Assignment Repository URL is: https://github.com/alfiyansys/5505EAS-Network-Concept

5505EAS-Network-Concept

Written by M Alfiyan Syamsuddin - 1225800008

The final semester assignment is to demonstrate how (L7 Application layer) socket programming works and its implementations on programming.

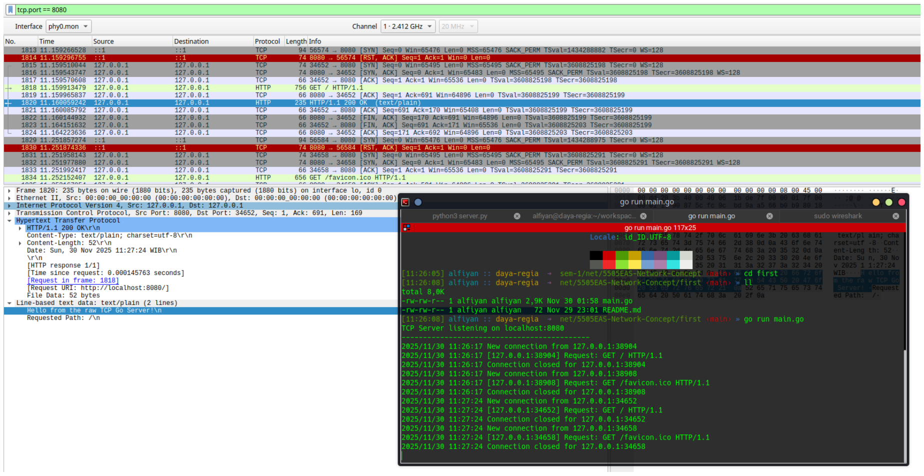
---

1. First Assignment: HTTP Server using RAW TCP Connections

   The first assignment is to demonstrate principles of RAW TCP socket programming and its implementations on HTTP L7 Application protocol layer. It written on Golang instead of python.

   The server is configurated to run on localhost and port 8080. Program begans with listening for incoming connections. Once a connection is established and entered main program loop, the server sends a response to the client using goroutine.

   The client's request is parsed, including IP address, port number, and HTTP method. If the HTTP protocol is present using buffered reader, the server will send a response using buffered writer that manually consutructed HTTP response.



```
No.  Time          Source     Destination Protocol Length Info
1813 11.159266528  ::1        ::1         TCP        94 56574 → 8080 [SYN] Seq=0 Win=65476 Len=0 MSS=65476 SACK_PERM TSval=1434288882 TSecr=0 WS=128
1814 11.159296755  ::1        ::1         TCP        74 8080 → 56574 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1815 11.159510044  127.0.0.1  127.0.0.1   TCP        74 34652 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3608825198 TSecr=0 WS=128
1816 11.159543747  127.0.0.1  127.0.0.1   TCP        74 8080 → 34652 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3608825198 TSecr=3608825198 WS=128
1817 11.159570608  127.0.0.1  127.0.0.1   TCP        66 34652 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3608825198 TSecr=3608825198
1818 11.159913479  127.0.0.1  127.0.0.1   HTTP      756 GET / HTTP/1.1
1819 11.159965837  127.0.0.1  127.0.0.1   TCP        66 8080 → 34652 [ACK] Seq=1 Ack=691 Win=64896 Len=0 TSval=3608825199 TSecr=3608825199
1820 11.160059242  127.0.0.1  127.0.0.1   HTTP      235 HTTP/1.1 200 OK  (text/plain)
1821 11.160085792  127.0.0.1  127.0.0.1   TCP        66 34652 → 8080 [ACK] Seq=691 Ack=170 Win=65408 Len=0 TSval=3608825199 TSecr=3608825199
1822 11.160144932  127.0.0.1  127.0.0.1   TCP        66 8080 → 34652 [FIN, ACK] Seq=170 Ack=691 Win=64896 Len=0 TSval=3608825199 TSecr=3608825199
1823 11.164151632  127.0.0.1  127.0.0.1   TCP        66 34652 → 8080 [FIN, ACK] Seq=691 Ack=171 Win=65536 Len=0 TSval=3608825203 TSecr=3608825199
1824 11.164223636  127.0.0.1  127.0.0.1   TCP        66 8080 → 34652 [ACK] Seq=171 Ack=692 Win=64896 Len=0 TSval=3608825203 TSecr=3608825203
1829 11.251857274  ::1        ::1         TCP        94 56584 → 8080 [SYN] Seq=0 Win=65476 Len=0 MSS=65476 SACK_PERM TSval=1434288975 TSecr=0 WS=128
1830 11.251874336  ::1        ::1         TCP        74 8080 → 56584 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1831 11.251958143  127.0.0.1  127.0.0.1   TCP        74 34658 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3608825291 TSecr=0 WS=128
1832 11.251977880  127.0.0.1  127.0.0.1   TCP        74 8080 → 34658 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3608825291 TSecr=3608825291 WS=128
1833 11.251992417  127.0.0.1  127.0.0.1   TCP        66 34658 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3608825291 TSecr=3608825291
1834 11.252152407  127.0.0.1  127.0.0.1   HTTP      656 GET /favicon.ico HTTP/1.1
```

```
▸ Frame 1820: 235 bytes on wire (1880 bits), 235 bytes captured (1880 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
▸ Transmission Control Protocol, Src Port: 8080, Dst Port: 34652, Seq: 1, Ack: 691, Len: 169
▾ Hypertext Transfer Protocol
  ▸ HTTP/1.1 200 OK\r\n
    Content-Type: text/plain; charset=utf-8\r\n
  ▸ Content-Length: 52\r\n
    Date: Sun, 30 Nov 2025 11:27:24 WIB\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.000145763 seconds]
    [Request in frame: 1818]
    [Request URI: http://localhost:8080/]
    File Data: 52 bytes
▾ Line-based text data: text/plain (2 lines)
    Hello from the raw TCP Go Server!\n
    Requested Path: /\n
```

```
Locale: id_ID.UTF-8

[11:26:05] alfiyan :: daya-regia  →  sem-1/net/5505EAS-Network-Concept ‹main› » cd first
[11:26:08] alfiyan :: daya-regia  →  net/5505EAS-Network-Concept/first ‹main› » ll
total 8,0K
-rw-rw-r-- 1 alfiyan alfiyan 2,9K Nov 30 01:58 main.go
-rw-rw-r-- 1 alfiyan alfiyan   72 Nov 29 23:01 README.md
[11:26:08] alfiyan :: daya-regia  →  net/5505EAS-Network-Concept/first ‹main› » go run main.go
TCP Server listening on localhost:8080
--------------------------------------------
2025/11/30 11:26:17 New connection from 127.0.0.1:38904
2025/11/30 11:26:17 [127.0.0.1:38904] Request: GET / HTTP/1.1
2025/11/30 11:26:17 Connection closed for 127.0.0.1:38904
2025/11/30 11:26:17 New connection from 127.0.0.1:38908
2025/11/30 11:26:17 [127.0.0.1:38908] Request: GET /favicon.ico HTTP/1.1
2025/11/30 11:26:17 Connection closed for 127.0.0.1:38908
2025/11/30 11:27:24 New connection from 127.0.0.1:34652
2025/11/30 11:27:24 [127.0.0.1:34652] Request: GET / HTTP/1.1
2025/11/30 11:27:24 Connection closed for 127.0.0.1:34652
2025/11/30 11:27:24 New connection from 127.0.0.1:34658
2025/11/30 11:27:24 [127.0.0.1:34658] Request: GET /favicon.ico HTTP/1.1
2025/11/30 11:27:24 Connection closed for 127.0.0.1:34658
```

Figure 1: RAW TCP based HTTP server access

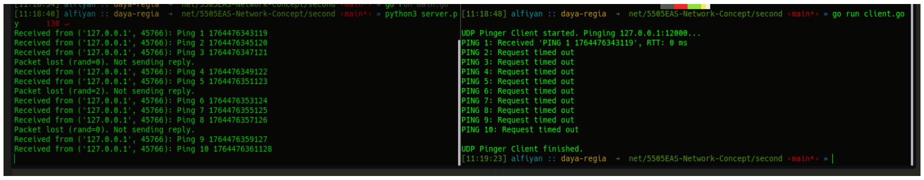
---

2. Second project: UDP Pinger Client Impelementations

The second assignment is to demonstrate how ping messages are implemented using UDP based connections. The server runtime is provided using python. The randominess is to simulate UDP transmission error.

The provided server code logic shouldn't be modified to preserve assignment constraints. The code is slightly modified to support debugging purposes.

Main assignment is to write up and implement pinger client. It written on Golang instead of python at first, until it was changed back to python as instructed.



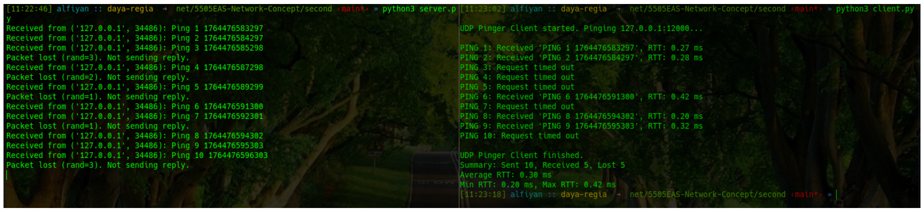
```
[11:18:34] alfiyan :: daya-regia → net/5505EAS-Network-Concept/second ‹main*› » go run main.go
[11:18:40] alfiyan :: daya-regia → net/5505EAS-Network-Concept/second ‹main*› » python3 server.p
y      130 ↵
Received from ('127.0.0.1', 45766): Ping 1 1764476343119
Received from ('127.0.0.1', 45766): Ping 2 1764476345120
Received from ('127.0.0.1', 45766): Ping 3 1764476347121
Packet lost (rand=0). Not sending reply.
Received from ('127.0.0.1', 45766): Ping 4 1764476349122
Received from ('127.0.0.1', 45766): Ping 5 1764476351123
Packet lost (rand=2). Not sending reply.
Received from ('127.0.0.1', 45766): Ping 6 1764476353124
Received from ('127.0.0.1', 45766): Ping 7 1764476355125
Received from ('127.0.0.1', 45766): Ping 8 1764476357126
Packet lost (rand=0). Not sending reply.
Received from ('127.0.0.1', 45766): Ping 9 1764476359127
Received from ('127.0.0.1', 45766): Ping 10 1764476361128
```

```
[11:18:48] alfiyan :: daya-regia → net/5505EAS-Network-Concept/second ‹main*› » go run client.go

UDP Pinger Client started. Pinging 127.0.0.1:12000...
PING 1: Received 'PING 1 1764476343119', RTT: 0 ms
PING 2: Request timed out
PING 3: Request timed out
PING 4: Request timed out
PING 5: Request timed out
PING 6: Request timed out
PING 7: Request timed out
PING 8: Request timed out
PING 9: Request timed out
PING 10: Request timed out

UDP Pinger Client finished.
[11:19:23] alfiyan :: daya-regia → net/5505EAS-Network-Concept/second ‹main*› »
```

Figure 2: Golang based UDP pinger client



```
[11:22:46] alfiyan :: daya-regia → net/5505EAS-Network-Concept/second ‹main*› » python3 server.p
y
Received from ('127.0.0.1', 34486): Ping 1 1764476583297
Received from ('127.0.0.1', 34486): Ping 2 1764476584297
Received from ('127.0.0.1', 34486): Ping 3 1764476585298
Packet lost (rand=3). Not sending reply.
Received from ('127.0.0.1', 34486): Ping 4 1764476587298
Packet lost (rand=2). Not sending reply.
Received from ('127.0.0.1', 34486): Ping 5 1764476589299
Packet lost (rand=1). Not sending reply.
Received from ('127.0.0.1', 34486): Ping 6 1764476591300
Received from ('127.0.0.1', 34486): Ping 7 1764476592301
Packet lost (rand=1). Not sending reply.
Received from ('127.0.0.1', 34486): Ping 8 1764476594302
Received from ('127.0.0.1', 34486): Ping 9 1764476595303
Received from ('127.0.0.1', 34486): Ping 10 1764476596303
Packet lost (rand=3). Not sending reply.
```

```
[11:23:02] alfiyan :: daya-regia → net/5505EAS-Network-Concept/second ‹main*› » python3 client.py

UDP Pinger Client started. Pinging 127.0.0.1:12000...

PING 1: Received 'PING 1 1764476583297', RTT: 0.27 ms
PING 2: Received 'PING 2 1764476584297', RTT: 0.28 ms
PING 3: Request timed out
PING 4: Request timed out
PING 5: Request timed out
PING 6: Received 'PING 6 1764476591300', RTT: 0.42 ms
PING 7: Request timed out
PING 8: Received 'PING 8 1764476594302', RTT: 0.20 ms
PING 9: Received 'PING 9 1764476595303', RTT: 0.32 ms
PING 10: Request timed out

UDP Pinger Client finished.
Summary: Sent 10, Received 5, Lost 5
Average RTT: 0.30 ms
Min RTT: 0.20 ms, Max RTT: 0.42 ms
[11:23:18] alfiyan :: daya-regia → net/5505EAS-Network-Concept/second ‹main*› »
```

Figure 3: Python based UDP pinger client

The Golang based is not working properly after first ping is because of how different programming language implements OS scheduling routine. As server is using python based is using IO blocking model, while Go is non-blocking.

Based on Pytho3 based UDP ping client, as seen in the image above, the server is running on port 12000. The client is running on localhost, while the server is running on the same machine. Both sides (server and client) are showing in what UNIX time both are communicating, as well as the sequence number.

Test results shows loss rate is 50% due to 0..4 range of loss randominess (loss rate should be 40~50% to be exact, valid). With RTT averaging about 0.3ms, max 0.42ms, min 0.2ms.