Keana Gindlesperger – ksgindle
1/17/2023

Part 1. Command Line Tasks – Linux (using GitBash)

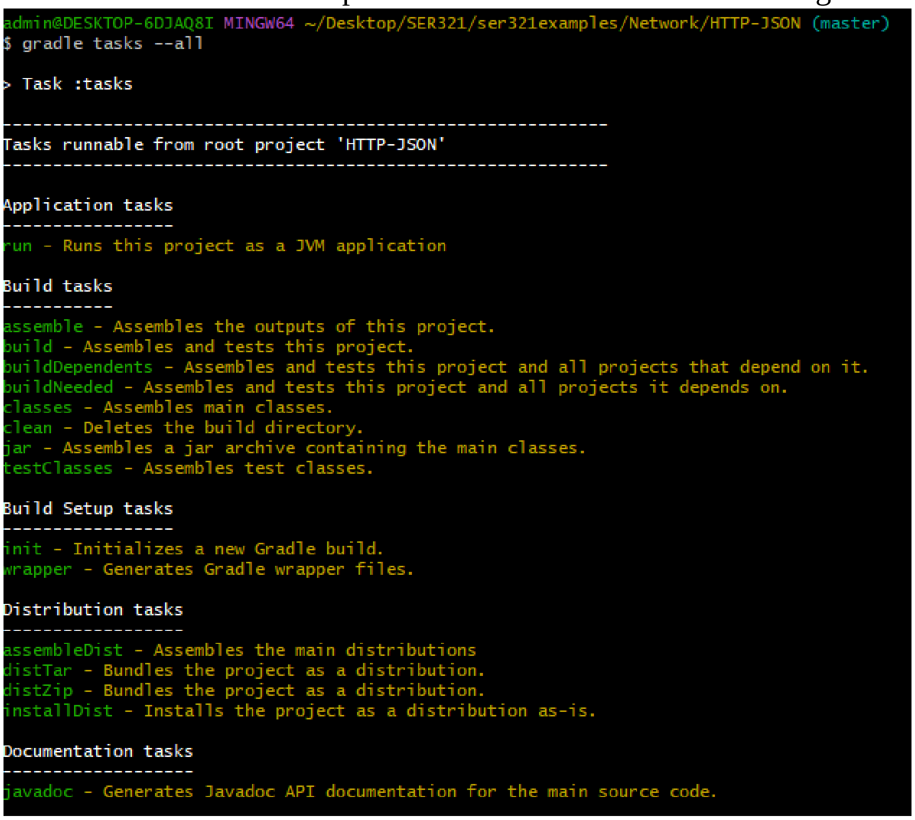
1. mkdir cli_assignment
2. cd cli_assignment
3. touch stuff.txt
4. cat > stuff.txt
   This is a file created for assignment 1.
   This has nothing of importance in it.
   CTRL + D to save these lines.
5. wc -l stuff.txt
   wc -w stuff.txt
6. echo ‘Appending text to the file’ >> stuff.txt
7. mkdir draft
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. ls -alR
14. zcat NASA_access_log_Aug95.gz
15. gzip -d NASA_access_log_Aug95.gz
16. mv NASA_access_log_Aug95 logs.txt
17. mv logs.txt /cli_assignment
18. head -100 logs.txt
19. head -100 logs.txt > logs_top_100.txt
20. tail -100 logs.txt
21. tail -100 logs.txt > logs_bottom_100.txt
22. cat logs_top_100.txt logs_bottom_100.txt > logs_snapshot.txt
23. echo ‘ksgindle: This is a great assignment 1/10/2023’ >> logs_snapshot.txt
24. less logs.txt
25. cut -d ‘%’ -f1 marks.csv | awk ‘NR!=1 {print}’
26. cut -d ‘%’ -f1 marks.csv | sort
27. awk -F’%’ ‘{ total += $3; n++} END { if (n >0) print total/ n; }’ marks.csv > done.txt
28. mv done.txt final/
29. mv done.txt average.txt

Keana Gindlesperger – ksgindle
1/17/2023

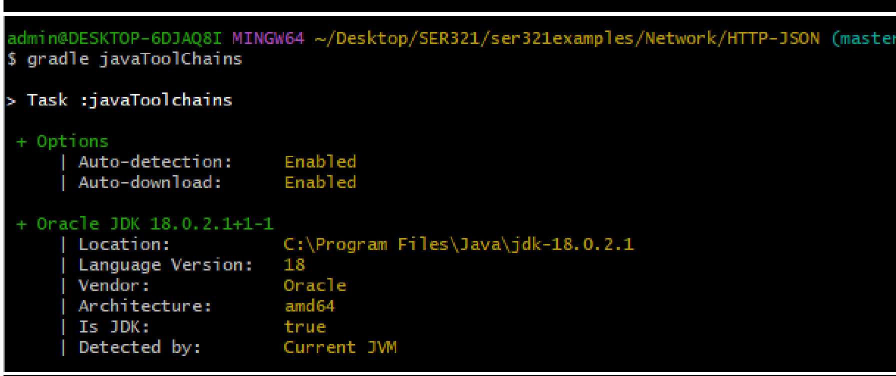
Part 2. Some Setup and Examples

2.2 Running Examples

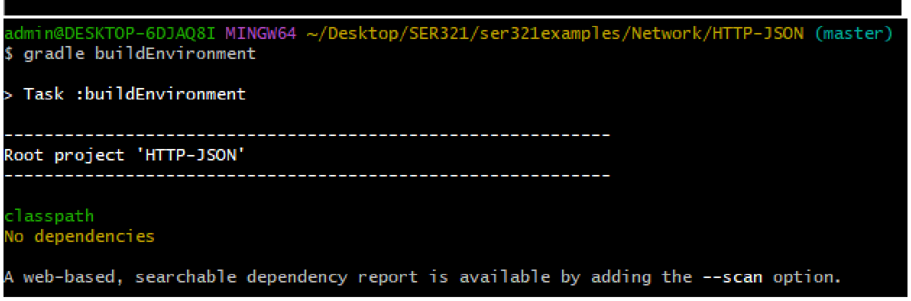
***1. Network/HTTP-JSON*:** ran the command gradle tasks -all and then chose a couple tasks presented to run. The tasks which were run are gradle javaToolChains and gradle buildEnvironment. From my understanding, this example showcases how to work with JSON files and find specific information about them through the command line.



```
admin@DESKTOP-6DJAQ8I MINGW64 ~/Desktop/SER321/ser321examples/Network/HTTP-JSON (master)
$ gradle tasks --all

> Task :tasks

------------------------------------------------------------
Tasks runnable from root project 'HTTP-JSON'
------------------------------------------------------------

Application tasks
-----------------
run - Runs this project as a JVM application

Build tasks
-----------
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles test classes.

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

Distribution tasks
------------------
assembleDist - Assembles the main distributions
distTar - Bundles the project as a distribution.
distZip - Bundles the project as a distribution.
installDist - Installs the project as a distribution as-is.

Documentation tasks
-------------------
javadoc - Generates Javadoc API documentation for the main source code.
```



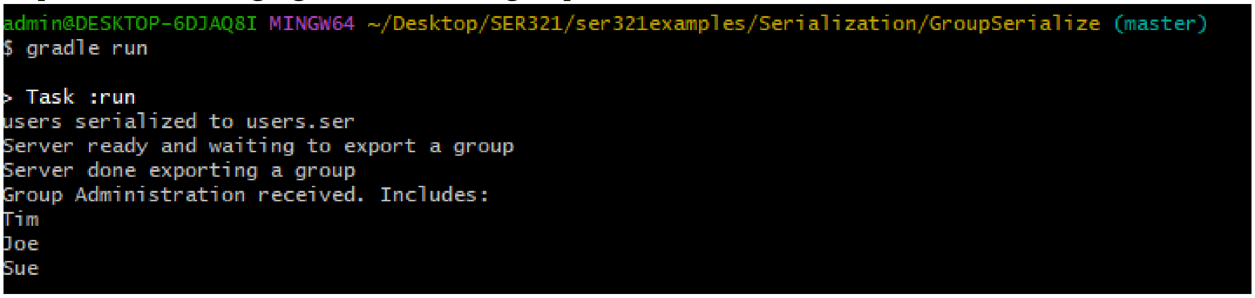
```
admin@DESKTOP-6DJAQ8I MINGW64 ~/Desktop/SER321/ser321examples/Network/HTTP-JSON (master
$ gradle javaToolChains

> Task :javaToolchains

 + Options
     | Auto-detection:     Enabled
     | Auto-download:      Enabled

 + Oracle JDK 18.0.2.1+1-1
     | Location:           C:\Program Files\Java\jdk-18.0.2.1
     | Language Version:   18
     | Vendor:             Oracle
     | Architecture:       amd64
     | Is JDK:             true
     | Detected by:        Current JVM
```



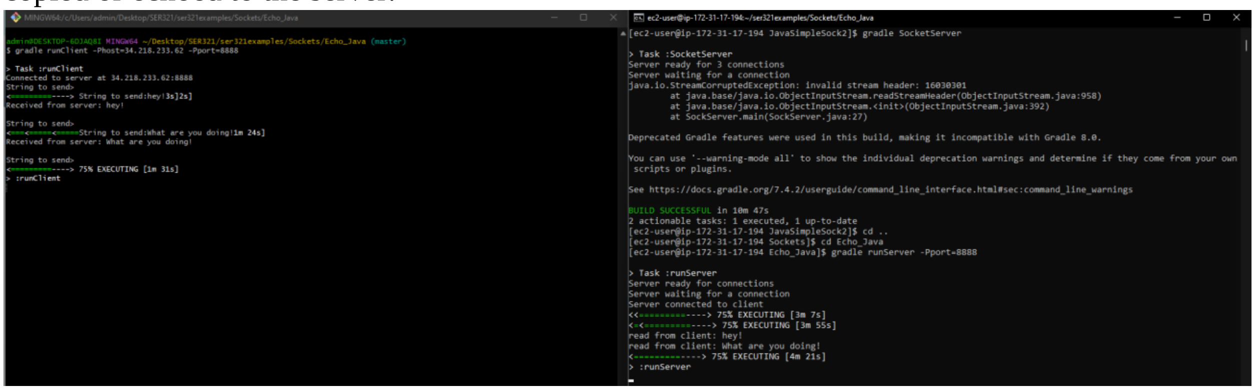
```
admin@DESKTOP-6DJAQ8I MINGW64 ~/Desktop/SER321/ser321examples/Network/HTTP-JSON (master)
$ gradle buildEnvironment

> Task :buildEnvironment

------------------------------------------------------------
Root project 'HTTP-JSON'
------------------------------------------------------------

classpath
No dependencies

A web-based, searchable dependency report is available by adding the --scan option.
```

Keana Gindlesperger – ksgindle
1/17/2023

***2. Serialization/GroupSerialize*:** ran the command gradle run. From doing so, the information for who has administration power was displayed. I would assume this is important for managing authentication groups as discussed in the README.



3. Sockets/Echo_Java: ran the commands gradle runServer-Pport=PORT and gradle runClient -Phost=HOST -Pport=PORT. The PORT and HOST were filled in accordingly as displayed in the screenshot. From what I ran, the commands appear to set up a connection between sockets to establish one as a client and the other as the server. The one established as a client can put anything in the command line that will then be copied or echoed to the server.



2.4 Set up your second system

Used AWS as second system.

Link to screencast: https://drive.google.com/file/d/1exO7K89fJgvuI92cyp4V-jl_sll22lIV/view?usp=sharing

Keana Gindlesperger – ksgindle
1/17/2023

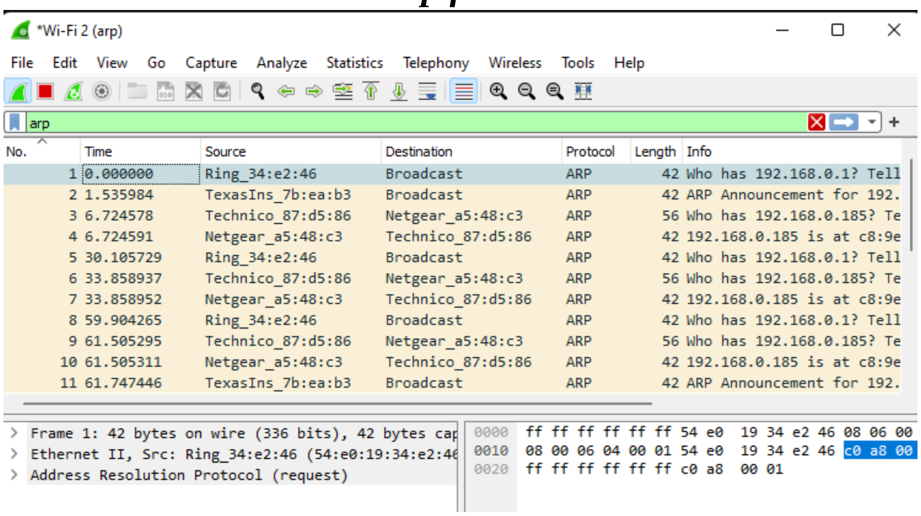
Part 3. Networking

3.1 Explore the Data Link Layer with ARP

Screen capture calls to identify network interface and gateway

```
admin@DESKTOP-6DJAQ8I MINGW64 ~
$ netstat -r
===========================================================================
Interface List
 23...18 c0 4d 95 47 e8 ......Realtek Gaming GbE Family Controller
 12...0a 00 27 00 00 0c ......VirtualBox Host-Only Ethernet Adapter
 13...ca 9e 43 a5 48 c3 ......Microsoft Wi-Fi Direct Virtual Adapter #2
  8...c8 9e 43 a5 48 c3 ......Microsoft Wi-Fi Direct Virtual Adapter #3
  7...c8 9e 43 a5 48 c3 ......NETGEAR A7000 WiFi USB3.0 Adapter
 20...98 8d 46 db dc ff ......Bluetooth Device (Personal Area Network)
 22...98 8d 46 db dc fb ......Intel(R) Dual Band Wireless-AC 3168
  1...........................Software Loopback Interface 1
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      192.168.0.1    192.168.0.211     65
          0.0.0.0          0.0.0.0      192.168.0.1    192.168.0.185     50
```

Wireshark instance with arp filter



Arp -a and arp -d commands

```
admin@DESKTOP-6DJAQ8I MINGW64 ~
$ arp -a

Interface: 192.168.0.185 --- 0x7
  Internet Address      Physical Address      Type
  192.168.0.1           94-6a-77-87-d5-86     dynamic
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static

Interface: 192.168.56.1 --- 0xc
  Internet Address      Physical Address      Type
  224.0.0.22            01-00-5e-00-00-16     static

Interface: 192.168.0.211 --- 0x16
  Internet Address      Physical Address      Type
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
```

```
admin@DESKTOP-6DJAQ8I MINGW64 ~
$ sudo arp -d 192.168.0.1 && arp -a

Interface: 192.168.0.185 --- 0x7
  Internet Address      Physical Address      Type
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static

Interface: 192.168.56.1 --- 0xc
  Internet Address      Physical Address      Type
  224.0.0.22            01-00-5e-00-00-16     static

Interface: 192.168.0.211 --- 0x16
  Internet Address      Physical Address      Type
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.22            01-00-5e-00-00-16     static
```

Keana Gindlesperger – ksgindle
1/17/2023

Updated trace in Wireshark

1337	11.524398	Technico_87:d5:86	Netgear_a5:48:c3	ARP	56	Who has 192.168.0.185? Tell
1338	11.524418	Netgear_a5:48:c3	Technico_87:d5:86	ARP	42	192.168.0.185 is at c8:9e:4
1765	13.426719	TexasIns_7b:ea:b3	Broadcast	ARP	42	ARP Announcement for 192.16
3537	25.407554	Ring_34:e2:46	Broadcast	ARP	42	Who has 192.168.0.1? Tell 1
8966	55.410902	Ring_34:e2:46	Broadcast	ARP	42	Who has 192.168.0.1? Tell 1

ARP request and reply

```
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Ring_34:e2:46 (54:e0:19:34:e2:46)
    Sender IP address: 192.168.0.66
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 192.168.0.1
```

```
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Netgear_a5:48:c3 (c8:9e:43:a5:48:c3)
    Sender IP address: 192.168.0.185
    Target MAC address: Technico_87:d5:86 (94:6a:77:87:d5:86)
    Target IP address: 192.168.0.1
```

1. What opcode is used to indicate a request? What about a reply?
   The opcode for a request is 1 and the opcode for a reply is 2.

2. How large is the ARP header for a request? What about for a reply? You will need to research this (hint: some sources define what belongs to the header differently, name which source you base your answer on)
   For IPv4, an ARP header for a request is 28 bytes. This stays the same for a reply. This information is found from practicalnetworking.net. Moreover, other sources like kevincurran.org and netometer.com also state the same information.

3. What value is carried on a request for the unknown target MAC address?
   The values carried on a request for an unknown target MAC address is the integer value -1 or the Hex value ffffffffffff

4. What Ethernet Type value indicates that ARP is the higher layer protocol?
   The Ethernet Type value of 0x806 indicated that ARP is the higher layer protocol.

3.2 Understanding TCP network sockets

<u>Command/Script used</u>

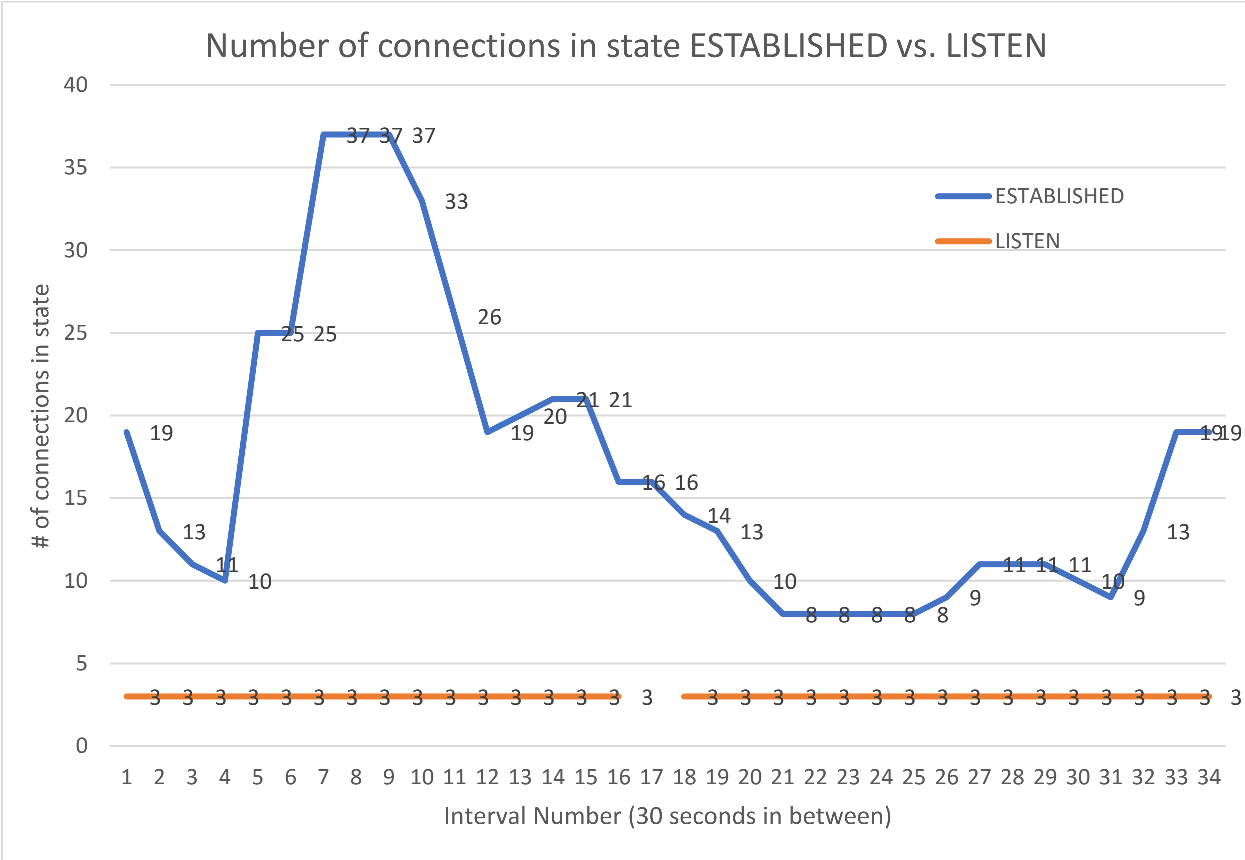
Terminal - keana@keana-VirtualBox: ~

File Edit View Terminal Tabs Help

```
keana@keana-VirtualBox:~$ while :; do $date; printf "%s\n""date"; netstat -at | grep 'ESTABLISHED\|LISTEN';
sleep 30; done >> finalOutput.txt
^C
```

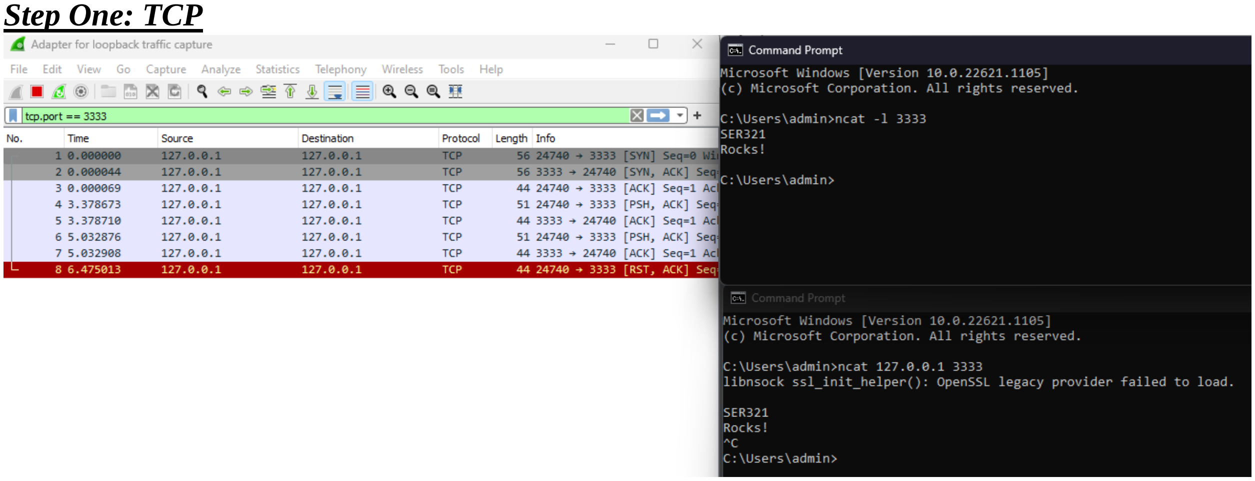
Keana Gindlesperger – ksgindle
1/17/2023

Graph of socket states over 12 minutes



3.3 Sniffing TCP/UDP traffic

Step One: TCP



a) Explain both the commands you used in detail. What did they actually do?
The two commands used together allow for communication on a network through a specific port. In these commands, we utilize TCP/IP to do so. The first command used ncat -l 3333 is a command to connect to/listen to a specific port number. In other words, it is used to read and write to network connections. In this case, we chose to listen to port number 3333. Then, we must open another terminal that will write and

Keana Gindlesperger – ksgindle
1/17/2023

listen to the same port. This occurs with the second command: ncat 127.0.0.1 3333. Essentially, we are connecting to the same port through the local host.

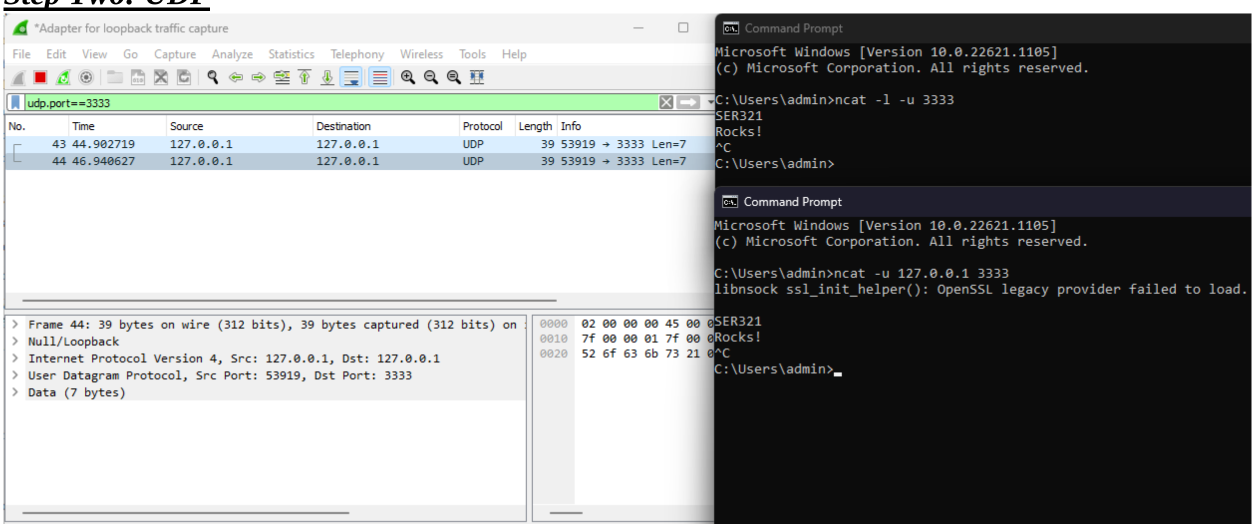
b) How many frames were send back and forth to capture these 2 lines (Frames: 4 – I counted all frames that were sent)?
5 frames were sent back and forth to capture these 2 lines.

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

d) How many packets were needed to capture the whole "process" (starting the communication, ending the communication)?
To capture the whole "process", 8 packets were needed.

e) How many bytes is the data (only the data) that was send?
The data was 14 bytes.

Step Two: UDP



a) Explain both the commands you used in detail. What did they do?
Both the commands are similar to the previous commands used to communicate on a network through a specific port; however, instead of utilizing TCP/IP, the commands utilize UDP. Moreover, the first command tells the terminal to listen to port 3333 using UDP and the second command also tells the terminal to write to port 3333 through UDP by providing the localhost address and the port of choice. Thus, the terminals can now communicate through port 3333 with the help of UDP.

Keana Gindlesperger – ksgindle
1/17/2023

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

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

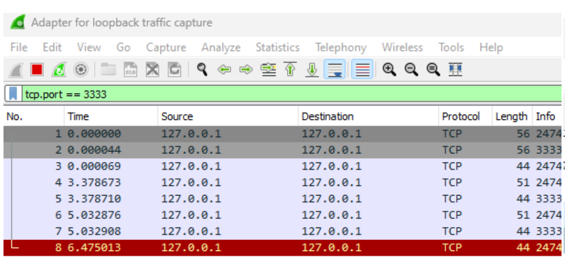
d) How many packets were needed to capture the whole "process" (starting the communication, ending the communication)?
   2 packets were needed to capture the whole “process”

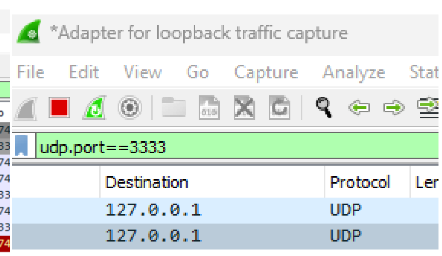
e) How many total bytes went over the wire?
   39 bytes went over the wire on each frame creating a total of 78 bytes.

f) How many bytes is the data (only the data) that was send?
   The data was 14 bytes

g) Basically, how many bytes was the whole process compared to the actual data that we did send?
   Compared to the actual data, the whole process was 78 bytes while the actual data was only 14 bytes.

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.
   In TCP, there is much more overhead. Specifically, this is because TCP/IP connects to the receiving computer/network directly while UDP sends out data and relies on devices in between to correctly deliver information. Additionally, the header for TCP connections is larger than UDP due to their connection-oriented protocol. This difference in overhead can be seen when evaluating the packet traces. The TCP/IP requires much more packets due to its direct connection while UDP skips the direct connection.





Keana Gindlesperger – ksgindle
1/17/2023

3.4 Internet Protocol (IP) Routing

Route 1 (Home Network from Desktop)

#	Country	Town	Lat	Lon	IP	Hostname	Latency (r	DNS Look	Distance
1	United St	Laguna Ni	33.5157	-117.711	2600:8802	(None)	3	89	0
2	United St	(Unknowi	37.751	-97.822	2600:8802	(None)	12	68	1856
3	United St	(Unknowi	37.751	-97.822	2001:578:	blackhole	12	17	0
4	United St	(Unknowi	37.751	-97.822	2001:578:	blackhole	13	69	0
5	United St	(Unknowi	37.751	-97.822	2001:578:	(None)	15	14	0
6	United St	(Unknowi	37.751	-97.822	2620:11a:	(None)	13	48	0
7	United St	(Unknowi	37.751	-97.822	2a04:4e42	(None)	16	15	0

```
C:\Users\admin>tracert www.asu.edu

Tracing route to pantheon-systems.map.fastly.net [2a04:4e42:6::645]
over a maximum of 30 hops:

  1     4 ms     5 ms     4 ms  2600:8802:3a07:5c00:966a:77ff:fe87:d586
  2    12 ms    12 ms    15 ms  2600:8802:3aff:ffff::1111
  3    13 ms    11 ms    15 ms  2001:578:400:7:f:0:1:800c
  4    12 ms    12 ms    13 ms  2001:578:400:4:4000::16
  5    15 ms    15 ms    14 ms  2001:578:1:0:172:17:249:32
  6    13 ms    13 ms    13 ms  2620:11a:c000:242:fa57::
  7    15 ms    15 ms    14 ms  2a04:4e42:6::645
```

Route 2 (Coffeeshop Network from Laptop)

```
C:\Users\keana>tracert www.asu.edu

Tracing route to pantheon-systems.map.fastly.net [2a04:4e42:6::645]
over a maximum of 30 hops:

  1    40 ms     3 ms     3 ms  2600:1012:b1c6:a954:1807:12cc:6ec2:b469
  2    51 ms    30 ms    28 ms  2600:1012:b1c6:a954:0:38:ab62:4d40
  3     *        *        *     Request timed out.
  4     *        *        *     Request timed out.
  5    42 ms    89 ms    34 ms  2001:4888:65:200e:62e:25:0:2
  6     *        *        *     Request timed out.
  7    56 ms    45 ms     *     2001:4888:6f:3092:62e:1:0:1
  8     *        *        *     Request timed out.
  9    62 ms    48 ms    45 ms  2001:506:0:1::72
 10     *        *        *     Request timed out.
 11     *        *        *     Request timed out.
 12    63 ms    42 ms    43 ms  2600:80a::75
 13    48 ms    61 ms    43 ms  2600:80a:85f::e
 14   442 ms    51 ms    46 ms  2a04:4e42:6::645

Trace complete.
```

a) Which is the fastest?
   The fastest network is the home network.
b) Which has the fewest hops?
   The network with fewest hops is the home network.

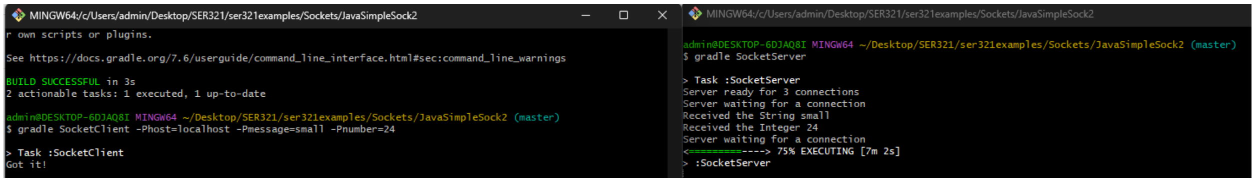
3.5 Running client servers in different ways

3.5.1 Running things locally

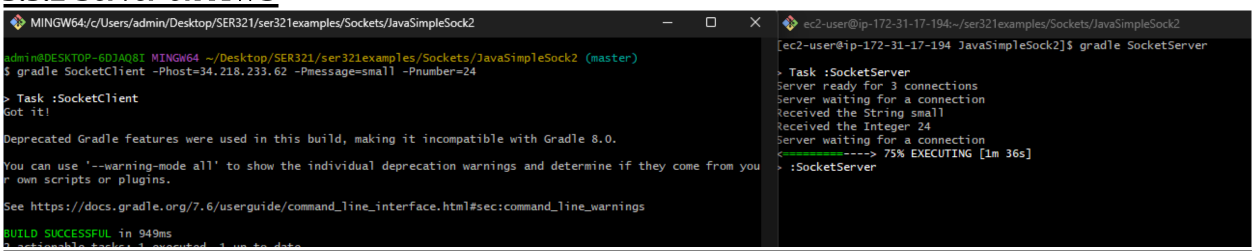
Link to recording:
https://drive.google.com/file/d/11G_l2MHfiPLfGlJ0f2K7mQyOKHPwOrgZ/view?usp=share_link

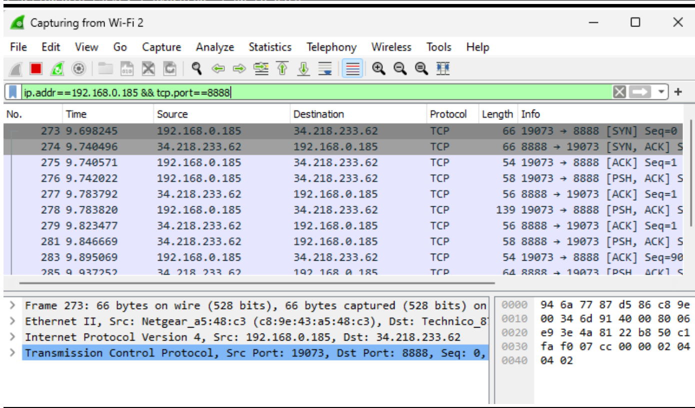
Screenshots of command line windows:

Keana Gindlesperger – ksgindle
1/17/2023



3.5.2 Server on AWS





To run the server from AWS and client locally and see the network traffic on Wireshark, a couple things had to changed. Firstly, I had to filter for the tcp port that they are communication through. Additionally, I used the Wi-Fi interface to view the packet control. Lastly, I added another filter of the ip address of the local gateway. Then, with the Gradle calls I only had to change the host address from localhost to the AWS host address.

3.5.3 Client on AWS

To run the client from AWS and the server locally and still see network traffic, issues may occur. Moreover, it cannot be done in the same way as in 3.5.2 because the user would have to determine what host address to connect to. When attempting to connect through the localhost IP address or gateway, the AWS client is unable to connect. Thus, a different connection must be made to have the client on AWS.

3.5.4 Client on AWS 2

It is harder to have a server running locally versus on AWS for a multitude of reasons. Moreover, a home router uses Network address translation to hide the subnet behind it. By doing so, multiple devices can communicate on the single global IP address. Therefore, when addressing the public IP from outside the network, you are not addressing the specific computer but rather the entire router. Thus, a rule in the router would need to be made that tells outside addresses to forward traffic to the specific computer/network through a specific port. This is called port forwarding.