

Understanding Software Dynamics Ch 6 Lab Report

I tried many times and cannot postprocess. Then I looked at the code and found that the book's commands given on p.106 are incorrect. The correct command:

```
./dumplogfile4 "ping" client4_20250929_092429_ubuntu2_392753.log  
>client4_20250929_092429_ubuntu2_392753.json  
./makeself show_rpc.html client4_20250929_092429_ubuntu2_392753.json  
client4_20250929_092429_ubuntu2_392753.html  
The above command needed to be run in /postproc.
```

useful command: `ip addr show`

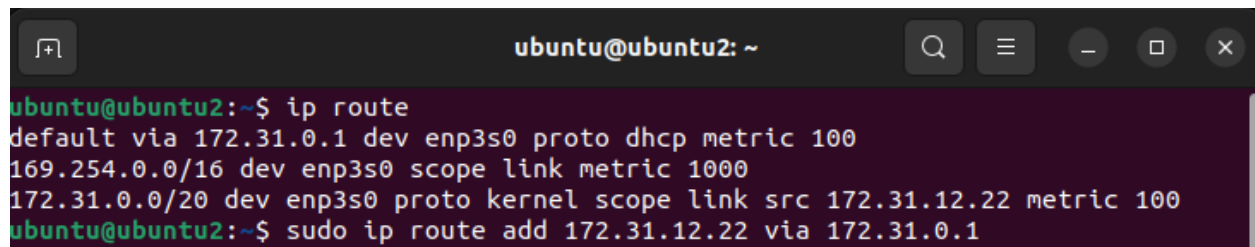
I run server and client in the same machine using

```
./client4_20250929_092429_ubuntu2_392753.html
```

And find out that the estimated network time is extremely small (1us), which is 1000x less than expected. It is because of the OS's internal routing optimization.

Next, I have to configure to bypass OS internal routing optimization (which keeps traffic local), so I modify the routing table.

modify_routing_table.png

A terminal window titled 'ubuntu@ubuntu2: ~' with search, menu, and window control icons. The terminal shows the following commands and output:

```
ubuntu@ubuntu2:~$ ip route  
default via 172.31.0.1 dev enp3s0 proto dhcp metric 100  
169.254.0.0/16 dev enp3s0 scope link metric 1000  
172.31.0.0/20 dev enp3s0 proto kernel scope link src 172.31.12.22 metric 100  
ubuntu@ubuntu2:~$ sudo ip route add 172.31.12.22 via 172.31.0.1
```

Then I verified there is tcp handshake between a client-server test by Wireshark.

wireshark_tcp_handshake.png(frame 127, 129, 130)

(simple client-server test:

```
$ python3 -m http.server 8000 --bind 172.31.12.22
```

```
$ curl http://172.31.12.22:8000)
```

Wireshark capture on interface enp3s0. Filter: ip.addr == 172.31.12.22 or ip.addr == 172.31.0.1

No.	Time	Source	Destination	Protocol	Length	Info
120	6.972066615	172.31.0.1	224.0.0.251	MDNS	85	Standard query 0x2a71 PTR 164.2.31.172.in-addr.arpa, "QM" question
121	6.972394312	172.31.0.1	224.0.0.251	MDNS	84	Standard query 0x2a72 PTR 66.0.31.172.in-addr.arpa, "QM" question
122	6.972691879	172.31.0.1	224.0.0.251	MDNS	86	Standard query 0x2a73 PTR 111.13.31.172.in-addr.arpa, "QM" question
123	6.983028228	172.31.12.22	142.250.76.238	QUIC	74	Protected Payload (KP0), DCID=f199ba3e9e9055c8
124	6.992982836	142.250.76.238	172.31.12.22	QUIC	671	Protected Payload (KP0)
125	7.017254072	142.250.76.238	172.31.12.22	QUIC	63	Protected Payload (KP0)
126	7.017478845	172.31.12.22	142.250.76.238	QUIC	77	Protected Payload (KP0), DCID=f199ba3e9e9055c8
127	7.023762863	1.194.172.138	172.31.12.22	TCP	66	443 → 55900 [ACK] Seq=1 Ack=720 Win=85 Len=0 TSval=2206824089 TSec...
128	7.045819859	142.250.76.238	172.31.12.22	QUIC	66	Protected Payload (KP0)
129	7.214214201	1.194.172.138	172.31.12.22	TLSv1.2	1443	Application Data
130	7.214267615	172.31.12.22	1.194.172.138	TCP	66	55900 → 443 [ACK] Seq=720 Ack=1378 Win=658 Len=0 TSval=3702786047 ...
131	7.472131406	172.31.0.1	224.0.0.251	MDNS	84	Standard query 0x2a74 PTR 17.1.31.172.in-addr.arpa, "QM" question
132	7.472376566	172.31.0.1	224.0.0.251	MDNS	84	Standard query 0x2a75 PTR 99.7.31.172.in-addr.arpa, "QM" question
133	7.472943639	172.31.0.1	224.0.0.251	MDNS	85	Standard query 0x2a76 PTR 18.12.31.172.in-addr.arpa, "QM" question
134	7.475035981	172.31.0.1	224.0.0.251	MDNS	86	Standard query 0x2a78 PTR 111.13.31.172.in-addr.arpa, "QM" question
135	7.475254889	172.31.0.1	224.0.0.251	MDNS	85	Standard query 0x2a79 PTR 164.2.31.172.in-addr.arpa, "QM" question
136	7.475398210	172.31.0.1	224.0.0.251	MDNS	84	Standard query 0x2a77 PTR 66.0.31.172.in-addr.arpa, "QM" question

Frame 130: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface enp3s0, id 0
 Ethernet II, Src: ASUSTekC_6b:53:54 (4c:ed:fb:6b:53:54), Dst: Tp-LinkT_b0:5d:88 (40:3f:8c:b0:5d:88)
 Internet Protocol Version 4, Src: 172.31.12.22, Dst: 1.194.172.138
 Transmission Control Protocol, Src Port: 55900, Dst Port: 443, Seq: 720, Ack: 1378, Len: 0

```

0000  40 3f 8c b0 5d 88 4c ed fb 6b 53 54 08 00 45 00  @?...].L. .kST..E.
0010  00 34 91 ac 40 00 40 06 42 96 ac 1f 0c 16 01 c2  -4-@.@. B.....
0020  ac 8a da 5c 01 bb b7 a7 91 b4 c7 f9 dc 41 80 10  ...\......A...
0030  02 92 66 a8 00 00 01 01 08 0a dc b4 07 ff 83 89  ..f.....
0040  77 58                                     wX
  
```

Wireshark enp3s0FGLD3.pcapng Packets: 139 · Displayed: 99 (71.2%) · Dropped: 0 (0.0%) Profile: Default

```

./client4 172.31.12.22 12345 ping
./client4 172.31.12.22 12345 -k 10 write -key "kkkkk" + -value "kkkkk" + 1000000
./client4 172.31.12.22 12345 -k 10 read -key "kkkkk" +
./client4 172.31.12.22 12345 quit
  
```

Important: 50ms is the standard magnitude for request-response latency. (RTT)

```

ubuntu@ubuntu2: ~/Documents/github/KUtrace-experiments
ubuntu@ubuntu2:~/Documents/github/KUtrace-experiments$ ./client4 172.31.12.22 12
345 ping
at client, server IP = ac1f0c16:3039

Histogram of floor log 2 buckets of usec response times
1 2+ 4+ us          1+ 2+ 4+ msec          1+ 2+ 4+ sec          1K+ 2k+ secs
|                  |                  |                  |
0 0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
1 RPCs,    0.5 msec, 0.000 TxMB, 0.000 RxMB total
1945.5 RPC/s (0.514 msec/RPC),    0.2 TxMB/s,    0.2 RxMB/s

client4_20250930_144727_ubuntu2_523896.log written
0.514ms  ubuntu@ubuntu2:~/Documents/github/KUtrace-experiments$ ./client4 172.31
.12.22 12345 -k 10 write -key "kkkkk" + -value "kkkkk" + 1000000
at client, server IP = ac1f0c16:3039
1.993ms 1.917ms 1.711ms 1.221ms 1.256ms 1.207ms 1.323ms 1.250ms 1.235ms
1.388ms

Histogram of floor log 2 buckets of usec response times
1 2+ 4+ us          1+ 2+ 4+ msec          1+ 2+ 4+ sec          1K+ 2k+ secs
|                  |                  |                  |
0 0 0 0 0 0 0 0 0 0  10 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
10 RPCs,   14.5 msec, 10.001 TxMB, 0.001 RxMB total
689.6 RPC/s (1.450 msec/RPC), 689.7 TxMB/s,    0.1 RxMB/s

client4_20250930_144756_ubuntu2_523938.log written
ubuntu@ubuntu2:~/Documents/github/KUtrace-experiments$ ./client4 172.31.12.22 12
345 -k 10 read -key "kkkkk" +
at client, server IP = ac1f0c16:3039
1.085ms 0.571ms 0.298ms 0.293ms 0.242ms 0.247ms 0.272ms 0.261ms 0.260ms
0.260ms

Histogram of floor log 2 buckets of usec response times
1 2+ 4+ us          1+ 2+ 4+ msec          1+ 2+ 4+ sec          1K+ 2k+ secs
|                  |                  |                  |
0 0 0 0 0 0 0 0 2 6 1  1 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
10 RPCs,    3.8 msec, 0.001 TxMB, 10.001 RxMB total
2639.2 RPC/s (0.379 msec/RPC),    0.3 TxMB/s, 2639.5 RxMB/s

client4_20250930_144804_ubuntu2_523948.log written
ubuntu@ubuntu2:~/Documents/github/KUtrace-experiments$ ./client4 172.31.12.22 12
345 quit
at client, server IP = ac1f0c16:3039

Histogram of floor log 2 buckets of usec response times
1 2+ 4+ us          1+ 2+ 4+ msec          1+ 2+ 4+ sec          1K+ 2k+ secs
|                  |                  |                  |
0 0 0 0 0 0 0 0 0 1 0  0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0
1 RPCs,    0.4 msec, 0.000 TxMB, 0.000 RxMB total
2469.1 RPC/s (0.405 msec/RPC),    0.2 TxMB/s,    0.2 RxMB/s

client4_20250930_144810_ubuntu2_523960.log written

```

6.1) client4_20250930_144727_ubuntu2_523896.html

estimate: 0.01ms pings are usually small ICMP packets. And the server only needs to validate the checksum.

reality: 0.5ms

6.2) client4_20250930_144756_ubuntu2_523938.html

estimate: 0.05ms for 1MB if written in RAM, because it involves key parsing, memory allocation, and data copy

Reality: 1.5ms each on average

6.3) client4_20250930_144804_ubuntu2_523948.html

estimate: 0.03ms because reads are usually faster than writes, because no memory allocation

Reality: 1ms for the first read, and 0.3ms on average for later reads

And all the response message transmission above just adds RTT/2 (from the server back to the client after processing).

Reason for difference:

- Software Overhead: Parsing, serialization, or protocol handling (e.g., TCP, HTTP stack overhead like kernel interrupts and socket handling) adds ~0.1-0.5 ms.
- System Load: complex memory management (e.g., allocation delay/fragmentation when searching for available memory, garbage collection to find free space) adds ~0.5-1 ms for 1 MB ops.
- Initialization: First read's 1 ms likely includes setup (e.g., cache miss, session init), while subsequent reads benefit from caching.

It is not likely to be because of disk IO, which would be 5-10ms.