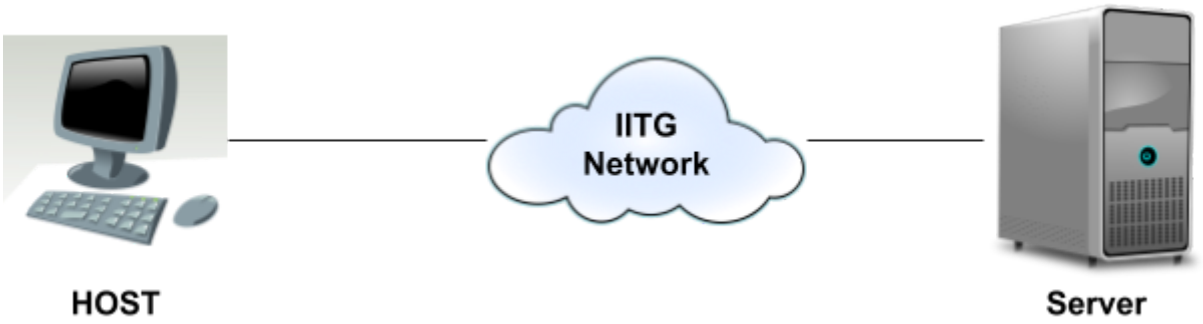
Assignment 3

Instructions:

1. Make sure that you read, understand, and follow these instructions carefully. Your cooperation will help to speed up the grading process.
2. Following are generic instructions. Make sure that you also check carefully and follow any specific instructions associated with particular questions.
3. In this assignment, you will explore the dynamics and challenges of time and memory management in operating systems.
4. Complete the assignment with the best of your own capabilities and follow the instructions as mentioned below.
    a. Create a folder named as your roll no.
    b. Create separate folders for each questions inside it and name them as Q-1, Q-2, etc.
    c. Each folder must contain:
        i. All your program files
        ii. Makefile to compile your programs if required.
        iii. Readme/report file to explain your assignment and answers to the questions.
    d. Compress the entire submission folder as a single file. After this step, the file name should be as your roll no. Example: 146101002.zip. (Find a sample submission in moodle). When done, submit the file in Moodle on or before the submission deadline.
5. Submission deadline: 08:00 PM, Wednesday 28th March, 2018 (Hard Deadline).

Ethical Guidelines (lab policy):

1. Deadlines: Deadlines should be met. Assignments submitted after their respective deadlines will not be considered for evaluation.
2. Cheating: You are expected to complete the assignment by yourself. Cases of unfair means and copying from another student will not be tolerated, even if you make cosmetic changes to them. If we suspect any form of cheating, we are compelled to award ZERO marks.
3. If you have problem meeting a deadline, it is suggested that you consult the instructor and not cheat.



1. Consider the scenario given in the figure. In this case consider your system as host/client. A Server (172.16.112.141) is hosted in our department. The server is already configured with two server applications: a. Python SimpleHTTPServer (port 8080) b. Iperf UDP server (port 5001). Perform the experiments as mentioned below.
   1.1. Use host as an UDP Client (with data generation rate = 20 Mbps) for UDP performance measurement using Iperf. The experiment should be executed for at-least 100s. Please check Iperf documentation for related commands.
   1.2. Using wget/curl as TCP client download a file from the following URL

http://172.16.112.141/cs558/sample.txt

Capture the packet traces using Wireshark during the experiments in your client system and answer the following questions.

a. What are the protocols being used during your experiments? Why are they being used?
b. What are the sizes of the TCP/UDP packets for both the experiments (1.1 and 1.2)? Are all the packets of same size?
c. What is the bandwidth for TCP? Find and plot the timing diagram of the TCP retransmission events and duplicate packet receiving.
d. Execute exp. 1.2 for variable UDP data generation rates (in Mbps) such as 1, 10, 20, 30, 40, 50 . Execute each of them for at-least 10 times.
   i. Plot the variation of UDP bandwidth with average and confidence interval with respect to data generation rates.
   ii. Plot the variation of UDP packets transmitted with average and confidence interval with respect to UDP bandwidth.
   iii. What are your observations from these plots?

[7]

---

2. **"tc** is used to configure Traffic Control in the Linux kernel"[1]. Use tc to insert additional delay of 500ms and bandwidth control limit of 1Mbps. Upon configuring your system, repeat the experiments mentioned in Q.1.

[3]

[1] https://linux.die.net/man/8/tc

3. Use a UDP client-server program[2] to implement TCP 3-way handshaking. Use the UDP payload bytes for implementing custom Sequence numbers and necessary TCP header fields.

[3]

3.1. Your implemented Wrapper TCP must incorporate **slow start** and ***fast retransmit*** policy.

[3+3]

3.2. Your implemented client and server application should communicate via loopback interface. Therefore, the underlying link between them has a high bandwidth with low latency. Such a type of links are called "fat-pipes". CUBIC TCP[3] is one of the TCP variant that works well in such cases. Implement CUBIC TCP to control congestion in your protocol.

[8]

3.3. Use tc to configure the interface delay from 100ms to 500ms in steps of 200ms and interface drop rate from 0% to 2% in steps of 0.5%. Use you protocol to transfer data in these configurations and plot the change of congestion window size vs time. From these plots, try to identify the minimum flow duration for which CUBIC reaches its steady state.

[3]

[2] https://www.cs.cmu.edu/afs/cs/academic/class/15213-f99/www/class26/

[3] https://tools.ietf.org/html/draft-rhee-tcp-cubic-00