#1 Play with the Traffic Shaper

The lab aims to play with the tc tool to emulate network settings (latency, bandwidth, and optional packet loss rate) on Linux.

Description

Platform Requirement

Please note that tc ***requires Linux kernel support*** to work correctly. Check the items listed here carefully to ensure you have a working environment.

1. It should work without problem if you run a Debian installation on a physical or virtual machine.
2. It should also work without problem if you use Docker Desktop on a Mac to run a Debian Linux container.
3. ***It DOES NOT WORK*** by default if you use Docker Desktop on ***Windows*** because the default WSL Linux kernel is compiled ***without*** queueing/scheduling features. Please follow the steps here to replace the default kernel first.

You have to ensure your runtime environment has the required commands, e.g., make, docker, and docker-compose. For example, you have to install make by yourself, e.g., sudo apt install make on Windows WSL (Debian/Ubuntu) or xcode-select --install on Mac OS.

1. Please download the package for this lab from here. Unpack the package in your preferred directory, switch into the **lab_shaper** directory, and then run **docker-compose up -d --build** to build everything.
2. **tc** is a tool to manipulate traffic control settings. One essential feature is the **qdisc** (the short name for *queueing discipline*) feature, which can be used to control network traffic sent out from a network interface. The command comes from the **iproute2** package. The required packages should be installed by default in our prepared environment.

The commands demonstrated in the following should be invoked in our docker runtime. To get the shell in the runtime, use the command:

docker exec -ti lab_shaper ash

3. To show **tc qdisc** rules, you can use the command **tc qdisc show**. You can try it in our runtime environment by using the command: **tc qdisc show**
4. In this lab, we use the **netem** (*Network Emulator*) qdisc to add network delay, set up packet loss rate, and/or limit bandwidth on an outbound network interface. To add *500us* network delay and limit the upper bound bandwidth at *100Mbps* on the local network interface ***lo***, we can use the command:

   tc qdisc add dev lo root netem delay 1ms rate 100Mbit

   You need ***root permission*** to change configurations when using the tc command, which should be available by default in our docker runtime.

5. You can try to set up a rule as mentioned above, use the ping command to **ping** localhost, and verify if it works as expected.
   ping -c 3 localhost
6. To remove the **netem** rule from the ***lo*** interface, use the command:
   tc qdisc del dev lo root netem

Demonstration

1. Follow the steps in the *Description* section to ensure that your tc works correctly in the runtime environment.
2. [50 pts] Run **/scripts/testping.py** and verify whether the latency reported from ping command matches the tc configuration. The testping.py script can be viewed here. The reported time values from the ping command should be twice the latency configured by tc.
3. [50 pts] Run **/scripts/testperf.py** and verify whether the reported from iperf3 command matches the tc configuration. The testperf.py script can be viewed here. The reported throughput should be less than the bandwidth limitation configured by tc
   We vary the throughput settings between 100Mbps and 1000Mbps. However, the reported throughput number might be ***much*** less than the expected one, depending on your hardware spec. Our experiences show that commodity hardware should be able to achieve a throughput of 500Mbps.

#2 How *Accurate* Can You Measure?

Please ensure you have a working Linux kernel that supports traffic queueing/scheduling, as used in Part #1.

This lab aims to implement a self-defined network protocol (either in TCP or UDP) to determine the network latency and bandwidth between a client and a server. You have to implement the client and the server by yourself.

Description

1. This lab shares the same package used in Part #1. If you have already completed Part #1, your runtime is ready.
2. Implement a client-server pair to measure the network latency and throughput between the client and the server. The communication between the client and the server should be done throught the **lo** interface, i.e., using the IP address 127.0.0.1.
3. We don’t have any protocol specification in this lab. Please define your own. However, the measured result output from your client must meet the format:
   # RESULTS: delay = {number} ms, bandwidth = {number} Mbps\n
   See the sample codes in dist/client.c and dist/server.c. Note that the current server does nothing, and the current client simply prints out

a *fixed/guessed* output.

4. Please implement all your codes in the dist directory, and build the codes using the command make all (or simply make) in the **lab_shaper** directory. We will compile your codes based on the scripts in **dist/Makefile** in a docker runtime. The client and the server executables must be linked as **static** binaries.
5. To test your programs, use the command **make testimpl**. The test script runs your server (only once), generates several randomized bandwidth-delay settings, and then runs your client for each of the settings. The testimpl.py script can be viewed here.

Demonstration

1. [20 pts] Ensure both make and make testimpl work as a charm.
   You have to implement your server and client in C or C++. The server will be invoked only once, and the client will be invoked for each test case.
2. [70 pts] The numbers of your measured results are visually similar to the expected output (Expected range: delay +/-10ms, bandwidth +/-30Mbps).
   For the latency, please measure the one-way latency. You may consider the gettimeofday(2) fucntion.
3. [10 pts] A ranking score based on the precision of your measurement. The score will be given based on the ranking of your implementation among all students in the class. There will be only four ranks. The top-ranked, second-ranked, third-ranked, and bottom-ranked implementations get 10pts, 7.5pts, 5pts, and 2.5pts, respectively. The rank will be sorted based on (1) measured latency, and then (2) measured bandwidth.