NETWORK TOOLS AND PROTOCOLS

Lab 17: Stochastic Fair Queueing (SFQ)

Document Version: 11-12-2019

Award 1829698
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”

Contents

Overview

This lab introduces to Stochastic Fair Queuing (SFQ), which is a queueing discipline aimed to ensure fairness between TCP flows. The lab describes the steps to conduct throughput tests that shows the benefits of isolating the dynamic of competing TCP flows by applying SFQ rules to a router egress' interface.

Objectives

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

1. Identify and describe the components of end-to-end latency.
2. Understand the scheduling process in a router.
3. Explain the impact of using SFQ to isolate the dynamic of competing TCP flows.
4. Visualize the interaction of competing TCP flows after SFQ is configured on a router's interface.

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.
2. Section 2: Lab topology.
3. Section 3: Testing the throughput of two competing TCP flows
4. Section 4: Configuring SFQ on switch S2.

1 Introduction

1.1 FIFO queueing discipline

First-in, first-out (FIFO) queuing is the most basic queue scheduling discipline. It is also the default queueing discipline in Linux and most of the routers[1]. In FIFO queuing, all packets are treated equally by placing them into a single queue, and then delivering them in the same order that they were placed into the queue (see Figure 1).

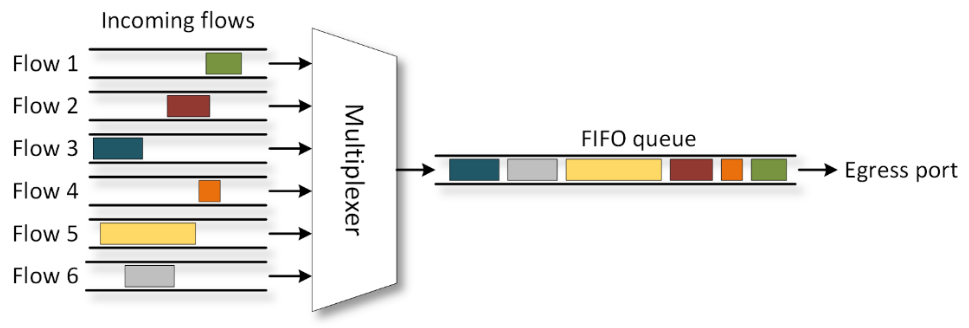


Figure 1. FIFO queuing discipline.

FIFO queuing presents advantages for software-based routers due to its low computational load on the system when compared with other queueing disciplines. Additionally, the behavior of a FIFO queue is very predictable, which means that packets are not reordered, and the queue delay is determined by the maximum depth of the queue. If the queue depth remains short, FIFO queuing provides simple contention resolution for network resources without adding significantly delay to the link.

However, a single FIFO queue does not allow routers to classify packets or set priorities. If a router uses a single FIFO queue, it will impact all flows equally, this means that the average queuing delay for all flows increases as congestion increases. As a result, FIFO queuing can result in increased delay, jitter, and loss for real-time applications. Another limitation of FIFO queuing is that bursty TCP flows can consume the entire buffer space of a FIFO queue, and that causes all other flows to be denied service until after the burst is serviced. This can result in increased delay, jitter, and loss for the other coexistent TCP flows

1.2 SFQ queueing discipline

Stochastic Fairness Queueing (SFQ) is a classless queueing discipline available for traffic control. SFQ does not shape traffic but only schedules the transmission of packets, based on classified flows[2]. The goal of the algorithm is to ensure fairness so that each flow can send data in turn thus, preventing any single flow from drowning out the rest. This feature may in fact have some effect in mitigating a Denial of Service attempts. SFQ is work-conserving and therefore always delivers a packet if it has one available.

SFQ classifies flows in a fixed set of queues serviced in strict round-robin order. The maximum number of queues is configurable (1024 by default in the Linux implementation). In order to assign a queue to an ingress packet, a hash function is applied to its 5-tuple determined by the IP source and destination, layer 4 port source and destination and layer 4 protocol number. Packets with the same hash are assigned to the same queue. Because of the hash, multiple sessions might end up in the same bucket, which would halve each session is chance of sending a packet, thus halving the effective speed available. To prevent this situation from becoming noticeable, SFQ changes its hashing algorithm quite often so that any two colliding sessions will only do so for a small number of seconds.

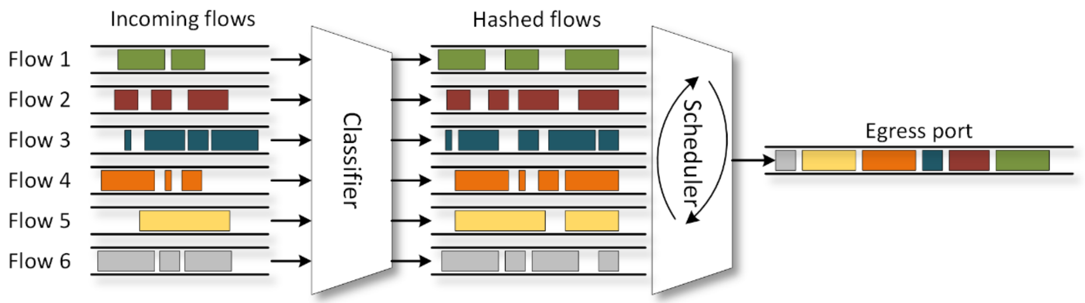


Figure 2. SFQ queuing discipline.

The basic `sfq` syntax used with `tc` is as follows:

```
tc qdisc [add | ...] dev [dev_id] root sfq perturb [seconds] quantum
```

- `tc`: Linux traffic control tool.
- `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 the token bucket algorithm on a specific interface, the operation will be `add`. To change or remove it, the operation will be `change` or `del`, respectively.
- `dev [dev_id]`: this parameter indicates the interface is to be subject to emulation.
- `sfq`: this parameter specifies the Stochastic Fair Queueing algorithm.
- `perturb`: It is used to specify the interval in seconds for queue algorithm perturbation. If the value is set to 0 indicate that no perturbation occurs. It is recommended avoiding low values to prevent packet reordering. Empirical evaluations recommend this value set to 10 seconds.
- `quantum`: Denotes the number of bytes which a flow can dequeue during a round of the round-robin process. It is recommended to set this value not less than the Maximum Transmission Unit (MTU).

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.

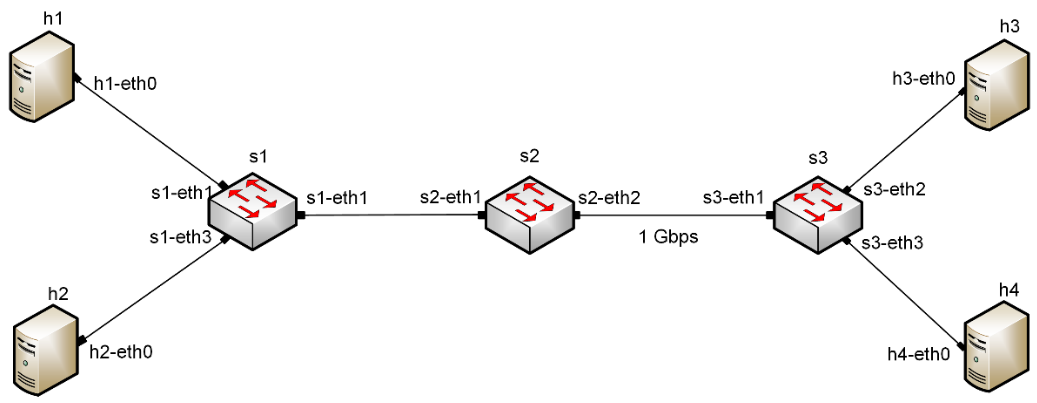


Figure 3. Lab topology.

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`.

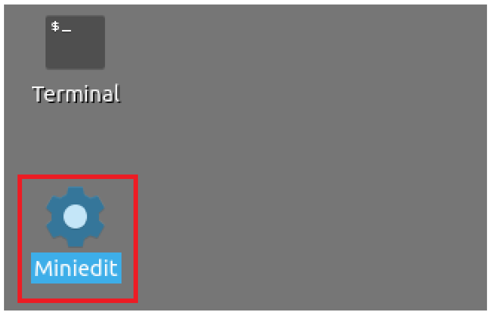


Figure 4. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 17.mn* topology file and click on *Open*.

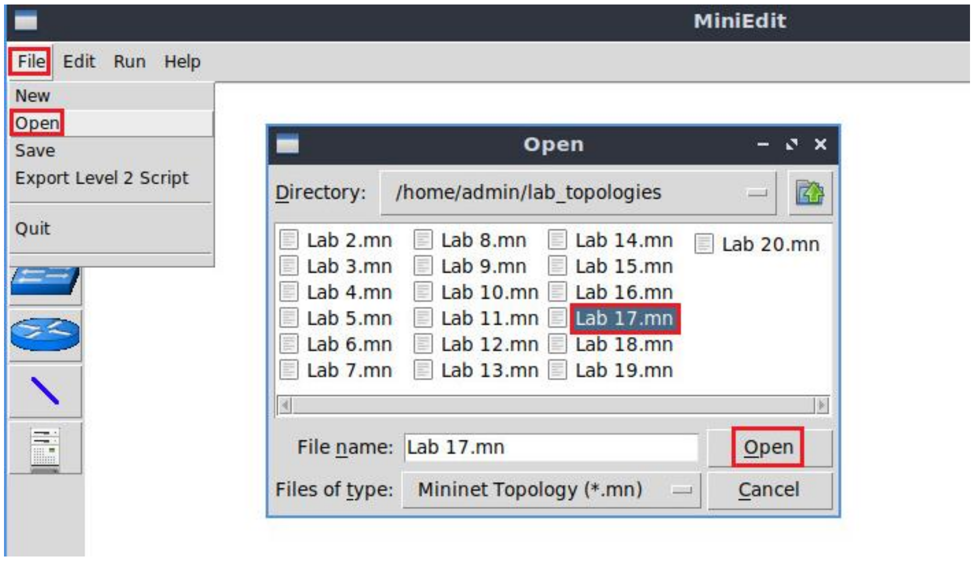


Figure 5. MiniEdit's *Open* dialog.

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

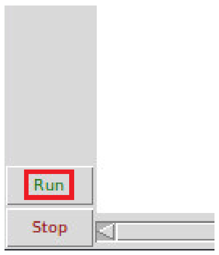


Figure 6. Running the emulation.

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

2.1 Starting host h1, host h2, host h3 and host h4

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.

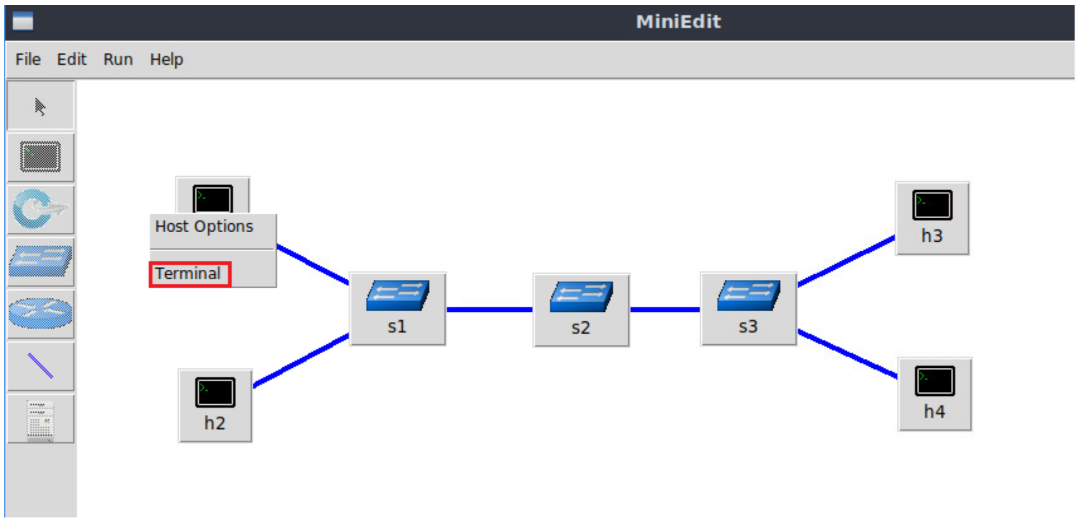
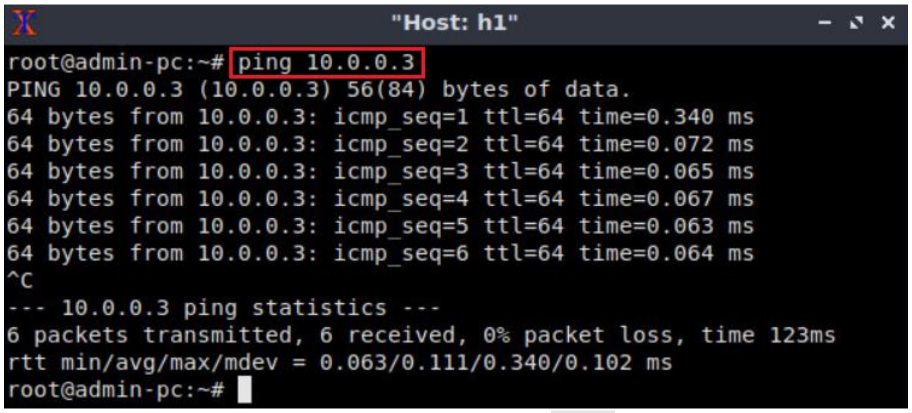


Figure 7. Opening a terminal on host h1.

Step 2. Apply the same steps on host h2 and host h3 and open their *Terminals*.

Step 3. Test connectivity between the end-hosts using the `ping` command. On host h1, type the command `ping 10.0.0.3`. This command tests the connectivity between host h1 and host h3. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.



```
"Host: h1"
root@admin-pc:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.340 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.072 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.065 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.067 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.063 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.064 ms
^C
--- 10.0.0.3 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 123ms
rtt min/avg/max/mdev = 0.063/0.111/0.340/0.102 ms
root@admin-pc:~#
```

Figure 8. Connectivity test using `ping` command.

2.2 Emulating high-latency WAN

This section emulates a high-latency WAN. We will emulate 20ms delay on switch S1's *s1-eth2* interface.

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

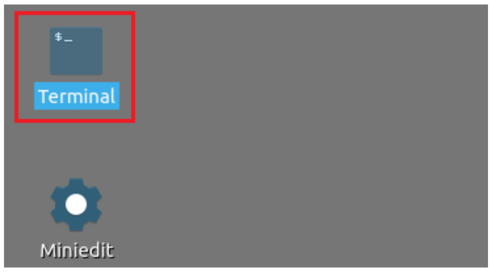
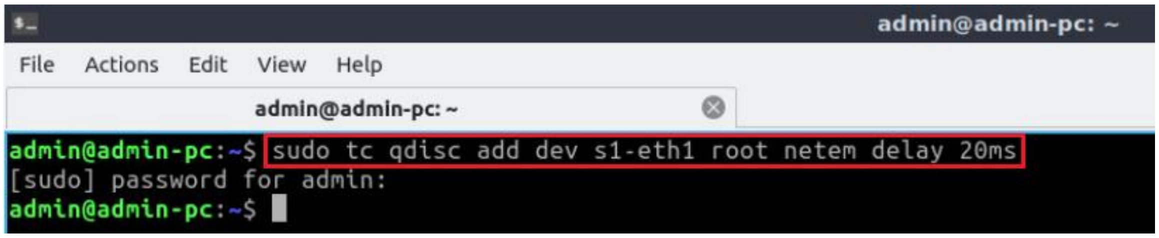


Figure 9. Shortcut to open a Linux terminal.

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 to switch S1's *s1-eth1* interface.

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



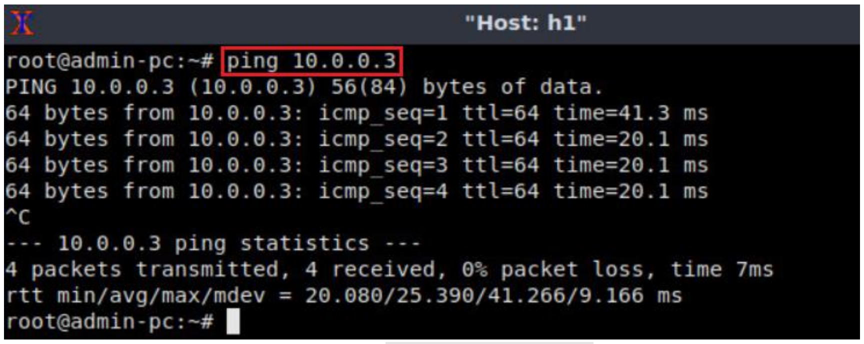
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth1 root netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 10. Adding delay of 20ms to 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 10.0.0.3`. 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 h3 (10.0.0.3), successfully receiving responses back.

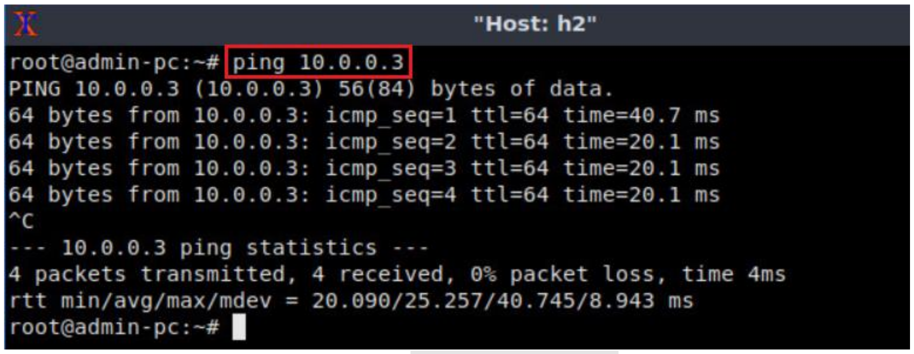


```
root@admin-pc:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=41.3 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.080/25.390/41.266/9.166 ms
root@admin-pc:~#
```

Figure 11. Output of `ping 10.0.0.3` 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 20.080, 25.390, 41.266, and 9.166 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 10.0.0.3`. The ping output in this test should be relatively similar to the results of the test initiated by host h1 in Step 1. To stop the test, press `Ctrl+c`.



```
root@admin-pc:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=40.7 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 4ms
rtt min/avg/max/mdev = 20.090/25.257/40.745/8.943 ms
root@admin-pc:~#
```

Figure 12. Output of `ping 10.0.0.3` 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 20.090, 25.257, 40.745, and 8.943 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

3 Testing the throughput of two competing TCP flows

In this section, you are going to tune the network devices in order to emulate a Wide Area Network (WAN). First, you will set the hosts' TCP buffers to 8 · BDP therefore, the bottleneck is not in the end-hosts. Then, you will add 20ms latency to switch S1's s1-eth1 interface. Additionally, you will set the bottleneck bandwidth to 1Gbps in switch S2's s2-eth2 interface. Finally, you will conduct throughput tests between two competing TCP flows which uses different congestion control algorithms (i.e. Cubic, BBR).

3.1 Bandwidth-delay product (BDP) and hosts' TCP buffer size

In the upcoming tests, the bandwidth is limited to 1 Gbps, and the RTT (delay or latency) is 20ms.

$$\text{BW} = 1{,}000{,}000{,}000 \text{ bits/second}$$

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

$$\begin{aligned}\text{BDP} &= 1{,}000{,}000{,}000 \cdot 0.02 = 20{,}000{,}000 \text{ bits}\\ &= 2{,}500{,}000 \text{ bytes} \approx 2.5 \text{ Mbytes}\end{aligned}$$

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

$$\text{BDP} = 2.5 \text{ Mbytes} = 2.5 \cdot 1024^2 \text{ bytes} = 2{,}621{,}440 \text{ bytes}$$

The default buffer size in Linux is 16 Mbytes, and only 8 Mbytes (half of the maximum buffer size) can be allocated. Since 8 Mbytes is greater than 2.5 Mbytes, then no need to tune the buffer sizes on end-hosts. However, in upcoming tests, we configure the buffer size on the switch to BDP. In addition, to ensure that the bottleneck is not the hosts' TCP buffers, we configure the buffers to 8·BDP (20,971,520).

3.2 Modifying hosts' buffer size

For the following calculation, the bottleneck bandwidth is considered as 1 Gbps, and the round-trip time latency as 20ms.

In order to have enough TCP buffer size, we will set the TCP sending and receiving buffer to $8 \cdot \text{BDP}$ in all hosts.

$$\text{BW} = 1{,}000{,}000{,}000 \text{ bits/second}$$

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

$$\begin{aligned}\text{BDP} &= 1{,}000{,}000{,}000 \cdot 0.02 = 20{,}000{,}000 \text{ bits}\\ &= 2{,}500{,}000 \text{ bytes} \approx 2.5 \text{ Mbytes}\end{aligned}$$

The send and receive TCP buffer sizes should be set to 8 · BDP to ensure the bottleneck is not in the end-hosts. For simplicity, we will use 2.5 Mbytes as the value for the BDP instead of 2,500,000 bytes.

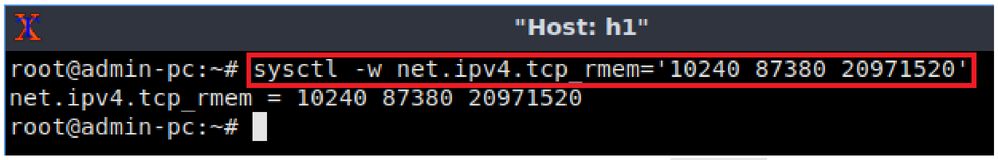
$$1\text{ Mbyte} = 1024^2\text{ bytes}$$

$$\text{BDP} = 2.5\text{ Mbytes} = 2.5 \cdot 1024^2\text{ bytes} = 2{,}621{,}440\text{ bytes}$$

$$8 \cdot \text{BDP} = 8 \cdot 2{,}621{,}440\text{ bytes} = 20{,}971{,}520\text{ bytes}$$

Step 1. At this point, we have calculated the maximum value of the TCP sending and receiving buffer size. In order to change the receiving buffer size, on host h1's terminal type the command shown below. The values set are: 10,240 (minimum), 87,380 (default), and 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```



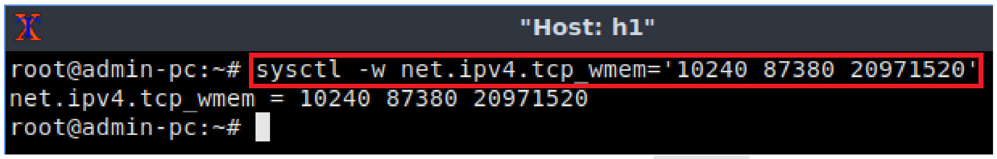
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
net.ipv4.tcp_rmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 13. Receive window change in `sysctl`.

The returned values are measured in bytes. 10,240 represents the minimum buffer size that is used by each TCP socket. 87,380 is the default buffer which is allocated when applications create a TCP socket. 20,971,520 is the maximum receive buffer that can be allocated for a TCP socket.

Step 2. 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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```



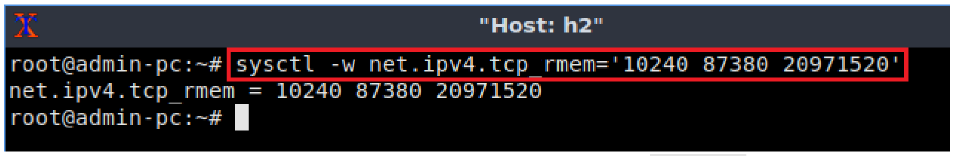
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
net.ipv4.tcp_wmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 14. Send window change in `sysctl`.

Next, the same commands must be configured on host h2, host h3, and host h4.

Step 3. To change the current receiver-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

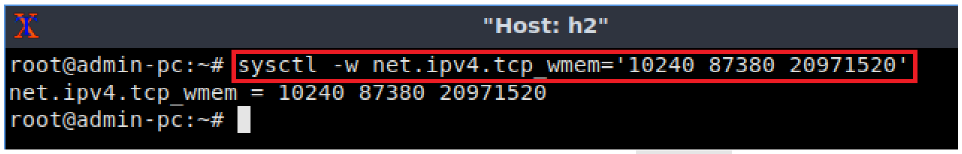


```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
net.ipv4.tcp_rmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 15. Receive window change in `sysctl`.

Step 4. 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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

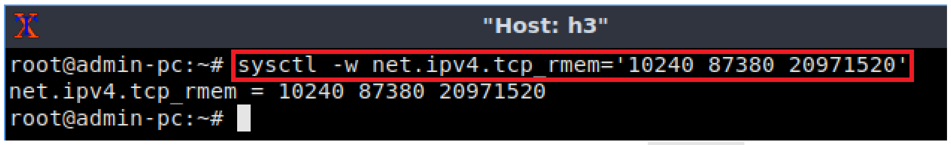


```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
net.ipv4.tcp_wmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 16. Send window change in `sysctl`.

Step 5. To change the current receiver-window size value(s), use the following command on host h3's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

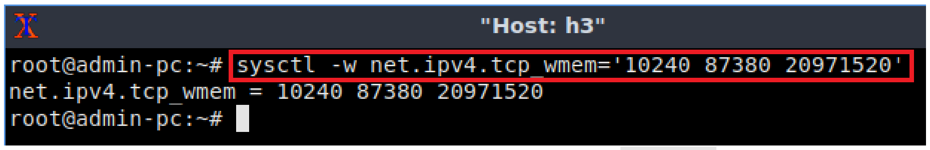


```
"Host: h3"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
net.ipv4.tcp_rmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 17. Receive window change in `sysctl`.

Step 6. To change the current send-window size value(s), use the following command on host h3's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

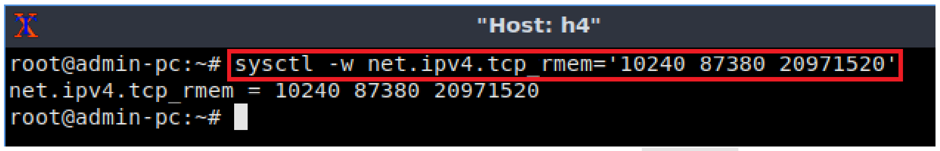


```
"Host: h3"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
net.ipv4.tcp_wmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 18. Send window change in `sysctl`.

Step 7. To change the current receiver-window size value(s), use the following command on host h4's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

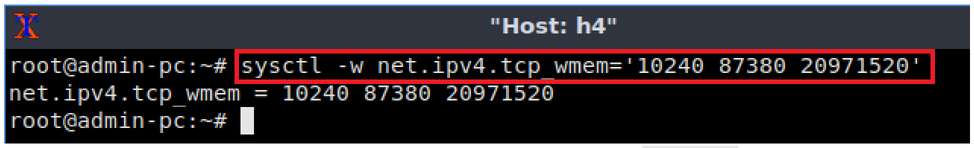


```
"Host: h4"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
net.ipv4.tcp_rmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 19. Receive window change in `sysctl`.

Step 8. To change the current send-window size value(s), use the following command on host h4's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```



```
"Host: h4"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
net.ipv4.tcp_wmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 20. Send window change in `sysctl`.

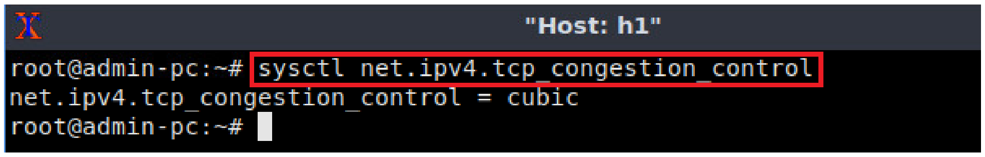
3.3 Changing congestion control algorithm in host h1 and host h2

In this part, you will set different congestion control algorithms in the host h1 and host h2 to cubic and BBR respectively. Consequently, you will have two TCP flows with different dynamic sharing the same bottleneck link.

The default congestion avoidance algorithm in the following test is cubic thus, there is no need to specify it manually.

Step 1. Verify that the congestion control algorithm is cubic by issuing the following command in host h1 terminal:

```
sysctl net.ipv4.tcp_congestion_control
```

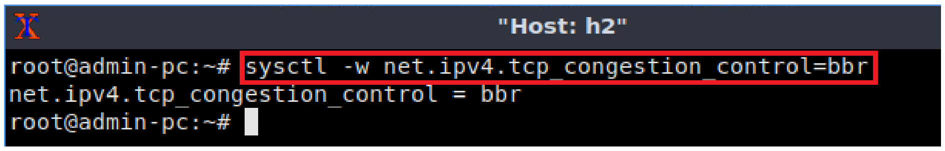


```
"Host: h1"
root@admin-pc:~# sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = cubic
root@admin-pc:~#
```

Figure 21. Verifying TCP congestion control algorithm in host h1.

Step 2. In host h2 terminal, type the following command to change the current TCP congestion control algorithm to BBR.

```
sysctl -w net.ipv4.tcp_congestion_control=bbr
```



```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=bbr
net.ipv4.tcp_congestion_control = bbr
root@admin-pc:~#
```

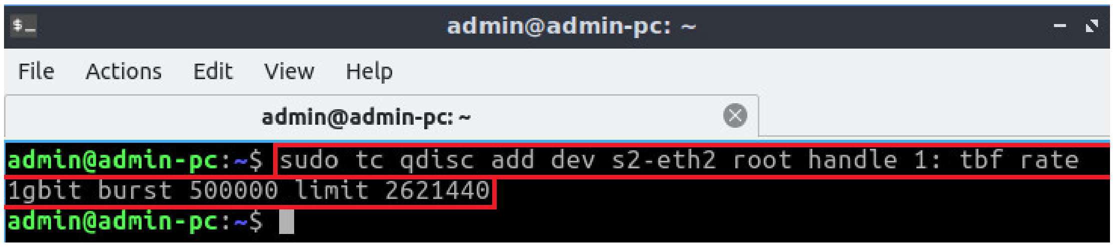
Figure 22. Changing TCP congestion control algorithm in host h2.

3.4 Setting switch S2's buffer size to BDP

Step 1. Apply `tbf` rate limiting rule on switch S2's *s2-eth2* interface. In the client's terminal, type the command below. When prompted for a password, type `password` and hit *Enter*.

- `rate`: 1gbit
- `burst`: 500,000
- `limit`: 2,621,440

```
sudo tc qdisc add dev s2-eth2 root: handle 1: tbf rate 1gbit burst 500000 limit
2621440
```



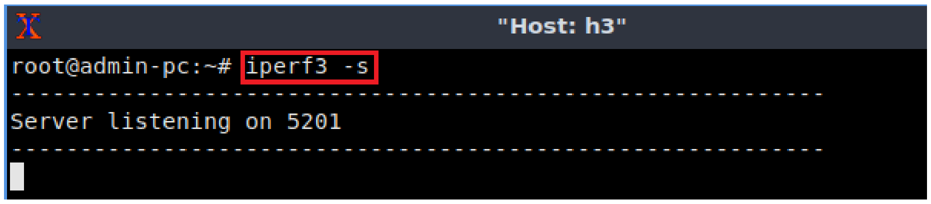
```
admin@admin-pc:~$ sudo tc qdisc add dev s2-eth2 root handle 1: tbf rate
1gbit burst 500000 limit 2621440
admin@admin-pc:~$
```

Figure 23. Limiting rate to 1 Gbps and setting the buffer size to BDP on switch S2's interface.

3.5 Throughput tests

Step 1. Launch iPerf3 in server mode on host h3's terminal.

```
iperf3 -s
```

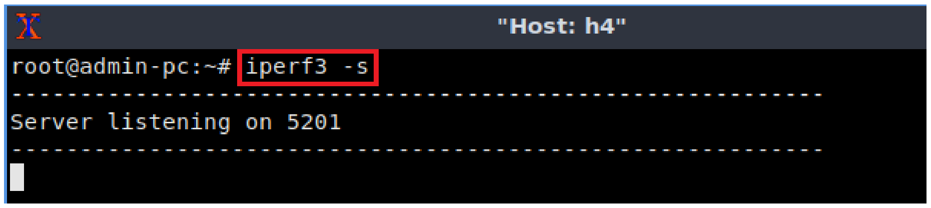


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 24. Starting iPerf3 server on host h3.

Step 2. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



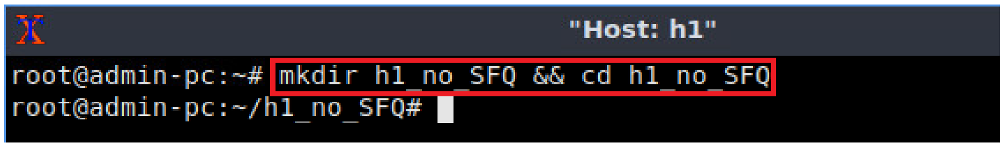
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 25. Starting iPerf3 server on host h4.

The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the *iperf3* commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 4 and Step 7 without executing them next, in Step 8 you will press *Enter* in host h1 and host h2 to execute them.

Step 3. In host h1, create a directory called *h1_no_SFQ* and navigate into it using the following command:

```
mkdir h1_no_SFQ && h1_no_SFQ
```

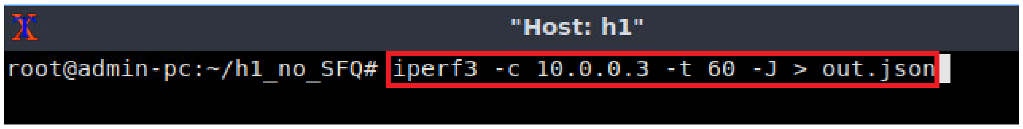


```
root@admin-pc:~# mkdir h1_no_SFQ && cd h1_no_SFQ
root@admin-pc:~/h1_no_SFQ#
```

Figure 26. Creating and navigating into directory *h1_no_SFQ*.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it. The `-J` option is used to display the output in JSON format. The redirection operator `>` is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 60 -J > out.json
```

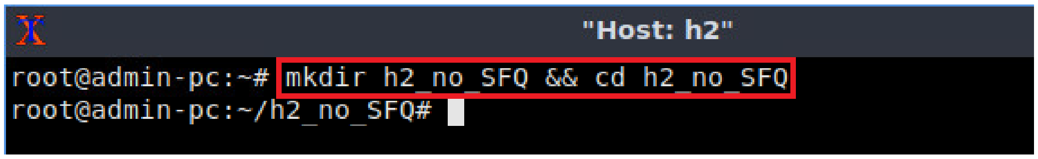


```
root@admin-pc:~/h1_no_SFQ# iperf3 -c 10.0.0.3 -t 60 -J > out.json
```

Figure 27. Running iPerf3 client on host h1.

Step 5. In host h2, create a directory called *h2_no_SFQ* and navigate into it using the following command:

```
mkdir h2_no_SFQ && h2_no_SFQ
```

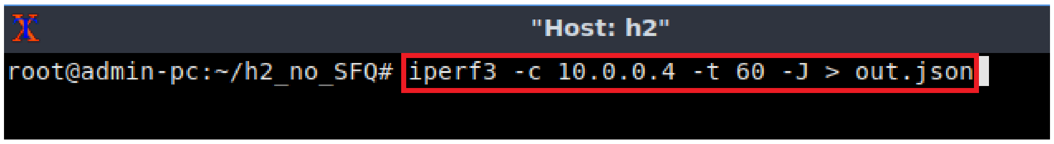


```
root@admin-pc:~# mkdir h2_no_SFQ && cd h2_no_SFQ
root@admin-pc:~/h2_no_SFQ#
```

Figure 28. Creating and navigating into directory *h2_no_SFQ*.

Step 6. Type the following iPerf3 command in host h2's terminal without executing it. The `-J` option is used to display the output in JSON format. The redirection operator `>` is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.4 -t 60 -J > out.json
```



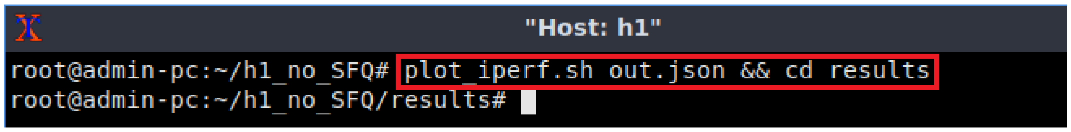
```
"Host: h2"
root@admin-pc:~/h2_no_SFQ# iperf3 -c 10.0.0.4 -t 60 -J > out.json
```

Figure 29. Running iPerf3 client on host h2.

Step 7. Press *Enter* to execute the commands shown in step 4 and step 6, first in host h1 terminal then, in host h3 terminal.

Step 8. After the iPerf3 test finishes on host h1, enter the following command.

```
plot_iperf.sh out.json && cd results
```

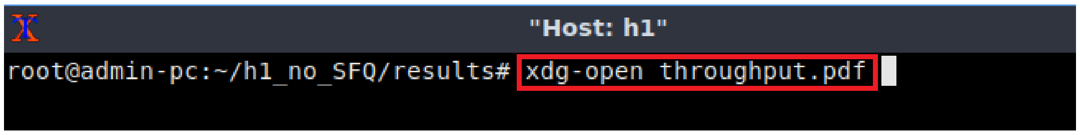


```
"Host: h1"
root@admin-pc:~/h1_no_SFQ# plot_iperf.sh out.json && cd results
root@admin-pc:~/h1_no_SFQ/results#
```

Figure 30. Generate plotting files and entering the *results* directory.

Step 9. Open the throughput file using the command below on host h1.

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



```
"Host: h1"
root@admin-pc:~/h1_no_SFQ/results# xdg-open throughput.pdf
```

Figure 31. Opening the *throughput.pdf* file.

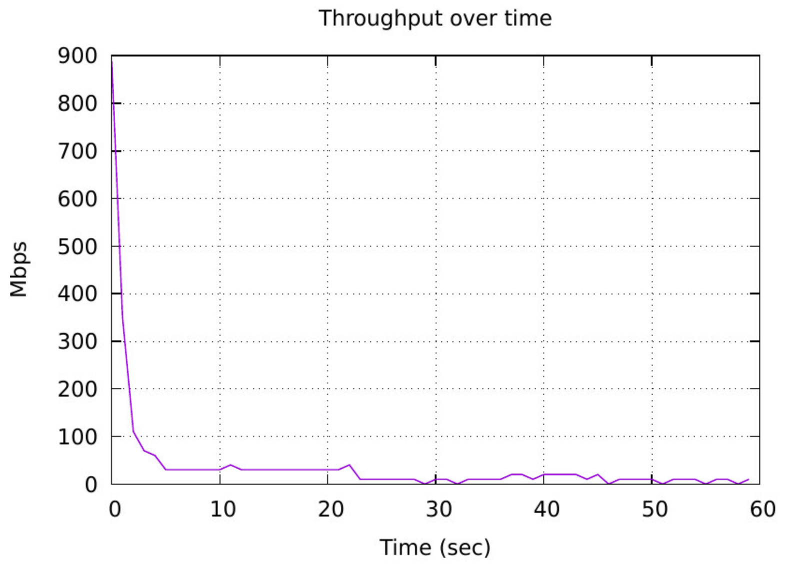
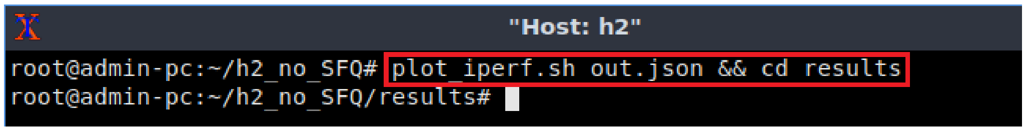


Figure 32. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is approximately 100 Mbps. It is observed that the Cubic flow collapses significantly since the link is not fairly shared with the other TCP flow.

Step 10. Close the *throughput.pdf* window then, in host h2, proceed similarly by typing the following command:

```
plot_iperf.sh out.json && cd results
```

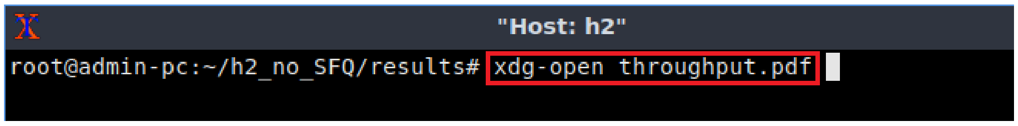


```
"Host: h2"
root@admin-pc:~/h2_no_SFQ# plot_iperf.sh out.json && cd results
root@admin-pc:~/h2_no_SFQ/results#
```

Figure 33. Generate plotting files and entering the *results* directory.

Step 11. In host h2 terminal, open the throughput file using the following command:

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



```
"Host: h2"
root@admin-pc:~/h2_no_SFQ/results# xdg-open throughput.pdf
```

Figure 34. Opening the *throughput.pdf* file.

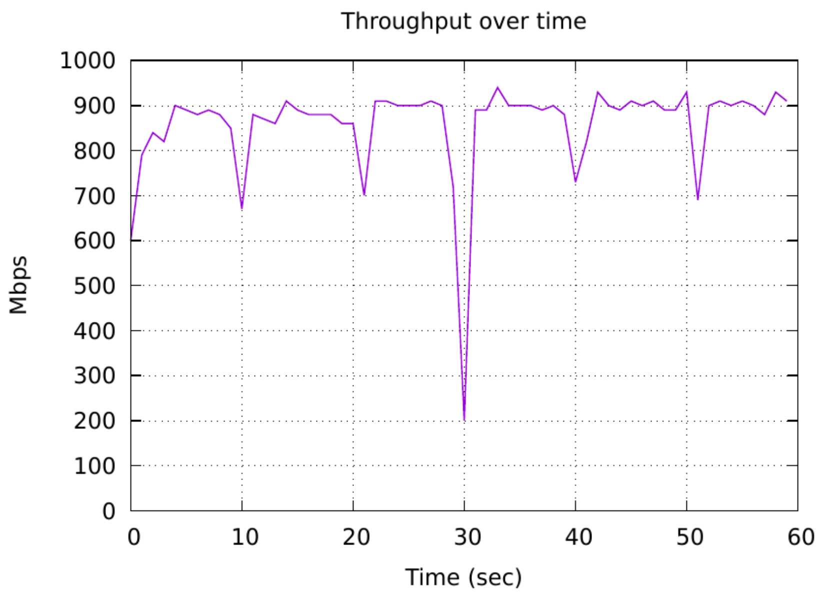


Figure 35. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 850 Mbps. It is observed that the BBR flow takes over almost the full link which upholds that the link is not fairly shared with the other TCP flow.

Step 12. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

4 Configuring SFQ on switch S2

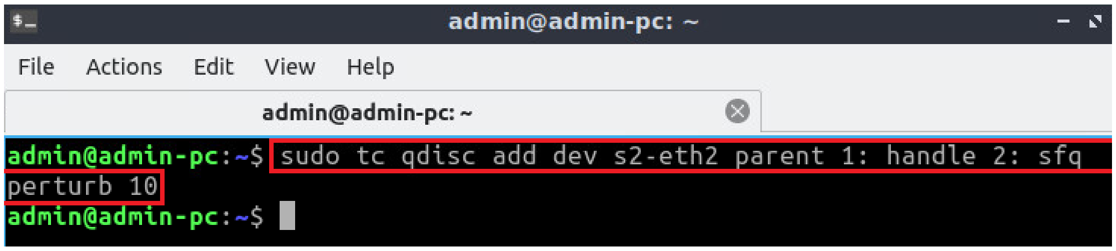
In this section, you are going to configure Stochastic Fair Queueing (SFQ) in switch S2's *s2-eth2* interface. Then, you will conduct throughput and latency measurements between host h1 and host h3.

4.1 Setting SFQ parameter on switch S2's egress interface

Step 1. Apply `sfq` rule on switch S2's *s2-eth2* interface. In the client's terminal, type the command below. When prompted for a password, type `password` and hit *Enter*.

- `perturb`: 10

```
sudo tc qdisc add dev s2-eth2 parent 1: handle 2: sfq perturb 10
```



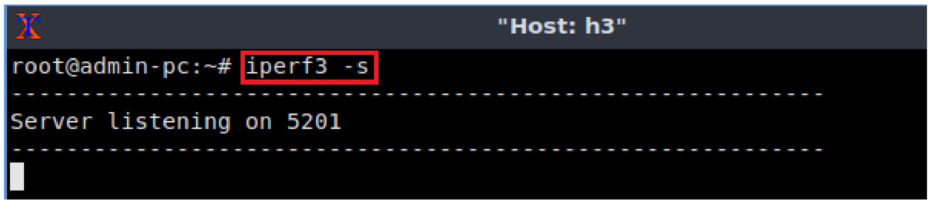
```
admin@admin-pc:~$ sudo tc qdisc add dev s2-eth2 parent 1: handle 2: sfq
perturb 10
admin@admin-pc:~$
```

Figure 36. Setting SFQ parameters on switch S2's *s2-eth2* interface.

4.2 Throughput tests

Step 1. Launch iPerf3 in server mode on host h3's terminal.

```
iperf3 -s
```

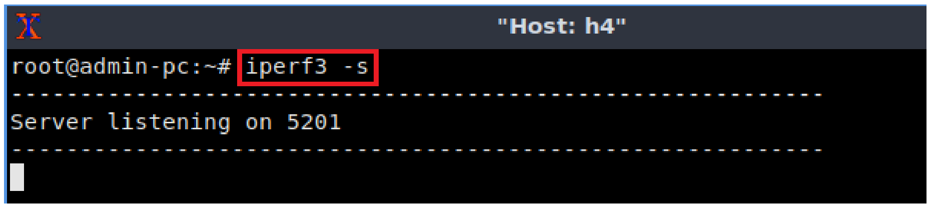


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 37. Starting iPerf3 server on host h3.

Step 2. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



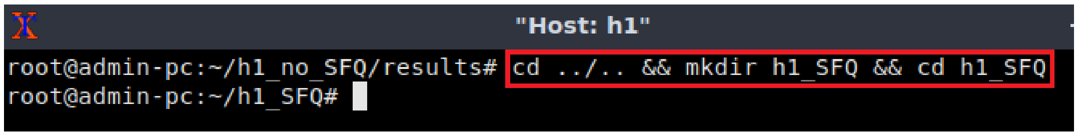
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 38. Starting iPerf3 server on host h4.

The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the *iperf3* commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 4 and Step 7 without executing them next, in Step 8 you will press *Enter* in host h1 and host h2 to execute them.

Step 3. In host h1, create a directory called *h1_SFQ* and navigate into it using the following command:

```
cd ../.. && mkdir h1_SFQ && h1_SFQ
```

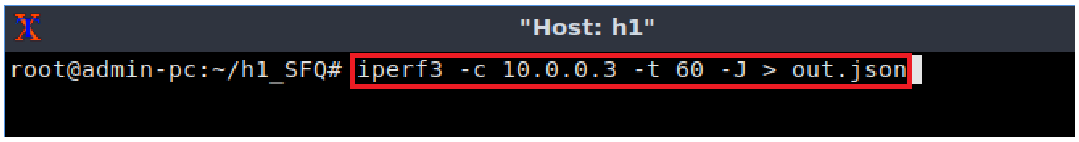


```
root@admin-pc:~/h1_no_SFQ/results# cd ../.. && mkdir h1_SFQ && cd h1_SFQ
root@admin-pc:~/h1_SFQ#
```

Figure 39. Creating and navigating into directory *h1_SFQ*.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it. The `-J` option is used to display the output in JSON format. The redirection operator `>` is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 60 -J > out.json
```

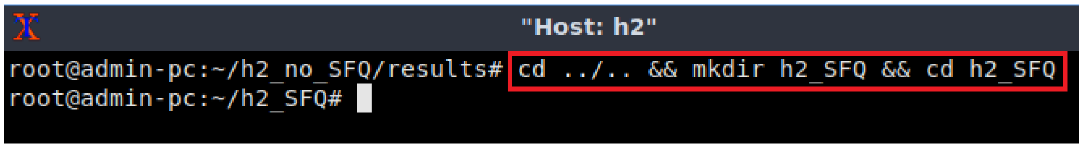


```
root@admin-pc:~/h1_SFQ# iperf3 -c 10.0.0.3 -t 60 -J > out.json
```

Figure 40. Running iPerf3 client on host h1.

Step 5. In host h2, create a directory called *h2_SFQ* and navigate into it using the following command:

```
cd ../.. && mkdir h2_SFQ && h2_SFQ
```

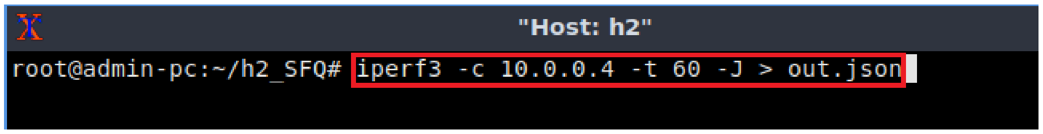


```
root@admin-pc:~/h2_no_SFQ/results# cd ../.. && mkdir h2_SFQ && cd h2_SFQ
root@admin-pc:~/h2_SFQ#
```

Figure 41. Creating and navigating into directory *h2_SFQ*.

Step 6. Type the following iPerf3 command in host h2's terminal without executing it. The `-J` option is used to display the output in JSON format. The redirection operator `>` is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.4 -t 60 -J > out.json
```



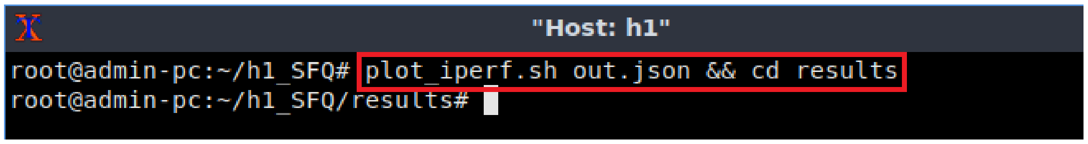
```
root@admin-pc:~/h2_SFQ# iperf3 -c 10.0.0.4 -t 60 -J > out.json
```

Figure 42. Running iPerf3 client on host h2.

Step 7. Press *Enter* to execute the commands shown in step 4 and step 6, first in host h1 terminal then, in host h3 terminal.

Step 8. After the iPerf3 test finishes on host h1, enter the following command.

```
plot_iperf.sh out.json && cd results
```

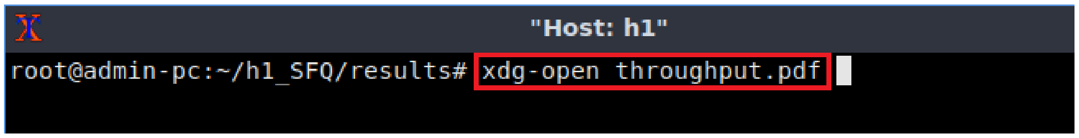


```
root@admin-pc:~/h1_SFQ# plot_iperf.sh out.json && cd results
root@admin-pc:~/h1_SFQ/results#
```

Figure 43. Generate plotting files and entering the *results* directory.

Step 9. Open the throughput file using the command below on host h1.

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



```
root@admin-pc:~/h1_SFQ/results# xdg-open throughput.pdf
```

Figure 44. Opening the *throughput.pdf* file.

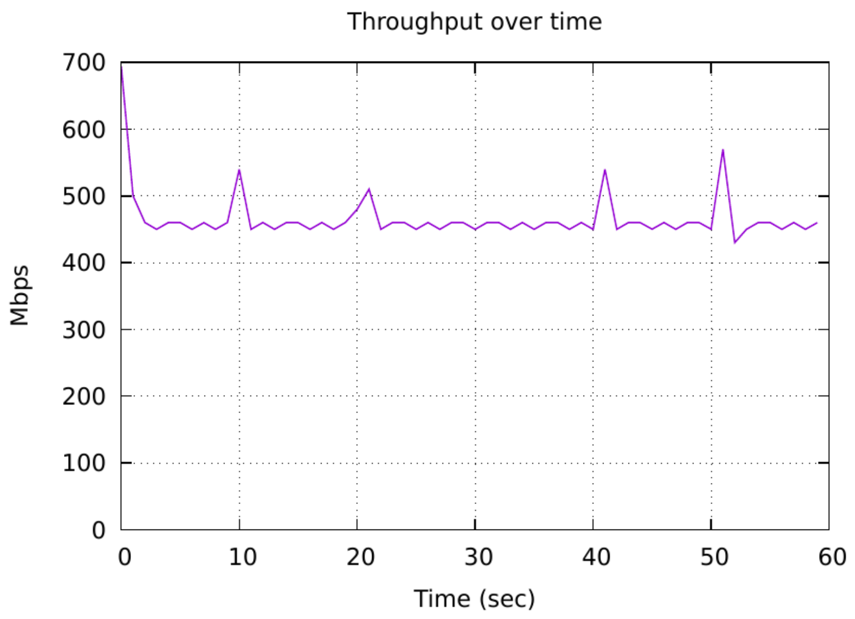
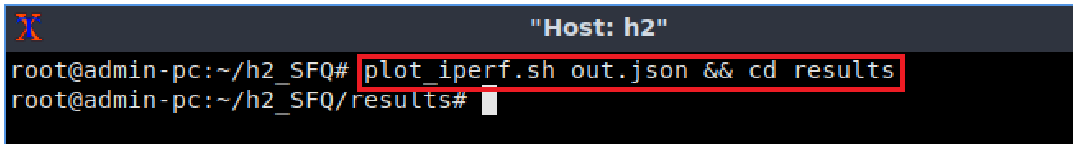


Figure 45. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is approximately 500 Mbps. It is observed that the Cubic flow uses the half part of the link. Notice also that there are spikes every 10 seconds which is consistent with the rehashing time specified by `perturb` parameter.

Step 10. In host h2, proceed similarly by typing the following command:

```
plot_iperf.sh out.json && cd results
```

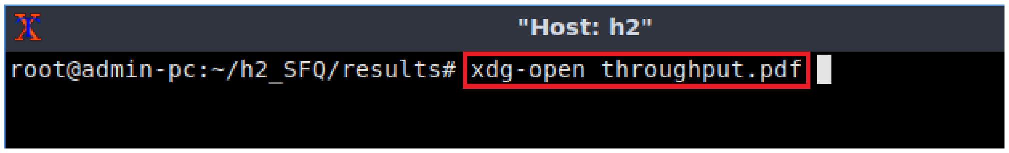


```
root@admin-pc:~/h2_SFQ# plot_iperf.sh out.json && cd results
root@admin-pc:~/h2_SFQ/results#
```

Figure 46. Generate plotting files and entering the *results* directory.

Step 11. In host h2 terminal, open the throughput file using the following command:

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



```
root@admin-pc:~/h2_SFQ/results# xdg-open throughput.pdf
```

Figure 47 Opening the *throughput.pdf* file.

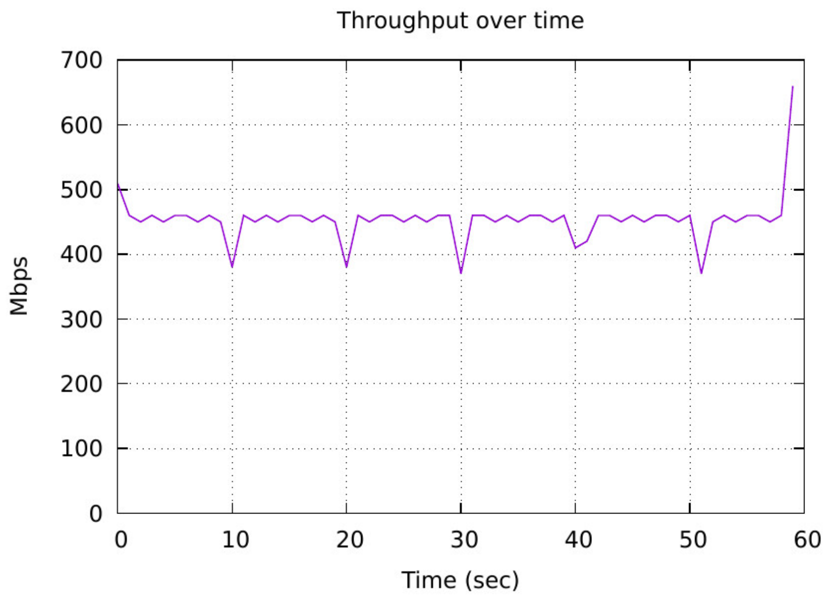


Figure 48. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is approximately 500 Mbps. It is observed that the BBR can fairly coexist with a Cubic flow since the dynamic of both flows do not interact as a result of SFQ is configured in switch S2's *s2-eth2* interface.

Step 12. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

This concludes Lab 17. Stop the emulation and then exit out of MiniEdit.

References

1. C. Semeria, “Supporting differentiated service classes: queue scheduling disciplines,” Juniper networks, pp. 11-14, 2001.
2. P. McKenney. “Stochastic fairness queueing,” In *Proceedings. IEEE INFOCOM90:* Ninth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 733-740, IEEE, 1990.
3. J. Kurose, K. Ross, “Computer networking, a top-down approach,” 7th Edition, Pearson, 2017.
4. C. Villamizar, C. Song, “High performance TCP in ansnet,” ACM Computer Communications Review, vol. 24, no. 5, pp. 45-60, Oct. 1994.
5. R. Bush, D. Meyer, “Some internet architectural guidelines and philosophy,” Internet Request for Comments, RFC Editor, RFC 3439, Dec. 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3439.txt.
6. J. Gettys, K. Nichols, “Bufferbloat: dark buffers in the internet,” Communications of the ACM, vol. 9, no. 1, pp. 57-65, Jan. 2012.
7. 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.