



UNIVERSITY OF
SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 16: Random Early Detection (RED)

Document Version: **10-10-2019**



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
1.1 Random Early Detection mechanism	4
2 Lab topology.....	6
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	13
4 Configuring RED on switch S2	18
4.1 Setting RED parameter on switch S2's egress interface	19
4.2 Throughput and latency tests	19
4.3 Changing the bandwidth to 100Mbps	24
4.4 Throughput and latency tests	25
References	30

Overview

This lab explains the Random Early Detection (RED) Active Queue Management (AQM) algorithm. This algorithm is aimed to mitigate high end-to-end latency by controlling the average queue length in routers' buffers. Throughput, latency and queue length measurements are conducted in this lab to verify the impact of the dropping policy provided RED.

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 RED handling the queuing policy in a router egress port.
4. Visualize queue occupancy in a router.
5. Analyze how RED 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 RED'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 RED on switch S2.

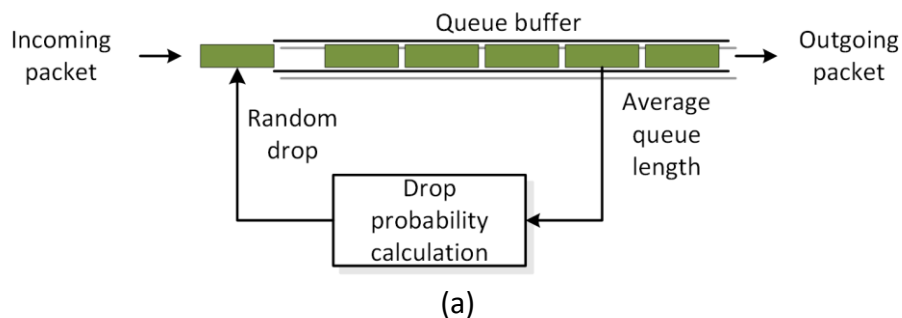
1 Introduction

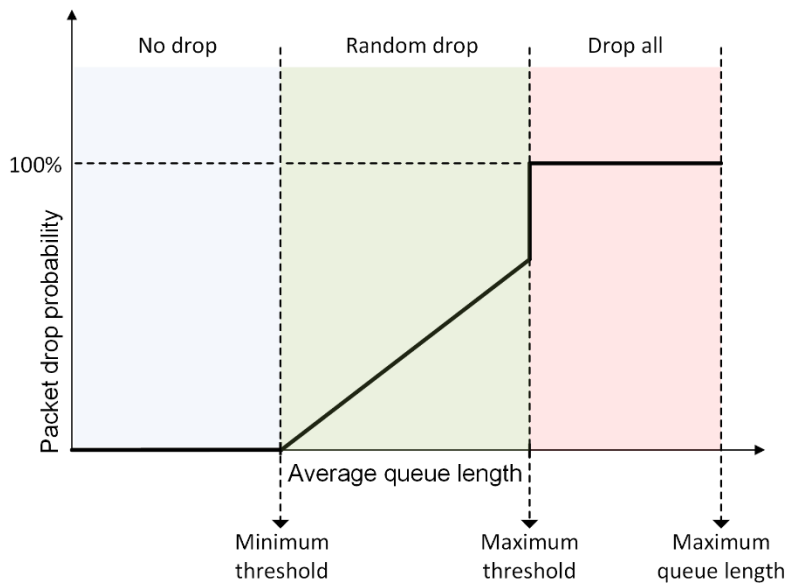
End-to-end-congestion control is widely used in the current Internet to prevent congestion collapse. However, because data traffic is inherently bursty, routers are provisioned with large buffers to absorb this burstiness and maintain high link utilization. The downside of these large buffers is that if traditional drop-tail buffer management is used, there will be high queuing delays at congested routers. Thus, drop-tail buffer management forces network operators to choose between high utilization (requiring large buffers), or low delay (requiring small buffers).

Random Early Detection (RED) was proposed by Floyd and Van Jacobson¹ to address network congestion in a responsive rather than reactive manner. The main goal of RED is to provide congestion avoidance by controlling the average queue size. Other goals are the avoidance of global synchronization and introduce fairness to reduce the bias against bursty traffic. TCP global synchronization happens to a TCP flow during periods of congestion when each sender reduces and then increase their transmission rate at the same time due to packet loss.

1.1 Random Early Detection mechanism

Figure 1(a) illustrates scenario where a router's buffer is managed by Random Early Detection. RED uses a low-pass filter with an exponential moving average to calculate the average queue size. Then, the average queue size is compared to two thresholds, a minimum threshold and a maximum threshold. Consequently, the packet drop probability is determined by the function shown in the Figure 1(b). When the average queue size is less than the minimum threshold, no packets are dropped. When the average queue size is greater than the maximum threshold, every arriving packet is marked therefore, they are dropped. When the average queue size is between the minimum and the maximum threshold, each arriving packet is marked with drop probability. Thus, RED has two separate algorithms. First, the algorithm for computing the average queue size that determines the degree of burstiness allowed in the queue. Secondly, the algorithm for calculating the packet marking probability, which determines how frequently the gateway marks or drop packets, given the current level of congestion. The goal is for the gateway to mark packets at evenly spaced intervals, in order to avoid biases global synchronization by marking packets to control the average queue size.





(b)

Figure 1. Behavior of Random Early Detection AQM. (a) Buffer managed by RED AQM. (b) RED dropping function.

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

```
tc qdisc [add | ...] dev [dev_id] root red limit [BYTES] max [BYTES] min
[BYTES] burst [BYTES] avpkt [BYTES] bandwidth [BPS] [probability
[RATE]|adaptive] ecn
```

- `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.
- `red`: This parameter specifies the Random Early Detection algorithm.
- `limit [BYTES]`: Hard limit on the real (not average) queue size in bytes. Further packets are dropped. Should be set higher than `max+burst`.
- `max [BYTES]`: This parameter specifies the maximum average queue size. After this value, the dropping probability is 100%. It is recommended to set this value to `limit/4`.
- `min [BYTES]`: This parameter specifies the minimum average queue size. Below this value, no packet is dropped. Above this threshold, the dropping probability is established by `probability` or it increases linearly if the parameter `adaptive` is set.
- `avpkt`: Used with `burst` to determine the time constant for average queue size calculations. It is suggested 1000 as good value.

- `burst [BYTES]`: Used for determining how fast the average queue size is influenced by the real queue size. Larger values make the calculation slower, allowing longer bursts of traffic before the marking or dropping phase starts. Empirical evaluations suggest the following guideline to set this value: $(2 \cdot \text{min} + \text{max}) / (3 \cdot \text{avpkt})$.
- `bandwidth [BPS]`: This value is optional and used to calculate the average queue size after any idle time. It should be set to the bandwidth of the interface. This parameter does not limit the rate. The default value is 10Mbps.
- `ecn`: This parameter enables RED to notify remote hosts that their rate exceeds the amount of bandwidth available. Non-ECN capable hosts can only be notified by dropping a packet.
- `probability`: This value specifies the dropping probability after the average queue length surpass the min threshold. It is specified as a floating point from 0.0 to 1.0. Suggested values are 0.01 or 0.02 (1% or 2% respectively).
- `adaptative`: This parameter sets a dynamic value to the dropping probability. This value varies from 1% to 50%.

In this lab, we will use the `red` AQM algorithm to contain 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.

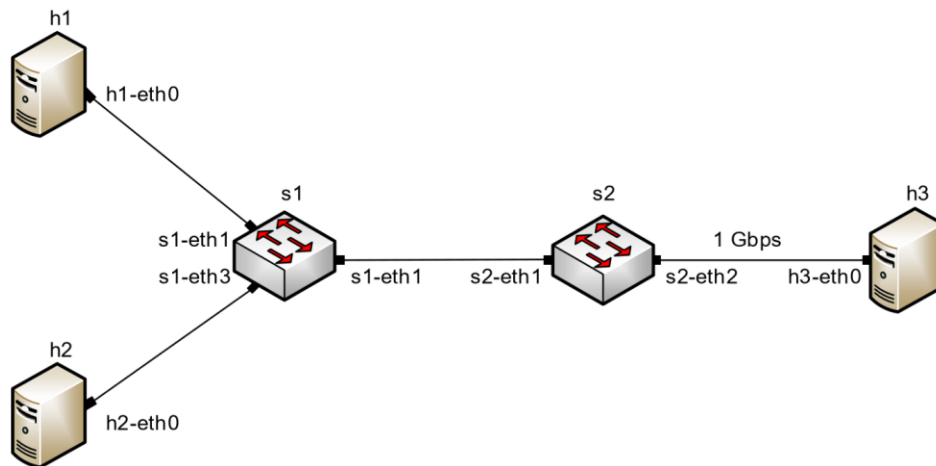


Figure 2. Lab topology.

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`.

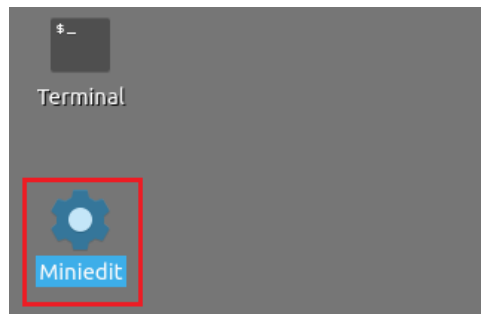


Figure 3. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 16.mn* topology file and click on *Open*.

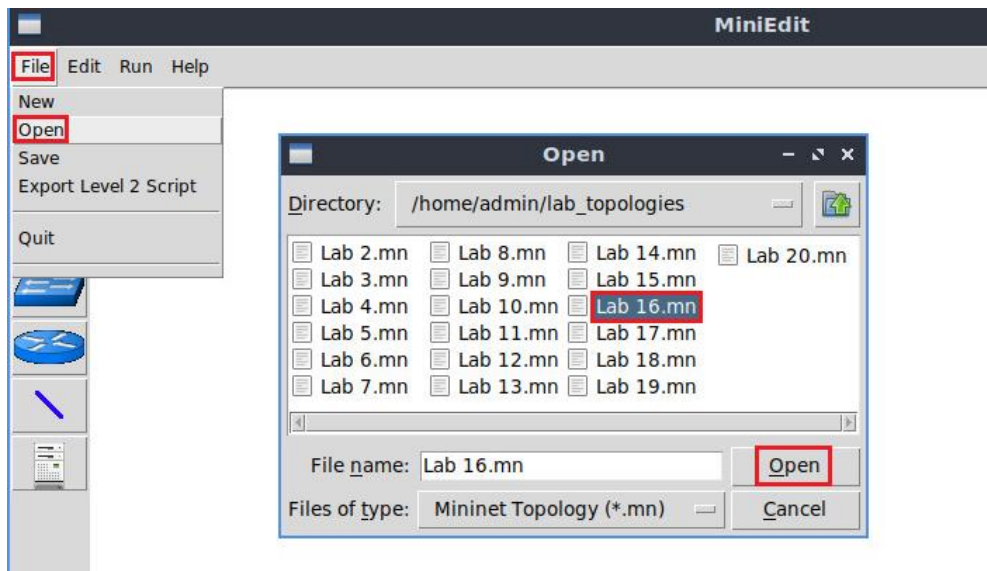


Figure 4. MiniEdit's *Open* dialog.

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.

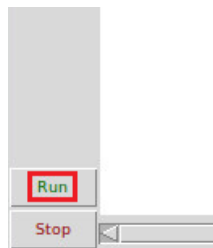


Figure 5. Running the emulation.

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.

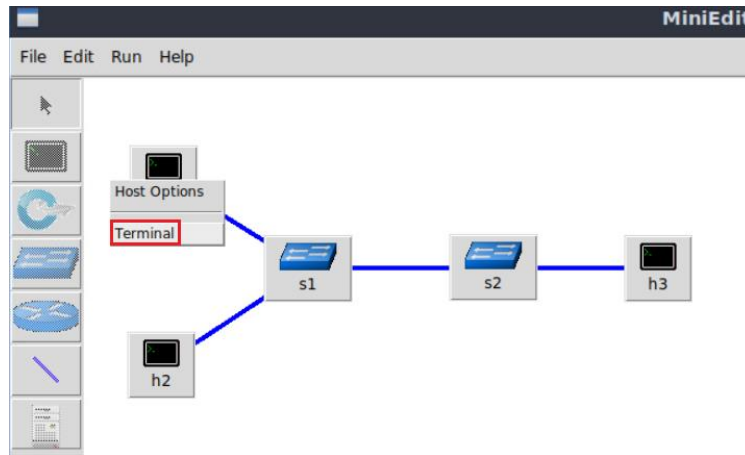


Figure 6. Opening a terminal on host h1.

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.

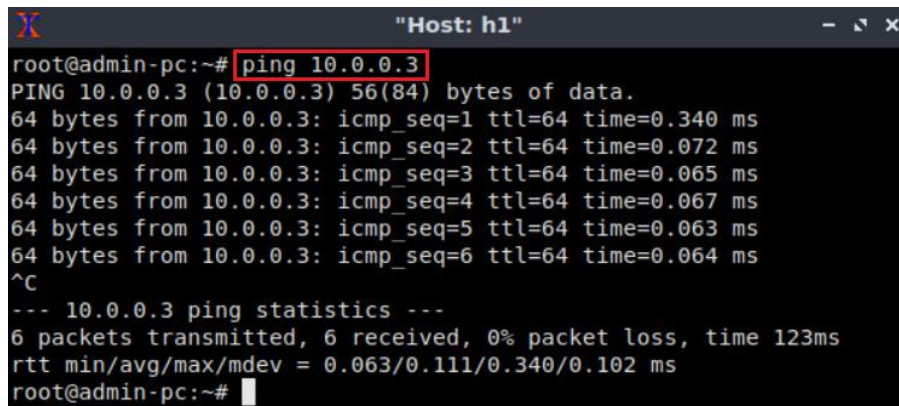


Figure 7. Connectivity test using `ping` command.

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.

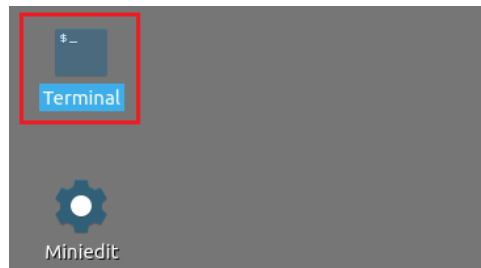


Figure 8. Shortcut to open a Linux terminal.

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
```

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`.

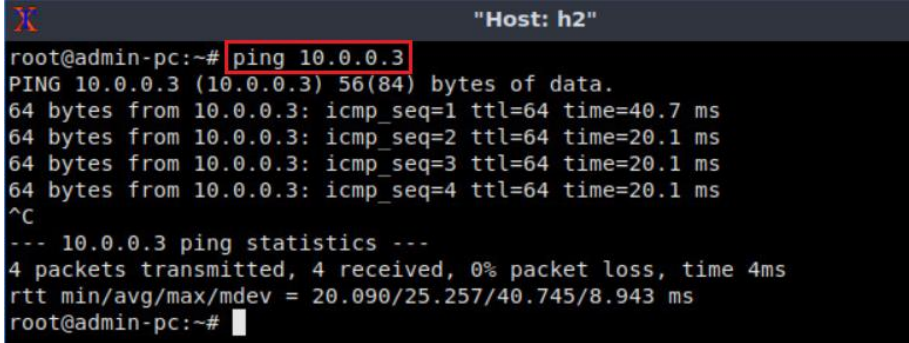
Step 1. On the terminal of host h1, type `ping 10.0.0.3`. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test. Host h1 (10.0.0.1) sent four packets to host h3 (10.0.0.3), successfully receiving responses back.

```
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 (`tbfb`) 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 `tbfb` 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 `tbfb` 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.

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

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

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

= 2,500,000 bytes \approx 2.5 Mbytes

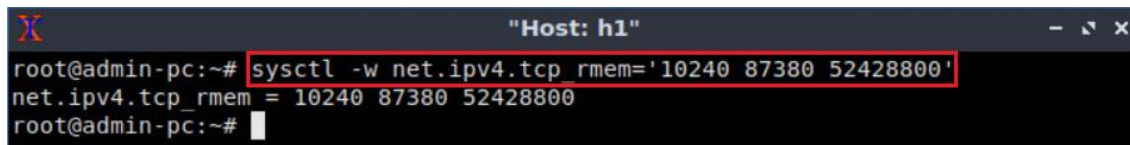
1 Mbyte = 1024^2 bytes

BDP = 2.5 Mbytes = $2.5 \cdot 1024^2$ bytes = 2,621,440 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:~#

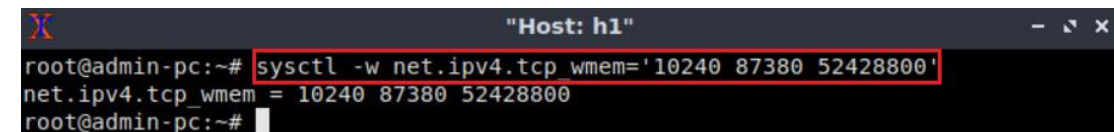
```

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:~#

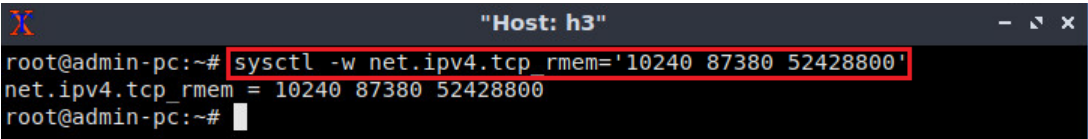
```

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'
```

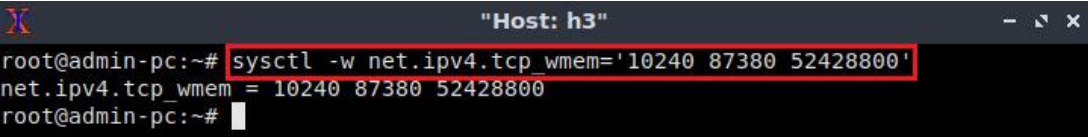


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#
```

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'
```



```
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 $10 \cdot \text{BDP}$

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`: 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 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
```

```
"Host: h3"
root@admin-pc:~# iperf3 -s
-----
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. 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 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.

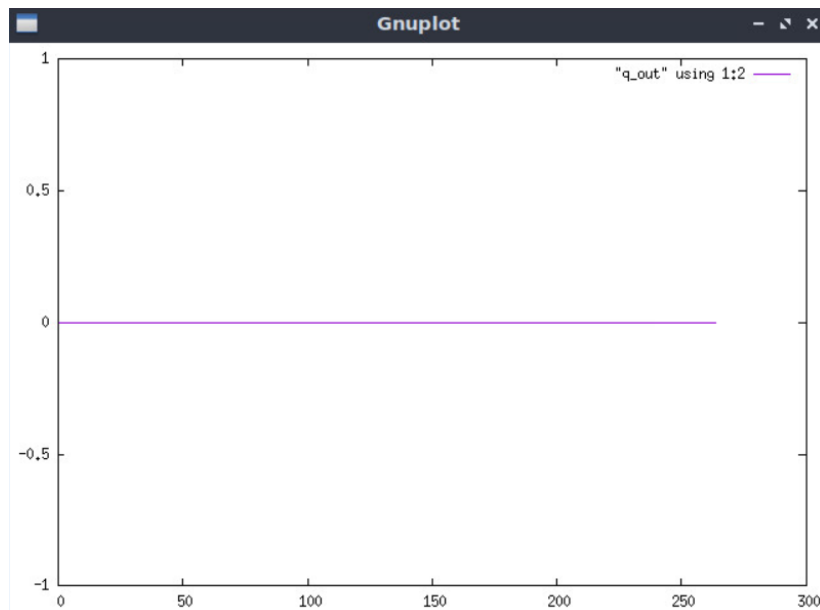


Figure 19. Queue occupancy on switch S2's s2-eth2 interface.

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
```

```

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

```

X                                     "Host: h1"
root@admin-pc:~/Drop_Tail# 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
```

```

X                                     "Host: h2"
root@admin-pc:~# 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.

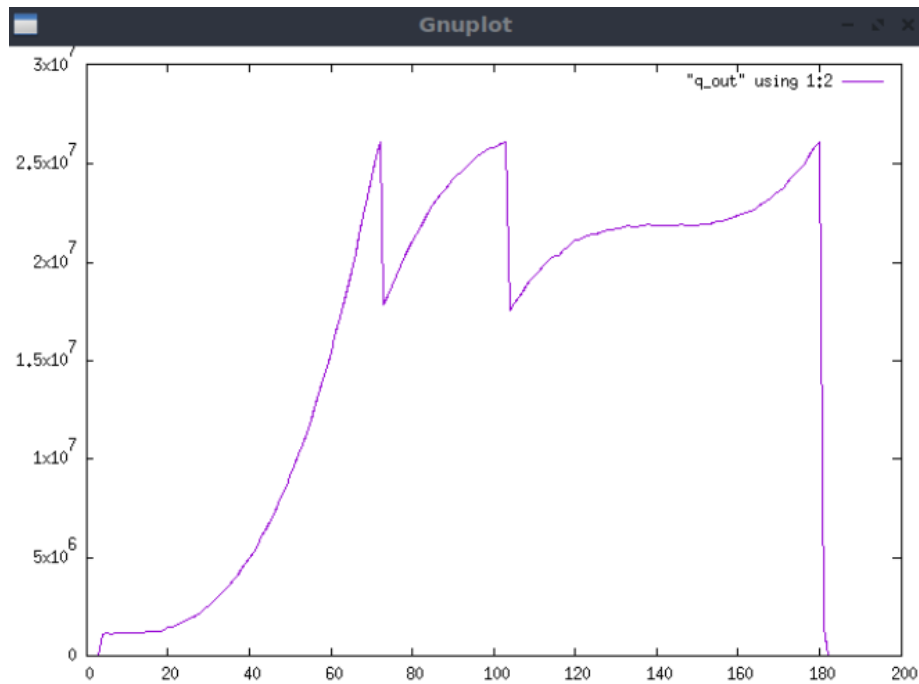


Figure 23. Queue occupancy on switch S2's *s2-eth2* interface.

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 `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
```

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

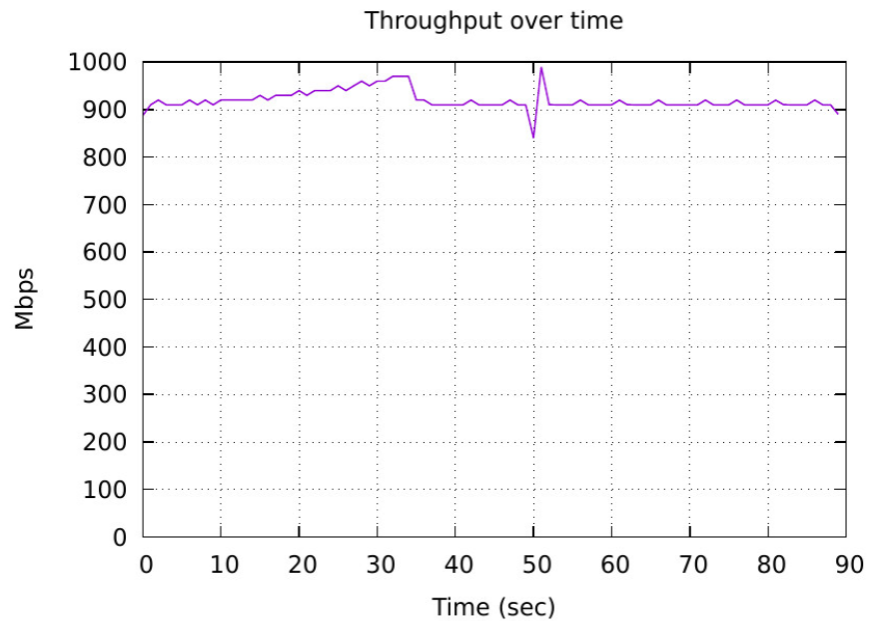


Figure 26. Measured throughput.

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
```

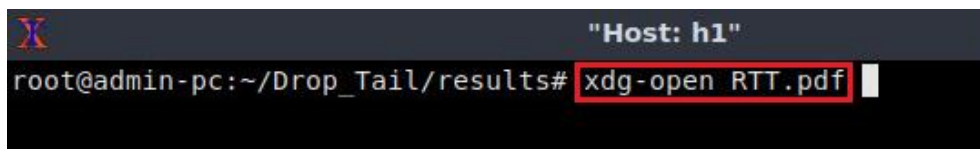


Figure 27. Opening the *RTT.pdf* file.

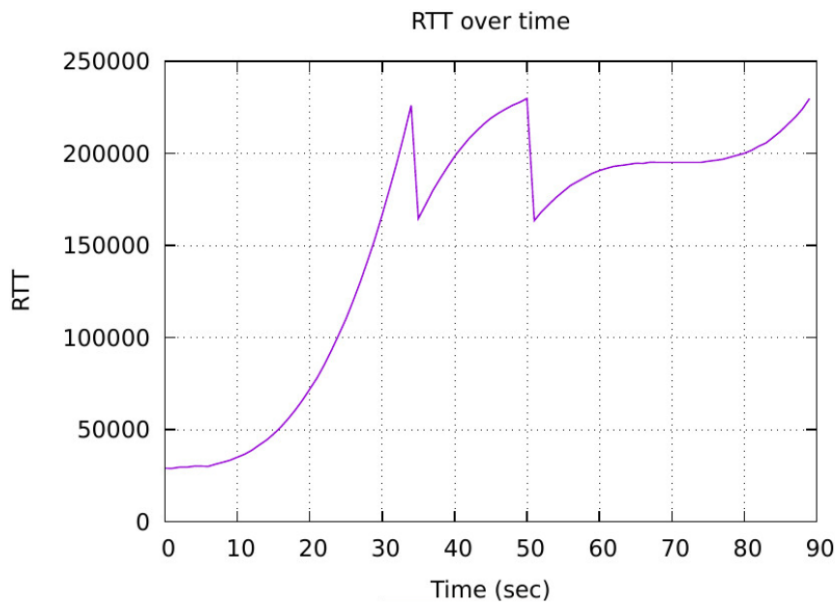


Figure 28. Measured round-trip time.

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.

```

X                                     "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
```

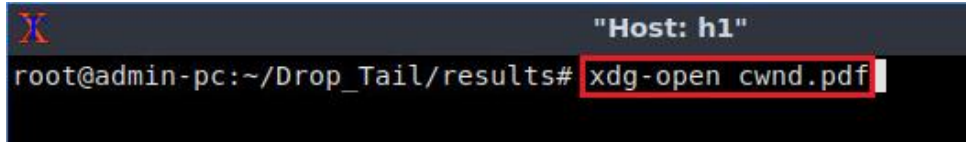


Figure 30. Opening the *cwnd.pdf* file.

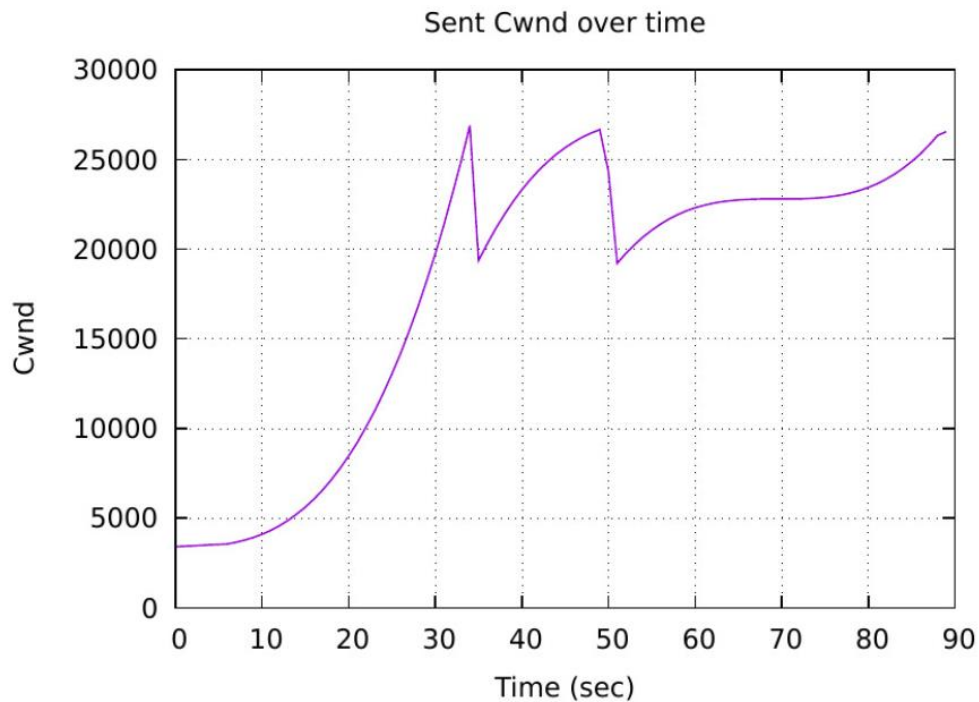


Figure 31. Congestion window evolution.

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 RED on switch S2

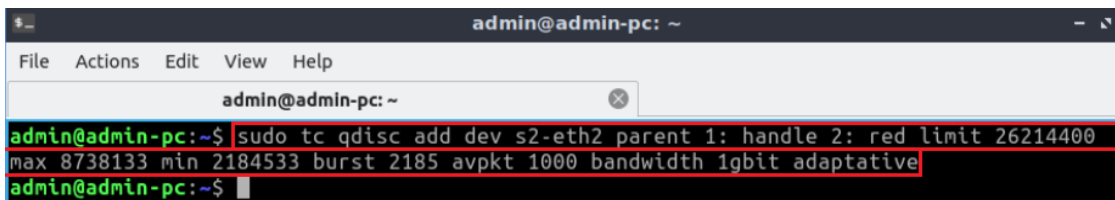
In this section, you are going to configure Random Early Detection in switch S2. Then, you will conduct throughput and latency measurements between host h1 and host h3. Note that the buffer size is set to 10·BDP.

4.1 Setting RED parameter on switch S2's egress interface

Step 1. Apply `tc` 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`: 26,214,400
- `max`: 8,738,133
- `min`: 2,184,533
- `burst`: 2185
- `avpkt`: 1000
- `bandwidth`: 1gbit
- `adaptative`

```
sudo tc qdisc add dev s2-eth2 parent 1: handle 2: red limit 26214400 max
8738133 min 2184533 burst 2185 avpkt 1000 bandwidth 1gbit adaptative
```



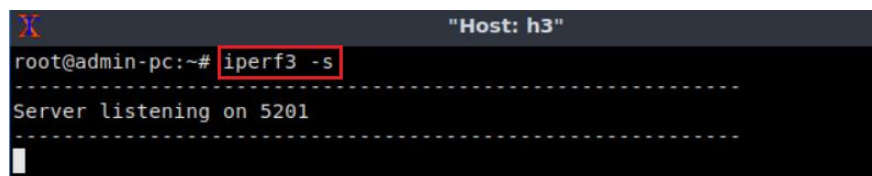
The screenshot shows a terminal window titled "admin@admin-pc: ~". The command `sudo tc qdisc add dev s2-eth2 parent 1: handle 2: red limit 26214400 max 8738133 min 2184533 burst 2185 avpkt 1000 bandwidth 1gbit adaptative` is entered and executed. The prompt returns to `admin@admin-pc:~$`.

Figure 32. Setting RED 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
```

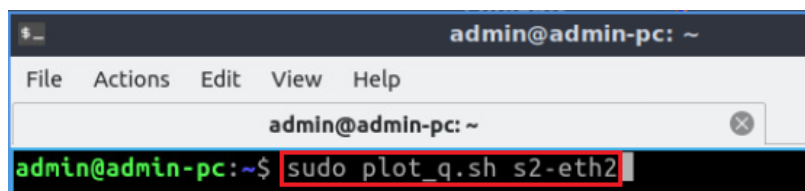


The screenshot shows a terminal window titled "Host: h3". The command `iperf3 -s` is entered and executed. The output shows "Server listening on 5201".

Figure 33. 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
```



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

Figure 34. 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.

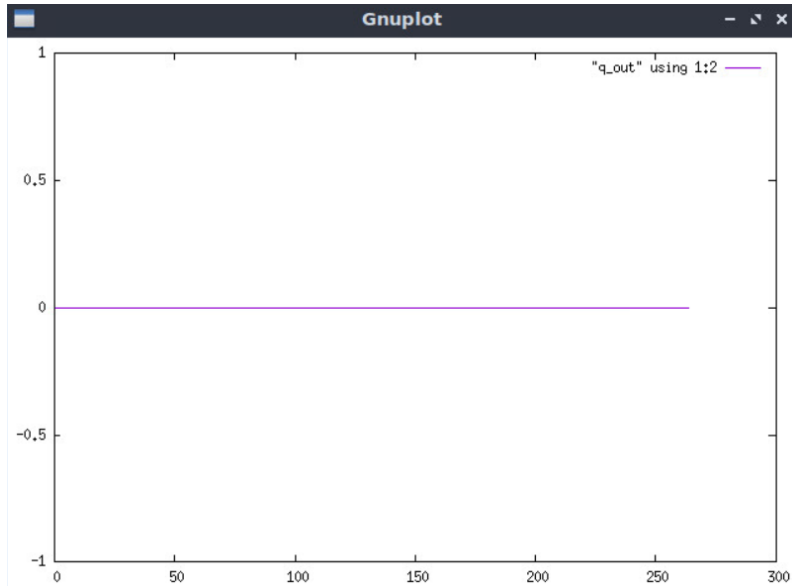


Figure 35. Queue occupancy on switch S2's s2-eth2 interface.

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

```
cd ../../ && mkdir RED && cd RED
```

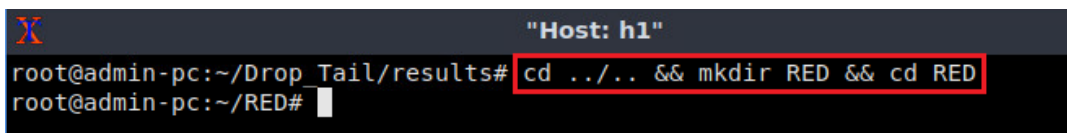


Figure 36. Creating and navigating into directory RED.

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
```

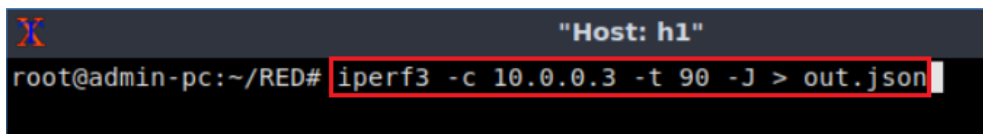


Figure 37. Running iPerf3 client on host h1.

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

```
ping 10.0.0.3 -c 90
```

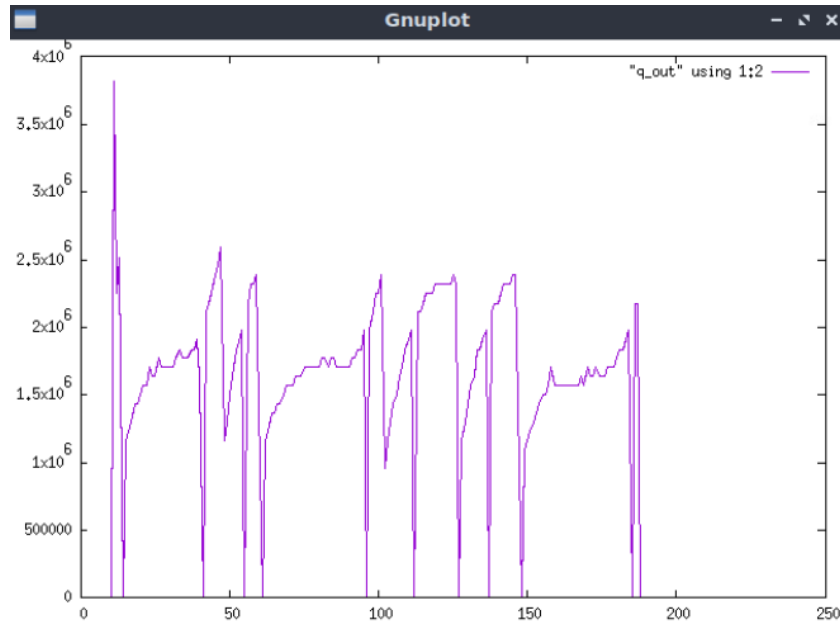
```

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

```

Figure 38. 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 39. Queue occupancy on switch S2's `s2-eth2` interface.

The graph above shows that the queue occupancy peaked around $3.5 \cdot 10^6$ 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:~/RED# plot_iperf.sh out.json && cd results
root@admin-pc:~/RED/results#

```

Figure 40. 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:~/RED/results# xdg-open throughput.pdf
    
```

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

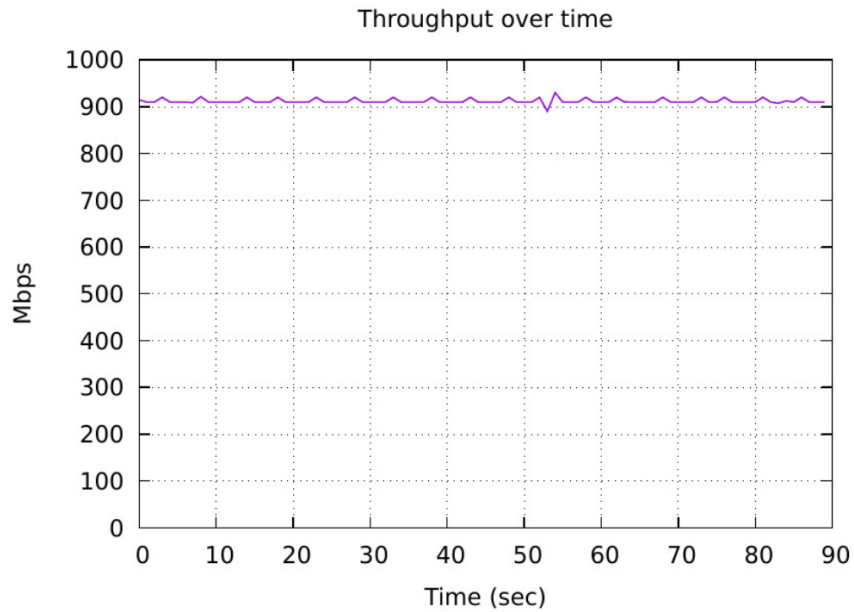


Figure 42. Measured throughput.

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 RED 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:~/10BDP/results# xdg-open RTT.pdf
    
```

Figure 43. Opening the *RTT.pdf* file.

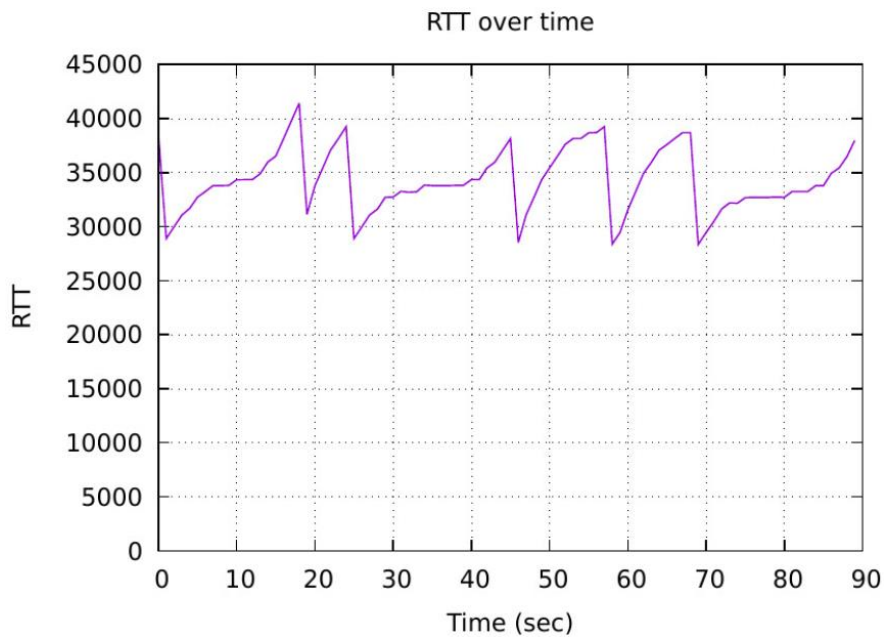


Figure 44. Measured Round-Trip Time.

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=34.3 ms
64 bytes from 10.0.0.3: icmp_seq=73 ttl=64 time=35.3 ms
64 bytes from 10.0.0.3: icmp_seq=74 ttl=64 time=37.3 ms
64 bytes from 10.0.0.3: icmp_seq=75 ttl=64 time=37.1 ms
64 bytes from 10.0.0.3: icmp_seq=76 ttl=64 time=27.5 ms
64 bytes from 10.0.0.3: icmp_seq=77 ttl=64 time=29.5 ms
64 bytes from 10.0.0.3: icmp_seq=78 ttl=64 time=31.9 ms
64 bytes from 10.0.0.3: icmp_seq=79 ttl=64 time=33.6 ms
64 bytes from 10.0.0.3: icmp_seq=80 ttl=64 time=34.9 ms
64 bytes from 10.0.0.3: icmp_seq=81 ttl=64 time=36.4 ms
64 bytes from 10.0.0.3: icmp_seq=82 ttl=64 time=34.3 ms
64 bytes from 10.0.0.3: icmp_seq=83 ttl=64 time=37.6 ms
64 bytes from 10.0.0.3: icmp_seq=84 ttl=64 time=27.8 ms
64 bytes from 10.0.0.3: icmp_seq=85 ttl=64 time=27.9 ms
64 bytes from 10.0.0.3: icmp_seq=86 ttl=64 time=29.7 ms
64 bytes from 10.0.0.3: icmp_seq=87 ttl=64 time=29.10 ms
64 bytes from 10.0.0.3: icmp_seq=88 ttl=64 time=31.3 ms
64 bytes from 10.0.0.3: icmp_seq=89 ttl=64 time=31.7 ms
64 bytes from 10.0.0.3: icmp_seq=90 ttl=64 time=31.6 ms

--- 10.0.0.3 ping statistics ---
90 packets transmitted, 90 received, 0% packet loss, time 229ms
rtt min/avg/max/mdev = 26.833/34.048/38.824/3.311 ms
root@admin-pc:~#
    
```

Figure 45. `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 26.833, 34.048, 38.824, and 3.311 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
```

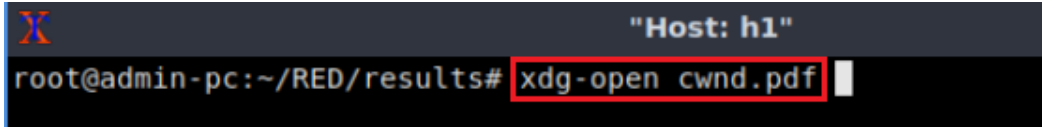


Figure 46. Opening the *cwnd.pdf* file.

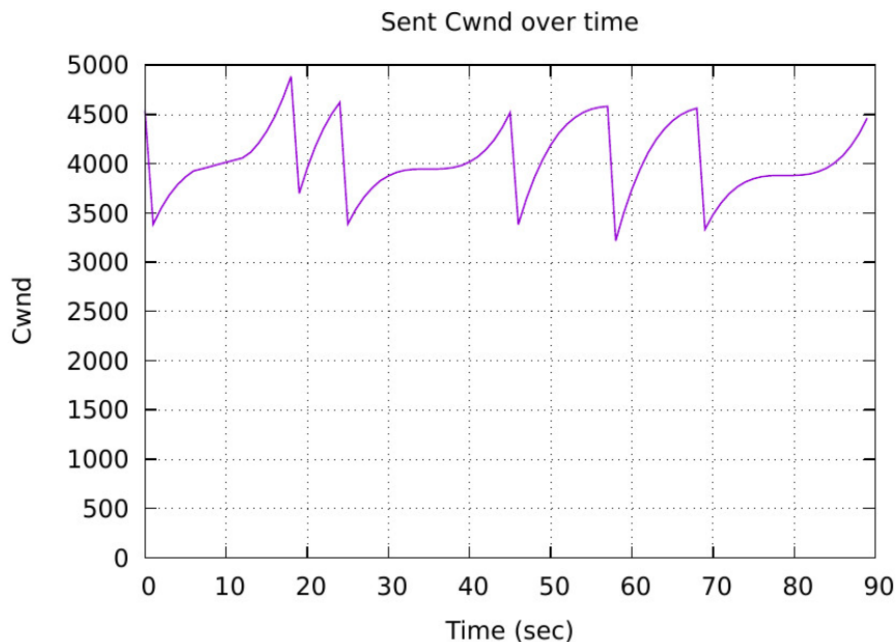


Figure 47. Evolution of the congestion window.

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
```

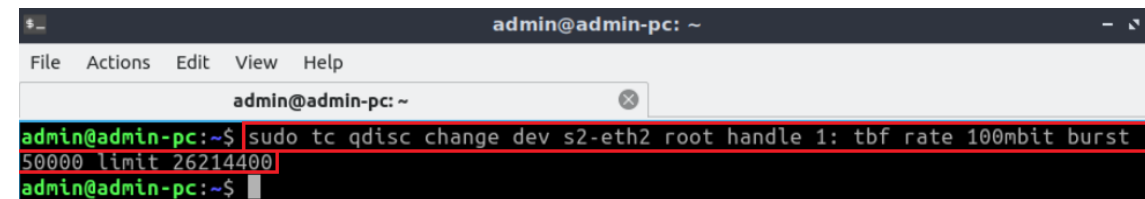


Figure 48. Limiting rate to 100 Mbps and keeping the buffer size to 10·BDP on switch S2's interface.

4.4 Throughput and latency tests

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

```
iperf3 -s
```

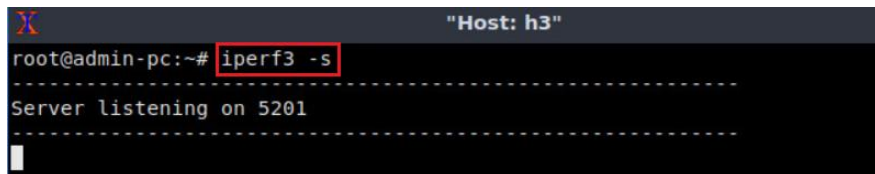


Figure 49. Starting iPerf3 server on host h3.

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
```

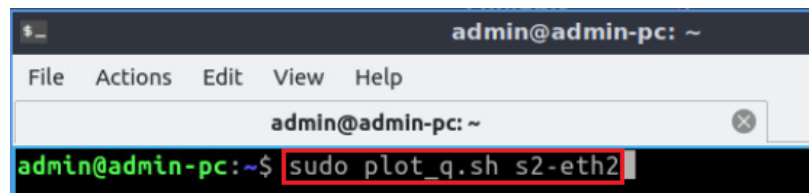


Figure 50. Plotting the queue occupancy on switch S2's `s2-eth2` interface.

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.

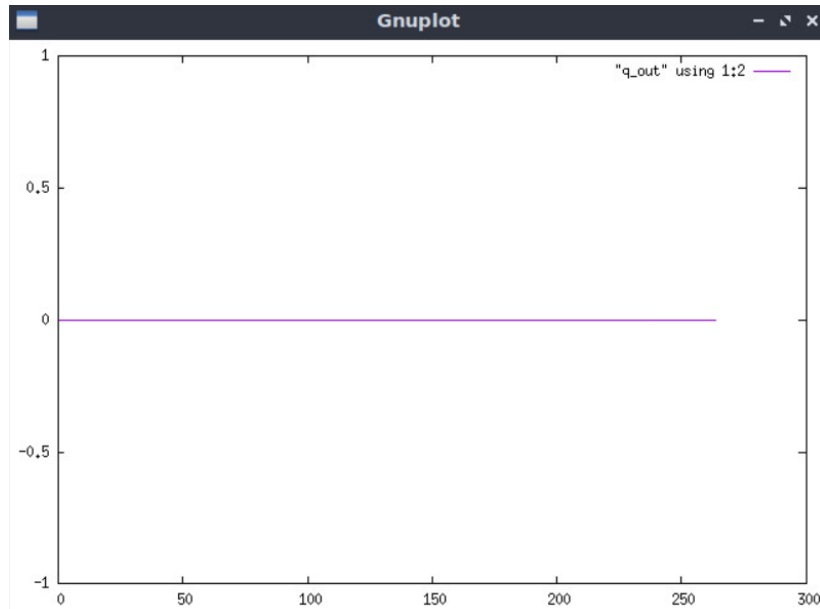


Figure 51. Queue occupancy on switch S2's *s2-eth2* interface.

Step 3. Exit from RED/results directory using the following command:

```
cd ..
```

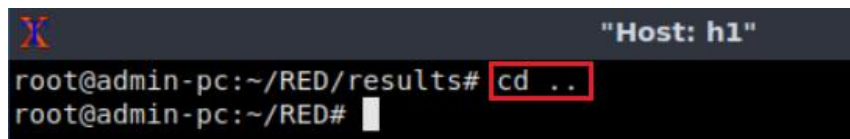


Figure 52. Creating and navigating into directory *1BDP*.

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
```

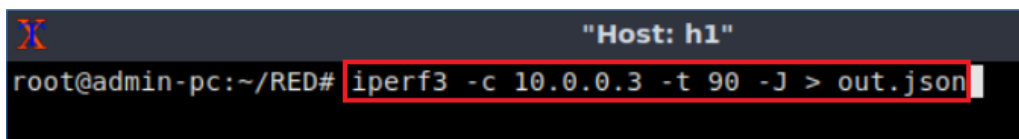


Figure 53. Running iPerf3 client on host h1.

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

```
ping 10.0.0.3 -c 90
```

```

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

```

Figure 54. 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.

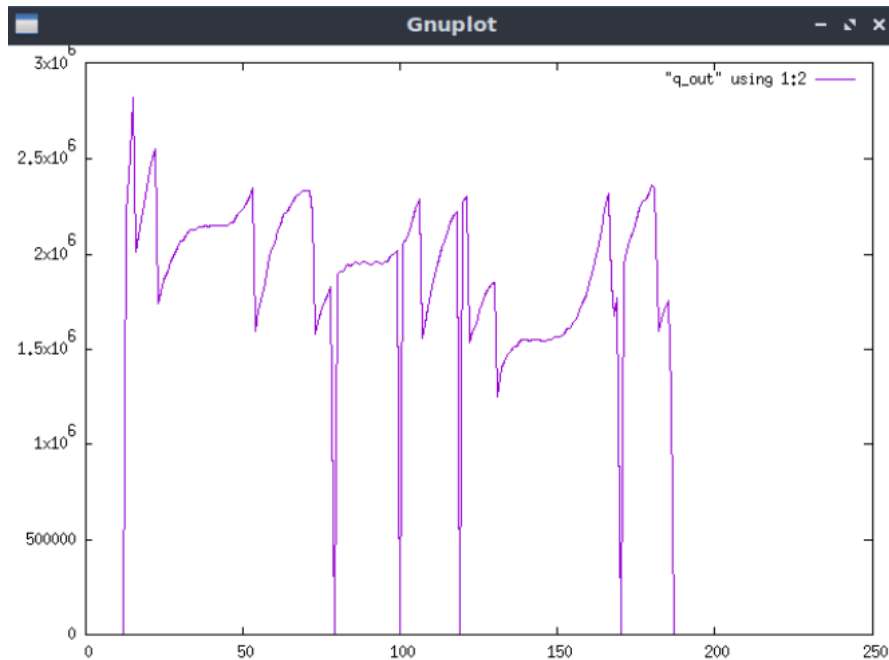


Figure 55. Queue occupancy on switch S2's s2-eth2 interface.

The graph above shows that the queue occupancy peaked over $2.5 \cdot 10^6$, which is around average queue length for a 1 Gbps link. However, in this case we set a 100 Mbps link when RED is configured to operate for 1 Gbps link therefore, the point of operation changed. Consequently, bufferbloat is experienced thus, it is necessary to reconfigure RED parameters in order to mitigate the excessive queue length.

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
```

```

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

```

Figure 56. 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
```



Figure 57. Opening the *throughput.pdf* file.

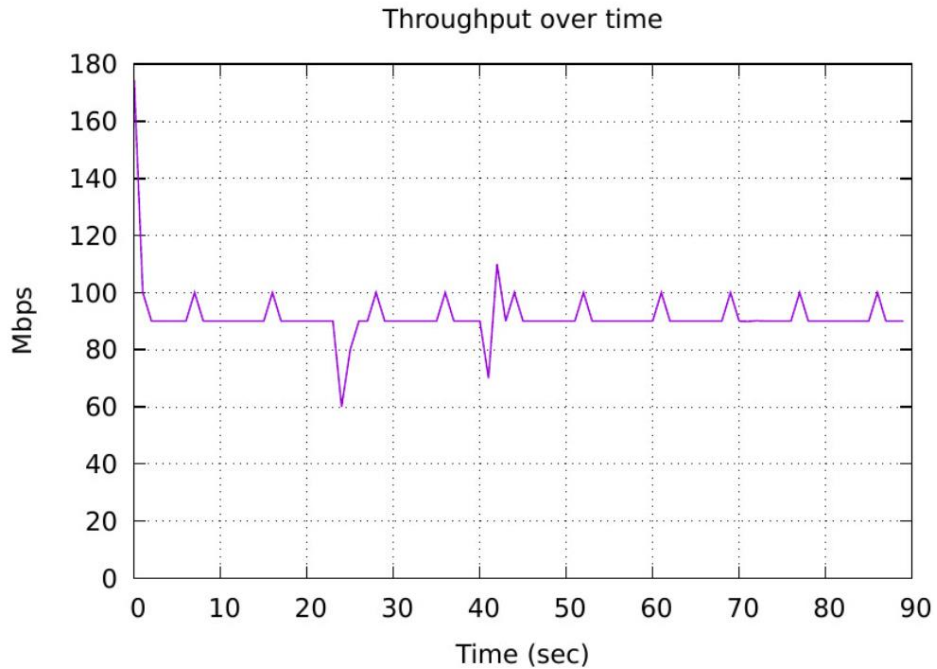


Figure 58. Measured throughput.

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
```

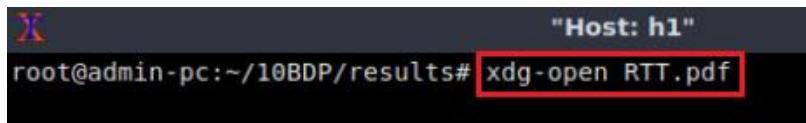


Figure 59. Opening the *RTT.pdf* file.

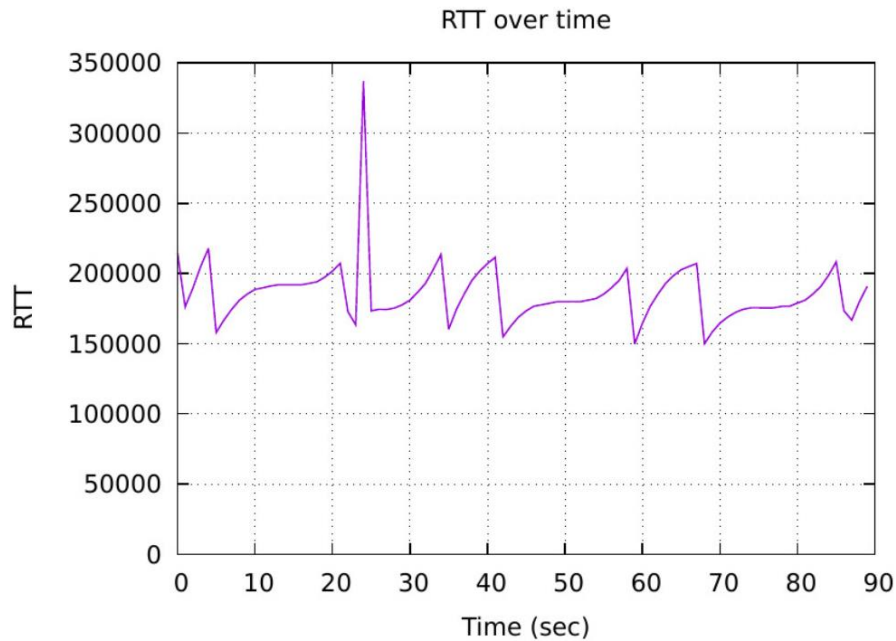


Figure 60. Measured Round-Trip Time.

The graph above shows that the RTT increased from approximately ten times the default latency (20ms). The output above shows that there is a bufferbloat problem as the average latency is significantly greater. Since RED is configured to operate on a 1 Gbps link, for this test the point of operation changed therefore, unnecessary delay is observed.

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=169 ms
64 bytes from 10.0.0.3: icmp_seq=73 ttl=64 time=171 ms
64 bytes from 10.0.0.3: icmp_seq=74 ttl=64 time=174 ms
64 bytes from 10.0.0.3: icmp_seq=75 ttl=64 time=174 ms
64 bytes from 10.0.0.3: icmp_seq=76 ttl=64 time=175 ms
64 bytes from 10.0.0.3: icmp_seq=77 ttl=64 time=174 ms
64 bytes from 10.0.0.3: icmp_seq=78 ttl=64 time=174 ms
64 bytes from 10.0.0.3: icmp_seq=79 ttl=64 time=176 ms
64 bytes from 10.0.0.3: icmp_seq=80 ttl=64 time=177 ms
64 bytes from 10.0.0.3: icmp_seq=81 ttl=64 time=178 ms
64 bytes from 10.0.0.3: icmp_seq=82 ttl=64 time=180 ms
64 bytes from 10.0.0.3: icmp_seq=83 ttl=64 time=185 ms
64 bytes from 10.0.0.3: icmp_seq=84 ttl=64 time=191 ms
64 bytes from 10.0.0.3: icmp_seq=85 ttl=64 time=198 ms
64 bytes from 10.0.0.3: icmp_seq=86 ttl=64 time=208 ms
64 bytes from 10.0.0.3: icmp_seq=87 ttl=64 time=160 ms
64 bytes from 10.0.0.3: icmp_seq=88 ttl=64 time=166 ms
64 bytes from 10.0.0.3: icmp_seq=89 ttl=64 time=180 ms
64 bytes from 10.0.0.3: icmp_seq=90 ttl=64 time=192 ms

--- 10.0.0.3 ping statistics ---
90 packets transmitted, 90 received, 0% packet loss, time 183ms
rtt min/avg/max/mdev = 148.914/186.175/468.728/33.481 ms
root@admin-pc:~#
    
```

Figure 61. `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 148.914, 186.175, 468.728 and 33.481 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 congestion window (*cwnd.pdf*) file using the command below.

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

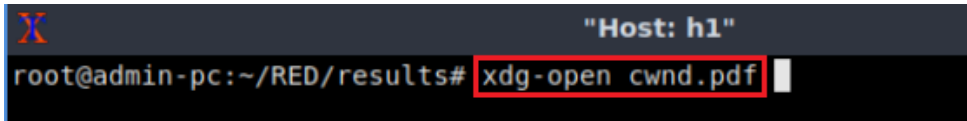


Figure 62. Opening the *cwnd.pdf* file.

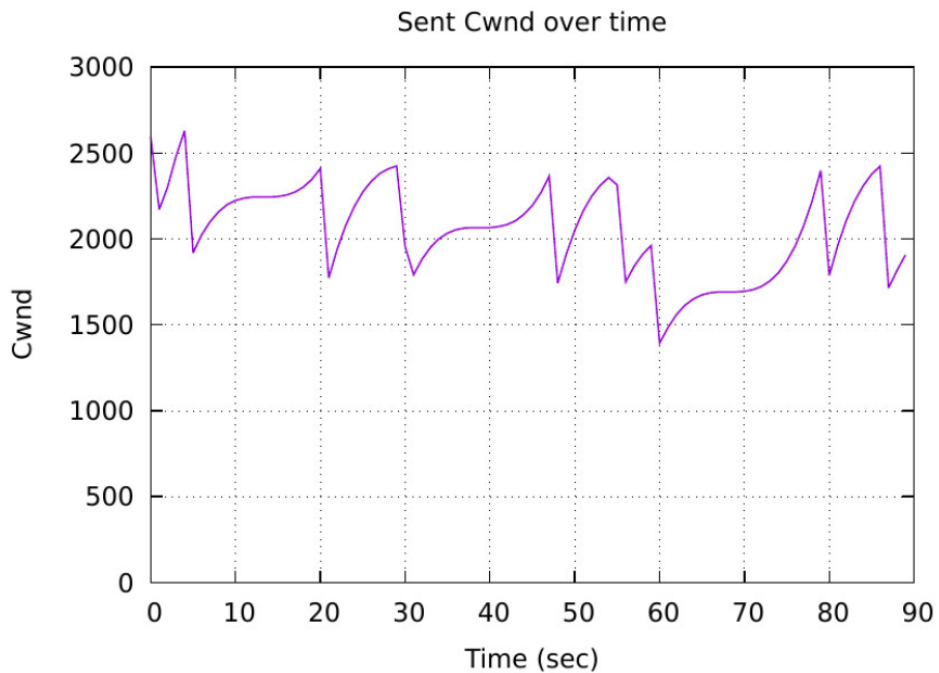


Figure 63. Evolution of the congestion window.

The graph above shows the evolution of the congestion window which peaked around 2.5 Mbytes.

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

This concludes Lab 16. Stop the emulation and then exit out of MiniEdit.

References

1. S. Floyd, V. Jacobson, "Random early detection gateways for congestion avoidance". *IEEE/ACM Transactions on networking*, 1993.

2. S. Floyd, R. Gummadi, S. Shenker. "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management." 2001.
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.