



UNIVERSITY OF  
**SOUTH CAROLINA**

## **NETWORK TOOLS AND PROTOCOLS**

### **Lab 5: Setting WAN Bandwidth with Token Bucket Filter (TBF)**

Document Version: **06-14-2019**



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 Token Bucket algorithm .....	3
2 Lab topology.....	5
2.1 Starting host h1 and host h2 .....	7
3 Rate limiting on end-hosts.....	8
3.1 Identify interface of host h1 and host h2 .....	8
3.2 Emulating 10 Gbps high-latency WAN .....	9
4 Rate limiting on switches .....	11
5 Combining NETEM and TBF .....	15
References .....	18

## Overview

This lab explains the Token Bucket Filter (TBF) queuing discipline which shapes incoming/outgoing traffic to limit the bandwidth. Throughput measurements are also conducted in this lab to verify the bandwidth-limiting configuration with TBF.

## Objectives

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

1. Understand the Token Bucket algorithm.
2. Use Token Bucket Filter (*tbfb*), which is a Linux implementation of the Token Bucket algorithm on network interfaces.
3. Understand how to combine queuing disciplines in Linux Traffic Control (*tc*).
4. Combine *tbfb* and *NETEM*.
5. Emulate WAN properties in Mininet.
6. Visualize iPerf3's output after modifying the network's parameters.

## 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 Token Bucket algorithm.
2. Section 2: Lab Topology.
3. Section 3: Rate limiting on end-hosts.
4. Section 4: Rate limiting on switches.
5. Section 5: Combining NETEM and TBF.

### 1 Introduction to Token Bucket algorithm

When simulating a Wide Area Network (WAN), it is sometimes necessary to limit the bandwidth of devices (end hosts and networking devices) to observe the network's behavior in different conditions.

The *Token Bucket* is an algorithm used in packet-switching networks to limit the bandwidth and the burstiness of the traffic. In summary, token bucket consists of adding tokens (represented as packets or packets' bytes) at a fixed rate to a fixed-capacity bucket. When a new packet arrives, the bucket is inspected to check the number of available tokens; if at least  $n$  tokens are available,  $n$  tokens are removed from the bucket, and the packet is sent to the network. Else, no tokens are removed, and the packet is considered *non-conformant*. In such case, the packet might be dropped, enqueued, or transmitted but marked as non-conformant. This algorithm is illustrated in Figure 1.

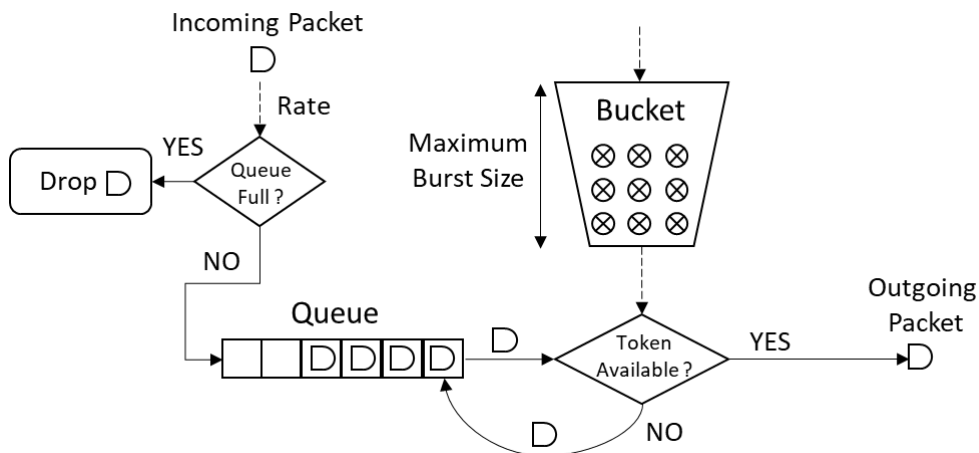


Figure 1. Token bucket filter.

The *rate*, which is the transmission speed, is determined by the frequency at which tokens are added to the bucket.

Another important property of the token bucket algorithm is *burstiness*; when the bucket becomes completely occupied (i.e. no packets are consuming tokens), new packets will consume tokens right away, without being limited. Burstiness is defined as the number of tokens that can fit in the bucket, or the bucket size.

To provide limits and control over the bursts, token bucket implementations often create another smaller bucket with a size equal to the *Maximum Transmission Unit (MTU)*, and a rate much faster than the original bucket (the *peak rate*). Its rate defines the maximum speed of bursts.

The token bucket algorithm implemented in Linux is the Token Bucket Filter (*tbfb*), which is a queuing discipline used in conjunction with the Linux Traffic Control (*tc*) to shape traffic.

Figure 2 depicts the main parameters used by `tbfb`.

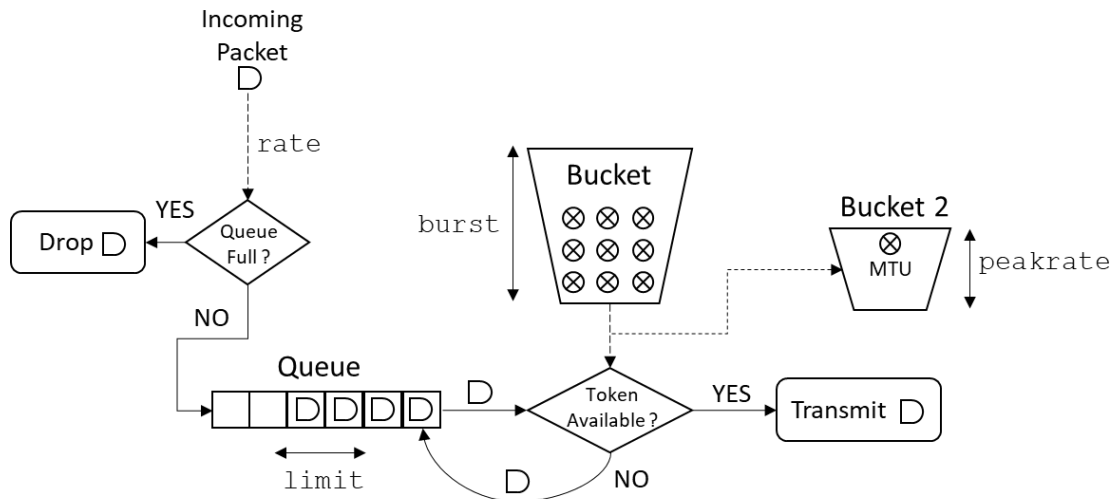


Figure 2. `tbf` parameters and architecture.

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

```
tc qdisc [add | ...] dev [dev_id] root tbf limit [BYTES] burst [BYTES] rate
[bps] [mtu BYTES] [ peakrate BPS ] [ latency TIME ]
```

- `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.
- `tbf`: this parameter specifies the Token Bucket Filter algorithm.
- `limit [BYTES]`: size of the packet queue in bytes.
- `burst [BYTES]`: number of bytes that can fit in the bucket.
- `rate [BPS]`: transmission speed, determined by the frequency at which tokens are added to the bucket.
- `mtu [BYTES]`: maximum transmission unit in bytes.
- `peakrate [BPS]`: the maximum speed of a burst.
- `latency [TIME]`: the maximum time a packet can wait in the queue.

In this lab, we will use the `tbf` queuing discipline to emulate the aforementioned parameters affecting the network behavior.

## 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.

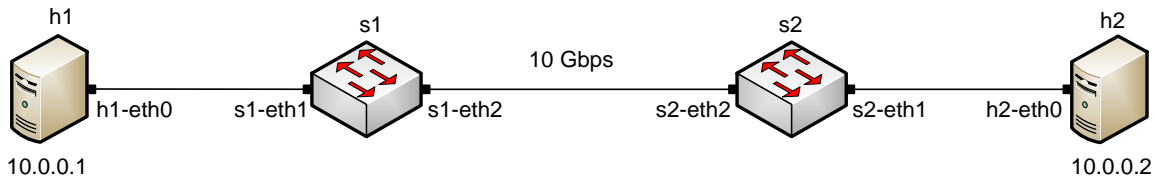


Figure 3. Lab topology.

**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`.

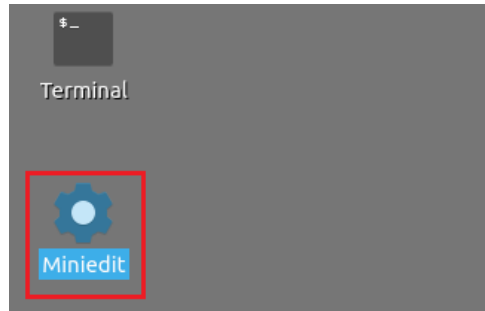


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 5.mn* topology file and click on *Open*.

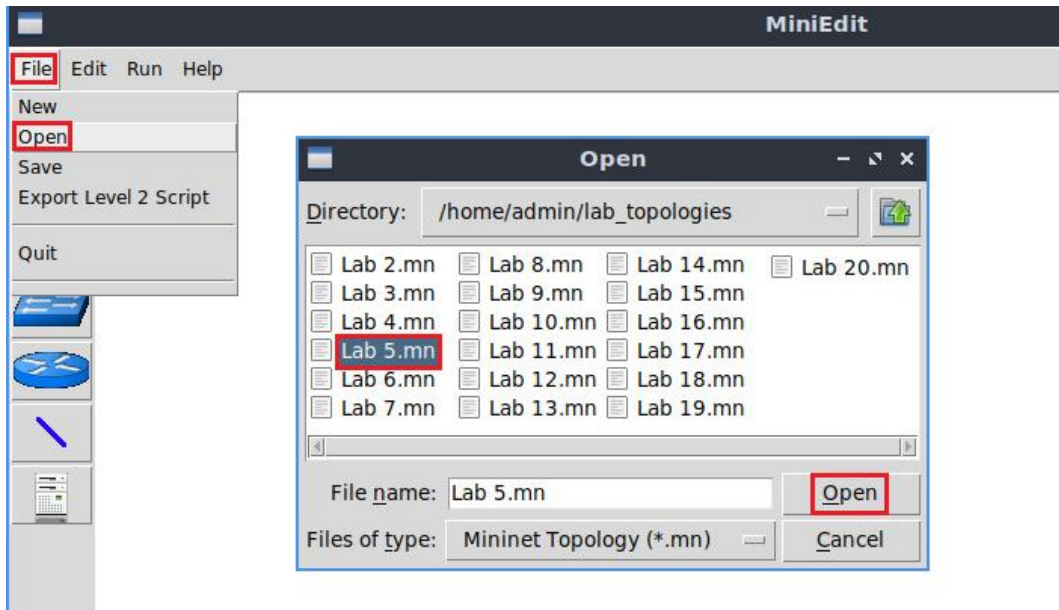


Figure 5. MiniEdit's *Open* dialog.

**Step 3.** Before starting the measurements between host h1 and host h2, the network must be started. Click on the *Run* button located at the bottom left of MiniEdit's window to start the emulation.

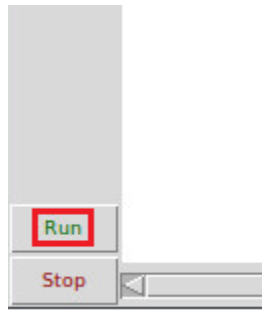


Figure 6. Running the emulation.

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

## 2.1 Starting host h1 and host h2

**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.

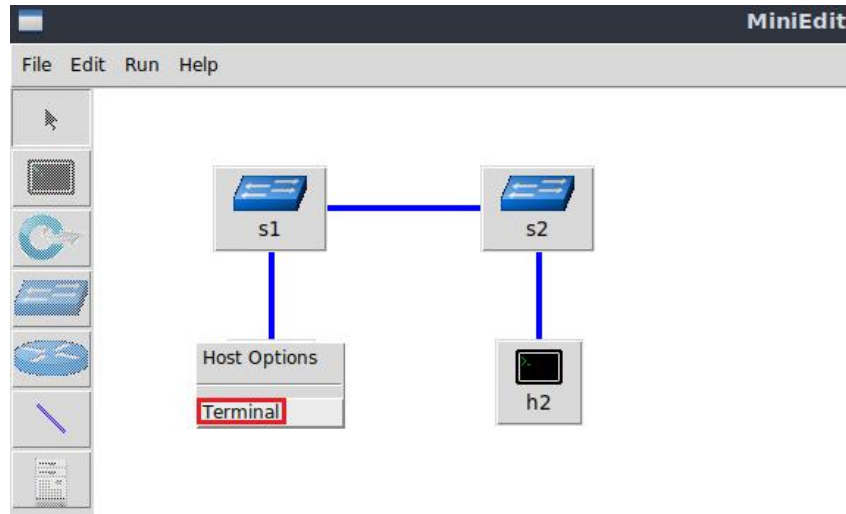


Figure 7. Opening a terminal on host h1.

**Step 2.** Apply the same steps on host h2 and open its *Terminal*.

**Step 3.** Test connectivity between the end-hosts using the `ping` command. On host h1, type the command `ping 10.0.0.2`. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.

```

Host: h1
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.33 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.044 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 91ms
rtt min/avg/max/mdev = 0.042/0.260/1.327/0.477 ms
root@admin-pc:~#

```

Figure 8. Connectivity test using `ping` command.

Figure 8 indicates that there is connectivity between host h1 and host h2.

### 3 Rate limiting on end-hosts

The `tc` command can be applied on the network interface of a device to shape egress traffic. In this section, the user will limit the sending rate of an end-host using the Token Bucket Filter (`tbf`), which is an implementation of the Token bucket algorithm.

#### 3.1 Identify interface of host h1 and host h2

According to the previous section, we must identify the interfaces on the connected hosts.

**Step 1.** On host h1, type the command `ifconfig` to display information related to its network interfaces and their assigned IP addresses.

```

Host: h1
root@admin-pc:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::f0d6:67ff:fe01:6041 prefixlen 64 scopeid 0x20<link>
    ether f2:d6:67:01:60:41 txqueuelen 1000 (Ethernet)
    RX packets 51 bytes 5112 (5.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 21 bytes 1678 (1.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@admin-pc:~#

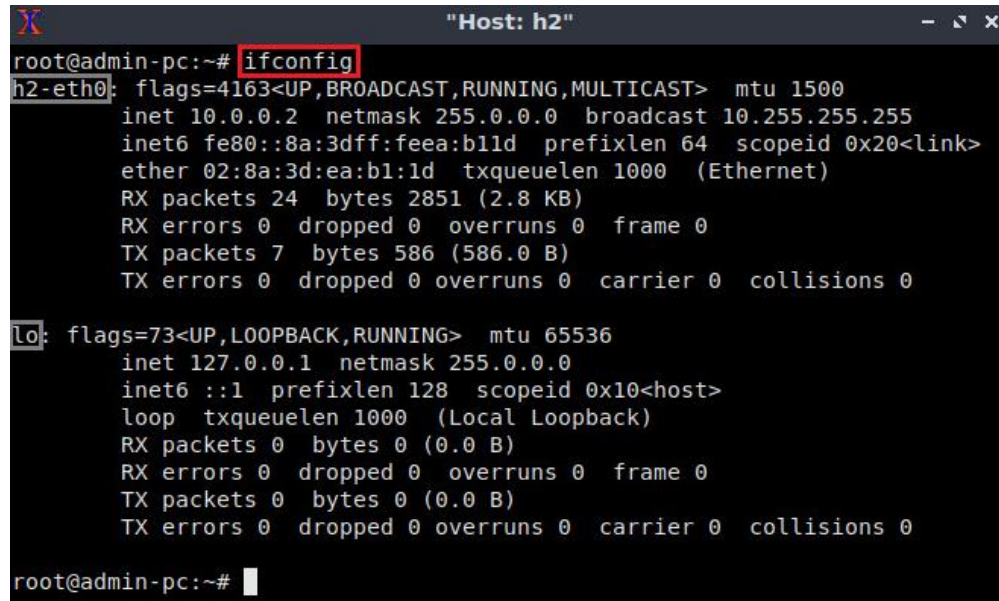
```

Figure 9. Output of `ifconfig` command on host h1.



The output of the `ifconfig` command indicates that host h1 has two interfaces: `h1-eth0` and `lo`. The interface `h1-eth0` at host h1 is configured with IP address 10.0.0.1 and subnet mask 255.0.0.0. This interface must be used in `tc` when emulating the network.

**Step 2.** In host h2's command line, type the command `ifconfig` as well.



```

root@admin-pc:~# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.2 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::8a:3dff:feea:b11d prefixlen 64 scopeid 0x20<link>
    ether 02:8a:3d:ea:b1:1d txqueuelen 1000 (Ethernet)
    RX packets 24 bytes 2851 (2.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 7 bytes 586 (586.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@admin-pc:~#

```

Figure 10. Output of `ifconfig` command on host h2.

The output of the `ifconfig` command indicates that host h2 has two interfaces: `h2-eth0` and `lo`. The interface `h2-eth0` at host h1 is configured with IP address 10.0.0.2 and subnet mask 255.0.0.0. This interface must be used in `tc` when emulating the network.

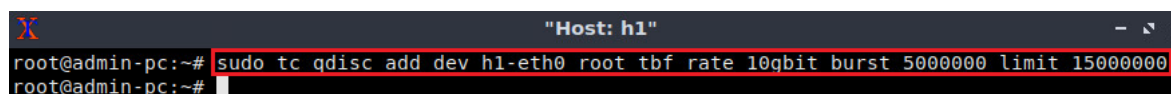
### 3.2 Emulating 10 Gbps high-latency WAN

In this section, you will use `tc` command on a network interface to control the egress rate.

**Step 1.** Modify the bandwidth of host h1 typing the command below. This command sets the bandwidth to 10 Gbps on host h1's `h1-eth0` interface. The `tc` parameters are the following:

- `rate`: 10gbit
- `burst`: 5,000,000
- `limit`: 15,000,000

```
sudo tc qdisc add dev h1-eth0 root tbf rate 10gbit burst 5000000 limit 15000000
```



```

root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root tbf rate 10gbit burst 5000000 limit 15000000
root@admin-pc:~#

```

Figure 10. Limiting rate with TBF to 10 Gbps.

This command can be summarized as follows:

- `sudo`: enable the execution of the command with higher security privileges.
- `tc`: invoke Linux's traffic control.
- `qdisc`: modify the queuing discipline of the network scheduler.
- `add`: create a new rule.
- `dev h1-eth0 root`: specify the interface on which the rule will be applied.
- `tbfb`: use the token bucket filter algorithm.
- `rate`: specify the transmission rate (10 Gbps).
- `burst`: number of bytes that can fit in the bucket (5,000,000).
- `limit`: queue size in bytes (15,000,000).

*Burst calculation:* `tbfb` requires setting a burst value when limiting the rate. This value must be high enough to allow your configured rate. Specifically, it must be at least the specified  $rate / HZ$ , where HZ is clock rate, configured as a kernel parameter, and can be extracted using the command shown below.

```
egrep '^CONFIG_HZ_[0-9]+' /boot/config-$(uname -r)
```

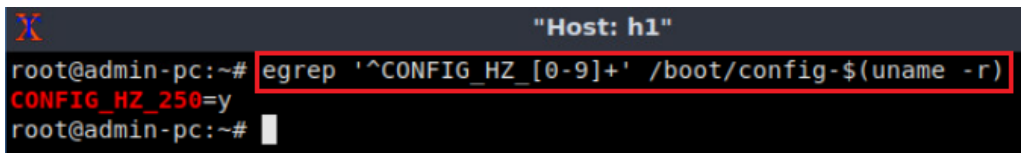


Figure 11. Retrieving system's HZ.

The HZ on Client1 is 250. Thus, to calculate the burst, we divide 10 Gbps by 250:

$$10 \text{ Gbps} = 10,000,000,000 \text{ bps}$$

$$Burst = \frac{10,000,000,000}{250} = 40,000,000 \text{ bits}$$

$$Burst = 40,000,000 \text{ bits} = 5,000,000 \text{ bytes}$$

The resulting value is to be used in the command as the *burst* value.

**Step 2.** The user can now verify the previous configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in the figure below:

```
iperf3 -s
```

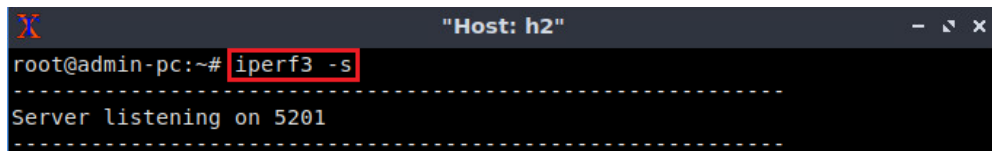
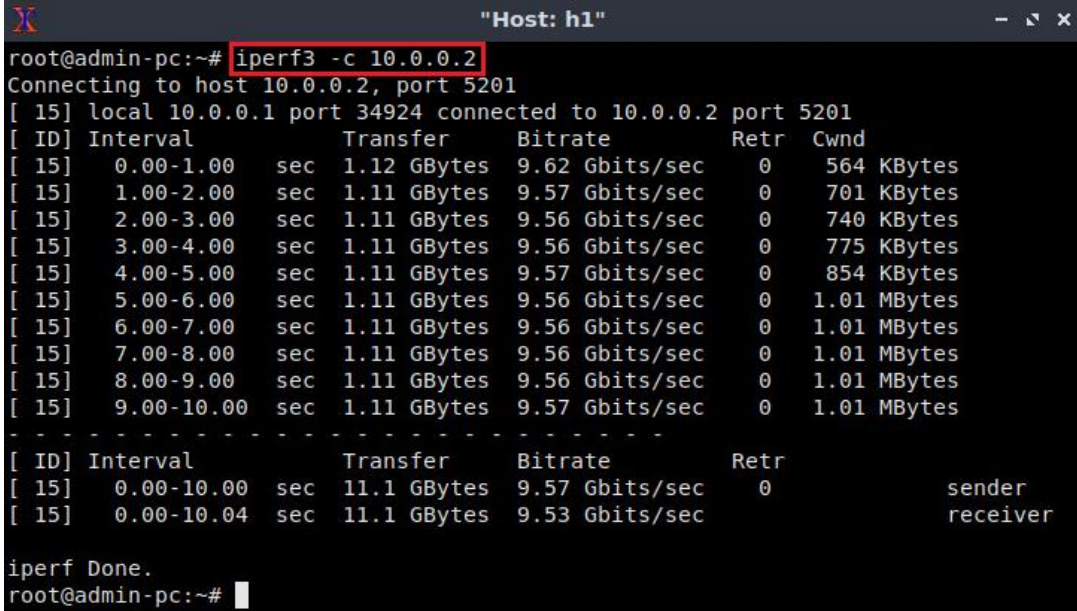


Figure 12. Host h2 running iPerf3 as server.

**Step 3.** Now to launch iPerf3 in client mode, run the command `iperf3 -c 10.0.0.2` in host h1's terminal as shown below:

```
iperf3 -c 10.0.0.2
```



```

root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 34924 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate      Retr  Cwnd
[ 15]  0.00-1.00      sec   1.12 GBytes  9.62 Gbits/sec    0   564 KBytes
[ 15]  1.00-2.00      sec   1.11 GBytes  9.57 Gbits/sec    0   701 KBytes
[ 15]  2.00-3.00      sec   1.11 GBytes  9.56 Gbits/sec    0   740 KBytes
[ 15]  3.00-4.00      sec   1.11 GBytes  9.56 Gbits/sec    0   775 KBytes
[ 15]  4.00-5.00      sec   1.11 GBytes  9.57 Gbits/sec    0   854 KBytes
[ 15]  5.00-6.00      sec   1.11 GBytes  9.56 Gbits/sec    0   1.01 MBytes
[ 15]  6.00-7.00      sec   1.11 GBytes  9.56 Gbits/sec    0   1.01 MBytes
[ 15]  7.00-8.00      sec   1.11 GBytes  9.56 Gbits/sec    0   1.01 MBytes
[ 15]  8.00-9.00      sec   1.11 GBytes  9.56 Gbits/sec    0   1.01 MBytes
[ 15]  9.00-10.00     sec   1.11 GBytes  9.57 Gbits/sec    0   1.01 MBytes
-----
[ ID] Interval           Transfer     Bitrate      Retr
[ 15]  0.00-10.00     sec   11.1 GBytes  9.57 Gbits/sec    0      sender
[ 15]  0.00-10.04     sec   11.1 GBytes  9.53 Gbits/sec                receiver

iperf Done.
root@admin-pc:~#

```

Figure 13. iPerf3's report after limiting the rate on host h1 to 10 Gbps.

The figure above shows the iPerf3 report after limiting the rate on host h1 using `tbf`. The average achieved throughputs are 9.57 Gbps (sender) and 9.53 Gbps (receiver). Since we executed the command on host h1's terminal, the rule was applied to host h1's network interface. However, it is also possible to limit the rate on the switch interfaces as explained next.

**Step 4.** In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

## 4 Rate limiting on switches

The previous section explained how to use the token bucket filter on end-hosts' network interfaces. In this section, we will explain how to apply the filter on switch interfaces. By limiting the rate on switch S1's `s1-eth2` interface, all communication sessions between switch S1 and switch S2 will be filtered by the applied rule(s).

In previous tests, we applied the command on host h1's terminal; switches, however, we do not have terminals where commands can be set and applied. Recall that we are using Mininet for this emulation, which creates virtual interfaces emulating the switch functionality. Therefore, these virtual interfaces can be identified using the `ifconfig` command, but this time, it should be issued on the client's terminal (e.g., the terminal located on the Desktop) and not on end-hosts (host h1 or host h2).

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

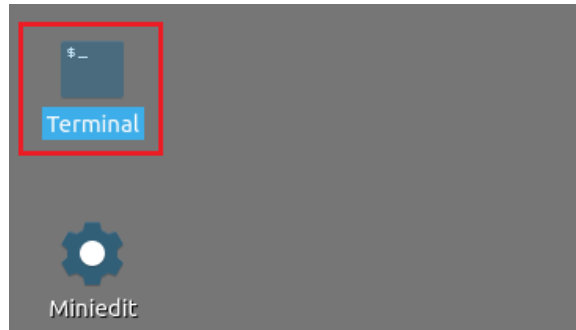


Figure 14. Shortcut to open a Linux terminal.

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 for execution.

**Step 2.** Type in the terminal the command `ifconfig` to display information related to its network interfaces.

```

admin@admin-pc: ~
File Actions Edit View Help
admin@admin-pc: ~
admin@admin-pc:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    ether 00:50:56:ae:1c:cd txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 198 bytes 32455 (32.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 19 base 0x2000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 4484 bytes 297536 (297.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4484 bytes 297536 (297.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::80d7:ff:fe14:b1a4 prefixlen 64 scopeid 0x20<link>
    ether 82:d7:00:14:b1:a4 txqueuelen 1000 (Ethernet)
    RX packets 216964 bytes 14138070762 (14.1 GB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 213105 bytes 14068378 (14.0 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::4411:aff:fef7:ff3e prefixlen 64 scopeid 0x20<link>
    ether 46:11:0a:f7:ff:3e txqueuelen 1000 (Ethernet)
    RX packets 213075 bytes 14064944 (14.0 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 216993 bytes 14138074110 (14.1 GB)
    
```

```
s2-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet6 fe80::fc8d:6dff:fe08:76b0 prefixlen 64 scopeid 0x20<link>
ether fe:8d:6d:08:76:b0 txqueuelen 1000 (Ethernet)
RX packets 213045 bytes 14061510 (14.0 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 217023 bytes 14138077544 (14.1 GB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s2-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet6 fe80::ec53:8ff:fe09:8cb7 prefixlen 64 scopeid 0x20<link>
ether ee:53:08:09:8c:b7 txqueuelen 1000 (Ethernet)
RX packets 216993 bytes 14138074110 (14.1 GB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 213075 bytes 14064944 (14.0 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

admin@admin-pc:~$
```

Figure 15. Output of `ifconfig` command on the client's terminal.

Figure 15 shows the network interfaces of the client:

- `s1-eth1` is the interface connecting switch S1 to host h1.
- `s1-eth2` is the interface connecting switch S1 to switch S2.
- `s2-eth1` is interface connecting switch S2 to host h2.
- `s2-eth2` is interface connecting switch S2 to switch S1.

**Step 3.** Remove the previous configuration on host h1. Write the following command on host h1's terminal:

```
sudo tc qdisc del dev h1-eth0 root
```

```
Host: h1
root@admin-pc:~# sudo tc qdisc del dev h1-eth0 root
root@admin-pc:~#
```

Figure 16. Deleting all rules on host h1's network scheduler.

**Step 4.** Apply `tbft` rate limiting rule on switch S1's interface which connects it to switch S2 (`s1-eth2`). In the Client's terminal, type the command below. When prompted for a password, type `password` and hit enter. The `tbft` parameters are the following:

- `rate`: 10gbit
- `burst`: 5,000,000
- `limit`: 15,000,000

```
sudo tc qdisc add dev s1-eth2 root tbf rate 10gbit burst 5000000 limit 15000000
```

```
admin@admin-pc: ~
File Actions Edit View Help
admin@admin-pc:~
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root tbf rate 10gbit burst 5000000 limit 15000000
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 17. Limiting rate with TBF to 10 Gbps on switch S1's interface.

**Step 5.** The user can now verify the previous configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in Figure 18:

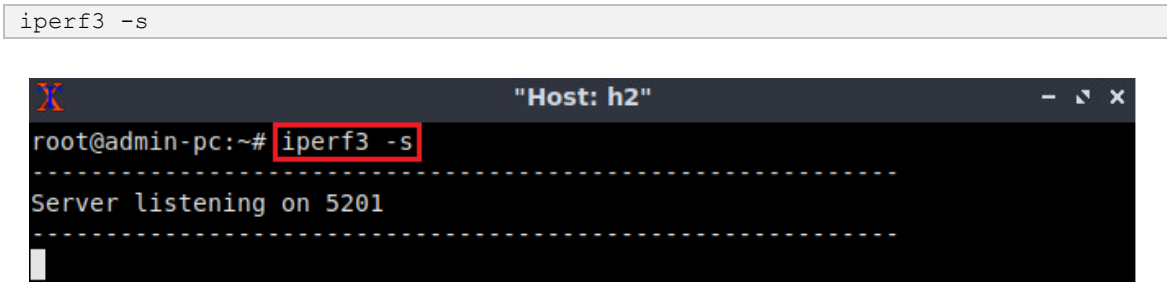


Figure 18. Host h2 running iPerf3 as server.

**Step 6.** Now to launch iPerf3 in client mode, run the command `iperf3 -c 10.0.0.2` in host h1's terminal as shown in the figure below:

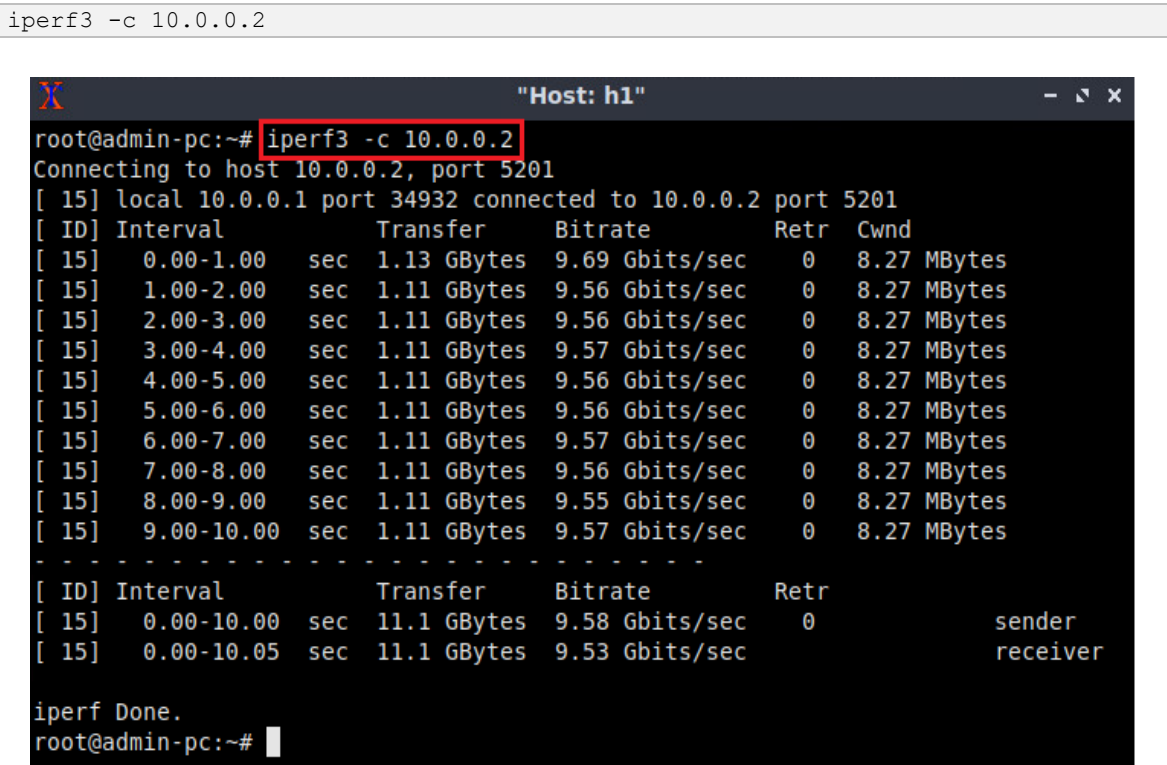


Figure 19. iPerf3's report after limiting the rate on switch S1 to 10 Gbps.

Again, the reported values match the desired throughput (10 Gbps). In practice, the reported throughput will not achieve the target (10 Gbps) but will achieve a throughput slightly less than the target.

**Step 7.** In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

## 5 Combining NETEM and TBF

NETEM is used to introduce delay, jitter, packet corruption, etc. TBF on the other hand can be used to limit the rate. However, this is not enough for emulating real networks, particularly WANs. Therefore, it is also possible to combine multiple impairments and activate them at the same time.

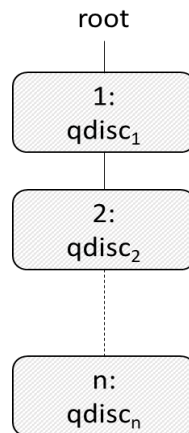


Figure 20. Chaining *qdiscs* hierarchy.

As shown in Figure 20, the first *qdisc* (*qdisc<sub>1</sub>*) is attached to the *root* label. Then, subsequent *qdiscs* can be attached to their *parents* by specifying the correct label. In this section, we will look at how to combine NETEM and TBF in order to have more properties emulated in our network. Specifically, we will introduce delay, jitter, and packet corruption, while specifying the rate on switch S1's interface.

**Step 1.** In the Client's terminal, type the following command to remove the previous configuration on switch S1.

```
sudo tc qdisc del dev s1-eth2 root
```

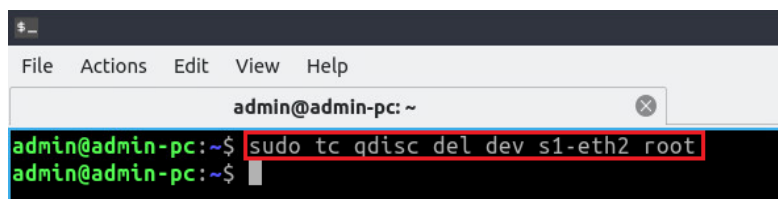
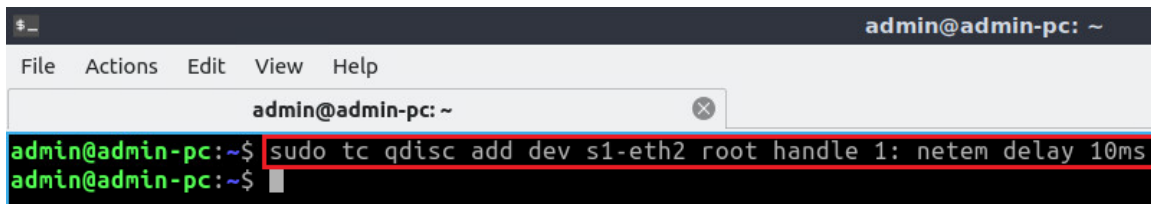


Figure 21. Deleting all rules on switch S1's *s1-eth2*.

**Step 2.** In the client's terminal, type the command below. When prompted for a password, type `password` and hit *Enter*.

```
sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 10ms
```



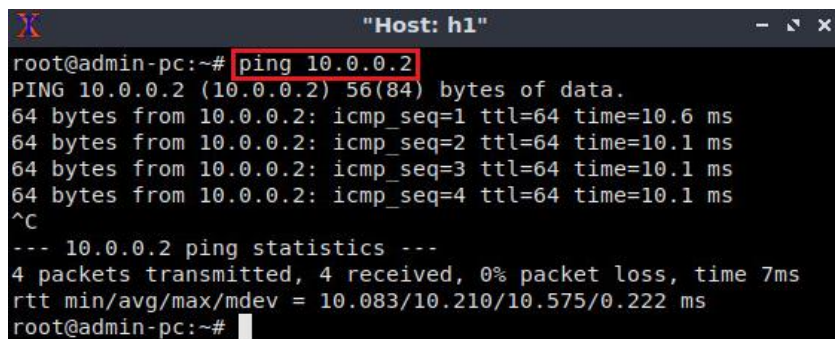
```
admin@admin-pc: ~
File Actions Edit View Help
admin@admin-pc: ~
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 10ms
admin@admin-pc:~$
```

Figure 22. Adding delay of 10ms to switch S1's *s1-eth2* interface.

The new keyword in this command is *handle* and its value reflects the number shown in Figure 22 above each *qdisc*. This means that our NETEM *qdisc* is attached to the root with the `handle 1:`.

**Step 3.** The user can now verify the previous configuration by using the `ping` tool to measure the Round-Trip Time (RTT). On the terminal of host h1, type `ping 10.0.0.2`. 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 h2 (10.0.0.2), successfully receiving responses back.

```
ping 10.0.0.2
```



```
Host: h1
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=10.6 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=10.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=10.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=10.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 10.083/10.210/10.575/0.222 ms
root@admin-pc:~#
```

Figure 23. Output of `ping 10.0.0.2` 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 10.083, 10.210, 10.575, and 0.222 milliseconds, respectively. Essentially, the standard deviation is an average of how far each ping RTT is from the average RTT. The higher the standard deviation, the more variable the RTT is.

**Step 4.** Now to add the second rule which applies rate limiting using `tbft`, issue the command shown below on the client's terminal. The `tbft` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 2gbit burst 1000000
limit 2500000
```



```
admin@admin-pc: ~
File Actions Edit View Help
admin@admin-pc: ~
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 2gbit
burst 1000000 limit 2500000
admin@admin-pc:~$
```

Figure 24. Adding a new rule while combining it with the previous.

**Step 5.** The user can now verify the previous configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in Figure 25:

```
iperf3 -s
```

```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----
Server listening on 5201
-----
```

Figure 25. Host h2 running iPerf3 as server.

**Step 6.** Now to launch iPerf3 in client mode again by running the command `iperf3 -c 10.0.0.2` in host h1's terminal as shown in Figure 26:

```
iperf3 -c 10.0.0.2
```

```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 34940 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate      Retr  Cwnd
[ 15]  0.00-1.00   sec    222 MBytes  1.86 Gbits/sec  426  3.41 MBytes
[ 15]  1.00-2.00   sec    228 MBytes  1.91 Gbits/sec   0  3.66 MBytes
[ 15]  2.00-3.00   sec    225 MBytes  1.89 Gbits/sec  450  2.70 MBytes
[ 15]  3.00-4.00   sec    231 MBytes  1.94 Gbits/sec   0  2.86 MBytes
[ 15]  4.00-5.00   sec    229 MBytes  1.92 Gbits/sec   0  2.98 MBytes
[ 15]  5.00-6.00   sec    228 MBytes  1.91 Gbits/sec   0  3.08 MBytes
[ 15]  6.00-7.00   sec    228 MBytes  1.91 Gbits/sec   0  3.15 MBytes
[ 15]  7.00-8.00   sec    229 MBytes  1.92 Gbits/sec   0  3.21 MBytes
[ 15]  8.00-9.00   sec    228 MBytes  1.91 Gbits/sec   0  3.24 MBytes
[ 15]  9.00-10.00  sec    229 MBytes  1.92 Gbits/sec   0  3.26 MBytes
-----
[ ID] Interval           Transfer     Bitrate      Retr
[ 15]  0.00-10.00  sec    2.22 GBytes  1.91 Gbits/sec  876  sender
[ 15]  0.00-10.04  sec    2.21 GBytes  1.89 Gbits/sec                receiver

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

Figure 26. iPerf3 throughput test after combining `qdiscs`.

The figure above shows the iPerf3 test output report. The average achieved throughputs are 1.86 Gbps (sender) and 1.84 Gbps (receiver).

**Step 7.** In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

This concludes Lab 5. Stop the emulation and then exit out of MiniEdit.

## References

1. Journey to the center of the linux kernel: traffic Control, shaping and QoS. [Online]. Available: [http://wiki.linuxwall.info/doku.php/en:ressources:dossiers:networking:traffic\\_control](http://wiki.linuxwall.info/doku.php/en:ressources:dossiers:networking:traffic_control).
2. How to use the linux traffic control panagiotis vouzis [Online]. Available: <https://netbeez.net/blog/how-to-use-the-linux-traffic-control>.