NETWORK TOOLS AND PROTOCOLS

Lab 14: Router’s Bufferbloat

Document Version: **07-07-2019**

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

Contents

Overview 3
Objectives 3
Lab settings 3
Lab roadmap 3
1 Introduction to bufferbloat 3
1.1 Packet delays 4
1.2 Bufferbloat 4
2 Lab topology 6
2.1 Starting host h1, host h2, and host h3 7
2.2 Emulating high-latency WAN 8
2.4 Testing connection 9
3 Testing throughput on a network with a small buffer-size switch 10
3.1 Setting switch S1's buffer size to 100·MTU 10
3.2 Bandwidth-delay product (BDP) and hosts' buffer size 11
3.3 Throughput test 13
4 Testing throughput on a network with a 1·BDP buffer-size switch 13
4.1 Setting switch S1's buffer size to 1·BDP 13
4.2 Throughput and latency tests 14
5 Testing throughput on a network with a large buffer-size switch 19
5.1 Setting switch S1's buffer size to 10·BDP 19
5.2 Throughput and latency tests 20
References 25

Overview

This lab discusses bufferbloat, a condition that occurs when a router or network device buffers too much data, leading to excessive delays. The lab describes the steps to conduct throughput tests on switched networks with different buffer sizes. Note that as the buffering process is similar in routers and switches, both terms are used interchangeably in this lab.

Objectives

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

1. Identify and describe the components of end-to-end delay.
2. Understand the buffering process in a router.
3. Explain the concept of bufferbloat.
4. Visualize queue occupancy in a router.
5. Analyze end-to-end delay and describe how queueing delay affects end-to-end delay on networks with large routers' buffer size.
6. Modify routers' buffer size to solve the bufferbloat problem.

Lab settings

The information in Table 1 provides the credentials of the machine containing Mininet.

Table 1**.** Credentials to access Client1 machine.

Device	Account	Password
Client1	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction to bufferbloat.
2. Section 2: Lab topology.
3. Section 3: Testing throughput on a network with a small buffer-size switch.
4. Section 4: Testing throughput on a network with a 1·BDP buffer-size switch.
5. Section 5: Testing throughput on a network with a large buffer-size switch.

1 Introduction to bufferbloat

1.1 Packet delays

As a packet travels from a sender to a receiver, it experiences several types of delays at each node (router / switch) along the path. The most important of these delays are the processing delay, queuing delay, transmission delay, and propagation delay (see Figure 1)[1].

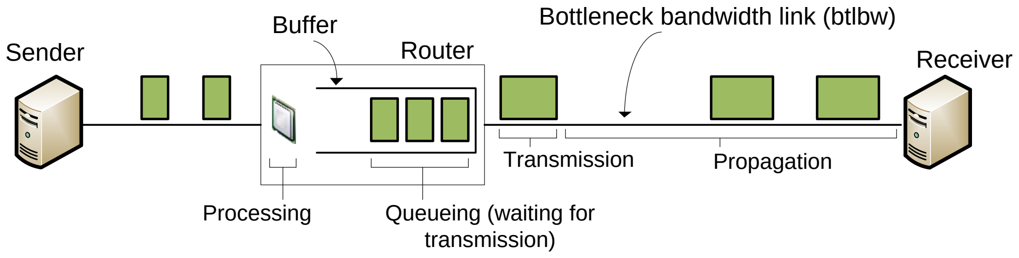

Buffer
Bottleneck bandwidth link (btlbw)
Sender
Router
Receiver
Transmission
Propagation
Processing
Queueing (waiting for transmission)

Figure 1. Delay components: processing, queueing, transmission, and propagation delays.

- Processing delay: The time required to examine the packet's header and determine where to direct the packet. For high-speed routers, this delay is on the order of microseconds or less.

- Transmission delay: The time required to put the bits on the *wire*. It is given by the packet size (in bits) divided by the bandwidth of the link (in bps). For example, for a 10 Gbps and 1,500-byte packet (12,000 bits), the transmission time is T = 12,000 / $10x10^9$ = 0.0012 milliseconds or 1.2 microseconds.

- Queueing delay: The time a packet waits for transmission onto the link. The length of the queuing delay of a packet depends on the number of earlier-arriving packets that are queued and waiting for transmission onto the link. Queuing delays can be on the order of microseconds to milliseconds.

- Propagation delay: Once a bit is placed into the link, it needs to propagate to the other end of the link. The time required to propagate across the link is the propagation delay. In local area networks (LANs) and datacenter environments, this delay is small (microseconds to few milliseconds); however, in Wide Area Networks (WANs) / long-distance connections, the propagation delay can be on the order of hundreds of milliseconds.

1.2 Bufferbloat

In modern networks composed of high-speed routers and switches, the processing and transmission delays may be negligible. The propagation delay can be considered as a constant (i.e., it has a fixed value). Finally, the dynamics of the queues in routers results in varying queueing delays. Ideally, this delay should be minimized.

An important consideration that affects the queuing delay is the router's buffer size. While there is no consensus on how large the buffer should be, the rule of thumb has been that the amount of buffering (in bits) in a router's port should equal the average Round-Trip Time (RTT) (in seconds) multiplied by the capacity C (in bits per seconds) of the port[2, 3]:

$$\text{Router's buffer size} = C \cdot \text{RTT [bits]}$$

A large-enough router's buffer size is essential for networks transporting big flows, as it absorbs transitory packet bursts and prevents losses. However, if a buffer size is excessively large, queues can be formed and substantial queueing delay be observed. This high latency produced by excess buffering of packets is referred to as bufferbloat.

The bufferbloat problem is caused by routers with large buffer size and end devices running TCP congestion control algorithms that constantly probe for additional bandwidth[4]. Consider Figure 2, where RT_{prop} refers to the end-to-end propagation delay from sender to receiver and then back (round-trip), and BDP refers to the bandwidth-delay product given by the product of the capacity of the bottleneck link along the path and RT_{prop}. RT_{prop} is a constant that depends on the physical distance between end devices. In the application limited region, the throughput increases as the amount of data generated by the application layer increases, while the RTT remains constant. The pipeline between sender and receiver becomes full when the inflight number of bits is equal to BDP, at the edge of the bandwidth limited region. Note that traditional TCP congestion control (e.g., Reno, Cubic, HTCP) will continue to increase the sending rate (inflight data) beyond the optimal operating point, as they probe for more bandwidth. This process is known as TCP additive increase rule. Since no packet loss is noted in the bandwidth limited region despite the increasing TCP rate (which is absorbed by the router's buffer), TCP keeps increasing the sending rate / inflight data, until eventually the router's buffer is full and a packet is drop (the amount of bits in the network is equal to BDP plus the buffer size of the router). Beyond the application limited region, the increase in queueing delay causes the bufferbloat problem.

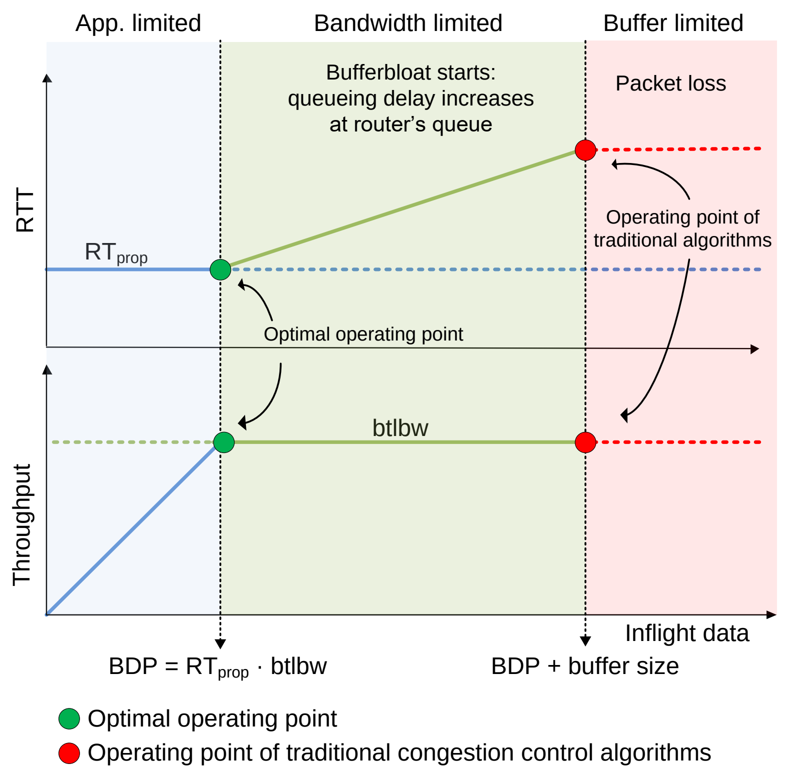


Figure 2. Throughput and RTT as a function of inflight data[5].

In this lab, the reader will conduct experiments and measure the throughput and RTT under different network conditions. By modifying a router's buffer size, the bufferbloat problem will be observed.

2 Lab topology

Let's get started with creating a simple Mininet topology using MiniEdit. The topology uses 10.0.0.0/8 which is the default network assigned by Mininet.

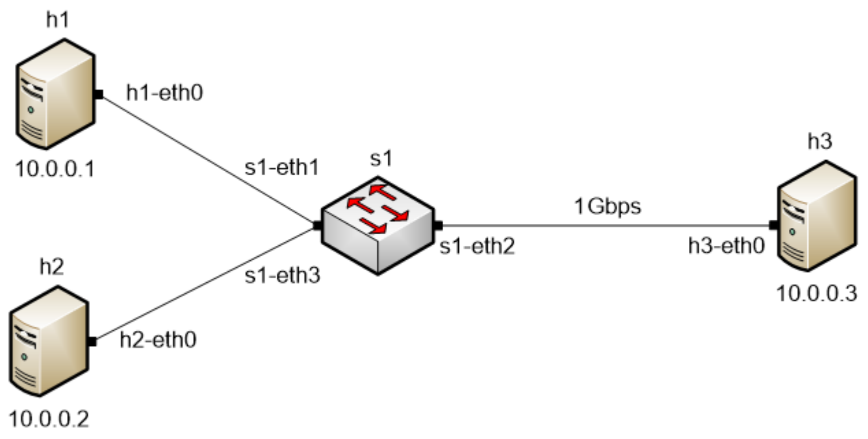


Figure 3. Lab topology.

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

Step 1. A shortcut to MiniEdit is located on the machine's Desktop. Start MiniEdit by clicking on MiniEdit's shortcut. When prompted for a password, type `password`.

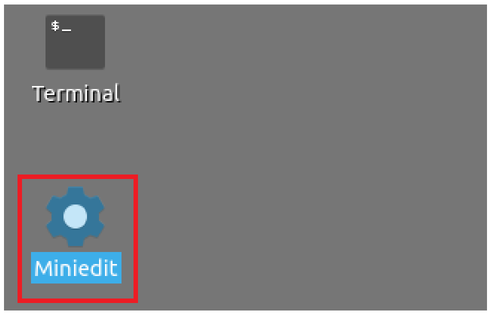


Figure 4. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 14.mn* topology file and click on *Open*.

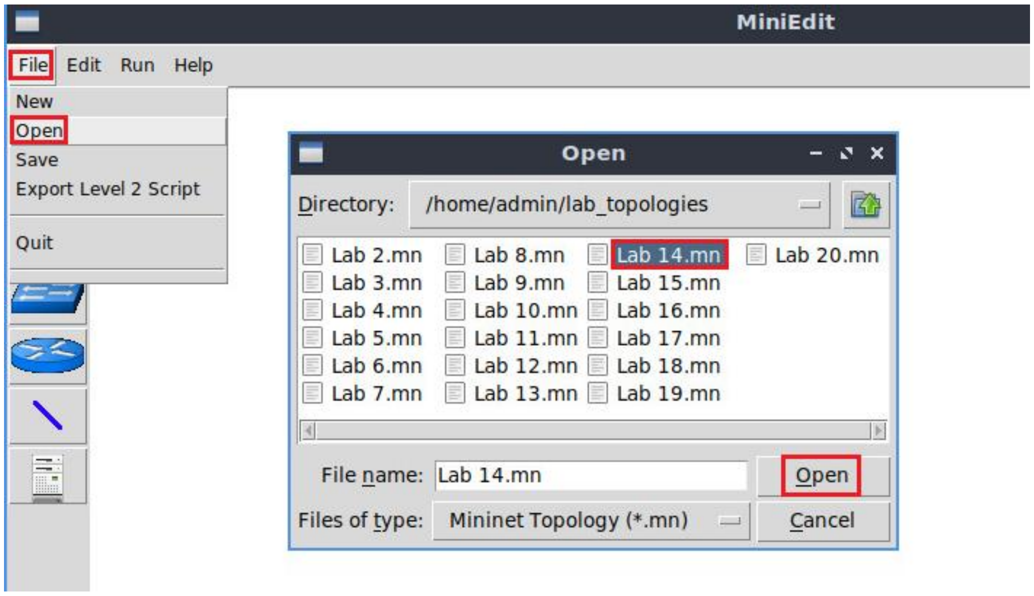


Figure 5. MiniEdit's *Open* dialog.

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

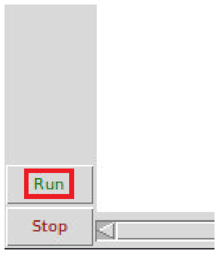


Figure 6. Running the emulation.

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

2.1 Starting host h1, host h2, 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.

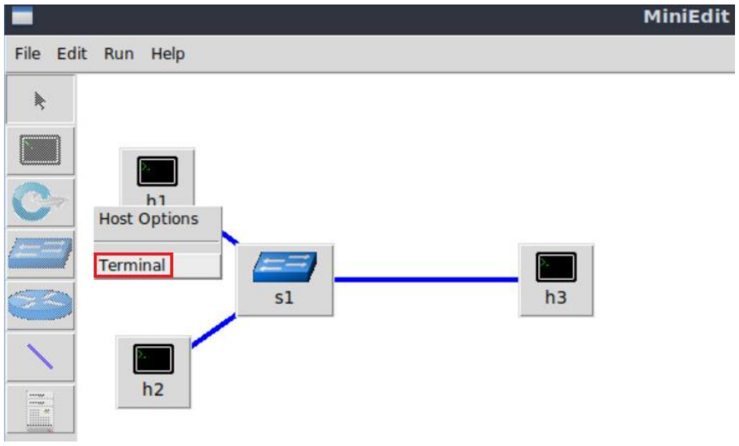
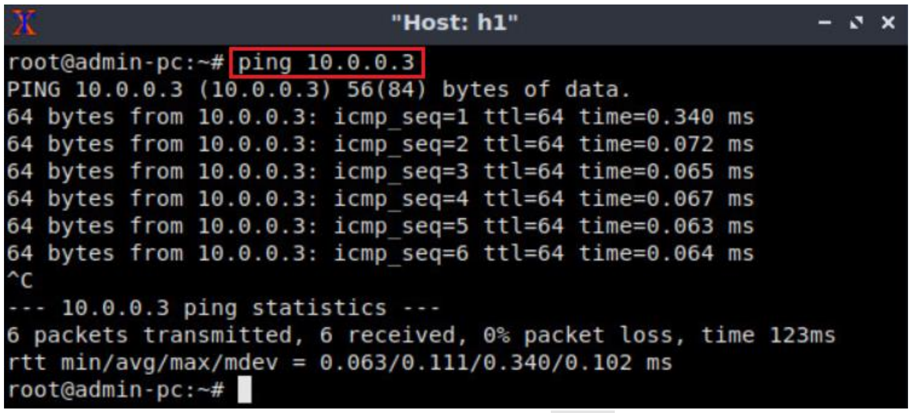


Figure 7. Opening a terminal on host h1.

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

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



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

Figure 8. Connectivity test using `ping` command.

2.2 Emulating high-latency WAN

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

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

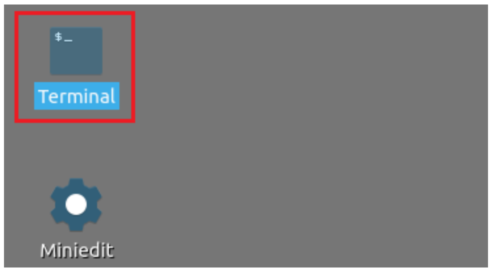
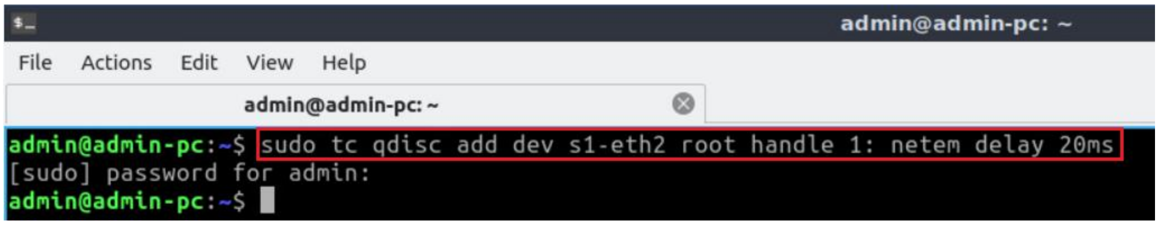


Figure 9. Shortcut to open a Linux terminal.

The Linux terminal is a program that opens a window and permits you to interact with a command-line interface (CLI). A CLI is a program that takes commands from the keyboard and sends them to the operating system to perform.

Step 2. In the terminal, type the command below. When prompted for a password, type `password` and hit *Enter*. This command introduces 10ms delay to switch S1's *s1-eth2* interface.

```
sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 20ms
```



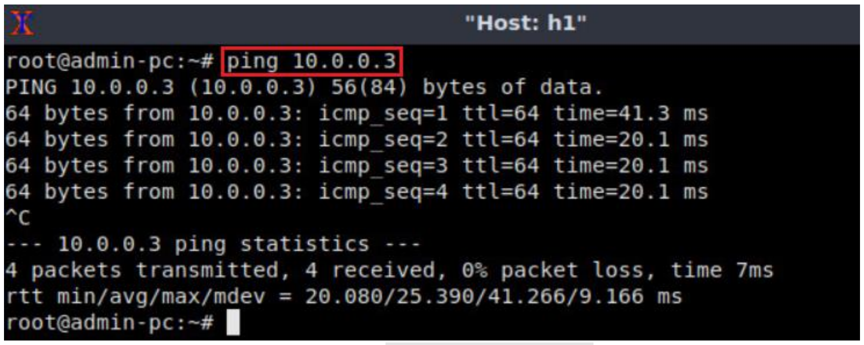
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 10. Adding delay of 10ms to switch S1's *s1-eth2* 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.

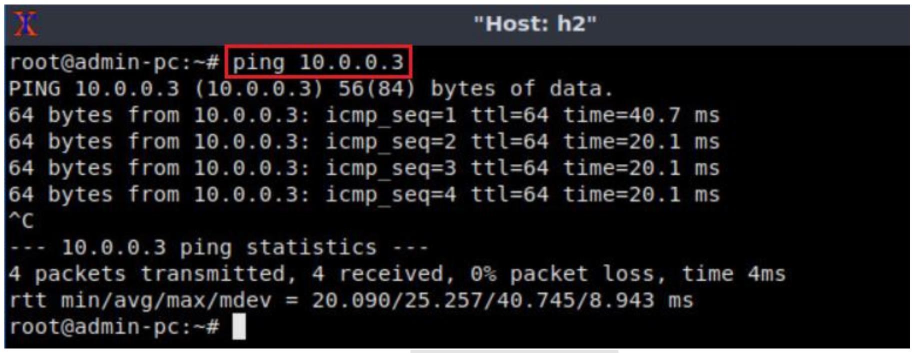


```
root@admin-pc:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=41.3 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=20.1 ms
^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 11. Output of `ping 10.0.0.3` command.

The result above indicates that all four packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 20.080, 25.390, 41.266, and 9.166 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

Step 2. On the terminal of host h2, type `ping 10.0.0.3`. The ping output in this test should be relatively similar to the results of the test initiated by host h1 in Step 1. To stop the test, press `Ctrl+c`.



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

Figure 12. Output of `ping 10.0.0.3` command.

The result above indicates that all four packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 20.090, 25.257, 40.745, and 8.943 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

3 Testing throughput on a network with a small buffer-size switch

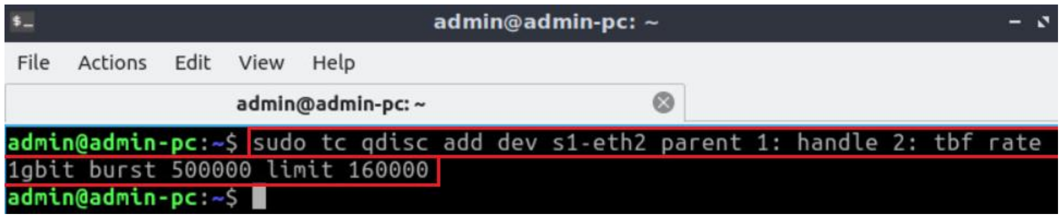
In this section, you are going to change the switch S1's buffer size to 100·MTU and emulate a 1 Gbps Wide Area Network (*WAN*) using the Token Bucket Filter (`tbf`). Then, you will test the throughput between host h1 and host h3. In this section, the MTU is 1600 bytes, thus the `tbf` limit value will be set to 100 · MTU = 160,000 bytes.

3.1 Setting switch S1's buffer size to 100·MTU

Step 1. Apply `tbf` rate limiting rule on switch S1's *s1-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`: 160,000

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 1gbit burst 500000
limit 160000
```



```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate
1gbit burst 500000 limit 160000
admin@admin-pc:~$
```

Figure 13. Limiting rate to 1 Gbps and setting the buffer size to 100·MTU on switch S1's interface.

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

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

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

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

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

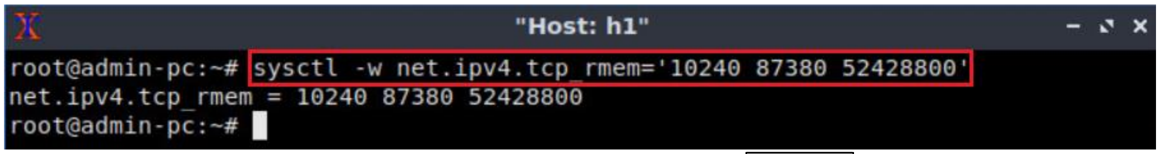
$$1 \text{ Mbyte} = 1024^2 \text{ bytes}$$

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

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·BDP. To ensure that the bottleneck is not the hosts' buffers, we configure the buffers to 10·BDP (26,214,400).

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 (2·10·BDP) as Linux only allocates half of the assigned value.

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



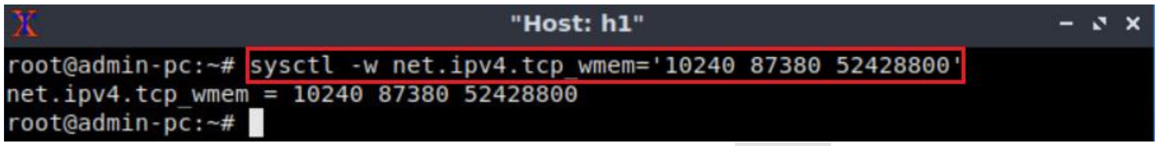
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 14. Receive window change in `sysctl`.

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'
```

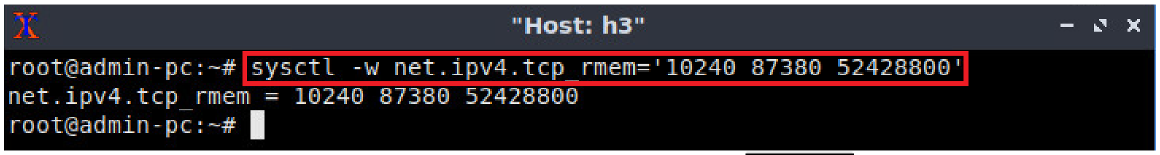


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 15. Send window change in `sysctl`.

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_wmem='10240 87380 52428800'
```

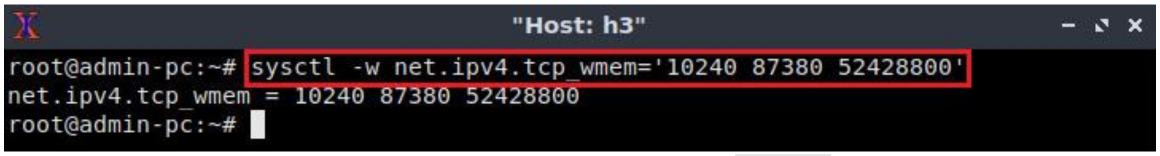


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 16. 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:~#
```

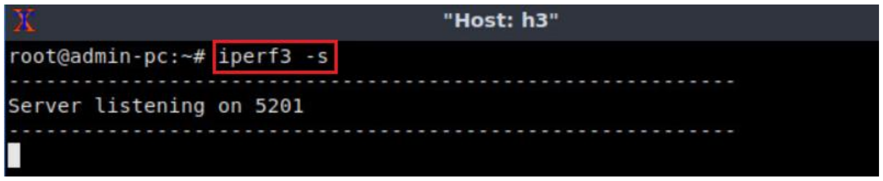
Figure 17. Send window change in `sysctl`.

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.

3.3 Throughput test

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

```
iperf3 -s
```

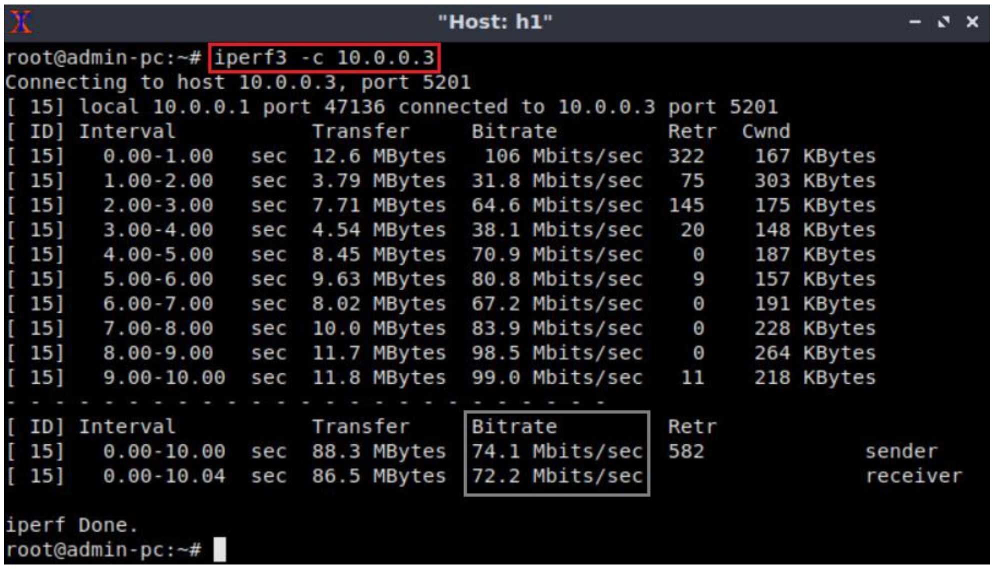


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 18. Starting iPerf3 server on host h3.

Step 2. Type the following iPerf3 command in host h1's terminal.

```
iperf3 -c 10.0.0.3
```



```
root@admin-pc:~# iperf3 -c 10.0.0.3
Connecting to host 10.0.0.3, port 5201
[ 15] local 10.0.0.1 port 47136 connected to 10.0.0.3 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec  12.6 MBytes   106 Mbits/sec  322    167 KBytes
[ 15]   1.00-2.00   sec  3.79 MBytes  31.8 Mbits/sec   75    303 KBytes
[ 15]   2.00-3.00   sec  7.71 MBytes  64.6 Mbits/sec  145    175 KBytes
[ 15]   3.00-4.00   sec  4.54 MBytes  38.1 Mbits/sec   20    148 KBytes
[ 15]   4.00-5.00   sec  8.45 MBytes  70.9 Mbits/sec    0    187 KBytes
[ 15]   5.00-6.00   sec  9.63 MBytes  80.8 Mbits/sec    9    157 KBytes
[ 15]   6.00-7.00   sec  8.02 MBytes  67.2 Mbits/sec    0    191 KBytes
[ 15]   7.00-8.00   sec  10.0 MBytes  83.9 Mbits/sec    0    228 KBytes
[ 15]   8.00-9.00   sec  11.7 MBytes  98.5 Mbits/sec    0    264 KBytes
[ 15]   9.00-10.00  sec  11.8 MBytes  99.0 Mbits/sec   11    218 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  88.3 MBytes  74.1 Mbits/sec  582             sender
[ 15]   0.00-10.04  sec  86.5 MBytes  72.2 Mbits/sec                  receiver

iperf Done.
root@admin-pc:~#
```

Figure 19. Running iPerf3 client on host h1.

The figure above shows the iPerf3 test output report. The average achieved throughput is 74.1 Mbps (sender) and 72.2 Mbps (receiver), and the number of retransmissions is 582. Note that the maximum throughput (1 Gbps) was not achieved. This is due to having a small buffer on the switch (100 · MTU).

4 Testing throughput on a network with a 1·BDP buffer-size switch

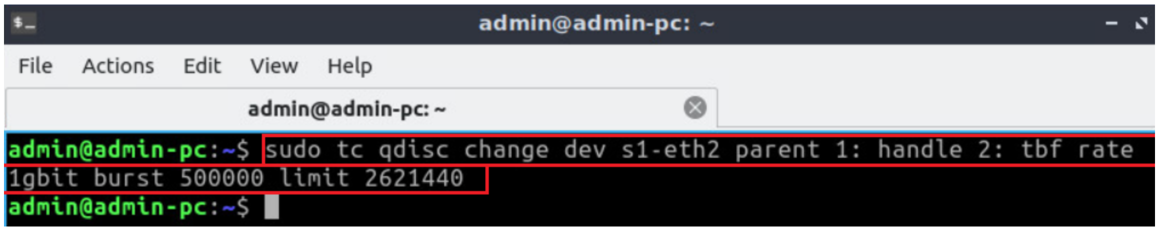
In this section, you are going to change the switch S1's buffer size to 1·BDP and emulate a 1 Gbps Wide Area Network (*WAN*) using the Token Bucket Filter (`tbf`). Then, you will test the throughput between host h1 and host h3. The BDP is 2,621,440 bytes, thus the `tbf` limit value will be set to 2,621,440.

4.1 Setting switch S1's buffer size to 1·BDP

Step 1. Apply `tbf` rate limiting rule on switch S1's *s1-eth2* interface. In the client's terminal, type the command below. When prompted for a password, type `password` and hit *Enter*.

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

```
sudo tc qdisc change dev s1-eth2 parent 1: handle 2: tbf rate 1gbit burst 500000
limit 2621440
```



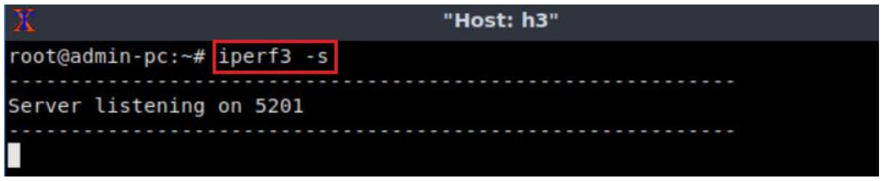
```
admin@admin-pc:~$ sudo tc qdisc change dev s1-eth2 parent 1: handle 2: tbf rate
1gbit burst 500000 limit 2621440
admin@admin-pc:~$
```

Figure 20. Limiting rate to 1 Gbps and setting the buffer size to 1·BDP on switch S1's interface.

4.2 Throughput and latency tests

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

```
iperf3 -s
```

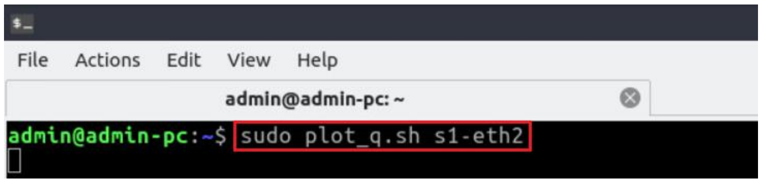


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 21. 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 s1-eth2
```



```
admin@admin-pc:~$ sudo plot_q.sh s1-eth2
```

Figure 22. Plotting the queue occupancy on switch S1's *s1-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 *s1-eth2* interface on switch S1, the queue occupancy is constantly 0.

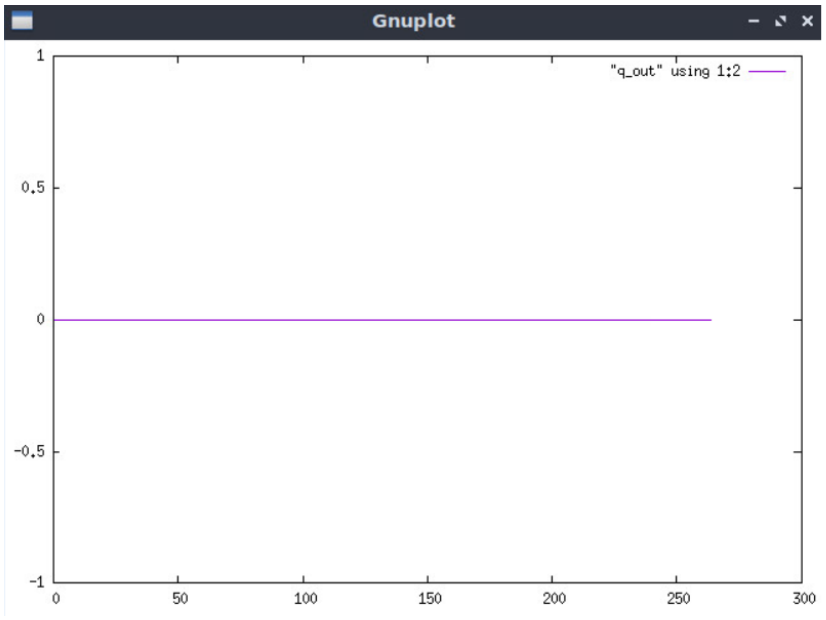
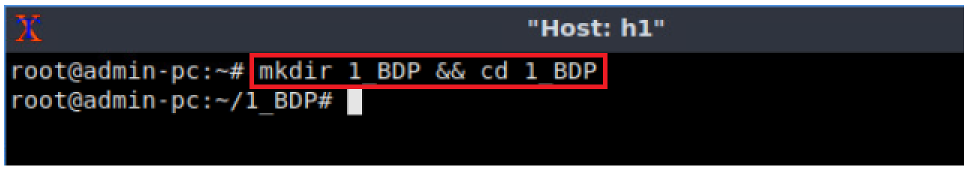


Figure 23. Queue occupancy on switch S1's *s1-eth2* interface.

Step 3. In host h1, create a directory called *1_BDP* and navigate into it using the following command:

```
mkdir 1_BDP && cd 1_BDP
```

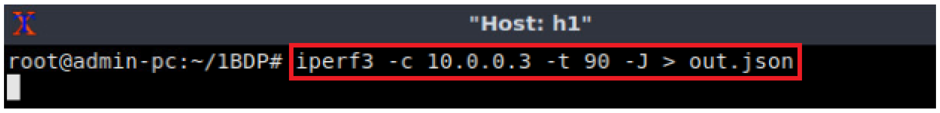


```
root@admin-pc:~# mkdir 1_BDP && cd 1_BDP
root@admin-pc:~/1_BDP#
```

Figure 24. Creating and navigating into directory *1_BDP*.

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
```

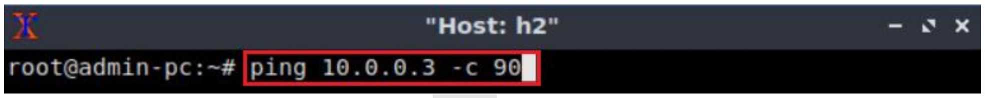


```
root@admin-pc:~/1BDP# iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

Figure 25. 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
```



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

Figure 26. 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.

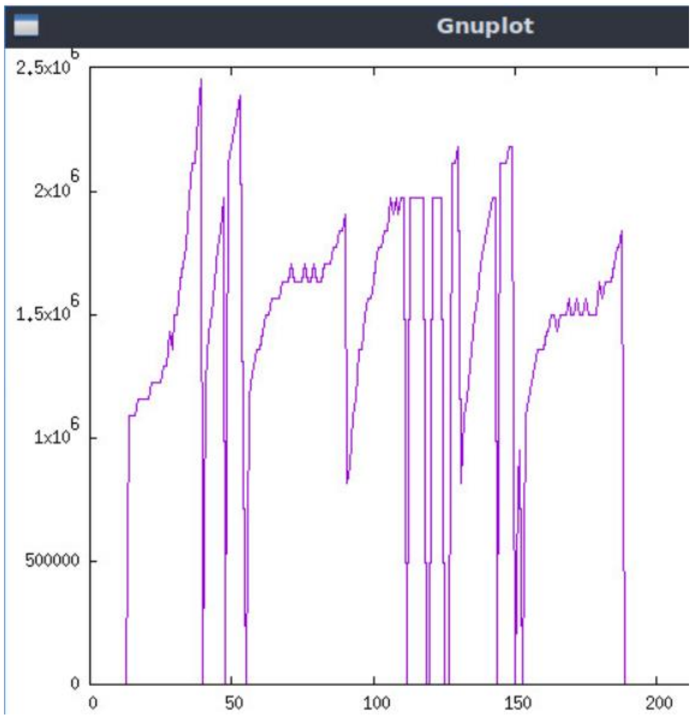


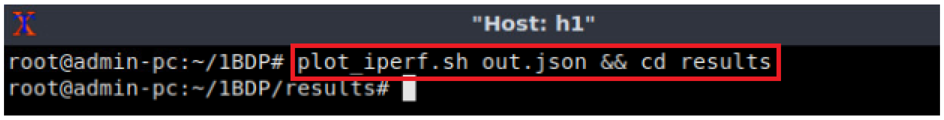
Figure 27. Queue occupancy on switch S1's *s1-eth2* interface.

The graph above shows that the queue occupancy peaked at $2.5 \cdot 10^6$, 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
```

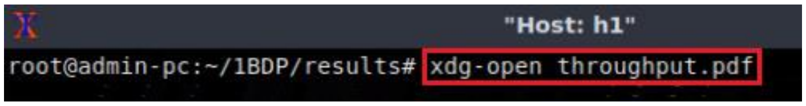


```
"Host: h1"
root@admin-pc:~/1BDP# plot_iperf.sh out.json && cd results
root@admin-pc:~/1BDP/results#
```

Figure 28. 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:~/1BDP/results# xdg-open throughput.pdf
```

Figure 29. Opening the *throughput.pdf* file.

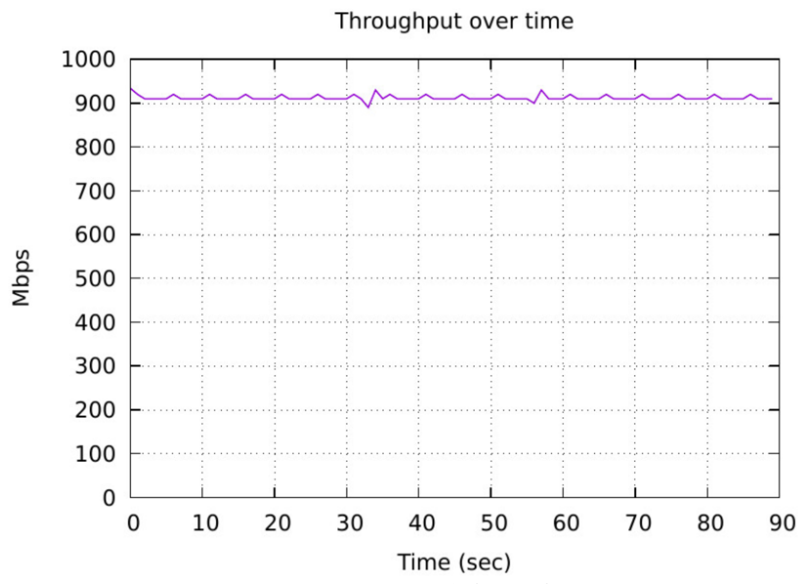
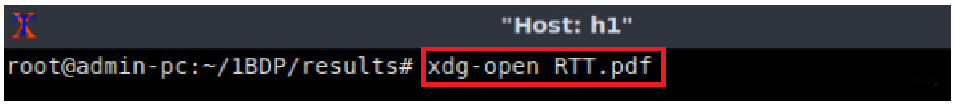


Figure 30. Measured throughput.

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 1BDP.

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:~/1BDP/results# xdg-open RTT.pdf
```

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

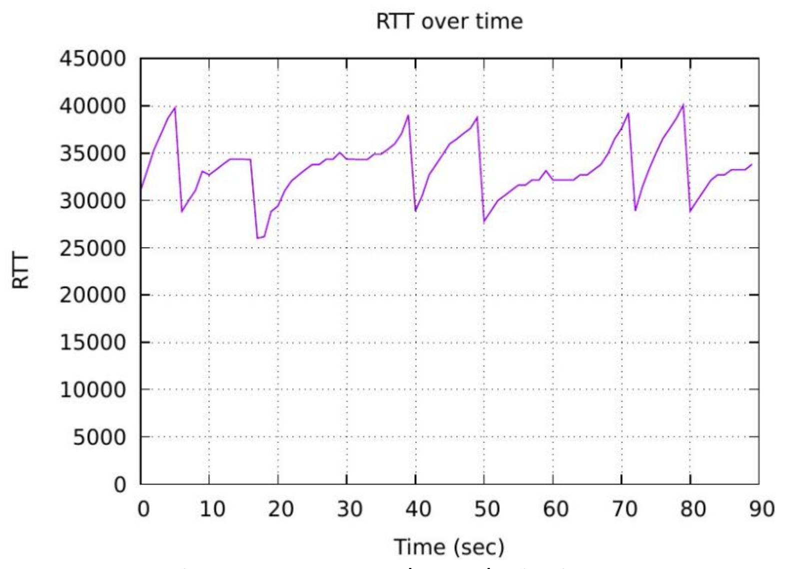
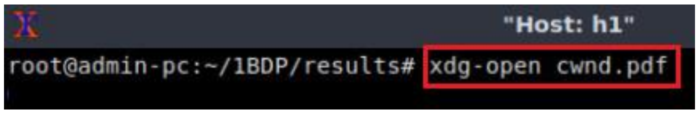


Figure 32. Measured round-trip time.

The graph above shows that the RTT was between 25000 microseconds (25ms) and 40000 microseconds (40ms). The output shows that there is no bufferbloat problem as the average latency is slightly greater than the configured delay (20ms).

Step 11. Close the *RTT.pdf* window then open the congestion window (cwnd) file using the command below.

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



```
"Host: h1"
root@admin-pc:~/1BDP/results# xdg-open cwnd.pdf
```

Figure 33. Opening the *cwnd.pdf* file.

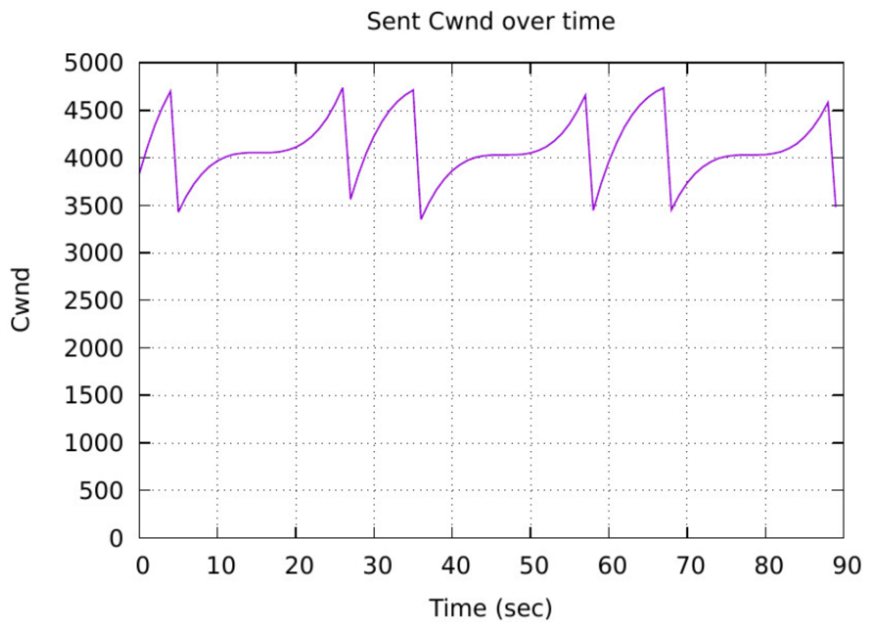
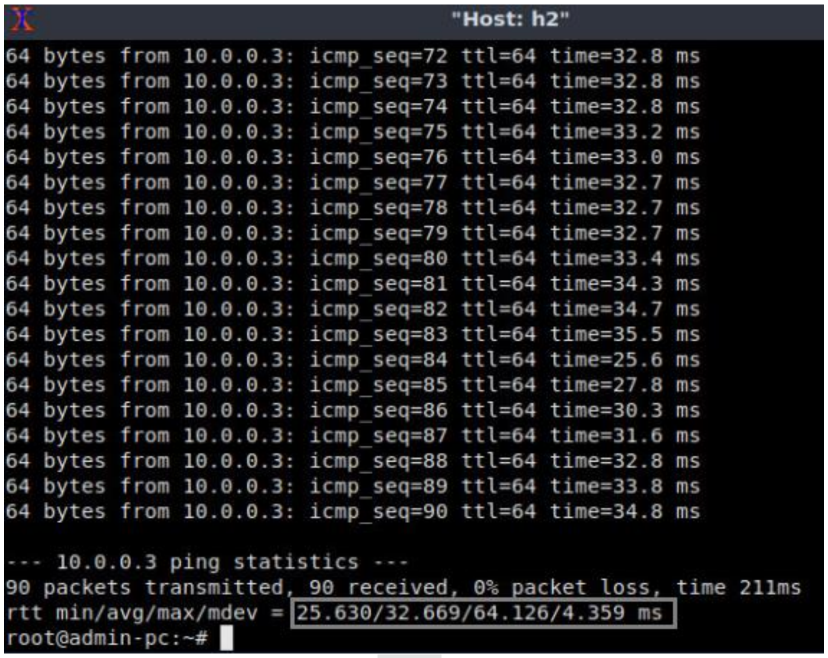


Figure 34. Congestion window evolution.

The graph above shows the evolution of the congestion window which peaked at 4.5 Mbytes. In the next test, we see how buffer size on the switch affect the congestion window evolution.

Step 12. Close the *cwnd.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=32.8 ms
64 bytes from 10.0.0.3: icmp_seq=73 ttl=64 time=32.8 ms
64 bytes from 10.0.0.3: icmp_seq=74 ttl=64 time=32.8 ms
64 bytes from 10.0.0.3: icmp_seq=75 ttl=64 time=33.2 ms
64 bytes from 10.0.0.3: icmp_seq=76 ttl=64 time=33.0 ms
64 bytes from 10.0.0.3: icmp_seq=77 ttl=64 time=32.7 ms
64 bytes from 10.0.0.3: icmp_seq=78 ttl=64 time=32.7 ms
64 bytes from 10.0.0.3: icmp_seq=79 ttl=64 time=32.7 ms
64 bytes from 10.0.0.3: icmp_seq=80 ttl=64 time=33.4 ms
64 bytes from 10.0.0.3: icmp_seq=81 ttl=64 time=34.3 ms
64 bytes from 10.0.0.3: icmp_seq=82 ttl=64 time=34.7 ms
64 bytes from 10.0.0.3: icmp_seq=83 ttl=64 time=35.5 ms
64 bytes from 10.0.0.3: icmp_seq=84 ttl=64 time=25.6 ms
64 bytes from 10.0.0.3: icmp_seq=85 ttl=64 time=27.8 ms
64 bytes from 10.0.0.3: icmp_seq=86 ttl=64 time=30.3 ms
64 bytes from 10.0.0.3: icmp_seq=87 ttl=64 time=31.6 ms
64 bytes from 10.0.0.3: icmp_seq=88 ttl=64 time=32.8 ms
64 bytes from 10.0.0.3: icmp_seq=89 ttl=64 time=33.8 ms
64 bytes from 10.0.0.3: icmp_seq=90 ttl=64 time=34.8 ms

--- 10.0.0.3 ping statistics ---
90 packets transmitted, 90 received, 0% packet loss, time 211ms
rtt min/avg/max/mdev = 25.630/32.669/64.126/4.359 ms
root@admin-pc:~#
```

Figure 35. `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 25.630, 32.669, 64.126, and 4.359 milliseconds, respectively. The output also verifies that there is no bufferbloat problem as the average latency (32.669) is slightly greater than the configured delay (20ms).

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

5 Testing throughput on a network with a large buffer-size switch

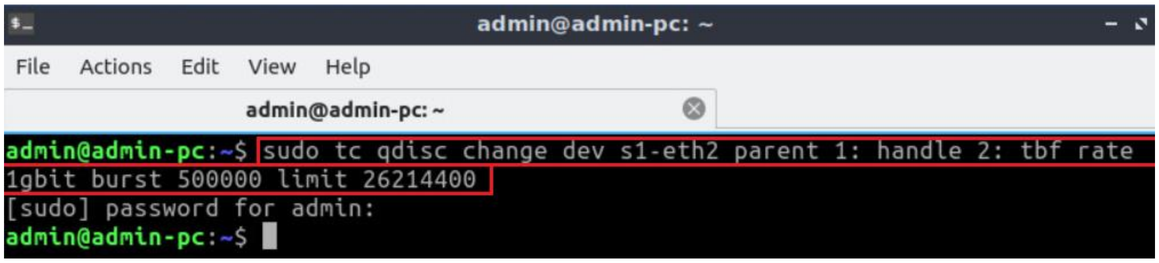
In this section, you are going to change the switch S1's buffer size to 10·BDP and emulate a 1 Gbps Wide Area Network (*WAN*) using the Token Bucket Filter (`tbf`). Then, you will test the throughput between host h1 and host h3. The BDP is 2,621,440 bytes, thus the `tbf` limit value will be set to 26,214,400.

5.1 Setting switch S1's buffer size to 10·BDP

Step 1. Apply `tbf` rate limiting rule on switch S1's *s1-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 change dev s1-eth2 parent 1: handle 2: tbf rate 1gbit burst
500000 limit 26214400
```



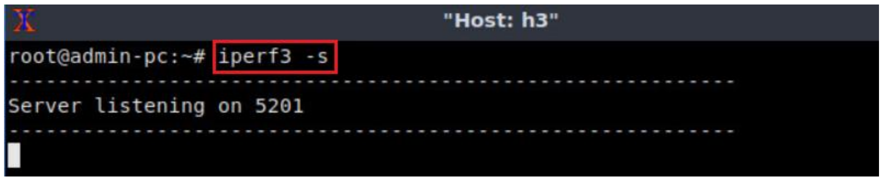
```
admin@admin-pc:~$ sudo tc qdisc change dev s1-eth2 parent 1: handle 2: tbf rate
1gbit burst 500000 limit 26214400
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 36. Limiting rate to 1 Gbps and setting the buffer size to 10·BDP on switch S1's interface.

5.2 Throughput and latency tests

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

```
iperf3 -s
```

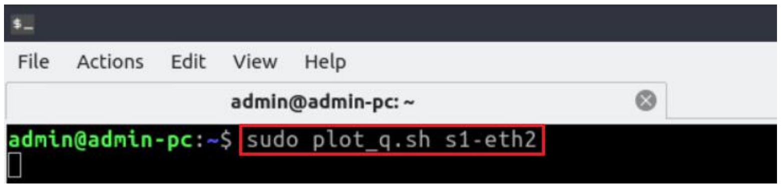


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 37. Starting iPerf3 server on host h3.

Step 2. 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 s1-eth2
```



```
admin@admin-pc:~$ sudo plot_q.sh s1-eth2
```

Figure 38. Plotting the queue occupancy on switch S1's *s1-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 *s1-eth2* interface on switch S1, the queue occupancy is constantly 0.

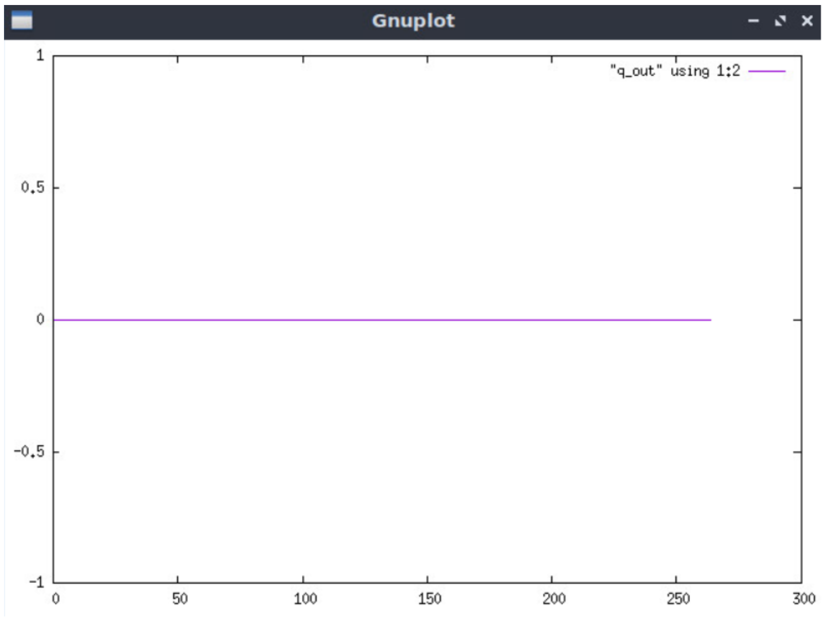
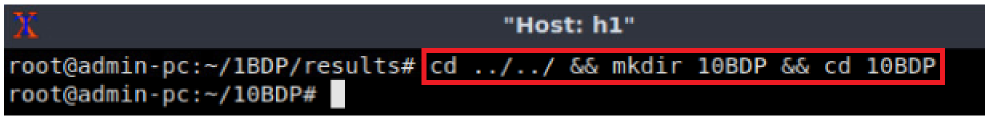


Figure 39. Queue occupancy on switch S1's *s1-eth2* interface.

Step 3. Exit from 1BDP/results directory, then create a directory *10BDP* and navigate into it using the following command.

```
cd ../../ && mkdir 10BDP && cd 10BDP
```

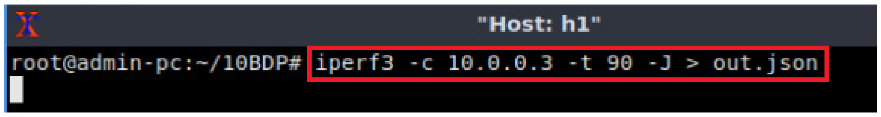


```
root@admin-pc:~/1BDP/results# cd ../../ && mkdir 10BDP && cd 10BDP
root@admin-pc:~/10BDP#
```

Figure 40. 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
```

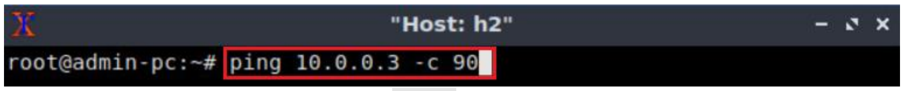


```
root@admin-pc:~/10BDP# iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

Figure 41. 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
```



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

Figure 42. 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.

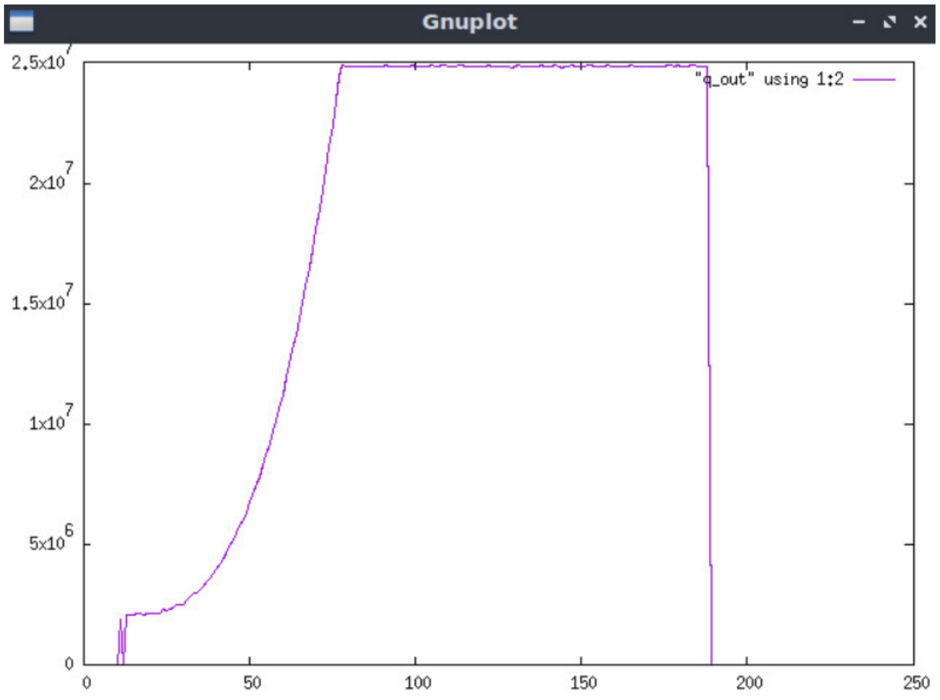


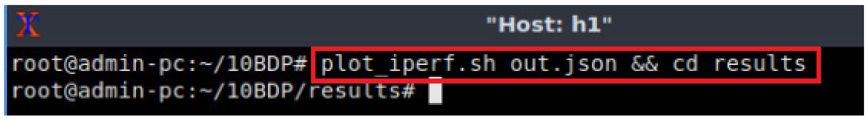
Figure 43. Queue occupancy on switch S1's *s1-eth2* interface.

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. Note that the buffer is almost always fully occupied, which will lead to an increase in the latency as demonstrated next.

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
```

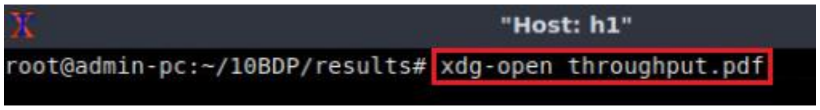


```
root@admin-pc:~/10BDP# plot_iperf.sh out.json && cd results
root@admin-pc:~/10BDP/results#
```

Figure 44. 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:~/10BDP/results# xdg-open throughput.pdf
```

Figure 45. Opening the *throughput.pdf* file.

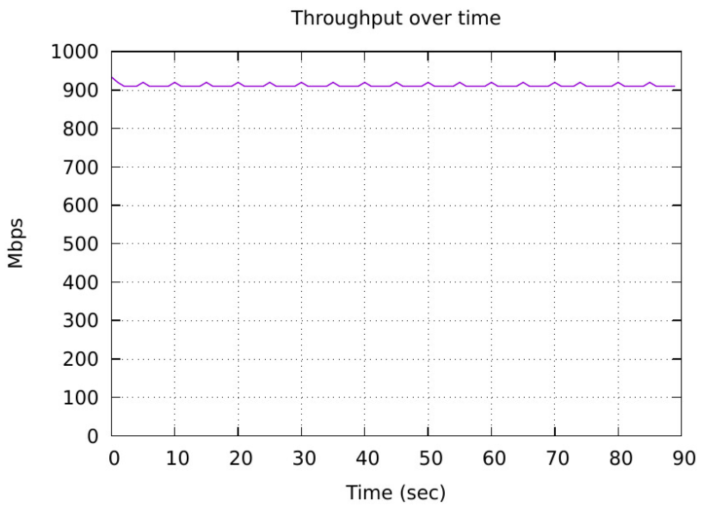
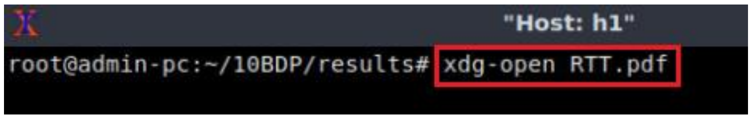


Figure 46. Measured throughput.

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 the switch's buffer size to 10·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:~/10BDP/results# xdg-open RTT.pdf
```

Figure 47. Opening the *RTT.pdf* file.

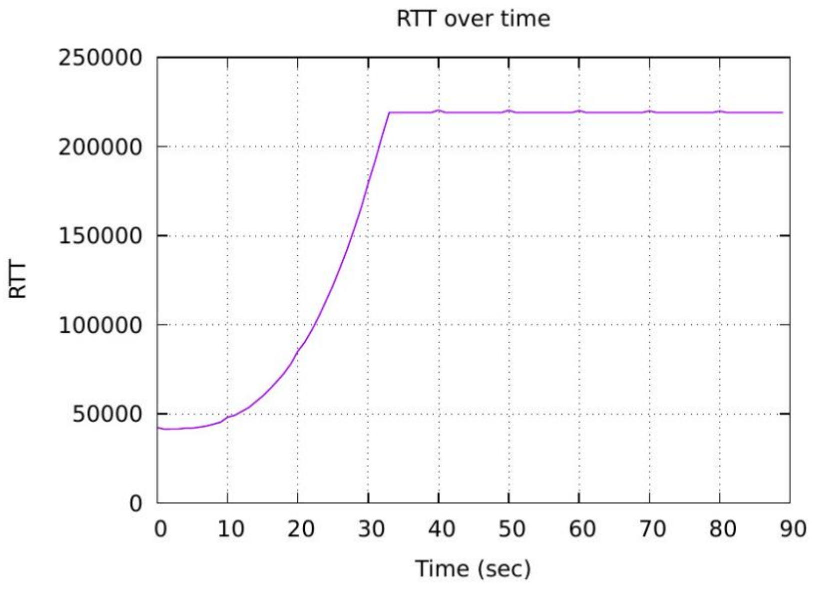
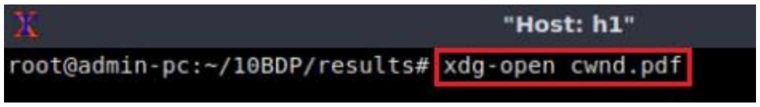


Figure 48. Measured Round-Trip Time.

The graph above shows that the RTT increased from approximately 50000 microseconds (50ms) to 230000 microseconds (230ms). The output above shows that there is a bufferbloat problem as the average latency is significantly greater than the configured delay (20ms). Since the buffer on the switch is accommodating a large congestion window, latency is increased as new incoming packets have to wait in the highly occupied queue.

Step 11. Close the *RTT.pdf* window then open the congestion window (cwnd) file using the command below.

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



```
"Host: h1"
root@admin-pc:~/10BDP/results# xdg-open cwnd.pdf
```

Figure 49. Opening the *cwnd.pdf* file.

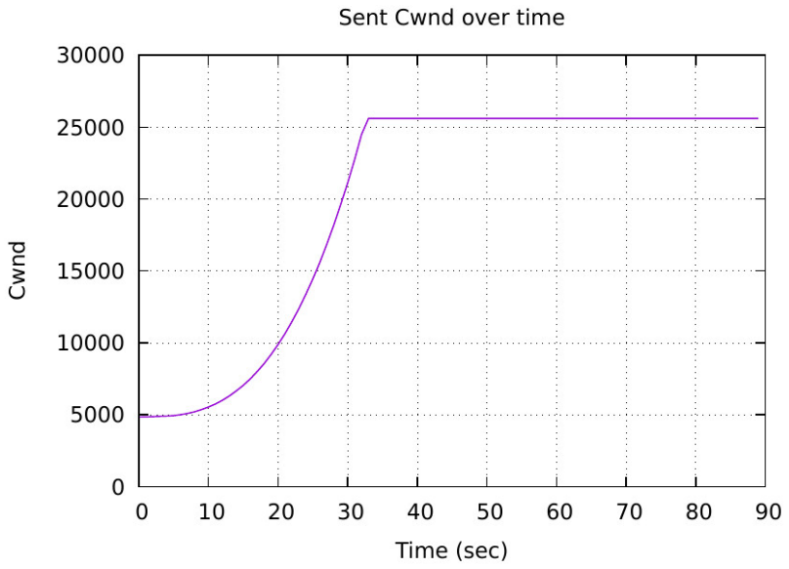
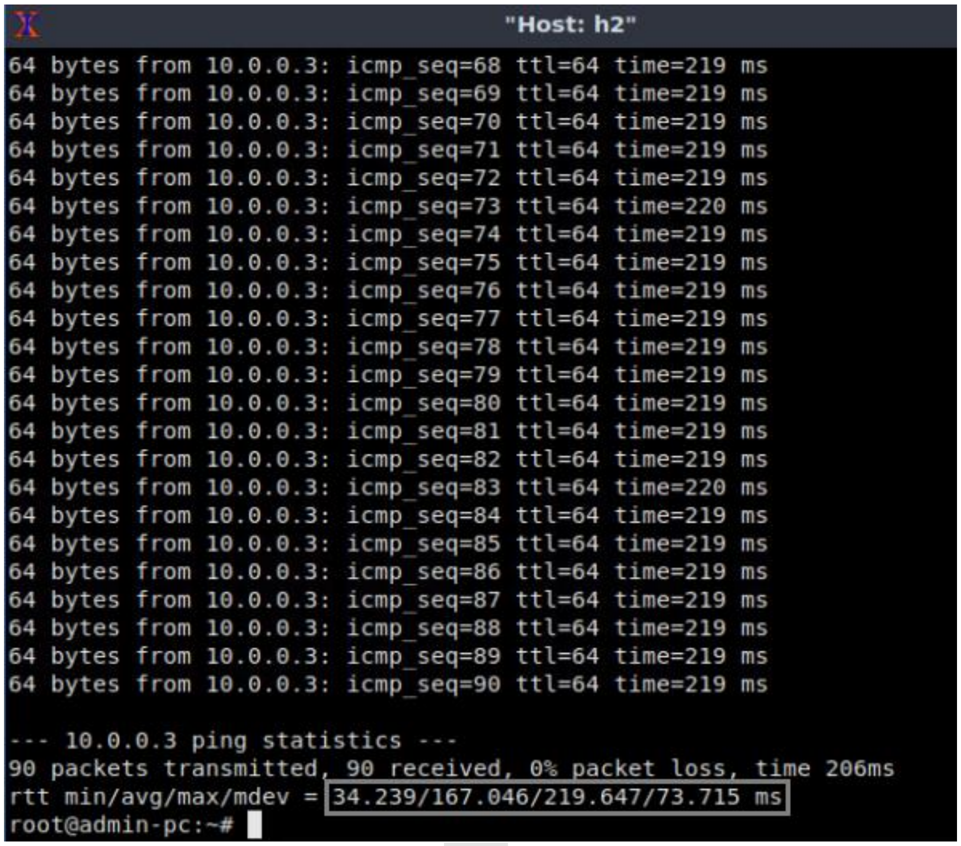


Figure 50. Congestion window evolution.

The graph above shows the evolution of the congestion window. Note how the congestion window peaked at 25.2 Mbytes compared to the previous test where it peaked at approximately 4.5 Mbytes. Since the queue size was configured with a large value, TCP continued to increase the congestion window as no packet losses were inferred.

Step 12. Close the *cwnd.pdf* window then go back to h2's terminal to see the `ping` output.



```
64 bytes from 10.0.0.3: icmp_seq=68 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=69 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=70 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=71 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=72 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=73 ttl=64 time=220 ms
64 bytes from 10.0.0.3: icmp_seq=74 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=75 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=76 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=77 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=78 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=79 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=80 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=81 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=82 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=83 ttl=64 time=220 ms
64 bytes from 10.0.0.3: icmp_seq=84 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=85 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=86 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=87 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=88 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=89 ttl=64 time=219 ms
64 bytes from 10.0.0.3: icmp_seq=90 ttl=64 time=219 ms

--- 10.0.0.3 ping statistics ---
90 packets transmitted, 90 received, 0% packet loss, time 206ms
rtt min/avg/max/mdev = 34.239/167.046/219.647/73.715 ms
root@admin-pc:~#
```

Figure 51. `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 34.239, 167.046, 219.647, and 73.715 milliseconds, respectively. The output also verifies that there is a bufferbloat problem as the average latency (167.046) is significantly greater than the configured delay (20ms).

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

This concludes Lab 14. Stop the emulation and then exit out of MiniEdit.

References

1. J. Kurose, K. Ross, "Computer networking, a top-down approach," 7th Edition, Pearson, 2017.
2. C. Villamizar, C. Song, "High performance TCP in ansnet," ACM Computer Communications Review, vol. 24, no. 5, pp. 45-60, Oct. 1994.
3. 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.
4. J. Gettys, K. Nichols, "Bufferbloat: dark buffers in the internet," Communications of the ACM, vol. 9, no. 1, pp. 57-65, Jan. 2012.

5. 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.