Github link:
https://github.com/alexjeffryallen/ser321-spring24-B-ajallen4.git

Linux System:

1: mkdir cli_assignment

2: cd cli_assignment

3: touch stuff.txt

4: cat > stuff.txt <<EOF
This is line 1.
This is line 2.
And this is line 3.
EOF

5: wc stuff.txt
Answer: 3 15 52 stuff.txt

6: cat <<EOF >> stuff.txt
This is additional text.
Appending more lines.
EOF

7. mkdir draft

8. mv stuff.txt draft/

9. cd draft
touch .secret.txt

10. cp -r draft final

11. mv draft draft.remove

12. mv draft.remove final/

13. cd cli_assignment
ls -l

14. wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Aug95.gz
zcat NASA_access_log_Aug95.gz

15. gunzip NASA_access_log_Aug95.gz

16. mv NASA_access_log_Aug95 logs.txt

17. mv logs.txt cli_assignment/

18. head -n 100 logs.txt

19. head -n 100 logs.txt > logs_top_100.txt

20. tail -n 100 logs.txt

21. tail -n 100 logs.txt > logs_bottom_100.txt

22. cat logs_top_100.txt logs_bottom_100.txt > logs_snapshot.txt

23. echo "ajallen4: This is a great assignment 03/12/2024" >> logs_snapshot.txt

24. less logs.txt

25. cut -d '%' -f 1 marks.csv | tail -n +2 > student_names.txt

26. cut -d '%' -f 4 marks.csv | sort

27. awk -F '%' '{ total += $3; count++ } END { if (count > 0) print total / count }' marks.csv

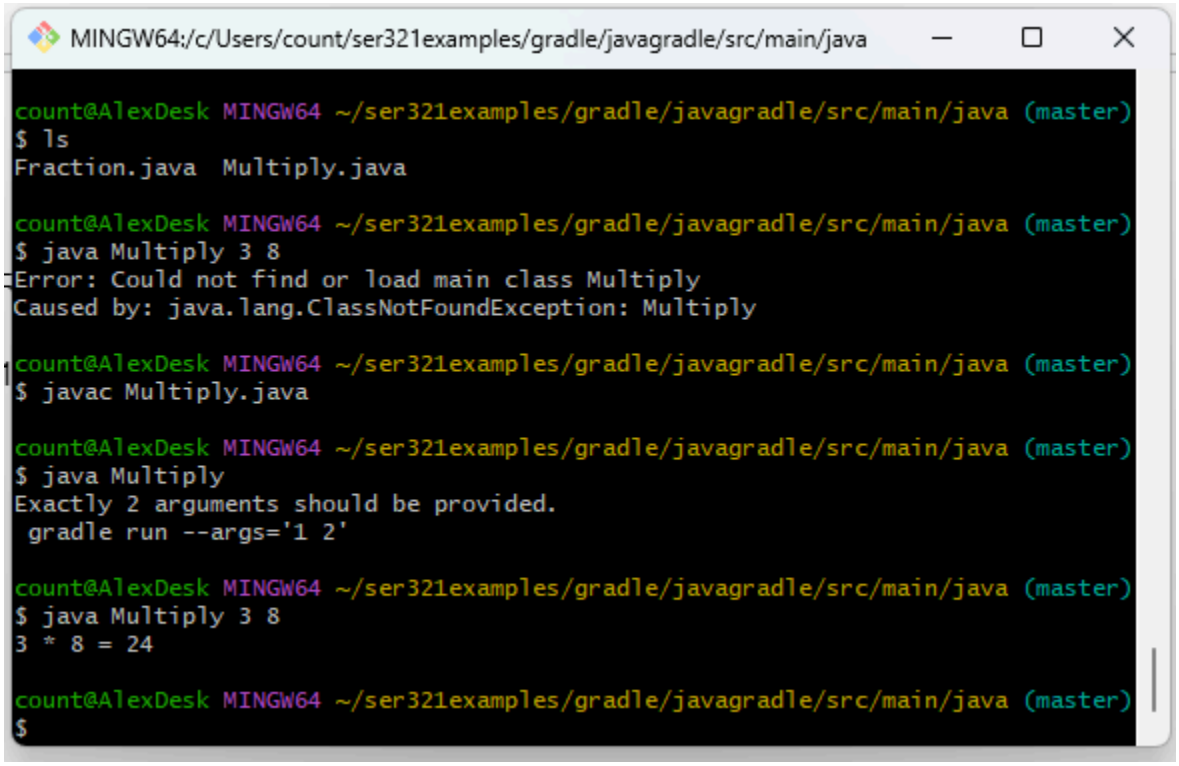
28. awk -F '%' '{ total += $3; count++ } END { if (count > 0) print total / count }' marks.csv > cli_assignment/done.txt

29. mv done.txt ../final/

30. mv done.txt average.txt

Running Examples:

1.
I ran Multiply.java



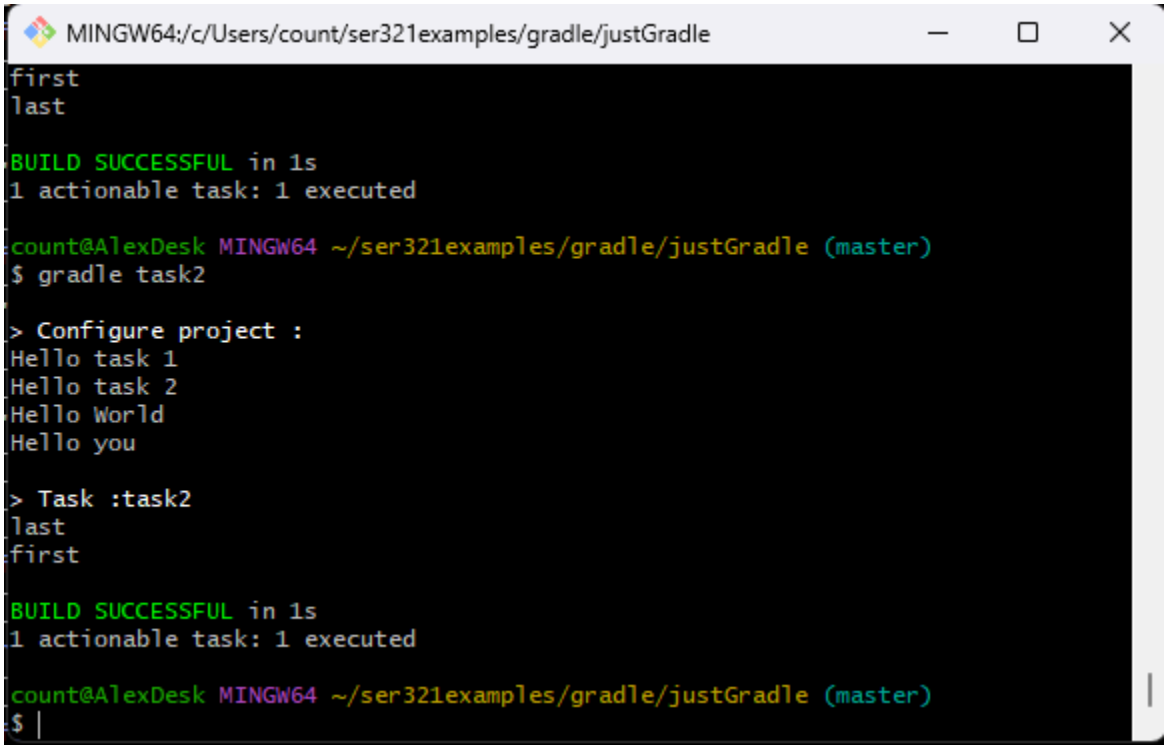
Explanation: This program takes two given arguments, such as 3 and 8 and multiplies them together, giving you the product, 24.

2.
I ran <gradle dependentComponents> in G-RPC.

```
extractTestProto - Extracts proto files/dependencies specified by 'protobuf' con
figuration
generateProto - Compiles Proto source for 'main'
generateTestProto - Compiles Proto source for 'test'
model - Displays the configuration model of project ':G-RPC'. [deprecated]
processResources - Processes main resources.
processTestResources - Processes test resources.
pythonProto
runClientJava - Run Client
runClientPython
runServerJava - Run Server
runServerPython

Rules
-----
Pattern: clean<TaskName>: Cleans the output files of a task.
Pattern: build<ConfigurationName>: Assembles the artifacts of a configuration.

Deprecated Gradle features were used in this build, making it incompatible with
Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and
 determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.4.2/userguide/command_line_interface.html#sec:comm
and_line_warnings

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed

count@AlexDesk MINGW64 ~/ser321examples/middleware/G-RPC (master)
$ gradle compileJava

Deprecated Gradle features were used in this build, making it incompatible with
Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and
 determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.4.2/userguide/command_line_interface.html#sec:comm
and_line_warnings

BUILD SUCCESSFUL in 1s
4 actionable tasks: 4 up-to-date

count@AlexDesk MINGW64 ~/ser321examples/middleware/G-RPC (master)
$ gradle dependentComponents

> Task :G-RPC:dependentComponents

------------------------------------------------------------
Project ':G-RPC' - gRPC Example
------------------------------------------------------------

No components.

Deprecated Gradle features were used in this build, making it incompatible with
Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and
 determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.4.2/userguide/command_line_interface.html#sec:comm
and_line_warnings

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
```

Explanation: I am not sure exactly what this task should do since it listed “no components” but, it should return a list of some sort of components that are dependent on something. I am pretty happy I have been able to get Gradle to seemingly be working though! 🙂

3.
I went into justGradle and ran <gradle task2>

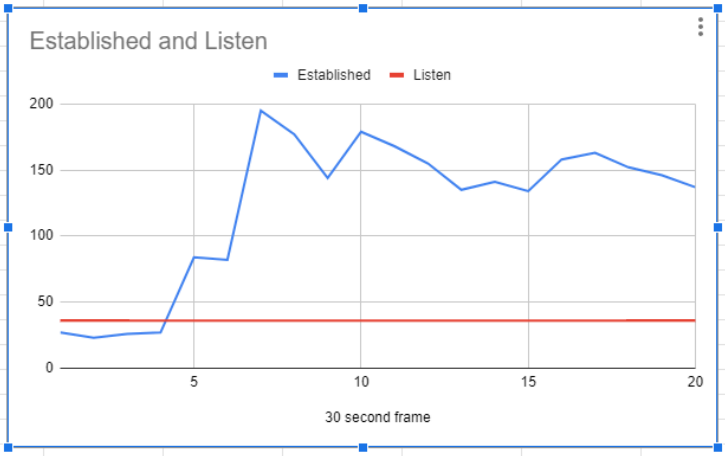


Explanation: This seems to just be a task made to test gradle and figure out using the command line. It could list more information but right now it only lists “last” and then “first” which is the opposite order from task1.

Socket Example video showing that I set up the server and client:

video.webm
https://drive.google.com/file/d/1DIP2I841Ipjj7apQRQqPrBgP80GYG7FQ/view?usp=sharing

	A	B	C
1	30 second frame	Established	Listen
2	1	27	36
3	2	23	36
4	3	26	36
5	4	27	36
6	5	84	36
7	6	82	36
8	7	195	36
9	8	177	36
10	9	144	36
11	10	179	36
12	11	168	36
13	12	155	36
14	13	135	36
15	14	141	36
16	15	134	36
17	16	158	36
18	17	163	36
19	18	152	36
20	19	146	36
21	20	137	36
22			




```
23:57:29 - Established: 27, Listen: 36
23:58:00 - Established: 23, Listen: 36
23:58:30 - Established: 26, Listen: 36
23:59:00 - Established: 27, Listen: 36
23:59:31 - Established: 84, Listen: 36
00:00:01 - Established: 82, Listen: 36
00:00:31 - Established: 195, Listen: 36
00:01:01 - Established: 177, Listen: 36
00:01:32 - Established: 144, Listen: 36
00:02:02 - Established: 179, Listen: 36
00:02:32 - Established: 168, Listen: 36
00:03:03 - Established: 155, Listen: 36
00:03:33 - Established: 135, Listen: 36
00:04:03 - Established: 141, Listen: 36
00:04:34 - Established: 134, Listen: 36
00:05:04 - Established: 134, Listen: 36
00:05:34 - Established: 158, Listen: 36
00:06:04 - Established: 163, Listen: 36
00:06:35 - Established: 152, Listen: 36
00:07:05 - Established: 146, Listen: 36
00:07:35 - Established: 137, Listen: 36
```

touch network_monitor.sh

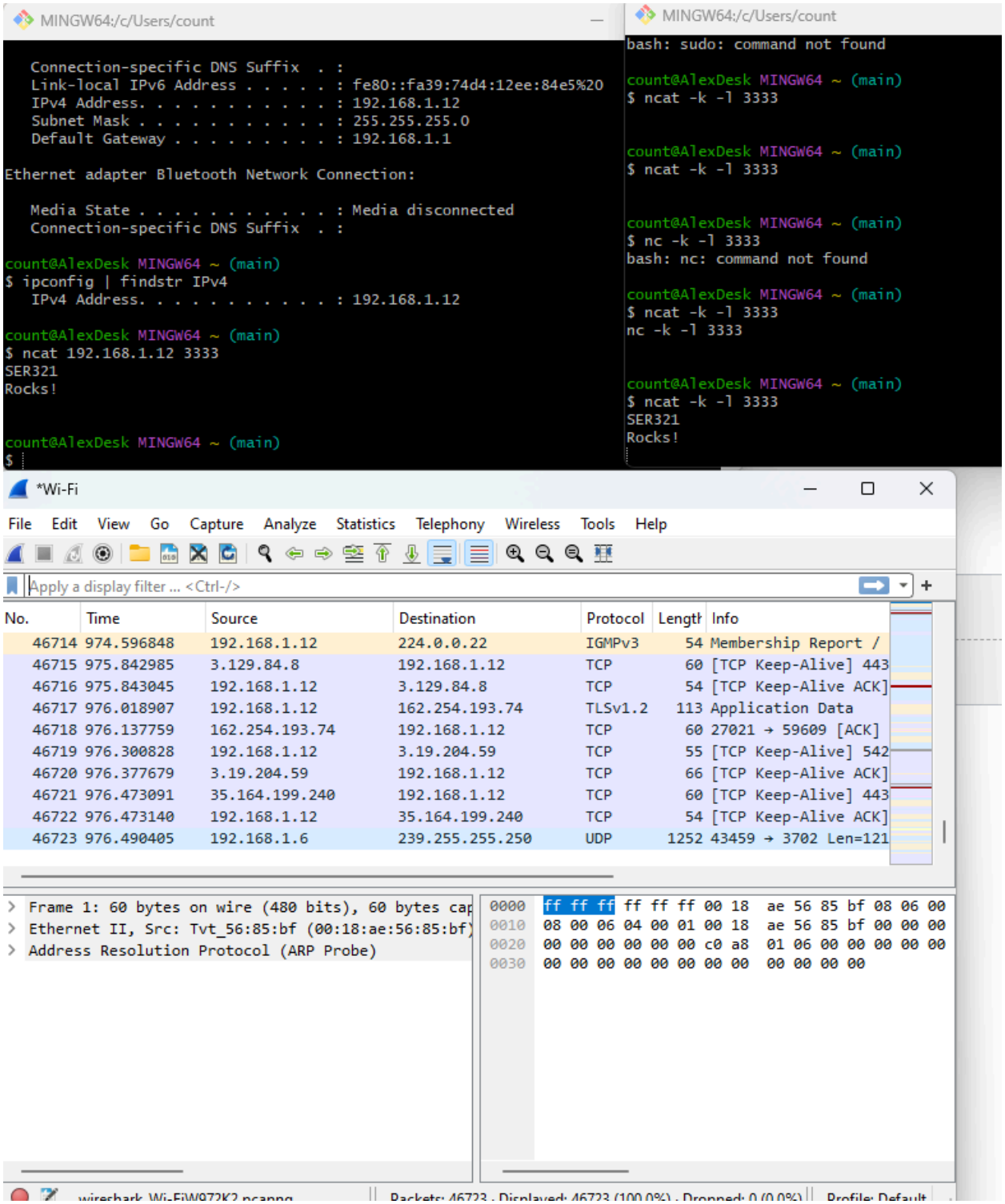
nano network_monitor.sh

```
#!/bin/bash

# Log file to store network activity data
LOG_FILE="network_activity.log"
```

```
# Create the log file if it doesn't exist
touch $LOG_FILE

while true; do
    echo "Time: $(date +%H:%M:%S)"

    # Count the number of sockets in ESTABLISHED state
    established_count=$(netstat -nat | grep 'ESTABLISHED' | wc -l)

    # Count the number of sockets in LISTEN state
    listen_count=$(netstat -nat | grep 'LISTEN' | wc -l)

    # Append the new data points to the log file
    echo "$(date +%H:%M:%S) - Established: $established_count, Listen:
$listen_count" >> $LOG_FILE

    # Display the current data points
    cat $LOG_FILE

    sleep 30
    clear
done
```

```
chmod +x network_monitor.sh

./network_monitor.sh
```

Task 3.2:


MINGW64:/c/Users/count
Connection-specific DNS Suffix . :
Link-local IPv6 Address . . . . . : fe80::fa39:74d4:12ee:84e5%20
IPv4 Address. . . . . : 192.168.1.12
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1
Ethernet adapter Bluetooth Network Connection:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
count@AlexDesk MINGW64 ~ (main)
$ ipconfig | findstr IPv4
IPv4 Address. . . . . : 192.168.1.12
count@AlexDesk MINGW64 ~ (main)
$ ncat 192.168.1.12 3333
SER321
Rocks!
count@AlexDesk MINGW64 ~ (main)
$
MINGW64:/c/Users/count
bash: sudo: command not found
count@AlexDesk MINGW64 ~ (main)
$ ncat -k -l 3333
count@AlexDesk MINGW64 ~ (main)
$ ncat -k -l 3333
count@AlexDesk MINGW64 ~ (main)
$ nc -k -l 3333
bash: nc: command not found
count@AlexDesk MINGW64 ~ (main)
$ ncat -k -l 3333
nc -k -l 3333
count@AlexDesk MINGW64 ~ (main)
$ ncat -k -l 3333
SER321
Rocks!
*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
46714 974.596848 192.168.1.12 224.0.0.22 IGMPv3 54 Membership Report /
46715 975.842985 3.129.84.8 192.168.1.12 TCP 60 [TCP Keep-Alive] 443
46716 975.843045 192.168.1.12 3.129.84.8 TCP 54 [TCP Keep-Alive ACK]
46717 976.018907 192.168.1.12 162.254.193.74 TLSv1.2 113 Application Data
46718 976.137759 162.254.193.74 192.168.1.12 TCP 60 27021 → 59609 [ACK]
46719 976.300828 192.168.1.12 3.19.204.59 TCP 55 [TCP Keep-Alive] 542
46720 976.377679 3.19.204.59 192.168.1.12 TCP 66 [TCP Keep-Alive ACK]
46721 976.473091 35.164.199.240 192.168.1.12 TCP 60 [TCP Keep-Alive] 443
46722 976.473140 192.168.1.12 35.164.199.240 TCP 54 [TCP Keep-Alive ACK]
46723 976.490405 192.168.1.6 239.255.255.250 UDP 1252 43459 → 3702 Len=121
Frame 1: 60 bytes on wire (480 bits), 60 bytes cap
Ethernet II, Src: Tvt_56:85:bf (00:18:ae:56:85:bf)
Address Resolution Protocol (ARP Probe)
0000 ff ff ff ff ff ff 00 18 ae 56 85 bf 08 06 00
0010 08 00 06 04 00 01 00 18 ae 56 85 bf 00 00 00
0020 00 00 00 00 00 00 c0 a8 01 06 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00

a) Explain both the commands you used in detail. What did they actually do?

The first command set up a connection to listen for information through a specific port (3333). The second command sent information to that specific IP address and port

.

b) How many frames were send back and forth to capture these 2 lines (Frames: 4 – I counted all frames that were sent)?

At the bottom of the wireshark page it says the summary is 46723 packets, I can’t find how many frames.

c) How many packets were send back and forth to capture only those 2 lines?

46723.

d) How many packets were needed to capture the whole "process" (starting the communication, ending the communication)?

46723.

e) How many bytes is the data (only the data) that was send?

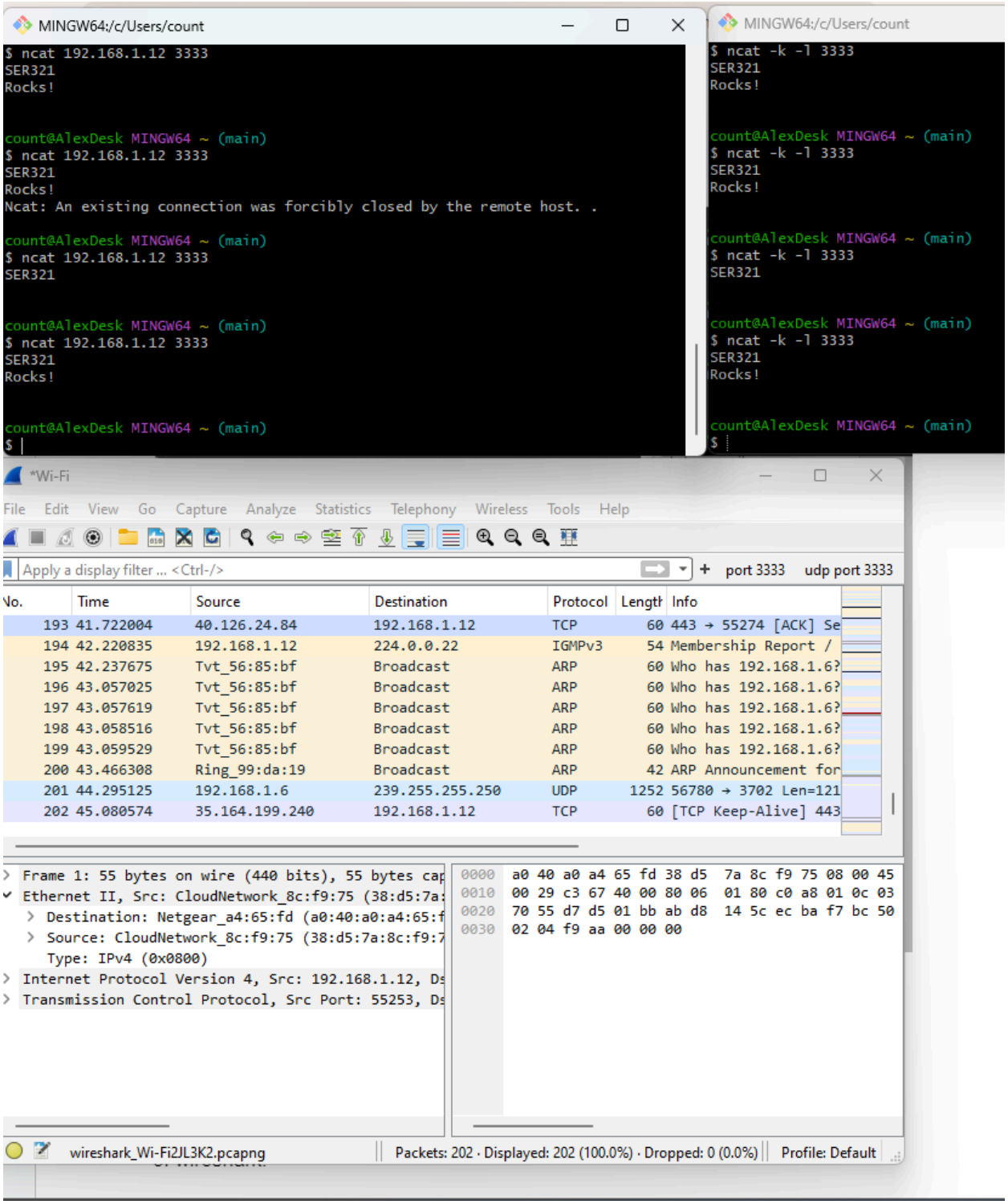
120.

f) How many total bytes went over the wire (back and forth) for the whole process?

60.

g) How much overhead was there. Basically how many bytes was the whole process compared to the actually data that we did send.

60.



a) Explain both the commands you used in detail. What did they actually do?

Same as before, set up a connection to listen on that port (3333) and then send information through that port so it would display on the first terminal.

b) How many frames were needed to capture those 2 lines?

I'm thinking maybe 1 because only 1 frame is listed on the bottom left section of wireshark.

c) How many packets were needed to capture those 2 lines?

The summary at the bottom says 202.

d) How many packets were needed to capture the whole "process" (starting the communication, ending the communication)?

202.

e) How many total bytes went over the wire?

55.

f) How many bytes is the data (only the data) that was send?

55.

g) Basically how many bytes was the whole process compared to the actually data that we did send.?

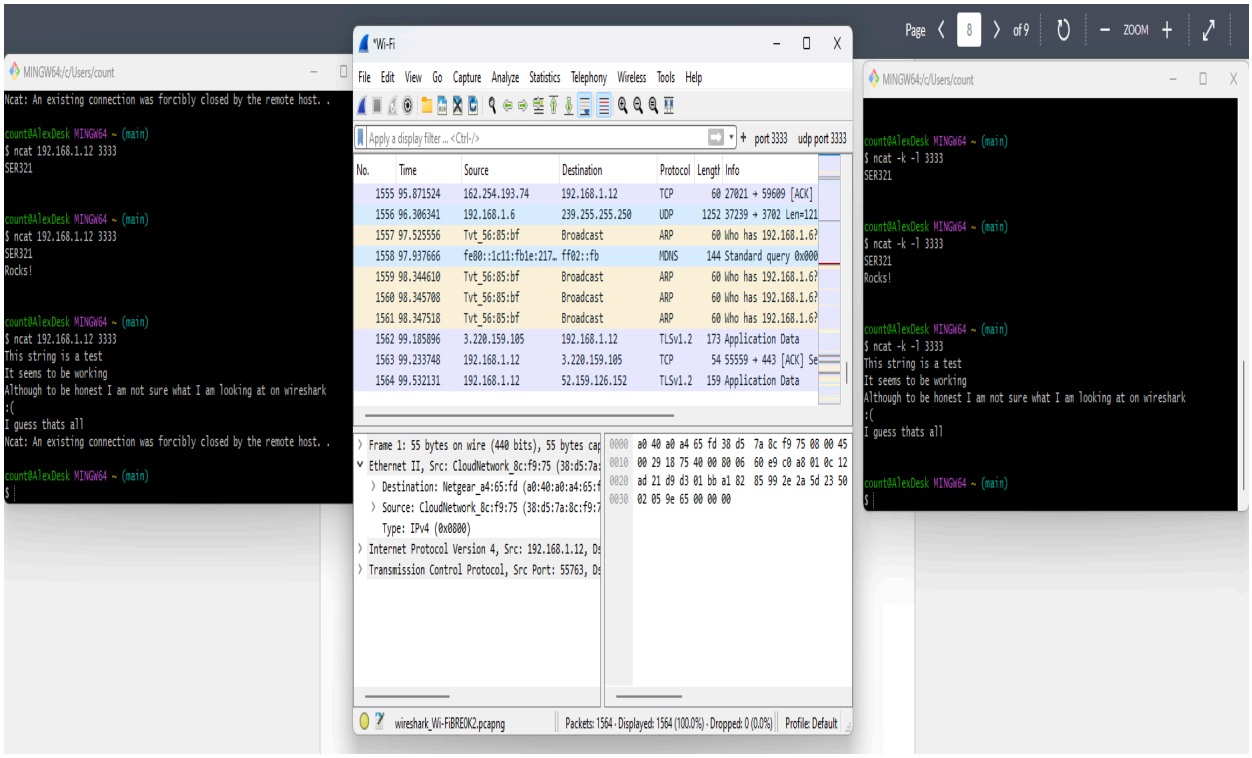
110.

h) 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

Seems like TCP uses a lot more overhead but I am not sure why.

3.3.1 video link:

https://drive.google.com/file/d/162Z7sOpy7GWe6ARvAze2ZD6jdmj1kas5/view?usp=sharing


MINGW64:/c/Users/count
Ncat: An existing connection was forcibly closed by the remote host. .
count@AlexDesk MINGW64 ~ (main)
$ ncat 192.168.1.12 3333
SER321
count@AlexDesk MINGW64 ~ (main)
$ ncat 192.168.1.12 3333
SER321
Rocks!
count@AlexDesk MINGW64 ~ (main)
$ ncat 192.168.1.12 3333
This string is a test
It seems to be working
Although to be honest I am not sure what I am looking at on wireshark
:(
I guess thats all
Ncat: An existing connection was forcibly closed by the remote host. .
count@AlexDesk MINGW64 ~ (main)
$
*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
port 3333 udp port 3333
No. Time Source Destination Protocol Length Info
1555 95.871524 162.254.193.74 192.168.1.12 TCP 60 27021 → 59609 [ACK]
1556 96.306341 192.168.1.6 239.255.255.250 UDP 1252 37239 → 3702 Len=121
1557 97.525556 Tvt_56:85:bf Broadcast ARP 60 Who has 192.168.1.6?
1558 97.937666 fe80::1c11:fb1e:217… ff02::fb MDNS 144 Standard query 0x000
1559 98.344610 Tvt_56:85:bf Broadcast ARP 60 Who has 192.168.1.6?
1560 98.345708 Tvt_56:85:bf Broadcast ARP 60 Who has 192.168.1.6?
1561 98.347518 Tvt_56:85:bf Broadcast ARP 60 Who has 192.168.1.6?
1562 99.185896 3.220.159.105 192.168.1.12 TLSv1.2 173 Application Data
1563 99.233748 192.168.1.12 3.220.159.105 TCP 54 55559 → 443 [ACK] Se
1564 99.532131 192.168.1.12 52.159.126.152 TLSv1.2 159 Application Data
Frame 1: 55 bytes on wire (440 bits), 55 bytes cap
Ethernet II, Src: CloudNetwork_8c:f9:75 (38:d5:7a:
Destination: Netgear_a4:65:fd (a0:40:a0:a4:65:f
Source: CloudNetwork_8c:f9:75 (38:d5:7a:8c:f9:7
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.1.12, Ds
Transmission Control Protocol, Src Port: 55763, Ds
0000 a0 40 a0 a4 65 fd 38 d5 7a 8c f9 75 08 00 45
0010 00 29 18 75 40 00 80 06 60 e9 c0 a8 01 0c 12
0020 ad 21 d9 d3 01 bb a1 82 85 99 2e 2a 5d 23 50
0030 02 05 9e 65 00 00 00
wireshark_Wi-FiBREOK2.pcapng
Packets: 1564 · Displayed: 1564 (100.0%) · Dropped: 0 (0.0%)
Profile: Default
Page 8 of 9
ZOOM
MINGW64:/c/Users/count
count@AlexDesk MINGW64 ~ (main)
$ ncat -k -l 3333
SER321
count@AlexDesk MINGW64 ~ (main)
$ ncat -k -l 3333
SER321
Rocks!
count@AlexDesk MINGW64 ~ (main)
$ ncat -k -l 3333
This string is a test
It seems to be working
Although to be honest I am not sure what I am looking at on wireshark
:(
I guess thats all
count@AlexDesk MINGW64 ~ (main)
$