CS252 : Lab-6

190050059, 190050118, 190050121

—

Contents of this .tar.gz file

1. client.c : sender side code
2. sender.c : receiver side code
3. recv.txt: output file
4. send.txt : input file
5. script.sh: Bash script for automation
6. Plotter.py : Python script to plot graphs
7. textfile.txt : All the results from each run (unprocessed)
8. results.txt : Processed results

(1st line after each delay and loss value is TCP_flavour_1, and 2nd line is TCP_flavour_2)

9. Figures : Contains all the plots
10. 100ms 1% : Contains the wireshark data for 100ms delay and 1% loss transmissions
11. 10ms 0.1% : Contains the wireshark data for 10ms delay and 0.1% loss transmissions

Steps to compile and run the programs

- Download the tar.gz and extract the files
- Use `chmod +rwx` to allow files to be executed as programs
- Run script.sh with sudo command : `sudo ./script.sh TCP_flavour_1 TCP_flavour_2`

Description of our code

Client.c (Usage: ./client localhost TCP_variant Server_Port)

- Number of arguments for this code must be 4, or else our code will directly return 1.
- We've used a string, i.e. char array called buffer of length 1024 to store the data that has to be sent to the server.
- If socket creation and binding do not go perfectly, we are exiting the code with an exit(1)/ exit(EXIT_FAILURE) command. Else, we are transmitting the file.
- We set the TCP variant using the setsockopt command.
- We used the read(..) function to read the file content from send.txt into the buffer.
- We used write to send the content from the buffer to the server.

- Finally, we output the time taken for the transmission and the number of bytes transmitted.

Server.c (Usage: ./server TCP_variant Server_Port)

- Number of arguments for this code must be 3, or else our code will directly return 1.
- We've used a string i.e. char array ,as mentioned above in client.c, of length 1024 to receive packets from the sender.
- If the socket creation or binding do not go well, we are exiting the code with an exit(1)/ exit(EXIT_FAILURE) command. Else we are trying to receive the packets from the sender in a while loop.
- We read the content sent by the client using the read command.
- If the content is corrupted, we terminate the code and exit with value 1.
- We use write to write the content from the buffer into recv.txt.
- We measure the number of bytes received in size.
- Finally, after successful transmission, we close the file pointers and the sockets and exit the code.

script.sh (Usage: ./script.sh TCP_variant_1 TCP_variant_2)

- This script automates the process of conducting 9*2*20 experiments.
- We start by compiling the server.c and client.c files.
- We make a directory for Figures.
- We set the MTU to 1500B using the ifconfig command.
- The find_port() function returns an unused port using the netstat command.
- We then loop over various delays and losses and run the transmission for 20 times for each of the variants.
- In each iteration, we set the respective loss and delay value using the tc command.
- We empty the recv.txt file, execute the C codes and then check if the input and output files are equal. That is, we use cmp -s send.txt recv.txt. We have a wait command before checking the above condition. We use this to allow server.txt to finish writing the output into the file.
- Note: If you want to speed up the code, remove this condition and just check if the number of bytes received by the server is the same as the number of bytes sent by the client. This will improve the script's performance tremendously.
- We generate 2 temporary files temp.txt and temp2.txt which store the output of the C files. We use this to measure the throughput in each iteration. The script automatically deletes these files after the execution.
- We use awk to perform some trivial calculations.
- The throughputs are written in textfile.txt in the format
    - Delay x loss y
    - Throughputs with the first TCP variant
    - Throughputs with the second TCP variant

- Finally, the plotting script is called to draw the graphs.

Plotter.py (Usage: python3 plotter.py)

- The script automatically calls this file but you can manually use it too. Make sure you have a complete textfile.txt before you run this. That is, ensure you have the values for all the 20*2*9 experiments.
- The code reads the contents into numpy arrays trivially using if else conditions.
- Then we use the numpy library to calculate the mean and the standard deviation.
- We use the matplotlib library to plot the scatterplots with error bars.