Programming Assignment: Webserver

Author: Anup Bharadwaj P538- Computer Networks

Write-up:

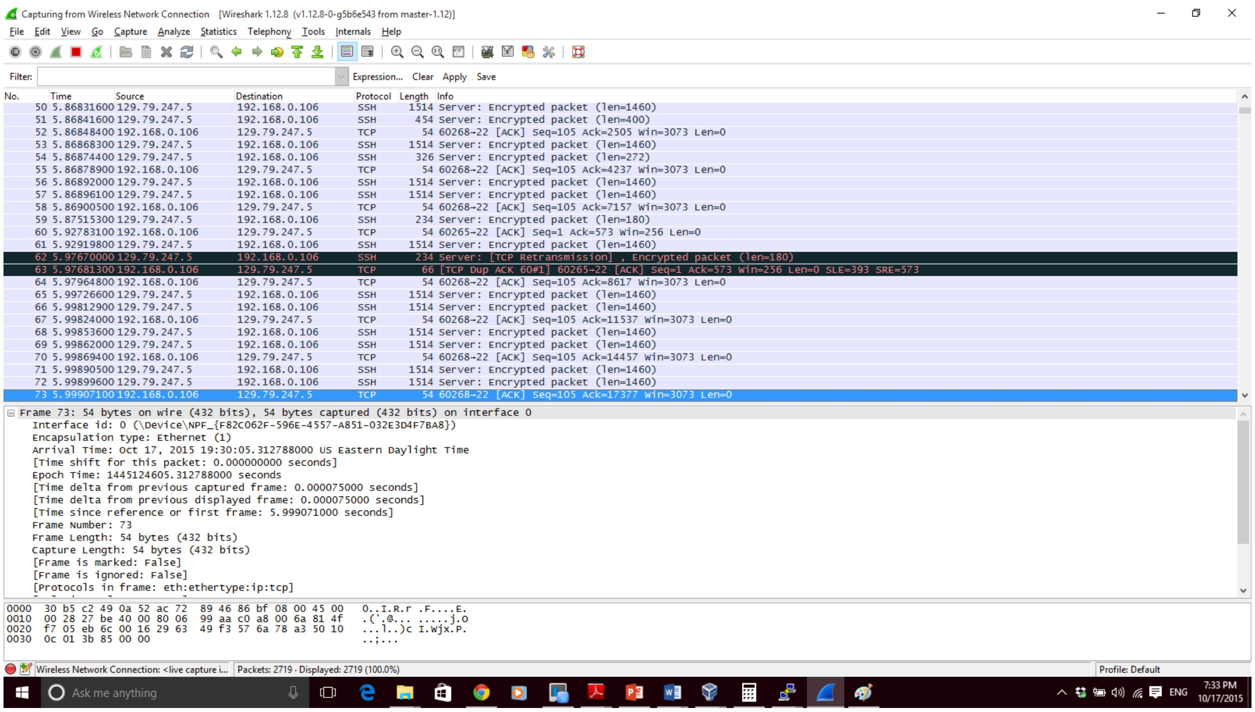
Persistent connections keep the socket open to process the requests from the client and hence are expected to be relatively more efficient compared to non-persistent connections where the client has to undergo a TCP handshake for every request. Therefore, non-persistent connections take 2 RTT more to send a request and get the response from the server.

In our client and the server application, one to eight 1.3 MB files are requested by the client using both persistent and non-persistent connections. The table below summarizes the time taken under each scenario. To perform the analysis, I have taken a text file named “xl.txt” which is 1.3 MB in size.

Number of Files * File size (MB)	Persistent Connections (Time in seconds)	Non-persistent Connections(Time in seconds)
1.3	0.55	0.57
2.6	2.07	1.5
3.9	3.57	2.04
5.2	4.78	3
6.5	8.61	4.2
7.8	8.74	4.9
9.1	10.76	5.9
10.4	11.86	6.8

Several test runs have been made and the average of the runs made in each scenario has been tabulated above. For non-persistent connections, the sum of the time taken to request each file has been taken. However for persistent connections, the time taken to fetch all the files at once is recorded. Therefore, the overall time taken for non-persistent connections could be much longer when we consider the time taken to send each requests after server closing the socket every time. Hence, even though the time taken in the case of persistent connections appears to be longer, it includes the overall time taken to fetch the contents of the files requested in order and hence is more efficient as it avoids a TCP handshake for every request.

It takes almost the same amount of time to request and receive a 1 MB file in both persistent and non-persistent connections. From the above table, it took 0.55 and 0.56 seconds for persistent and non-persistent connections respectively to request and receive a 1.3 MB file from the server.



Multi-threaded server:

The multi-threaded server application was much more efficient in handling both persistent and non-persistent connections compared to the single threaded server application. Since each thread was handling requests in parallel, the client was able to receive the response with at least 20-30% lesser time compared to a single threaded system.

Scenario:

Three client applications were made to request for a 1.3 MB file at the same time to a single threaded server application. The second and the third client applications could only receive files after the first client application finished its transaction. The overall time taken for the server to respond to all the three clients was 8.27 seconds. However, with a multi-threaded server, it took just 3.18 seconds to do the same task. The performance of the multi-threaded server was a little better with persistent connections in the case where the client was sequentially requesting for files. Had it been a case where the client was requesting for multiple files simultaneously, even better performance could be seen with multi-threading.

Client and server using UDP:

The time taken to receive a response in a UDP server and client application was much faster than TCP. In one of the test runs conducted, the server was able to push contents in 0.5 seconds to the client. But the data transfer using UDP is not reliable like TCP as it is always prone to packet loss. However, I did not experience any packet loss as the application was run on the silo server having 100 MBPS switched Ethernet with very low congestion.