INP111 Lab06 Week #9 (2022-11-10)

Date: 2022-11-10

#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 Ubuntu 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 Ubuntu 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 (https://md.zoolab.org/s/eHduEC62B) to replace the default kernel first.

1. `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.

Please install the package first. It should be installed by default in our prepared environment.

2. To show `tc qdisc` rules, you can use the command `tc qdisc show`.

3. 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.

4. 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.

5. You can install the `iperf3` package and use it to measure the throughput configured for the selected interface. `iperf3` is a tool used to measure throughput between the client and the server. Suppose we use TCP port 9997 to run the test. You have to run the server first using the command `iperf3 -s -p 9997`.
   You can then run the client on the same machine using the command:
   `iperf3 -c localhost -p 9997` and observe the numbers reported from `iperf3`.

6. To remove the `netem` rule from the ***lo*** interface, use the command:
   `tc qdisc del dev lo root netem`.

Demo

1. Follow the steps in the *Description* section to ensure that your `tc` works correctly in your environment.

2. [50 pts] If the latency reported from `ping` command matches the `tc` configuration. You may download and run our test script from here (testping.sh) (https://inp111.zoolab.org/lab06.1/testping.sh.txt). The reported TTL values from the `ping` command should be twice the latency configured by `tc`.

3. [50 pts] If the throughput reported from the `iperf3` server and client matches the `tc` configuration. You have to run the iperf3 server at port 9997 manually. You may download and run our test script from here (testperf.sh) (https://inp111.zoolab.org/lab06.1/testperf.sh.txt). The reported throughput should be less than the bandwidth limitation configured by `tc`

#2 How *Fast* Can You Deliver?

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

This lab aims to implement a TCP server and a TCP client to determine the maximum throughput of a network link.

Description

1. Implement a TCP server ✓ that accepts one argument (the port number) and binds two ports (port and port+1). The former (port) is a command channel used to handle client commands, and the latter (port+1) is a data sink used for receiving data from connected sockets without replies.
   ↳ receive
   ↳ listen.
2. The server command channel has to support the following case-sensitive commands. The commands are sent in plain text, and each sent command must be ended with a newline `\n` character.
   - **/reset**: The server maintains a **counter** to count how many bytes have been received from ***all*** the data sink connections. This command resets the value of the **counter** to zero. The response format of this command is `<time> RESET <counter-value-before-reset>\n`. For example, `1665376871.854357 RESET 1139146752`.
   - **/ping**: This command checks the server's liveness. The response format of this command is `<time> PONG\n`. For example, `1665376954.021821 PONG`.
   - **/report**: This command is used to get the current **counter** value on the server. The response format of this command is `<time> REPORT <counter-value> <elapsed-time>s <measured-megabits-per-second>Mbps\n`. Note that the ***elapsed-time*** is the time (in seconds) between the current time and the last time invoking the **/reset** command. The ***measured-megabits-per-second*** is simply obtained by $(8.0 * \text{counter}/1,000,000.0/\text{elapsed-time})$. For example, `1665377140.485579 REPORT 0 268.631222s 0.000000Mbps`.
   - **/clients**: This command reports how many data sink connections are currently served by the server. The response format of this command is `<time> CLIENTS <number-of-connected-data-sink-connections>\n`. For example, `1665377742.371157 CLIENTS 5`.
3. The data sink accepts any incoming TCP connection requests. Once connected, it receives data from the connected clients, adds how many bytes are received to the **counter**, and drops the received data.

4. Implement a TCP client to interact with your implemented TCP server. A typical procedure for your client should be:
    - Connect to the server's command channel.
    - Create some connections to the server's data sink channel.
    - Send **/reset** command.
    - Keep sending data to the server using the data sink connections until a user terminates the client.
    - On receipt of the termination signal, the client closes all data sink connections and ensures that all the data sink connections are closed.
    - Send **/report** commands to the server and show the measured throughput.

Demonstration

1. [10 pts] Run your server and ensure it binds on both the TCP port 9998 (the command channel) and port 9999 (the data sink channel).
2. [40 pts] Play with your server manually using the `nc` command. Each of the four commands (`/ping` , `/reset` , `/clients` , and `/report`) is worth 10 pts. A list of sample commands used to test your server are listed below for your reference.

```
echo '/ping' | timeout 1 nc localhost 9998
echo '/reset' | timeout 1 nc localhost 9998
(timeout 10 nc localhost 9999 &)
(timeout 10 nc localhost 9999 &)
(timeout 10 nc localhost 9999 &)
echo '/clients' | timeout 1 nc localhost 9998
echo 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA' | timeout 1 nc localhost 9999
echo 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA' | timeout 1 nc localhost 9999
echo 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA' | timeout 1 nc localhost 9999
sleep 1
echo '/report' | timeout 1 nc localhost 9998
```

3. [50pts] Test your server and client with our sample script available here (test.sh) (https://inp111.zoolab.org/lab06.2/test.sh.txt). You have to run your server manually before running the testing script. Check the throughput reported from your implementation.
    - You get 50 pts if ***all*** the reported throughput is higher than 85% of the configured bandwidth limitation.

- You get 30 pts if ***all*** the reported throughput is higher than 70% of the configured bandwidth limitation.
- You get 20 pts if ***all*** the reported throughput is higher than 55% of the configured bandwidth limitation.
- You get 10 pts if ***all*** the reported throughput is higher than 40% of the configured bandwidth limitation.