hw5

HW 5

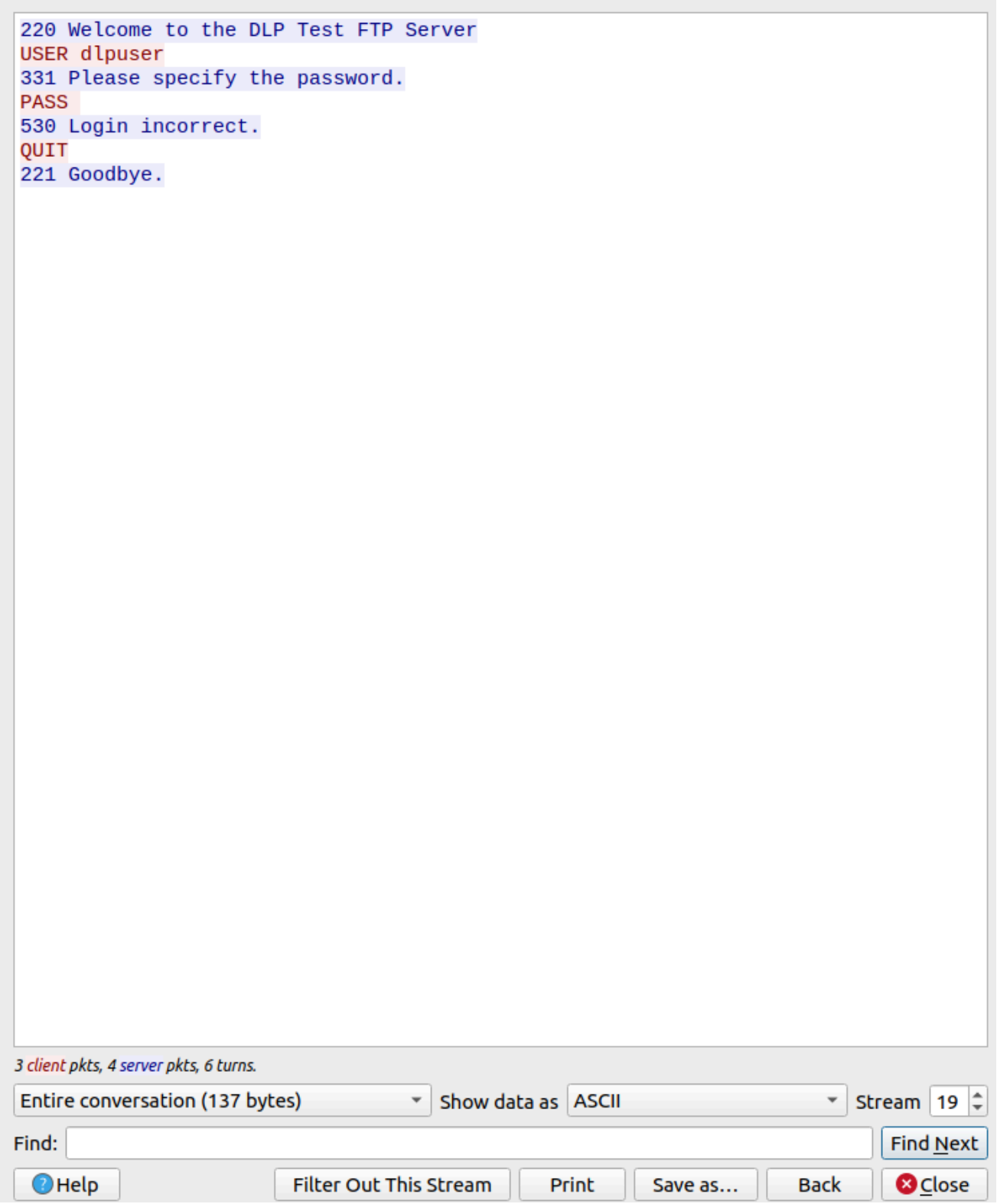
Part 1

1. a) TCP is a protocol that is reliable with flow and congestion control. If there are any dropped packets or errors or anything of the sort, a new packet will be delivered that is correct. The delivery of the packets is also controlled so the end user is not overwhelmed. UDP on the other hand is much faster but is not reliable. There is no guarantee that the packet reached and/or isn't corrupted. This is because unlike TCP, UDP has no connections it is just sending a packet. There is no feedback (ACK) like with TCP for when the packet reaches correctly.
   b) QUIC runs on top of UDP so it is low latency and uses the UDP transport method. Like TCP, it has congestion/flow control and other similar mechanisms to make it reliable. It is significantly faster than TCP though. It also has multiplexing allowing for multiple data to be sent over one connection as well as encryption built in (instead of TLS)
   c) QUIC is supported by all browsers and HTTP/3. It is also over half of Google's full traffic. According to https://w3techs.com/technologies/details/ce-quic it is used by a little over 8% of all websites
2. 01010011 + 01100110 = 10111001 + 01110100 = 0010_1101, complement = 1101_0010. The complement is used because you can add the original sum to the complement and then get all 1s. This is a lot easier for the computer to check than individual bits. The complement also makes it endian independent. If just the sum is used then the computer doesn't know if the start is the largest number or the end. Errors are detected if the sum of all the bits + the checksum given does not equal all 1s. 1-bit errors will always be detected while 2-bit errors will not be.
3. a) Source is 1500 and destination is 1200
   b) Yes, the source IP will be different for both packets
   c) https://en.wikipedia.org/wiki/File_Transfer_Protocol 20 and 21
   d) They will all pass through a welcoming socket which establishes a separate socket for each of host A and B. The separate sockets will be ID'd using the source port and IP address. The sockets will be different but the number will be the same
4. a) Sequence numbers are needed to make sure all the packets arrive and the correct order they are supposed to be in. The receiver doesn't know the order or how many packets in total so this allows for the receiver to realize a packet is missing and keep them in the correct order
   b) Timers are needed in case a packet is dropped. If a packet is dropped, the sender

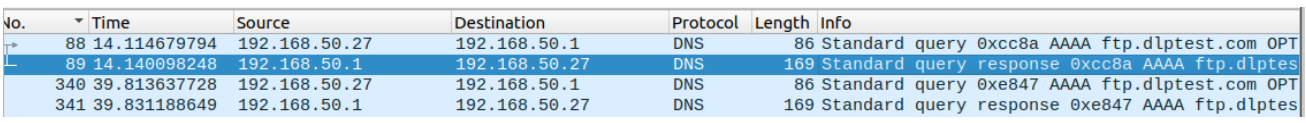
doesn't know so they set a timer so that after a certain point, they assume it is dropped and they send another one. If there wasn't one then the packet exchange would stall as the sender would be waiting to receive a packet before sending another one.
c) Yes because if the delay is constant then the timer should be that constant time. The only time it is not needed is if there is a guarantee that packets wont be dropped.

5. a) Telnet is on port 23 and it is like an SSH. You access a CLI remotely but it is insecure with no encryption so it is kind of obsolete.
b)
i) A: Arbitrary, lets say 8100. S = 23
ii) B: Arbitrary, lets say 8101. S = 23
iii) A: Same as before 8100. S = 23
iv) B: Same as before 8101. S = 23
v) Yes, the sources aren't aware of each other so they could randomly use the same one. It doesn't matter since the source IP is also used as the key
vi) If they are the same host then it needs to be different because then source IP would be the same and if the port is too then the key would be the same making it impossible to distinguish

Part 2

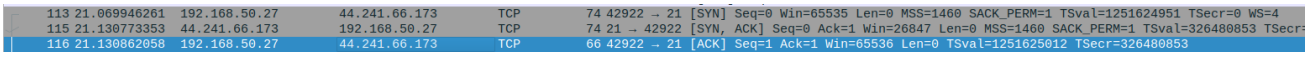
1. 

I could see me messing up the password (but not the password itself) as well as me doing it correctly. The password is blank whether it is right or not

2. 

No.	Time	Source	Destination	Protocol	Length	Info
88	14.114679794	192.168.50.27	192.168.50.1	DNS	86	Standard query 0xcc8a AAAA ftp.dlptest.com OPT
89	14.140098248	192.168.50.1	192.168.50.27	DNS	169	Standard query response 0xcc8a AAAA ftp.dlptes
340	39.813637728	192.168.50.27	192.168.50.1	DNS	86	Standard query 0xe847 AAAA ftp.dlptest.com OPT
341	39.831188649	192.168.50.1	192.168.50.27	DNS	169	Standard query response 0xe847 AAAA ftp.dlptes

DNS was used to find it and the local DNS found it successfully, probably because this is my second time trying it as I forgot to turn on wireshark the first time. The IP was 44.241.66.173

3. 

   The first TCP is a syn and is in a different color. Wireshark specifies it as a [SYN] packet making it easier to spot. It is also outgoing. The one after was a SYN ACK so that is how I was sure it was correct and that was incoming which is how I know it was correct.

4.

Table 1	
Source IP	192.168.50.27
Dest IP	44.241.66.173
Source port	42922
Dest port	21
seq num	0

ack num	The info section shows nothing but when I look at the sumamry it says 0
	`Acknowledgment Number: 0`
	`Acknowledgment number (raw): 0`

header length	40 bytes
window size	65535

```
    Source Address: 192.168.50.27
    Destination Address: 44.241.66.173
▾ Transmission Control Protocol, Src Port: 42922, Dst Port: 21, Seq: 0, Len: 0
    Source Port: 42922
    Destination Port: 21
    [Stream index: 4]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1735970723
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  ▸ Flags: 0x002 (SYN)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x7436 [unverified]
```

Only SYN

```
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
    .... .... .0.. = Reset: Not set
  ▸ .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
```

Table 2	
Source IP	44.241.66.173
Dest IP	192.168.50.27
Source port	21
Dest port	42922
seq num	0
ack num	1
header length	40 bytes
window size	26847

```
    Source Address: 44.241.66.173
    Destination Address: 192.168.50.27
▾ Transmission Control Protocol, Src Port: 21, Dst Port: 42922, Seq: 0, Ack: 1, Len: 0
    Source Port: 21
    Destination Port: 42922
    [Stream index: 4]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 558078767
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1735970724
    1010 .... = Header Length: 40 bytes (10)
  ▸ Flags: 0x012 (SYN, ACK)
    Window: 26847
    [Calculated window size: 26847]
    Checksum: 0x8783 [unverified]
```

SYN and ACK

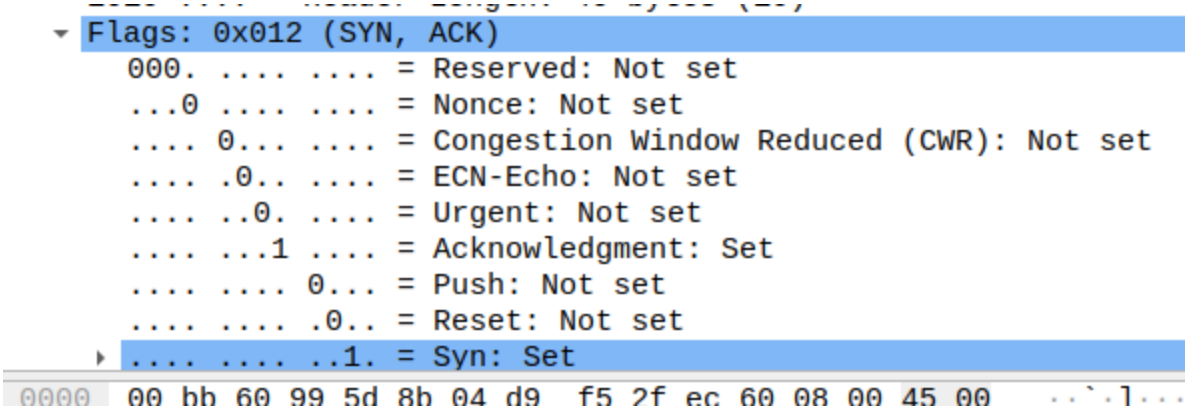
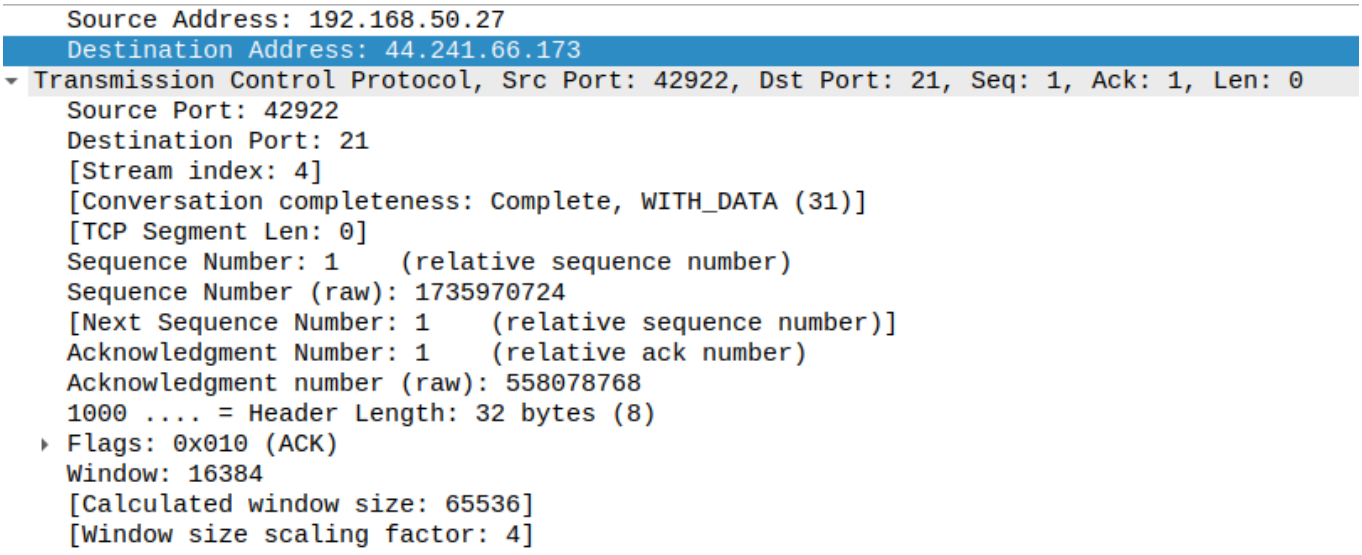
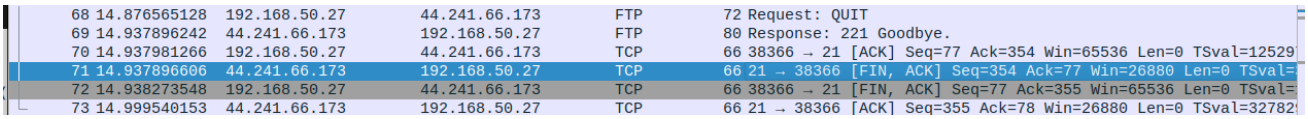


Table 3	
Source IP	192.168.50.27
Dest IP	44.241.66.173
Source port	42922
Dest port	21
seq num	1
ack num	1
header length	32 bytes
window size	65535



Only ACK

```
▾ Flags: 0x010 (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
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
```

5. 

68	14.876565128	192.168.50.27	44.241.66.173	FTP	72	Request: QUIT
69	14.937896242	44.241.66.173	192.168.50.27	FTP	80	Response: 221 Goodbye.
70	14.937981266	192.168.50.27	44.241.66.173	TCP	66	38366 → 21 [ACK] Seq=77 Ack=354 Win=65536 Len=0 TSval=12529
71	14.937896606	44.241.66.173	192.168.50.27	TCP	66	21 → 38366 [FIN, ACK] Seq=354 Ack=77 Win=26880 Len=0 TSval=
72	14.938273548	192.168.50.27	44.241.66.173	TCP	66	38366 → 21 [FIN, ACK] Seq=77 Ack=355 Win=65536 Len=0 TSval=
73	14.999540153	44.241.66.173	192.168.50.27	TCP	66	21 → 38366 [ACK] Seq=355 Ack=78 Win=26880 Len=0 TSval=32782

So first, the server sends a FIN, ACK packet to me starting the data close. The ACK is to notify the last packet received by the server and the fin is to start the end of communication. The client then responds with a FIN ACK acknowledging it got the last FIN and sending its own FIN telling the server it is done on its side too. This termination ensures all the info before it is received and that both sides acknowledge each others termination