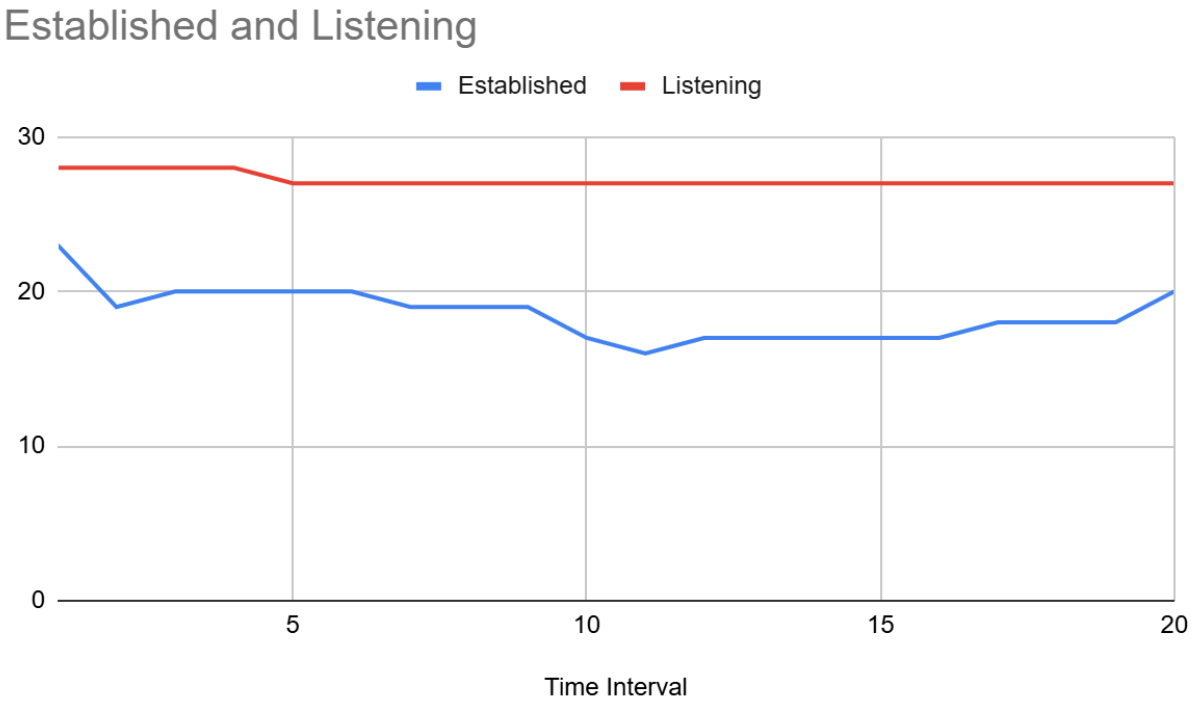
Task 1.1:

1. Command Script

```
@echo off
echo Time, ESTABLISHED, LISTENING > netstat_log.csv
:loop
for /f "tokens=*" %%t in ('powershell -command "Get-Date -Format 'HH:mm:ss'"') do set timestamp=%%t
set /a established=0
set /a listening=0
for /f "tokens=1,4" %%a in ('netstat -an ^| findstr /I "ESTABLISHED LISTENING"') do (
    if "%%b"=="ESTABLISHED" set /a established+=1
    if "%%b"=="LISTENING" set /a listening+=1
)
echo %timestamp%, %established%, %listening% >> netstat_log.csv
echo %timestamp% - ESTABLISHED: %established%, LISTENING: %listening%
timeout /t 30 /nobreak >nul
goto loop
```

2. Graph



Task 1.2

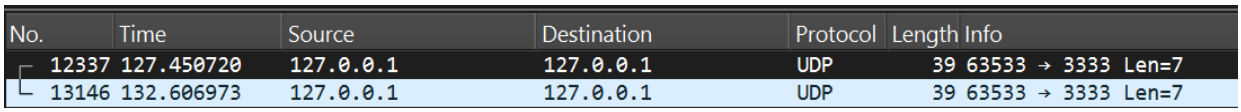
Step 1

No.	Time	Source	Destination	Protocol	Length	Info
7461	74.324680	127.0.0.1	127.0.0.1	TCP	56	49775 → 3333 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
7462	74.324755	127.0.0.1	127.0.0.1	TCP	56	3333 → 49775 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
7463	74.324784	127.0.0.1	127.0.0.1	TCP	44	49775 → 3333 [ACK] Seq=1 Ack=1 Win=65280 Len=0
7560	77.583938	127.0.0.1	127.0.0.1	TCP	52	49775 → 3333 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=8
7561	77.583992	127.0.0.1	127.0.0.1	TCP	44	3333 → 49775 [ACK] Seq=1 Ack=9 Win=65280 Len=0
7576	85.778167	127.0.0.1	127.0.0.1	TCP	58	49775 → 3333 [PSH, ACK] Seq=9 Ack=1 Win=65280 Len=14
7577	85.778225	127.0.0.1	127.0.0.1	TCP	44	3333 → 49775 [ACK] Seq=1 Ack=23 Win=65280 Len=0
8930	106.887942	127.0.0.1	127.0.0.1	TCP	44	49775 → 3333 [RST, ACK] Seq=23 Ack=1 Win=0 Len=0

1. Explain both the commands you used in detail. What did they actually do?
    a. nc -k -l 3333: Starts listening and keeps the connection open at the port 3333.
    b. ncat 127.0.0.1 3333: Sends data to local host at port 3333.

2. How many packets were send back and forth so the client/server could send/receive these two lines?
    a. 4 packets were used to send request to send and receive messages.
3. How many packets were needed back and forth to capture the whole "process" (starting the communication, ending the communication, sending the lines)?
    a. 8 total packets were needed for the entire process.
4. How many bytes is the data (only the data) that was sent from client to server?
    a. A total of 21 bytes were sent to the server from the client.
5. How many total bytes went over the wire (back and forth) for the whole process?
    a. A total of 353 bytes went back and forth during the process.
6. How much overhead was there? Basically how many bytes was the whole process compared to the actual data that we did send.
    a. 332 bytes is the overhead for the whole process.

Step 2



No.	Time	Source	Destination	Protocol	Length	Info
12337	127.450720	127.0.0.1	127.0.0.1	UDP	39	63533 → 3333 Len=7
13146	132.606973	127.0.0.1	127.0.0.1	UDP	39	63533 → 3333 Len=7

1. Explain both the commands you used in detail. What did they actually do?
    a. nc -k -l -u 3333: Starts listening and keeps the connection open at port 3333, and uses UDP instead of TCP.
    b. nc -u 127.0.0.1 3333: Sends data to the local host at port 3333 using UDP instead of TCP.
2. How many packets were sent back and forth so the client/server could send/receive these two lines?
    a. Only 2 packets were needed to send and receive the data between client and server.
3. How many packets were needed to capture the whole "process" (starting the communication, ending the communication)?
    a. Only 2 packets were needed to capture the whole process
4. How many bytes is the data (only the data) that was sent?
    a. 14 bytes of data was sent.
5. How many total bytes went over the wire? Basically how many bytes was the whole process compared to the actual data that we did send?
    a. 78 bytes went over the wire.
6. What is the difference in relative overhead between UDP and TCP and why? Specifically, what kind of information was exchanged in TCP that was not exchanged in UDP? Show the relative parts of the packet traces.
    a. The difference in overhead between TCP and UDP for my example is 268 bytes because it requires a lot more packets to send a message with TCP than UDP. Some of the packets sent from the client to the server are acknowledgement packets to let the server know the client wants to send a message and the server to send an acknowledgement packet to say it can receive a message.

Task 1.3

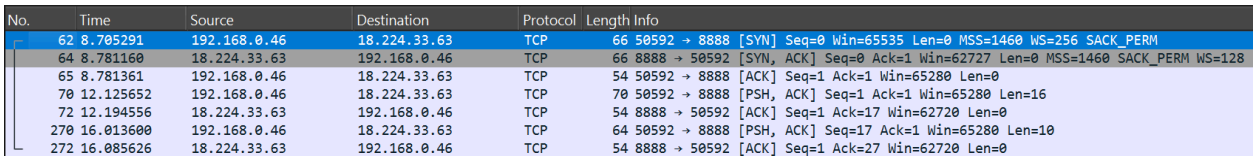
1.3.1

Video:

https://youtu.be/5UUYzpLVP38

Command Line:

1.3.2



No.	Time	Source	Destination	Protocol	Length	Info
62	8.705291	192.168.0.46	18.224.33.63	TCP	66	50592 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
64	8.781160	18.224.33.63	192.168.0.46	TCP	66	8888 → 50592 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=1460 SACK_PERM WS=128
65	8.781361	192.168.0.46	18.224.33.63	TCP	54	50592 → 8888 [ACK] Seq=1 Ack=1 Win=65280 Len=0
70	12.125652	192.168.0.46	18.224.33.63	TCP	70	50592 → 8888 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=16
72	12.194556	18.224.33.63	192.168.0.46	TCP	54	8888 → 50592 [ACK] Seq=1 Ack=17 Win=62720 Len=0
270	16.013600	192.168.0.46	18.224.33.63	TCP	64	50592 → 8888 [PSH, ACK] Seq=17 Ack=1 Win=65280 Len=10
272	16.085626	18.224.33.63	192.168.0.46	TCP	54	8888 → 50592 [ACK] Seq=1 Ack=27 Win=62720 Len=0

1. Command Line

2. The only changes I needed to make was the ip address for the client, instead of using the local ip address I used the public ip address from the AWS instance. I also used a different port, but I think 3333 can still be used if it's free. For wire shark the only change I made was to use WIFI and filter for a TCP port 8888.

1.3.3

1. I can not make a server locally, because of the issue of my local network having more security. I can not do it the same way from 1.3.2 because it is much more secure on my local network. It'll be difficult for the AWS client to connect to the server.

1.3.4

1. The router gives public ip addresses for multiple devices, and also assigns private ip addresses for each device for identification. When a request is sent in the router from AWS it's only given a public ip, but not a private ip that identifies the device AWS is trying to connect to. It's easier with a server on AWS because it has it's own public ip address that can make it easy to identify. One solution to connect from AWS to a local server is by port forwarding so AWS can easily access the server. The problem with making a server locally and accessing from the outside world is you have no way of accessing through the network because it won't have access to the private ip because of the security.