

Assignment 3

Question 1: Understanding UDP Protocol

#1-A:

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

The screenshot shows a Wireshark packet capture of a UDP packet. The packet list at the top shows two packets, both of which are UDP. The packet details pane shows the structure of the packet, including the Ethernet II header, Internet Protocol Version 4 header, and User Datagram Protocol header. The UDP header fields are expanded, showing the source port (49154), destination port (6667), length (188), and checksum (0xf1ae). The protocol number is shown as 17 (decimal) and 0x11 (hexadecimal). The packet bytes pane at the bottom shows the raw data of the packet, with the 0x11 value highlighted in the second byte of the UDP header.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.4	255.255.255.255	UDP	230	49154 → 6667 Len=188
11	5.327570	192.168.0.4	255.255.255.255	UDP	230	49154 → 6667 Len=188

```

0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 216
Identification: 0x08ba (2234)
> Flags: 0x0000
...0 0000 0000 0000 = Fragment offset: 0
Time to live: 255
Protocol: UDP (17)
Header checksum: 0xf1ae [validation disabled]
[Header checksum status: Unverified]
Source: 192.168.0.4
  
```

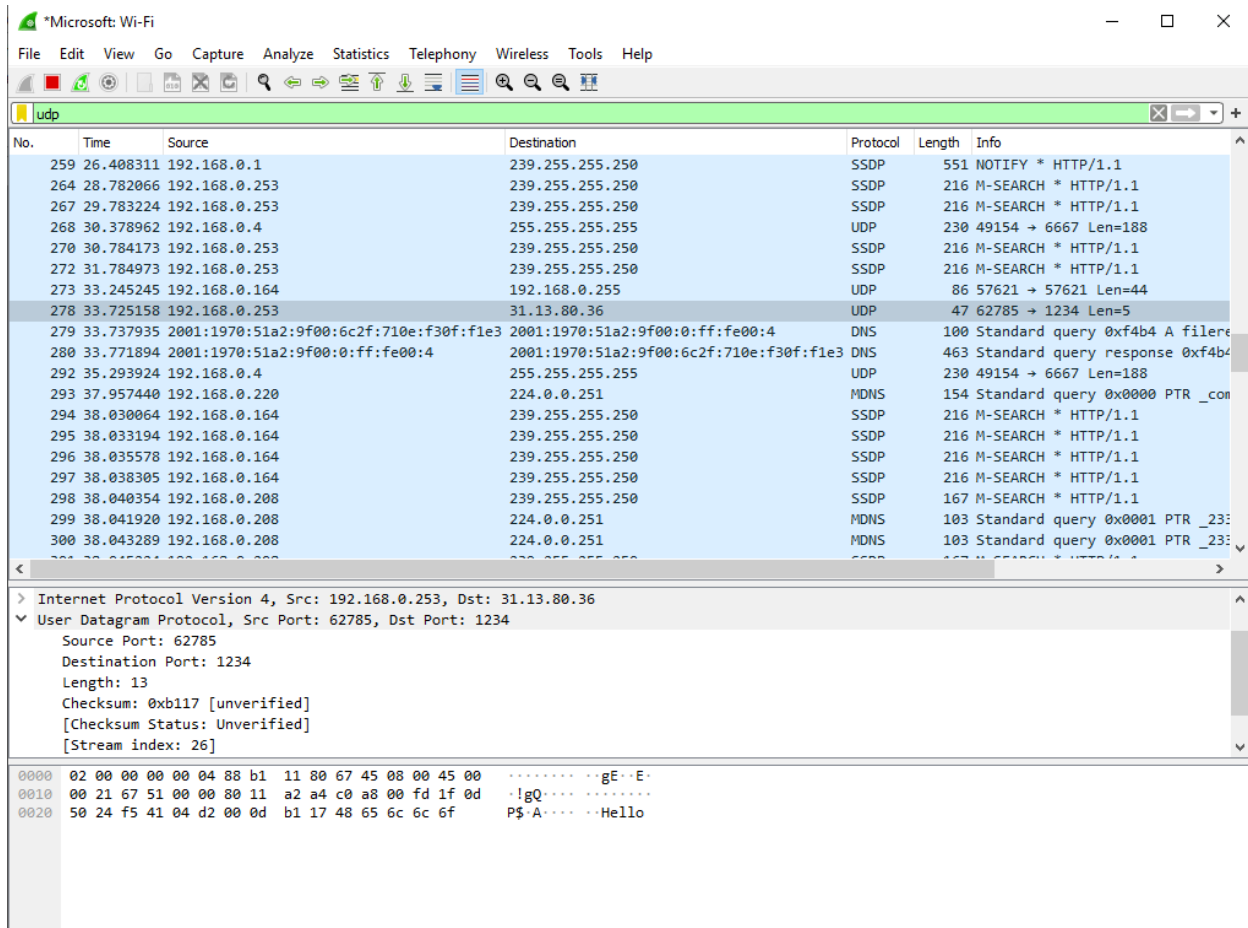
0010 00 d8 08 ba 00 00 f1 ae c0 a8 00 04 ff ff
 0020 ff ff c0 02 1a 0b 00 c4 34 a3 00 00 55 aa 00 004...U...
 0030 00 00 00 00 00 13 00 00 00 ac 00 00 00 00 23 d0#

As clearly seen in the above screenshot, UDP's protocol number is 17 as a decimal, and 0x11 in hexadecimal notation. I have highlighted these fields in the wireshark packet contents body above.

#1-B:

To calculate UDP checksum, we first must know that in addition to its own header, UDP checksum uses a pseudo header, consisting of the original source IP, destination IP, reserved (0000 0000), protocol (0x11), and the length from the UDP header. This gets added with the actual UDP header, consisting of a source port, destination port, length, and actual data.

The following screenshot illustrates the UDP packet I used for this question. It has as it's data the string "Hello."



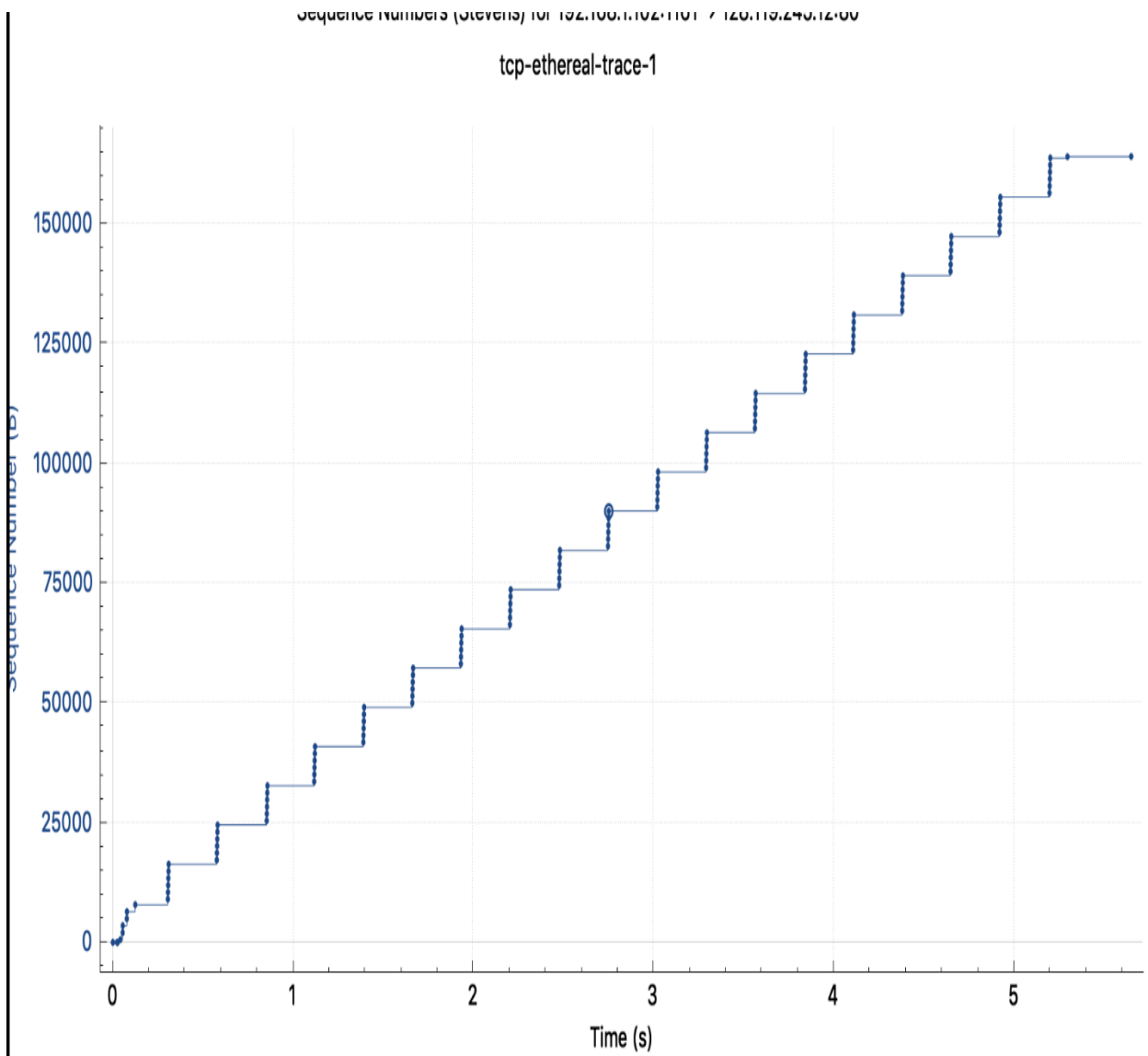
Byte #	Data/Content (in hex)	Current rolling sum
PSEUDO HEADER	PSEUDO HEADER	0
26-29 (Source IP)	C0A8 00FD	C1A5
30-33 (Destination IP)	1F0D 5024	130D6 -> 30D7 (after overflow addition)
23 (UDP Protocol) plus reserve	0011	30E8
38-39 (UDP Length)	000D	30F5
UDP HEADER	UDP HEADER	30F5
34-35 (UDP Source port)	F1AA	1229F -> 22A0 (after overflow addition)
36-37 (UDP Destination port)	04D2	2772
38-39 (UDP Length)	000D	277F
42-46 (UDP Data)	4865 6c6c 6f00	4B51

TAKE ONE'S COMPLEMENT		FFFF – 4B51 = B4AE (Calculated Checksum)
40-41 (Actual Checksum)	0xB4AE	CORRECT!!!!

Comparing my calculated checksum with the actual wireshark captured checksum, we can verify that my solution was correct.

Question 2: Understanding TCP Protocol

#2-A:



The TCP slowstart phase begins when the connection is initialized (When the HTTP POST segment is sent out). Although the identification of the TCP slowstart phase and congestion avoidance phase depends on the value of the congestion window size of the TCP sender, we unfortunately cannot obtain

the exact value of the congestion window size directly from the Time Sequence Graph. However, we can estimate the lower bound of this value by the amount of outstanding data, as this represents the amount of data without acknowledgement. Also, we know that the TCP window is constrained by the receiver window size and the receiver buffer can act as the upper bound of the TCP window size. Since the receiver buffer isn't the bottleneck in this trace, we can use the lower bound of the TCP window size.

However, despite this, we cannot determine the exact end of the slow start phase and the start of the congestion avoidance phase for this trace. The major reason behind this is that the TCP sender is not pushing enough throughput to enter the congestion state. This means that before the end of the slow start phase, the application is already stopping transmission temporally. In the above graph, however, we can see the estimated congestion avoidance phase represented by the flatline, when the sequence number isn't increasing with regards to time.

In the text, the idealized behaviour of TCP assumes that TCP senders are extremely aggressive in sending data. Since traffic may congest the network layer, TCP senders ideally should follow the AIMD algorithm to detect packet loss, and should drop their sending window size. However, in practice, TCP behaviour also depends greatly on the application. For example, in a web application, some web objects are extremely small, and thus, before the end of the slow start phase, the transmission is already over, thus suffering from the long delays of the slow start phase of TCP.

#2-B:

Bytes 46-47 indicate the flags that are set in the TCP header, corresponding to which type of TCP segment is being sent. *Note: only half of byte 46 is used for the Flags, the other half is part of the Header Length: which is 32 Bytes (8).

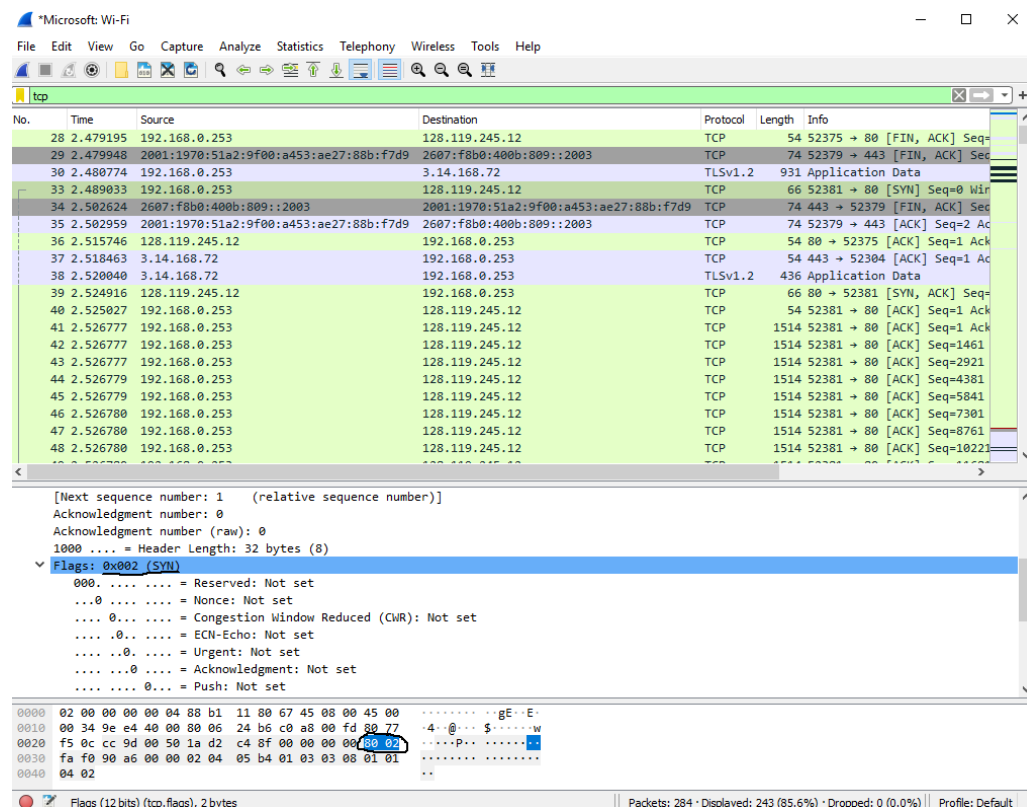
#2-C:

First, I will add the table:

Segment Type	Data/Content of Byte #46-47 (Hex)	Data/Content of Byte #46-47 (Binary)
SYN	0x002	0000 0000 0010
SYN-ACK	0x012	0000 0001 0010
ACK	0x010	0000 0001 0000
DATA	0x010	0000 0001 0000
FIN-ACK	0x011	0000 0001 0001

Next, I will add the screenshots:

SYN:



Microsoft Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp

No.	Time	Source	Destination	Protocol	Length	Info
28	2.479195	192.168.0.253	128.119.245.12	TCP	54	52375 → 80 [FIN, ACK] Seq=...
29	2.479948	2001:1970:51a2:9f00:a453:ae27:88b:f7d9	2607:f8b0:400b:809::2003	TCP	74	52379 → 443 [FIN, ACK] Seq=...
30	2.480774	192.168.0.253	3.14.168.72	TLsv1.2	931	Application Data
33	2.489033	192.168.0.253	128.119.245.12	TCP	66	52381 → 80 [SYN] Seq=0 Win=...
34	2.502624	2607:f8b0:400b:809::2003	2001:1970:51a2:9f00:a453:ae27:88b:f7d9	TCP	74	443 → 52379 [FIN, ACK] Seq=...
35	2.502959	2001:1970:51a2:9f00:a453:ae27:88b:f7d9	2607:f8b0:400b:809::2003	TCP	74	52379 → 443 [ACK] Seq=2 Ac...
36	2.515746	128.119.245.12	192.168.0.253	TCP	54	80 → 52375 [ACK] Seq=1 Ac...
37	2.518463	3.14.168.72	192.168.0.253	TCP	54	443 → 52304 [ACK] Seq=1 Ac...
38	2.520040	3.14.168.72	192.168.0.253	TLsv1.2	436	Application Data
39	2.524916	128.119.245.12	192.168.0.253	TCP	66	80 → 52381 [SYN, ACK] Seq=...
40	2.525027	192.168.0.253	128.119.245.12	TCP	54	52381 → 80 [ACK] Seq=1 Ac...
41	2.526777	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=1 Ac...
42	2.526777	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=1461
43	2.526777	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=2921
44	2.526779	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=4381
45	2.526779	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=5841
46	2.526780	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=7301
47	2.526780	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=8761
48	2.526780	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=10221

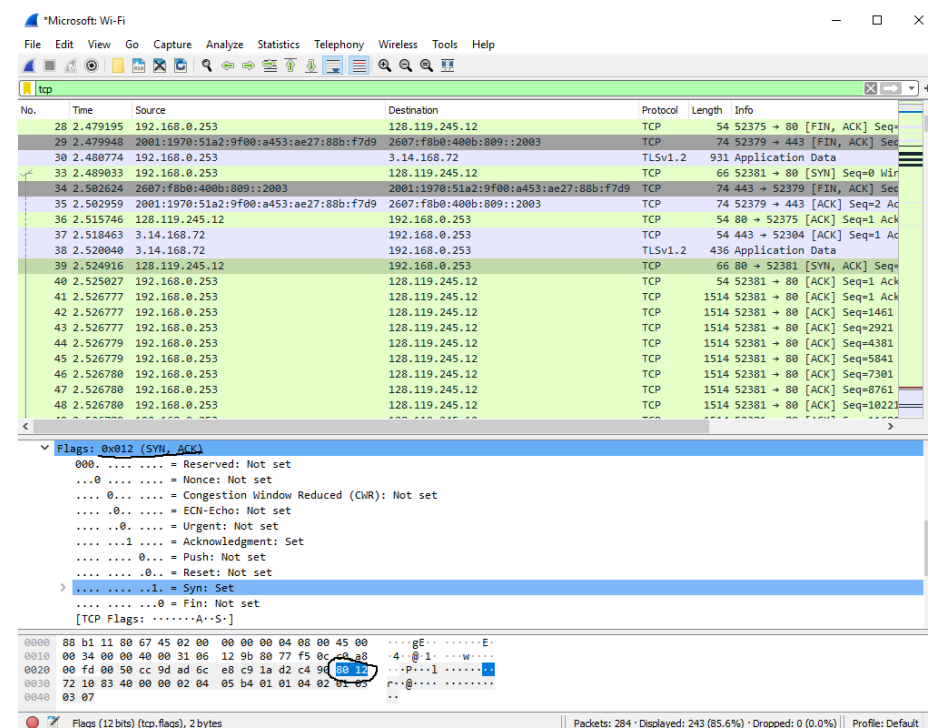
[Next sequence number: 1 (relative sequence number)]
 Acknowledgment number: 0
 Acknowledgment number (raw): 0
 1000 = Header Length: 32 bytes (8)
 Flags: 0x002 (SYN)
 000. = Reserved: Not set
 ...0 = Nonce: Not set
0. = Congestion Window Reduced (CWR): Not set
0. = ECN-Echo: Not set
0. = Urgent: Not set
0. = Acknowledgment: Not set
0. = Push: Not set

0000 02 00 00 00 00 04 88 b1 11 80 67 45 08 00 45 00gE...E
 0010 00 34 9e e4 40 00 00 06 24 b6 c0 a8 00 fd 80 77 4. @. \$.w
 0020 f5 0c cc 9d 00 50 1a d2 c4 8f 00 00 00 60 0aP.....
 0030 fa f0 90 a6 00 00 02 04 05 b4 01 03 08 01 01
 0040 04 02

Flags (12 bits) (tcp.flags), 2 bytes

Packets: 284 · Displayed: 243 (85.6%) · Dropped: 0 (0.0%) · Profile: Default

SYN-ACK:



Microsoft Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp

No.	Time	Source	Destination	Protocol	Length	Info
28	2.479195	192.168.0.253	128.119.245.12	TCP	54	52375 → 80 [FIN, ACK] Seq=...
29	2.479948	2001:1970:51a2:9f00:a453:ae27:88b:f7d9	2607:f8b0:400b:809::2003	TCP	74	52379 → 443 [FIN, ACK] Seq=...
30	2.480774	192.168.0.253	3.14.168.72	TLsv1.2	931	Application Data
33	2.489033	192.168.0.253	128.119.245.12	TCP	66	52381 → 80 [SYN] Seq=0 Win=...
34	2.502624	2607:f8b0:400b:809::2003	2001:1970:51a2:9f00:a453:ae27:88b:f7d9	TCP	74	443 → 52379 [FIN, ACK] Seq=...
35	2.502959	2001:1970:51a2:9f00:a453:ae27:88b:f7d9	2607:f8b0:400b:809::2003	TCP	74	52379 → 443 [ACK] Seq=2 Ac...
36	2.515746	128.119.245.12	192.168.0.253	TCP	54	80 → 52375 [ACK] Seq=1 Ac...
37	2.518463	3.14.168.72	192.168.0.253	TCP	54	443 → 52304 [ACK] Seq=1 Ac...
38	2.520040	3.14.168.72	192.168.0.253	TLsv1.2	436	Application Data
39	2.524916	128.119.245.12	192.168.0.253	TCP	66	80 → 52381 [SYN, ACK] Seq=...
40	2.525027	192.168.0.253	128.119.245.12	TCP	54	52381 → 80 [ACK] Seq=1 Ac...
41	2.526777	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=1 Ac...
42	2.526777	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=1461
43	2.526777	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=2921
44	2.526779	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=4381
45	2.526779	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=5841
46	2.526780	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=7301
47	2.526780	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=8761
48	2.526780	192.168.0.253	128.119.245.12	TCP	1514	52381 → 80 [ACK] Seq=10221

Flags: 0x012 (SYN, ACK)
 000. = Reserved: Not set
 ...0 = Nonce: Not set
0. = Congestion Window Reduced (CWR): Not set
0. = ECN-Echo: Not set
0. = Urgent: Not set
1. = Acknowledgment: Set
0. = Push: Not set
0. = Reset: Not set
1. = Syn: Set
0. = Fin: Not set
 [TCP Flags:A.S.]

0000 88 b1 11 80 67 45 02 00 00 00 00 04 08 00 45 00gE...E
 0010 00 34 00 00 00 31 06 12 9b 00 77 f5 0c 00 00 4. @. l. ...w
 0020 00 fd 00 00 cc 9d ad 6c e0 c9 1a d2 c4 90 00 00P+...l.....
 0030 72 10 83 40 00 00 02 04 05 b4 01 01 04 02 01 07
 0040 03 07

Flags (12 bits) (tcp.flags), 2 bytes

Packets: 284 · Displayed: 243 (85.6%) · Dropped: 0 (0.0%) · Profile: Default

ACK:

The screenshot displays the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons for file operations, capture control, and analysis. The main window is divided into three panes:

- Packet List Pane:** Shows a list of captured packets. Packet 40 is selected, which is a TCP packet with source 192.168.0.253 and destination 128.119.245.12. The packet size is 60 bytes.
- Packet Details Pane:** Shows the hierarchical structure of the selected packet. The 'Flags' field is expanded, showing '0x010 (ACK)'. The 'Reserved' bit is set, and the 'Ack' bit is set.
- Packet Bytes Pane:** Shows the raw data of the packet in hexadecimal and ASCII. The bytes '60 10' are highlighted, which correspond to the 'ACK' flag in the TCP header.

The status bar at the bottom indicates that 284 packets are displayed, with 243 (85.6%) shown and 0 dropped. The profile is set to 'Default'.

FIN-ACK:

The screenshot displays the Wireshark network protocol analyzer interface. At the top, the title bar reads "Microsoft: Wi-Fi". The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons for packet capture and analysis. The main window is divided into three panes:

- Packet List:** Shows a list of captured packets. The selected packet is #36, a TCP packet from 192.168.0.253 to 128.119.245.12, with sequence number 52375 and length 54. The protocol is TCP.
- Packet Details:** Shows the structure of the selected packet. The top section is the Ethernet II header, and the bottom section is the TCP header. The TCP header fields are expanded, showing the source port 52375 and destination port 52381. The sequence number is 52375, and the acknowledgment number is 52381. The window size is 65535, and the flags are SYN.
- Packet Bytes:** Shows the raw data of the packet in hexadecimal and ASCII. The sequence number 52375 is highlighted in blue.

The status bar at the bottom indicates "Packets: 284 / Displayed: 243 (85.6%) / Dropped: 0 (0.0%)".

#2-D:

Similar to UDP, to calculate TCP checksum, we first must know that in addition to its own header, TCP checksum uses a pseudo header, consisting of the original source IP, destination IP, reserved (0000 0000), protocol (0x11), and the length from the TCP header.

Byte #	Data/Content (in hex)	Current rolling sum
PSEUDO HEADER	PSEUDO HEADER	0
26-29 (Source IP)	C0A8 00FD	C1A5
30-33 (Destination IP)	82D3 1035	154AD -> 54AE After Overflow addition
23 (TCP Protocol) plus reserve	0006	54B4
16-17 (TCP Length)	0029	54DD
UDP HEADER	UDP HEADER	54DD
34-35 (Source Port)	EC59	14136 -> 4137 After Overflow Addition
36-37 (Destination Port)	01BB	42F2
38-41 (Sequence Number)	737D 1FF8	D667
42-45 (Acknowledgement Number)	E3B4 7538	2F55
46-47 (Header Length + Flags)	5010	7F65
48-49 (Window Size Value)	00FF	8064
52-53 (Urgent Pointer)	0000	8064
54 (Data)	0000	8064
TAKE ONE'S COMPLEMENT		FFFF – 8064 = 7FAF Calculated Checksum
50-51 (Actual Checksum)		7FAF CORRECT!!!!