Assignment-1 of Computer Networking

Liu Yuxi @ October 12, 2018

Contents

1 Webserver **3**
- 1.1 Basic target . . . 3
- 1.2 Optional target . . . 3
  - 1.2.1 Multithreaded server . . . 3
  - 1.2.2 HTTP client . . . 3
- 1.3 Implementation . . . 3
  - 1.3.1 Server . . . 3
  - 1.3.2 Client . . . 4
- 1.4 Conclusion . . . 4

2 UDPPinger **5**
- 2.1 Basic target . . . 5
- 2.2 Optional target . . . 5
  - 2.2.1 RTTs . . . 5
  - 2.2.2 Heartbeat . . . 5
- 2.3 Implementation . . . 5
  - 2.3.1 UDPPingerClient . . . 5
  - 2.3.2 UDPHeartbeatServer . . . 6
  - 2.3.3 UDPHeartbeatClient . . . 6
- 2.4 Conclusion . . . 6

3 MailClient **7**
- 3.1 Basic target . . . 7
- 3.2 Optional Exercises . . . 7
  - 3.2.1 TLS or SSL . . . 7
  - 3.2.2 Send images . . . 7
- 3.3 Implementation . . . 7
  - 3.3.1 MailClient . . . 7

3.3.2 SMTP protocol 7
3.4 Conclusion 8

4 WebProxyServer **9**
4.1 Basic target 9
4.2 Optional target 9
4.2.1 404 Not Found 9
4.2.2 POST 9
4.2.3 Caching 9
4.3 Implementation 9
4.3.1 WebProxyServer 9
4.4 Conclusion 10

1 Webserver

1.1 Basic target

Develop a web server that handles HTTP requests. The web server accepts and parses the HTTP request, gets the requested file from the file system, creates an HTTP response message consisting of the requested file preceded by header lines, and then sends the response directly to the client. If the requested file is not present in the server, the server sends an HTTP ”404 Not Found” message back to the client.

1.2 Optional target

1.2.1 Multithreaded server

Implement a Multithreaded server that is capable of serving muliple requests simultaneously.

1.2.2 HTTP client

Write a HTTP client to test the server

1.3 Implementation

1.3.1 Server

(a) Build a socket to bind a adress - (id, post)

(b) Establish a connection by $.accept()$ method constantly

(c) Set a threading with target as a function to deal with the connection and arguments as the new socket and new address

(d) Recieve the message by $.recv()$ method

(e) Try to open the requested file in the file system

(f) Send HTTP message ”HTTP/1.1 200 OK” and the data if the file exists

(g) Send HTTP message ”HTTP/1.1 404 Not Found” if the file doesn’t exists

1.3.2 Client

(a) Connect the server's socket by $.connect()$ method

(b) Send "Get /filename HTTP/1.1" to get the file and receive the data

1.4 Conclusion

Learn to take advantage of some basic methods of socket and get a good command of the fundamental interactive mechanism of the web server and the client. Besides, realize some formats of HTTP messsages.

2 UDPPinger

2.1 Basic target

Use UDP sockets to send and receive datagram packets. Learn how to set a proper socket timeout. Compute the statistics such as packet loss rate.

2.2 Optional target

2.2.1 RTTs

Calculate the round-trip time for each packet and report their minimum, maximum and average. In addition, calculate the packet loss rate.

2.2.2 Heartbeat

The Heartbeat is used to check if an application is up and running and to report one-way packet loss. The client sends a sequence number and current timestamp in the UDP packet to the server, which is listening for the Heartbeat (i.e., the UDP packets) of the client. Upon receiving the packets, the server calculates the time difference and reports any lost packets. If the Heartbeat packets are missing for some specified period of time, we can assume that the client application has stopped.

2.3 Implementation

2.3.1 UDPPingerClient

(a) Use $.settimeout()$ to set a timeout of each packet

(b) Use $time()$ to get the precise time and calculate the time difference the get the RTTs

(c) Use a list to store the RTTs and get the max, min and ave

(d) Write down the number of the ”lost”(timeout) packets

2.3.2 UDPHeartbeatServer

(a) Receive the packets and write down the time and their sequence number in two lists

(b) Calculate the time difference between the forward packet and the current packet

(c) Print the lost packets between the forward packet and the current packet

2.3.3 UDPHeartbeatClient

(a) Send sequence number and local time with a random loss

2.4 Conclusion

Learn the fundamental mechanism of UDP and the definition of RTT and packet loss rate.

3 MailClient

3.1 Basic target

Develop a simple mail client that sends email to any recipient. The client will need to connect to a mail server, dialogue with the mail server using the SMTP protocol, and send an email message to the mail server.

3.2 Optional Exercises

3.2.1 TLS or SSL

Add a Transport Layer Security(TLS) or Secure Sockets Layer(SSL) for authentication and security reasons

3.2.2 Send images

Modify the client such that it can send emails with both text and images

3.3 Implementation

3.3.1 MailClient

(a) Choose (”smtp.qq.com”, 587) as the mail server (port 25 is ok without TLS nor SSL)

(b) Follow the rules of SMTP protocol to interact with the server by methods $.send()$ and $.recv()$

3.3.2 SMTP protocol

(a) HELO QQ email

(b) STARTLS

(c) HELO

(d) AUTH LOGIN

(e) (Username)

(f) (Password, actually authorization code in smtp.qq.com)

(g) MAIL FROM : ⟨ @qq.com ⟩

(h) RCPT TO : ⟨ @ ⟩

(i) DATA

(j) Header

(k) MIME-Header

(l) Text-Header

(m) Message data

(n) Image header

(o) Image data

(p) Frontier

(q) QUIT

3.4 Conclusion

Learn to interact with the mail server following SMTP protocol. Learn some headers for sending messages both with text and images. Learn the TLS and SSL for security and authentication.

4 WebProxyServer

4.1 Basic target

Develop a small web proxy server which is able to cache web pages. The proxy server only understands simple GET-requests, but is able to handle all kinds of objects not just HTML pages.

4.2 Optional target

4.2.1 404 Not Found

When the client requests an object which is not available, send the ”404 Not Found” response.

4.2.2 POST

Add support for POST-request.

4.2.3 Caching

When the proxy gets a request, it checks if the requested object is cached, and if yes, it returns the object from the cache, without contacting the server. If the object is not cached, the proxy retrieves the object from the server, returns it to the client and caches a copy for future requests.

4.3 Implementation

4.3.1 WebProxyServer

(a) Establish a cache in the local file system

(b) Establish a tcpSerSocket binded with (’127.0.0.1’, 2335) and keep listening

(c) Accept a (tcpCliSocket, addr) from tcpSerSocket.accept()

(d) Receive message and extract the url or filename

(e) Encode the filename(or url) by MD5 and look up the corresponding file in the local cache (Or just the original filename for HelloWorld.html)

(f) If the file exists, proxy server finds a chache hit and generate some response messages(status, header, data)

(g) Catch the IOError when the cache misses and create a new socket on the proxy server

(h) Try to connect the host(created by the url) an port 80

(i) Get the data by GET-message sended to the socket and store the data

(j) Write data down to the local cache and also send it to the tcpCliSocket if successful connection

(k) Otherwise, if the status is '404', send '404 Not Found'

4.4 Conclusion

Learn to establish tcp connections to implement a proxy server. Learn some knowledge about different ports and the format of response(header, status, data, etc.). Learn to encode the file by MD5 in the local file system.