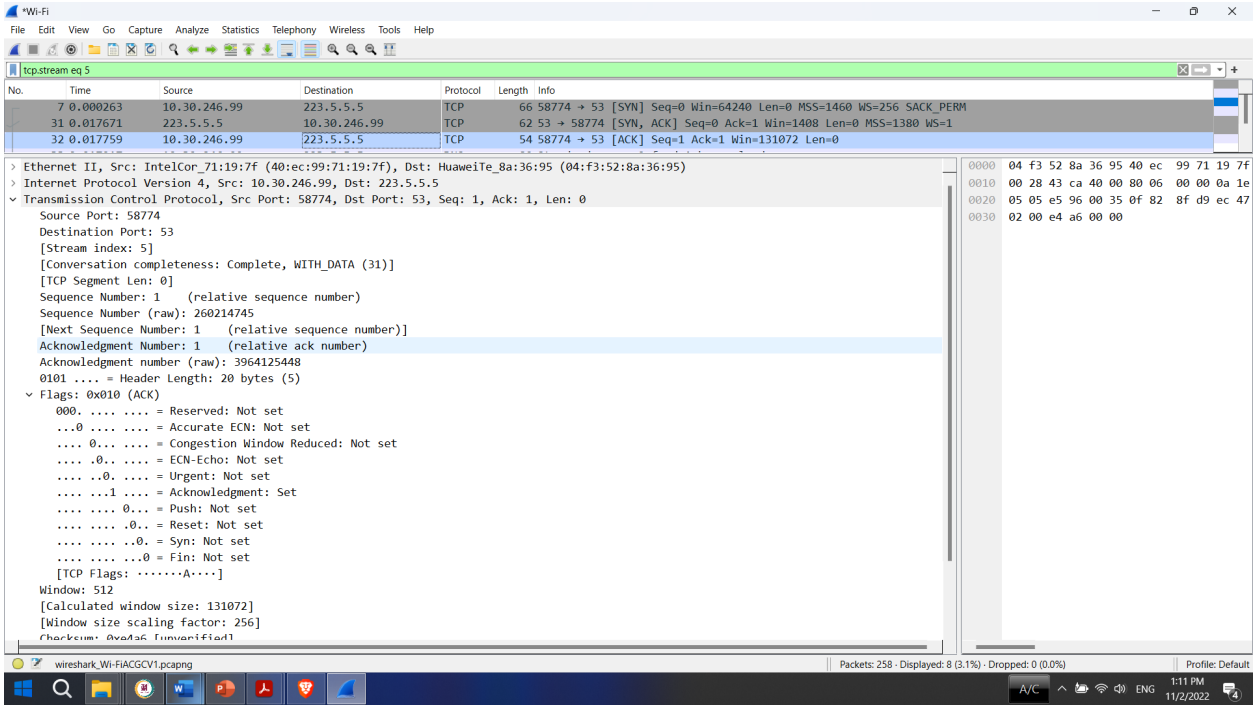

Figure 6: (client) ACK

- (client) ACK:
  - Flag = ACK;
  - Seq. No = 1;
  - ACK No = 1;

Main source 10.30.246.99(Client) and destination 223.5.5.5(Server)

We can see from the experiment the 3 way communication was done successfully.

2. Capture TCP 4-way handshake via Wireshark.

Solution

Step 1 (FIN From Client) – Suppose that the client application decides it wants to close the connection. (Note that the server could also choose to close the connection). This causes the client to send a TCP segment with the FIN bit set to 1 to the server and to enter the FIN_WAIT_1 state. While in the FIN_WAIT_1 state, the client waits for a TCP segment from the server with an acknowledgment (ACK).

Step 2 (ACK From Server) – When the Server received the FIN bit segment from Sender (Client), Server Immediately sends acknowledgement (ACK) segment to the Sender (Client).

Step 3 (Client waiting) – While in the FIN_WAIT_1 state, the client waits for a TCP segment from the server with an acknowledgment. When it receives this segment, the client enters the FIN_WAIT_2 state. While in the FIN_WAIT_2 state, the client waits for another segment from the server with the FIN bit set to 1.

Step 4 (FIN from Server) – The server sends the FIN bit segment to the Sender(Client) after some time when the Server sends the ACK segment (because of some closing process in the Server).

Step 5 (ACK from Client) – When the Client receives the FIN bit segment from the Server, the client acknowledges the server's segment and enters the TIME_WAIT state. The TIME_WAIT state lets the client resend the final acknowledgment in case the ACK is lost. The time spent by clients in the TIME_WAIT state depends on their implementation, but their typical values are 30 seconds, 1 minute, and 2 minutes. After the wait, the connection formally closes and all resources on the client-side (including port numbers and buffer data) are released.

Process of capturing

Step 1: Start browser and wait few minutes.

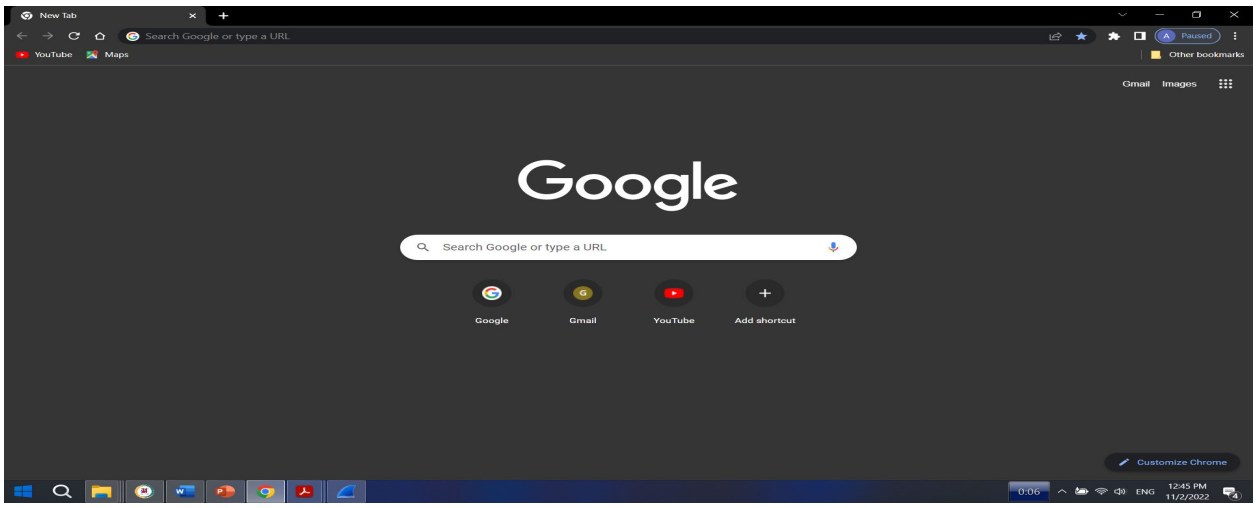

Figure: Opening The Browser

Step 2: Start Wireshark and wait wait few minutes .

Double-click the Wireshark icon 

, which is located on the desktop.

Step 3: Press wifi capturing packets

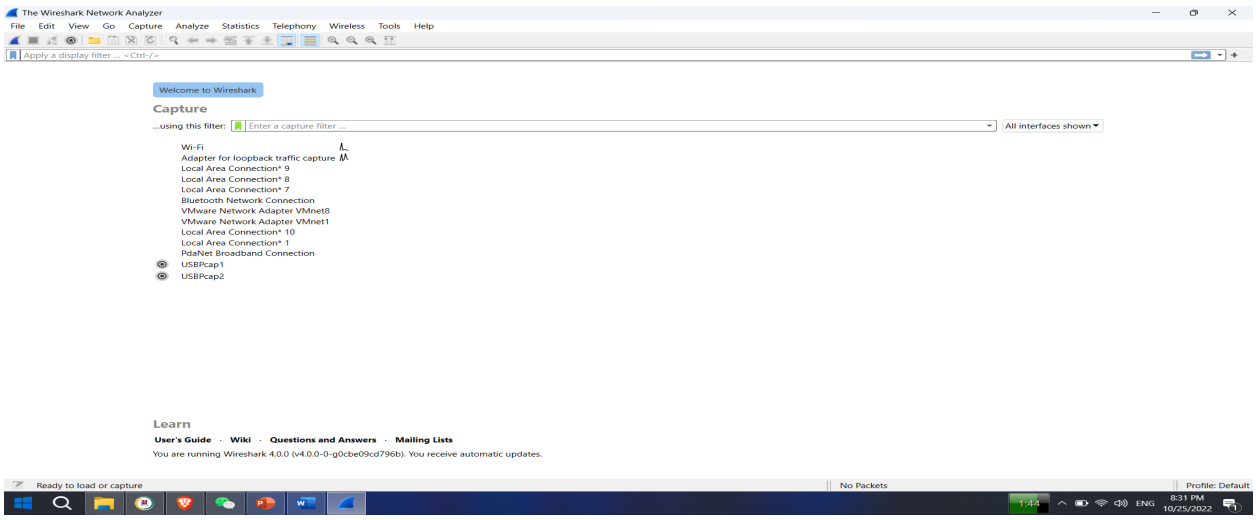

Figure: Opened Wireshark Application and pressing wifi

Step 4:

(1) Press this button  to capture network, pasted the link the browser http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html

(2) When we saw the GET http message along with TCP 4 way handshake ,we press stop the button  .

Step 5:

4-way handshake analysis via wireshark

- **FIN, ACK - ACK - FIN, ACK - ACK**

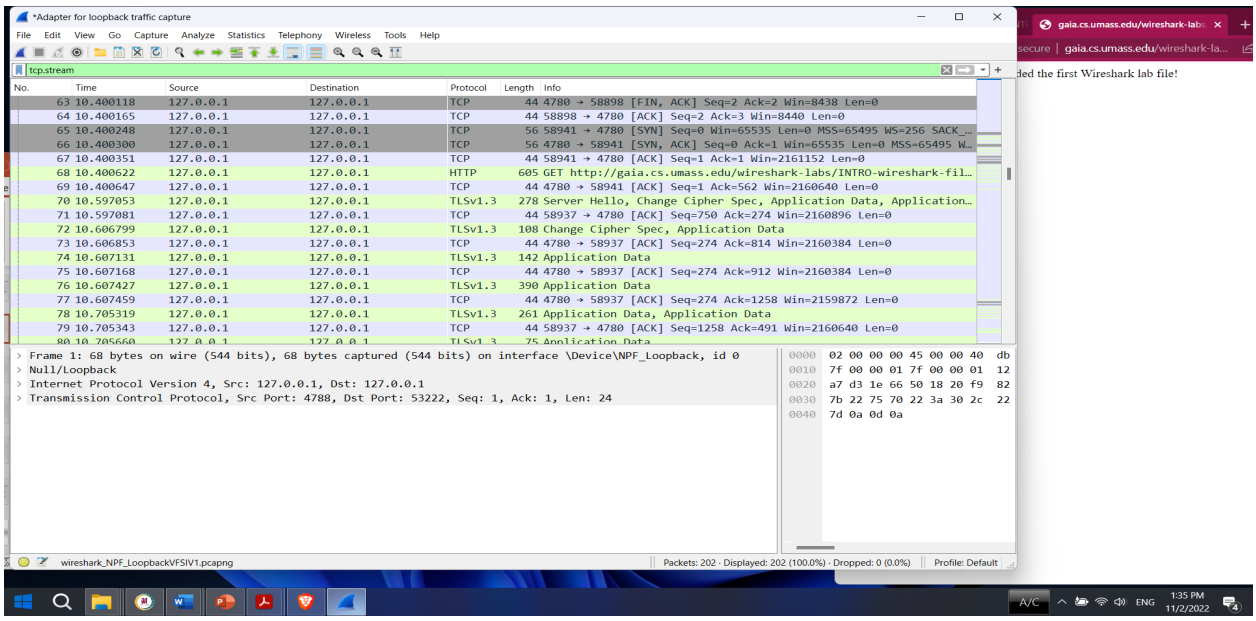

Figure 1: Http Get message

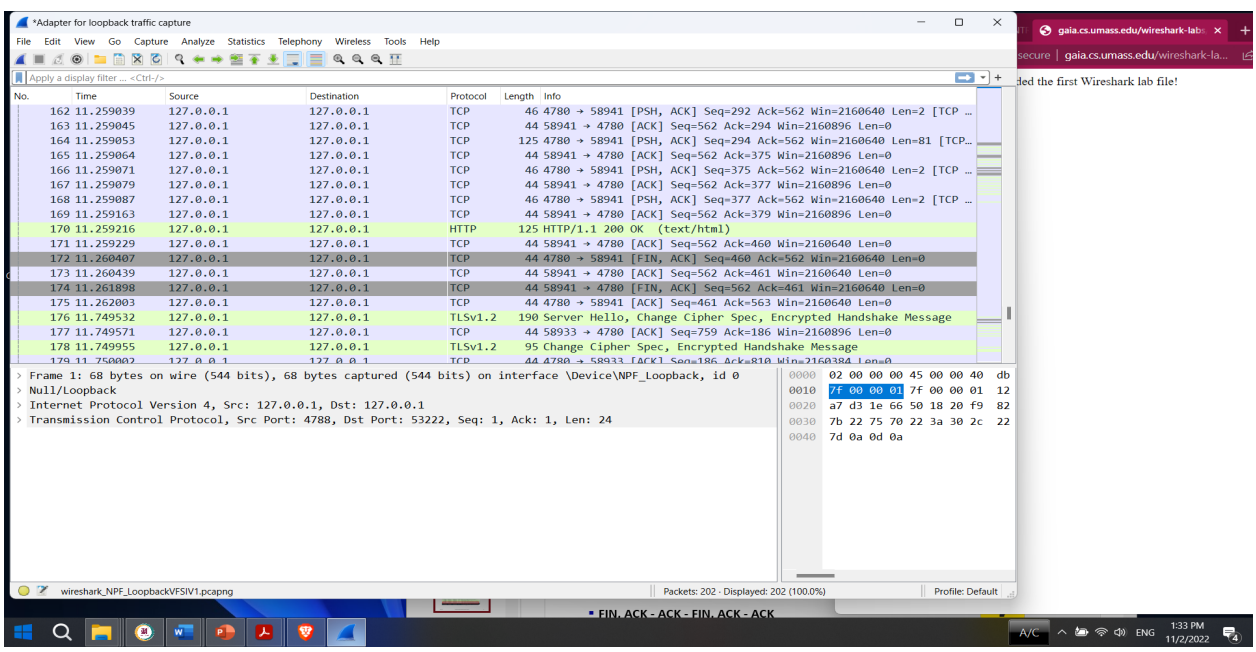

Figure : Http Get message (1)

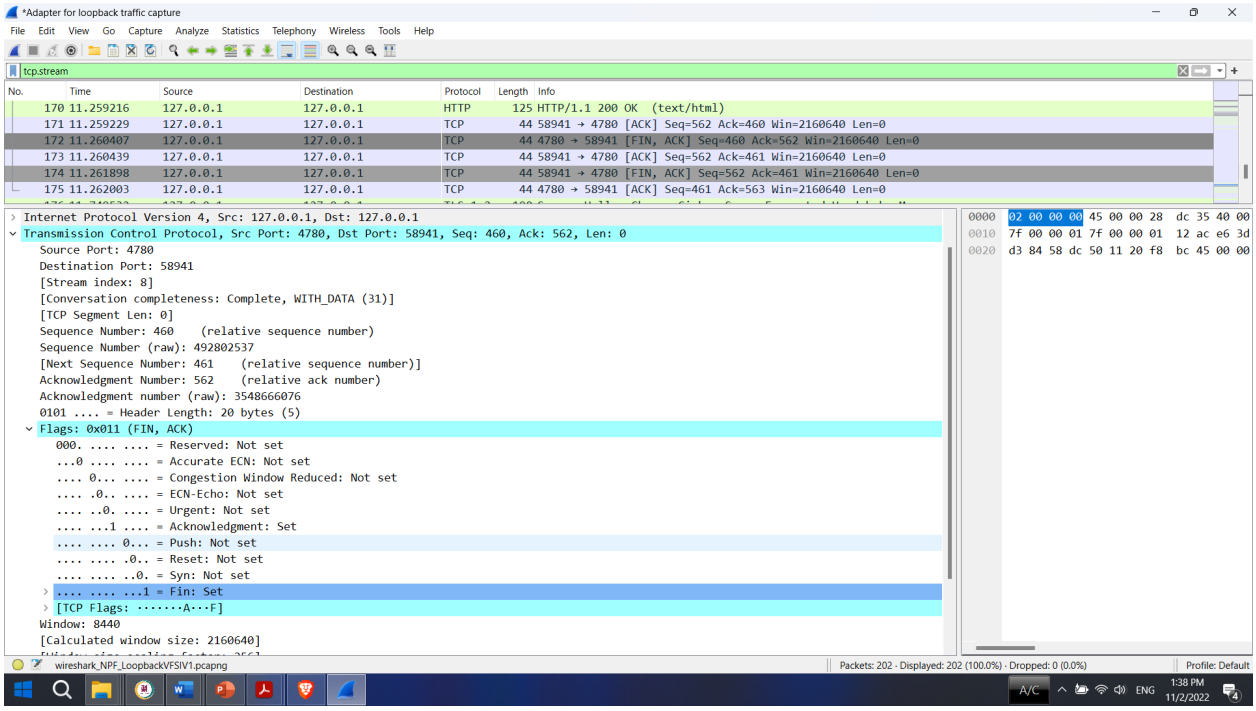

Figure : FIN,ACK (Server)

- FIN,ACK (Server)
  - Seq. No = 460;
  - ACK No = 562

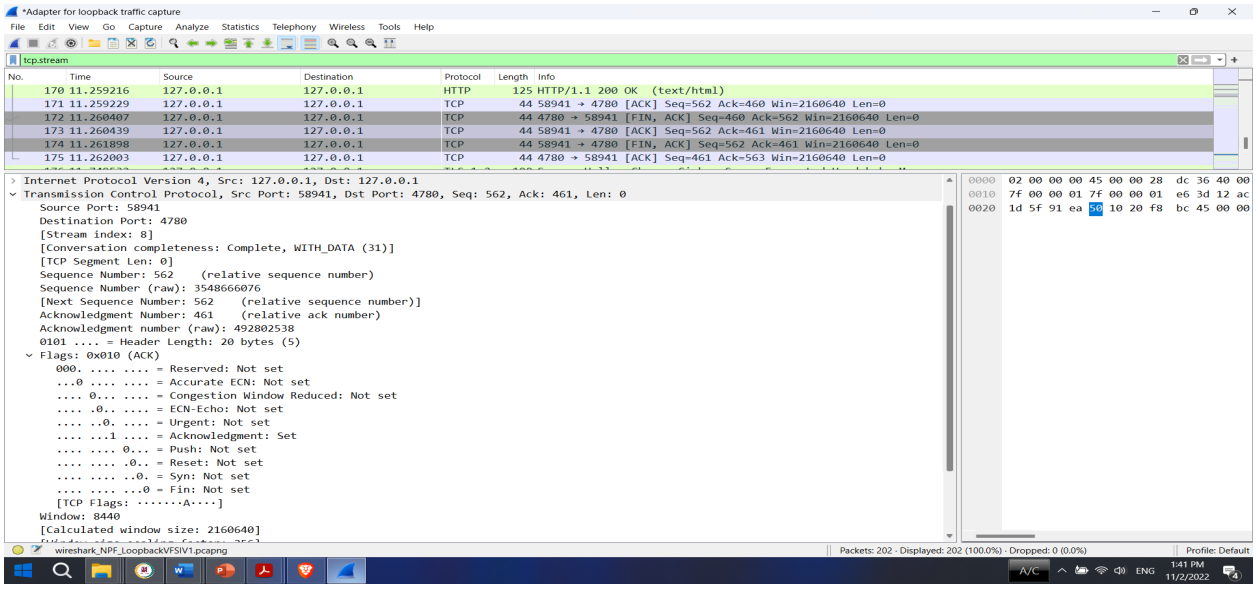

Figure : ACK (Client)

- ACK (Client)
  - Seq. No = 562;
  - ACK No = 461

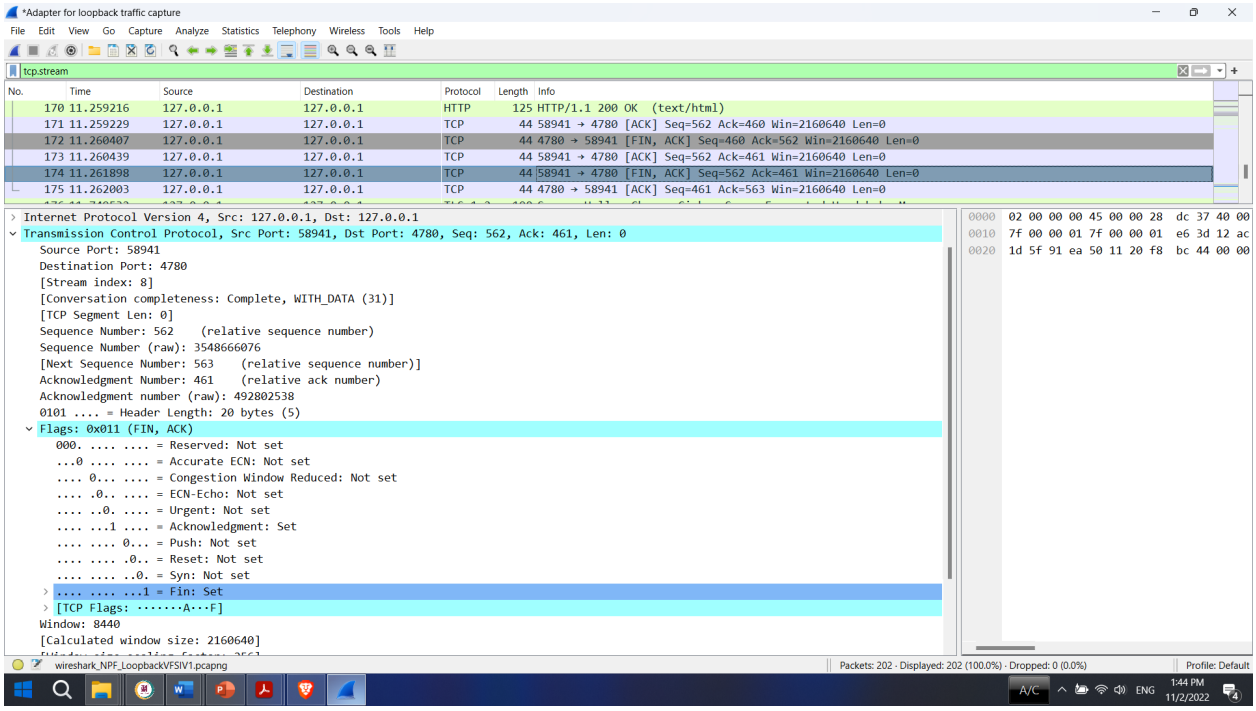

Figure : FIN, ACK (Client)

- FIN, ACK (Client)
  - Seq. No = 562;
  - ACK No = 461

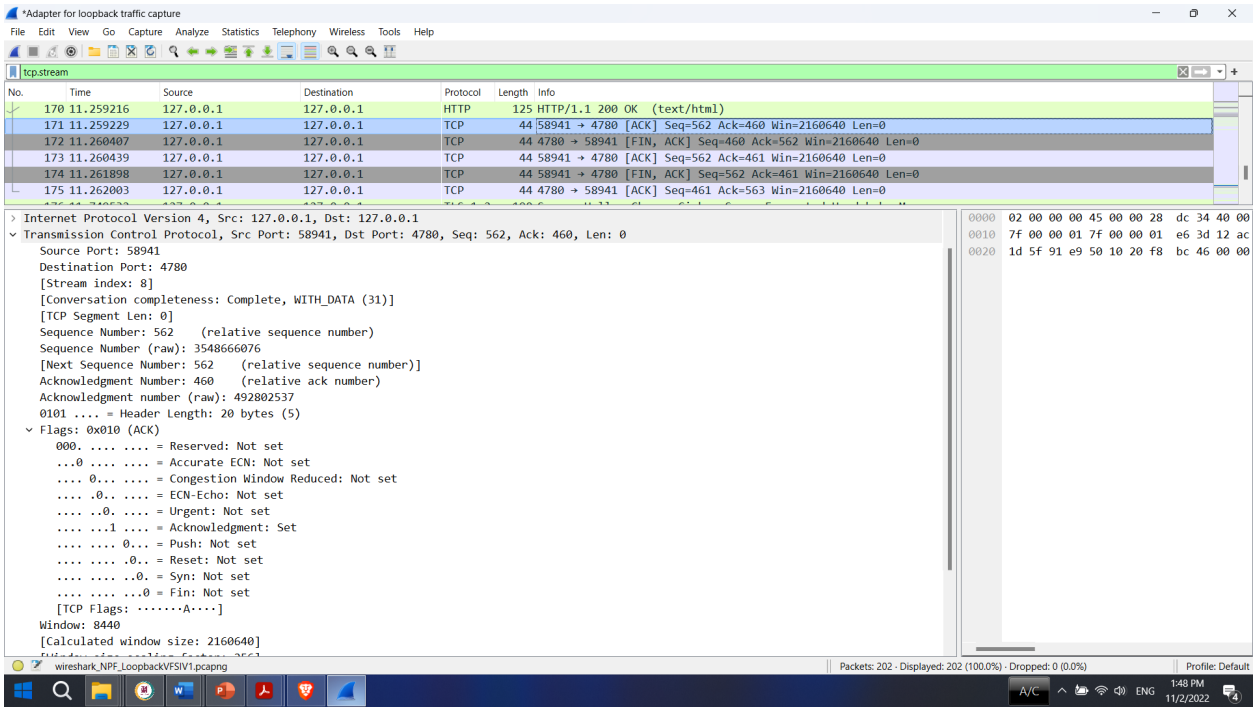

Figure : ACK (Server)

- ACK (Server)
  - ACK No = 460

3. Try the TCP sockets example (bonus)

TCP Server:

- Using create (), Create TCP socket.
- Using bind (), Bind the socket to server address.
- Using listen (), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
- Using accept (), At this point, connection is established between client and server, and they are ready to transfer data.

TCP Server:

Code

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int connfd)
{
	char buff[MAX];
	int n;
	// infinite loop for chat
	for (;;) {
		bzero(buff, MAX);

		// read the message from client and copy it in buffer
		read(connfd, buff, sizeof(buff));
		// print buffer which contains the client contents
		printf("From client: %s\t To client : ", buff);
		bzero(buff, MAX);
		n = 0;
		// copy server message in the buffer
		while ((buff[n++] = getchar()) != '\n')
			;

		// and send that buffer to client
		write(connfd, buff, sizeof(buff));
```

```
		// if msg contains "Exit" then server exit and chat ended.
		if (strncmp("exit", buff, 4) == 0) {
			printf("Server Exit...\n");
			break;
		}
	}
}

// Driver function
int main()
{
	int sockfd, connfd, len;
	struct sockaddr_in servaddr, cli;

	// socket create and verification
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd == -1) {
		printf("socket creation failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully created..\n");
	bzero(&servaddr, sizeof(servaddr));

	// assign IP, PORT
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
	servaddr.sin_port = htons(PORT);

	// Binding newly created socket to given IP and verification
	if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
		printf("socket bind failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully binded..\n");

	// Now server is ready to listen and verification
	if ((listen(sockfd, 5)) != 0) {
		printf("Listen failed...\n");
		exit(0);
	}
	else
		printf("Server listening..\n");
	len = sizeof(cli);

	// Accept the data packet from client and verification
	connfd = accept(sockfd, (SA*)&cli, &len);
	if (connfd < 0) {
		printf("server accept failed...\n");
		exit(0);
	}
	else
		printf("server accept the client...\n");

	// Function for chatting between client and server
	func(connfd);
```

```
	// After chatting close the socket
	close(sockfd);
}
```

TCP Client:

Code

```
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
	char buff[MAX];
	int n;
	for (;;) {
		bzero(buff, sizeof(buff));
		printf("Enter the string : ");
		n = 0;
		while ((buff[n++] = getchar()) != '\n')
			;
		write(sockfd, buff, sizeof(buff));
		bzero(buff, sizeof(buff));
		read(sockfd, buff, sizeof(buff));
		printf("From Server : %s", buff);
		if ((strncmp(buff, "exit", 4)) == 0) {
			printf("Client Exit...\n");
			break;
		}
	}
}

int main()
{
	int sockfd, connfd;
	struct sockaddr_in servaddr, cli;

	// socket create and verification
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd == -1) {
		printf("socket creation failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully created..\n");
	bzero(&servaddr, sizeof(servaddr));

	// assign IP, PORT
```

```
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	servaddr.sin_port = htons(PORT);

	// connect the client socket to server socket
	if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
		!= 0) {
		printf("connection with the server failed...\n");
		exit(0);
	}
	else
		printf("connected to the server..\n");

	// function for chat
	func(sockfd);

	// close the socket
	close(sockfd);
}
```

Compilation in linux using C language –

Server side:

gcc server.c -o server
./server

Client side:

gcc client.c -o client
./client

Output

Server side:

```
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
From client: hi
     To client : hello
From client: exit
     To client : exit
Server Exit...
```

Client side:

```
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```