

Matthew Lowber

SER 321

Assignment 1

<https://github.com/mlowber/ser321-spring25B-mlowber>

Part I

1) System: Windows 10 (using GitBash)

1. mkdir cli_assignment
2. cd cli_assignment
3. touch stuff.txt
4. cat > stuff.txt
5. wc stuff.txt
6. cat >> stuff.txt
7. mkdir draft
8. mv stuff.txt draft/
9. cd draft
 - touch .secret.txt (added '.' in front of file name to make it hidden on linux)
 - attrib +h .secret.txt (added 'hidden' attribute to file to make it hidden in windows)
10. cd ..
 - cp -r draft final
11. mv draft draft.remove
12. mv draft.remove final/
13. ls -l
14. zcat ../NASA_access_log_Aug95.gz | head
15. gunzip ../NASA_access_log_Aug95.gz
16. mv ../NASA_access_log_Aug95 logs.txt
17. 'logs.txt' already renamed into 'cli_assignment/' during step 16
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 "mlowber: This is a great assignment \$(date)" >> logs_snapshot.txt
24. less logs.txt
25. cut -d '%' -f 1 ../marks.csv | tail -n +2
26. cut -d '%' -f 4 ../marks.csv | tail -n +2 | sort -n
27. cut -d '%' -f 3 ../marks.csv | tail -n +2 | awk '{sum += \$1} END {print sum/NR}'
28. cut -d '%' -f 3 ../marks.csv | tail -n +2 | awk '{sum += \$1} END {print sum/NR}' > done.txt
29. mv done.txt final/
30. mv final/done.txt final/average.txt

2.1) <https://github.com/mlowber/ser321-spring25B-mlowber>

2.2)

JustGradle

```
Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session 8/SER
321/Assignment 1/ser321examples/Gradle/JustGradle (master)
$ gradle tasks

> Configure project :
Hello task 1
Hello task 2
Hello World
Hello you

> Task :tasks

-----
Tasks runnable from root project 'JustGradle'
-----

Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Help tasks
-----
buildEnvironment - Displays all buildscript dependencies declared in root projec
t 'JustGradle'.
dependencies - Displays all dependencies declared in root project 'JustGradle'.
dependencyInsight - Displays the insight into a specific dependency in root proj
ect 'JustGradle'.
help - Displays a help message.
javaToolchains - Displays the detected java toolchains.
outgoingVariants - Displays the outgoing variants of root project 'JustGradle'.
projects - Displays the sub-projects of root project 'JustGradle'.
properties - Displays the properties of root project 'JustGradle'.
resolvableConfigurations - Displays the configurations that can be resolved in r
oot project 'JustGradle'.
tasks - Displays the tasks runnable from root project 'JustGradle'.

Just a task tasks
-----
task1 - Shows how doFirst and doLast works
task2 - Shows how doFirst and doLast works -- reversed

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>

BUILD SUCCESSFUL in 837ms
1 actionable task: 1 executed

Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session 8/SER
321/Assignment 1/ser321examples/Gradle/JustGradle (master)
$
```

This example is a minimal gradle project with no java code. It's most likely used to explore how gradle works as a build system, define tasks manually, and learn gradle's DSL.

JavaSimpleSock2

```
MINGW64:/d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assig...
Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2 (master)
$ gradle SocketServer

> Task :SocketServer
Server ready for 3 connections
Server waiting for a connection
Received the String HI
Received the Integer 100
Server waiting for a connection
<===== 75% EXECUTING [45s]
> :SocketServer

MINGW64:/d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assig...
Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2 (master)
$ gradle SocketClient
Starting a Gradle Daemon, 1 busy Daemon could not be reused, use --status for details

> Task :SocketClient
Got it!

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/7.5/userguide/command_line_interface.html#sec:command_line_warnings

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

Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2 (master)
$
```

This example is of a simple client-server interaction using java sockets. The server listens on a port and handles communication from a client. The client connects to the server, sends a string and an int, and receives a response.

FirstThread

```
Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session 8/SER
321/Assignment 1/ser321examples/Threads/FirstThread (master)
$ gradle run

> Task :run
Hello from 0 loop=0
Hello from 2 loop=0
Hello from 4 loop=0
Hello from 3 loop=0
Hello from 1 loop=0
Hello from 0 loop=1
Hello from 0 loop=2
Hello from 0 loop=3
Hello from 0 loop=4
Hello from 1 loop=1
Hello from 2 loop=1
Hello from 1 loop=2
Hello from 3 loop=1
Hello from 1 loop=3
Hello from 4 loop=1
Hello from 2 loop=2
Hello from 1 loop=4
Hello from 3 loop=2
Hello from 2 loop=3
Hello from 4 loop=2
Hello from 2 loop=4
Hello from 3 loop=3
Hello from 4 loop=3
Hello from 3 loop=4
Hello from 4 loop=4

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/7.5/userguide/command\_line\_interface.html#sec:command\_line\_warnings

BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 executed

Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session 8/SER
321/Assignment 1/ser321examples/Threads/FirstThread (master)
$ |
```

This example is of basic multithreading in java. The program creates multiple threads which print to the console and shows how threads can run concurrently and interleave their outputs.

2.3) Folder uploaded to github directory

2.4) <https://youtu.be/-PtLxRaryWc>

3.1)

PowerShell script

monitor-netstat.ps1

\$interval = 30

\$duration = 600

\$iterations = \$duration / \$interval

\$outputFile = "netstat_log.csv"

"Time,Established,Listening" | Out-File -Encoding UTF8 \$outputFile

for (\$i = 0; \$i -lt \$iterations; \$i++) {

 \$time = Get-Date -Format "HH:mm:ss"

 \$netstat = netstat -an

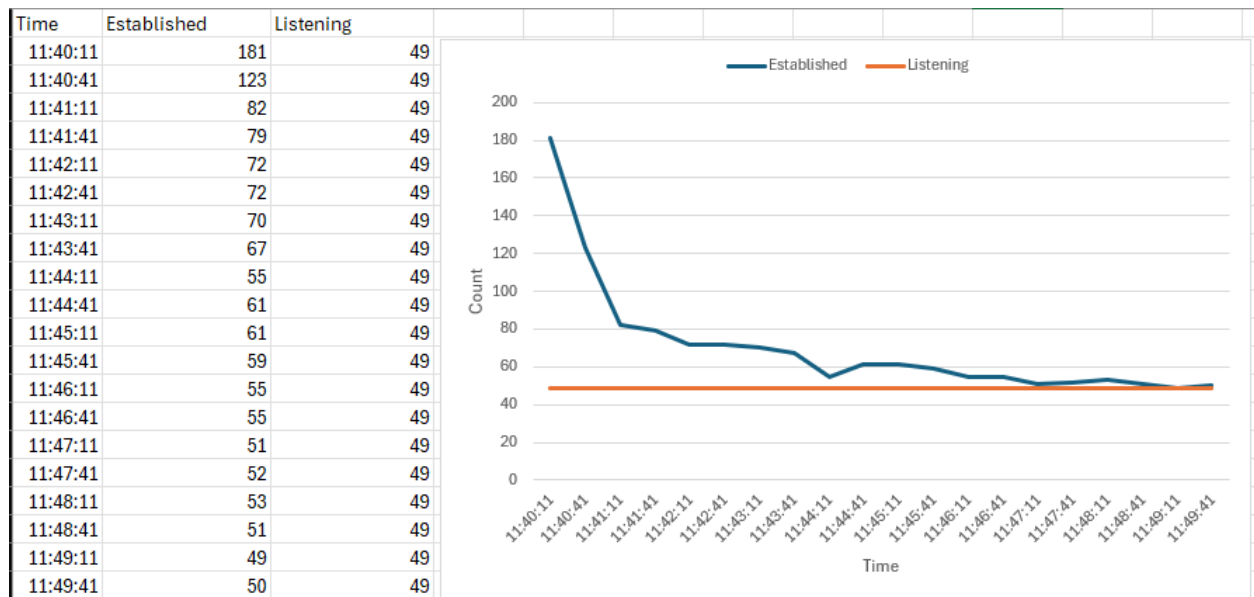
 \$established = (\$netstat | Select-String "ESTABLISHED").Count

 \$listening = (\$netstat | Select-String "LISTENING").Count

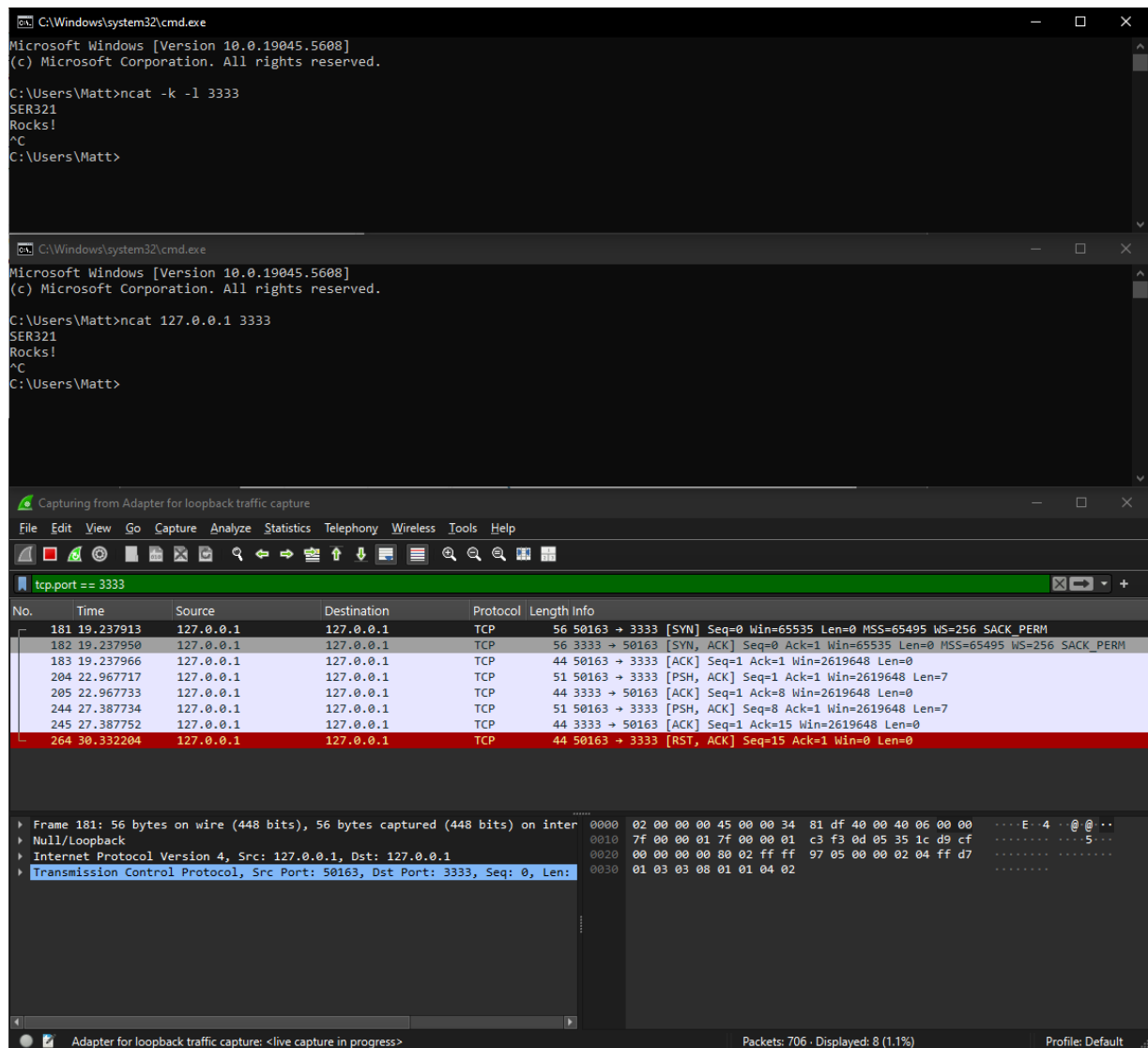
 "\$time,\$established,\$listening" | Out-File -Encoding UTF8 -Append \$outputFile

 Start-Sleep -Seconds \$interval

}

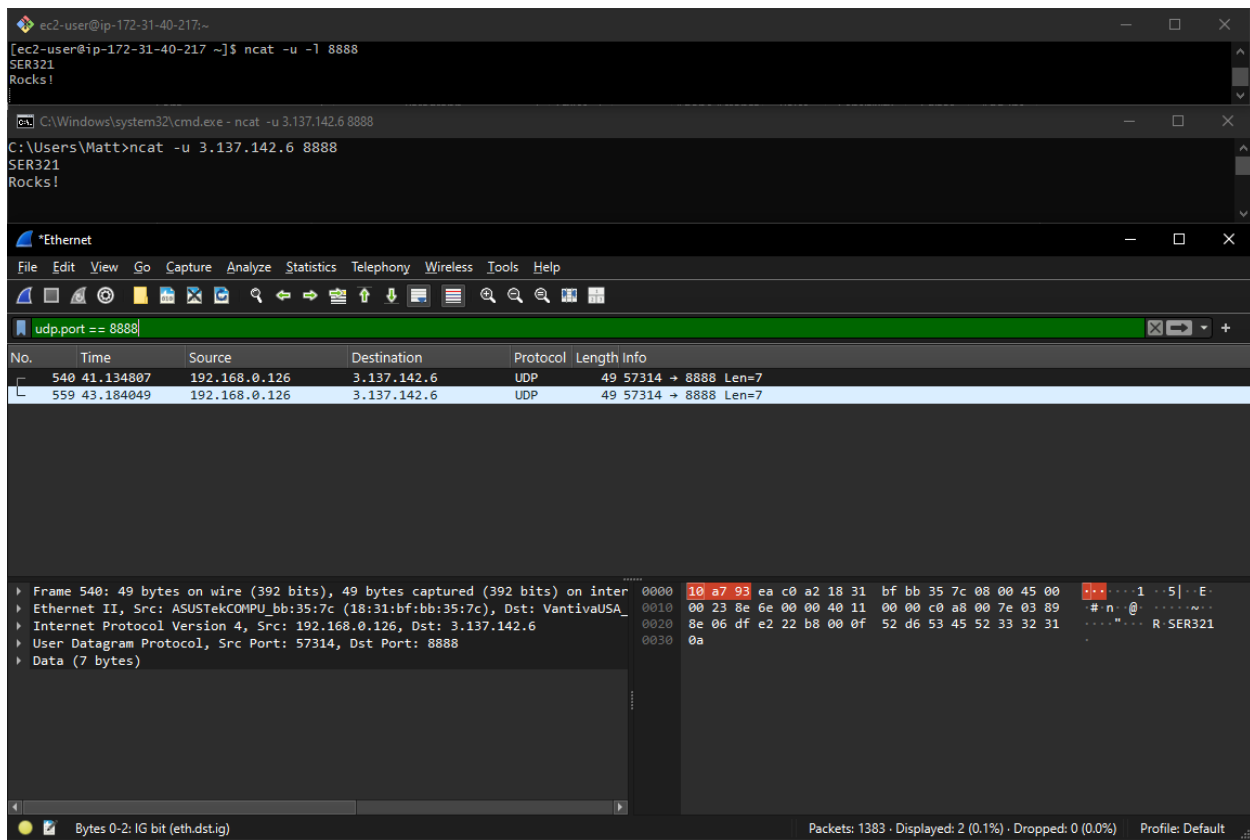


3.2) Step One (TCP)



- (terminal 1) `ncat -k -l 3333`
starts a TCP server listening on port 3333, `-k` keeps it running after a connection ends, and `-l` puts it in listen mode
(terminal 2) `ncat 127.0.0.1 3333`
connects to the TCP server at 127.0.0.1 (local loopback) on port 3333, and whatever is typed into the terminal after the connection is sent to the server.
- 4: 204, 205, 244, 245
- 4, same as above. Two data packets and 2 acks.
- 8
- 16 bytes (6 characters + 2 bytes for `\r\n` = 8 bytes each)
- $56+56+44+51+44+51+44+44 = 390$ bytes
- $390-16 = 374$ bytes (~95.9%) of overhead

3.2) Step 2 (UDP)



- a) (on EC2) `ncat -u -l 8888`
listens for incoming UDP messages on port 8888, -u enables UDP mode, -l makes it a listener, and prints any data received just like the TCP example
(on local) `ncat -u 3.137.142.6 8888`
connects to UDP server at the EC2 public IP address on port 8888, -u enables UDP mode, and sends the packets as before
- b) 2: 540, 559
- c) 2 packets
- d) UDP is connectionless, there is no handshake/teardown. 2 packets sent/captured.
- e) $49 + 49 = 98$ bytes
- f) 16 bytes, same as before
- g) $98 - 16 = 82$ bytes (~84%) of overhead
- h) UDP has lower overhead than TCP because it doesn't require a handshake (no syn/syn-ack/ack) and doesn't include acknowledgements or sequence numbers, but it also does not guarantee delivery or correct ordering. TCP sends extra packets for connection, acknowledgements, and teardown. As a comparison, for 16 bytes of data: TCP used 8 packets and 390 bytes, UDP used 2 packets and 98 bytes. Relative overhead was much higher in TCP due to additional headers and reliability features.

3.3.1) <https://youtu.be/E9KEU5R3Kig>

```
MINGW64:/d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2
Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2 (master)
$ gradle SocketServer

> Task :SocketServer
Server ready for 3 connections
Server waiting for a connection
Received the String Hello
Received the Integer 42
Server waiting for a connection
Received the String Wireshar
Received the Integer 99
Server waiting for a connection
<----- 75% EXECUTING [15m 8s]
> :SocketServer

MINGW64:/d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2
Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2 (master)
$ gradle SocketClient -PHost=127.0.0.1 -Pmessage="Hello" -Pnumber=42
Starting a Gradle Daemon, 1 busy and 1 stopped Daemons could not be reused, use --status for details

> Task :SocketClient
Got it!

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/7.5/userguide/command_line_interface.html#sec:command_line_warnings

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

Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2 (master)
$ gradle SocketClient -PHost=127.0.0.1 -Pmessage="Wireshar" -Pnumber=99

> Task :SocketClient
Got it!

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/7.5/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 895ms
2 actionable tasks: 1 executed, 1 up-to-date

Matt@DESKTOP-QIIQ3VK MINGW64 /d/Documents/school work/ASU/Fall 25/Session B/SER 321/Assignment 1/ser321examples/Sockets/JavaSimpleSock2 (master)
$
```


3.3.2)

The screenshot displays two windows. The top window is a terminal running on an EC2 instance, showing the execution of a Java socket server and client. The bottom window is Wireshark, capturing network traffic on the Ethernet adapter with a filter for 'tcp.port == 8888'.

Terminal Output:

```
ec2-user@ip-172-31-40-217:~/ser321examples/Sockets/JavaSimpleSock2
[ec2-user@ip-172-31-40-217 JavaSimpleSock2]$ gradle SocketServer
Starting a Gradle Daemon (subsequent builds will be faster)

> Task :SocketServer
Server ready for 3 connections
Server waiting for a connection
Received the String Hello
Received the Integer 42
Server waiting for a connection
<-----> 75% EXECUTING [3m 18s]
> :SocketServer
```

Wireshark Output:

Filter: tcp.port == 8888

No.	Time	Source	Destination	Protocol	Length	Info
798	40.093985	192.168.0.126	3.137.142.6	TCP	66	54674 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
801	40.166465	3.137.142.6	192.168.0.126	TCP	66	8888 → 54674 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=1460 SACK_PERM WS=128
802	40.166521	192.168.0.126	3.137.142.6	TCP	54	54674 → 8888 [ACK] Seq=1 Ack=1 Win=262656 Len=0
803	40.168269	192.168.0.126	3.137.142.6	TCP	58	54674 → 8888 [PSH, ACK] Seq=1 Ack=1 Win=262656 Len=4
805	40.231013	3.137.142.6	192.168.0.126	TCP	60	8888 → 54674 [ACK] Seq=1 Ack=5 Win=62848 Len=0
806	40.231013	3.137.142.6	192.168.0.126	TCP	60	[TCP Dup ACK 805#1] 8888 → 54674 [ACK] Seq=1 Ack=5 Win=62848 Len=0
807	40.231038	192.168.0.126	3.137.142.6	TCP	139	54674 → 8888 [PSH, ACK] Seq=5 Ack=1 Win=262656 Len=85
808	40.295399	3.137.142.6	192.168.0.126	TCP	60	8888 → 54674 [ACK] Seq=1 Ack=90 Win=62848 Len=0
809	40.314953	3.137.142.6	192.168.0.126	TCP	60	8888 → 54674 [PSH, ACK] Seq=1 Ack=90 Win=62848 Len=4
811	40.363754	192.168.0.126	3.137.142.6	TCP	54	54674 → 8888 [ACK] Seq=90 Ack=5 Win=262656 Len=0
814	40.429924	3.137.142.6	192.168.0.126	TCP	64	8888 → 54674 [PSH, ACK] Seq=5 Ack=90 Win=62848 Len=10
815	40.429791	192.168.0.126	3.137.142.6	TCP	54	54674 → 8888 [FIN, ACK] Seq=90 Ack=15 Win=262656 Len=0
816	40.534671	3.137.142.6	192.168.0.126	TCP	60	8888 → 54674 [ACK] Seq=15 Ack=91 Win=62848 Len=0

Detailed packet 806 information:

Frame 806: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF{...} Ethernet II, Src: VantivaUSA_ea:c0:a2:10:a7:93:ea:c0:a2, Dst: ASUSTekCOMPU_bb:35:7c:00:02:00:00 Internet Protocol Version 4, Src: 3.137.142.6, Dst: 192.168.0.126 Transmission Control Protocol, Src Port: 8888, Dst Port: 54674, Seq: 1, Ack: 5, Len: 60

Packet 806 Hex Dump:

```
0000 18 31 bf bb 35 7c 10 a7 93 ea c0 a2 08 00 45 00 .1.5[...E
0010 00 28 48 c9 40 00 fc 06 e3 50 03 89 8e 06 c0 a8 .(H@...P
0020 00 7e 22 b8 d5 92 27 66 64 a8 19 5f 01 57 50 10 ~"...f d...WP
0030 01 eb bc 24 00 00 00 00 00 00 00 00 00 00 00 00
```

To run the server on AWS, I ssh'd into my EC2 instance, navigated to JavaSimpleSock2, and ran: gradle SocketServer

On my local machine, I ran the client and connected to the EC2 server using: gradle SocketClient -Phost=[my EC2 public address] -Pmessage="Hello" -Pnumber=42

For wireshark, I switched from the loopback adapter to my ethernet adapter to capture the network traffic and applied the 'tcp.port = 8888' filter to isolate packets for this connection.

As for differences, instead of communicating over 127.0.0.1 or localhost, the connection is now over the internet connecting to a public IP, packet latency was slightly higher in wireshark, and port access had to be manually configured in AWS (unlike localhost). Otherwise, the client/server interaction remained the same.

3.3.3) Running the client on AWS and the server on my local machine would likely not work—at least not the same way as in 3.3.2. The main issue is that my local machine is behind a NAT which blocks unsolicited inbound traffic from the internet as a security feature. In 3.3.2 it worked because AWS servers are publicly addressable and I opened port 8888 in the security group. To make the reverse work, I would need to port forward 8888 on my home router to my local machine, probably configure a firewall exception, use my public IP address using `ipconfig`, and use that public IP in the AWS client's `-Phost=...` argument. The setup is more complex and less secure by default, which is why it would be typically avoided.

3.3.4) Mostly answered in 3.3.3 by coincidence. In a typical home network, my computer has a private IP address (192.168.x.x) assigned by my router. This address is only valid inside my local network and is not directly reachable via the internet. When I run a client locally to connect to an AWS server, the connection works easily because AWS assigns the server a public IP address, my router allows outgoing connections so traffic can leave the network and reach AWS, and the server responds to that connection and the router lets the response back in.

However, the reverse is not allowed by default. So if a client on AWS tries to connect to my local server, the router doesn't know which local device should handle the request and blocks the incoming connection for security. To make it work, I would need to set up port forwarding as explained in 3.3.3, possibly configure a static IP for my computer, adjust firewall settings to allow external traffic, and share my public IP with the AWS client—all of which is more complicated and introduces a lot of security risks, which is why cloud servers like AWS are often used instead.