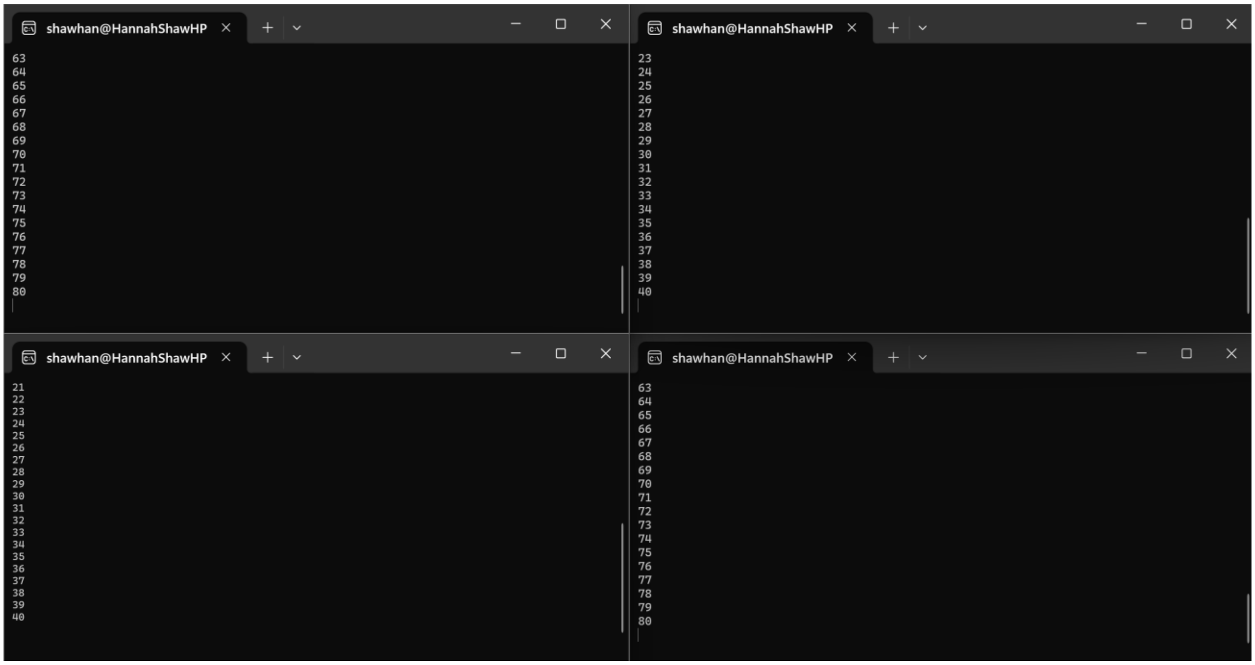
Hannah Shaw – CS536 – Lab6

With multiple tunneling hops (msec)	Base UDPPING application (msec)
Beginning to send 80 ping messages Completed sending of ping messages Minimum: 2.517 Maximum: 7.059 Mean: 4.275 Standard Deviations: 1.193	Begining to send 80 ping messages Completed sending of ping messages Minimum: 0.161 Maximum: 0.547 Mean: 0.219 Standard Deviations: 0.067

Not surprisingly, my applications reported a slower rtt time using the tunnel, with a larger standard deviation. This is likely due to the multiple hops increasing the rtt for each packet, and the added complexity of the multiple hops affecting the standard deviation. When Just pinging back and forth directly, the single step between two machines in the same lab is expectedly quick and consistent.

My applications correctly reported the number of packets that were processed. Hop1 (top left) and hop4 (bottom right) reported processing all 80 ping packets, and hop2(top right) and hop3(bottom left) reported processing half the packets, processing 40 packets each.



```
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
```

```
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

```
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

```
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
```

(The reason I print out the number of packets every iteration of the loop instead of printing a final total at the end is to save headache as I'm able to run the clients in the background of the source terminal and not have double digit terminals).

Bonus question:

Hannah Shaw – CS536 – Lab6

The reported statistics from my file transport app:

File transport through the tunnel: Elapsed time in milliseconds: 46229

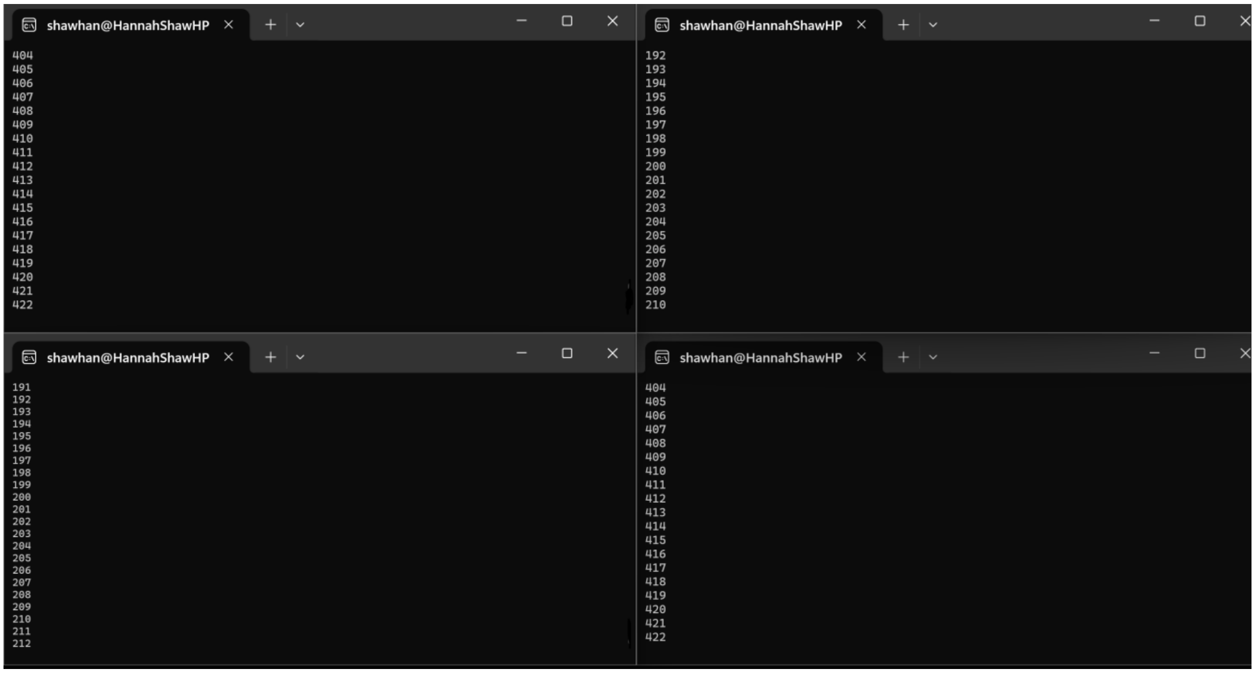
File transport directly to client: Elapsed time in milliseconds: 46057

I saw a similar output showing the number of packets processed by each hop. I am surprised to see the varition in values, as the previous values were perfectly what I expected, but I will attribute this to the number of packets needed to send the file that isnt %20 = 0 like the 80 ping packets were. Nevertheless they follow an intuitive pattern where they will send 10 packets at a time to alternating hops.

Similarly, our elapsed time is expectedly slower when sent through the tunnel, where more transfers need to be made to transfer a single packet. In fact, in order to properly run this, I needed to lengthen the time on my alarms, as otherwise every single packet would time out and the transfer would eventually give up.

Hop1 (top left) hop2(top right)

hop3(bottom left) hop4 (bottom right)



```
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
```

```
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
```

```
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
```

```
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
```

(The reason I print out the number of packets every iteration of the loop instead of printing a final total at the end is to save headache as I'm able to run the clients in the background of the source terminal and not have double digit terminals).

Hannah Shaw – CS536 – Lab6

The following is just my process for testing, not relevant to grading.

If of interest:

Name	Machine	Machine IPv4 Address
source	amber10	10.168.53.**19**
hop1	amber01	10.168.53.**10**
hop2	amber02	10.168.53.**11**
hop3	amber03	10.168.53.**12**
hop4	amber04	10.168.53.**13**
destination	amber05	10.168.53.**14**

step	On machine:	Run:	Port number:
1	amber05	./udppings 50505 secret	
2	amber04	./tvpns 10.168.53.**13** 55555 hannah	
3	amber10	./tvpnc 10.168.53.**13** 55555 hannah 10.168.53.**11** 10.168.53.**14** 50505 &	a
4	amber02	./tvpns 10.168.53.**11** 55555 hannah	
5	amber10	./tvpnc 10.168.53.**11** 55555 hannah 10.168.53.**10** 10.168.53.**13** a &	b
6	amber03	./tvpns 10.168.53.**12** 55555 hannah	
7	amber10	./tvpnc 10.168.53.**12** 55555 hannah 10.168.53.**10** 10.168.53.**13** a &	c
8	amber01	./tvpns 10.168.53.**10** 55555 hannah	
9	amber10	./tvpnc 10.168.53.**10** 55555 hannah 10.168.53.**19** 10.168.53.**11** b &	d
10	amber10	./tvpnc 10.168.53.**10** 55555 hannah 10.168.53.**19** 10.168.53.**12** c &	e
11	amber10	./udppingc 10.168.53.10 d secret 50506 80	

```
>>./tvpns serverip portnum secret

>>./tvpnc serverip serverport secret sourceip destip destport

>>./udppings portnum secret

>>./udppingc serverip serverport secret portnum pcount
```

Hannah Shaw – CS536 – Lab6

Important:

1. The labels a, b, c, d, and e are labels given for the port number determined by the tunneling server and output at the tunneling client. Replace the labels with the respective port number before running. I have made the font of all such labels purple.
2. In **step 11**, I use port number d. This is dependant on how you code it. It is possible that port number e would work for your code.
3. In **step 2**, I state the client IPv4 address as the IPv4 address of hop2 (10.168.53.11). Technically hop3 is also a client. This is also dependent on how you code it. This works for my code. The recvfrom in tvpns should populate the struct in such a way that it will be sent back to the appropriate hop.
4. I recommend running the tvpnc commands with an & on the end to limit terminals

Notes:

1. Port numbers 55555, 50505, and 50506 are arbitrary, you can choose other ones as long as you make the ones i made match, match.
2. Secrets are arbitrary, I used “hannah” when dealing with the tunnel, and “secret” when dealing with udpping. Again, other strings can be chosen, and even some of the “hannah”s could not match, but for simplest results, make the ones I made match, match.
3. I run the ping server first – this just ensures that the port we need for it is available before we set the whole thing up.