



UNIVERSITY OF
SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 10: Measuring TCP Fairness

Document Version: **06-14-2019**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Fairness concepts.....	3
1.1 TCP bandwidth allocation	3
1.2 TCP fairness index calculation	5
2 Lab topology.....	6
2.1 Starting host h1 and host h2	8
2.2 Emulating 10 Gbps high-latency WAN	8
2.3 Testing connection	10
3 Calculating fairness among parallel flows	13
4 Calculating fairness among several hosts with the same congestion control algorithm.....	14
5 Calculating fairness among hosts with different congestion control algorithms.....	17
References	19

Overview

This lab introduces TCP fairness in Wide Area Networks (WAN) and explains how competing TCP connections converge to fairness. The lab describes how to calculate the TCP fairness index, a metric that quantifies how fair the aggregate connection is divided between active connections. Finally, the lab conducts throughput tests in an emulated high-latency network and derives the fairness index.

Objectives

By the end of this lab, students should be able to:

1. Define TCP fairness.
2. Calculate TCP fairness index.
3. Emulate a WAN and calculating fairness index among parallel streams.
4. Emulate a WAN and calculating fairness index among competing TCP connections.

Lab settings

The information in Table 1 provides the credentials of the machine containing Mininet.

Table 1. Credentials to access Client1 machine.

Device	Account	Password
Client1	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Fairness concepts.
2. Section 2: Lab topology.
3. Section 3: Calculating fairness among parallel flows.
4. Section 4: Calculating fairness index with different congestion control algorithms.

1 Fairness concepts

1.1 TCP bandwidth allocation

Many networks do not use any bandwidth reservation mechanism for TCP flows passing through a router. Instead, routers simply make forwarding decisions based on the destination field of the IP header. As a result, flows may attempt to use as much bandwidth as possible. In this situation, it is the TCP congestion control algorithm that allocates bandwidth to the competing flows.

Consider the scenario where two TCP flows share a bottleneck link with bandwidth capacity R , as illustrated in Figure 1. Assume that the two senders are in equal conditions (round-trip time, maximum segment size, configuration parameters) and that they use the same congestion control algorithm. Furthermore, assume that the two flows are in steady state and that the congestion control algorithm operates according to the additive increase multiplicative decrease (AIMD) rule¹. A fair bandwidth allocation would result in a bandwidth partition of $R/2$ to each flow.

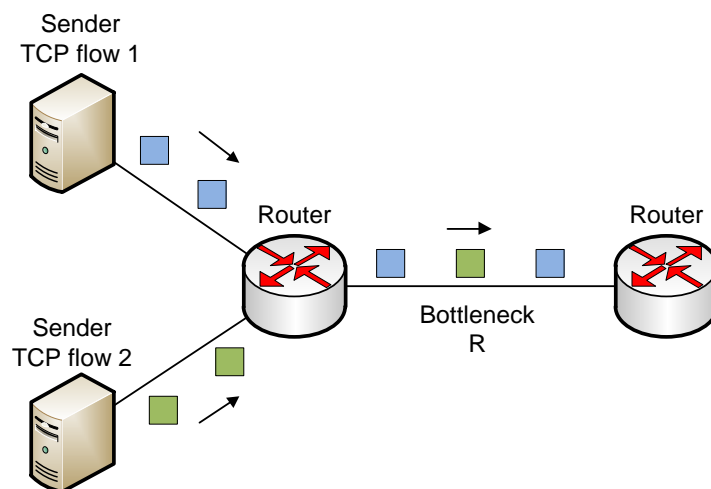


Figure 1. Two TCP flows that share a bottleneck link of capacity R .

Figure 2 shows the bandwidth allocation region for the two flows¹. The bandwidth allocation to flow 1 is on x-axis and to flow 2 is on the y-axis. If TCP is to share the bottleneck bandwidth equally between the two flows, then the bandwidth will fall along the fairness line emanating from the origin. Note that the origin $(0, 0)$ is a fair but undesirable solution. When the allocations sum to 100% of the bottleneck capacity, the allocation is efficient. This is shown by the efficiency line. Note that potential *efficient* solutions include points A $(R, 0)$ and points B $(0, R)$. On point A, flow 1 receives 100% of the capacity, and on point B flow 2 receives 100% of the capacity. Clearly, these solutions are not desirable, as they lead to starvation and unfairness.

Assume that the sending rates of senders 1 and 2 at a given time are indicated by point p_1 . As the amount of aggregate bandwidth jointly consumed by the two flows is less than R , no loss will occur, and TCP will gently increase the bandwidth allocation (this process is called additive increase phase). Eventually, the bandwidth jointly consumed by the two connections will be greater than R , and a packet loss will occur at a point, say p_2 . TCP reacts to a packet loss by aggressively decreasing the sending rate by half (this operation is called multiplicate decrease). The resulting bandwidth allocations are realized at point p_3 . Since the joint bandwidth use is less than R at point p_3 , TCP will again increase the allocation to flows 1 and 2. Eventually, the TCP additive increase phase will lead to the

operating point p_4 , where a loss will again occur, and the two flows again will see a decrease in the bandwidth allocation, and so on. The bandwidth realized by the two flows eventually will fluctuate along the fairness line, near the optimal operating point Opt ($R/2, R/2$). Chiu and Jain¹ describe the reasons of why TCP converges to a fair and efficient allocation. This convergence occurs independently of the starting point^{2, 3}.

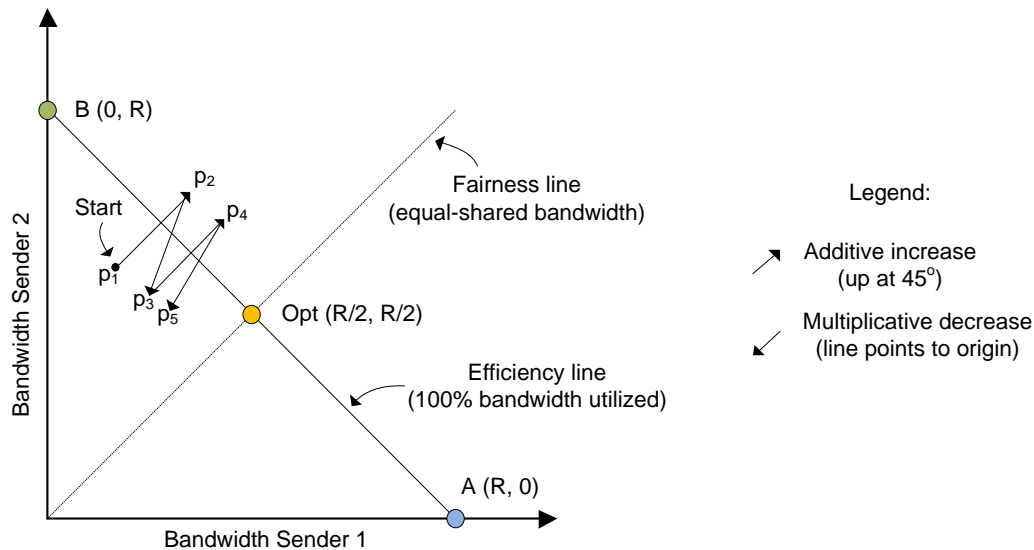


Figure 2. Bandwidth allocation region realized by two competing TCP flows.

1.2 TCP fairness index calculation

A useful index to quantify fairness is Jain's index⁴. The index has the following properties:

1. *Population size independence*: the index is applicable to any number of flows.
2. *Scale and metric independence*: the index is independent of scale, i.e., the unit of measurement does not matter.
3. *Boundedness*: the index is bounded between 0 and 1. A totally fair system has an index of 1 and a totally unfair system has an index of 0.
4. *Continuity*: the index is continuous. Any change in allocation is reflected in the fairness index.

Jain's fairness index is given by the following equation:

$$I = \frac{(\sum_{i=1}^n T_i)^2}{n \sum_{i=1}^n T_i^2}$$

where

- I is the fairness index, with values between 0 and 1.
- n is the total number of flows.
- T_1, T_2, \dots, T_n are the measured throughput of individual flows.

As an example of fairness index calculation, consider the three flows shown in Figure 3. Given the bottleneck capacity of 9 Gbps, assume that the bandwidth allocations for flows 1, 2, and 3 are 5 Gbps, 3 Gbps, and 1 Gbps. The fairness index for this allocation is:

$$I = \frac{(\sum_{i=1}^3 T_i)^2}{3 \sum_{i=1}^3 T_i^2} = \frac{(5 \cdot 10^9 + 3 \cdot 10^9 + 1 \cdot 10^9)^2}{3 \cdot ((5 \cdot 10^9)^2 + (3 \cdot 10^9)^2 + (1 \cdot 10^9)^2)} = 0.77$$

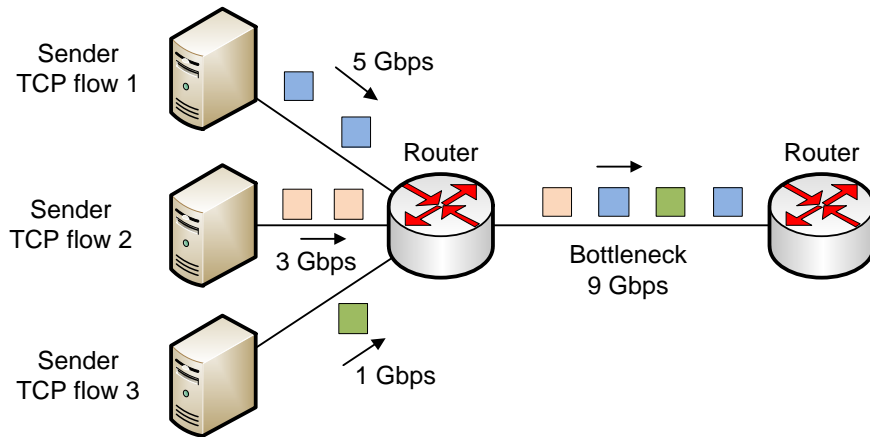


Figure 3. Three TCP flows that share a bottleneck link of capacity 9 Gbps.

Note that by property 2 (scale and metric independence), the fairness index of the above example is the same as that of an allocation of 5 Mbps, 3 Mbps, and 1 Mbps (or more generally, an allocation of 5, 3, and 1 units). Also, note that an optimal fair allocation of 3 Gbps to each flow would produce a fairness index of 1.

2 Lab topology

Let's get started with creating a simple Mininet topology using MiniEdit. The topology uses 10.0.0.0/8 which is the default network assigned by Mininet.

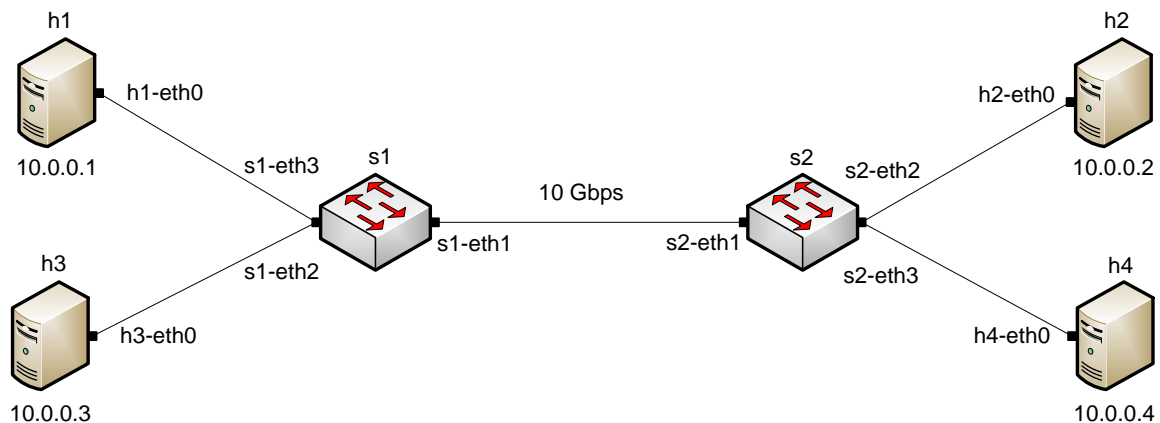


Figure 4. Lab topology

The above topology uses 10.0.0.0/8 which is the default network assigned by Mininet.

Step 1. A shortcut to MiniEdit is located on the machine's Desktop. Start MiniEdit by clicking on MiniEdit's shortcut. When prompted for a password, type `password`.

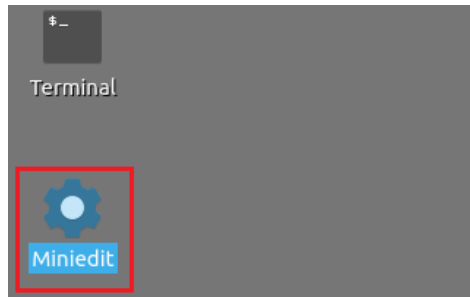


Figure 5. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 10.mn* topology file and click on *Open*.

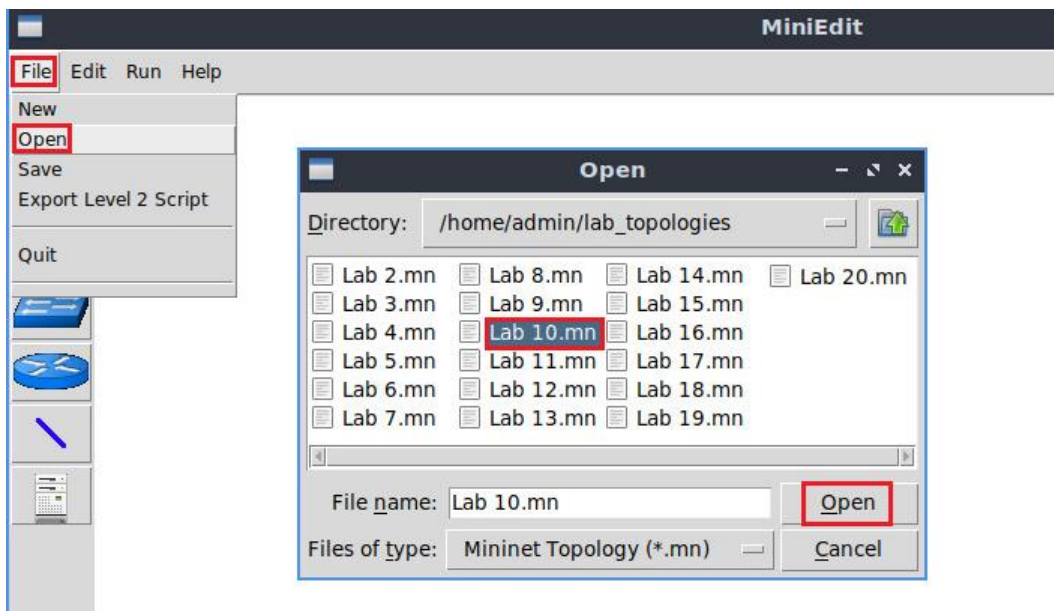


Figure 6. MiniEdit's *Open* dialog.

Step 3. Before starting the measurements between host h1 and host h2, the network must be started. Click on the *Run* button located at the bottom left of MiniEdit's window to start the emulation.

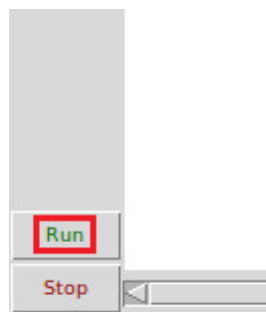


Figure 7. Running the emulation.

The above topology uses 10.0.0.0/8 which is the default network assigned by Mininet.

2.1 Starting host h1 and host h2

Step 1. Hold the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

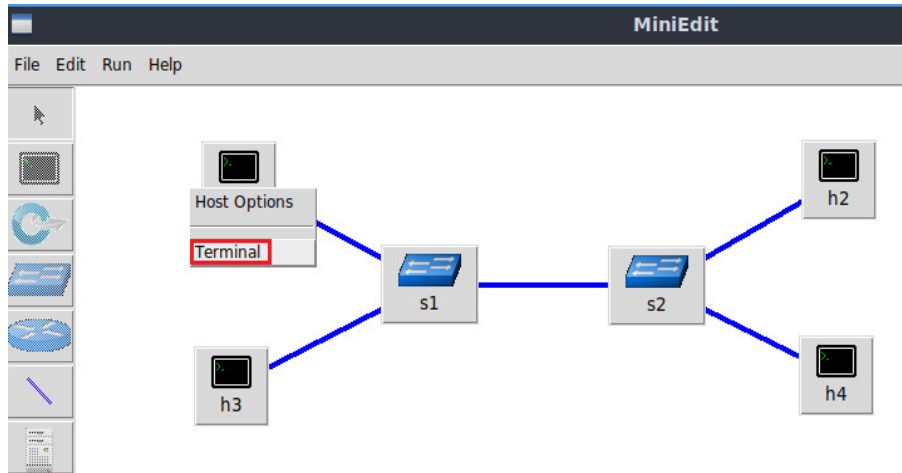


Figure 8. Opening a terminal on host h1.

Step 2. Apply the same steps on host h2 and open its *Terminal*.

Step 3. Test connectivity between the end-hosts using the `ping` command. On host h1, type the command `ping 10.0.0.2`. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.

```

"Host: h1"
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.33 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.044 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 91ms
rtt min/avg/max/mdev = 0.042/0.260/1.327/0.477 ms
root@admin-pc:~#

```

Figure 9. Connectivity test using `ping` command.

Figure 9 indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

2.2 Emulating 10 Gbps high-latency WAN

This section emulates a high-latency WAN. We will first emulate 20ms delay between switch S1 and switch S2 and measure the throughput. Then, we will set the bandwidth between host h1 and host h2 to 10 Gbps.

Step 1. Launch a Linux terminal by holding the `Ctrl+Alt+T` keys or by clicking on the Linux terminal icon.



Figure 10. Shortcut to open a Linux terminal.

The Linux terminal is a program that opens a window and permits you to interact with a Command-Line Interface (CLI). A CLI is a program that takes commands from the keyboard and sends them to the operating system for execution.

Step 2. In the terminal, type the command below. When prompted for a password, type `password` and hit *Enter*. This command introduces 20ms delay on switch S1's `s1-eth1` interface.

```
sudo tc qdisc add dev s1-eth1 root handle 1: netem delay 20ms
```

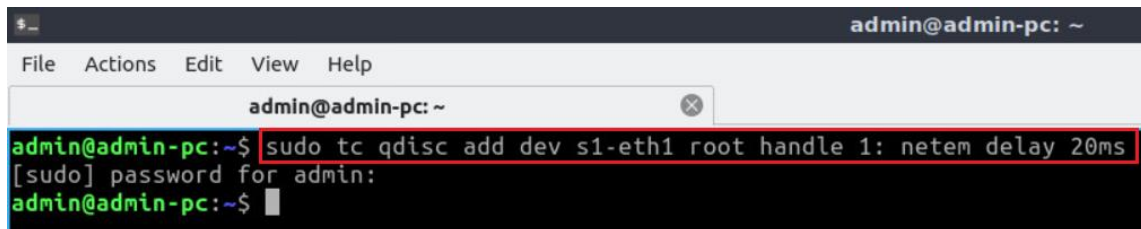


Figure 11. Adding delay of 20ms to switch S1's `s1-eth1` interface.

Step 3. Modify the bandwidth of the link connecting the switch S1 and switch S2: on the same terminal, type the command below. This command sets the bandwidth to 10 Gbps on switch S1's `s1-eth2` interface. The `tbfb` parameters are the following:

- `rate`: 10gbit
- `burst`: 5,000,000
- `limit`: 15,000,000

```
sudo tc qdisc add dev s1-eth1 parent 1: handle 2: tbf rate 10gbit burst 5000000 limit 15000000
```

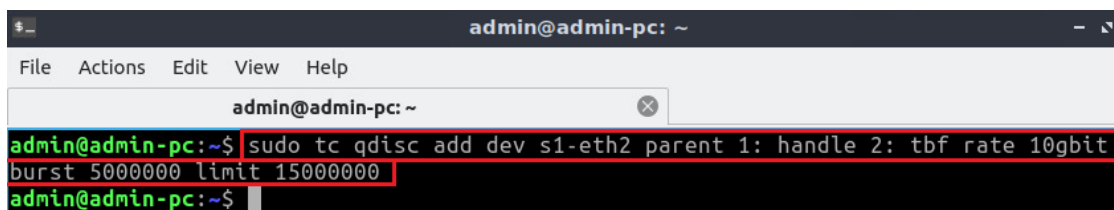
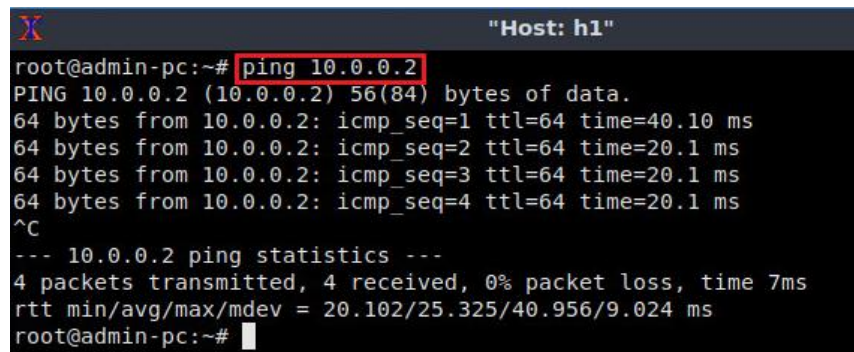


Figure 12. Limiting the bandwidth to 10 Gbps on switch S1's `s1-eth1` interface.

2.3 Testing connection

To test connectivity, you can use the command `ping`.

Step 1. On the terminal of host h1, type `ping 10.0.0.2`. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test. Host h1 (10.0.0.1) sent four packets to host h2 (10.0.0.2), successfully receiving responses back.



```

Host: h1
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=40.10 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.102/25.325/40.956/9.024 ms
root@admin-pc:~#
  
```

Figure 13. Output of `ping 10.0.0.2` command.

The result above indicates that all four packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 20.102, 25.325, 40.956, and 9.024 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

Step 2. On the terminal of host h2, type `ping 10.0.0.1`. The ping output in this test should be relatively close to the results of the test initiated by host h1 in Step 1. To stop the test, press `Ctrl+c`.

Step 3. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```



```

Host: h2
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----
  
```

Figure 14. Starting iPerf3 server on host h2.

Step 4. Launch iPerf3 in client mode on host h1's terminal.

```
iperf3 -c 10.0.0.2
```

```

Host: h1
root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 19] local 10.0.0.1 port 60040 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer             Bitrate             Retr  Cwnd
[ 19]  0.00-1.00   sec    320 MBytes         2.68 Gbits/sec      17   16.4 MBytes
[ 19]  1.00-2.00   sec    372 MBytes         3.12 Gbits/sec       0   16.4 MBytes
[ 19]  2.00-3.00   sec    388 MBytes         3.25 Gbits/sec       0   16.4 MBytes
[ 19]  3.00-4.00   sec    372 MBytes         3.13 Gbits/sec       0   16.4 MBytes
[ 19]  4.00-5.00   sec    395 MBytes         3.31 Gbits/sec       0   16.4 MBytes
[ 19]  5.00-6.00   sec    392 MBytes         3.29 Gbits/sec       0   16.4 MBytes
[ 19]  6.00-7.00   sec    391 MBytes         3.28 Gbits/sec       0   16.4 MBytes
[ 19]  7.00-8.00   sec    394 MBytes         3.30 Gbits/sec       0   16.4 MBytes
[ 19]  8.00-9.00   sec    394 MBytes         3.30 Gbits/sec       0   16.4 MBytes
[ 19]  9.00-10.00  sec    391 MBytes         3.28 Gbits/sec       0   16.4 MBytes
-----
[ ID] Interval           Transfer             Bitrate             Retr
[ 19]  0.00-10.00  sec   3.72 GBytes         3.20 Gbits/sec      17
[ 19]  0.00-10.04  sec   3.72 GBytes         3.18 Gbits/sec
iperf Done.
root@admin-pc:~#

```

Figure 15. Running iPerf3 client on host h1.

Although the link was configured to 10 Gbps, the test results show that the achieved throughput is 3.20 Gbps. This is because the TCP buffer size was not modified at this point.

Step 5. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

Step 6. To change the current receive-window size value(s), we calculate the Bandwidth-Delay Product by performing the following calculation:

$$BW = 10,000,000,000 \text{ bits/second}$$

$$RTT = 0.02 \text{ seconds}$$

$$BDP = 10,000,000,000 * 0.02 = 200,000,000 \text{ bits} \\ = 25,000,000 \text{ bytes} \approx 25 \text{ Mbytes}$$

The send and receive buffer sizes should be set to $2 \cdot BDP$. We will use the 25 Mbytes value for the BDP instead of 25,000,000 bytes.

$$1 \text{ Mbyte} = 1024^2 \text{ bytes}$$

$$BDP = 25 \text{ Mbytes} = 25 * 1024^2 \text{ bytes} = 26,214,400 \text{ bytes}$$

$$2 \cdot BDP = 2 \cdot 26,214,400 \text{ bytes} = 52,428,800 \text{ bytes}$$

Now, we have calculated the maximum value of the TCP sending and receiving buffer size. In order to apply the new values, on host h1's terminal type the command showed down below. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
```

```

X                                     "Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~# █

```

Figure 16. Receive window change in `sysctl`.

Step 7. To change the current send-window size value(s), use the following command on host h1's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
```

```

X                                     "Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~# █

```

Figure 17. Send window change in `sysctl`.

Next, the same commands must be configured on host h2.

Step 8. To change the current receive-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
```

```

X                                     "Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~# █

```

Figure 18. Receive window change in `sysctl`.

Step 9. To change the current send-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
```

```

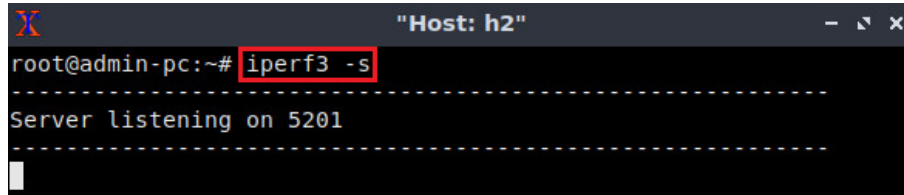
X                                     "Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~# █

```

Figure 19. Send window change in `sysctl`.

Step 10. The user can now verify the rate limit configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal:

```
iperf3 -s
```



```

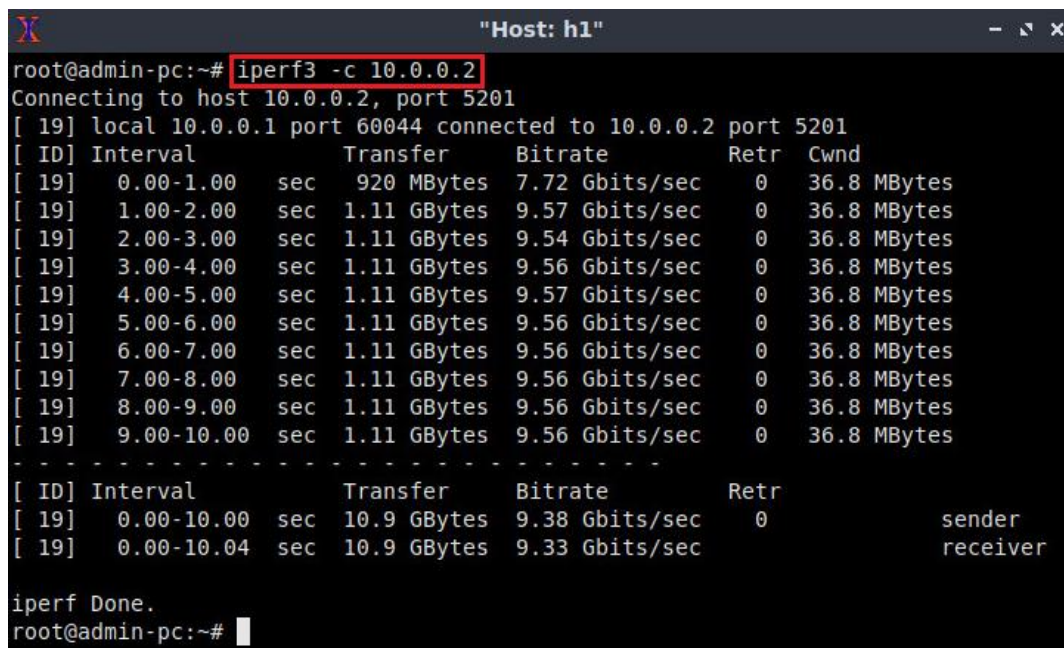
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----

```

Figure 20. Host h2 running iPerf3 as server.

Step 11. Now to launch iPerf3 in client mode again by running the command `iperf3 -c 10.0.0.2` in host h1's terminal:

```
iperf3 -c 10.0.0.2
```



```

root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 19] local 10.0.0.1 port 60044 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate      Retr  Cwnd
[ 19]  0.00-1.00       sec   920 MBytes  7.72 Gbits/sec    0   36.8 MBytes
[ 19]  1.00-2.00       sec   1.11 GBytes  9.57 Gbits/sec    0   36.8 MBytes
[ 19]  2.00-3.00       sec   1.11 GBytes  9.54 Gbits/sec    0   36.8 MBytes
[ 19]  3.00-4.00       sec   1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]  4.00-5.00       sec   1.11 GBytes  9.57 Gbits/sec    0   36.8 MBytes
[ 19]  5.00-6.00       sec   1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]  6.00-7.00       sec   1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]  7.00-8.00       sec   1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]  8.00-9.00       sec   1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]  9.00-10.00      sec   1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
-----
[ ID] Interval           Transfer     Bitrate      Retr
[ 19]  0.00-10.00      sec  10.9 GBytes  9.38 Gbits/sec    0      sender
[ 19]  0.00-10.04      sec  10.9 GBytes  9.33 Gbits/sec                receiver

iperf Done.
root@admin-pc:~#

```

Figure 21. iPerf3 throughput test.

Note the measured throughput now is approximately 9.38 Gbps, which is close to the value assigned in our `tbw` rule (10 Gbps).

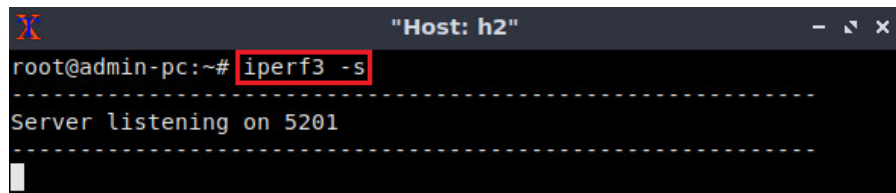
Step 12. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3 Calculating fairness among parallel flows

In this section, an iPerf3 test that includes several parallel streams is conducted, followed by the calculation of the fairness index.

Step 1. Now a test is conducted using parallel streams. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in Figure 22:

```
iperf3 -s
```

```

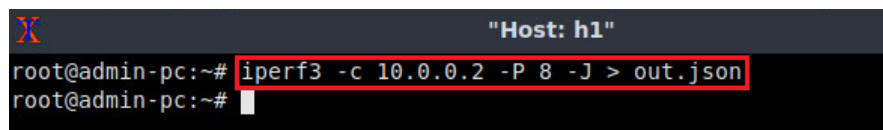
Host: h2
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----

```

Figure 22. Host h2 running iPerf3 as server.

Step 2. Now the iPerf3 client should be launched with the `-P` option specified to start parallel streams. The `-J` option is also specified to indicate that JSON output is desired, and the redirection operator `>` to store the output in a file. Run the following command in host h1's terminal as shown in Figure 23:

```
iperf3 -c 10.0.0.2 -P 8 -J > out.json
```



```

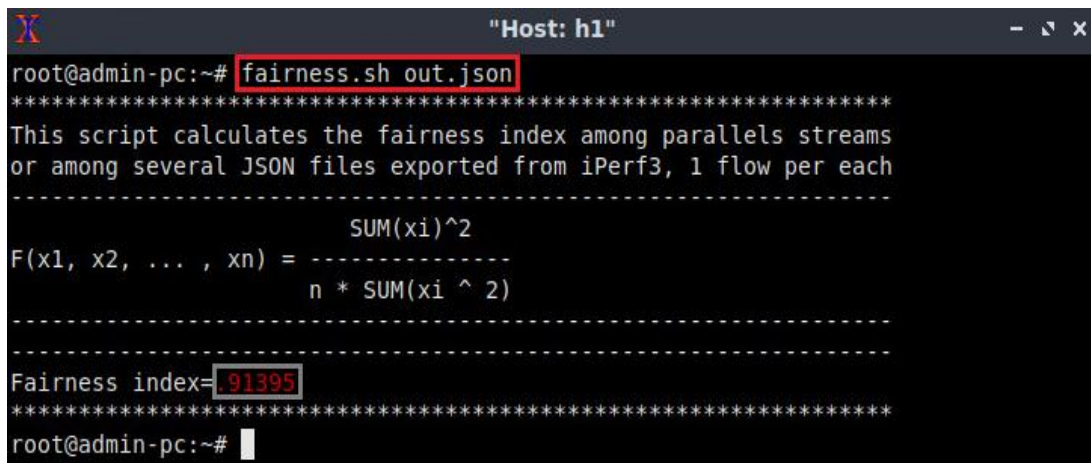
Host: h1
root@admin-pc:~# iperf3 -c 10.0.0.2 -P 8 -J > out.json
root@admin-pc:~#

```

Figure 23. Host h1 running iPerf3 as client with 8 parallel streams and saving output in file.

Step 3. The client includes a script called `fairness.sh`. Basically, this script accepts as input the JSON file exported by iPerf3 and calculates the fairness index. Run the following command to calculate the fairness index:

```
fairness.sh out.json
```



```

Host: h1
root@admin-pc:~# fairness.sh out.json
*****
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----
                SUM(xi)^2
F(x1, x2, ... , xn) = -----
                n * SUM(xi ^ 2)
-----

Fairness index=.91395
*****
root@admin-pc:~#

```

Figure 24. Calculating the fairness index between the parallel streams.

In the above test, the fairness index is .91395, or 91% fair. Note that this result may vary according to the result of your emulation test.

Step 4. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

4 Calculating fairness among several hosts with the same congestion control algorithm

In the previous section, we calculated the fairness index among several parallel streams, all initiated by a single host. In this section we calculate the fairness index among two transmitting devices. Specifically, an iPerf3 client is executed on host h1 and connected to host h2 (iPerf3 server); another iPerf3 client is executed on host h3 and connected to host h4 (iPerf3 server).

To calculate the fairness index, the transmitting hosts should initiate their transmissions simultaneously. Since it is difficult to start the clients at the same time, the client's machine provides a script that automates this process.

Step 1. Close the terminals of host h1 and host h2.

Step 2. Go to Mininet's terminal, i.e., the one launched when MiniEdit was started.

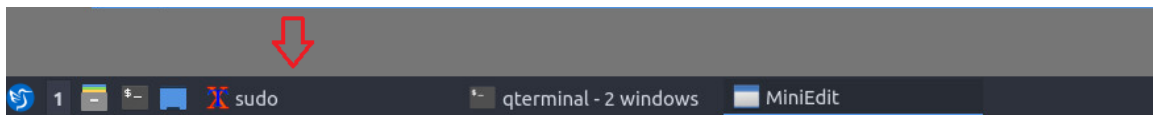


Figure 25. Opening Mininet's terminal.

```
sudo
[sudo] password for admin:
topo=None
Getting Hosts and Switches.
Getting Links.
*** Configuring hosts
h4 h1 h2 h3
**** Starting 0 controllers

**** Starting 2 switches
s1 s2
No NetFlow targets specified.
No sFlow targets specified.

NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exiting will prevent MiniEdit from quitting and will prevent you from starting the network again during this session.

*** Starting CLI:
mininet> |
```

Figure 26. Mininet's terminal.

Step 3. Issue the following command on Mininet's terminal as shown in the figure below.

```
source concurrent_same_algo
```

```

sudo
mininet> source concurrent_same_algo

Setting h1's congestion control algorithm to Reno:
--> sysctl -w net.ipv4.tcp_congestion_control=reno

Setting h3's congestion control algorithm to Reno:
--> sysctl -w net.ipv4.tcp_congestion_control=reno

Modifying TCP buffer size on all devices... (10Gbps, 20ms delay)
--> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
--> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'

h1 connected to h2. Transmitting for 120 seconds, please wait ...
h3 connected to h4. Transmitting for 120 seconds, please wait ...

*****
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----
                                SUM(xi)^2
F(x1, x2, ... , xn) = -----
                        n * SUM(xi ^ 2)

```

Figure 27. Running the tests simultaneously for 120 seconds. Both host h1 and host h3 are using Reno.

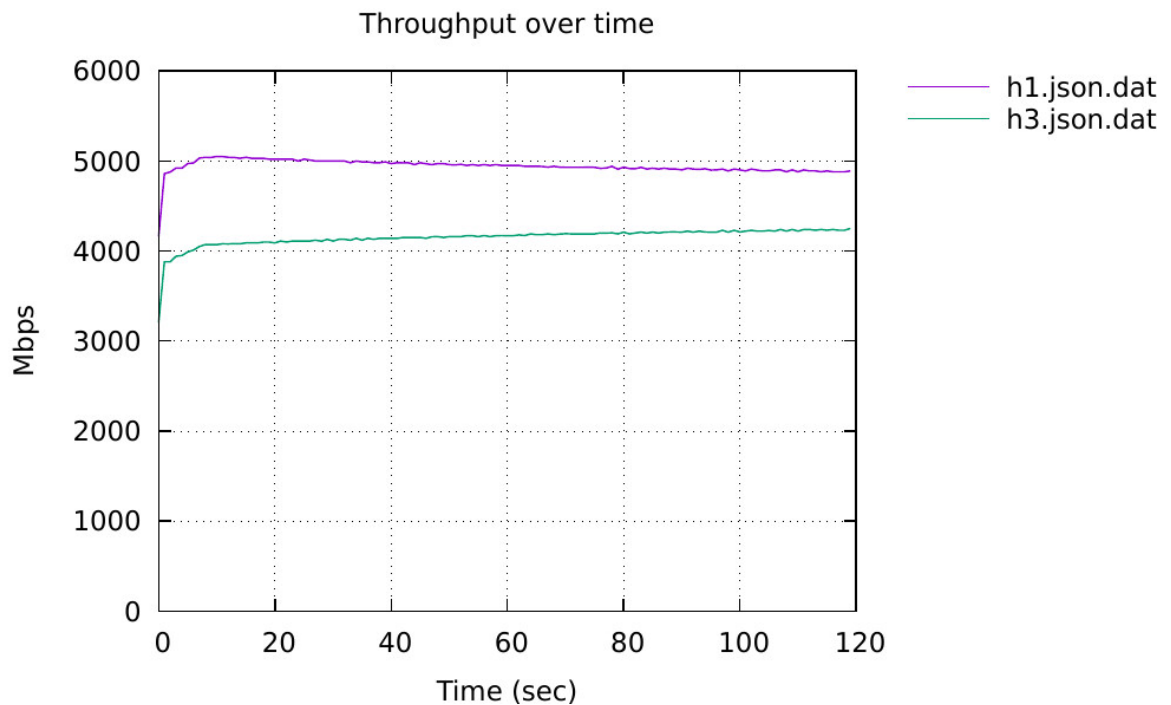


Figure 28. Throughput of host h1 and host h3.

The above graph shows that the throughput of host h1 is close to that of host h3. Therefore, the fairness index should be close to 1 (100%). Note that this result may vary according to the result of your emulation test.

Step 4. Close the graph window and go back to Mininet's terminal. The fairness index is displayed at the end as shown in the figure below.


```

sudo
--> sysctl -w net.ipv4.tcp_congestion_control=reno

Modifying TCP buffer size on all devices... (10Gbps, 20ms delay)
--> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
--> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'

h1 connected to h2. Transmitting for 120 seconds, please wait ...
h3 connected to h4. Transmitting for 120 seconds, please wait ...

*****
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----
                        SUM(xi)^2
F(x1, x2, ... , xn) = -----
                      n * SUM(xi ^ 2)
-----

Fairness index=.99595
*****

```

Figure 29. Calculated fairness index.

The above figure shows a fairness index of .99595. This value indicates that the bottleneck bandwidth was 99% fairly shared among host h1 and host h3. Note that this result may vary according to the result of your emulation test.

5 Calculating fairness among hosts with different congestion control algorithms

In the previous test, we calculated the fairness index while using the same congestion control algorithm (Reno). In this section we repeat the test, but with host h1 using Reno and host h3 using BBR.

Step 1. Go back to Mininet's terminal, i.e., the one launched when MiniEdit was started.

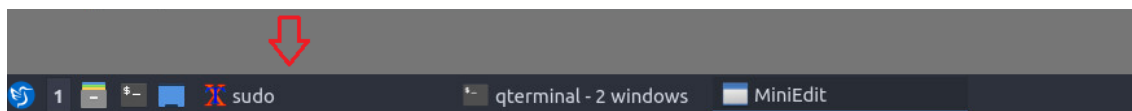


Figure 30. Opening Mininet's terminal.

Step 2. Issue the following command on Mininet's terminal as shown in the figure below.

```
source concurrent_diff_algo
```

```

sudo
mininet> source concurrent_diff_algo

Setting h1's congestion control algorithm to Reno:
--> sysctl -w net.ipv4.tcp_congestion_control=reno

Setting h3's congestion control algorithm to BBR:
--> sysctl -w net.ipv4.tcp_congestion_control=bbr

Modifying TCP buffer size on all devices... (10Gbps, 20ms delay)
--> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
--> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'

h1 connected to h2. Transmitting for 20 seconds, please wait ...

h3 connected to h4. Transmitting for 20 seconds, please wait ...

*****
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----
                                SUM(xi)^2
F(x1, x2, ... , xn) = -----
                        n * SUM(xi ^ 2)

```

Figure 31. Running the tests simultaneously for 20 seconds. Host h1 is using Reno while host h3 is using BBR.

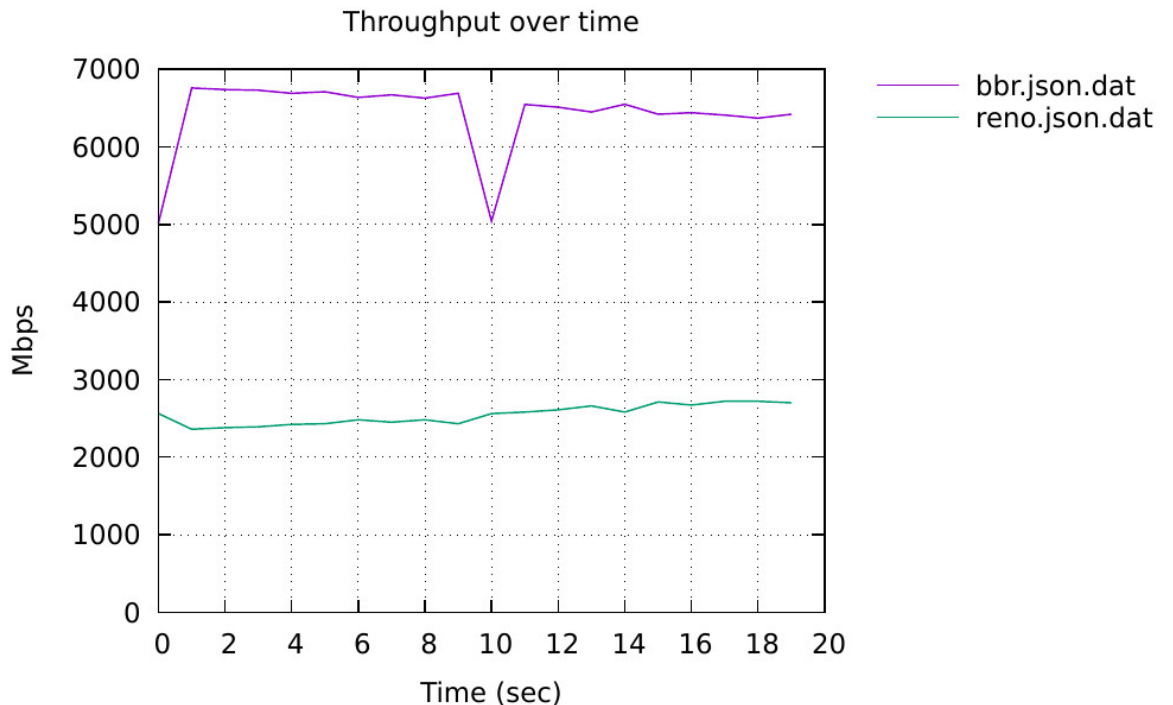


Figure 32. Throughput of host h1 and host h3.

The above graph shows that the device configured with BBR has a larger bandwidth allocation than that configured with Reno. Therefore, the fairness index will not be close to 1 (100%).

Step 3. Close the graph window and go back to Mininet's terminal. The fairness index is displayed at the end as shown in the figure below.

```

sudo
Setting h3's congestion control algorithm to BBR:
--> sysctl -w net.ipv4.tcp_congestion_control=bbr

Modifying TCP buffer size on all devices... (10Gbps, 20ms delay)
--> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
--> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'

h1 connected to h2. Transmitting for 20 seconds, please wait ...
h3 connected to h4. Transmitting for 20 seconds, please wait ...

*****
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----
                SUM(xi)^2
F(x1, x2, ... , xn) = -----
                n * SUM(xi ^ 2)
-----

Fairness index=.86036
*****

```

Figure 33. Calculated fairness index.

The above figure shows a fairness index of .86036 (~ 86%), which is very far from 100%. This value indicates that the bottleneck bandwidth was 86% fairly shared among host h1 and host h3. Note that this result may vary according to the result of your emulation test.

This concludes Lab 10. Stop the emulation and then exit out of MiniEdit.

References

1. D. Chiu, R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Journal of Computer Networks and ISDN Systems*, vol. 17, issue 1, pp. 1-14, Jun. 1989.
2. A. Tanenbaum, D. Wetherall, "Computer networks," 5th Edition, Prentice Hall, 2011.
3. J. Kurose, K. Ross, "Computer networking, a top-down approach," 7th Edition, Pearson, 2017.
4. R. Jain, D. Chiu, W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *DEC Research Report TR-301*, Sep. 1984.