



UNIVERSITY OF
SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 12: TCP Rate Control with Pacing

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 Introduction to TCP pacing	4
1.1 TCP pacing essentials	4
1.2 Use case: TCP pacing on a 100 Gbps network	5
1.3 Fair queueing details	6
2 Lab topology.....	7
2.1 Starting host h1 and host h2	8
2.2 Emulating 10 Gbps high-latency WAN	9
2.3 Testing connection	10
3 Enabling TCP pacing with tc and fq.....	13
4 Enabling TCP pacing from application	15
5 Concurrent transmission without pacing	17
6 Concurrent transmission with pacing.....	19
7 Parallel streams and without pacing	21
8 Parallel streams and with pacing	23
References	25

Overview

This lab introduces TCP pacing, which is a technique that evenly spaces out packets and minimizes traffic burstiness and packet losses. The focus in this lab is on Fair Queueing (FQ)-based pacing in high-latency Wide Area Networks (WANs). The lab describes the steps to conduct throughput tests that encompass TCP pacing and to compare the performance of TCP pacing against regular (non-paced) TCP.

Objectives

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

1. Define TCP pacing.
2. Understand FQ-based pacing.
3. Enable TCP pacing in Linux.
4. Compare the performance of paced TCP vs. non-paced TCP.
5. Understand pacing effect on parallel streams.
6. Emulate a WAN and calculate the coefficient of variation of flows.

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: Introduction to TCP pacing.
2. Section 2: Lab topology.
3. Section 3: Enabling TCP pacing with tc and fq.
4. Section 4: Enabling TCP pacing from application.
5. Section 5: Concurrent transmission without pacing.
6. Section 6: Concurrent transmission with pacing.
7. Section 7: Parallel streams and without pacing.
8. Section 8: Parallel streams and with pacing.

1 Introduction to TCP pacing

1.1 TCP pacing essentials

Data transmission can be bursty, resulting in packets being buffered at routers and switches and dropped at times. End devices can contribute to the problem by sending a large number of packets in a short period of time. If those packets were transmitted at a steady pace, the formation of queues could be reduced, avoiding packet losses.

TCP pacing is a technique by which a transmitter evenly spaces or paces packets at a pre-configured rate. It has been applied for years in enterprise networks¹, with mixed results. However, its recent application to data transfers in high-throughput high-latency networks and science demilitarized zones (Science DMZs) suggests that its use has several advantages². TCP pacing has also been applied to datacenter environments³.

The existing TCP congestion control algorithms, except for BBR⁴, indicate how much data is allowed for transmission. Those algorithms do not provide a time period over which that data should be transmitted and how the data should be spread to mitigate potential bursts. The rate, however, can be enforced by a packet scheduler such as a fair queue (FQ)⁵. The packet scheduler organizes the flow of packets of each TCP connection through the network stack to meet policy objectives. Some Linux distributions such as CentOS⁶ implement FQ scheduling in conjunction with TCP pacing^{4, 7}.

FQ is intended for locally generated traffic (e.g., a sender device, such as data transfer node (DTN) in Science DMZs). Figure 1 illustrates the operation of FQ pacing. Application 1 generates green packets, and application 2 generates blue packets. Each application opens a TCP connection. FQ paces each connection according to the desired rate, evenly spacing out packets within an application based on the desired rate. The periods T_1 and T_2 represent the time-space used for connections 1 and 2 respectively.

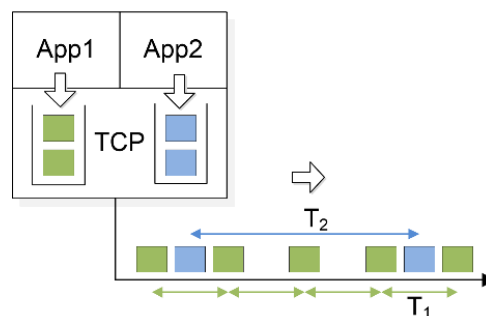


Figure 1. TCP pacing. Packets of applications 1 and 2 are evenly spaced by T_1 and T_2 time units.

TCP pacing reduces the typical TCP sawtooth behavior⁸ and is effective when there are rate mismatches along the path between the sender and the receiver. This is the case, for example, when the ingress port of a router has a capacity of 100 Gbps, and the egress port has a capacity of 10 Gbps. Because of the TCP congestion control mechanism, the sawtooth behavior always emerges. As TCP continues to increase the size of the congestion window, eventually the bottleneck link becomes full while the rest of the links

become underutilized. These mismatches produce a continuous circle of additive increases and multiplicative decreases⁸.

1.2 Use case: TCP pacing on a 100 Gbps network

With the increase of big data transfers across networks, network professionals have recently explored the impact of pacing on large flows⁸. Figure 2(a) shows the results of data transfers over the Energy Science Network (ESnet). ESnet is a high-performance network that carries science traffic for the U.S. Department of Energy. As of 2018, this network is transporting more than 200 petabytes per month. The path capacity and round-trip time (RTT) between end devices, referred to as DTNs, are 100 Gbps and 92 milliseconds respectively. Transfers use TCP Cubic congestion control algorithm⁹ without pacing and a maximum segment size (MSS) of 1,500 bytes. Four concurrent TCP connections are generated from a single source DTN to a single destination DTN. These four connections exhibit the typical sawtooth behavior¹⁰, which in part is attributed to the inability of switches to absorb traffic bursts. Figure 2(b) shows the behavior of TCP Cubic with FQ pacing. The pacing rate for the four TCP connections is approximately 20 Gbps (curves are overlapped at nearly 20 Gbps). The throughput is slightly lower than 20 Gbps per connection. However, notice how the sawtooth behavior is reduced and stable rates are obtained.

In general, TCP FQ pacing is also effective when there are rate mismatches along the path between the sender and the receiver. This is the case, for example, when the ingress port of a router has a capacity of 100 Gbps and the egress port has a capacity of 10 Gbps. As TCP increases the congestion window during the additive increase phase, eventually the bottleneck link becomes full while the rest of the links become underutilized. The mismatches produce a continuous circle of additive increases and multiplicative decreases, thus generating the sawtooth behavior.

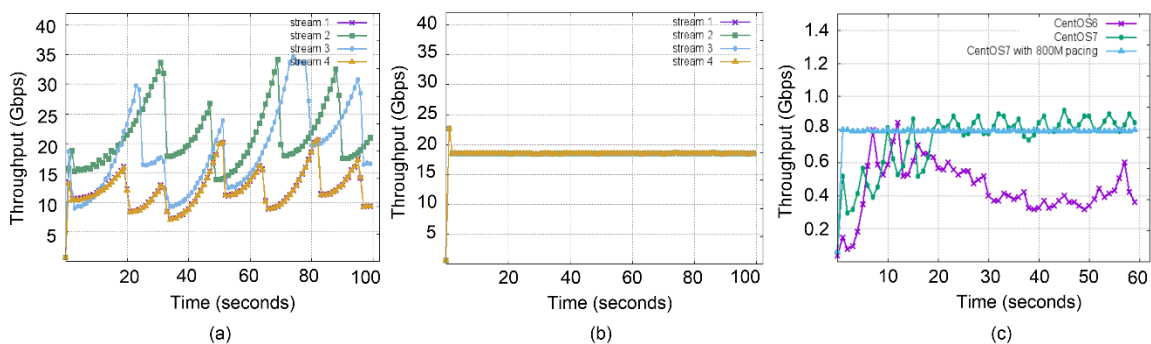


Figure 2. Impact of TCP pacing on throughput. (a) Data transfers of four parallel TCP connections across a 100 Gbps, 92 milliseconds RTT path. (b) The same data transfer as in (a) but using TCP pacing. (c) Data transfers between two DTNs connected by a path with a bottleneck link of 1 Gbps. The curves show the performance when the DTNs use different Linux operating systems (violet: CentOS 6; green: CentOS 7, and blue: CentOS7 with pacing). The results are reproduced from⁸.

Figure 2(c) shows the data transfer between two DTNs over ESnet. One DTN is in Amarillo, Texas, and the other DTN is in New York City. Although the WAN connecting the two sites has 100 Gbps capacity, one of the DTNs is attached to the network via a 1 Gbps network

interface card. Thus, the entirety of the path includes multiple 100 Gbps links and one bottleneck link of 1 Gbps. The figure shows three curves: the throughput when both DTNs are based on Linux CentOS⁶ Version 6 (violet), the throughput when DTNs are based on Linux CentOS Version 7 (green), and the throughput when DTNs are based on Linux CentOS Version 7 and packets are paced at 800 Mbps (blue). Note that pacing also leads to much more stable behaviors, almost removing the TCP sawtooth behavior.

1.3 Fair queueing details

In Linux-based systems, network traffic can be controlled by Queueing Disciplines (*qdisc*) used in conjunction with the Traffic Control (*tc*) tool. In this lab we focus on the most commonly used queueing discipline: FQ. In this queueing discipline, aggregate queues are used to associate token buckets in order to limit the transmission rate.

FQ performs flow separation to achieve pacing; it is designed to follow the requirements set by the TCP stack⁵. Generally, a flow is considered all packets pertaining to a particular socket. FQ uses the red-black tree data structure to index and track the state of single flows as shown in Figure 3(a)¹¹. A red-black tree is a binary search tree which ensures that no path in the tree is more than twice long as any other. This property ensures that tree operations have a logarithmic complexity. FQ achieves fairness through the Deficit Round Robin (DRR) algorithm¹², illustrated in Figure 3(b). The DRR is an algorithm that allows each flow passing through a network device to have a nearly perfect fairness and requires only a constant number of operations per packet. FQ uses the leaky bucket queue where transmitting timestamps (indexed on the read-black tree) are derived from the pacing rate specified by the user and the packet size. FQ is a non-work conserving scheduler, therefore, it can have idle scheduled resources even if there are jobs ready to be scheduled.

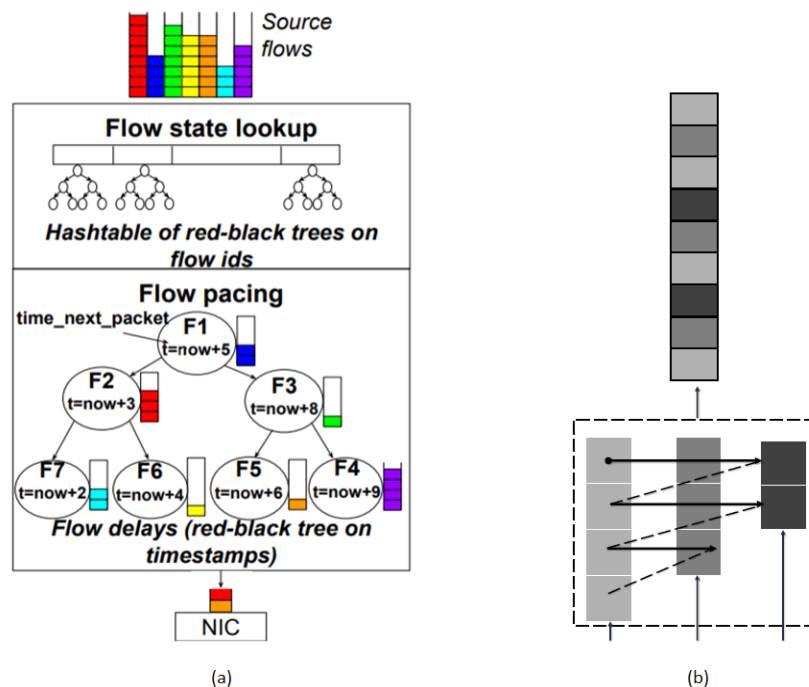


Figure 3. (a) FQ-pacing. (b) Deficit Round-Robin (DRR) algorithm.

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.

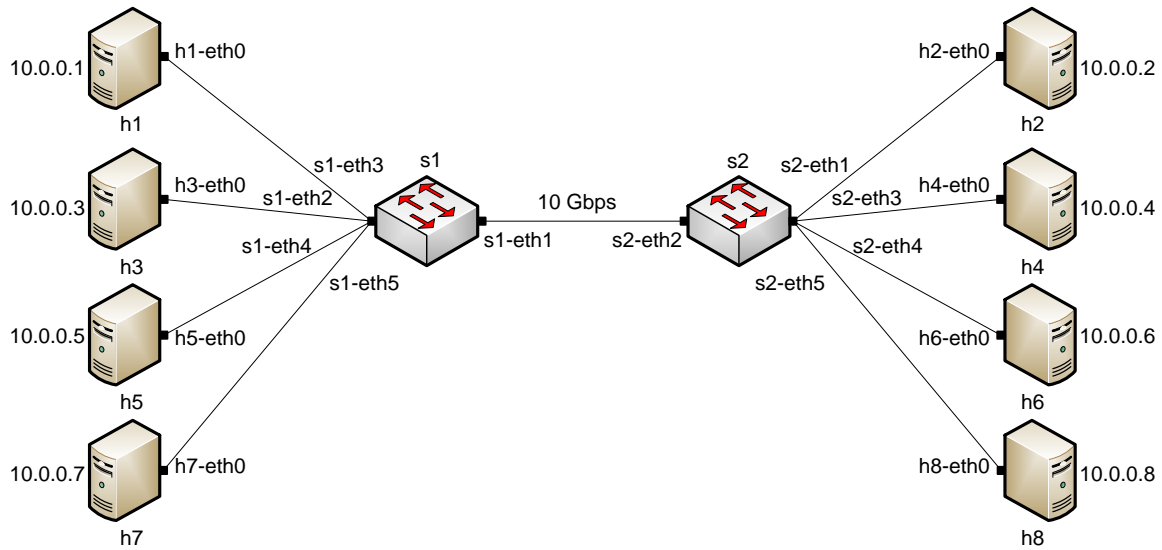


Figure 4. Lab topology.

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`.

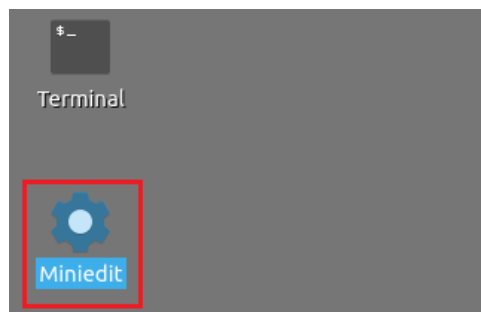


Figure 5. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 12.mn* topology file and click on *Open*.

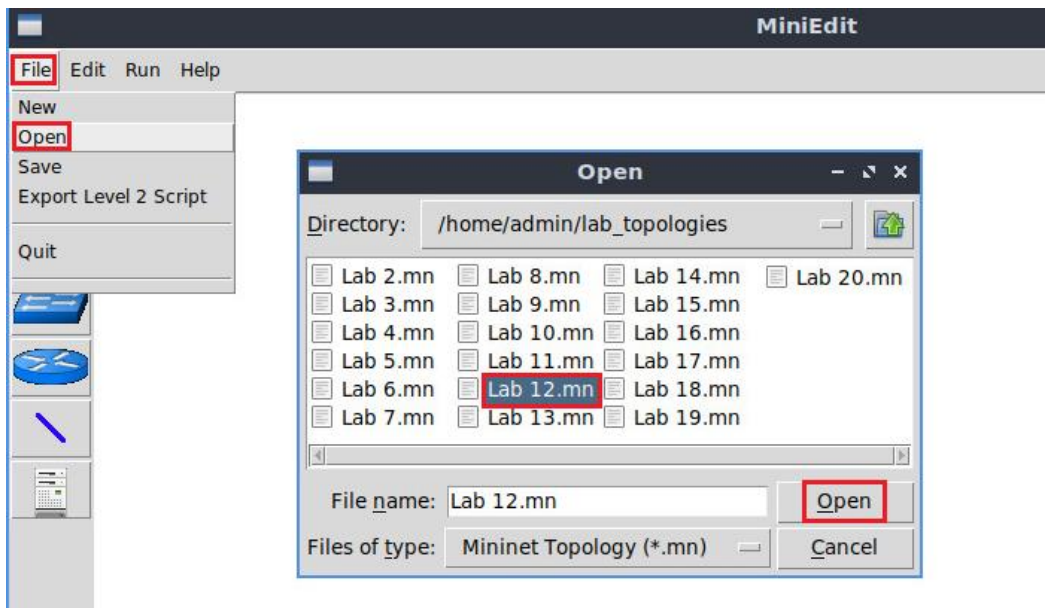


Figure 6. MiniEdit's *Open* dialog.

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.

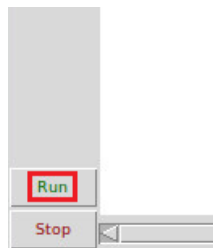


Figure 7. Running the emulation.

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

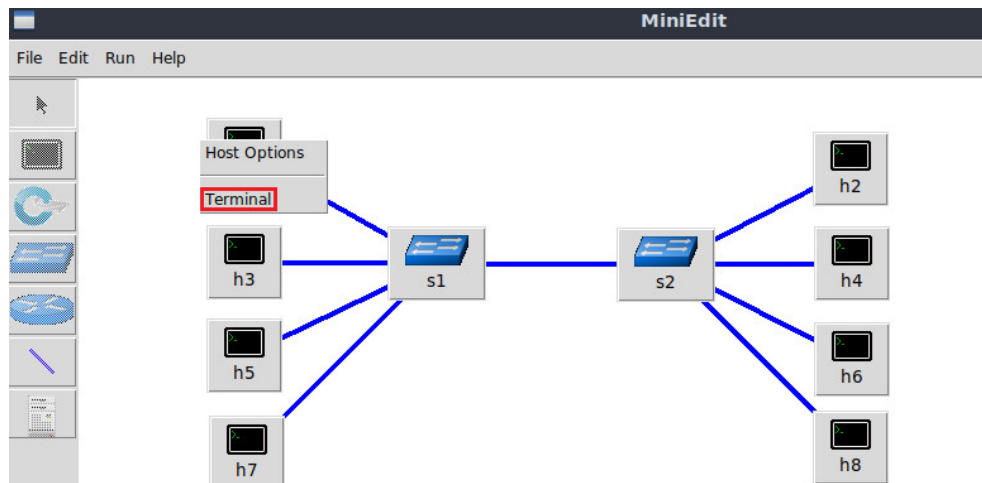


Figure 8. Opening a terminal on host h1.

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.

```

X "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.

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 hosts 1 and 2 to 10 Gbps.

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

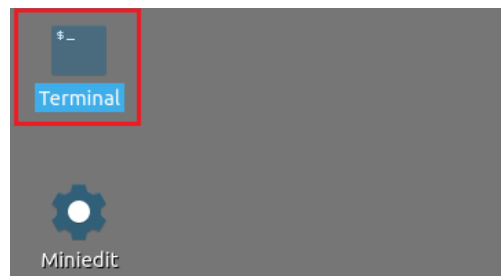


Figure 10. Shortcut to open a Linux terminal.

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 to perform.

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 10Gbps on switch S1's `s1-eth2` interface. The `tbw` parameters are the following:

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

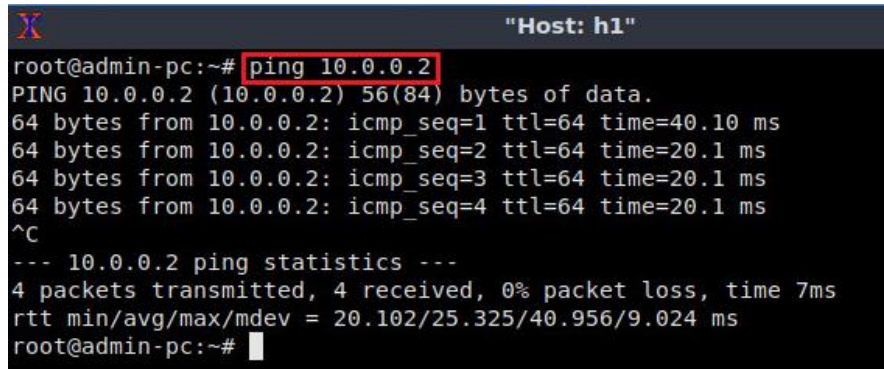
```
sudo tc qdisc add dev s1-eth1 parent 1: handle 2: tbf rate 10gbit burst 5000000 limit 15000000
```

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

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

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

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

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.

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

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

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

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'
```

```

"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 14. Receive window change in `sysctl`.

Step 3. 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'
```

```

"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 15. Send window change in `sysctl`.

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

Step 4. 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'
```

```

"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 16. Receive window change in `sysctl`.

Step 5. 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'
```

```

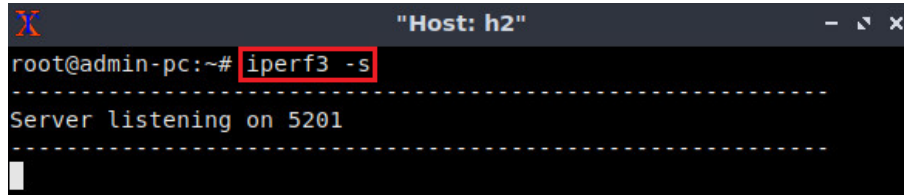
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#

```

Figure 17. Send window change in `sysctl`.

Step 6. 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
```



```

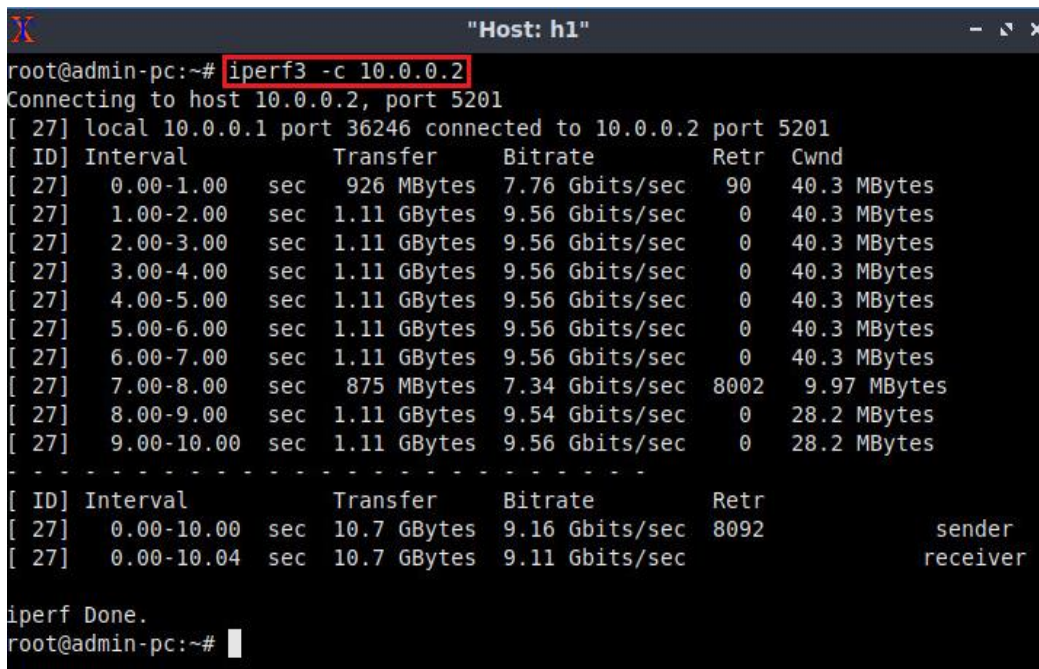
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----

```

Figure 18. Host h2 running iPerf3 as server.

Step 7. 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
[ 27] local 10.0.0.1 port 36246 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate      Retr  Cwnd
[ 27]  0.00-1.00       sec   926 MBytes  7.76 Gbits/sec    90  40.3 MBytes
[ 27]  1.00-2.00       sec   1.11 GBytes  9.56 Gbits/sec     0  40.3 MBytes
[ 27]  2.00-3.00       sec   1.11 GBytes  9.56 Gbits/sec     0  40.3 MBytes
[ 27]  3.00-4.00       sec   1.11 GBytes  9.56 Gbits/sec     0  40.3 MBytes
[ 27]  4.00-5.00       sec   1.11 GBytes  9.56 Gbits/sec     0  40.3 MBytes
[ 27]  5.00-6.00       sec   1.11 GBytes  9.56 Gbits/sec     0  40.3 MBytes
[ 27]  6.00-7.00       sec   1.11 GBytes  9.56 Gbits/sec     0  40.3 MBytes
[ 27]  7.00-8.00       sec   875 MBytes  7.34 Gbits/sec   8002  9.97 MBytes
[ 27]  8.00-9.00       sec   1.11 GBytes  9.54 Gbits/sec     0  28.2 MBytes
[ 27]  9.00-10.00      sec   1.11 GBytes  9.56 Gbits/sec     0  28.2 MBytes
-----
[ ID] Interval           Transfer     Bitrate      Retr
[ 27]  0.00-10.00      sec   10.7 GBytes  9.16 Gbits/sec   8092
[ 27]  0.00-10.04      sec   10.7 GBytes  9.11 Gbits/sec
iperf Done.
root@admin-pc:~#

```

Figure 19. iPerf3 throughput test.

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

Step 8. 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 Enabling TCP pacing with tc and fq

The user enables fair queuing using a command line utility called `tc`. The basic `tc` syntax used with `fq` is as follows:

```
sudo tc qdisc [add|del|replace|change|show] dev dev_id root fq opts
```

`sudo`: enables the execution of the command with higher security privileges.

`tc`: invokes Linux's traffic control.

`qdisc`: a queue discipline (qdisc) is a set of rules that determine the order in which packets arriving from the IP protocol output are served. The queue discipline is applied to a packet queue to decide when to send each packet.

`[add | del | replace | change | show]`: this is the operation on qdisc. For example, to add delay on a specific interface, the operation will be `add`. To change or remove delay on the specific interface, the operation will be `change` or `del`.

`dev id`: this parameter indicates the interface to be subject to emulation.

`fq`: this parameter enables fair queuing qdisc.

`opts`: this parameter indicates the amount of delay, packet loss, duplication, corruption, and others.

Step 1. In host h1, type the following command:

```
sudo tc qdisc add dev h1-eth0 root fq maxrate 5gbit
```

This command can be summarized as follows:

`sudo`: enable the execution of the command with higher security privileges.

`tc`: invoke Linux's traffic control.

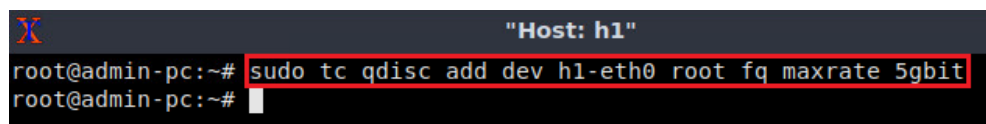
`qdisc`: modify the queuing discipline of the network scheduler.

`add`: create a new rule.

`dev h1-eth0`: specify the interface on which the rule will be applied.

`fq`: use the fair queuing qdiscs.

`maxrate 5gbit`: Maximum sending rate of a flow (default is unlimited). Enables pacing on a maximum rate of 5 Gbps.



```

Host: h1
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root fq maxrate 5gbit
root@admin-pc:~#

```

Figure 20. Enabling fair queuing pacing with a maximum rate of 5 Gbps to the interface `h1-eth0` on host h1.

Step 2. The user can now verify pacing 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
```



```

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

```

Figure 21. Host h2 running iPerf3 as server.

Step 3. Now to launch iPerf3 in client mode again by running the command `iperf3 -c 10.0.0.2 -O 5` in host h1's terminal. The `-O` option is used to specify the number of seconds to omit in the resulting report.

```
iperf3 -c 10.0.0.2 -O 5
```

```

root@admin-pc:~# iperf3 -c 10.0.0.2 -O 5
Connecting to host 10.0.0.2, port 5201
[ 27] local 10.0.0.1 port 36258 connected to 10.0.0.2 port 5201
[ ID] Interval      Transfer      Bitrate      Retr  Cwnd
[ 27] 0.00-1.00    sec  19.3 MBytes  162 Mbits/sec  0    891 KBytes (omitted)
[ 27] 1.00-2.00    sec  202 MBytes  1.70 Gbits/sec  0    9.50 MBytes (omitted)
[ 27] 2.00-3.00    sec  564 MBytes  4.73 Gbits/sec  0    12.4 MBytes (omitted)
[ 27] 3.00-4.00    sec  569 MBytes  4.77 Gbits/sec  0    12.4 MBytes (omitted)
[ 27] 4.00-5.00    sec  570 MBytes  4.78 Gbits/sec  0    12.4 MBytes (omitted)
[ 27] 0.00-1.00    sec  570 MBytes  4.78 Gbits/sec  0    12.4 MBytes
[ 27] 1.00-2.00    sec  569 MBytes  4.77 Gbits/sec  32   12.4 MBytes
[ 27] 2.00-3.00    sec  570 MBytes  4.78 Gbits/sec  0    12.4 MBytes
[ 27] 3.00-4.00    sec  570 MBytes  4.78 Gbits/sec  0    12.4 MBytes
[ 27] 4.00-5.00    sec  569 MBytes  4.77 Gbits/sec  0    12.4 MBytes
[ 27] 5.00-6.00    sec  570 MBytes  4.78 Gbits/sec  0    12.4 MBytes
[ 27] 6.00-7.00    sec  568 MBytes  4.76 Gbits/sec  0    12.4 MBytes
[ 27] 7.00-8.00    sec  570 MBytes  4.78 Gbits/sec  0    12.4 MBytes
[ 27] 8.00-9.00    sec  570 MBytes  4.78 Gbits/sec  0    12.4 MBytes
[ 27] 9.00-10.00   sec  569 MBytes  4.77 Gbits/sec  0    12.4 MBytes
-----
[ ID] Interval      Transfer      Bitrate      Retr
[ 27] 0.00-10.00   sec  5.56 GBytes  4.78 Gbits/sec  32
[ 27] 0.00-10.04   sec  5.58 GBytes  4.78 Gbits/sec
iperf Done.
root@admin-pc:~#

```

Figure 22. iPerf3 throughput test.

The figure above shows the iPerf3 test output report. The average achieved throughput is 4.78 Gbps (sender) and 4.78 Gbps (receiver), which is close to the assigned pacing value (5 Gbps).

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 Enabling TCP pacing from application

An application can specify a maximum pacing rate using the `SO_MAX_PACING_RATE` setsockopt call. This packet scheduler adds delay between packets to respect rate limitation set on each socket. Application specific setting via `SO_MAX_PACING_RATE` is ignored only if it is larger than the `maxrate` value assigned with `fq` (if any).

In iPerf3, the option `--fq-rate` sets a rate to be used with fair-queueing based socket-level pacing, in bits per second.

Step 1. Remove previous `qdiscs` on host h1's `h1-eth0` interface.

```
sudo tc qdisc del dev h1-eth0 root
```

```

Host: h1
root@admin-pc:~# sudo tc qdisc del dev h1-eth0 root
root@admin-pc:~#

```

Figure 23. Removing *qdiscs* on host h1's *h1-eth0* interface.

Step 2. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal:

```
iperf3 -s
```

```

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

```

Figure 24. Host h2 running iPerf3 as server.

Step 3. Now launch iPerf3 in client mode by running the command `iperf3 -c 10.0.0.2 -O 5 --fq-rate 5gbit` in host h1's terminal. The `-O 5` option is used to specify the number of seconds to omit in the resulting report (5 seconds), and the `--fq-rate` is used to enable pacing through the `SO_MAX_PACING_RATE` setsockopt call.

```
iperf3 -c 10.0.0.2 -O 5 --fq-rate 5gbit
```

```

Host: h1
root@admin-pc:~# iperf3 -c 10.0.0.2 -O 5 --fq-rate 5gbit
Connecting to host 10.0.0.2, port 5201
[ 27] local 10.0.0.1 port 36266 connected to 10.0.0.2 port 5201
[ ID] Interval          Transfer      Bitrate      Retr  Cwnd
[ 27]  0.00-1.00      sec   9.39 MBytes  78.7 Mbits/sec    0   355 KBytes  (omitted)
[ 27]  1.00-2.00      sec  41.4 MBytes  348 Mbits/sec    0   1.24 MBytes  (omitted)
[ 27]  2.00-3.00      sec  170 MBytes  1.43 Gbits/sec    0   5.56 MBytes  (omitted)
[ 27]  3.00-4.00      sec  370 MBytes  3.10 Gbits/sec    0   8.83 MBytes  (omitted)
[ 27]  4.00-5.00      sec  504 MBytes  4.23 Gbits/sec    0  11.1 MBytes
[ 27]  0.00-1.00      sec   540 MBytes  4.53 Gbits/sec    0  11.7 MBytes
[ 27]  1.00-2.00      sec   541 MBytes  4.54 Gbits/sec    0  11.7 MBytes
[ 27]  2.00-3.00      sec   541 MBytes  4.54 Gbits/sec    0  11.7 MBytes
[ 27]  3.00-4.00      sec   541 MBytes  4.54 Gbits/sec    0  11.7 MBytes
[ 27]  4.00-5.00      sec   540 MBytes  4.53 Gbits/sec    0  11.7 MBytes
[ 27]  5.00-6.00      sec   540 MBytes  4.53 Gbits/sec    0  11.7 MBytes
[ 27]  6.00-7.00      sec   538 MBytes  4.51 Gbits/sec    0  11.7 MBytes
[ 27]  7.00-8.00      sec   541 MBytes  4.54 Gbits/sec    0  11.7 MBytes
[ 27]  8.00-9.00      sec   536 MBytes  4.50 Gbits/sec    0  11.7 MBytes
[ 27]  9.00-10.00     sec   540 MBytes  4.53 Gbits/sec    0  11.7 MBytes
-----
[ ID] Interval          Transfer      Bitrate      Retr
[ 27]  0.00-10.00     sec   5.27 GBytes  4.53 Gbits/sec    0
[ 27]  0.00-10.04     sec   5.29 GBytes  4.53 Gbits/sec
iperf Done.
root@admin-pc:~#

```

Figure 25. iPerf3 throughput test with pacing enabled by iPerf3 application.

5 Concurrent transmission without pacing

In the previous section, we applied pacing on a single host (host h1) and we measured the average throughput. In this section we run a test where four clients (host h1, host h3, host h5, and host h7) are transmitting simultaneously to four servers (host h2, host h4, host h6, and host h8), while sharing the same bottleneck link (link connecting switch S1 to switch S2).

Since it is difficult to start the four clients at the same time, Client1'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.

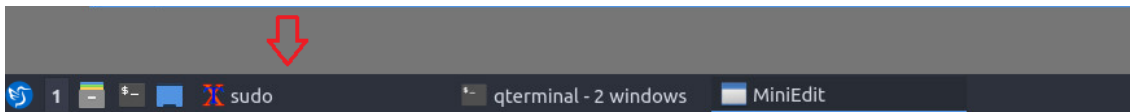


Figure 26. Opening Mininet's terminal.

```
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 27. Mininet's terminal.

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

```
source concurrent_no_pacing
```

```

sudo
mininet> source concurrent_no_pacing
Removing previous qdiscs on end-hosts.
--> tc qdisc del dev dev_eth root

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 ...
h5 connected to h6. Transmitting for 20 seconds, please wait ...
h7 connected to h8. 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 28. Running the tests simultaneously for 20 seconds without applying pacing.

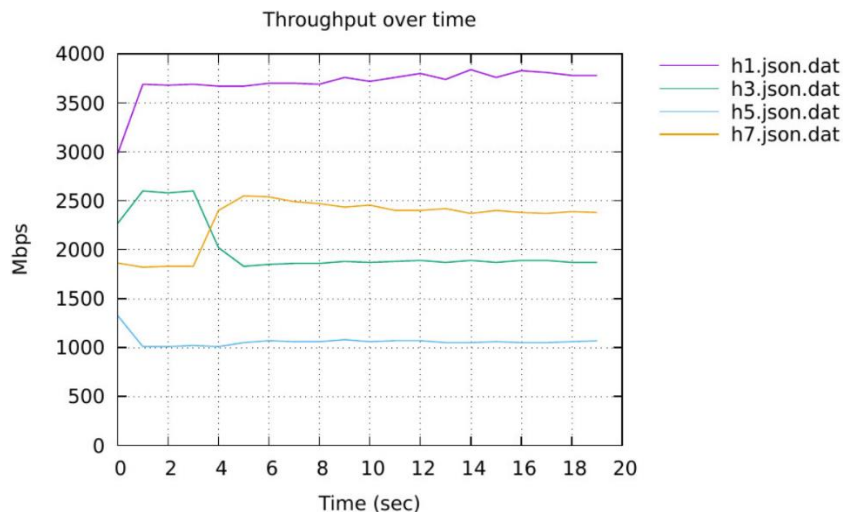


Figure 29. Throughput of host h1, host h3, host h5 and host h7.

The above graph shows that the throughput of host h1, host h3, host h5 and host h7. It is clear from the figure that there are variations in the flows. Moreover, the bottleneck bandwidth was not evenly shared among the hosts, which decreases the fairness index from 100%.

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_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 ...
h5 connected to h6. Transmitting for 20 seconds, please wait ...
h7 connected to h8. 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=.83588
*****

```

Figure 30. Calculated fairness index.

The above figure shows a fairness index of .83588. This value indicates that the bottleneck bandwidth was approximately 83% evenly shared among host h1, host h3, host h5, and host h7.

6 Concurrent transmission with pacing

In the previous section, we ran a test where four clients (host h1, host h3, host h5, and host h7) are transmitting simultaneously to four servers (host h2, host h4, host h6, and host h8), while sharing the same bottleneck link (link connecting switch S1 to switch S2) without applying pacing. In this section we repeat the same test, but with pacing enabled on host h1, host h3, host h5 and host h7.

Since it is difficult to start the four clients at the same time, Client1's machine provides a script that automates this process.

Step 1. Using same Mininet's terminal, issue the following command on Mininet's terminal as shown in the figure below.

```
source concurrent_pacing
```

```

sudo
mininet> source concurrent_pacing
Removing previous qdiscs on end-hosts.
---> tc qdisc del dev dev_eth root

Enabling fq pacing to a maxrate of 2.5gbit per host.
---> tc qdisc add dev dev_eth root fq maxrate 2.5gbit

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 ...
h5 connected to h6. Transmitting for 20 seconds, please wait ...

```

Figure 31. Running the tests simultaneously for 20 seconds while applying pacing.

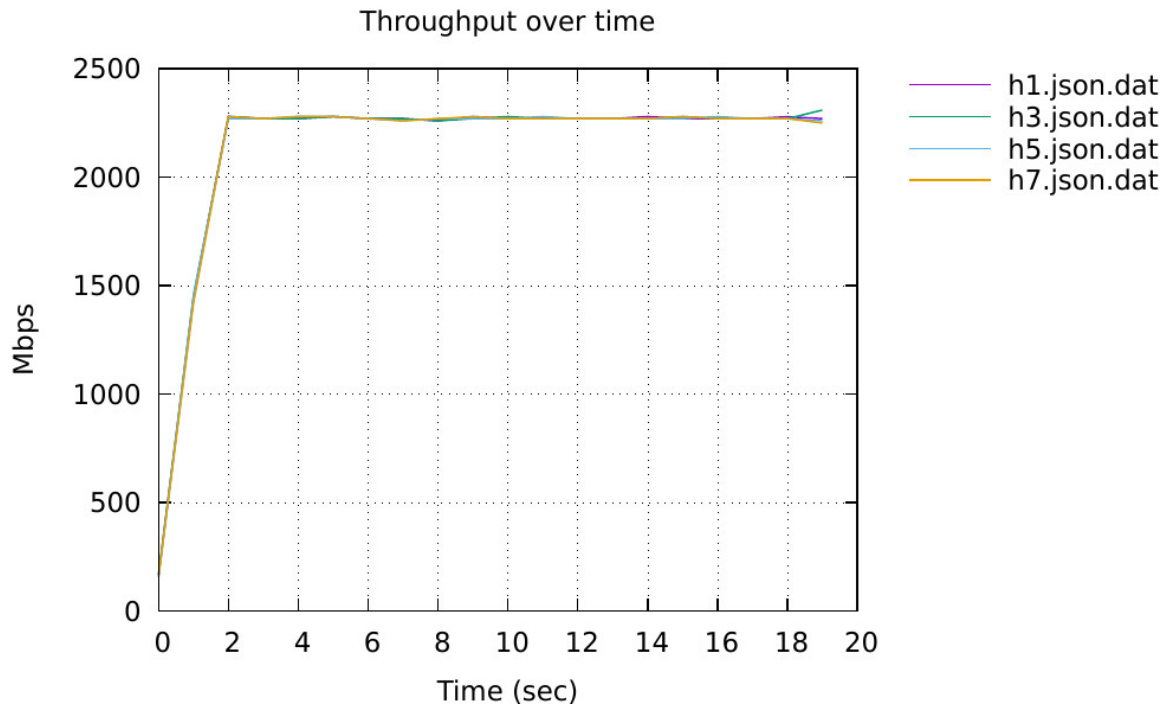


Figure 32. Throughput of host h1, host h3, host h5 and host h7 after applying pacing.

The above graph shows that the throughput of host h1, host h3, host h5 and host h7 with pacing enabled. It is clear from the figure that there are less variations in the flows compared to the non-paced flows. Moreover, the bottleneck bandwidth is now better shared among the hosts.

Step 2. 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_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 ...
h5 connected to h6. Transmitting for 20 seconds, please wait ...
h7 connected to h8. 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= .99999
*****
    
```

Figure 33. Calculated fairness index.

The above figure shows a fairness index of .99999. The fairness index here is better than the previous test .83588. Therefore, pacing generally improves fairness among transmitting hosts.

7 Parallel streams and without pacing

In the previous tests, four clients (host h1, host h3, host h5, and host h7) were transmitting simultaneously to four servers (host h2, host h4, host h6, and host h8), while sharing the same bottleneck link (link connecting switch S1 to switch S2). In this section only one client (host h1) is transmitting to one server (host h2) while using five parallel streams.

Step 1. In MiniEdit, 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.

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

Step 3. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal:

```
iperf3 -s
```

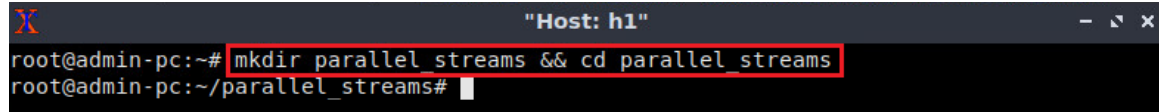
```

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

Figure 34. Host h2 running iPerf3 as server.

Step 4. Create and enter to a new directory *parallel_streams*:

```
mkdir parallel_streams && cd parallel_streams
```



```

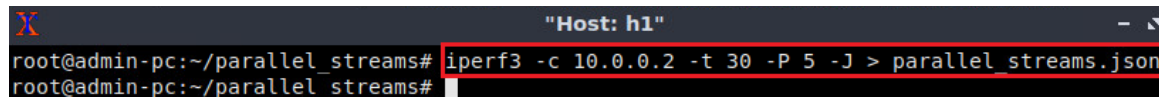
Host: h1
root@admin-pc:~# mkdir parallel_streams && cd parallel_streams
root@admin-pc:~/parallel_streams#

```

Figure 35. Creating and entering a new directory *parallel_streams*.

Step 5. Launch iPerf3 in client mode on host h1's terminal. The `-J` option is used to produce a JSON output and the redirection operator `>` to send the standard output to a file.

```
iperf3 -c 10.0.0.2 -t 30 -P 5 -J > parallel_streams.json
```



```

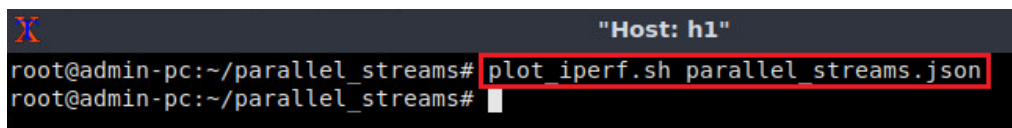
Host: h1
root@admin-pc:~/parallel_streams# iperf3 -c 10.0.0.2 -t 30 -P 5 -J > parallel_streams.json
root@admin-pc:~/parallel_streams#

```

Figure 36. Running iPerf3 client on host h1 with 5 parallel streams for 30 seconds, and redirecting the output to *parallel_streams.json*.

Step 6. Once the test is finished, in order to generate the output plots for iPerf3's JSON file run the following command:

```
plot_iperf.sh parallel_streams.json
```



```

Host: h1
root@admin-pc:~/parallel_streams# plot_iperf.sh parallel_streams.json
root@admin-pc:~/parallel_streams#

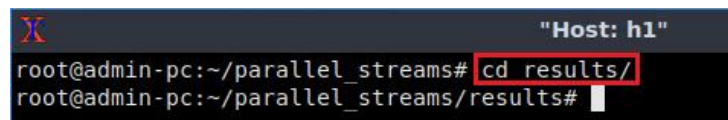
```

Figure 37. `plot_iperf.sh` script generating output results.

This plotting script generates PDF files for the following fields: congestion window (*cwnd.pdf*), retransmits (*retransmits.pdf*), Round-Trip Time (*RTT.pdf*), throughput (*throughput.pdf*), maximum transmission unit (*MTU.pdf*), bytes transferred (*bytes.pdf*). These files are stored in a directory *results* created in the same directory where the script was executed.

Step 7. Navigate to the results folder using the `cd` command.

```
cd results/
```



```

Host: h1
root@admin-pc:~/parallel_streams# cd results/
root@admin-pc:~/parallel_streams/results#

```

Figure 38. Entering the results directory using the `cd` command.

Step 8. Open the *throughput.pdf* file, use the following command:

```
xdg-open throughput.pdf
```

```

Host: h1
root@admin-pc:~/parallel_streams/results# xdg-open throughput.pdf
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'

```

Figure 39. Opening the *throughput.pdf* file using `xdg-open`.

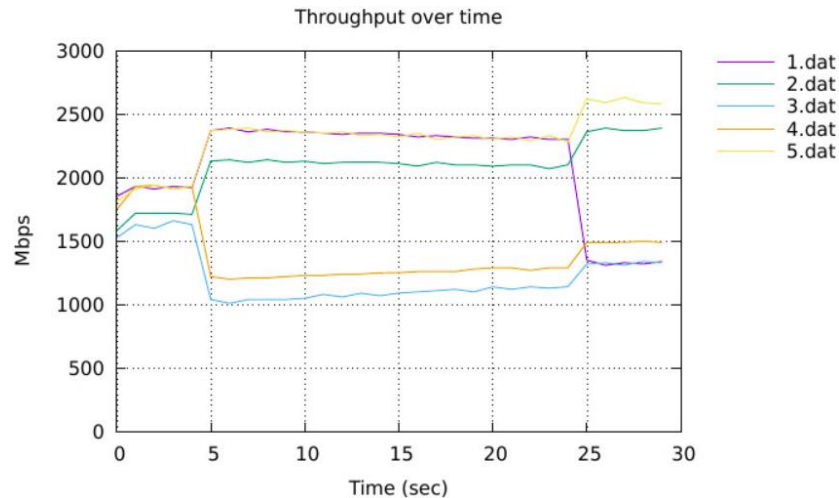


Figure 40. Throughput of 5 parallel streams initiated by host h1 without pacing.

Step 9. Close *throughput.pdf* file and stop the server by pressing `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

Step 10. Exit the *parallel_streams/results* directory by using the following command on host h1's terminal:

```
cd ../../
```

```

Host: h1
root@admin-pc:~/parallel_streams/results# cd ../../
root@admin-pc:~#

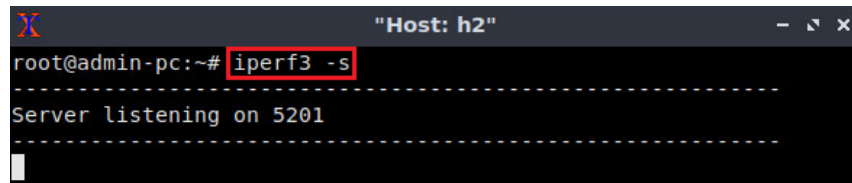
```

Figure 41. Exiting the *reno/results* directory.

8 Parallel streams and with pacing

Step 1. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal:

```
iperf3 -s
```



```

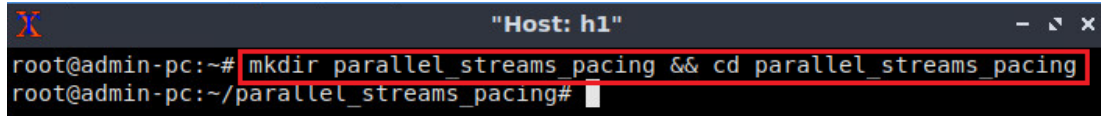
Host: h2
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----

```

Figure 42. Host h2 running iPerf3 as server.

Step 2. Create and enter to a new directory *parallel_streams_pacing*:

```
mkdir parallel_streams_pacing && cd parallel_streams_pacing
```



```

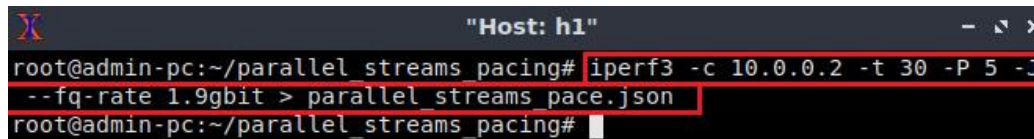
Host: h1
root@admin-pc:~# mkdir parallel_streams_pacing && cd parallel_streams_pacing
root@admin-pc:~/parallel_streams_pacing#

```

Figure 43. Creating and entering a new directory *parallel_streams_pacing*.

Step 3. Launch iPerf3 in client mode on host h1's terminal. The `-J` option is used to produce a JSON output and the redirection operator `>` to send the standard output to a file. The `-P` is used to specify the number of parallel streams, and the `--fq-rate` is used to enable pacing through the `SO_MAX_PACING_RATE` setsockopt call. In this test, pacing is applied to a maximum rate of 1.9 Gbps per stream, and 5 * 1.9 Gbps (9.5 Gbps) total for all streams. Note that assigning a pacing rate slightly less than the maximum bandwidth (10 Gbps in our case) reduces packet lost and the variations of flows.

```
iperf3 -c 10.0.0.2 -t 30 -P 5 -J --fq-rate 1.9gbit > parallel_streams_pace.json
```



```

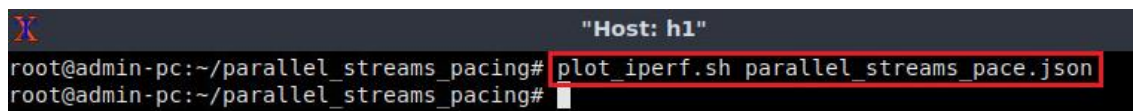
Host: h1
root@admin-pc:~/parallel_streams_pacing# iperf3 -c 10.0.0.2 -t 30 -P 5 -J
--fq-rate 1.9gbit > parallel_streams_pace.json
root@admin-pc:~/parallel_streams_pacing#

```

Figure 44. Running iPerf3 client on host h1 with 5 parallel streams for 30 seconds with pacing enabled, and redirecting the output to *parallel_streams_pace.json*.

Step 4. Once the test is finished, type the command, to generate the output plots for iPerf3's JSON file run the following command:

```
plot_iperf.sh parallel_streams_pace.json
```



```

Host: h1
root@admin-pc:~/parallel_streams_pacing# plot_iperf.sh parallel_streams_pace.json
root@admin-pc:~/parallel_streams_pacing#

```

Figure 45. `plot_iperf.sh` script generating output results.

This plotting script generates PDF files for the following fields: congestion window (*cwnd.pdf*), retransmits (*retransmits.pdf*), Round-Trip Time (*RTT.pdf*), throughput (*throughput.pdf*), maximum transmission unit (*MTU.pdf*), bytes transferred (*bytes.pdf*). These files are stored in a directory *results* created in the same directory where the script was executed.

Step 5. Navigate to the results folder using the `cd` command.

```
cd results/
```

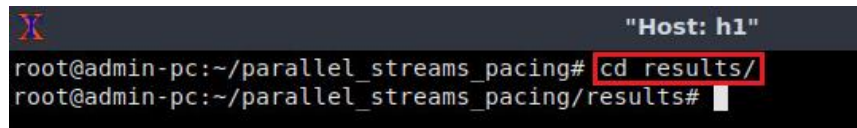


Figure 46. Entering the *results* directory using the `cd` command.

Step 6. Open the *throughput.pdf* file, use the following command:

```
xdg-open throughput.pdf
```

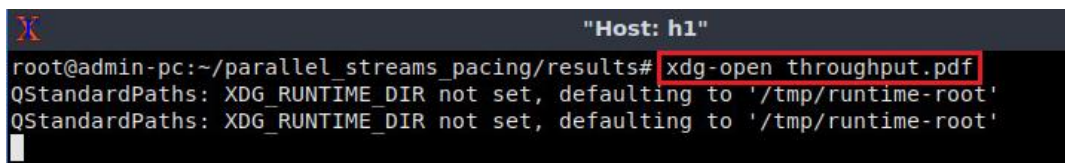


Figure 47. Opening the *throughput.pdf* file using `xdg-open`.

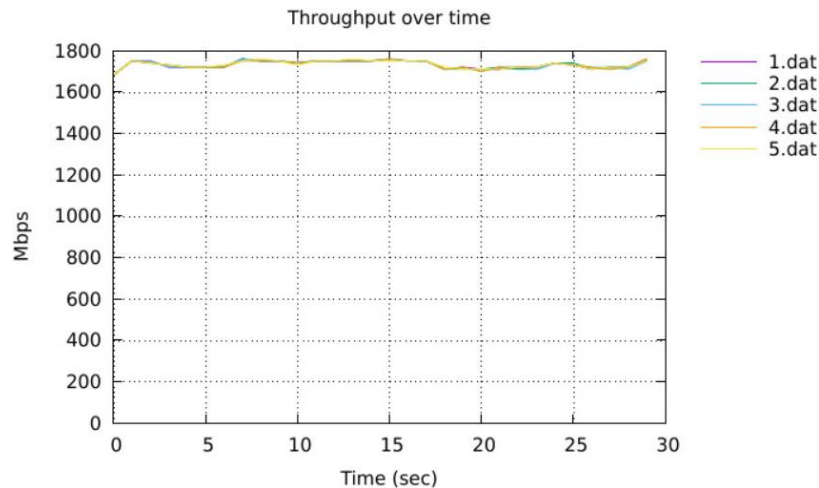


Figure 48. Throughput of 5 parallel streams initiated by host h1 with pacing applied to a maximum rate of 1.9 Gbps per stream.

The graph above shows how the advantages of applying pacing when using parallel streams. Compared to figure 40, the flows have less variations and the fairness among these flows is improved.

This concludes Lab 12. Stop the emulation and then exit out of MiniEdit.

References

1. A. Aggarwal, S. Savage, T. Anderson, "Understanding the performance of TCP pacing," in Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), Mar. 2000.

2. B. Tierney, N. Hanford, D. Ghosal, "Optimizing data transfer nodes using packet pacing: a journey of discovery," in Workshop on Innovating the Network for Data-Intensive Science, Nov. 2015.
3. M. Ghobadi, Y. Ganjali, "TCP pacing in data center networks," in IEEE Annual Symposium on High-Performance Interconnects (HOTI), Aug. 2013.
4. N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, V. Jacobson, "BBR: congestion-based congestion control," *Communications of the ACM*, vol 60, no. 2, pp. 58-66, Feb. 2017.
5. Fair Queue traffic policing. [Online]. Available: <http://man7.org/linux/man-pages/man8/tc-fq.8.html>
6. The centos project. [Online]. Available: <https://www.centos.org>
7. J. Corbet, "TSO sizing and the FQ scheduler," *LWN.net Online Magazine*, Aug. 2013. [Online]. Available: <https://lwn.net/Articles/564978>
8. B. Tierney, "Improving performance of 40G/100G data transfer nodes," in 2016 Technology Exchange Workshop, Sep. 2016. [Online]. Available: <https://meetings.internet2.edu/2016-technologyexchange/detail/10004333/>
9. I. Rhee, L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM Special Interest Group on Operating Systems Operating System Review*, vol. 42, issue 5, pp. 64-74, Jul. 2008.
10. J. Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 303-314, Sep. 1998.
11. A. Saeed, N. Dukkipati, V. Valancius, C. Contavalli, A. Vahdat, "Carousel: scalable traffic shaping at end hosts," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 404-417, Aug. 2017.
12. M. Shreedhar, G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, issue 3, pp. 375-385, Jun. 1996.