Wireshark Project **Data Communication and Networking** **Name: Sahil Anande**

1.

a)

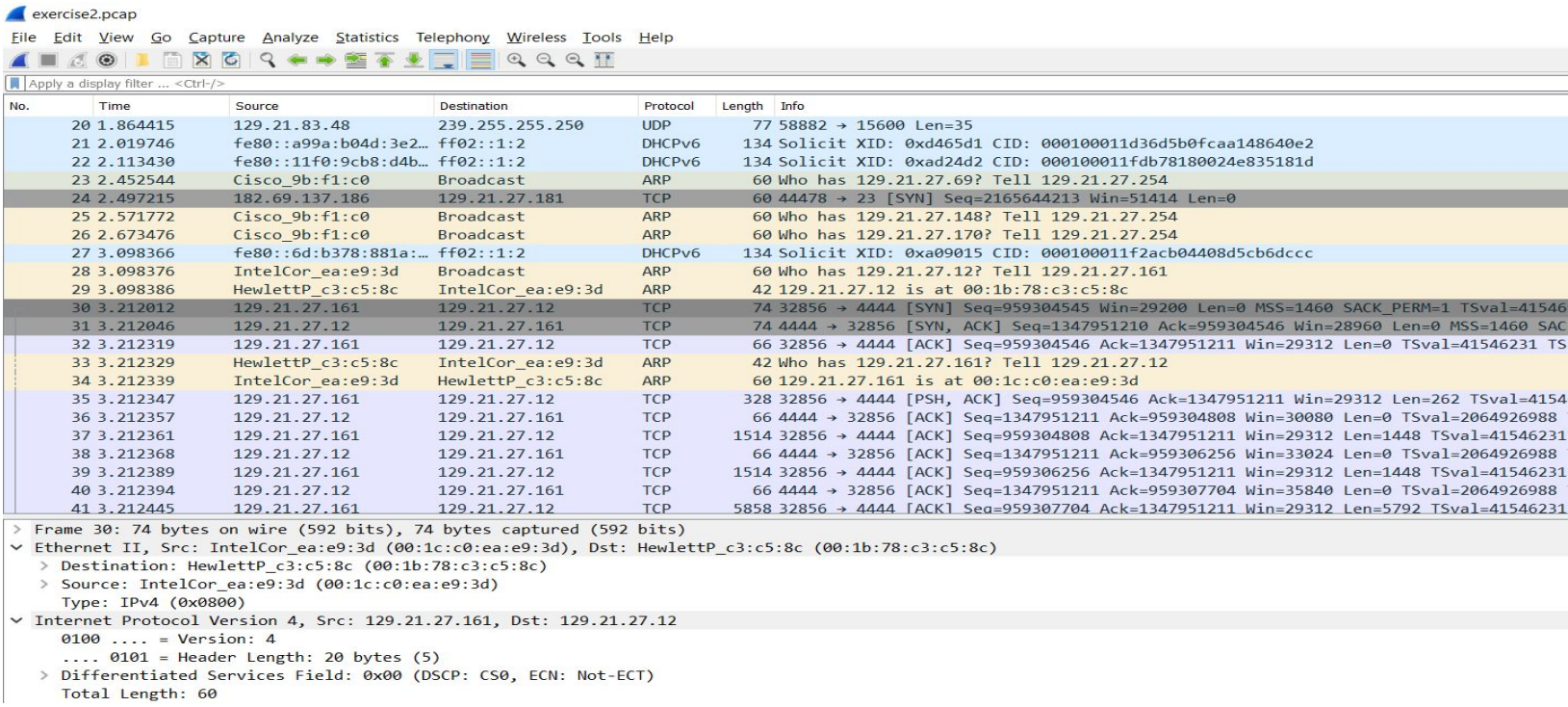

Fig 1: Ethernet II and 74 bytes on the wire.

Ethernet II mentioned in the link layer is a wired protocol used for data transfer. Thus, we can conclude that the network interface from where traffic was captured was wired (an Ethernet interface).

b)

Different packets/protocols	Layers encapsulated
DHCPv6	Physical, Data Link, Network, Transport
STP	Physical, Data Link
ARP	Physical, Data Link
UDP	Physical, Data Link, Network, Transport
TCP	Physical, Data Link, Network, Transport

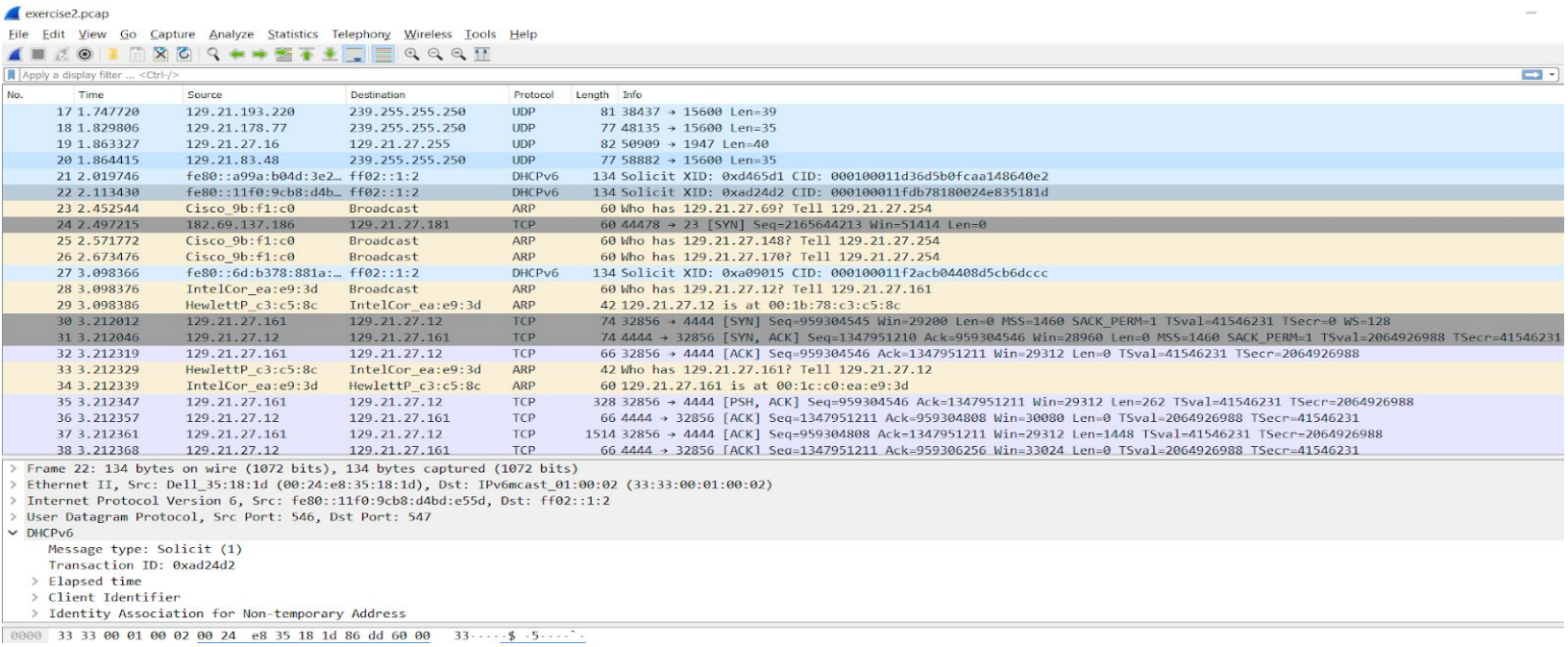

Fig 2: the layers encapsulated for packet 22, which has the DHCPv6 protocol. Check the packet detail window

From the above figure, we can interpret that the DHCPv6 protocol packet has the Physical layer, Data Link layer, Network layer, and Transport layer encapsulated.
Similarly, we can obtain this information for other packets/protocols. All the different protocols available in the used pcap file are presented in a tabular form above.

c) Packet numbers 28 and 29 follow the Address Resolution Protocol (ARP). Packet 28 is an ARP request and packet 29 is an ARP reply. Source and destination addresses are MAC addresses. In ARP exchange the sender knows the IP address of the destination but has no entry of its MAC address in it's MAC table. By sending an ARP request, which is a broadcast message i.e the destination MAC address field is all '1' in the frame, IP address is the destination IP address. When the broadcast frame is received at the node whose IP address is included in the frame, it acknowledges the frame and sends an ARP reply back only to the source with its MAC address included in the frame.
In our pcap file, in packet 28, the source has MAC address 00:1c:c0:ea:e9:3d (IntelCor_ea:e9:3d) and IP address as 129.21.27.161. It sents a broadcast to all the nodes in the LAN with destination MAC address as ff:ff:ff:ff:ff:ff (Broadcast) and destination IP address as 129.21.27.12. When the node with IP address 129.21.27.12 receives the broadcast message it has acknowledged it and sends an ARP reply back to the source with its MAC address included in the frame.

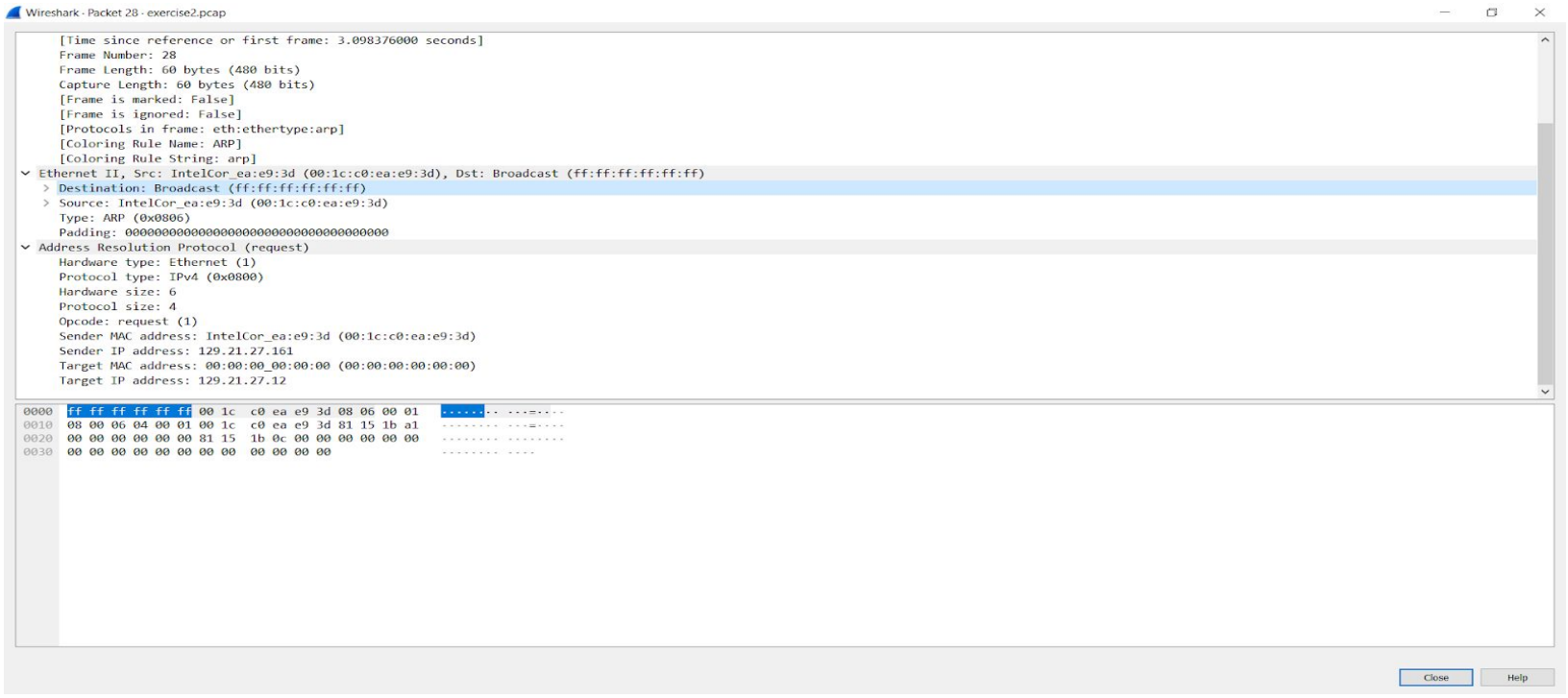

Fig 3: ARP request (Packet 28), Destination address -> MAC address ff:ff:ff:ff:ff:ff

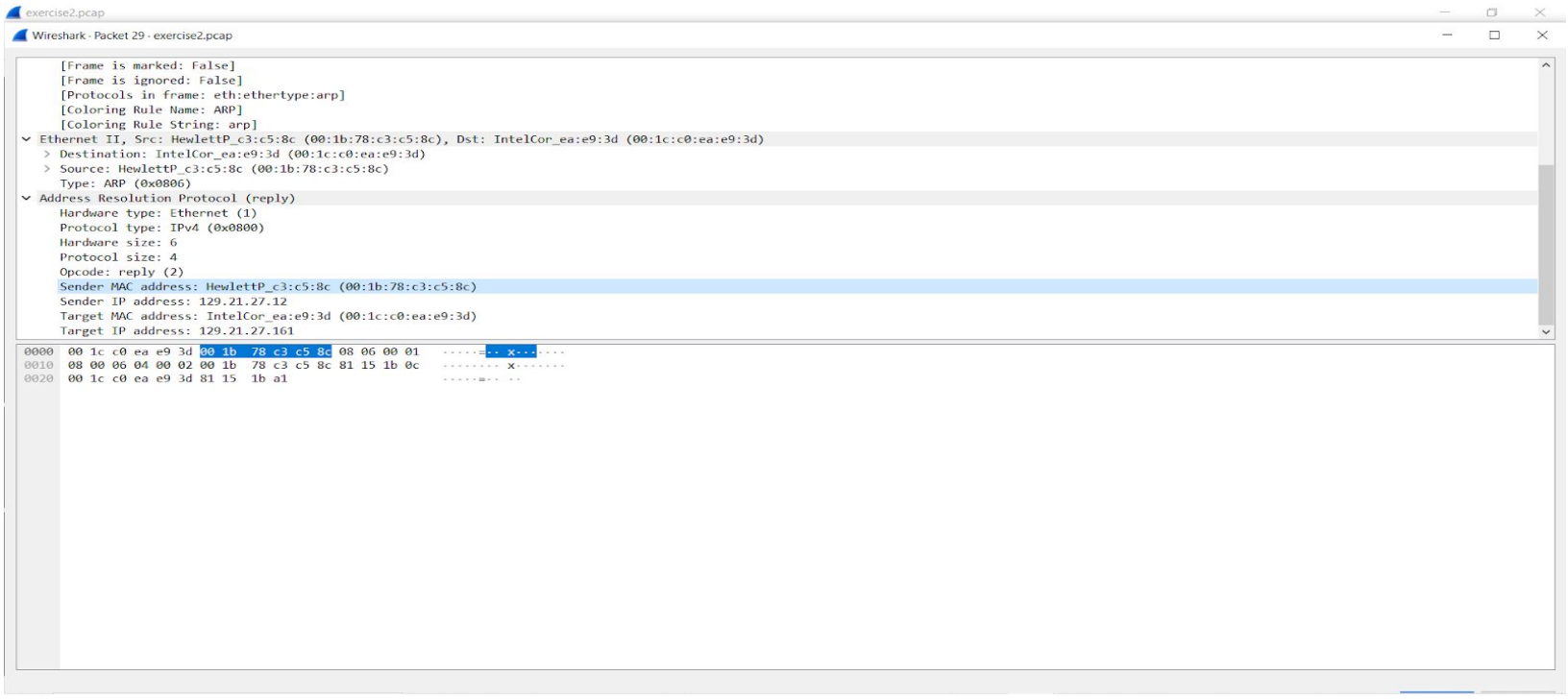

Fig 4: ARP reply (Packet 29), Destination MAC address (HewlettP) obtained.

d) The pcap file used contains a TCP connection for file transfer from source to destination. The two IP addresses involved in this exchange are 129.21.27.12 and 129.21.27.161. Packets 28, 29 and packets 33, 34 are the ARP packets used to obtain the MAC address of one another. TCP exchange starts at packet 30 where the source IP address 129.21.27.161 sends an SYN (Synchronize Sequence Number) to destination IP address 129.21.27.12 for synchronizing the exchange. The destination IP address 129.21.27.12 sends back an SYN, ACK to acknowledge this synchronization and to tell the source that it is ready for TCP exchange in the packet 31. Packet 32 denotes the ACK sent by the source to destination that confirms the reliable connection made between them and tells the destination that the transfer will start now. These three packets denote the 3-way handshake of TCP. From packets 35 to packet 82 there is data transfer from source to destination. Packet 35 includes the first payload “RFC 791”. Packet 83 includes data transfer from destination to source, the data was “I got your file”. From packet 84 to 87 the TCP termination takes place. In packet 84 the destination node sends a packet with FIN and ACK to the source denoting that it wants to terminate the TCP connection. Packet 85 is the ACK sent by the source node to the destination acknowledging the request. Also, in packet 86 source node sends a packet with FIN and ACK flag high to denote that graceful termination of the TCP will be done from both the end. In packet 87, the destination node sends a packet with ACK flag high and the TCP connection is terminated.

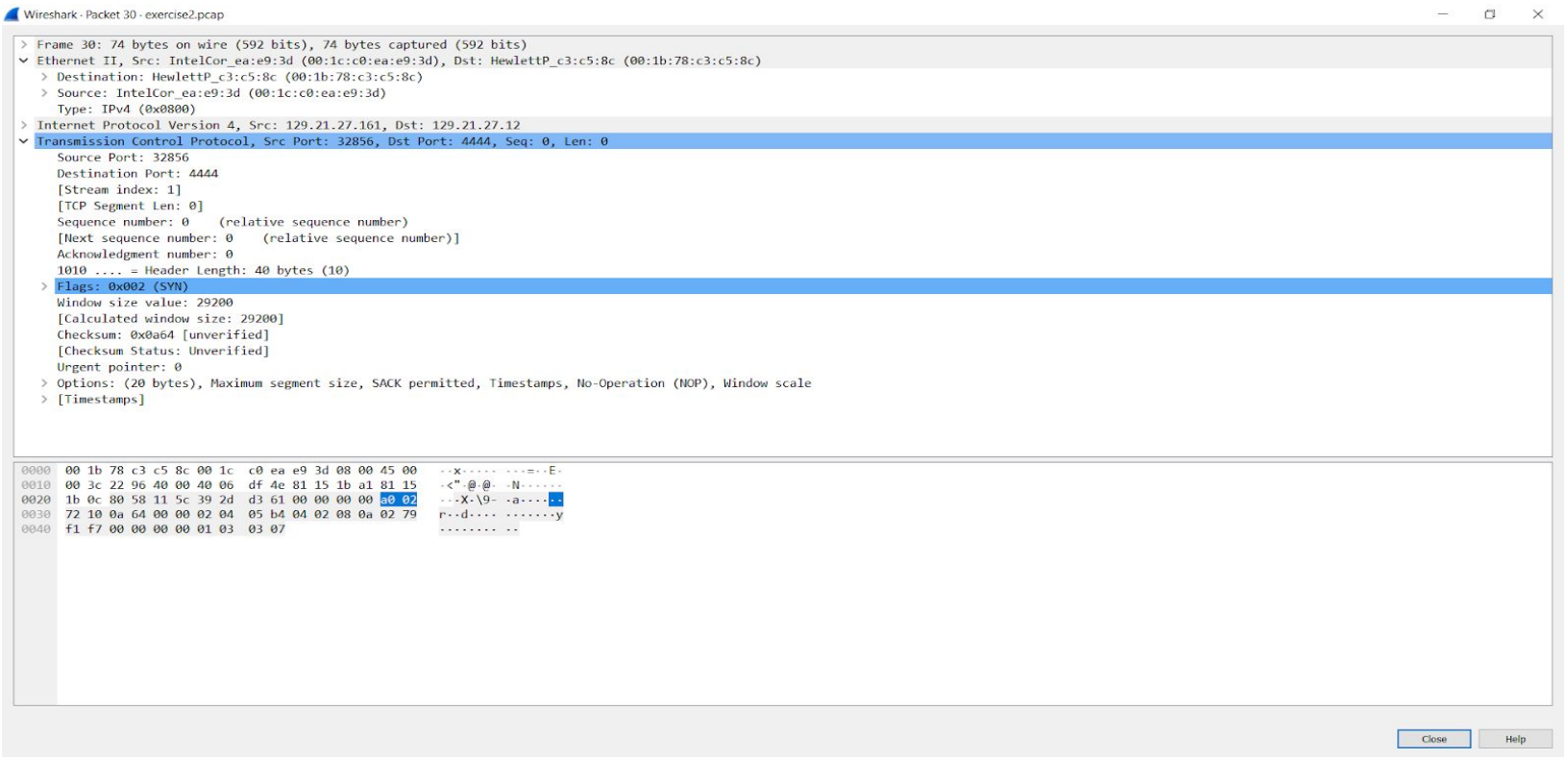

Fig 5: Packet 30, SYN flag for synchronization of sequence number between source and destination

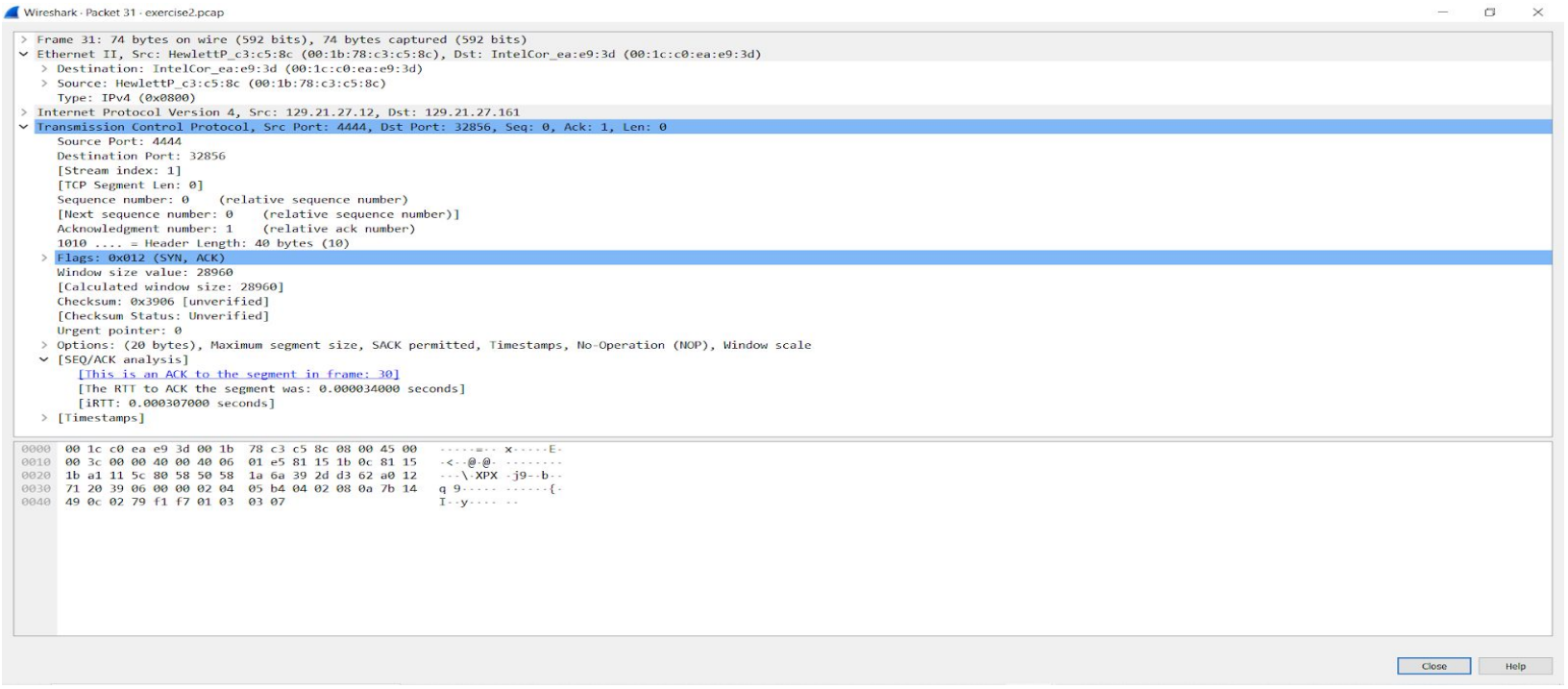

Fig 6: Packet 31, SYN+ACK received from the destination

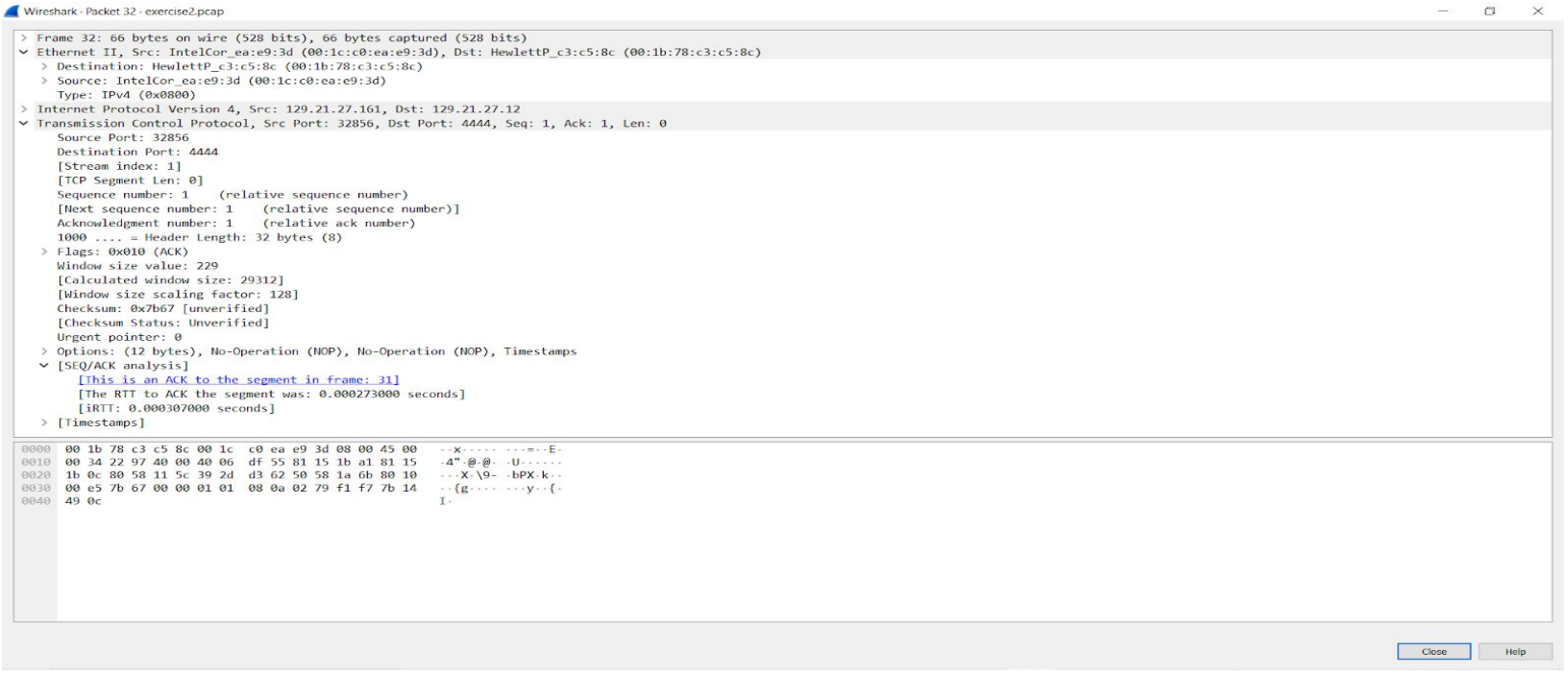

Fig 7: Packet 32, ACK sent by the source to destination to denote the 3-way handshake complete and data transfer will begin.

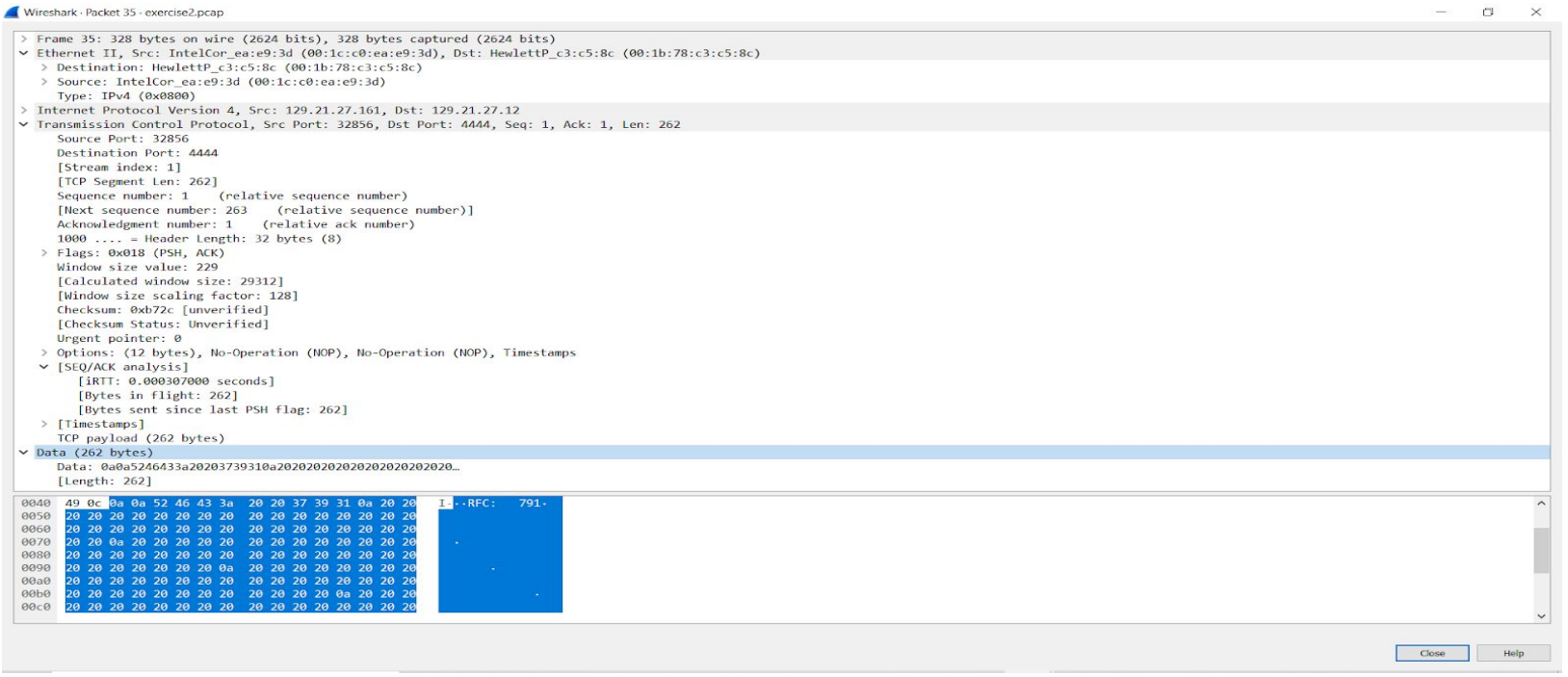

Fig 8: Packet 35, the source sends the first payload to the destination

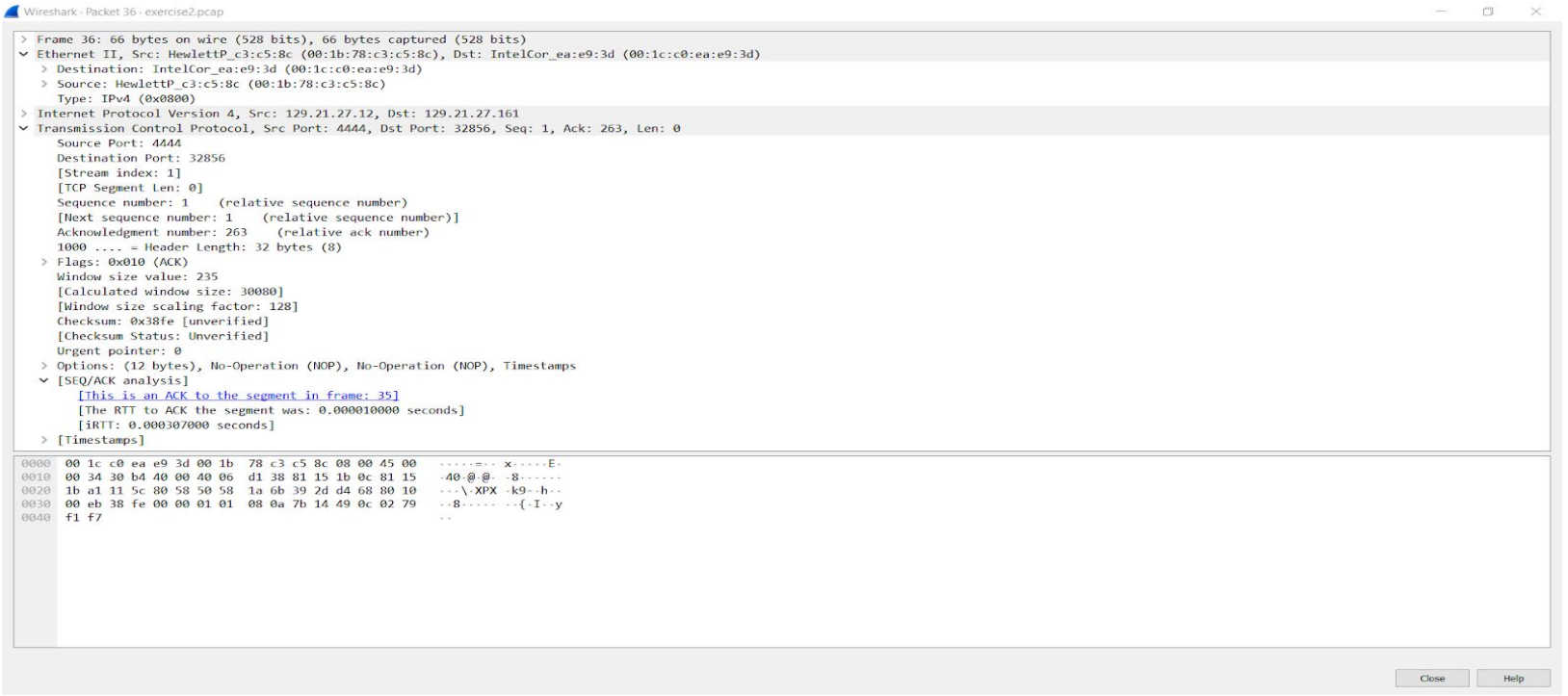

Fig 9: Packet 36. ACK sent back to the source which denotes that packet 35 data received.

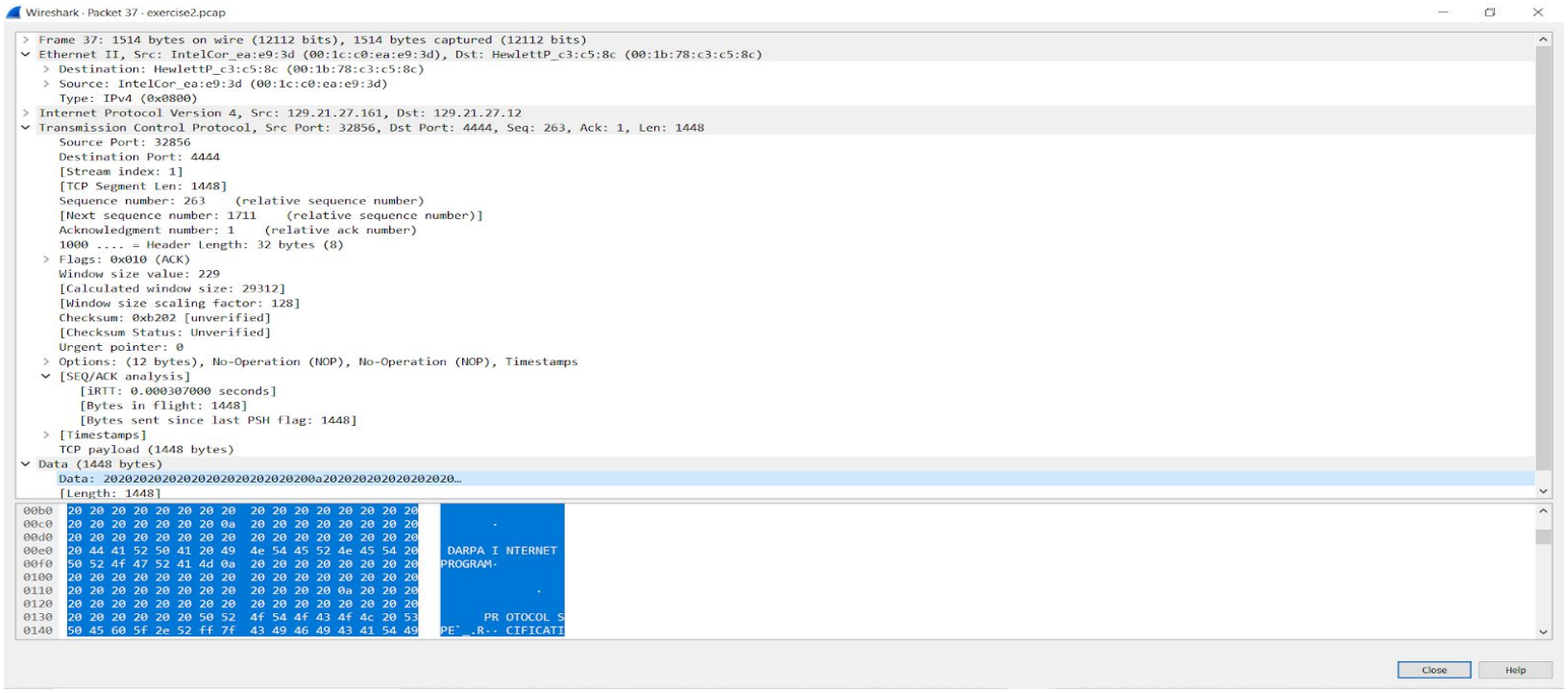

Fig 10: Packet 37, 2nd payload send to the destination

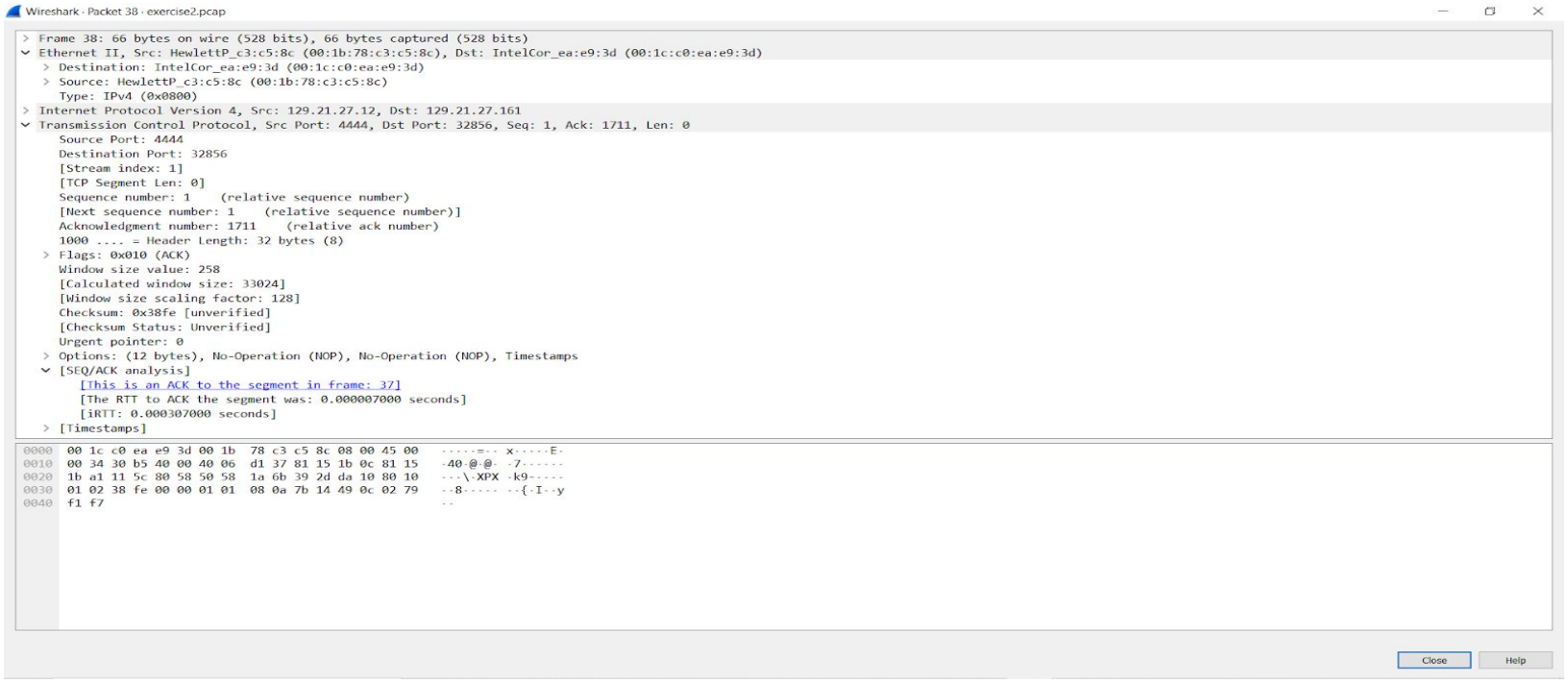

Fig 11: Packet 38, ACK for the packet 37 received correctly

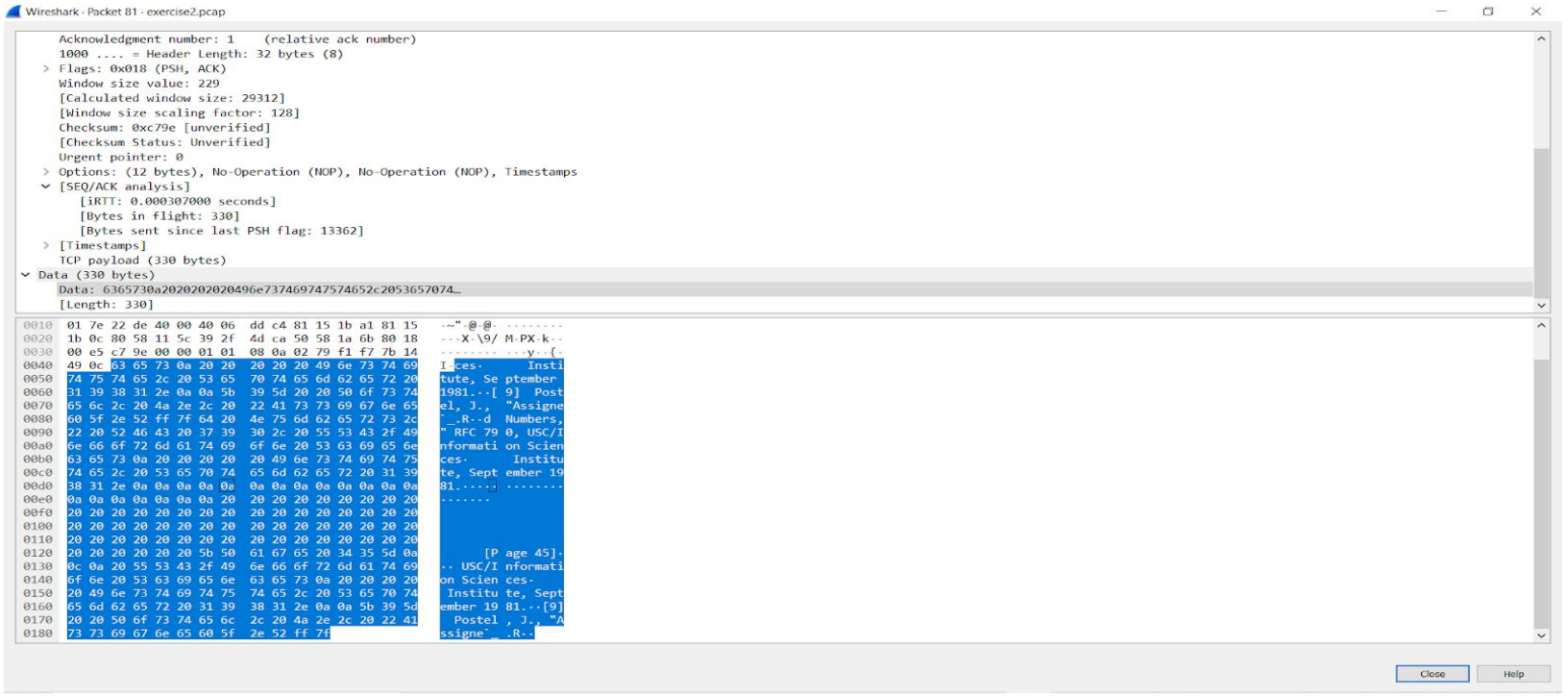

Fig 12: The Last payload sent by the source

Fig13: Data sent by the destination to the source after completion of the file transfer

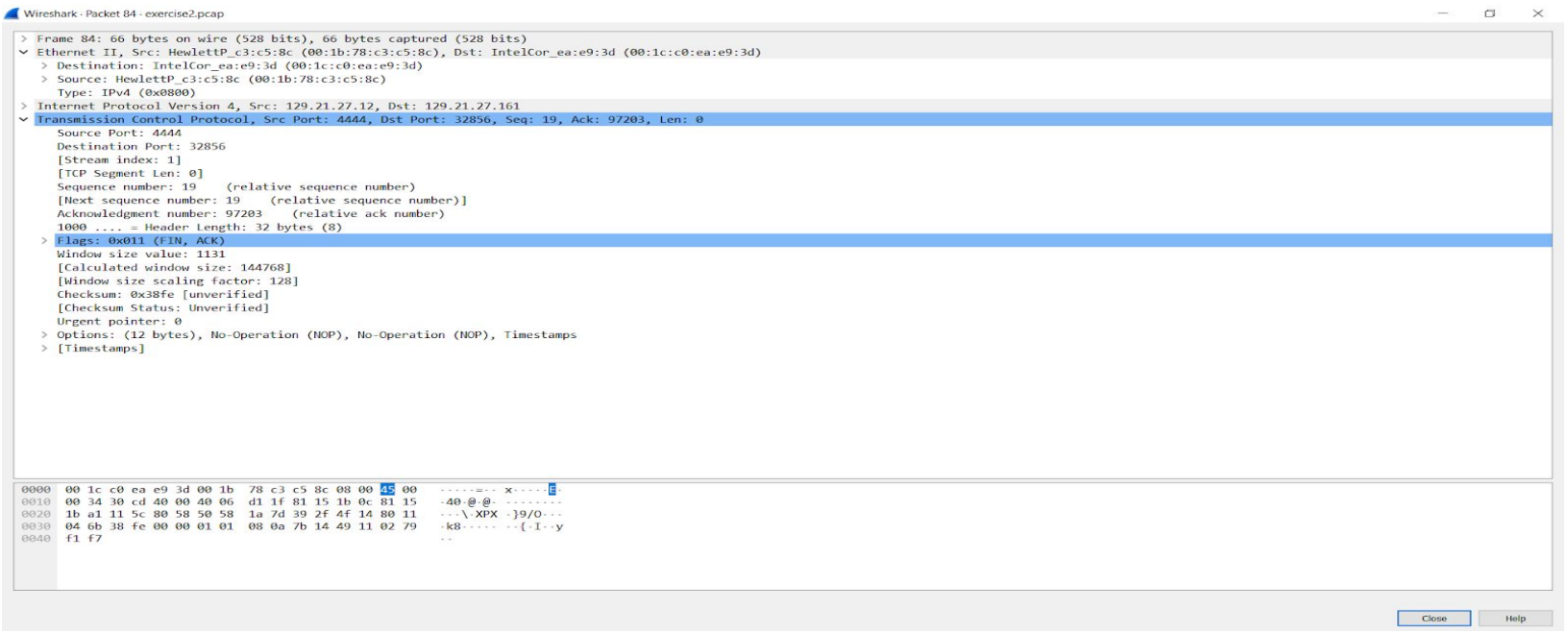

Fig 14: FIN+ACK send by destination to initiate the termination of the TCP connection

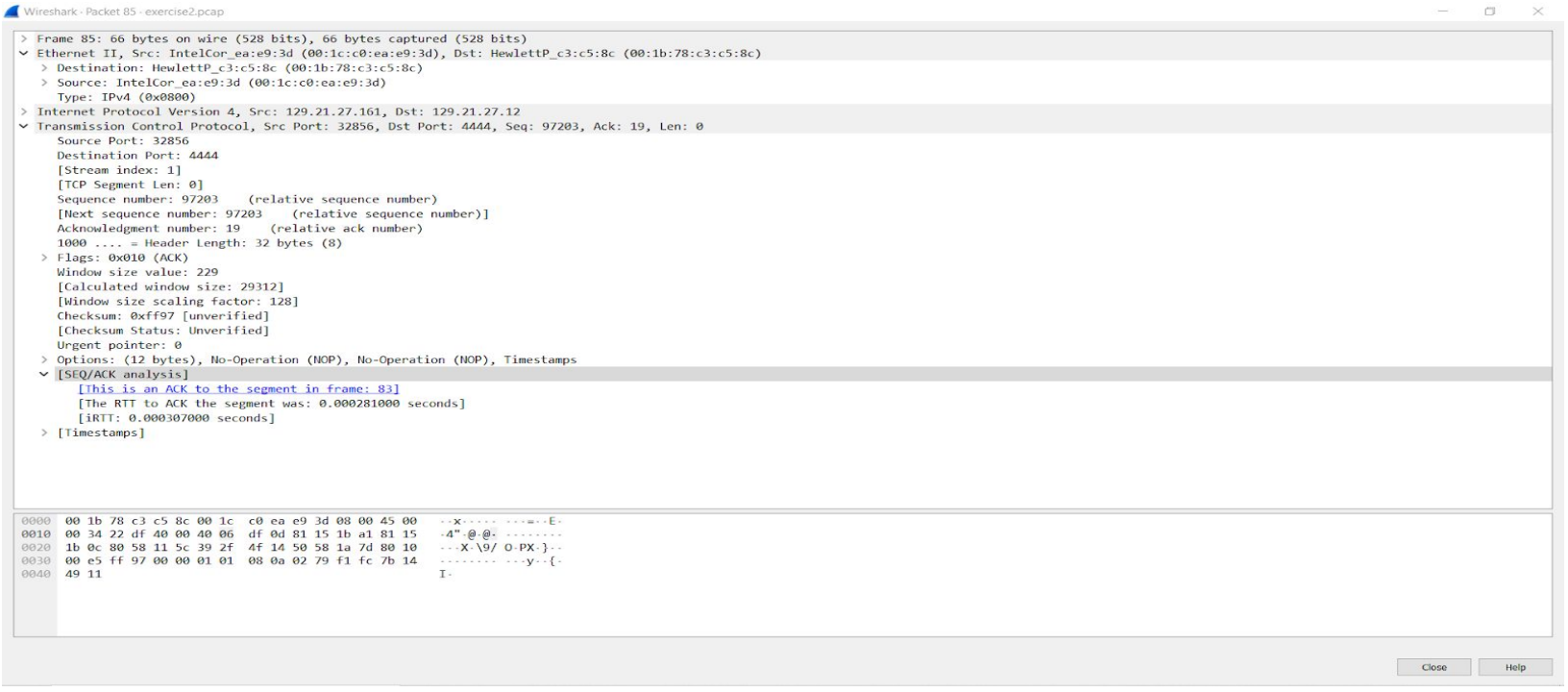

Fig 15: ACK send by the source for the FIN+ACK of destination

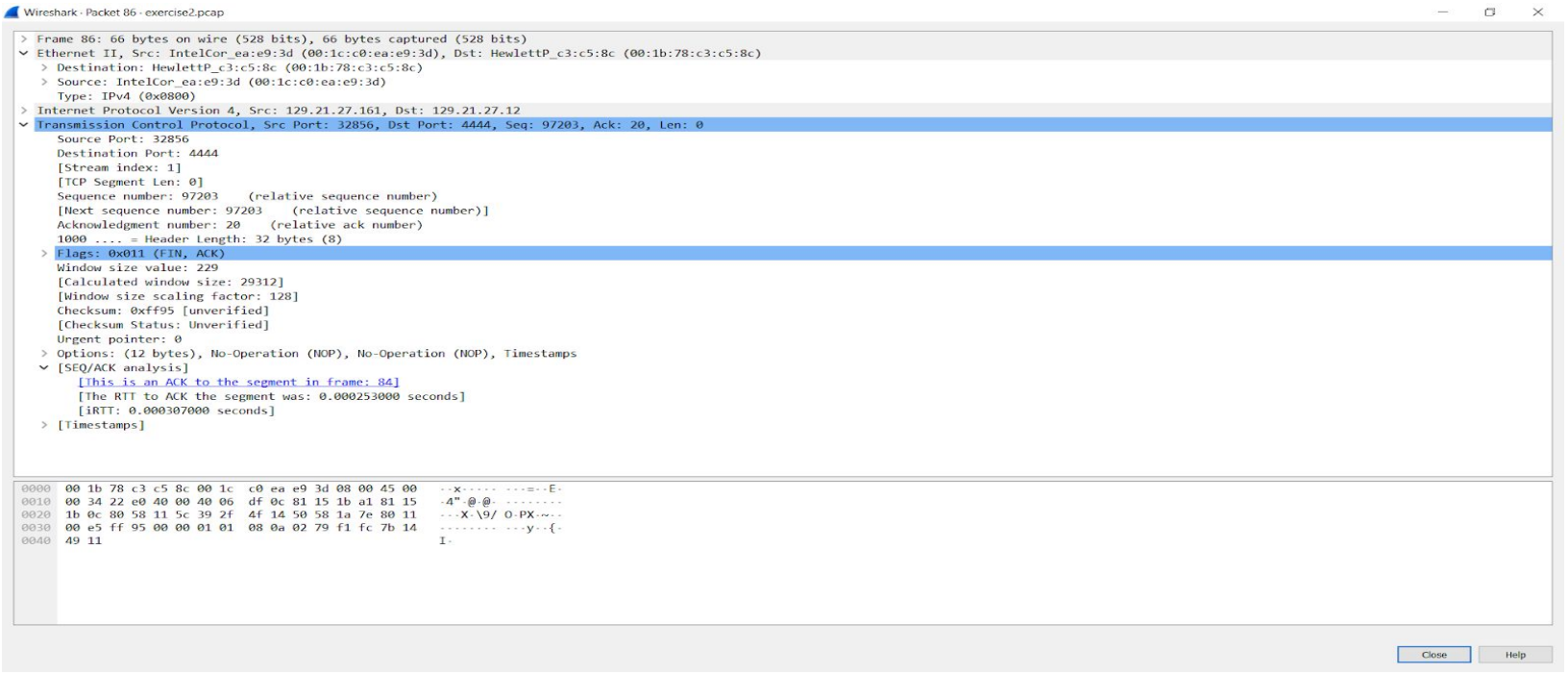

Fig 16: Graceful termination from both the end, FIN+ACK send by the source

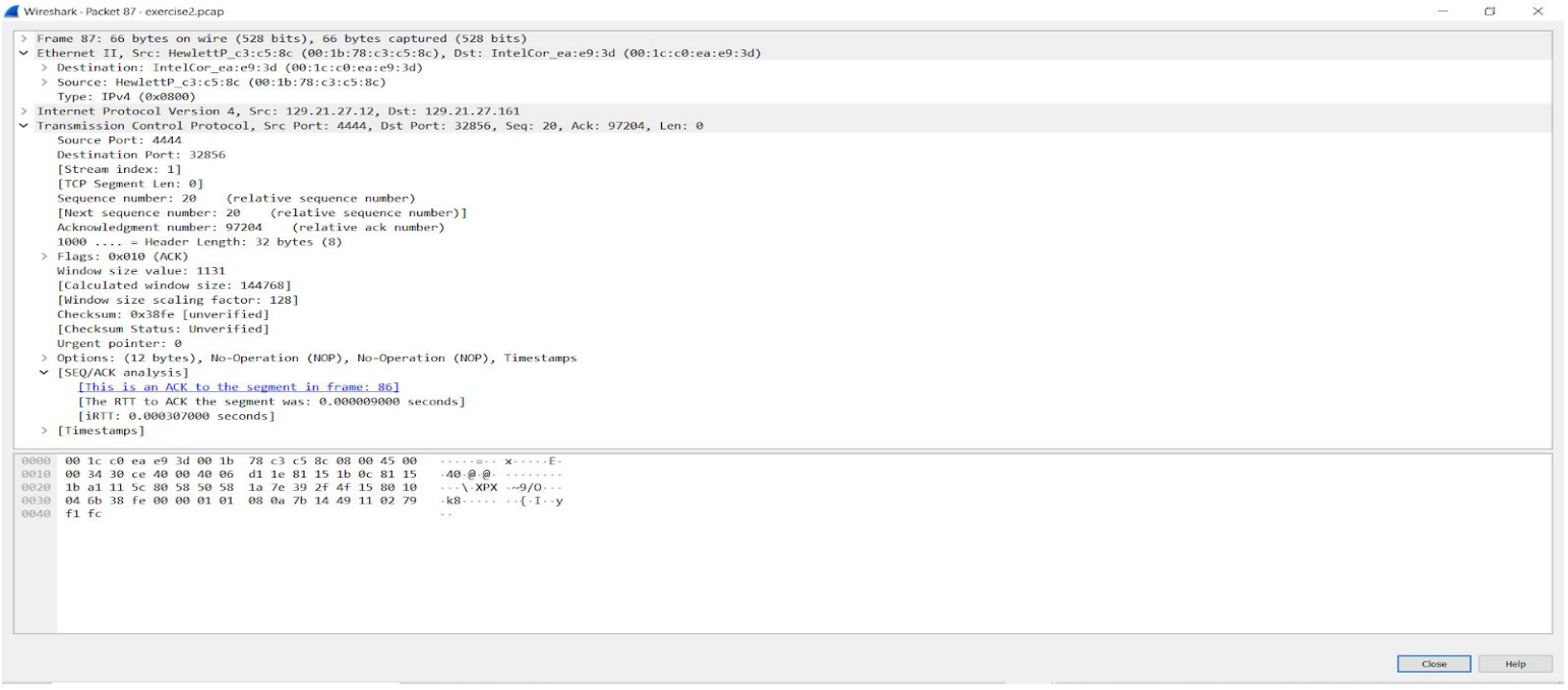

Fig 17: ACK sent by the receiver and TCP connection is terminated.

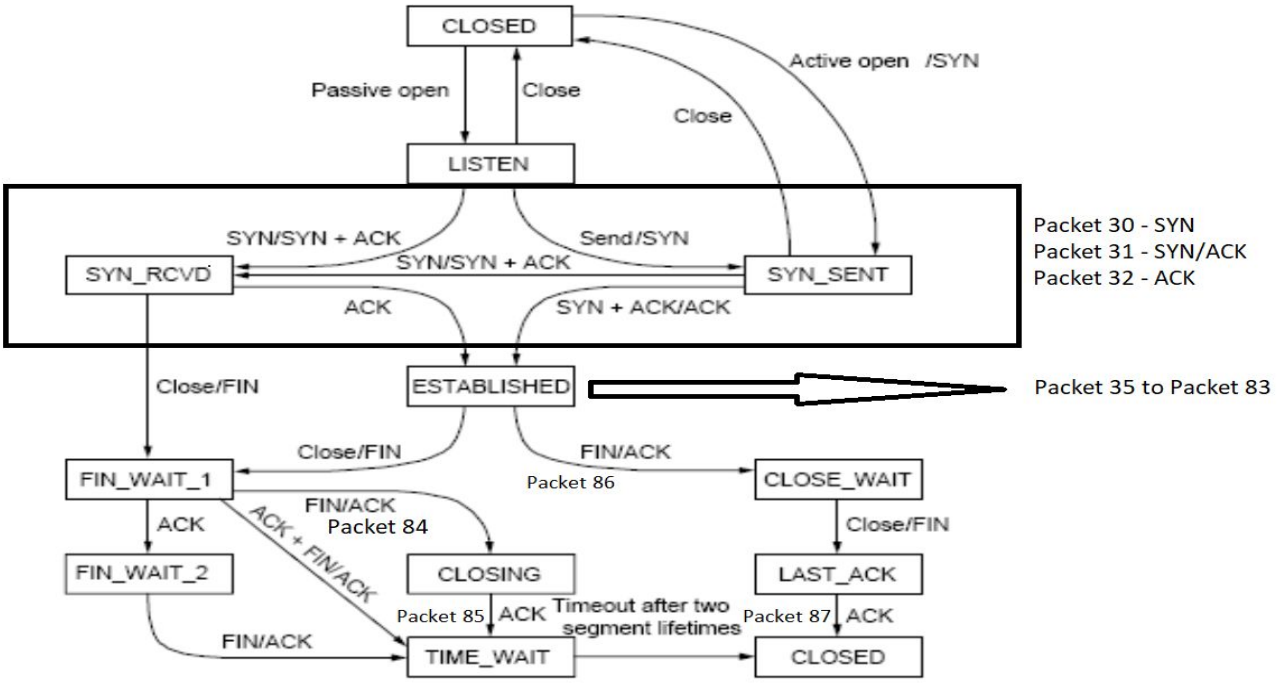

Fig 18: TCP state Diagram

2

b) A structure creates a data type to used group items of possibly different types into a single type. struct sniff_ethernet is used to store the parameters included in the ethernet frame. It contains two unsigned character arrays of six bytes for source and destination MAC addresses, which are 6 bytes in length. It also contains a two-byte unsigned short variable to get the ether type from the header which specifies whether the frame type is IP, ARP, etc.
“ethernet = (struct sniff_ethernet*)(pcktbuf); “ is typecasting a pointer. In the code, pcktbuf is defined as an unsigned char array of 65535 bytes. By using the above command, the array of unsigned char is typecasted to a struct using a pointer. Typecasting changes the variable type for that operation into a different datatype mentioned in the code.

The other examples of typecasting using a pointer in the same function are,
ip = (struct sniff_ip*)(pcktbuf + SIZE_ETHERNET)
tcp = (struct sniff_tcp*)(pcktbuf + SIZE_ETHERNET + *size_ip)
payload = (u_char *)(pcktbuf + SIZE_ETHERNET + *size_ip + *size_tcp)

“int isgetTCPIP(BYTE *pcktbuf, u_int *size_ip, u_int *size_tcp)” function takes input as a pointer to the pcktbuf which points to the start of the packet buffer, a pointer to IP size and TCP size. ethernet = (struct sniff_ethernet*)(pcktbuf), as discussed before changes the type of variable pcktbuf to a struct of sniff_ethernet so that the Ethernet frame can be sniffed and information of

MAC addresses and ether type can be obtained from the packet buffer (pcktbuf). After this, the ethernet header is added to the packet buffer and forwarded to the IP layer i.e data link layer. Using ip = (struct sniff_ip*)(pcktbuf + SIZE_ETHERNET); the frame is assigned a type struct sniff_ip so that using the object “ip” of sniff_ip we can access the parameters defined inside that function for our encapsulation process. The size of IP is compared and passed ahead only if it satisfies the requirement of header length. With the help of tcp = (struct sniff_tcp*)(pcktbuf + SIZE_ETHERNET + *size_ip); the data link layer packet is now typecasted using the sniff_tcp structure pointer. The TCP header size is obtained and verified if it is of the correct length. By adding all the headers, namely Ethernet, IP, and TCP we get the location of where data/payload starts i.e. payload = (u_char *)(pcktbuf + SIZE_ETHERNET + *size_ip + *size_tcp);

This function is separating headers from Ethernet protocol, IP protocol and TCP protocol which resides in data link layer, network layer and transport layer respectively.

c)

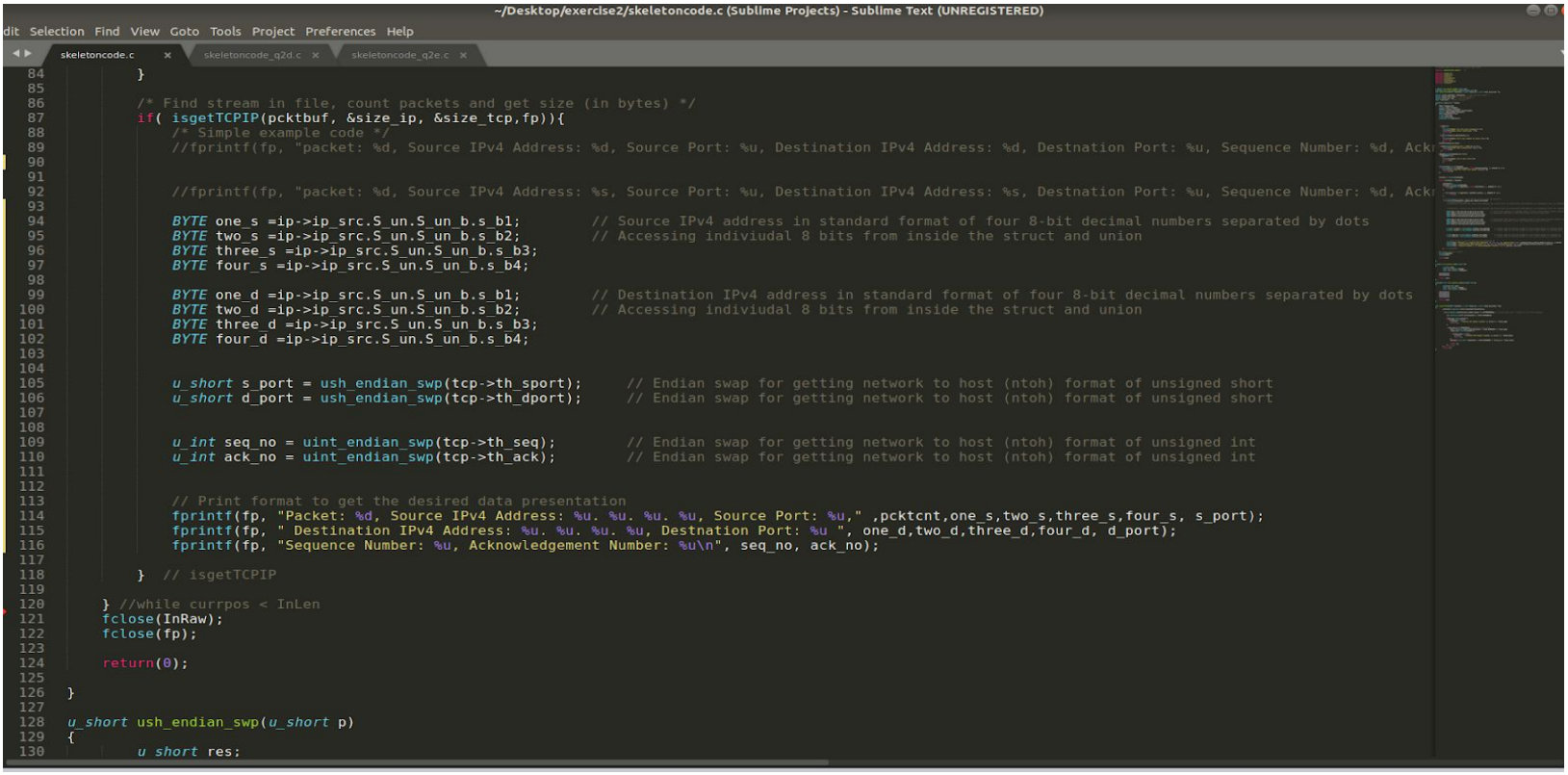

Fig 19: Code to obtain Source IPv4 address and port number, Destination IPv4 address and port number, Sequence number, Acknowledgement number

The encap.h file contains a struct in_addr_new which further contains a union to store the values of IPv4 address in parts as a single-byte i.e 8 bits. I have referenced to those bytes through various structures and stored in different variables which would help in writing them with

a specific format in the file. Similar operations are carried out for destination IPv4 addresses as well.

To overcome the “endian” problem, source and destination port are endian swapped using the ush_endian_swp function for unsigned short values.

Also, the sequence number and acknowledgment number are endian swapped to get the proper results using uint_endian_swp function for unsigned int values.

Sequence numbers and acknowledgment numbers are not relative to each other unlike the output shown in the Wireshark portal in question 1d.

```
Packet: 24, Source IPv4 Address: 182. 69. 137. 186, Source Port: 44478, Destination IPv4 Address: 182. 69. 137. 186, Destnation Port: 23 Sequence Number: 2165644213, Acknowledgement Number: 0
Packet: 30, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959304545, Acknowledgement Number: 0
Packet: 31, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951210, Acknowledgement Number: 959304546
Packet: 32, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959304546, Acknowledgement Number: 1347951211
Packet: 35, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959304546, Acknowledgement Number: 1347951211
Packet: 36, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959304808
Packet: 37, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959304808, Acknowledgement Number: 1347951211
Packet: 38, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959306256
Packet: 39, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959306256, Acknowledgement Number: 1347951211
Packet: 40, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959307704
Packet: 41, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959307704, Acknowledgement Number: 1347951211
Packet: 42, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959313496
Packet: 43, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959313496, Acknowledgement Number: 1347951211
Packet: 44, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959319288
Packet: 45, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959319288, Acknowledgement Number: 1347951211
Packet: 46, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959321052
Packet: 47, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959321052, Acknowledgement Number: 1347951211
Packet: 48, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959326844
Packet: 49, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959326844, Acknowledgement Number: 1347951211
Packet: 50, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959331188
Packet: 51, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959331188, Acknowledgement Number: 1347951211
Packet: 52, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959335532
Packet: 53, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959335532, Acknowledgement Number: 1347951211
Packet: 54, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959340440
Packet: 55, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959340440, Acknowledgement Number: 1347951211
Packet: 56, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959343336
Packet: 57, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959343336, Acknowledgement Number: 1347951211
Packet: 58, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959347680
Packet: 59, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959347680, Acknowledgement Number: 1347951211
Packet: 60, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959350576
Packet: 61, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959350576, Acknowledgement Number: 1347951211
Packet: 62, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959354920
Packet: 63, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959354920, Acknowledgement Number: 1347951211
Packet: 64, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959362160
Packet: 65, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959362160, Acknowledgement Number: 1347951211
Packet: 66, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959367952
Packet: 67, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959367952, Acknowledgement Number: 1347951211
Packet: 68, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959373744
Packet: 69, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959373744, Acknowledgement Number: 1347951211
Packet: 70, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959379536
Packet: 71, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959379536, Acknowledgement Number: 1347951211
Packet: 72, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959385328
Packet: 73, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959385328, Acknowledgement Number: 1347951211
Packet: 74, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959388386
Packet: 75, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959388386, Acknowledgement Number: 1347951211
Packet: 76, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959392730
Packet: 77, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959392730, Acknowledgement Number: 1347951211
Packet: 78, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959397074
Packet: 79, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959397074, Acknowledgement Number: 1347951211
Packet: 80, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959401418
Packet: 81, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959401418, Acknowledgement Number: 1347951211
Packet: 82, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959401748
Packet: 83, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951211, Acknowledgement Number: 959401748
Packet: 84, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951229, Acknowledgement Number: 959401748
Packet: 85, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959401748, Acknowledgement Number: 1347951229
Packet: 86, Source IPv4 Address: 129. 21. 27. 161, Source Port: 32856, Destination IPv4 Address: 129. 21. 27. 161, Destnation Port: 4444 Sequence Number: 959401748, Acknowledgement Number: 1347951230
Packet: 87, Source IPv4 Address: 129. 21. 27. 12, Source Port: 4444, Destination IPv4 Address: 129. 21. 27. 12, Destnation Port: 32856 Sequence Number: 1347951230, Acknowledgement Number: 959401749
```

Fig 20: Information for each TCP packet displayed

d)

~/Desktop/exercise2/skeletoncode_q2d.c (Sublime Projects) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

skeletoncode.c | skeletoncode_q2d.c | skeletoncode_q2e.c

```
#include <sys/types.h>
#include "encap.h"
#include <arpa/inet.h>
#include <string.h>


u_short ush_endian_swp(u_short p);
unsigned int uint_endian_swp(unsigned int p);
int isgetTCPIP(BYTE *pcktbuf, u_int *size_ip, u_int *size_tcp,FILE *);

struct sniff_ethernet *ethernet;  /* The ethernet header */
struct sniff_tcp *tcp; /* The TCP header */
struct sniff_ip *ip;  /* The IP header */
char *payload; /* Packet payload */

main(int argc,char **argv)
{
    FILE *InRaw,*fp;
    struct stat filedat;
    off64_t InLen, currpos;
    struct pcap_file_header pcapfilehdr;
    struct pcap_pkthdr pckthdr;
    BYTE pcktbuf[65535];
    u_int size_ip;
    u_int size_tcp;
    unsigned int pcktcnt=0;
    int totalsize = 0;

    // Defining Source IPv4 Address in individual bytes
    int check_ip1 = 129;
    int check_ip2 = 21;
    int check_ip3 = 27;
    int check_ip4 = 161;


    if(argc<2)
        {
        printf("ERROR: Too few input arguments\n");
        printf("Usage: encap input.pcap \n");
        return 0;
        }
    if(stat(*(argv+1),&filedat)==-1)
        {
        printf("ERROR: Can't get length of input file\n");
        return 0;
        }
    InLen=filedat.st_size;
```

Fig 21: Source IPv4 defined as four int variables to later compare with the result obtained from code

~/Desktop/exercise2/skeletoncode_q2d.c (Sublime Projects) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

skeletoncode.c | skeletoncode_q2d.c | skeletoncode_q2e.c

```
        if (fread((char *) &pcktbuf, pckthdr.caplen, 1, InRaw) != 1) {
                break;
        }

        /* Find stream in file, count packets and get size (in bytes) */
        if( isgetTCPIP(pcktbuf, &size_ip, &size_tcp,fp)){
            /* Simple example code */
            u_short ip_frame_length = ush_endian_swp(ip->ip_len);

            u_short tcp_packet_length = ip_frame_length - size_ip;

            u_short i_packet_size = tcp_packet_length - size_tcp;


            totalsize += i_packet_size;

            unsigned int seq_no = uint_endian_swp(tcp->th_seq);
            unsigned int ack_no = uint_endian_swp(tcp->th_ack);

            BYTE one_s =ip->ip_src.S_un.S_un_b.s_b1;        // Source IPv4 address in standard format of four 8-bit decimal numbers separated by dots
            BYTE two_s =ip->ip_src.S_un.S_un_b.s_b2;        // Accessing indiviudal 8 bits from inside the struct and union
            BYTE three_s =ip->ip_src.S_un.S_un_b.s_b3;
            BYTE four_s =ip->ip_src.S_un.S_un_b.s_b4;


            // Check if payload is actually containing data or not
            if (i_packet_size != 0)
            {
                // Last data sent by Receiver is not the fragment of file transmission thus discarded. Only transmission from Source is considered
                if (check_ip1==one_s && check_ip2==two_s && check_ip3==three_s && check_ip4==four_s)
                {
                    fprintf(fp, "Packet no.: %d, Size of Payload is  %u bytes, Sequence Number: %u\n",pcktcnt, i_packet_size, seq_no);
                }

            }


        }  // isgetTCPIP

    } //while currpos < InLen

    fprintf(fp, "Total file size is %d bytes\n",totalsize);
    fclose(InRaw);
    fclose(fp);

    return(0);
}
```

Fig 22: Operations carried out to find each packet data size

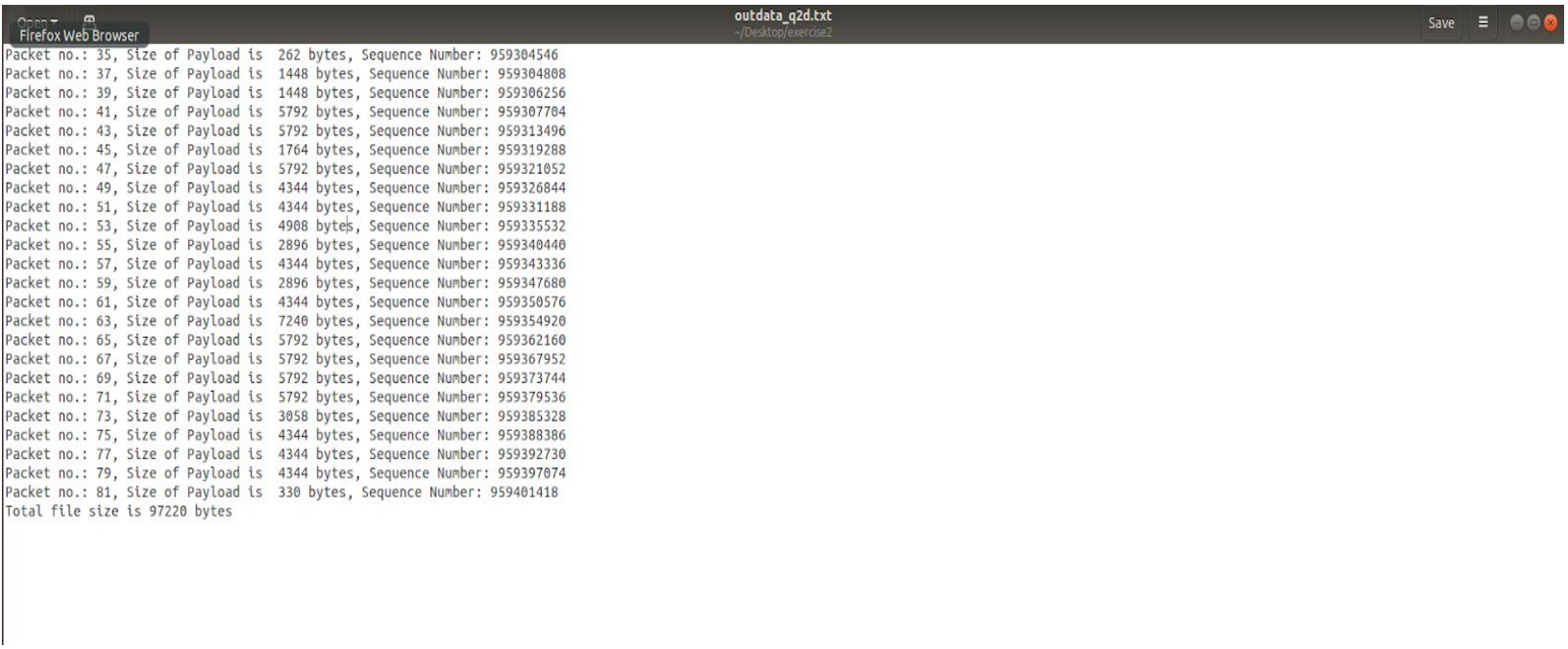

Fig 23: Output for question 2d

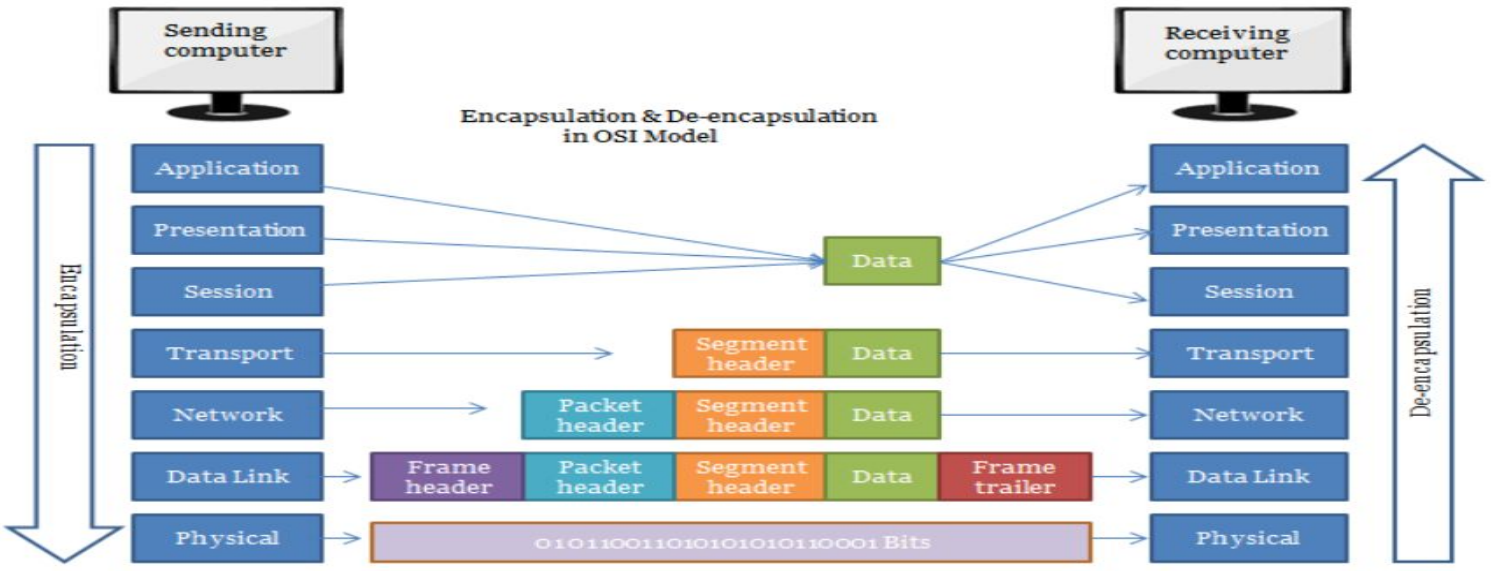

Fig 24: Encapsulation & De-encapsulation in OSI model

We know that different layers add their header to the message coming from the upper layers. For accessing the data, these header files must be removed. In Fig 22, the IP header is

removed from the IP segment. The resultant message still contains the TCP header which is removed to obtain the payload (Data) and its size.
We need to get only those packets that contain the fragment of file data transfer. Packet 83 sends an acknowledgment data from the receiver to the source which is not the data of the file transferred. Hence this packet is discarded here with the use of source IPv4 address comparison.

For getting the un-relative sequence and acknowledgment number, Edit -> Preference -> Protocols -> TCP -> Turn off the relative sequence number.

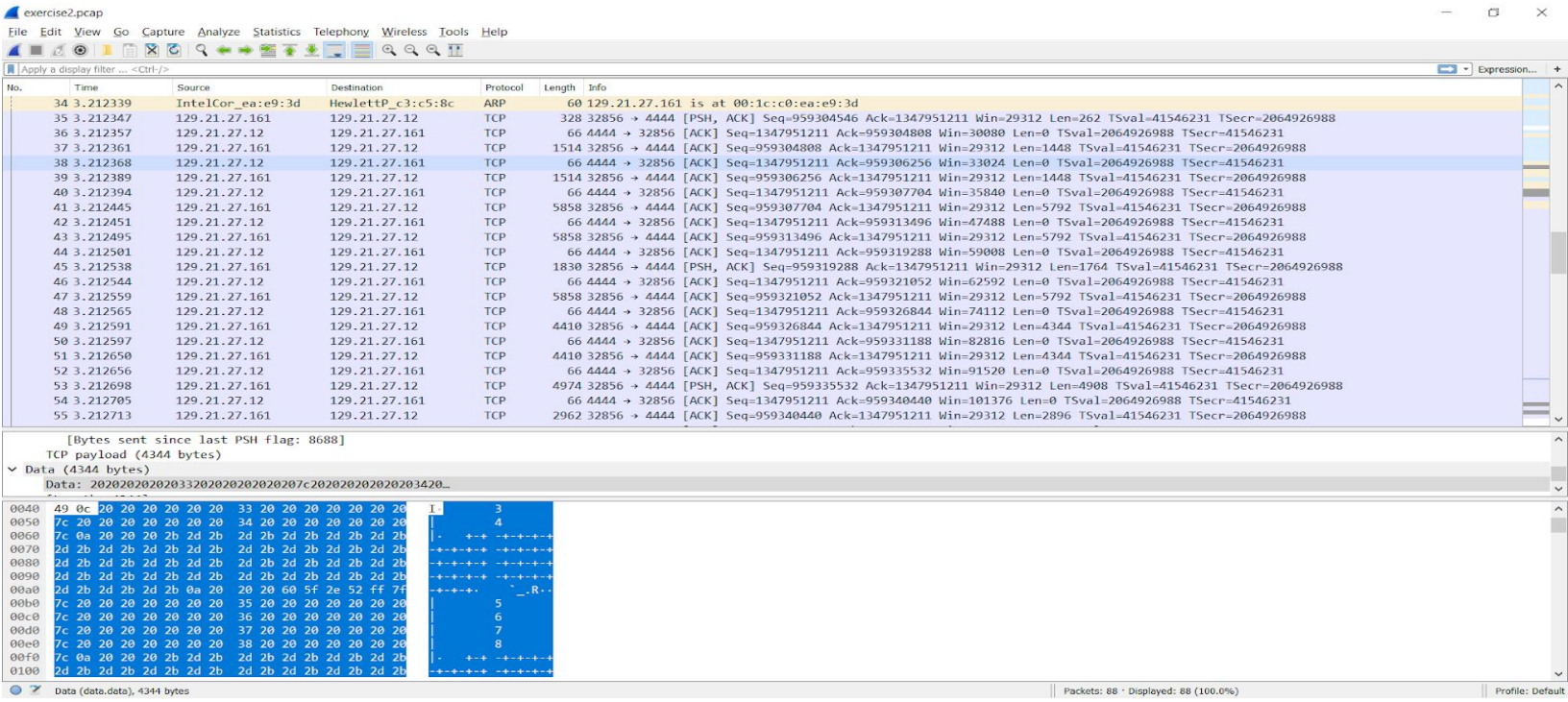

Fig 25: Updating the sequence and acknowledgment number

We can also check if our obtained packet sizes are correct by using the sequence numbers. We know that the sequence number helps the TCP on both sides keep track of how much data has been transferred and to put the data back into the correct order if it is received in the wrong order, and to request data when it has been lost in transit. It is the first byte indexed in that packet.

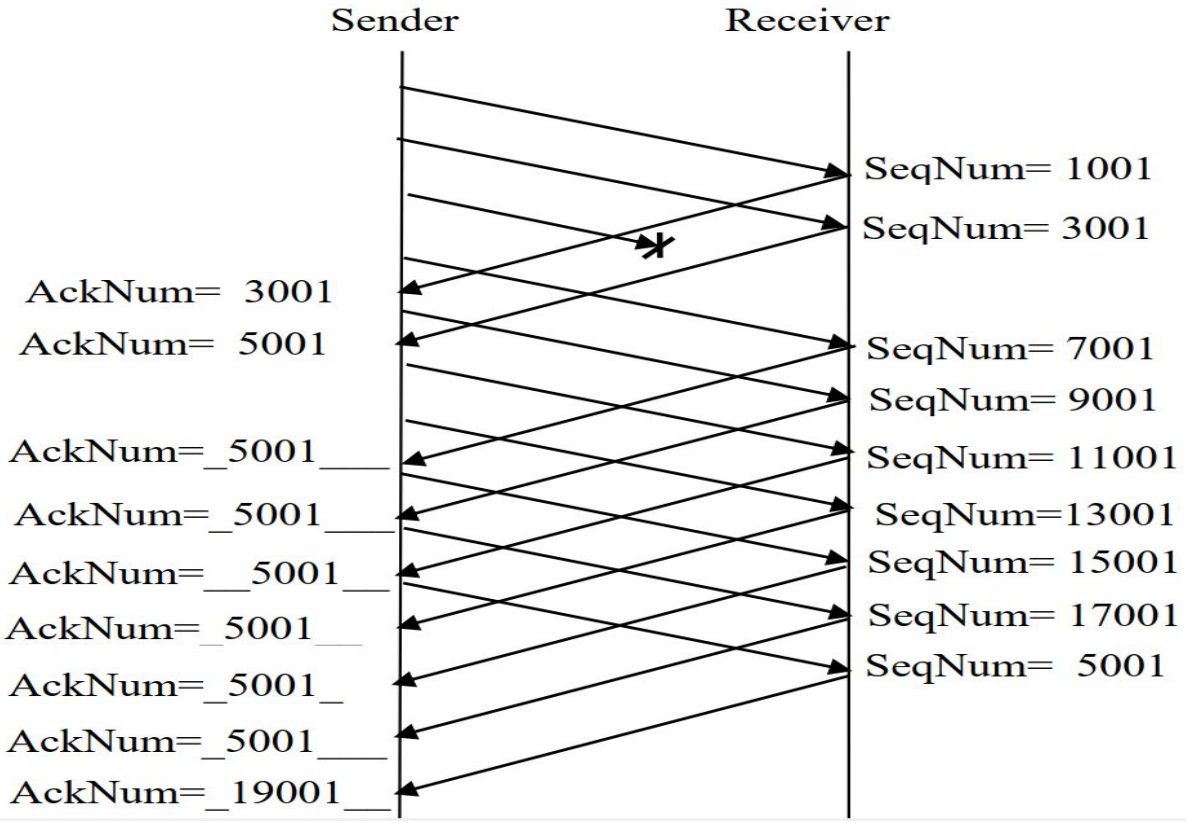

Fig 26: The data transfer in the first packets would be 3001-1001 = 2000 bytes.

Therefore consider packets 37 and 39 which are transmitted by source and include data in it.
Sequence number_P37 = 959304808
Sequence number_P39 = 959306256
Data transferred in packet 37 = 959306256 - 959304808 = 1448 bytes

So, we can confirm that the packet data size obtained from the code is correct. Similarly, we can also check the Wireshark portal to confirm this result.

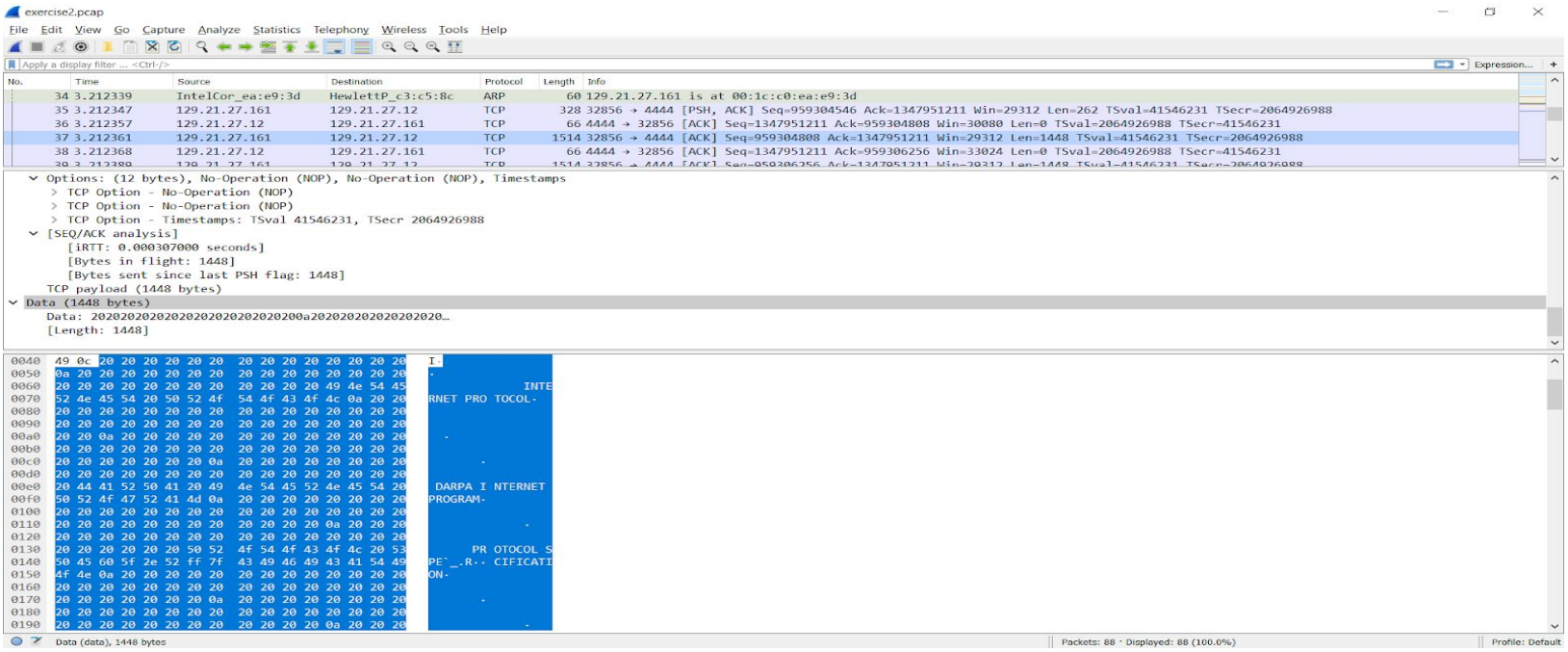

Fig 27: Packet 37 has a payload of 1448 bytes

The total file size is 97.22 kilobytes

e)

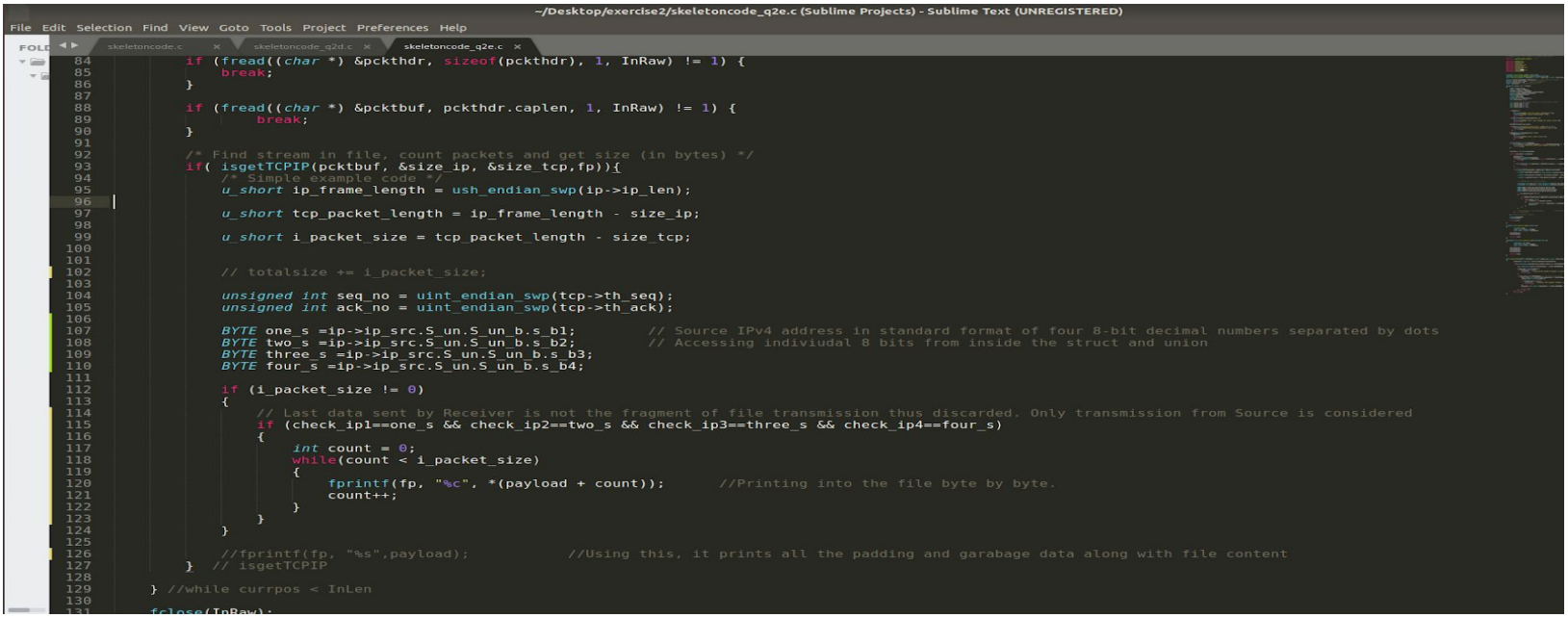

Fig 28: Writing the file data used for transfer in the pcap file

Packet 83 sends an acknowledgment data from the receiver to the source which is not the data of the file transferred. Hence this packet is discarded here with the use of source IPv4 address comparison. I have used byte to byte data write into the file using the count variable.

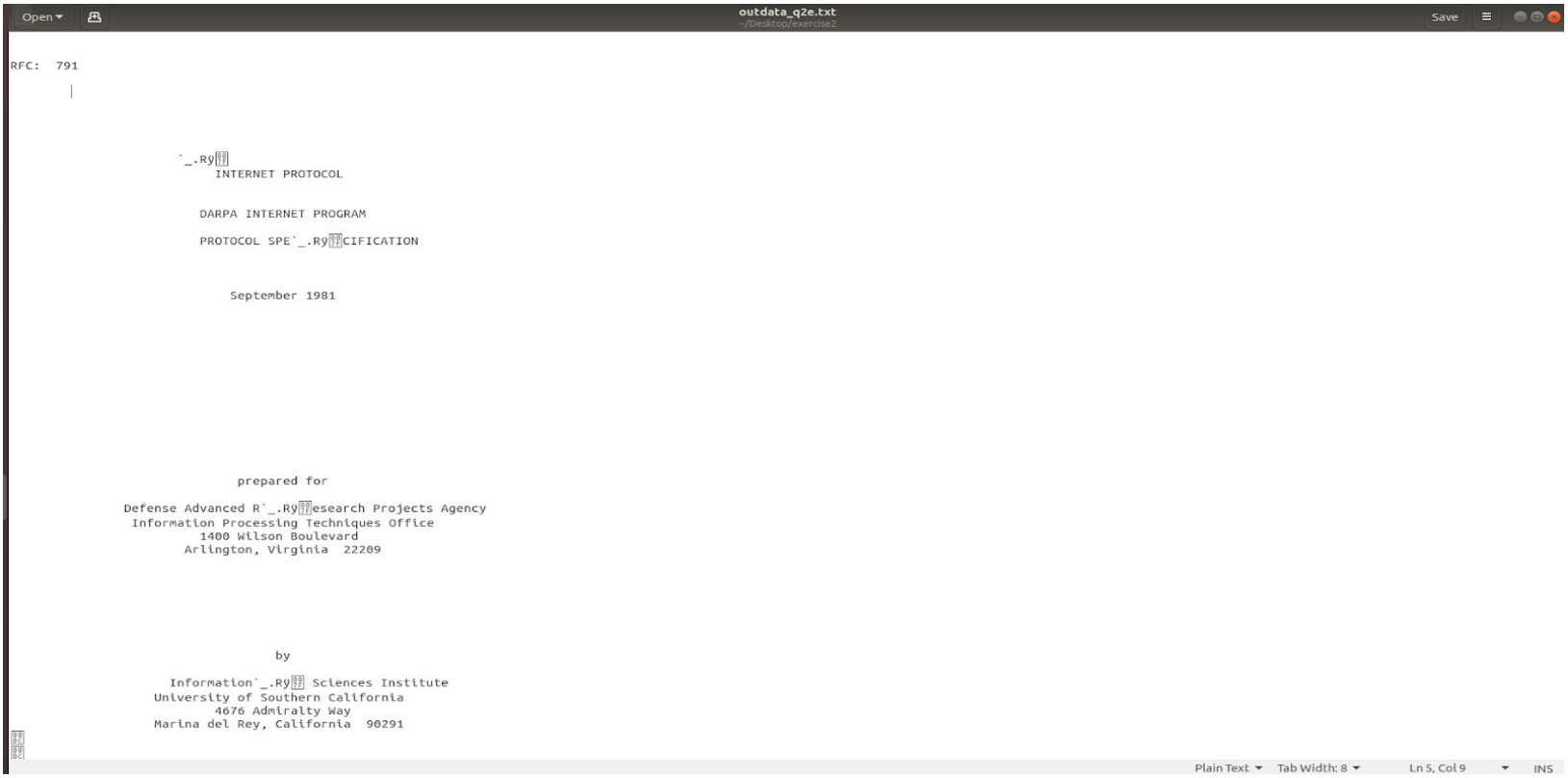

Fig 29: Text file used in the TCP packet transfer

The subject of the file is RFC 791, Internet Protocol, DARPA Internet Protocol Specifications.

References:

https://www.wireshark.org/docs/wsug_html_chunked/ChCustPreferencesSection.html
https://www.computernetworkingnotes.com/ccna-study-guide/data-encapsulation-and-de-encapsulation-explained.html