

Command line tasks

System: Ubuntu 20.04

1. `mkdir cli_assignment`
2. `cd cli_assignment`
3. `touch stuff.txt`
4. `cat >> stuff.txt`
`line1`
`line2`
`line3`
5. `wc -w -l stuff.txt`
6. `cat >> stuff.txt`
`new line`
`another new line`
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 -Rla`
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 ~/School/SER321/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. `cat >> logs_snapshot.txt`
`caburdet: This is a great assignment 5/18/2021`
24. `less logs.txt`
25. `cut -d '%' -f 1 marks.csv | sed "1 d"`
26. `cut -d '%' -f 4 marks.csv | sort -r`
27. `cat marks.csv | awk -F '%' '{sum += $3} END {print sum/NR}'`
28. `cat marks.csv | awk -F '%' '{sum += $3} END {print sum/NR}' > done.txt`
29. `mv done.txt final`
30. `mv final/done.txt final/avearge.txt`

Some Setup and Examples

2.1. Setup a GitHub repo to submit your assignments

See github

2.2. Running examples

```
chris@chris-Z270X-UD3:~$ cd School/SER321/ser321examples
chris@chris-Z270X-UD3:~/School/SER321/ser321examples$ ls
gradle  Middleware  Network  README.md  Serialization  Sockets  Threads
chris@chris-Z270X-UD3:~/School/SER321/ser321examples$ cd Middleware
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Middleware$ ls
build.gradle  G-RPC  JMS  settings.gradle
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Middleware$ cd G-RPC
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Middleware/G-RPC$ ls
build  build.gradle  README.md  src
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Middleware/G-RPC$ gradle runServerJava
> Task :G-RPC:runServerJava
Server running ...
Received from client: Hello there Server
<=====--> 87% EXECUTING [2m 2s]
> :G-RPC:runServerJava
BUILD SUCCESSFUL in 4s
6 actionable tasks: 1 executed, 5 up-to-date
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Middleware/G-RPC$
```

```
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Threads/ThreadsShareData$ gradle run
> Task :run
Started thread #5
Started thread #2
Started thread #4
Started thread #1
Started thread #3
Shareable data with value 25 accessed by thread 5 count is 0
Shareable data with value 25 accessed by thread 3 count is 0
Shareable data with value 25 accessed by thread 4 count is 0
Shareable data with value 25 accessed by thread 3 count is 1
Shareable data with value 25 accessed by thread 2 count is 0
Shareable data with value 25 accessed by thread 1 count is 0
Shareable data with value 26 changed by thread 3 count is 2
Shareable data with value 26 accessed by thread 4 count is 1
Shareable data with value 26 accessed by thread 5 count is 1
Shareable data with value 27 changed by thread 4 count is 2
Shareable data with value 28 changed by thread 5 count is 2
Shareable data with value 28 accessed by thread 2 count is 1
Shareable data with value 28 accessed by thread 1 count is 1
Shareable data with value 29 changed by thread 1 count is 2
Shareable data with value 30 changed by thread 2 count is 2
BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 executed
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Threads/ThreadsShareData$
```

```
cdFirstThread: command not found
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Threads$ cd FirstThread
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Threads/FirstThread$ gradle run

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

BUILD SUCCESSFUL in 664ms
2 actionable tasks: 2 executed
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Threads/FirstThread$
```

2.3. Understanding Gradle

See Github

2.4. Set up your second system

Second System: AWS

Link: <https://youtu.be/301RtmcNGN0>

Network Traffic

3.1. Explore the Data Link Layer with ARP

Step 1: Capture a Trace

1.

```
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/sockets/JavaSimpleSock2$ ifconfig
enp0s31f6: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.50.70 netmask 255.255.255.0 broadcast 192.168.50.255
    inet6 fe80::6cb5:2c95:2189:ec94 prefixlen 64 scopeid 0x20<link>
    ether 1c:1b:0d:6f:bd:cb txqueuelen 1000 (Ethernet)
    RX packets 6907973 bytes 9613858906 (9.6 GB)
    RX errors 0 dropped 82 overruns 0 frame 0
    TX packets 2583110 bytes 285261163 (285.2 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 16 memory 0xef400000-ef420000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 88888 bytes 12075904 (12.0 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 88888 bytes 12075904 (12.0 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp4s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.50.191 netmask 255.255.255.0 broadcast 192.168.50.255
    inet6 fe80::5826:9f62:8b04:9a28 prefixlen 64 scopeid 0x20<link>
    ether e0:d4:e8:68:f8:df txqueuelen 1000 (Ethernet)
    RX packets 26555 bytes 3318249 (3.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 15761 bytes 2033158 (2.0 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

2.

```
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/sockets/JavaSimpleSock2$ route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default RT-AX3000-9820 0.0.0.0 UG 100 0 0 enp0s31f6
default RT-AX3000-9820 0.0.0.0 UG 600 0 0 wlp4s0
link-local 0.0.0.0 255.255.0.0 U 1000 0 0 enp0s31f6
192.168.50.0 0.0.0.0 255.255.255.0 U 100 0 0 enp0s31f6
192.168.50.0 0.0.0.0 255.255.255.0 U 600 0 0 wlp4s0
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/sockets/JavaSimpleSock2$
```

3.

Capturing from wlp4s0 (arp)

No.	Time	Source	Destination	Protocol	Length	Info
3	66.863011374	ASUSTekC_10:98:20	IntelCor_68:f8:df	ARP	42	Who has 192.168.50.191?
4	66.863033187	IntelCor_68:f8:df	ASUSTekC_10:98:20	ARP	42	192.168.50.191 is at e6
5	119.696542537	HuiZhouG_a9:8f:95	IntelCor_68:f8:df	ARP	42	Who has 192.168.50.191?
6	119.696563856	IntelCor_68:f8:df	HuiZhouG_a9:8f:95	ARP	42	192.168.50.191 is at e6
7	143.325065640	14:98:77:3c:12:b2	Broadcast	ARP	60	Who has 169.254.255.255?
8	182.367193363	TexasIns_7f:9b:b9	Broadcast	ARP	42	ARP Announcement for 19
9	200.769802354	14:98:77:3c:12:b2	Broadcast	ARP	60	ARP Announcement for 19

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface wlp4s0, id 0

4.

```
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Sockets/JavaSimpleSock2$ arp -a
LivingRoomTV (192.168.50.176) at 44:d8:78:a9:8f:95 [ether] on wlp4s0
RT-AX3000-9820 (192.168.50.1) at a8:5e:45:10:98:20 [ether] on wlp4s0
LivingRoomTV (192.168.50.176) at 44:d8:78:a9:8f:95 [ether] on enp0s31f6
RT-AX3000-9820 (192.168.50.1) at a8:5e:45:10:98:20 [ether] on enp0s31f6
```

```
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Sockets/JavaSimpleSock2$ sudo arp -d 192.168.50.1 && arp -a
[sudo] password for chris:
LivingRoomTV (192.168.50.176) at 44:d8:78:a9:8f:95 [ether] on wlp4s0
RT-AX3000-9820 (192.168.50.1) at <incomplete> on wlp4s0
LivingRoomTV (192.168.50.176) at 44:d8:78:a9:8f:95 [ether] on enp0s31f6
LivingRoomTV (192.168.50.176) at 44:d8:78:a9:8f:95 [ether] on enp0s31f6
```

5.

1	0.000000000	14:98:77:3c:12:b2	Broadcast	ARP	60	Who has 169.254.255.255? Tell 192.168.50.155
---	-------------	-------------------	-----------	-----	----	--

Step 2: Inspect the Trace

```
▼ Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: 14:98:77:3c:12:b2 (14:98:77:3c:12:b2)
  Sender IP address: 192.168.50.155
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 169.254.255.255

▼ Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: ASUSTekC_10:98:20 (a8:5e:45:10:98:20)
  Sender IP address: 192.168.50.1
  Target MAC address: IntelCor_68:f8:df (e0:d4:e8:68:f8:df)
  Target IP address: 192.168.50.191
```

Step 3: Details of ARP over Ethernet

What opcode is used to indicate a request?

1

What opcode is used to indicate a reply?

2

How large is the ARP header for a request?

28 bytes

How large is the ARP header for a reply?

28bytes

What value is carried on a request for the unknown target MAC address?

00:00:00_00:00:00

What Ethernet Type value indicates that ARP is the higher layer protocol?

0x806

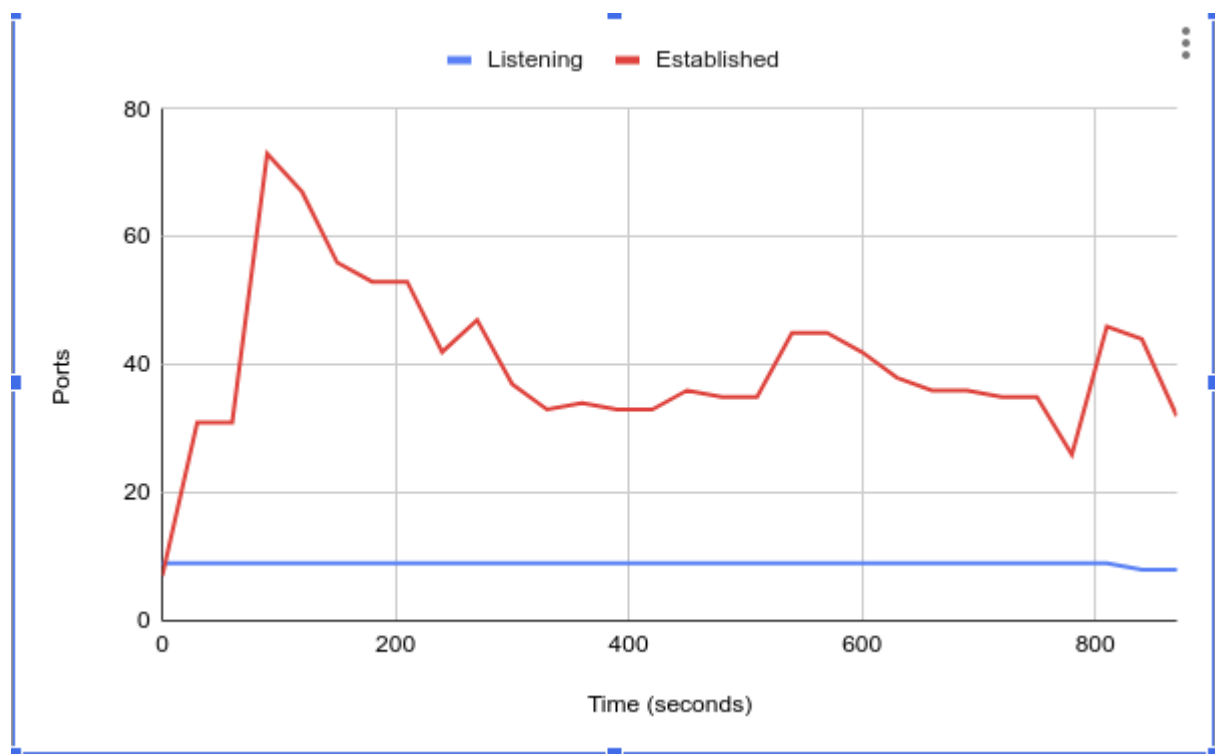
3.2 Understanding TCP Network Sockets

data:

```
LISTEN : 9
ESTABLISHED : 7
LISTEN : 9
ESTABLISHED : 31
LISTEN : 9
ESTABLISHED : 31
LISTEN : 9
ESTABLISHED : 73
LISTEN : 9
ESTABLISHED : 67
LISTEN : 9
ESTABLISHED : 56
LISTEN : 9
ESTABLISHED : 53
LISTEN : 9
ESTABLISHED : 53
LISTEN : 9
ESTABLISHED : 42
LISTEN : 9
ESTABLISHED : 47
LISTEN : 9
ESTABLISHED : 37
LISTEN : 9
ESTABLISHED : 33
LISTEN : 9
ESTABLISHED : 34
LISTEN : 9
ESTABLISHED : 33
LISTEN : 9
ESTABLISHED : 33
LISTEN : 9
ESTABLISHED : 36
LISTEN : 9
ESTABLISHED : 30
LISTEN : 9
ESTABLISHED : 36
LISTEN : 9
ESTABLISHED : 35
LISTEN : 9
ESTABLISHED : 35
LISTEN : 9
ESTABLISHED : 45
LISTEN : 9
ESTABLISHED : 45
```

LISTEN : 9
ESTABLISHED : 42
LISTEN : 9
ESTABLISHED : 38
LISTEN : 9
ESTABLISHED : 36
LISTEN : 9
ESTABLISHED : 36
LISTEN : 9
ESTABLISHED : 35
LISTEN : 9
ESTABLISHED : 35
LISTEN : 9
ESTABLISHED : 26
LISTEN : 9
ESTABLISHED : 46
LISTEN : 8
ESTABLISHED : 44
LISTEN : 8
ESTABLISHED : 32

graph:

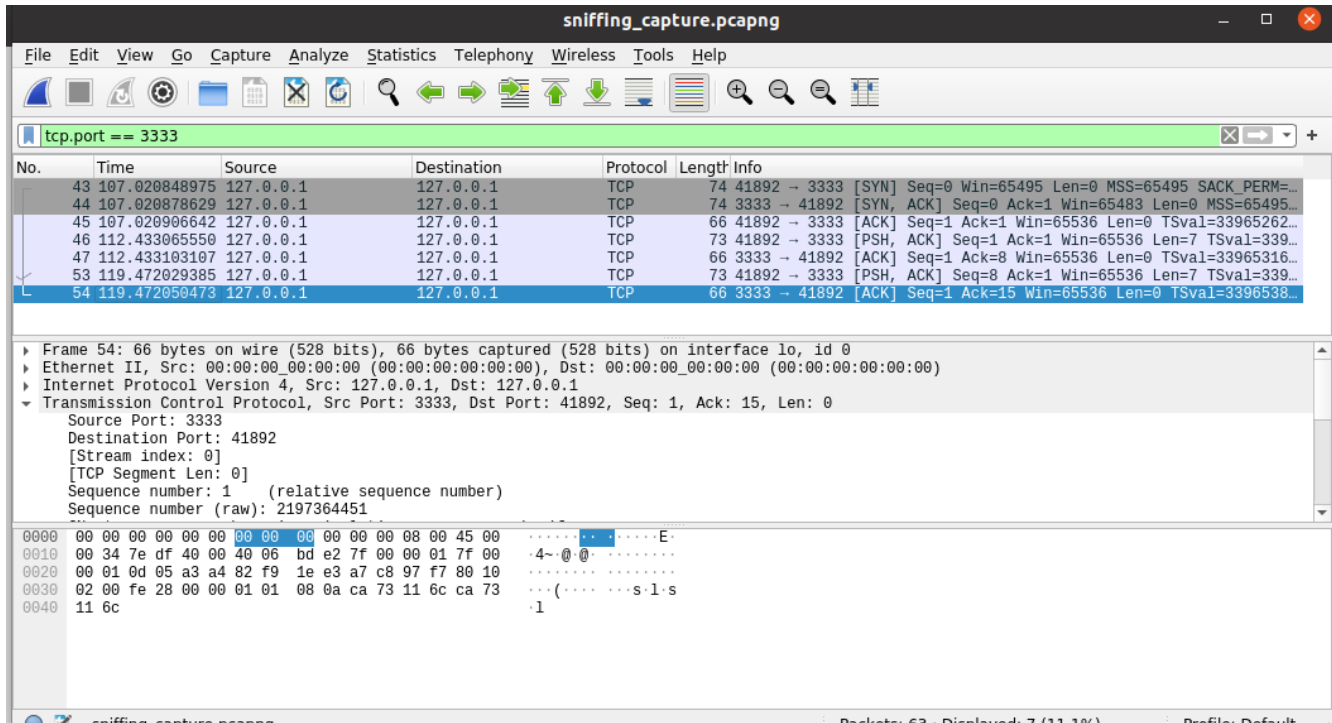


3.3 Sniffing TCP/UDP Traffic

Step 1: TCP

```
chris@chris-Z270X-UD3:~/School/SER321/ser321examples/Sockets/JavaSimpleSock2$ nc -k -l 3333
SER321
Rocks!
^C
```

```
chris@chris-Z270X-UD3:~$ nc 127.0.0.1 3333
SER321
Rocks!
^C
```



How many frames were needed to capture those 2 lines?

7

How many packets were needed to capture those 2 lines?

7

How many total bytes went over the wire? How much overhead was there

Total: 492 bytes

Data: 14 bytes

Overhead: 478 bytes

Step 2: UDP

```
chris@chris-Z270X-UD3:~/School/SER321$ nc -k -l -u 3333
^C
chris@chris-Z270X-UD3:~/School/SER321$
```

```
chris@chris-Z270X-UD3:~$ nc -u 127.0.0.1 3333
SER321
Rocks!
^C
```

The image shows a Wireshark packet capture window titled "*Loopback: lo". The filter bar shows "udp.port == 3333". The packet list shows two packets:

No.	Time	Source	Destination	Protocol	Length	Info
9	22.452505977	127.0.0.1	127.0.0.1	UDP	49	44008 → 3333 Len=7
10	24.748497259	127.0.0.1	127.0.0.1	UDP	49	44008 → 3333 Len=7

The packet details for the first packet (Frame 9) are:

- Frame 9: 49 bytes on wire (392 bits), 49 bytes captured (392 bits) on interface lo, id 0
- Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- User Datagram Protocol, Src Port: 44008, Dst Port: 3333
- Data (7 bytes)

The packet bytes are displayed in hexadecimal and ASCII:

```
0000  00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E
0010  00 23 85 75 40 00 40 11 b7 52 7f 00 00 01 7f 00  u@.0. R
0020  00 01 ab e8 0d 05 00 0f fe 22 53 45 52 33 32 31  ..... "SER321
0030  0a
```

How many frames were needed to capture those 2 lines?

2

How many packets were needed to capture those 2 lines?

2

How many total bytes went over the wire? How much overhead was there

Total: 98 bytes

Data: 14 bytes

Overhead: 84 bytes

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.

3.4 Internet Protocol (IP) Routing

```
chris@chris-Z270X-UD3:~/School/SER321$ traceroute www.asu.edu
traceroute to www.asu.edu.cdn.cloudflare.net (104.16.51.14), 64 hops max
 1  192.168.50.1  4.203ms  5.136ms  3.573ms
 2  10.96.56.1  11.533ms  10.532ms  10.636ms
 3  100.127.78.58  13.267ms  11.383ms  10.945ms
 4  100.120.100.50  12.381ms  14.122ms  11.460ms
 5  68.1.4.42  90.715ms  54.387ms  56.074ms
 6  141.101.73.251  58.270ms  56.215ms  56.489ms
 7  104.16.51.14  56.500ms  57.038ms  56.587ms
```

```
chris@chris-Z270X-UD3:~/School/SER321$ traceroute www.asu.edu
traceroute to www.asu.edu.cdn.cloudflare.net (104.16.51.14), 64 hops max
 1  192.168.66.80  5.482ms  4.609ms  7.120ms
 2  192.168.50.1  12.374ms  18.511ms  17.581ms
 3  10.96.56.1  30.497ms  97.202ms  129.007ms
 4  100.127.78.58  21.900ms  34.374ms  16.507ms
 5  100.120.100.50  40.228ms  32.935ms  28.890ms
 6  68.1.4.42  81.390ms  79.002ms  86.369ms
 7  141.101.73.251  65.626ms  75.750ms  227.191ms
 8  104.16.51.14  130.762ms  164.402ms  81.578ms
```

```
caburdet@general5:~$ ping -c 5 www.asu.edu
PING www.asu.edu.cdn.cloudflare.net (104.16.51.14) 56(84) bytes of data.
64 bytes from 104.16.51.14 (104.16.51.14): icmp_seq=1 ttl=34 time=9.26 ms
64 bytes from 104.16.51.14 (104.16.51.14): icmp_seq=2 ttl=34 time=9.78 ms
64 bytes from 104.16.51.14 (104.16.51.14): icmp_seq=3 ttl=34 time=9.28 ms
64 bytes from 104.16.51.14 (104.16.51.14): icmp_seq=4 ttl=34 time=9.31 ms
64 bytes from 104.16.51.14 (104.16.51.14): icmp_seq=5 ttl=31 time=9.67 ms

--- www.asu.edu.cdn.cloudflare.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 9.261/9.462/9.780/0.218 ms
```

Which is the fastest?

SSH

Which has the fewest hops?

SSH