



UNIVERSITY OF
SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 19: Proportional Integral Controller-Enhanced (PIE)

Document Version: **11-23-2019**



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	3
2 Lab topology.....	5
2.1 Starting host h1, host h2, and host h3	7
2.1 Emulating high-latency WAN	8
2.4 Testing connection	9
3 Testing throughput on a network using Drop Tail AQM algorithm.....	10
3.1 Bandwidth-delay product (BDP) and hosts' TCP buffer size	10
3.2 Setting switch S2's buffer size to $10 \cdot \text{BDP}$	12
3.3 Throughput and latency tests	12
4 Configuring PIE on switch S2.....	17
4.1 Setting PIE parameter on switch S2's egress interface	17
4.2 Throughput and latency tests	18
4.3 Changing the bandwidth to 100Mbps	23
4.4 Throughput and latency tests	24
References	29

Overview

This lab introduces to Proportional Integral Controller-Enhanced (PIE) Active Queue Management (AQM) algorithm. This algorithm is aimed to mitigate high end-to-end latency by controlling the average queue length. PIE manages the queue length by using a Proportional Integral (PI) controller which is known for removing steady state errors. Throughput, latency and queue length measurements are conducted in this lab to verify the impact of the dropping policy provided PIE.

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 buffering process in a router.
3. Explain the impact of PIE handling the queuing policy in a router egress port.
4. Visualize queue occupancy in a router.
5. Analyze how PIE manages the queue length in order to allow end-hosts to achieve high throughput and low latency.
6. Modify the network condition in order to evaluate the performance on PIE's dropping policy.

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 throughput on a network using Drop Tail AQM algorithm.
4. Section 4: Configuring PIE on switch S2.

1 Introduction

The increasing number of time-sensitive applications in the Internet brings a set of challenges to control end-to-end delay. To avoid packet loss, many service providers or data center operators require vendors to increase router’s buffer as much as possible. Due to the decrease in memory chip prices, these requests are easily satisfied assuring low packet loss and high TCP throughput however, it suffers from a major downside. The TCP protocol continuously increases its sending rate and causes network buffers to fill up. TCP cuts its rate only when it receives a packet drop or mark that is interpreted as a congestion signal. Nevertheless, drops and marks usually occur when network buffers are full or almost full. As a result, excess buffers, initially designed to avoid packet drops, would lead to highly elevated queueing latency and latency variation. The design of a queue management scheme should not only should allow short-term burst to smoothly pass, but also should control the average latency in the presence of bursty TCP flows.

Active queue management (AQM) algorithms are designed to tackle the aforementioned problem. New algorithms are beginning to emerge to control queueing latency directly to address the bufferbloat problem¹. In this lab, it is presented Proportional Integral Controller-Enhanced AQM algorithm (PIE). PIE randomly drops an incoming packet at the onset of the congestion. The congestion detection, however, is based on the queueing latency instead of the queue length. Furthermore, PIE also uses the derivative (rate of change) of the queueing latency to help determine congestion levels and an appropriate response. The design parameters of PIE are chosen via control theory stability analysis. While these parameters can be fixed to work in various traffic conditions, they could be made self-tuning to optimize system performance.

1.1 The PIE algorithm

As illustrated in Figure 1, PIE is comprised of three simple basic components: a) random dropping at enqueueing; b) periodic drop probability update; c) latency calculation². When a packet arrives, a random decision is made regarding whether to drop the packet. The drop probability is updated periodically based on how far the current latency is away from the target and whether the queueing latency is currently trending up or down. The queueing latency can be obtained using direct measurements or using estimations calculated from the queue length and the dequeue rate.

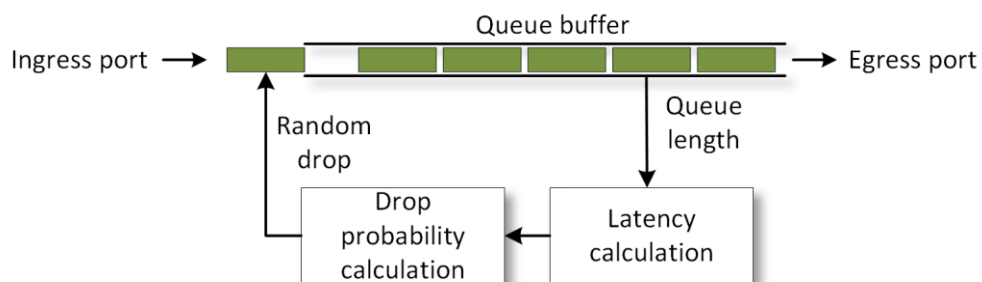


Figure 1. Buffer managed by PIE AQM.

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

```
tc qdisc [add | ...] dev [dev_id] root pie limit [PACKETS] target [SECONDS]
tupdate [SECONDS] alpha [0-32] beta [0-32] ecn|noecn bytemode|nobytemode
```

- `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.
- `pie`: This parameter specifies the Proportional Integral Controller-Enhanced AQM algorithm.
- `limit [BYTES]`: Limit on the queue size in packets. Incoming packets are dropped when this limit is reached. Default is 1000 packets.
- `target`: Denotes the expected queue delay. The default value is 15ms.
- `tupdate`: Specifies the frequency at which the system drop probability is calculated. The default value is 15ms.
- `alpha/beta`: Alpha and beta are parameters chosen to set the proportional and integral gain in the controller. With these values the drop probability is calculated. These values should be in the range between 0 and 32.
- `ecn/noecn`: Is used to mark packets instead of dropping `ecn` to turn on ecn mode, `noecn` to turn off ecn mode. By default, ecn is turned off.
- `bytemode/nobytemode`: Is used to scale drop probability proportional to packet size `bytemode` to turn on bytemode, `nobytemode` to turn off bytemode. By default, bytemode is turned off.

In this lab, we will use the `pie` AQM algorithm 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.

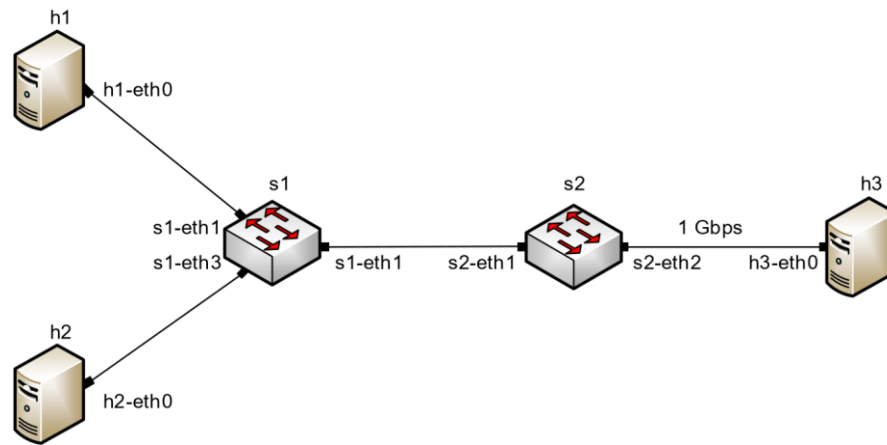


Figure 2. Lab topology.

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`.

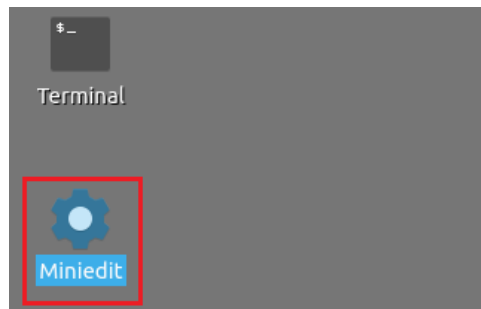


Figure 3. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 19.mn* topology file and click on *Open*.

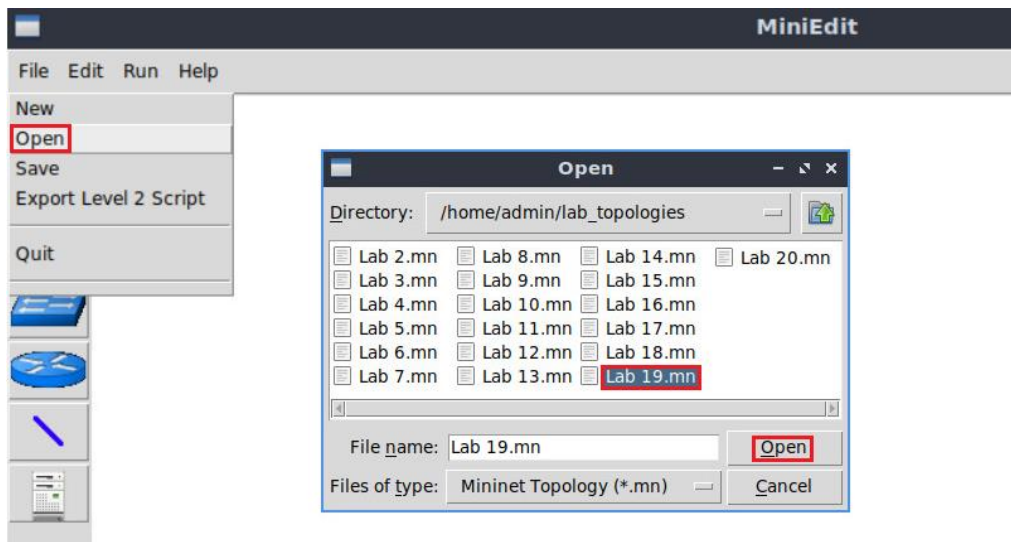


Figure 4. MiniEdit's *Open* dialog.

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.

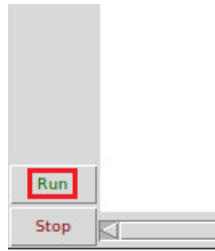


Figure 5. Running the emulation.

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

2.1 Starting host h1, host h2, and host h3

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.

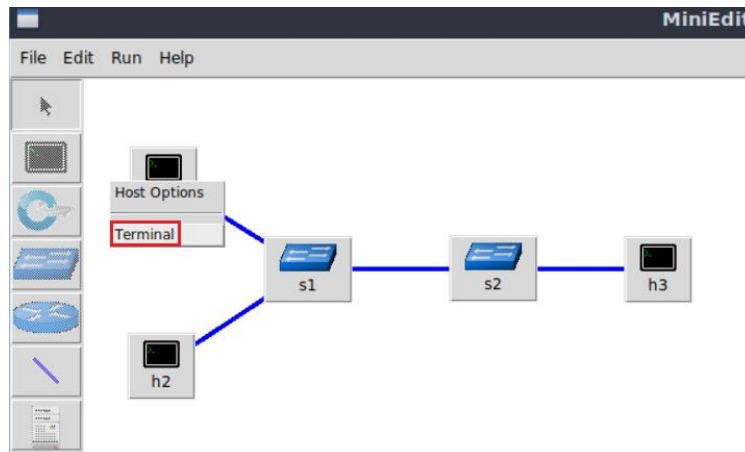


Figure 6. Opening a terminal on host h1.

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 7. Connectivity test using `ping` command.

2.1 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 8. 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
    
```

```

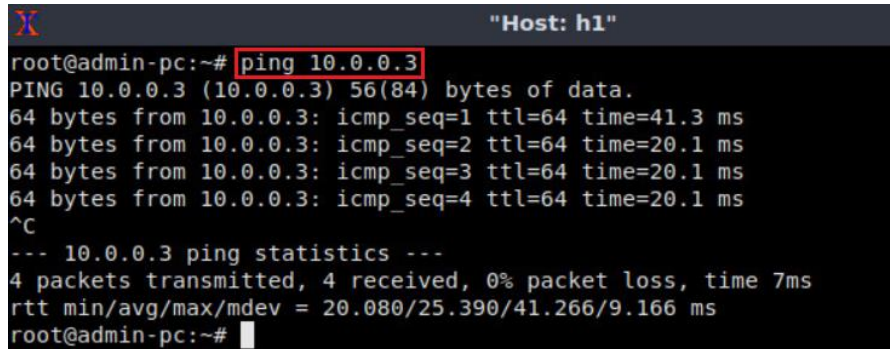
admin@admin-pc: ~
File Actions Edit View Help
admin@admin-pc: ~
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth1 root netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$
    
```

Figure 9. Adding delay of 20ms to switch S1's `s1-eth1` interface.

2.4 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.



```

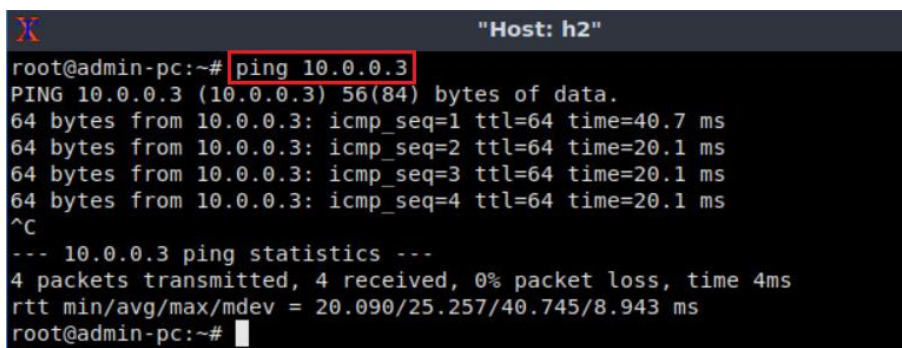
Host: h1
root@admin-pc:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=41.3 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=20.1 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 10. 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 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 20.090, 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 throughput on a network using Drop Tail AQM algorithm

In this section, you are going to change the switch S2's buffer size to $10 \cdot \text{BDP}$ and emulate a 1 Gbps Wide Area Network (WAN) using the Token Bucket Filter (tbf) as well as hosts' h1 and h3 TCP sending and receiving windows. The AQM algorithm is Drop Tail, which works dropping newly arriving packets when the queue is full therefore, the parameter that is configured is the queue size which is given by the limit value set with the tbf rule. Then, you will test the throughput between host h1 and host h3. In this section, $10 \cdot \text{BDP}$ is 25 Mbytes, thus the tbf limit value will be set to $10 \cdot \text{BDP} = 26,214,400$ bytes.

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.

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

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

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

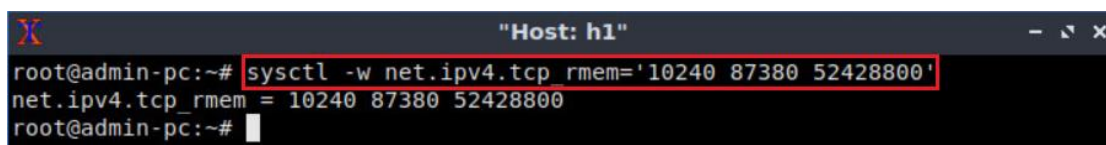
$$1 \text{ Mbyte} = 1024^2 \text{ bytes}$$

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

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 $10 \cdot \text{BDP}$. 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).

Step 1. Now, 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). The maximum value is doubled ($2 \cdot 10 \cdot \text{BDP}$) as Linux only allocates half of the assigned value.

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



```

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

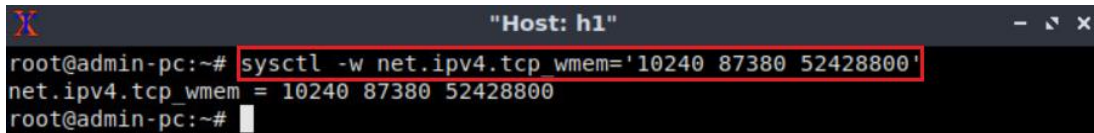
```

Figure 12. 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). The maximum value is doubled as Linux allocates only half of the assigned value.

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



```

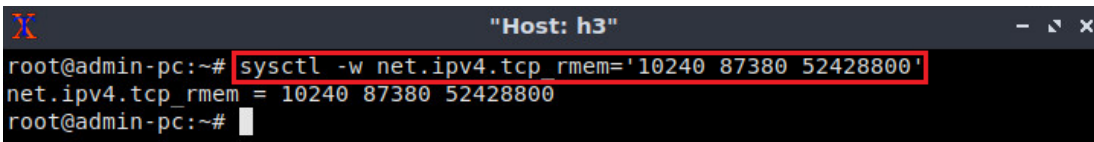
Host: h1
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#

```

Figure 13. Send window change in `sysctl`.

Step 3. Now, we have calculated the maximum value of the TCP sending and receiving buffer size. In order to change the receiving buffer size, on host h3's terminal type the command shown below. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum). The maximum value is doubled as Linux allocates only half of the assigned value.

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



```


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 14. Receive window change in `sysctl`.

Step 4. 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). The maximum value is doubled as Linux allocates only half of the assigned value.

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



```

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 15. Send 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.

3.2 Setting switch S2's buffer size to 10 · BDP

Step 1. Apply `tbw` 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
```

The screenshot shows a terminal window titled "admin@admin-pc: ~". The command `sudo tc qdisc add dev s2-eth2 root handle 1: tbf rate 1gbit burst 500000 limit 26214400` is entered and executed. The prompt returns to `admin@admin-pc:~$`.

Figure 16. Limiting rate to 1 Gbps and setting the buffer size to 10 · BDP on switch S2's interface.

3.3 Throughput and latency tests

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

```
iperf3 -s
```

The screenshot shows a terminal window titled "Host: h3". The command `iperf3 -s` is entered and executed. The output shows "Server listening on 5201".

Figure 17. 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.

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

The screenshot shows a terminal window titled "admin@admin-pc: ~". The command `sudo plot_q.sh s2-eth2` is entered and executed.

Figure 18. 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.

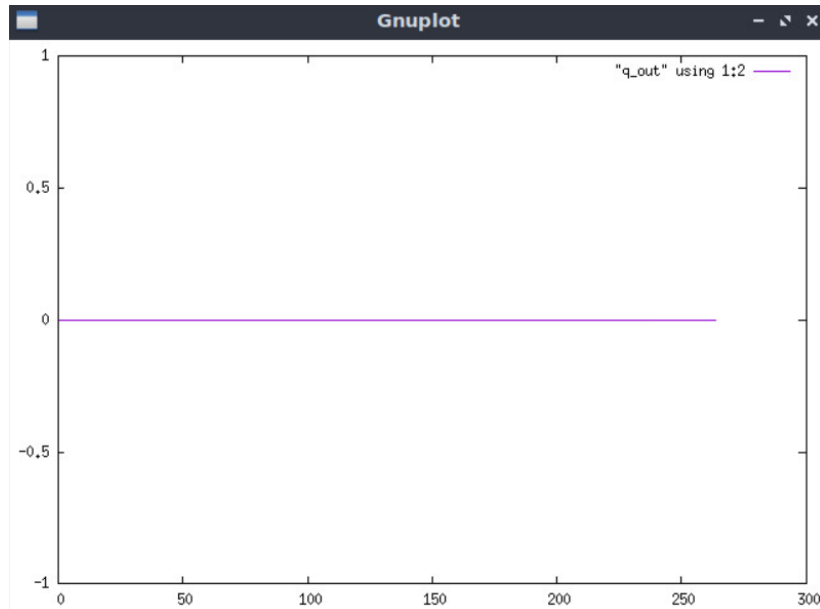


Figure 19. Queue occupancy on switch S2's *s2-eth2* interface.

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

```
mkdir Drop_Tail && cd Drop_Tail
```

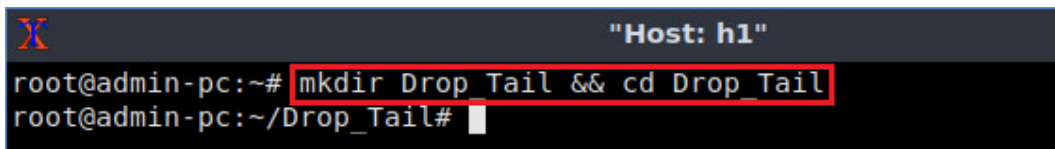


Figure 20. Creating and navigating into directory *Drop_Tail*.

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 90 -J > out.json
```

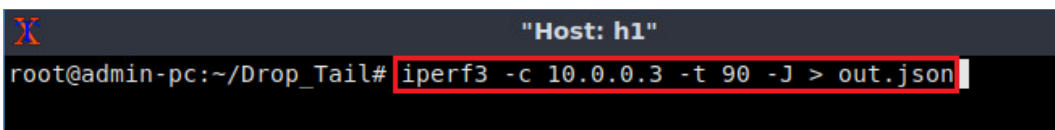


Figure 21. Running iPerf3 client on host h1.

Step 5. Type the following `ping` command in host h2's terminal without executing it.

```
ping 10.0.0.3 -c 90
```

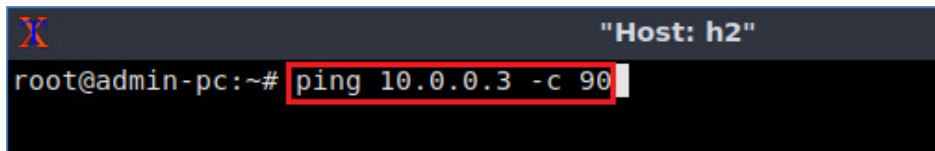


Figure 22. Typing `ping` command on host h2.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal. Then, go back to the queue plotting window and observe the queue occupancy.

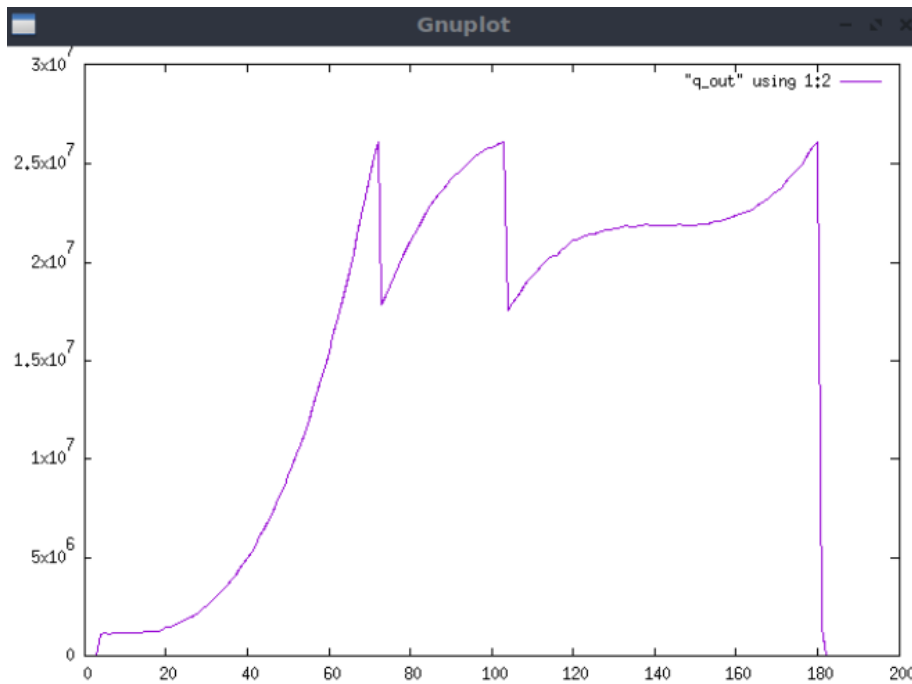


Figure 23. Queue occupancy on switch S2's `s2-eth2` interface.

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 7. In the queue plotting window, press the `q` key on your keyboard to stop plotting the queue.

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

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

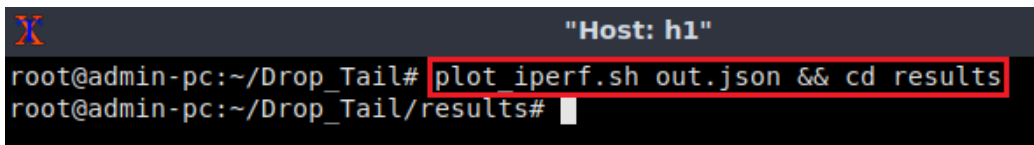


Figure 24. Generate plotting files and entering the `results` directory.

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

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

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

Figure 25. Opening the *throughput.pdf* file.

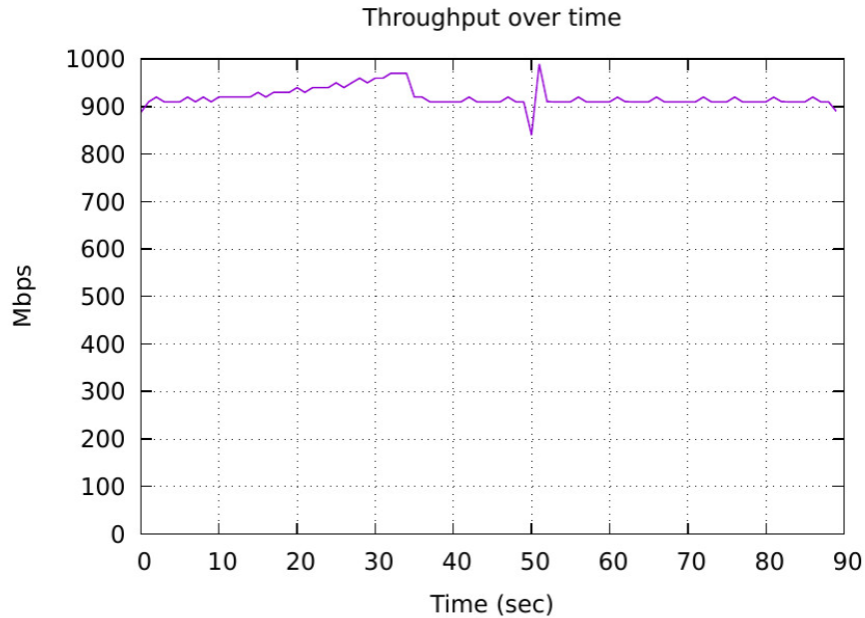


Figure 26. Measured throughput.

The figure above shows the iPerf3 test output report for the last 90 seconds. The average achieved throughput is approximately 900 Mbps. We can see now that the maximum throughput was almost achieved (1 Gbps) when we set the switch's buffer size to $10 \cdot \text{BDP}$.

Step 10. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

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

Figure 27. Opening the *RTT.pdf* file.

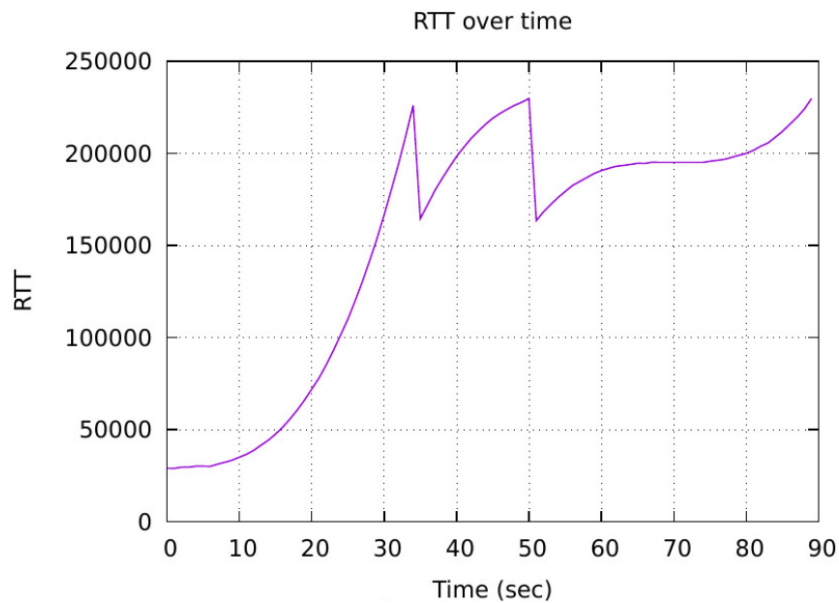


Figure 28. Measured round-trip time.

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 11. Close the *RTT.pdf* window then go back to h2's terminal to see the `ping` output.

```

Host: h2
64 bytes from 10.0.0.3: icmp_seq=72 ttl=64 time=227 ms
64 bytes from 10.0.0.3: icmp_seq=73 ttl=64 time=228 ms
64 bytes from 10.0.0.3: icmp_seq=74 ttl=64 time=164 ms
64 bytes from 10.0.0.3: icmp_seq=75 ttl=64 time=165 ms
64 bytes from 10.0.0.3: icmp_seq=76 ttl=64 time=169 ms
64 bytes from 10.0.0.3: icmp_seq=77 ttl=64 time=173 ms
64 bytes from 10.0.0.3: icmp_seq=78 ttl=64 time=177 ms
64 bytes from 10.0.0.3: icmp_seq=79 ttl=64 time=180 ms
64 bytes from 10.0.0.3: icmp_seq=80 ttl=64 time=183 ms
64 bytes from 10.0.0.3: icmp_seq=81 ttl=64 time=185 ms
64 bytes from 10.0.0.3: icmp_seq=82 ttl=64 time=187 ms
64 bytes from 10.0.0.3: icmp_seq=83 ttl=64 time=190 ms
64 bytes from 10.0.0.3: icmp_seq=84 ttl=64 time=190 ms
64 bytes from 10.0.0.3: icmp_seq=85 ttl=64 time=191 ms
64 bytes from 10.0.0.3: icmp_seq=86 ttl=64 time=192 ms
64 bytes from 10.0.0.3: icmp_seq=87 ttl=64 time=193 ms
64 bytes from 10.0.0.3: icmp_seq=88 ttl=64 time=194 ms
64 bytes from 10.0.0.3: icmp_seq=89 ttl=64 time=194 ms
64 bytes from 10.0.0.3: icmp_seq=90 ttl=64 time=20.1 ms

--- 10.0.0.3 ping statistics ---
90 packets transmitted, 90 received, 0% packet loss, time 103ms
rtt min/avg/max/mdev = 20.083/192.823/228.407/26.954 ms
root@admin-pc:~#
    
```

Figure 29. `ping` test result.

The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 20.083, 192.823, 228.407, and 26.954 milliseconds, respectively. The

output also verifies that there is bufferbloat as the average latency (192.823) is significantly greater than the configured delay (20ms).

Step 12. Open the congestion window (*cwnd.pdf*) file using the command below.

```
xdg-open cwnd.pdf
```

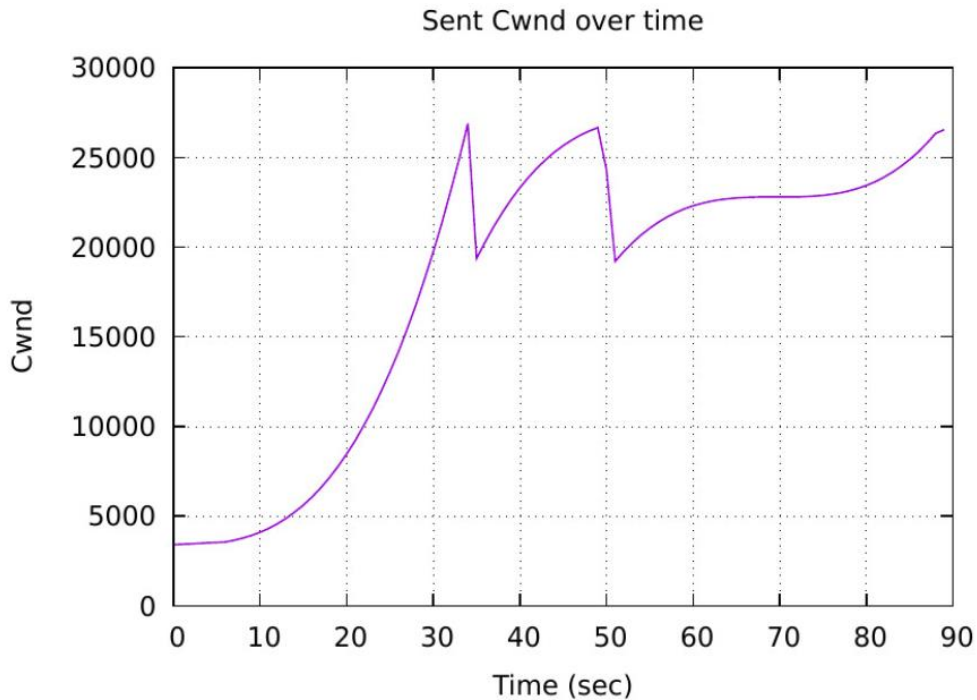


Figure 30. Congestion window evolution.

The graph above shows the evolution of the congestion window which peaked at 2.5 Mbytes. In the next section you will configure Random Early Detection on switch S2 and observe how the algorithm controls the queue length.

Step 13. To stop *iperf3* server in host h3 press `Ctrl+c`.

4 Configuring PIE on switch S2

In this section, you are going to configure PIE in switch S2's *s2-eth2* interface. Then, you will conduct throughput and latency measurements between host h1 and host h3. Note that the buffer size is set to 10·BDP.

4.1 Setting PIE parameter on switch S2's egress interface

Step 1. Apply `pie` 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*.

- `limit`: 17476
- `target`: 22ms
- `tupdate`: 5ms
- `alpha`: 2
- `beta`: 25

```
sudo tc qdisc add dev s2-eth2 parent 1: handle 2: pie limit 17476 target 22ms  
tupdate 5ms alpha 2 beta 25
```

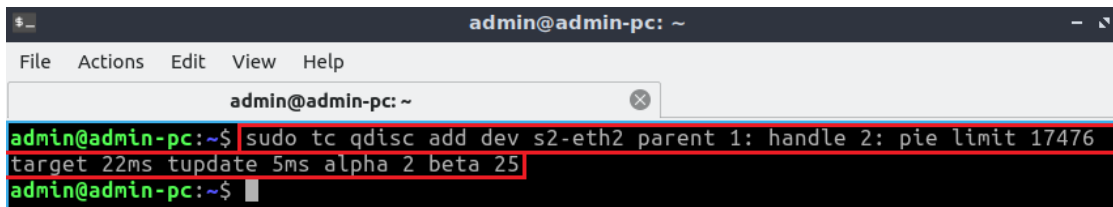


Figure 31. Setting PIE parameters on switch S2's s2-eth2 interface.

4.2 Throughput and latency tests

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

```
iperf3 -s
```

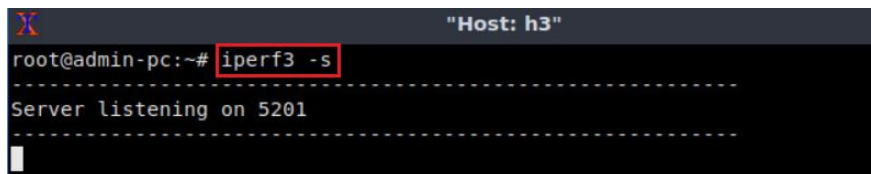


Figure 32. Starting iPerf3 server on host h3.

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
```

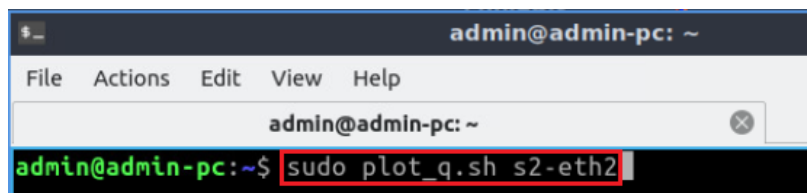


Figure 33. Plotting the queue occupancy on switch S2's s2-eth2 interface.

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.

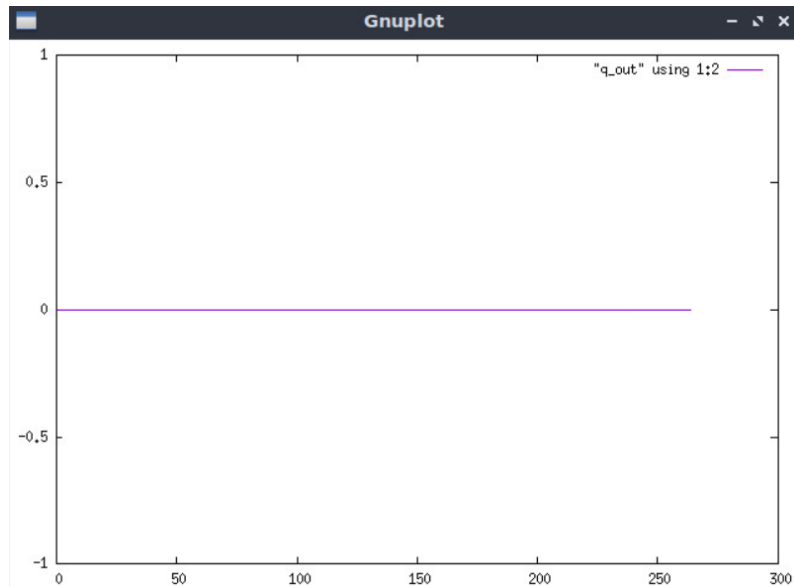


Figure 34. Queue occupancy on switch S2's s2-eth2 interface.

Step 3. Exit from *Drop_Tail/results* directory, then create a directory *PIE* and navigate into it using the following command.

```
cd ../../ && mkdir PIE && cd PIE
```

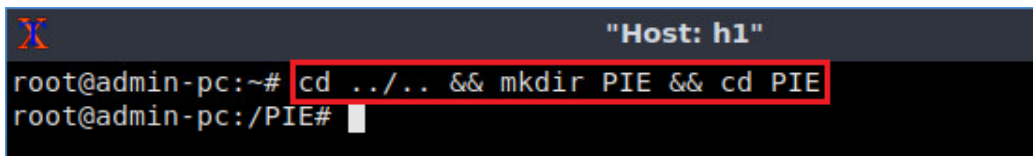


Figure 35. Creating and navigating into directory *RED*.

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 90 -J > out.json
```

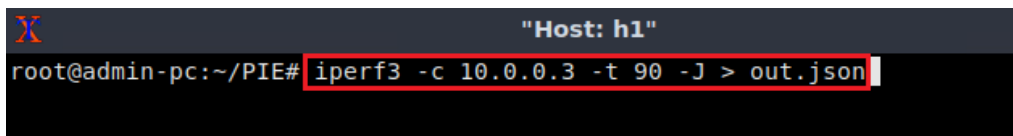


Figure 36. Running iPerf3 client on host h1.

Step 5. Type the following `ping` command in host h2's terminal without executing it.

```
ping 10.0.0.3 -c 90
```

```

X "Host: h2"
root@admin-pc:~# ping 10.0.0.3 -c 90
    
```

Figure 37. Typing `ping` command on host h2.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal. Then, go back to the queue plotting window and observe the queue occupancy.

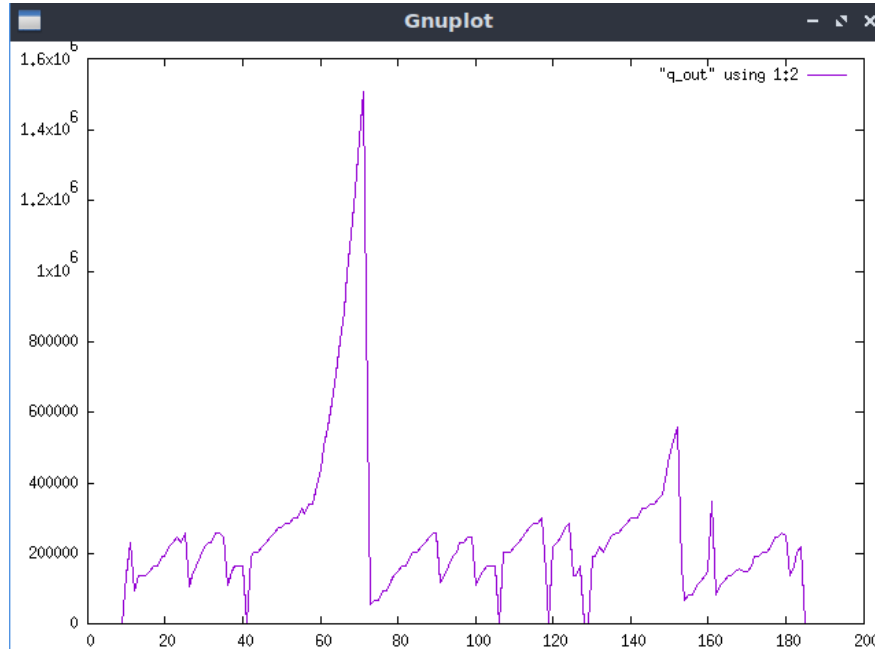


Figure 38. Queue occupancy on switch S2's `s2-eth2` interface.

The graph above shows that the queue occupancy peaked around $1.6 \cdot 10^6$ bytes, which is closer to a buffer of BDP size.

Step 7. In the queue plotting window, press the `S` key on your keyboard to stop plotting the queue.

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

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

```

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

Figure 39. 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:/PIE/results# xdg-open throughput.pdf
    
```

Figure 40. Opening the *throughput.pdf* file.

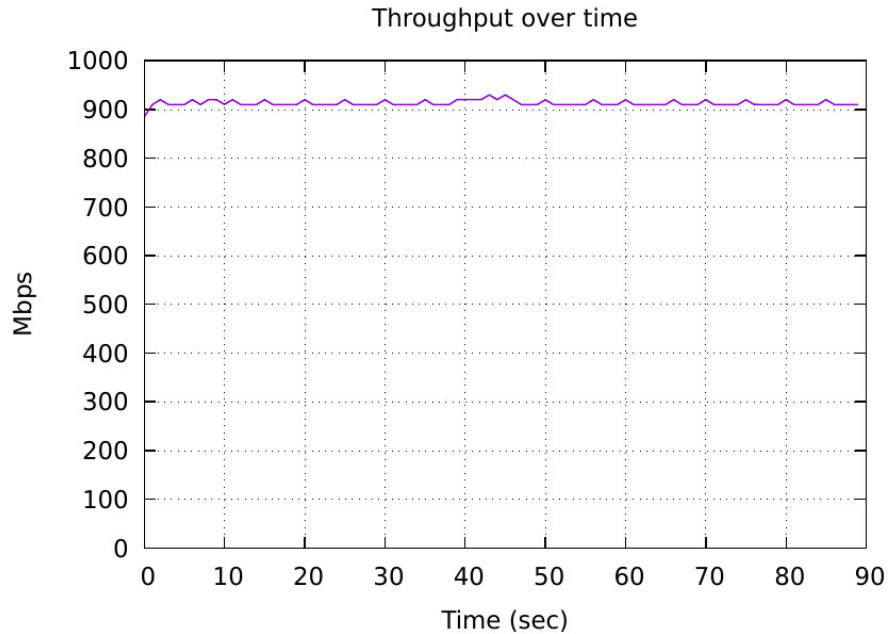


Figure 40. Measured throughput.

The figure above shows the iPerf3 test output report for the last 90 seconds. The average achieved throughput is 900 Mbps. We can see now that the maximum throughput is also achieved (1 Gbps) when we set PIE at the egress port of switch S2.

Step 10. Close the *throughput.pdf* window then open the Round-Trip Time (*RTT.pdf*) file using the command below.

```

xdg-open RTT.pdf
    
```

```

Host: h1
root@admin-pc:~/PIE/results# xdg-open RTT.pdf
    
```

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

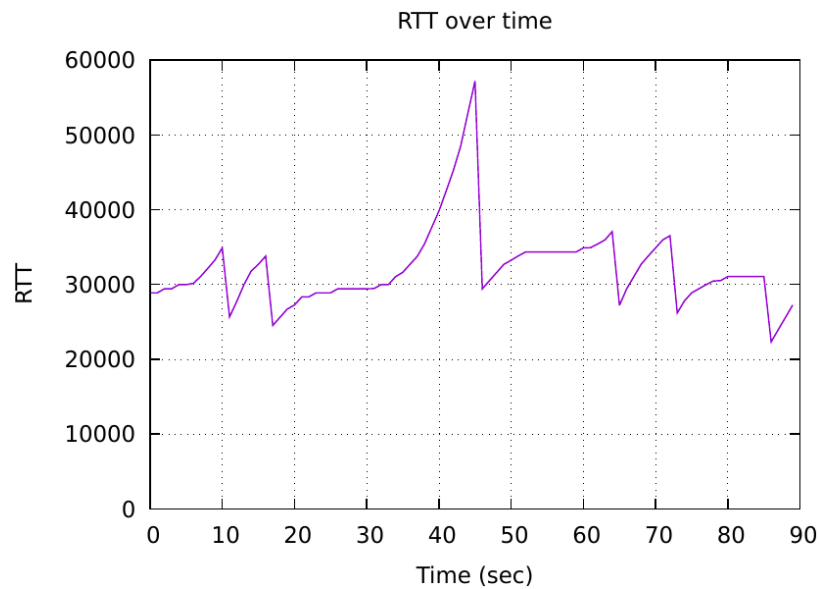


Figure 42. Measured Round-Trip Time.

The graph above shows that the RTT was contained between 30ms and 40ms which is not significantly greater than the configured delay (20ms) thus, there is not bufferbloat. Since the AQM algorithm configured on the switch is applying a dropping policy to prevent unnecessary delays.

Step 11. Close the *RTT.pdf* window then go back to h2's terminal to see the `ping` output.

```

"Host: h2"
64 bytes from 10.0.0.3: icmp_seq=72 ttl=64 time=31.1 ms
64 bytes from 10.0.0.3: icmp_seq=73 ttl=64 time=33.3 ms
64 bytes from 10.0.0.3: icmp_seq=74 ttl=64 time=24.6 ms
64 bytes from 10.0.0.3: icmp_seq=75 ttl=64 time=25.6 ms
64 bytes from 10.0.0.3: icmp_seq=76 ttl=64 time=26.8 ms
64 bytes from 10.0.0.3: icmp_seq=77 ttl=64 time=27.3 ms
64 bytes from 10.0.0.3: icmp_seq=78 ttl=64 time=27.8 ms
64 bytes from 10.0.0.3: icmp_seq=79 ttl=64 time=28.4 ms
64 bytes from 10.0.0.3: icmp_seq=80 ttl=64 time=28.5 ms
64 bytes from 10.0.0.3: icmp_seq=81 ttl=64 time=28.9 ms
64 bytes from 10.0.0.3: icmp_seq=82 ttl=64 time=28.9 ms
64 bytes from 10.0.0.3: icmp_seq=83 ttl=64 time=28.9 ms
64 bytes from 10.0.0.3: icmp_seq=84 ttl=64 time=29.3 ms
64 bytes from 10.0.0.3: icmp_seq=85 ttl=64 time=28.9 ms
64 bytes from 10.0.0.3: icmp_seq=86 ttl=64 time=28.9 ms
64 bytes from 10.0.0.3: icmp_seq=87 ttl=64 time=28.9 ms
64 bytes from 10.0.0.3: icmp_seq=88 ttl=64 time=29.5 ms
64 bytes from 10.0.0.3: icmp_seq=89 ttl=64 time=29.5 ms
64 bytes from 10.0.0.3: icmp_seq=90 ttl=64 time=30.5 ms

--- 10.0.0.3 ping statistics ---
90 packets transmitted, 90 received, 0% packet loss, time 200ms
rtt min/avg/max/mdev = 20.078/64.165/217.367/70.417 ms
root@admin-pc:~#
    
```

Figure 43. `ping` test result.

The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip

Time (RTT) were 20.078, 64.165, 217.367, and 70.417 milliseconds, respectively. The output also verifies that there is not bufferbloat as the average latency (34.048) is not significantly greater than the configured delay (20ms).

Step 12. Open the congestion window (*cwnd.pdf*) file using the command below.

```
xdg-open cwnd.pdf
```

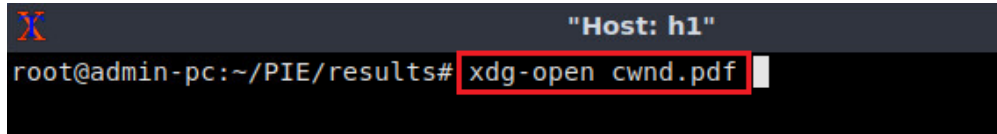


Figure 44. Opening the *cwnd.pdf* file.

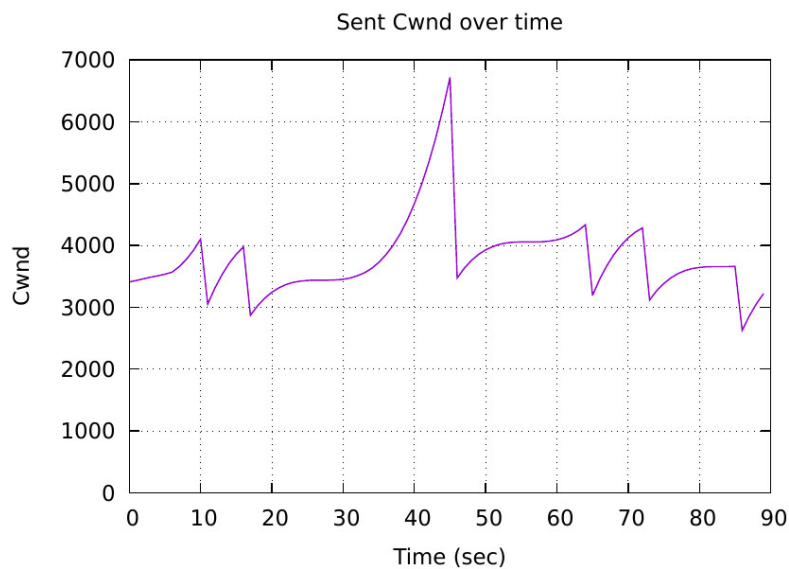


Figure 45. Evolution of the congestion window.

The graph above shows the evolution of the congestion window which peaked around 5 Mbytes. In the next section you will maintain the current parameters of Random Early Detection on switch S2 however, you will change the link rate in order to verify if the algorithm performs well if the network condition changes.

Step 13. To stop *iperf3* server in host h3 press `Ctrl+c`.

4.3 Changing the bandwidth to 100Mbps

This section is aimed to analyze the impact of changing the bandwidth to 100 Mbps while RED is tuned to work with the previous network condition. The results will show that RED requires a reconfiguration if the network conditions changes (i.e., latency, bandwidth, loss rate). First, you will change the bandwidth to 100 Mbps then, you will observe the queue occupancy, RTT and congestion window in order to evaluate the performance of RED when the network condition changes.

Step 1. Apply `tbw` 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
```

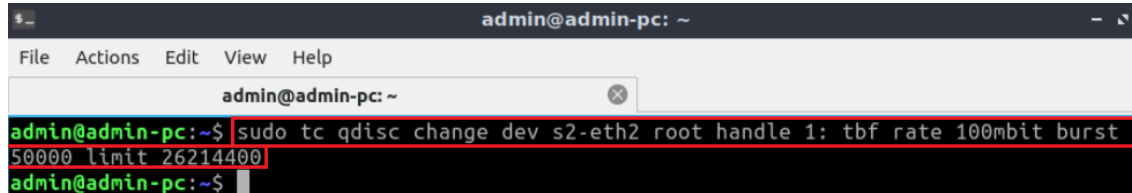


Figure 46. Limiting rate to 100 Mbps and keeping the buffer size to 10·BDP on switch S2's interface.

4.4 Throughput and latency tests

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

```
iperf3 -s
```

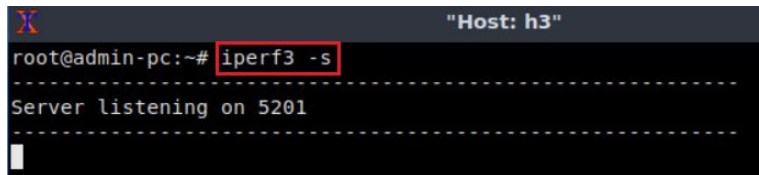


Figure 47. Starting iPerf3 server on host h3.

Step 2. In the Client's terminal, type the command below to plot the switch's queue in real-time.

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

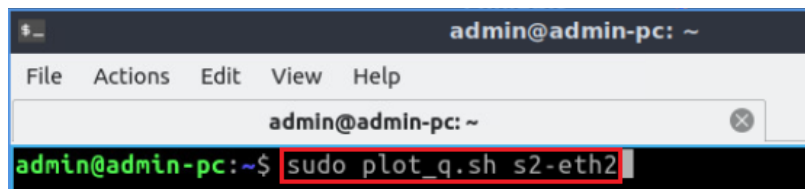


Figure 48. Plotting the queue occupancy on switch S2's `s2-eth2` interface.

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.

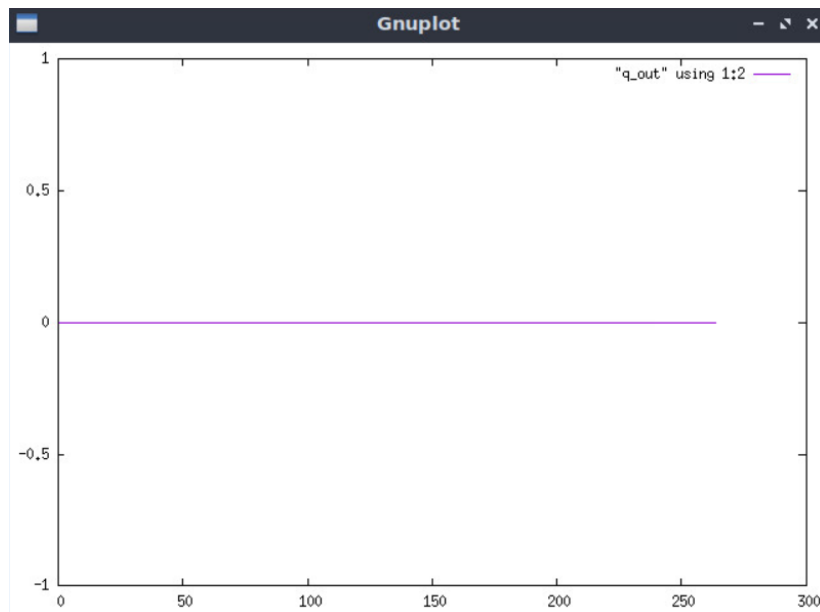


Figure 49. Queue occupancy on switch S2's s2-eth2 interface.

Step 3. Exit from *PIE/results* directory, then create a directory *PIE_100Mbps* and navigate into it using the following command.

```
cd ../../ && mkdir PIE_100Mbps && cd PIE_100Mbps
```

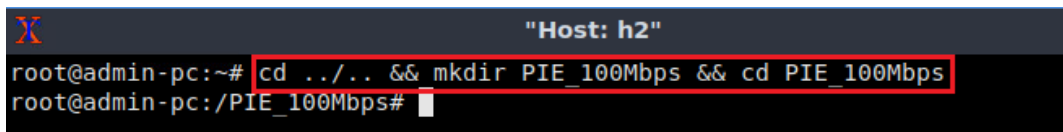


Figure 50. Creating and navigating into directory *PIE_100Mbps*.

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 90 -J > out.json
```

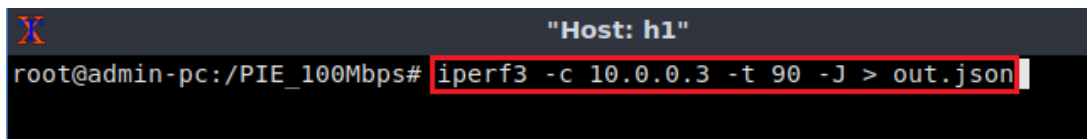


Figure 51. Running iPerf3 client on host h1.

Step 5. Type the following `ping` command in host h2's terminal without executing it.

```
ping 10.0.0.3 -c 90
```

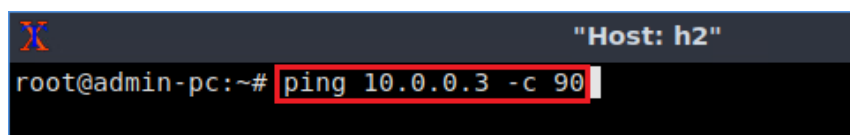


Figure 52. Typing `ping` command on host h2.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal. Then, go back to the queue plotting window and observe the queue occupancy.

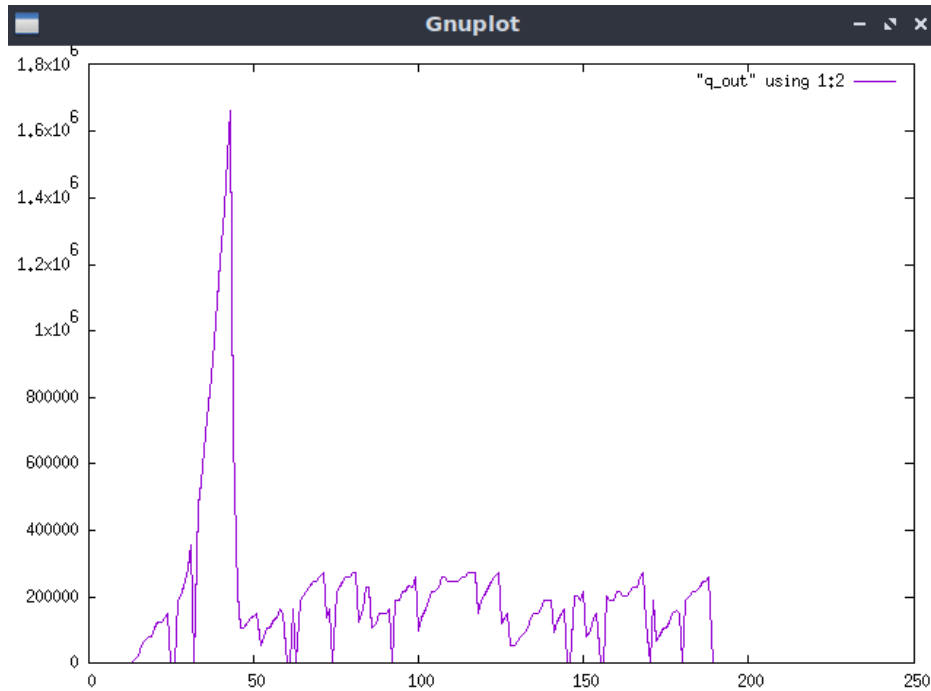


Figure 53. Queue occupancy on switch S2's s2-eth2 interface.

The graph above shows that the queue occupancy peaked around $1.7 \cdot 10^6$. In this case we set a 100 Mbps link when PIE is configured to operate for 1 Gbps link therefore, the point of operation changed. However, bufferbloat is not experienced because PIE controller configuration does not depend on the network condition, it depends on the queue length and latency as shown in section 1.1.

Step 7. In the queue plotting window, press the `S` key on your keyboard to stop plotting the queue.

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

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

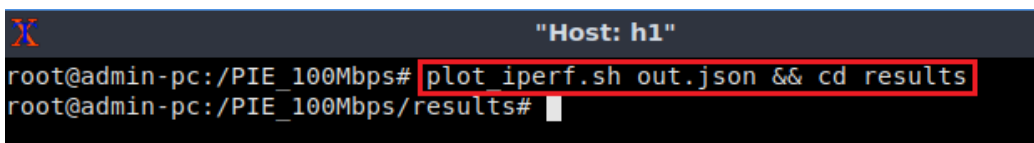


Figure 54. Generate plotting files and entering the *results* directory.

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

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

```
root@admin-pc:~/PIE_100Mbps/results# xdg-open throughput.pdf
```

Figure 55. Opening the *throughput.pdf* file.

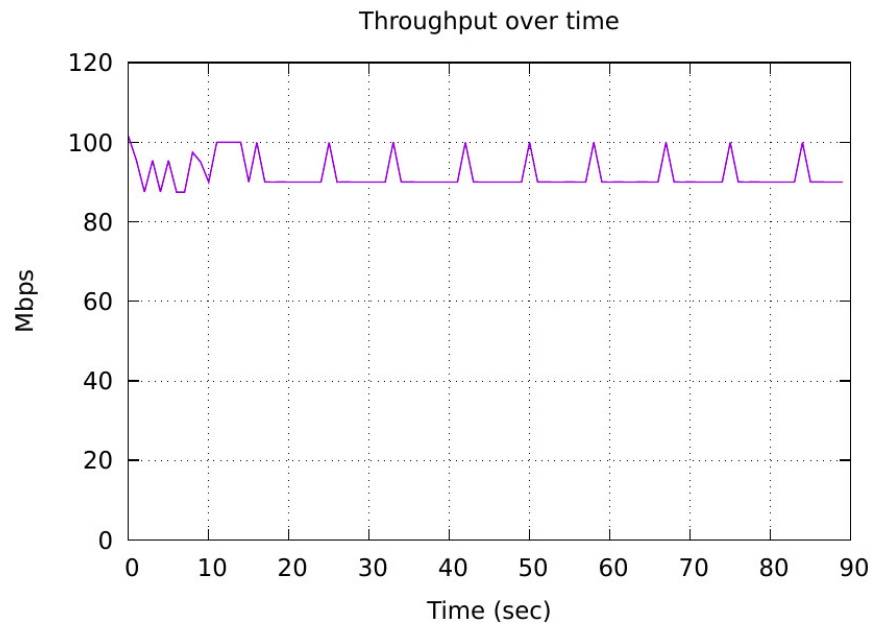


Figure 55. Measured throughput.

The figure above shows the iPerf3 test output report for the last 90 seconds. The average achieved throughput is 100 Mbps.

Step 10. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

```
root@admin-pc:~/PIE_100Mbps/results# xdg-open RTT.pdf
```

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

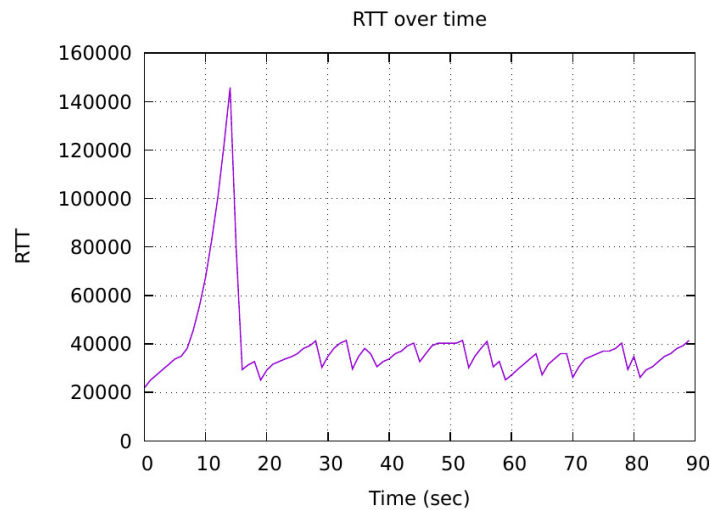


Figure 57. Measured Round-Trip Time.

The graph above shows that the RTT peaked at 10 seconds up to 140ms then, the latency went down and stabilize around 40ms which is closer to the default latency (20ms). This output shows that there is not bufferbloat as the average latency is not significantly greater as in the previous section. Since PIE is configured to operate on a 1 Gbps link, for this test the point of operation changed therefore, this change does not affect the queue management policy.

Step 11. Close the *RTT.pdf* window then go back to h2's terminal to see the `ping` output.

```

X                                     "Host: h2"
64 bytes from 10.0.0.3: icmp_seq=72 ttl=64 time=29.1 ms
64 bytes from 10.0.0.3: icmp_seq=73 ttl=64 time=32.3 ms
64 bytes from 10.0.0.3: icmp_seq=74 ttl=64 time=34.5 ms
64 bytes from 10.0.0.3: icmp_seq=75 ttl=64 time=35.5 ms
64 bytes from 10.0.0.3: icmp_seq=76 ttl=64 time=35.10 ms
64 bytes from 10.0.0.3: icmp_seq=77 ttl=64 time=35.10 ms
64 bytes from 10.0.0.3: icmp_seq=78 ttl=64 time=37.1 ms
64 bytes from 10.0.0.3: icmp_seq=79 ttl=64 time=39.3 ms
64 bytes from 10.0.0.3: icmp_seq=80 ttl=64 time=39.9 ms
64 bytes from 10.0.0.3: icmp_seq=81 ttl=64 time=32.7 ms
64 bytes from 10.0.0.3: icmp_seq=82 ttl=64 time=26.2 ms
64 bytes from 10.0.0.3: icmp_seq=83 ttl=64 time=27.9 ms
64 bytes from 10.0.0.3: icmp_seq=84 ttl=64 time=29.9 ms
64 bytes from 10.0.0.3: icmp_seq=85 ttl=64 time=31.1 ms
64 bytes from 10.0.0.3: icmp_seq=86 ttl=64 time=33.4 ms
64 bytes from 10.0.0.3: icmp_seq=87 ttl=64 time=35.3 ms
64 bytes from 10.0.0.3: icmp_seq=88 ttl=64 time=37.7 ms
64 bytes from 10.0.0.3: icmp_seq=89 ttl=64 time=38.8 ms
64 bytes from 10.0.0.3: icmp_seq=90 ttl=64 time=41.0 ms

--- 10.0.0.3 ping statistics ---
90 packets transmitted, 90 received, 0% packet loss, time 191ms
rtt min/avg/max/mdev = 22.073/37.541/139.705/16.828 ms
root@admin-pc:/PIE_100Mbps#
    
```

Figure 58. `ping` test result.

The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip

Time (RTT) were 22.073, 37.541, 139.705 and 16.828 milliseconds, respectively. The output also verifies that there is a bufferbloat problem as the average latency (186.175) is significantly greater than the configured delay (20ms).

Step 12. Close the *RTT.pdf* window then open the retransmissions (retransmits) file using the command below.

```
xdg-open cwnd.pdf
```

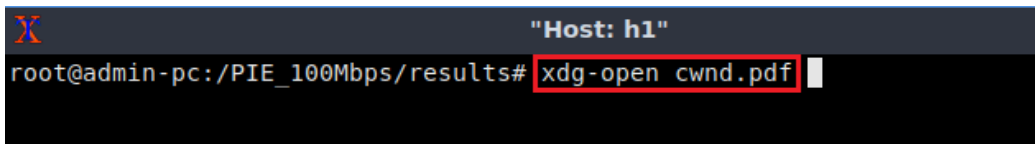


Figure 59. Opening the *cwnd.pdf* file.

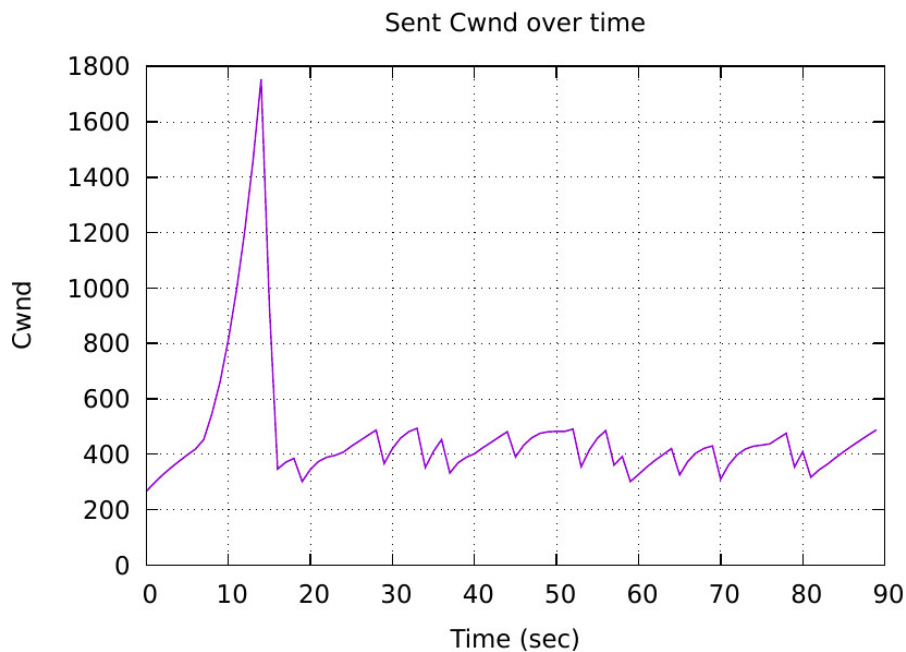


Figure 60. Evolution of the congestion window.

The graph above shows the evolution of the congestion window which peaked around 1.8 Mbytes.

Step 13. To stop *iperf3* server in host h3 press `Ctrl+c`.

This concludes Lab 19. Stop the emulation and then exit out of MiniEdit.

References

1. Bufferbloat project, "Bufferbloat" [Online]. Available: <https://www.bufferbloat.net/projects/>
2. IETF draft, "PIE: A lightweight control scheme to address the bufferbloat problem," [Online]. Available: <https://tools.ietf.org/html/draft-ietf-aqm-pie-10>

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.