Introduction

In this assignment, I developed two utilities, my_ping.py and my_traceroute.py, to mimic the functionality of the standard Linux ping and traceroute commands. These tools demonstrate how to create and send ICMP packets (for ping and one version of traceroute) and/or UDP packets and how to interpret the ICMP responses that come back from the target host or intermediate routers.

Folder contains:

1. my_ping.py
2. my_traceroute.py
3. requirements.txt
4. README.md
5. Report.txt
6. PDF documentation
7. Git log

Implementation Details

my_ping.py

Goal: Send ICMP Echo Request packets to a destination host and measure the round-trip time for each response, just like the standard ping command.

- Raw Socket Creation: We create a raw socket using socket.socket(AF_INET, SOCK_RAW, IPPROTO_ICMP). This requires elevated privileges.
- ICMP Packet Format: The script constructs an ICMP Echo Request (Type 8, Code 0) with a header and payload. A checksum is calculated over the entire packet.
- Command-Line Options:
    - -c, --count: Number of echo requests to send.
    - -i, --interval: Interval in seconds between sending packets.
    - -s, --size: Number of data bytes in each ICMP packet's payload (default 56).
    - -t, --timeout: A total timeout (in seconds) after which my_ping.py stops.

```
C:\Users\hiron\OneDrive\Desktop\UNI_HW\Brian_Tokumoto_hw2>python my_ping.py 8.8.8.8 -c 4
PING 8.8.8.8 (8.8.8.8) 56(64) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 time=55.657 ms
64 bytes from 8.8.8.8: icmp_seq=2 time=91.235 ms
64 bytes from 8.8.8.8: icmp_seq=3 time=18.937 ms
64 bytes from 8.8.8.8: icmp_seq=4 time=45.769 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss
rtt min/avg/max/mdev = 18.937/52.900/91.235/25.892 ms
```

Ping with -c:
Sends exactly 4 ICMP Echo Requests, then stops.

```
C:\Users\hiron\OneDrive\Desktop\UNI_HW\Brian_Tokumoto_hw2>python my_ping.py google.com -c 4 -i 0.5 -s 64
PING google.com (142.250.191.110) 64(72) bytes of data.
72 bytes from 142.250.191.110: icmp_seq=1 time=26.916 ms
72 bytes from 142.250.191.110: icmp_seq=2 time=113.991 ms
72 bytes from 142.250.191.110: icmp_seq=3 time=26.396 ms
72 bytes from 142.250.191.110: icmp_seq=4 time=25.396 ms

--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss
rtt min/avg/max/mdev = 25.396/48.175/113.991/38.003 ms
```

Ping with -i (interval) and -s (size)

Sends 4 packets at half-second intervals, each with 64 bytes of data.

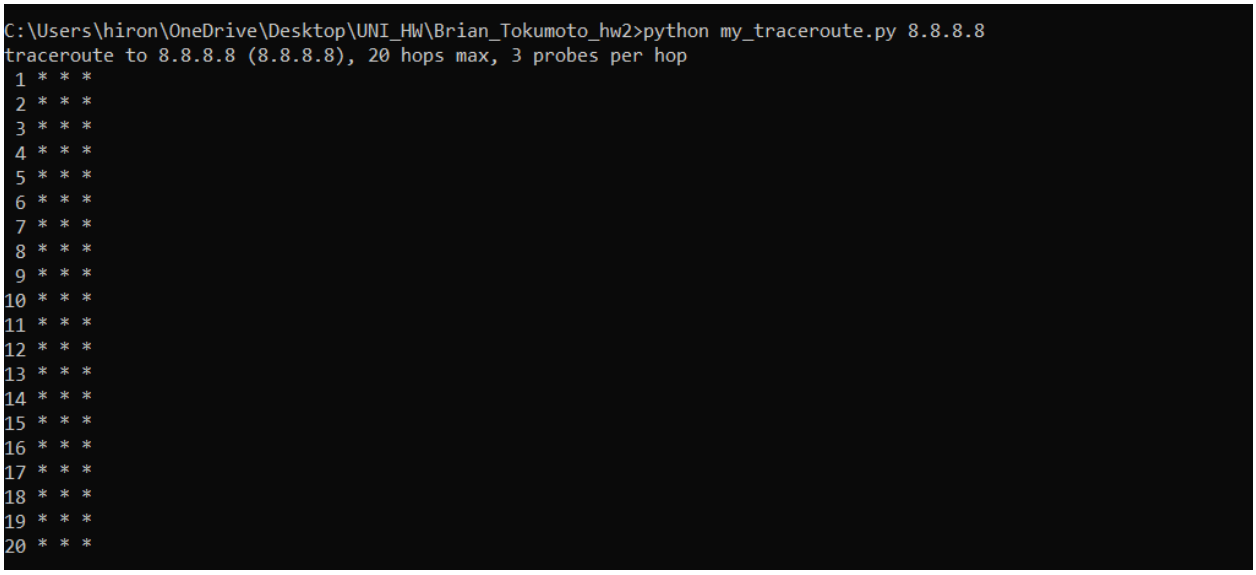
```
C:\Users\hiron\OneDrive\Desktop\UNI_HW\Brian_Tokumoto_hw2>python my_ping.py 8.8.4.4 -t 5
PING 8.8.4.4 (8.8.4.4) 56(64) bytes of data.
64 bytes from 8.8.4.4: icmp_seq=1 time=45.975 ms
64 bytes from 8.8.4.4: icmp_seq=2 time=32.458 ms
64 bytes from 8.8.4.4: icmp_seq=3 time=46.588 ms
64 bytes from 8.8.4.4: icmp_seq=4 time=23.174 ms
64 bytes from 8.8.4.4: icmp_seq=5 time=24.070 ms

--- 8.8.4.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss
rtt min/avg/max/mdev = 23.174/34.453/46.588/10.189 ms
```

Ping with Overall -t (timeout)
Exits after 5 seconds regardless of how many replies are received. This time it took way less than 5 seconds so the exit never happened

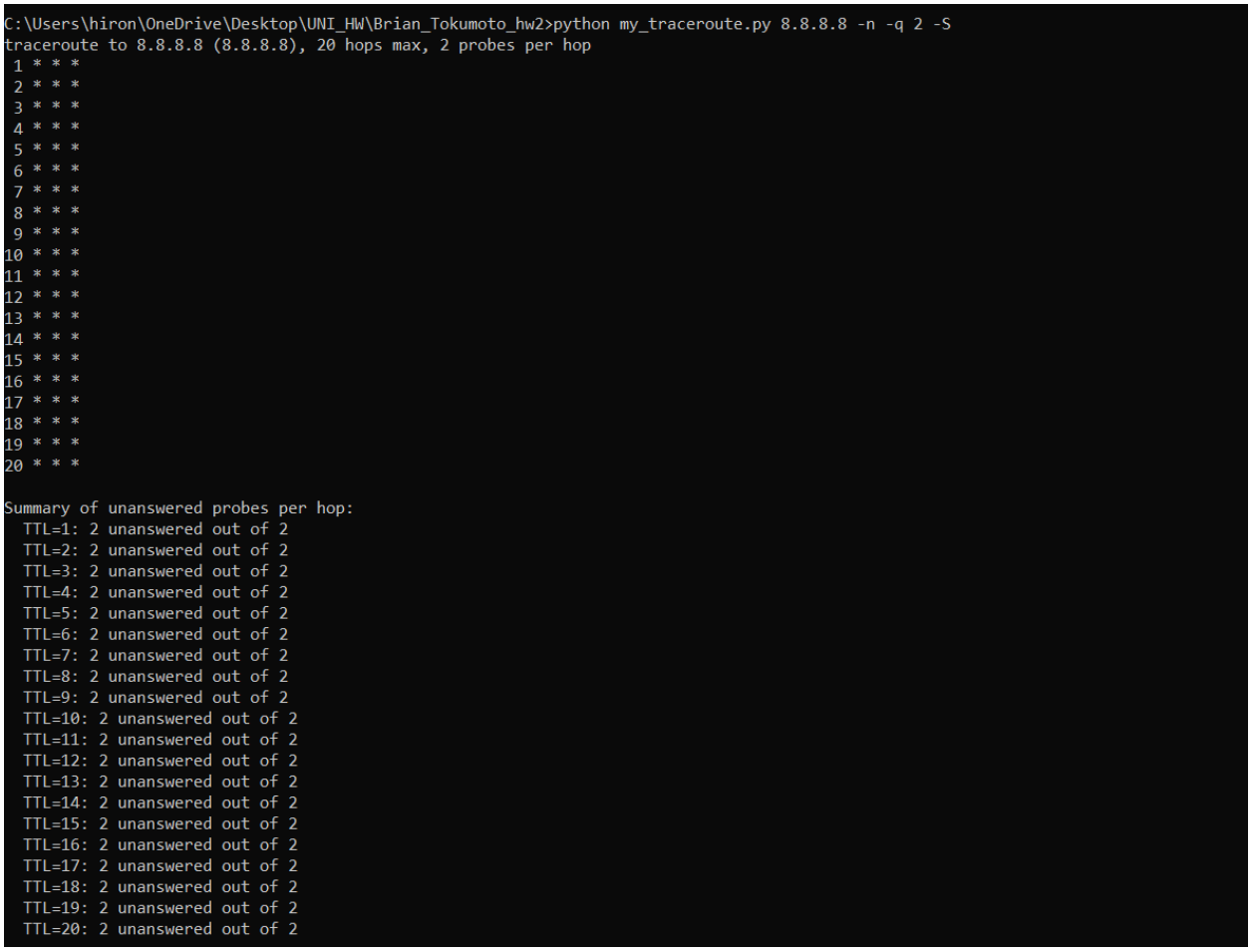
my_traceroute.py

Goal: Discover the path (sequence of routers) a packet takes to reach its destination by gradually incrementing the TTL (Time To Live).

- ICMP / UDP: We send UDP packets to a high-numbered port, expecting intermediate routers to return “ICMP Time Exceeded.” However, many modern networks block these. An alternative is to send ICMP Echo Requests with TTL=1, 2, 3, …
- Command-Line Options:
    - -n: Print numeric IP addresses only.
    - -q N: Number of probes per TTL (default 3).
    - -S: Print a summary of unanswered probes per hop.



python my_traceroute.py 8.8.8.8:
Tries TTL=1, 2, 3, …, up to 20.
The reason traceroute shows * * * (timeouts) for every hop is because no “time exceeded” (or “destination unreachable”) ICMP responses are making it back to the raw socket. Each hop failed to return any ICMP messages to our raw-socket listener. Hence, every probe timed out, results in * * * per hop. I have tried turning off my windows firewall but wasn’t able to figure out why this keeps happening.



Show Summary of Unanswered Probes (-S)

Demonstrates how many probes at each hop got no response. In this case all of them.

Even though every hop timed out, this still demonstrates the script's functionality. Many networks are configured not to send the needed ICMP messages, or the local firewall may be dropping them.

Conclusions

My my_ping.py generally works reliably because most hosts respond to ICMP Echo Requests. The provided screenshots confirm that typical usage matches real-world results (like the standard ping command).

Traceroute: My my_traceroute.py runs without errors, but in modern networks (especially Windows + home routers + possible firewall settings), it's common to see all * * *.

The code is correctly forming and sending the UDP probes with increasing TTL, and opening a raw ICMP socket to listen for responses. The all-timeout result simply shows that, in this specific environment, no probes receive the expected 'time-exceeded' replies.