Report for assignment 1 **Zhanghao Wu (516030910593)**

CS391 Computer Networking ACM Class, Zhiyuan College, SJTU

Prof. **Yanmin Zhu** Due Date: Oct 16, 2018

TA. **Haobing Liu** Submit Date: October 14, 2018

1 Web server

1.1 Basic Exercise

This assignment ask us to write a web server, which can response to HTTP request, using socket to send and receive HTTP packets.

The way to solve this task is quite simple. Firstly, the server creates a socket which listens to a specific port on local host. Whenever a client tries to connect to the IP/port the server runs on, the server starts a logical link with it and waits for the HTTP request from the client. If the file requested by the client exists, the server replies 'HTTP/1.1 200 OK' followed by content of the file. Otherwise, the server replies 'HTTP/1.1 404 Not Found' to the client implies that it fails to find the file requested.

Figure 1 shows the result of the execution.

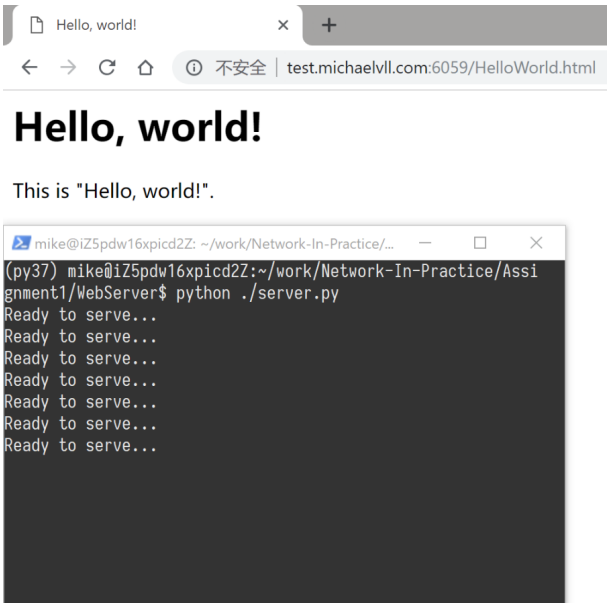

Figure 1: The running result of web server

1.2 Optional Exercises

I implemented a multi-thread web server and a client for this assignment.

- For the server, whenever a connection is raised, the server start a thread to handle the HTTP request from the client.
- For the client, it sends HTTP request constantly in order to test the server.

2 UDP pinger

2.1 Basic Exercise

This assignment ask us to write a UDP ping client, which can send 10 pings to the server and wait up to 1 second for a reply for each pings.

To solve the problem, I create a socket first with a param telling that its a socket for UDP. Then I bind it to a port. Then it sends pings to the server and receive the messages from it sequentially.

Figure 2 shows the running result of the pinger.

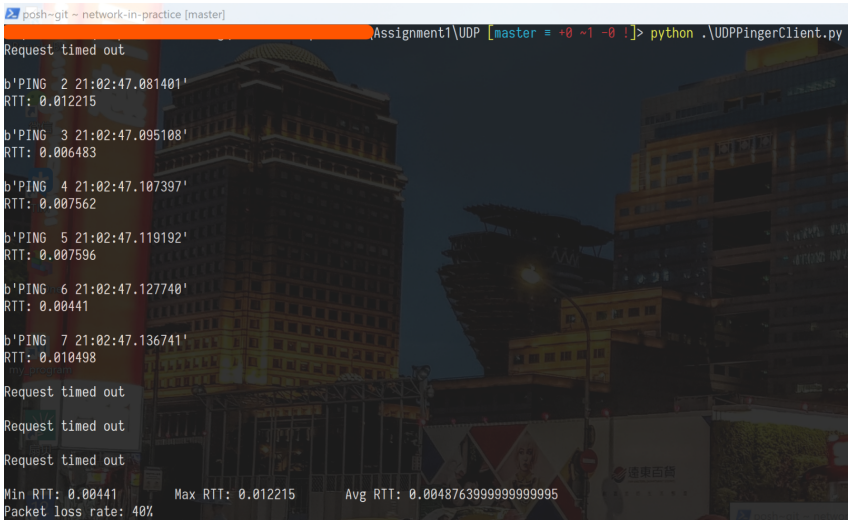

Figure 2: The running result of udp pinger

2.2 Optional Exercises

I add some code to find out the minimum, maximum, and average RTTs at the end of all pings from the client, by calculating the difference between receiving time and sending time. In

addition, calculate the packet loss rate (in percentage)(option 1), using the sequence number in the packets. It is also shown in Figure 2.

Also, I implemented the UDP Heartbeat server and client (option 2). The server wait the heartbeat from the client constantly until a new hearbeat packet does not arrive for 5 seconds. Timestamps and pakcet IDs are added to the packets and the server can calculate the duration and packet loss using these informations.

Figure 3 shows the running result of the udp heartbeat.

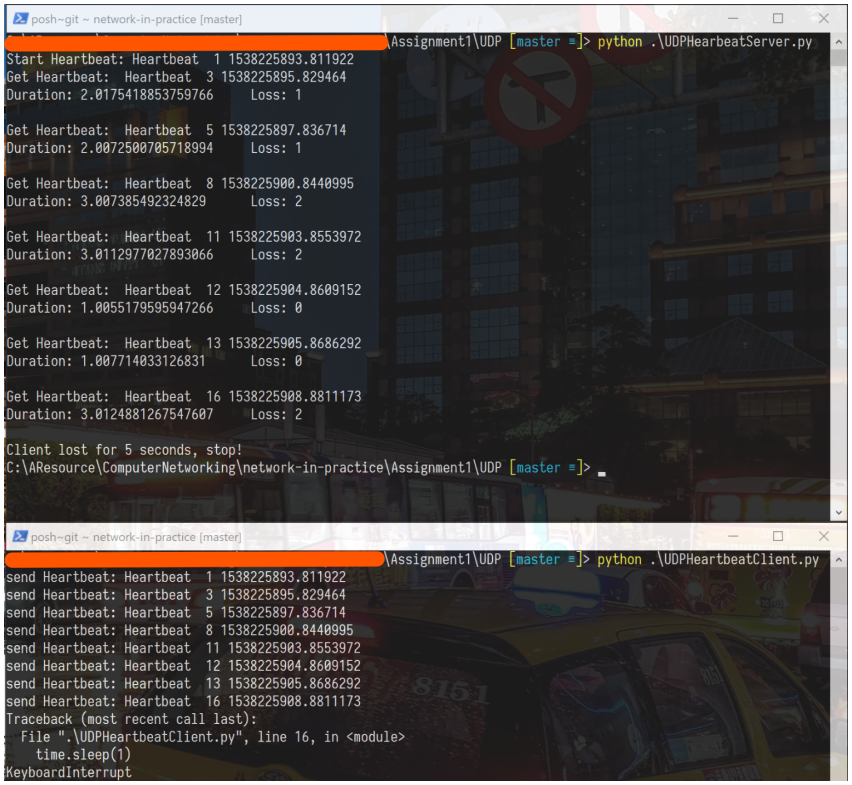

Figure 3: The running result of udp heartbeat

2.3 Some thought

By solving the two problems above, I get some understanding of TCP and UDP in connection. For a TCP connection, when a client needs to send messages to the server, it needs to start a unique port to port logical connection with the server. However, in UDP connection, there is no needs to start a logical connection. And each message sending uses a disposable logical connection.

3 Mail client

3.1 Basic Exercise

This problem asks us to implement a simple mail client taht sends mail to any recipient.

In order to send something to a email address, the mail client needs firstly connect to a email server. After sending identification informations, including my own email address and the password in base64, the mail client is connected to the email server. And now the client can send text to any recipient by following the SMTP protocol.

Figure 4 shows the mail I send from my mail client (connected to qq mail server) to my SJTU mail box.

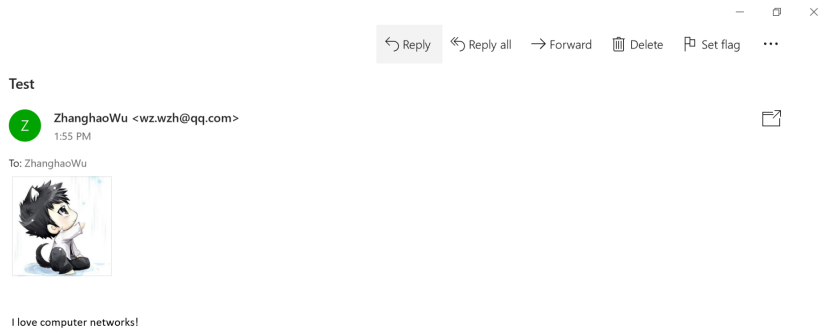

Figure 4: A mail sent by my mail client

3.2 Optional Exercises

The first optional exercise asks us to connect the mail server in TLS or SSl for authentication and security reasons. To do that, the port my mail client connects to should be changed to the port that support TLS or SSL connection of the email server. Also the mail client should send a command telling the email server to start a secure connection.

The second optional exercise asks us to send image with the text using our mail client. By following the 'MIME' protocol, just adding some headers, some extra predefined contents and the encoded image in the message, the recipient can receive the mail with text and image. And Figure 4, shows the image and text I sent.

4 Web proxy

4.1 Basic Exercise

The basic request of this task asks us to implement a web proxy server.

When a HTTP request received, the proxy server seperate the hostname from the header of the message. And it connect with the 80 port of the host. The proxy server sends a command to the host requesting the files needed by the client and send them to the client.

Figure 5 shows the result of the proxy server I implemented.

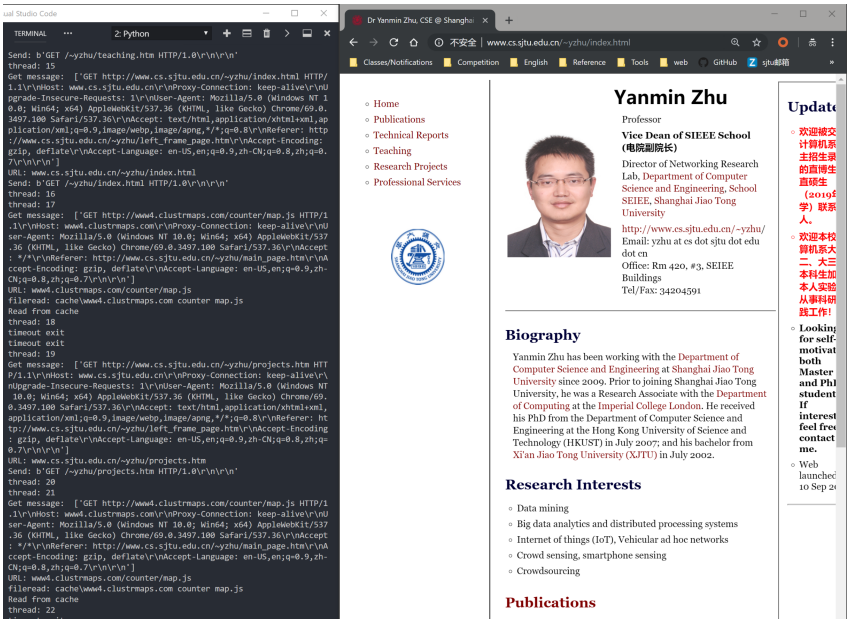

Figure 5: The result of the proxy server

4.2 Optional Exercises

When the proxy server receives a HTTP request from the client requesting a not existing host. The step trying to connect the host will raise a exception and the proxy server will return a '404 Not Found' response to the client.

I also added a cache to the proxy server. The proxy server will look for the cache file related to the HTTP request from the client. If the cache file exists, the proxy server will send it back to the client. Or, otherwise, the proxy server will request the host for the content needed like it did before. When getting a new content from the host, it saves it to the cache.

4.3 Some thought

After implemented the proxy server, I found that when there are a lot of contents in a website, it may takes long time to load the website. So I change the proxy server to be multi-thread for dealing with the request from the client. And each thread will keep running until it has not received request from client for 5 seconds.

However, nowadays, most of the website are encrypted by HTTPS protocol. Each time the content is requested, it will be encrypted in different ways, so there may be some challenges to implement a cache for HTTPS.