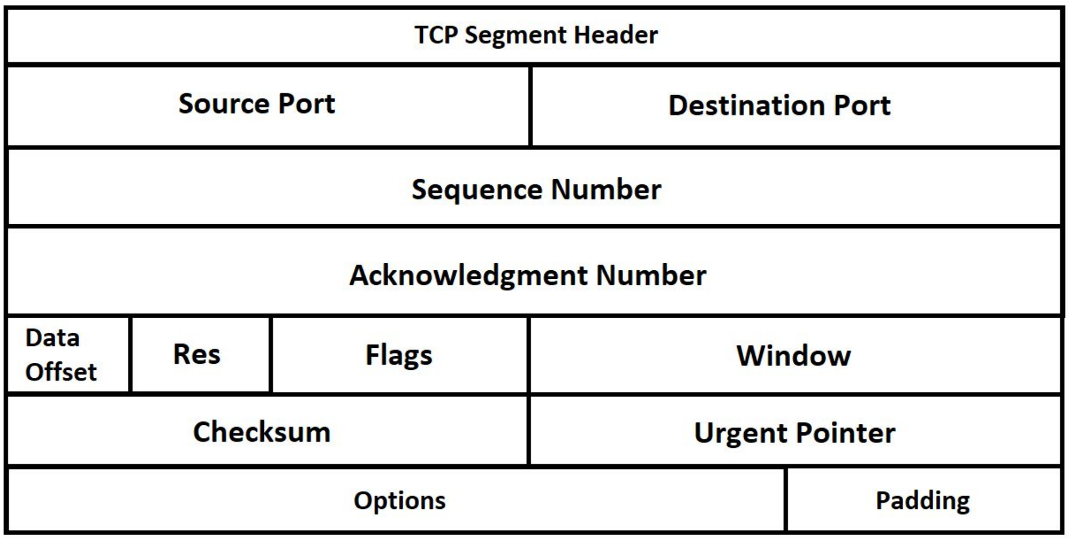
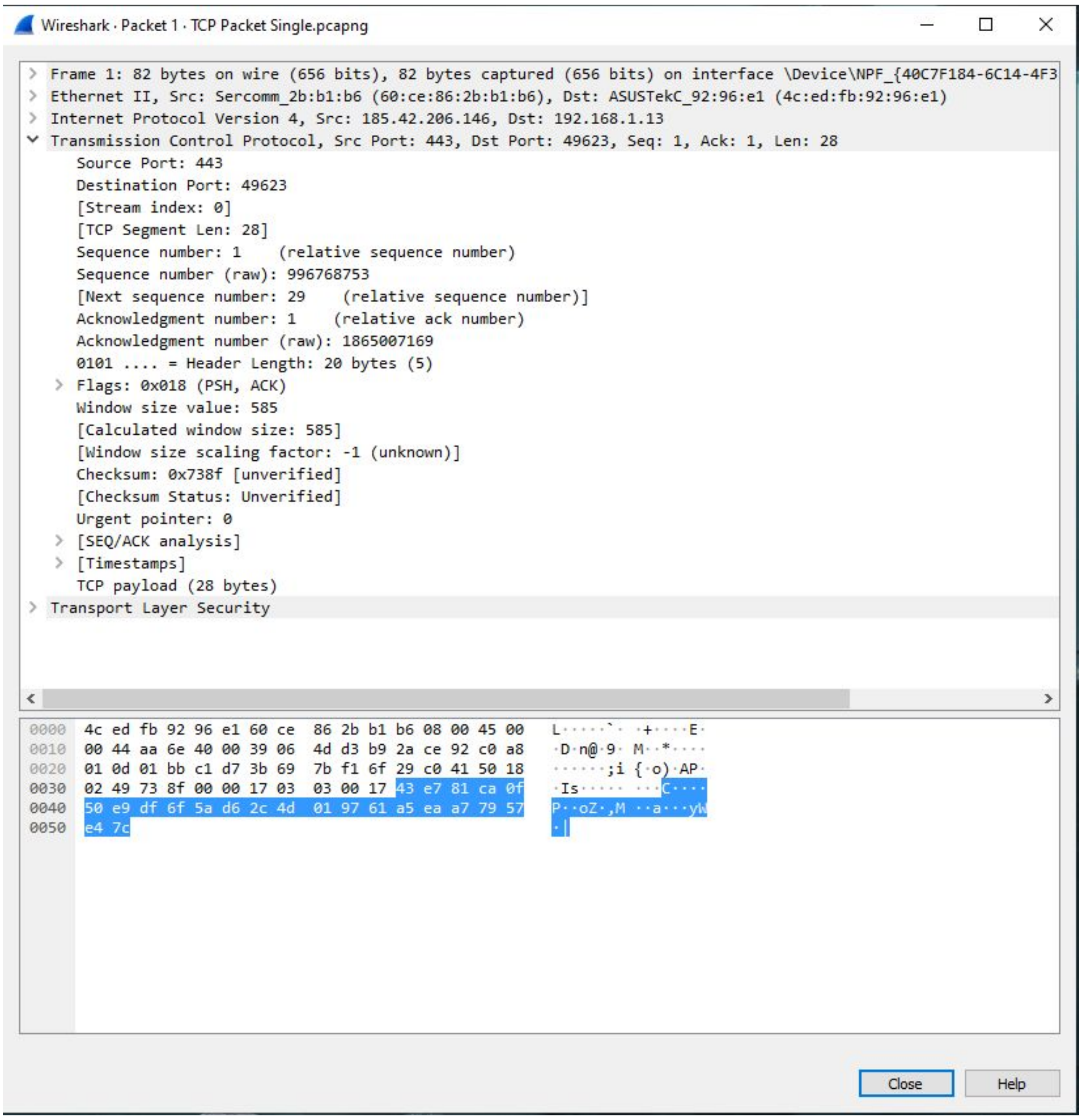
Lab 5

Question 1:





```
> Frame 1: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface \Device\NPF_{40C7F184-6C14-4F3
> Ethernet II, Src: Sercomm_2b:b1:b6 (60:ce:86:2b:b1:b6), Dst: ASUSTekC_92:96:e1 (4c:ed:fb:92:96:e1)
> Internet Protocol Version 4, Src: 185.42.206.146, Dst: 192.168.1.13
v Transmission Control Protocol, Src Port: 443, Dst Port: 49623, Seq: 1, Ack: 1, Len: 28
     Source Port: 443
     Destination Port: 49623
     [Stream index: 0]
     [TCP Segment Len: 28]
     Sequence number: 1    (relative sequence number)
     Sequence number (raw): 996768753
     [Next sequence number: 29    (relative sequence number)]
     Acknowledgment number: 1    (relative ack number)
     Acknowledgment number (raw): 1865007169
     0101 .... = Header Length: 20 bytes (5)
   > Flags: 0x018 (PSH, ACK)
     Window size value: 585
     [Calculated window size: 585]
     [Window size scaling factor: -1 (unknown)]
     Checksum: 0x738f [unverified]
     [Checksum Status: Unverified]
     Urgent pointer: 0
   > [SEQ/ACK analysis]
   > [Timestamps]
     TCP payload (28 bytes)
> Transport Layer Security
```

```
0000  4c ed fb 92 96 e1 60 ce  86 2b b1 b6 08 00 45 00   L·····`· ·+····E·
0010  00 44 aa 6e 40 00 39 06  4d d3 b9 2a ce 92 c0 a8   ·D·n@·9· M··*····
0020  01 0d 01 bb c1 d7 3b 69  7b f1 6f 29 c0 41 50 18   ······;i {·o)·AP·
0030  02 49 73 8f 00 00 17 03  03 00 17 43 e7 81 ca 0f   ·Is····· ···C····
0040  50 e9 df 6f 5a d6 2c 4d  01 97 61 a5 ea a7 79 57   P··oZ·,M ··a···yW
0050  e4 7c                                              ·|
```

Lab 5

Source Port - Is the identification number used by the computer that is sending the packet to identify the port the packet is being sent from. In this scenario it is 443 which is uncommon as the majority of the time the port number is above 1024.

Destination Port - Is the identification number used by the computer that is receiving the packet to identify the port that the packet is being sent to. In this scenario the port id is 49623 which is uncommon as the majority of the time the port number is below 1024.

Sequence Number - Used to segment the data into TCP segments and reassemble them on the other side. The sequence number allows the TCP software on both ends to keep track of how much data has been transferred and to reassemble the data into the correct order, if it is received in the wrong order, and to request data when it has been lost in transit. In this case the SYN flag is clear (0), thus meaning that this is the accumulated sequence number of the first data byte of this segment for the current session.

Acknowledgement Number - If the ACK flag is set then the value of this field is the next sequence number that the sender of the ACK is expecting. This acknowledges receipt of all prior bytes (if any). The first ACK sent by each end acknowledges the other end's initial sequence number itself, but no data. In this case this is true as the ACK flag is set to 1 meaning the next expected sequence value is 29.

Data Offset: Specifies the size of the TCP header in 32-bit words. In this the size of the TCP header is 20 bytes or 5 bit words.

Reserved: For future use and should be set to zero which is true in this case.

Flags: Contains 9 1-bit flags (control bits) as follows:
- Nonce: concealment protection
- Congestion: Congestion window reduced (CWR) flag is set by the sending host to indicate that it received a TCP segment with the ECE flag set and responded in congestion control mechanism. This is not the case in this scenario.
- ECN-Echo: As SYN is set to 0 then this serves as an indication of network congestion (or impending congestion) to the TCP sender.
- Urgent: Indicates that the Urgent pointer field is significant which is not the case here.
- Acknowledgment: Indicates that the Acknowledgment field is significant. All packets after the initial SYN packet sent by the client should have this flag set. Which is the case in this scenario.
- Push: Asks to push the buffered data to the receiving application. Which is true in this case.
- Reset: Reset the connection which is not true in this case.
- Syn: Synchronize sequence numbers. Only the first packet sent from each end should have this flag set. Some other flags and fields change meaning based on this flag, and some are only valid when it is set, and others when it is clear. In this case it is set to true.
- FIN: Last packet from sender.

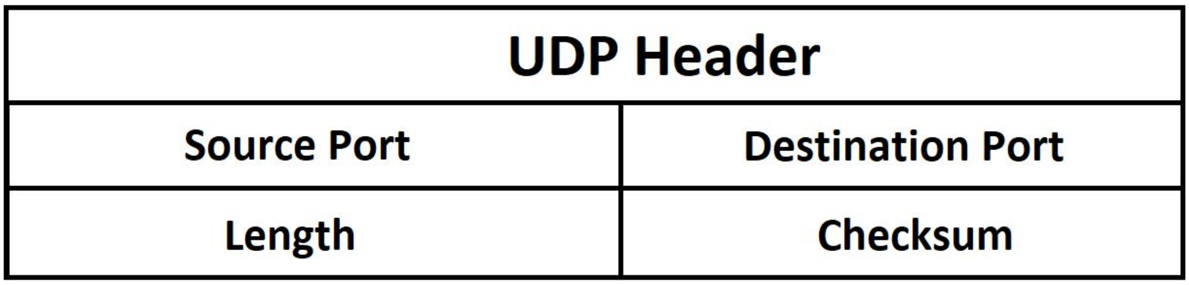
Window: The size of the receive window, which specifies the number of window size units that the sender of this segment is currently willing to receive. In this case the size is 585.

Lab 5

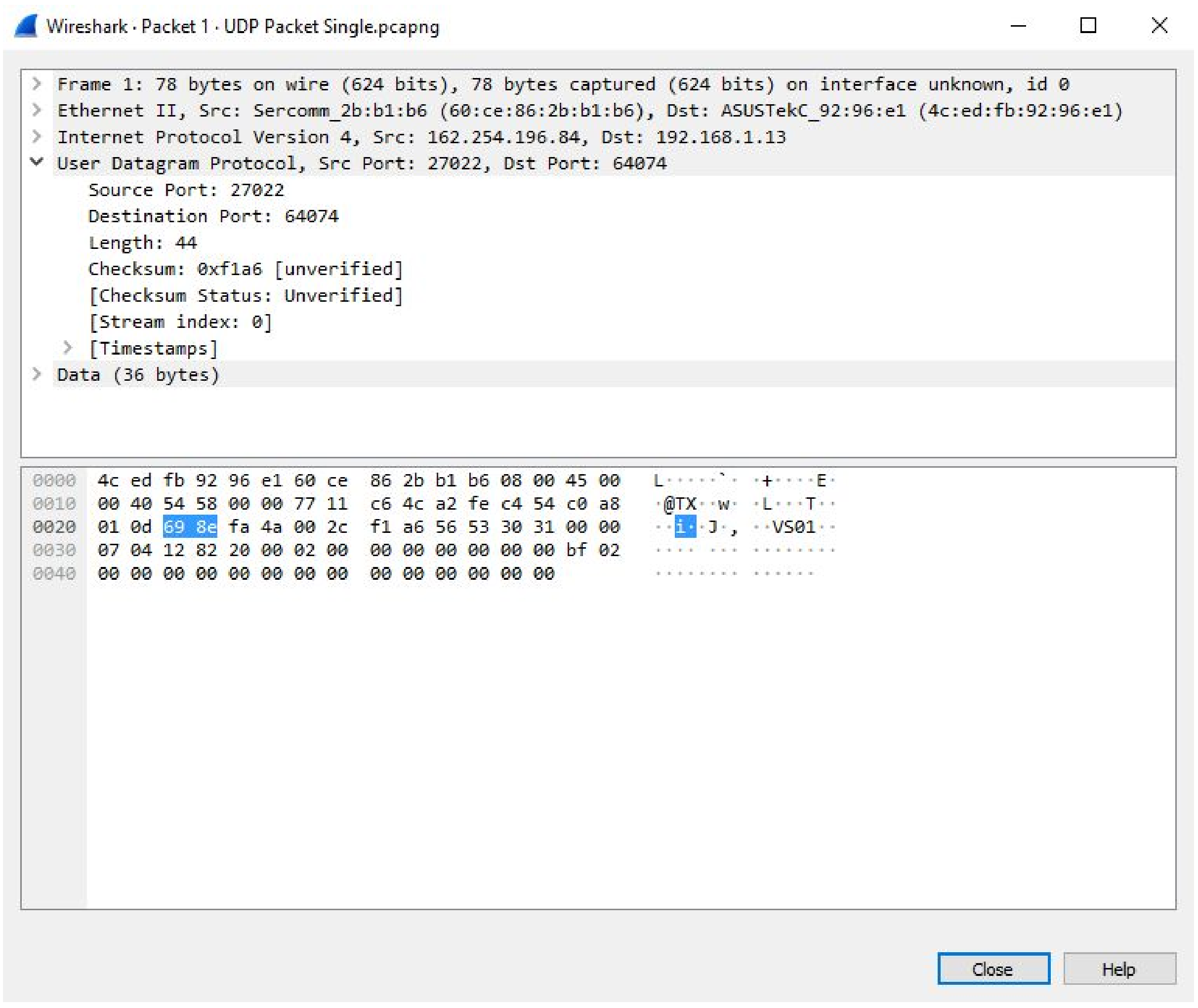
Checksum: The 16-bit checksum field is used for error-checking of the TCP header, the payload and an IP pseudo-header. The pseudo-header consists of the source IP address, the destination IP address, the protocol number for the TCP protocol (6) and the length of the TCP headers and payload (in bytes). In this case the value of the checksum is 0x738f.

Urgent Pointer: If the Urgent flag is set, then this 16-bit field is an offset from the sequence number indicating the last urgent data byte. This is not the case here and thus the Urgent Pointer is set to default value.

Question 2:



UDP Header	
Source Port	Destination Port
Length	Checksum



Wireshark · Packet 1 · UDP Packet Single.pcapng

```
> Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface unknown, id 0
> Ethernet II, Src: Sercomm_2b:b1:b6 (60:ce:86:2b:b1:b6), Dst: ASUSTekC_92:96:e1 (4c:ed:fb:92:96:e1)
> Internet Protocol Version 4, Src: 162.254.196.84, Dst: 192.168.1.13
v User Datagram Protocol, Src Port: 27022, Dst Port: 64074
    Source Port: 27022
    Destination Port: 64074
    Length: 44
    Checksum: 0xf1a6 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  > [Timestamps]
> Data (36 bytes)
```

```
0000  4c ed fb 92 96 e1 60 ce  86 2b b1 b6 08 00 45 00   L·····`· ·+····E·
0010  00 40 54 58 00 00 77 11  c6 4c a2 fe c4 54 c0 a8   ·@TX··w· ·L···T··
0020  01 0d 69 8e fa 4a 00 2c  f1 a6 56 53 30 31 00 00   ··i··J·, ··VS01··
0030  07 04 12 82 20 00 02 00  00 00 00 00 00 00 bf 02   ········ ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00         ········ ······
```

Source Port - Is the identification number used by the computer that is sending the packet to identify the port the packet is being sent from. In this scenario it is 27022.

Lab 5

Destination Port - Is the identification number used by the computer that is receiving the packet to identify the port that the packet is being sent to. In this scenario the port id is 64074.

Length: The length in bytes of the UDP header and any encapsulated data. In this case the size is 44 bytes.

Checksum: The checksum field may be used for error-checking of the header and data. This field is optional in IPv4, and mandatory in IPv6. The field carries all-zeros if unused. In this case the value of the checksum is 0xf1a6.

Question 3:
a2fe + c454 = 16752
16752 + c0a8 = 227FA
227FA + 010d = 22907
22907 + 0011 = 22918
22918 + 002c = 22944
22944 + 698e = 292D2
292D2 + fa4a = 38D1C
38D1C + 002c = 38D48
38D48 + 5653 = 3E39B
3E39B + 3031 = 413CC
413CC + 0000 = 413CC
413CC + 0704 = 41AD0
41AD0 + 1282 = 42D52
42D52 + 2000 = 44D52
44D52 + 0200 = 44F52
44F52 + 0000 = 44F52
44F52 + 0000 = 44F52
44F52 + 0000 = 44F52
44F52 + bf02 = 50E54

1010000111001010100 Add upper with lower
=
0E59

To Binary
0000111001011001

One's Compliment
1111000110100110

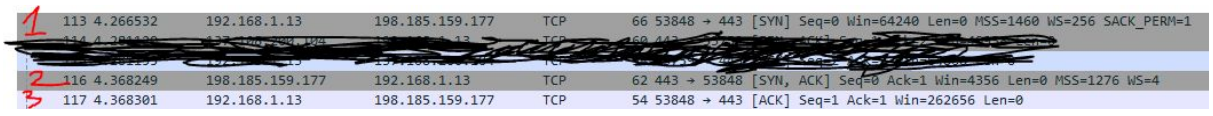
Convert to Hex
F1a6

Question 4

Lab 5

Unfortunately I was not able to pinpoint any of the files from a streaming service while running wireshark. Though it should be using a combination of TCP and UDP protocols in which case the UDP just be mostly as the services does not care if some frames are lost as speed is the top priority.

Question 5



```
1   113 4.266532    192.168.1.13      198.185.159.177   TCP      66 53848 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
[redacted]
2   116 4.368249    198.185.159.177   192.168.1.13      TCP      62 443 → 53848 [SYN, ACK] Seq=0 Ack=1 Win=4356 Len=0 MSS=1276 WS=4
3   117 4.368301    192.168.1.13      198.185.159.177   TCP      54 53848 → 443 [ACK] Seq=1 Ack=1 Win=262656 Len=0
```

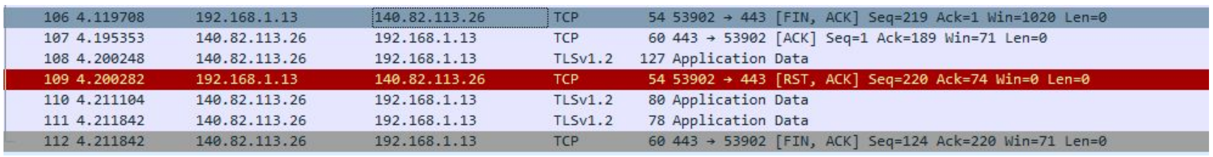
In this scenario I try to connect ot github:

Step 1 (SYN) : In the first step, the client wants to establish a connection with the server, so it sends a segment with SYN(Synchronize Sequence Number) which informs the server that client is likely to start communication and with what sequence number it starts segments with. In this case I inform github.

Step 2 (SYN + ACK): Server responds to the client request with SYN-ACK signal bits set. Acknowledgement(ACK) stands for the fact the response of segment it received and SYN stand for that with what sequence number it is likely to start the segments with. In this case github responds back.

Step 3 (ACK) : In the final part the client acknowledges the response of the server and they both establish a reliable connection with which they will start the actual data transfer. In this case I establish the transfer of data with github.

Question 6



```
106 4.119708    192.168.1.13      140.82.113.26     TCP       54 53902 → 443 [FIN, ACK] Seq=219 Ack=1 Win=1020 Len=0
107 4.195353    140.82.113.26     192.168.1.13      TCP       60 443 → 53902 [ACK] Seq=1 Ack=189 Win=71 Len=0
108 4.200248    140.82.113.26     192.168.1.13      TLSv1.2  127 Application Data
109 4.200282    192.168.1.13      140.82.113.26     TCP       54 53902 → 443 [RST, ACK] Seq=220 Ack=74 Win=0 Len=0
110 4.211104    140.82.113.26     192.168.1.13      TLSv1.2   80 Application Data
111 4.211842    140.82.113.26     192.168.1.13      TLSv1.2   78 Application Data
112 4.211842    140.82.113.26     192.168.1.13      TCP       60 443 → 53902 [FIN, ACK] Seq=124 Ack=220 Win=71 Len=0
```

Step 1 - I am trying to close the connection with github and thus I send them a FIN and ACK flag to inform them that I am trying to close the connection.

Step 2 - When the server receives the FIN flag it sends back the ACK flag back to me.

Step 3 - I then enter the second waiting face to receive FIN flag from the server side.

Step 4 - Server sends FIN bit segment to me after some time when Server sends the ACK segment because I am closing it back.

Step 5 - Lastly I would send a final ACK to inform the server that it should release all my data as I have fully closed connection with them.