



UNIVERSITY OF
SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 18: Controlled Delay (CoDel) Active Queue Management

Document Version: **11-18-2019**



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	4
1.1 CoDel active queue management	4
1.2 Fair queueing CoDel active queue management	6
2 Lab topology.....	8
2.1 Starting host h1, host h2, host h3 and host h4	9
2.2 Emulating high-latency WAN.....	10
2.3 Testing connection	11
3 Testing the throughput of two competing TCP flows.....	12
3.1 Bandwidth-delay product (BDP) and hosts' TCP buffer size	12
3.2 Modifying hosts' buffer size	13
3.3 Changing congestion control algorithm in host h1 and host h2	16
3.4 Setting switch S2's buffer size to BDP	16
3.5 Throughput tests	17
4 Configuring CoDel on switch S2	24
4.1 Setting CoDel parameter on switch S2's egress interface.....	24
4.2 Throughput tests	25
5 Configuring Fq_CoDel on switch S2	31
5.1 Setting Fq_CoDel parameters on switch S2's egress interface	31
5.2 Throughput test.....	32
6 Changing the bandwidth to 100Mbps	39
6.1 Throughput and latency tests.....	39
References	46

Overview

This lab explains how Controlled Delay (CoDel) Active Queue Management (AQM) algorithm is used to manage outgoing TCP traffic in order to achieve low end-to-end latency and ensure fairness. Additionally, it is presented FQ_CoDel (Fair Queueing with Controlled Delay) AQM, a combination of fair queuing and CoDel algorithms which is aimed to mitigate bufferbloat and ensure fairness. Along this lab, throughput, latency and queue occupancy measurements are conducted in an emulated high-latency network showing the features of both algorithms.

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 features of AQM algorithms.
3. Explain how CoDel algorithm contains the queue length in order to avoid bufferbloat.
4. Observe how FQ_CoDel ensures low end-to-end latency and fairness.
5. Visualize the benefits of isolating the dynamic of competing TCP 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.
2. Section 2: Lab topology.
3. Section 3: Testing the throughput of two competing TCP flows
4. Section 4: Configuring CoDel on switch S2.
5. Section 5: Configuring FQ_CoDel on switch S2.
6. Section 6: Changing the bandwidth to 100Mbps.

1 Introduction

The persistently problem of bufferbloat has been discussed since the early 80's. Bufferbloat is caused by oversizing router's queues that hold traffic that cannot be immediately forwarded consequently, unnecessary latency is experienced. The deployment of Active Queue Management (AQM) algorithms started being notorious in 1998. The aim of those algorithms is to solve the increasing problem of bufferbloat. Despite this awareness, the problem has only gotten worse as growth in memory density per Moore's Law fueled an exponential increase in buffer pool size. Efforts to deploy AQM algorithms have been frustrated by difficult configuration and negative impact on network utilization. This bufferbloat problem has become more and more important throughout the Internet but particularly at the consumer edge. Queue management has become more critical due to increased consumer use of the Internet, mixing large video transactions with time-critical VoIP and gaming¹.

Network buffers are designed to absorb the packet bursts that occur naturally in statistically multiplexed networks. Buffers helpfully absorb the queues created by reasonable packet network behavior such as short-term mismatches in traffic arrival and departure. Unfortunately, other less useful network behaviors can cause queues to fill, and their effects are not nearly as benign. Discussion of these issues and the reason why the solution is not simply creating smaller buffers. To understand queue management, it is critical to understand the difference between the necessary, useful "good" queue and the counterproductive "bad" queue. An effective AQM algorithm remediates bufferbloat at a bottleneck without affecting the hops where buffers are not bloated. However, the development and deployment of AQM algorithms are frequently subject to misconceptions about the cause of packet queues in network buffers.

1.1 CoDel active queue management

In order to mitigate the bufferbloat problem, it is presented Controlled Delay² (CoDel), an innovative Active Queue Management (AQM) that adapts to changing links rates and it is suitable for deployment and experimentation in Linux-based routers. The goal of CoDel³ is to contain the queuing latency while maximizing the throughput. CoDel does not require any parameters tuning and it has been designed to work across a wide range of conditions with different links and round-trip times. Figure 1 illustrate a scenario where CoDel AQM is used to manage the queue. Firstly, a timestamp is added to every incoming packet. Then, by measuring the departure time of every packet in the queue, it is determined for how long a packet was waiting in the queue. Consequently, CoDel algorithm determines whether the enqueued packets are going to be dropped or not.

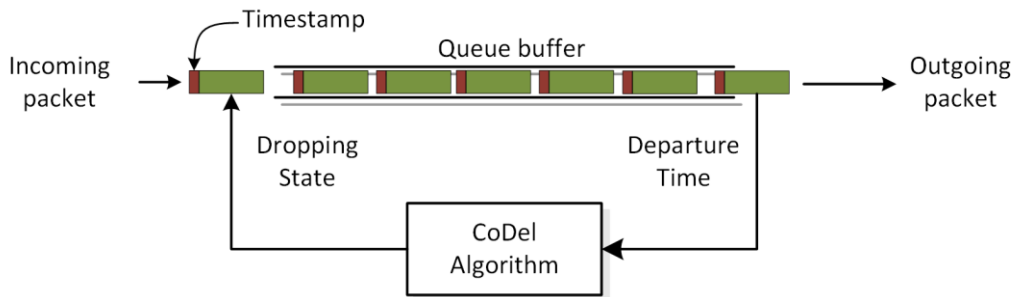


Figure 1. Buffer managed by CoDel AQM.

The algorithm depicted in Listing 1 shows in more detail how CoDel works⁴: as mentioned before, timestamp is added to each packet at the ingress side of the queue in order to measure the packet departure time at the egress side when packets are dequeued. The departure time is then compared to a target delay `TARGET` (5ms by default), if it is below the target the packet is forwarded, otherwise the algorithm starts a timer and forwards the packet. When packets are dequeued, CoDel checks the departure time, and if it gets below the target, the timer is stopped. However, if the timer reaches the value of interval `INTERVAL` (100ms by default), CoDel enters the dropping state which is left when the departure time gets below the target delay. During this state, whenever the interval timer expires a packet is dropped, the timer is reset, and the next timer duration is reconfigured. Thus, the longer the departure time stays above the target delay, the higher the packet dropping frequency. Finally, as soon as the measured departure time of an outgoing packet gets below the target, the dropping state is left, and interval is restored to its default value.

```
Paket p
StatefulObject s

if(p.qdelay < TARGET)
    s.dropping = false
    count = 0
if(s.dropping == false)
    s.dropping = true
    s.drop_next_packet = now + INTERVAL
if(s.dropping && s.drop_next_packet >= now)
    drop()
    count++
    s.drop_next_packet = now + INTERVAL / sqrt(now)
```

Listing 1. Simplified CoDel pseudocode.

In summary, CoDel algorithm considers three scenarios:

- If the queueing delay is below `TARGET`, a packet is never dropped.
- If `TARGET` is exceeded by more than `INTERVAL` time units, the first packet will be dropped.
- From `now`, the interval between dropping packets is getting smaller, until `TARGET` delay is reached.

The complete algorithm considers other factors like the duration since the last dropping phase. The Linux implementation used in this lab is based on the full algorithm.

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

```
tc qdisc [add | ...] dev [dev_id] root codell limit [PACKETS] target [SECONDS]
interval [SECONDS] ecn|noecn ce_threshold
```

- `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.
- `codell`: this parameter enables the Controlled Delay (CoDel) algorithm.
- `limit`: this parameter specifies hard limit on the real queue size in packets. When this limit is reached, incoming packets are dropped. If the value is lowered, packets are dropped so that the new limit is met. The default value is 1000 packets.
- `target`: denotes the acceptable minimum standing/persistent queue delay. This minimum delay is identified by tracking the local minimum queue delay that packets experience. The default and recommended value is 5ms.
- `interval`: this parameter is intended to ensure that the measured minimum delay does not become too stale. The minimum delay must be experienced in the last epoch of length `interval`. It should be set on the order of the worst-case RTT through the bottleneck to give endpoints sufficient time to react. The default value is 100ms.
- `ecn|noecn`: it is used to mark packets instead of dropping them. If `ecn` has been enabled, `noecn` can be used to turn it off and vice-a-versa. By default, `ecn` is turned off.
- `ce_threshold`: sets a threshold above which all packets are marked with ECN Congestion Experienced. This is useful for DCTCP-style congestion control algorithms that require marking at very shallow queueing thresholds.

1.2 Fair queueing CoDel active queue management

FQ_CoDel (Fair Queuing Controlled Delay) is a queuing discipline that combines Fair Queuing with the CoDel AQM scheme. FQ_CoDel uses a stochastic model to classify incoming packets into different flows. It is aimed to provide a fair share of the bandwidth to all the flows using the queue.

Figure 2 shows that FQ_CoDel consists of two logical parts: 1) the scheduler, which selects which queue to dequeue a packet from, and 2) The CoDel AQM which works on each of the queues. Since FQ_CoDel mixes packets from multiple flows, it reduces the impact of bursty traffic. It also provides isolation for applications namely DNS, web and videoconferencing traffic. Additionally, it improves network utilization by keeping the

queue lengths short and it can be implemented in a memory and CPU efficient fashion across a wide range of hardware.

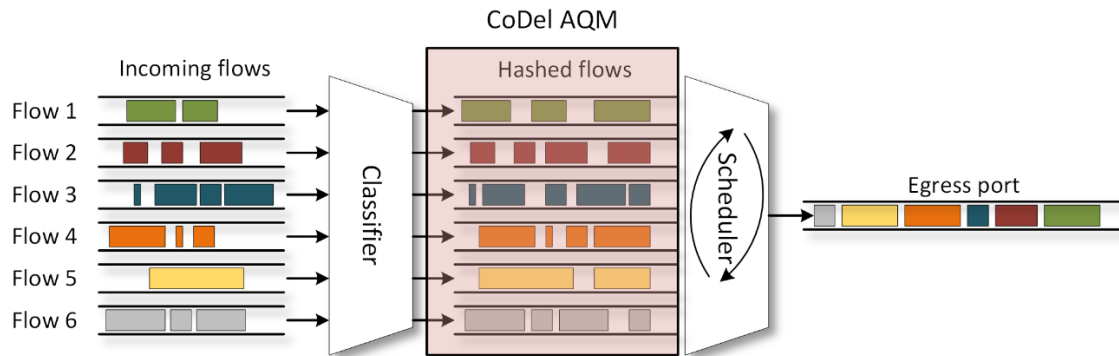


Figure 2. Buffer managed by FQ_CoDel AQM.

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

```
tc qdisc [add | ...] dev [dev_id] root fq_codel limit [PACKETS] target
[SECONDS] flows [NUMBER] interval [SECONDS] quantum [BYTES] ecn|noecn
ce_threshold
```

- `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.
- `fq_codel`: this parameter enables the Fair Queueing Controlled Delay (FQ_CoDel) algorithm.
- `limit`: this parameter specifies hard limit on the real queue size in packets. When this limit is reached, incoming packets are dropped. If the value is lowered, packets are dropped so that the new limit is met. The default value is 1000 packets.
- `target`: denotes the acceptable minimum standing/persistent queue delay. This minimum delay is identified by tracking the local minimum queue delay that packets experience. The default and recommended value is 5ms.
- `flows`: this parameter specifies the number of flows into which the incoming packets are classified. Due to the stochastic nature of hashing, multiple flows may end up being hashed into the same slot. Newer flows have priority over older ones. This parameter can be set only at load time since memory has to be allocated for the hash table. Default value is 1024.
- `interval`: this parameter is intended to ensure that the measured minimum delay does not become too stale. The minimum delay must be experienced in the last epoch of length `interval`. It should be set on the order of the worst-case

RTT through the bottleneck to give endpoints sufficient time to react. The default value is 100ms.

- `quantum`: denotes the number of bytes used as deficit in the fair queuing algorithm. The default value is 1514 bytes which corresponds to the Ethernet MTU plus the hardware header length of 14 bytes.
- `ecn|noecn`: it is used to mark packets instead of dropping them. If `ecn` has been enabled, `noecn` can be used to turn it off and vice-a-versa. By default, `ecn` is turned off.
- `ce_threshold`: sets a threshold above which all packets are marked with ECN Congestion Experienced. This is useful for DCTCP-style congestion control algorithms that require marking at very shallow queuing thresholds.

In this lab, we will use `codell` and `fq_codel` AQM algorithms to control the queue size at the egress port of a router.

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.

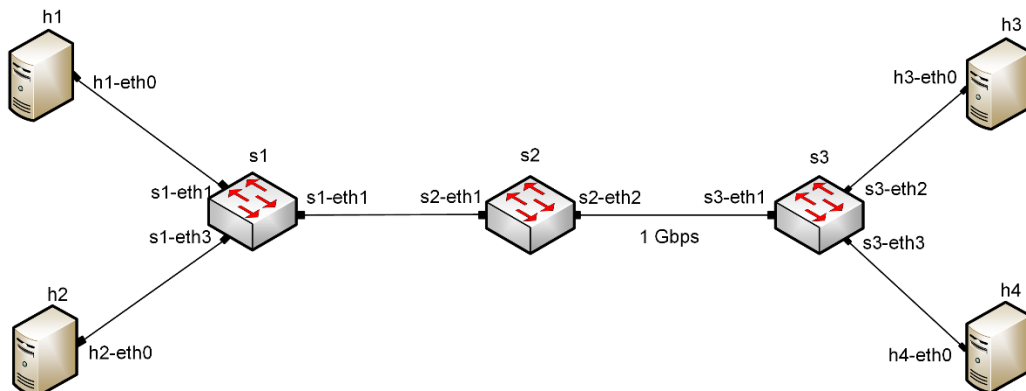


Figure 3. Lab topology.

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`.

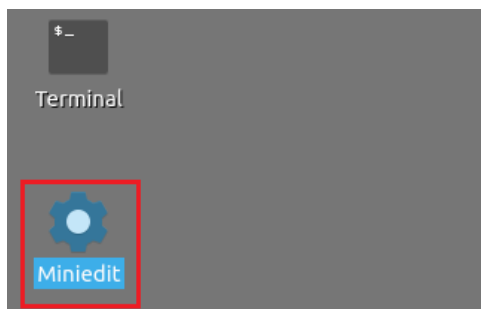


Figure 4. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 18.mn* topology file and click on *Open*.

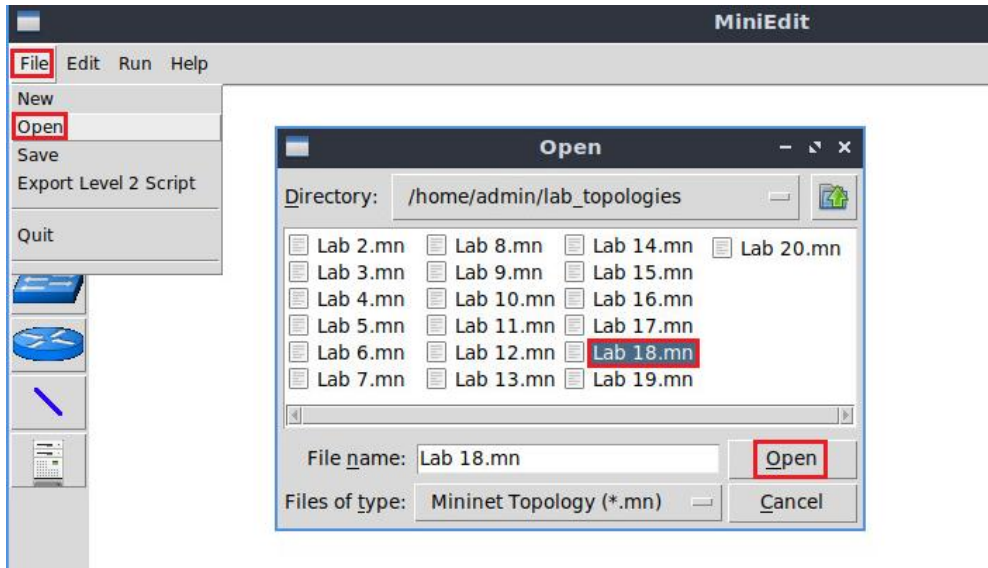


Figure 5. MiniEdit's *Open* dialog.

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.

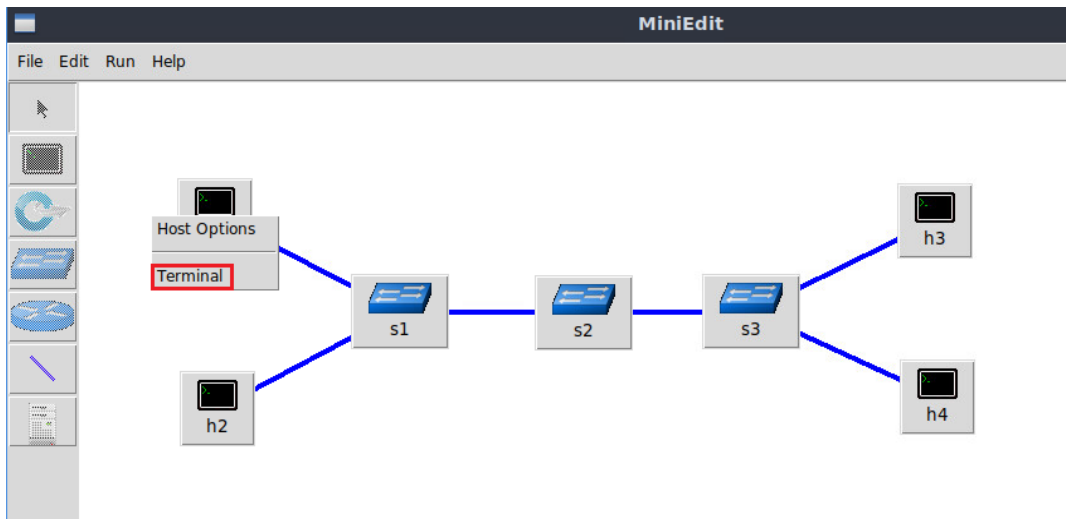


Figure 7. Opening a terminal on host h1.

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.

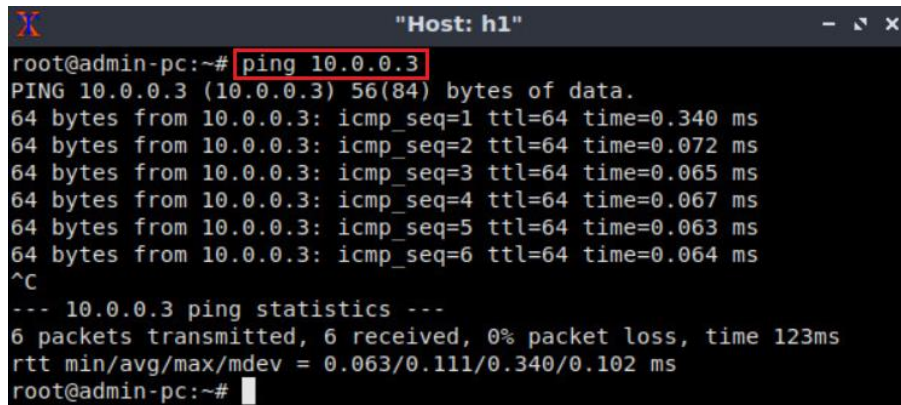


Figure 8. Connectivity test using `ping` command.

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.

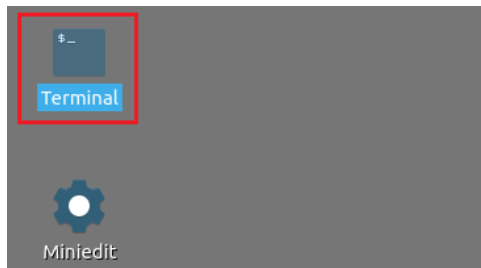


Figure 9. Shortcut to open a Linux terminal.

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
```

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`.

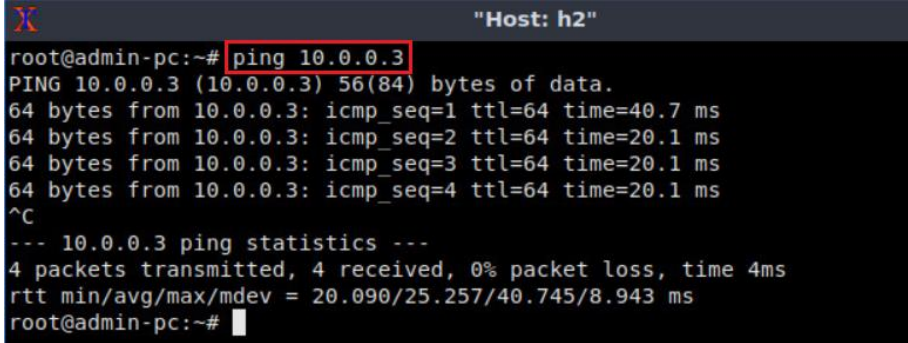
Step 1. On the terminal of host h1, type `ping 10.0.0.3`. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test. Host h1 (10.0.0.1) sent four packets to host h3 (10.0.0.3), successfully receiving responses back.

```
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`.



```

Host: h2
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 $20 \cdot \text{BDP}$ 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 however, the buffer will be intentionally oversized to $20 \cdot \text{BDP}$ therefore, you should expect bufferbloat. 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.

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

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

$$\text{BDP} = 1,000,000,000 \cdot 0.02 = 20,000,000 \text{ bits}$$

$$= 2,500,000 \text{ bytes} \approx 2.5 \text{ Mbytes}$$

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

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

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 $20 \cdot \text{BDP}$ (52,428,800).

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 $20 \cdot \text{BDP}$ 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 $20 \cdot \text{BDP}$ 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.

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

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

$$20 \cdot \text{BDP} = 20 \cdot 2,621,440 \text{ bytes} = 52,428,800 \text{ bytes}$$

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 52,428,800 (maximum).

```
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 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. 52,428,800 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 52,428,800 (maximum).

```
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 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 52,428,800 (maximum).

```
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 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 52,428,800 (maximum).

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

```

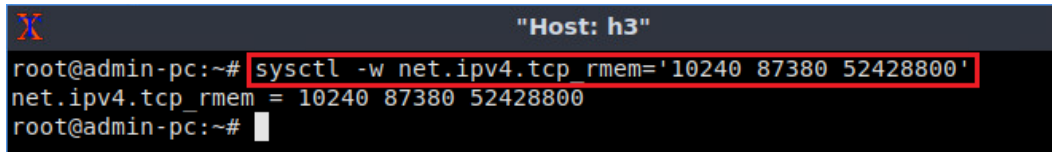
X "Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#

```

Figure 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 52,428,800 (maximum).

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



```

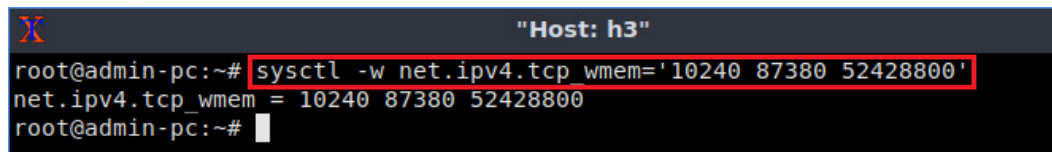
X                                     "Host: h3"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#

```

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 52,428,800 (maximum).

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



```

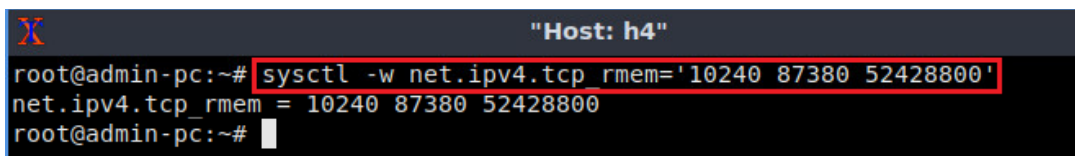
X                                     "Host: h3"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#

```

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 52,428,800 (maximum).

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



```

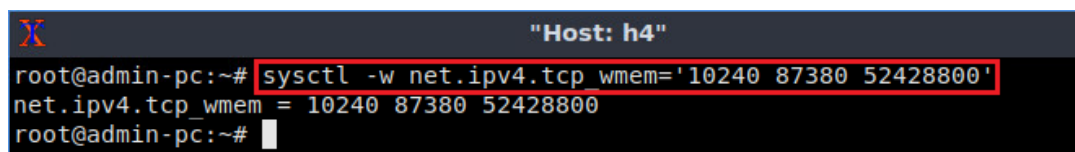
X                                     "Host: h4"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#

```

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 52,428,800 (maximum).

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



```

X                                     "Host: h4"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#

```

Figure 20. Send window change in `sysctl`.

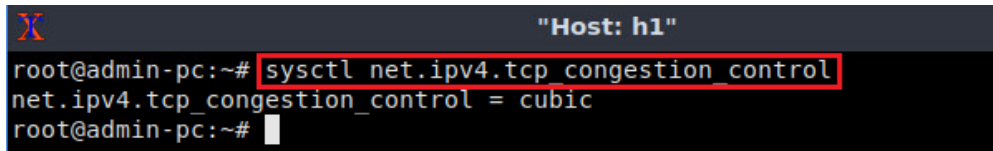
3.3 Changing congestion control algorithm in host h1 and host h2

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
```



```

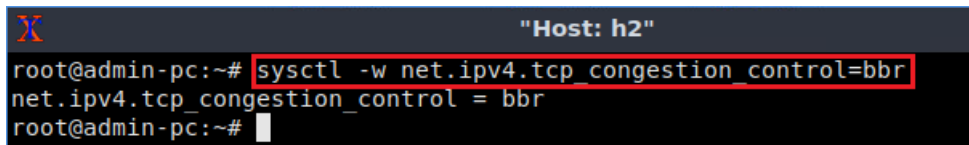
Host: h1
root@admin-pc:~# sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = cubic
root@admin-pc:~#

```

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=bbp
```



```

Host: h2
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=bbp
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 `tbpf` 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`: 26,214,400

```
sudo tc qdisc add dev s2-eth2 root: handle 1: tbf rate 1gbit burst 500000 limit 26214400
```



```
admin@admin-pc: ~
File Actions Edit View Help
admin@admin-pc: ~
admin@admin-pc:~$ sudo tc qdisc add dev s2-eth2 root handle 1: tbf rate
1gbit burst 500000 limit 26214400
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
```

```
Host: h3
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
```

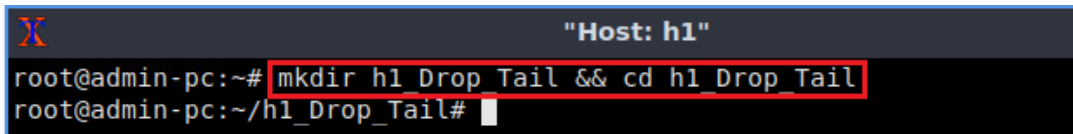
```
Host: h4
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----
```

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 6 without executing them next, in Step 7 you will press *Enter* in host h1 and host h2 to execute them.

Step 3. In host h1, create a directory called *h1_Drop_Tail* and navigate into it using the following command:

```
mkdir h1_Drop_Tail && cd h1_Drop_Tail
```



```

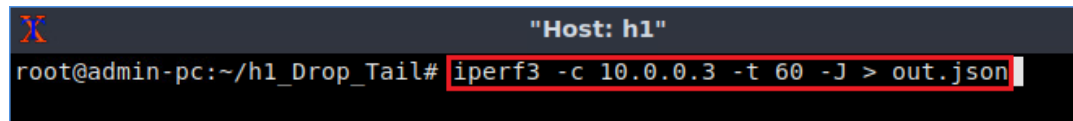
X "Host: h1"
root@admin-pc:~# mkdir h1_Drop_Tail && cd h1_Drop_Tail
root@admin-pc:~/h1_Drop_Tail#

```

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
```



```

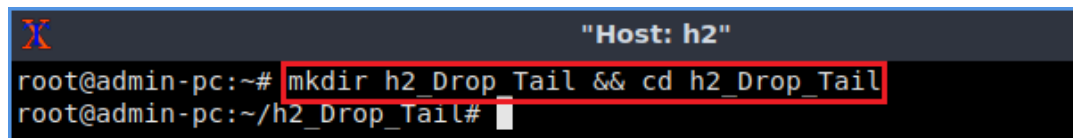
X "Host: h1"
root@admin-pc:~/h1_Drop_Tail# iperf3 -c 10.0.0.3 -t 60 -J > out.json

```

Figure 27. Running iPerf3 client on host h1.

Step 5. In host h2, create a directory *h1_Drop_Tail* and navigate into it using the following command:

```
mkdir h2_Drop_Tail && h2_Drop_Tail
```



```

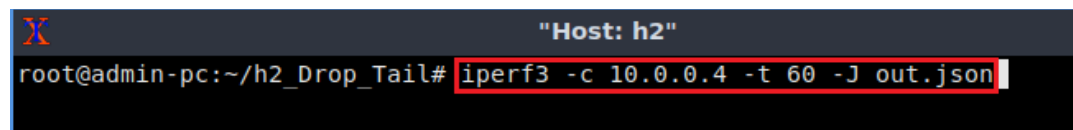
X "Host: h2"
root@admin-pc:~# mkdir h2_Drop_Tail && cd h2_Drop_Tail
root@admin-pc:~/h2_Drop_Tail#

```

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
```



```

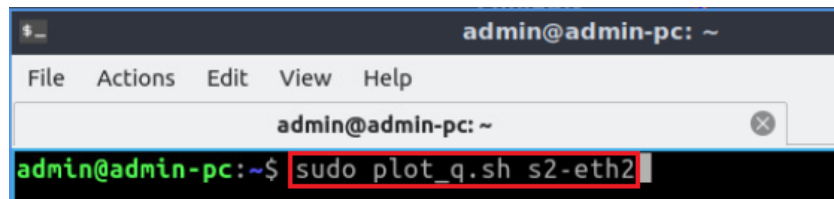
X "Host: h2"
root@admin-pc:~/h2_Drop_Tail# iperf3 -c 10.0.0.4 -t 60 -J out.json

```

Figure 29. Running iPerf3 client on host h2.

Step 7. In the Client's terminal, type the command below to plot the switch's queue in real-time. When prompted for a password, type `password` and hit *Enter*.

```
sudo plot_q.sh s2-eth2
```



```
admin@admin-pc: ~  
File Actions Edit View Help  
admin@admin-pc: ~  
admin@admin-pc:~$ sudo plot_q.sh s2-eth2
```

Figure 30. Plotting the queue occupancy on switch S2's s2-eth2 interface.

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through s2-eth2 interface on switch S2, the queue occupancy is constantly 0.

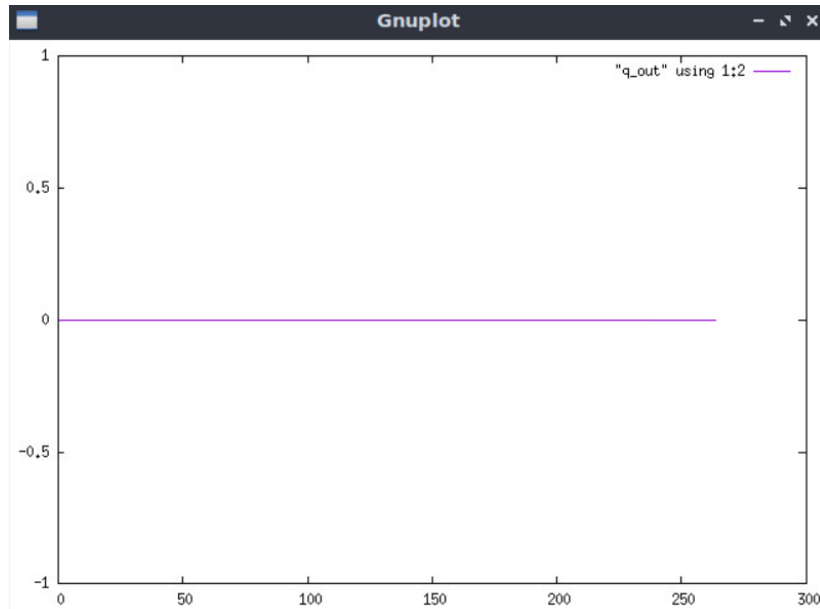


Figure 31. Queue occupancy on switch S2's s2-eth2 interface.

Step 8. Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal.

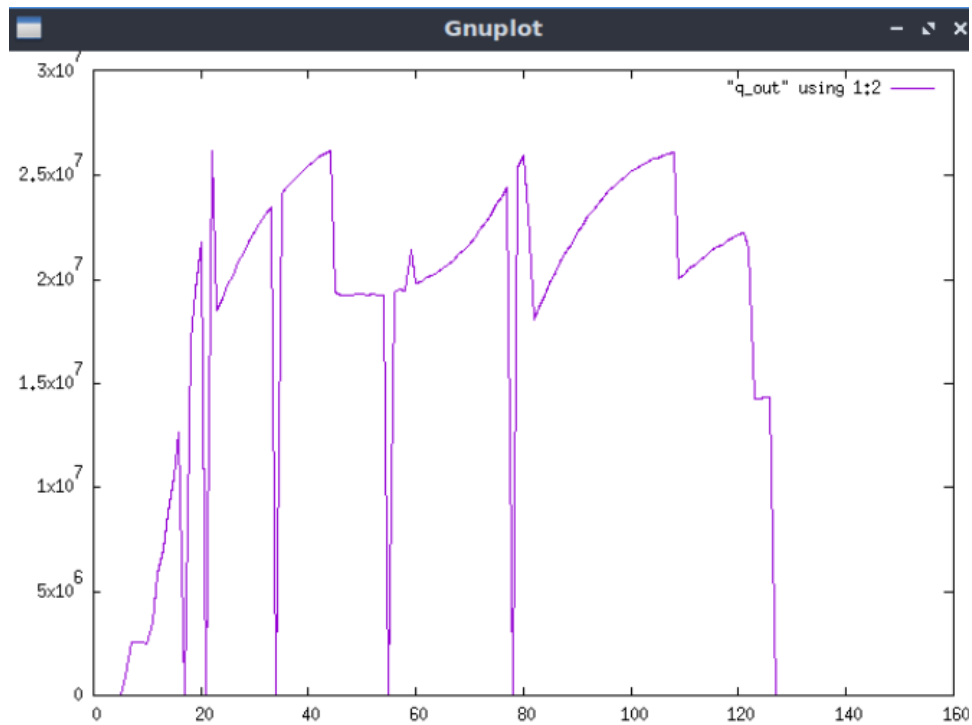


Figure 32. Queue occupancy on switch S2's s2-eth2 interface.

The graph above shows that the queue occupancy peaked at $2.5 \cdot 10^7$, which is the maximum buffer size we configure on the switch.

Step 9. In the queue plotting window, press the `S` key on your keyboard to stop plotting the queue.

Step 10. After the iPerf3 test finishes on host h1, enter the following command:

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

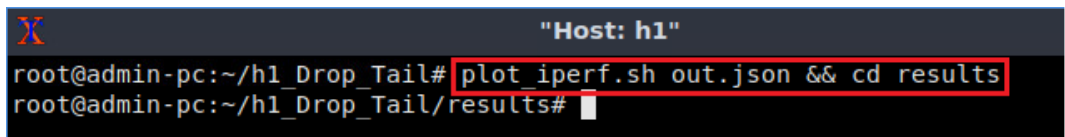


Figure 33. Generate plotting files and entering the *results* directory.

Step 11. Open the throughput file using the command below on host h1.

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

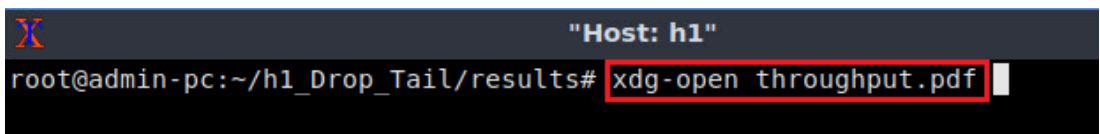


Figure 34. Opening the *throughput.pdf* file.

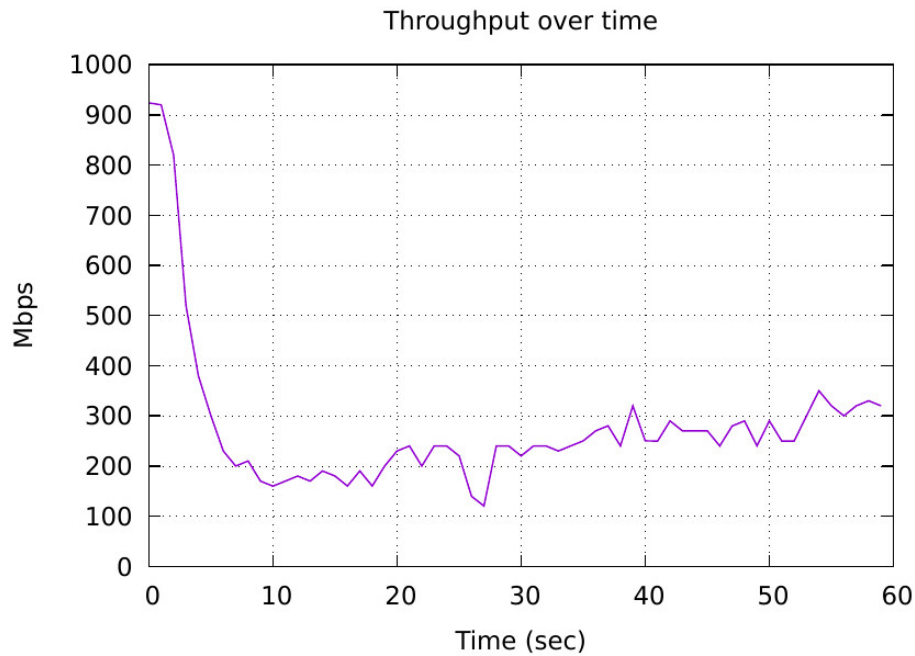
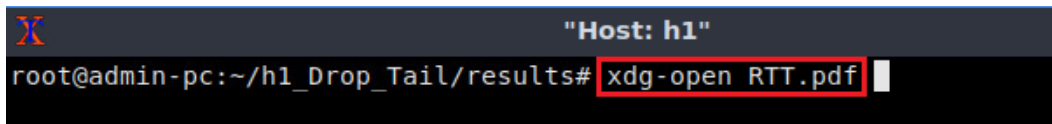


Figure 35. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 250 Mbps. It is observed that the Cubic flow collapses significantly since the link is not fairly shared with the other TCP flow.

Step 12. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

```
xdg-open RTT.pdf
```

Figure 36. Opening the *RTT.pdf* file.

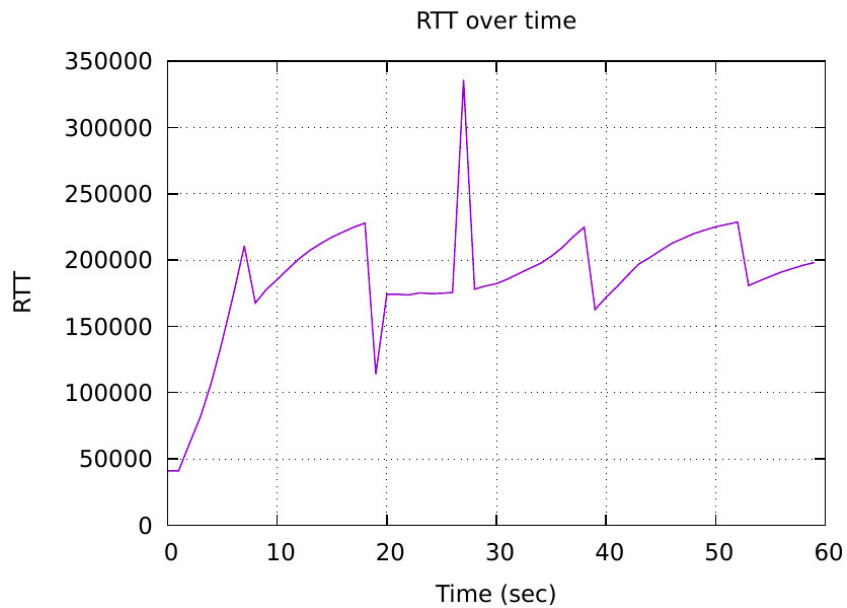


Figure 37. Measured round-trip time.

The graph above shows that the RTT was approximately 200,000 microseconds (200ms). The output shows that there is bufferbloat as the average latency is at least ten times greater than the configured delay (20ms).

Step 13. Close the *RTT.pdf* window then, in host h2, proceed similarly by typing the following command:

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

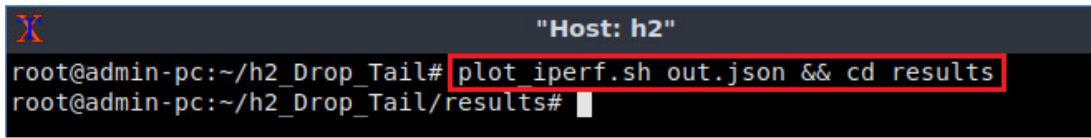


Figure 38. Generate plotting files and entering the *results* directory.

Step 14. In host h2 terminal, open the throughput file using the following command:

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

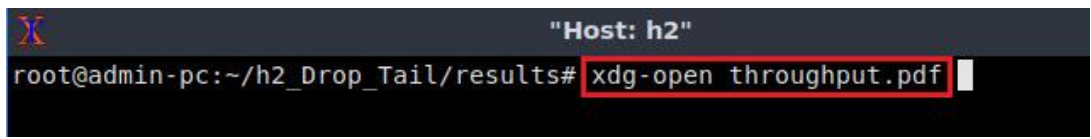


Figure 39. Opening the *throughput.pdf* file.

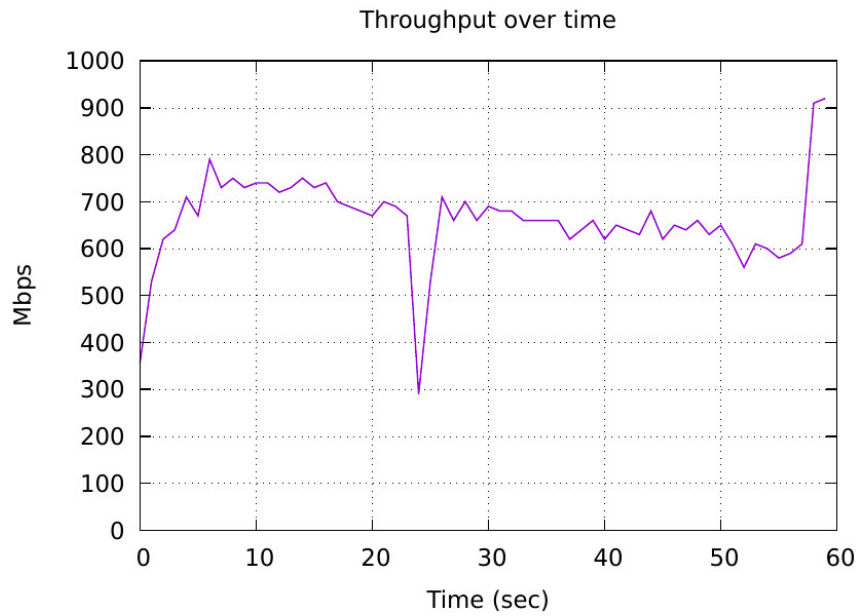


Figure 40. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 700 Mbps. It is observed that the BBR flow uses the major part of the available bandwidth which shows that the link is not fairly shared with the other TCP flow.

Step 15. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

```
xdg-open RTT.pdf
```

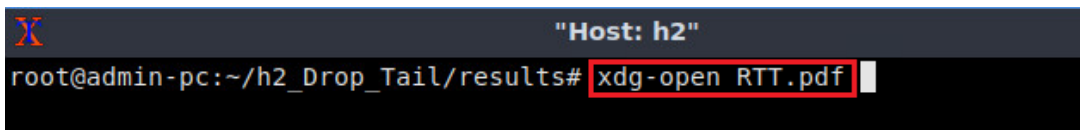


Figure 41. Opening the *RTT.pdf* file.

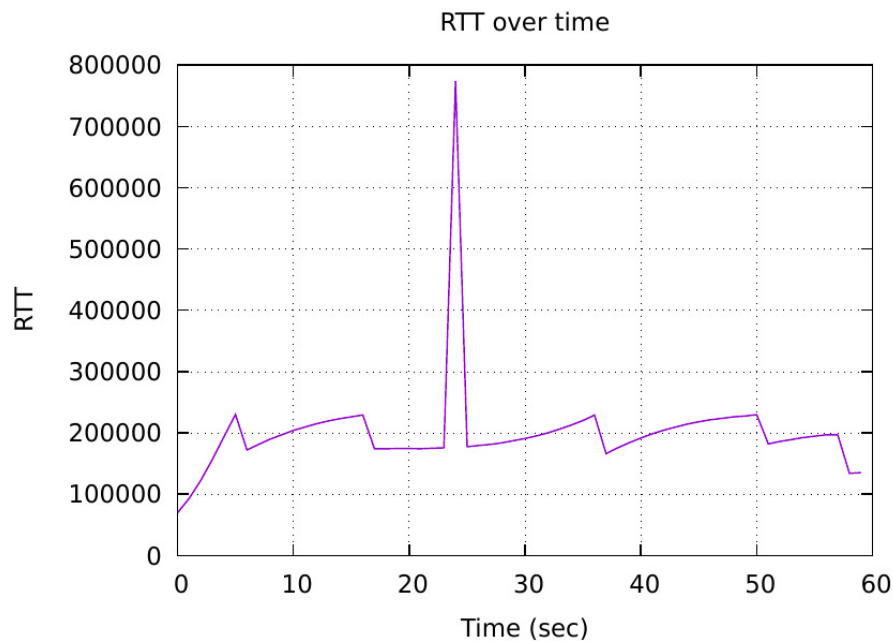


Figure 42. Measured round-trip time.

The graph above shows that the RTT was approximately 200,000 microseconds (200ms) as well. The output shows that there is bufferbloat as the average latency is at least ten times greater than the configured delay (20ms).

Step 16. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

4 Configuring CoDel on switch S2

In the previous section you have observed bufferbloat, due to oversizing switch S2's *s2-eth2* interface. In this section, you are going to configure CoDel AQM in switch S2's *s2-eth2* interface. Then, you will conduct throughput and latency measurements between the hosts.

4.1 Setting CoDel parameter on switch S2's egress interface

Step 1. Apply `codell` 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*.

- `limit`: 17476
- `target`: 5ms
- `interval`: 100ms

```
sudo tc qdisc add dev s2-eth2 parent 1: handle 2: codell limit 17476 target 5ms interval 100ms
```



```
admin@admin-pc: ~
File Actions Edit View Help
admin@admin-pc: ~
admin@admin-pc:~$ sudo tc qdisc add dev s2-eth2 parent 1: handle 2: codel
limit 17476 target 5ms interval 100ms
admin@admin-pc:~$
```

Figure 43. Setting CoDel 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
```

```
Host: h3
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----
```

Figure 44. Starting iPerf3 server on host h3.

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

```
iperf3 -s
```

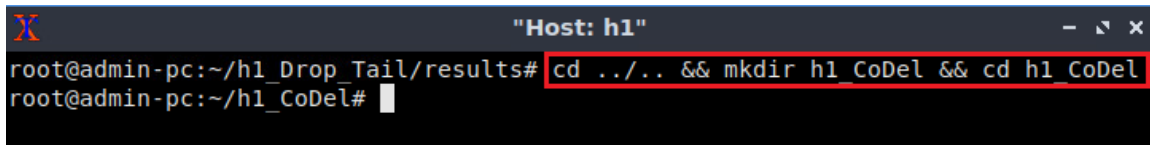
```
Host: h4
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----
```

Figure 45. 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 6 without executing them next, in Step 7 you will press *Enter* in host h1 and host h2 to execute them.

Step 3. In host h1, exit from the previous folders then, create a directory called *h1_CoDel* and navigate into it using the following command:

```
cd ../../ && mkdir h1_CoDel && cd h1_CoDel
```



```

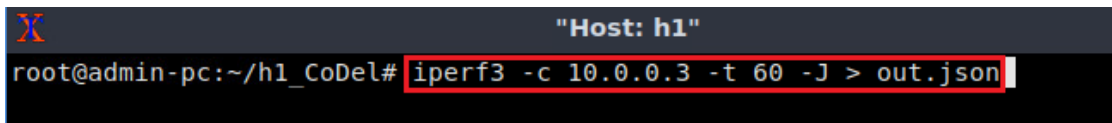
Host: h1
root@admin-pc:~/h1_Drop_Tail/results# cd ../../ && mkdir h1 CoDel && cd h1 CoDel
root@admin-pc:~/h1_CoDel#

```

Figure 46. Creating and navigating into directory *h1_CoDel*.

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
```



```

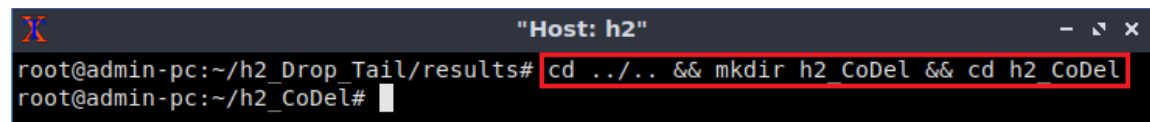
Host: h1
root@admin-pc:~/h1_CoDel# iperf3 -c 10.0.0.3 -t 60 -J > out.json

```

Figure 47. Running iPerf3 client on host h1.

Step 5. In host h2, exit from the previous folders then, create a directory called *h2_CoDel* and navigate into it using the following command:

```
cd ../../ && mkdir h2_CoDel && cd h2_CoDel
```



```

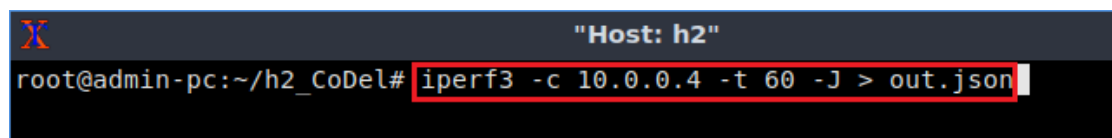
Host: h2
root@admin-pc:~/h2_Drop_Tail/results# cd ../../ && mkdir h2 CoDel && cd h2 CoDel
root@admin-pc:~/h2_CoDel#

```

Figure 48. Creating and navigating into directory *h2_CoDel*.

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
```



```

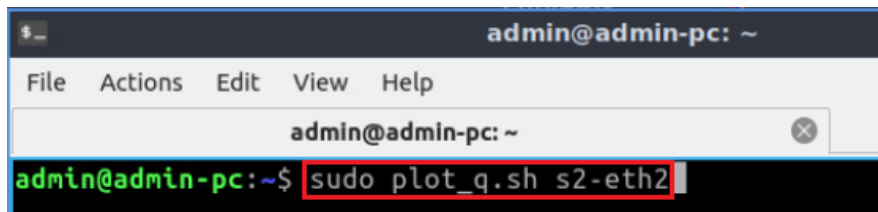
Host: h2
root@admin-pc:~/h2_CoDel# iperf3 -c 10.0.0.4 -t 60 -J > out.json

```

Figure 49 Running iPerf3 client on host h2.

Step 7. In the Client's terminal, type the command below to plot the switch's queue in real-time. When prompted for a password, type `password` and hit *Enter*.

```
sudo plot_q.sh s2-eth2
```



```
admin@admin-pc: ~  
File Actions Edit View Help  
admin@admin-pc: ~  
admin@admin-pc:~$ sudo plot_q.sh s2-eth2
```

Figure 50. Plotting the queue occupancy on switch S2's s2-eth2 interface.

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through s2-eth2 interface on switch S2, the queue occupancy is constantly 0.

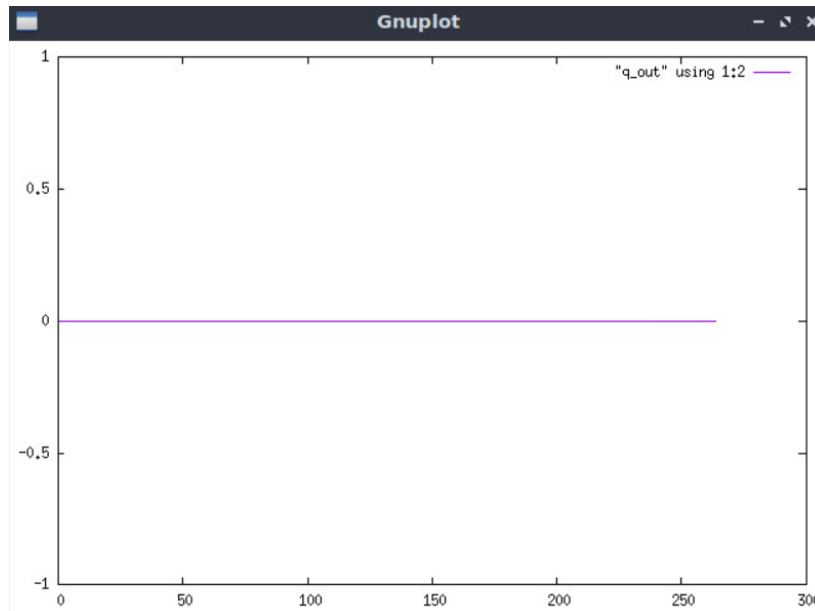


Figure 51. Queue occupancy on switch S2's s2-eth2 interface.

Step 8. Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal.

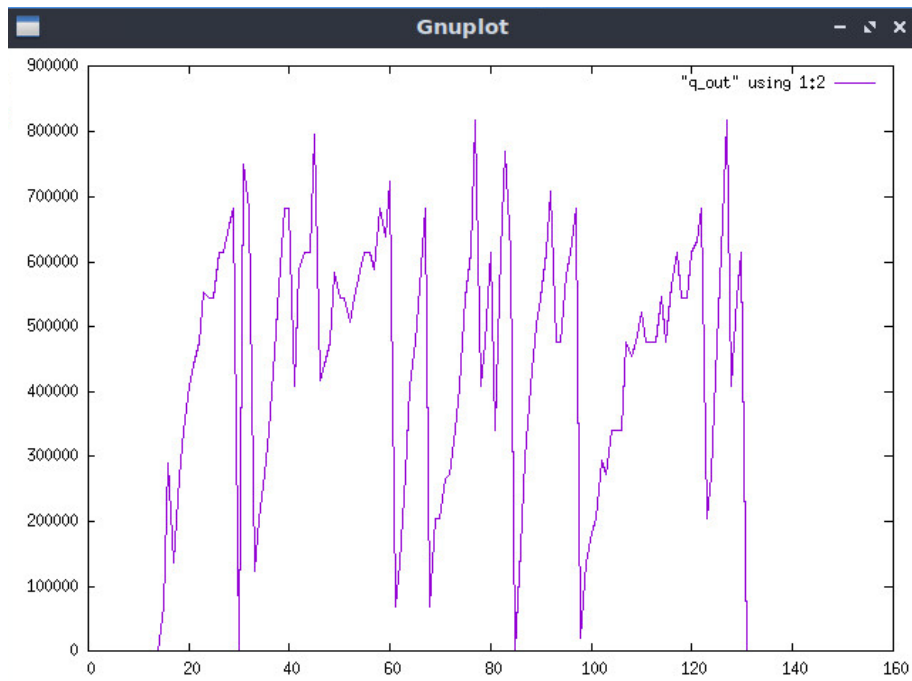


Figure 52. Queue occupancy on switch S2's s2-eth2 interface.

The graph above shows that the queue occupancy peaked around 900,000 bytes, which is below the maximum buffer size we configure on the switch 26,214,400 bytes (10 ·BDP).

Step 9. In the queue plotting window, press the `[S]` key on your keyboard to stop plotting the queue.

Step 10. After the iPerf3 test finishes on host h1, enter the following command:

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

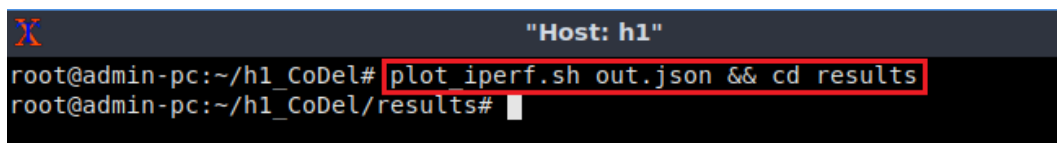


Figure 53. Generate plotting files and entering the *results* directory.

Step 11. Open the throughput file using the command below on host h1.

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

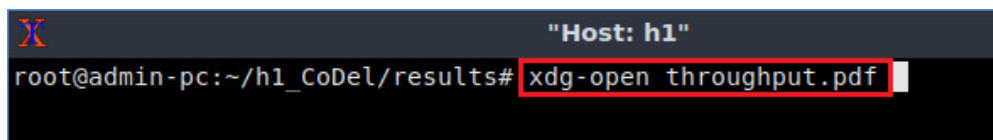


Figure 54. Opening the *throughput.pdf* file.

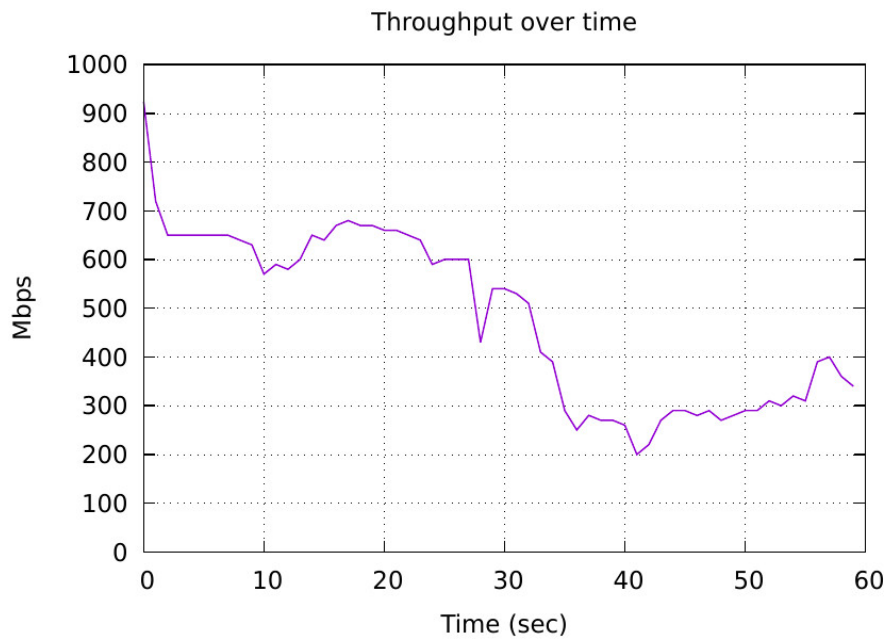


Figure 55. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 400 Mbps. It is observed that the Cubic flow collapses significantly since the link is not fairly shared with the other TCP flow.

Step 12. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

```
xdg-open RTT.pdf
```

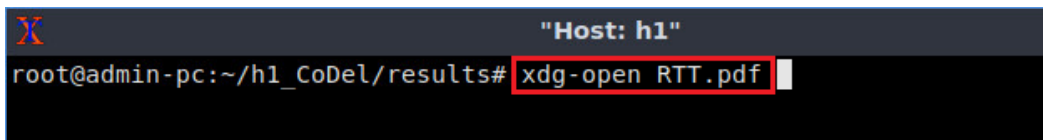


Figure 56. Opening the *RTT.pdf* file.

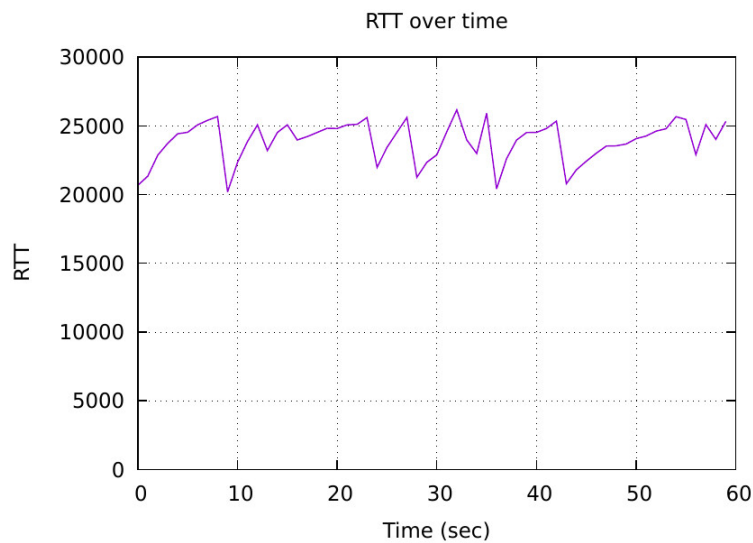


Figure 57. Measured round-trip time.

The graph above shows that the RTT was approximately 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not exceeding the configured delay (20ms).

Step 13. Close the *RTT.pdf* window then, in host h2, proceed similarly by typing the following command:

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

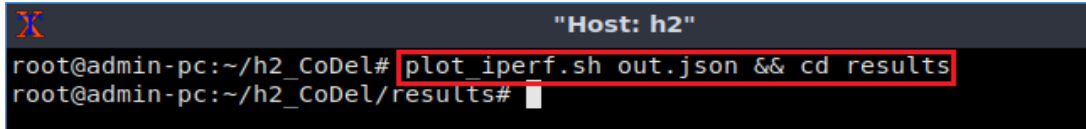


Figure 58. Generate plotting files and entering the *results* directory.

Step 14. In host h2 terminal, open the throughput file using the following command:

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

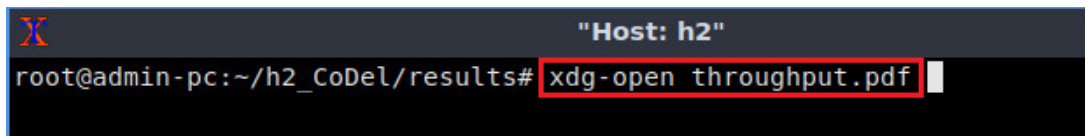


Figure 59. Opening the *throughput.pdf* file.

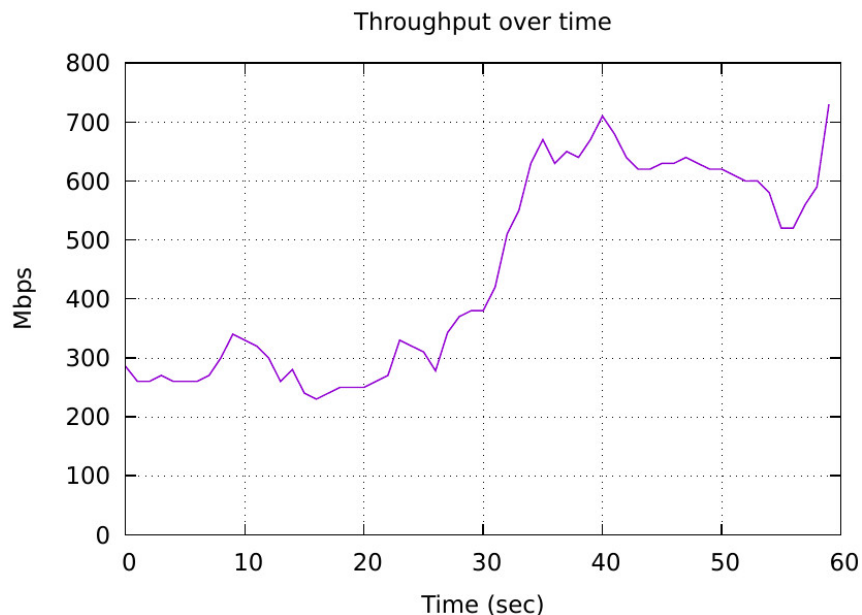


Figure 60. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 400 Mbps. It is observed that the BBR flow takes the major part of the available bandwidth which shows that the link is not fairly shared with the other TCP flow.

Step 15. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

```
xdg-open RTT.pdf
```

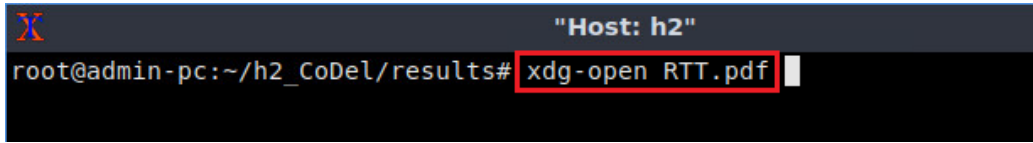


Figure 61. Opening the *RTT.pdf* file.

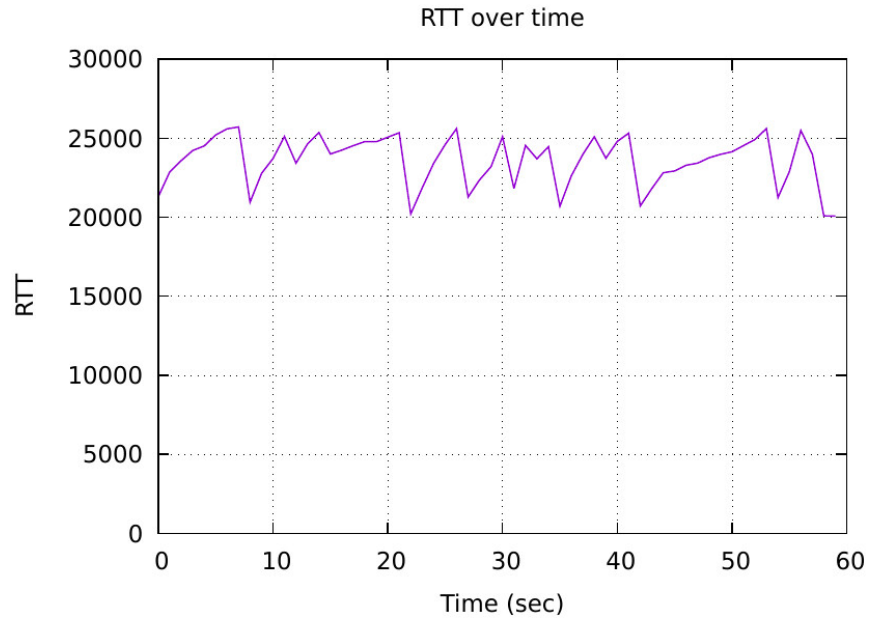


Figure 62. Measured round-trip time.

The graph above shows that the RTT was approximately 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not exceeding the configured delay (20ms).

Step 16. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

5 Configuring Fq_CoDel on switch S2

In the previous section, you have observed that despite getting low queue occupancy and, consequently mitigating bufferbloat, the link share was not fair between the two TCP flows. In this section, you are going to configure Fq_CoDel in switch S2's *s2-eth2* interface to mitigate bufferbloat and at the same time to ensure fairness. Then, you will conduct throughput and latency measurements between the hosts.

5.1 Setting Fq_CoDel parameters on switch S2's egress interface

Step 1. Delete the previous `tc` rule by issuing the command shown below.

```
sudo tc qdisc del dev s2-eth2 parent 1: handle 2:
```

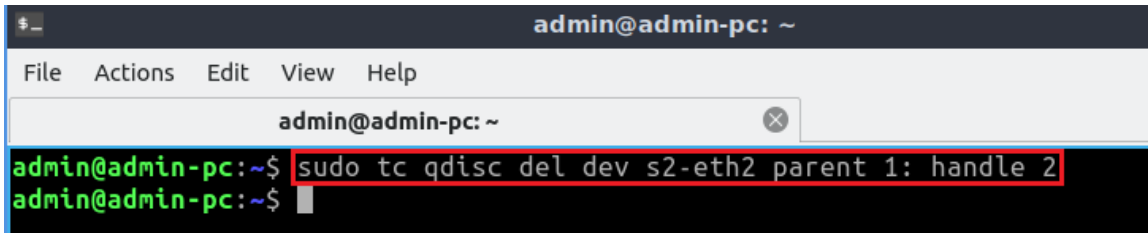


Figure 63. Deleting previous `tc` rule on switch S2's `s2-eth2` interface.

Step 2. Apply `fq_codel` 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*.

- `limit`: 17476
- `target`: 5ms
- `interval`: 100ms
- `flows`: 2

```
sudo tc qdisc add dev s2-eth2 parent 1: handle 2: fq_codel limit 17476 target 5ms interval 100ms flows 2
```

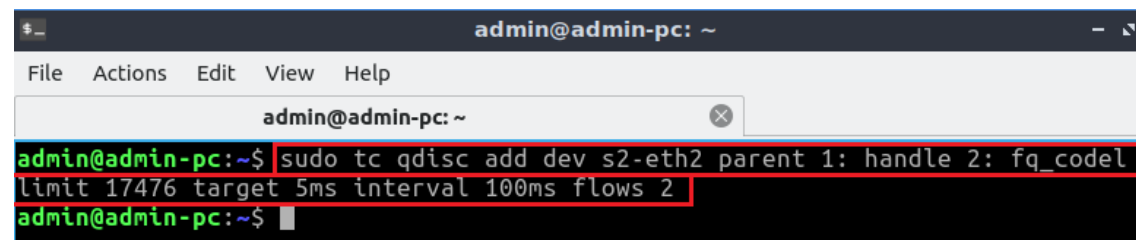


Figure 64. Setting `Fq_CoDel` parameters on switch S2's `s2-eth2` interface.

5.2 Throughput test

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

```
iperf3 -s
```

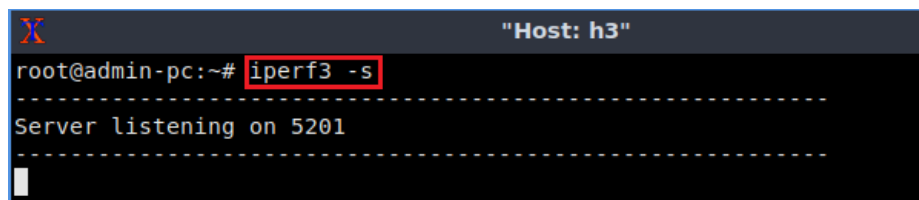
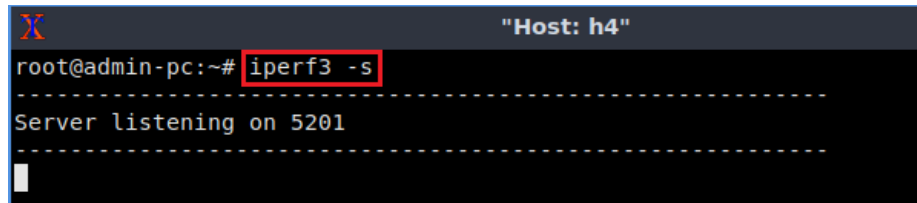


Figure 65. Starting `iPerf3` server on host h3.

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

```
iperf3 -s
```

```

Host: h4
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----

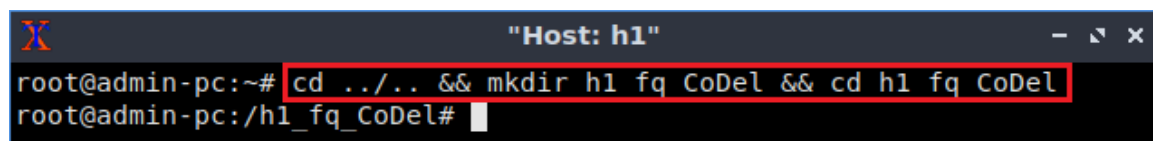
```

Figure 66. 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 6 without executing them next, in Step 7 you will press *Enter* in host h1 and host h2 to execute them.

Step 3. In host h1, exit from the previous folders then, create a directory called *h1_CoDel* and navigate into it using the following command:

```
cd ../../ && mkdir h1_fq_CoDel && cd h1_fq_CoDel
```



```

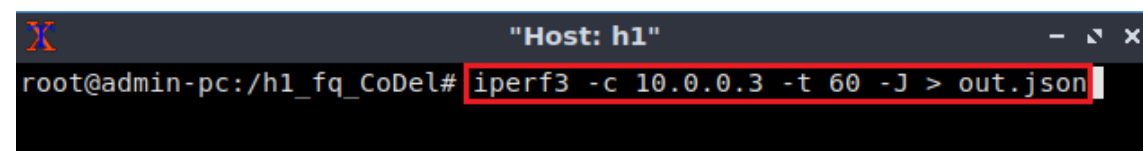
Host: h1
root@admin-pc:~# cd ../../ && mkdir h1 fq CoDel && cd h1 fq CoDel
root@admin-pc:/h1_fq_CoDel#

```

Figure 67. Creating and navigating into directory *h1_fq_CoDel*.

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
```



```

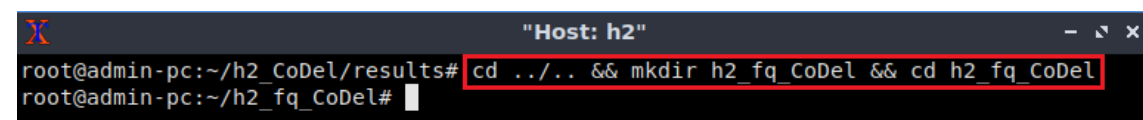
Host: h1
root@admin-pc:/h1_fq_CoDel# iperf3 -c 10.0.0.3 -t 60 -J > out.json

```

Figure 68. Running iPerf3 client on host h1.

Step 5. In host h2, exit from the previous folders then, create a directory called *h2_CoDel* and navigate into it using the following command:

```
cd ../../ && mkdir h2_fq_CoDel && cd h2_fq_CoDel
```



```

Host: h2
root@admin-pc:~/h2_CoDel/results# cd ../../ && mkdir h2_fq_CoDel && cd h2_fq_CoDel
root@admin-pc:~/h2_fq_CoDel#

```

Figure 69. Creating and navigating into directory *h2_fq_CoDel*.

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
```

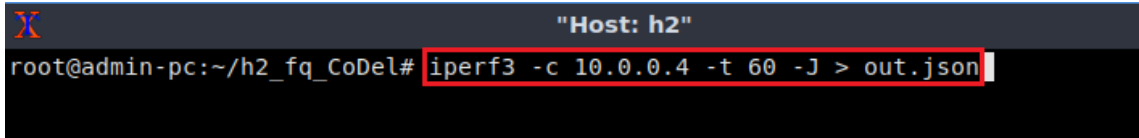


Figure 70. Running iPerf3 client on host h2.

Step 7. In the Client's terminal, type the command below to plot the switch's queue in real-time. When prompted for a password, type `password` and hit *Enter*.

```
sudo plot_q.sh s2-eth2
```

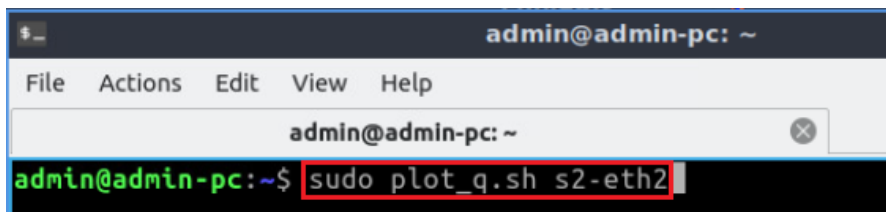


Figure 71. Plotting the queue occupancy on switch S2's s2-eth2 interface.

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through `s2-eth2` interface on switch S2, the queue occupancy is constantly 0.

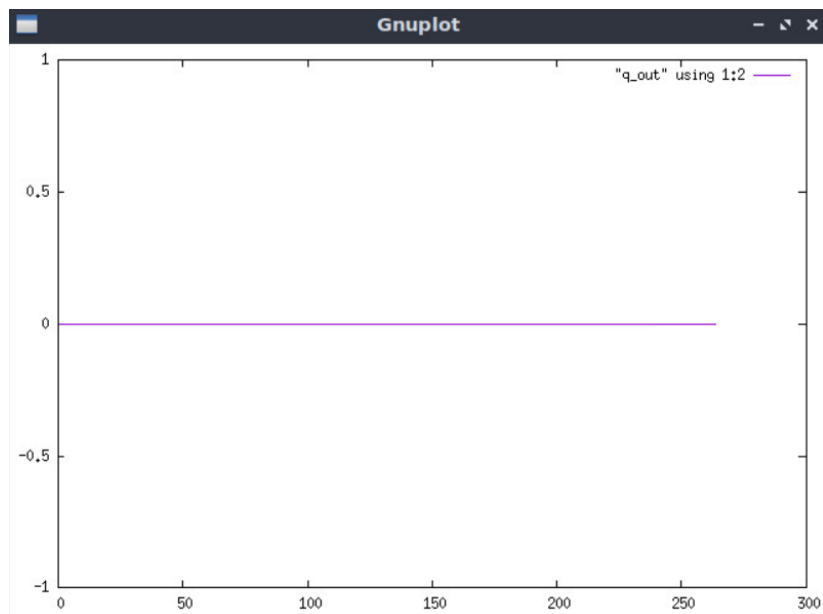


Figure 72. Queue occupancy on switch S2's s2-eth2 interface.

Step 8. Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal.

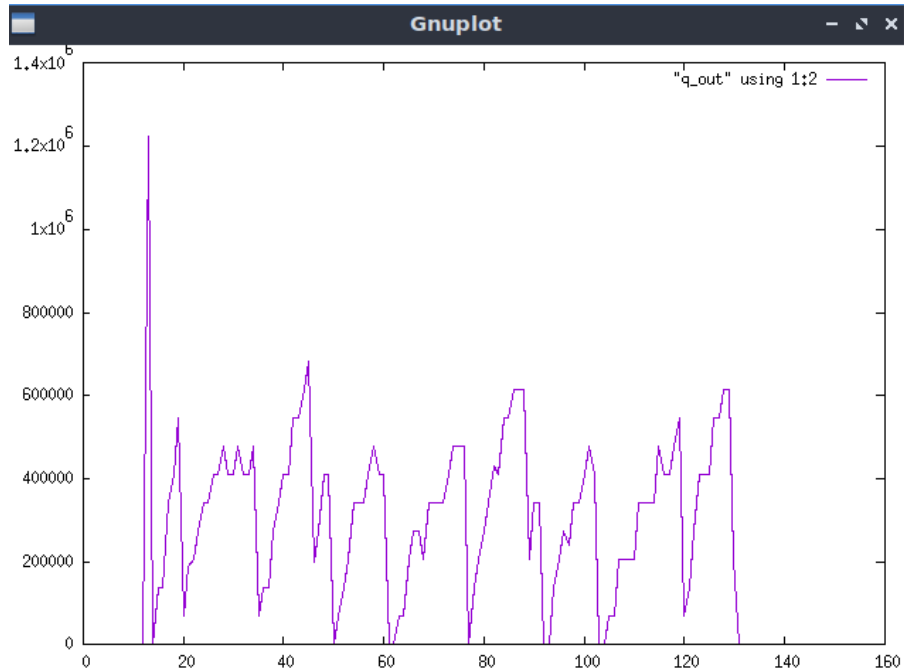


Figure 73. Queue occupancy on switch S2's s2-eth2 interface.

The graph above shows that the queue occupancy peaked around 1,200,000 bytes, which is below the maximum buffer size we configure on the switch 26,441,00 bytes (10 ·BDP).

Step 9. In the queue plotting window, press the `S` key on your keyboard to stop plotting the queue.

Step 10. After the iPerf3 test finishes on host h1, enter the following command:

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

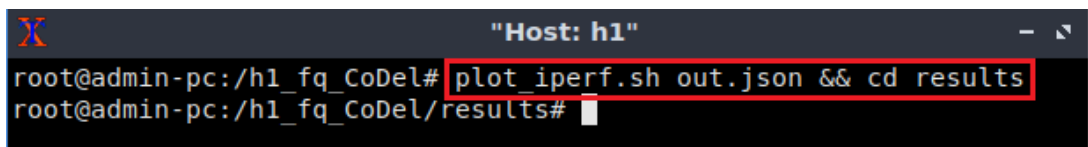


Figure 74. Generate plotting files and entering the *results* directory.

Step 11. Open the throughput file using the command below on host h1.

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

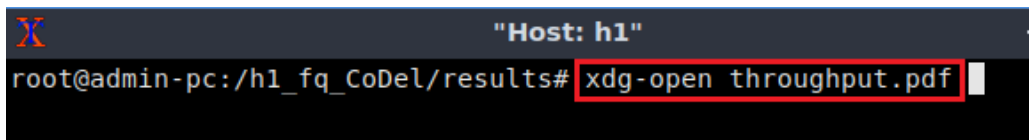


Figure 75. Opening the *throughput.pdf* file.

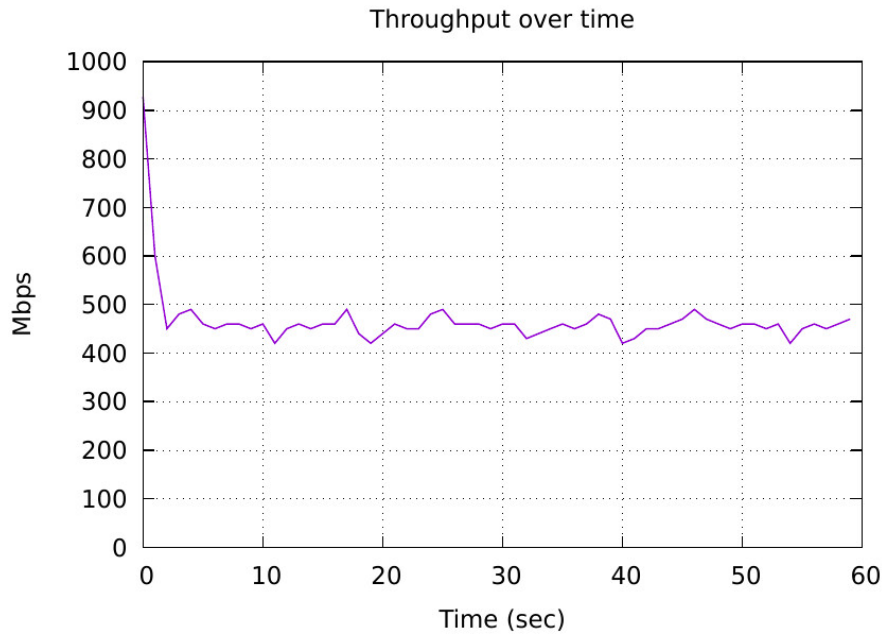


Figure 76. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 450 Mbps. It is observed that the Cubic flow uses the half part of the available bandwidth.

Step 12. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

```
xdg-open RTT.pdf
```

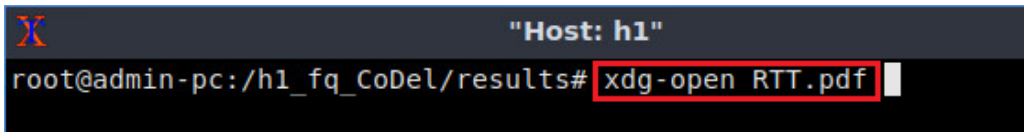


Figure 77. Opening the *RTT.pdf* file.

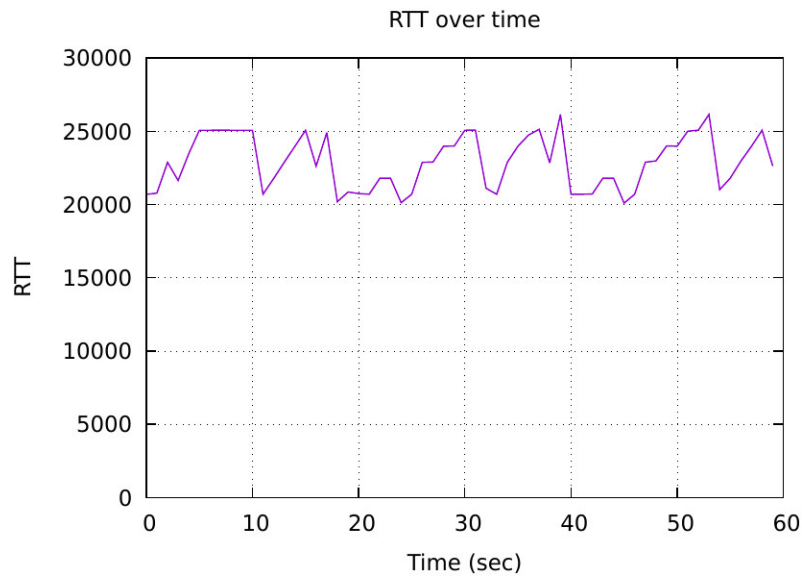


Figure 78. Measured round-trip time.

The graph above shows that the RTT was approximately 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not considerably exceeding the configured delay (20ms).

Step 13. Close the *RTT.pdf* window then, in host h2, proceed similarly by typing the following command:

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

```

X "Host: h2"
root@admin-pc:~/h2_fq_CoDel# plot iperf.sh out.json && cd results
root@admin-pc:~/h2_fq_CoDel/results#

```

Figure 79. Generate plotting files and entering the *results* directory.

Step 14. In host h2 terminal, open the throughput file using the following command:

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

```

X "Host: h2"
root@admin-pc:~/h2_fq_CoDel/results# xdg-open throughput.pdf

```

Figure 80. Opening the *throughput.pdf* file.

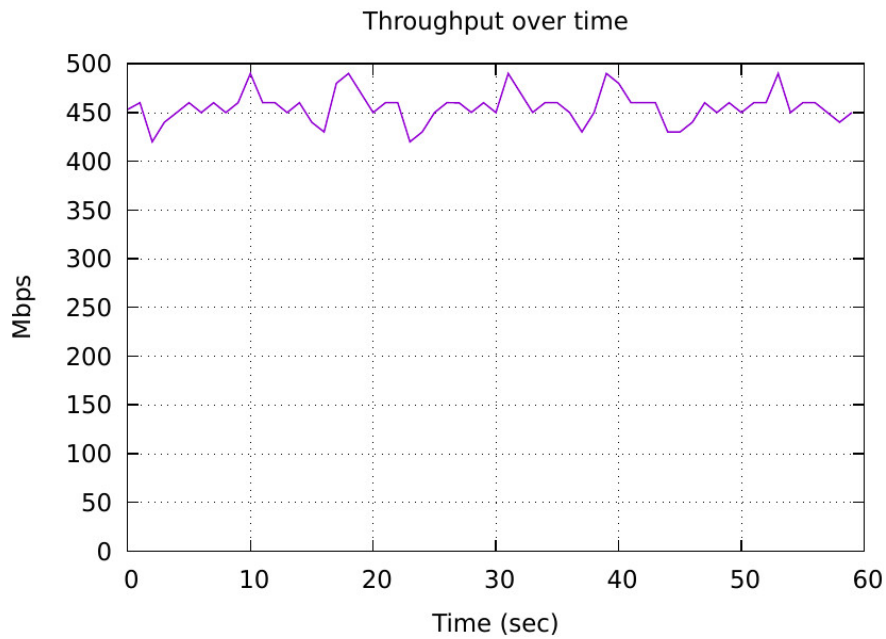


Figure 81. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 450 Mbps. It is observed that the BBR flow uses the half part of the available bandwidth thus, it is fairly sharing the link with the other flow.

Step 15. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

```
xdg-open RTT.pdf
```

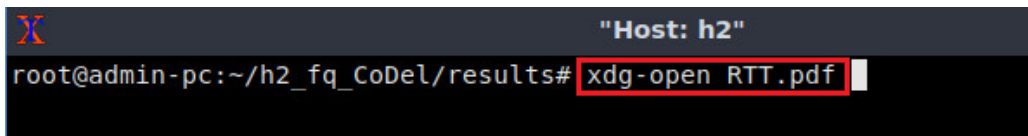


Figure 82. Opening the *RTT.pdf* file.

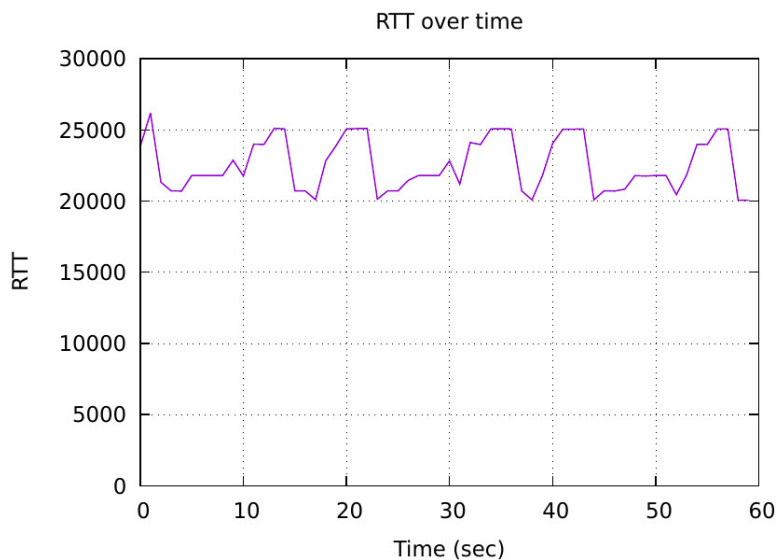


Figure 83. Measured round-trip time.

The graph above shows that the RTT was around 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not considerably exceeding the configured delay (20ms).

Step 16. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

6 Changing the bandwidth to 100Mbps

This section is aimed to analyze the impact of changing the bandwidth to 100 Mbps while Fq_CoDel is configured to work considering the previous network condition. The results will show Fq_CoDel does not requires a reconfiguration if the network conditions changes (i.e, latency, bandwidth, loss rate). In this section, you will change the bandwidth to 100 Mbps then, you will observe the throughput, queue occupancy and, Round-Trip Time.

Step 1. Apply `tbft` 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`: 100mbit
- `burst`: 50,000
- `limit`: 26,214,400

```
sudo tc qdisc change dev s2-eth2 root: handle 1: tbf rate 100mbit burst 50000
limit 26214400
```

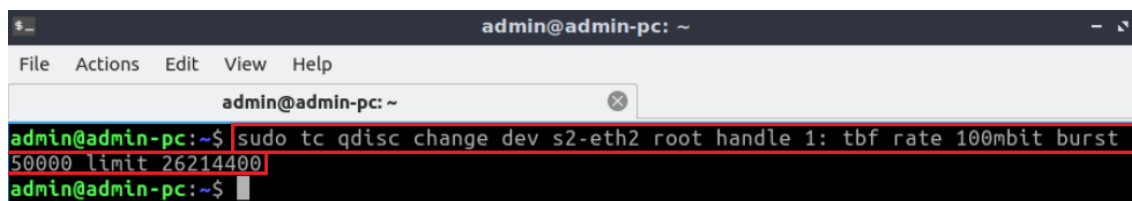


Figure 84. Limiting rate to 100 Mbps and keeping the buffer size to 100·BDP on switch S2's interface.

Notice that the link rate is changed to 100 Mbps however, the buffer size is maintained at 26,214,400 bytes which for this case corresponds to 100·BDP

6.1 Throughput and latency tests

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

```
iperf3 -s
```

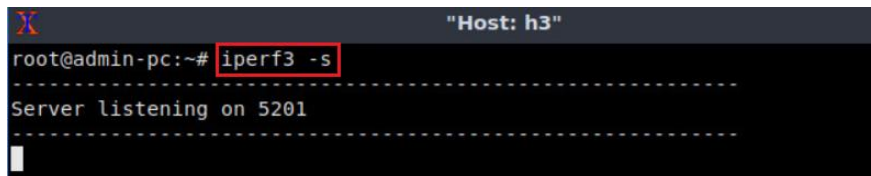


Figure 85. Starting iPerf3 server on host h3.

Step 2. In the Client’s terminal, type the command below to plot the switch’s queue in real-time. When prompted for a password, type `password` and hit *Enter*.

```
sudo plot_q.sh s2-eth2
```

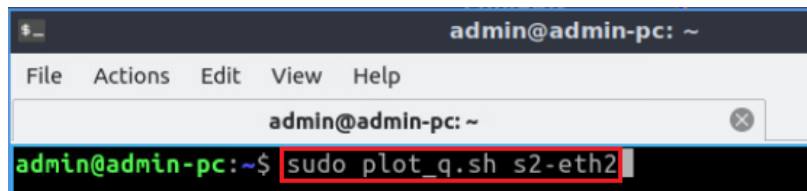


Figure 86. Plotting the queue occupancy on switch S2’s s2-eth2 interface.

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through *s2-eth2* interface on switch S2, the queue occupancy is constantly 0.

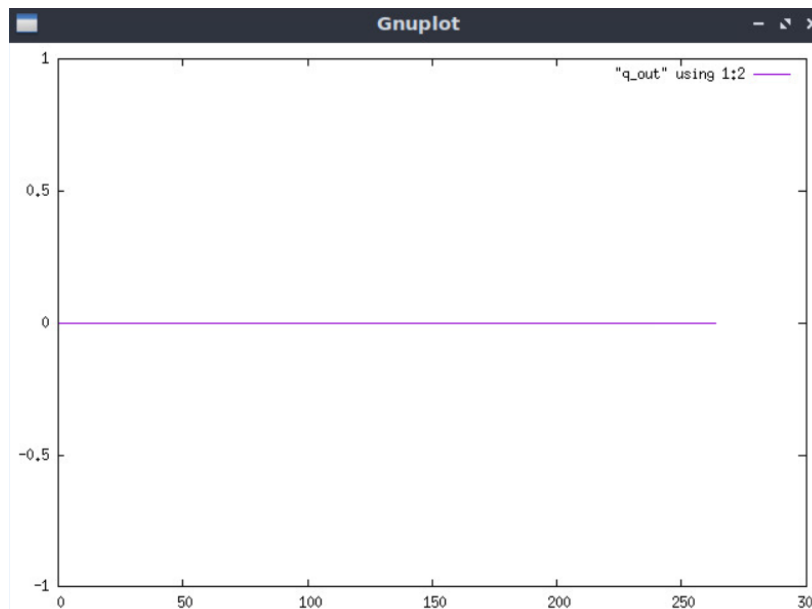
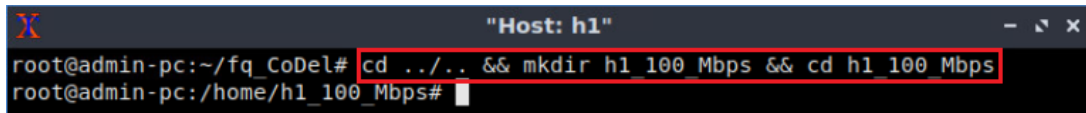


Figure 87. Queue occupancy on switch S2’s s2-eth2 interface.

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 6 without executing them next, in Step 7 you will press *Enter* in host h1 and host h2 to execute them.

Step 3. In host h1, exit from the previous folders then, create a directory called *h1_100_Mbps* and navigate into it using the following command:

```
cd ../../ && mkdir h1_100_Mbps && cd h1_100_Mbps
```

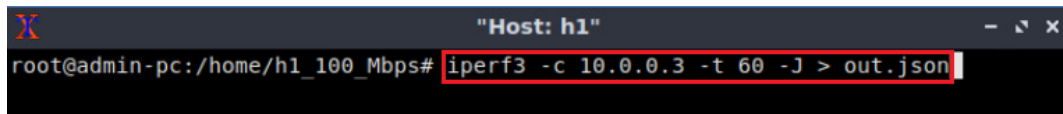


A terminal window titled "Host: h1" showing the command `cd ../../ && mkdir h1_100_Mbps && cd h1_100_Mbps` being entered and executed. The prompt changes from `root@admin-pc:~/fq_CoDel#` to `root@admin-pc:/home/h1_100_Mbps#`.

Figure 88. Creating and navigating into directory *h1_100_Mbps*.

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
```

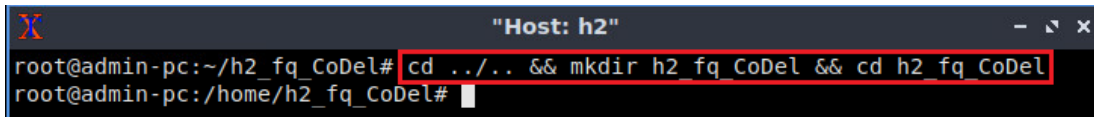


A terminal window titled "Host: h1" showing the command `iperf3 -c 10.0.0.3 -t 60 -J > out.json` being typed at the prompt `root@admin-pc:/home/h1_100_Mbps#`.

Figure 89. Running iPerf3 client on host h1.

Step 5. In host h2, exit from the previous folders then, create a directory called *h2_100_Mbps* and navigate into it using the following command:

```
cd ../../ && mkdir h2_100_Mbps && cd h2_100_Mbps
```

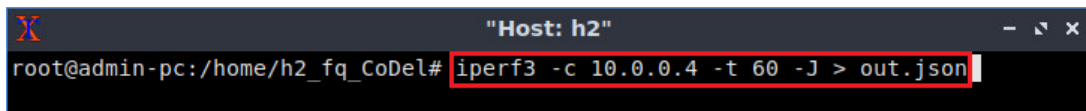


A terminal window titled "Host: h2" showing the command `cd ../../ && mkdir h2_100_Mbps && cd h2_100_Mbps` being entered and executed. The prompt changes from `root@admin-pc:~/h2 fq_CoDel#` to `root@admin-pc:/home/h2_100_Mbps#`.

Figure 90. Creating and navigating into directory *h2_100_Mbps*.

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
```



A terminal window titled "Host: h2" showing the command `iperf3 -c 10.0.0.4 -t 60 -J > out.json` being typed at the prompt `root@admin-pc:/home/h2_100_Mbps#`.

Figure 91. Running iPerf3 client on host h2.

Step 7. In the Client's terminal, type the command below to plot the switch's queue in real-time. When prompted for a password, type `password` and hit *Enter*.

```
sudo plot_q.sh s2-eth2
```

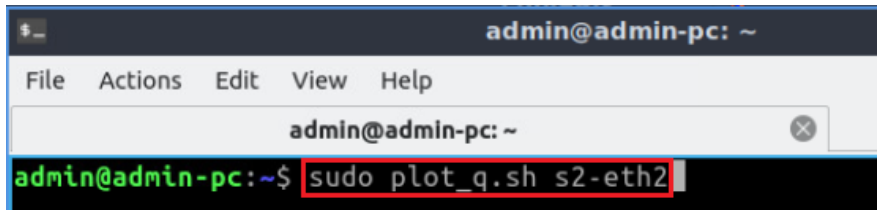


Figure 92. Plotting the queue occupancy on switch S2's s2-eth2 interface.

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through s2-eth2 interface on switch S2, the queue occupancy is constantly 0.

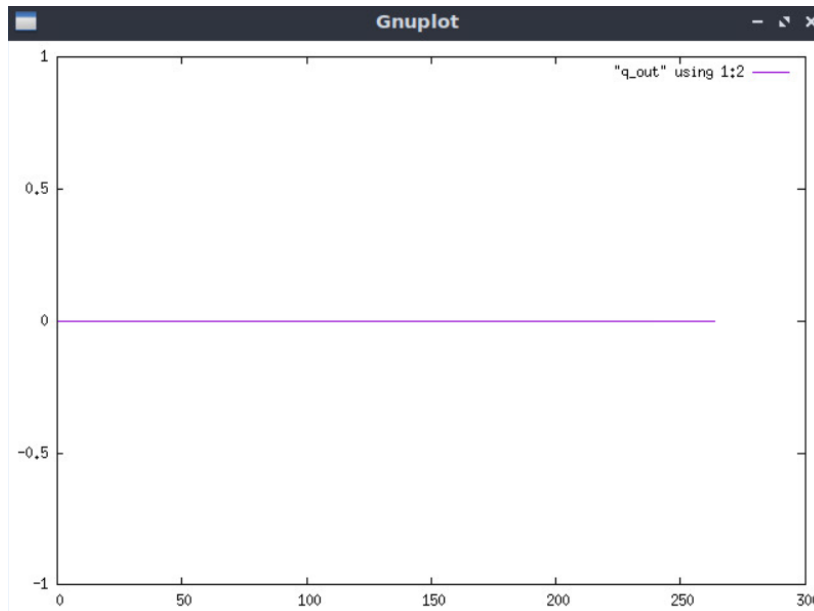


Figure 93. Queue occupancy on switch S2's s2-eth2 interface.

Step 8. Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal.

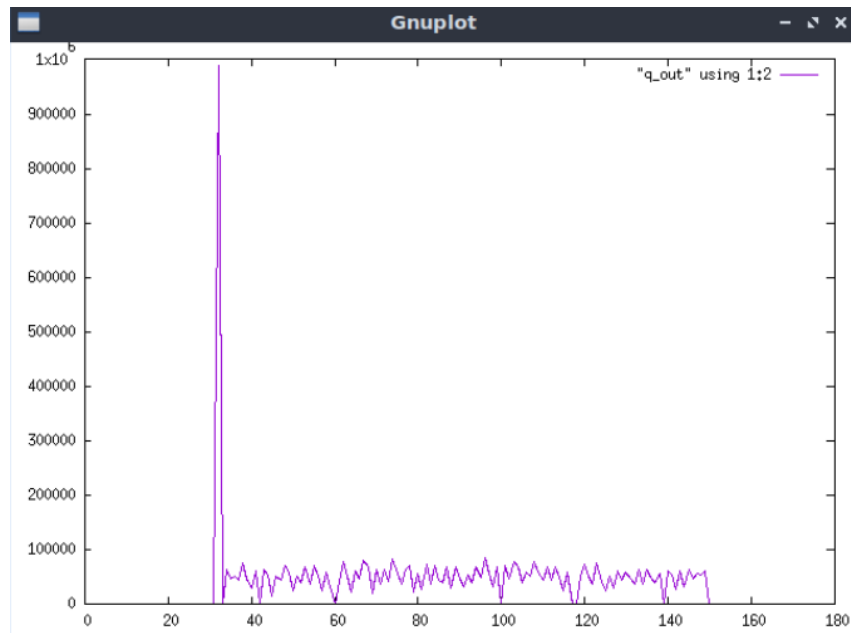


Figure 94. Queue occupancy on switch S2's s2-eth2 interface.

The graph above shows that the queue occupancy peaked at $1 \cdot 10^6$, which is below the maximum buffer size we configure on the switch 26,214,400 bytes.

Step 9. In the queue plotting window, press the `S` key on your keyboard to stop plotting the queue.

Step 10. After the iPerf3 test finishes on host h1, enter the following command:

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

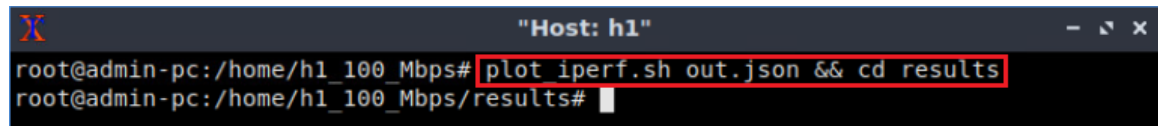


Figure 95. Generate plotting files and entering the *results* directory.

Step 11. Open the throughput file using the command below on host h1.

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

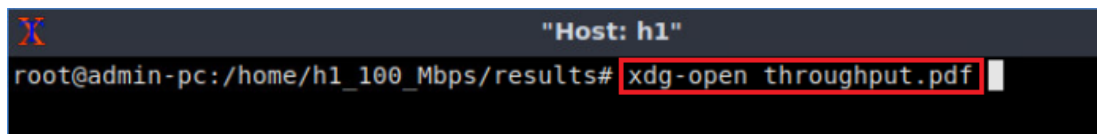


Figure 96. Opening the *throughput.pdf* file.

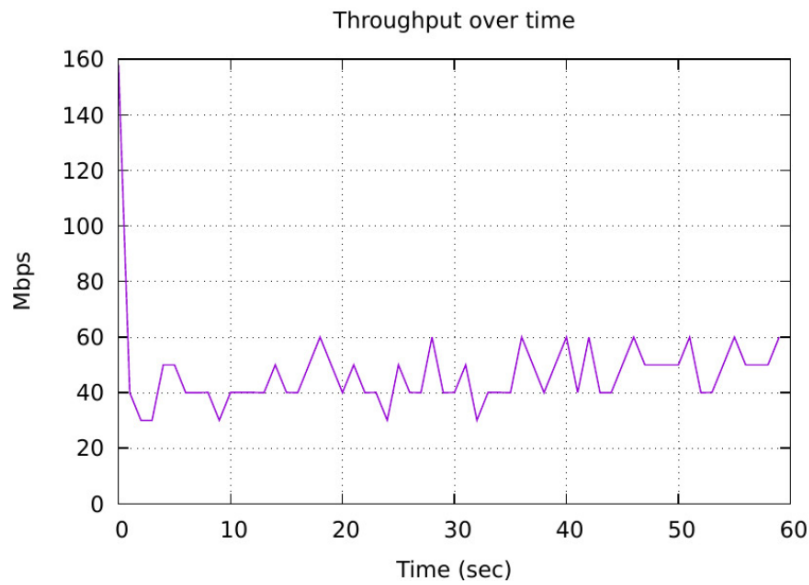


Figure 97. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 50 Mbps. It is observed that the Cubic flow uses the half part of the available bandwidth.

Step 12. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

```
xdg-open RTT.pdf
```

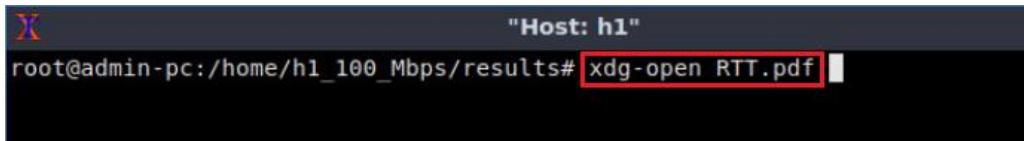


Figure 98. Opening the *RTT.pdf* file.

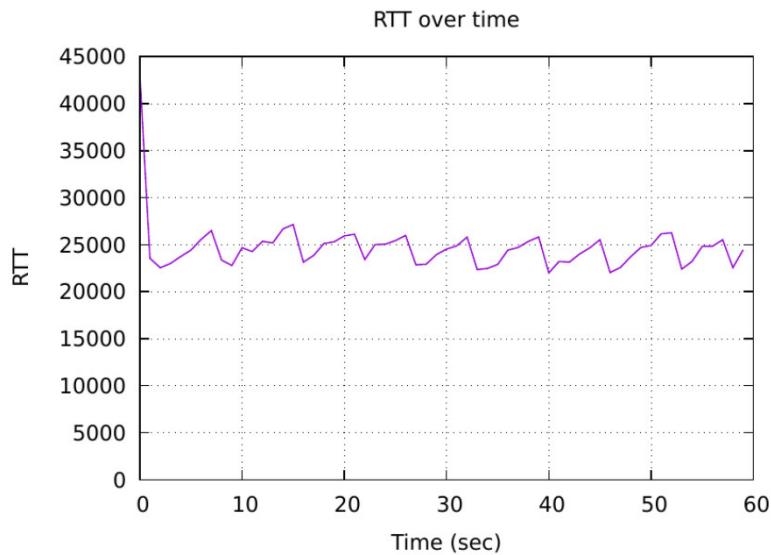
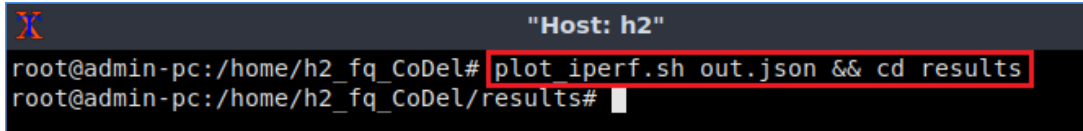


Figure 99. Measured round-trip time.

The graph above shows that the RTT was around 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not considerably exceeding the configured delay (20ms).

Step 13. Close the *RTT.pdf* window then, in host h2, proceed similarly by typing the following command:

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



```

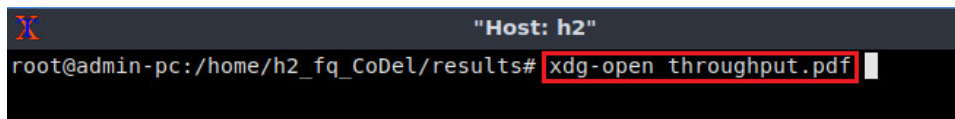
Host: h2
root@admin-pc:/home/h2_fq_CoDel# plot_iperf.sh out.json && cd results
root@admin-pc:/home/h2_fq_CoDel/results#

```

Figure 100. Generate plotting files and entering the *results* directory.

Step 14. In host h2 terminal, open the throughput file using the following command:

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



```

Host: h2
root@admin-pc:/home/h2_fq_CoDel/results# xdg-open throughput.pdf

```

Figure 101. Opening the *throughput.pdf* file.

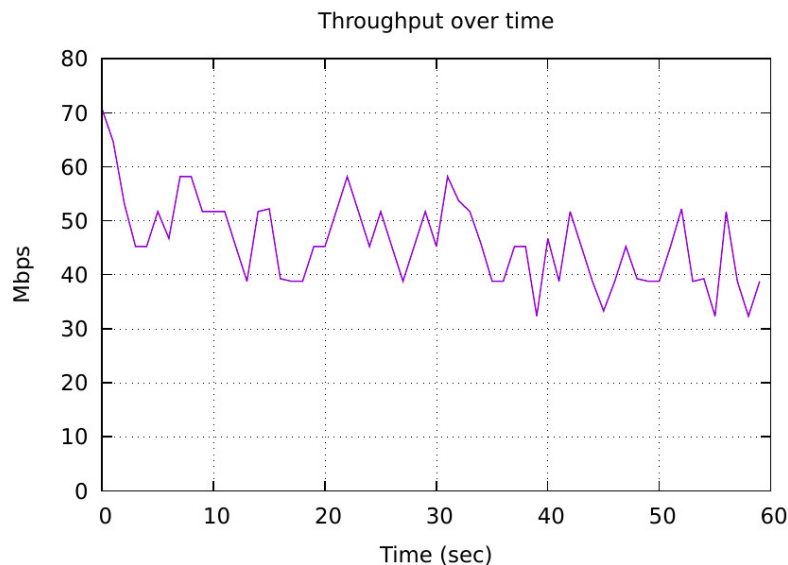
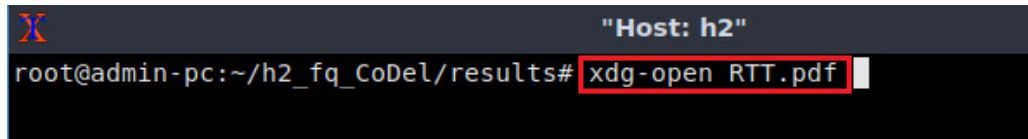


Figure 102. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 50 Mbps as well. It is observed that the BBR flow uses the half part of the available bandwidth thus, it is fairly sharing the link with the other flow. One of the most remarkable features of Fq_CoDel is that it works well even if the network condition changes.

Step 15. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

```
xdg-open RTT.pdf
```



```

Host: h2
root@admin-pc:~/h2_fq_CoDel/results# xdg-open RTT.pdf

```

Figure 103. Opening the *RTT.pdf* file.

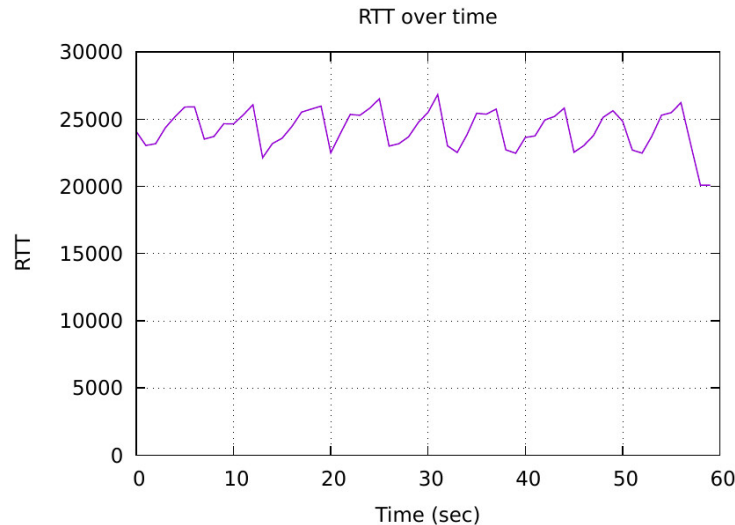


Figure 104. Measured round-trip time.

The graph above shows that the RTT was around 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not considerably exceeding the configured delay (20ms).

Step 16. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

This concludes Lab 18. Stop the emulation and then exit out of MiniEdit.

References

1. K. Nicolas, E. Lochin, O. Mehani. "Revisiting old friends: is CoDel really achieving what RED cannot?," (2014).
2. K. Nichols, V. Jacobson, "Controlling queue delay". *ACM Queue*, (2012).
3. J. Gettys, K. Nichols, "Bufferbloat: dark buffers in the internet," *Communications of the ACM*, (2012).
4. K. Nichols, V. Jacobson, A. McGregor, and A. Iyengar, "Controlled delay active queue management," Internet engineering task force, Request for comments 8289, 2018

5. J. Kurose, K. Ross, "Computer networking, a top-down approach," 7th Edition, Pearson, 2017.
6. C. Villamizar, C. Song, "High performance TCP in ansnet," ACM Computer Communications Review, vol. 24, no. 5, pp. 45-60, Oct. 1994.
7. 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>.
8. J. Gettys, K. Nichols, "Bufferbloat: dark buffers in the internet," Communications of the ACM, vol. 9, no. 1, pp. 57-65, Jan. 2012.
9. 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.