NETWORK TOOLS AND PROTOCOLS

Lab 1: Introduction to Mininet

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 Mininet 3
2 Invoking Mininet using the CLI........ 5
2.1 Invoking Mininet using the default topology........ 5
2.2 Testing connectivity 8
3 Building and emulating a network in Mininet using the GUI 9
3.1 Building the network topology........ 9
3.2 Testing connectivity 11
3.3 Automatic assignment of IP addresses 13
3.4 Saving and loading a Mininet topology........ 15
References 16

Overview

This lab provides an introduction to Mininet, a virtual testbed used for testing network tools and protocols. It demonstrates how to invoke Mininet from the command-line interface (CLI) utility and how to build and emulate topologies using a graphical user interface (GUI) application.

Objectives

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

1. Understand what Mininet is and why it is useful for testing network topologies.
2. Invoke Mininet from the CLI.
3. Construct network topologies using the GUI.
4. Save/load Mininet topologies using the GUI.

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 Mininet.
2. Section 2: Invoking Mininet using the CLI.
3. Section 3: Building and emulating a network in Mininet using the GUI.

1 Introduction to Mininet

Mininet is a virtual testbed enabling the development and testing of network tools and protocols. With a single command, Mininet can create a realistic virtual network on any type of machine (Virtual Machine (VM), cloud-hosted, or native). Therefore, it provides an inexpensive solution and streamlined development running in line with production networks[1]. Mininet offers the following features:

- Fast prototyping for new networking protocols.

- Simplified testing for complex topologies without the need of buying expensive hardware.
- Realistic execution as it runs real code on the Unix and Linux kernels.
- Open source environment backed by a large community contributing extensive documentation.

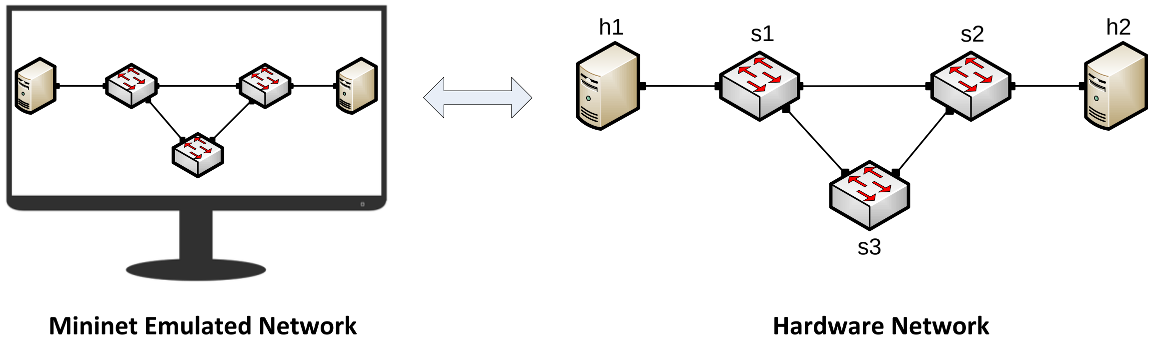


Figure 1. Hardware network vs. Mininet emulated network.

Mininet is useful for development, teaching, and research as it is easy to customize and interact with it through the CLI or the GUI. Mininet was originally designed to experiment with *OpenFlow*[2] and *Software-Defined Networking (SDN)*[3]. This lab, however, only focuses on emulating a simple network environment without SDN-based devices.

Mininet's logical nodes can be connected into networks. These nodes are sometimes called containers, or more accurately, *network namespaces*. Containers consume sufficiently few resources that networks of over a thousand nodes have created, running on a single laptop. A Mininet container is a process (or group of processes) that no longer has access to all the host system's native network interfaces. Containers are then assigned virtual Ethernet interfaces, which are connected to other containers through a virtual switch[4]. Mininet connects a host and a switch using a virtual Ethernet (veth) link. The veth link is analogous to a wire connecting two virtual interfaces, as illustrated below.

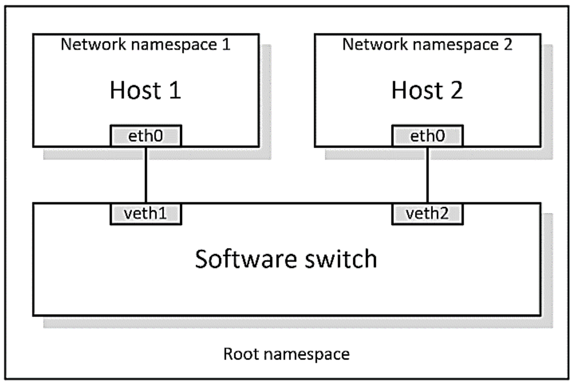


Figure 2. Network namespaces and virtual Ethernet links.

Each container is an independent network namespace, a lightweight virtualization feature that provides individual processes with separate network interfaces, routing tables, and Address Resolution Protocol (ARP) tables.
Mininet provides network emulation opposed to simulation, allowing all network software at any layer to be simply run *as is*; i.e. nodes run the native network software of

the physical machine. In a simulator environment on the other hand, applications and protocol implementations need to be ported to run within the simulator before they can be used[4].

2 Invoking Mininet using the CLI

The first step to start Mininet using the CLI is to start a Linux terminal.

2.1 Invoking Mininet using the default topology

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

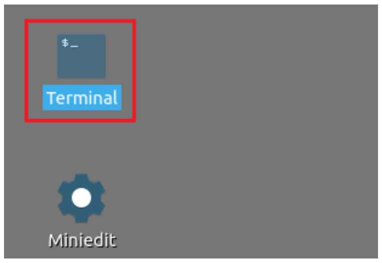
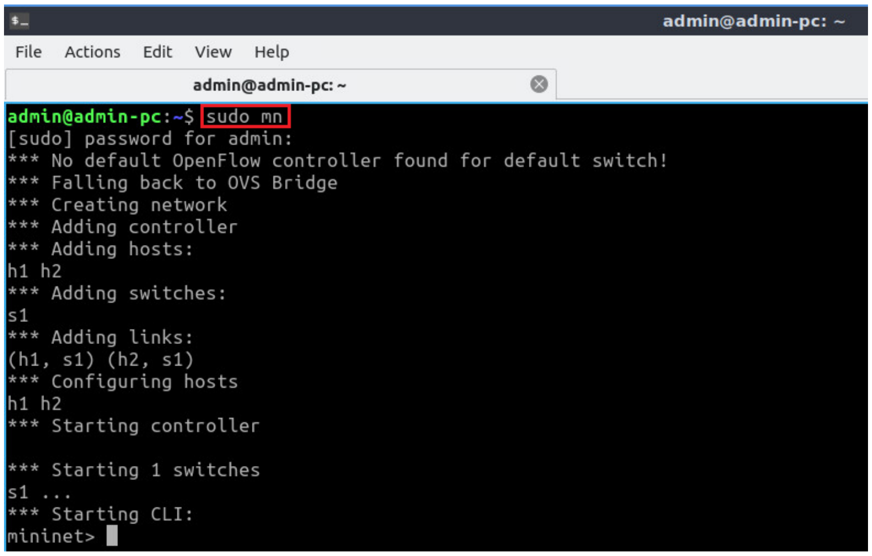


Figure 3. Shortcut to open a Linux terminal.

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. To start a minimal topology, enter the command `sudo mn` at the CLI. When prompted for a password, type `password` and hit enter. Note that the password will not be visible as you type it.



```
admin@admin-pc:~$ sudo mn
[sudo] password for admin:
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> 
```

Figure 4. Starting Mininet using the CLI.

The above command starts Mininet with a minimal topology, which consists of a switch connected to two hosts as shown below.

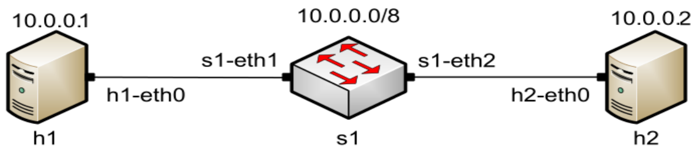
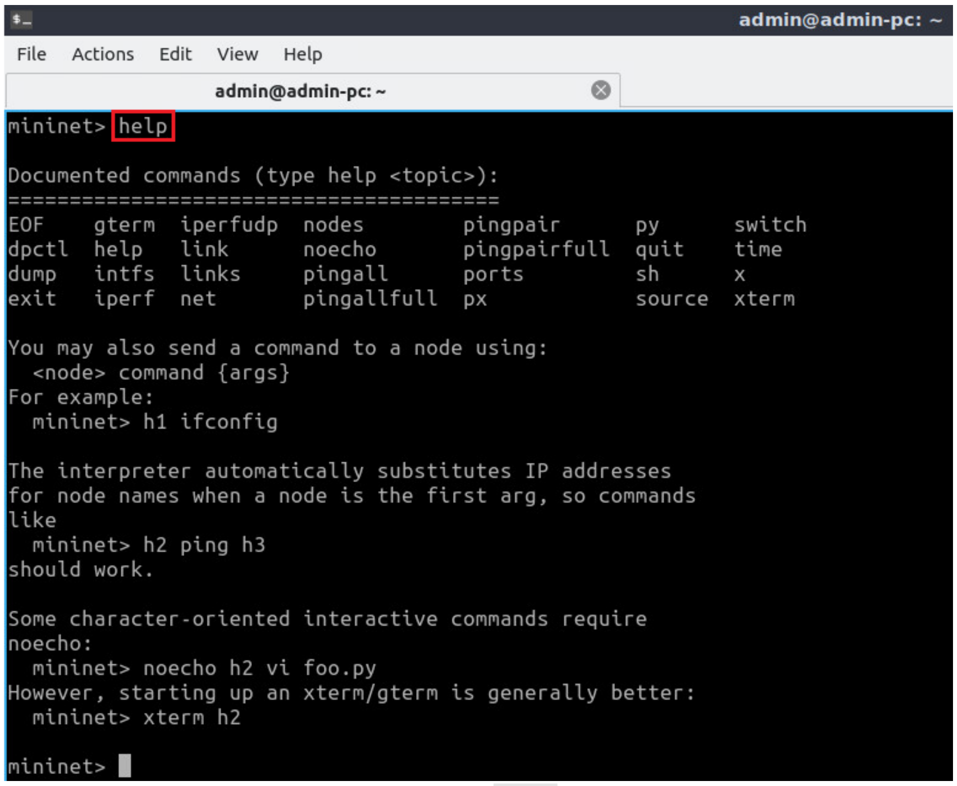


Figure 5. Mininet's default minimal topology.

When issuing the `sudo mn` command, Mininet initializes the topology and launches its command line interface which looks like this:

```
mininet>
```

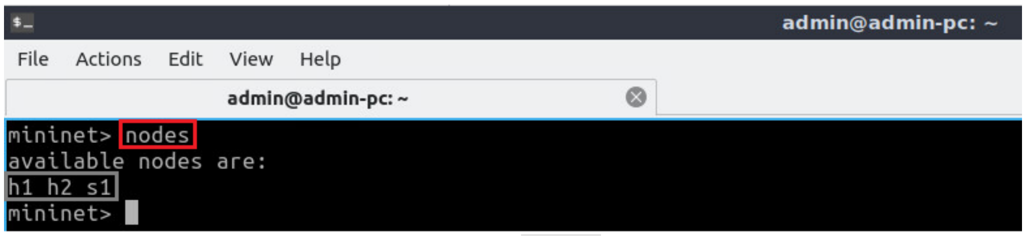
Step 3. To display the list of Mininet CLI commands and examples on their usage, type the command `help` in the Mininet CLI:



```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    gterm  iperfudp  nodes        pingpair      py      switch
dpctl  help   link      noecho       pingpairfull  quit    time
dump   intfs  links     pingall      ports         sh      x
exit   iperf  net       pingallfull  px            source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2

mininet>
```

Figure 6. Mininet's `help` command.

Step 4. To display the available nodes, type the command `nodes`:

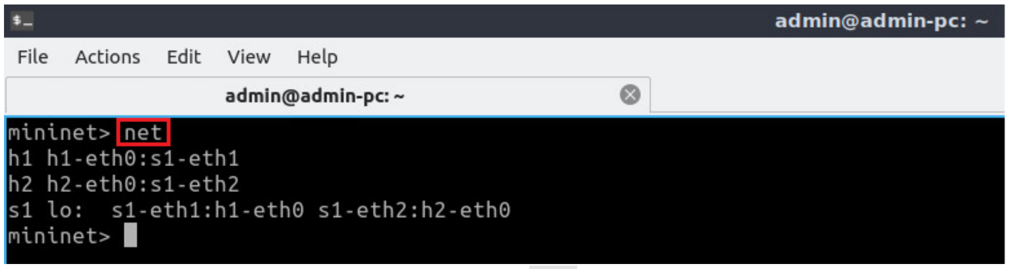


```
mininet> nodes
available nodes are:
h1 h2 s1
mininet>
```

Figure 7. Mininet's `nodes` command.

The output of this command shows that there are two hosts (host h1 and host h2) and a switch (s1).

Step 5. It is useful sometimes to display the links between the devices in Mininet to understand the topology. Issue the command `net` in the Mininet CLI to see the available links.



```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
mininet> 
```

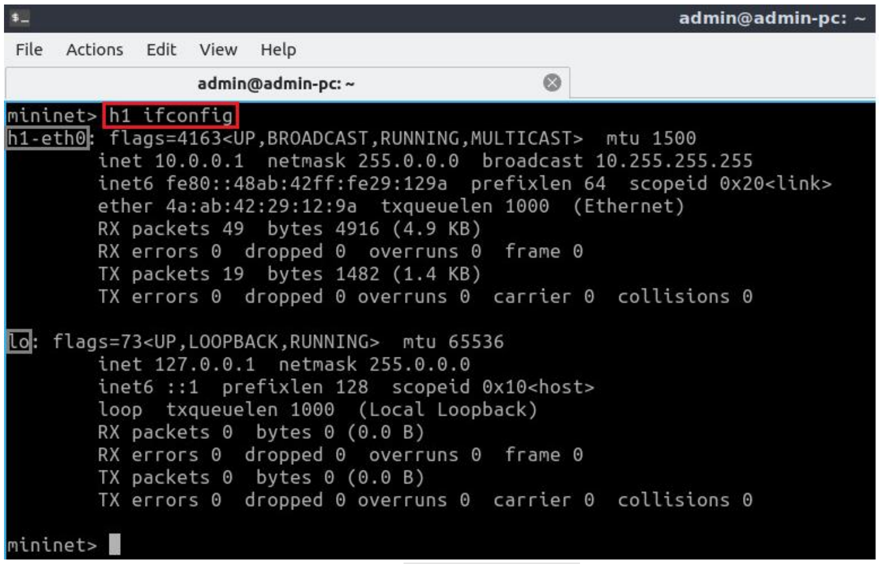
Figure 8. Mininet's `net` command.

The output of this command shows that:

1. Host h1 is connected using its network interface *h1-eth0* to the switch on interface *s1-eth1*.
2. Host h2 is connected using its network interface *h2-eth0* to the switch on interface *s1-eth2*.
3. Switch s1:
    a. has a loopback interface *lo*.
    b. connects to *h1-eth0* through interface *s1-eth1*.
    c. connects to *h2-eth0* through interface *s1-eth2*.

Mininet allows you to execute commands at a specific device. To issue a command for a specific node, you must specify the device first, followed by the command.

Step 6. Issue the command `h1 ifconfig`.



```
mininet> h1 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::48ab:42ff:fe29:129a  prefixlen 64  scopeid 0x20<link>
        ether 4a:ab:42:29:12:9a  txqueuelen 1000  (Ethernet)
        RX packets 49  bytes 4916 (4.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19  bytes 1482 (1.4 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

mininet> 
```

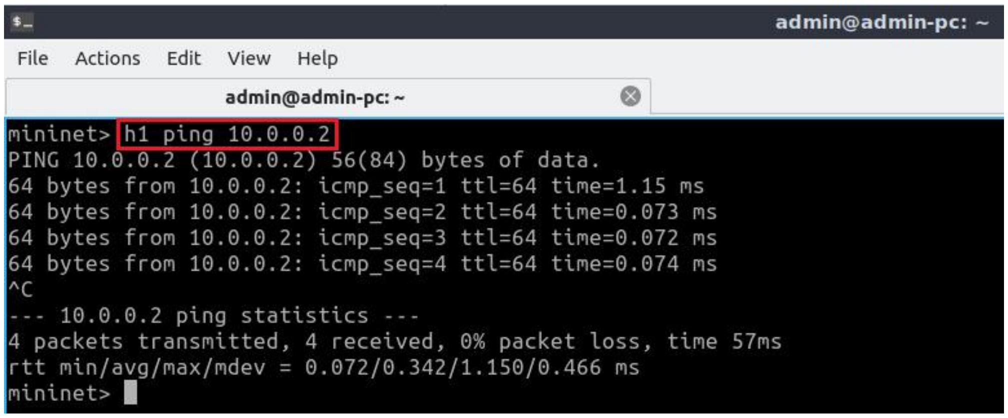
Figure 9. Output of `h1 ifconfig` command.

This command executes the `ifconfig` Linux command on host h1. The command shows host h1's interfaces. The display indicates that host h1 has an interface *h1-eth0* configured with IP address 10.0.0.1, and another interface lo configured with IP address 127.0.0.1 (loopback interface).

2.2 Testing connectivity

Mininet's default topology assigns the IP addresses 10.0.0.1/8 and 10.0.0.2/8 to host h1 and host h2 respectively. To test connectivity between them, you can use the command `ping`. The `ping` command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the remote computer and waiting for a response. Information available includes how many responses are returned and how long it takes for them to return.

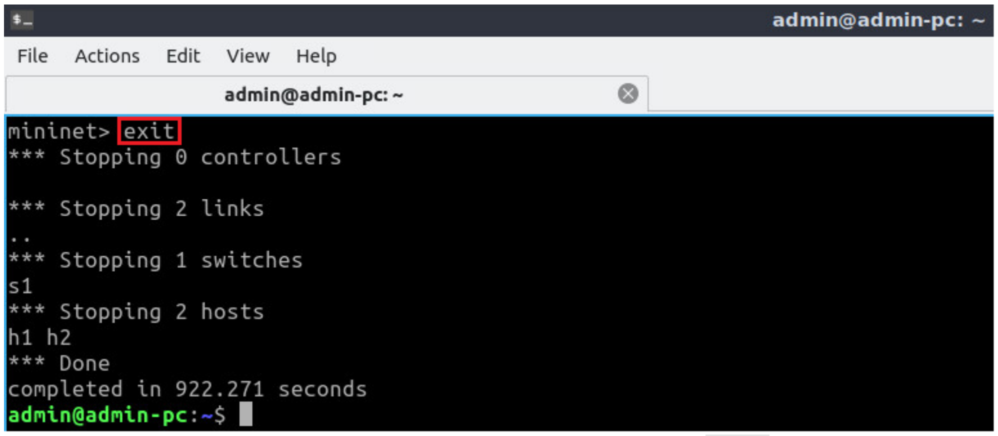
Step 1. On the CLI, type `h1 ping 10.0.0.2`. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test. Host h1 (10.0.0.1) sent four packets to host h2 (10.0.0.2) and successfully received the expected responses.



```
mininet> h1 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=1.15 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.074 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 57ms
rtt min/avg/max/mdev = 0.072/0.342/1.150/0.466 ms
mininet> 
```

Figure 10. Connectivity test between host h1 and host h2.

Step 2. Stop the emulation by typing `exit`.



```
mininet> exit
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 922.271 seconds
admin@admin-pc:~$ 
```

Figure 11. Stopping the emulation using `exit`.

The command `sudo mn -c` is often used on the Linux terminal (not on the Mininet CLI) to clean a previous instance of Mininet (e.g., after a crash).

3 Building and emulating a network in Mininet using the GUI

In this section, you will use the application MiniEdit[5] to deploy the topology illustrated below. MiniEdit is a simple GUI network editor for Mininet.

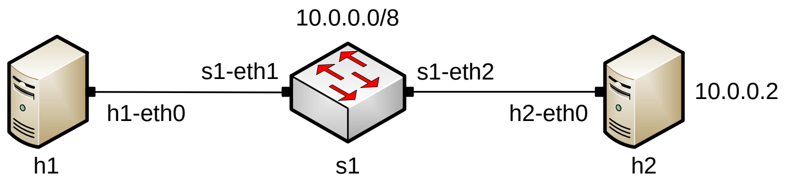


Figure 12. Lab topology.

3.1 Building the network 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`.

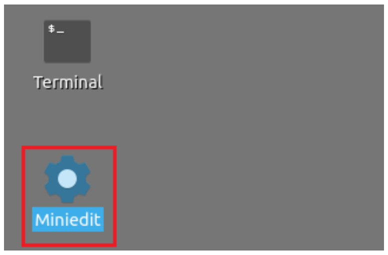


Figure 13. MiniEdit Desktop shortcut.

MiniEdit will start, as illustrated below.

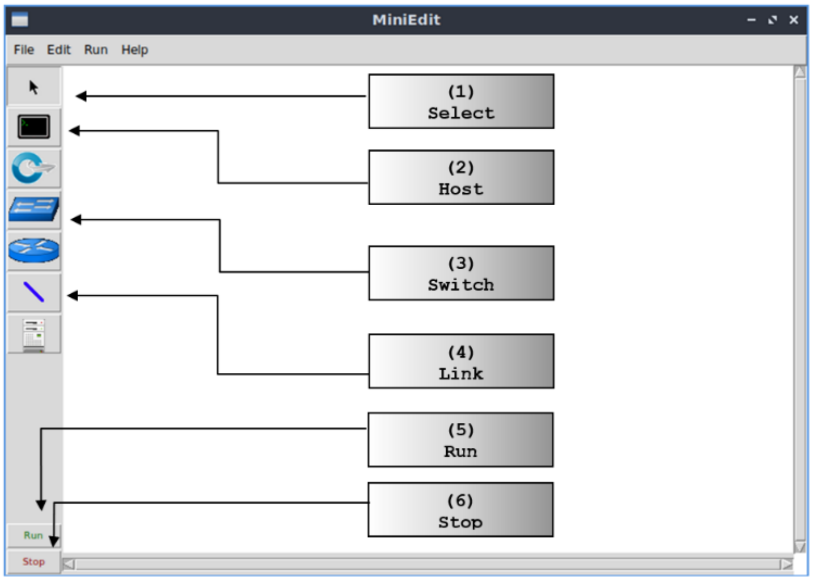


Figure 14. MiniEdit Graphical User Interface (GUI).

The main buttons are:

1. *Select*: allows selection/movement of the devices. Pressing *Del* on the keyboard after selecting the device removes it from the topology.
2. *Host*: allows addition of a new host to the topology. After clicking this button, click anywhere in the blank canvas to insert a new host.
3. *Switch*: allows addition of a new switch to the topology. After clicking this button, click anywhere in the blank canvas to insert the switch.
4. *Link*: connects devices in the topology (mainly switches and hosts). After clicking this button, click on a device and drag to the second device to which the link is to be established.
5. *Run*: starts the emulation. After designing and configuring the topology, click the run button.
6. *Stop*: stops the emulation.

Step 2. To build the topology of Figure 12, two hosts and one switch must be deployed. Deploy these devices in MiniEdit, as shown below.

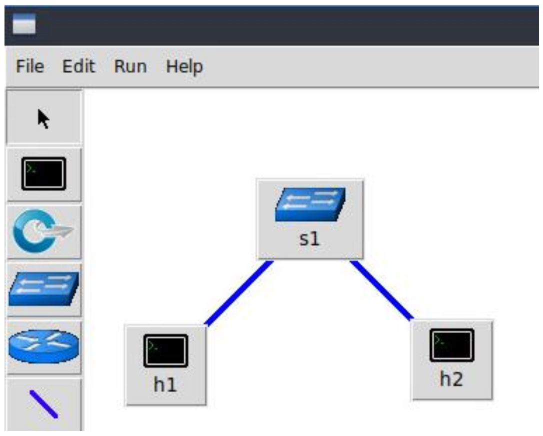
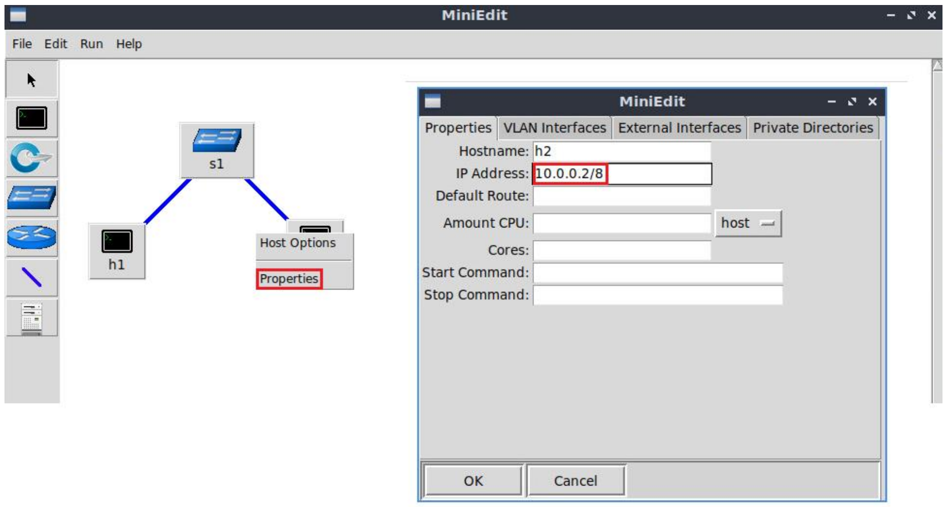


Figure 15. MiniEdit's topology.

Use the buttons described in the previous step to add and connect devices. The configuration of IP addresses is described in Step 3.

Step 3. Configure the IP addresses at host h1 and host h2. Host h1's IP address is 10.0.0.1/8 and host h2's IP address is 10.0.0.2/8. A host can be configured by holding the right click and selecting properties on the device. For example, host h2 is assigned the IP address 10.0.0.2/8 in the figure below.



Hostname: h2
IP Address: 10.0.0.2/8
Default Route:
Amount CPU: host
Cores:
Start Command:
Stop Command:

Figure 16. Configuration of a host's properties.

3.2 Testing connectivity

Before testing the connection between host h1 and host h2, the emulation must be started.

Step 1. Click on the *Run* button to start the emulation. The emulation will start and the buttons of the MiniEdit panel will gray out, indicating that they are currently disabled.

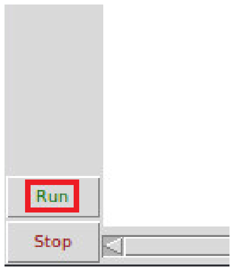


Figure 17. Starting the emulation.

Step 2. Open a terminal on host h1 by holding the right click on host h1 and selecting *Terminal*. This opens a terminal on host h1 and allows the execution of commands on the host h1. Repeat the procedure on host h2.

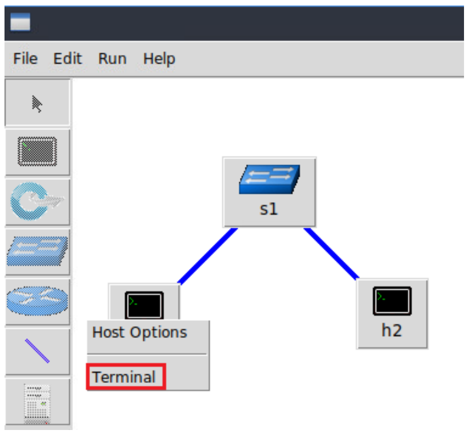
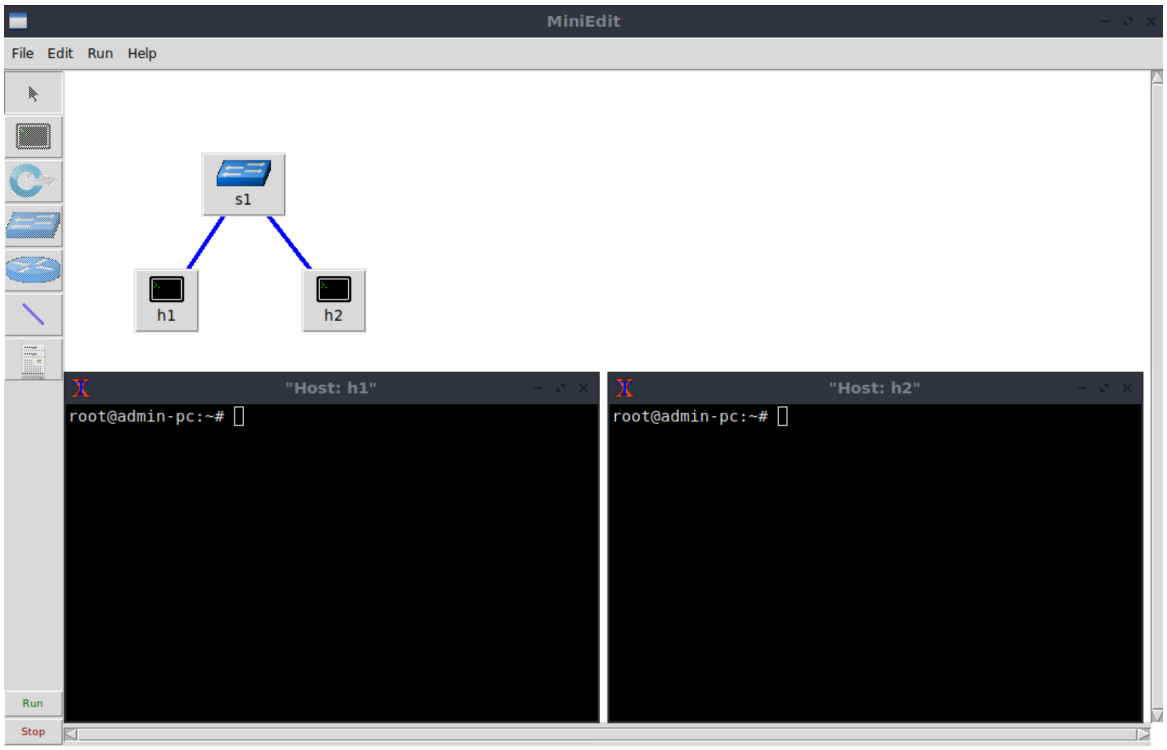


Figure 18. Opening a terminal on host h1.

The network and terminals at host h1 and host h2 will be available for testing.

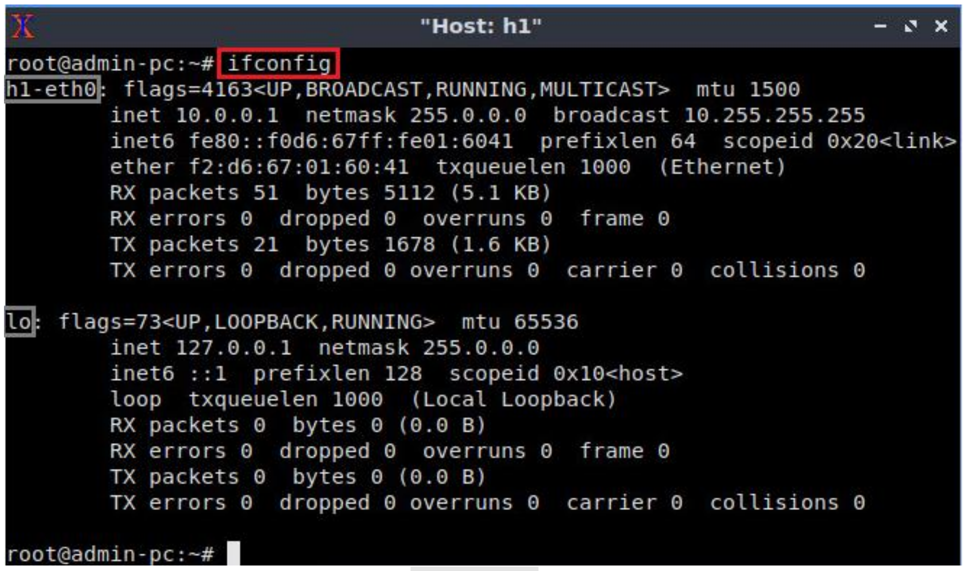


```
root@admin-pc:~#
```

```
root@admin-pc:~#
```

Figure 19. Terminals at host h1 and host h2.

Step 3. On host h1's terminal, type the command `ifconfig` to display its assigned IP addresses. The interface *h1-eth0* at host h1 should be configured with the IP address 10.0.0.1 and subnet mask 255.0.0.0.

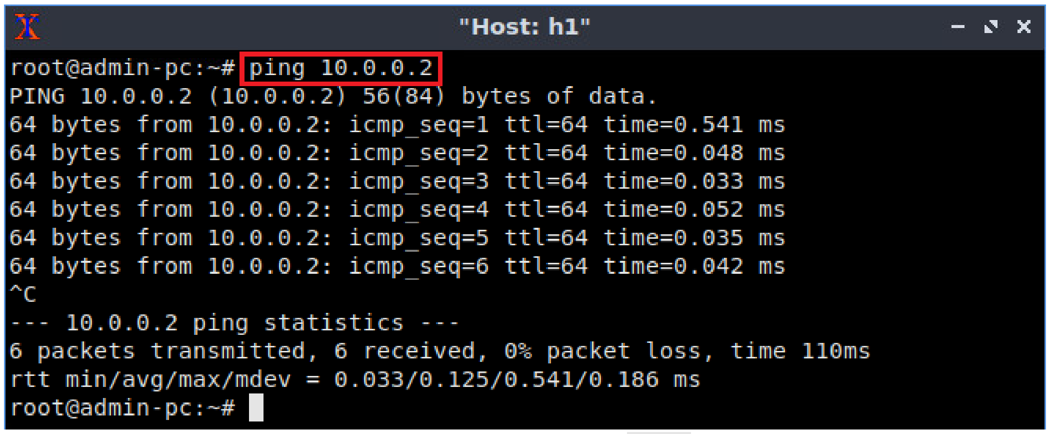


```
root@admin-pc:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::f0d6:67ff:fe01:6041  prefixlen 64  scopeid 0x20<link>
        ether f2:d6:67:01:60:41  txqueuelen 1000  (Ethernet)
        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 20. Output of `ifconfig` command on host h1.

Repeat Step 3 on host h2. Its interface *h2-eth0* should be configured with IP address 10.0.0.2 and subnet mask 255.0.0.0.

Step 4. On host h1's terminal, 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 (10.0.0.1) sent six packets to host h2 (10.0.0.2) and successfully received the expected responses.



```
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=0.541 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.033 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.042 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 110ms
rtt min/avg/max/mdev = 0.033/0.125/0.541/0.186 ms
root@admin-pc:~# 
```

Figure 21. Connectivity test using `ping` command.

Step 5. Stop the emulation by clicking on the *Stop* button.

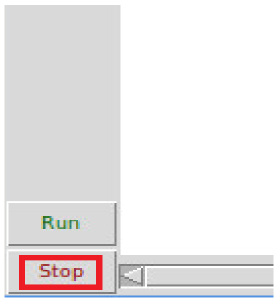
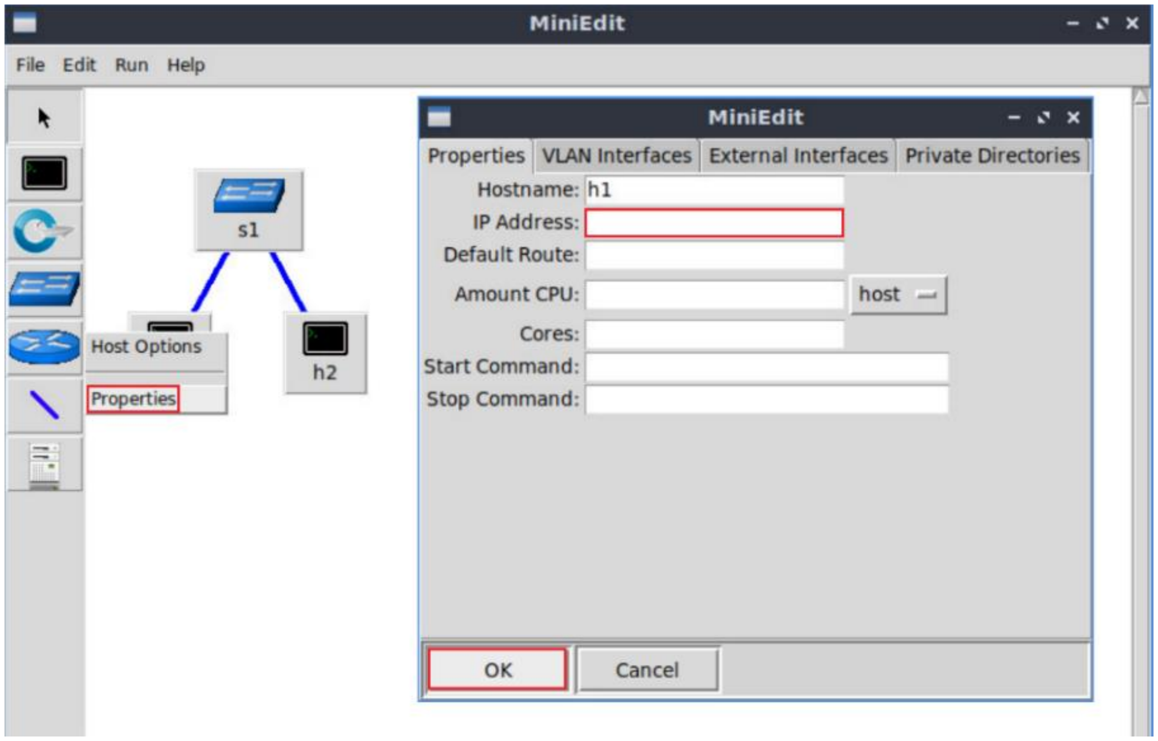


Figure 22. Stopping the emulation.

3.3 Automatic assignment of IP addresses

In the previous section, you manually assigned IP addresses to host h1 and host h2. An alternative is to rely on Mininet for an automatic assignment of IP addresses (by default, Mininet uses automatic assignment), which is described in this section.

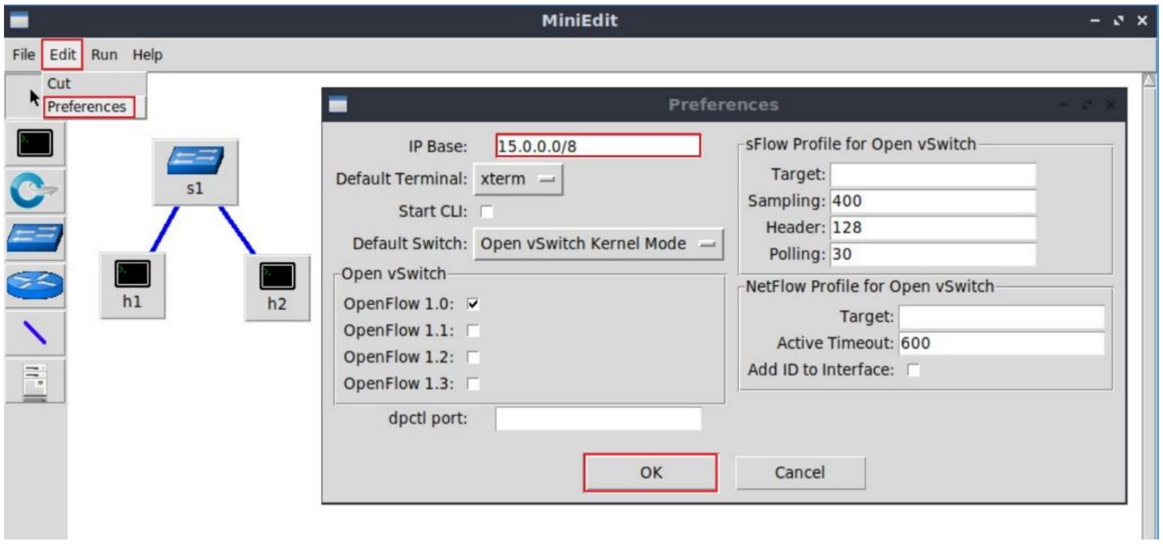
Step 1. Remove the manually assigned IP address from host h1. Hold right click on host h1, *Properties*. Delete the IP address, leaving it unassigned, and press the *OK* button as shown below. Repeat the procedure on host h2.



Hostname: h1
IP Address:
Default Route:
Amount CPU: host
Cores:
Start Command:
Stop Command:

Figure 23. Host h1 properties.

Step 2. Click on *Edit*, *Preferences* button. The default IP base is 10.0.0.0/8. Modify this value to 15.0.0.0/8, and then press the *OK* button.



IP Base: 15.0.0.0/8
Default Terminal: xterm
Start CLI: ☐
Default Switch: Open vSwitch Kernel Mode
Open vSwitch
OpenFlow 1.0: ✓
OpenFlow 1.1: ☐
OpenFlow 1.2: ☐
OpenFlow 1.3: ☐
dpctl port:
sFlow Profile for Open vSwitch
Target:
Sampling: 400
Header: 128
Polling: 30
NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface: ☐

Figure 24. Modification of the IP Base (network address and prefix length).

Step 3. Run the emulation again by clicking on the *Run* button. The emulation will start and the buttons of the MiniEdit panel will be disabled.

Step 4. Open a terminal on host h1 by holding the right click on host h1 and selecting Terminal.

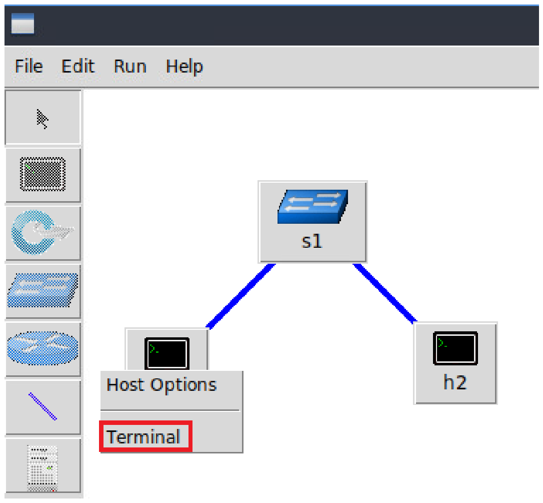
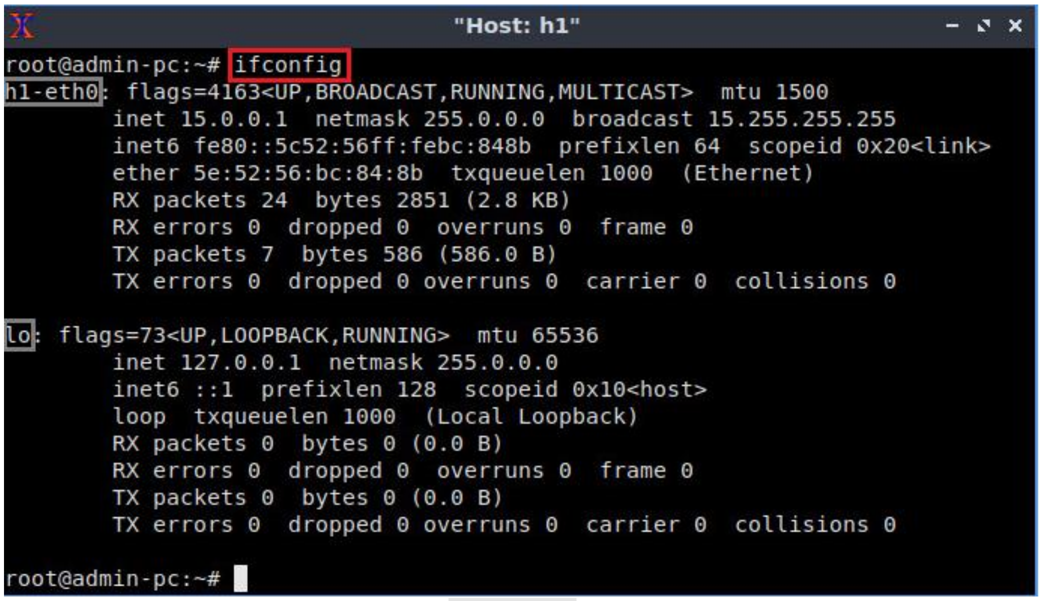


Figure 25. Opening a terminal on host h1.

Step 5. Type the command `ifconfig` to display the IP addresses assigned to host h1. The interface *h1-eth0* at host h1 now has the IP address 15.0.0.1 and subnet mask 255.0.0.0.



```
root@admin-pc:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 15.0.0.1  netmask 255.0.0.0  broadcast 15.255.255.255
        inet6 fe80::5c52:56ff:febc:848b  prefixlen 64  scopeid 0x20<link>
        ether 5e:52:56:bc:84:8b  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 26. Output of `ifconfig` command on host h1.

You can also verify the IP address assigned to host h2 by repeating Steps 4 and 5 on host h2's terminal. The corresponding interface *h2-eth0* at host h2 has now the IP address 15.0.0.2 and subnet mask 255.0.0.0.

Step 6. Stop the emulation by clicking on *Stop* button.

3.4 Saving and loading a Mininet topology

It is often useful to save the network topology, particularly when its complexity increases. MiniEdit enables you to save the topology to a file.

Step 1. To save your topology, click on *File* then *Save*. Provide a name for the topology and save on your machine.

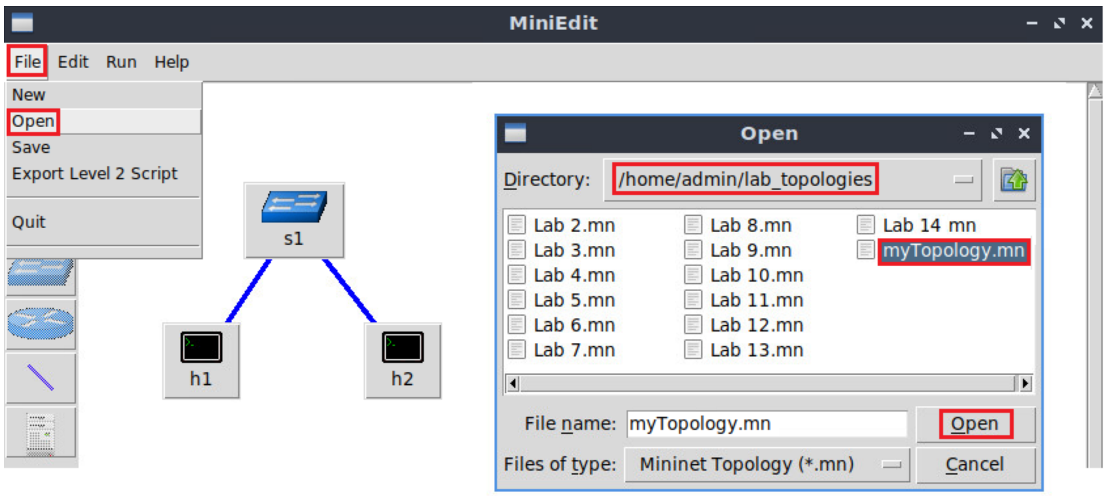


Figure 27. Saving the topology.

Step 2. To load the topology, click on *File* then *Open*. Locate the topology file and click on *Open*. The topology will be loaded again to MiniEdit.

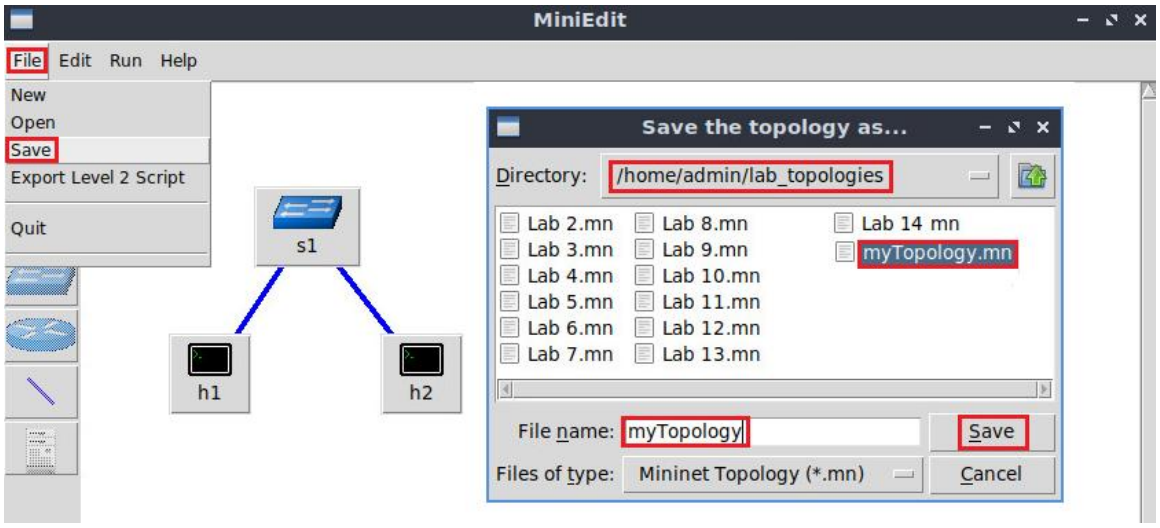


Figure 28. Opening a topology.

The upcoming labs' topologies are already built and stored in the folder */home/admin/lab_topologies* located in the Client's home directory. The *Open* dialog is used to avoid manually rebuilding each lab's topology.

This concludes Lab 1. Stop the emulation and then exit out of MiniEdit and Linux terminal.

References

1. Mininet walkthrough. [Online]. Available: http://Mininet.org.

2. N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, p. 69, 2008.
3. J. Esch, "Prolog to, software-defined networking: a comprehensive survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 10–13, 2015.
4. P. Dordal, "An Introduction to computer networks," [Online]. Available: https://intronetworks.cs.luc.edu/.
5. B. Lantz, G. Gee, "MiniEdit: a simple network editor for Mininet," 2013. [Online]. Available: https://github.com/Mininet/Mininet/blob/master/examples.

NETWORK TOOLS AND PROTOCOLS

Lab 2: Introduction to Iperf3

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 iPerf 3
2. Lab topology 4
 2.1 Starting host h1 and host h2 6
3. Using iPerf3 (client and server commands) 6
 3.1 Starting client and server 7
 3.2 Setting transmitting time period 8
 3.3 Setting time interval 9
 3.4 Changing the number of bytes to transmit 10
 3.5 Specifying the transport-layer protocol 11
 3.6 Changing port number 13
 3.7 Export results to JSON file 13
 3.8 Handle one client 14
4. Plotting iPerf3 results 15
References 17

Overview

This lab briefly introduces iPerf3 and explains how it can be used to measure and test network throughput in a designed network topology. It demonstrates how to invoke both client-side and server-side options from the command line utility.

Objectives

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

1. Understand throughput and how it differs from bandwidth in network systems.
2. Create iPerf3 tests with various settings on a designed network topology.
3. Understand and analyze iPerf3's test output.
4. Visualize iPerf3's output using a custom plotting script.

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 iPerf3.
2. Section 2: Lab topology.
3. Section 3: Using iPerf3 (client and server commands).
4. Section 4: Plotting iPerf3's results.

1 Introduction to iPerf

Bandwidth is a physical property of a transmission media that depends on factors such as the construction and length of wire or fiber. To network engineers, bandwidth is the maximum data rate of a channel, a quantity measured in bits per second (bps)[1]. Having a high-bandwidth link does not always guarantee high network performance. In fact, several factors may affect the performance such as latency, packet loss, jitter, and others.

In the context of a communication session between two end devices along a network path, *throughput* is the rate in bps at which the sending process can deliver bits to the receiving process. Because other sessions will be sharing the bandwidth along the network path, and because these other sessions will recur, the available throughput can fluctuate with time[2]. Note, however, that sometimes the terms throughput and bandwidth are used interchangeably.

iPerf3 is a real-time network throughput measurement tool. It is an open source, cross-platform client-server application that can be used to measure the throughput between the two end devices. A typical iPerf3 output contains a timestamped report of the amount of data transferred and the throughput measured.

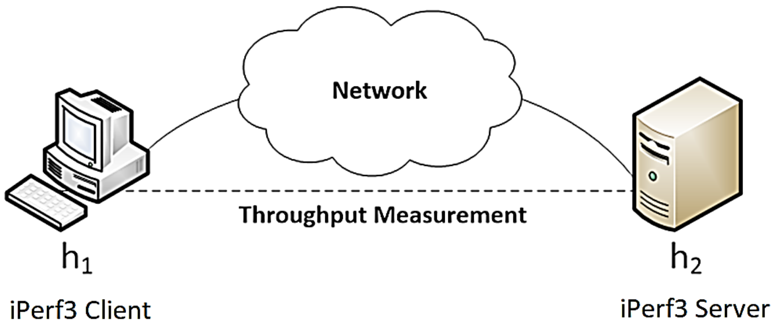


Figure 1. Throughput measurement with iPerf3.

Measuring throughput is particularly useful when experiencing network bandwidth issues such as delay, packet loss, etc. iPerf3 can operate on Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Stream Control Transmission Protocol (SCTP).

In iPerf3, the user can set *client* and *server* configurations via options and parameters and can create data flows to measure the throughput between the two end hosts in a unidirectional or bidirectional way. iPerf3 outputs a timestamped report of the amount of data transferred and the throughput measured[3].

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.

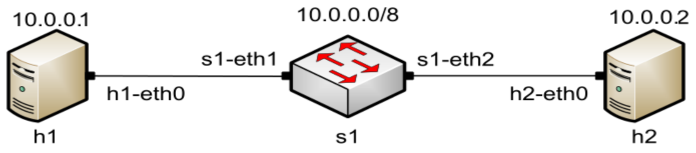


Figure 2. Mininet’s default minimal topology.

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`.

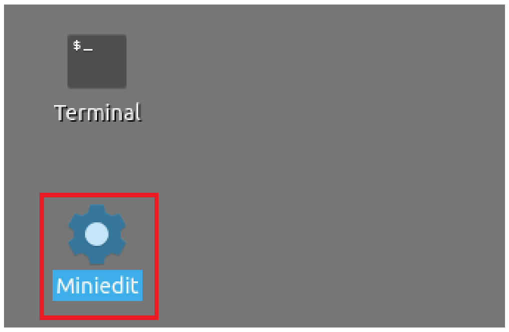


Figure 3. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 2.mn* topology file in the default directory, */home/admin/lab_topologies*, and click on *Open*.

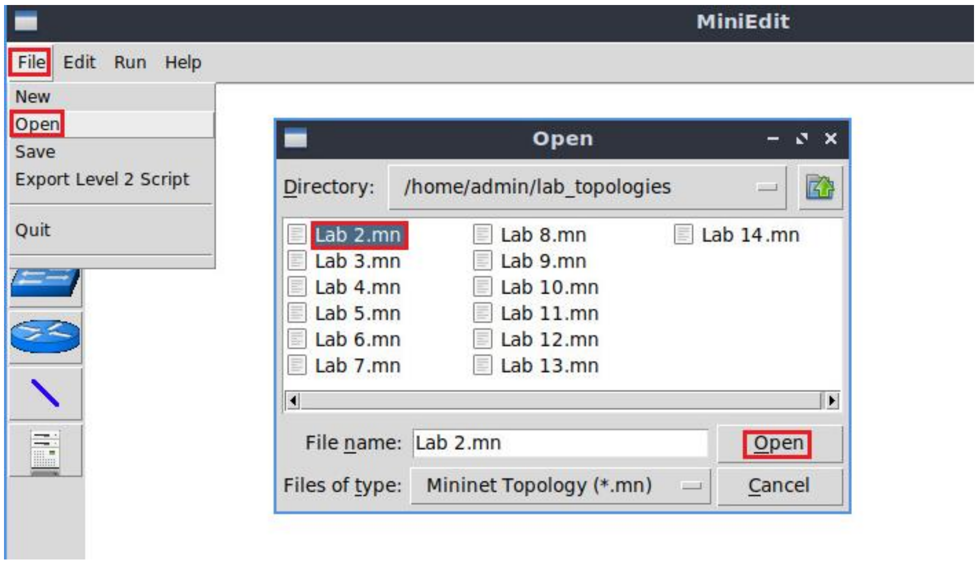


Figure 4. MiniEdit's *Open* dialog.

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.

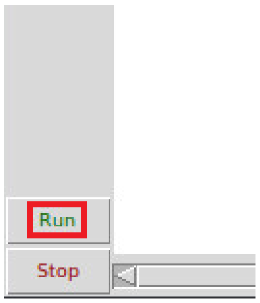


Figure 5. Running the emulation.

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

2.1 Starting host h1 and host h2

Step 1. Hold the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

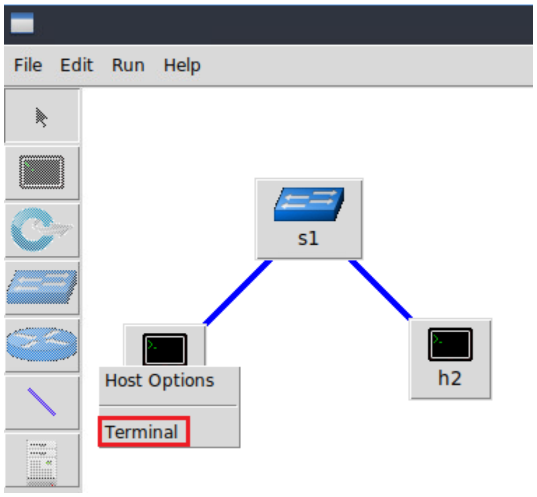
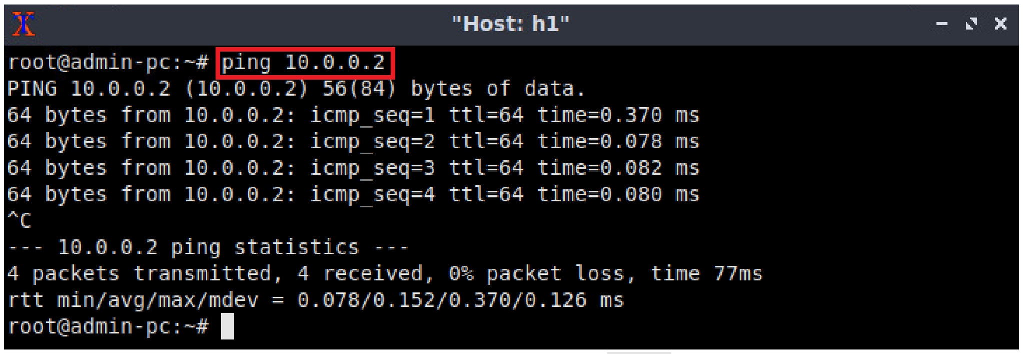


Figure 6. Opening a terminal on host h1.

Step 2. 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.



```
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=0.370 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.080 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 77ms
rtt min/avg/max/mdev = 0.078/0.152/0.370/0.126 ms
root@admin-pc:~#
```

Figure 7. Connectivity test using `ping` command.

The figure above indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

3 Using iPerf3 (client and server commands)

Since the initial setup and configuration are done, it is time to start a simple throughput measurement. The user interacts with iPerf3 using the `iperf3` command. The basic `iperf3` syntax used on both the client and the server is as follows:

```
iperf3 [-s|-c] [ options ]
```

3.1 Starting client and server

Step 1. Hold the right-click on host h2 and select *Terminal*. This opens the terminal of host h2 and allows the execution of commands on that host.

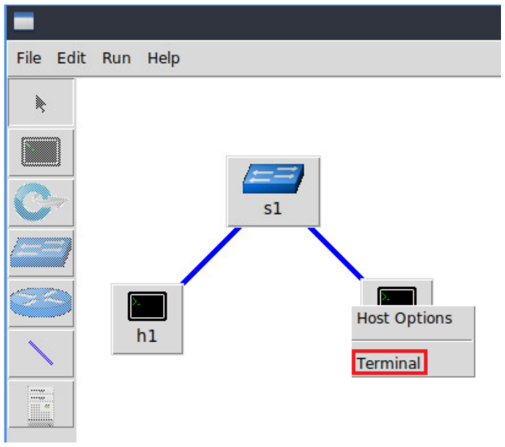
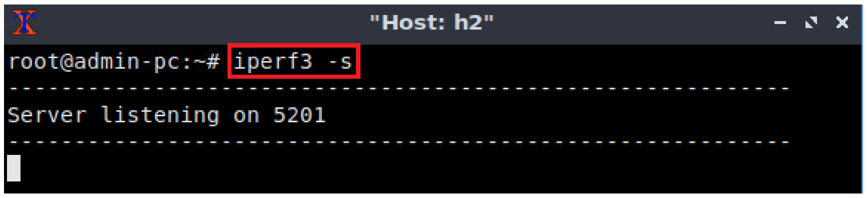


Figure 8. Opening a terminal on host h2.

Step 2. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in the figure below:

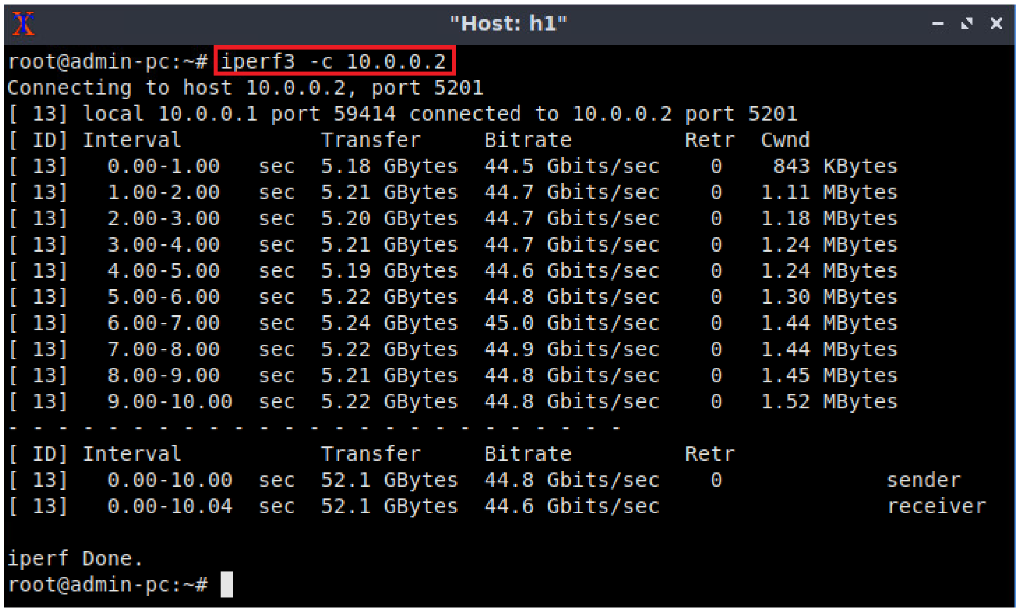
```
iperf3 -s
```


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```
Figure 9. Host h2 running iPerf3 server.

The parameter `-s` in the command above indicates that the host is configured as a server. Now, the server is listening on port 5201 waiting for incoming connections.

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 in the figure below:

```
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
[ 13] local 10.0.0.1 port 59414 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 13]   0.00-1.00   sec  5.18 GBytes  44.5 Gbits/sec    0    843 KBytes
[ 13]   1.00-2.00   sec  5.21 GBytes  44.7 Gbits/sec    0   1.11 MBytes
[ 13]   2.00-3.00   sec  5.20 GBytes  44.7 Gbits/sec    0   1.18 MBytes
[ 13]   3.00-4.00   sec  5.21 GBytes  44.7 Gbits/sec    0   1.24 MBytes
[ 13]   4.00-5.00   sec  5.19 GBytes  44.6 Gbits/sec    0   1.24 MBytes
[ 13]   5.00-6.00   sec  5.22 GBytes  44.8 Gbits/sec    0   1.30 MBytes
[ 13]   6.00-7.00   sec  5.24 GBytes  45.0 Gbits/sec    0   1.44 MBytes
[ 13]   7.00-8.00   sec  5.22 GBytes  44.9 Gbits/sec    0   1.44 MBytes
[ 13]   8.00-9.00   sec  5.21 GBytes  44.8 Gbits/sec    0   1.45 MBytes
[ 13]   9.00-10.00  sec  5.22 GBytes  44.8 Gbits/sec    0   1.52 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 13]   0.00-10.00  sec  52.1 GBytes  44.8 Gbits/sec    0             sender
[ 13]   0.00-10.04  sec  52.1 GBytes  44.6 Gbits/sec                  receiver

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

Figure 10. Host h1 running iPerf3 as client.

The parameter `-c` in command above indicates that host h1 is configured as a client. The parameter 10.0.0.2 is the server's (host h2) IP address. Once the test is completed, a summary report on both the client and the server is displayed containing the following data:

- *ID*: identification number of the connection.
- *Interval*: time interval to periodically report throughput. By default, the time interval is 1 second.
- *Transfer:* how much data was transferred in each time interval.
- *Bitrate:* the measured throughput in in each time interval.
- *Retr:* the number of TCP segments retransmitted in each time interval. This field increases when TCP segments are lost in the network due to congestion or corruption.
- *Cwnd:* indicates the congestion windows size in each time interval. TCP uses this variable to limit the amount of data the TCP client can send before receiving the acknowledgement of the sent data.

The summarized data, which starts after the last dashed line, shows the total amount of transferred data is 52.1 Gbyte and the throughput 44.8 Gbps.

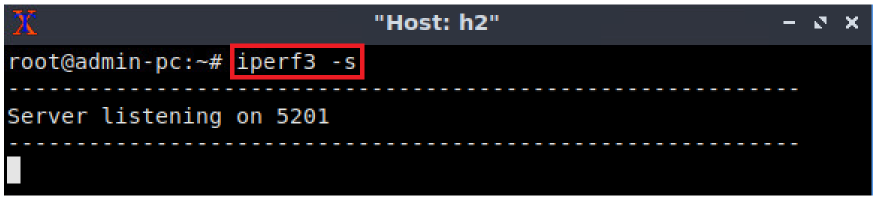
Step 4. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too. The summarized data on the server is similar to that of the client side's and must be interpreted in the same way.

3.2 Setting transmitting time period

Setting the transmission time period is configured solely on the client. To change the default transmission time, apply the following steps:

Step 1. Start the iPerf3 server on host h2.

```
iperf3 -s
```

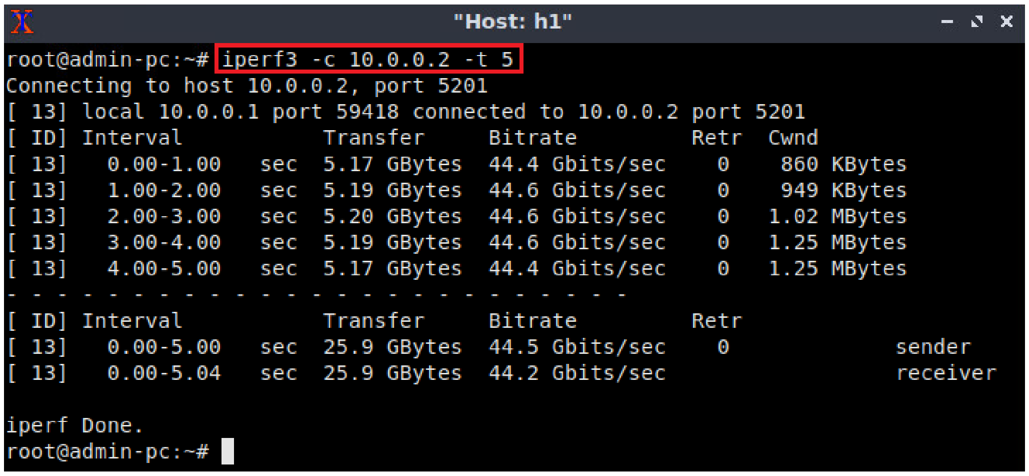


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 11. Host h2 running iPerf3 as server.

Step 2. Start the iPerf3 client with the `-t` option followed by the number of seconds.

```
iperf3 -c 10.0.0.2 -t 5
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 5
Connecting to host 10.0.0.2, port 5201
[ 13] local 10.0.0.1 port 59418 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 13]   0.00-1.00   sec  5.17 GBytes  44.4 Gbits/sec    0    860 KBytes
[ 13]   1.00-2.00   sec  5.19 GBytes  44.6 Gbits/sec    0    949 KBytes
[ 13]   2.00-3.00   sec  5.20 GBytes  44.6 Gbits/sec    0   1.02 MBytes
[ 13]   3.00-4.00   sec  5.19 GBytes  44.6 Gbits/sec    0   1.25 MBytes
[ 13]   4.00-5.00   sec  5.17 GBytes  44.4 Gbits/sec    0   1.25 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 13]   0.00-5.00   sec  25.9 GBytes  44.5 Gbits/sec    0             sender
[ 13]   0.00-5.04   sec  25.9 GBytes  44.2 Gbits/sec                  receiver

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

Figure 12. Host h1 transmitting for 5 seconds.

The above command starts an iPerf3 client for a 5-second time period transmitting at an average rate of 44.5 Gbps.

Step 3. 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.

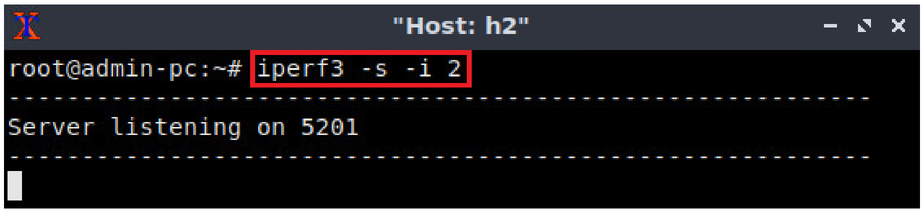
3.3 Setting time interval

In this test, the user will configure the client to perform a throughput test with 2-seconds reporting time interval on both the client and the server. Note the default 1-second interval period in Figure 12.

The `-i` option allows setting the reporting interval time in seconds. In this case the value should be set to 2 seconds on both the client and the server**.**

Step 1. Setting the interval value on the server (host h2's terminal):

```
iperf3 -s -i 2
```

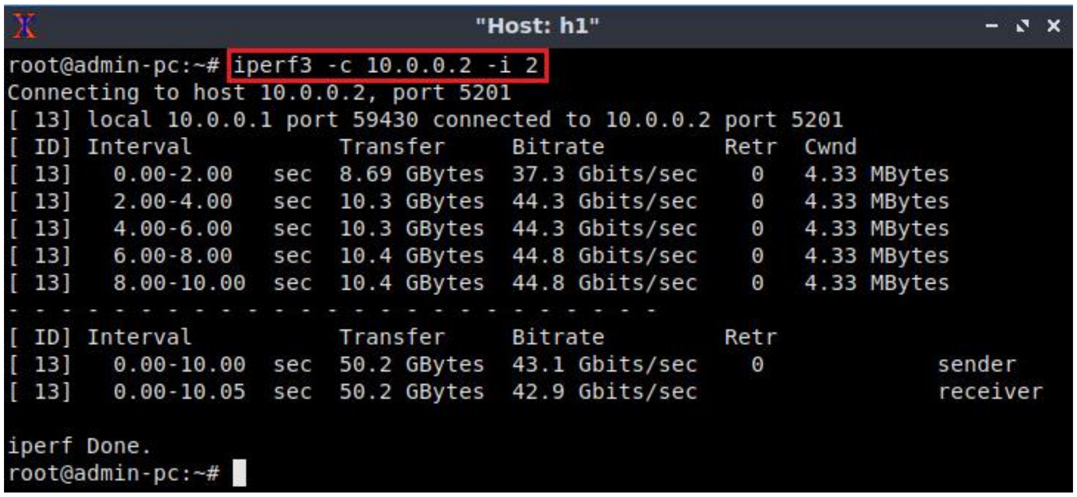


```
root@admin-pc:~# iperf3 -s -i 2
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 13. Host h2 running iPerf3 as server.

Step 2. Setting the interval value on the client (host h1's terminal):

```
iperf3 -c 10.0.0.2 -i 2
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2 -i 2
Connecting to host 10.0.0.2, port 5201
[ 13] local 10.0.0.1 port 59430 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 13]   0.00-2.00   sec  8.69 GBytes  37.3 Gbits/sec    0   4.33 MBytes
[ 13]   2.00-4.00   sec  10.3 GBytes  44.3 Gbits/sec    0   4.33 MBytes
[ 13]   4.00-6.00   sec  10.3 GBytes  44.3 Gbits/sec    0   4.33 MBytes
[ 13]   6.00-8.00   sec  10.4 GBytes  44.8 Gbits/sec    0   4.33 MBytes
[ 13]   8.00-10.00  sec  10.4 GBytes  44.8 Gbits/sec    0   4.33 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 13]   0.00-10.00  sec  50.2 GBytes  43.1 Gbits/sec    0             sender
[ 13]   0.00-10.05  sec  50.2 GBytes  42.9 Gbits/sec                  receiver

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

Figure 14. Host h1 and host h2 reporting every 2 seconds.

Note that the `-i` option can be specified differently on the client and the server. For example, if the `-i` option is specified with the value 3 on the client only, then the client will be reporting every 3 seconds while the server will be reporting every second (the default `-i` value).

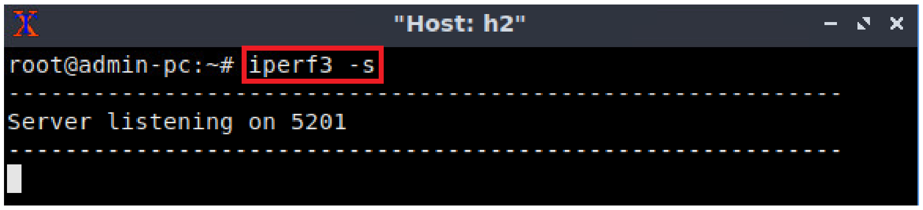
Step 3. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3.4 Changing the number of bytes to transmit

In this test, the client is configured to send a specific amount of data by setting the number of bytes to transmit. By default, iPerf3 performs the throughput measurement for 10 seconds. However, with this configuration, the client will keep sending packets until all the bytes specified by the user were sent.

Step 1. Type the following command on host h2's terminal to start the iPerf3 server.

```
iperf3 -s
```



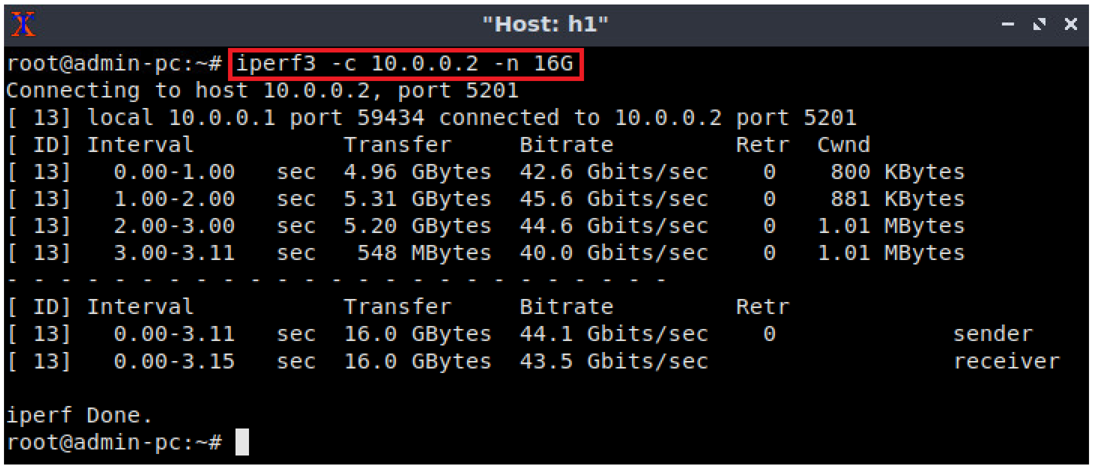
```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 15. Host h2 running iPerf3 as server.

Step 2. This configuration is only set on the client (host h1's terminal) using the `-n` option as follows:

```
iperf3 -c 10.0.0.2 -n 16G
```

The `-n` option in the above command indicates the amount of data to transmit: 16 Gbytes. The user can specify other scale values, for example, `16M` is used to send 16 Mbytes.



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -n 16G
Connecting to host 10.0.0.2, port 5201
[ 13] local 10.0.0.1 port 59434 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 13]   0.00-1.00   sec  4.96 GBytes  42.6 Gbits/sec    0    800 KBytes
[ 13]   1.00-2.00   sec  5.31 GBytes  45.6 Gbits/sec    0    881 KBytes
[ 13]   2.00-3.00   sec  5.20 GBytes  44.6 Gbits/sec    0   1.01 MBytes
[ 13]   3.00-3.11   sec   548 MBytes  40.0 Gbits/sec    0   1.01 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 13]   0.00-3.11   sec  16.0 GBytes  44.1 Gbits/sec    0             sender
[ 13]   0.00-3.15   sec  16.0 GBytes  43.5 Gbits/sec                  receiver

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

Figure 16. Host h1 sending 16 Gbps of data.

Note the total time spent for sending the 16 Gbytes of data is 3.11 seconds and not the default transmitting time used by iPerf3 (10 seconds).

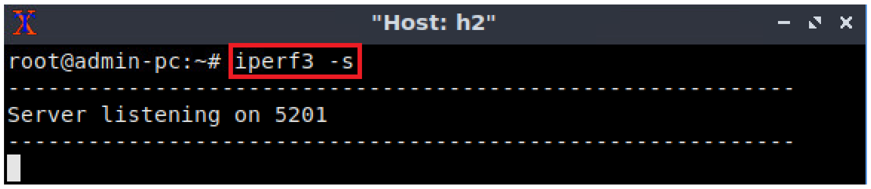
Step 3. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3.5 Specifying the transport-layer protocol

So far, the throughput measurements were conducted on the TCP protocol, which is the default configuration protocol. In order to change the protocol to UDP, the user must invoke the option `-u` on the client side. Similarly, the option `--sctp` is used for the SCTP protocol. iPerf3 automatically detects the transport-layer protocol on the server side.

Step 1. Start the iPerf3 server on host h2.

```
iperf3 -s
```

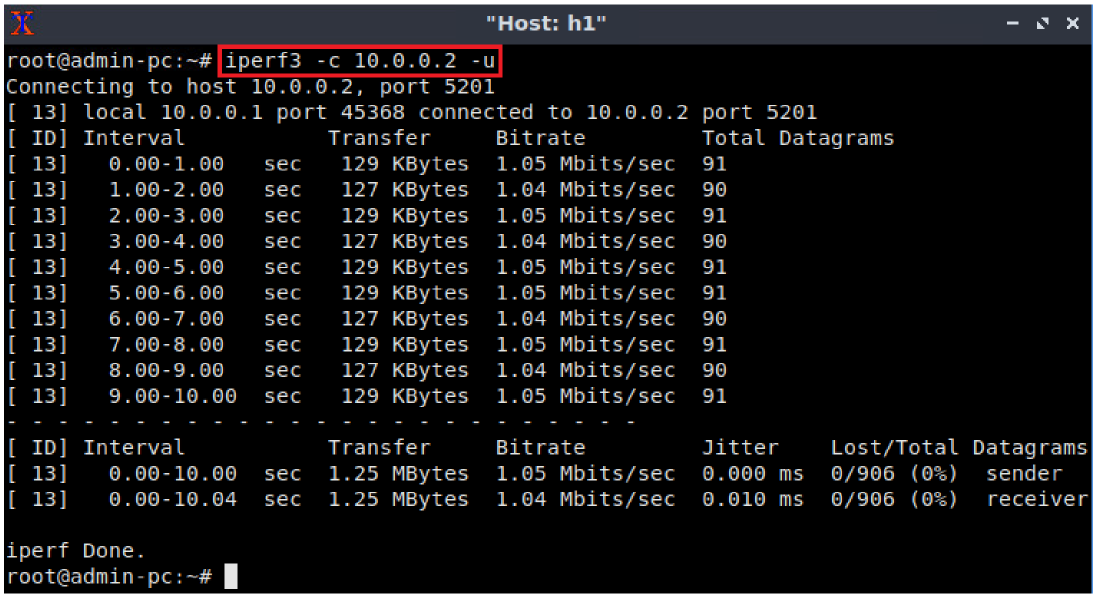


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 17. Host h2 running iPerf3 as server.

Step 2. Specify UDP as the transport-layer protocol using the `-u` option as follows.

```
iperf3 -c 10.0.0.2 -u
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -u
Connecting to host 10.0.0.2, port 5201
[ 13] local 10.0.0.1 port 45368 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[ 13]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91
[ 13]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  90
[ 13]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91
[ 13]   3.00-4.00   sec   127 KBytes  1.04 Mbits/sec  90
[ 13]   4.00-5.00   sec   129 KBytes  1.05 Mbits/sec  91
[ 13]   5.00-6.00   sec   129 KBytes  1.05 Mbits/sec  91
[ 13]   6.00-7.00   sec   127 KBytes  1.04 Mbits/sec  90
[ 13]   7.00-8.00   sec   129 KBytes  1.05 Mbits/sec  91
[ 13]   8.00-9.00   sec   127 KBytes  1.04 Mbits/sec  90
[ 13]   9.00-10.00  sec   129 KBytes  1.05 Mbits/sec  91
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[ 13]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/906 (0%)  sender
[ 13]   0.00-10.04  sec  1.25 MBytes  1.04 Mbits/sec  0.010 ms  0/906 (0%)  receiver

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

Figure 18. Host h1 sending UDP datagrams.

Once the test is completed, it will show the following summarized data:

- *ID, Interval, Transfer, Bitrate*: Same as TCP.
- *Jitter*: the difference in packet delay.
- *Lost/Total*: indicates the number of lost datagrams over the total number sent to the server (and percentage).

After the dashed lines, the summary is displayed, showing the total amount of transferred data (1.25 Mbytes) and the maximum achieved bandwidth (1.05 Mbps), over a time period of 10 seconds. The Jitter, which indicates in milliseconds (ms) the variance of time delay between data packets over a network, has a value of 0.010ms. Finally, the lost datagrams value is 0 (zero) and the total datagram which the server has received was 906, and thus, the loss rate is 0%. These values are reported on the server as well.

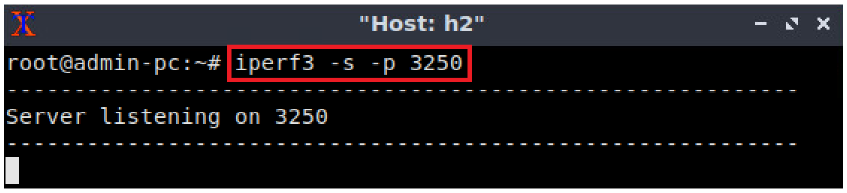
Step 3. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3.6 Changing port number

If the user wishes to measure throughput on a specific port, the `-p` option is used to configure both the client and the server to send/receive packets or datagrams on the specified port.

Step 1. Start the iPerf3 server on host h2. Use the `-p` option to specify the listening port.

```
iperf3 -s -p 3250
```

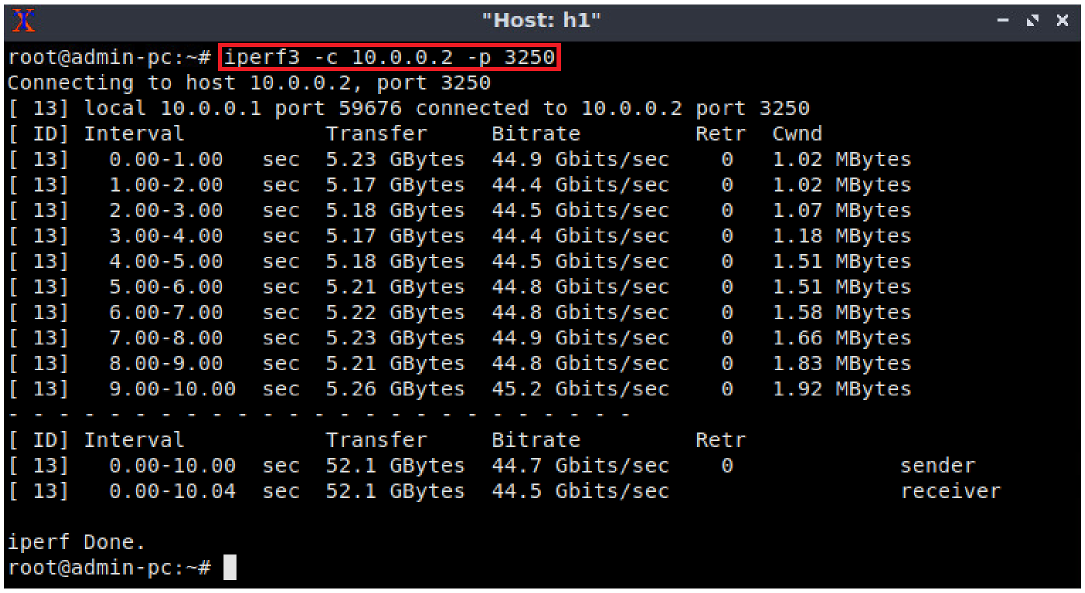


```
"Host: h2"
root@admin-pc:~# iperf3 -s -p 3250
-----------------------------------------------------------
Server listening on 3250
-----------------------------------------------------------
```

Figure 19. Host h2 running iPerf3 as server on port 3250.

Step 2. Start the iPerf3 client on host h1. Use the `-p` option to specify the server's listening port.

```
iperf3 -c 10.0.0.2 -p 3250
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -p 3250
Connecting to host 10.0.0.2, port 3250
[ 13] local 10.0.0.1 port 59676 connected to 10.0.0.2 port 3250
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 13]   0.00-1.00   sec  5.23 GBytes  44.9 Gbits/sec    0   1.02 MBytes
[ 13]   1.00-2.00   sec  5.17 GBytes  44.4 Gbits/sec    0   1.02 MBytes
[ 13]   2.00-3.00   sec  5.18 GBytes  44.5 Gbits/sec    0   1.07 MBytes
[ 13]   3.00-4.00   sec  5.17 GBytes  44.4 Gbits/sec    0   1.18 MBytes
[ 13]   4.00-5.00   sec  5.18 GBytes  44.5 Gbits/sec    0   1.51 MBytes
[ 13]   5.00-6.00   sec  5.21 GBytes  44.8 Gbits/sec    0   1.51 MBytes
[ 13]   6.00-7.00   sec  5.22 GBytes  44.8 Gbits/sec    0   1.58 MBytes
[ 13]   7.00-8.00   sec  5.23 GBytes  44.9 Gbits/sec    0   1.66 MBytes
[ 13]   8.00-9.00   sec  5.21 GBytes  44.8 Gbits/sec    0   1.83 MBytes
[ 13]   9.00-10.00  sec  5.26 GBytes  45.2 Gbits/sec    0   1.92 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 13]   0.00-10.00  sec  52.1 GBytes  44.7 Gbits/sec    0             sender
[ 13]   0.00-10.04  sec  52.1 GBytes  44.5 Gbits/sec                  receiver

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

Figure 20. Host h2 running on port 3250.

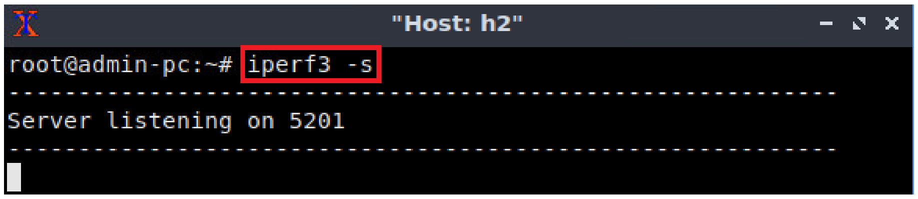
Step 3. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3.7 Export results to JSON file

JSON (JavaScript Object Notation) is a lightweight data-interchange format. iPerf3 allows exporting the test results to a JSON file, which makes it easy for other applications to parse the file and interpret the results (e.g. plot the results).

Step 1. Start the iPerf3 server on host h2.

```
iperf3 -s
```

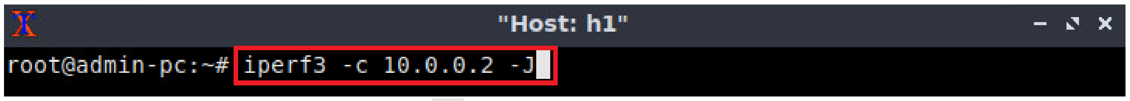


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 21. Host h2 running iPerf3 as server.

Step 2. Start the iPerf3 client on host h1. Specify the `-J` option to display the output in JSON format.

```
iperf3 -c 10.0.0.2 -J
```

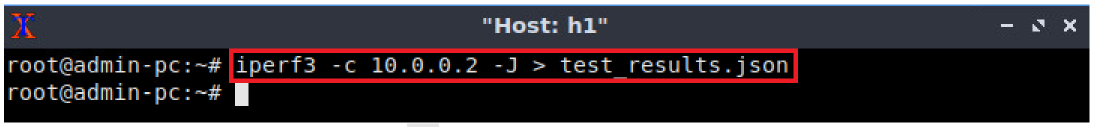


```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -J
```

Figure 22. Host h1 using `-J` to output JSON to standard output (*stdout*).

The `-J` option outputs JSON text to the screen through standard output (*stdout*) after the test is done (10 seconds by default). It is often useful to export the output to a file that can be parsed later by other programs. This can be done by redirecting the standard output to a file using the redirection operator in Linux `>`.

```
iperf3 -c 10.0.0.2 -J > test_results.json
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -J > test_results.json
root@admin-pc:~#
```

Figure 23. Host h1 using `-J` to output JSON and redirecting *stdout* to file.

After creating the JSON file, the `ls` command is used to verify that the file is created. The `cat` command can be used to display the file's contents.

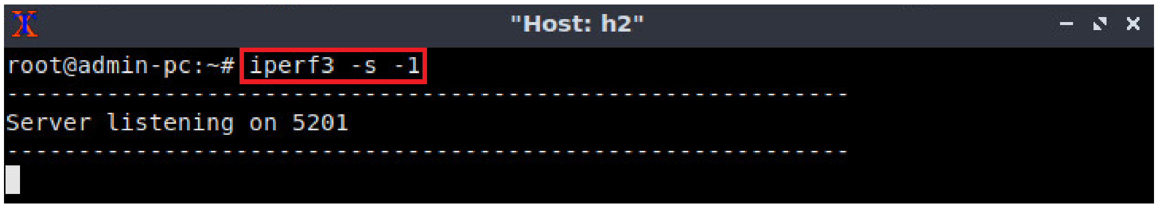
Step 3. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3.8 Handle one client

By default, an iPerf3 server keeps listening to incoming connections. To allow the server to handle one client and then stop, the `-1` option is added to the server.

Step 1. Start the iPerf3 server on host h2. Use the `-1` option to accept only one client.

```
iperf3 -s -1
```

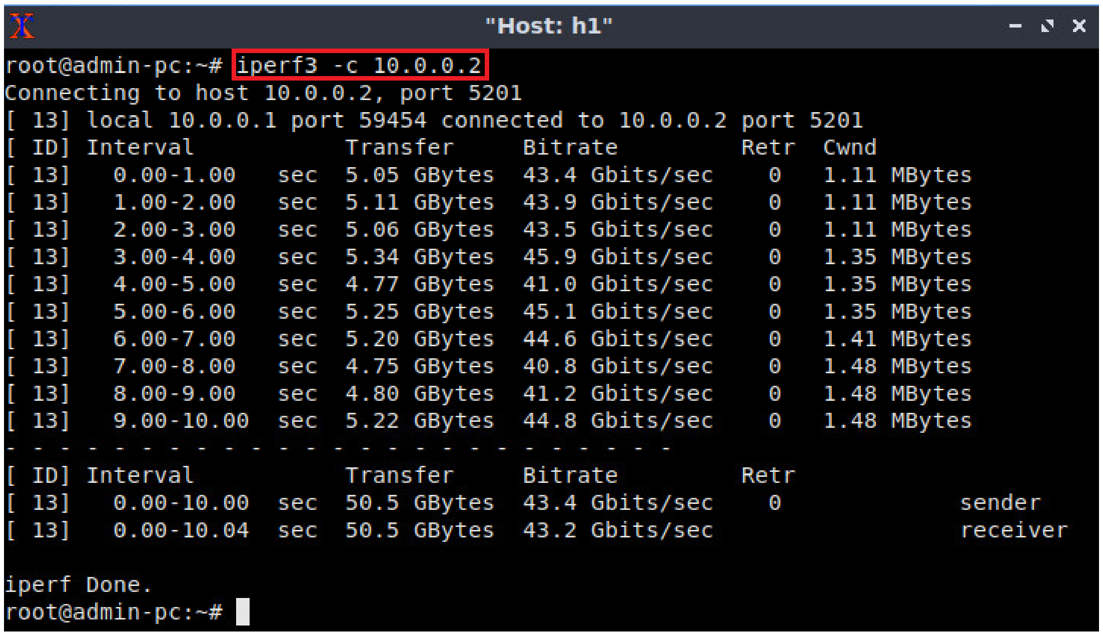


```
"Host: h2"
root@admin-pc:~# iperf3 -s -1
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 24. Host h2 running a server with one connection only.

Step 2. Start the iPerf3 client on host h1.

```
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
[ 13] local 10.0.0.1 port 59454 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 13]   0.00-1.00   sec  5.05 GBytes  43.4 Gbits/sec    0   1.11 MBytes
[ 13]   1.00-2.00   sec  5.11 GBytes  43.9 Gbits/sec    0   1.11 MBytes
[ 13]   2.00-3.00   sec  5.06 GBytes  43.5 Gbits/sec    0   1.11 MBytes
[ 13]   3.00-4.00   sec  5.34 GBytes  45.9 Gbits/sec    0   1.35 MBytes
[ 13]   4.00-5.00   sec  4.77 GBytes  41.0 Gbits/sec    0   1.35 MBytes
[ 13]   5.00-6.00   sec  5.25 GBytes  45.1 Gbits/sec    0   1.35 MBytes
[ 13]   6.00-7.00   sec  5.20 GBytes  44.6 Gbits/sec    0   1.41 MBytes
[ 13]   7.00-8.00   sec  4.75 GBytes  40.8 Gbits/sec    0   1.48 MBytes
[ 13]   8.00-9.00   sec  4.80 GBytes  41.2 Gbits/sec    0   1.48 MBytes
[ 13]   9.00-10.00  sec  5.22 GBytes  44.8 Gbits/sec    0   1.48 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 13]   0.00-10.00  sec  50.5 GBytes  43.4 Gbits/sec    0             sender
[ 13]   0.00-10.04  sec  50.5 GBytes  43.2 Gbits/sec                  receiver

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

Figure 25. Host h1 running an iPerf3 client.

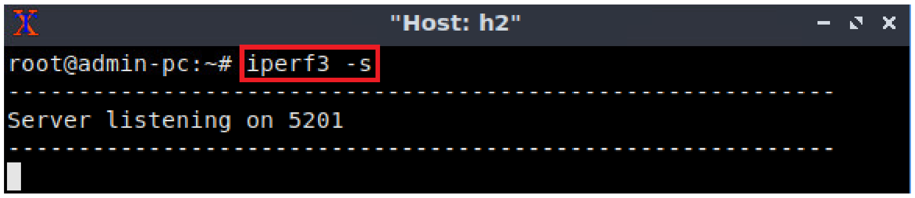
After this test is finished, the server stops immediately.

4 Plotting iPerf3 results

In section 3.7, iPerf3's result was exported to a JSON file to be processed by other applications. A script called `plot_iperf.sh` is installed and configured on the Client's machine. It accepts a JSON file as input and generates PDF files plotting several variables produced by iPerf3.

Step 1. Start the iPerf3 server on host h2.

```
iperf3 -s
```

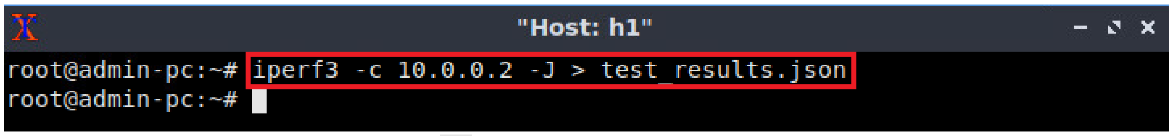


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 26. Host h2 running iPerf3 as server.

Step 2. Start the iPerf3 client on host h1. Specify the `-J` option to produce the output in JSON format and redirect the output to the file *test_results.json*. Any data previously stored in this file will be replaced with current output as the `>` operator is being used here.

```
iperf3 -c 10.0.0.2 -J > test_results.json
```

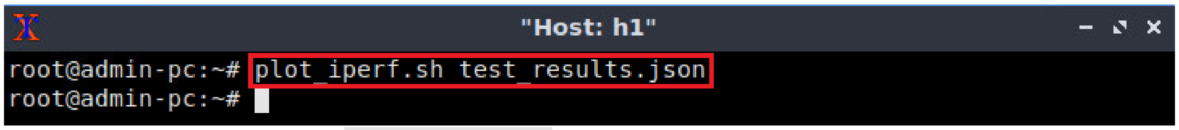


```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -J > test_results.json
root@admin-pc:~#
```

Figure 27. Host h1 using `-J` to output JSON and redirecting *stdout* to file.

Step 3. To generate the output for iPerf3's JSON file run the following command:

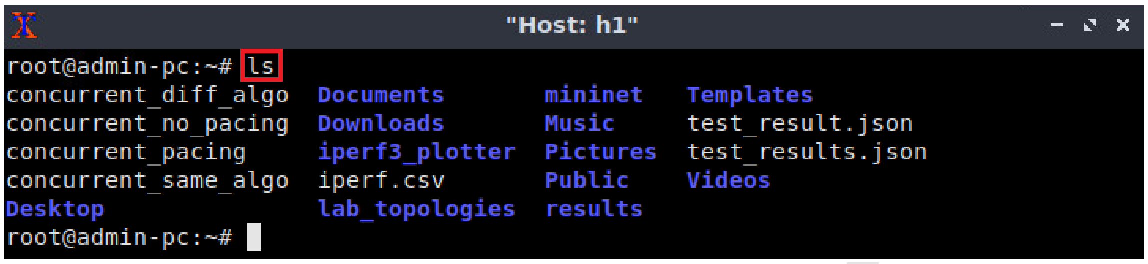
```
plot_iperf.sh test_results.json
```



```
"Host: h1"
root@admin-pc:~# plot_iperf.sh test_results.json
root@admin-pc:~#
```

Figure 28. `plot_iperf.sh` script generating output results.

This plotting script generates PDF files for the following fields: congestion window (*cwnd.pdf*), retransmits (*retransmits.pdf*), Round-Trip Time (*RTT.pdf*), Round-Trip Time variance (*RTT_Var.pdf*), throughput (*throughput.pdf*), maximum transmission unit (*MTU.pdf*), bytes transferred (*bytes.pdf*). The plotting script also generates a CSV file (1.dat) which can be used by other applications. These files are stored in a directory *results* created in the same directory where the script was executed as shown in the figure below.

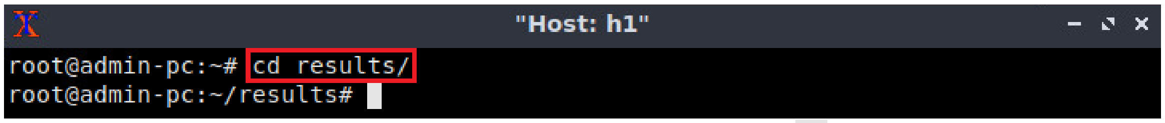


```
"Host: h1"
root@admin-pc:~# ls
concurrent_diff_algo  Documents       mininet   Templates
concurrent_no_pacing  Downloads       Music     test_result.json
concurrent_pacing     iperf3_plotter  Pictures  test_results.json
concurrent_same_algo  iperf.csv       Public    Videos
Desktop               lab_topologies  results
root@admin-pc:~#
```

Figure 29. Listing the current directory's contents using the `ls` command.

Step 4. Navigate to the results folder using the `cd` command.

```
cd results/
```

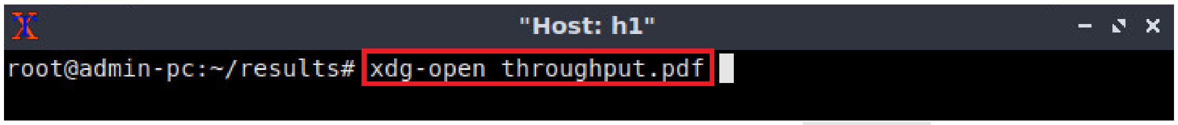


```
"Host: h1"
root@admin-pc:~# cd results/
root@admin-pc:~/results#
```

Figure 30. Entering the results directory using the `cd` command.

Step 5. To open any of the generated files, use the `xdg-open` command followed by the file name. For example, to open the throughput.pdf file, use the following command:

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



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

Figure 31. Opening the *throughput.pdf* file using `xdg-open`.

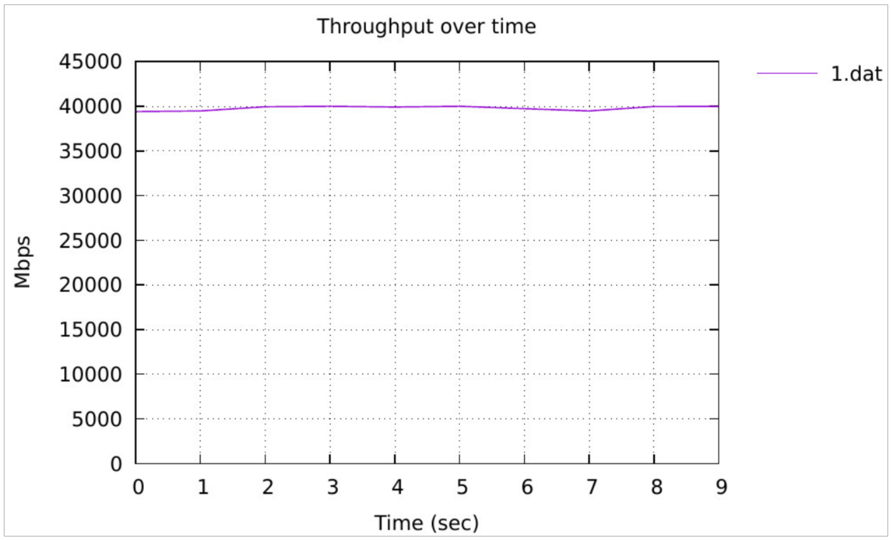


Figure 32. *throughput.pdf* output.

Step 6. 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 2. Stop the emulation and then exit out of MiniEdit.

References

1. A. Tanenbaum, D. Wetherall, "Computer networks," 5th Edition, Prentice Hall, 2011.
2. J. Kurose, K. Ross, "Computer networking, a top-down approach," 7th Edition, Pearson, 2017.
3. Invoking Iperf3 [Online]. Available: https://software.es.net/iperf/invoking.html.

UNIVERSITY OF
SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 3: Emulating WAN with NETEM I: Latency, Jitter

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 network emulators and NETEM 4
1.1 NETEM 4
1.2 WANs and delay 4
2 Lab topology 5
2.1 Starting host h1 and host h2 6
3 Adding/changing delay to emulate a WAN 7
3.1 Identify interface of host h1 and host h2 8
3.2 Add delay to interface connecting to WAN 9
3.3 Changing the delay in emulated WAN 11
4 Restoring original values (deleting the rules) 12
5 Adding jitter to emulated WAN 13
5.1 Add jitter to interface connecting to WAN 13
6 Adding correlation value for jitter and delay 14
7 Delay distribution 16
References 17

Overview

This lab introduces NETEM and explains how it can be used to emulate real-world scenarios while having control on parameters that affect the performance of networks. Network parameters include latency, jitter, packet loss, reordering, and corruption. Correlation values between network parameters will also be set to provide a more realistic network environment.

Objectives

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

1. Understand delay in networks and how to measure it.
2. Understand Linux queuing disciplines (qdisc) architecture.
3. Deploy emulated WANs characterized by large delays using NETEM and Mininet.
4. Perform measurements after introducing delays to an emulated WAN.
5. Deploy emulated WANs characterized by delays, jitters, and corresponding correlation values.
6. Modify the delay distribution of an emulated WAN.

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 network emulators and NETEM.
2. Section 2: Lab topology.
3. Section 3: Adding/changing delay to emulate a WAN.
4. Section 4: Restoring original values (deleting the rules).
5. Section 5: Adding jitter to emulated WAN.
6. Section 6: Adding correlation value for jitter and delay.
7. Section 7: Delay distribution.

1 Introduction to network emulators and NETEM

Network emulators play an important role for the research and development of network protocols and applications. Network emulators provide the ability to perform tests of realistic scenarios in a controlled manner, which is very difficult on production networks. This is particularly complex for researchers who develop and test tools for *Wide Area Networks (WANs)* and for multi-domain environments.

1.1 NETEM

One of the most popular network emulators is *NETEM*[1,2], a Linux network emulator for testing the performance of real applications over a virtual network. The virtual network may reproduce long-distance WANs in the lab environment. These scenarios facilitate the test and evaluation of protocols and devices from the application layer to the data-link layer under a variety of conditions. NETEM allows the user to modify parameters such as delay, jitter, packet loss, duplication and re-ordering of packets.

NETEM is implemented in Linux and consists of two portions: a small kernel module for a queuing discipline and a command line utility to configure it. Figure 1 shows the basic architecture of Linux queuing disciplines. The queuing disciplines exist between the IP protocol output and the network device. The default queuing discipline is a simple packet first-in first-out (FIFO) queue. A queuing discipline is a simple object with two interfaces. One interface queues packets to be sent and the other interface releases packets to the network device. The queuing discipline makes the policy decision of which packets to send, which packets to delay, and which packets to drop. A classful queueing discipline, such as NETEM, has configurable internal modules.

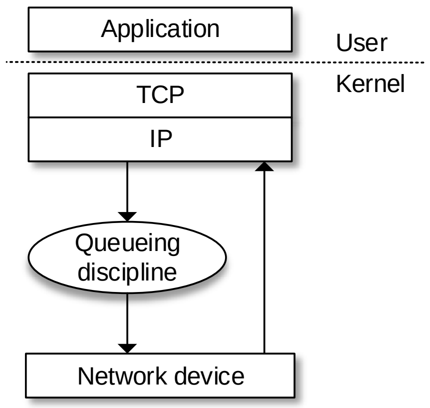


Figure 1. Linux queueing discipline.

1.2 WANs and delay

In networks, there are several processes and devices that contribute to the end-to-end delay between a sender node and a destination node. Many times, the end-to-end delay is dominated by the WAN's propagation delay. Consider two adjacent switches A and B connected by a WAN. Once a bit is pushed onto the WAN by switch A, it needs to propagate to switch B. The time required to propagate from the beginning of the WAN to switch B is the propagation delay. The bit propagates at the propagation speed of the WAN's link. The propagation speed depends on the physical medium (that is, fiber optics, twisted-pair copper wire, etc) and is in the range of $2\text{x}10^8$ meters/sec to $3\text{x}10^8$ meters/sec, which is equal to, or a little less than, the speed of light. The propagation delay is the distance between two switches divided by the propagation speed. Once the last bit of the packet propagates to switch B, it and all the preceding bits of the packet are stored in switch B[3].

Network tools usually estimate delay for troubleshooting and performance measurements. For example, an estimate of end-to-end delay is the Round-Trip Time (RTT), which is the time it takes for a small packet to travel from sender to receiver and then back to the sender. The RTT includes packet-propagation delays, packet-queuing delays in intermediate routers and switches, and packet-processing. As mentioned above, if the propagation delay dominates other delay components (as in the case of many WANs), then RTT is also an estimate of the propagation delay.

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.

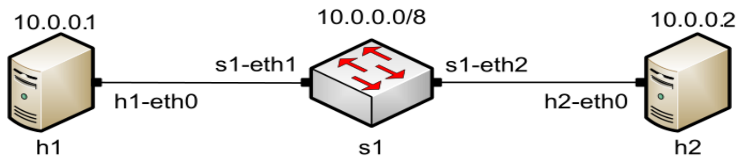


Figure 2. Lab topology.

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`.

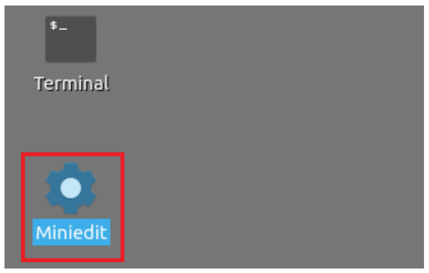


Figure 3. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 3.mn* topology file and click on *Open*.

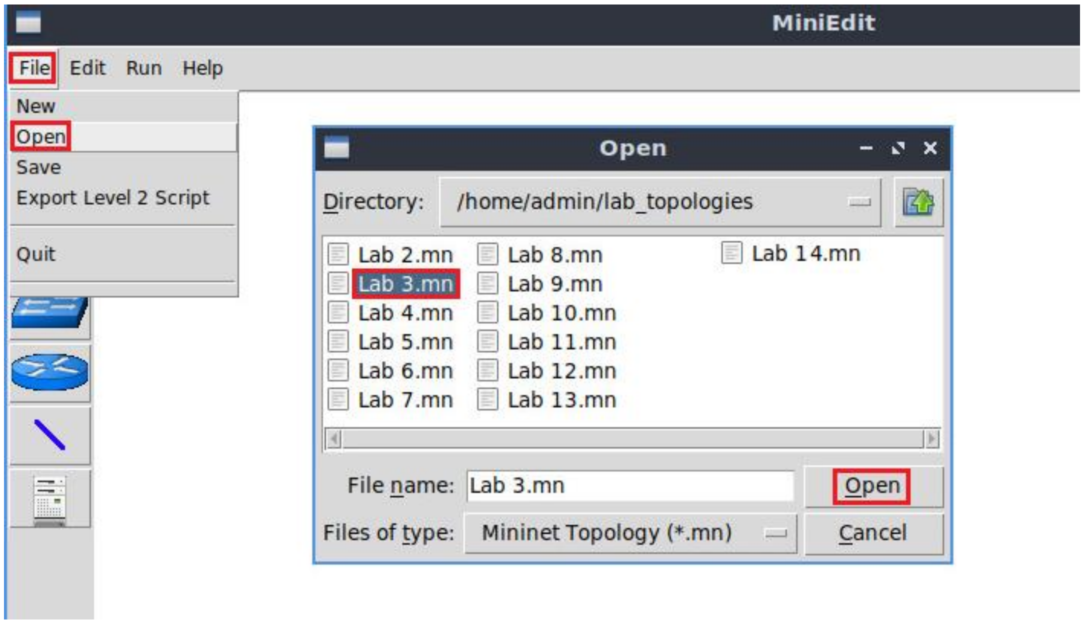


Figure 4. MiniEdit's *Open* dialog.

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.

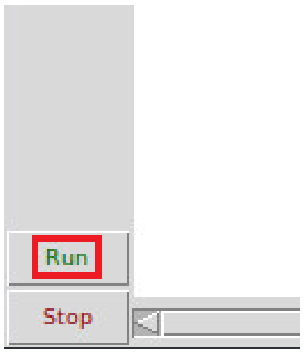


Figure 5. Running the emulation.

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

2.1 Starting host h1 and host h2

Step 1. Hold the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on host h1.

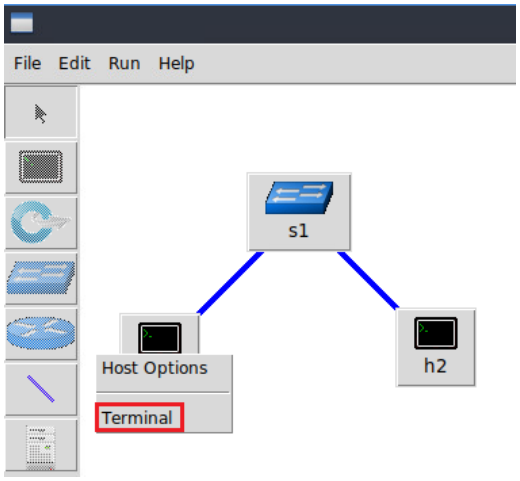
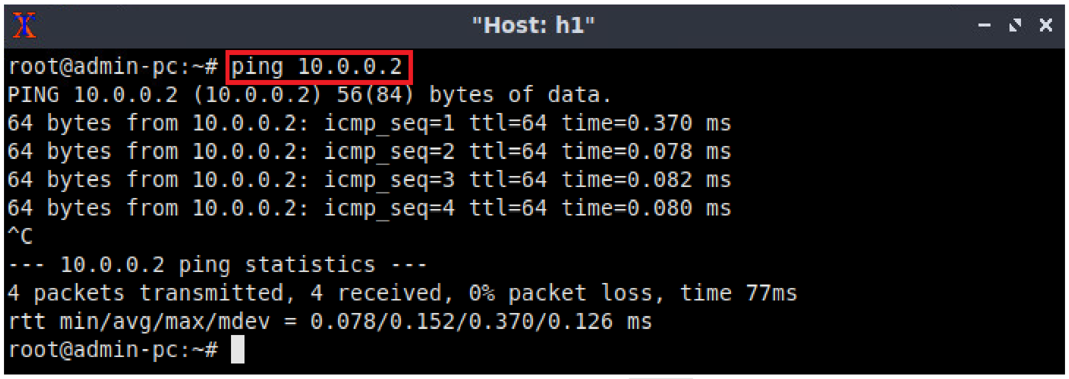


Figure 6. Opening a terminal on host h1.

Step 2. 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.



```
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=0.370 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.080 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 77ms
rtt min/avg/max/mdev = 0.078/0.152/0.370/0.126 ms
root@admin-pc:~#
```

Figure 7. Connectivity test using `ping` command.

The figure above indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

3 Adding/changing delay to emulate a WAN

The user invokes NETEM using the command line utility called `tc` [4, 5]. With no additional parameters, NETEM behaves as a basic FIFO queue with no delay, loss, duplication, or reordering of packets. The basic `tc` syntax used with NETEM is as follows:

```
sudo tc qdisc [add|del|replace|change|show] dev dev_id root netem opts
```

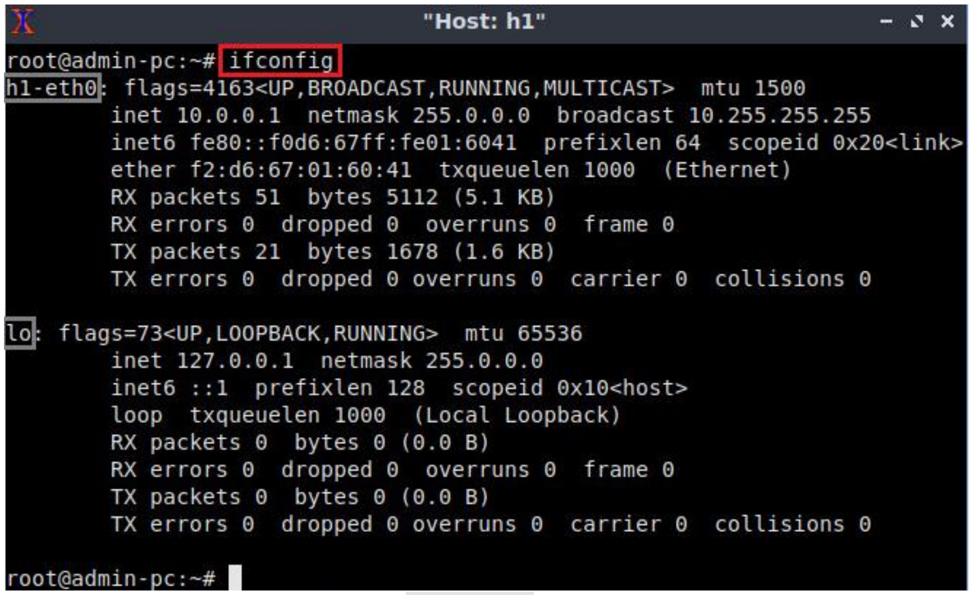
- `sudo`: enable the execution of the command with higher security privileges.
- `tc`: command used to interact with NETEM.

- `qdisc`**:** a queue discipline (qdisc) is a set of rules that determine the order in which packets arriving from the IP protocol output (see Figure 1) 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 delay on a specific interface, the operation will be `add`. To change or remove delay on the specific interface, the operation will be `change` or `del`.
- `dev_id`**:** this parameter indicates the interface to be subject to emulation.
- `opts`**:** this parameter indicates the amount of delay, packet loss, duplication, corruption, and others.

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.

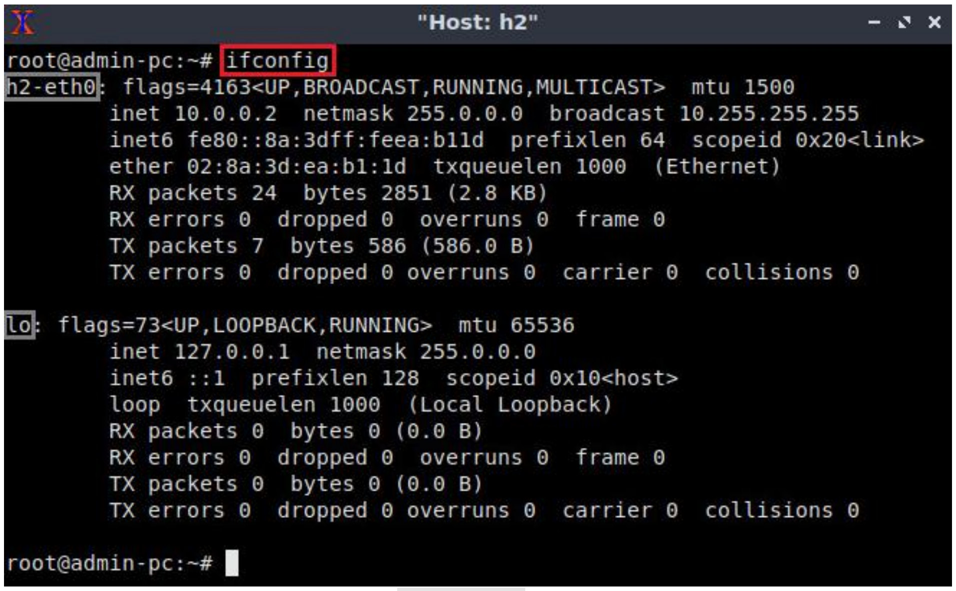


```
root@admin-pc:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::f0d6:67ff:fe01:6041  prefixlen 64  scopeid 0x20<link>
        ether f2:d6:67:01:60:41  txqueuelen 1000  (Ethernet)
        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 8. 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 WAN.

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



```
"Host: h2"
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 9. 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 WAN.

3.2 Add delay to interface connecting to WAN

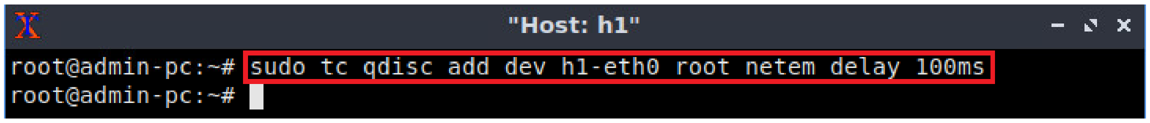
Network emulators emulate delays by introducing them to an interface. For example, the delay introduced to a switch A's interface that is connected to a switch B's interface may represent the propagation delay of a WAN connecting both switches. In this section, you will use `netem` command to insert delay to a network interface.

Step 1. In host h1, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem delay 100ms
```

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`: specify the interface on which the rule will be applied.
- `netem`: use the network emulator.
- `delay 100ms`: inject delay of 100ms.



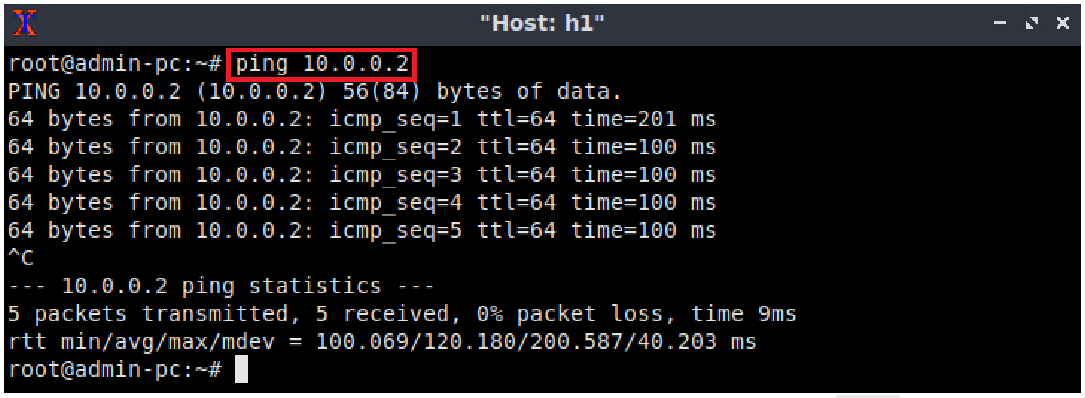
```
"Host: h1"
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem delay 100ms
root@admin-pc:~#
```

Figure 10. Adding 100ms delay to the interface *h1-eth0*.

The above command adds a delay of 100 milliseconds (ms) to the output interface, exclusively.

Step 2. The user can verify now that the connection from host h1 to host h2 has a delay of 100 milliseconds by using the `ping` command from host h1:

```
ping 10.0.0.2
```



```
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=201 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=100 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 9ms
rtt min/avg/max/mdev = 100.069/120.180/200.587/40.203 ms
root@admin-pc:~# 
```

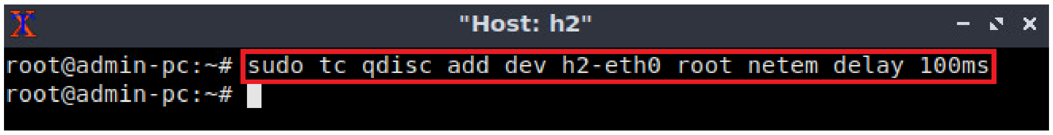
Figure 11. Verifying latency after emulating delay using `ping`.

The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 100.069, 120.180, 200.587, and 40.203 milliseconds respectively.

Note that the above scenario emulates 100 milliseconds latency on the interface of host h1 connecting to the switch. In order to emulate a WAN where the delay is bidirectional, a delay of 100 milliseconds must also be added to the corresponding interface on host h2.

Step 3. In host h2's terminal, type the following command:

```
sudo tc qdisc add dev h2-eth0 root netem delay 100ms
```

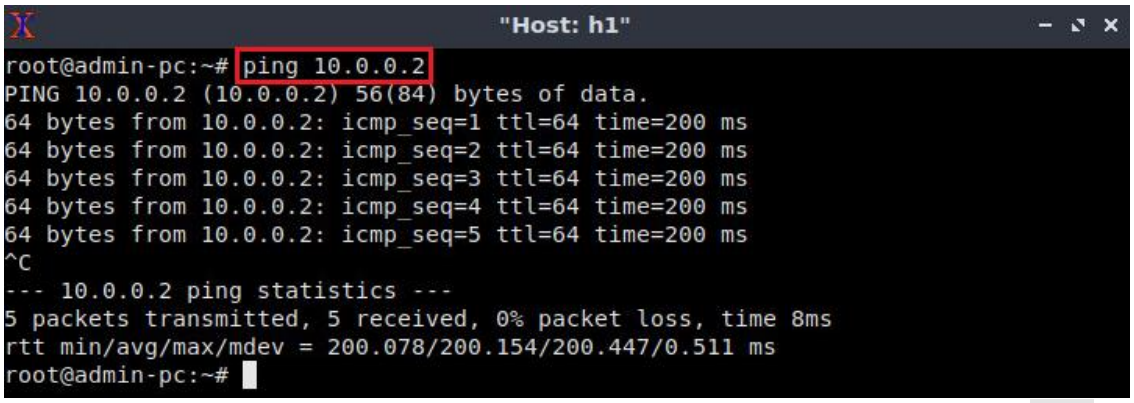


```
root@admin-pc:~# sudo tc qdisc add dev h2-eth0 root netem delay 100ms
root@admin-pc:~# 
```

Figure 12. Adding 100ms delay to the interface *h2-eth0*.

Step 4. The user can verify now that the connection between host h1 and host h2 has an RTT of 200 milliseconds (100ms from host h1 to host h2 plus 100ms from host h2 to host h1) by retyping the `ping` command on host h1's terminal:

```
ping 10.0.0.2
```



```
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=200 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=200 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=200 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=200 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=200 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 8ms
rtt min/avg/max/mdev = 200.078/200.154/200.447/0.511 ms
root@admin-pc:~#
```

Figure 13. Verifying latency after emulating delay on both host h1 and host h2 using `ping`.

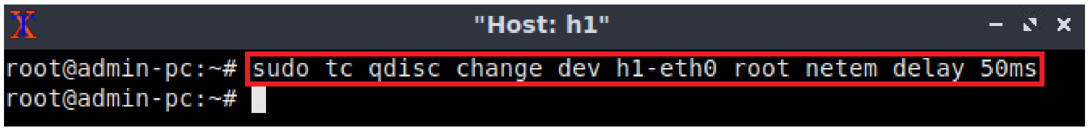
The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 200.078, 200.154, 204.447, and 0.511 milliseconds respectively.

3.3 Changing the delay in emulated WAN

In this section, the user will change the delay from 100 milliseconds to 50 milliseconds in both sender and receiver. The RTT will be 100 milliseconds now.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc change dev h1-eth0 root netem delay 50ms
```



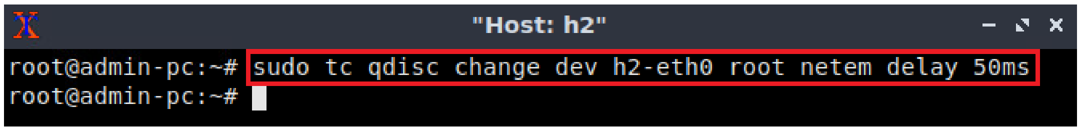
```
root@admin-pc:~# sudo tc qdisc change dev h1-eth0 root netem delay 50ms
root@admin-pc:~#
```

Figure 14. Changing delay on the interface *h1-eth0*.

The new option added here is `change`, which changes the previously set delay to 50 milliseconds.

Step 2. Apply also the above step on host h2's terminal to change the delay to 50ms:

```
sudo tc qdisc change dev h2-eth0 root netem delay 50ms
```

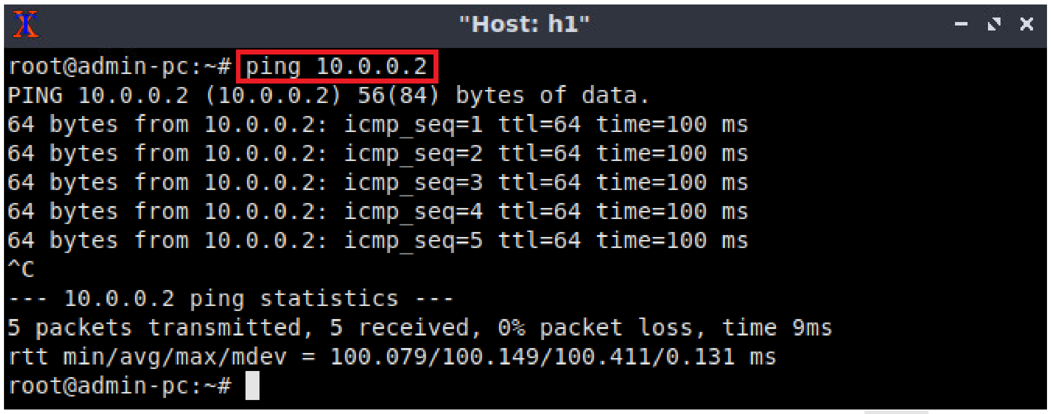


```
root@admin-pc:~# sudo tc qdisc change dev h2-eth0 root netem delay 50ms
root@admin-pc:~#
```

Figure 15. Changing delay to the interface *h2-eth0*.

Step 3. The user can verify now that the connection from host h1 to host h2 has a delay of 100 milliseconds by using the `ping` command from host h1's terminal:

```
ping 10.0.0.2
```



```
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=100 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=100 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 9ms
rtt min/avg/max/mdev = 100.079/100.149/100.411/0.131 ms
root@admin-pc:~#
```

Figure 16. Verifying latency after emulating 100ms delay using `ping`.

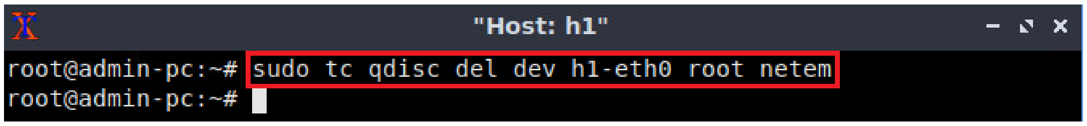
The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 100.079, 100.149, 100.411, and 0.131 milliseconds respectively.

4 Restoring original values (deleting the rules)

In this section, the user will restore the default configuration in both sender and receiver by deleting all the rules applied to the network scheduler of an interface.

Step 1. In host h1's terminal, type the following command:

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



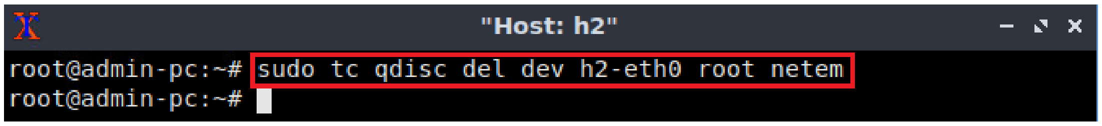
```
root@admin-pc:~# sudo tc qdisc del dev h1-eth0 root netem
root@admin-pc:~#
```

Figure 17. Deleting all rules on interface *h1-eth0*.

The new option added here is `del`, which deletes the previously set rules on a given interface. As a result, the `tc` qdisc will restore its default values of the device *h1-eth0*.

Step 2. Apply the same steps to remove rules on host h2. In host h2's terminal, type the following command:

```
sudo tc qdisc del dev h2-eth0 root netem
```



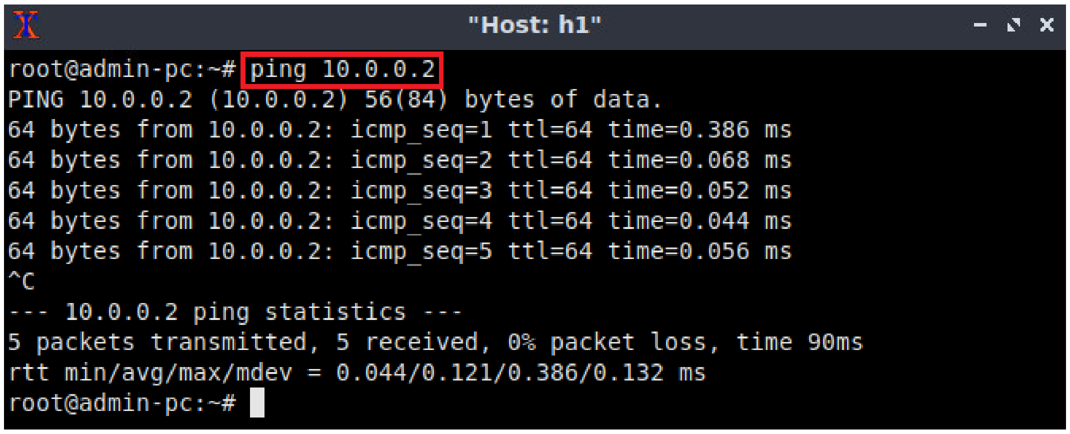
```
root@admin-pc:~# sudo tc qdisc del dev h2-eth0 root netem
root@admin-pc:~#
```

Figure 18. Deleting all rules on interface *h2-eth0*.

As a result, the `tc` queueing discipline will restore its default values of the device *h2-eth0*.

Step 3. The user can now verify that the connection from host h1 to host h2 has no explicit delay set by using the `ping` command from host h1's terminal:

```
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=0.386 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.052 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.056 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 90ms
rtt min/avg/max/mdev = 0.044/0.121/0.386/0.132 ms
root@admin-pc:~#
```

Figure 19. Verifying latency after deleting all rules on both devices.

The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 0.044, 0.121, 0.386, and 0.132 milliseconds respectively.

5 Adding jitter to emulated WAN

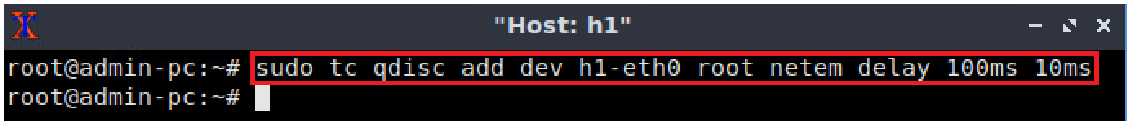
Networks do not exhibit constant delay; the delay may vary based on other traffic flows contending for the same path. Jitter is the variation of delay time. The delay parameters are described by the average value (μ), standard deviation (σ), and correlation. By default, NETEM uses a uniform distribution, so that the delay is within μ ± σ.

5.1 Add jitter to interface connecting to WAN

In this section, the user will add delay of 100 milliseconds with a random variation of ± 10 milliseconds. Before doing so, make sure to restore the default configuration of the interfaces on host h1 and host h2 by applying the commands of Section 4. Then, apply the commands below.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem delay 100ms 10ms
```



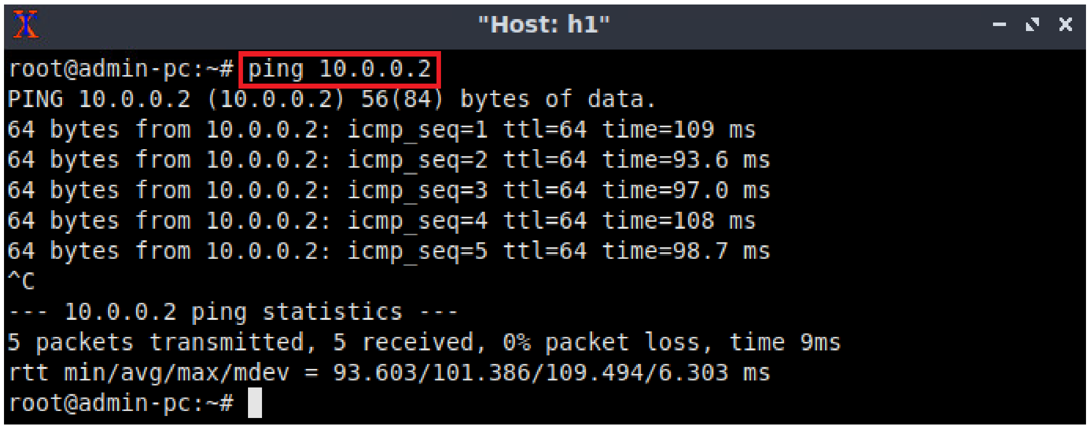
```
"Host: h1"
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem delay 100ms 10ms
root@admin-pc:~#
```

Figure 20. Add 100ms delay with ± 10 millisecond.

The new value added here represents jitter which defines the delay variation. Therefore, all packets leaving host h1 via interface *h1-eth0* will experience a delay of 100ms, with a random variation of ± 10ms.

Step 2. The user can now verify that the connection from host h1 to host h2 has 100ms delay with ± 10 millisecond random variation by using the `ping` command on host h1's terminal:

```
ping 10.0.0.2
```



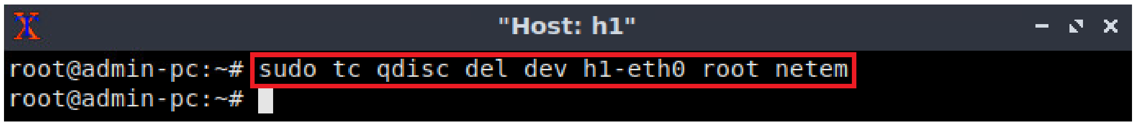
```
"Host: h1"
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=109 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=93.6 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=97.0 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=108 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=98.7 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 9ms
rtt min/avg/max/mdev = 93.603/101.386/109.494/6.303 ms
root@admin-pc:~#
```

Figure 21. Verifying RTT after adding 100 millisecond delay and 10 millisecond jitter on interface *h1-eth0*.

The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 93.603, 101.386, 109.494, and 6.303 milliseconds respectively. Note that we are only adding jitter to the interface of host h1 at this point.

Step 3. In host h1's terminal, type the following command to delete previous configurations:

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



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

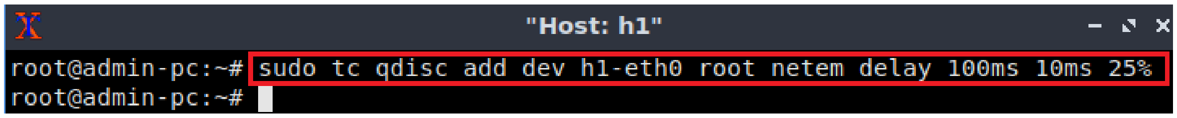
Figure 22. Deleting all rules on interface *h1-eth0*.

6 Adding correlation value for jitter and delay

The correlation parameter controls the relationship between successive pseudo-random values. In this section, the user will add a delay of 100 milliseconds with a variation of ± 10 milliseconds while adding a correlation value. Before doing so, make sure to restore the default configuration of the interfaces on host h1 and host h2 by applying the commands of Section 4. Then, apply the commands below.

Step 1. In host h1 terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem delay 100ms 10ms 25%
```



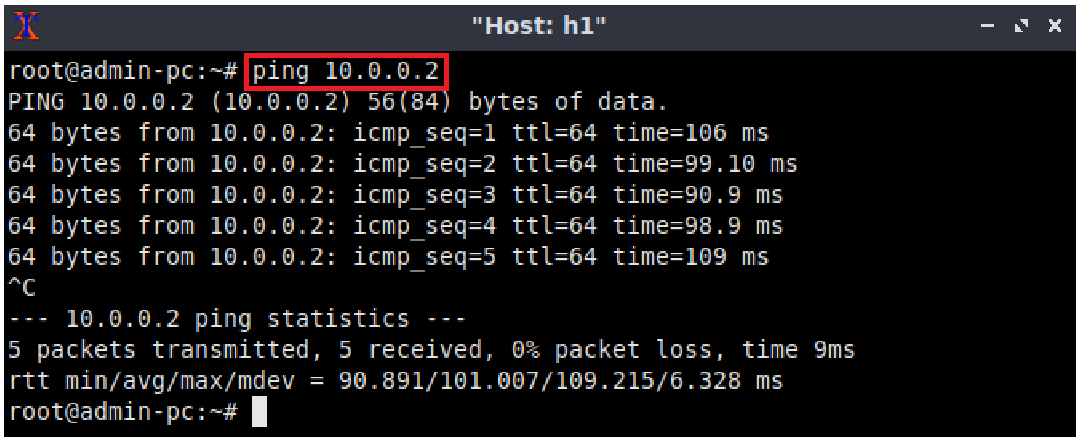
```
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem delay 100ms 10ms 25%
root@admin-pc:~#
```

Figure 23. Adding a correlation value of 25%.

The new value added here represents the correlation value for jitter and delay. Therefore, all packets leaving the device host h1 on the interface *h1-eth0* will experience a 100ms delay time, with a random variation of ± 10 millisecond with the next random packet depending 25% on the previous one.

Step 2. Now, the user can test the connection from host h1 to host h2 by using the `ping` command on host h1's terminal:

```
ping 10.0.0.2
```



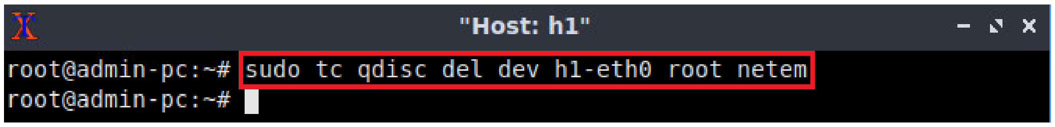
```
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=106 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=99.10 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=90.9 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=98.9 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=109 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 9ms
rtt min/avg/max/mdev = 90.891/101.007/109.215/6.328 ms
root@admin-pc:~#
```

Figure 24. Verifying latency after setting the correlation value.

The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 90.891, 101.007, 109.215, and 6.328 milliseconds respectively.

Step 3. In host h1's terminal, type the following command to delete previous configurations:

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



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

Figure 25. Deleting all rules on interface *h1-eth0*.

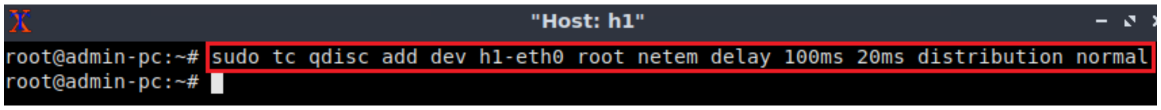
7 Delay distribution

NETEM permits user to specify a distribution that describes how delays vary in the network. Usually delays are not uniform, so it may be convenient to use a non-uniform distribution such as normal, pareto, or pareto-normal. For this test, the user will specify a normal distribution for the delay in the emulated network. Before doing so, make sure to restore the default configuration of the interfaces on host h1 and host h2 by applying the commands of Section 4. Then, apply the commands below.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem delay 100ms 20ms distribution normal
```

The new option added here (`distribution`) represents the delay distribution type. We define the delay to have a normal distribution, which provides a more realistic emulation of WAN networks. As a result, all packets leaving the host h1 on the interface *h1-eth0* will experience delay time which is normally distributed between the range of 100ms ± 20ms.

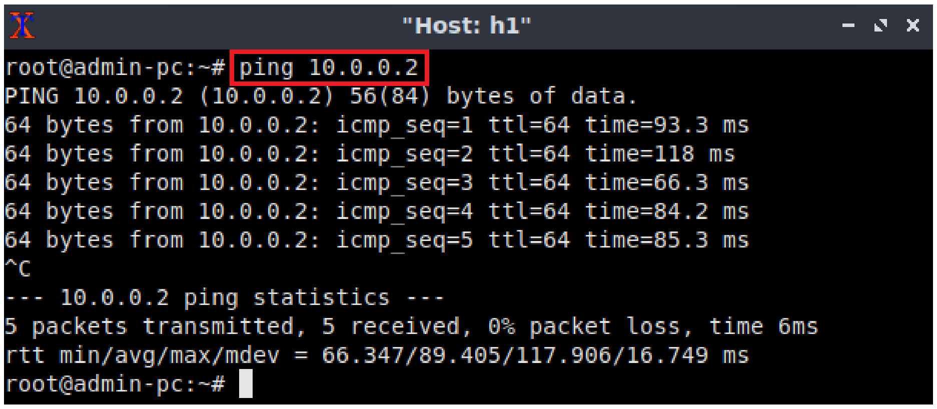


```
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem delay 100ms 20ms distribution normal
root@admin-pc:~#
```

Figure 26. Adding normal distribution of delay.

Step 2. The user can now verify if the configuration was successfully done in the previous step (Step 1) by using the `ping` command on host h1's terminal:

```
ping 10.0.0.2
```



```
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=93.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=118 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=66.3 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=84.2 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=85.3 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 6ms
rtt min/avg/max/mdev = 66.347/89.405/117.906/16.749 ms
root@admin-pc:~#
```

Figure 27. Verifying latency after using normal distribution.

The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 66.347, 89.405, 117.906, and 16.749 milliseconds respectively.

This concludes Lab 3. Stop the emulation and then exit out of MiniEdit.

References

1. Linux foundation. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem.
2. S. Hemminger, “Network emulation with NETEM,” Linux conf au. 2005, pp. 18-23. 2005.
3. J. Kurose, K. Ross, “Computer networking, a top-down approach,” 7th Edition, Pearson, 2017.
4. How to use the linux traffic control panagiotis vouzis [Online]. Available: https://netbeez.net/blog/how-to-use-the-linux-traffic-control.
5. M. Brown, F. Bolelli, N. Patriciello, “Traffic control howto,” Guide to IP Layer Network, 2006.

UNIVERSITY OF SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 4: Emulating WAN with NETEM II: Packet Loss, Duplication, Reordering, and Corruption

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 network emulators and NETEM 3
2 Lab topology........ 5
2.1 Testing connectivity between two hosts 6
3 Adding/changing packet loss 7
3.1 Identify interface of host h1 and host h2........ 8
3.2 Add packet loss to the interface connecting to the WAN 9
3.3 Restore default values........ 12
3.4 Add correlation value for packet loss to interface connecting to WAN 13
4 Adding packet corruption 14
4.1 Add packet corruption to an interface connected to the WAN........ 14
5 Add packet reordering 16
6 Add packet duplication 17
References 18

Overview

This lab continues the description of NETEM and how to use it to emulate Wide Area Networks (WANs). Besides delay, this lab focuses on other parameters such as packet loss, packet duplication, reordering, and packet corruption. These parameters affect the performance of protocols and networks.

Objectives

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

1. Deploy emulated WANs characterized by parameters such as delay, packet loss, packet corruption, packet reordering, and packet duplication.
2. Measure the performance of WANs characterized by different parameter values.
3. Visualize WAN performance measures.

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 network emulators and NETEM.
2. Section 2: Lab topology.
3. Section 3: Adding/changing packet loss.
4. Section 4: Adding packet corruption.
5. Section 5: Adding packet reordering.
6. Section 6: Adding packet duplication.

1 Introduction to network emulators and NETEM

Part I of Emulating WAN with NETEM described how to use NETEM to emulate WANs characterized by long delays. Part I also explained how the end-to-end delay can be dominated by the WAN's propagation delay and how the Round-Trip Time (RTT) estimates this delay.

In addition to delay, many WANs and LANs are subject to packet loss, reordering, corruption, and duplication.

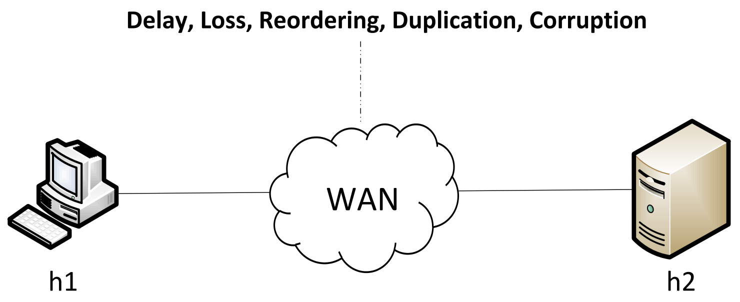


Figure 1. Parameters affecting throughput in a WAN.

The above situations are described follows:

1. Packet loss: a condition that occurs when a packet travelling across a network fails to reach its destination. Packet loss may have a large impact on high-throughput high-latency networks. A common cause of packet loss is the inability of routers to hold packets arriving at a rate higher than the departure rate. Even in cases where the high packet arrival rate is only temporary (e.g., short-term traffic bursts), the router is limited by the amount of buffer memory used to momentarily store packets. When packet loss occurs, TCP reduces the congestion window and consequently the throughput by half. Packet loss must be mitigated by using best-practice network designs, such as Science DMZ.

2. Packet reordering: a condition that occurs when packets are received in a different order from which they were sent. Packet reordering, also known as out-of-order packet delivery, is typically the result of packets following different routes to reach their destination. Packet reordering may deteriorate the throughput of TCP connections in high-throughput high-latency networks. For each segment received out of order, a TCP receiver sends an acknowledgement (ACK) for the last correctly received segment. Once the TCP sender receives three acknowledgements for the same segment (triple duplicate ACK), the sender considers that the receiver did not correctly receive the packet following the packet that is being acknowledged three times. It then proceeds to reduce the congestion window and throughput by half.

3. Packet corruption: corruption of bits comprising a packet may (mostly) occur at the physical layer. Two adjacent devices are connected by a physical channel (e.g., fiber, twisted-pair copper wire, etc). The physical layer accepts a raw bit stream and delivers it to the data-link layer. If corruption occurs, some bits may have different values than those originally sent by the sender node. The receiver node then simply discards the packet. As a result, the TCP sender process will not

receive an acknowledgement for the corresponding segment and will consider it as a lost segment. The TCP sender process will subsequently decrease the congestion window and throughput by half.

4. Packet duplication: a condition where multiple copies of a packet are present in the network and received by the destination. Packet duplication is the result of retransmissions, where a sender node retransmits unacknowledged (NACK) packets.

Packet loss, reordering, and corruption (the last two are interpreted as packet loss also by the TCP sender) lead to a drastic reduction of throughput. In this lab, we will use the NETEM tool to emulate these situations affecting end-to-end performance.

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.

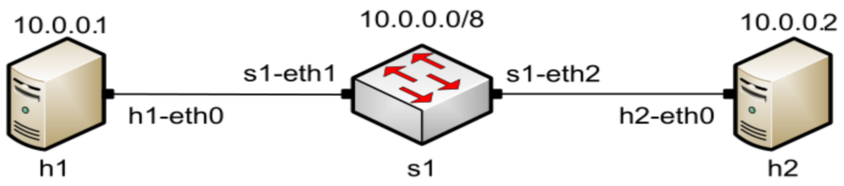


Figure 2. Lab topology.

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`.

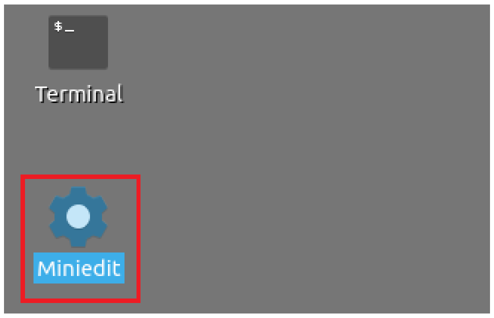


Figure 3. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 4.mn* topology file and click on *Open*.

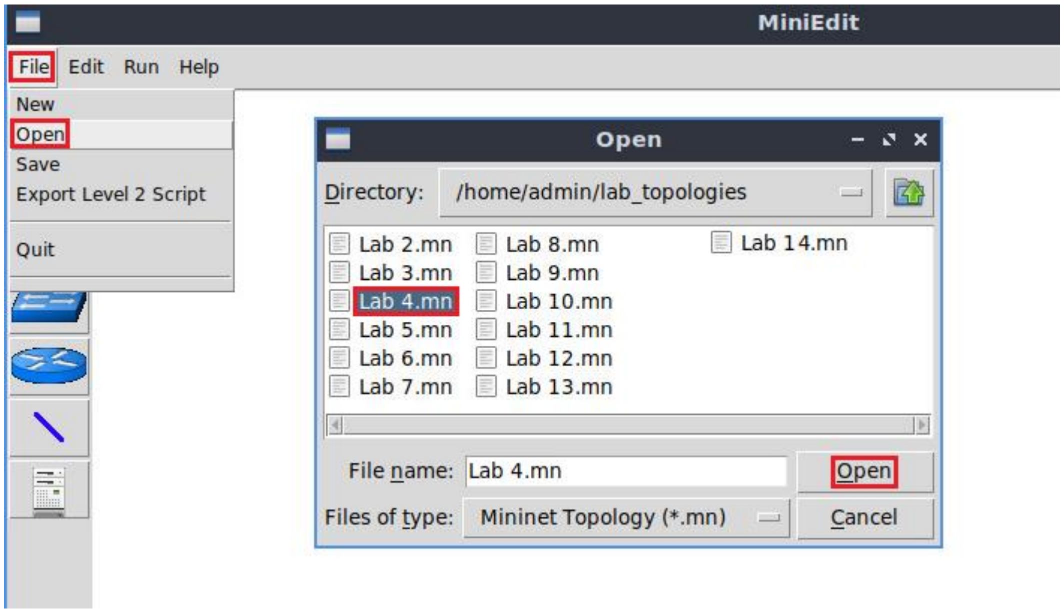


Figure 4. MiniEdit's *Open* dialog.

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.

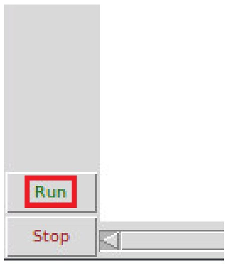


Figure 5. Running the emulation.

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

2.1 Testing connectivity between two hosts

Step 1. Hold the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on host h1.

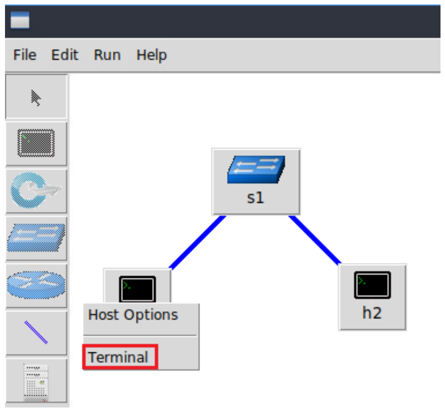
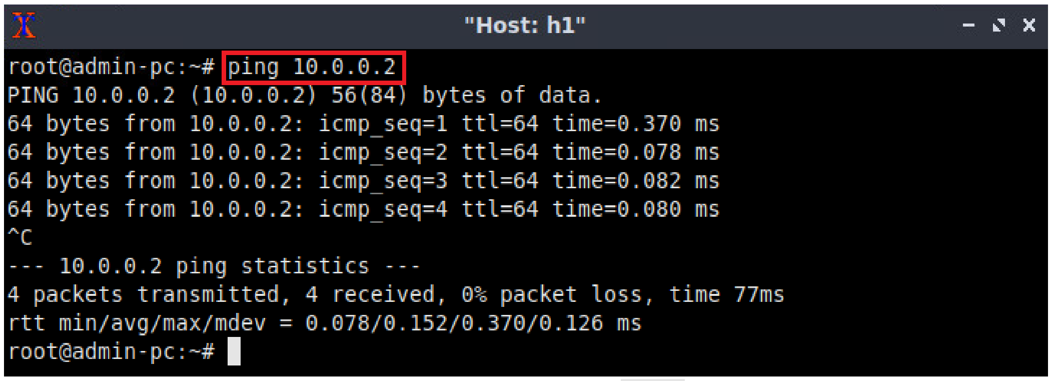


Figure 6. Opening a terminal on host h1.

Step 2. 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.



```
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=0.370 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.080 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 77ms
rtt min/avg/max/mdev = 0.078/0.152/0.370/0.126 ms
root@admin-pc:~# 
```

Figure 7. Connectivity test using `ping` command.

The figure above indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

3 Adding/changing packet loss

The user invokes NETEM using the command line utility called `tc` [4, 5]. With no additional parameters, NETEM behaves as a basic FIFO queue with no delay, loss, duplication, or reordering of packets. The basic `tc` syntax used with NETEM is as follows:

```
sudo tc qdisc [add|del|replace|change|show] dev dev_id root netem opts
```

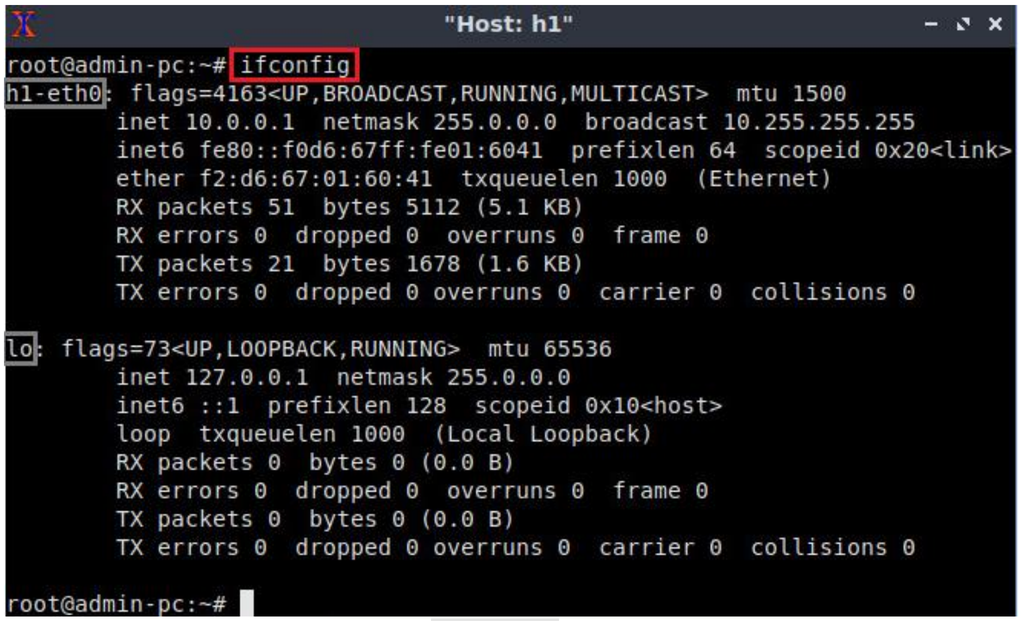
- `sudo`: enable the execution of the command with higher security privileges.
- `tc`: command used to interact with NETEM.
- `qdisc`: a queue discipline (qdisc) is a set of rules that determine the order in which packets arriving from the IP protocol output are served. The queue discipline is applied 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 delay on a specific interface, the operation will be `add`. To change or remove delay on the specific interface, the operation will be `change` or `del`.
- `dev_id`: this parameter indicates the interface to be subject to emulation.
- `opts`: this parameter indicates the amount of delay, packet loss, duplication, corruption, and others.

3.1 Identify interface of host h1 and host h2

In this 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.

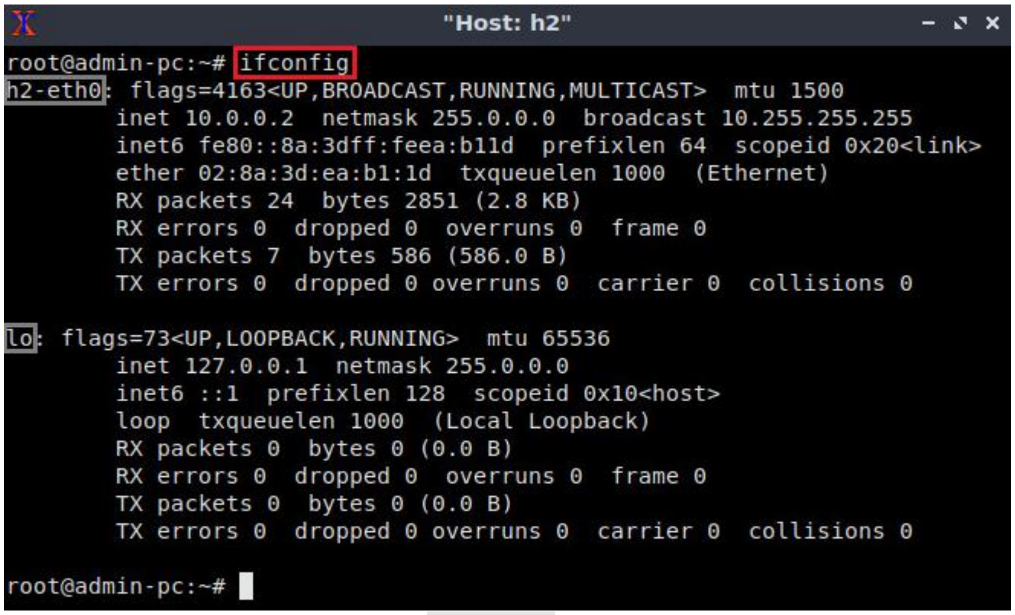


```
root@admin-pc:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::f0d6:67ff:fe01:6041  prefixlen 64  scopeid 0x20<link>
        ether f2:d6:67:01:60:41  txqueuelen 1000  (Ethernet)
        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 8. 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 h2 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 WAN.

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



```
"Host: h2"
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 9. Output of `ifconfig` command on host h2.

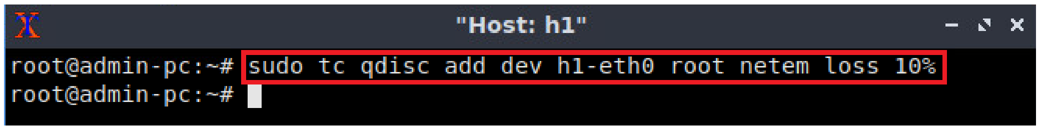
The output of the `ifconfig` command indicates that host h2 has two interfaces: *h2-eth0* and *lo*. The interface *h2-eth0* at host h1 is configured with IP address 10.0.0.2 and subnet mask 255.0.0.0. This interface must be used in `tc` when emulating the WAN.

3.2 Add packet loss to the interface connecting to the WAN

In a network, packets may be lost during transmission due to factors such as bit errors and network congestion. The rate of packets that are lost is often measured as a percentage of lost packets with respect to the number of sent packets. In this section, you will use `netem` command to insert packet loss on a network interface.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem loss 10%
```



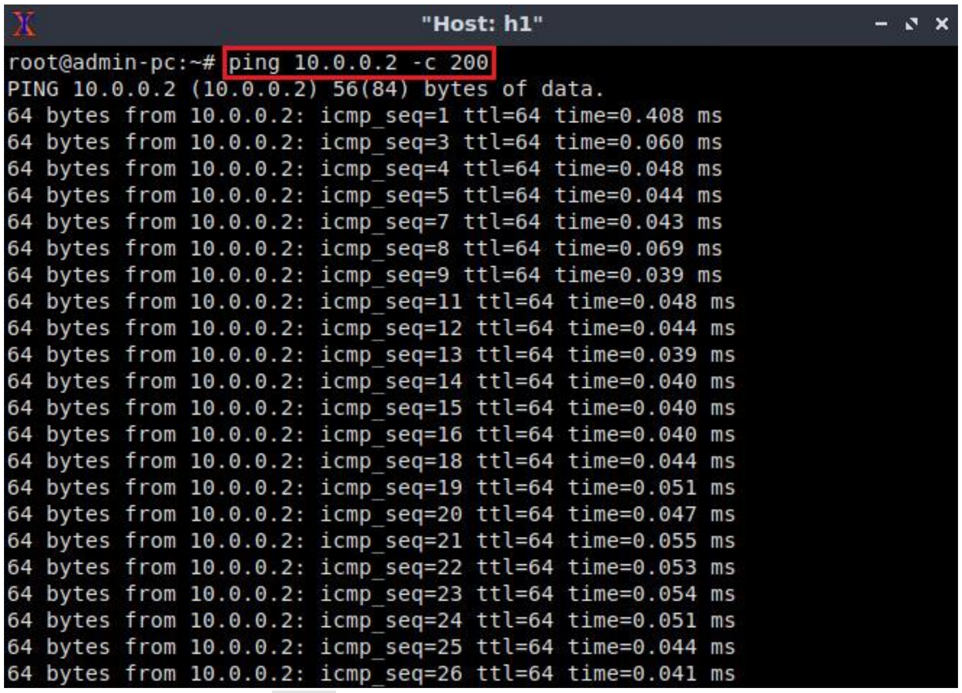
```
"Host: h1"
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem loss 10%
root@admin-pc:~# 
```

Figure 10. Adding 10% packet loss to host h1's interface *h1-eth0*.

The above command adds a 10% packet loss to host h1's interface *h1-eth0*.

Step 2. The user can verify now that the connection from host h1 to host h2 has packet losses by using the `ping` command from host h1's terminal. The `-c` option specifies the total number of packets to send.

```
ping 10.0.0.2 -c 200
```



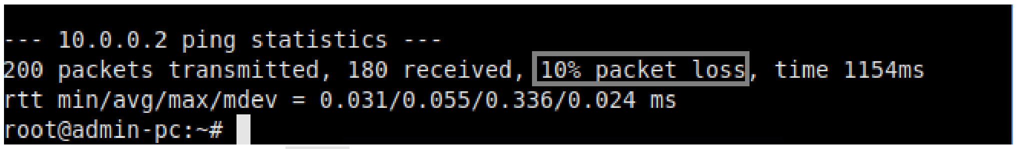
```
"Host: h1"
root@admin-pc:~# ping 10.0.0.2 -c 200
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.408 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=23 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=24 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=0.041 ms
```

Figure 11. `ping` command after introducing packet loss.

In the figure 11, host h1 sends 200 ping packets to host h2. Note the *icmp_seq* values demonstrated in the figure above.

You can see that *icmp_seq*=2, 6, 10 and 17 are missing due to packet losses. Resulting packet loss will likely vary in each emulation.

Figure 12 shows the summary report of the previous command. By default, `ping` reports the percentage of packet loss after finishing the transmission. In our test, ping reported a packet loss rate of 10%. The measured packet loss rate will tend to become closer to the configured loss rate as more trials are performed.



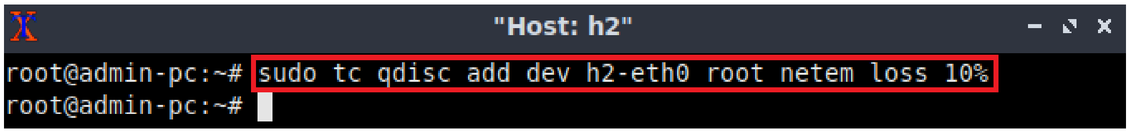
```
--- 10.0.0.2 ping statistics ---
200 packets transmitted, 180 received, 10% packet loss, time 1154ms
rtt min/avg/max/mdev = 0.031/0.055/0.336/0.024 ms
root@admin-pc:~#
```

Figure 12. `ping` summary report showing 10% packet loss.

Note that the above scenario emulates 10% packet loss on the unidirectional link from host h1 to host h2. If we want to emulate packet loss on both directions, a packet loss of 10% must also be added to host h2.

Step 3. In host h2's terminal, type the following command:

```
sudo tc qdisc add dev h2-eth0 root netem loss 10%
```

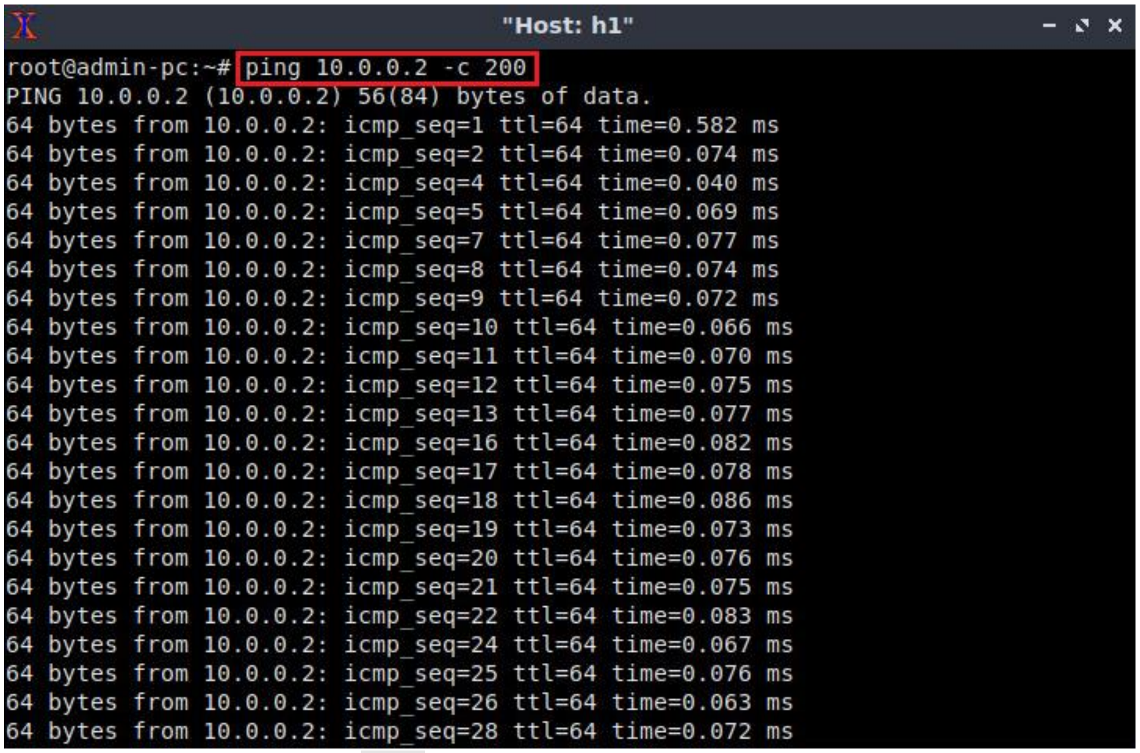


```
root@admin-pc:~# sudo tc qdisc add dev h2-eth0 root netem loss 10%
root@admin-pc:~# 
```

Figure 13. Adding 10% packet loss to host h2's interface *h2-eth0*.

Step 4. The user can verify now that the connection between host h1 and host h2 has more packets losses (10% from host h1 + 10% from host h2) by retyping the `ping` command on host h1's terminal:

```
ping 10.0.0.2 -c 200
```



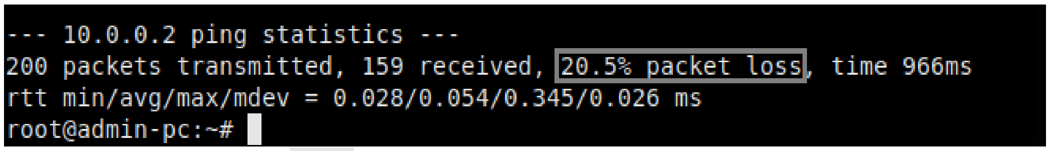
```
root@admin-pc:~# ping 10.0.0.2 -c 200
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.582 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.066 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.078 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=21 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=24 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=28 ttl=64 time=0.072 ms
```

Figure 14. `ping` command after introducing packet loss.

In the figure 14, host h1 sends 200 ping packets to host h2. Note the *icmp_seq* values demonstrated in the figure above.

You can see that *icmp_seq*=3, 6, 10, 14, 23 and 27 are missing due to packet losses. Resulting packet loss will likely vary in each emulation.

Figure 14 shows the summary report of the previous command. By default, `ping` reports the percentage of packet loss after finishing the transmission. In our test, ping reported a packet loss rate of 10%. The measured packet loss rate will tend to become closer to the configured loss rate as more trials are performed.



```
--- 10.0.0.2 ping statistics ---
200 packets transmitted, 159 received, 20.5% packet loss, time 966ms
rtt min/avg/max/mdev = 0.028/0.054/0.345/0.026 ms
root@admin-pc:~#
```

Figure 15. `ping` summary report showing 20.5% packet loss.

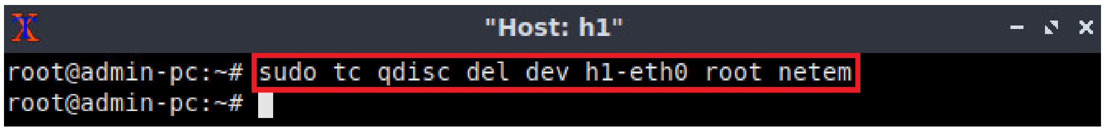
The result above indicates that 159 out of 200 packets were received successfully (20.5% packet loss).

3.3 Restore default values

To remove the packet loss added in Section 3.2 and restore the default configuration, you must delete the rules of the interfaces on host h1 and host h2.

Step 1. In host h1's terminal, type the following command:

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

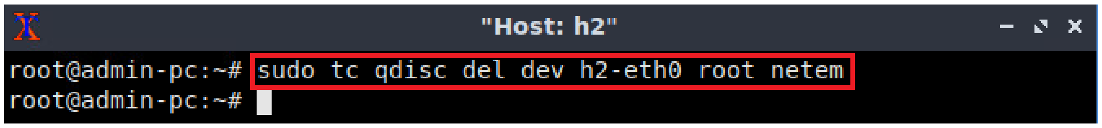


```
root@admin-pc:~# sudo tc qdisc del dev h1-eth0 root netem
root@admin-pc:~#
```

Figure 16. Deleting all rules on interface *h1-eth0*.

Step 2. Apply the same steps to remove rules on host h2. In host h2's terminal, type the following command:

```
sudo tc qdisc del dev h2-eth0 root netem
```



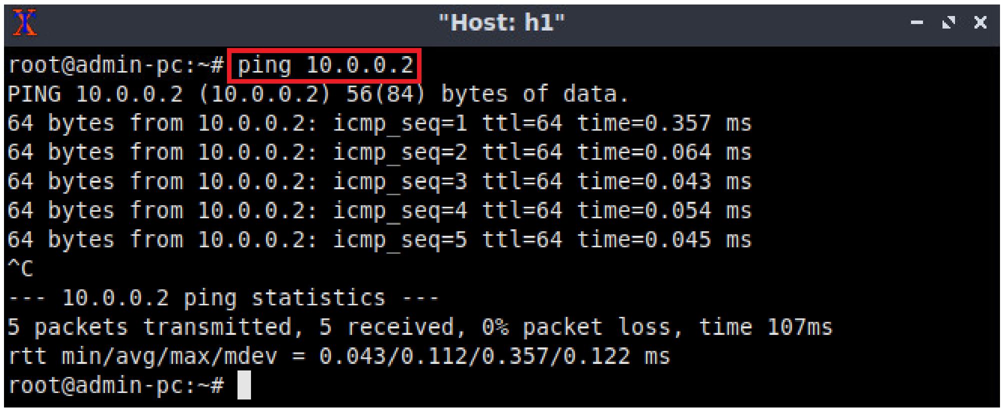
```
root@admin-pc:~# sudo tc qdisc del dev h2-eth0 root netem
root@admin-pc:~#
```

Figure 17. Deleting all rules on interface *h2-eth0*.

As a result, the `tc` queueing discipline will restore its default values of the device *h2-eth0*.

Step 3. Now, the user can verify that the connection from host h1 to host h2 has no explicit packet loss configured by using the `ping` command from host h1's terminal, press `Ctrl+c` to stop the test:

```
ping 10.0.0.2
```



```
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=0.357 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.045 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 107ms
rtt min/avg/max/mdev = 0.043/0.112/0.357/0.122 ms
root@admin-pc:~# 
```

Figure 18. Verifying latency after deleting all rules on both devices.

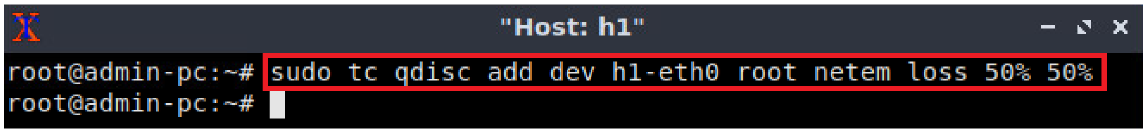
The result above indicates that all five packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 0.043, 0.112, 0.357, and 0.122 milliseconds respectively.

3.4 Add correlation value for packet loss to interface connecting to WAN

An optional correlation may be added. Adding correlation causes the random number generator to be less random and can be used to emulate packet burst losses[1].

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem loss 50% 50%
```



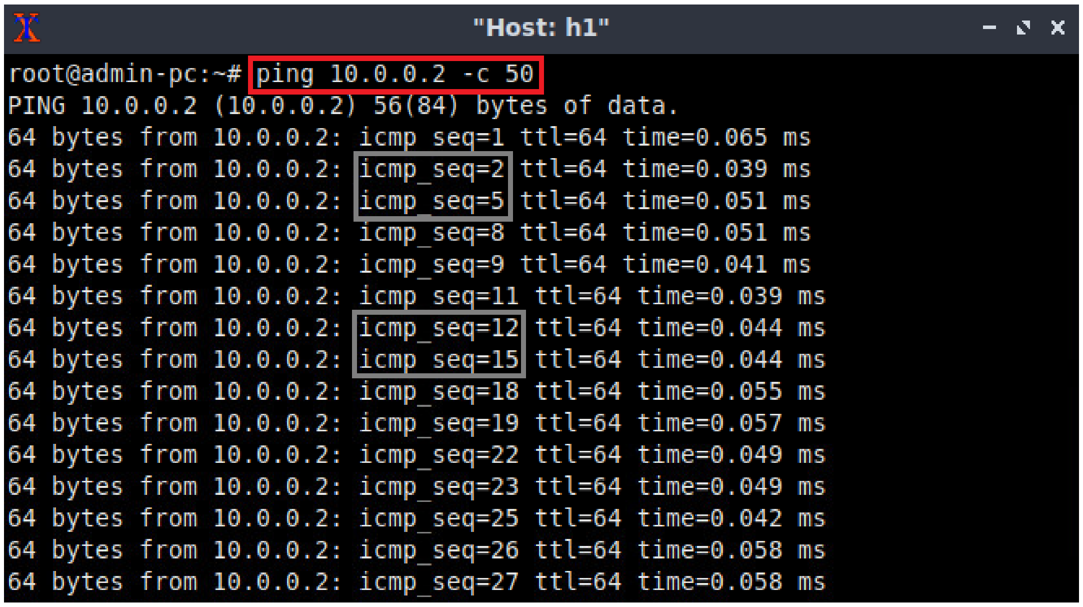
```
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem loss 50% 50%
root@admin-pc:~# 
```

Figure 19. Verifying latency after deleting all rules on both devices.

The above command introduces a packet loss rate of 50%, and each successive probability depends 50% on the last one[1]. Note that a packet loss rate this high is unlikely.

Step 2. The user can verify now that the connection from host h1 to host h2 has packet losses by using the `ping` command from host h1's terminal.

```
ping 10.0.0.2 -c 50
```



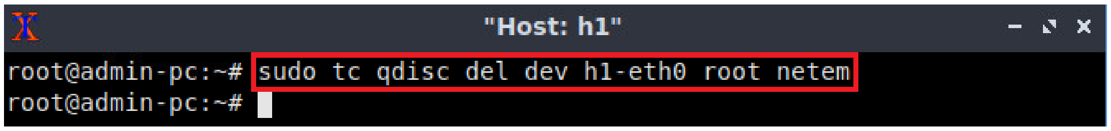
```
root@admin-pc:~# ping 10.0.0.2 -c 50
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.041 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.057 ms
64 bytes from 10.0.0.2: icmp_seq=22 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=23 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=25 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=26 ttl=64 time=0.058 ms
64 bytes from 10.0.0.2: icmp_seq=27 ttl=64 time=0.058 ms
```

Figure 20. `ping` in progress showing successive packet loss.

The result above shows an example where successive packets were dropped: [3, 4, 6, 10,], [13, 14, 16, 17, 20, 21], etc.

Step 3. In host h1's terminal, type the following command to delete previous configurations:

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



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

Figure 21. Deleting all rules on interface *h1-eth0*.

4 Adding packet corruption

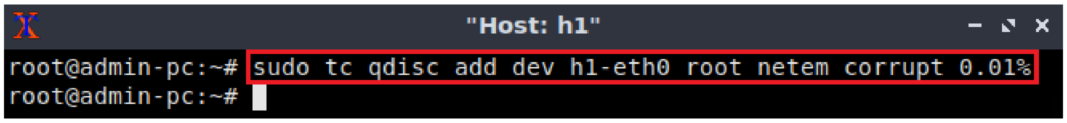
Besides packet loss, packet corruption can be introduced with NETEM.

4.1 Add packet corruption to an interface connected to the WAN

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem corrupt 0.01%
```

The new value added here represents packet corruption percentage (0.01%).

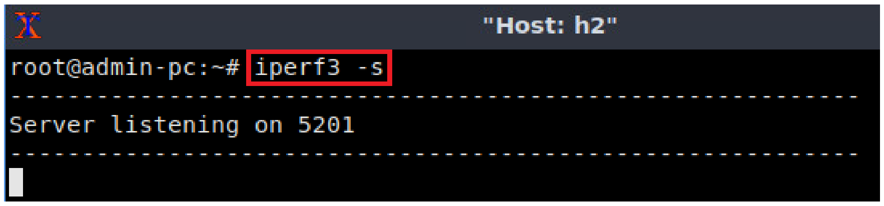


```
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem corrupt 0.01%
root@admin-pc:~#
```

Figure 22. Adding packets corruption (0.01%) to interface *h1-eth0*.

Step 2. The user can now verify the previous configuration by using the `iperf3` tool to check the retransmissions. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal.

```
iperf3 -s
```

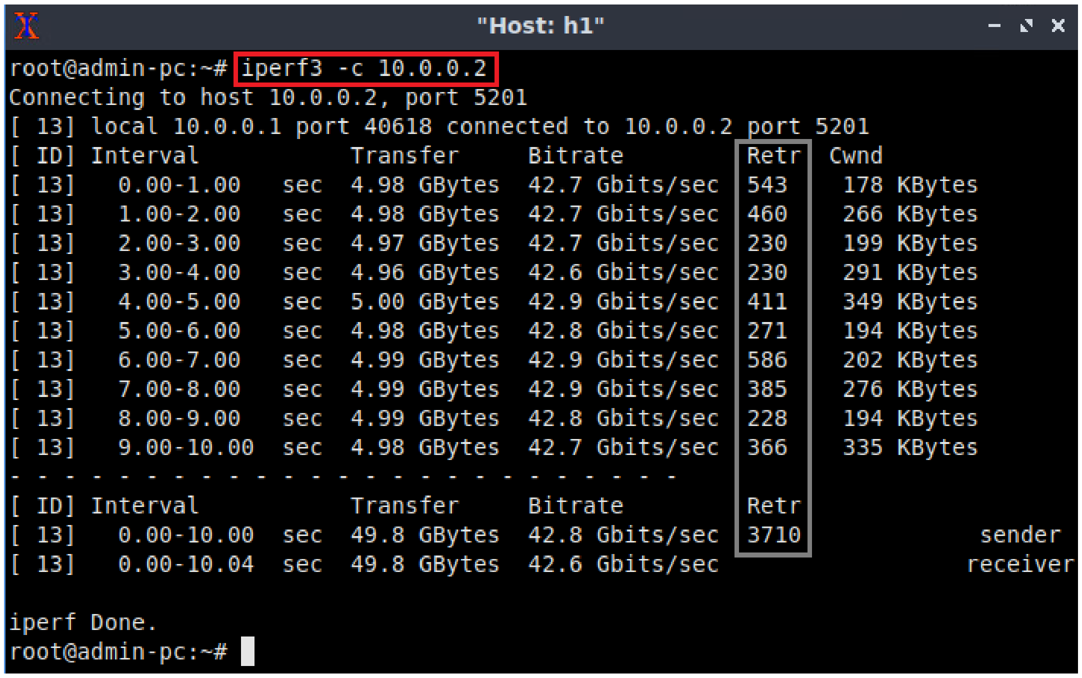


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 23. Host h2 running iPerf3 as server.

Step 3. To launch iPerf3 in client mode, run the command `iperf3 -c 10.0.0.2` in host h1's terminal.

```
iperf3 -c 10.0.0.2
```



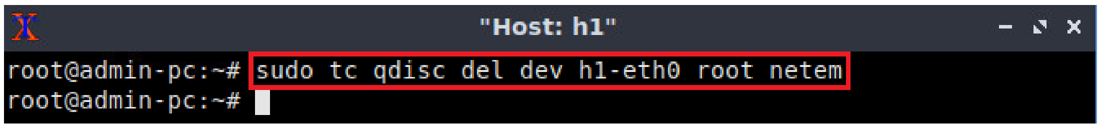
```
root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 13] local 10.0.0.1 port 40618 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 13]   0.00-1.00   sec  4.98 GBytes  42.7 Gbits/sec  543    178 KBytes
[ 13]   1.00-2.00   sec  4.98 GBytes  42.7 Gbits/sec  460    266 KBytes
[ 13]   2.00-3.00   sec  4.97 GBytes  42.7 Gbits/sec  230    199 KBytes
[ 13]   3.00-4.00   sec  4.96 GBytes  42.6 Gbits/sec  230    291 KBytes
[ 13]   4.00-5.00   sec  5.00 GBytes  42.9 Gbits/sec  411    349 KBytes
[ 13]   5.00-6.00   sec  4.98 GBytes  42.8 Gbits/sec  271    194 KBytes
[ 13]   6.00-7.00   sec  4.99 GBytes  42.9 Gbits/sec  586    202 KBytes
[ 13]   7.00-8.00   sec  4.99 GBytes  42.9 Gbits/sec  385    276 KBytes
[ 13]   8.00-9.00   sec  4.99 GBytes  42.8 Gbits/sec  228    194 KBytes
[ 13]   9.00-10.00  sec  4.98 GBytes  42.7 Gbits/sec  366    335 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 13]   0.00-10.00  sec  49.8 GBytes  42.8 Gbits/sec  3710             sender
[ 13]   0.00-10.04  sec  49.8 GBytes  42.6 Gbits/sec                  receiver

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

Figure 24. Retransmissions after packets corruption.

The figure above shows the retransmission values on each time interval (1 second). The total number of retransmitted packets, due to packet corruption, is 3710. This verifies that packet corruption was indeed, applied to the interface on host h1.

Step 4. In host h1's terminal, type the following command to delete previous configurations:

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



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

Figure 25. Deleting all rules on interface *h1-eth0*.

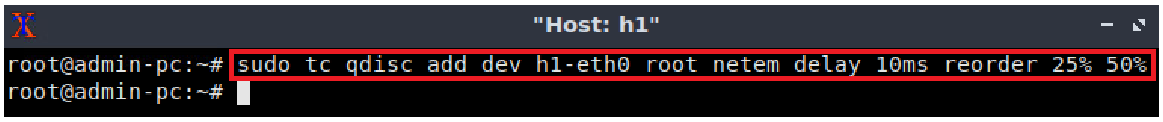
Step 5. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too. The summarized data on the server is similar to that of the client side's and must be interpreted in the same way.

5 Add packet reordering

Packets are sometimes not delivered in the same order they were sent. In order to emulate reordering in NETEM, the `reorder` option is used. Proceed with the steps below.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc add dev h1-eth0 root netem delay 10ms reorder 25% 50%
```



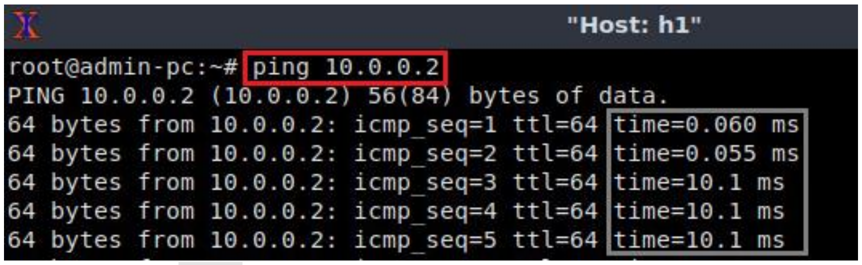
```
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem delay 10ms reorder 25% 50%
root@admin-pc:~# 
```

Figure 26. Adding packet reordering.

In this command, 25% of the packets (with a correlation value of 50%) will be sent immediately, while the remainder 75% will be delayed by 10ms.

Step 2. The user can verify the effect of packet reorder by using the `ping` command on host h1's terminal, press `Ctrl+c` to stop the test:

```
ping 10.0.0.2
```



```
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=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.055 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
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=10.1 ms
```

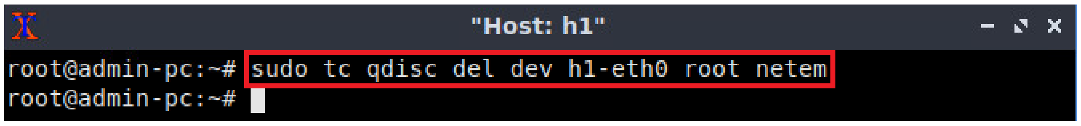
Figure 27. `ping` test illustrating the effect of packet reordering.

Consider the first four packets of the figure above. The first and second packets did not experience delay (one out of four, or 25%), while the next three packets experienced a delay of ~10 milliseconds (three out of four, or 75%). The measured reordering rate will tend to become closer to the configured reordering rate as more trials are performed.

It is possible that your first packet will experience delay, but this effect will eventually occur in future tests.

Step 3. In host h1's terminal, type the following command to delete previous configurations:

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



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

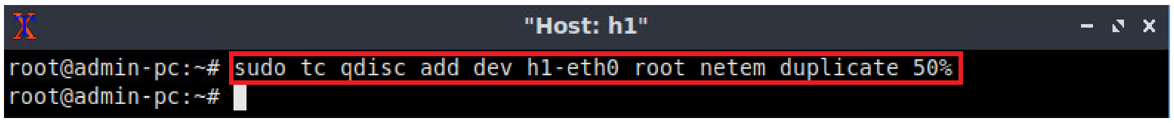
Figure 28. Deleting all rules on interface *h1-eth0*.

6 Add packet duplication

Duplicate packets may be present in networks as a result of retransmissions. NETEM provides the option `duplicate` to inject duplicate packets. Before introducing packet corruption, make sure to restore the default configuration of the interfaces on host h1 and host h2 by applying the commands of Section 3.3. Then, proceeds with the following steps.

Step 1. In host h1's terminal, type the following command:

```
sudo tc qdisc change dev h1-eth0 root netem duplicate 50%
```



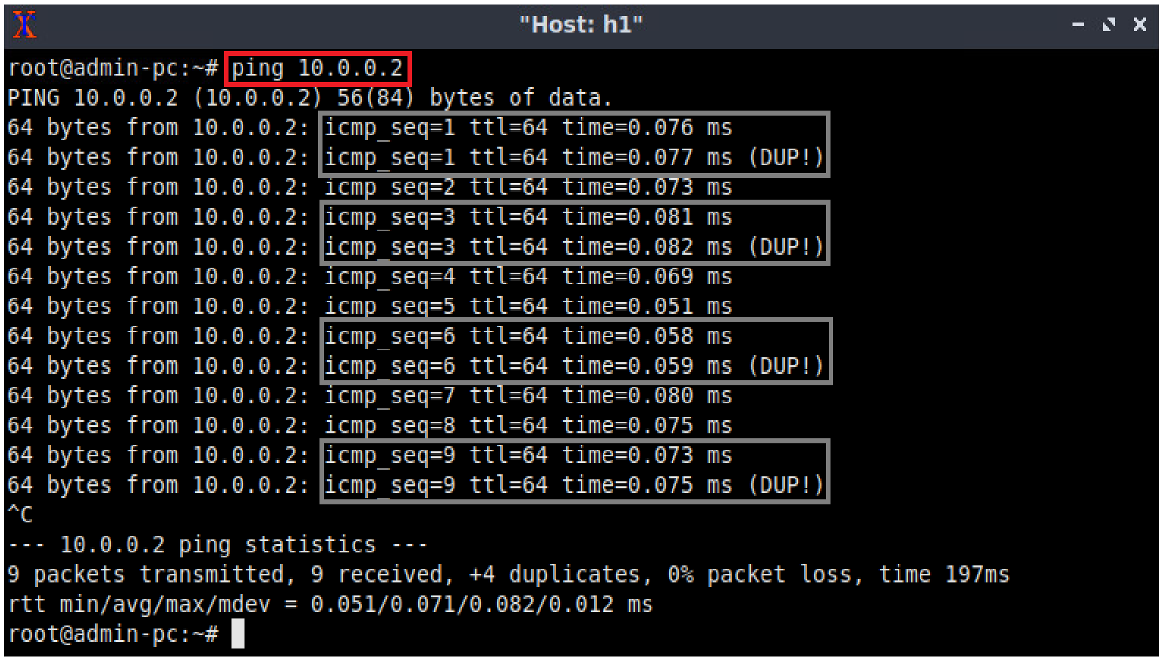
```
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root netem duplicate 50%
root@admin-pc:~#
```

Figure 29. Adding packet duplication.

The above command will produce a duplication of 50% (i.e., 50% of the packets will be received twice at the destination).

Step 2. The user can verify the effect of packet duplication by using the `ping` command on host h1's terminal, press `Ctrl+c` to stop the test:

```
ping 10.0.0.2
```



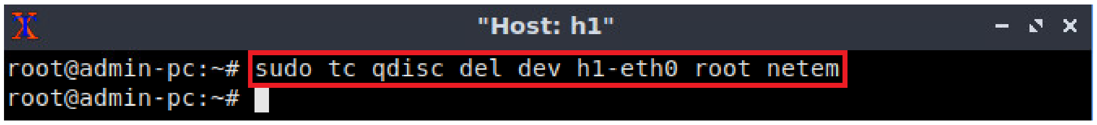
```
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.077 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.082 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.051 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.058 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.059 ms (DUP!)
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.075 ms (DUP!)
^C
--- 10.0.0.2 ping statistics ---
9 packets transmitted, 9 received, +4 duplicates, 0% packet loss, time 197ms
rtt min/avg/max/mdev = 0.051/0.071/0.082/0.012 ms
root@admin-pc:~#
```

Figure 30. `ping` test illustrating the effect of packet duplication.

The result above indicates that five duplicate packets were received. Duplicate packets are also marked with (DUP!). The measured rate of duplicate packets will tend to become closer to the configured rate as more trials are performed.

Step 3. In host h1's terminal, type the following command to delete previous configurations:

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



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

Figure 31. Deleting all rules on interface *h1-eth0*.

This concludes Lab 4. Stop the emulation and then exit out of MiniEdit.

References

1. Linux foundation. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem.
2. S. Hemminger, "Network emulation with NETEM," Linux conf au. 2005, pp. 18-23. 2005.
3. How to use the linux traffic control panagiotis vouzis [Online]. Available: https://netbeez.net/blog/how-to-use-the-linux-traffic-control.
4. M. Brown, F. Bolelli, N. Patriciello, "Traffic control howto," Guide to IP Layer Network, 2006.

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 (*tbf*), which is a Linux implementation of the Token Bucket algorithm on network interfaces.
3. Understand how to combine queueing disciplines in Linux Traffic Control (*tc*).
4. Combine *tbf* 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.

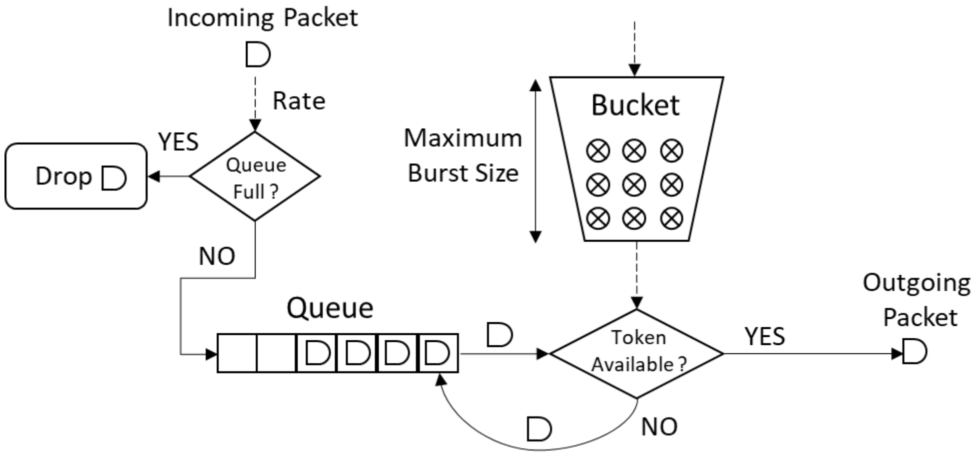


Figure 1. Token bucket filter.

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 (*tbf*), 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 `tbf`.

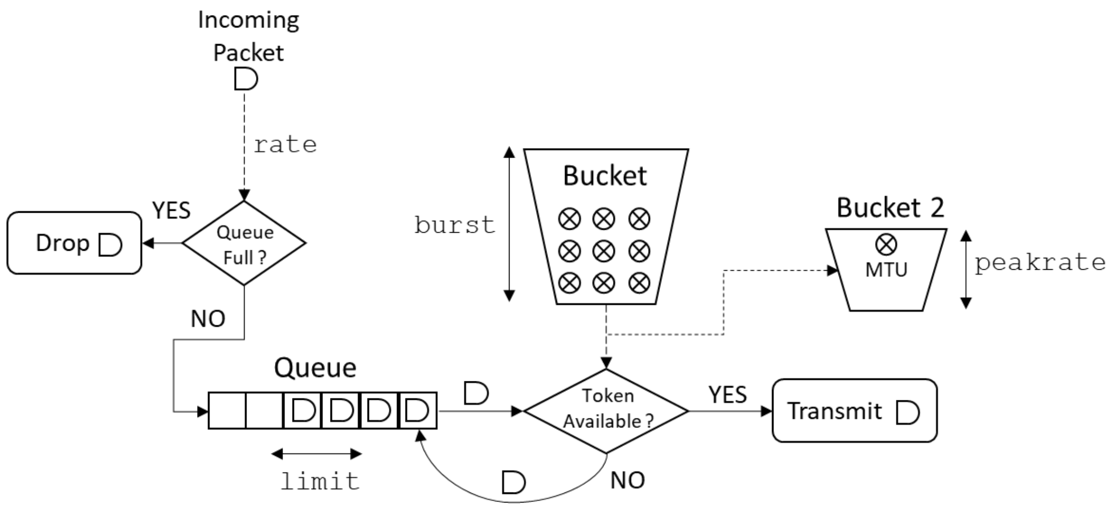


Figure 2. `tbf` parameters and architecture.

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` queueing 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.

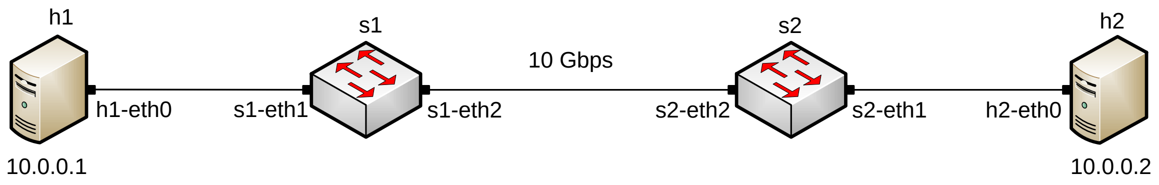


Figure 3. Lab topology.

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`.

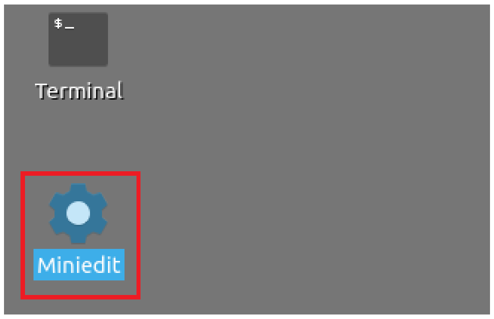


Figure 4. MiniEdit shortcut.

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*.

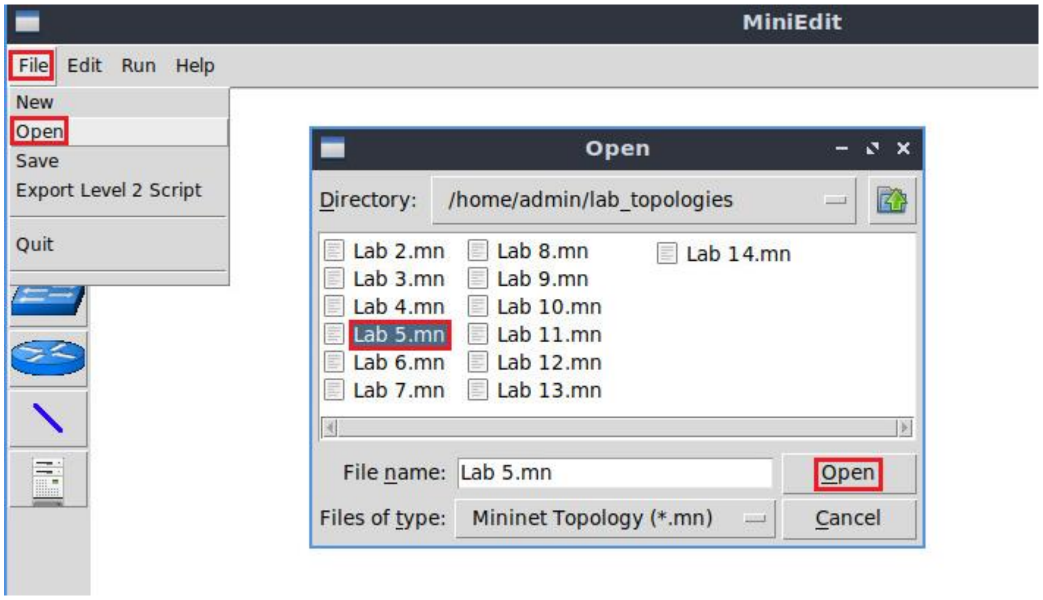


Figure 5. MiniEdit's *Open* dialog.

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.

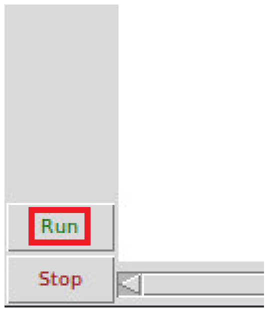


Figure 6. Running the emulation.

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.

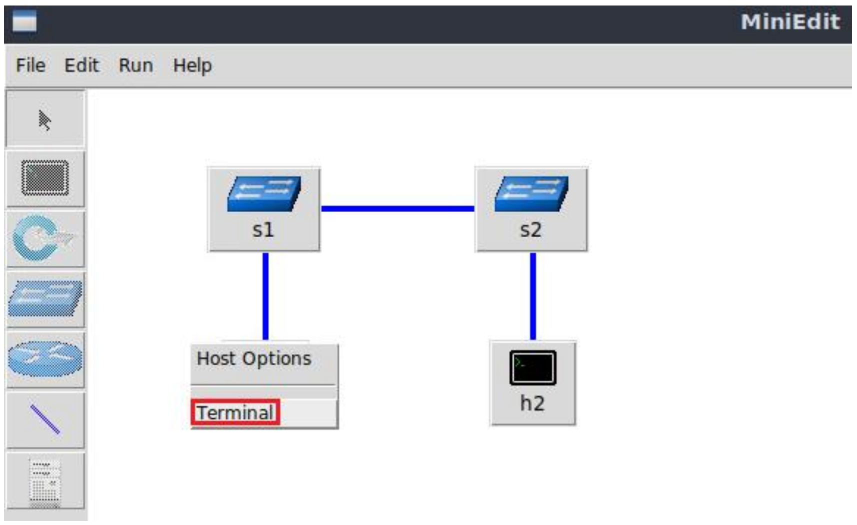
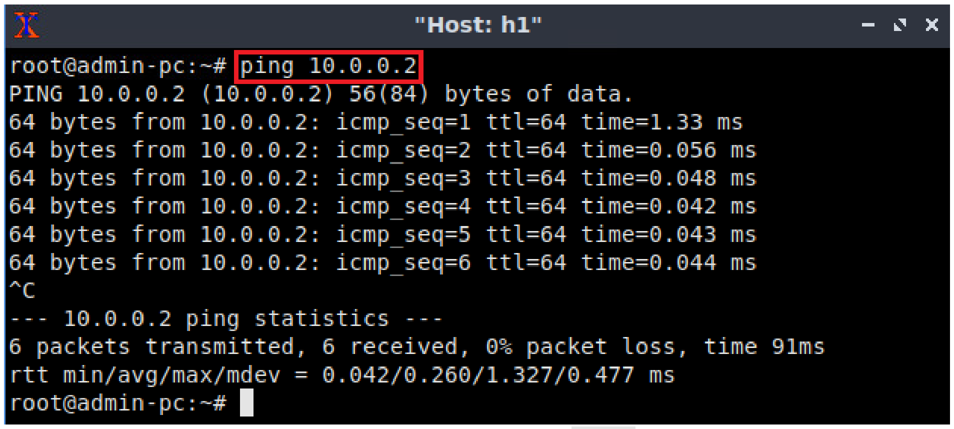


Figure 7. Opening a terminal on host h1.

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.



```
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.

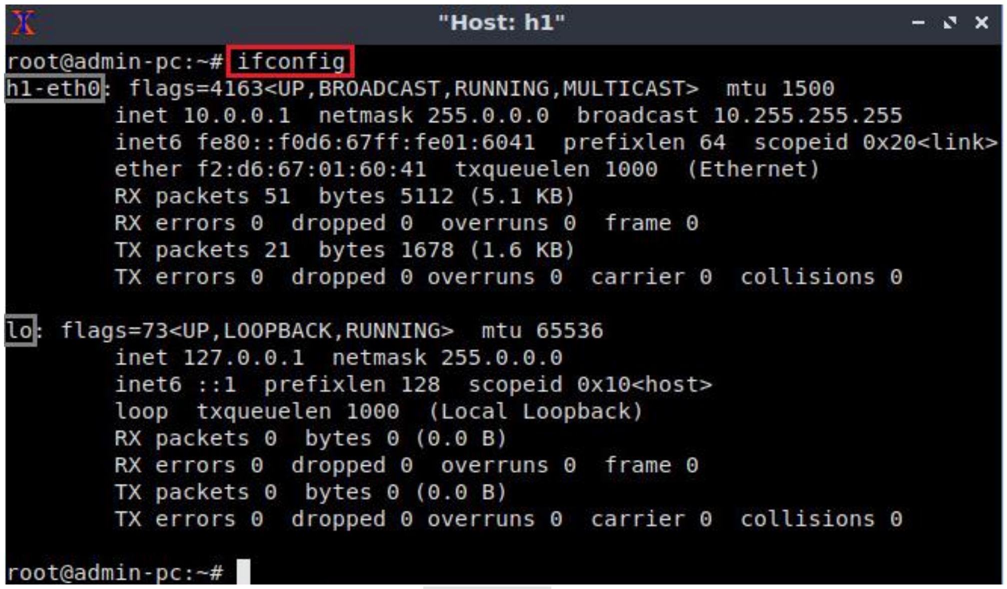
3 Rate limiting on end-hosts

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.

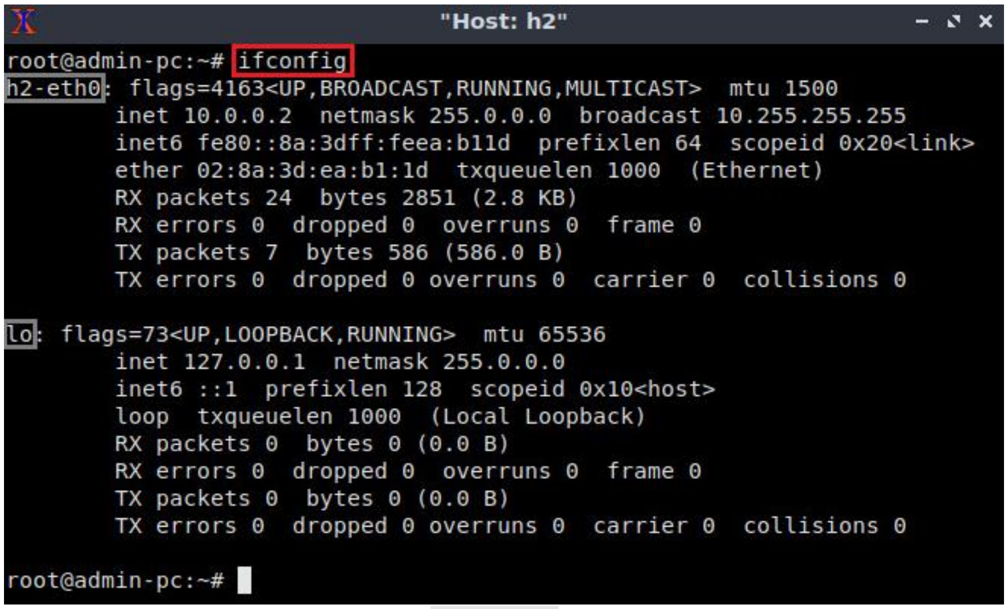


```
root@admin-pc:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::f0d6:67ff:fe01:6041  prefixlen 64  scopeid 0x20<link>
        ether f2:d6:67:01:60:41  txqueuelen 1000  (Ethernet)
        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.

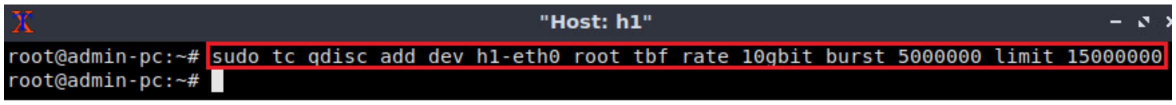
3.2 Emulating 10 Gbps high-latency WAN

In this section, you will use `tbf` 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 `tbf` 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:~#
```

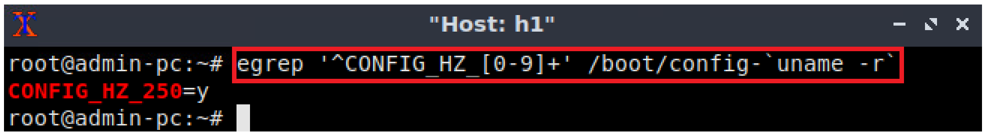
Figure 10. Limiting rate with TBF to 10 Gbps.

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.
- `tbf`: 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: `tbf` 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 following command:

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


```
root@admin-pc:~# egrep '^CONFIG_HZ_[0-9]+' /boot/config-`uname -r`
CONFIG_HZ_250=y
root@admin-pc:~#
```
Figure 11. Retrieving system's HZ.

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

10 Gbps = 10,000,000,000 bps

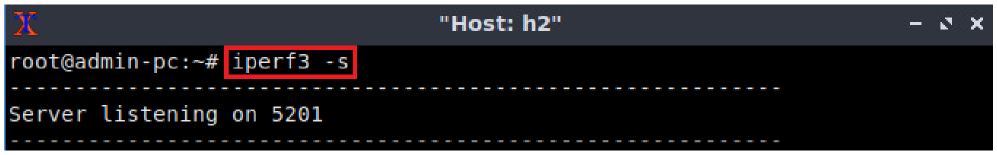
$$Burst = \frac{10{,}000{,}000{,}000}{250} = 40{,}000{,}000\ bits$$

Burst = 40,000,000 bits = 5,000,000 bytes

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

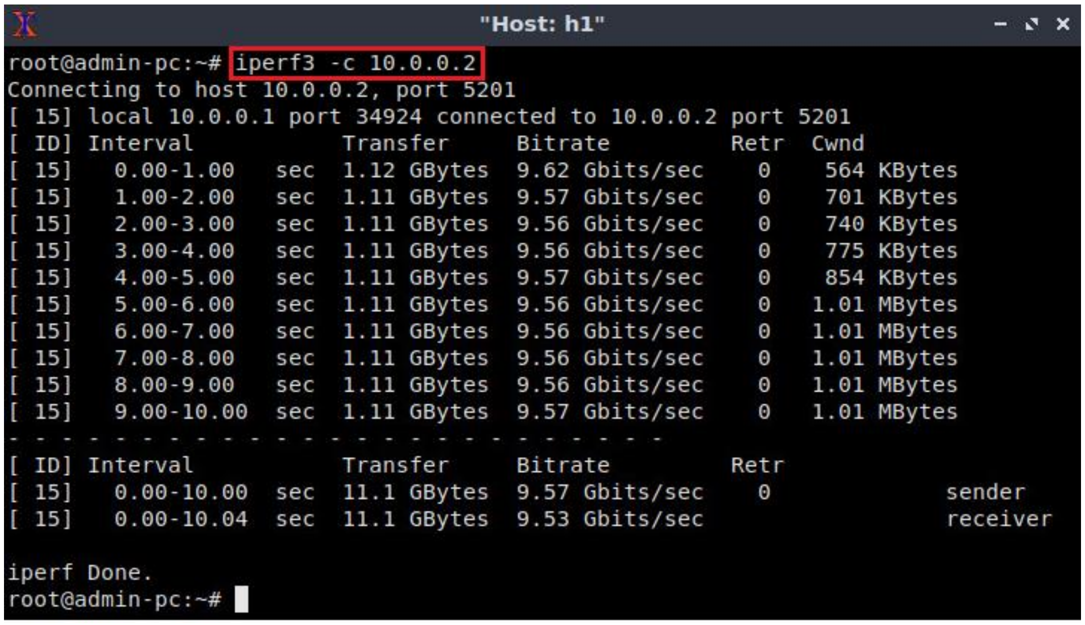
Step 2. The user can now verify the previous configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in the figure below:

```
iperf3 -s
```


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

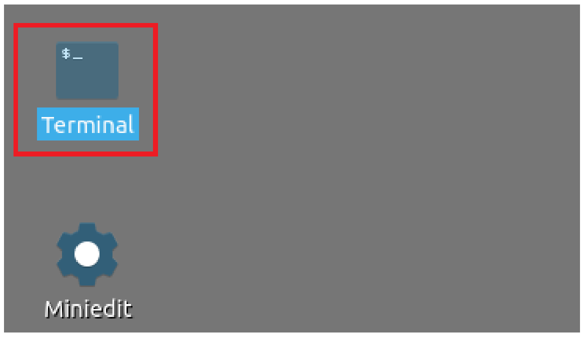
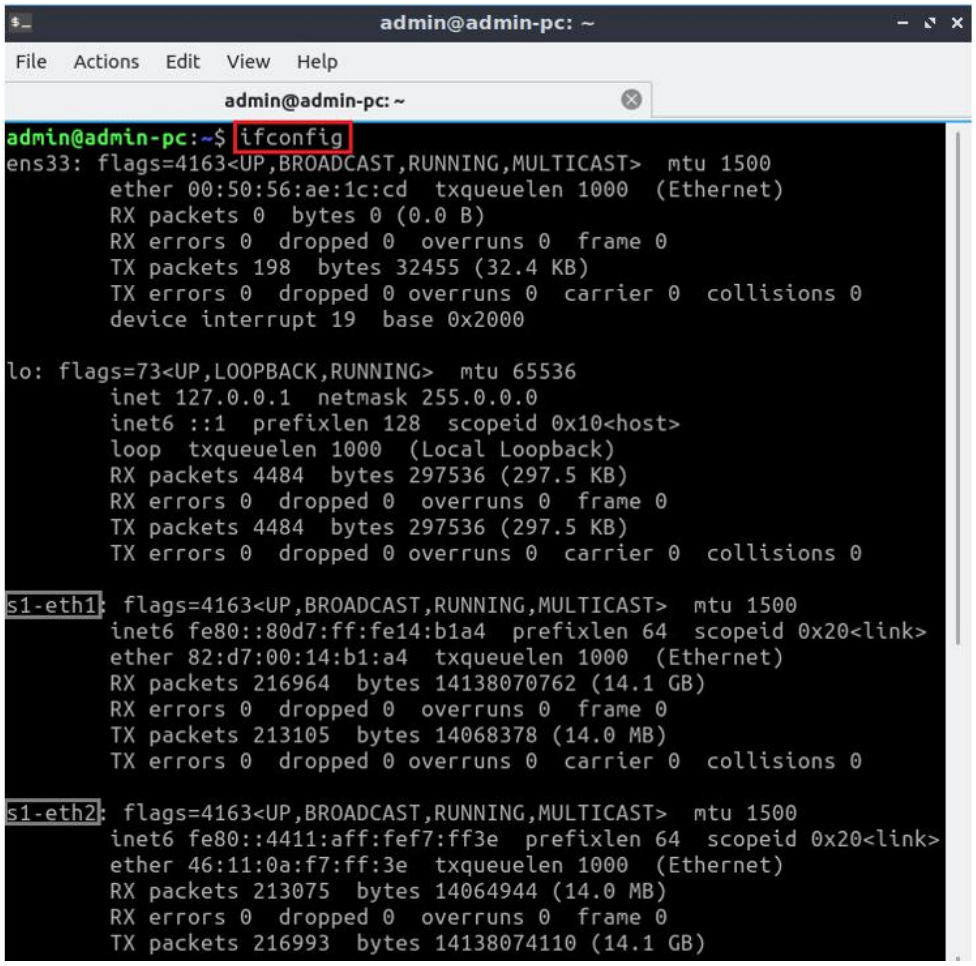


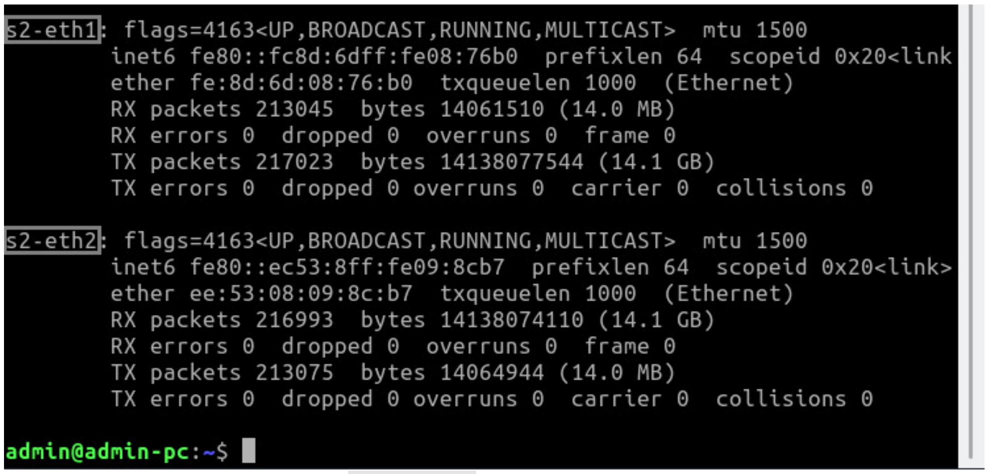
Figure 14. Shortcut to open a Linux terminal.

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:~$ 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:~$ 
```

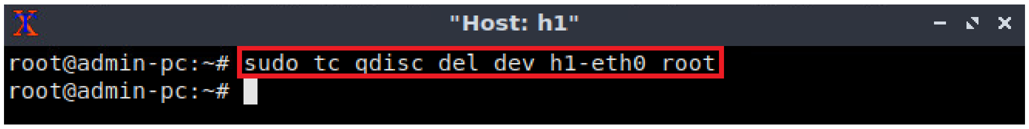
Figure 15. Output of `ifconfig` command on the client's terminal.

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
```



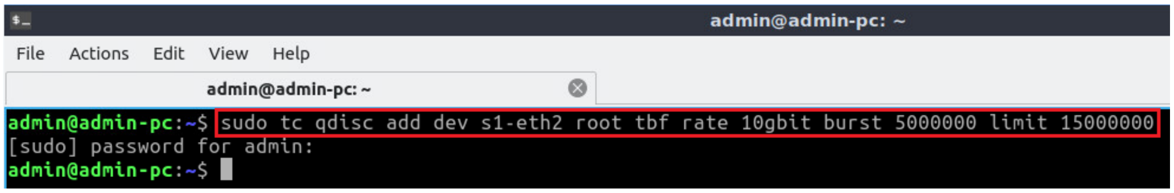
```
root@admin-pc:~# sudo tc qdisc del dev h1-eth0 root
root@admin-pc:~# 
```

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

Step 4. Apply `tbf` rate limiting rule on switch S1's interface which connects it to switch S2 (*s1-eth2*). In the Client1's terminal, type the command below. When prompted for a password, type `password` and hit enter. The `tbf` 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
```

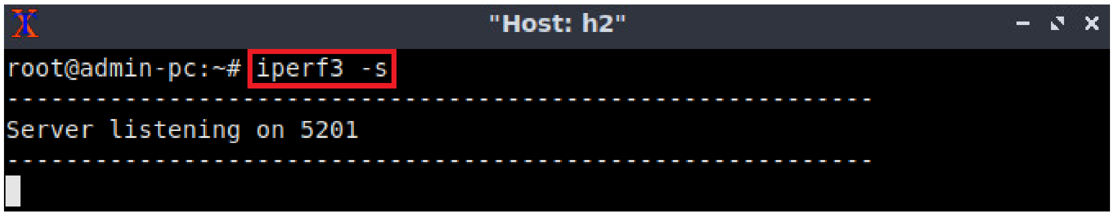


```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root tbf rate 10gbit burst 5000000 limit 15000000
[sudo] password for admin:
admin@admin-pc:~$ 
```

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:

```
iperf3 -s
```

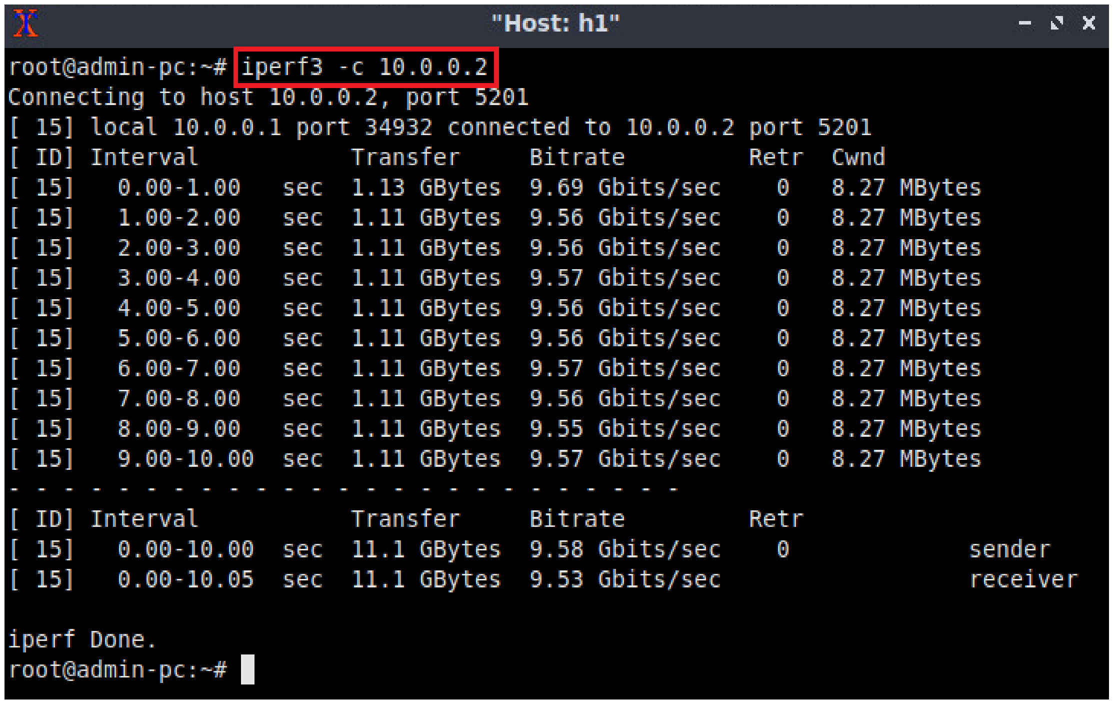


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

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:

```
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 34932 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec  1.13 GBytes  9.69 Gbits/sec    0   8.27 MBytes
[ 15]   1.00-2.00   sec  1.11 GBytes  9.56 Gbits/sec    0   8.27 MBytes
[ 15]   2.00-3.00   sec  1.11 GBytes  9.56 Gbits/sec    0   8.27 MBytes
[ 15]   3.00-4.00   sec  1.11 GBytes  9.57 Gbits/sec    0   8.27 MBytes
[ 15]   4.00-5.00   sec  1.11 GBytes  9.56 Gbits/sec    0   8.27 MBytes
[ 15]   5.00-6.00   sec  1.11 GBytes  9.56 Gbits/sec    0   8.27 MBytes
[ 15]   6.00-7.00   sec  1.11 GBytes  9.57 Gbits/sec    0   8.27 MBytes
[ 15]   7.00-8.00   sec  1.11 GBytes  9.56 Gbits/sec    0   8.27 MBytes
[ 15]   8.00-9.00   sec  1.11 GBytes  9.55 Gbits/sec    0   8.27 MBytes
[ 15]   9.00-10.00  sec  1.11 GBytes  9.57 Gbits/sec    0   8.27 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  11.1 GBytes  9.58 Gbits/sec    0             sender
[ 15]   0.00-10.05  sec  11.1 GBytes  9.53 Gbits/sec                  receiver

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

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.

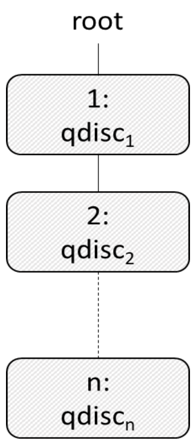
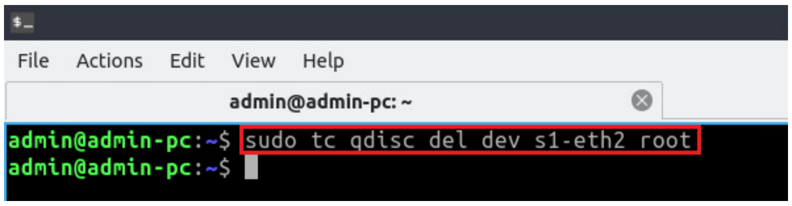


Figure 20. Chaining *qdiscs* hierarchy.

As shown in Figure 20, the first *qdisc* (*qdisc$_1$*) 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
```

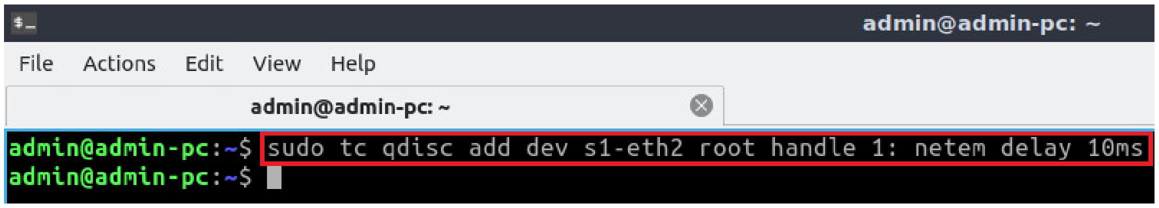


```
admin@admin-pc:~$ sudo tc qdisc del dev s1-eth2 root
admin@admin-pc:~$ 
```

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
```



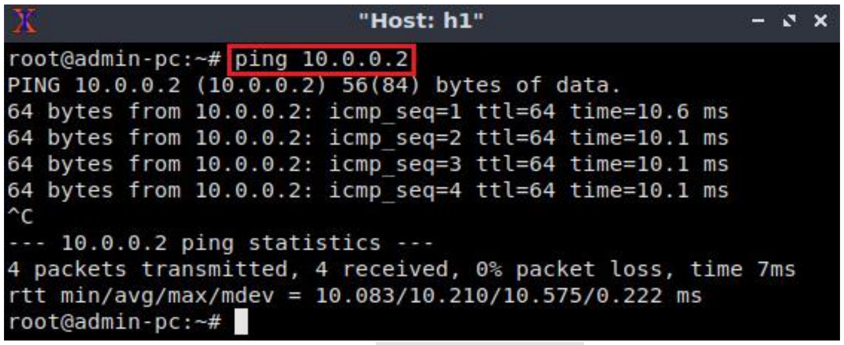
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 10ms
admin@admin-pc:~$
```

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
```



```
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:~#
```

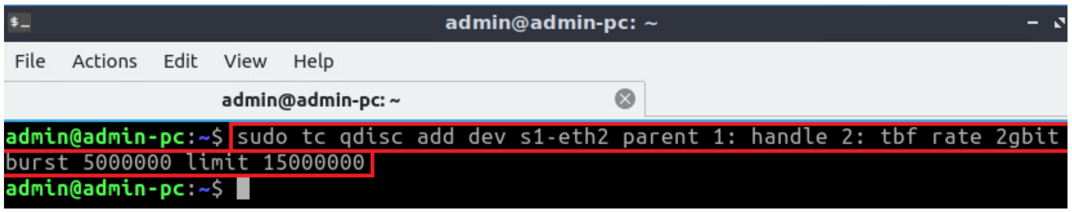
Figure 23. Output of `ping 10.0.0.2` command.

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 tbf, issue the command shown below on the client's terminal. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 2gbit burst 1000000
limit 2000000
```

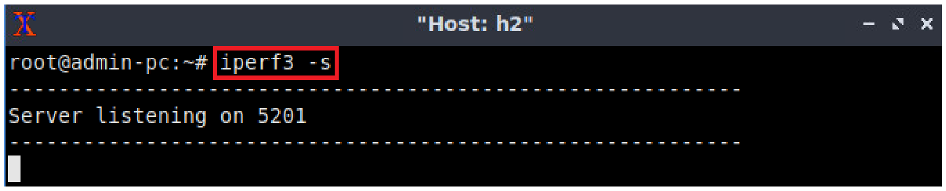


```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 2gbit
burst 5000000 limit 15000000
admin@admin-pc:~$
```

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
```

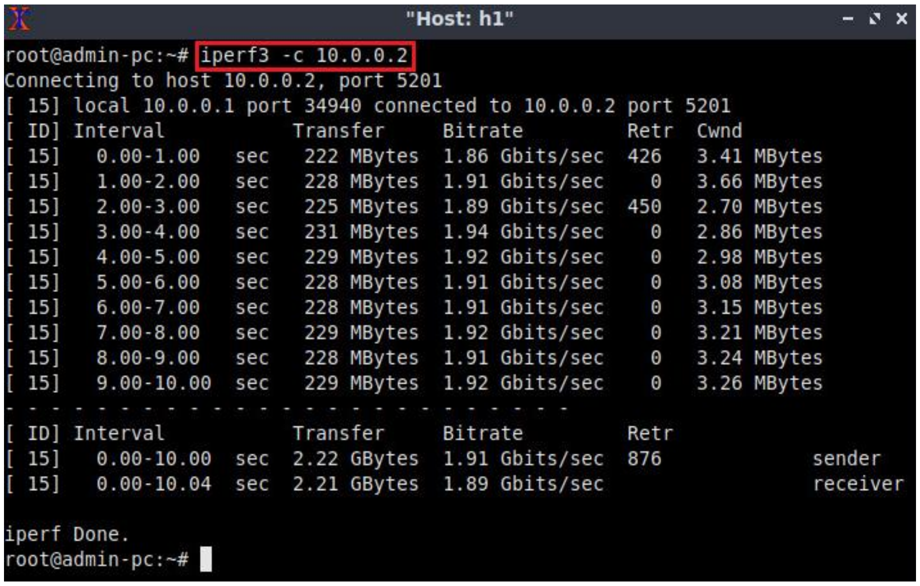


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

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
```



```
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.
2. How to use the linux traffic control panagiotis vouzis [Online]. Available: https://netbeez.net/blog/how-to-use-the-linux-traffic-control.

UNIVERSITY OF SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 6: Understanding Traditional TCP Congestion Control (HTCP, Cubic, Reno)

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 TCP 3
1.1 TCP review 4
1.2 TCP throughput 4
1.3 TCP packet loss event 5
1.4 Impact of packet loss in high-latency networks 6
2 Lab topology.......... 7
2.1 Starting host h1 and host h2 8
2.2 Emulating 10 Gbps high-latency WAN with packet loss 9
2.3 Testing connection 10
3 Introduction to sysctl 11
3.1 Read sysctl parameters 11
3.2 Write sysctl parameters 12
3.3 Configuring sysctl.conf file 12
4 Congestion control algorithms and sysctl 14
4.1 Inspect and install/load congestion control algorithms 15
4.2 Inspect the default (current) congestion control algorithm 16
4.3 Modify the default (current) congestion control algorithm 17
5 iPerf3 throughput test 17
5.1 Throughput test without delay 18
5.1.1 TCP Reno 18
5.1.2 Hamilton TCP (HTCP) 19
5.1.3 TCP Cubic 21
5.2 Throughput test with 30ms delay 22
5.2.1 TCP Reno 23
5.2.2 Hamilton TCP (HTCP) 24
5.2.3 TCP Cubic 26
References 27

Overview

This lab reviews key features and behavior of Transmission Control Protocol (TCP) that have a large impact on data transfers over high-throughput, high-latency networks. The lab describes the behavior of TCP's congestion control algorithm, its impact on throughput, and how to modify the congestion control algorithm in a Linux machine.

Objectives

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

1. Describe the basic operation of TCP congestion control algorithm and its impact on high-throughput networks.
2. Explain the concepts of congestion window, bandwidth probing, and Additive-Increase Multiplicative-Decrease (AIMD).
3. Understand TCP throughput calculation.
4. Understand the impact of packet loss on high-latency networks.
5. Deploy emulated WANs in Mininet.
6. Modify the TCP congestion control algorithm in Linux using *sysctl* tool.
7. Compare TCP Reno, HTCP, and Cubic with injected packet loss.
8. Compare TCP Reno, HTCP, and Cubic with both injected delay and packet loss.

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:

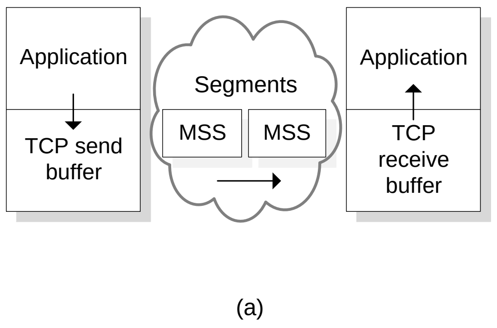
1. Section 1: Introduction to TCP.
2. Section 2: Lab topology.
3. Section 3: Introduction to *sysctl*.
4. Section 4: Congestion control algorithms and *sysctl*.
5. Section 5: iPerf3 throughput test.

1 Introduction to TCP

1.1 TCP review

Big data applications require the transmission of large amounts of data between end devices. Data must be correctly delivered from one device to another; e.g., from an instrument to a Data Transfer Node (DTN). Reliability is one of the services provided by TCP and a reason why TCP is the protocol used by most data transfer tools. Thus, understanding the behavior of TCP is essential for the design and operation of networks used to transmit big data.

TCP receives data from the application layer and places it in the TCP send buffer, as shown in Figure 1(a). Data is typically broken into Maximum Segment Size (MSS) units. Note that "segment" here refers to the Protocol Data Unit (PDU) at the transport layer, and sometimes the terms packet and segment are interchangeably used. The MSS is simply the Maximum Transmission Unit (MTU) minus the combined lengths of the TCP and IP headers (typically 40 bytes). Ethernet's normal MTU is 1,500 bytes. Thus, MSS's typical value is 1,460. The TCP header is shown in Figure 1(b).



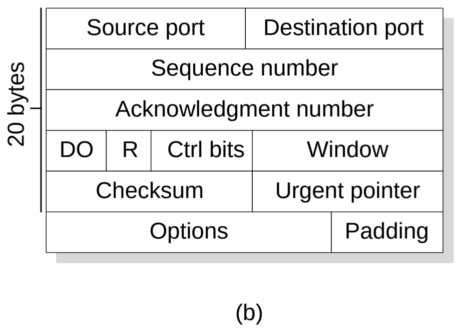


Figure 1. (a) TCP Connection. (b) TCP header.

For reliability, TCP uses two fields of the TCP header to convey information to the sender: sequence number and acknowledgement (ACK) number. The sequence number is the byte-stream number of the first byte in the segment. The acknowledgement number that the receiver puts in its segment is the sequence number of the next byte the receiver is expecting from the sender. In the example of Figure 2(a), after receiving the first two segments containing sequence number 90 (which contains bytes 90-99) and 100 (bytes 100-109), the receiver sends a segment with acknowledge number 110. This segment is called cumulative acknowledgement.

1.2 TCP throughput

The TCP rate limitation is defined by the receive buffer shown in Figure 1(a). If this buffer size is too small, TCP must constantly wait until an acknowledgement arrives before sending more segments. This limitation is removed by setting a large receive buffer size.

A second limitation is imposed by the congestion control mechanism operating at the sender side, which keeps track of a variable called congestion window. The congestion window, referred to as *cwnd* (in bytes), imposes a constraint on the rate at which a TCP sender can send traffic. The *cwnd* value is the amount of unacknowledged data at the sender. To see this, note that at the beginning of every Round-Trip Time (RTT), the sender can send *cwnd* bytes of data into the connection; at the end of the RTT the sender receives acknowledgments for the data. Thus, the sender's send rate is roughly *cwnd*/RTT bytes/sec. By adjusting the value of *cwnd*, the sender can therefore adjust the rate at which it sends data into the connection.

$$\text{TCP Throughput} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ [bytes/second]}$$

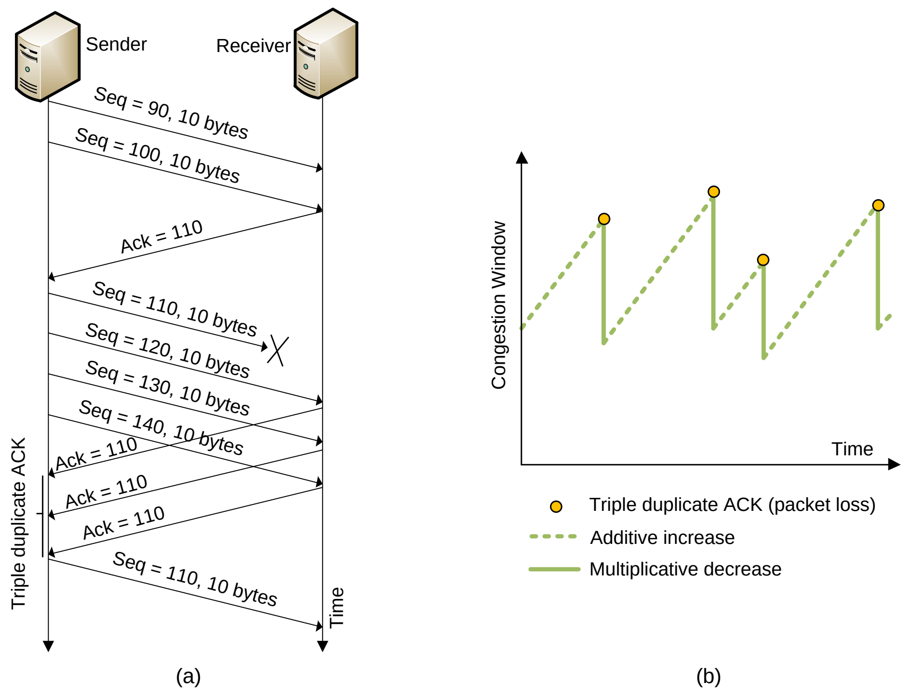


Figure 2. (a) TCP operation. (b) Adaptation of TCP's congestion window.

1.3 TCP packet loss event

TCP is a reliable transport protocol that requires each segment be acknowledged. If an acknowledgement for an outstanding segment is not received, TCP retransmits that segment. Alternatively, instead of waiting for a timeout-triggered retransmission, the sender can also detect a packet loss before the timeout by detecting duplicate ACKs. A duplicate ACK is an ACK that re-acknowledges a segment for which the sender has already received. If the TCP sender receives three duplicate ACKs for the same segment, TCP interprets this event as packet loss due to congestion and reduces the congestion window *cwnd* by half. This congestion window reduction is known as multiplicative decrease.

In steady state (ignoring the initial TCP period when a connection begins), a packet loss will be detected by a triple duplicate ACK. After decreasing *cwnd* by half, and as long as no other packet loss is detected, TCP will slowly increase *cwnd* again by 1 MSS per RTT. This congestion control phase essentially produces an additive increase in the congestion window. For this reason, TCP congestion control is referred to as an Additive-Increase Multiplicative-Decrease (AIMD) form of congestion control. AIMD gives rise to the "saw tooth" behavior shown in Figure 2(b), which also illustrates the idea of TCP "probing" for bandwidth—TCP linearly increases its congestion window size (and hence its transmission rate) until a triple duplicate-ACK event occurs. It then decreases its congestion window size by a factor of two but then again begins increasing it linearly, probing to see if there is additional available bandwidth.

1.4 Impact of packet loss in high-latency networks

During the additive increase phase, TCP only increases *cwnd* by 1 MSS every RTT period. This feature makes TCP very sensitive to packet loss on high-latency networks, where the RTT is large.

Consider Figure 3, which shows the TCP throughput of a data transfer across a 10 Gbps path. The packet loss rate is 1/22,000, or 0.0046%. The purple curve is the throughput in a loss-free environment; the green curve is the theoretical throughput computed according to the equation below, where *L* is the packet loss rate.

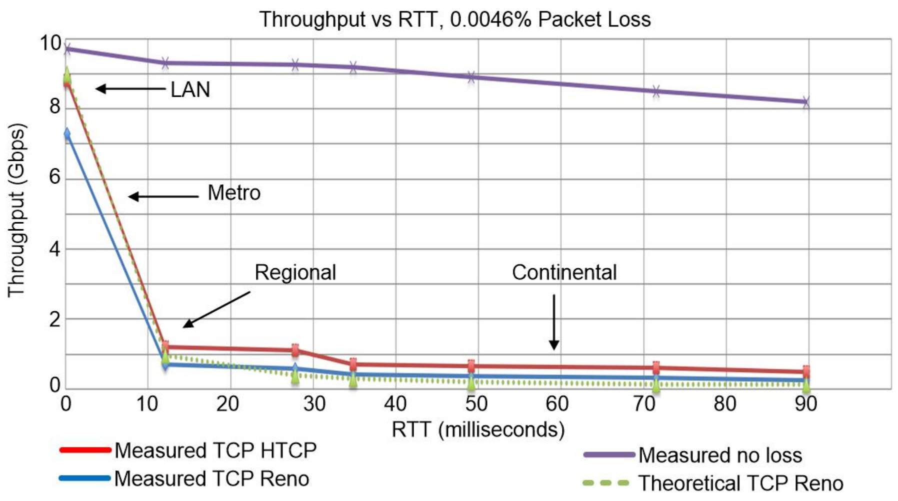


Figure 3. Throughput vs Round-Trip Time (RTT), for two devices connected via a 10 Gbps path. The performance of two TCP implementations are provided: Reno[1] (blue) and Hamilton TCP[2] (HTCP) (red). The theoretical performance with packet losses (green) and the measured throughput without packet losses (purple) are also shown[3].

$$\text{TCP Throughput} \approx \frac{\text{MSS}}{\text{RTT}\,\sqrt{L}} \quad \text{[bytes / second]}$$

The equation above indicates that the throughput of a TCP connection in steady state is directly proportional to the maximum segment size (MSS) and inversely proportional to the Round-Trip Time (RTT) and the square root of the packet loss rate (*L*). The red and blue curves are real throughput measurements of two popular implementations of TCP: Reno[1] and Hamilton TCP (HTCP)[2]. Because TCP interprets losses as network congestion, it reacts by decreasing the rate at which packets are sent. This problem is exacerbated as the latency increases between the communicating hosts. Beyond LAN transfers, the throughput decreases rapidly to less than 1 Gbps. This is often the case when research collaborators sharing data are geographically distributed.

TCP Reno is an early congestion control algorithm. TCP Cubic[4], HTCP[5], and BBR[6] are more recent congestion control algorithms, which have demonstrated improvements with respect to TCP Reno.

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.

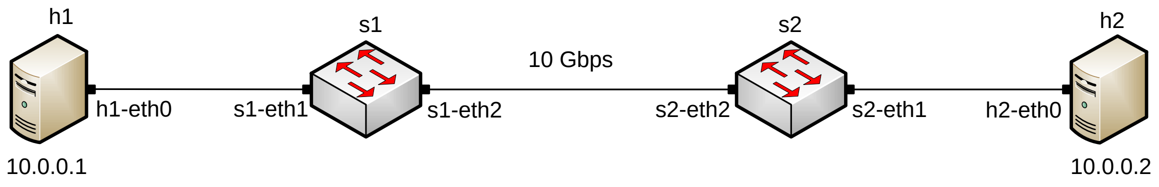


Figure 4. Lab topology.

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`.

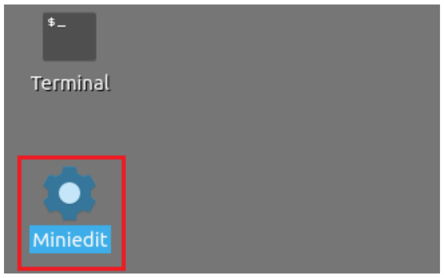


Figure 5. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 6.mn* topology file and click on *Open*.

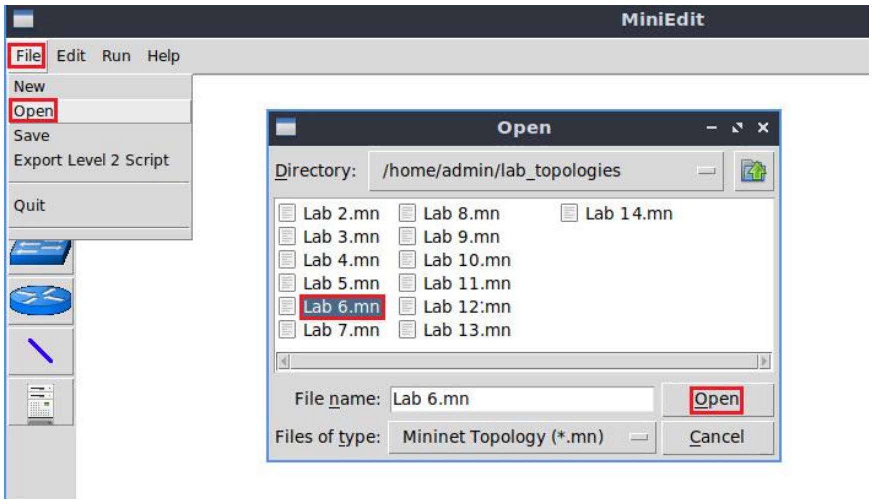


Figure 6. MiniEdit shortcut.

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.

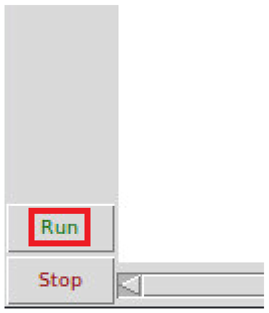


Figure 7. Running the emulation.

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 the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on host h1.

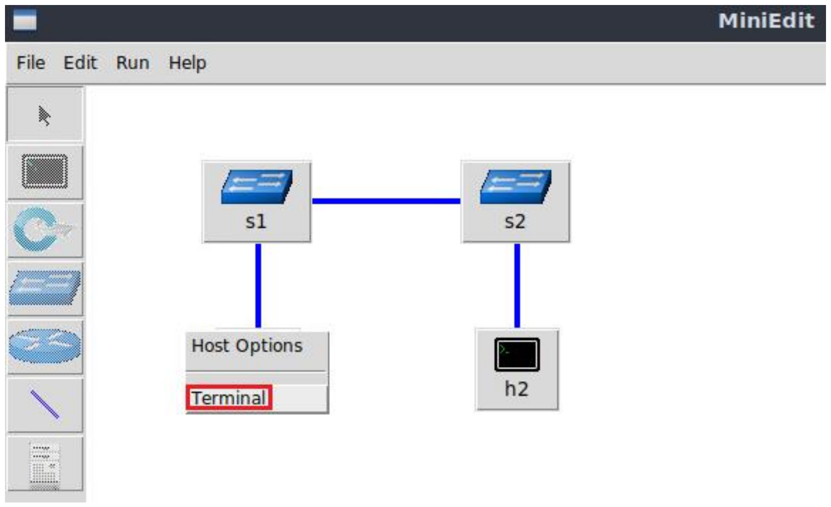
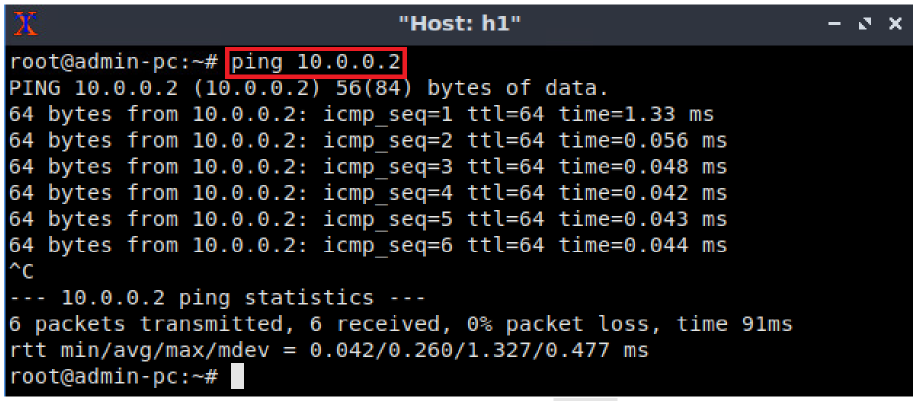


Figure 8. Opening a terminal on host h1.

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.



```
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 9. Connectivity test using `ping` command.

Figure 9 indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

2.2 Emulating 10 Gbps high-latency WAN with packet loss

This section emulates a high-latency WAN, which is used to validate the results observed in Figure 3. We will first set the bandwidth between host h1 and host h2 to 10 Gbps. Then we will emulate packet losses between switch S1 and switch S2 and measure the throughput.

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

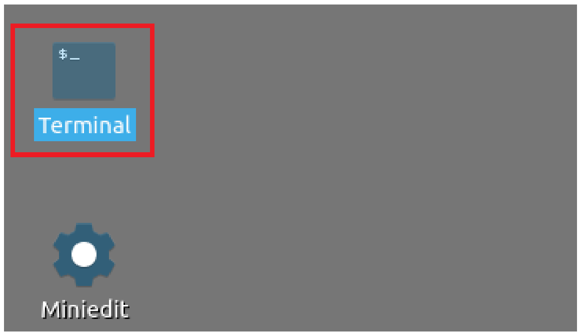
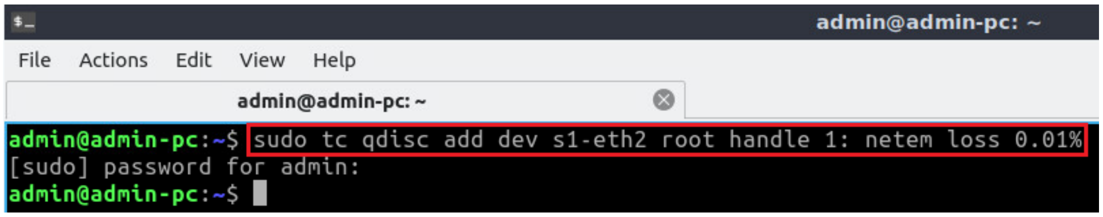


Figure 10. Shortcut to open a Linux terminal.

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.

```
sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 0.01%
```



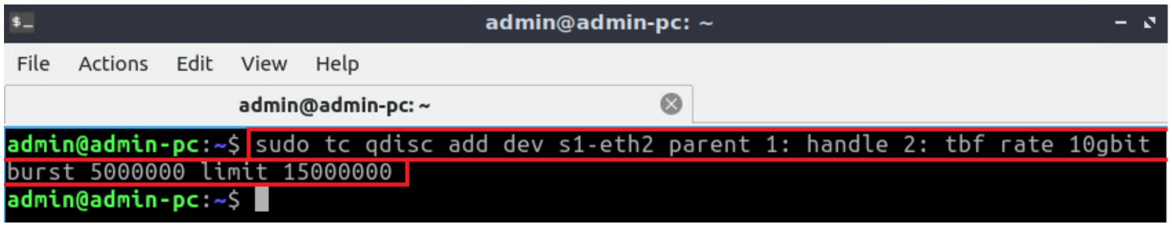
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 0.01%
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 11. Adding 0.01% packet loss rate to switch S1's *s1-eth2* interface.

Step 3. Modify the bandwidth of the link connecting the switch S1 and switch S2; on the same terminal, type the command below. This command sets the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit burst 5000000
limit 15000000
```



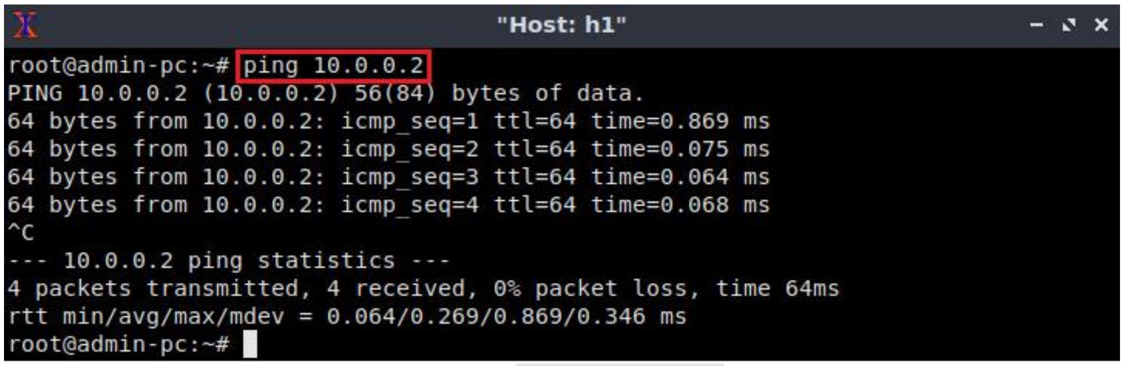
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$
```

Figure 12. Limiting the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface.

2.3 Testing connection

To test connectivity, you can use the command `ping`.

Step 1. On the terminal of host h1, type `ping 10.0.0.2`. 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.



```
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=0.869 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.068 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 64ms
rtt min/avg/max/mdev = 0.064/0.269/0.869/0.346 ms
root@admin-pc:~#
```

Figure 13. 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 0.064, 0.269, 0.869, and 0.346 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.

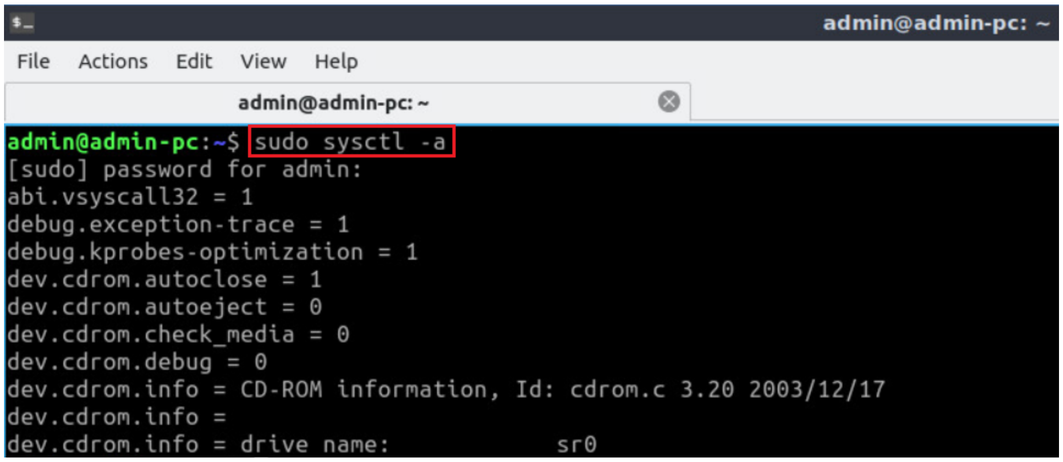
Step 2. On the terminal of host h2, type `ping 10.0.0.1`. The ping output in this test should be relatively similar to the results of the test initiated by host h1 in Step 1. To stop the test, press `Ctrl+c`.

3 Introduction to sysctl

sysctl is a tool for dynamically changing parameters in the Linux operating system[7]. It allows users to modify kernel parameters dynamically without rebuilding the Linux kernel.

Step 1. Run the command below on the Client1's terminal. When prompted for a password, type `password` and hit enter.

```
sudo sysctl -a
```



```
admin@admin-pc:~$ sudo sysctl -a
[sudo] password for admin:
abi.vsyscall32 = 1
debug.exception-trace = 1
debug.kprobes-optimization = 1
dev.cdrom.autoclose = 1
dev.cdrom.autoeject = 0
dev.cdrom.check_media = 0
dev.cdrom.debug = 0
dev.cdrom.info = CD-ROM information, Id: cdrom.c 3.20 2003/12/17
dev.cdrom.info =
dev.cdrom.info = drive name:            sr0
```

Figure 14. Listing all system parameters in Linux.

This command produces a large output containing the kernel parameters and their values. This is represented in a key-value pair. For instance, `net.ipv4.ip_forward = 0` implies that the key `net.ipv4.ip_forward` has the value `0`.

3.1 Read sysctl parameters

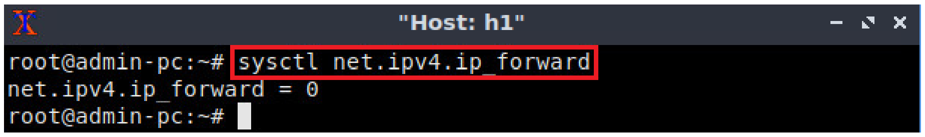
It is often useful to search for specific keys without having to manually locate the needed key. This can be achieved using the following command:

```
sysctl <key>
```

Where *<key>* is replaced by the needed key. For example, the command `sysctl net.ipv4.ip_forward` returns `net.ipv4.ip_forward = 0`.

Step 1. Run the following command on the host h1's terminal:

```
sysctl net.ipv4.ip_forward
```



```
root@admin-pc:~# sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
root@admin-pc:~#
```

Figure 15. Reading the value of a given key.

3.2 Write sysctl parameters

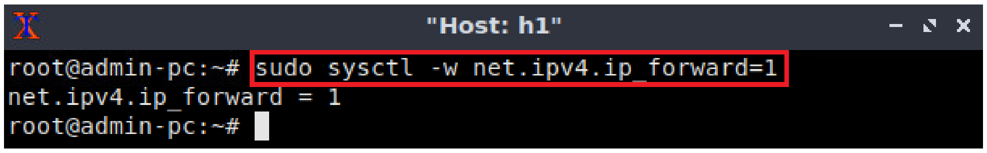
It is also very useful to modify kernel parameters on the fly. The `-w` switch is added to the sysctl to "write" a value for a specific key.

```
sysctl -w <key>=<value>
```

Step 1. For example, if the user decides to enable IP forwarding (i.e., to configure a device as a router), then the following command is used:

```
sudo sysctl -w net.ipv4.ip_forward=1
```

Run the above command on the host h1's terminal:



```
root@admin-pc:~# sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@admin-pc:~#
```

Figure 16. Modifying a system parameter.

The changes made to a parameter using this command are temporary. Therefore, a new boot resets the value of a key to its default value. Also, when stopping MiniEdit's emulation, the configured parameters are reset.

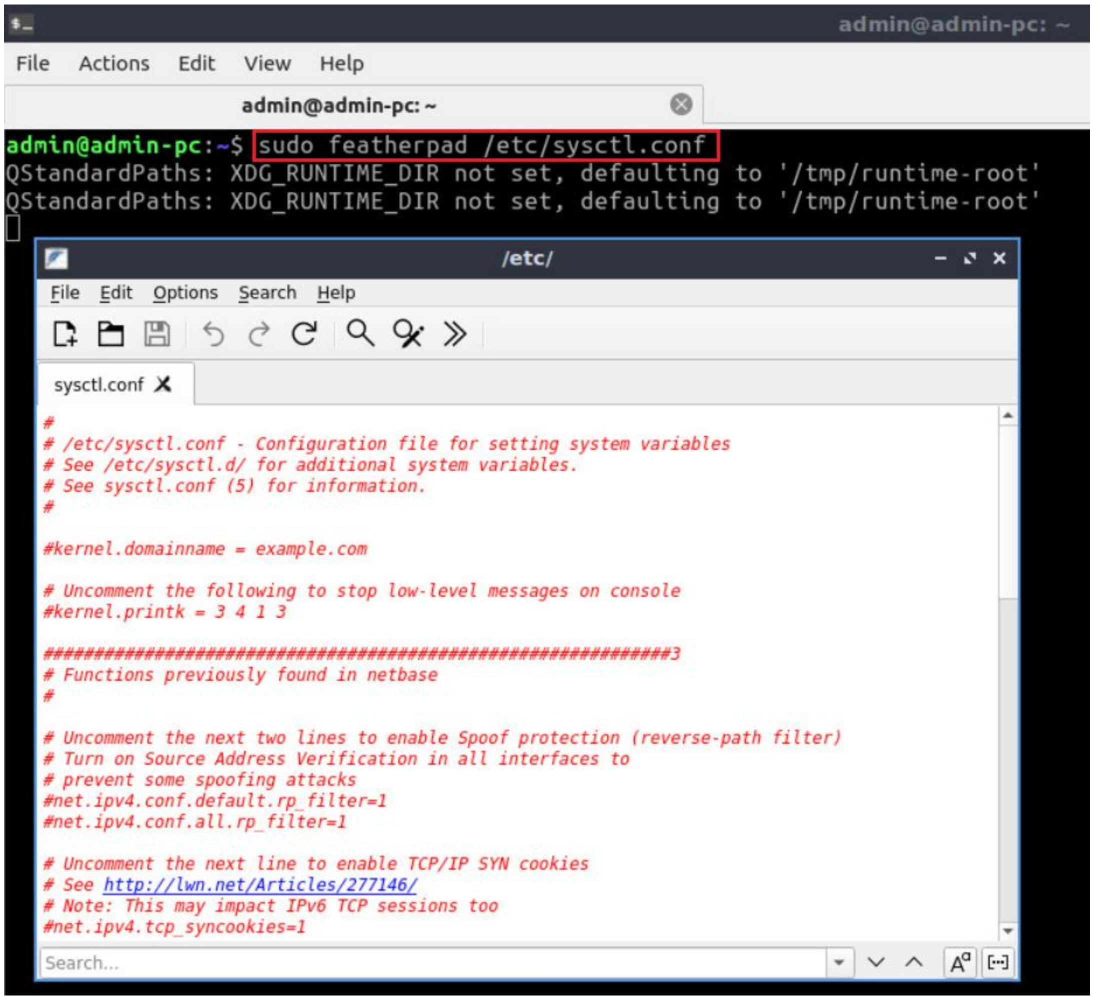
3.3 Configuring sysctl.conf file

If the user wishes to permanently modify the value of a specific key, then the key-value pair must be stored within the file */etc/sysctl.conf*.

Step 1. In the Linux terminal, open the */etc/sysctl.conf* file using your favorite text editor. Run the following command on the Client1's terminal. When prompted for a password, type `password` and hit enter.

```
sudo featherpad /etc/sysctl.conf
```

This is a text file that can be edited in any text editor (`vim`, `nano`, etc.). For simplicity, we use a Graphical User Interface (GUI)-based text editor (`featherpad`).



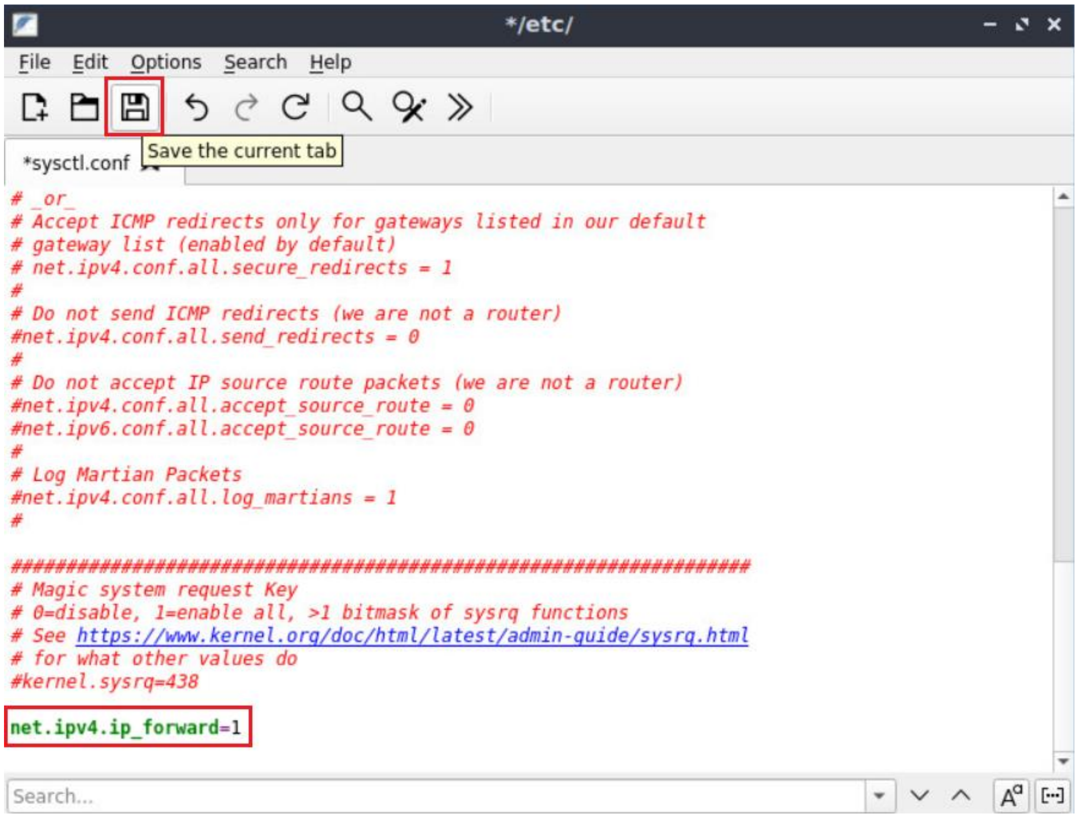
```
admin@admin-pc:~$ sudo featherpad /etc/sysctl.conf
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

```
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3

##############################################################3
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1
```

Figure 17. Opening the */etc/sysctl.conf* file.

Step 2. Keys and values are appended to this file. Enable IP forwarding permanently on the system by append `net.ipv4.ip_forward=1` to the */etc/sysctl.conf* file and save it. Once you have saved the file, close the text editor.

```
net.ipv4.ip_forward=1
```



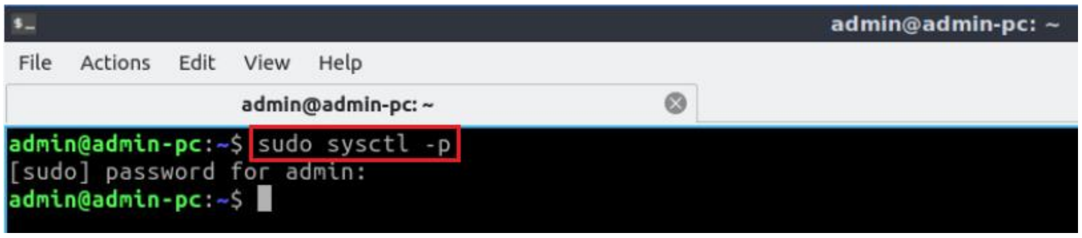
```
# _or_
# Accept ICMP redirects only for gateways listed in our default
# gateway list (enabled by default)
# net.ipv4.conf.all.secure_redirects = 1
#
# Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#

###################################################################
# Magic system request Key
# 0=disable, 1=enable all, >1 bitmask of sysrq functions
# See https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html
# for what other values do
#kernel.sysrq=438

net.ipv4.ip_forward=1
```

Figure 18. Appending key+value to the */etc/sysctl.conf* file and saving.

Step 3. To refresh the system with the new parameters, the `-p` switch is passed to the `sysctl` command as follows:

```
sudo sysctl -p
```

When prompted for a password, type `password` and hit enter.



```
admin@admin-pc:~$ sudo sysctl -p
[sudo] password for admin:
admin@admin-pc:~$ 
```

Figure 19. Loading new *sysctl.conf* parameters.

Now, even after a new system boot (or reboot), the system will have IP forwarding enabled.

4 Congestion control algorithms and sysctl

Congestion control algorithms can be inspected and modified using the `sysctl` command and the */etc/sysctl.conf* file. Specifically, the following operations are possible:

1. Check the installed congestion control algorithms on the system.

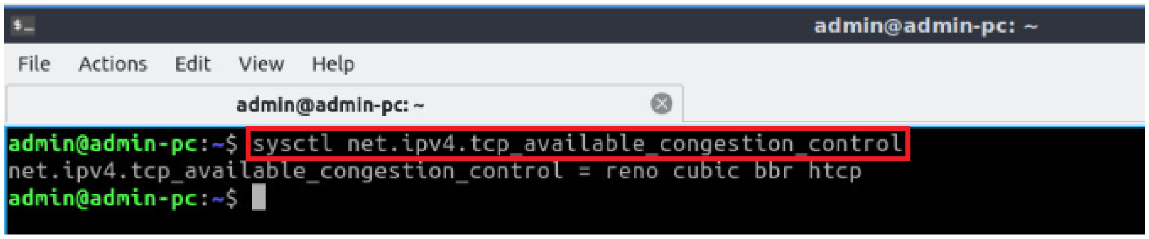
2. Inspect the default congestion control algorithm (i.e., the current algorithm used by the system).
3. Modify the congestion control algorithm.

4.1 Inspect and install/load congestion control algorithms

In Linux, it is possible to check the available TCP congestion control algorithms installed on the system with the command below.

Step 1. Execute the command below on the Client1's terminal.

```
sysctl net.ipv4.tcp_available_congestion_control
```



```
admin@admin-pc:~$ sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = reno cubic bbr htcp
admin@admin-pc:~$
```

Figure 20. Displaying the system's available congestion control algorithms.

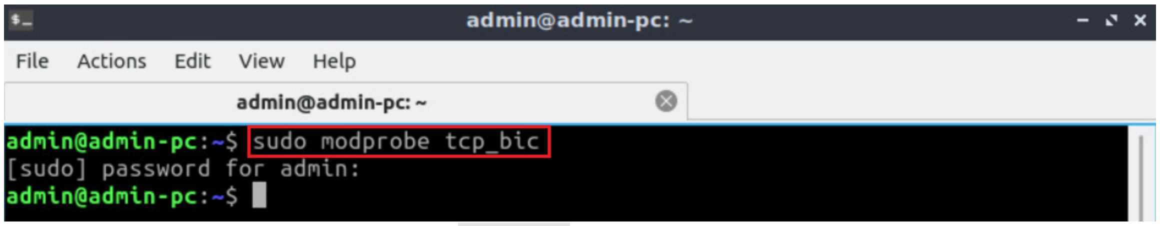
Usually, the default congestion control algorithm is CUBIC or Reno, depending on the installed operating system. A list of some of the possible output is:

- `reno`: Traditional TCP used by almost all other Operating Systems. Characterized by slow start, congestion avoidance, and fast retransmission via triple duplicate ACKs.
- `cubic`: CUBIC-TCP. Optimized congestion control algorithm for high bandwidth networks with high latency. Operates in a similar but more systematic fashion than BIC-TCP, in which its congestion window is a cubic function of time since the last packet loss, with the inflection point set to the window prior to the congestion event.
- `bic`: BIC-TCP. Congestion window utilizes a binary search algorithm to find the largest congestion window that will last the maximum amount of time.
- `htcp`: Hamilton TCP. A loss-based algorithm using additive-increase and multiplicative-decrease to control TCP's congestion window.
- `vegas`: TCP Vegas. Emphasizes packet delay, rather than packet loss, as a signal to help determine the rate at which to send packets.
- `bbr`: a new algorithm, discussed in future labs. Measures bottleneck bandwidth and Round-Trip Propagation (RTP) time in its execution of congestion control.

If the above command does not return a specific congestion control algorithm, it means that it is not loaded on the distribution.

Step 2. The command used in Step 1 listed three algorithms: `reno cubic bbr`. To install a new algorithm, its corresponding kernel module must be loaded. This can be done using `insmod` or `modprobe` commands. For example, to load the BIC-TCP module, use the following command on the Client1's terminal:

```
sudo modprobe tcp_bic
```



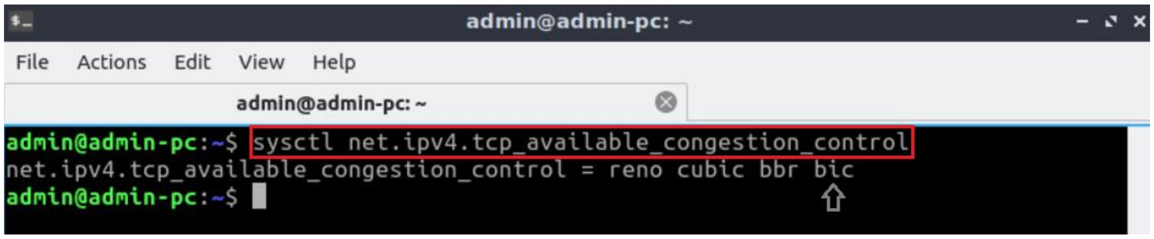
```
admin@admin-pc:~$ sudo modprobe tcp_bic
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 21. Loading `tcp_bic` module into the Linux kernel.

`modprobe` and `insmod` commands require high `sudo` privileges to insert kernel modules. When prompted for a password, type `password` and hit enter.

Step 3. To verify that the BIC-TCP algorithm is loaded, execute the below command on the Client1's terminal.

```
sysctl net.ipv4.tcp_available_congestion_control
```



```
admin@admin-pc:~$ sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = reno cubic bbr bic
admin@admin-pc:~$
```

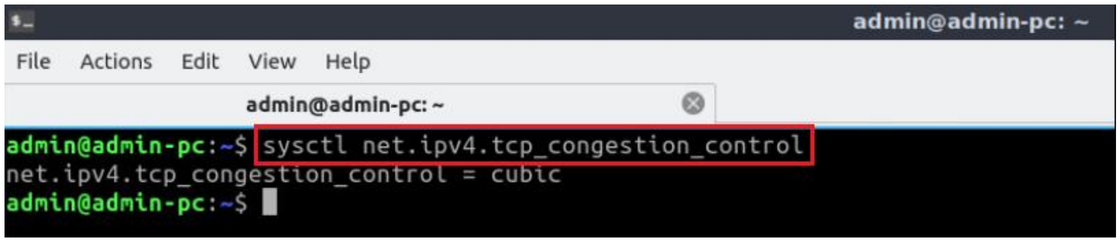
Figure 22. Displaying the system's available congestion control algorithms after loading TCP-BIC.

4.2 Inspect the default (current) congestion control algorithm

To check which TCP congestion control is currently being used by the Linux kernel, the *net.ipv4.tcp_congestion_control sysctl* key is read. This key can be read on an end-host's terminal (host h1 or host h2) or on the Client1's terminal.

Step 1. Execute the following command on the Client1's terminal to determine the current congestion control algorithm.

```
sysctl net.ipv4.tcp_congestion_control
```



```
admin@admin-pc:~$ sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = cubic
admin@admin-pc:~$
```

Figure 23. Current TCP congestion control algorithm.

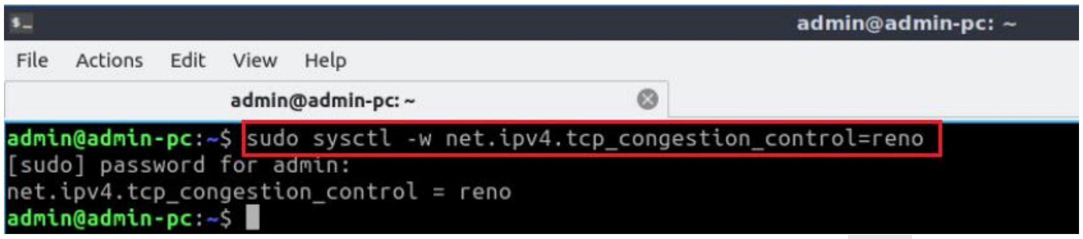
The output shows that the default congestion control algorithm is Cubic. Note that applications can set this value (congestion control algorithm) for individual connections.

4.3 Modify the default (current) congestion control algorithm

To temporarily change the TCP congestion control algorithm, the `sysctl` command is used with the `-w` switch on the *net.ipv4.tcp_congestion_control* key.

Step 1. To modify the current algorithm to TCP Reno, the following command is used. Execute the command below on the Client1's terminal. When prompted for a password, type `password` and hit enter.

```
sudo sysctl -w net.ipv4.tcp_congestion_control=reno
```



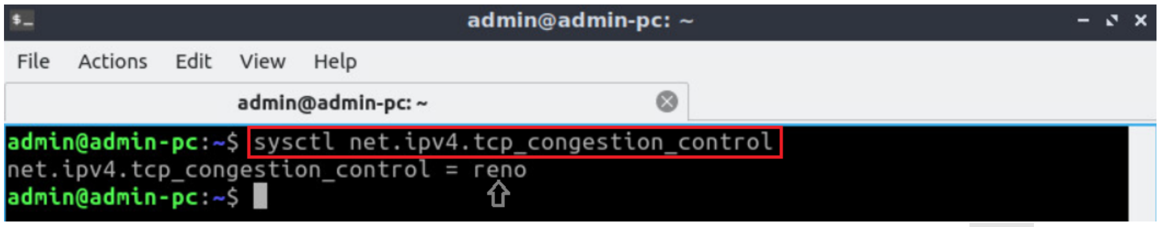
```
admin@admin-pc:~$ sudo sysctl -w net.ipv4.tcp_congestion_control=reno
[sudo] password for admin:
net.ipv4.tcp_congestion_control = reno
admin@admin-pc:~$
```

Figure 24. Modifying the congestion control algorithm to `reno`.

If no error occurred in the assignment (e.g., the module is not installed on the system), the output echoes back the new key-value pair, i.e.: `net.ipv4.tcp_congestion_control=reno`

Step 2. Execute the following command on the Client1's terminal to determine the current congestion control algorithm.

```
sysctl net.ipv4.tcp_congestion_control
```



```
admin@admin-pc:~$ sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = reno
admin@admin-pc:~$
```

Figure 25. Current TCP congestion control algorithm after modifying to `reno`.

The output shows that the default congestion control algorithm is now Reno instead of Cubic.

5 iPerf3 throughput test

In this section, the throughput between host h1 and host h2 is measured using different congestion control algorithms, namely Reno, HTCP, and Cubic. Moreover, the test is repeated using various injected delays to observe the throughput variations depending on each congestion control algorithm and the selected RTT.

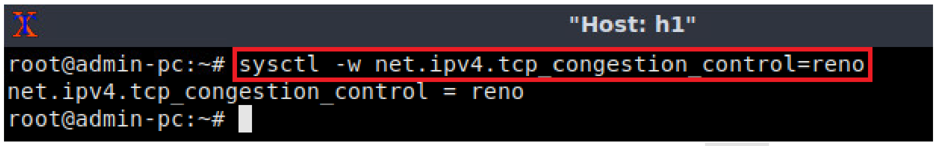
5.1 Throughput test without delay

In this test, we measure the throughput between host h1 and host h2 without introducing delay on the switch S1's *s1-eth2* interface.

5.1.1 TCP Reno

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Reno by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=reno
```

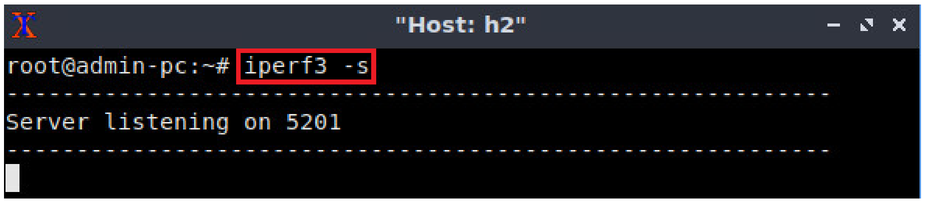


```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
root@admin-pc:~#
```

Figure 26. Changing TCP congestion control algorithm to `reno` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

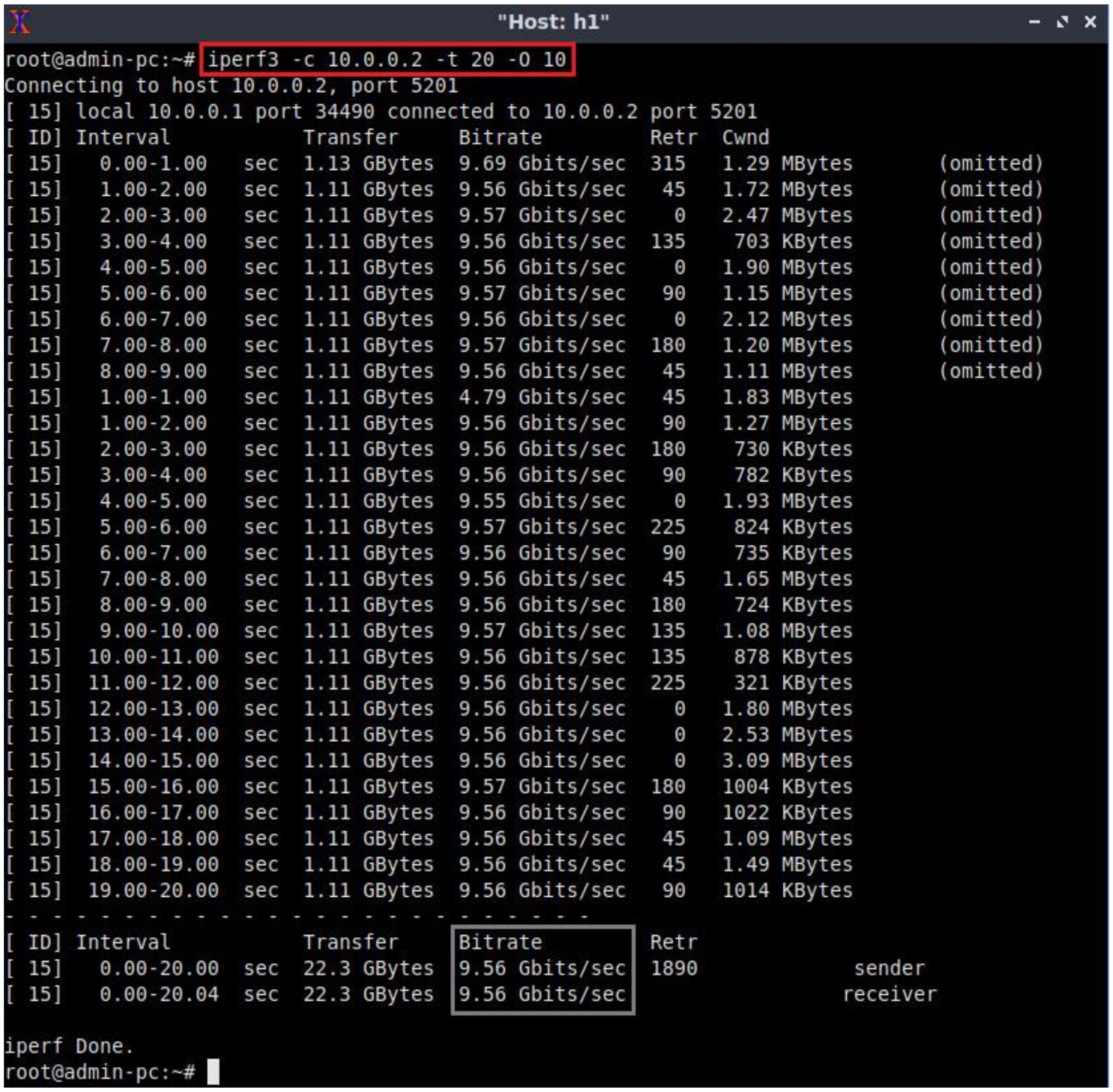


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 27. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in client mode on host h1 's terminal. The `-O` option is used to specify the number of seconds to omit in the resulting report. Note that this option is a capitalized 'O', not a zero.

```
iperf3 -c 10.0.0.2 -t 20 -O 10
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 20 -O 10
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 34490 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec  1.13 GBytes  9.69 Gbits/sec  315   1.29 MBytes       (omitted)
[ 15]   1.00-2.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.72 MBytes       (omitted)
[ 15]   2.00-3.00   sec  1.11 GBytes  9.57 Gbits/sec    0   2.47 MBytes       (omitted)
[ 15]   3.00-4.00   sec  1.11 GBytes  9.56 Gbits/sec  135    703 KBytes       (omitted)
[ 15]   4.00-5.00   sec  1.11 GBytes  9.56 Gbits/sec    0   1.90 MBytes       (omitted)
[ 15]   5.00-6.00   sec  1.11 GBytes  9.57 Gbits/sec   90   1.15 MBytes       (omitted)
[ 15]   6.00-7.00   sec  1.11 GBytes  9.56 Gbits/sec    0   2.12 MBytes       (omitted)
[ 15]   7.00-8.00   sec  1.11 GBytes  9.57 Gbits/sec  180   1.20 MBytes       (omitted)
[ 15]   8.00-9.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.11 MBytes       (omitted)
[ 15]   1.00-1.00   sec  1.11 GBytes  4.79 Gbits/sec   45   1.83 MBytes
[ 15]   1.00-2.00   sec  1.11 GBytes  9.56 Gbits/sec   90   1.27 MBytes
[ 15]   2.00-3.00   sec  1.11 GBytes  9.56 Gbits/sec  180    730 KBytes
[ 15]   3.00-4.00   sec  1.11 GBytes  9.56 Gbits/sec   90    782 KBytes
[ 15]   4.00-5.00   sec  1.11 GBytes  9.55 Gbits/sec    0   1.93 MBytes
[ 15]   5.00-6.00   sec  1.11 GBytes  9.57 Gbits/sec  225    824 KBytes
[ 15]   6.00-7.00   sec  1.11 GBytes  9.56 Gbits/sec   90    735 KBytes
[ 15]   7.00-8.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.65 MBytes
[ 15]   8.00-9.00   sec  1.11 GBytes  9.56 Gbits/sec  180    724 KBytes
[ 15]   9.00-10.00  sec  1.11 GBytes  9.57 Gbits/sec  135   1.08 MBytes
[ 15]  10.00-11.00  sec  1.11 GBytes  9.56 Gbits/sec  135    878 KBytes
[ 15]  11.00-12.00  sec  1.11 GBytes  9.56 Gbits/sec  225    321 KBytes
[ 15]  12.00-13.00  sec  1.11 GBytes  9.56 Gbits/sec    0   1.80 MBytes
[ 15]  13.00-14.00  sec  1.11 GBytes  9.56 Gbits/sec    0   2.53 MBytes
[ 15]  14.00-15.00  sec  1.11 GBytes  9.56 Gbits/sec    0   3.09 MBytes
[ 15]  15.00-16.00  sec  1.11 GBytes  9.57 Gbits/sec  180   1004 KBytes
[ 15]  16.00-17.00  sec  1.11 GBytes  9.56 Gbits/sec   90   1022 KBytes
[ 15]  17.00-18.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.09 MBytes
[ 15]  18.00-19.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.49 MBytes
[ 15]  19.00-20.00  sec  1.11 GBytes  9.56 Gbits/sec   90   1014 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-20.00  sec  22.3 GBytes  9.56 Gbits/sec  1890             sender
[ 15]   0.00-20.04  sec  22.3 GBytes  9.56 Gbits/sec                  receiver

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

Figure 28. Running iPerf3 client on host h1.

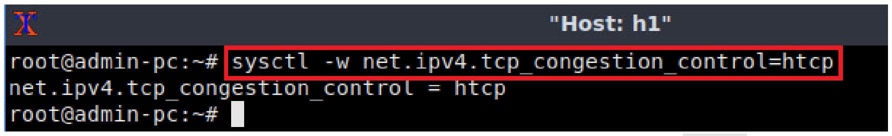
The figure above shows the iPerf3 test output report. The average achieved throughput is 9.56 Gbps (sender) and 9.56 Gbps (receiver), and the number of retransmissions is 1890 (due to the injected packet loss-- 0.01%).

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.

5.1.2 Hamilton TCP (HTCP)

Step 1. In host h1’s terminal, change the TCP congestion control algorithm to HTCP by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=htcp
```

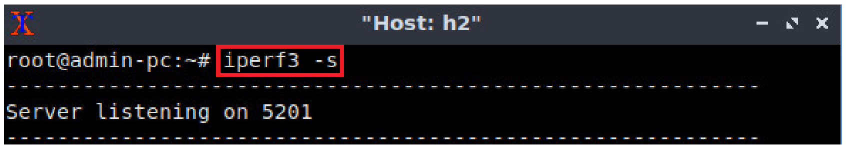


```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=htcp
net.ipv4.tcp_congestion_control = htcp
root@admin-pc:~# 
```

Figure 29. Changing TCP congestion control algorithm to `htcp` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

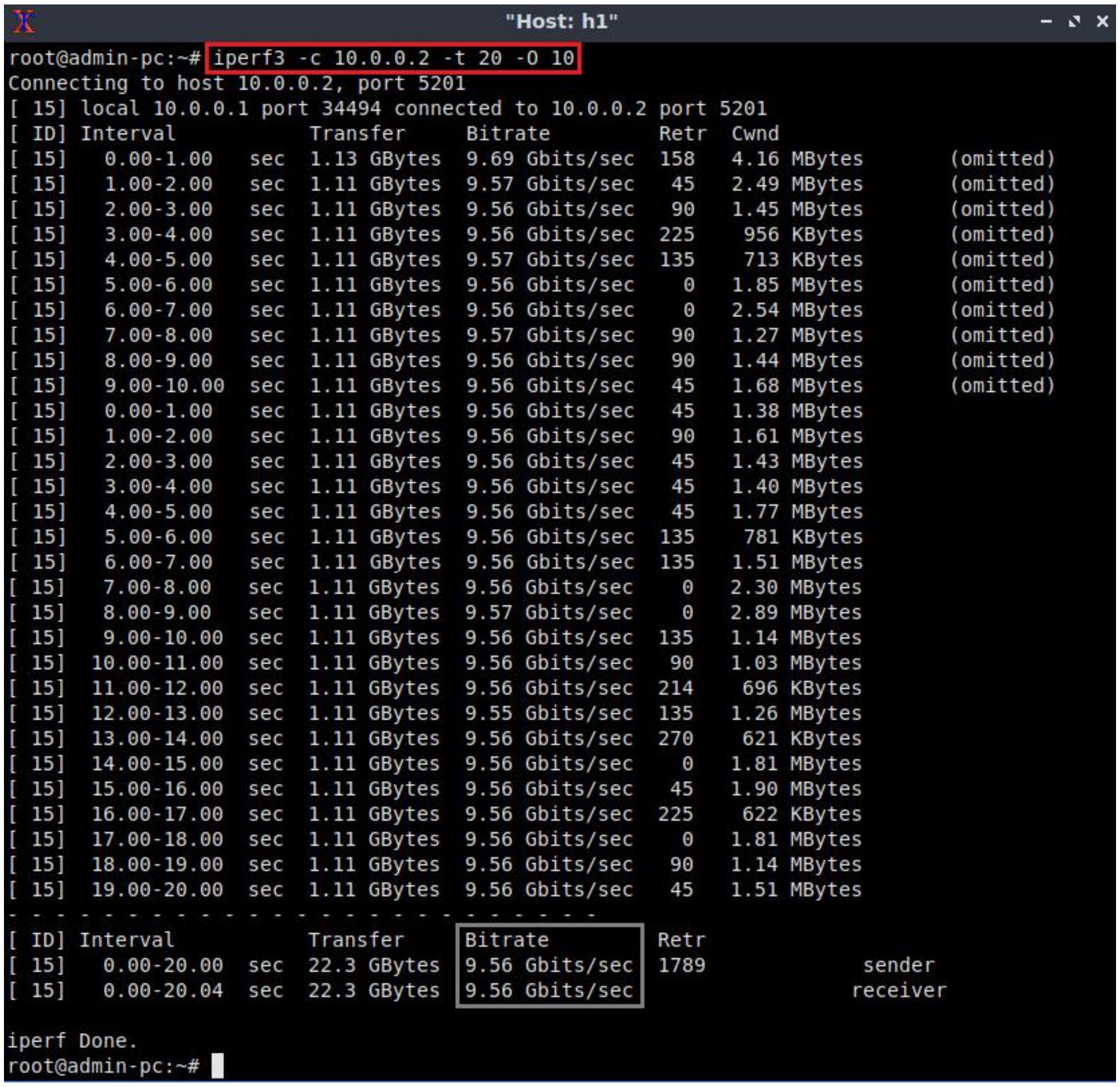


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 30. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in client mode on host h1's terminal:

```
iperf3 -c 10.0.0.2 -t 20 -O 10
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 20 -O 10
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 34494 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec  1.13 GBytes  9.69 Gbits/sec  158   4.16 MBytes       (omitted)
[ 15]   1.00-2.00   sec  1.11 GBytes  9.57 Gbits/sec   45   2.49 MBytes       (omitted)
[ 15]   2.00-3.00   sec  1.11 GBytes  9.56 Gbits/sec   90   1.45 MBytes       (omitted)
[ 15]   3.00-4.00   sec  1.11 GBytes  9.56 Gbits/sec  225    956 KBytes       (omitted)
[ 15]   4.00-5.00   sec  1.11 GBytes  9.57 Gbits/sec  135    713 KBytes       (omitted)
[ 15]   5.00-6.00   sec  1.11 GBytes  9.56 Gbits/sec    0   1.85 MBytes       (omitted)
[ 15]   6.00-7.00   sec  1.11 GBytes  9.56 Gbits/sec    0   2.54 MBytes       (omitted)
[ 15]   7.00-8.00   sec  1.11 GBytes  9.57 Gbits/sec   90   1.27 MBytes       (omitted)
[ 15]   8.00-9.00   sec  1.11 GBytes  9.56 Gbits/sec   90   1.44 MBytes       (omitted)
[ 15]   9.00-10.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.68 MBytes       (omitted)
[ 15]   0.00-1.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.38 MBytes
[ 15]   1.00-2.00   sec  1.11 GBytes  9.56 Gbits/sec   90   1.61 MBytes
[ 15]   2.00-3.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.43 MBytes
[ 15]   3.00-4.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.40 MBytes
[ 15]   4.00-5.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.77 MBytes
[ 15]   5.00-6.00   sec  1.11 GBytes  9.56 Gbits/sec  135    781 KBytes
[ 15]   6.00-7.00   sec  1.11 GBytes  9.56 Gbits/sec  135   1.51 MBytes
[ 15]   7.00-8.00   sec  1.11 GBytes  9.56 Gbits/sec    0   2.30 MBytes
[ 15]   8.00-9.00   sec  1.11 GBytes  9.57 Gbits/sec    0   2.89 MBytes
[ 15]   9.00-10.00  sec  1.11 GBytes  9.56 Gbits/sec  135   1.14 MBytes
[ 15]  10.00-11.00  sec  1.11 GBytes  9.56 Gbits/sec   90   1.03 MBytes
[ 15]  11.00-12.00  sec  1.11 GBytes  9.56 Gbits/sec  214    696 KBytes
[ 15]  12.00-13.00  sec  1.11 GBytes  9.55 Gbits/sec  135   1.26 MBytes
[ 15]  13.00-14.00  sec  1.11 GBytes  9.56 Gbits/sec  270    621 KBytes
[ 15]  14.00-15.00  sec  1.11 GBytes  9.56 Gbits/sec    0   1.81 MBytes
[ 15]  15.00-16.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.90 MBytes
[ 15]  16.00-17.00  sec  1.11 GBytes  9.56 Gbits/sec  225    622 KBytes
[ 15]  17.00-18.00  sec  1.11 GBytes  9.56 Gbits/sec    0   1.81 MBytes
[ 15]  18.00-19.00  sec  1.11 GBytes  9.56 Gbits/sec   90   1.14 MBytes
[ 15]  19.00-20.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.51 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-20.00  sec  22.3 GBytes  9.56 Gbits/sec  1789             sender
[ 15]   0.00-20.04  sec  22.3 GBytes  9.56 Gbits/sec                  receiver

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

Figure 31. Running iPerf3 client on host h1.

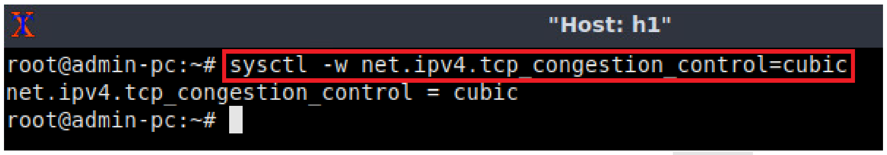
The figure above shows the iPerf3 test output report. The average achieved throughput is 9.56 Gbps (sender) and 9.56 Gbps (receiver), and the number of retransmissions is 1789 (due to the injected packet loss-- 0.01%).

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.

5.1.3 TCP Cubic

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Cubic by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=cubic
```

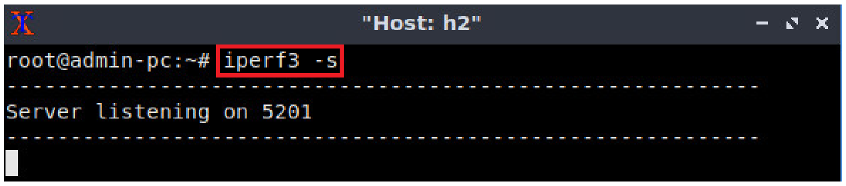


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=cubic
net.ipv4.tcp_congestion_control = cubic
root@admin-pc:~# 
```

Figure 32. Changing TCP congestion control algorithm to `cubic` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

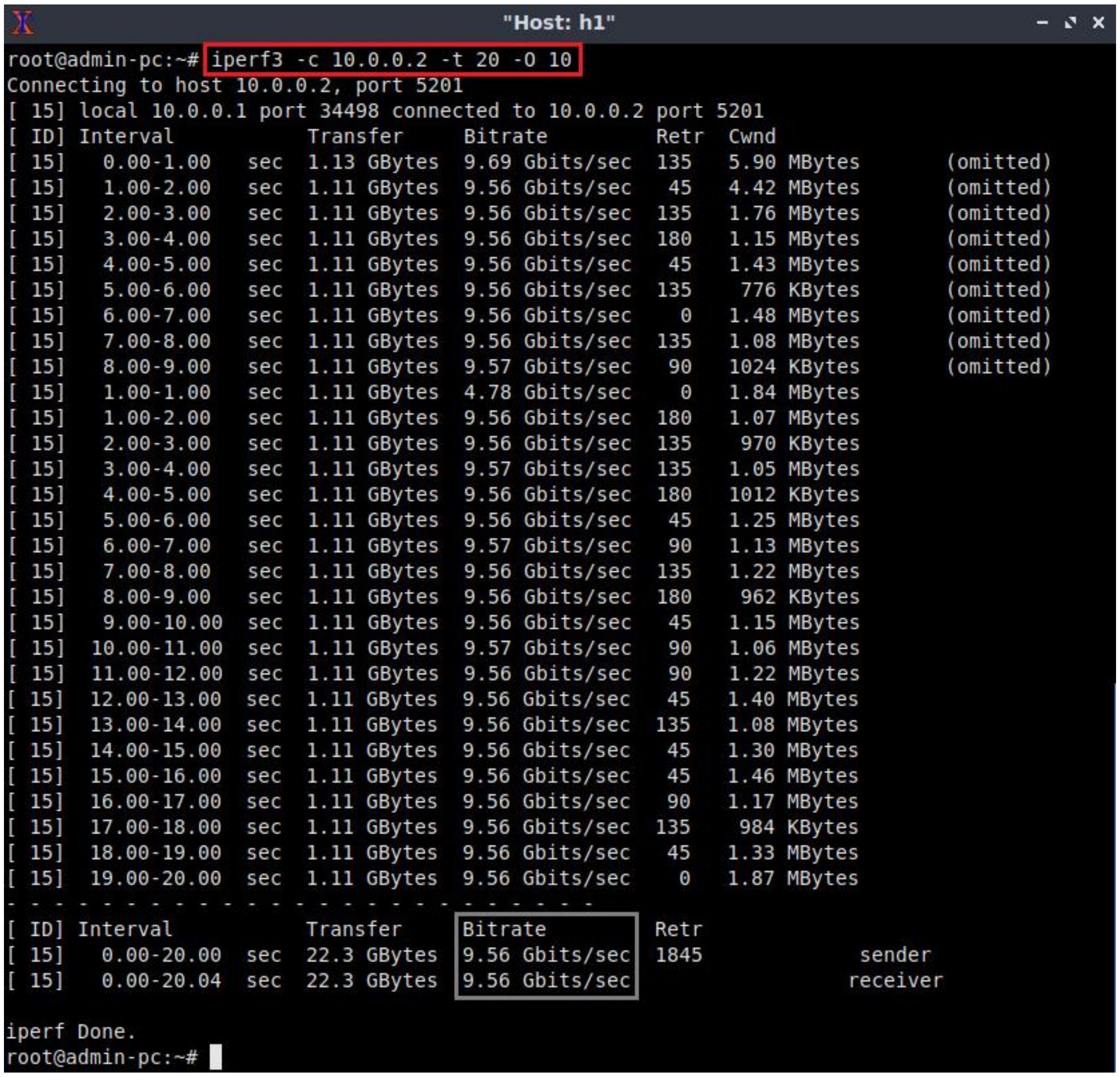


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 33. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in client mode on host h1's terminal:

```
iperf3 -c 10.0.0.2 -t 20 -O 10
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 20 -O 10
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 34498 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec  1.13 GBytes  9.69 Gbits/sec  135   5.90 MBytes       (omitted)
[ 15]   1.00-2.00   sec  1.11 GBytes  9.56 Gbits/sec   45   4.42 MBytes       (omitted)
[ 15]   2.00-3.00   sec  1.11 GBytes  9.56 Gbits/sec  135   1.76 MBytes       (omitted)
[ 15]   3.00-4.00   sec  1.11 GBytes  9.56 Gbits/sec  180   1.15 MBytes       (omitted)
[ 15]   4.00-5.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.43 MBytes       (omitted)
[ 15]   5.00-6.00   sec  1.11 GBytes  9.56 Gbits/sec  135    776 KBytes       (omitted)
[ 15]   6.00-7.00   sec  1.11 GBytes  9.56 Gbits/sec    0   1.48 MBytes       (omitted)
[ 15]   7.00-8.00   sec  1.11 GBytes  9.56 Gbits/sec  135   1.08 MBytes       (omitted)
[ 15]   8.00-9.00   sec  1.11 GBytes  9.57 Gbits/sec   90   1024 KBytes       (omitted)
[ 15]   1.00-1.00   sec  1.11 GBytes  4.78 Gbits/sec    0   1.84 MBytes
[ 15]   1.00-2.00   sec  1.11 GBytes  9.56 Gbits/sec  180   1.07 MBytes
[ 15]   2.00-3.00   sec  1.11 GBytes  9.56 Gbits/sec  135    970 KBytes
[ 15]   3.00-4.00   sec  1.11 GBytes  9.57 Gbits/sec  135   1.05 MBytes
[ 15]   4.00-5.00   sec  1.11 GBytes  9.56 Gbits/sec  180   1012 KBytes
[ 15]   5.00-6.00   sec  1.11 GBytes  9.56 Gbits/sec   45   1.25 MBytes
[ 15]   6.00-7.00   sec  1.11 GBytes  9.57 Gbits/sec   90   1.13 MBytes
[ 15]   7.00-8.00   sec  1.11 GBytes  9.56 Gbits/sec  135   1.22 MBytes
[ 15]   8.00-9.00   sec  1.11 GBytes  9.56 Gbits/sec  180    962 KBytes
[ 15]   9.00-10.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.15 MBytes
[ 15]  10.00-11.00  sec  1.11 GBytes  9.57 Gbits/sec   90   1.06 MBytes
[ 15]  11.00-12.00  sec  1.11 GBytes  9.56 Gbits/sec   90   1.22 MBytes
[ 15]  12.00-13.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.40 MBytes
[ 15]  13.00-14.00  sec  1.11 GBytes  9.56 Gbits/sec  135   1.08 MBytes
[ 15]  14.00-15.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.30 MBytes
[ 15]  15.00-16.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.46 MBytes
[ 15]  16.00-17.00  sec  1.11 GBytes  9.56 Gbits/sec   90   1.17 MBytes
[ 15]  17.00-18.00  sec  1.11 GBytes  9.56 Gbits/sec  135    984 KBytes
[ 15]  18.00-19.00  sec  1.11 GBytes  9.56 Gbits/sec   45   1.33 MBytes
[ 15]  19.00-20.00  sec  1.11 GBytes  9.56 Gbits/sec    0   1.87 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-20.00  sec  22.3 GBytes  9.56 Gbits/sec  1845             sender
[ 15]   0.00-20.04  sec  22.3 GBytes  9.56 Gbits/sec                  receiver

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

Figure 34. Running iPerf3 client on host h1.

The figure above shows the iPerf3 test output report. The average achieved throughput is 9.56 Gbps (sender) and 9.56 Gbps (receiver), and the number of retransmissions is 1845 (due to the injected packet loss-- 0.01%).

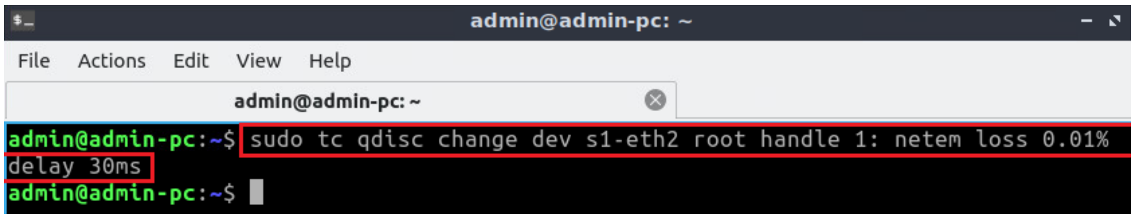
Step 4. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

5.2 Throughput test with 30ms delay

In this test, we measure the throughput between host h1 and host h2 while introducing 30ms delay on the switch S1's *s1-eth2* interface. Apply the following steps:

Step 1. On the client's terminal, run the following command to modify the previous rule to include 30ms delay. When prompted for a password, type `password` and hit enter.

```
sudo tc qdisc change dev s1-eth2 root handle 1: netem loss 0.01% delay 30ms
```



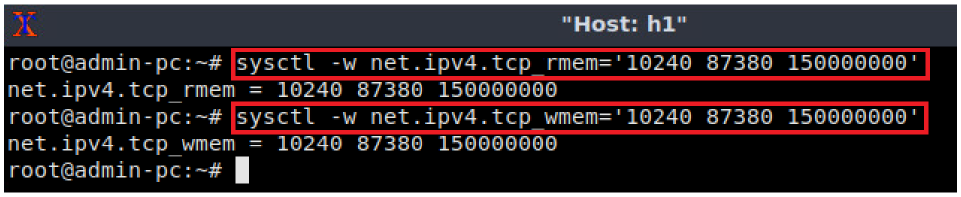
```
admin@admin-pc:~$ sudo tc qdisc change dev s1-eth2 root handle 1: netem loss 0.01%
delay 30ms
admin@admin-pc:~$
```

Figure 35. Injecting 30ms delay on switch S1's *s1-eth2* interface.

Step 2. In host h1's terminal, modify the TCP buffer size by typing the following commands: *sysctl -w net.ipv4.tcp_rmem='10,240 87,380 150,000,000'* and *sysctl -w net.ipv4.tcp_wmem='10,240 87,380 150,000,000'*. This TCP buffer is explained later in future labs.

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'
```

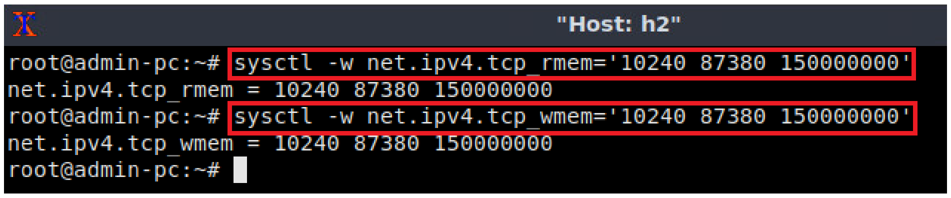
```
sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'
```



```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'
net.ipv4.tcp_rmem = 10240 87380 150000000
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'
net.ipv4.tcp_wmem = 10240 87380 150000000
root@admin-pc:~#
```

Figure 36. Modifying the TCP buffer size on host h1.

Step 3. In host h2's terminal, also modify the TCP buffer size by typing the following commands: *sysctl -w net.ipv4.tcp_rmem='10,240 87,380 150,000,000'* and *sysctl -w net.ipv4.tcp_wmem='10,240 87,380 150,000,000'*.



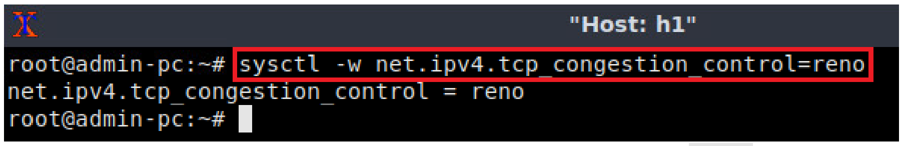
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'
net.ipv4.tcp_rmem = 10240 87380 150000000
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'
net.ipv4.tcp_wmem = 10240 87380 150000000
root@admin-pc:~#
```

Figure 37. Modifying the TCP buffer size on host h2.

5.2.1 TCP Reno

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Reno by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=reno
```

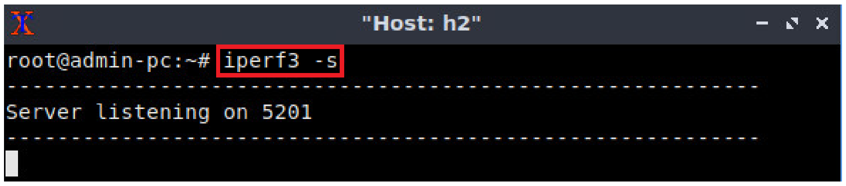


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
root@admin-pc:~#
```

Figure 38. Changing TCP congestion control algorithm to `reno` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

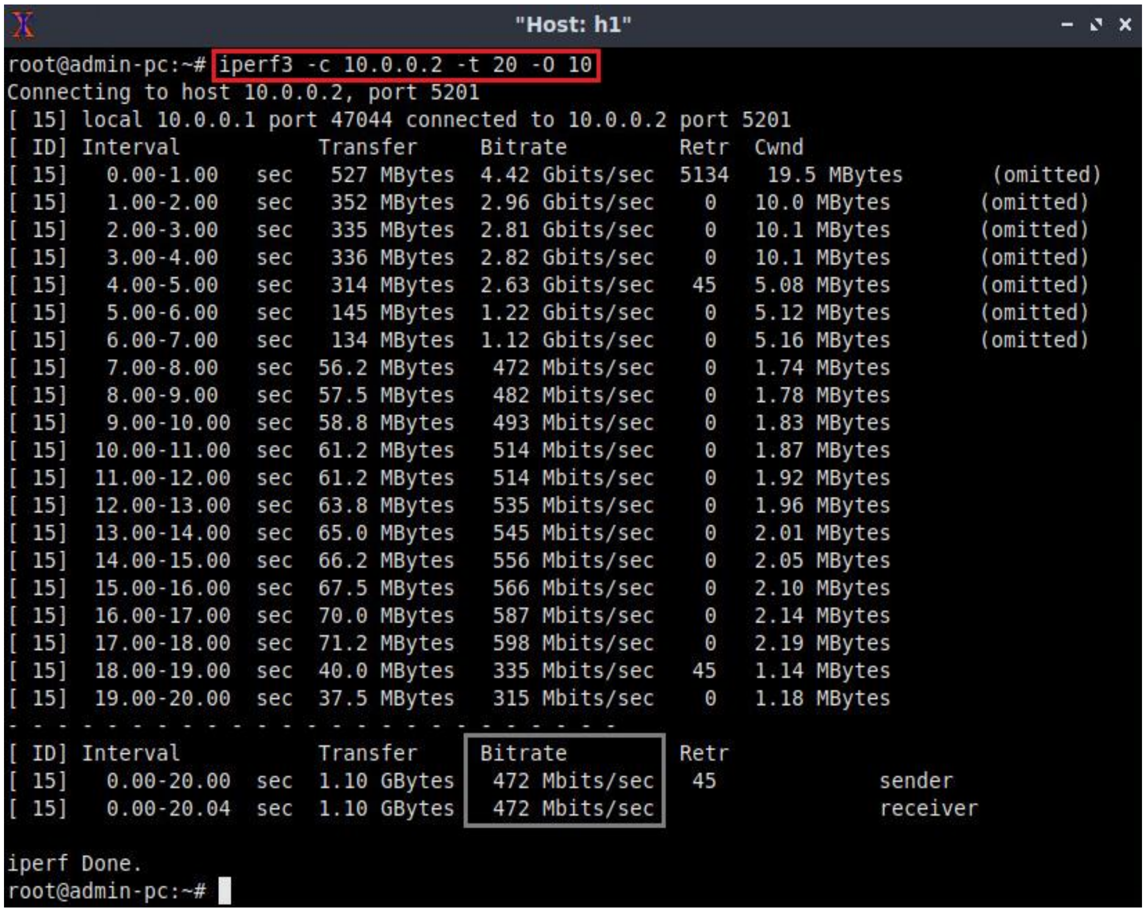


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 39. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in client mode on host h1's terminal. The `-O` option is used to specify the number of seconds to omit in the resulting report.

```
iperf3 -c 10.0.0.2 -t 20 -O 10
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 20 -O 10
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 47044 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   527 MBytes  4.42 Gbits/sec  5134   19.5 MBytes       (omitted)
[ 15]   1.00-2.00   sec   352 MBytes  2.96 Gbits/sec    0   10.0 MBytes       (omitted)
[ 15]   2.00-3.00   sec   335 MBytes  2.81 Gbits/sec    0   10.1 MBytes       (omitted)
[ 15]   3.00-4.00   sec   336 MBytes  2.82 Gbits/sec    0   10.1 MBytes       (omitted)
[ 15]   4.00-5.00   sec   314 MBytes  2.63 Gbits/sec   45   5.08 MBytes       (omitted)
[ 15]   5.00-6.00   sec   145 MBytes  1.22 Gbits/sec    0   5.12 MBytes       (omitted)
[ 15]   6.00-7.00   sec   134 MBytes  1.12 Gbits/sec    0   5.16 MBytes       (omitted)
[ 15]   7.00-8.00   sec  56.2 MBytes   472 Mbits/sec    0   1.74 MBytes
[ 15]   8.00-9.00   sec  57.5 MBytes   482 Mbits/sec    0   1.78 MBytes
[ 15]   9.00-10.00  sec  58.8 MBytes   493 Mbits/sec    0   1.83 MBytes
[ 15]  10.00-11.00  sec  61.2 MBytes   514 Mbits/sec    0   1.87 MBytes
[ 15]  11.00-12.00  sec  61.2 MBytes   514 Mbits/sec    0   1.92 MBytes
[ 15]  12.00-13.00  sec  63.8 MBytes   535 Mbits/sec    0   1.96 MBytes
[ 15]  13.00-14.00  sec  65.0 MBytes   545 Mbits/sec    0   2.01 MBytes
[ 15]  14.00-15.00  sec  66.2 MBytes   556 Mbits/sec    0   2.05 MBytes
[ 15]  15.00-16.00  sec  67.5 MBytes   566 Mbits/sec    0   2.10 MBytes
[ 15]  16.00-17.00  sec  70.0 MBytes   587 Mbits/sec    0   2.14 MBytes
[ 15]  17.00-18.00  sec  71.2 MBytes   598 Mbits/sec    0   2.19 MBytes
[ 15]  18.00-19.00  sec  40.0 MBytes   335 Mbits/sec   45   1.14 MBytes
[ 15]  19.00-20.00  sec  37.5 MBytes   315 Mbits/sec    0   1.18 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-20.00  sec  1.10 GBytes   472 Mbits/sec   45             sender
[ 15]   0.00-20.04  sec  1.10 GBytes   472 Mbits/sec                  receiver

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

Figure 40. Running iPerf3 client on host h1.

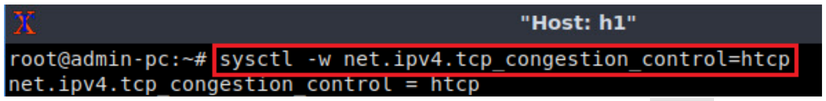
The figure above shows the iPerf3 test output report. The average achieved throughput is 472 Mbps (sender) and 472 Mbps (receiver), and the number of retransmissions is 45.

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.

5.2.2 Hamilton TCP (HTCP)

Step 1. In host h1's terminal, change the TCP congestion control algorithm to HTCP by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=htcp
```

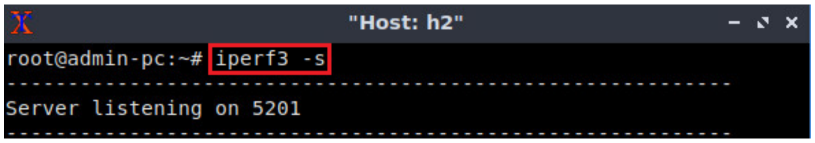


```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=htcp
net.ipv4.tcp_congestion_control = htcp
```

Figure 41. Changing TCP congestion control algorithm to `htcp` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

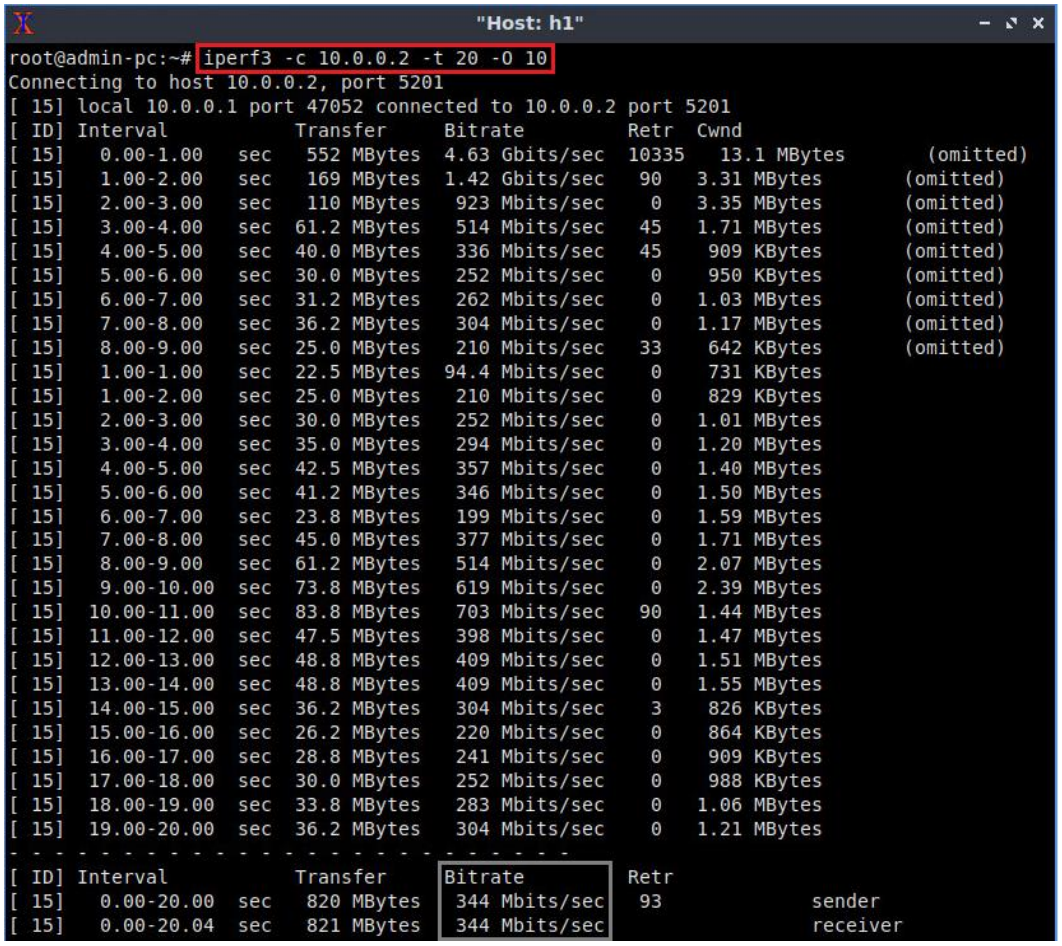


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 42. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in client mode on host h1's terminal:

```
iperf3 -c 10.0.0.2 -t 20 -O 10
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 20 -O 10
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 47052 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   552 MBytes  4.63 Gbits/sec  10335   13.1 MBytes       (omitted)
[ 15]   1.00-2.00   sec   169 MBytes  1.42 Gbits/sec   90   3.31 MBytes       (omitted)
[ 15]   2.00-3.00   sec   110 MBytes   923 Mbits/sec    0   3.35 MBytes       (omitted)
[ 15]   3.00-4.00   sec  61.2 MBytes   514 Mbits/sec   45   1.71 MBytes       (omitted)
[ 15]   4.00-5.00   sec  40.0 MBytes   336 Mbits/sec   45    909 KBytes       (omitted)
[ 15]   5.00-6.00   sec  30.0 MBytes   252 Mbits/sec    0    950 KBytes       (omitted)
[ 15]   6.00-7.00   sec  31.2 MBytes   262 Mbits/sec    0   1.03 MBytes       (omitted)
[ 15]   7.00-8.00   sec  36.2 MBytes   304 Mbits/sec    0   1.17 MBytes       (omitted)
[ 15]   8.00-9.00   sec  25.0 MBytes   210 Mbits/sec   33    642 KBytes       (omitted)
[ 15]   1.00-1.00   sec  22.5 MBytes  94.4 Mbits/sec    0    731 KBytes
[ 15]   1.00-2.00   sec  25.0 MBytes   210 Mbits/sec    0    829 KBytes
[ 15]   2.00-3.00   sec  30.0 MBytes   252 Mbits/sec    0   1.01 MBytes
[ 15]   3.00-4.00   sec  35.0 MBytes   294 Mbits/sec    0   1.20 MBytes
[ 15]   4.00-5.00   sec  42.5 MBytes   357 Mbits/sec    0   1.40 MBytes
[ 15]   5.00-6.00   sec  41.2 MBytes   346 Mbits/sec    0   1.50 MBytes
[ 15]   6.00-7.00   sec  23.8 MBytes   199 Mbits/sec    0   1.59 MBytes
[ 15]   7.00-8.00   sec  45.0 MBytes   377 Mbits/sec    0   1.71 MBytes
[ 15]   8.00-9.00   sec  61.2 MBytes   514 Mbits/sec    0   2.07 MBytes
[ 15]   9.00-10.00  sec  73.8 MBytes   619 Mbits/sec    0   2.39 MBytes
[ 15]  10.00-11.00  sec  83.8 MBytes   703 Mbits/sec   90   1.44 MBytes
[ 15]  11.00-12.00  sec  47.5 MBytes   398 Mbits/sec    0   1.47 MBytes
[ 15]  12.00-13.00  sec  48.8 MBytes   409 Mbits/sec    0   1.51 MBytes
[ 15]  13.00-14.00  sec  48.8 MBytes   409 Mbits/sec    0   1.55 MBytes
[ 15]  14.00-15.00  sec  36.2 MBytes   304 Mbits/sec    3    826 KBytes
[ 15]  15.00-16.00  sec  26.2 MBytes   220 Mbits/sec    0    864 KBytes
[ 15]  16.00-17.00  sec  28.8 MBytes   241 Mbits/sec    0    909 KBytes
[ 15]  17.00-18.00  sec  30.0 MBytes   252 Mbits/sec    0    988 KBytes
[ 15]  18.00-19.00  sec  33.8 MBytes   283 Mbits/sec    0   1.06 MBytes
[ 15]  19.00-20.00  sec  36.2 MBytes   304 Mbits/sec    0   1.21 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-20.00  sec   820 MBytes   344 Mbits/sec   93             sender
[ 15]   0.00-20.04  sec   821 MBytes   344 Mbits/sec                  receiver
```

Figure 43. Running iPerf3 client on host h1.

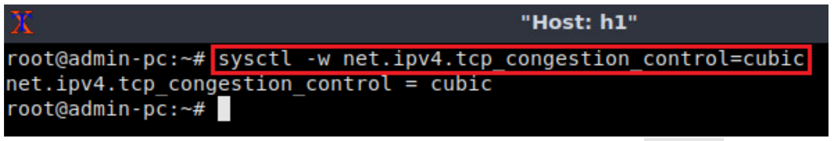
The figure above shows the iPerf3 test output report. The average achieved throughput is 344 Mbps (sender) and 344 Mbps (receiver), and the number of retransmissions is 93.

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.

5.2.3 TCP Cubic

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Cubic by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=cubic
```

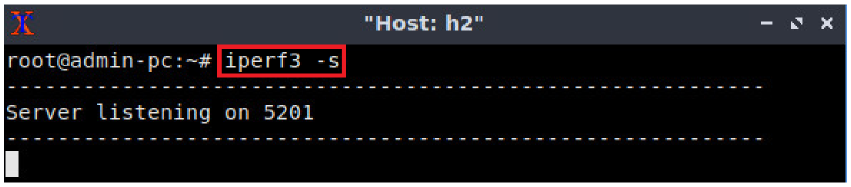


```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=cubic
net.ipv4.tcp_congestion_control = cubic
root@admin-pc:~#
```

Figure 44. Changing TCP congestion control algorithm to `cubic` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

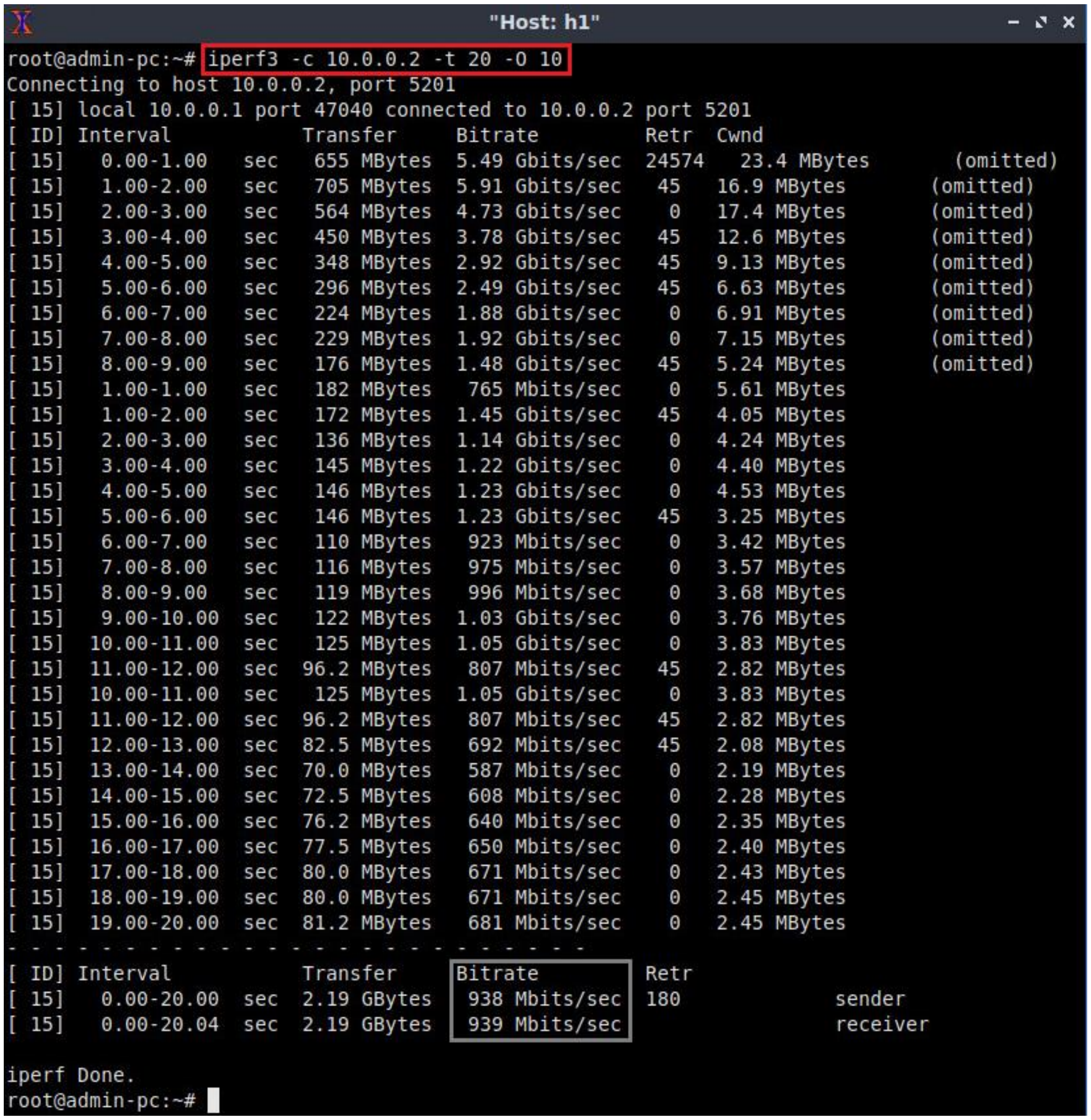


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 45. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in client mode on host h1's terminal:

```
iperf3 -c 10.0.0.2 -t 20 -O 10
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 20 -O 10
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 47040 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   655 MBytes  5.49 Gbits/sec  24574   23.4 MBytes       (omitted)
[ 15]   1.00-2.00   sec   705 MBytes  5.91 Gbits/sec   45   16.9 MBytes       (omitted)
[ 15]   2.00-3.00   sec   564 MBytes  4.73 Gbits/sec    0   17.4 MBytes       (omitted)
[ 15]   3.00-4.00   sec   450 MBytes  3.78 Gbits/sec   45   12.6 MBytes       (omitted)
[ 15]   4.00-5.00   sec   348 MBytes  2.92 Gbits/sec   45   9.13 MBytes       (omitted)
[ 15]   5.00-6.00   sec   296 MBytes  2.49 Gbits/sec   45   6.63 MBytes       (omitted)
[ 15]   6.00-7.00   sec   224 MBytes  1.88 Gbits/sec    0   6.91 MBytes       (omitted)
[ 15]   7.00-8.00   sec   229 MBytes  1.92 Gbits/sec    0   7.15 MBytes       (omitted)
[ 15]   8.00-9.00   sec   176 MBytes  1.48 Gbits/sec   45   5.24 MBytes       (omitted)
[ 15]   1.00-1.00   sec   182 MBytes   765 Mbits/sec    0   5.61 MBytes
[ 15]   1.00-2.00   sec   172 MBytes  1.45 Gbits/sec   45   4.05 MBytes
[ 15]   2.00-3.00   sec   136 MBytes  1.14 Gbits/sec    0   4.24 MBytes
[ 15]   3.00-4.00   sec   145 MBytes  1.22 Gbits/sec    0   4.40 MBytes
[ 15]   4.00-5.00   sec   146 MBytes  1.23 Gbits/sec    0   4.53 MBytes
[ 15]   5.00-6.00   sec   146 MBytes  1.23 Gbits/sec   45   3.25 MBytes
[ 15]   6.00-7.00   sec   110 MBytes   923 Mbits/sec    0   3.42 MBytes
[ 15]   7.00-8.00   sec   116 MBytes   975 Mbits/sec    0   3.57 MBytes
[ 15]   8.00-9.00   sec   119 MBytes   996 Mbits/sec    0   3.68 MBytes
[ 15]   9.00-10.00  sec   122 MBytes  1.03 Gbits/sec    0   3.76 MBytes
[ 15]  10.00-11.00  sec   125 MBytes  1.05 Gbits/sec    0   3.83 MBytes
[ 15]  11.00-12.00  sec  96.2 MBytes   807 Mbits/sec   45   2.82 MBytes
[ 15]  10.00-11.00  sec   125 MBytes  1.05 Gbits/sec    0   3.83 MBytes
[ 15]  11.00-12.00  sec  96.2 MBytes   807 Mbits/sec   45   2.82 MBytes
[ 15]  12.00-13.00  sec  82.5 MBytes   692 Mbits/sec   45   2.08 MBytes
[ 15]  13.00-14.00  sec  70.0 MBytes   587 Mbits/sec    0   2.19 MBytes
[ 15]  14.00-15.00  sec  72.5 MBytes   608 Mbits/sec    0   2.28 MBytes
[ 15]  15.00-16.00  sec  76.2 MBytes   640 Mbits/sec    0   2.35 MBytes
[ 15]  16.00-17.00  sec  77.5 MBytes   650 Mbits/sec    0   2.40 MBytes
[ 15]  17.00-18.00  sec  80.0 MBytes   671 Mbits/sec    0   2.43 MBytes
[ 15]  18.00-19.00  sec  80.0 MBytes   671 Mbits/sec    0   2.45 MBytes
[ 15]  19.00-20.00  sec  81.2 MBytes   681 Mbits/sec    0   2.45 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-20.00  sec  2.19 GBytes   938 Mbits/sec  180             sender
[ 15]   0.00-20.04  sec  2.19 GBytes   939 Mbits/sec                  receiver

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

Figure 46. Running iPerf3 client on host h1.

The figure above shows the iPerf3 test output report. The average achieved throughput is 938 Mbps (sender) and 939 Mbps (receiver), and the number of retransmissions is 180.

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.

This concludes Lab 6. Stop the emulation and then exit out of MiniEdit and Linux terminal.

References

1. K. Fall, S. Floyd, "Simulation-based comparisons of tahoe, reno, and sack TCP," Computer Communication Review, vol. 26, issue 3, Jul. 1996.

2. D. Leith, R. Shorten, Y. Lee, "H-TCP: a framework for congestion control in high-speed and long-distance networks," Hamilton Institute Technical Report, Aug. 2005. [Online]. Available: http://www.hamilton.ie/net/htcp2005.pdf.
3. E. Dart, L. Rotman, B. Tierney, M. Hester, J. Zurawski, "The science DMZ: a network design pattern for data-intensive science," in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Nov. 2013.
4. S. Ha, I., Rhee, L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," ACM SIGOPS operating systems review, vol. 42, issue 5, pp. 64-74, Jul. 2008.
5. D. Leith, R. Shorten, Y. Lee, "H-TCP: a framework for congestion control in high-speed and long-distance networks," Hamilton Institute Technical Report, Aug. 2005. [Online]. Available: http://www.hamilton.ie/net/htcp2005.pdf.
6. 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.
7. System information variables – sysctl (7). [Online]. Available: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt.

UNIVERSITY OF
SOUTH CAROLINA

NETWORK TOOLS AND PROTOCOLS

Lab 7: Understanding Rate-based TCP Congestion Control (BBR)

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 TCP 3
1.1 Traditional TCP congestion control review 3
1.2 Traditional congestion control limitations 4
1.3 TCP BBR 5
2 Lab topology 8
2.1 Starting host h1 and host h2 9
2.2 Emulating 1 Gbps high-latency WAN with packet loss 10
2.3 Testing connection 11
3 iPerf3 throughput test 12
3.1 Throughput test without delay 12
3.1.1 TCP Reno 12
3.1.2 TCP BBR 13
3.2 Throughput test with 30ms delay 15
3.2.1 TCP Reno 16
3.2.2 TCP BBR 19
References 22

Overview

This lab describes a new type of TCP congestion control algorithm called Bottleneck Bandwidth and Round-Trip Time (BBR). The lab conducts experimental results using TCP BBR and contrasts these results with those obtained using traditional congestion control algorithms such as a Reno and HTCP.

Objectives

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

1. Describe the basic operation of TCP BBR.
2. Describe differences between rate-based congestion control and window-based loss-based congestion control.
3. Modify the TCP congestion control algorithm in Linux using sysctl tool.
4. Compare the throughput performance of TCP Reno and BBR in high-throughput high-latency networks.

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 TCP.
2. Section 2: Lab Topology.
3. Section 3: iPerf3 Throughput Test.

1 Introduction to TCP

1.1 Traditional TCP congestion control review

TCP congestion control was introduced in the late 1980s. For many years, the main algorithm of congestion control was TCP Reno[1]. Subsequently, multiple algorithms were proposed based on Reno's enhancements. The goal of congestion control is to determine how much capacity is available in the network, so that a source knows how many packets it can safely have in transit (inflight). Once a source has these packets in transit, it uses the arrival of an acknowledgement (ACK) as a signal that one of its packets has left the network and that it is therefore safe to insert a new packet into the network without adding to the level of congestion. By using ACKs to pace the transmission of packets, TCP is said to be self-clocking[2].

A major task of the congestion control algorithm is to determine the available capacity. In steady state, TCP Reno maintains an estimate of the Round-Trip Time (RTT) -the time to send a packet and receive the corresponding ACK-. If the ACK stream shows that no packets are lost in transit, Reno increases the sending rate by one additional segment each RTT interval. This period is known as the additive increase. Note that "segment" here refers to the protocol data unit (PDU) at the transport layer, and that sometimes the terms packet and segment are interchangeably used. Eventually, the increasing flow rate saturates the bottleneck link at a router, which drops a packet. The TCP receiver signals the missing packet by sending an ACK in response to an out-of-order received segment, as illustrated in Figure 1(a). Once the TCP sender receives three duplicate ACKs for the same out-of-order segment, it interprets this event as packet loss due to congestion and reduces the sending rate by half. This reduction is known as multiplicative decrease. Once the loss is repaired, Reno resumes the additive increase phase. This iteration of additive increase multiplicative decrease (AIMD) periods is shown in Figure 1(b).

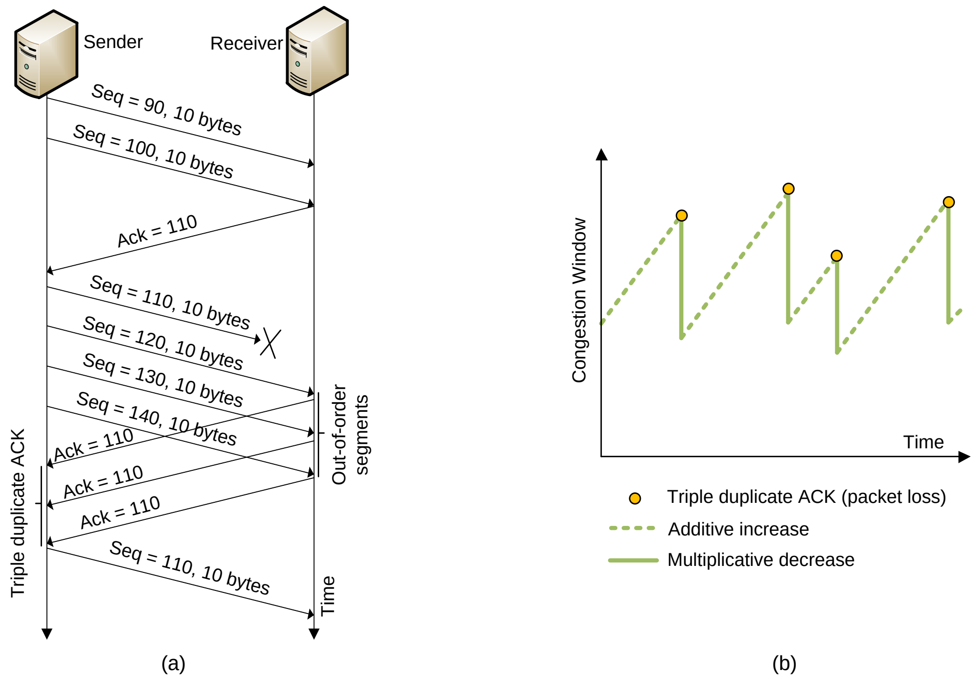


Figure 1. (a) TCP operation. (b) Evolution of TCP's congestion window.

1.2 Traditional congestion control limitations

While Reno has proven to perform adequately in the past, when the bulk of the TCP connections carried trivial applications such as web browsing and email, it faces severe limitations in high-throughput connections that are needed for grid computing and big science data transfers. Reno's average TCP throughput can be approximated by the following equation[2]:

$$\text{TCP Throughput} \approx \frac{\text{MSS}}{\text{RTT}\sqrt{L}} \ \text{[bytes / second]}$$

The equation above indicates that the throughput of a TCP connection in steady state is directly proportional to the maximum segment size (MSS) and inversely proportional to the product of Round-Trip Time (RTT) and the square root of the packet loss rate (L). Essentially, the equation above indicates that the TCP throughput is very sensitive to packet loss. In such environments Reno cannot achieve high throughput, especially in high-latency scenarios. Figure 2 validates the above equation. It shows the throughput as a function of RTT, for two devices connected by a 10 Gbps path. The performance of two TCP AIMD-based implementations are provided: Reno[1] (blue) and Hamilton TCP[3], better known as HTCP (red). The theoretical performance (using the above equation) with packet losses (green) and the measured throughput without packet losses (purple) are also shown. Figure 2 is reproduced from[4].

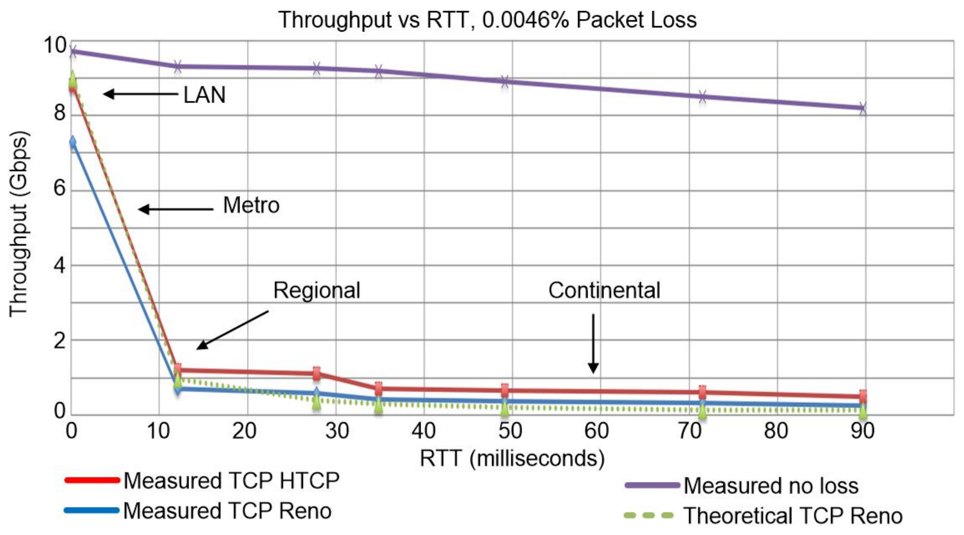


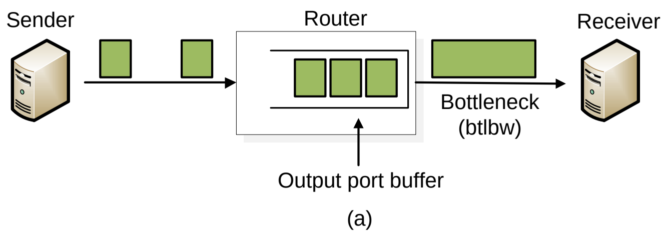
Figure 2. Throughput vs Round-Trip Time (RTT) for two devices connected via a 10 Gbps path. The performance of two TCP implementations are provided: Reno[1] (blue) and HTCP (red). The theoretical performance with packet losses (green) and the measured throughput without packet losses (purple) are also shown.

1.3 TCP BBR

The main issue surrounding traditional congestion control algorithms in high-speed high-latency networks is that the sender cannot recover from the packet loss and multiplicative decrease, even when the packet losses are sporadic. When the RTT is large, increasing the congestion window (and thus the sending rate) by only 1 MSS every RTT is too slow.

BBR[5] is a new congestion control algorithm that does not adhere to the AIMD rule and the above equation. BBR is a rate-based algorithm, meaning that at any given time it sends data at a rate that is independent of current packet losses. Note that this feature is a drastic departure from traditional congestion control algorithms, which operate by reducing the sending rate by half each time a packet loss is detected.

The behavior of BBR can be described using Figure 3, which shows a TCP's viewpoint of an end-to-end connection. At any time, the connection has exactly one slowest link, or bottleneck bandwidth (*btlbw*), that determines the location where queues are formed. When router buffers are large, traditional congestion control keeps them full (i.e., they keep increasing the rate during the additive increase phase). When buffers are small, traditional congestion control misinterprets a loss as a signal of congestion, leading to low throughput. The output port queue increases when the input link arrival rate exceeds *btlbw*. The throughput of loss-based congestion control is less than *btlbw* because of the frequent packet losses.



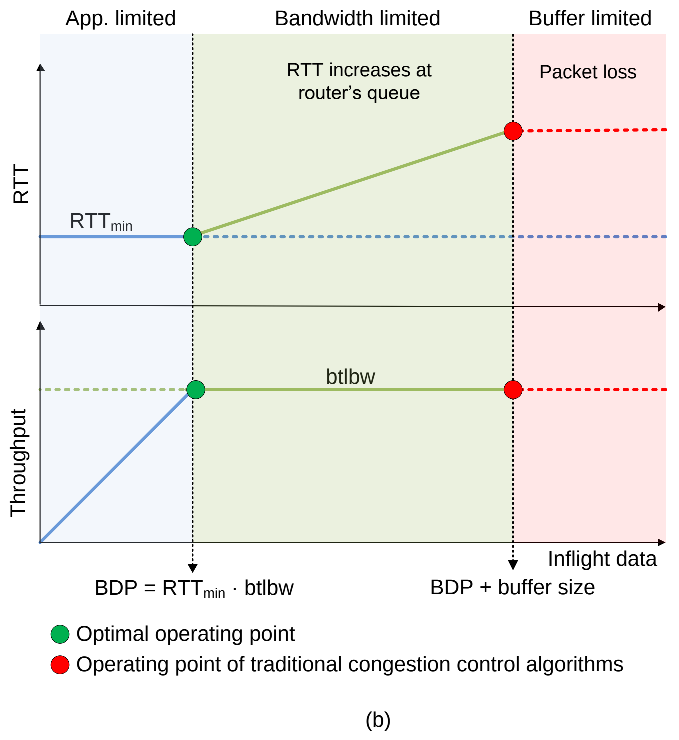


Figure 3. TCP viewpoint of a connection and relation between throughput and RTT. (a) Simplified TCP interpretation of the connection. (b) Throughput and RTT, as a function of in-flight data.

Figure 3(b) illustrates the RTT and throughput with the amount of data inflight[5]. RTT$_{min}$ is the propagation time with no queueing component (the network is not congested). In the application limited region, the delivery rate/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 the bandwidth multiplied by the RTT. This number is also called bandwidth-delay product (BDP) and quantifies the number of bits that can be inflight if the sender continuously sends segments. In the bandwidth limited region, the queue size at the router of Figure 3(a) starts increasing, resulting in an increase of the RTT. The throughput remains constant, as the bottleneck link is fully utilized. Finally, when no buffer is available at the router to store arriving packets (the number of inflight bits is equal to BDP plus the buffer size of the router), these are dropped.

It is important to understand that packets to be sent are paced at the estimated bottleneck rate, which is intended to avoid network queuing that would otherwise be encountered when the network performs rate adaptation at the bottleneck point. The intended operational model here is that the sender is passing packets into the network at a rate that is not anticipated to encounter queuing at any point within the entire path. This is a significant contrast to protocols such as Reno, which tends to send packet bursts at the epoch of the RTT and relies on the network’s queues to perform rate adaptation in the interior of the network if the burst sending rate is higher than the bottleneck capacity.

BBR also periodically probes for additional bandwidth. It spends one RTT interval deliberately sending at a rate that is higher than the current estimate bottleneck bandwidth. Specifically, it sends data at 125% the bottleneck bandwidth. If the available bottleneck bandwidth has not changed, then the increased sending rate will cause a queue to form at the bottleneck. This will cause the ACK signaling to reveal an increased RTT, but the bottleneck bandwidth estimate will be unaltered. If this is the case, then the sender will subsequently send at a compensating reduced sending rate for an RTT interval. The reduced rate is set to 75% the bottleneck bandwidth, allowing the bottleneck queue to drain. On the other hand, if the available bottleneck bandwidth estimate has increased because of this probe, then the sender will operate according to this new bottleneck bandwidth estimate. The entire cycle duration lasts eight RTTs and is repeated indefinitely in steady state.

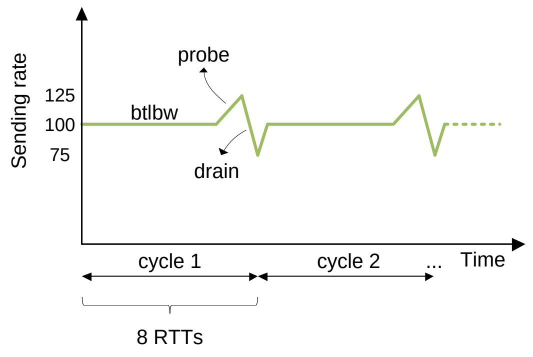


Figure 4. The rate used by the sender is the estimate bottleneck bandwidth (*btlbw*). During the probe period (1 RTT duration), the sender probes for additional bandwidth, sending at a rate of 125% of the bottleneck bandwidth. During the subsequent period, drain (1 RTT duration), the sender reduces the rate to 75% of the bottleneck bandwidth, thus allowing any bottleneck queue to drain.

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.

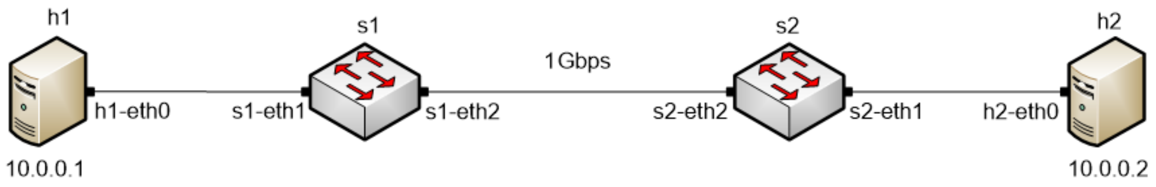


Figure 5. Lab topology.

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`.

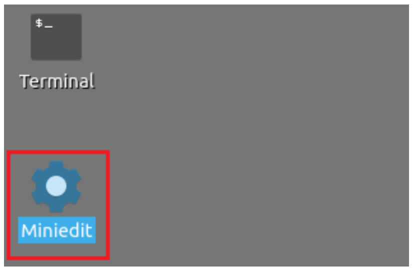


Figure 6. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 7.mn* topology file and click on *Open*.

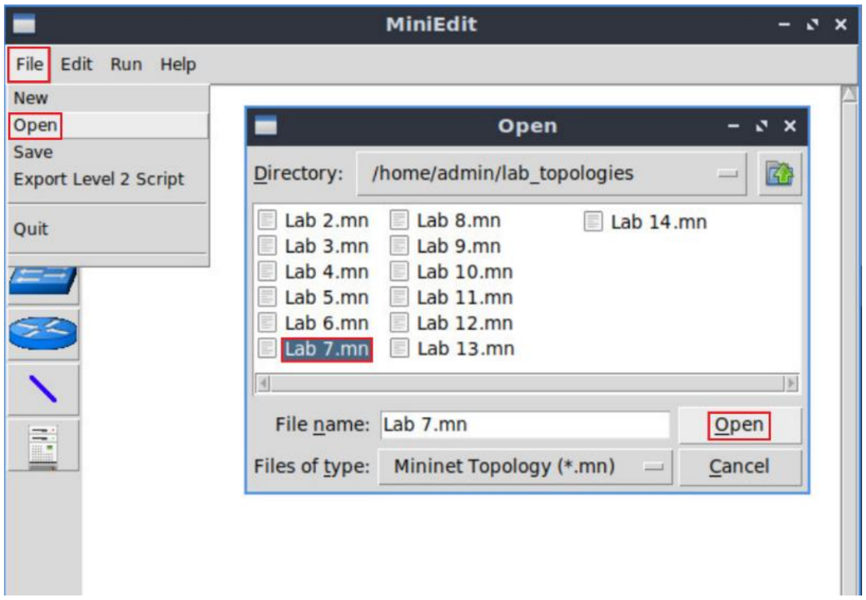


Figure 7. MiniEdit's *Open* dialog.

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.

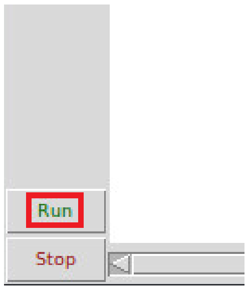


Figure 8. Running the emulation.

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 the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

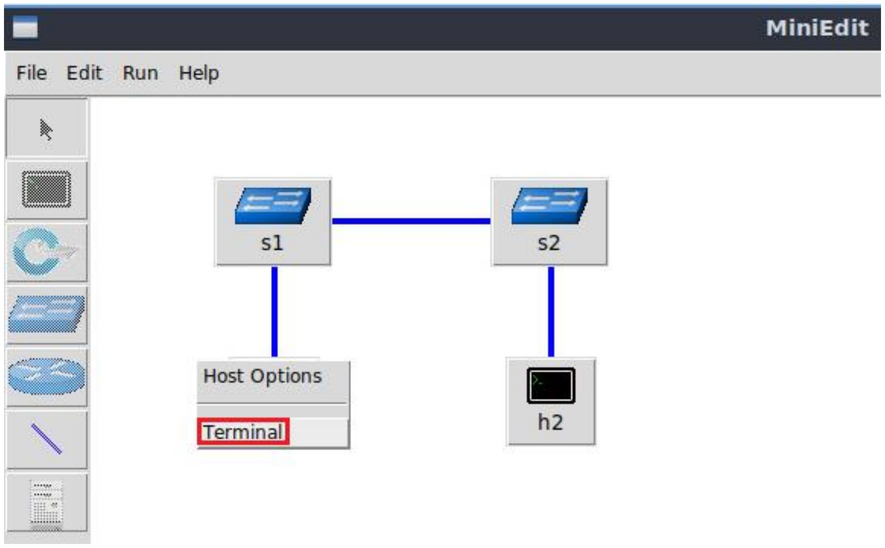


Figure 9. Opening a terminal on host h1.

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.

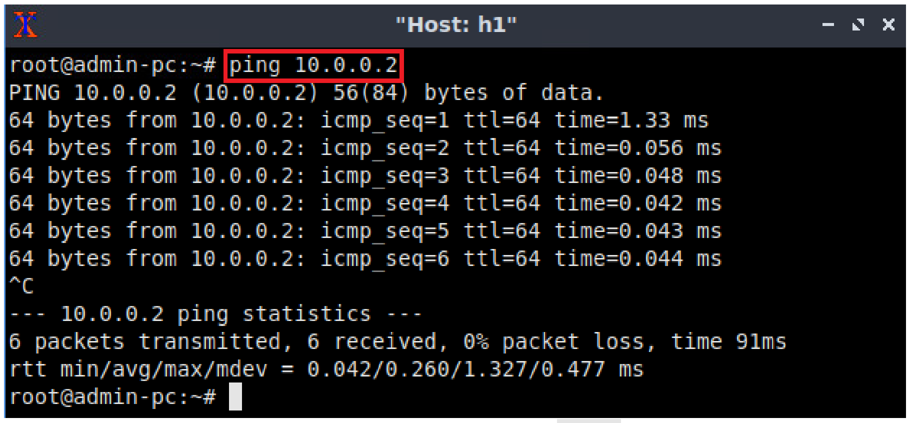

```
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.33 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.042 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 10. Connectivity test using `ping` command.

Figure 10 indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

2.2 Emulating 1 Gbps high-latency WAN with packet loss

This section emulates a high-latency WAN, which is used to validate the results observed in Figure 3. We will first set the bandwidth between host h1 and host h2 to 1 Gbps. Then we will emulate packet losses between switch S1 and switch S2, and measure the throughput.

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

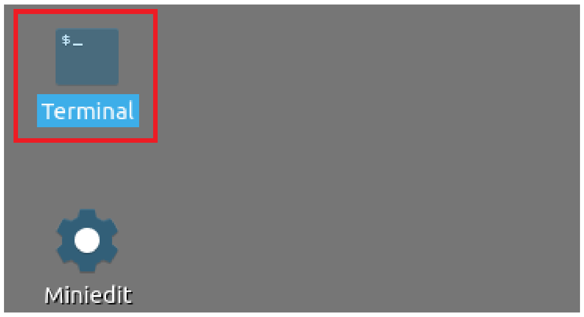
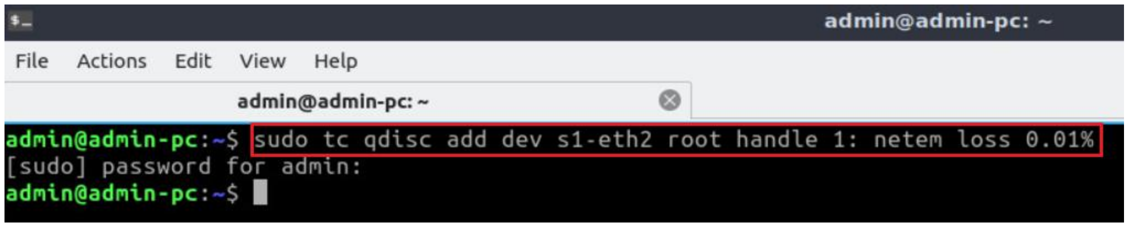


Figure 11. Shortcut to open a Linux terminal.

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. In the terminal, type the below command. When prompted for a password, type `password` and hit enter. This command basically introduces a 0.01% packet loss rate on switch S1's *s1-eth2* interface.

```
sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 0.01%
```



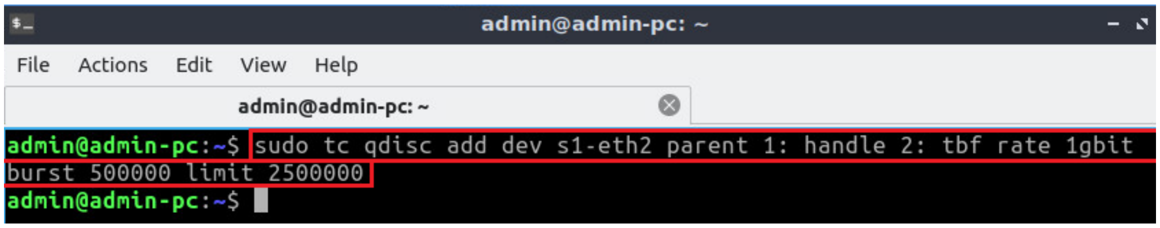
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 0.01%
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 12. Adding 0.01% packet loss rate to switch S1's s1-eth2 interface.

Step 3. Modify the bandwidth of the link connecting the switch S1 and switch S2: on the same terminal, type the command below. This command sets the bandwidth to 1 Gbps on switch S1's *s1-eth2* interface. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 1gbit burst 500000
limit 2500000
```



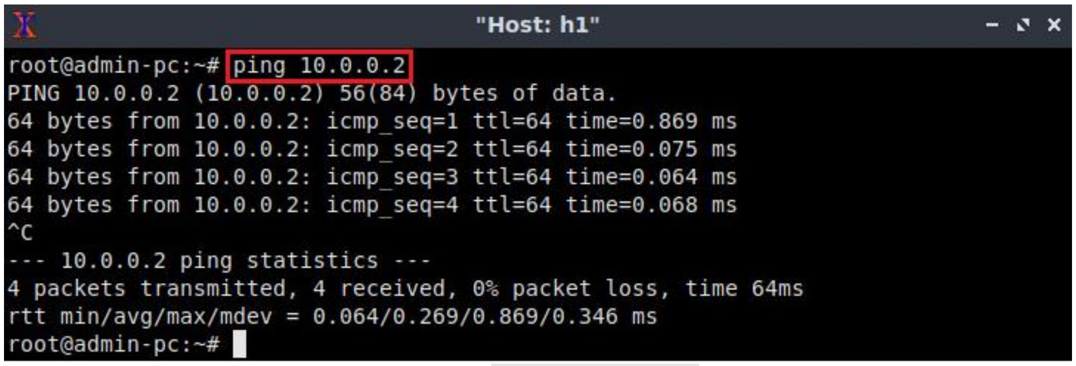
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 1gbit
burst 500000 limit 2500000
admin@admin-pc:~$
```

Figure 13. Limiting the bandwidth to 1 Gbps on switch S1's s1-eth2 interface.

2.3 Testing connection

To test connectivity, you can use the command `ping`.

Step 1. On the terminal of host h1, type `ping 10.0.0.2`. 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.



```
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=0.869 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.068 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 64ms
rtt min/avg/max/mdev = 0.064/0.269/0.869/0.346 ms
root@admin-pc:~#
```

Figure 14. 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 0.064, 0.269, 0.869, and 0.346 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 2. On the terminal of host h2, type `ping 10.0.0.1`. The ping output in this test should be relatively close to the results of the test initiated by host h1 in Step 1. To stop the test, press `Ctrl+c`.

3 iPerf3 throughput test

In this section, the throughput between host h1 and host h2 is measured using two congestion control algorithms: Reno and BBR. Moreover, the test is repeated using various injected delays to observe the throughput variations depending on each congestion control algorithm and the selected RTT.

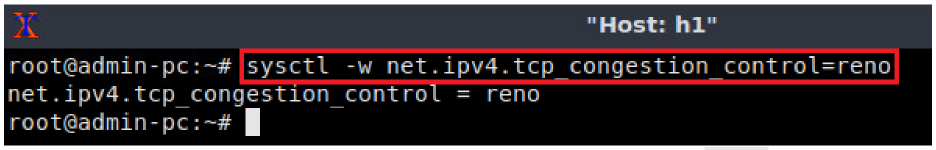
3.1 Throughput test without delay

In this test, we measure the throughput between host h1 and host h2 without introducing delay on the switch S1's *s1-eth2* interface.

3.1.1 TCP Reno

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Reno by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=reno
```

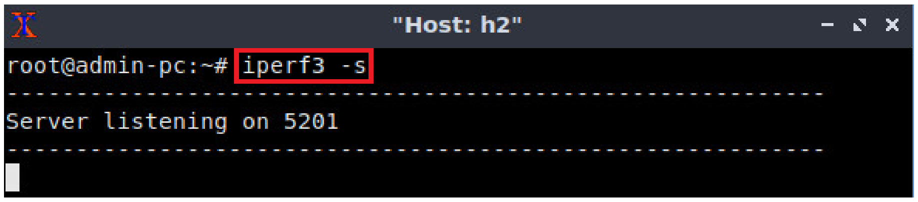


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
root@admin-pc:~#
```

Figure 15. Changing TCP congestion control algorithm to `reno` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

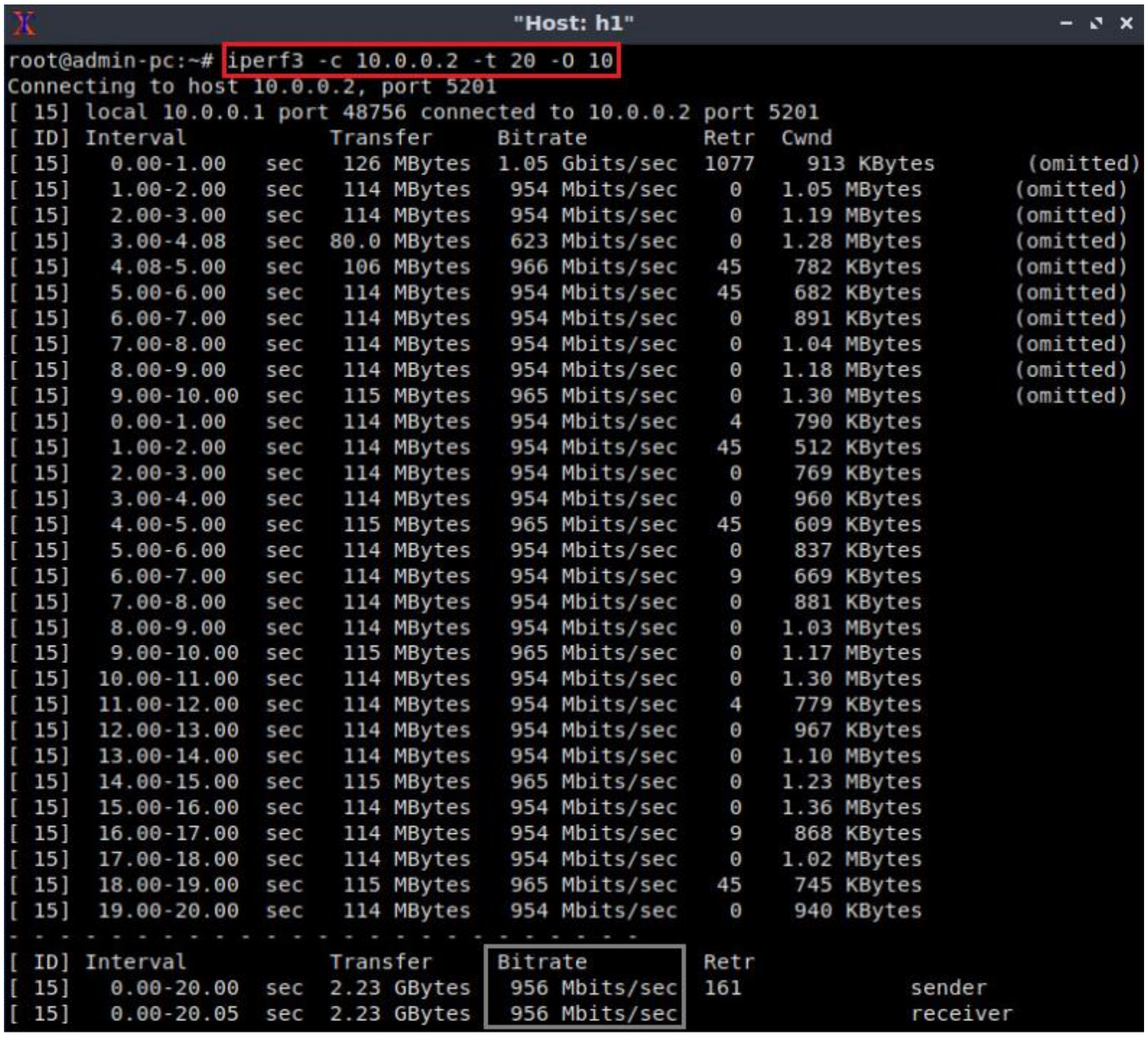


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 16. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in client mode on host h1's terminal. The `-O` option is used to specify the number of seconds to omit in the resulting report.

```
iperf3 -c 10.0.0.2 -t 20 -O 10
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 20 -O 10
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 48756 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   126 MBytes  1.05 Gbits/sec  1077    913 KBytes       (omitted)
[ 15]   1.00-2.00   sec   114 MBytes   954 Mbits/sec    0   1.05 MBytes       (omitted)
[ 15]   2.00-3.00   sec   114 MBytes   954 Mbits/sec    0   1.19 MBytes       (omitted)
[ 15]   3.00-4.08   sec  80.0 MBytes   623 Mbits/sec    0   1.28 MBytes       (omitted)
[ 15]   4.08-5.00   sec   106 MBytes   966 Mbits/sec   45    782 KBytes       (omitted)
[ 15]   5.00-6.00   sec   114 MBytes   954 Mbits/sec   45    682 KBytes       (omitted)
[ 15]   6.00-7.00   sec   114 MBytes   954 Mbits/sec    0    891 KBytes       (omitted)
[ 15]   7.00-8.00   sec   114 MBytes   954 Mbits/sec    0   1.04 MBytes       (omitted)
[ 15]   8.00-9.00   sec   114 MBytes   954 Mbits/sec    0   1.18 MBytes       (omitted)
[ 15]   9.00-10.00  sec   115 MBytes   965 Mbits/sec    0   1.30 MBytes       (omitted)
[ 15]   0.00-1.00   sec   114 MBytes   954 Mbits/sec    4    790 KBytes
[ 15]   1.00-2.00   sec   114 MBytes   954 Mbits/sec   45    512 KBytes
[ 15]   2.00-3.00   sec   114 MBytes   954 Mbits/sec    0    769 KBytes
[ 15]   3.00-4.00   sec   114 MBytes   954 Mbits/sec    0    960 KBytes
[ 15]   4.00-5.00   sec   115 MBytes   965 Mbits/sec   45    609 KBytes
[ 15]   5.00-6.00   sec   114 MBytes   954 Mbits/sec    0    837 KBytes
[ 15]   6.00-7.00   sec   114 MBytes   954 Mbits/sec    9    669 KBytes
[ 15]   7.00-8.00   sec   114 MBytes   954 Mbits/sec    0    881 KBytes
[ 15]   8.00-9.00   sec   114 MBytes   954 Mbits/sec    0   1.03 MBytes
[ 15]   9.00-10.00  sec   115 MBytes   965 Mbits/sec    0   1.17 MBytes
[ 15]  10.00-11.00  sec   114 MBytes   954 Mbits/sec    0   1.30 MBytes
[ 15]  11.00-12.00  sec   114 MBytes   954 Mbits/sec    4    779 KBytes
[ 15]  12.00-13.00  sec   114 MBytes   954 Mbits/sec    0    967 KBytes
[ 15]  13.00-14.00  sec   114 MBytes   954 Mbits/sec    0   1.10 MBytes
[ 15]  14.00-15.00  sec   115 MBytes   965 Mbits/sec    0   1.23 MBytes
[ 15]  15.00-16.00  sec   114 MBytes   954 Mbits/sec    0   1.36 MBytes
[ 15]  16.00-17.00  sec   114 MBytes   954 Mbits/sec    9    868 KBytes
[ 15]  17.00-18.00  sec   114 MBytes   954 Mbits/sec    0   1.02 MBytes
[ 15]  18.00-19.00  sec   115 MBytes   965 Mbits/sec   45    745 KBytes
[ 15]  19.00-20.00  sec   114 MBytes   954 Mbits/sec    0    940 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-20.00  sec  2.23 GBytes   956 Mbits/sec  161             sender
[ 15]   0.00-20.05  sec  2.23 GBytes   956 Mbits/sec                  receiver
```

Figure 17. Running iPerf3 client on host h1.

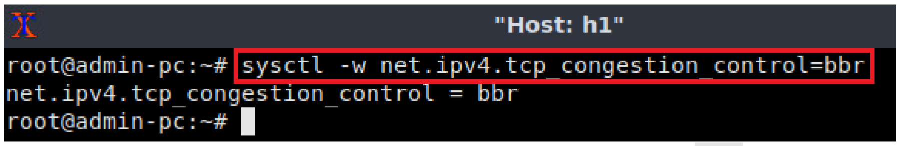
The figure above shows the iPerf3 test output report. The average achieved throughputs are 956 Mbps (sender) and 956 Mbps (receiver), and the number of retransmissions is 161 (due to the injected packet loss - 0.01%).

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.

3.1.2 TCP BBR

Step 1. In host h1's terminal, change the TCP congestion control algorithm to BBR by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=bbr
```

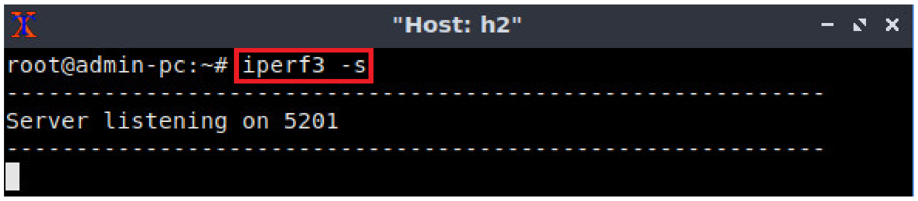


```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=bbr
net.ipv4.tcp_congestion_control = bbr
root@admin-pc:~#
```

Figure 18. Changing TCP congestion control algorithm to `bbr` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

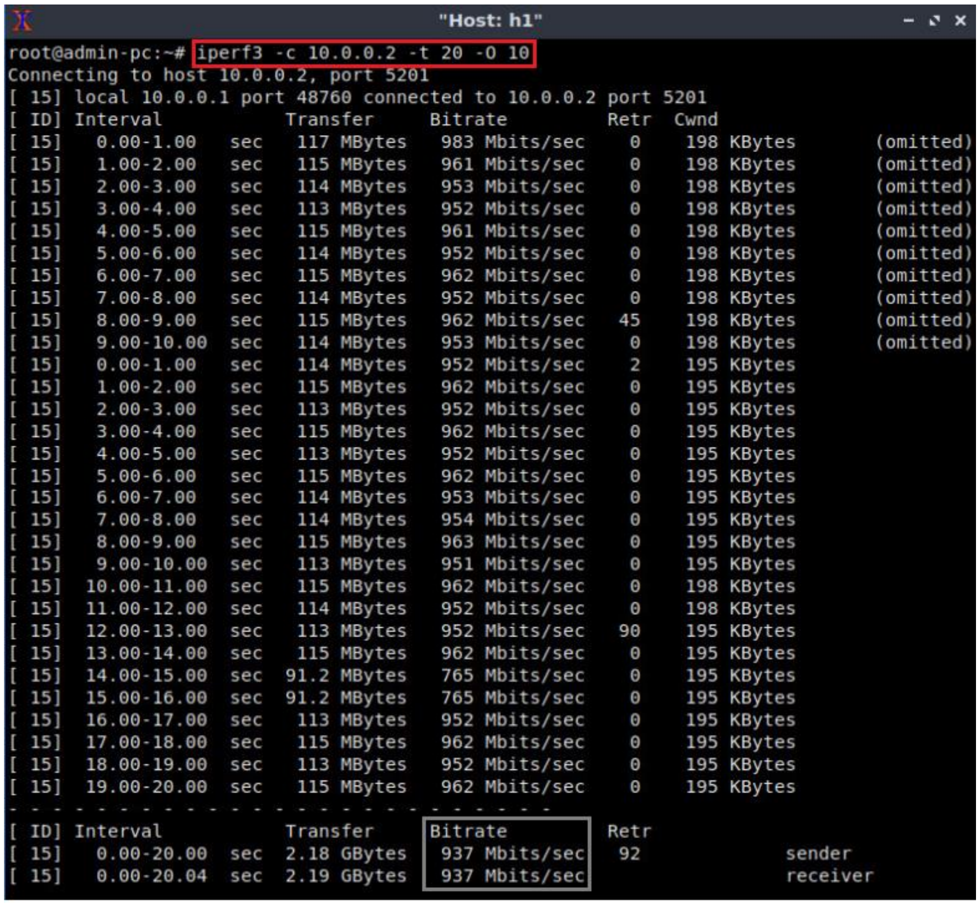


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 19. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in client mode on host h1's terminal:

```
iperf3 -c 10.0.0.2 -t 20 -O 10
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 20 -O 10
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 48760 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   117 MBytes   983 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   1.00-2.00   sec   115 MBytes   961 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   2.00-3.00   sec   114 MBytes   953 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   3.00-4.00   sec   113 MBytes   952 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   4.00-5.00   sec   115 MBytes   961 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   5.00-6.00   sec   114 MBytes   952 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   6.00-7.00   sec   115 MBytes   962 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   7.00-8.00   sec   114 MBytes   952 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   8.00-9.00   sec   115 MBytes   962 Mbits/sec   45    198 KBytes       (omitted)
[ 15]   9.00-10.00  sec   114 MBytes   953 Mbits/sec    0    198 KBytes       (omitted)
[ 15]   0.00-1.00   sec   114 MBytes   952 Mbits/sec    2    195 KBytes
[ 15]   1.00-2.00   sec   115 MBytes   962 Mbits/sec    0    195 KBytes
[ 15]   2.00-3.00   sec   113 MBytes   952 Mbits/sec    0    195 KBytes
[ 15]   3.00-4.00   sec   115 MBytes   962 Mbits/sec    0    195 KBytes
[ 15]   4.00-5.00   sec   113 MBytes   952 Mbits/sec    0    195 KBytes
[ 15]   5.00-6.00   sec   115 MBytes   962 Mbits/sec    0    195 KBytes
[ 15]   6.00-7.00   sec   114 MBytes   953 Mbits/sec    0    195 KBytes
[ 15]   7.00-8.00   sec   114 MBytes   954 Mbits/sec    0    195 KBytes
[ 15]   8.00-9.00   sec   115 MBytes   963 Mbits/sec    0    195 KBytes
[ 15]   9.00-10.00  sec   113 MBytes   951 Mbits/sec    0    195 KBytes
[ 15]  10.00-11.00  sec   115 MBytes   962 Mbits/sec    0    198 KBytes
[ 15]  11.00-12.00  sec   114 MBytes   952 Mbits/sec    0    198 KBytes
[ 15]  12.00-13.00  sec   113 MBytes   952 Mbits/sec   90    195 KBytes
[ 15]  13.00-14.00  sec   115 MBytes   962 Mbits/sec    0    195 KBytes
[ 15]  14.00-15.00  sec  91.2 MBytes   765 Mbits/sec    0    195 KBytes
[ 15]  15.00-16.00  sec  91.2 MBytes   765 Mbits/sec    0    195 KBytes
[ 15]  16.00-17.00  sec   113 MBytes   952 Mbits/sec    0    195 KBytes
[ 15]  17.00-18.00  sec   115 MBytes   962 Mbits/sec    0    195 KBytes
[ 15]  18.00-19.00  sec   113 MBytes   952 Mbits/sec    0    195 KBytes
[ 15]  19.00-20.00  sec   115 MBytes   962 Mbits/sec    0    195 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-20.00  sec  2.18 GBytes   937 Mbits/sec   92             sender
[ 15]   0.00-20.04  sec  2.19 GBytes   937 Mbits/sec                  receiver
```

Figure 20. Running iPerf3 client on host h1.

Figure 20 shows the iPerf3 test output report. The average achieved throughputs are 937 Mbps (sender) and 937 Mbps (receiver), and the number of retransmissions is 92 (due to the injected packet loss - 0.01%).

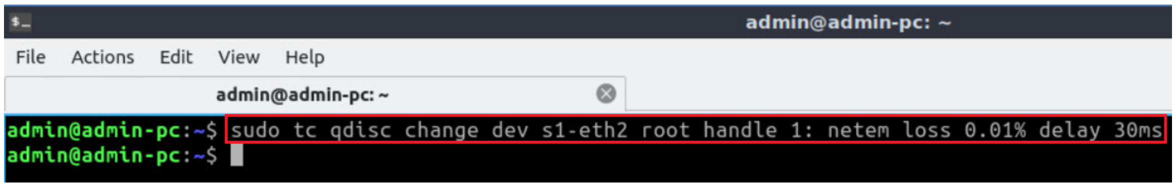
Step 4. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3.2 Throughput test with 30ms delay

In this test, we measure the throughput between host h1 and host h2 while introducing 30ms delay on the switch S1's *s1-eth2* interface. Apply the following steps:

Step 1. In order to add delay to the switch 1 or interface s1-eth2, go back to the Client's terminal, run the following command to modify the previous rule to include 30ms delay:

```
sudo tc qdisc change dev s1-eth2 root handle 1: netem loss 0.01% delay 30ms
```



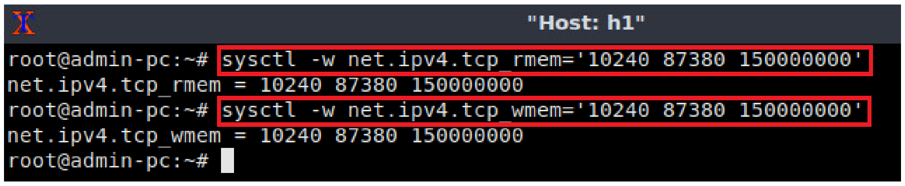
```
admin@admin-pc:~$ sudo tc qdisc change dev s1-eth2 root handle 1: netem loss 0.01% delay 30ms
admin@admin-pc:~$ 
```

Figure 21. Injecting 30ms delay on switch S1's *s1-eth2* interface.

Step 2. In host h1's terminal, modify the TCP buffer size by typing the following commands: *sysctl -w net.ipv4.tcp_rmem='10,240 87,380 150,000,000'* and *sysctl -w net.ipv4.tcp_wmem='10,240 87,380 150,000,000'*. This TCP buffer is explained later in future labs.

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'
```

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'
```

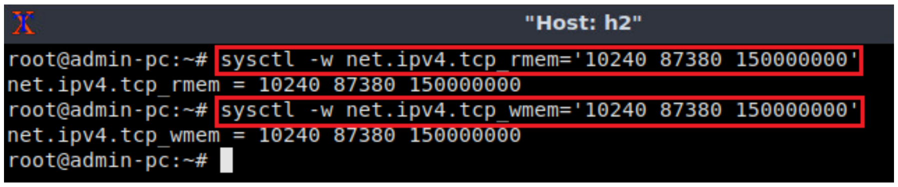


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'
net.ipv4.tcp_rmem = 10240 87380 150000000
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'
net.ipv4.tcp_wmem = 10240 87380 150000000
root@admin-pc:~# 
```

Figure 22. Modifying the TCP buffer size on host h1.

Step 3. In host h2's terminal, also modify the TCP buffer size by typing the following commands: *sysctl -w net.ipv4.tcp_rmem='10,240 87,380 150,000,000'* and *sysctl -w net.ipv4.tcp_wmem='10,240 87,380 150,000,000'*.

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'
sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'
```



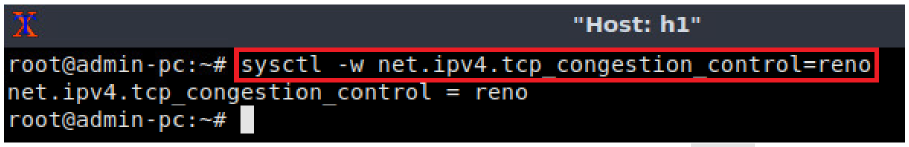
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'
net.ipv4.tcp_rmem = 10240 87380 150000000
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'
net.ipv4.tcp_wmem = 10240 87380 150000000
root@admin-pc:~#
```

Figure 23. Modifying the TCP buffer size on host h2.

3.2.1 TCP Reno

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Reno by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=reno
```

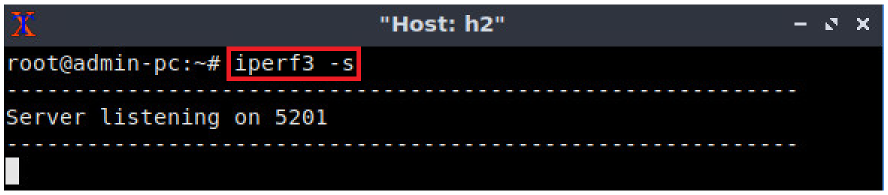


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
root@admin-pc:~#
```

Figure 24. Changing TCP congestion control algorithm to `reno` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

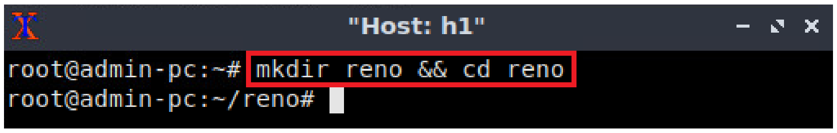


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 25. Starting iPerf3 server on host h2.

Step 3. Create and enter to a new directory *reno* on host h1's terminal:

```
mkdir reno && cd reno
```

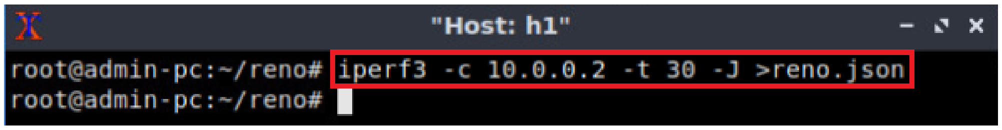


```
root@admin-pc:~# mkdir reno && cd reno
root@admin-pc:~/reno#
```

Figure 26. Creating and entering a new directory *reno*.

Step 4. Launch iPerf3 in client mode on host h1's terminal. The `-J` option is used to produce a JSON output and the redirection operator `>` to send the standard output to a file.

```
iperf3 -c 10.0.0.2 -t 30 -J > reno.json
```

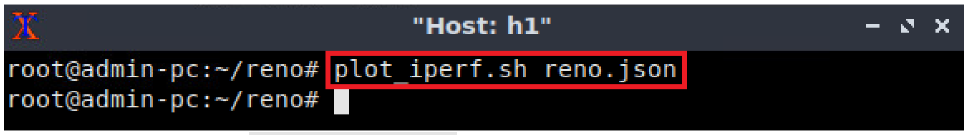


```
"Host: h1"
root@admin-pc:~/reno# iperf3 -c 10.0.0.2 -t 30 -J >reno.json
root@admin-pc:~/reno#
```

Figure 27. Running iPerf3 client on host h1 and redirecting the output to *reno.json*.

Step 5. Once the test is finished, type the following command to generate the output plots for iPerf3's JSON file:

```
plot_iperf.sh reno.json
```



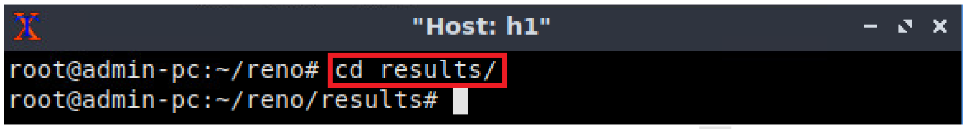
```
"Host: h1"
root@admin-pc:~/reno# plot_iperf.sh reno.json
root@admin-pc:~/reno#
```

Figure 28. `plot_iperf.sh` script generating output results.

This plotting script generates PDF files for the following fields: congestion window (*cwnd.pdf*), retransmits (*retransmits.pdf*), Round-Trip Time (*RTT.pdf*), Round-Trip Time variance (*RTT_Var.pdf*), throughput (*throughput.pdf*), maximum transmission unit (*MTU.pdf*), bytes transferred (*bytes.pdf*). The plotting script also generates a CSV file (*1.dat*) to be used by applicable programs. These files are stored in a directory *results* created in the same directory where the script was executed as shown in the figure below.

Step 6. Navigate to the results folder using the `cd` command.

```
cd results/
```

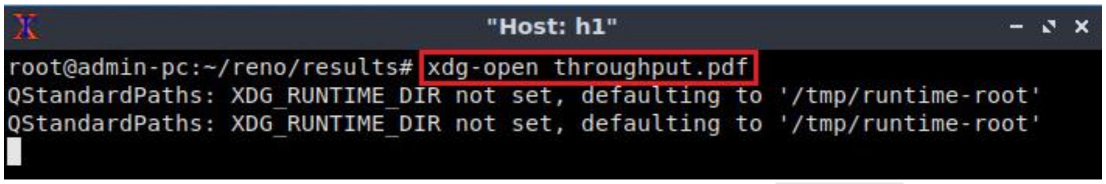


```
"Host: h1"
root@admin-pc:~/reno# cd results/
root@admin-pc:~/reno/results#
```

Figure 29. Entering the results directory using the `cd` command.

Step 7. To open any of the generated files, use the `xdg-open` command followed by the file name. For example, to open the *throughput.pdf* file, use the following command:

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



```
"Host: h1"
root@admin-pc:~/reno/results# xdg-open throughput.pdf
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

Figure 30. Opening the *throughput.pdf* file using `xdg-open`.

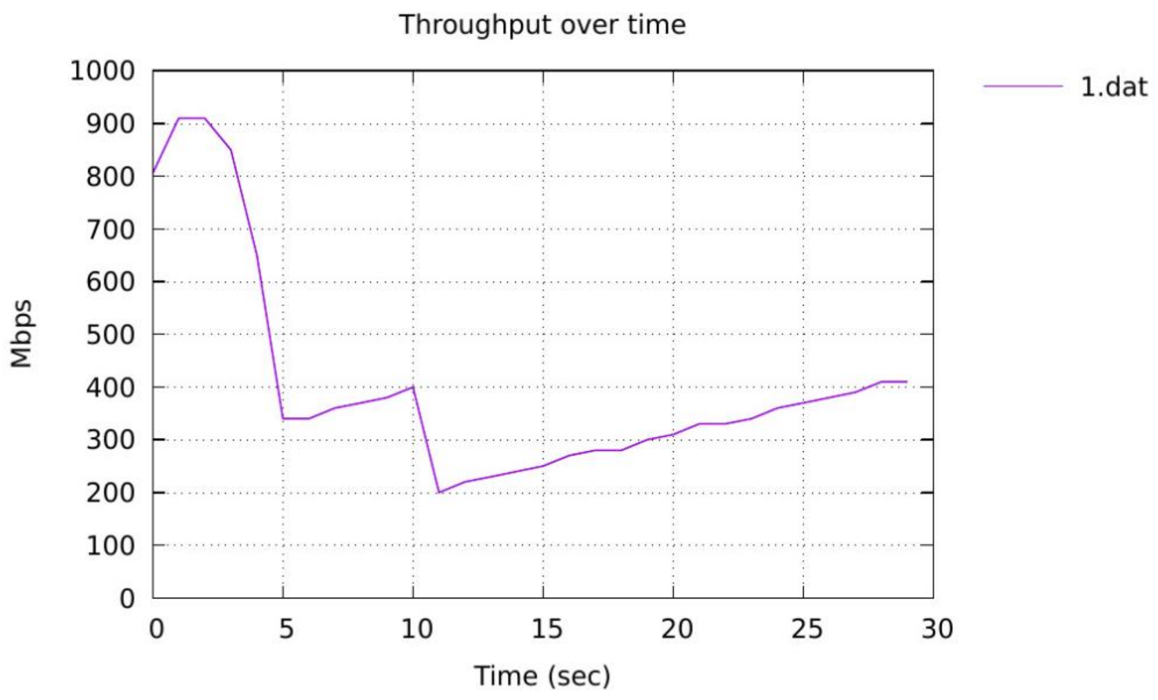
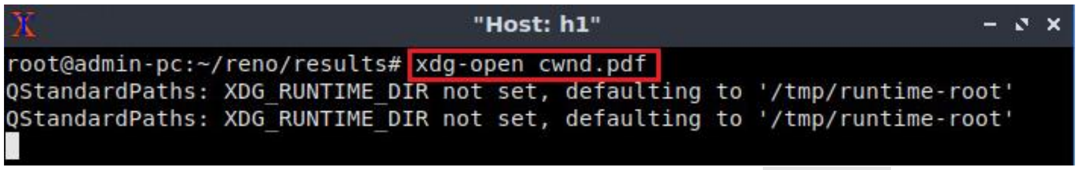


Figure 31. Reno's throughput.

Step 8. Close the *throughput.pdf* file and open the *cwnd.pdf* file using the following command:

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



```
root@admin-pc:~/reno/results# xdg-open cwnd.pdf
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

Figure 32. Opening the *throughput.pdf* file using `xdg-open`.

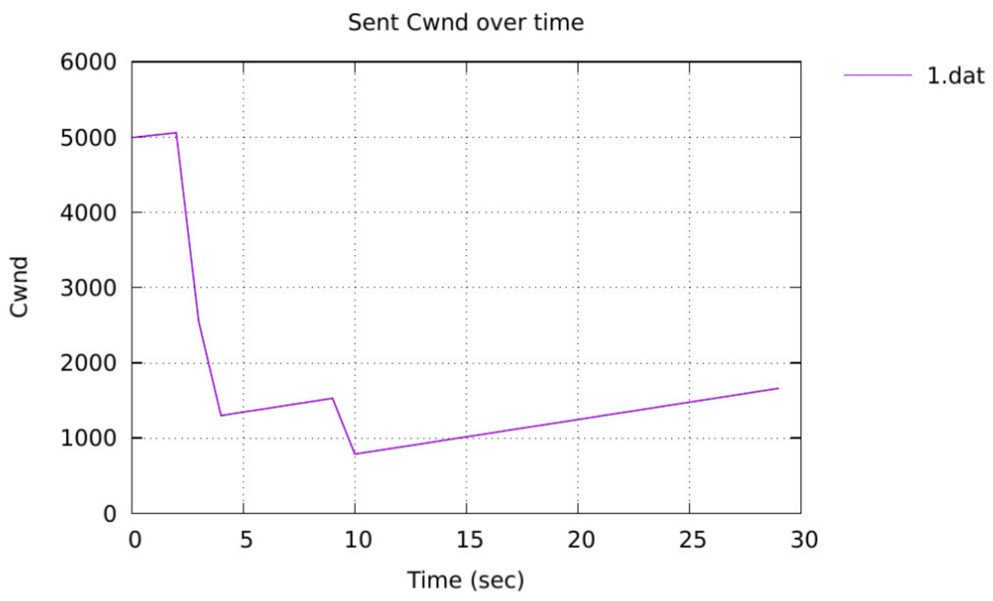
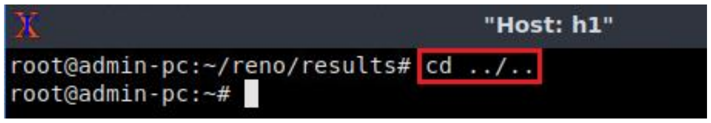


Figure 33. Reno's congestion window.

Step 9. 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.

Step 10. Exit the */reno/results* directory by using the following command on host h1's terminal:

```
cd ../..
```



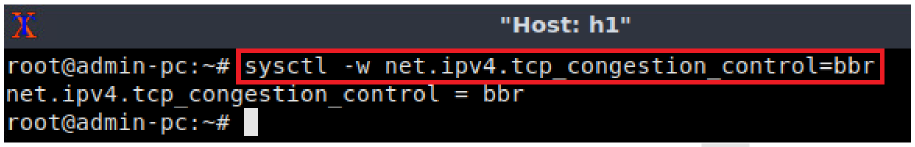
```
root@admin-pc:~/reno/results# cd ../..
root@admin-pc:~#
```

Figure 34. Exiting the */reno/results* directory.

3.2.2 TCP BBR

Step 1. In host h1's terminal, change the TCP congestion control algorithm to BBR by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=bbr
```

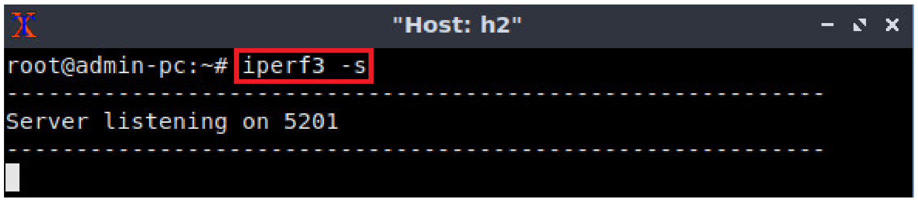


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=bbr
net.ipv4.tcp_congestion_control = bbr
root@admin-pc:~#
```

Figure 35. Changing TCP congestion control algorithm to `bbr` on host h1.

Step 2. Launch iPerf3 in server mode on host h2's terminal:

```
iperf3 -s
```

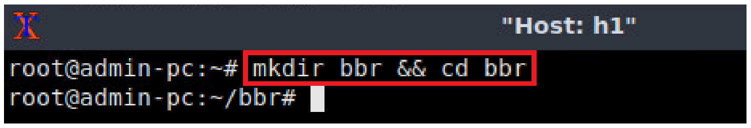


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 36. Starting iPerf3 server on host h2.

Step 3. Create and enter to a new directory *bbr* host h1's terminal*:*

```
mkdir bbr && cd bbr
```

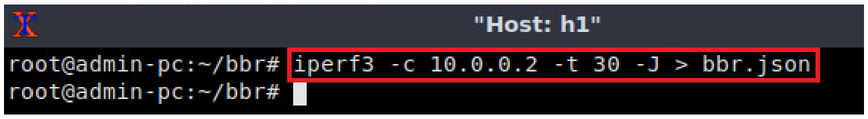


```
root@admin-pc:~# mkdir bbr && cd bbr
root@admin-pc:~/bbr#
```

Figure 37. Creating and entering a new directory *bbr* .

Step 4. Launch iPerf3 in client mode on host h1's terminal. The `-J` option is used to produce a JSON output and the redirection operator `>` to send the standard output to a file.

```
iperf3 -c 10.0.0.2 -t 30 -J > bbr.json
```

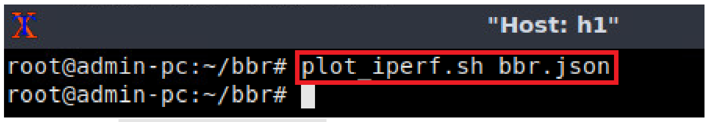


```
"Host: h1"
root@admin-pc:~/bbr# iperf3 -c 10.0.0.2 -t 30 -J > bbr.json
root@admin-pc:~/bbr#
```

Figure 38. Running iPerf3 client on host h1 and redirecting the output to *bbr.json*.

Step 5. To generate the output plots for iPerf3's JSON file run the following command:

```
plot_iperf.sh bbr.json
```



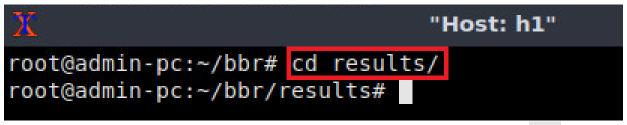
```
"Host: h1"
root@admin-pc:~/bbr# plot_iperf.sh bbr.json
root@admin-pc:~/bbr#
```

Figure 39. `plot_iperf.sh` script generating output results.

This plotting script generates PDF files for the following fields: congestion window (*cwnd.pdf*), retransmits (*retransmits.pdf*), Round-Trip Time (*RTT.pdf*), Round-Trip Time variance (*RTT_Var.pdf*), throughput (*throughput.pdf*), maximum transmission unit (*MTU.pdf*), bytes transferred (*bytes.pdf*). The plotting script also generates a CSV file (*1.dat*) to be used by applicable programs. These files are stored in a directory *results* created in the same directory where the script was executed as shown in the figure below.

Step 6. Navigate to the results folder using the `cd` command.

```
cd results/
```

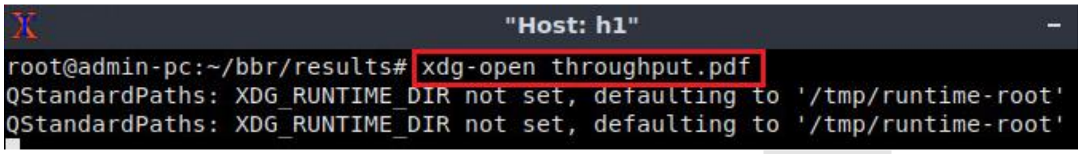


```
"Host: h1"
root@admin-pc:~/bbr# cd results/
root@admin-pc:~/bbr/results#
```

Figure 40. Entering the results directory using the `cd` command.

Step 7. To open any of the generated files, use the `xdg-open` command followed by the file name. For example, to open the *throughput.pdf* file, use the following command:

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



```
"Host: h1"
root@admin-pc:~/bbr/results# xdg-open throughput.pdf
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

Figure 41. Opening the *throughput.pdf* file using `xdg-open`.

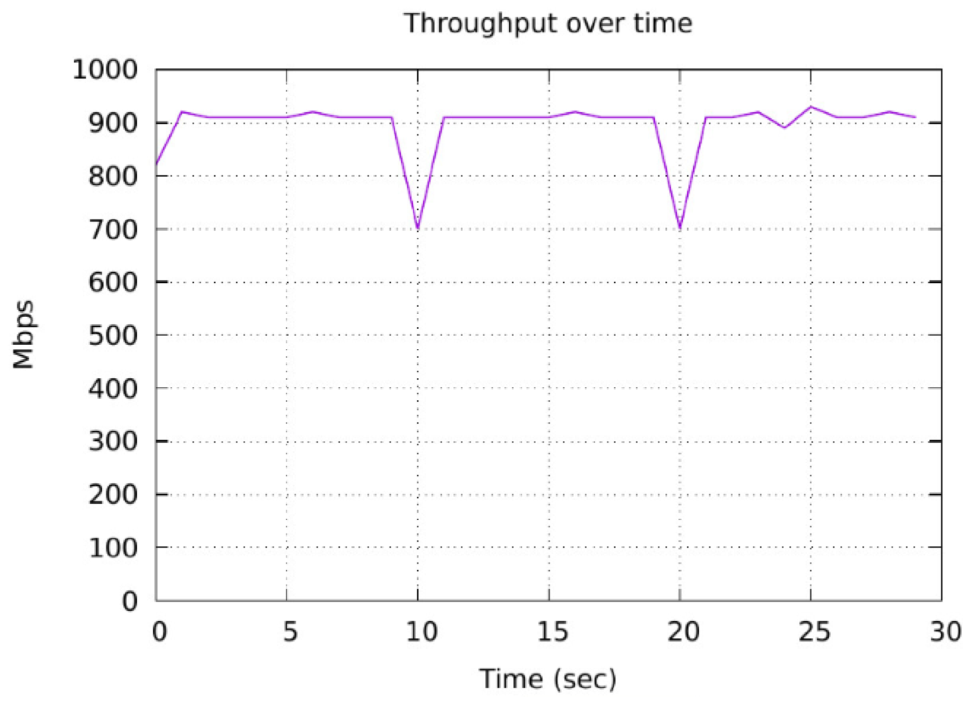
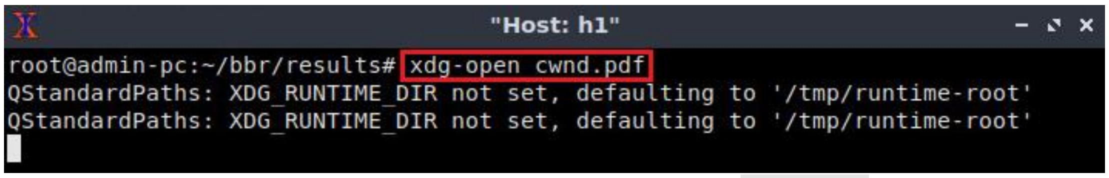


Figure 42. BBR's throughput.

Step 8. Figure 42 shows that in steady state, BBR has already attained the maximum throughput, which is over 900 Mbps (the bottleneck bandwidth is 1 Gbps, with an observed effective bandwidth of ~937 Gbps). Note also the periodic (short) drain intervals, where the throughput decreases to ~75% of maximum throughput, as discussed in Section 1.3. To proceed, close the *throughput.pdf* file and open the *cwnd.pdf* file using the following command:

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



```
root@admin-pc:~/bbr/results# xdg-open cwnd.pdf
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

Figure 43. Opening the *cwnd.pdf* file using `xdg-open`.

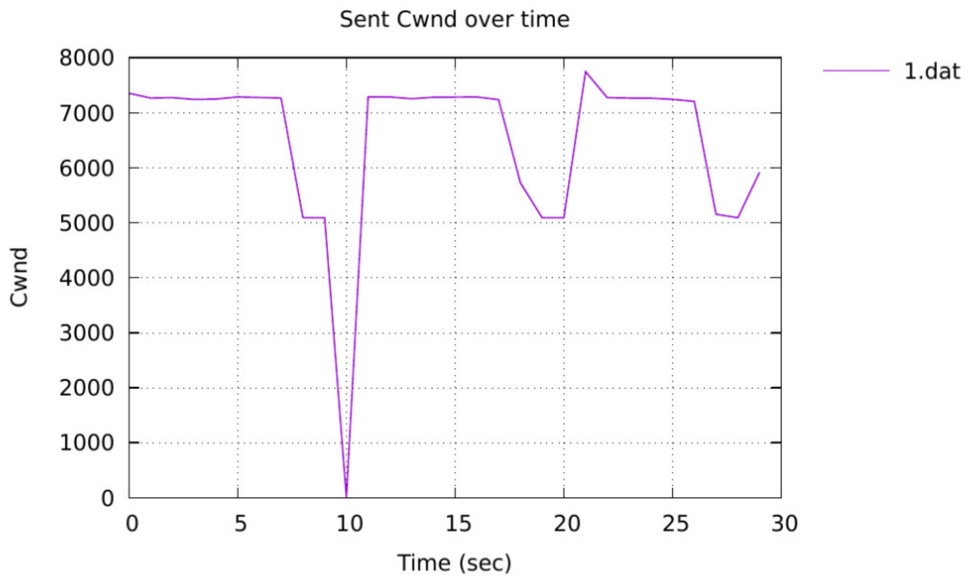
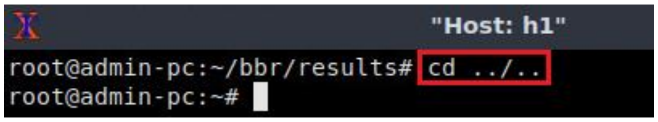


Figure 44. BBR's congestion window.

Step 9. 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.

Step 10. Exit the */bbr/results* directory by using the following command on host h1's terminal:

```
cd ../..
```



```
root@admin-pc:~/bbr/results# cd ../..
root@admin-pc:~#
```

Figure 45. Exiting the */bbr/results* directory.

It is clear from the above test that when introducing delay, BBR preforms significantly better than Reno.

This concludes Lab 7. Stop the emulation and then exit out of MiniEdit.

References

1. K. Fall, S. Floyd, "Simulation-based comparisons of tahoe, reno, and sack TCP," Computer Communication Review, vol. 26, issue 3, Jul. 1996.
2. J. Kurose, K. Ross, "Computer networking, a top down approach," Pearson, 6th Edition, 2017.
3. D. Leith, R. Shorten, Y. Lee, "H-TCP: a framework for congestion control in high-speed and long-distance networks," Hamilton Institute Technical Report, Aug. 2005. [Online]. Available: http://www.hamilton.ie/net/htcp2005.pdf.
4. E. Dart, L. Rotman, B. Tierney, M. Hester, J. Zurawski, "The science DMZ: a network design pattern for data-intensive science," in Proceedings of the International

Conference on High Performance Computing, Networking, Storage and Analysis, Nov. 2013.

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.
6. S. Ha, I., Rhee, L. Xu, “CUBIC: a new TCP-friendly high-speed TCP variant,” ACM SIGOPS operating systems review, vol. 42, issue 5, pp. 64-74, Jul. 2008.
7. Leith D, Shorten R. H-TCP: TCP congestion control for high bandwidth-delay product paths. draft-leith-tcp-htcp-06 (work in progress). 2008 Apr.
8. System information variables – sysctl(7). [Online]. Available: https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt.

NETWORK TOOLS AND PROTOCOLS

Lab 8: Bandwidth-delay Product and TCP Buffer Size

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 TCP buffers, BDP, and TCP window 3
1.1 TCP buffers 3
1.2 Bandwidth-delay product.. 4
1.3 Practical observations on setting TCP buffer size 5
1.4 TCP window size calculated value.. 7
1.5 Zero window.. 8
2 Lab topology.. 8
2.1 Starting host h1 and host h2 9
2.2 Emulating 10 Gbps high-latency WAN 10
3 BDP and buffer size 13
3.1 Window size in sysctl.. 13
4 Modifying buffer size and throughput test.. 15
References 17

Overview

This lab explains the bandwidth-delay product (BDP) and how to modify the TCP buffer size in Linux systems. Throughput measurements are also conducted to test and verify TCP buffer configurations based on the BDP.

Objectives

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

1. Understand BDP.
2. Define and calculate TCP window size.
3. Modify the TCP buffer size with sysctl, based on BDP calculations.
4. Emulate WAN properties in Mininet.
5. Achieve full throughput in WANs by modifying the size of TCP buffers.

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 TCP buffers, BDP, and TCP window.
2. Section 2: Lab topology.
3. Section 3: BDP and buffer size experiments.
4. Section 4: Modifying buffer size and throughput test.

1 Introduction to TCP buffers, BDP, and TCP window

1.1 TCP buffers

The TCP send and receive buffers may impact the performance of Wide Area Networks (WAN) data transfers. Consider Figure 1. At the sender side, TCP receives data from the application layer and places it in the TCP send buffer. Typically, TCP fragments the data in the buffer into maximum segment size (MSS) units. In this example, the MSS is 100 bytes. Each segment carries a sequence number, which is the byte-stream number of the first byte in the segment. The corresponding acknowledgement (Ack) carries the number of the next expected byte (e.g., Ack-101 acknowledges bytes 1-100, carried by the first segment). At the receiver, TCP receives data from the network layer and places it into the TCP receive buffer. TCP delivers the data *in order* to the application. E.g., bytes contained in a segment, say segment 2 (bytes 101-200), cannot be delivered to the application layer before the bytes contained in segment 1 (bytes 1-100) are delivered to the application. At any given time, the TCP receiver indicates the TCP sender how many bytes the latter can send, based on how much free buffer space is available at the receiver.

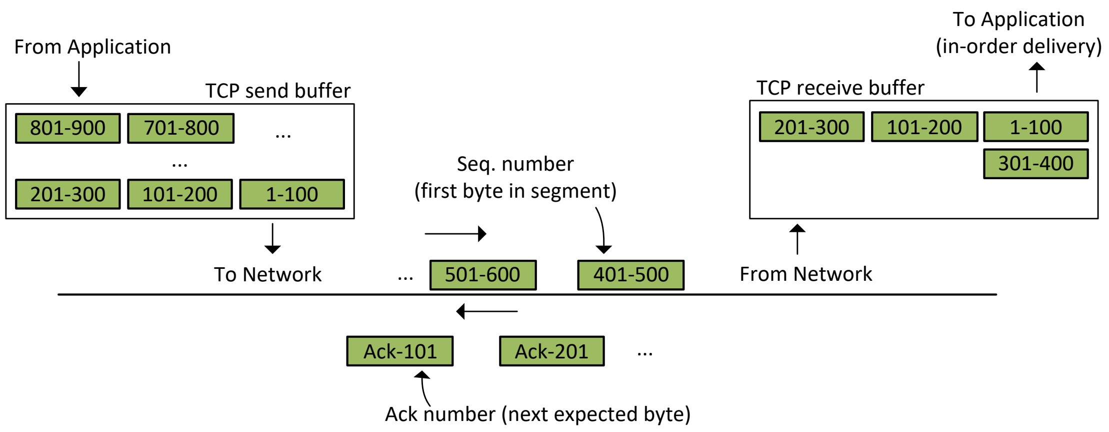


Figure 1. TCP send and receive buffers.

1.2 Bandwidth-delay product

In many WANs, the round-trip time (RTT) is dominated by the propagation delay. Long RTTs along and TCP buffer size have throughput implications. Consider a 10 Gbps WAN with a 50-millisecond RTT. Assume that the TCP send and receive buffer sizes are set to 1 Mbyte (1 Mbyte = 1024^2 bytes = 1,048,576 bytes = 1,048,576 · 8 bits = 8,388,608 bits). With a bandwidth (Bw) of 10 Gbps, this number of bits is approximately transmitted in

$$\mathrm{T_{tx}} = \frac{\#\ \mathrm{bits}}{\mathrm{Bw}} = \frac{8{,}388{,}608}{10 \cdot 10^9} = 0.84\ \mathrm{milliseconds.}$$

I.e., after 0.84 milliseconds, the content of the TCP send buffer will be completely sent. At this point, TCP must wait for the corresponding acknowledgements, which will only start arriving at t = 50 milliseconds. This means that the sender only uses 0.84/50 or 1.68% of the available bandwidth.

The solution to that above problem lies in allowing the sender to continuously transmit segments until the corresponding acknowledgments arrive back. Note that the first acknowledgement arrives after an RTT. The number of bits that can be transmitted in a RTT period is given by the bandwidth of the channel in bits per second multiplied by the

RTT. This quantity is referred to as the bandwidth-delay product (BDP). For the above example, the buffer size must be greater than or equal to the BDP:

$$\text{TCP buffer size} \geq \text{BDP} = (10 \cdot 10^9)(50 \cdot 10^{-3}) = 500{,}000{,}000 \text{ bits} = 62{,}500{,}000 \text{ bytes}.$$

The first factor ($10 \cdot 10^9$) is the bandwidth; the second factor ($50 \cdot 10^{-3}$) is the RTT. For practical purposes, the TCP buffer can be also expressed in Mbytes (1 Mbyte = 1024^2 bytes) or Gbytes (1 Gbyte = 1024^3 bytes). The above expression, in Mbytes, is

$$\text{TCP buffer size} \geq 62{,}500{,}000 \text{ bytes} = 59.6 \text{ Mbytes} \approx 60 \text{ Mbytes}.$$

1.3 Practical observations on setting TCP buffer size

Linux systems configuration. Linux assumes that half of the send/receive TCP buffers are used for internal structures. Thus, only half of the buffer size is used to store segments. This implies that if a TCP connection requires certain buffer size, then the administrator must configure the buffer size equals to twice that size. For the previous example, the TCP buffer size must be:

$$\text{TCP buffer size} \geq 2 \cdot 60 \text{ Mbytes} = 120 \text{ Mbytes}.$$

Packet loss scenarios and TCP BBR[1]. TCP provides a reliable, in-order delivery service. Reliability means that bytes successfully received must be acknowledged. In-order delivery means that the receiver only delivers bytes to the application layer in sequential order. The memory occupied by those bytes will be deallocated from the receive buffer after passing the bytes to the application layer. This process has some performance implications, as illustrated next. Consider Figure 2, which shows a TCP receive buffer. Assume that the segment carrying bytes 101-200 is lost. Although the receiver has successfully received bytes 201-900, it cannot deliver to the application layer until bytes 101-200 are received. Note that the receive buffer may become full, which would block the sender from utilizing the channel.

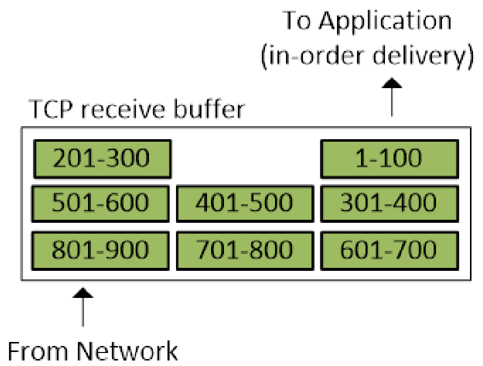


Figure 2. TCP receive buffer. Although bytes 301-900 have been received, they cannot be delivered to the application until the segment carrying bytes 201-300 are received.

While setting the buffer size equal to the BDP is acceptable when traditional congestion control algorithms are used (e.g., Reno[2], Cubic[3], HTCP[4]), this size may not allow the full utilization of the channel with BBR[1]. In contrast to other algorithms, BBR does not reduce the transmission rate after a packet loss. For example, suppose that a packet sent at t = 0 is lost, as shown in Figure 3. At t = RTT, the acknowledgement identifying the packet to retransmit is received. By then, the sender has sent BDP bits, which will be stored in the receive buffer. This data cannot be delivered yet to the application, because of the in-order delivery requirement. Since the receive buffer has a capacity of BDP only, the sender is temporarily blocked until the acknowledgement for the retransmitted data is received at t = 2·RTT. Thus, the throughput over the period t = 0 to t = 2·RTT is reduced by half:

$$\text{throughput} = \frac{\#\text{ bits transmitted}}{\text{period}} = \frac{\text{Bw} \cdot \text{RTT}}{2 \cdot \text{RTT}} = \frac{\text{Bw}}{2}.$$

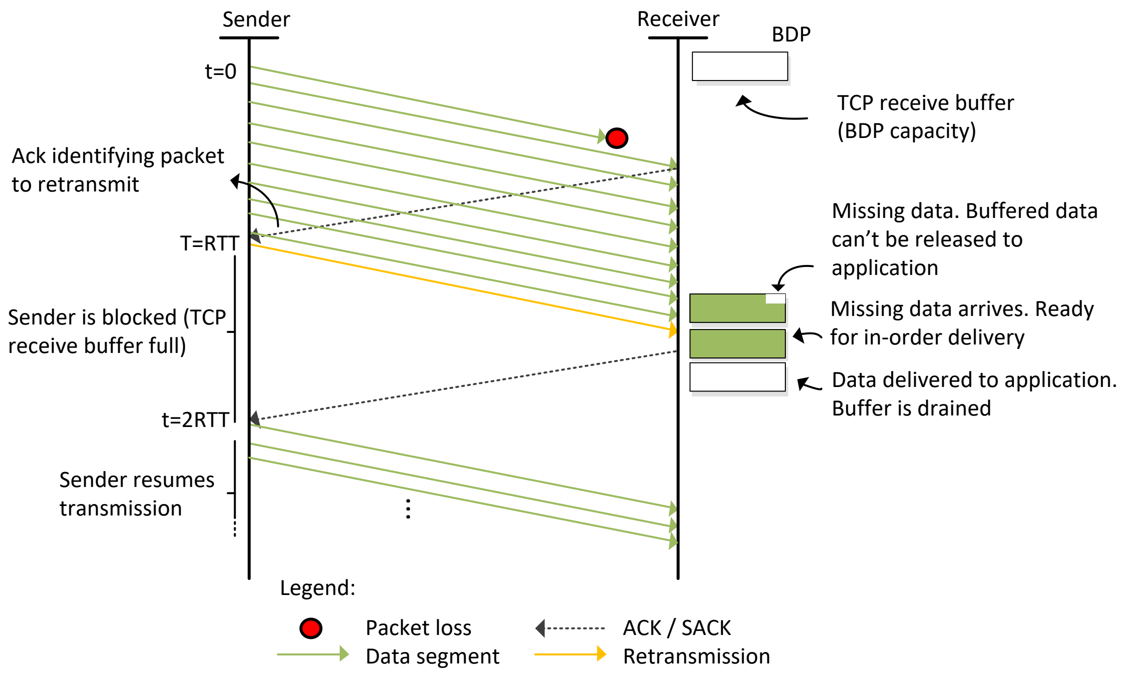

Sender
Receiver
BDP
t=0
TCP receive buffer (BDP capacity)
Ack identifying packet to retransmit
T=RTT
Missing data. Buffered data can't be released to application
Sender is blocked (TCP receive buffer full)
Missing data arrives. Ready for in-order delivery
Data delivered to application. Buffer is drained
t=2RTT
Sender resumes transmission
Legend:
Packet loss
Data segment
ACK / SACK
Retransmission

Figure 3. A scenario where a TCP receive buffer size of BDP cannot prevent throughput degradation.

With BBR, to fully utilize the available bandwidth, the TCP send and receive buffers must be large enough to prevent such situation. Figure 4 shows an example on how a TCP buffer size of 2·BDP mitigates packet loss.

High to moderate packet loss scenarios, using TCP BBR:

TCP send/receive buffer ≥ several BDPs (e.g., 4 · BDP)

Continuing with the example of Section 1.2, in a Linux system using TCP BBR, the send/receive buffers for a BDP of 60 Mbytes in a high to moderate packet loss scenario should be:

$$\text{TCP buffer size} \geq (2 \cdot 60 \text{ Mbytes}) \cdot 4 = 480 \text{ Mbytes}.$$

The factor 2 is because of the Linux systems configuration, and the factor 4 is because of the use of TCP BBR in a high to moderate packet loss scenario.

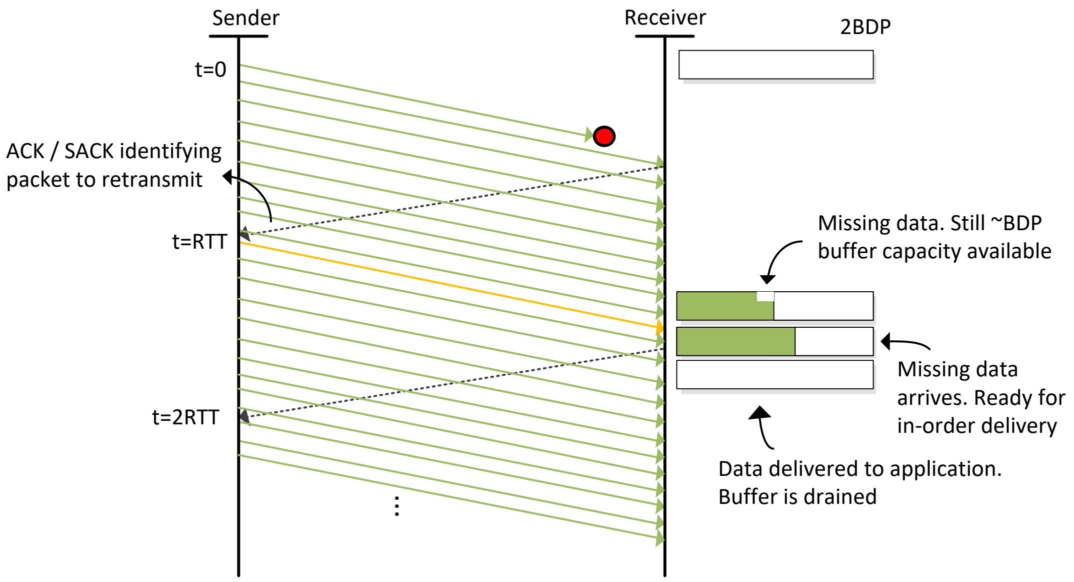

Sender
Receiver
2BDP
t=0
ACK / SACK identifying packet to retransmit
t=RTT
t=2RTT
Missing data. Still ~BDP buffer capacity available
Missing data arrives. Ready for in-order delivery
Data delivered to application. Buffer is drained

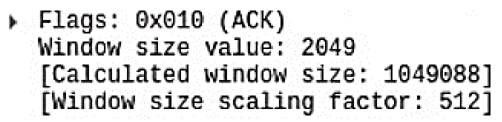
Figure 4. A scenario where a TCP buffer size of 2·BDP mitigates packet loss.

1.4 TCP window size calculated value

The receiver must constantly communicate with the sender to indicate how much free buffer space is available in the TCP receive buffer. This information is carried in a TCP header field called window size. The window size has a maximum value of 65,535 bytes, as the header value allocated for the window size is two bytes long (16 bits; 2^{16}-1 = 65,535). However, this value is not large enough for high-bandwidth high-latency networks. Therefore, *TCP window scale option* was standardized in RFC 1323[5]. By using this option, the calculated window size may be increased up to a maximum value of 1,073,725,440 bytes. When advertising its window, a device also advertises the *scale factor* (multiplier) that will be used throughout the session. The TCP window size is calculated as follows:

$$\text{Scaled TCP}_{\text{Win}} = \text{TCP}_{\text{Win}} \cdot \text{Scaling Factor}$$

As an example, consider the following example. For an advertised TCP window of 2,049 and a scale factor of 512, then the final window size is 1,049,088 bytes. Figure 5 displays a packet inspected in Wireshark protocol analyzer for this numerical example.



```
▸ Flags: 0x010 (ACK)
  Window size value: 2049
  [Calculated window size: 1049088]
  [Window size scaling factor: 512]
```

Figure 5. Window Scaling in Wireshark.

1.5 Zero window

When the TCP buffer is full, a window size of zero is advertised to inform the other end that it cannot receive any more data. When a client sends a TCP window of zero, the server will pause its data transmission, and waits for the client to recover. Once the client is recovered, it digests data, and inform the server to resume the transmission again by setting again the TCP window.

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.

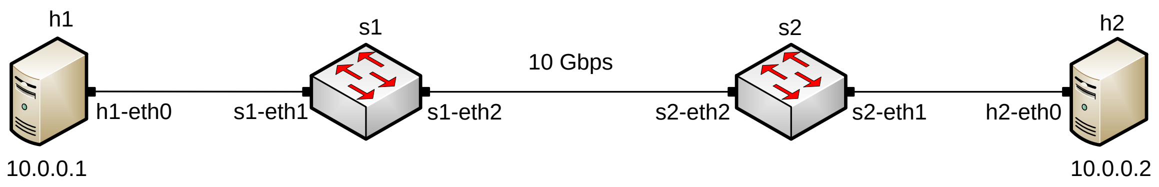


Figure 6. Lab topology.

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`.

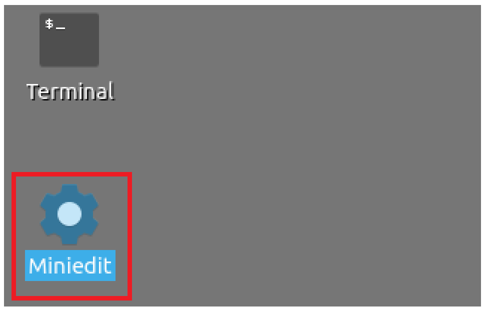


Figure 7. Miniedit shortcut.

Step 2. On Miniedit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *lab8.mn* topology file and click on *Open*.

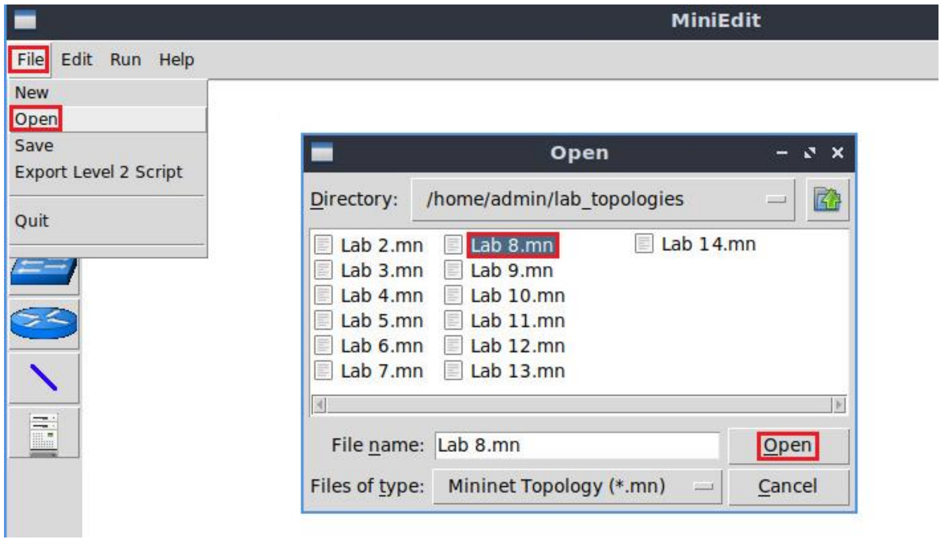


Figure 8. Miniedit's *Open* dialog.

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.

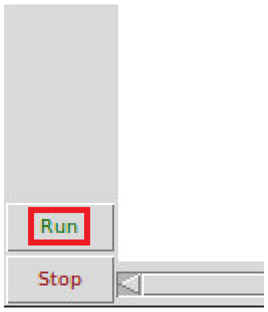


Figure 9. Running the emulation.

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 the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

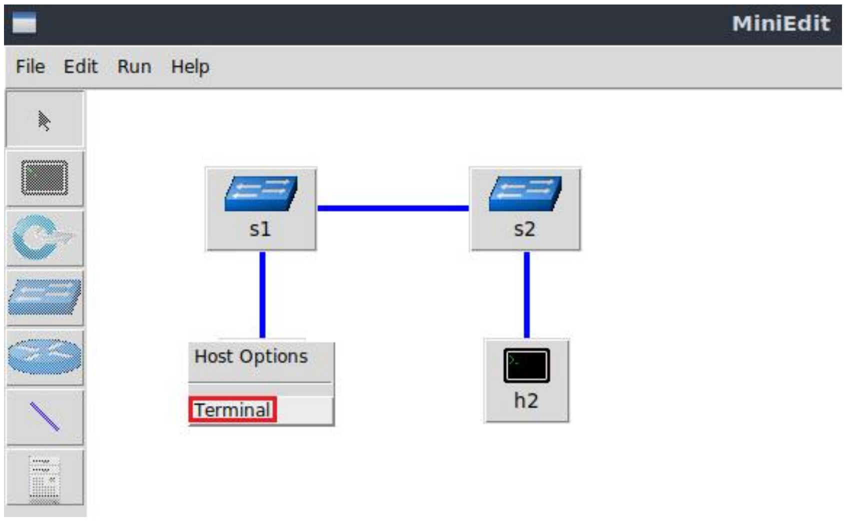
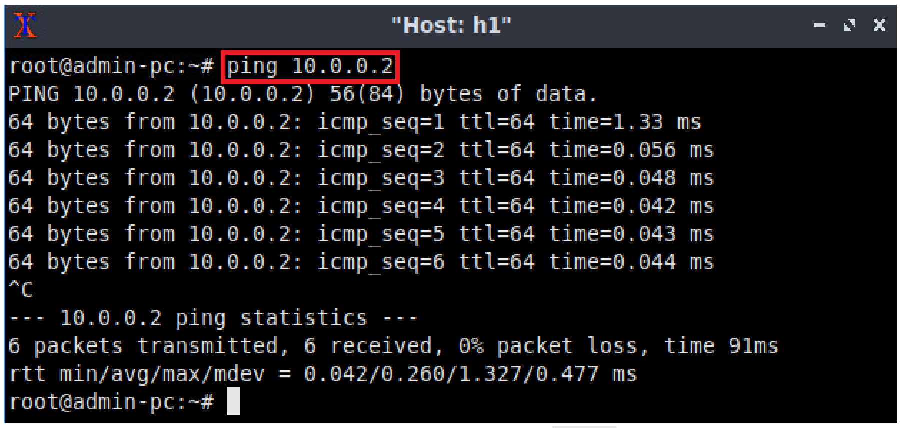


Figure 10. Opening a terminal on host h1.

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.



```
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 11. Connectivity test using `ping` command.

Figure 11 indicates that there is connectivity between host h1 and host h2.

2.2 Emulating 10 Gbps high-latency WAN

This section emulates a high-latency WAN by introducing delays to the network. We will first set the bandwidth between hosts 1 and 2 to 10 Gbps. Then, we will emulate a 20 ms delay and measure the throughput.

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

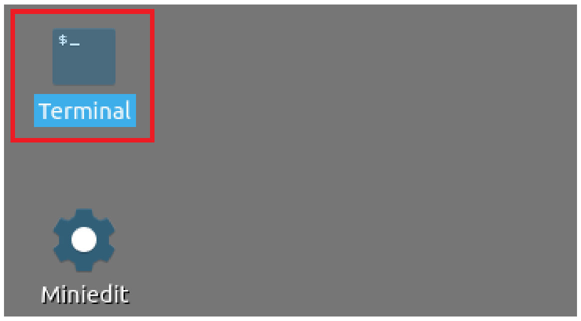
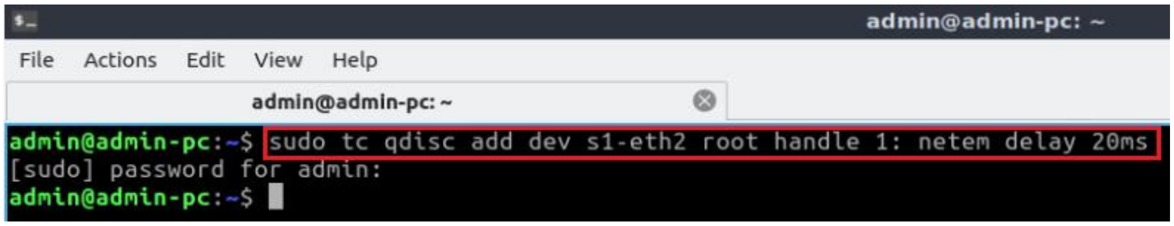


Figure 12. Shortcut to open a Linux terminal.

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 on S1's *s1-eth2* interface.

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



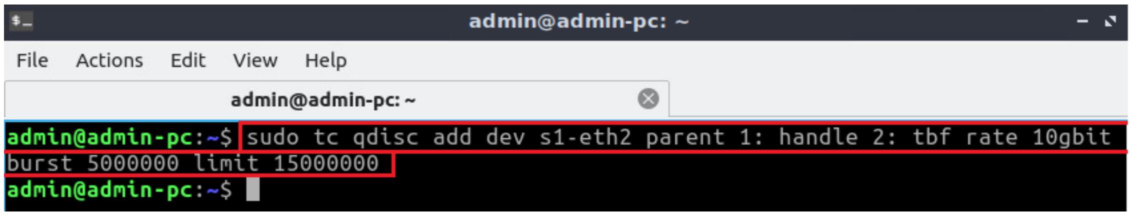
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 13. Adding 20ms delay to switch S1's *s1-eth2* interface.

Step 3. Modify the bandwidth of the link connecting the switches S1 and S2: on the same terminal, type the command below. This command sets the bandwidth to 10 Gbps on S1's *s1-eth2* interface. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit burst 5000000
limit 15000000
```

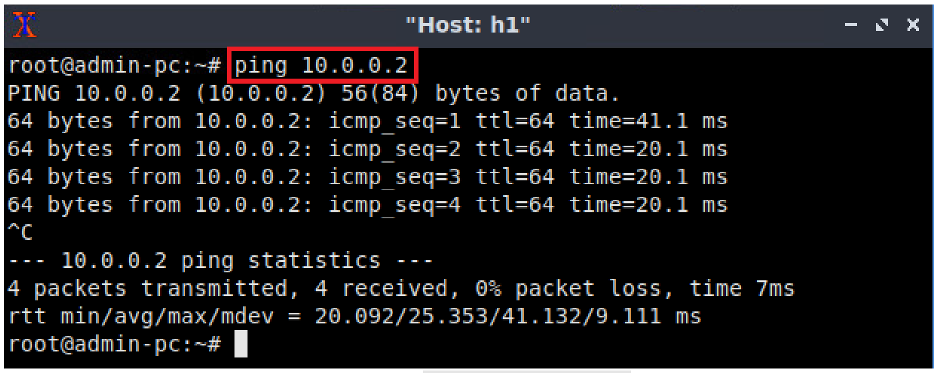


```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$
```

Figure 14. Limiting the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface.

Step 3. On h1's terminal, type `ping 10.0.0.2`. This command tests the connectivity between host h1 and host h2. The test was initiated by h1 as the command is executed on h1's terminal.

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.



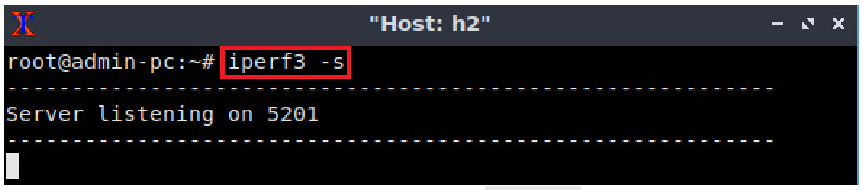
```
"Host: h1"
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=41.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.092/25.353/41.132/9.111 ms
root@admin-pc:~#
```

Figure 15. 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 20.092, 25.353, 41.132, and 9.111 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

Step 4. The user can now verify the rate limit configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in H2's terminal:

```
iperf3 -s
```

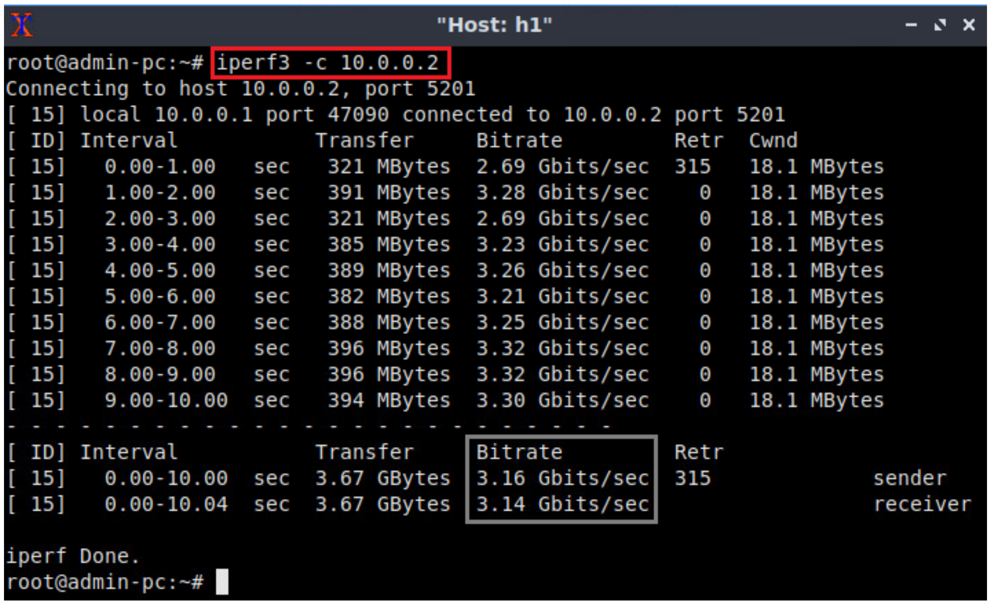


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 16. Host h2 running `iperf3` as server.

Step 5. Now to launch iPerf3 in client mode again by running the command `iperf3 -c 10.0.0.2` in h1's terminal:

```
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 47090 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   321 MBytes  2.69 Gbits/sec  315   18.1 MBytes
[ 15]   1.00-2.00   sec   391 MBytes  3.28 Gbits/sec    0   18.1 MBytes
[ 15]   2.00-3.00   sec   321 MBytes  2.69 Gbits/sec    0   18.1 MBytes
[ 15]   3.00-4.00   sec   385 MBytes  3.23 Gbits/sec    0   18.1 MBytes
[ 15]   4.00-5.00   sec   389 MBytes  3.26 Gbits/sec    0   18.1 MBytes
[ 15]   5.00-6.00   sec   382 MBytes  3.21 Gbits/sec    0   18.1 MBytes
[ 15]   6.00-7.00   sec   388 MBytes  3.25 Gbits/sec    0   18.1 MBytes
[ 15]   7.00-8.00   sec   396 MBytes  3.32 Gbits/sec    0   18.1 MBytes
[ 15]   8.00-9.00   sec   396 MBytes  3.32 Gbits/sec    0   18.1 MBytes
[ 15]   9.00-10.00  sec   394 MBytes  3.30 Gbits/sec    0   18.1 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  3.67 GBytes  3.16 Gbits/sec  315             sender
[ 15]   0.00-10.04  sec  3.67 GBytes  3.14 Gbits/sec                  receiver

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

Figure 17. iPerf3 throughput test.

Notice the measured throughput now is approximately 3 Gbps, which is different than the value assigned in our `tbf` rule. Next, we explain why the 10 Gbps maximum theoretical bandwidth is not achieved.

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.

3 BDP and buffer size

In connections that have a small BDP (either because the link has a low bandwidth or because the latency is small), buffers are usually small. However, in high-bandwidth high-latency networks, where the BDP is large, a larger buffer is required to achieve the maximum theoretical bandwidth.

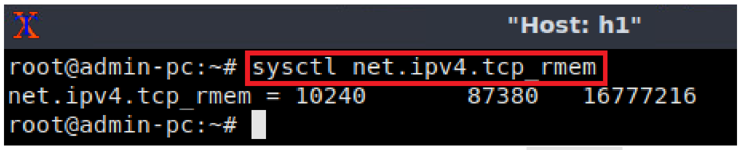
3.1 Window size in sysctl

The tool *sysctl* is used for dynamically changing parameters in the Linux operating system. It allows users to modify kernel parameters dynamically without rebuilding the Linux kernel.

The *sysctl key* for the receive window size is `net.ipv4.tcp_rmem` and the send window size is `net.ipv4.tcp_wmem`

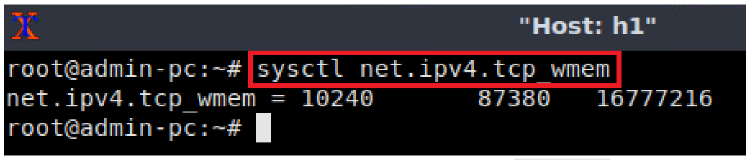
Step 1. To read the current receiver window size value of host h1, use the following command on h1's terminal:

```
sysctl net.ipv4.tcp_rmem
```


```
root@admin-pc:~# sysctl net.ipv4.tcp_rmem
net.ipv4.tcp_rmem = 10240        87380    16777216
root@admin-pc:~#
```
Figure 18. Receive window read in `sysctl`.

Step 2. To read the current send window size value of host h1, use the following command on h1's terminal:

```
sysctl net.ipv4.tcp_wmem
```


```
root@admin-pc:~# sysctl net.ipv4.tcp_wmem
net.ipv4.tcp_wmem = 10240        87380    16777216
root@admin-pc:~#
```
Figure 19. Send window read in `sysctl`.

The returned values of both keys (`net.ipv4.tcp_rmem` and `net.ipv4.tcp_wmem`) are measured in bytes. The first number represents the minimum buffer size that is used by each TCP socket. The middle one is the default buffer which is allocated when applications create a TCP socket. The last one is the maximum receive buffer that can be allocated for a TCP socket.

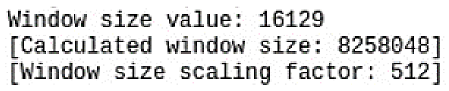
The default values used by Linux are:

- Minimum: 10,240
- Default: 87,380
- Maximum: 16,777,216

In the previous test (10 Gbps, 20ms delay), the buffer size was not modified on end-hosts. The BDP for the above test is:

$$\text{BDP} = (10 \cdot 10^9) \cdot (20 \cdot 10^{-3}) = \text{ 200,000,000 bits } = \text{25,000,000 bytes } \approx \text{25 Mbytes.}$$

Note that this value is significantly greater than the maximum buffer size (16 Mbytes), and therefore, the pipe is not getting filled, which leads to network resources underutilization. Moreover, since Linux systems by default uses half of the send/receive TCP buffers for internal kernel structures (see Section 1.3 Linux systems configuration), only half of the buffer size is used to store TCP segments. Figure 20 shows the calculated window size of a sample packet of the previous test- approximately 8 Mbytes. This is 50% of the default buffer size used by Linux (16 Mbytes).


```
Window size value: 16129
[Calculated window size: 8258048]
[Window size scaling factor: 512]
```
Figure 20. Sample window size from previous test.

Note that the observation in Figure 20 reinforces the best practice described in Section 1.3: in Linux systems, the TCP buffer size must be at least twice the BDP.

4 Modifying buffer size and throughput test

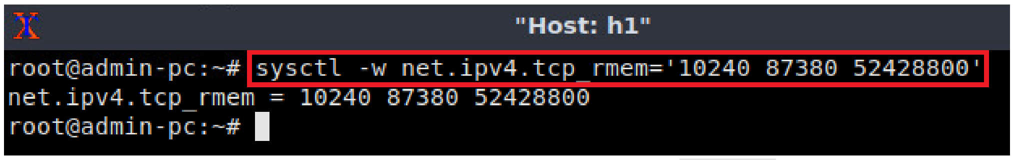
This section repeats the throughput test of Section 4 after modifying the buffer size according to the formula describe above. This test assumes the same network parameters introduced in the previous test, therefore, the bandwidth is limited to 10 Gbps, and the RTT (delay or latency) is 20ms. The send and receive buffer sizes should be set to at least 2 · BDP (if BBR is used as the congestion control algorithm, this should be set to even a larger value, as described in Section 1). We will use 25 Mbytes value for the BDP instead of 25,000,000 bytes (1 Mbyte = 1024^2 bytes).

$$\text{BDP} = 25 \text{ Mbytes} = 25 \cdot 1024^2 \text{ bytes} = 26{,}214{,}400 \text{ bytes}$$

$$\text{TCP buffer size} = 2 \cdot \text{BDP} = 2 \cdot 26{,}214{,}400 \text{ bytes} = 52{,}428{,}800 \text{ bytes}$$

Step 1. To change the TCP receive receive-window size value(s), use the following command on h1's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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



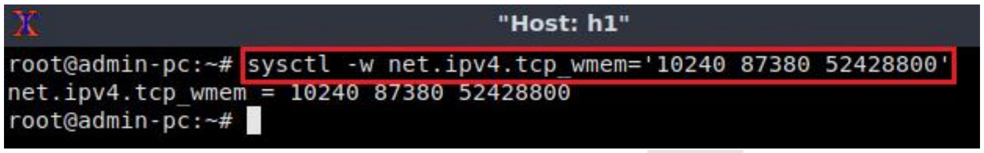
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 21. 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 h1's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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



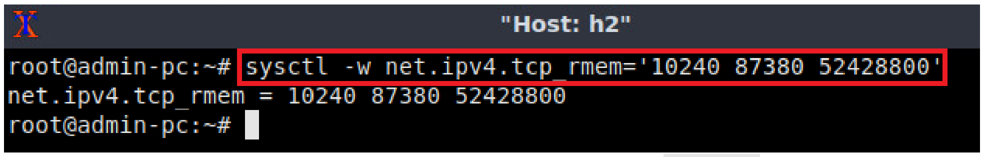
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 22. Send window change in `sysctl`.

Next, the same commands must be configured on host h2.

Step 3. To change the current receiver-window size value(s), use the following command on h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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



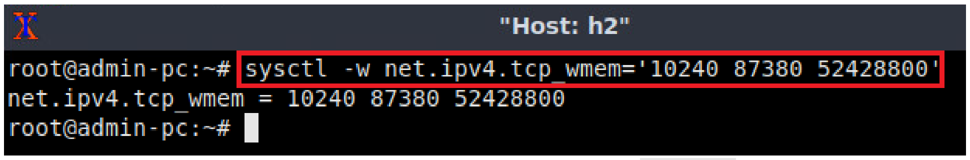
"Host: h2"

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

Figure 23. Receive window change in `sysctl`.

Step 4. To change the current send-window size value(s), use the following command on h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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



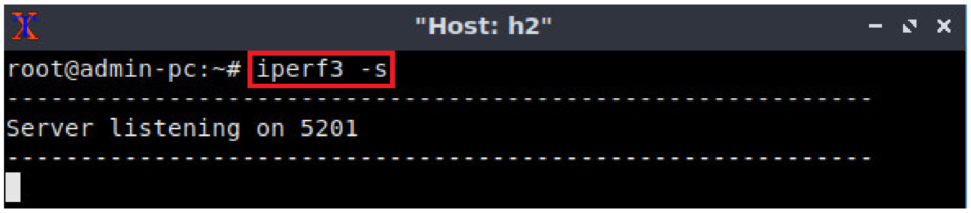
"Host: h2"

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

Figure 24. Send window change in `sysctl`.

Step 5. The user can now verify the rate limit configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in h2's terminal:

```
iperf3 -s
```



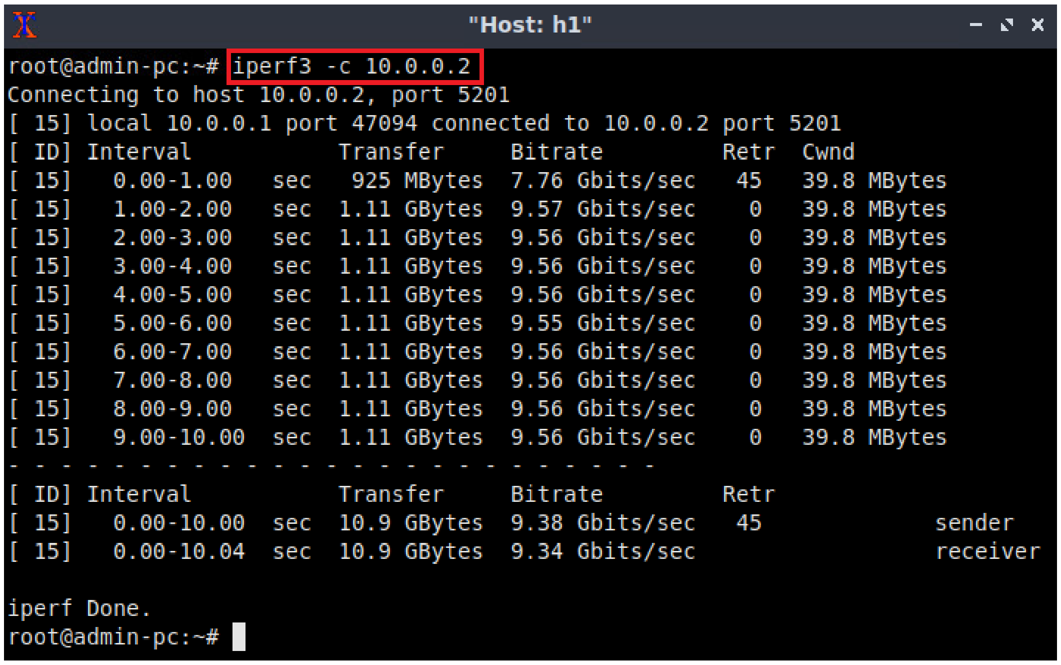
"Host: h2"

```
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 h1's terminal:

```
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 47094 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   925 MBytes  7.76 Gbits/sec   45   39.8 MBytes
[ 15]   1.00-2.00   sec  1.11 GBytes  9.57 Gbits/sec    0   39.8 MBytes
[ 15]   2.00-3.00   sec  1.11 GBytes  9.56 Gbits/sec    0   39.8 MBytes
[ 15]   3.00-4.00   sec  1.11 GBytes  9.56 Gbits/sec    0   39.8 MBytes
[ 15]   4.00-5.00   sec  1.11 GBytes  9.56 Gbits/sec    0   39.8 MBytes
[ 15]   5.00-6.00   sec  1.11 GBytes  9.55 Gbits/sec    0   39.8 MBytes
[ 15]   6.00-7.00   sec  1.11 GBytes  9.56 Gbits/sec    0   39.8 MBytes
[ 15]   7.00-8.00   sec  1.11 GBytes  9.56 Gbits/sec    0   39.8 MBytes
[ 15]   8.00-9.00   sec  1.11 GBytes  9.56 Gbits/sec    0   39.8 MBytes
[ 15]   9.00-10.00  sec  1.11 GBytes  9.56 Gbits/sec    0   39.8 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  10.9 GBytes  9.38 Gbits/sec   45             sender
[ 15]   0.00-10.04  sec  10.9 GBytes  9.34 Gbits/sec                  receiver

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

Figure 26. iPerf3 throughput test.

Note the measured throughput now is approximately 10 Gbps, which is close to the value assigned in our `tbf` rule (10 Gbps).

This concludes Lab 8. Stop the emulation and then exit out of MiniEdit and Linux terminal.

References

1. 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.
2. K. Fall, S. Floyd, “Simulation-based comparisons of tahoe, reno, and sack TCP,” Computer Communication Review, vol. 26, issue 3, Jul. 1996.
3. S. Ha, I., Rhee, L. Xu, “CUBIC: a new TCP-friendly high-speed TCP variant,” ACM SIGOPS operating systems review, vol. 42, issue 5, pp. 64-74, Jul. 2008.
4. D. Leith, R. Shorten, Y. Lee, “H-TCP: a framework for congestion control in high-speed and long-distance networks,” Hamilton Institute Technical Report, Aug. 2005. [Online]. Available: http://www.hamilton.ie/net/htcp2005.pdf
5. V. Jacobson, R. Braden, D. Borman, “TCP extensions for high performance,” Internet Request for Comments, RFC Edit, RFC 1323, May 1992. [Online]. Available: https://tools.ietf.org/rfc/rfc1323.txt

NETWORK TOOLS AND PROTOCOLS

Lab 9: Enhancing TCP Throughput with Parallel Streams

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 TCP parallel streams 3
1.1 Parallel stream fundamentals 3
1.2 Advantages of parallel streams 4
2 Lab topology........ 6
2.1 Starting host h1 and host h2 7
2.2 Emulating 10 Gbps high-latency WAN 8
2.3 Testing connection 9
3 Parallel streams to overcome TCP buffer limitation 11
4 Parallel streams to combat packet loss 12
4.1 Limit rate and add packet loss on switch S1’s s1-eth2 interface 12
4.2 Test with parallel streams 14
References 16

Overview

This lab introduces TCP parallel streams in Wide Area Networks (WANs) and explains how they are used to achieve higher throughput. Then, throughput tests using parallel streams are conducted.

Objectives

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

1. Understand TCP parallel streams.
2. Describe the advantages of TCP parallel streams.
3. Specify the number of parallel streams in an iPerf3 test.
4. Conduct tests and measure performance of parallel streams on an emulated WAN.

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 TCP parallel streams.
2. Section 2: Lab topology.
3. Section 3: Parallel streams in a high-latency high-bandwidth WAN.
4. Section 4: Parallel streams with packet loss.

1 Introduction to TCP parallel streams

1.1 Parallel stream fundamentals

Parallel Streams are multiple TCP connections opened by an application to increase performance and maximize the throughput between communicating hosts. With parallel streams, data blocks for a single file transmitted from a sender to a receiver are distributed over the multiple streams. Figure 1 shows the basic model. A control channel is established between the sender and the receiver to coordinate the data transfer. The actual transfer occurs over the parallel streams, collectively referred to as data channels. In this context, the term stream is a synonym of flow and connection.

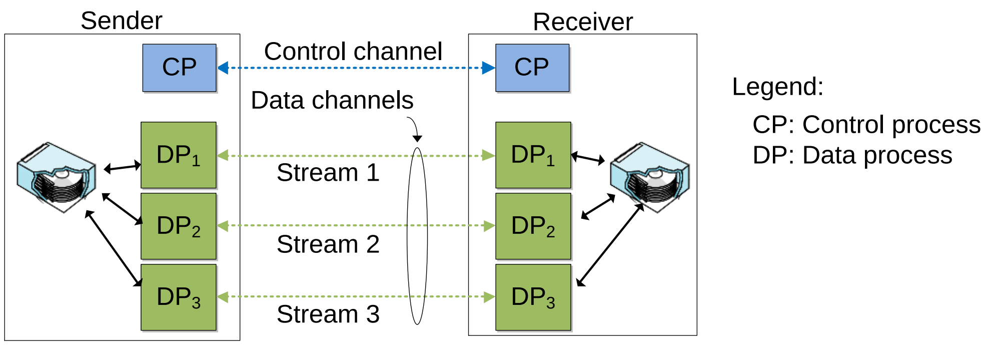


Figure 1. Data transfer model with parallel streams.

1.2 Advantages of parallel streams

Transferring large files over high-latency WANs with parallel streams have multiple benefits, as describe next.

Combat random packet loss not due congestion: assume that packet loss occurs randomly rather than due congestion. In steady state, the average throughput of a single TCP stream is given by[1]:

$$\textbf{Average throughput} \approx \frac{\textbf{MSS}}{\textbf{RTT}\,\sqrt{L}} \text{ bytes per second,}$$

where MSS is the maximum segment size and L is the packet loss rate. The above equation indicates that the throughput is directly proportional to the MSS and inversely proportional to RTT and the square root of L. When an application uses K parallel streams and if RTT, packet loss, and MSS are the same in each stream, the aggregate average throughput is the aggregation of the K single stream throughputs[2]:

$$\text{Aggregate average throughput} \approx \sum_{i=1}^{K} \frac{MSS}{RTT\sqrt{L}} = K \cdot \frac{MSS}{RTT\sqrt{L}} \text{ bytes per second.}$$

Thus, an application opening K parallel connections essentially creates a large virtual MSS on the aggregate connection that is K times the MSS of a single connection[2].

The TCP throughput follows the additive increase multiplicative decrease (AIMD) rule: TCP continuously probes for more bandwidth and increases the throughput of a connection by approximately 1 MSS per RTT as long as no packet loss occurs (additive increase phase). When a packet loss occurs, the throughput is reduced by half (multiplicative decrease event). Figure 2 illustrates the AIMD behavior for two connections with different MSSs. The MSS of the green connection is six than the MSS of the red connection. Since during the additive increase phase TCP increases the throughput by one MSS every RTT, the speed at which the throughput increases is proportional to the MSS (i.e., the larger the MSS the faster the recovery after a packet loss).

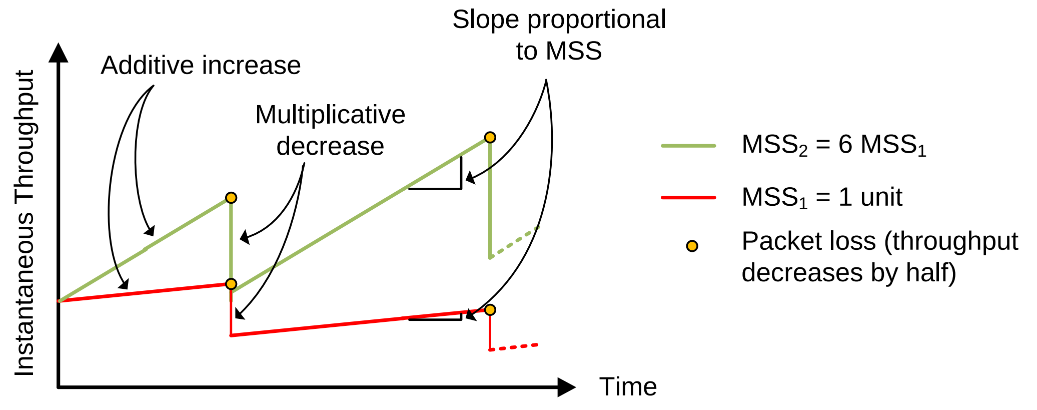


Figure 2. Additive increase multiplicative decrease (AIMD) behavior. The green curve corresponds to the throughput when the MSS is six times that of the red curve.

Mitigate TCP round-trip time (RTT) bias*:* when different flows with different RTTs share a given bottleneck link, TCP's throughput is inversely proportional to the RTT[3]. This is also noted in the equations discussed above. Hence, low-RTT flows get a higher share of the bandwidth than high-RTT flows. Thus, for transfers across high-latency WANs, one approach to combat the higher (unfair) bandwidth allocated to low-latency connections is by using parallel streams. By doing so, even if each high-latency stream receives less amount of bandwidth than low-latency flows, the aggregate throughput of the parallel streams can be high.

Overcome TCP buffer limitation: TCP receives data from the application layer and places it in the TCP buffer, as shown in Figure 3. TCP implements flow control by requiring the receiver indicate how much spare room is available in the TCP receive buffer. For a full utilization of the path, the TCP send and receive buffers must be greater than or equal to the bandwidth-delay product (BDP). This buffer size value is the maximum number of bits that can be outstanding (in-flight) if the sender continuously sends segments. If the buffer size is less than the bandwidth-delay product, then throughput will not be maximized. One solution to overcome small TCP buffer size situations is by using parallel streams. Essentially, an application opening K parallel connections creates a large buffer size on the aggregate connection that is K times the buffer size of a single connection.

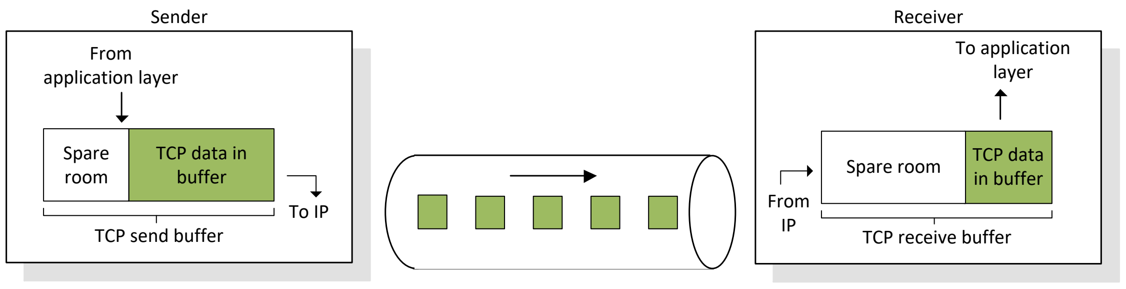


Figure 3. TCP send and receive buffers.

In this lab, we will explore the use of parallel streams to overcome TCP buffer limitation and to mitigate random packet loss.

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.

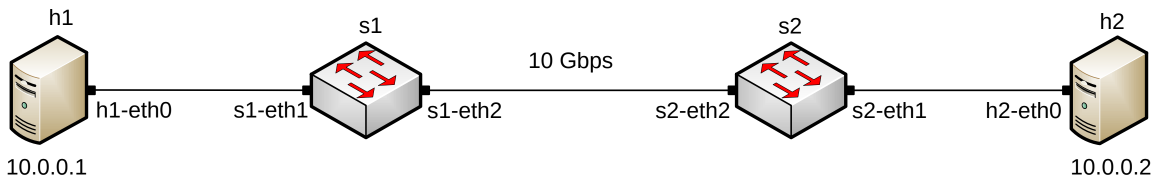


Figure 4. Lab topology.

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`.

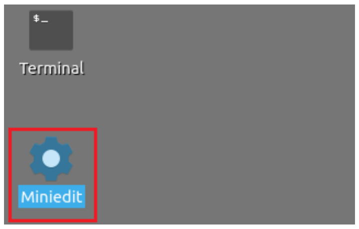


Figure 5. MiniEdit shortcut.

Step 2. On MiniEdit’s menu bar, click on *File* then *Open* to load the lab’s topology. Locate the *Lab 9.mn* topology file and click on *Open*.

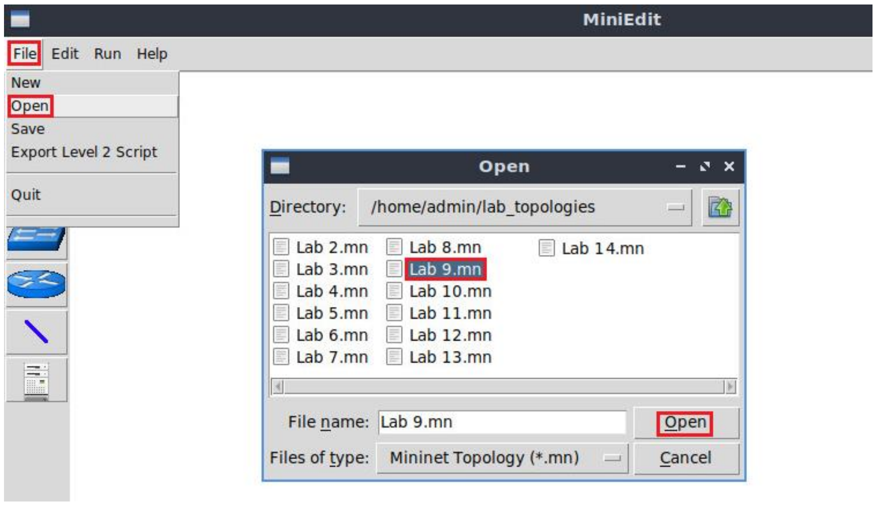


Figure 6. MiniEdit’s *Open* dialog.

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.

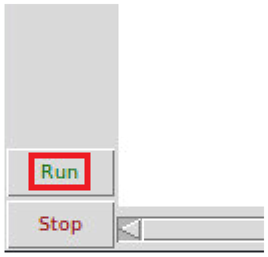


Figure 7. Running the emulation.

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 the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

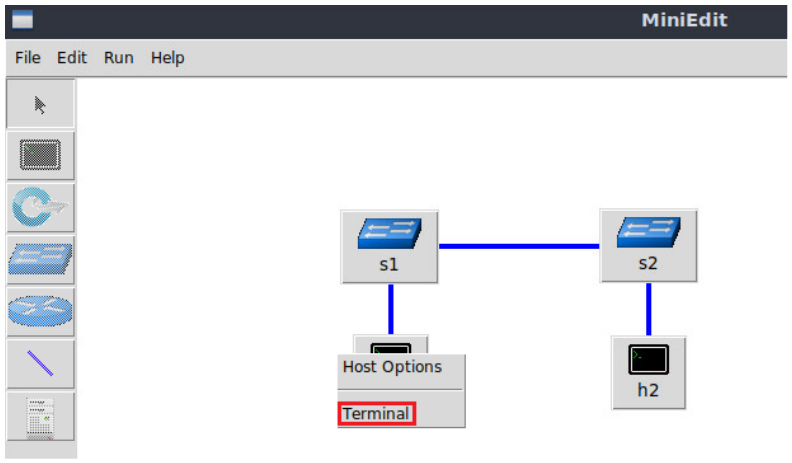
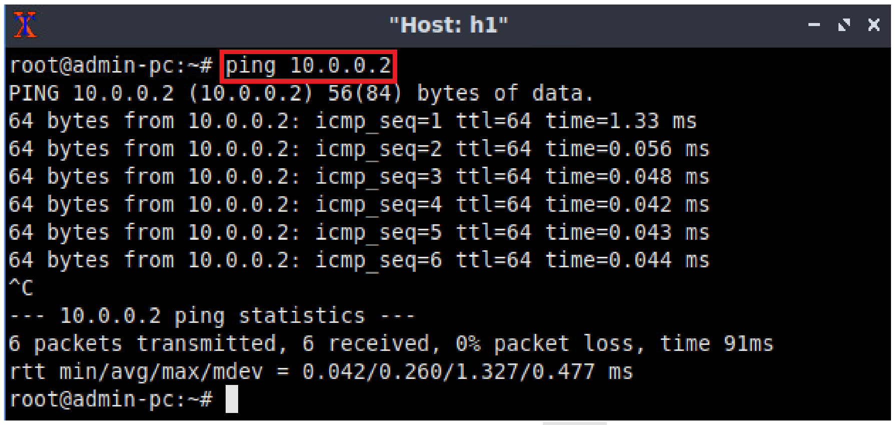


Figure 8. Opening a terminal on host h1.

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

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.



```
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 9. Connectivity test using `ping` command.

Figure 9 indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

2.2 Emulating 10 Gbps high-latency WAN

This section emulates a high-latency WAN. We will first emulate 20ms delay between switch S1 and switch S2 to measure the throughput. Then, we will set the bandwidth between host h1 and host h2 to 10 Gbps.

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

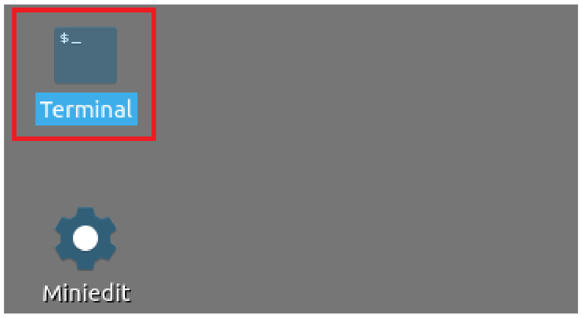
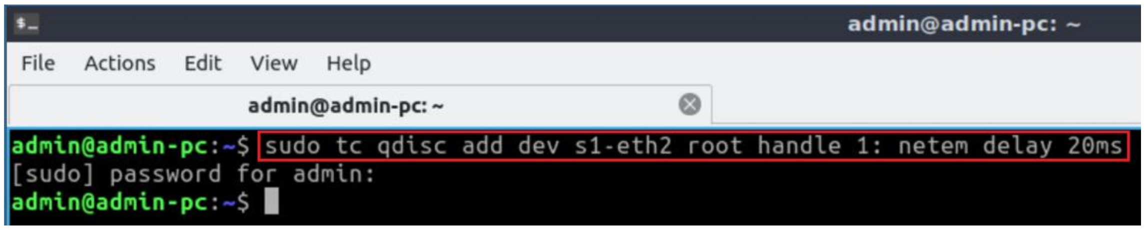


Figure 10. Shortcut to open a Linux terminal.

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. In the terminal, type the command below. When prompted for a password, type `password` and hit enter. This command introduces 20ms delay on switch S1's *s1-eth2* interface.

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



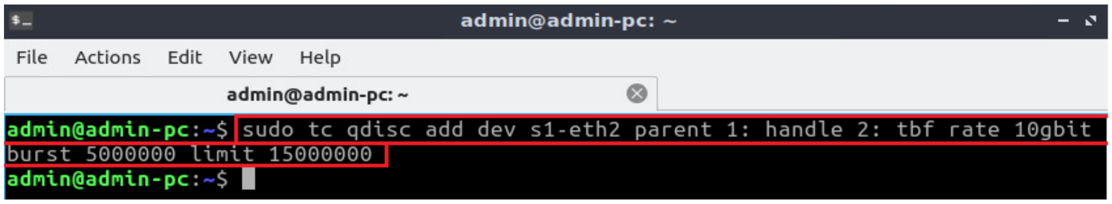
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 11. Adding delay of 20ms to switch S1's *s1-eth2* interface.

Step 3. Modify the bandwidth of the link connecting the switch S1 and switch S2: on the same terminal, type the command below. This command sets the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit burst 5000000
limit 15000000
```



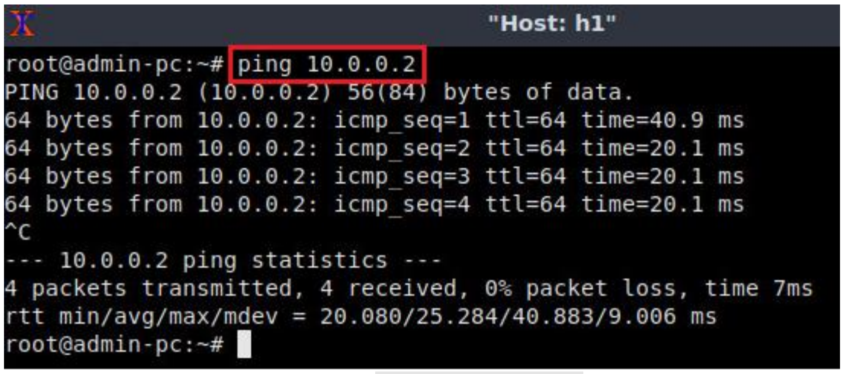
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$
```

Figure 12. Limiting the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface.

2.3 Testing connection

To test connectivity, you can use the command `ping`.

Step 1. On the terminal of host h1, type `ping 10.0.0.2`. 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.



```
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=40.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.080/25.284/40.883/9.006 ms
root@admin-pc:~#
```

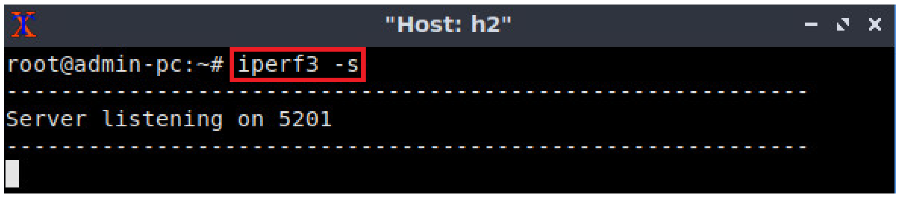
Figure 13. Output of `ping 10.0.0.2` command.

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 20.080, 25.284, 40.883, and 9.006 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 10.0.0.1`. The ping output in this test should be relatively close to the results of the test initiated by host h1 in Step 1. To stop the test, press `Ctrl+c`.

Step 3. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

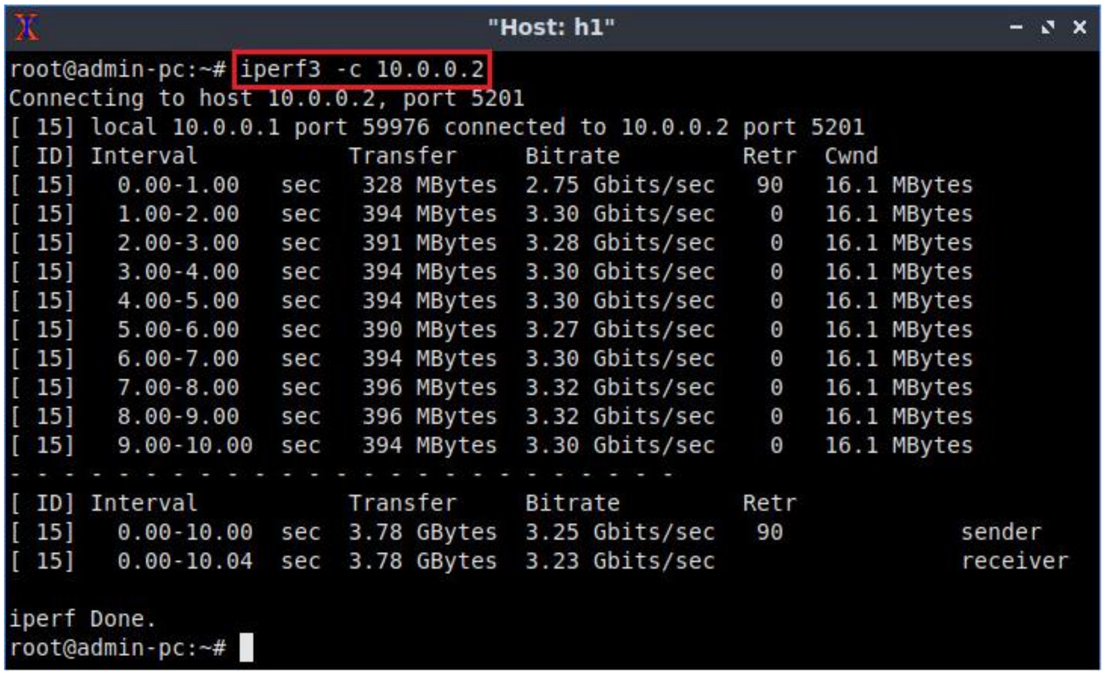


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 14. Starting iPerf3 server on host h2.

Step 4. Launch iPerf3 in client mode on host h1 's terminal. To stop the test, press `Ctrl+c`.

```
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 59976 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   328 MBytes  2.75 Gbits/sec   90   16.1 MBytes
[ 15]   1.00-2.00   sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
[ 15]   2.00-3.00   sec   391 MBytes  3.28 Gbits/sec    0   16.1 MBytes
[ 15]   3.00-4.00   sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
[ 15]   4.00-5.00   sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
[ 15]   5.00-6.00   sec   390 MBytes  3.27 Gbits/sec    0   16.1 MBytes
[ 15]   6.00-7.00   sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
[ 15]   7.00-8.00   sec   396 MBytes  3.32 Gbits/sec    0   16.1 MBytes
[ 15]   8.00-9.00   sec   396 MBytes  3.32 Gbits/sec    0   16.1 MBytes
[ 15]   9.00-10.00  sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  3.78 GBytes  3.25 Gbits/sec   90             sender
[ 15]   0.00-10.04  sec  3.78 GBytes  3.23 Gbits/sec                  receiver

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

Figure 15. Running iPerf3 client on host h1.

Although the link was configured to 10 Gbps, the test results show that the achieved throughput is 3.22 Gbps. This is because the TCP buffer size is less than the bandwidth-delay product. In the upcoming section, we run a throughput test without modifying the TCP buffer size, but with multiple parallel streams.

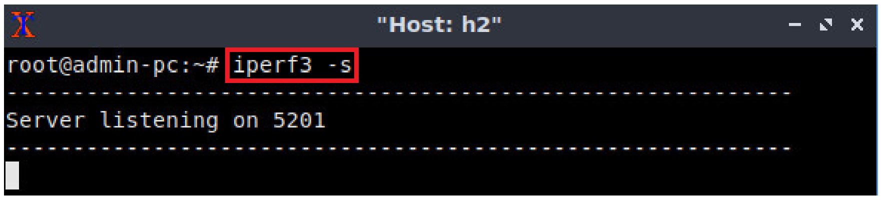
Step 5. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3 Parallel streams to overcome TCP buffer limitation

In this section, parallel streams are specified by the client when executing the throughput test in iPerf3. The iPerf3 server should start as usual, without specifying any additional options or parameters.

Step 1. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown the figure below:

```
iperf3 -s
```

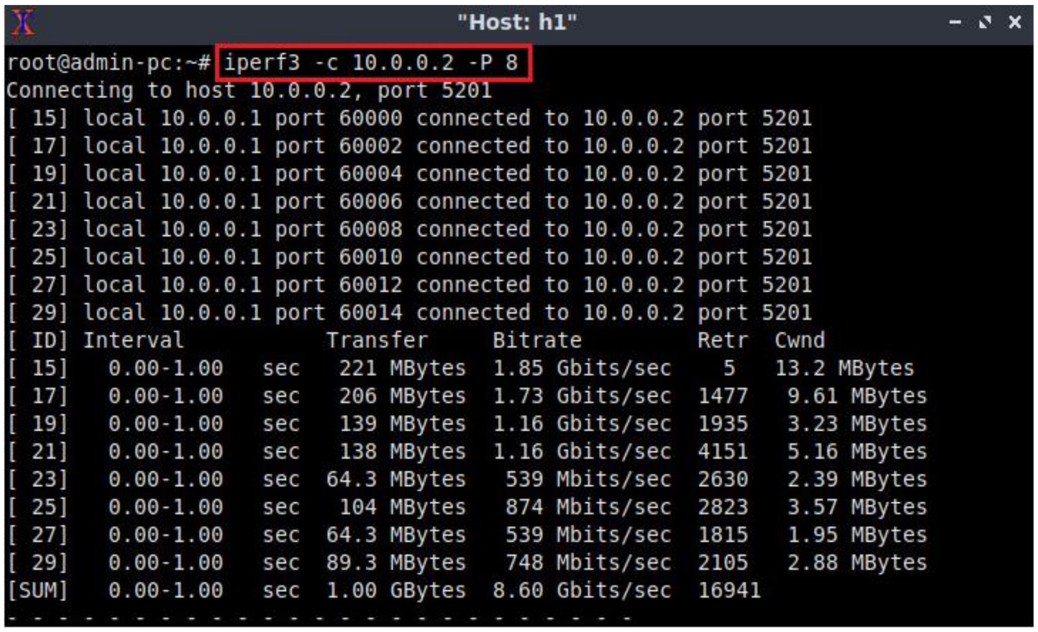


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 16. Host h2 running iPerf3 as server.

Step 2. Now the iPerf3 client should be launched with the `-P` option specified (not to be confused with the `-p` option which specifies the listening port number). This option specifies the number of parallel streams. Run the following command in host h1's terminal:

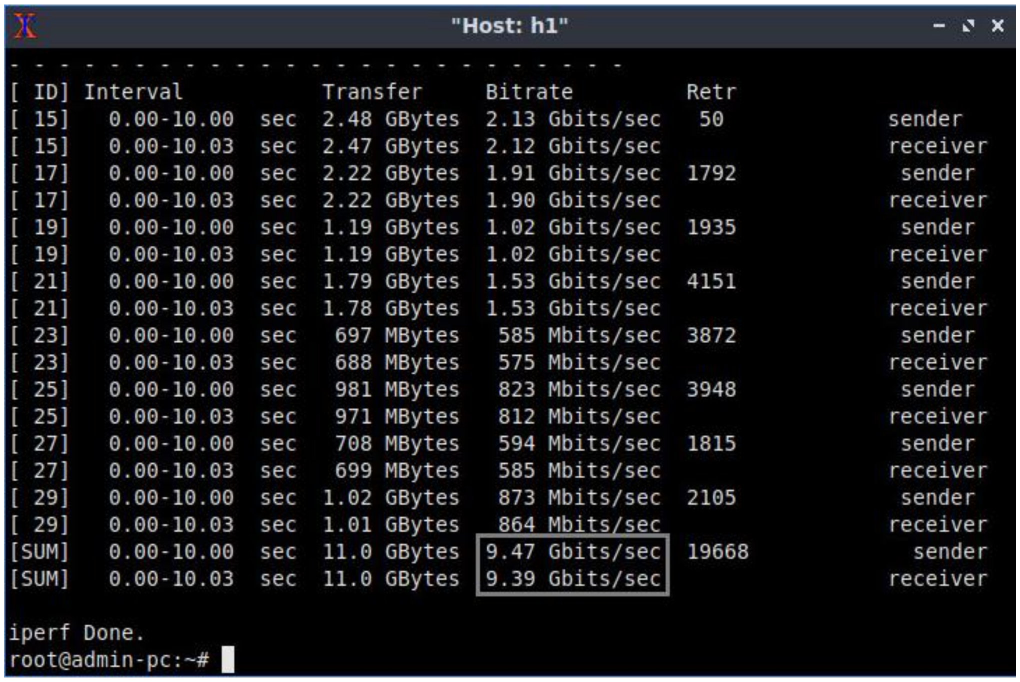
```
iperf3 -c 10.0.0.2 -P 8
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2 -P 8
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 60000 connected to 10.0.0.2 port 5201
[ 17] local 10.0.0.1 port 60002 connected to 10.0.0.2 port 5201
[ 19] local 10.0.0.1 port 60004 connected to 10.0.0.2 port 5201
[ 21] local 10.0.0.1 port 60006 connected to 10.0.0.2 port 5201
[ 23] local 10.0.0.1 port 60008 connected to 10.0.0.2 port 5201
[ 25] local 10.0.0.1 port 60010 connected to 10.0.0.2 port 5201
[ 27] local 10.0.0.1 port 60012 connected to 10.0.0.2 port 5201
[ 29] local 10.0.0.1 port 60014 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   221 MBytes  1.85 Gbits/sec    5   13.2 MBytes
[ 17]   0.00-1.00   sec   206 MBytes  1.73 Gbits/sec  1477   9.61 MBytes
[ 19]   0.00-1.00   sec   139 MBytes  1.16 Gbits/sec  1935   3.23 MBytes
[ 21]   0.00-1.00   sec   138 MBytes  1.16 Gbits/sec  4151   5.16 MBytes
[ 23]   0.00-1.00   sec  64.3 MBytes   539 Mbits/sec  2630   2.39 MBytes
[ 25]   0.00-1.00   sec   104 MBytes   874 Mbits/sec  2823   3.57 MBytes
[ 27]   0.00-1.00   sec  64.3 MBytes   539 Mbits/sec  1815   1.95 MBytes
[ 29]   0.00-1.00   sec  89.3 MBytes   748 Mbits/sec  2105   2.88 MBytes
[SUM]   0.00-1.00   sec  1.00 GBytes  8.60 Gbits/sec  16941
- - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 17. iPerf3 throughput test with parallel streams.

The above command uses 8 parallel streams. Note that 8 sockets are now opened on different local ports, and their streams are connected to the server, ready for transmitting data and performing the throughput test.


```
"Host: h1"
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  2.48 GBytes  2.13 Gbits/sec   50             sender
[ 15]   0.00-10.03  sec  2.47 GBytes  2.12 Gbits/sec                  receiver
[ 17]   0.00-10.00  sec  2.22 GBytes  1.91 Gbits/sec  1792             sender
[ 17]   0.00-10.03  sec  2.22 GBytes  1.90 Gbits/sec                  receiver
[ 19]   0.00-10.00  sec  1.19 GBytes  1.02 Gbits/sec  1935             sender
[ 19]   0.00-10.03  sec  1.19 GBytes  1.02 Gbits/sec                  receiver
[ 21]   0.00-10.00  sec  1.79 GBytes  1.53 Gbits/sec  4151             sender
[ 21]   0.00-10.03  sec  1.78 GBytes  1.53 Gbits/sec                  receiver
[ 23]   0.00-10.00  sec   697 MBytes   585 Mbits/sec  3872             sender
[ 23]   0.00-10.03  sec   688 MBytes   575 Mbits/sec                  receiver
[ 25]   0.00-10.00  sec   981 MBytes   823 Mbits/sec  3948             sender
[ 25]   0.00-10.03  sec   971 MBytes   812 Mbits/sec                  receiver
[ 27]   0.00-10.00  sec   708 MBytes   594 Mbits/sec  1815             sender
[ 27]   0.00-10.03  sec   699 MBytes   585 Mbits/sec                  receiver
[ 29]   0.00-10.00  sec  1.02 GBytes   873 Mbits/sec  2105             sender
[ 29]   0.00-10.03  sec  1.01 GBytes   864 Mbits/sec                  receiver
[SUM]   0.00-10.00  sec  11.0 GBytes  9.47 Gbits/sec  19668             sender
[SUM]   0.00-10.03  sec  11.0 GBytes  9.39 Gbits/sec                  receiver

iperf Done.
root@admin-pc:~#
```
Figure 18. iPerf3 throughput test with parallel streams summary output.

Note the measured throughput now is approximately 9.5 Gbps, which is close to the value assigned in the `tbf` rule (10 Gbps).

Step 3. 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 Parallel streams to combat packet loss

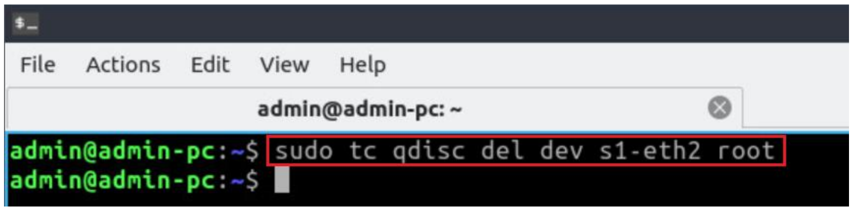
Packet loss is inevitable in real-world networks. This section explores the use of parallel streams to mitigate packet loss not due congestion (i.e., random packet loss), and compares the performance of single and parallel streams.

4.1 Limit rate and add packet loss on switch S1's s1-eth2 interface

In this topology, rate limiting is applied on switch S1's interface which connects it to switch S2 (*s1-eth2*) and 1% packet loss is introduced.

Step 1. Before applying any additional configuration, the previous rules assigned on the switch's interface must be deleted. To remove these, type the following command on the Client's terminal. When prompted for a password, type `password` and hit enter.

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

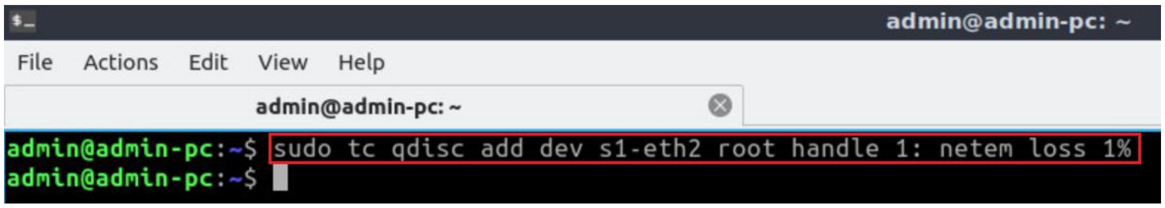


```
admin@admin-pc:~$ sudo tc qdisc del dev s1-eth2 root
admin@admin-pc:~$
```

Figure 19. Deleting previous rules on switch S1's *s1-eth2* interface.

Step 2. On the same terminal, type the below command to add 1% packet loss.

```
sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 1%
```



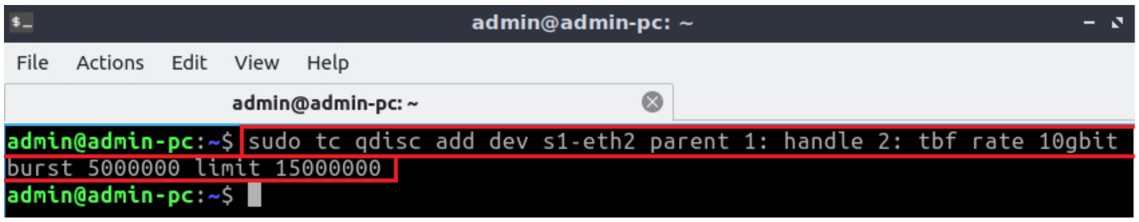
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 1%
admin@admin-pc:~$
```

Figure 20. Adding 1% packet loss to switch S1's *s1-eth2* interface.

Step 3. Modify the bandwidth of the link connecting the switch S1 and switch S2: on the same terminal, type the command below. This command sets the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit burst 5000000
limit 15000000
```

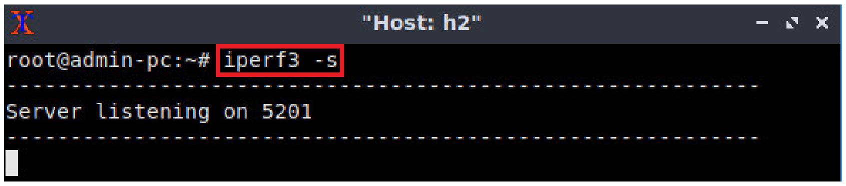


```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$
```

Figure 21. Limiting the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface.

Step 3. The user can now verify the rate limit 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 the figure below:

```
iperf3 -s
```

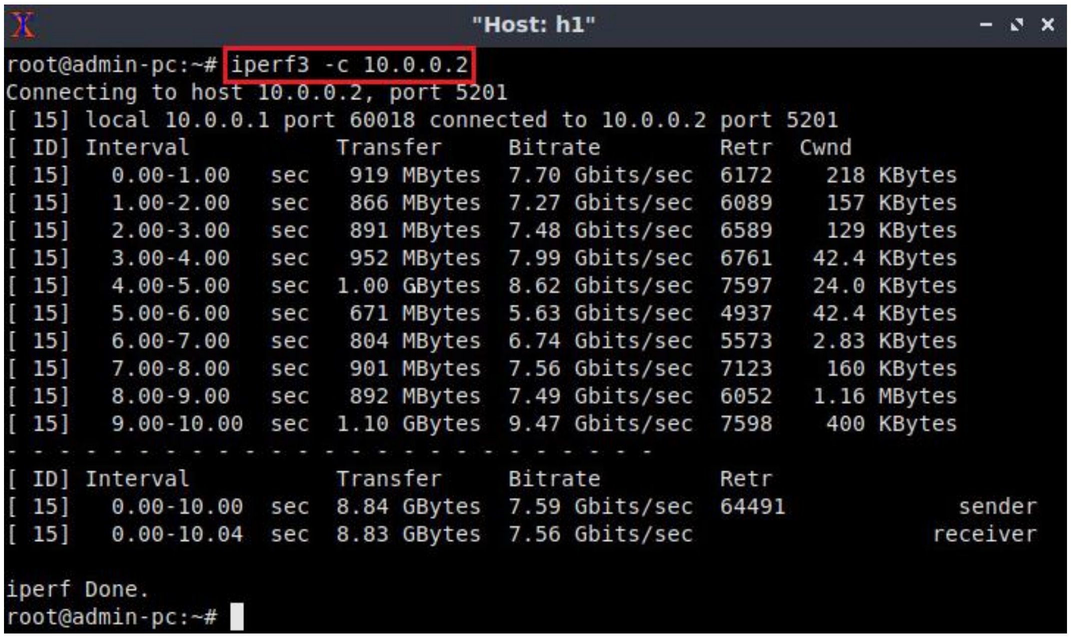


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 22. Starting iPerf3 server on host h2.

Step 4. Launch iPerf3 in client mode on host h1 's terminal. To stop the test, press `Ctrl+c`.

```
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 60018 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   919 MBytes  7.70 Gbits/sec  6172    218 KBytes
[ 15]   1.00-2.00   sec   866 MBytes  7.27 Gbits/sec  6089    157 KBytes
[ 15]   2.00-3.00   sec   891 MBytes  7.48 Gbits/sec  6589    129 KBytes
[ 15]   3.00-4.00   sec   952 MBytes  7.99 Gbits/sec  6761   42.4 KBytes
[ 15]   4.00-5.00   sec  1.00 GBytes  8.62 Gbits/sec  7597   24.0 KBytes
[ 15]   5.00-6.00   sec   671 MBytes  5.63 Gbits/sec  4937   42.4 KBytes
[ 15]   6.00-7.00   sec   804 MBytes  6.74 Gbits/sec  5573   2.83 KBytes
[ 15]   7.00-8.00   sec   901 MBytes  7.56 Gbits/sec  7123    160 KBytes
[ 15]   8.00-9.00   sec   892 MBytes  7.49 Gbits/sec  6052   1.16 MBytes
[ 15]   9.00-10.00  sec  1.10 GBytes  9.47 Gbits/sec  7598    400 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  8.84 GBytes  7.59 Gbits/sec  64491             sender
[ 15]   0.00-10.04  sec  8.83 GBytes  7.56 Gbits/sec                  receiver

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

Figure 23. Running iPerf3 client on host h1.

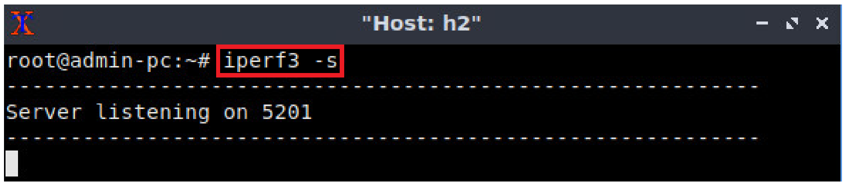
Note the measured throughput now is approximately 7.6 Gbps, which is different than the value assigned in the `tbf` rule (10 Gbps).

Step 5. 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.

4.2 Test with parallel streams

Step 1. Now the test is repeated while using parallel streams. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in Figure 24:

```
iperf3 -s
```

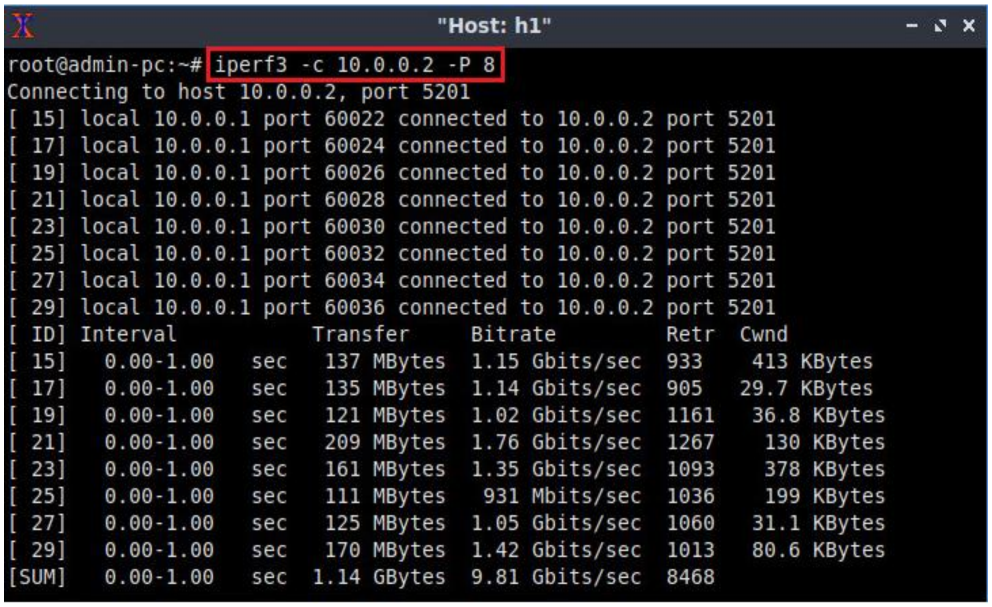


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 24. Host h2 running iPerf3 as server.

Step 2. Now the iPerf3 client should be launched with the `-P` option specified (not to be confused with the `-p` option which specifies the listening port number). This option specifies the number of parallel streams. Run the following command in host h1's terminal:

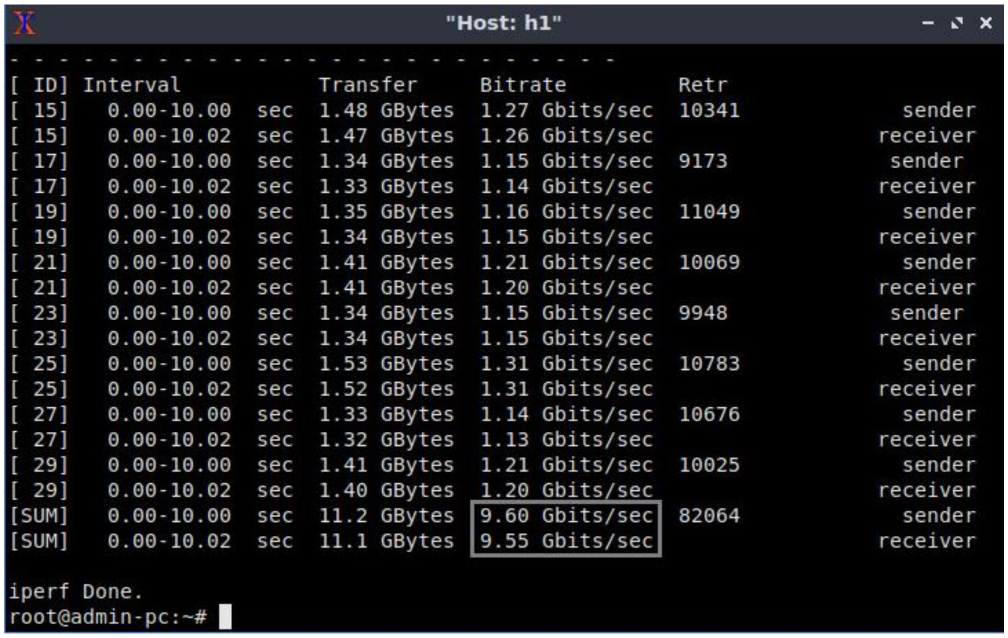
```
iperf3 -c 10.0.0.2 -P 8
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2 -P 8
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 60022 connected to 10.0.0.2 port 5201
[ 17] local 10.0.0.1 port 60024 connected to 10.0.0.2 port 5201
[ 19] local 10.0.0.1 port 60026 connected to 10.0.0.2 port 5201
[ 21] local 10.0.0.1 port 60028 connected to 10.0.0.2 port 5201
[ 23] local 10.0.0.1 port 60030 connected to 10.0.0.2 port 5201
[ 25] local 10.0.0.1 port 60032 connected to 10.0.0.2 port 5201
[ 27] local 10.0.0.1 port 60034 connected to 10.0.0.2 port 5201
[ 29] local 10.0.0.1 port 60036 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   137 MBytes  1.15 Gbits/sec  933    413 KBytes
[ 17]   0.00-1.00   sec   135 MBytes  1.14 Gbits/sec  905   29.7 KBytes
[ 19]   0.00-1.00   sec   121 MBytes  1.02 Gbits/sec  1161   36.8 KBytes
[ 21]   0.00-1.00   sec   209 MBytes  1.76 Gbits/sec  1267    130 KBytes
[ 23]   0.00-1.00   sec   161 MBytes  1.35 Gbits/sec  1093    378 KBytes
[ 25]   0.00-1.00   sec   111 MBytes   931 Mbits/sec  1036    199 KBytes
[ 27]   0.00-1.00   sec   125 MBytes  1.05 Gbits/sec  1060   31.1 KBytes
[ 29]   0.00-1.00   sec   170 MBytes  1.42 Gbits/sec  1013   80.6 KBytes
[SUM]   0.00-1.00   sec  1.14 GBytes  9.81 Gbits/sec  8468
```

Figure 25. Host h1 running iPerf3 as client with 8 parallel streams.

The above command uses 8 parallel streams. Note that 8 sockets are now opened on different local ports, and their streams are connected to the server, ready for transmitting data and performing the throughput test.



```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  1.48 GBytes  1.27 Gbits/sec  10341             sender
[ 15]   0.00-10.02  sec  1.47 GBytes  1.26 Gbits/sec                  receiver
[ 17]   0.00-10.00  sec  1.34 GBytes  1.15 Gbits/sec  9173             sender
[ 17]   0.00-10.02  sec  1.33 GBytes  1.14 Gbits/sec                  receiver
[ 19]   0.00-10.00  sec  1.35 GBytes  1.16 Gbits/sec  11049             sender
[ 19]   0.00-10.02  sec  1.34 GBytes  1.15 Gbits/sec                  receiver
[ 21]   0.00-10.00  sec  1.41 GBytes  1.21 Gbits/sec  10069             sender
[ 21]   0.00-10.02  sec  1.41 GBytes  1.20 Gbits/sec                  receiver
[ 23]   0.00-10.00  sec  1.34 GBytes  1.15 Gbits/sec  9948             sender
[ 23]   0.00-10.02  sec  1.34 GBytes  1.15 Gbits/sec                  receiver
[ 25]   0.00-10.00  sec  1.53 GBytes  1.31 Gbits/sec  10783             sender
[ 25]   0.00-10.02  sec  1.52 GBytes  1.31 Gbits/sec                  receiver
[ 27]   0.00-10.00  sec  1.33 GBytes  1.14 Gbits/sec  10676             sender
[ 27]   0.00-10.02  sec  1.32 GBytes  1.13 Gbits/sec                  receiver
[ 29]   0.00-10.00  sec  1.41 GBytes  1.21 Gbits/sec  10025             sender
[ 29]   0.00-10.02  sec  1.40 GBytes  1.20 Gbits/sec                  receiver
[SUM]   0.00-10.00  sec  11.2 GBytes  9.60 Gbits/sec  82064             sender
[SUM]   0.00-10.02  sec  11.1 GBytes  9.55 Gbits/sec                  receiver

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

Figure 26. iPerf3 throughput test with parallel streams summary output.

Note the measured throughput now is approximately 9.6 Gbps, which is close to the value assigned in our `tbf` rule (10 Gbps). In conclusion, parallel streams are beneficial when the packet loss rate is high. As shown in the previous test, when using parallel streams, the host was able to achieve the maximum theoretical bandwidth.

This concludes Lab 9. Stop the emulation and then exit out of MiniEdit.

References

1. M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," ACM Computer Communication Review, vol. 27, no 3, pp. 67-82, Jul. 1997.
2. T. Hacker, B. Athey, B. Noble, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," in Proceedings of the Parallel and Distributed Processing Symposium, Apr. 2001.
3. J. Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, pp. 303-314, Sep. 1998.

NETWORK TOOLS AND PROTOCOLS

Lab 10: Measuring TCP Fairness

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 Fairness concepts 3
  1.1 TCP bandwidth allocation 3
  1.2 TCP fairness index calculation 5
2 Lab topology 6
  2.1 Starting host h1 and host h2 8
  2.2 Emulating 10 Gbps high-latency WAN 8
  2.3 Testing connection 10
3 Calculating fairness among parallel flows 13
4 Calculating fairness among several hosts with the same congestion control algorithm 14
5 Calculating fairness among hosts with different congestion control algorithms..... 17
References 19

Overview

This lab introduces TCP fairness in Wide Area Networks (WAN) and explains how competing TCP connections converge to fairness. The lab describes how to calculate the TCP fairness index, a metric that quantifies how fair the aggregate connection is divided between active connections. Finally, the lab conducts throughput tests in an emulated high-latency network and derives the fairness index.

Objectives

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

1. Define TCP fairness.
2. Calculate TCP fairness index.
3. Emulate a WAN and calculating fairness index among parallel streams.
4. Emulate a WAN and calculating fairness index among competing TCP connections.

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: Fairness concepts.
2. Section 2: Lab topology.
3. Section 3: Calculating fairness among parallel flows.
4. Section 4: Calculating fairness index with different congestion control algorithms.

1 Fairness concepts

1.1 TCP bandwidth allocation

Many networks do not use any bandwidth reservation mechanism for TCP flows passing through a router. Instead, routers simply make forwarding decisions based on the destination field of the IP header. As a result, flows may attempt to use as much bandwidth as possible. In this situation, it is the TCP congestion control algorithm that allocates bandwidth to the competing flows.

Consider the scenario where two TCP flows share a bottleneck link with bandwidth capacity R, as illustrated in Figure 1. Assume that the two senders are in equal conditions (round-trip time, maximum segment size, configuration parameters) and that they use the same congestion control algorithm. Furthermore, assume that the two flows are in steady state and that the congestion control algorithm operates according to the additive increase multiplicative decrease (AIMD) rule[1]. A fair bandwidth allocation would result in a bandwidth partition of R/2 to each flow.

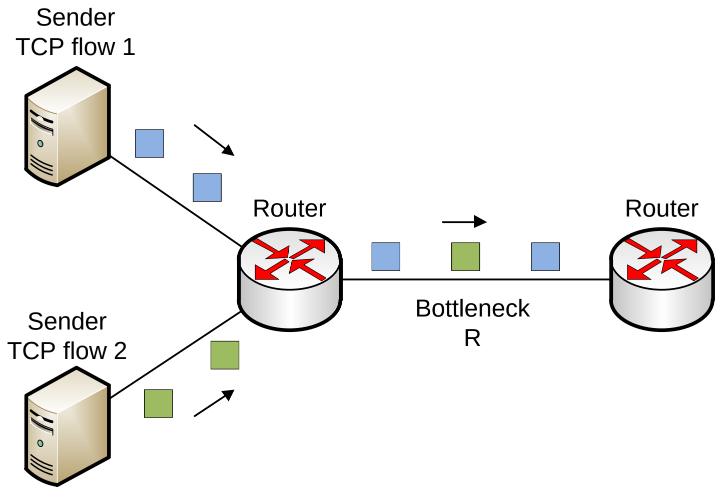


Figure 1. Two TCP flows that share a bottleneck link of capacity R.

Figure 2 shows the bandwidth allocation region for the two flows[1]. The bandwidth allocation to flow 1 is on x-axis and to flow 2 is on the y-axis. If TCP is to share the bottleneck bandwidth equally between the two flows, then the bandwidth will fall along the fairness line emanating from the origin. Note that the origin (0, 0) is a fair but undesirable solution. When the allocations sum to 100% of the bottleneck capacity, the allocation is efficient. This is shown by the efficiency line. Note that potential *efficient* solutions include points A (R, 0) and points B (0, R). On point A, flow 1 receives 100% of the capacity, and on point B flow 2 receives 100% of the capacity. Clearly, these solutions are not desirable, as they lead to starvation and unfairness.

Assume that the sending rates of senders 1 and 2 at a given time are indicated by point p_1. As the amount of aggregate bandwidth jointly consumed by the two flows is less than R, no loss will occur, and TCP will gently increase the bandwidth allocation (this process is called additive increase phase). Eventually, the bandwidth jointly consumed by the two connections will be greater than R, and a packet loss will occur at a point, say p_2. TCP reacts to a packet loss by aggressively decreasing the sending rate by half (this operation is called multiplicate decrease). The resulting bandwidth allocations are realized at point p_3. Since the joint bandwidth use is less than R at point p_3, TCP will again increase the allocation to flows 1 and 2. Eventually, the TCP additive increase phase will lead to the operating point p4, where a loss will again occur, and the two flows again will see a decrease in the bandwidth allocation, and so on. The bandwidth realized by the two flows eventually will fluctuate along the fairness line, near the optimal operating point Opt (R/2, R/2). Chiu and Jain[1] describe the reasons of why TCP converges to a fair and efficient allocation. This convergence occurs independently of the starting point[2, 3].

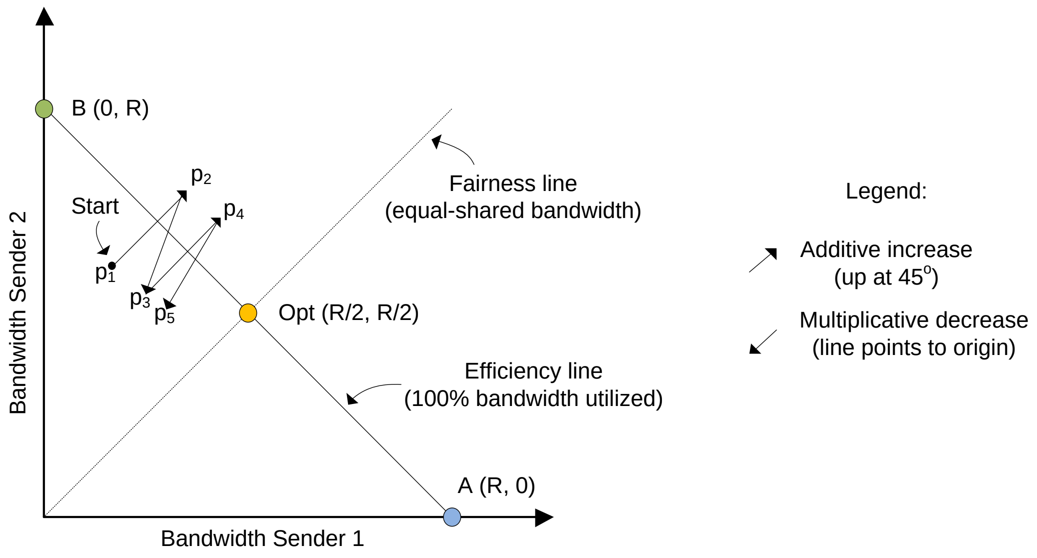


Figure 2. Bandwidth allocation region realized by two competing TCP flows.

1.2 TCP fairness index calculation

A useful index to quantify fairness is Jain's index[4]. The index has the following properties:

1. *Population size independence:* the index is applicable to any number of flows.
2. *Scale and metric independence:* the index is independent of scale, i.e., the unit of measurement does not matter.
3. *Boundedness:* the index is bounded between 0 and 1. A totally fair system has an index of 1 and a totally unfair system has an index of 0.
4. *Continuity:* the index is continuous. Any change in allocation is reflected in the fairness index.

Jain's fairness index is given by the following equation:

$$I = \frac{(\sum_{i=1}^{n} T_i)^2}{n \sum_{i=1}^{n} {T_i}^2}$$

where

- I is the fairness index, with values between 0 and 1.
- n is the total number of flows.
- $T_1, T_2, \ldots, T_n$ are the measured throughput of individual flows.

As an example of fairness index calculation, consider the three flows shown in Figure 3. Given the bottleneck capacity of 9 Gbps, assume that the bandwidth allocations for flows 1, 2, and 3 are 5 Gbps, 3 Gbps, and 1 Gbps. The fairness index for this allocation is:

$$I = \frac{(\sum_{i=1}^{3} T_i)^2}{3\ \sum_{i=1}^{3} {T_i}^2} = \frac{(5 \cdot 10^9 + 3 \cdot 10^9 +\ 1 \cdot 10^9)^2}{3\ \cdot ((5 \cdot 10^9)^2 +\ (3 \cdot 10^9)^2 +\ (1 \cdot 10^9)^2)} = 0.77$$

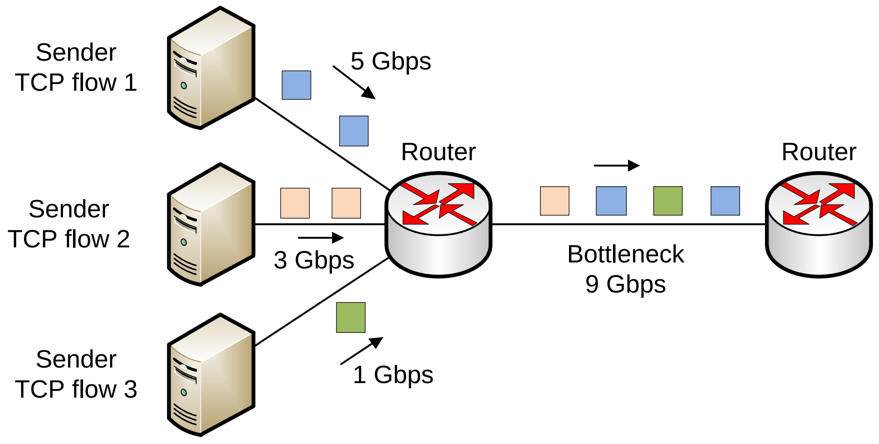


Figure 3. Three TCP flows that share a bottleneck link of capacity 9 Gbps.

Note that by property 2 (scale and metric independence), the fairness index of the above example is the same as that of an allocation of 5 Mbps, 3 Mbps, and 1 Mbps (or more generally, an allocation of 5, 3, and 1 units). Also, note that an optimal fair allocation of 3 Gbps to each flow would produce a fairness index of 1.

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.

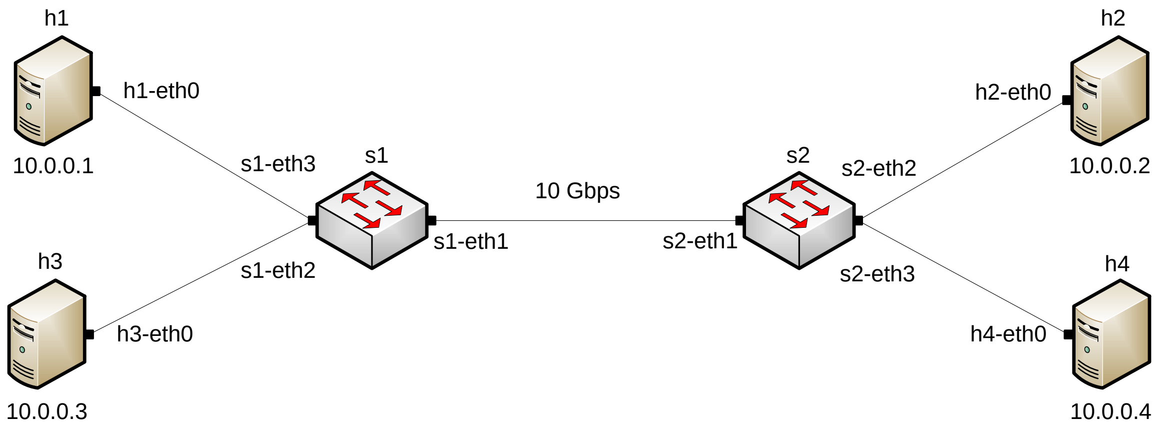


Figure 4. Lab topology

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`.

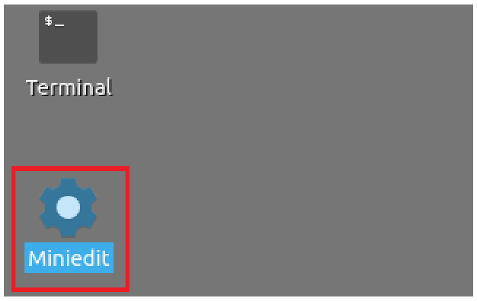


Figure 5. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 10.mn* topology file and click on *Open*.

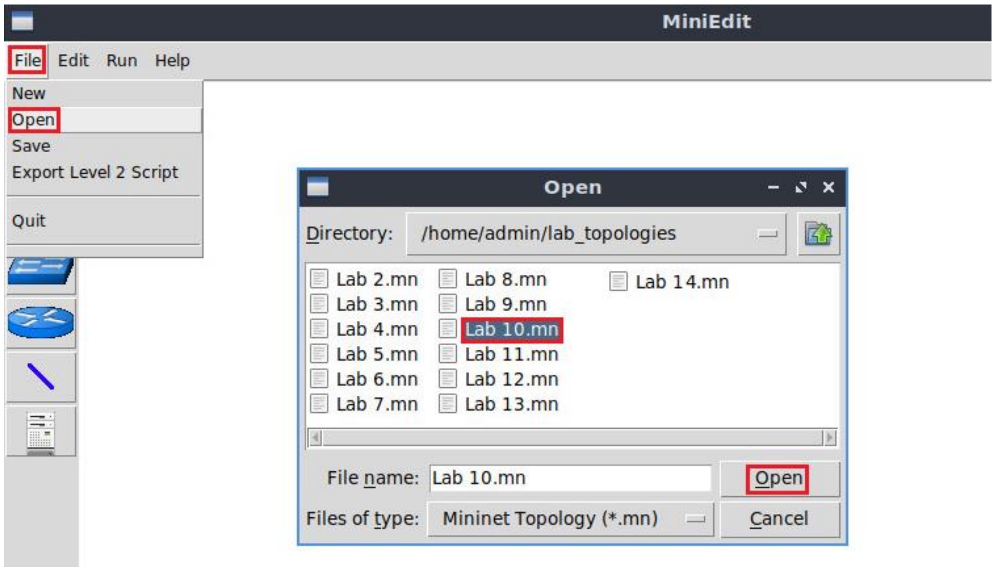


Figure 6. MiniEdit's *Open* dialog.

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.

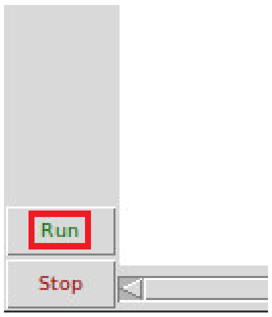


Figure 7. Running the emulation.

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 the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

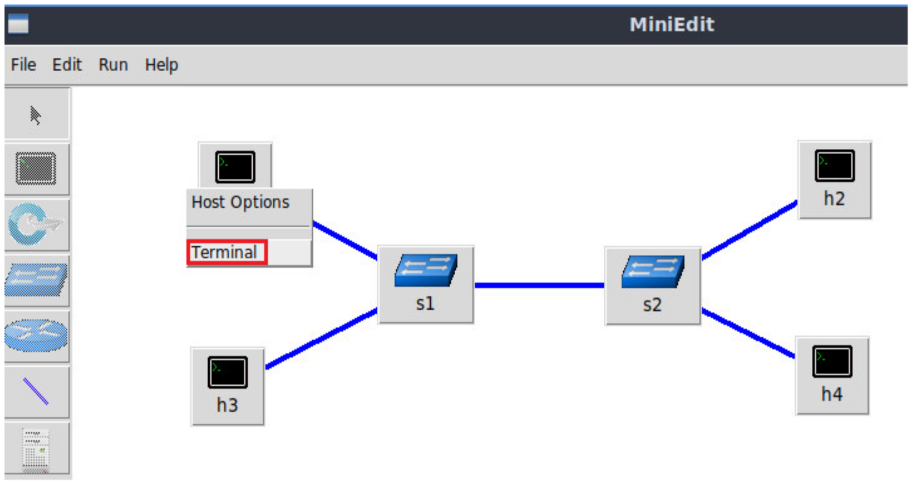
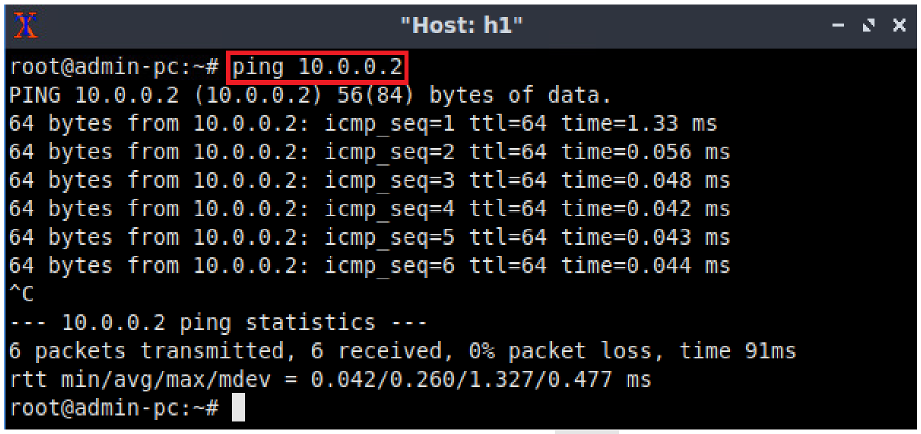


Figure 8. Opening a terminal on host h1.

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.



```
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 9. Connectivity test using `ping` command.

Figure 9 indicates that there is connectivity between host h1 and host h2. Thus, we are ready to start the throughput measurement process.

2.2 Emulating 10 Gbps high-latency WAN

This section emulates a high-latency WAN. We will first emulate 20ms delay between switch S1 and switch S2 and measure the throughput. Then, we will set the bandwidth between host h1 and host h2 to 10 Gbps.

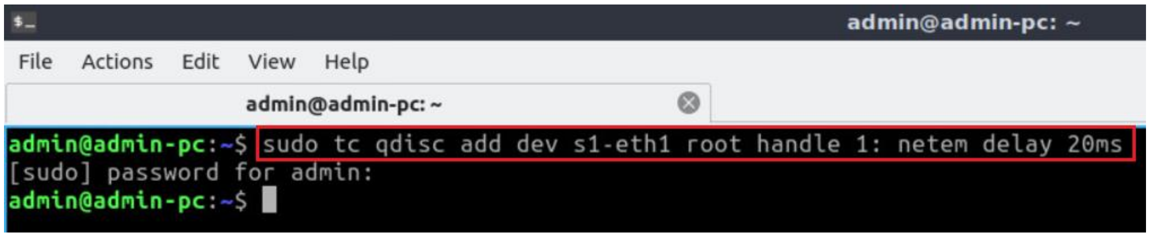
Step 1. Launch a Linux terminal by holding the `Ctrl+Alt+T` keys or by clicking on the Linux terminal icon.

Figure 10. 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. In the terminal, type the command below. When prompted for a password, type `password` and hit *Enter*. This command introduces 20ms delay on switch S1's *s1-eth1* interface.

```
sudo tc qdisc add dev s1-eth1 root handle 1: netem delay 20ms
```



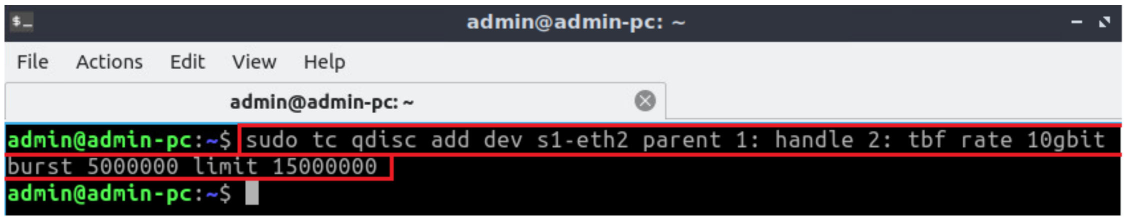
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth1 root handle 1: netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$ 
```

Figure 11. Adding delay of 20ms to switch S1's *s1-eth1* interface.

Step 3. Modify the bandwidth of the link connecting the switch S1 and switch S2: on the same terminal, type the command below. This command sets the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth1 parent 1: handle 2: tbf rate 10gbit burst 5000000
limit 15000000
```



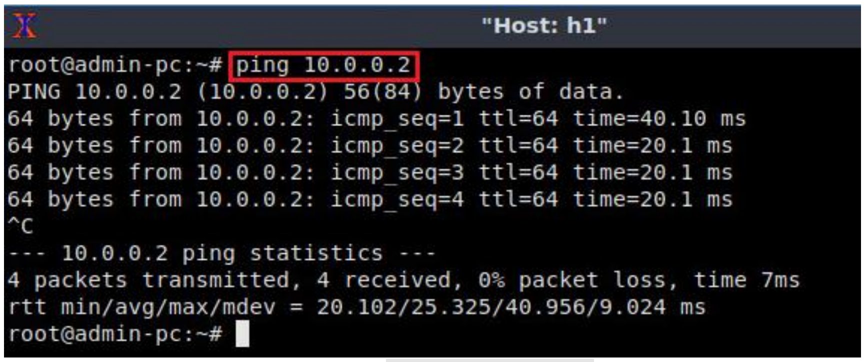
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$ 
```

Figure 12. Limiting the bandwidth to 10 Gbps on switch S1's *s1-eth1* interface.

2.3 Testing connection

To test connectivity, you can use the command `ping`.

Step 1. On the terminal of host h1, type `ping 10.0.0.2`. 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.



```
"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=40.10 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.102/25.325/40.956/9.024 ms
root@admin-pc:~#
```

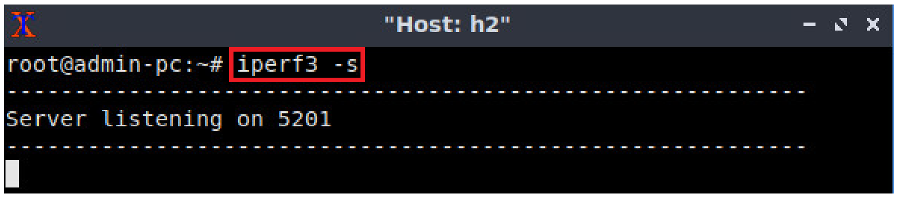
Figure 13. Output of `ping 10.0.0.2` command.

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 20.102, 25.325, 40.956, and 9.024 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 10.0.0.1`. The ping output in this test should be relatively close to the results of the test initiated by host h1 in Step 1. To stop the test, press `Ctrl+c`.

Step 3. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

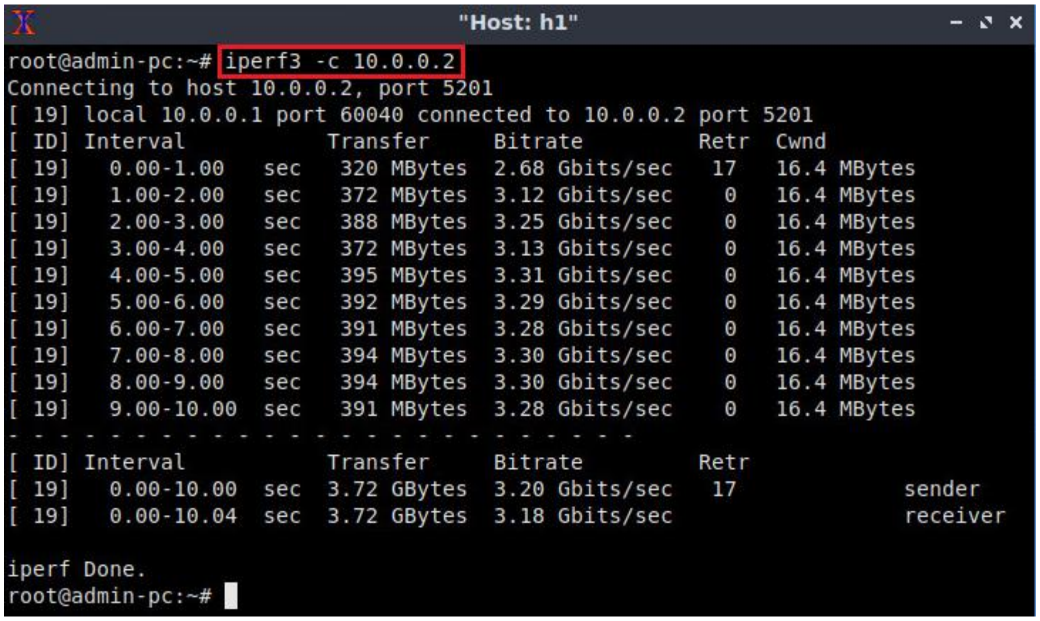


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 14. Starting iPerf3 server on host h2.

Step 4. Launch iPerf3 in client mode on host h1 's terminal.

```
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
[ 19] local 10.0.0.1 port 60040 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 19]   0.00-1.00   sec   320 MBytes  2.68 Gbits/sec   17   16.4 MBytes
[ 19]   1.00-2.00   sec   372 MBytes  3.12 Gbits/sec    0   16.4 MBytes
[ 19]   2.00-3.00   sec   388 MBytes  3.25 Gbits/sec    0   16.4 MBytes
[ 19]   3.00-4.00   sec   372 MBytes  3.13 Gbits/sec    0   16.4 MBytes
[ 19]   4.00-5.00   sec   395 MBytes  3.31 Gbits/sec    0   16.4 MBytes
[ 19]   5.00-6.00   sec   392 MBytes  3.29 Gbits/sec    0   16.4 MBytes
[ 19]   6.00-7.00   sec   391 MBytes  3.28 Gbits/sec    0   16.4 MBytes
[ 19]   7.00-8.00   sec   394 MBytes  3.30 Gbits/sec    0   16.4 MBytes
[ 19]   8.00-9.00   sec   394 MBytes  3.30 Gbits/sec    0   16.4 MBytes
[ 19]   9.00-10.00  sec   391 MBytes  3.28 Gbits/sec    0   16.4 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-10.00  sec  3.72 GBytes  3.20 Gbits/sec   17             sender
[ 19]   0.00-10.04  sec  3.72 GBytes  3.18 Gbits/sec                  receiver

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

Figure 15. Running iPerf3 client on host h1.

Although the link was configured to 10 Gbps, the test results show that the achieved throughput is 3.20 Gbps. This is because the TCP buffer size was not modified at this point.

Step 5. 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.

Step 6. To change the current receive-window size value(s), we calculate the Bandwidth-Delay Product by performing the following calculation:

$$BW = 10{,}000{,}000{,}000 \; bits/second$$

$$RTT = 0.02 \; seconds$$

$$BDP = 10{,}000{,}000{,}000 * 0.02 = 200{,}000{,}000 \; bits$$
$$= 25{,}000{,}000 \; bytes \approx 25 \; Mbytes$$

The send and receive buffer sizes should be set to 2 · BDP. We will use the 25 Mbytes value for the BDP instead of 25,000,000 bytes.

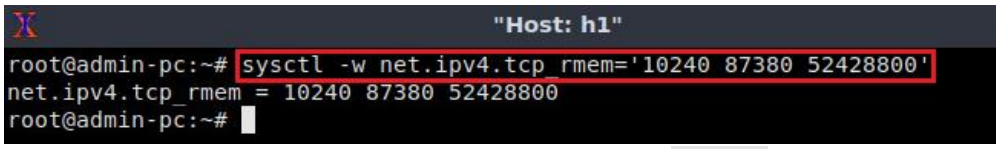
$$1 \; Mbyte = 1024^2 \; bytes$$

$$BDP = 25 \; Mbytes = 25 * 1024^2 \; bytes = 26{,}214{,}400 \; bytes$$

$$2 \cdot BDP = 2 \cdot 26{,}214{,}400 \; bytes = 52{,}428{,}800 \; bytes$$

Now, we have calculated the maximum value of the TCP sending and receiving buffer size. In order to apply the new values, on host h1's terminal type the command showed down below. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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

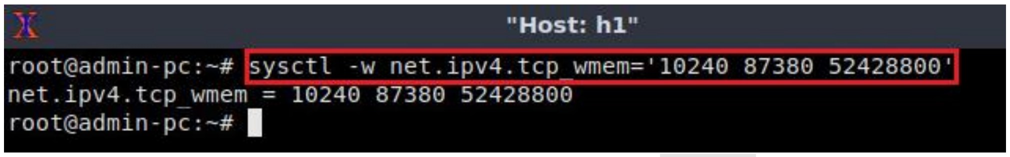


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 16. Receive window change in `sysctl`.

Step 7. 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, calculated by doubling the BDP).

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



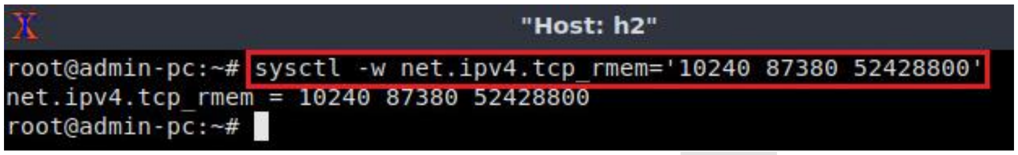
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 17. Send window change in `sysctl`.

Next, the same commands must be configured on host h2.

Step 8. To change the current receive-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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

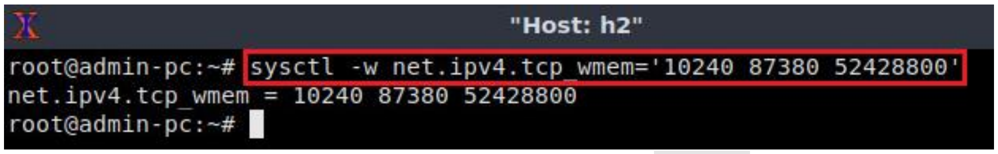


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 18. Receive window change in `sysctl`.

Step 9. To change the current send-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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

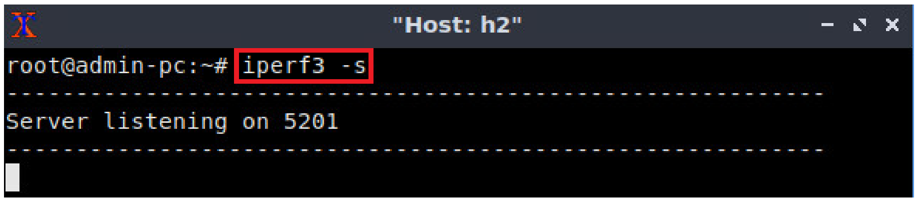


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#
```

Figure 19. Send window change in `sysctl`.

Step 10. The user can now verify the rate limit 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:

```
iperf3 -s
```

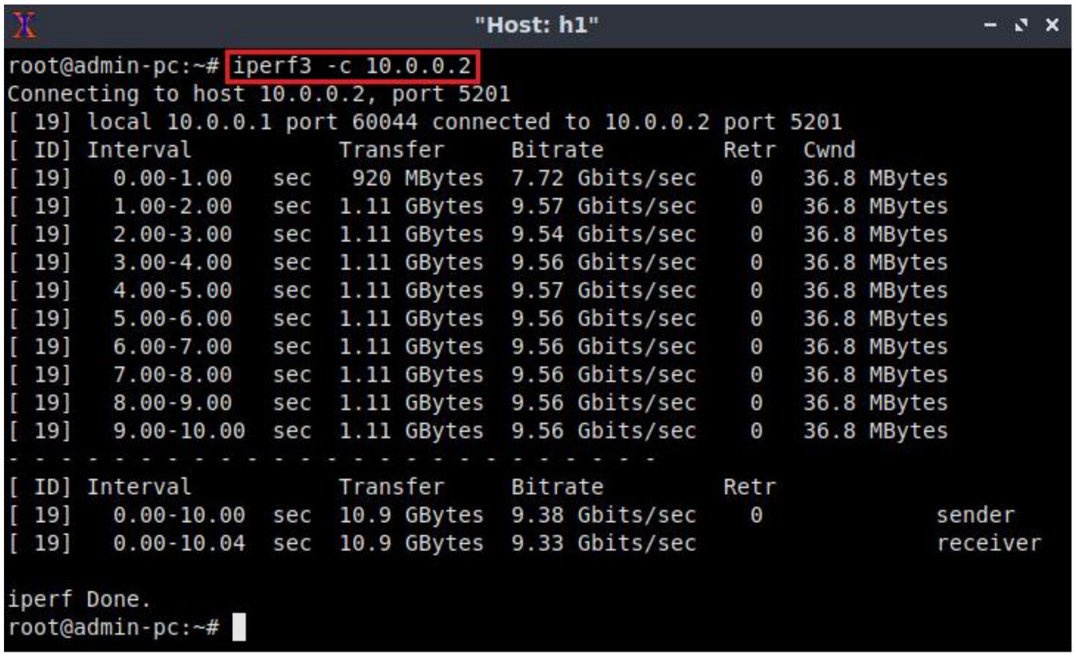


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 20. Host h2 running iPerf3 as server.

Step 11. Now to launch iPerf3 in client mode again by running the command `iperf3 -c 10.0.0.2` in host h1's terminal:

```
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
[ 19] local 10.0.0.1 port 60044 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 19]   0.00-1.00   sec   920 MBytes  7.72 Gbits/sec    0   36.8 MBytes
[ 19]   1.00-2.00   sec  1.11 GBytes  9.57 Gbits/sec    0   36.8 MBytes
[ 19]   2.00-3.00   sec  1.11 GBytes  9.54 Gbits/sec    0   36.8 MBytes
[ 19]   3.00-4.00   sec  1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]   4.00-5.00   sec  1.11 GBytes  9.57 Gbits/sec    0   36.8 MBytes
[ 19]   5.00-6.00   sec  1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]   6.00-7.00   sec  1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]   7.00-8.00   sec  1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]   8.00-9.00   sec  1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
[ 19]   9.00-10.00  sec  1.11 GBytes  9.56 Gbits/sec    0   36.8 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-10.00  sec  10.9 GBytes  9.38 Gbits/sec    0             sender
[ 19]   0.00-10.04  sec  10.9 GBytes  9.33 Gbits/sec                  receiver

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

Figure 21. iPerf3 throughput test.

Note the measured throughput now is approximately 9.38 Gbps, which is close to the value assigned in our `tbf` rule (10 Gbps).

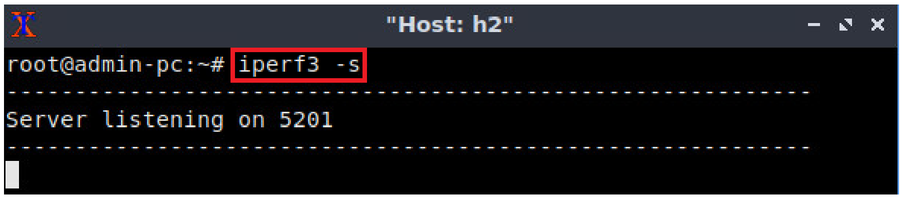
Step 12. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3 Calculating fairness among parallel flows

In this section, an iPerf3 test that includes several parallel streams is conducted, followed by the calculation of the fairness index.

Step 1. Now a test is conducted using parallel streams. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in Figure 22:

```
iperf3 -s
```

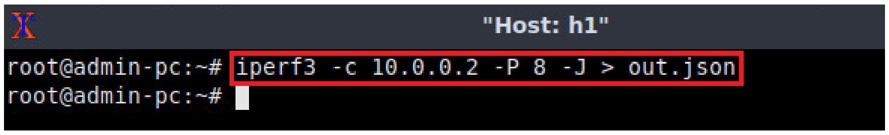


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 22. Host h2 running iPerf3 as server.

Step 2. Now the iPerf3 client should be launched with the `-P` option specified to start parallel streams. The -`J` option is also specified to indicate that JSON output is desired, and the redirection operator `>` to store the output in a file. Run the following command in host h1's terminal as shown in Figure 23:

```
iperf3 -c 10.0.0.2 -P 8 -J > out.json
```

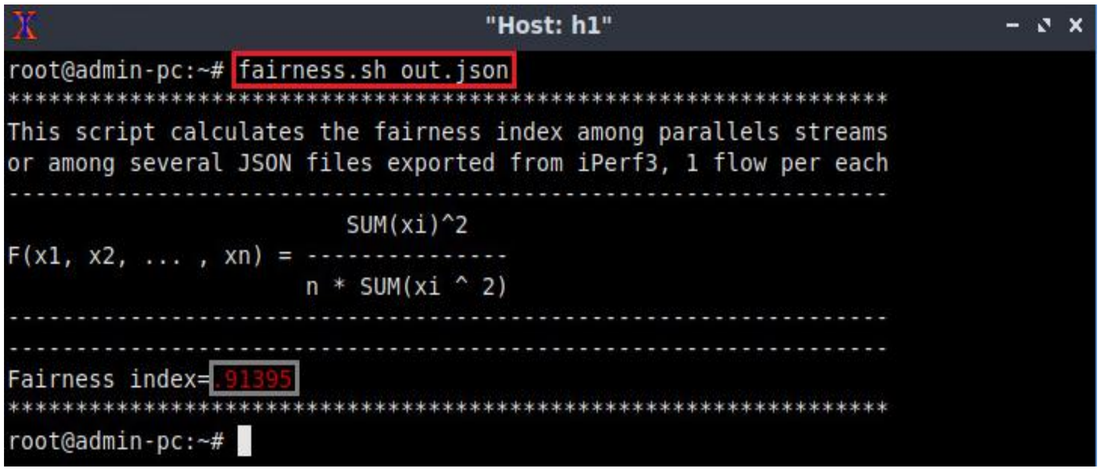


```
root@admin-pc:~# iperf3 -c 10.0.0.2 -P 8 -J > out.json
root@admin-pc:~#
```

Figure 23. Host h1 running iPerf3 as client with 8 parallel streams and saving output in file.

Step 3. The client includes a script called `fairness.sh`. Basically, this script accepts as input the JSON file exported by iPerf3 and calculates the fairness index. Run the following command to calculate the fairness index:

```
fairness.sh out.json
```



```
root@admin-pc:~# fairness.sh out.json
*****************************************************************
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----------------------------------------------------------------
                         SUM(xi)^2
F(x1, x2, ... , xn) = ---------------
                      n * SUM(xi ^ 2)
-----------------------------------------------------------------
-----------------------------------------------------------------
Fairness index=.91395
*****************************************************************
root@admin-pc:~#
```

Figure 24. Calculating the fairness index between the parallel streams.

In the above test, the fairness index is .91395, or 91% fair. Note that this result may vary according to the result of your emulation test.

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 Calculating fairness among several hosts with the same congestion control algorithm

In the previous section, we calculated the fairness index among several parallel streams, all initiated by a single host. In this section we calculate the fairness index among two transmitting devices. Specifically, an iPerf3 client is executed on host h1 and connected to host h2 (iPerf3 server); another iPerf3 client is executed on host h3 and connected to host h4 (iPerf3 server).

To calculate the fairness index, the transmitting hosts should initiate their transmissions simultaneously. Since it is difficult to start the clients at the same time, the client's machine provides a script that automates this process.

Step 1. Close the terminals of host h1 and host h2.

Step 2. Go to Mininet's terminal, i.e., the one launched when MiniEdit was started.

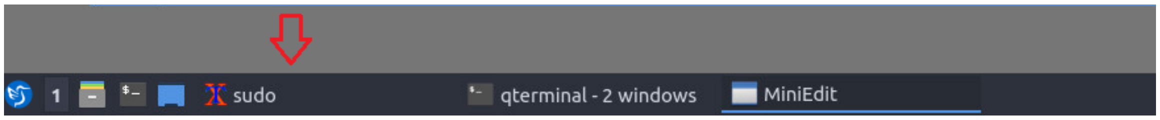
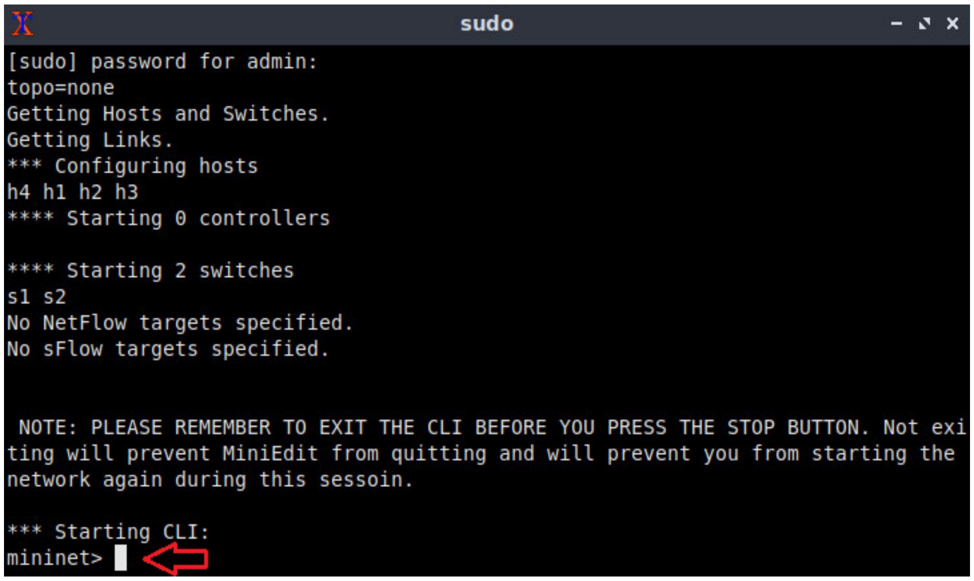


Figure 25. Opening Mininet's terminal.

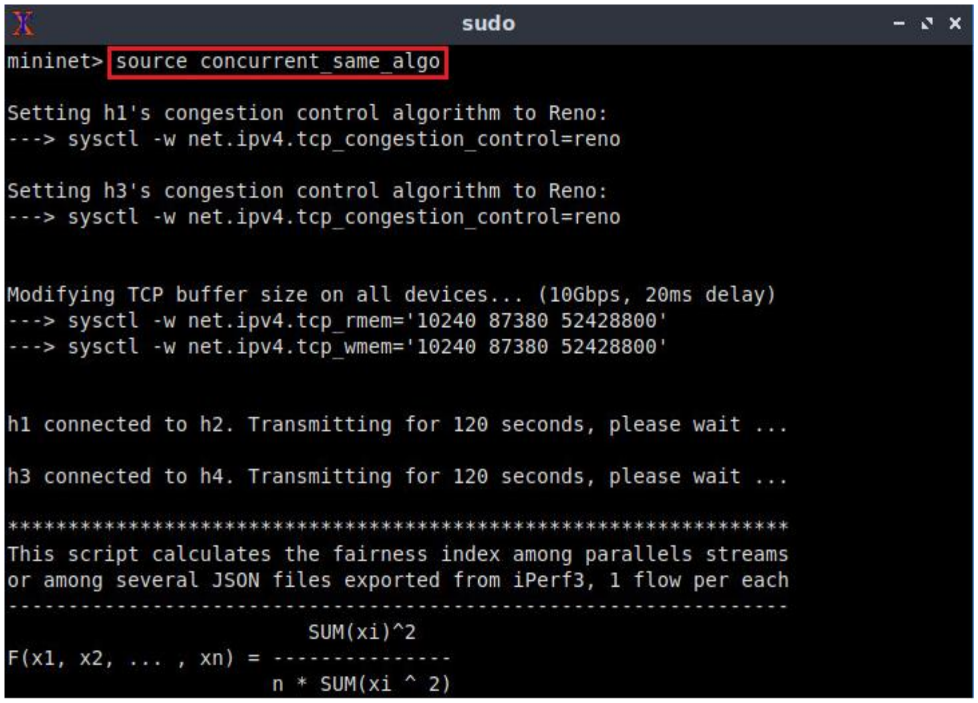


```
[sudo] password for admin:
topo=none
Getting Hosts and Switches.
Getting Links.
*** Configuring hosts
h4 h1 h2 h3
**** Starting 0 controllers

**** Starting 2 switches
s1 s2
No NetFlow targets specified.
No sFlow targets specified.


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exi
ting will prevent MiniEdit from quitting and will prevent you from starting the
network again during this sessoin.

*** Starting CLI:
mininet>
```

Figure 26. Mininet's terminal.

Step 3. Issue the following command on Mininet's terminal as shown in the figure below.

```
source concurrent_same_algo
```



```
mininet> source concurrent_same_algo

Setting h1's congestion control algorithm to Reno:
---> sysctl -w net.ipv4.tcp_congestion_control=reno

Setting h3's congestion control algorithm to Reno:
---> sysctl -w net.ipv4.tcp_congestion_control=reno


Modifying TCP buffer size on all devices... (10Gbps, 20ms delay)
---> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
---> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'


h1 connected to h2. Transmitting for 120 seconds, please wait ...

h3 connected to h4. Transmitting for 120 seconds, please wait ...

*****************************************************************
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----------------------------------------------------------------
                         SUM(xi)^2
F(x1, x2, ... , xn) = ---------------
                       n * SUM(xi ^ 2)
```

Figure 27. Running the tests simultaneously for 120 seconds. Both host h1 and host h3 are using Reno.

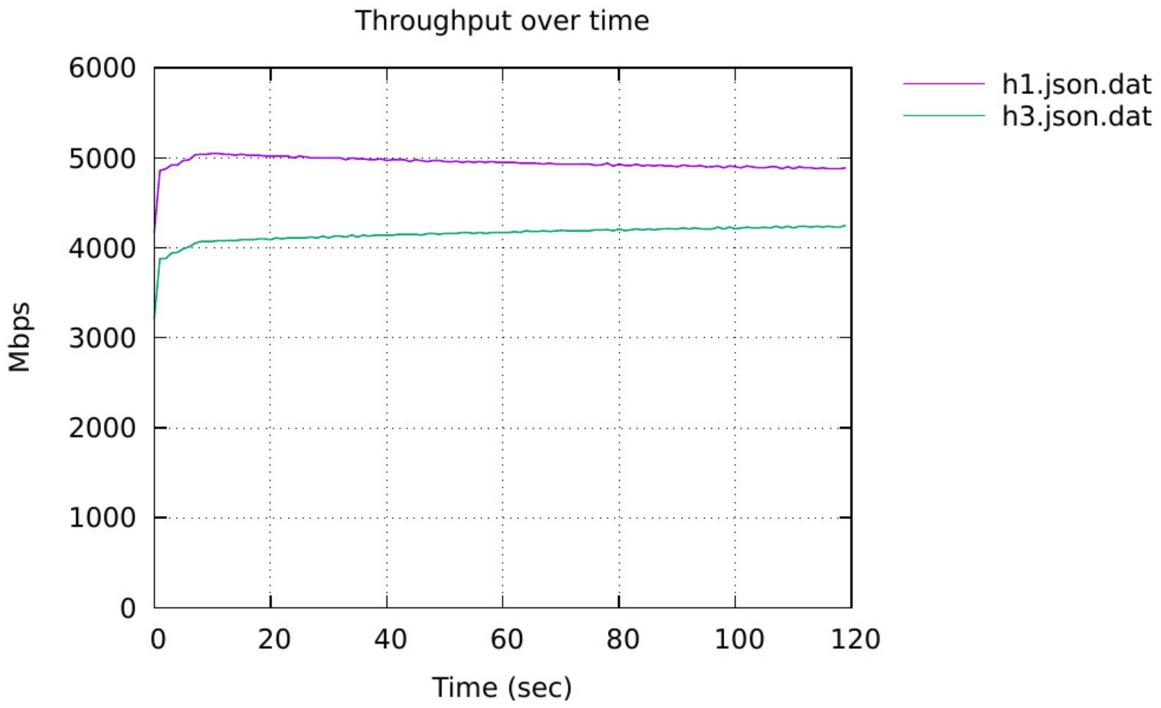
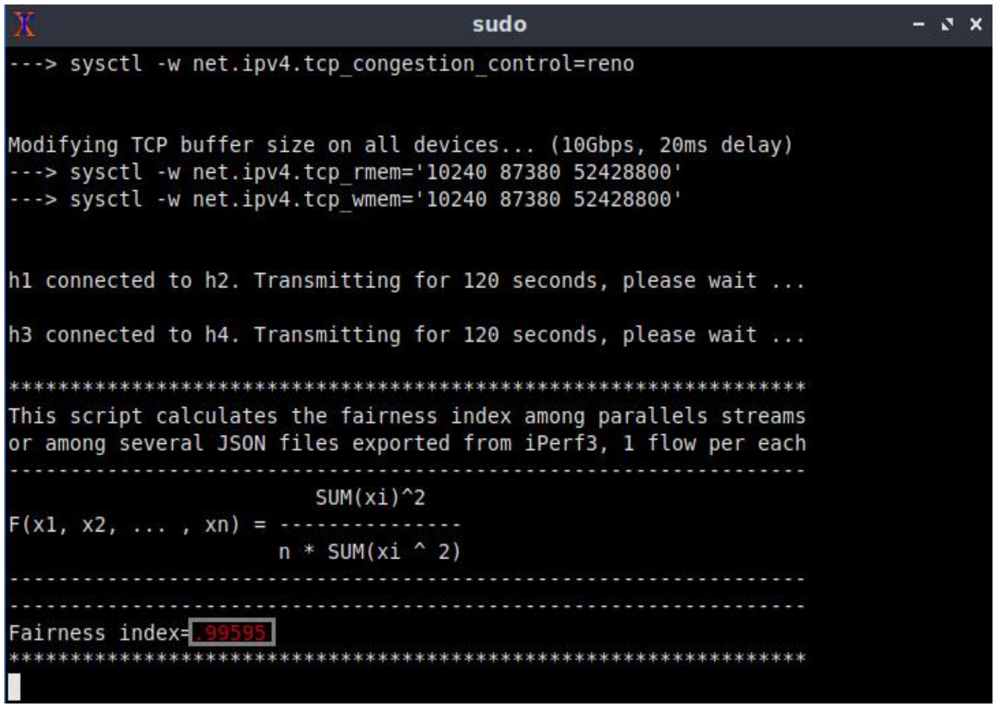


Figure 28. Throughput of host h1 and host h3.

The above graph shows that the throughput of host h1 is close to that of host h3. Therefore, the fairness index should be close to 1 (100%). Note that this result may vary according to the result of your emulation test.

Step 4. Close the graph window and go back to Mininet's terminal. The fairness index is displayed at the end as shown in the figure below.



```
---> sysctl -w net.ipv4.tcp_congestion_control=reno


Modifying TCP buffer size on all devices... (10Gbps, 20ms delay)
---> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
---> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'


h1 connected to h2. Transmitting for 120 seconds, please wait ...

h3 connected to h4. Transmitting for 120 seconds, please wait ...

*****************************************************************
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----------------------------------------------------------------
                         SUM(xi)^2
F(x1, x2, ... , xn) = --------------
                      n * SUM(xi ^ 2)
-----------------------------------------------------------------
-----------------------------------------------------------------
Fairness index=.99595
*****************************************************************
```

Figure 29. Calculated fairness index.

The above figure shows a fairness index of .99595. This value indicates that the bottleneck bandwidth was 99% fairly shared among host h1 and host h3. Note that this result may vary according to the result of your emulation test.

5 Calculating fairness among hosts with different congestion control algorithms

In the previous test, we calculated the fairness index while using the same congestion control algorithm (Reno). In this section we repeat the test, but with host h1 using Reno and host h3 using BBR.

Step 1. Go back to Mininet’s terminal, i.e., the one launched when MiniEdit was started.

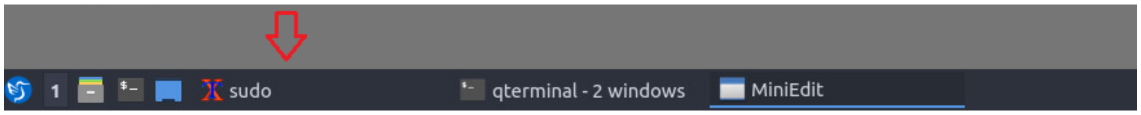
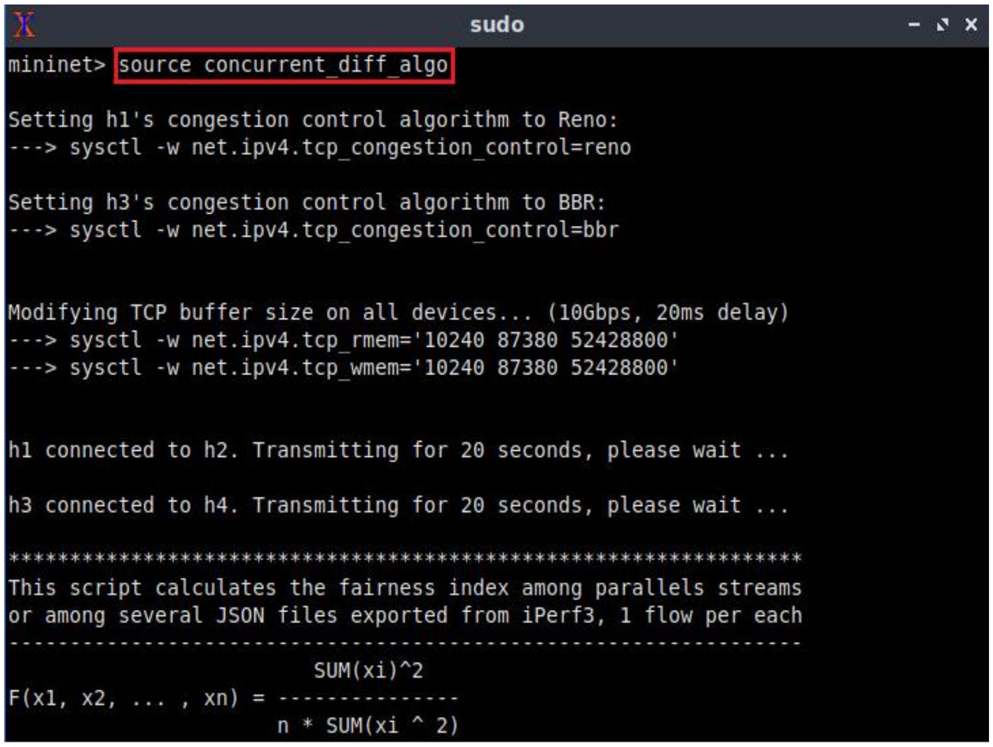


Figure 30. Opening Mininet’s terminal.

Step 2. Issue the following command on Mininet’s terminal as shown in the figure below.

```
source concurrent_diff_algo
```



```
mininet> source concurrent_diff_algo

Setting h1's congestion control algorithm to Reno:
---> sysctl -w net.ipv4.tcp_congestion_control=reno

Setting h3's congestion control algorithm to BBR:
---> sysctl -w net.ipv4.tcp_congestion_control=bbr


Modifying TCP buffer size on all devices... (10Gbps, 20ms delay)
---> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
---> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'


h1 connected to h2. Transmitting for 20 seconds, please wait ...

h3 connected to h4. Transmitting for 20 seconds, please wait ...

*****************************************************************
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----------------------------------------------------------------
                         SUM(xi)^2
F(x1, x2, ... , xn) = ---------------
                      n * SUM(xi ^ 2)
```

Figure 31. Running the tests simultaneously for 20 seconds. Host h1 is using Reno while host h3 is using BBR.

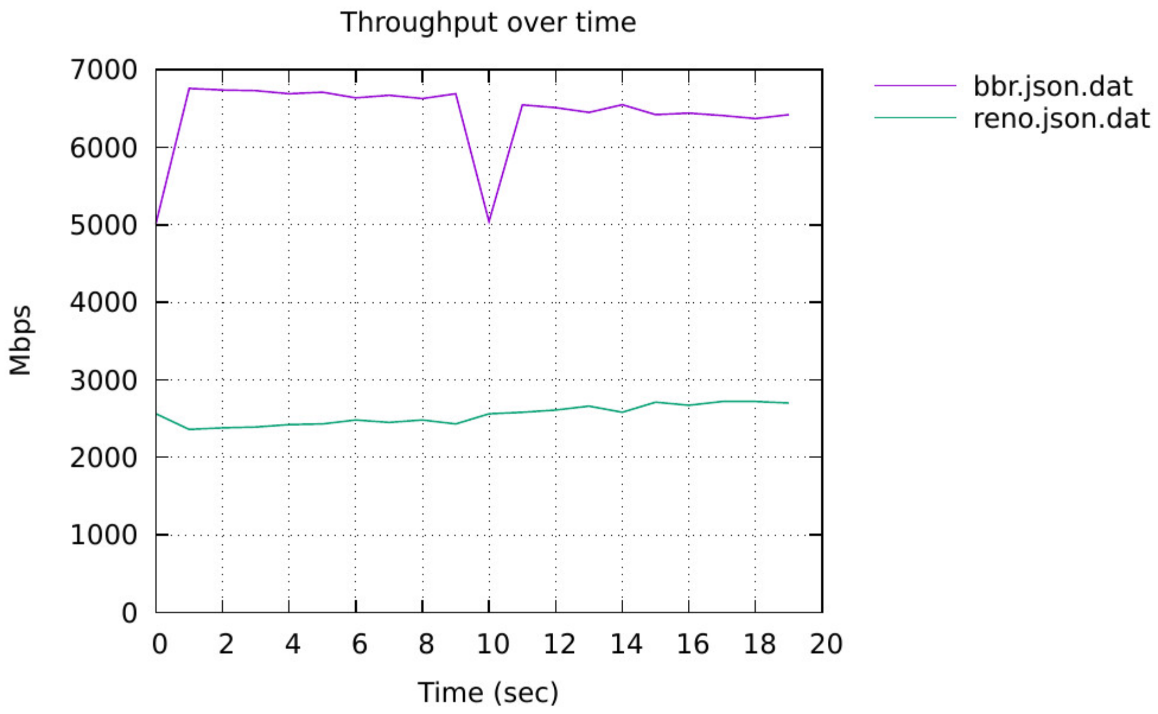
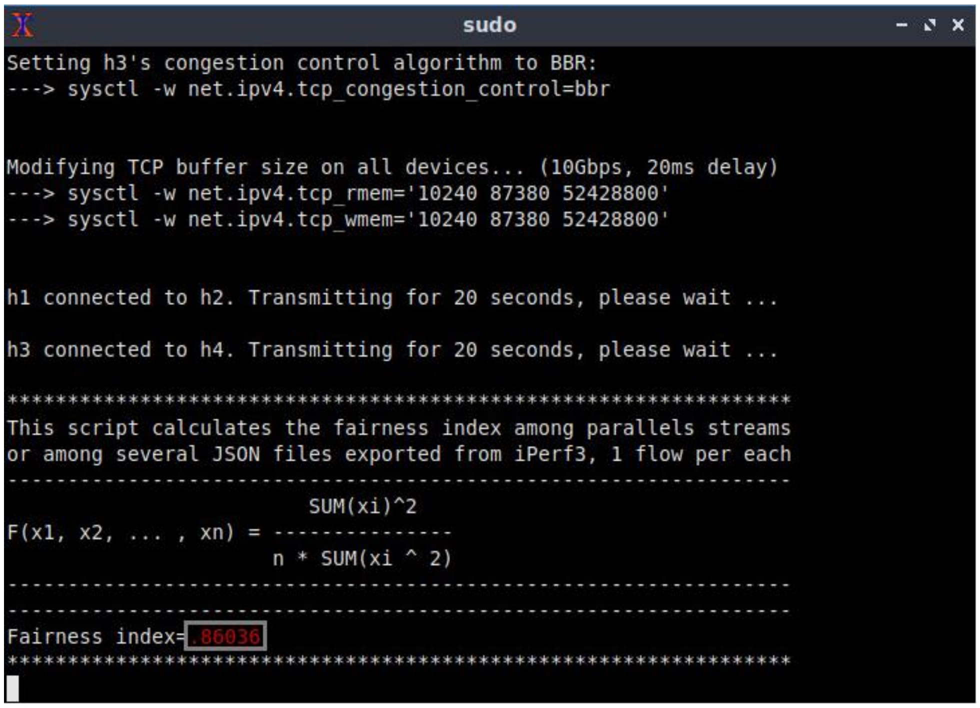


Figure 32. Throughput of host h1 and host h3.

The above graph shows that the device configured with BBR has a larger bandwidth allocation than that configured with Reno. Therefore, the fairness index will not be close to 1 (100%).

Step 3. Close the graph window and go back to Mininet's terminal. The fairness index is displayed at the end as shown in the figure below.



```
Setting h3's congestion control algorithm to BBR:
---> sysctl -w net.ipv4.tcp_congestion_control=bbr


Modifying TCP buffer size on all devices... (10Gbps, 20ms delay)
---> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
---> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'


h1 connected to h2. Transmitting for 20 seconds, please wait ...

h3 connected to h4. Transmitting for 20 seconds, please wait ...

*****************************************************************
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----------------------------------------------------------------
                          SUM(xi)^2
F(x1, x2, ... , xn) = ---------------
                       n * SUM(xi ^ 2)
-----------------------------------------------------------------
-----------------------------------------------------------------
Fairness index=.86036
*****************************************************************
```

Figure 33. Calculated fairness index.

The above figure shows a fairness index of .86036 (~ 86%), which is very far from 100%. This value indicates that the bottleneck bandwidth was 86% fairly shared among host h1 and host h3. Note that this result may vary according to the result of your emulation test.

This concludes Lab 10. Stop the emulation and then exit out of MiniEdit.

References

1. D. Chiu, R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," Journal of Computer Networks and ISDN Systems, vol. 17, issue 1, pp. 1-14, Jun. 1989.
2. A. Tanenbaum, D. Wetherall, "Computer networks," 5th Edition, Prentice Hall, 2011.
3. J. Kurose, K. Ross, "Computer networking, a top-down approach," 7th Edition, Pearson, 2017.
4. R. Jain, D. Chiu, W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC Research Report TR-301, Sep. 1984.

NETWORK TOOLS AND PROTOCOLS

Lab 11: Router's Buffer Size

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 3
  1.1 Introduction to switching 4
  1.2 Router architecture 4
  1.3 Where does packet loss occur? 5
  1.4 Buffer size 5
2 Lab topology 6
  2.1 Starting host h1, host h2, host h3 and host h4 7
  2.2 Modifying hosts' buffer size 9
  2.3 Emulating high-latency WAN 12
  2.4 Testing connection 13
3 Testing throughput with 100·MTU switch's buffer size 14
  3.1 Setting switch S1's buffer size to 100·MTU 14
  3.2 TCP Cubic 14
  3.3 TCP Reno 16
  3.4 TCP BBR 18
4 Testing throughput with one BDP switch's buffer size 20
  4.1 Changing switch S1's buffer size to one BDP 20
  4.2 TCP Cubic 21
  4.3 TCP Reno 23
  4.4 TCP BBR 25
5 Emulating high-latency WAN with packet loss 27
  5.1 TCP Cubic 27
  5.2 TCP Reno 29
  5.3 TCP BBR 31
References 33

Overview

This lab reviews the internal architecture of routers and switches. These devices are essential in high-speed networks, as they must be capable of absorbing transient packet bursts generated by large flows and thus avoid packet loss. The lab describes the buffer requirements to absorb such traffic fluctuations, which are then validated by experimental results.

Objectives

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

1. Describe the internal architecture of routers and switches.
2. Understand the importance of buffers of routers and switches to prevent packet loss.
3. Conduct experiments with routers and switches of variable buffer sizes.
4. Calculate the buffer size required by routers and switches to absorb transient bursts.
5. Use experimental results to draw conclusions and make appropriate decision related to routers' and switches' buffers.

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 with 100*MTU switch's buffer size.
4. Section 4: Testing throughput with one BDP switch's buffer size.
5. Section 5: Emulating high-latency WAN with packet loss.

1 Introduction

1.1 Introduction to switching

Two essential functions performed by routers are routing and forwarding. Routing refers to the determination of the route taken by packets. Forwarding refers to the switching of a packet from the input port to the appropriate output port. The term switching is also used interchangeably with forwarding. Traditional routing approaches such as static and dynamic routing (e.g., Open Shortest Path First (OSPF)[1], BGP[2]) are used in the implementation of high-speed networks, e.g., Science DMZs. Routing events, such as routing table updates, occur at the millisecond, second, or minute timescale, and best practices used in regular enterprise networks are applicable to high-speed networks as well. These functions are sometimes collectively referred to as the control plane and are usually implemented in software and execute on the routing processor (typically a traditional CPU), see Figure 1. On the other hand, with transmission rates of 10 Gbps and above, the forwarding operations related to moving packets from input to output interfaces at very high speed must occur at the nanosecond timescale. Thus, forwarding operations, collectively referred to as forwarding or data plane, are executed in specialized hardware and optimized for performance.

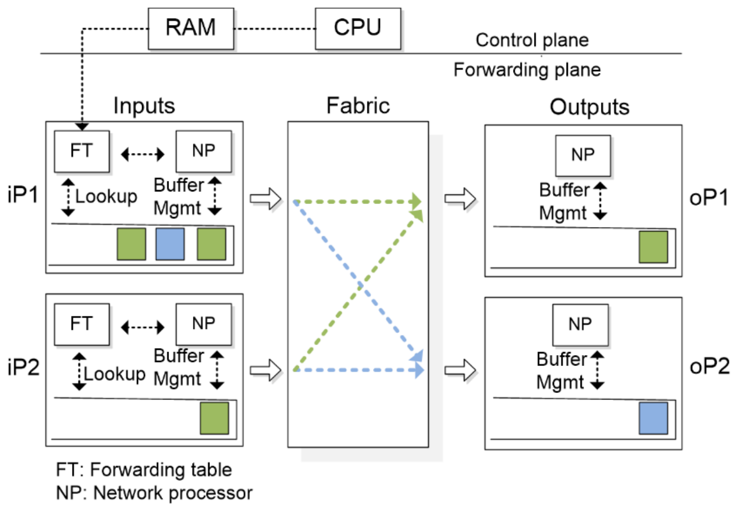


Figure 1. A generic router architecture.

Since forwarding functionality is common in both routers and switches, this lab reviews the architecture and forwarding-related attributes of switches. These attributes are applicable to routers as well; thus, for this lab, the terms switch and router are used interchangeably.

1.2 Router architecture

Consider the generic router architecture that is shown in Figure 1. Modern routers may have a network processor (NP) and a table derived from the routing table in each port, which is referred to as the forwarding table (FT) or forwarding information base (FIB). The router in Figure 1 has two input ports, iP1 and iP2, with their respective queues. iP1 has three packets in its queue, which will be forwarded to output ports oP1 (green packets) and oP2 (blue packet) by the fabric. A switch fabric moves packets from input to output ports. Switch fabric designs are shared memory, crossbar network, and bus. In shared memory switches, packets are written into a memory location by an input port and then read from that memory location by the output port. Crossbar switches implement a matrix of pathways that can be configured to connect any input port to any output port. Bus switches use a shared bus to move packets from the input ports to the output ports[3].

Router queues/buffers absorb traffic fluctuations. Even in the absence of congestion, fluctuations are present, resulting mostly from coincident traffic bursts[4]. Consider an input buffer implemented as a first-in first-out in the router of Figure 1. As iP1 and iP2 both have one packet to be forwarded to oP1 at the front of the buffer, only one of them, say the packet at iP2, will be forwarded to oP1. The consequence of this is that not only the first packet must wait at iP1. Also, the second packet that is queued at iP1 must wait, even though there is no contention for oP2. This phenomenon is known as Head-Of-Line (HOL) blocking[5]. To avoid HOL blocking, many switches use output buffering, a mixture of internal and output buffering, or techniques emulating output buffering such as Virtual Output Queueing (VOQ).

1.3 Where does packet loss occur?

Packet queues may form at both the input ports and the output ports. The location and extent of queueing (either at the input port queues or the output port queues) will depend on the traffic load, the relative speed of the switching fabric, and the line speed[5]. However, in modern switches with large switching rate capability, queues are commonly formed at output or transmission ports. A main contributing factor is the coincident arrivals of traffic bursts from different input ports that must be forwarded to the same output port. If transmission rates of input and output ports are the same, then packets from coincident arrivals must be momentarily buffered.

Note, however, that buffers will only prevent packet losses in case of transient traffic bursts. If those were not transient but permanent, such as approximately constant bit rates from large file transfers, the aggregate rate of input ports will surpass the rate of the output port. Thus, the output buffer would be permanently full, and the router would drop packets.

Packet loss occurs when a router drops the packet. It is the queues within a router, where such packets are dropped and lost.

1.4 Buffer size

From the above observation, a key question is how large should buffers be to absorb the fluctuations generated by TCP flows. 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[6, 7].

$$\text{Router's buffer size} = \text{C} \cdot \text{RTT [bits]} \quad \text{(single / small number of flows)}$$

Note that RTT is the average of individual RTTs. For example, if there are five TCP flows sharing a router's link (port), the RTT used in the equation above is the average value of the five flows, and the capacity C is the router's port capacity. E.g., for 250 millisecond connections and a 10 Gbps port, the router's buffer size equals 2.5 Gbits. The above quantity is a conservative value that can be used in high-throughput high-latency networks.

In 2004, Appenzeller et al.[8] presented a study that suggests that when there is a large number of TCP flows passing through a link, say N (e.g., hundreds, thousands or more), the amount of buffering can be reduced to:

$$\text{Router's buffer size} = \frac{\text{C} \cdot \text{RTT}}{\sqrt{N}} \text{ [bits]} \quad \text{(large number of flows N)}$$

This result is observed when there is no dominant flow and the router aggregates hundreds, thousands, or more flows. The observed effect is that the fluctuation of the sum of congestion windows are smoothed, and the buffer size at an output port can be reduced to the expression given above. Note that N can be very large for campus and backbone networks, and the reduction in needed buffer size can become considerable.

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.

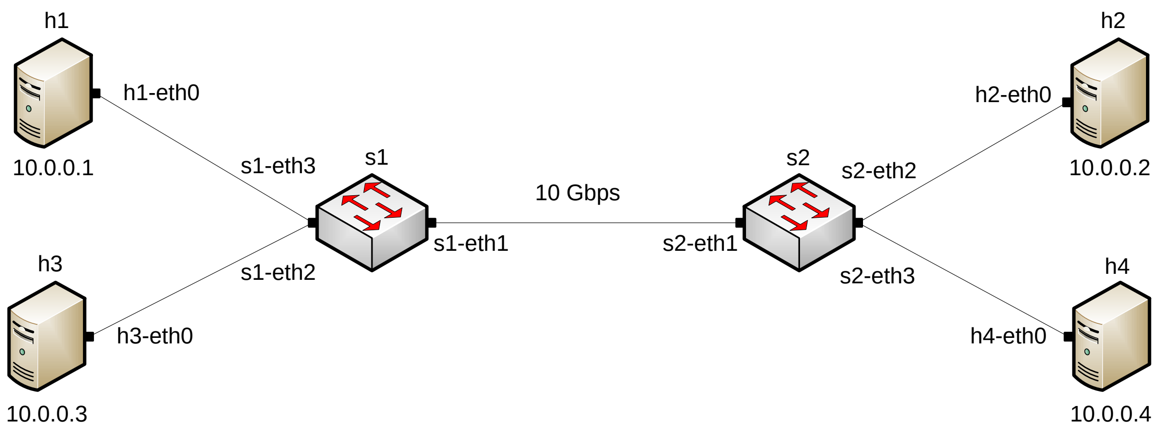


Figure 2. Lab topology.

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`.

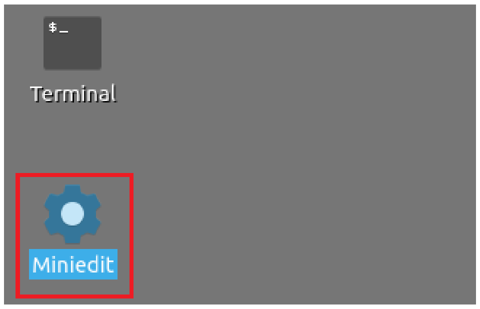


Figure 3. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 11.mn* topology file and click on *Open*.

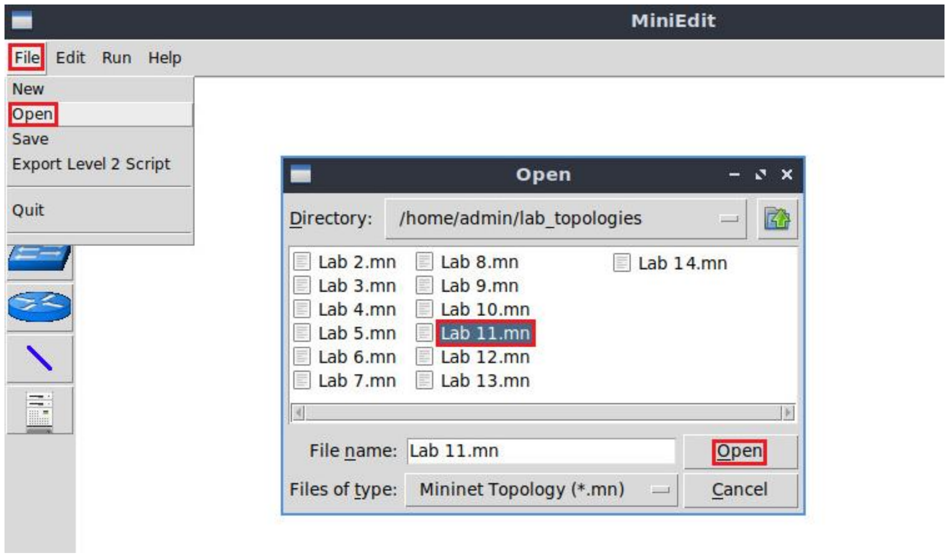


Figure 4. Miniedit's *Open* dialog.

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.

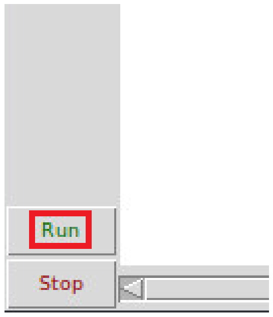


Figure 5. Running the emulation.

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

2.1 Starting host h1, host h2, host h3 and host h4

Step 1. Hold the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

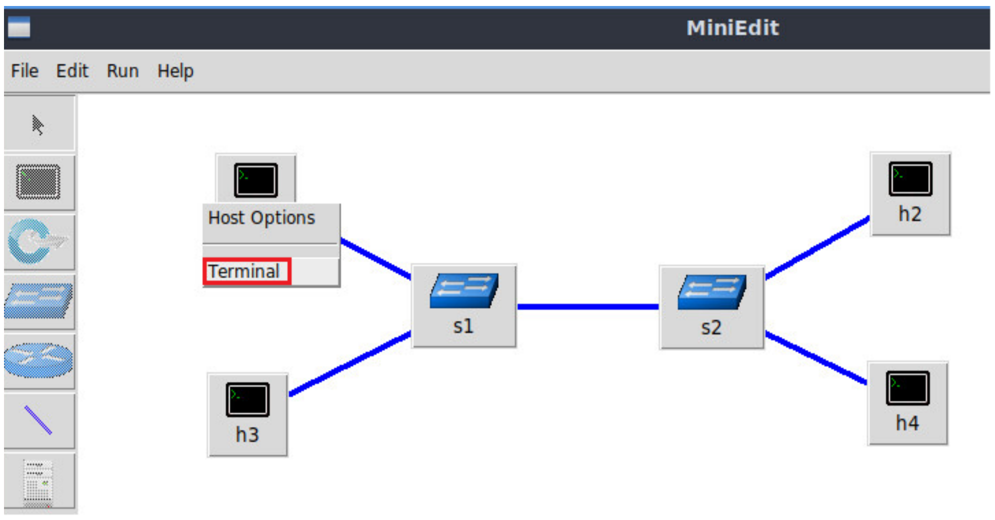
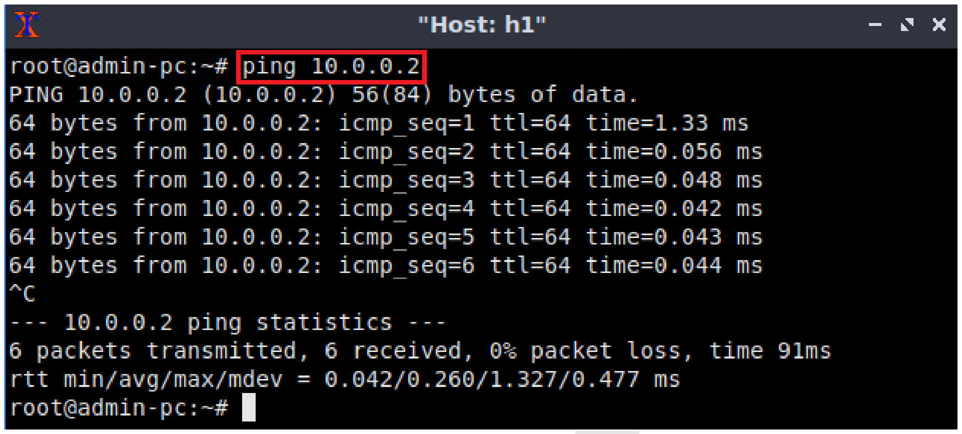


Figure 6. Opening a terminal on host h1.

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

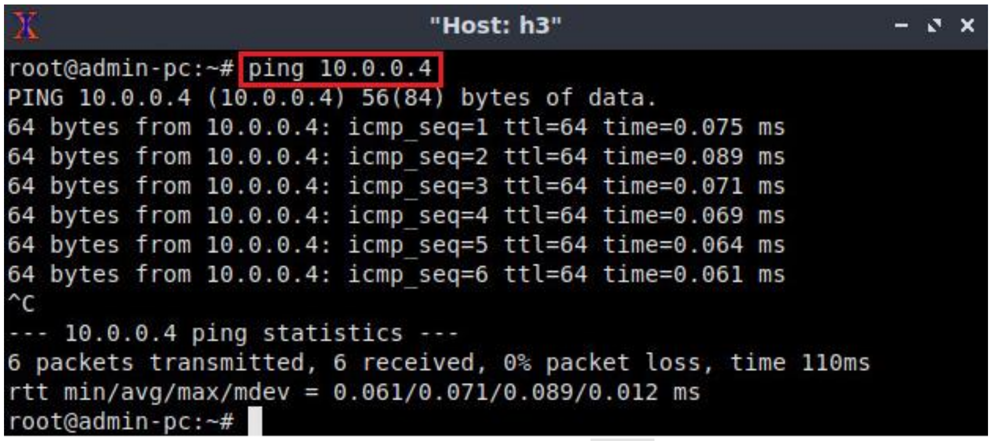
Step 3. Test connectivity between the end-hosts using the `ping` command. On host h1, type the command `ping 10.0.0.2`. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.



```
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 7. Connectivity test using `ping` command.

Step 4. Test connectivity between the end-hosts using the `ping` command. On host h3, type the command `ping 10.0.0.4`. This command tests the connectivity between host h3 and host h4. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.



```
root@admin-pc:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.075 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.089 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.069 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.064 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.061 ms
^C
--- 10.0.0.4 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 110ms
rtt min/avg/max/mdev = 0.061/0.071/0.089/0.012 ms
root@admin-pc:~#
```

Figure 8. Connectivity test using `ping` command.

2.2 Modifying hosts' buffer size

The following tests the bandwidth is limited to 10 Gbps, and the RTT (delay or latency) is 20ms.

In order to have enough TCP buffer size, we will set the sending and receiving buffer to $5 \cdot \text{BDP}$ in all hosts.

$$\text{BW} = 10{,}000{,}000{,}000 \text{ bits/second}$$

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

$$\begin{aligned}\text{BDP} &= 10{,}000{,}000{,}000 \cdot 0.02 = 200{,}000{,}000 \text{ bits}\\ &= 25{,}000{,}000 \text{ bytes} \approx 25 \text{ Mbytes}\end{aligned}$$

The send and receive buffer sizes should be set to 5 · BDP. We will use the 25 Mbytes value for the BDP instead of 25,000,000 bytes.

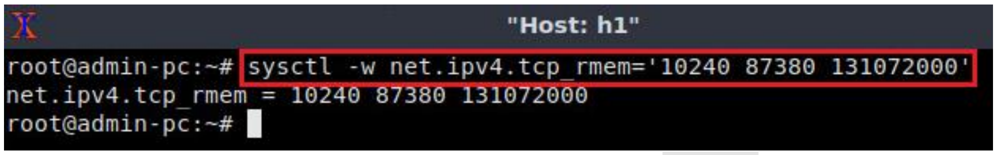
$$1 \text{ Mbyte} = 1024^2 \text{ bytes}$$

$$\text{BDP} = 25 \text{ Mbytes} = 25 \cdot 1024^2 \text{ bytes} = 26{,}214{,}400 \text{ bytes}$$

$$5 \cdot \text{BDP} = 5 \cdot 26{,}214{,}400 \text{ bytes} = 131{,}072{,}000 \text{ bytes}$$

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 131,072,000 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
```



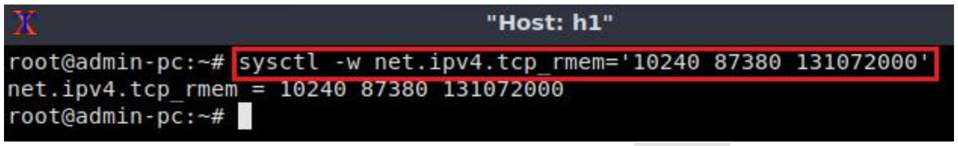
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
net.ipv4.tcp_rmem = 10240 87380 131072000
root@admin-pc:~#
```

Figure 9. 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. 131,072,000 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 131,072,000 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
```



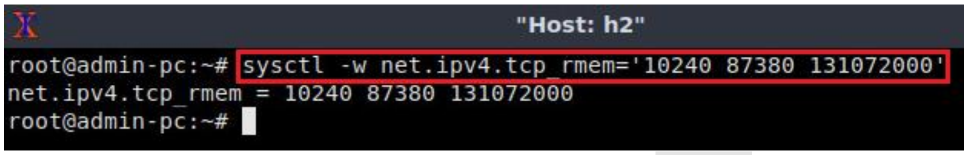
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
net.ipv4.tcp_rmem = 10240 87380 131072000
root@admin-pc:~#
```

Figure 10. Send window change in `sysctl`.

Next, the same commands must be configured on host h2, host h3, and host h4.

Step 3. To change the current receiver-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 131,072,000 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
```

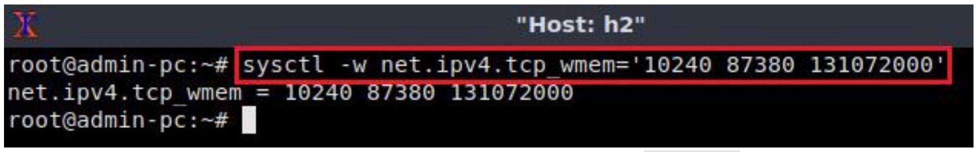


```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
net.ipv4.tcp_rmem = 10240 87380 131072000
root@admin-pc:~#
```

Figure 11. Receive window change in `sysctl`.

Step 4. To change the current send-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 131,072,000 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
```

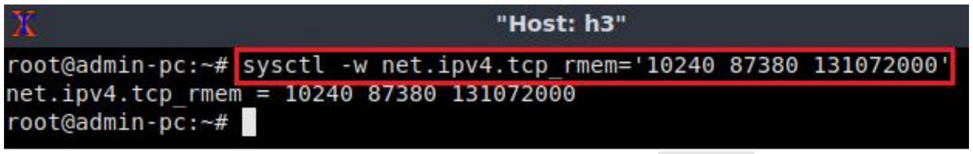


```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
net.ipv4.tcp_wmem = 10240 87380 131072000
root@admin-pc:~#
```

Figure 12. Send window change in `sysctl`.

Step 5. To change the current receiver-window size value(s), use the following command on host h3's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 131,072,000 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
```

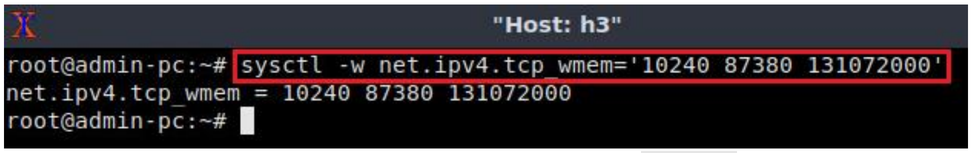


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
net.ipv4.tcp_rmem = 10240 87380 131072000
root@admin-pc:~#
```

Figure 13. Receive window change in `sysctl`.

Step 6. To change the current send-window size value(s), use the following command on host h3's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 131,072,000 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
```

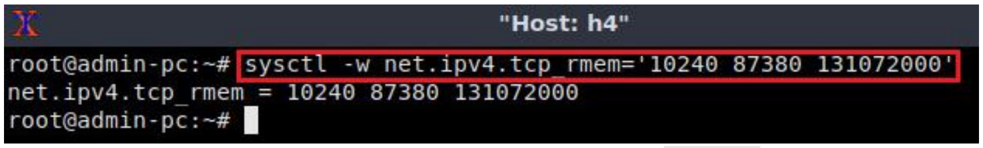


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
net.ipv4.tcp_wmem = 10240 87380 131072000
root@admin-pc:~#
```

Figure 14. Send window change in `sysctl`.

Step 7. To change the current receiver-window size value(s), use the following command on host h4's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 131,072,000 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
```

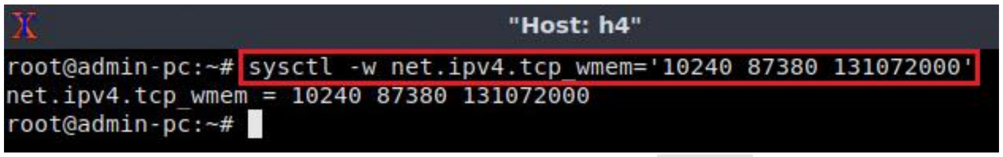


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
net.ipv4.tcp_rmem = 10240 87380 131072000
root@admin-pc:~#
```

Figure 15. Receive window change in `sysctl`.

Step 8. To change the current send-window size value(s), use the following command on host h4's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 131,072,000 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
```



```
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
net.ipv4.tcp_wmem = 10240 87380 131072000
root@admin-pc:~#
```

Figure 16. Send window change in `sysctl`.

2.3 Emulating high-latency WAN

This section emulates a high-latency WAN. We will first emulate 20ms delay between switches, setting 10ms delay on switch S1 and 10ms delay on switch S2, resulting in 20ms of Round-Trip Time (*RTT)*.

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

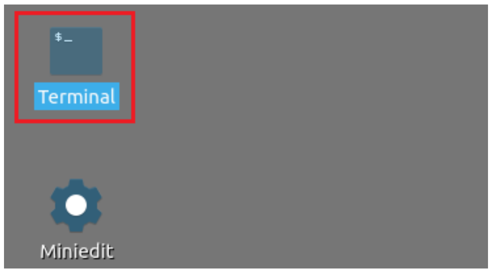
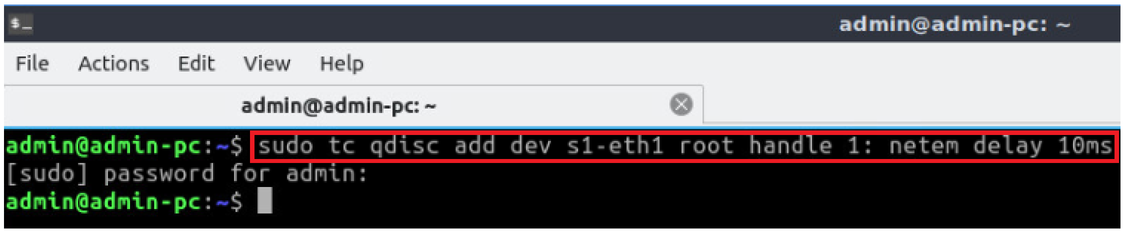


Figure 17. Shortcut to open a Linux terminal.

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-eth1* interface.

```
sudo tc qdisc add dev s1-eth1 root handle 1: netem delay 10ms
```

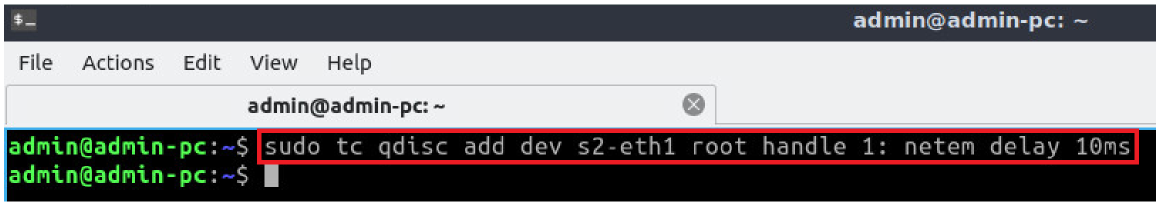


```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth1 root handle 1: netem delay 10ms
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 18. Adding delay of 10ms to switch S1's *s1-eth1* interface.

Step 3. Similarly, repeat again the previous step to set a 10ms delay to switch S2's interface. When prompted for a password, type `password` and hit *Enter*. This command introduces 10ms delay on switch S2's *s2-eth1* interface.

```
sudo tc qdisc add dev s2-eth1 root handle 1: netem delay 10ms
```



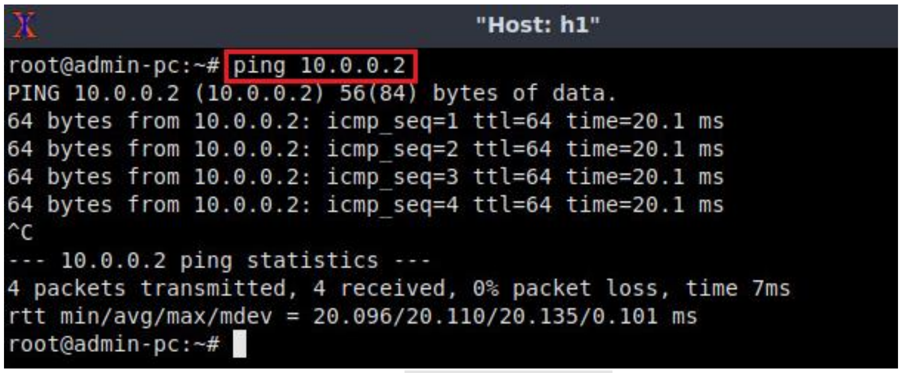
```
admin@admin-pc:~$ sudo tc qdisc add dev s2-eth1 root handle 1: netem delay 10ms
admin@admin-pc:~$
```

Figure 19. Adding delay of 10ms to switch S2's *s2-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 10.0.0.2`. 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.

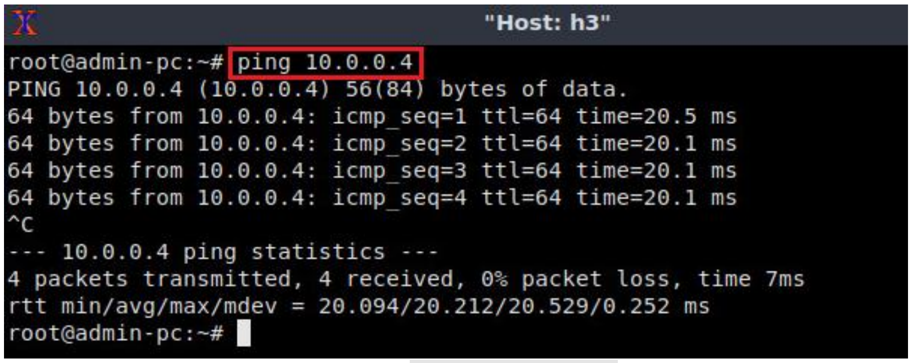


```
"Host: h1"
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.096/20.110/20.135/0.101 ms
root@admin-pc:~# 
```

Figure 20. 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 20.096, 20.110, 20.135, and 0.101 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

Step 2. On the terminal of host h3, type `ping 10.0.0.4`. 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: h3"
root@admin-pc:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=20.5 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.094/20.212/20.529/0.252 ms
root@admin-pc:~# 
```

Figure 21. Output of `ping 10.0.0.4` 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 20.094, 20.212, 20.529, and 0.252 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

3 Testing throughput with 100·MTU switch's buffer size

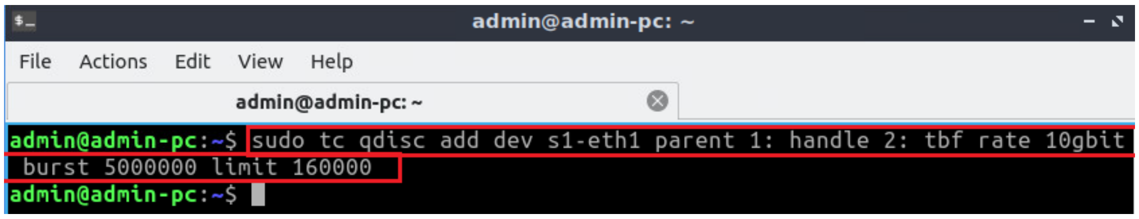
In this section, you are going to change the switch S1's buffer size to 100·MTU and emulate a 10 Gbps Wide Area Network (*WAN*) using the Token Bucket Filter (`tbf`). Then, you will test the throughput between host h1 and host h2 while there is another TCP flow between host h3 and host h4. On each test, you will modify the congestion control algorithm in host h1, namely, *cubic*, *reno* and *bbr*. The congestion control algorithm will still be *cubic* in host h3 for all tests. 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-eth1* interface. In the client's terminal, type the command below. When prompted for a password, type `password` and hit *Enter*.

- `rate`: 10gbit
- `burst`: 5,000,000
- `limit`: 160,000

```
sudo tc qdisc add dev s1-eth1 parent 1: handle 2: tbf rate 10gbit burst 5000000
limit 160000
```



```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth1 parent 1: handle 2: tbf rate 10gbit
 burst 5000000 limit 160000
admin@admin-pc:~$
```

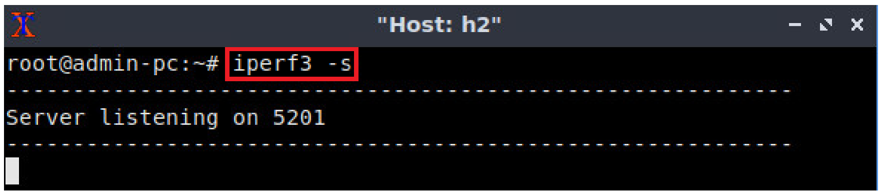
Figure 22. Limiting rate to 10 Gbps and setting the buffer size to 100·MTU on switch S1's interface.

3.2 TCP Cubic

The default congestion avoidance algorithm in the following test is *cubic* thus, there is no need to specify it manually.

Step 1. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

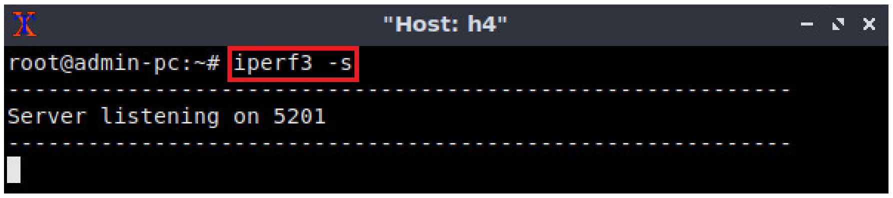


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 23. Starting iPerf3 server on host h2.

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

```
iperf3 -s
```



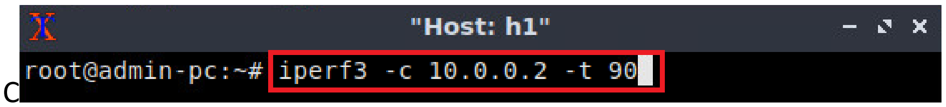
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 24. Starting iPerf3 server on host h4.

The following two steps should be executed almost simultaneously thus, you will type the commands displayed in Step 3 and Step 4 then, in Step 5 you will execute them.

Step 3. Type the following iPerf3 command in host h1's terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

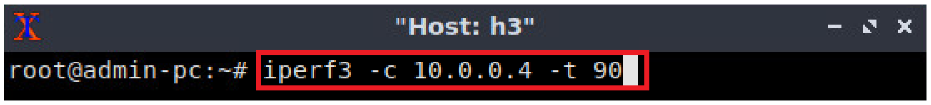


```
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 25. Typing iPerf3 client command on host h1.

Step 4. Type the following iPerf3 command in host h3's terminal without executing it.

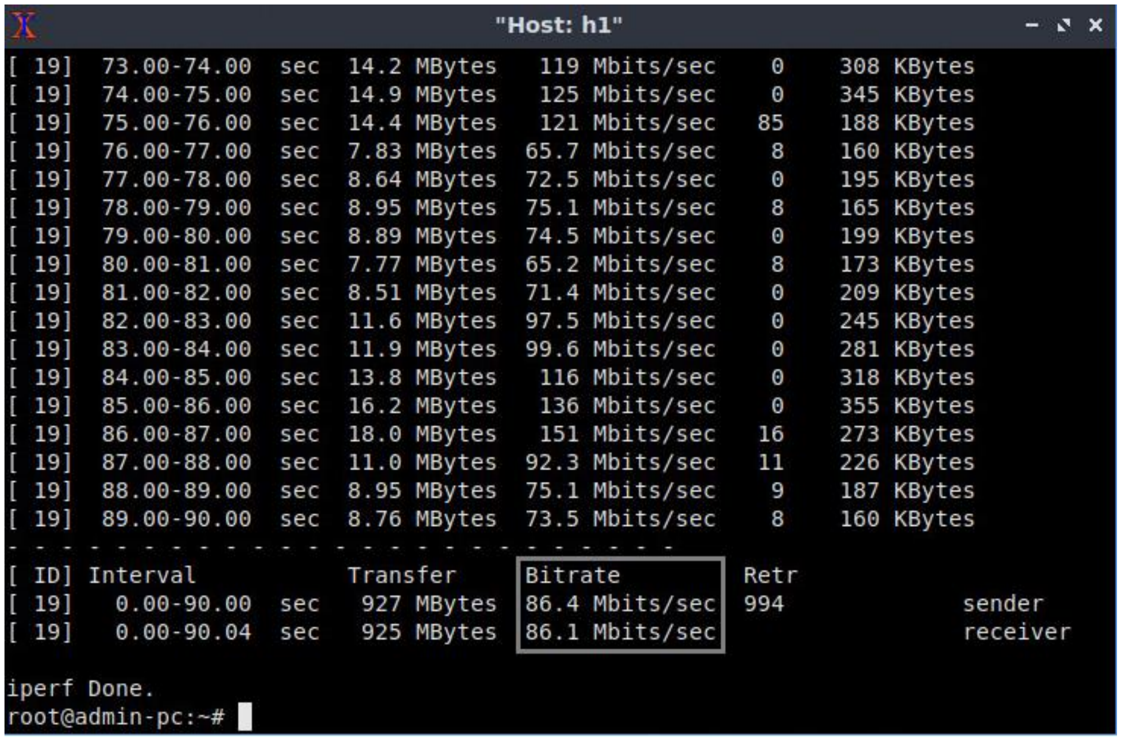
```
iperf3 -c 10.0.0.2 -t 90
```



```
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 26. Typing iPerf3 client command on host h3.

Step 5. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
"Host: h1"
[ 19]  73.00-74.00  sec  14.2 MBytes   119 Mbits/sec    0    308 KBytes
[ 19]  74.00-75.00  sec  14.9 MBytes   125 Mbits/sec    0    345 KBytes
[ 19]  75.00-76.00  sec  14.4 MBytes   121 Mbits/sec   85    188 KBytes
[ 19]  76.00-77.00  sec  7.83 MBytes  65.7 Mbits/sec    8    160 KBytes
[ 19]  77.00-78.00  sec  8.64 MBytes  72.5 Mbits/sec    0    195 KBytes
[ 19]  78.00-79.00  sec  8.95 MBytes  75.1 Mbits/sec    8    165 KBytes
[ 19]  79.00-80.00  sec  8.89 MBytes  74.5 Mbits/sec    0    199 KBytes
[ 19]  80.00-81.00  sec  7.77 MBytes  65.2 Mbits/sec    8    173 KBytes
[ 19]  81.00-82.00  sec  8.51 MBytes  71.4 Mbits/sec    0    209 KBytes
[ 19]  82.00-83.00  sec  11.6 MBytes  97.5 Mbits/sec    0    245 KBytes
[ 19]  83.00-84.00  sec  11.9 MBytes  99.6 Mbits/sec    0    281 KBytes
[ 19]  84.00-85.00  sec  13.8 MBytes   116 Mbits/sec    0    318 KBytes
[ 19]  85.00-86.00  sec  16.2 MBytes   136 Mbits/sec    0    355 KBytes
[ 19]  86.00-87.00  sec  18.0 MBytes   151 Mbits/sec   16    273 KBytes
[ 19]  87.00-88.00  sec  11.0 MBytes  92.3 Mbits/sec   11    226 KBytes
[ 19]  88.00-89.00  sec  8.95 MBytes  75.1 Mbits/sec    9    187 KBytes
[ 19]  89.00-90.00  sec  8.76 MBytes  73.5 Mbits/sec    8    160 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec   927 MBytes  86.4 Mbits/sec  994             sender
[ 19]   0.00-90.04  sec   925 MBytes  86.1 Mbits/sec                  receiver

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

Figure 27. Running iPerf3 client on host h1.

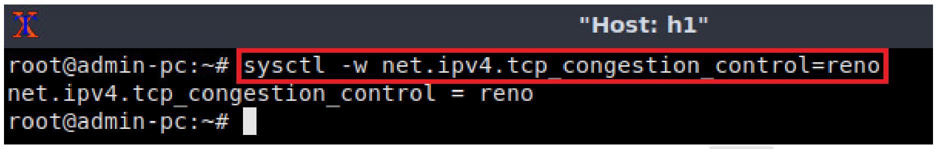
The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 86.4 Mbps (sender) and 86.1 Mbps (receiver), and the number of retransmissions is 994. Host h3's results are similar to the above, however we are just focused on host h1's results.

Step 6. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

3.3 TCP Reno

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Reno by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=reno
```



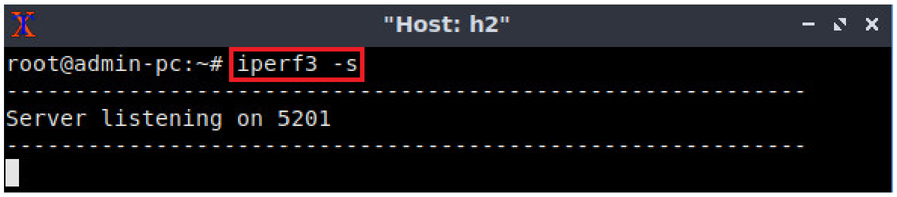
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
root@admin-pc:~#
```

Figure 28. Changing TCP congestion control algorithm to `reno` in host h1.

Note that host h3's congestion control algorithm is cubic by default.

Step 2. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

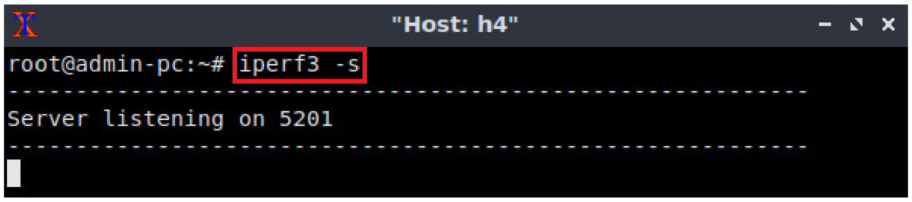


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 29. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```



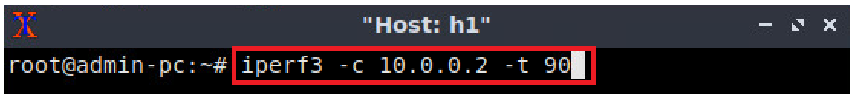
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 30. Starting iPerf3 server on host h4.

The following two steps should be executed almost simultaneously thus, you will type the commands displayed in Step 3 and Step 4 then, in Step 5 you will execute them.

Step 4. Type the following iPerf3 command in host h1’s terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

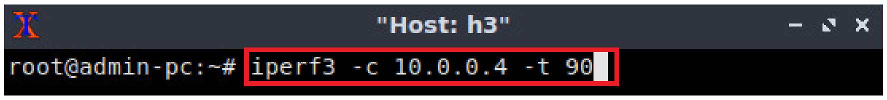


```
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 31. Typing iPerf3 client command on host h1.

Step 5. Type the following iPerf3 command in host h3’s terminal without executing it.

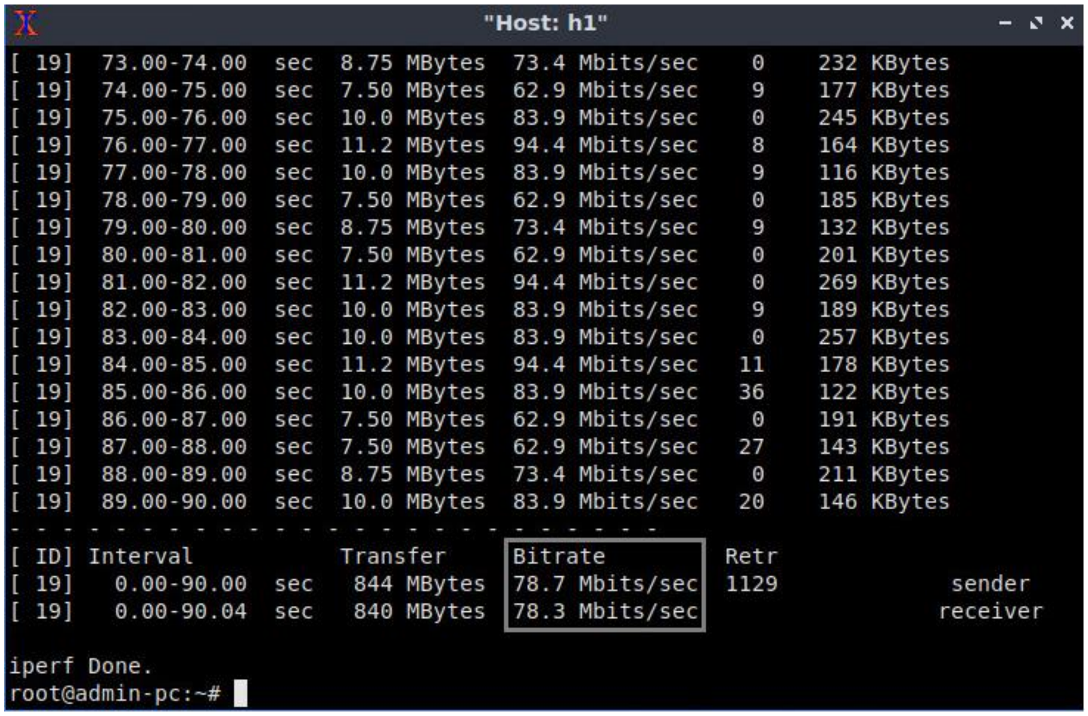
```
iperf3 -c 10.0.0.2 -t 90
```



```
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 32. Typing iPerf3 client command on host h3.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
[ 19]  73.00-74.00  sec  8.75 MBytes  73.4 Mbits/sec    0    232 KBytes
[ 19]  74.00-75.00  sec  7.50 MBytes  62.9 Mbits/sec    9    177 KBytes
[ 19]  75.00-76.00  sec  10.0 MBytes  83.9 Mbits/sec    0    245 KBytes
[ 19]  76.00-77.00  sec  11.2 MBytes  94.4 Mbits/sec    8    164 KBytes
[ 19]  77.00-78.00  sec  10.0 MBytes  83.9 Mbits/sec    9    116 KBytes
[ 19]  78.00-79.00  sec  7.50 MBytes  62.9 Mbits/sec    0    185 KBytes
[ 19]  79.00-80.00  sec  8.75 MBytes  73.4 Mbits/sec    9    132 KBytes
[ 19]  80.00-81.00  sec  7.50 MBytes  62.9 Mbits/sec    0    201 KBytes
[ 19]  81.00-82.00  sec  11.2 MBytes  94.4 Mbits/sec    0    269 KBytes
[ 19]  82.00-83.00  sec  10.0 MBytes  83.9 Mbits/sec    9    189 KBytes
[ 19]  83.00-84.00  sec  10.0 MBytes  83.9 Mbits/sec    0    257 KBytes
[ 19]  84.00-85.00  sec  11.2 MBytes  94.4 Mbits/sec   11    178 KBytes
[ 19]  85.00-86.00  sec  10.0 MBytes  83.9 Mbits/sec   36    122 KBytes
[ 19]  86.00-87.00  sec  7.50 MBytes  62.9 Mbits/sec    0    191 KBytes
[ 19]  87.00-88.00  sec  7.50 MBytes  62.9 Mbits/sec   27    143 KBytes
[ 19]  88.00-89.00  sec  8.75 MBytes  73.4 Mbits/sec    0    211 KBytes
[ 19]  89.00-90.00  sec  10.0 MBytes  83.9 Mbits/sec   20    146 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec   844 MBytes  78.7 Mbits/sec  1129             sender
[ 19]   0.00-90.04  sec   840 MBytes  78.3 Mbits/sec                  receiver

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

Figure 33. Running iPerf3 client on host h1.

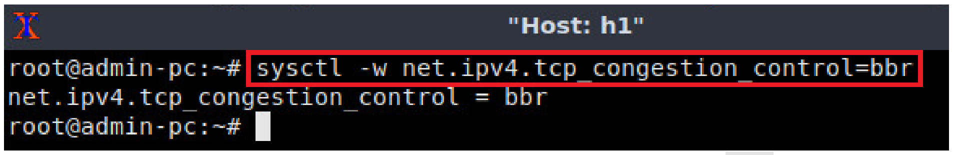
The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 78.7 Mbps (sender) and 78.3 Mbps (receiver), and the number of retransmissions is 1129. Host h3's results are similar to the figure above, however we are just focused on host h1's results.

Step 7. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

3.4 TCP BBR

Step 1. In host h1's terminal, change the TCP congestion control algorithm to BBR by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=bbr
```



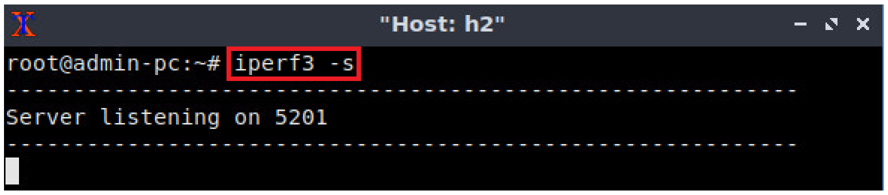
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=bbr
net.ipv4.tcp_congestion_control = bbr
root@admin-pc:~#
```

Figure 34. Changing TCP congestion control algorithm to `bbr` in host h1.

Note that host h3's congestion control algorithm is cubic by default.

Step 2. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

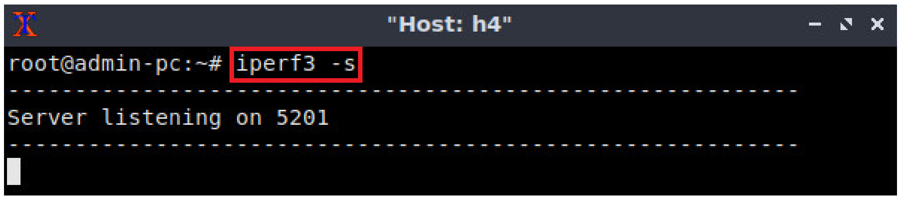


```
"Host: h2"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 35. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



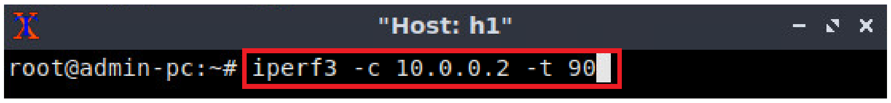
```
"Host: h4"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 36. Starting iPerf3 server on host h4.

The following two steps should be executed almost simultaneously, thus you will type the commands displayed in Step 3 and Step 4, then in Step 5 you will execute them.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

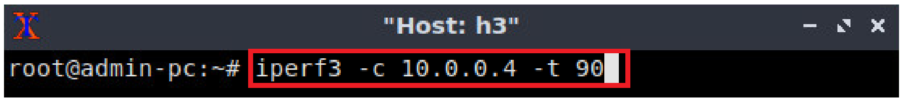


```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 37. Typing iPerf3 client command on host h1.

Step 5. Type the following iPerf3 command in host h3's terminal without executing it.

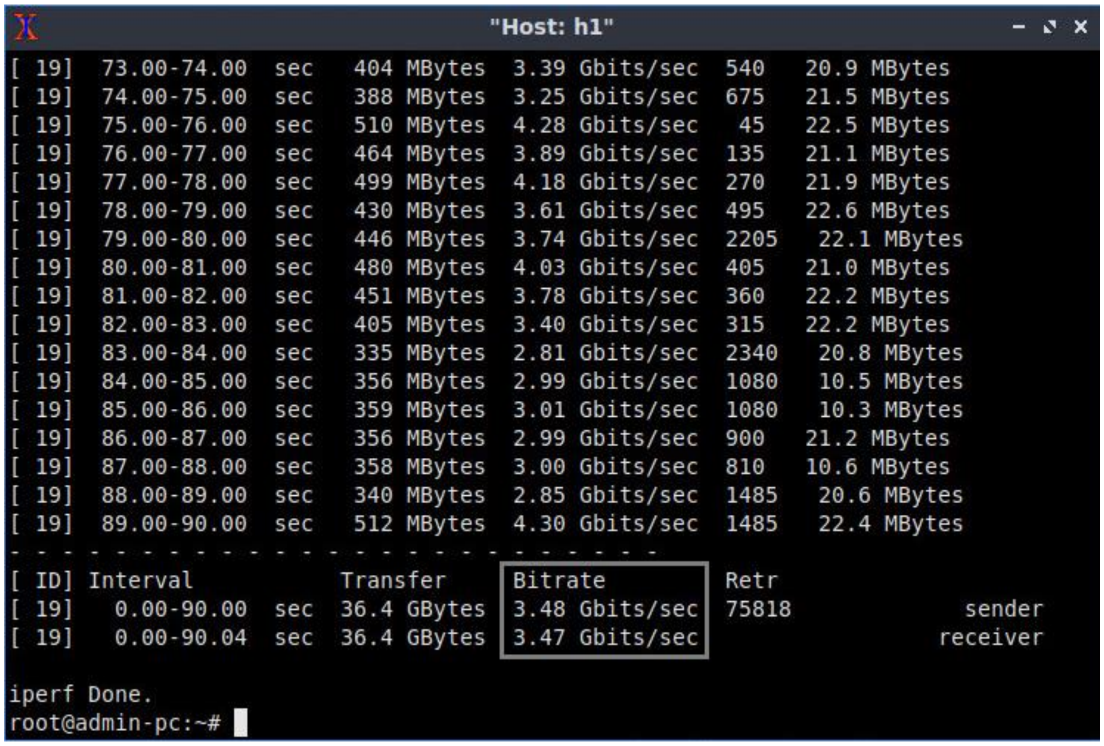
```
iperf3 -c 10.0.0.2 -t 90
```



```
"Host: h3"
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 38. Typing iPerf3 client command on host h3.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
"Host: h1"
[ 19]  73.00-74.00  sec   404 MBytes  3.39 Gbits/sec  540   20.9 MBytes
[ 19]  74.00-75.00  sec   388 MBytes  3.25 Gbits/sec  675   21.5 MBytes
[ 19]  75.00-76.00  sec   510 MBytes  4.28 Gbits/sec   45   22.5 MBytes
[ 19]  76.00-77.00  sec   464 MBytes  3.89 Gbits/sec  135   21.1 MBytes
[ 19]  77.00-78.00  sec   499 MBytes  4.18 Gbits/sec  270   21.9 MBytes
[ 19]  78.00-79.00  sec   430 MBytes  3.61 Gbits/sec  495   22.6 MBytes
[ 19]  79.00-80.00  sec   446 MBytes  3.74 Gbits/sec  2205   22.1 MBytes
[ 19]  80.00-81.00  sec   480 MBytes  4.03 Gbits/sec  405   21.0 MBytes
[ 19]  81.00-82.00  sec   451 MBytes  3.78 Gbits/sec  360   22.2 MBytes
[ 19]  82.00-83.00  sec   405 MBytes  3.40 Gbits/sec  315   22.2 MBytes
[ 19]  83.00-84.00  sec   335 MBytes  2.81 Gbits/sec  2340   20.8 MBytes
[ 19]  84.00-85.00  sec   356 MBytes  2.99 Gbits/sec  1080   10.5 MBytes
[ 19]  85.00-86.00  sec   359 MBytes  3.01 Gbits/sec  1080   10.3 MBytes
[ 19]  86.00-87.00  sec   356 MBytes  2.99 Gbits/sec  900   21.2 MBytes
[ 19]  87.00-88.00  sec   358 MBytes  3.00 Gbits/sec  810   10.6 MBytes
[ 19]  88.00-89.00  sec   340 MBytes  2.85 Gbits/sec  1485   20.6 MBytes
[ 19]  89.00-90.00  sec   512 MBytes  4.30 Gbits/sec  1485   22.4 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec  36.4 GBytes  3.48 Gbits/sec  75818             sender
[ 19]   0.00-90.04  sec  36.4 GBytes  3.47 Gbits/sec                  receiver

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

Figure 39. Running iPerf3 client on host h1.

The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 3.48 Gbps (sender) and 3.47 Gbps (receiver), and the number of retransmissions is 75818. Note that the congestion control algorithm used in host h1 is *bbr* and in host h3 is *cubic*.

Step 7. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

4 Testing throughput with one BDP switch's buffer size

In this section, you are going to change the switch S1 buffer size to one BDP (26,214,400) using the Token Bucket Filter (`tbf`). Then, you will test the throughput between host h1 and host h2 while there is another TCP flow between host h3 and host h4. On each test, you will modify the congestion control algorithm in host h1 namely, *cubic*, *reno* and *bbr*. The congestion control algorithm will still *cubic* in host 3 for all tests. In this section, the `tbf` limit value will be set to one BDP = 26,214,400 bytes.

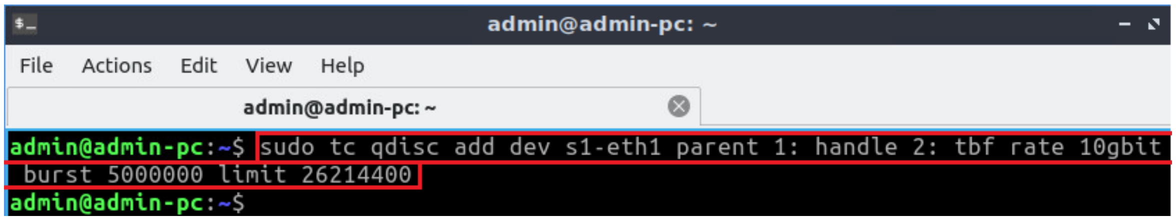
4.1 Changing switch S1's buffer size to one BDP

Step 1. Apply `tbf` rate limiting rule on switch S1's *s1-eth1* interface. In the client's terminal, type the command below. When prompted for a password, type `password` and hit *Enter*.

- `rate`: 10gbit

- `burst`: 5,000,000
- `limit`: 26,214,400

```
sudo tc qdisc change dev s1-eth1 parent 1: handle 2: tbf rate 10gbit burst
5000000 limit 26214400
```



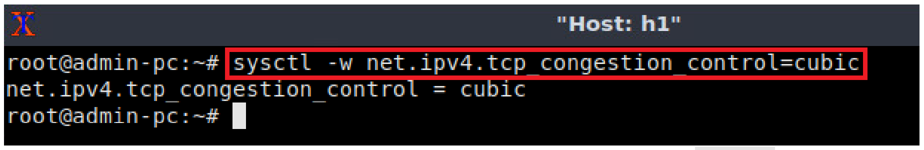
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth1 parent 1: handle 2: tbf rate 10gbit
 burst 5000000 limit 26214400
admin@admin-pc:~$
```

Figure 40. Changing the buffer size to one BDP on switch S1's *s1-eth1* interface.

4.2 TCP Cubic

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Cubic by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=cubic
```



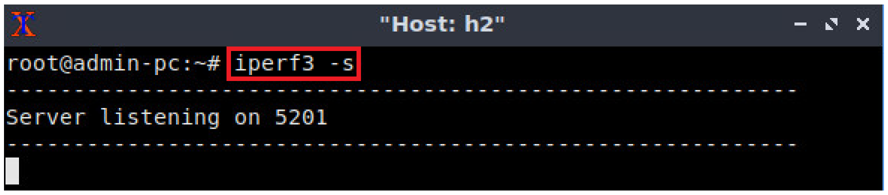
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=cubic
net.ipv4.tcp_congestion_control = cubic
root@admin-pc:~#
```

Figure 41. Changing TCP congestion control algorithm to `cubic` in host h1.

Note that host h3's congestion control algorithm is cubic by default.

Step 2. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

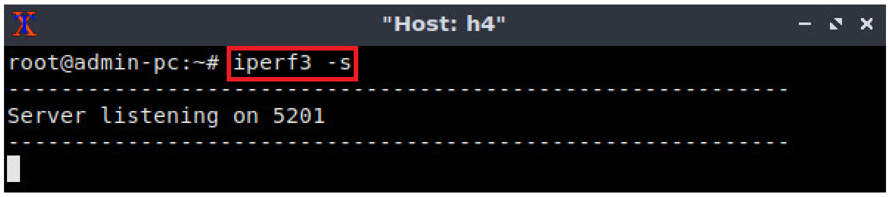


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 42. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



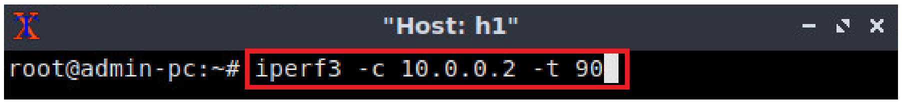
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 43. Starting iPerf3 server on host h4.

The following two steps should be executed almost simultaneously, thus you will type the commands displayed in Step 3 and Step 4, then in Step 5 you will execute them.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

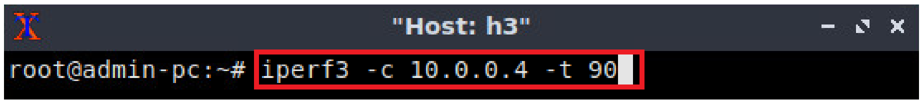


```
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 44. Typing iPerf3 client command on host h1.

Step 5. Type the following iPerf3 command in host h3's terminal without executing it.

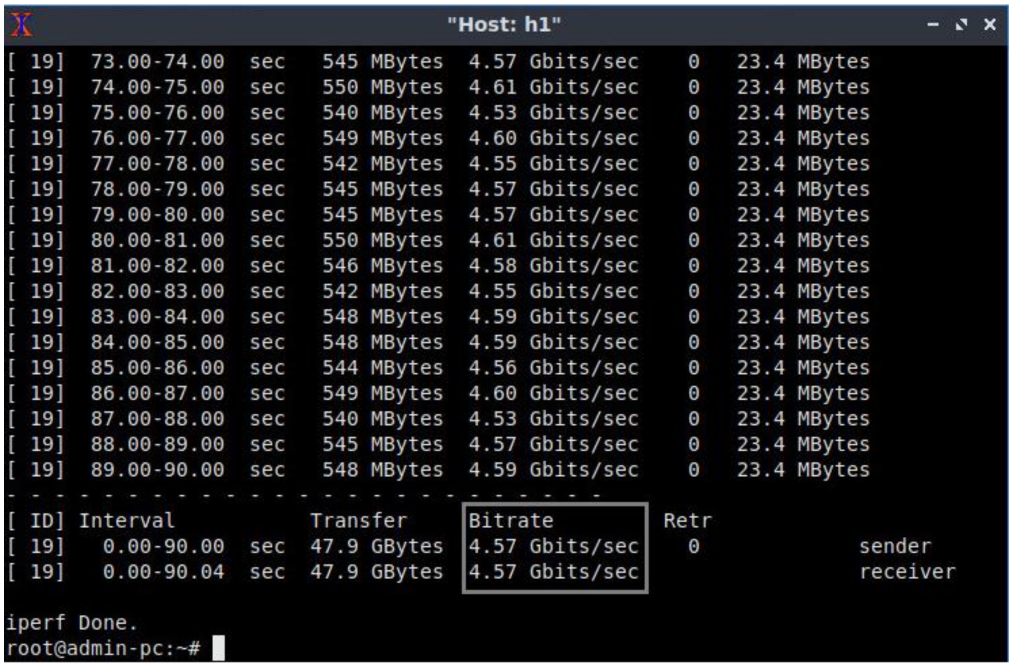
```
iperf3 -c 10.0.0.2 -t 90
```



```
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 45. Typing iPerf3 client command on host h3.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
[ 19]  73.00-74.00  sec   545 MBytes  4.57 Gbits/sec    0   23.4 MBytes
[ 19]  74.00-75.00  sec   550 MBytes  4.61 Gbits/sec    0   23.4 MBytes
[ 19]  75.00-76.00  sec   540 MBytes  4.53 Gbits/sec    0   23.4 MBytes
[ 19]  76.00-77.00  sec   549 MBytes  4.60 Gbits/sec    0   23.4 MBytes
[ 19]  77.00-78.00  sec   542 MBytes  4.55 Gbits/sec    0   23.4 MBytes
[ 19]  78.00-79.00  sec   545 MBytes  4.57 Gbits/sec    0   23.4 MBytes
[ 19]  79.00-80.00  sec   545 MBytes  4.57 Gbits/sec    0   23.4 MBytes
[ 19]  80.00-81.00  sec   550 MBytes  4.61 Gbits/sec    0   23.4 MBytes
[ 19]  81.00-82.00  sec   546 MBytes  4.58 Gbits/sec    0   23.4 MBytes
[ 19]  82.00-83.00  sec   542 MBytes  4.55 Gbits/sec    0   23.4 MBytes
[ 19]  83.00-84.00  sec   548 MBytes  4.59 Gbits/sec    0   23.4 MBytes
[ 19]  84.00-85.00  sec   548 MBytes  4.59 Gbits/sec    0   23.4 MBytes
[ 19]  85.00-86.00  sec   544 MBytes  4.56 Gbits/sec    0   23.4 MBytes
[ 19]  86.00-87.00  sec   549 MBytes  4.60 Gbits/sec    0   23.4 MBytes
[ 19]  87.00-88.00  sec   540 MBytes  4.53 Gbits/sec    0   23.4 MBytes
[ 19]  88.00-89.00  sec   545 MBytes  4.57 Gbits/sec    0   23.4 MBytes
[ 19]  89.00-90.00  sec   548 MBytes  4.59 Gbits/sec    0   23.4 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec  47.9 GBytes  4.57 Gbits/sec    0             sender
[ 19]   0.00-90.04  sec  47.9 GBytes  4.57 Gbits/sec                  receiver

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

Figure 46. Running iPerf3 client on host h1.

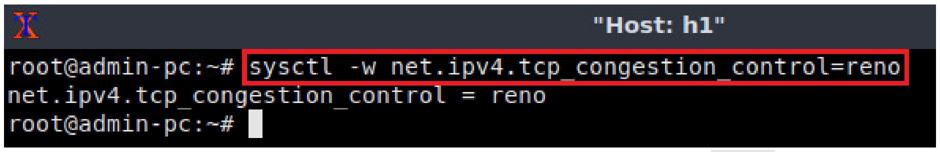
The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 4.57 Gbps (sender) and 4.57 Gbps (receiver), and the number of retransmissions is 0. Note that the congestion avoidances algorithm used in host h1 and host h2 is *cubic*. Similar results are found in host h3 terminal.

Step 7. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

4.3 TCP Reno

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Reno by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=reno
```



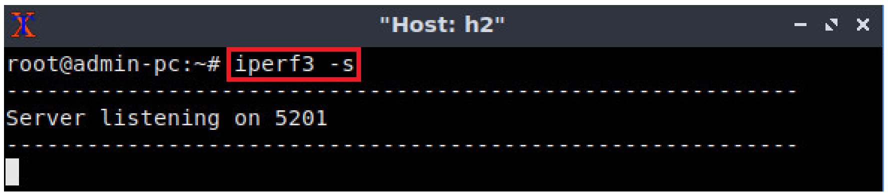
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
root@admin-pc:~#
```

Figure 47. Changing TCP congestion control algorithm to `reno` in host h1.

Note that host h3's congestion control algorithm is cubic by default.

Step 2. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

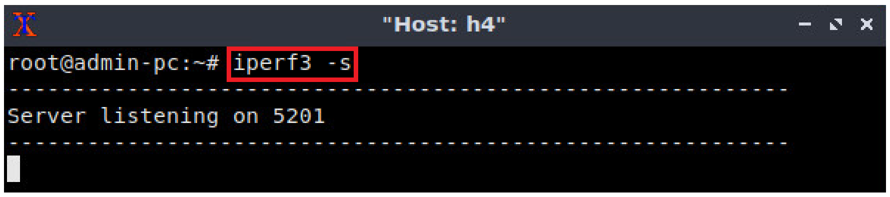


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 48. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



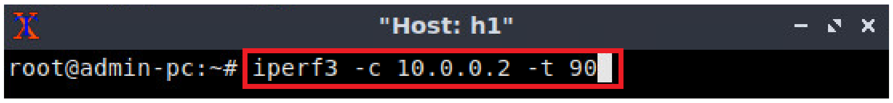
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 49. Starting iPerf3 server on host h4.

The following two steps should be executed almost simultaneously, thus you will type the commands displayed in Step 3 and Step 4, then in Step 5 you will execute them.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

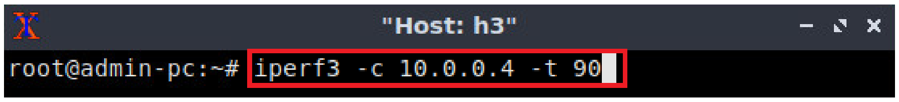


```
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 50. Typing iPerf3 client command on host h1.

Step 5. Type the following iPerf3 command in host h3's terminal without executing it.

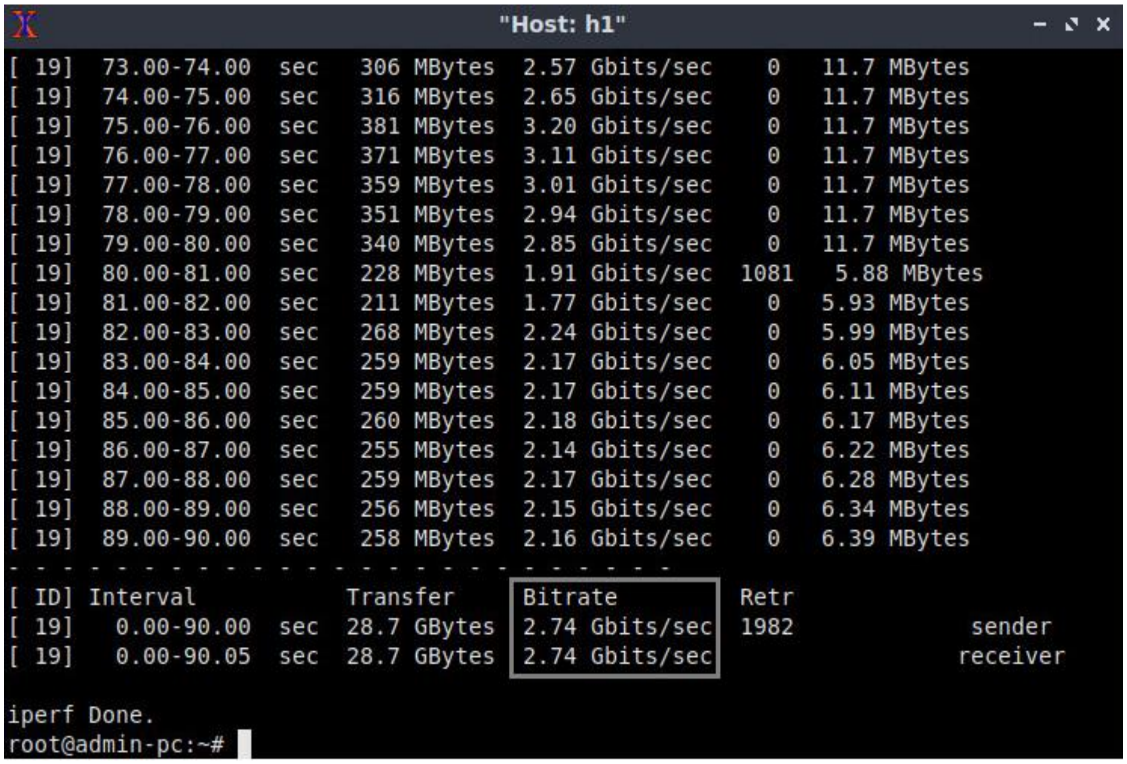
```
iperf3 -c 10.0.0.2 -t 90
```



```
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 51. Typing iPerf3 client command on host h3.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
[ 19]  73.00-74.00  sec   306 MBytes  2.57 Gbits/sec    0   11.7 MBytes
[ 19]  74.00-75.00  sec   316 MBytes  2.65 Gbits/sec    0   11.7 MBytes
[ 19]  75.00-76.00  sec   381 MBytes  3.20 Gbits/sec    0   11.7 MBytes
[ 19]  76.00-77.00  sec   371 MBytes  3.11 Gbits/sec    0   11.7 MBytes
[ 19]  77.00-78.00  sec   359 MBytes  3.01 Gbits/sec    0   11.7 MBytes
[ 19]  78.00-79.00  sec   351 MBytes  2.94 Gbits/sec    0   11.7 MBytes
[ 19]  79.00-80.00  sec   340 MBytes  2.85 Gbits/sec    0   11.7 MBytes
[ 19]  80.00-81.00  sec   228 MBytes  1.91 Gbits/sec  1081    5.88 MBytes
[ 19]  81.00-82.00  sec   211 MBytes  1.77 Gbits/sec    0   5.93 MBytes
[ 19]  82.00-83.00  sec   268 MBytes  2.24 Gbits/sec    0   5.99 MBytes
[ 19]  83.00-84.00  sec   259 MBytes  2.17 Gbits/sec    0   6.05 MBytes
[ 19]  84.00-85.00  sec   259 MBytes  2.17 Gbits/sec    0   6.11 MBytes
[ 19]  85.00-86.00  sec   260 MBytes  2.18 Gbits/sec    0   6.17 MBytes
[ 19]  86.00-87.00  sec   255 MBytes  2.14 Gbits/sec    0   6.22 MBytes
[ 19]  87.00-88.00  sec   259 MBytes  2.17 Gbits/sec    0   6.28 MBytes
[ 19]  88.00-89.00  sec   256 MBytes  2.15 Gbits/sec    0   6.34 MBytes
[ 19]  89.00-90.00  sec   258 MBytes  2.16 Gbits/sec    0   6.39 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec  28.7 GBytes  2.74 Gbits/sec  1982             sender
[ 19]   0.00-90.05  sec  28.7 GBytes  2.74 Gbits/sec                  receiver

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

Figure 52. Running iPerf3 client on host h1.

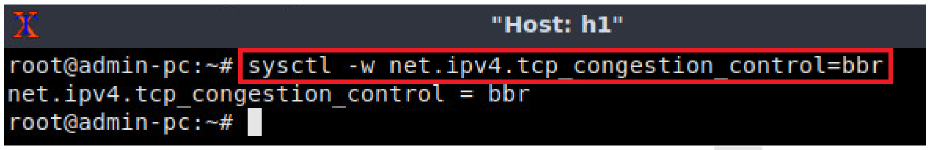
The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 2.74 Gbps (sender) and 2.74 Gbps (receiver), and the number of retransmissions is 1982. Note that the congestion avoidances algorithm used in host h1 is *reno* and in host h2 is *cubic*. Host h3's results are similar to the figure above, however we are just focused on host h1's results.

Step 7. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

4.4 TCP BBR

Step 1. In host h1's terminal, change the TCP congestion control algorithm to BBR by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=bbr
```



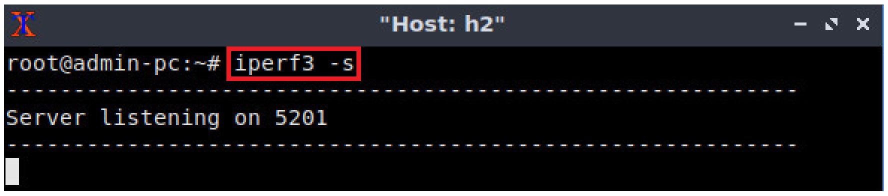
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=bbr
net.ipv4.tcp_congestion_control = bbr
root@admin-pc:~#
```

Figure 53. Changing TCP congestion control algorithm to `bbr` in host h1.

Note that host h3's congestion control algorithm is cubic by default.

Step 2. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

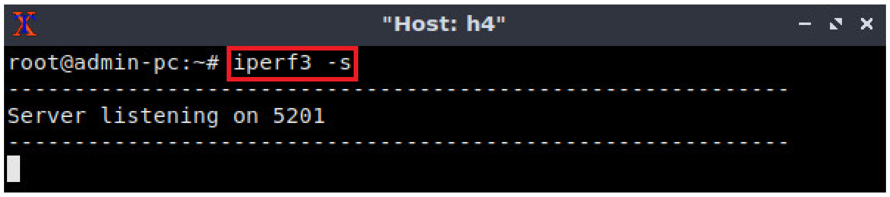


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 54. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



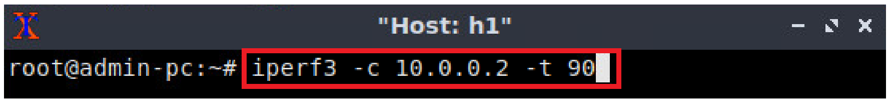
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 55. Starting iPerf3server on host h4.

The following two steps should be executed almost simultaneously thus, you will type the commands displayed in Step 3 and Step 4 then, in Step 5 you will execute them.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

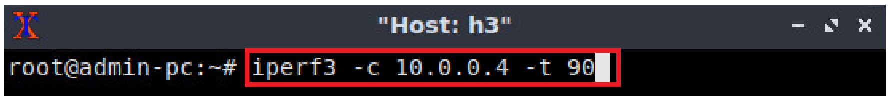


```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 56. Typing iPerf3 client command on host h1.

Step 5. Type the following iPerf3 command in host h3's terminal without executing it.

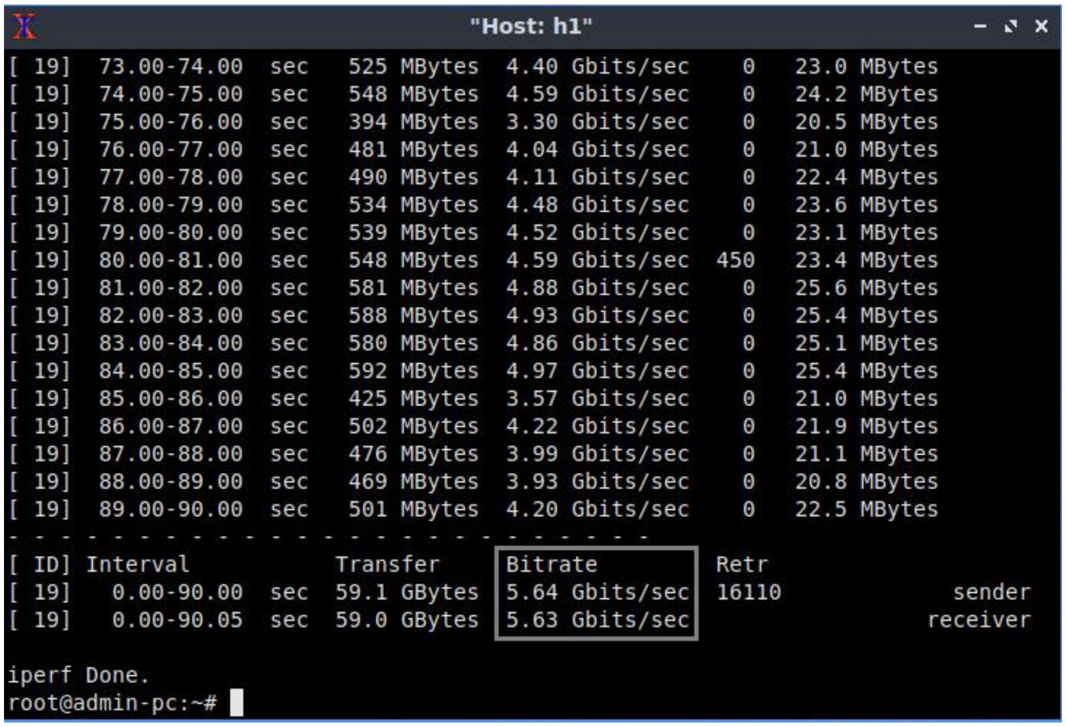
```
iperf3 -c 10.0.0.2 -t 90
```



```
"Host: h3"
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 57. Typing iPerf3 client command on host h3.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
"Host: h1"
[ 19]  73.00-74.00  sec   525 MBytes  4.40 Gbits/sec    0   23.0 MBytes
[ 19]  74.00-75.00  sec   548 MBytes  4.59 Gbits/sec    0   24.2 MBytes
[ 19]  75.00-76.00  sec   394 MBytes  3.30 Gbits/sec    0   20.5 MBytes
[ 19]  76.00-77.00  sec   481 MBytes  4.04 Gbits/sec    0   21.0 MBytes
[ 19]  77.00-78.00  sec   490 MBytes  4.11 Gbits/sec    0   22.4 MBytes
[ 19]  78.00-79.00  sec   534 MBytes  4.48 Gbits/sec    0   23.6 MBytes
[ 19]  79.00-80.00  sec   539 MBytes  4.52 Gbits/sec    0   23.1 MBytes
[ 19]  80.00-81.00  sec   548 MBytes  4.59 Gbits/sec  450   23.4 MBytes
[ 19]  81.00-82.00  sec   581 MBytes  4.88 Gbits/sec    0   25.6 MBytes
[ 19]  82.00-83.00  sec   588 MBytes  4.93 Gbits/sec    0   25.4 MBytes
[ 19]  83.00-84.00  sec   580 MBytes  4.86 Gbits/sec    0   25.1 MBytes
[ 19]  84.00-85.00  sec   592 MBytes  4.97 Gbits/sec    0   25.4 MBytes
[ 19]  85.00-86.00  sec   425 MBytes  3.57 Gbits/sec    0   21.0 MBytes
[ 19]  86.00-87.00  sec   502 MBytes  4.22 Gbits/sec    0   21.9 MBytes
[ 19]  87.00-88.00  sec   476 MBytes  3.99 Gbits/sec    0   21.1 MBytes
[ 19]  88.00-89.00  sec   469 MBytes  3.93 Gbits/sec    0   20.8 MBytes
[ 19]  89.00-90.00  sec   501 MBytes  4.20 Gbits/sec    0   22.5 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec  59.1 GBytes  5.64 Gbits/sec  16110             sender
[ 19]   0.00-90.05  sec  59.0 GBytes  5.63 Gbits/sec                  receiver

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

Figure 58. Running iPerf3 client on host h1.

The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 5.64 Gbps (sender) and 5.63 Gbps (receiver), and the number of retransmissions is 16,110. Note that the congestion avoidances algorithm used in host h1 is *bbr* and in host h3 is *cubic*. Host h3's results are similar to the figure above, however we are just focused on host h1's results.

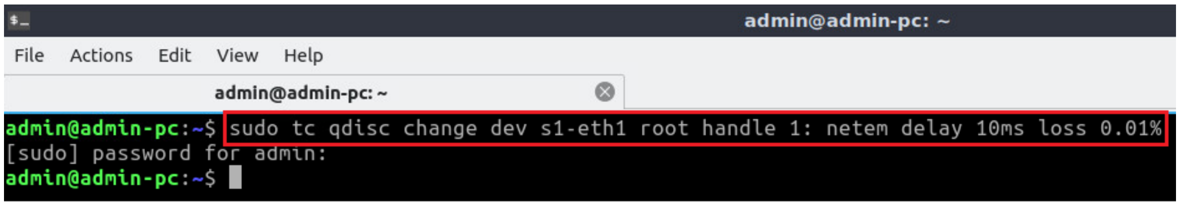
Step 7. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

5 Emulating high-latency WAN with packet loss

This section emulates a high-latency WAN with packet loss. We already have set a 20ms RTT on the switches. Now, you will add 0.01% packet loss on the switch S1. Note that the switch S1's buffer size is set to one BDP.

Step 1. In the terminal, type the command below. When prompted for a password, type `password` and hit *Enter*. This command introduces 0.01% packet loss on switch S1's *s1-eth1* interface.

```
sudo tc qdisc change dev s1-eth1 root handle 1: netem delay 10ms loss 0.01%
```



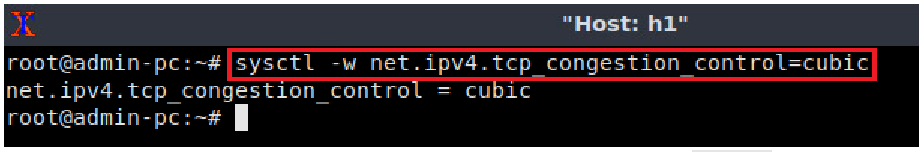
```
admin@admin-pc:~$ sudo tc qdisc change dev s1-eth1 root handle 1: netem delay 10ms loss 0.01%
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 59. Adding delay of 0.01% to switch S1's *s1-eth1* interface.

5.1 TCP Cubic

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Cubic by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=cubic
```



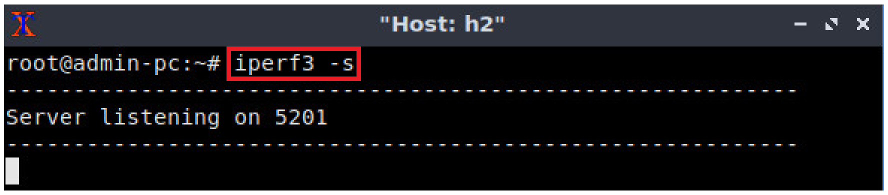
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=cubic
net.ipv4.tcp_congestion_control = cubic
root@admin-pc:~#
```

Figure 60. Changing TCP congestion control algorithm to `cubic` in host h1.

Note that host h3's congestion control algorithm is Cubic by default.

Step 2. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

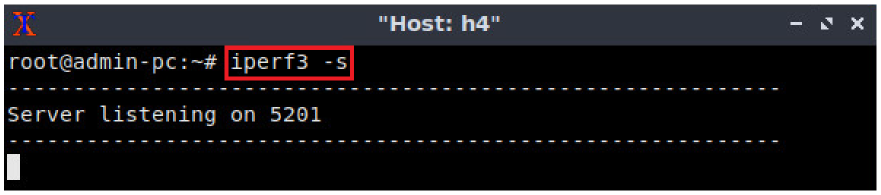


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 61. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



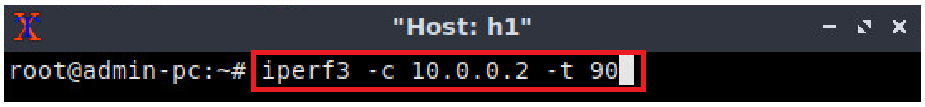
```
"Host: h4"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 62. Starting iPerf3 server on host h4.

The following two steps should be executed almost simultaneously thus, you will type the commands displayed in Step 3 and Step 4 then, in Step 5 you will execute them.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

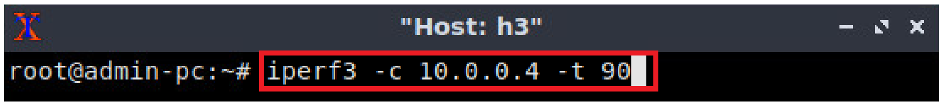


```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 63. Typing iPerf3 client command on host h1.

Step 5. Type the following iPerf3 command in host h3's terminal without executing it.

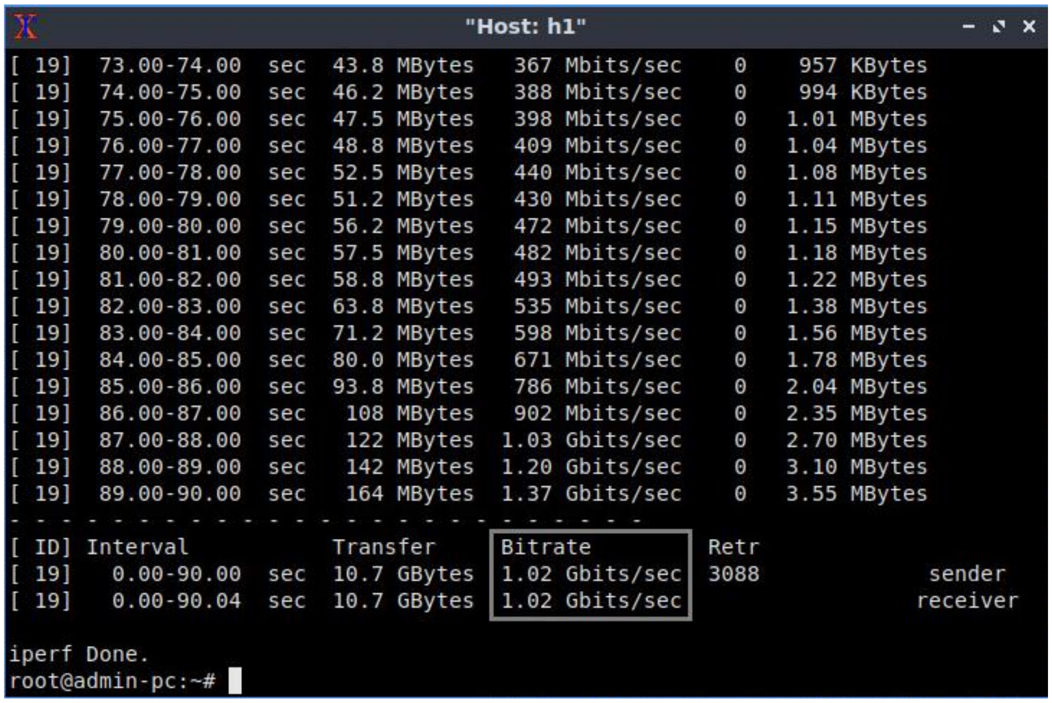
```
iperf3 -c 10.0.0.2 -t 90
```



```
"Host: h3"
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 64. Typing iPerf3 client command on host h3.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
"Host: h1"
[ 19]  73.00-74.00  sec  43.8 MBytes   367 Mbits/sec    0    957 KBytes
[ 19]  74.00-75.00  sec  46.2 MBytes   388 Mbits/sec    0    994 KBytes
[ 19]  75.00-76.00  sec  47.5 MBytes   398 Mbits/sec    0   1.01 MBytes
[ 19]  76.00-77.00  sec  48.8 MBytes   409 Mbits/sec    0   1.04 MBytes
[ 19]  77.00-78.00  sec  52.5 MBytes   440 Mbits/sec    0   1.08 MBytes
[ 19]  78.00-79.00  sec  51.2 MBytes   430 Mbits/sec    0   1.11 MBytes
[ 19]  79.00-80.00  sec  56.2 MBytes   472 Mbits/sec    0   1.15 MBytes
[ 19]  80.00-81.00  sec  57.5 MBytes   482 Mbits/sec    0   1.18 MBytes
[ 19]  81.00-82.00  sec  58.8 MBytes   493 Mbits/sec    0   1.22 MBytes
[ 19]  82.00-83.00  sec  63.8 MBytes   535 Mbits/sec    0   1.38 MBytes
[ 19]  83.00-84.00  sec  71.2 MBytes   598 Mbits/sec    0   1.56 MBytes
[ 19]  84.00-85.00  sec  80.0 MBytes   671 Mbits/sec    0   1.78 MBytes
[ 19]  85.00-86.00  sec  93.8 MBytes   786 Mbits/sec    0   2.04 MBytes
[ 19]  86.00-87.00  sec   108 MBytes   902 Mbits/sec    0   2.35 MBytes
[ 19]  87.00-88.00  sec   122 MBytes  1.03 Gbits/sec    0   2.70 MBytes
[ 19]  88.00-89.00  sec   142 MBytes  1.20 Gbits/sec    0   3.10 MBytes
[ 19]  89.00-90.00  sec   164 MBytes  1.37 Gbits/sec    0   3.55 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec  10.7 GBytes  1.02 Gbits/sec  3088             sender
[ 19]   0.00-90.04  sec  10.7 GBytes  1.02 Gbits/sec                  receiver

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

Figure 65. Running iPerf3 client on host h1.

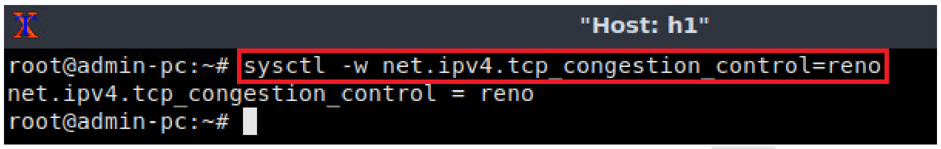
The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 1.02 Gbps (sender) and 1.02 Gbps (receiver), and the number of retransmissions is 3088. Note that the congestion control algorithm used in host h1 and host h2 is *cubic*. Host h3's results are similar to the figure above, however we are just focused on host h1's results.

Step 7. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

5.2 TCP Reno

Step 1. In host h1's terminal, change the TCP congestion control algorithm to Reno by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=reno
```



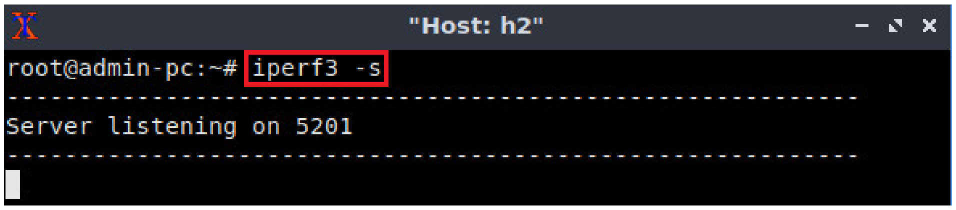
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno
root@admin-pc:~#
```

Figure 66. Changing TCP congestion control algorithm to `reno` in host h1.

Note that host h3's congestion control algorithm is cubic by default.

Step 2. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

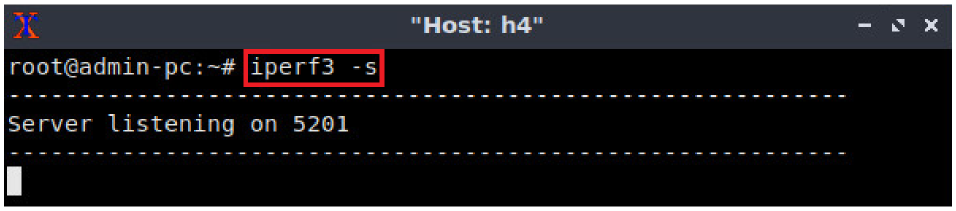


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 67. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



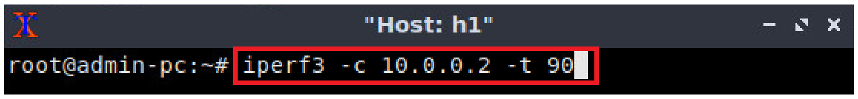
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 68. Starting iPerf3 server on host h4.

The following two steps should be executed almost simultaneously, thus you will type the commands displayed in Step 3 and Step 4, then in Step 5 you will execute them.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

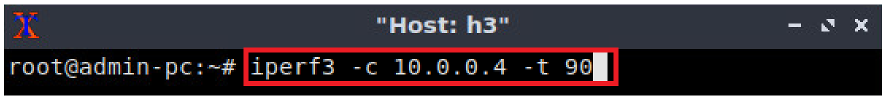


```
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 69. Typing iPerf3 client command on host h1.

Step 5. Type the following iPerf3 command in host h3's terminal without executing it.

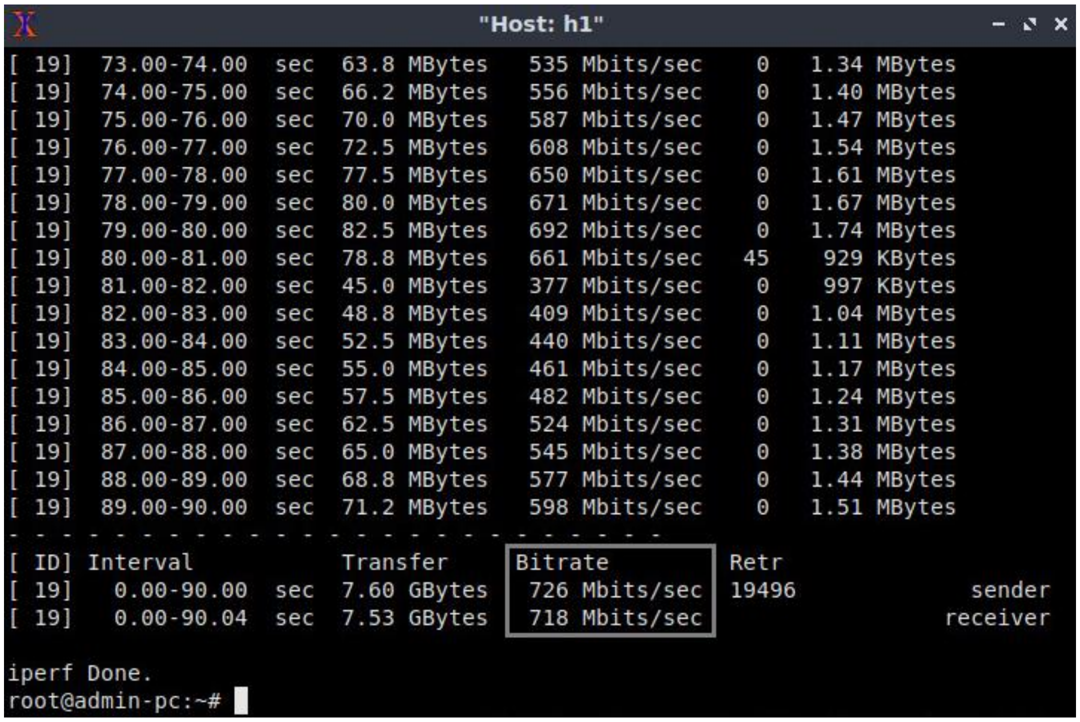
```
iperf3 -c 10.0.0.2 -t 90
```



```
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 70. Typing iPerf3 client command on host h3.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
"Host: h1"
[ 19]  73.00-74.00  sec  63.8 MBytes   535 Mbits/sec    0   1.34 MBytes
[ 19]  74.00-75.00  sec  66.2 MBytes   556 Mbits/sec    0   1.40 MBytes
[ 19]  75.00-76.00  sec  70.0 MBytes   587 Mbits/sec    0   1.47 MBytes
[ 19]  76.00-77.00  sec  72.5 MBytes   608 Mbits/sec    0   1.54 MBytes
[ 19]  77.00-78.00  sec  77.5 MBytes   650 Mbits/sec    0   1.61 MBytes
[ 19]  78.00-79.00  sec  80.0 MBytes   671 Mbits/sec    0   1.67 MBytes
[ 19]  79.00-80.00  sec  82.5 MBytes   692 Mbits/sec    0   1.74 MBytes
[ 19]  80.00-81.00  sec  78.8 MBytes   661 Mbits/sec   45    929 KBytes
[ 19]  81.00-82.00  sec  45.0 MBytes   377 Mbits/sec    0    997 KBytes
[ 19]  82.00-83.00  sec  48.8 MBytes   409 Mbits/sec    0   1.04 MBytes
[ 19]  83.00-84.00  sec  52.5 MBytes   440 Mbits/sec    0   1.11 MBytes
[ 19]  84.00-85.00  sec  55.0 MBytes   461 Mbits/sec    0   1.17 MBytes
[ 19]  85.00-86.00  sec  57.5 MBytes   482 Mbits/sec    0   1.24 MBytes
[ 19]  86.00-87.00  sec  62.5 MBytes   524 Mbits/sec    0   1.31 MBytes
[ 19]  87.00-88.00  sec  65.0 MBytes   545 Mbits/sec    0   1.38 MBytes
[ 19]  88.00-89.00  sec  68.8 MBytes   577 Mbits/sec    0   1.44 MBytes
[ 19]  89.00-90.00  sec  71.2 MBytes   598 Mbits/sec    0   1.51 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec  7.60 GBytes   726 Mbits/sec  19496             sender
[ 19]   0.00-90.04  sec  7.53 GBytes   718 Mbits/sec                  receiver

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

Figure 71. Running iPerf3 client on host h1.

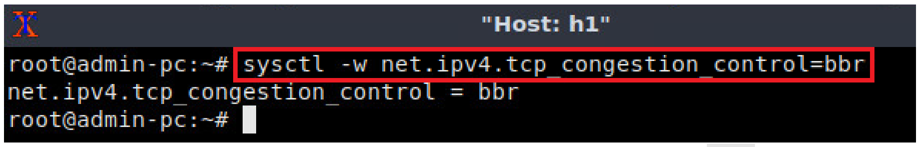
The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 726 Mbps (sender) and 718 Mbps (receiver), and the number of retransmissions is 19,496. Note that the congestion control algorithm used in host h1 is *reno* and in host h2 is *cubic*. Host h3's results are similar to the figure above, however we are just focused on host h1's results.

Step 7. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

5.3 TCP BBR

Step 1. In host h1's terminal, change the TCP congestion control algorithm to BBR by typing the following command:

```
sysctl -w net.ipv4.tcp_congestion_control=bbr
```



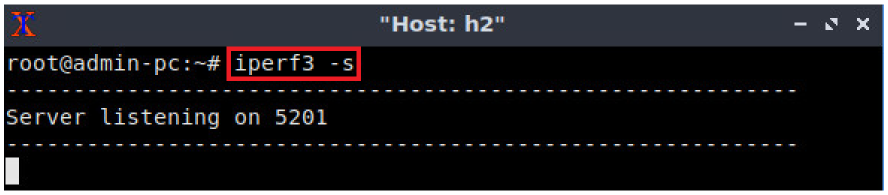
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_congestion_control=bbr
net.ipv4.tcp_congestion_control = bbr
root@admin-pc:~#
```

Figure 72. Changing TCP congestion control algorithm to `bbr` in host h1.

Note that host h3's congestion control algorithm is cubic by default.

Step 2. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

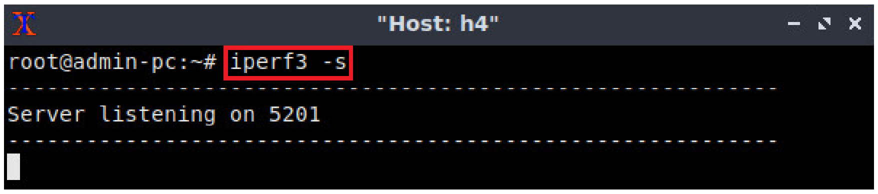


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 73. Starting iPerf3 server on host h2.

Step 3. Launch iPerf3 in server mode on host h4's terminal.

```
iperf3 -s
```



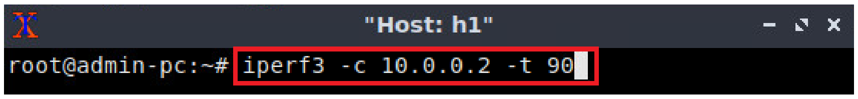
```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 74. Starting iPerf3 server on host h4.

The following two steps should be executed almost simultaneously, thus you will type the commands displayed in Step 3 and Step 4, then in Step 5 you will execute them.

Step 4. Type the following iPerf3 command in host h1's terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

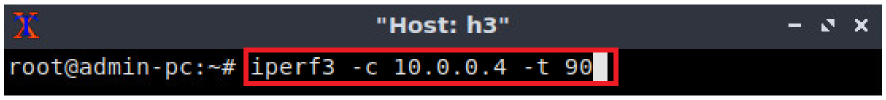


```
root@admin-pc:~# iperf3 -c 10.0.0.2 -t 90
```

Figure 75. Typing iPerf3 client command on host h1.

Step 5. Type the following iPerf3 command in host h3's terminal without executing it.

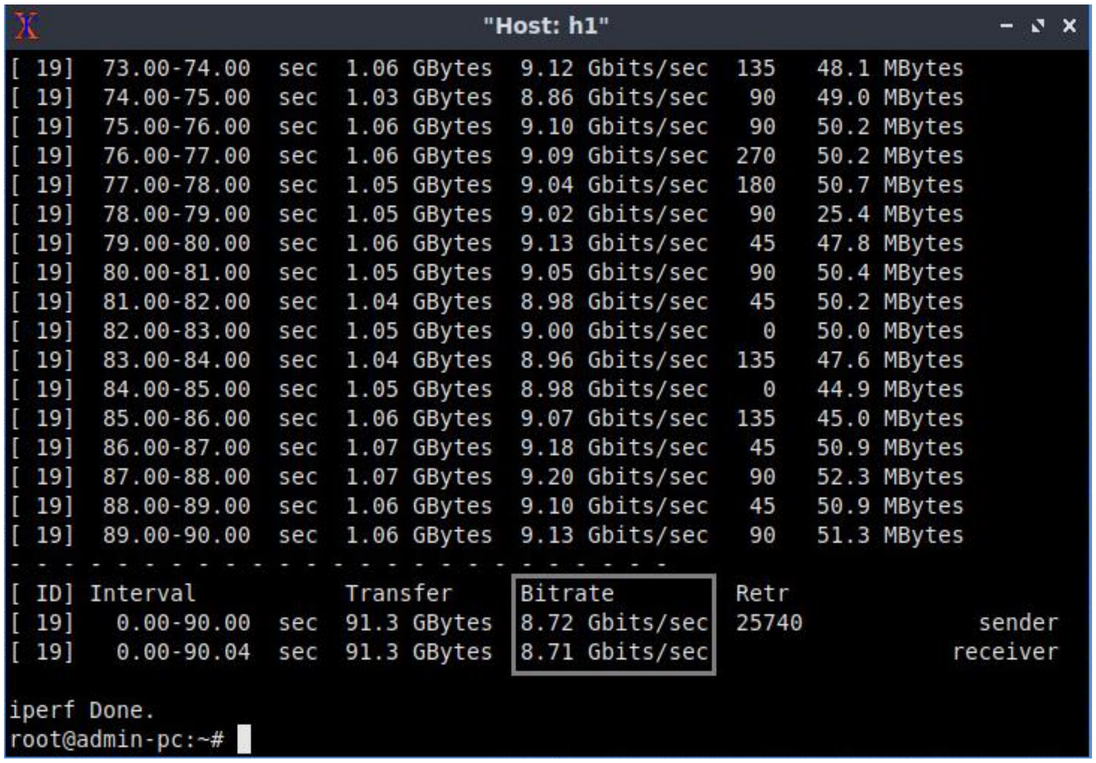
```
iperf3 -c 10.0.0.2 -t 90
```



```
root@admin-pc:~# iperf3 -c 10.0.0.4 -t 90
```

Figure 76. Typing iPerf3 client command on host h3.

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.



```
"Host: h1"
[ 19]  73.00-74.00  sec  1.06 GBytes  9.12 Gbits/sec  135   48.1 MBytes
[ 19]  74.00-75.00  sec  1.03 GBytes  8.86 Gbits/sec   90   49.0 MBytes
[ 19]  75.00-76.00  sec  1.06 GBytes  9.10 Gbits/sec   90   50.2 MBytes
[ 19]  76.00-77.00  sec  1.06 GBytes  9.09 Gbits/sec  270   50.2 MBytes
[ 19]  77.00-78.00  sec  1.05 GBytes  9.04 Gbits/sec  180   50.7 MBytes
[ 19]  78.00-79.00  sec  1.05 GBytes  9.02 Gbits/sec   90   25.4 MBytes
[ 19]  79.00-80.00  sec  1.06 GBytes  9.13 Gbits/sec   45   47.8 MBytes
[ 19]  80.00-81.00  sec  1.05 GBytes  9.05 Gbits/sec   90   50.4 MBytes
[ 19]  81.00-82.00  sec  1.04 GBytes  8.98 Gbits/sec   45   50.2 MBytes
[ 19]  82.00-83.00  sec  1.05 GBytes  9.00 Gbits/sec    0   50.0 MBytes
[ 19]  83.00-84.00  sec  1.04 GBytes  8.96 Gbits/sec  135   47.6 MBytes
[ 19]  84.00-85.00  sec  1.05 GBytes  8.98 Gbits/sec    0   44.9 MBytes
[ 19]  85.00-86.00  sec  1.06 GBytes  9.07 Gbits/sec  135   45.0 MBytes
[ 19]  86.00-87.00  sec  1.07 GBytes  9.18 Gbits/sec   45   50.9 MBytes
[ 19]  87.00-88.00  sec  1.07 GBytes  9.20 Gbits/sec   90   52.3 MBytes
[ 19]  88.00-89.00  sec  1.06 GBytes  9.10 Gbits/sec   45   50.9 MBytes
[ 19]  89.00-90.00  sec  1.06 GBytes  9.13 Gbits/sec   90   51.3 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 19]   0.00-90.00  sec  91.3 GBytes  8.72 Gbits/sec  25740             sender
[ 19]   0.00-90.04  sec  91.3 GBytes  8.71 Gbits/sec                  receiver

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

Figure 77. Running iPerf3 client on host h1.

The figure above shows the iPerf3 test output report by the last 20 seconds. The average achieved throughput is 8.72 Gbps (sender) and 8.71 Gbps (receiver), and the number of retransmissions is 25,740. Note that the congestion avoidances algorithm used in host h1 is *bbr* and in host h3 is *cubic*.

Step 7. In order to stop the server, press `Ctrl+c` in host h2's and host h4's terminals. The user can see the throughput results in the server side too.

This concludes Lab 11. Stop the emulation and then exit out of MiniEdit.

References

1. J. Moy, "Open shortest path first (OSPF) Version 2," Internet Request for Comments, RFC Editor, RFC 2328, Apr. 1998. [Online]. Available: https://www.ietf.org/rfc/rfc2328.txt.
2. Y. Rekhter, T. Li, S. Hares, "Border gateway protocol 4," Internet Request for Comments, RFC Editor, RFC 4271, Jan. 2006. [Online]. Available: https://tools.ietf.org/html/rfc4271.
3. J. Crichigno, E. Bou-Harb, N. Ghani, "A comprehensive tutorial on Science DMZ," IEEE Communications Surveys and Tutorials, 2019.
4. 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.
5. J. Kurose, K. Ross, "Computer networking: a top-down approach," 7th Edition, Pearson, 2017.

6. C. Villamizar, C. Song, "High performance TCP in ansnet," ACM Computer Communications Review, vol. 24, no. 5, pp. 45-60, Oct. 1994.
7. R. Bush, D. Meyer, "Some internet architectural guidelines and philosophy," Internet Request for Comments, RFC Editor, RFC 3439, Dec. 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3439.txt.
8. G. Appenzeller, I. Keslassy, N. McKeown, "Sizing router buffers," in Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 281-292, Oct. 2004.

NETWORK TOOLS AND PROTOCOLS

Lab 12: TCP Rate Control with Pacing

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 TCP pacing 4
1.1 TCP pacing essentials 4
1.2 Use case: TCP pacing on a 100 Gbps network 5
1.3 Fair queueing details 6
2 Lab topology........ 7
2.1 Starting host h1 and host h2 8
2.2 Emulating 10 Gbps high-latency WAN 9
2.3 Testing connection 10
3 Enabling TCP pacing with tc and fq........ 13
4 Enabling TCP pacing from application 15
5 Concurrent transmission without pacing 17
6 Concurrent transmission with pacing 19
7 Parallel streams and without pacing 21
8 Parallel streams and with pacing 23
References 25

Overview

This lab introduces TCP pacing, which is a technique that evenly spaces out packets and minimizes traffic burstiness and packet losses. The focus in this lab is on Fair Queueing (FQ)-based pacing in high-latency Wide Area Networks (WANs). The lab describes the steps to conduct throughput tests that encompass TCP pacing and to compare the performance of TCP pacing against regular (non-paced) TCP.

Objectives

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

1. Define TCP pacing.
2. Understand FQ-based pacing.
3. Enable TCP pacing in Linux.
4. Compare the performance of paced TCP vs. non-paced TCP.
5. Understand pacing effect on parallel streams.
6. Emulate a WAN and calculate the coefficient of variation of flows.

Lab settings

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

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

Device	Account	Password
Client1	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction to TCP pacing.
2. Section 2: Lab topology.
3. Section 3: Enabling TCP pacing with tc and fq.
4. Section 4: Enabling TCP pacing from application.
5. Section 5: Concurrent transmission without pacing.
6. Section 6: Concurrent transmission with pacing.
7. Section 7: Parallel streams and without pacing.
8. Section 8: Parallel streams and with pacing.

1 Introduction to TCP pacing

1.1 TCP pacing essentials

Data transmission can be bursty, resulting in packets being buffered at routers and switches and dropped at times. End devices can contribute to the problem by sending a large number of packets in a short period of time. If those packets were transmitted at a steady pace, the formation of queues could be reduced, avoiding packet losses.

TCP pacing is a technique by which a transmitter evenly spaces or paces packets at a pre-configured rate. It has been applied for years in enterprise networks[1], with mixed results. However, its recent application to data transfers in high-throughput high-latency networks and science demilitarized zones (Science DMZs) suggests that its use has several advantages[2]. TCP pacing has also been applied to datacenter environments[3].

The existing TCP congestion control algorithms, except for BBR[4], indicate how much data is allowed for transmission. Those algorithms do not provide a time period over which that data should be transmitted and how the data should be spread to mitigate potential bursts. The rate, however, can be enforced by a packet scheduler such as a fair queue (FQ)[5]. The packet scheduler organizes the flow of packets of each TCP connection through the network stack to meet policy objectives. Some Linux distributions such as CentOS[6] implement FQ scheduling in conjunction with TCP pacing[4, 7].

FQ is intended for locally generated traffic (e.g., a sender device, such as data transfer node (DTN) in Science DMZs). Figure 1 illustrates the operation of FQ pacing. Application 1 generates green packets, and application 2 generates blue packets. Each application opens a TCP connection. FQ paces each connection according to the desired rate, evenly spacing out packets within an application based on the desired rate. The periods T_1 and T_2 represent the time-space used for connections 1 and 2 respectively.

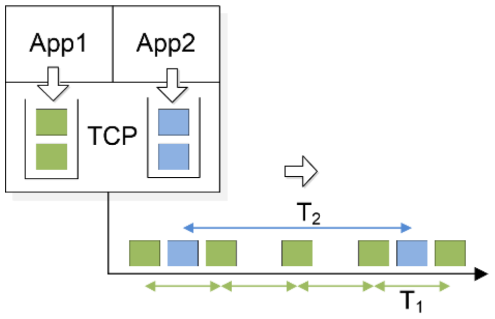


Figure 1. TCP pacing. Packets of applications 1 and 2 are evenly spaced by T_1 and T_2 time units.

TCP pacing reduces the typical TCP sawtooth behavior[8] and is effective when there are rate mismatches along the path between the sender and the receiver. This is the case, for example, when the ingress port of a router has a capacity of 100 Gbps, and the egress port has a capacity of 10 Gbps. Because of the TCP congestion control mechanism, the sawtooth behavior always emerges. As TCP continues to increase the size of the congestion window, eventually the bottleneck link becomes full while the rest of the links become underutilized. These mismatches produce a continuous circle of additive increases and multiplicative decreases[8].

1.2 Use case: TCP pacing on a 100 Gbps network

With the increase of big data transfers across networks, network professionals have recently explored the impact of pacing on large flows[8]. Figure 2(a) shows the results of data transfers over the Energy Science Network (ESnet). ESnet is a high-performance network that carries science traffic for the U.S. Department of Energy. As of 2018, this network is transporting more than 200 petabytes per month. The path capacity and round-trip time (RTT) between end devices, referred to as DTNs, are 100 Gbps and 92 milliseconds respectively. Transfers use TCP Cubic congestion control algorithm[9] without pacing and a maximum segment size (MSS) of 1,500 bytes. Four concurrent TCP connections are generated from a single source DTN to a single destination DTN. These four connections exhibit the typical sawtooth behavior[10], which in part is attributed to the inability of switches to absorb traffic bursts. Figure 2(b) shows the behavior of TCP Cubic with FQ pacing. The pacing rate for the four TCP connections is approximately 20 Gbps (curves are overlapped at nearly 20 Gbps). The throughput is slightly lower than 20 Gbps per connection. However, notice how the sawtooth behavior is reduced and stable rates are obtained.

In general, TCP FQ pacing is also effective when there are rate mismatches along the path between the sender and the receiver. This is the case, for example, when the ingress port of a router has a capacity of 100 Gbps and the egress port has a capacity of 10 Gbps. As TCP increases the congestion window during the additive increase phase, eventually the bottleneck link becomes full while the rest of the links become underutilized. The mismatches produce a continuous circle of additive increases and multiplicative decreases, thus generating the sawtooth behavior.

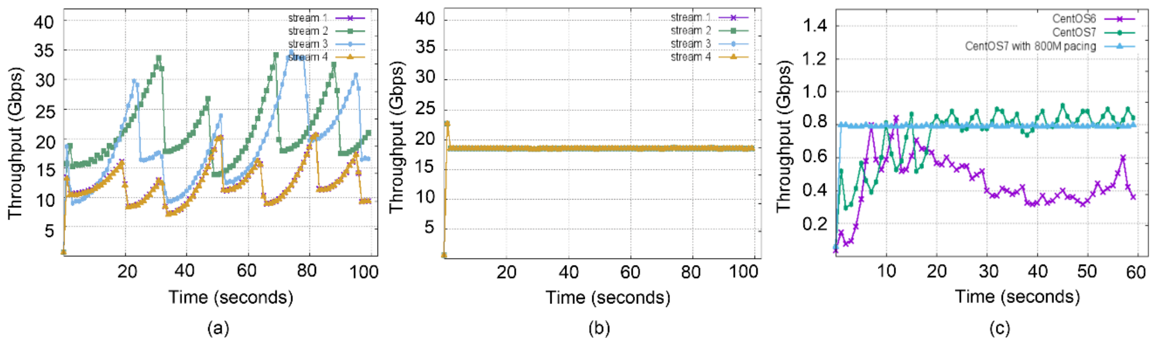


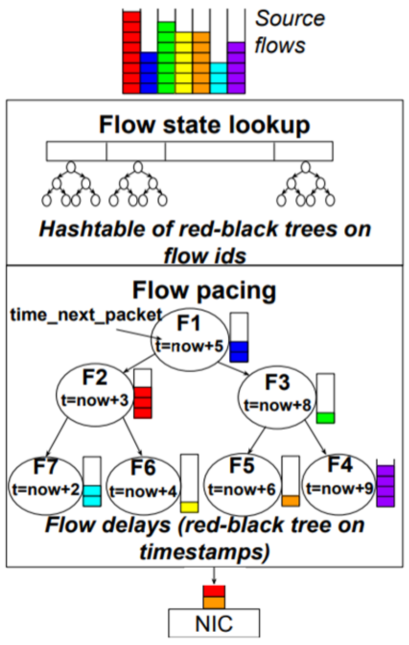
Figure 2. Impact of TCP pacing on throughput. (a) Data transfers of four parallel TCP connections across a 100 Gbps, 92 milliseconds RTT path. (b) The same data transfer as in (a) but using TCP pacing. (c) Data transfers between two DTNs connected by a path with a bottleneck link of 1 Gbps. The curves show the performance when the DTNs use different Linux operating systems (violet: CentOS 6; green: CentOS 7, and blue: CentOS7 with pacing). The results are reproduced from[8].

Figure 2(c) shows the data transfer between two DTNs over ESnet. One DTN is in Amarillo, Texas, and the other DTN is in New York City. Although the WAN connecting the two sites has 100 Gbps capacity, one of the DTNs is attached to the network via a 1 Gbps network interface card. Thus, the entirety of the path includes multiple 100 Gbps links and one bottleneck link of 1 Gbps. The figure shows three curves: the throughput when both DTNs are based on Linux CentOS[6] Version 6 (violet), the throughput when DTNs are based on Linux CentOS Version 7 (green), and the throughput when DTNs are based on Linux CentOS Version 7 and packets are paced at 800 Mbps (blue). Note that pacing also leads to much more stable behaviors, almost removing the TCP sawtooth behavior.

1.3 Fair queueing details

In Linux-based systems, network traffic can be controlled by Queueing Disciplines (*qdisc*) used in conjunction with the Traffic Control (*tc*) tool. In this lab we focus on the most commonly used queueing discipline: FQ. In this queueing discipline, aggregate queues are used to associate token buckets in order to limit the transmission rate.

FQ performs flow separation to achieve pacing; it is designed to follow the requirements set by the TCP stack[5]. Generally, a flow is considered all packets pertaining to a particular socket. FQ uses the red-black tree data structure to index and track the state of single flows as shown in Figure 3(a)[11]. A red-black tree is a binary search tree which ensures that no path in the tree is more than twice long as any other. This property ensures that tree operations have a logarithmic complexity. FQ achieves fairness through the Deficit Round Robin (DRR) algorithm[12], illustrated in Figure 3(b). The DRR is an algorithm that allows each flow passing through a network device to have a nearly perfect fairness and requires only a constant number of operations per packet. FQ uses the leaky bucket queue where transmitting timestamps (indexed on the read-black tree) are derived from the pacing rate specified by the user and the packet size. FQ is a non-work conserving scheduler, therefore, it can have idle scheduled resources even if there are jobs ready to be scheduled.



(a)

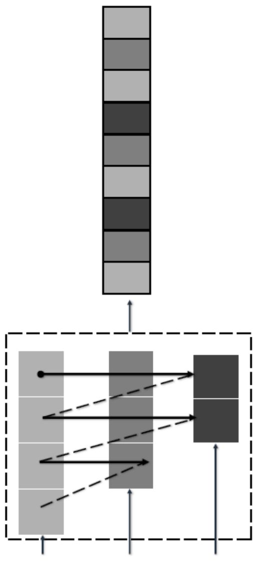

(b)

Figure 3. (a) FQ-pacing. (b) Deficit Round-Robin (DRR) algorithm.

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.

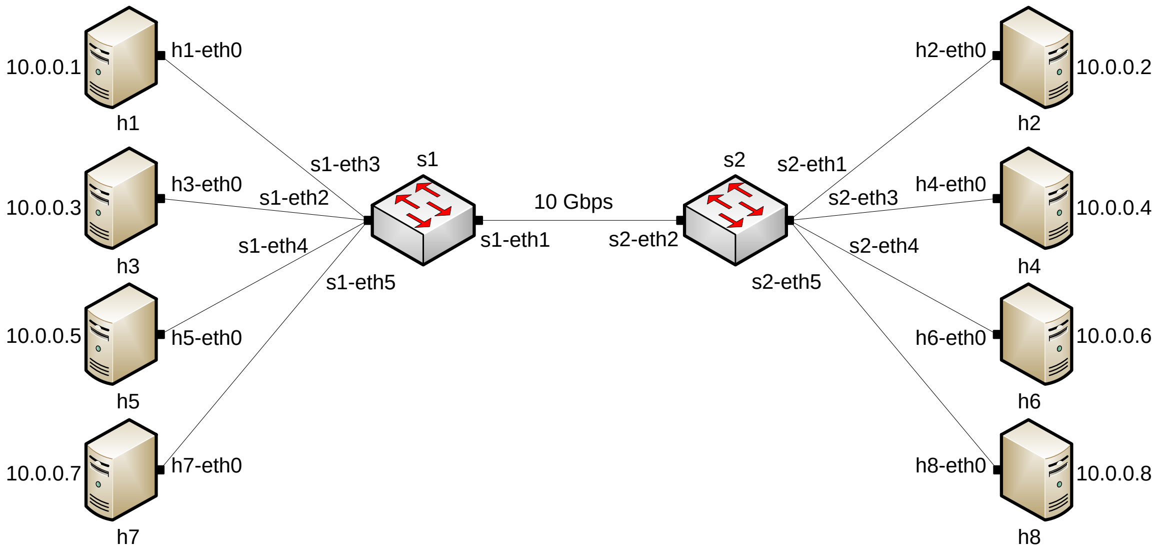


Figure 4. Lab topology.

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`.

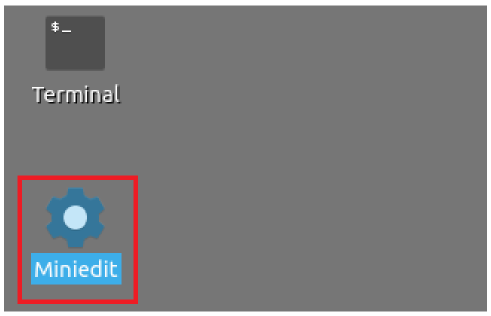


Figure 5. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 12.mn* topology file and click on *Open*.

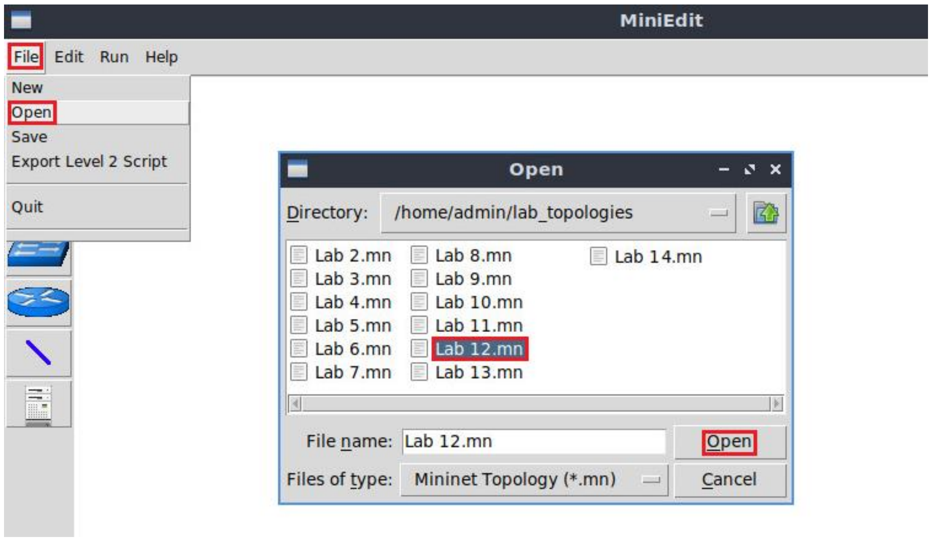


Figure 6. MiniEdit's *Open* dialog.

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.

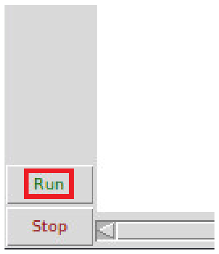


Figure 7. Running the emulation.

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.

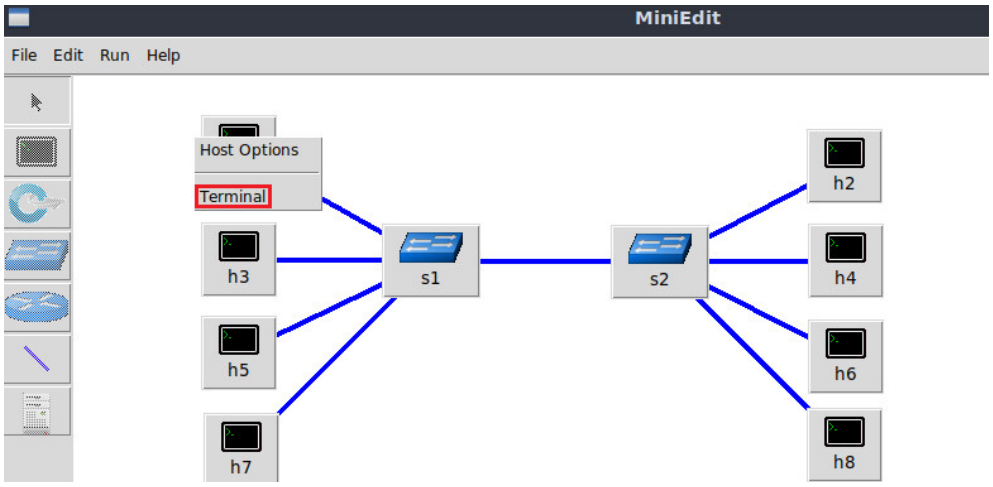
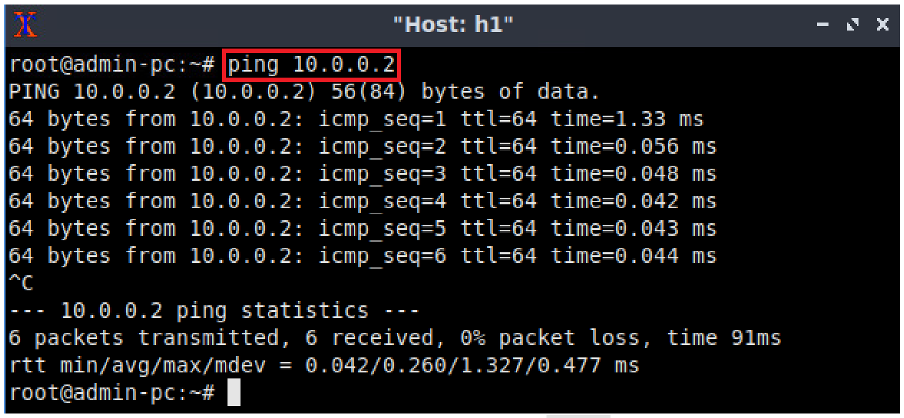


Figure 8. Opening a terminal on host h1.

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.



```
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 9. Connectivity test using `ping` command.

2.2 Emulating 10 Gbps high-latency WAN

This section emulates a high-latency WAN. We will first emulate 20ms delay between switch S1 and switch S2 and measure the throughput. Then, we will set the bandwidth between hosts 1 and 2 to 10 Gbps.

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

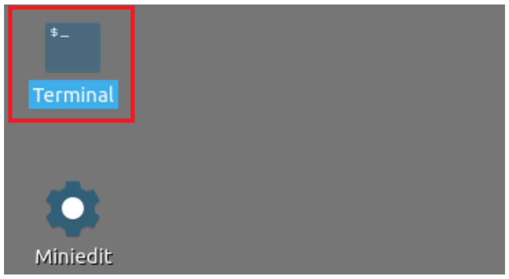
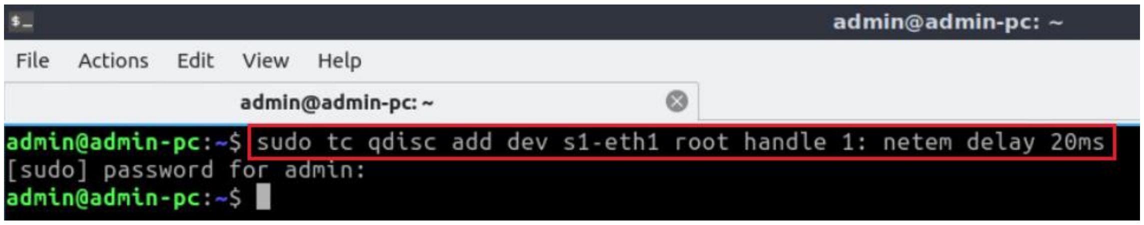


Figure 10. Shortcut to open a Linux terminal.

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 on switch S1's *s1-eth1* interface.

```
sudo tc qdisc add dev s1-eth1 root handle 1: netem delay 20ms
```



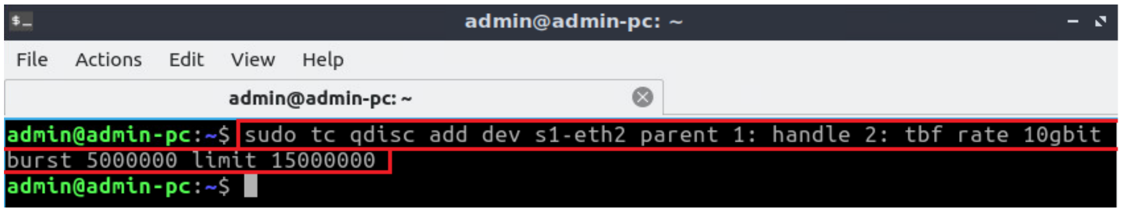
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth1 root handle 1: netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$ 
```

Figure 11. Adding delay of 20ms to switch S1's *s1-eth1* interface.

Step 3. Modify the bandwidth of the link connecting the switch S1 and switch S2: on the same terminal, type the command below. This command sets the bandwidth to 10Gbps on switch S1's *s1-eth2* interface. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth1 parent 1: handle 2: tbf rate 10gbit burst 5000000
limit 15000000
```



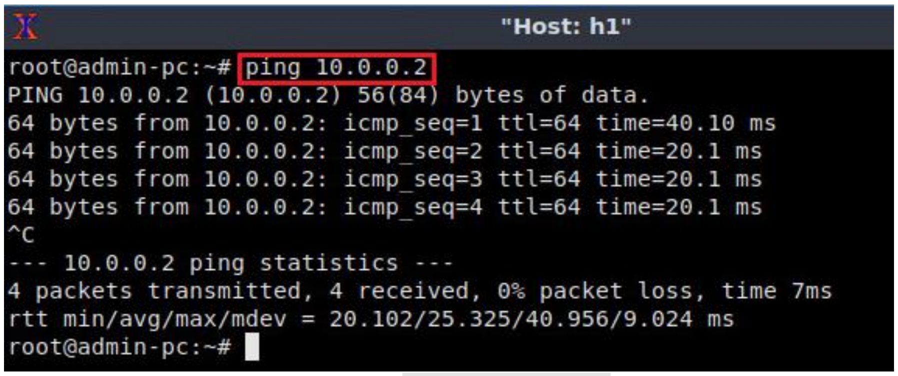
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$ 
```

Figure 12. Limiting the bandwidth to 10 Gbps on switch S1's *s1-eth1* interface.

2.3 Testing connection

To test connectivity, you can use the command `ping`.

Step 1. On the terminal of host h1, type `ping 10.0.0.2`. 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.



```
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=40.10 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.102/25.325/40.956/9.024 ms
root@admin-pc:~# 
```

Figure 13. 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 20.102, 25.325, 40.956, and 9.024 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

Step 2. To change the current receive-window size value(s), we calculate the Bandwidth-Delay Product by performing the following calculation:

$$\text{BW} = 10{,}000{,}000{,}000 \text{ bits/second}$$

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

$$\begin{aligned}\text{BDP} &= 10{,}000{,}000{,}000 \cdot 0.02 = 200{,}000{,}000 \text{ bits}\\ &= 25{,}000{,}000 \text{ bytes} \approx 25 \text{ Mbytes}\end{aligned}$$

The send and receive buffer sizes should be set to 2 · BDP. We will use the 25 Mbytes value for the BDP instead of 25,000,000 bytes.

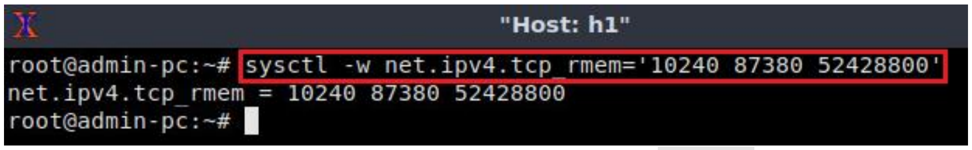
$$1 \text{ Mbyte} = 1024^2 \text{ bytes}$$

$$\text{BDP} = 25 \text{ Mbytes} = 25 \cdot 1024^2 \text{ bytes} = 26{,}214{,}400 \text{ bytes}$$

$$\text{TCP buffer size} = 2 \cdot \text{BDP} = 2 \cdot 26{,}214{,}400 \text{ bytes} = 52{,}428{,}800 \text{ bytes}$$

Now, we have calculated the maximum value of the TCP sending and receiving buffer size. In order to apply the new values, on host h1's terminal type the command showed down below. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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

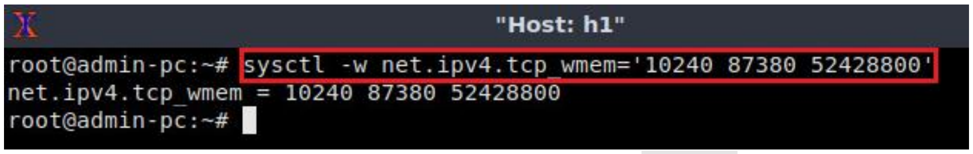


```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~# 
```

Figure 14. Receive window change in `sysctl`.

Step 3. 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, calculated by doubling the BDP).

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



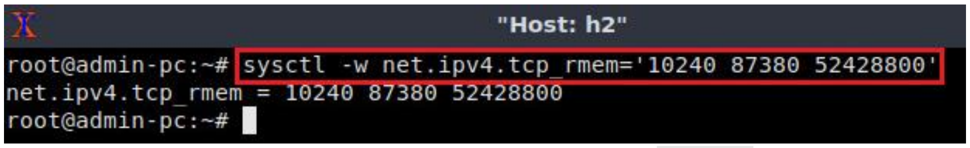
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~# 
```

Figure 15. Send window change in `sysctl`.

Next, the same commands must be configured on host h2.

Step 4. To change the current receive-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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

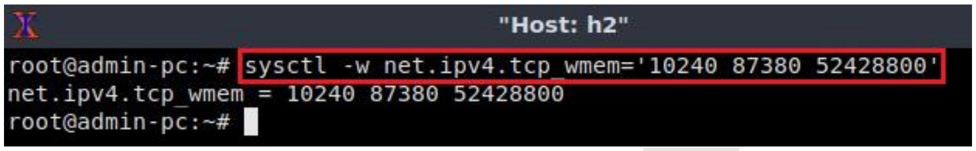


```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~# 
```

Figure 16. Receive window change in `sysctl`.

Step 5. To change the current send-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum, calculated by doubling the BDP).

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

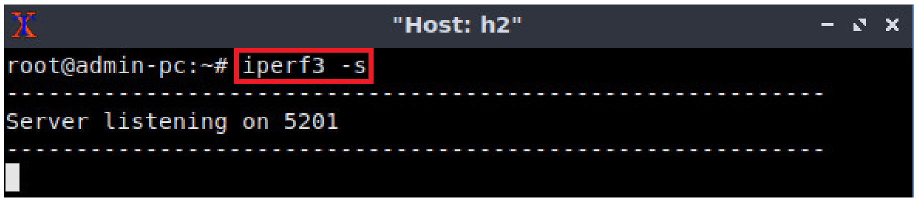


```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~# 
```

Figure 17. Send window change in `sysctl`.

Step 6. The user can now verify the rate limit 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:

```
iperf3 -s
```

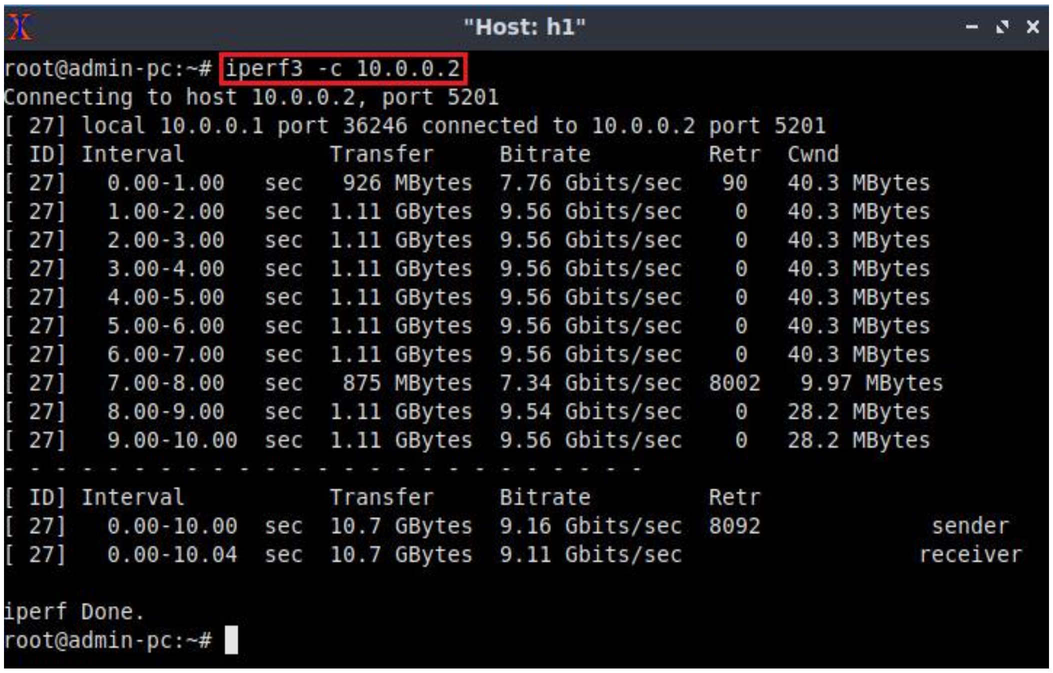


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 18. Host h2 running iPerf3 as server.

Step 7. Now to launch iPerf3 in client mode again by running the command `iperf3 -c 10.0.0.2` in host h1's terminal:

```
iperf3 -c 10.0.0.2
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 27] local 10.0.0.1 port 36246 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 27]   0.00-1.00   sec   926 MBytes  7.76 Gbits/sec   90   40.3 MBytes
[ 27]   1.00-2.00   sec  1.11 GBytes  9.56 Gbits/sec    0   40.3 MBytes
[ 27]   2.00-3.00   sec  1.11 GBytes  9.56 Gbits/sec    0   40.3 MBytes
[ 27]   3.00-4.00   sec  1.11 GBytes  9.56 Gbits/sec    0   40.3 MBytes
[ 27]   4.00-5.00   sec  1.11 GBytes  9.56 Gbits/sec    0   40.3 MBytes
[ 27]   5.00-6.00   sec  1.11 GBytes  9.56 Gbits/sec    0   40.3 MBytes
[ 27]   6.00-7.00   sec  1.11 GBytes  9.56 Gbits/sec    0   40.3 MBytes
[ 27]   7.00-8.00   sec   875 MBytes  7.34 Gbits/sec  8002   9.97 MBytes
[ 27]   8.00-9.00   sec  1.11 GBytes  9.54 Gbits/sec    0   28.2 MBytes
[ 27]   9.00-10.00  sec  1.11 GBytes  9.56 Gbits/sec    0   28.2 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 27]   0.00-10.00  sec  10.7 GBytes  9.16 Gbits/sec  8092             sender
[ 27]   0.00-10.04  sec  10.7 GBytes  9.11 Gbits/sec                  receiver

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

Figure 19. iPerf3 throughput test.

Note the measured throughput is approximately 10 Gbps, which is close to the value assigned in our `tbf` rule (10 Gbps).

Step 8. 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.

3 Enabling TCP pacing with tc and fq

The user enables fair queuing using a command line utility called `tc`. The basic `tc` syntax used with `fq` is as follows:

```
sudo tc qdisc [add|del|replace|change|show] dev dev_id root fq opts
```

`sudo`**:** enables the execution of the command with higher security privileges.

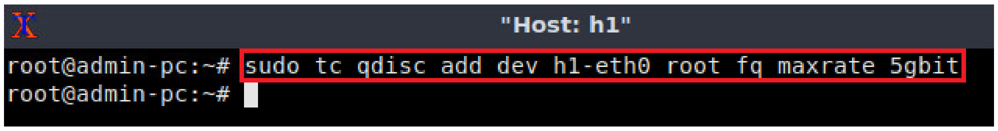
`tc`: invokes Linux's traffic control.
`qdisc`: a queue discipline (qdisc) is a set of rules that determine the order in which packets arriving from the IP protocol output are served. The queue discipline is applied to a packet queue to decide when to send each packet.
`[add | del | replace | change | show]`: this is the operation on qdisc. For example, to add delay on a specific interface, the operation will be `add`. To change or remove delay on the specific interface, the operation will be `change` or `del`.
`dev_id`: this parameter indicates the interface to be subject to emulation.
`fq`: this parameter enables fair queuing qdisc.
`opts`: this parameter indicates the amount of delay, packet loss, duplication, corruption, and others.

Step 1. In host h1, type the following command:

```
sudo tc qdisc add dev h1-eth0 root fq maxrate 5gbit
```

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`: specify the interface on which the rule will be applied.
`fq`: use the fair queueing qdics.
`maxrate 5gbit`: Maximum sending rate of a flow (default is unlimited). Enables pacing on a maximum rate of 5 Gbps.

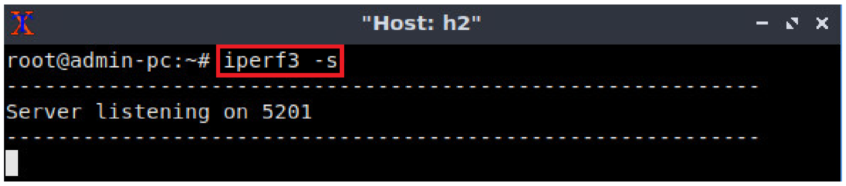


```
root@admin-pc:~# sudo tc qdisc add dev h1-eth0 root fq maxrate 5gbit
root@admin-pc:~# 
```

Figure 20. Enabling fair queuing pacing with a maximum rate of 5 Gbps to the interface *h1-eth0* on host h1.

Step 2. The user can now verify pacing 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:

```
iperf3 -s
```

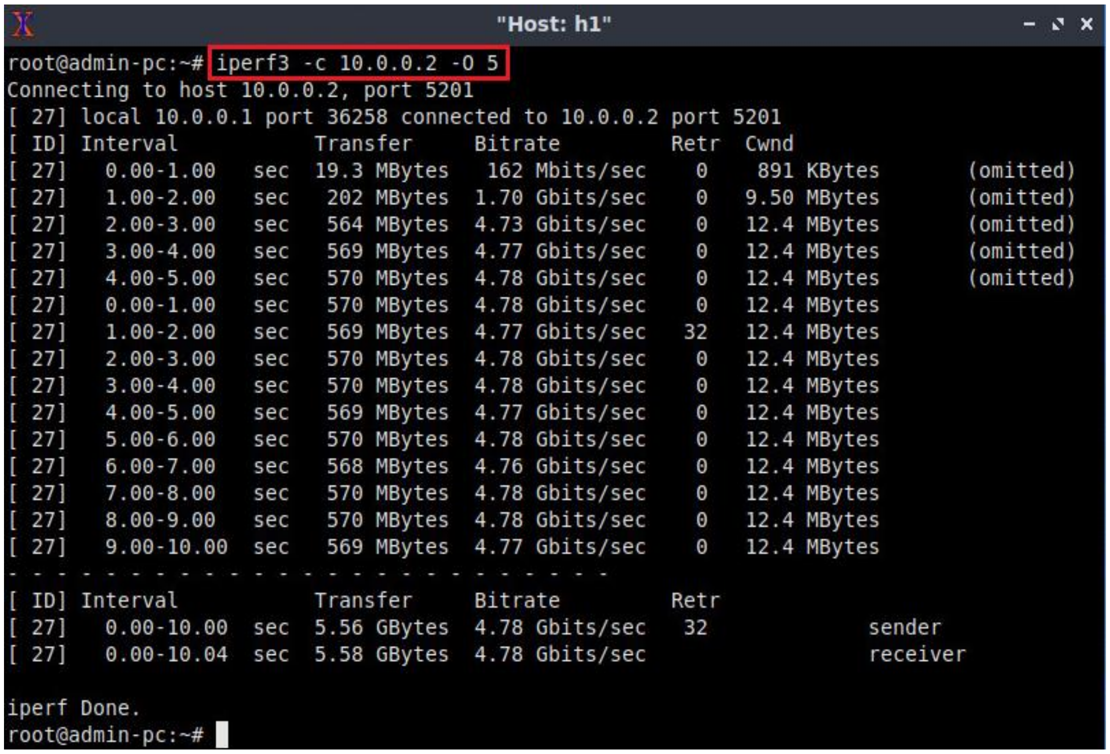


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 21. Host h2 running iPerf3 as server.

Step 3. Now to launch iPerf3 in client mode again by running the command `iperf3 -c 10.0.0.2 -O 5` in host h1's terminal. The `-O` option is used to specify the number of seconds to omit in the resulting report.

```
iperf3 -c 10.0.0.2 -O 5
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2 -O 5
Connecting to host 10.0.0.2, port 5201
[ 27] local 10.0.0.1 port 36258 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 27]   0.00-1.00   sec  19.3 MBytes   162 Mbits/sec    0    891 KBytes       (omitted)
[ 27]   1.00-2.00   sec   202 MBytes  1.70 Gbits/sec    0   9.50 MBytes       (omitted)
[ 27]   2.00-3.00   sec   564 MBytes  4.73 Gbits/sec    0   12.4 MBytes       (omitted)
[ 27]   3.00-4.00   sec   569 MBytes  4.77 Gbits/sec    0   12.4 MBytes       (omitted)
[ 27]   4.00-5.00   sec   570 MBytes  4.78 Gbits/sec    0   12.4 MBytes       (omitted)
[ 27]   0.00-1.00   sec   570 MBytes  4.78 Gbits/sec    0   12.4 MBytes
[ 27]   1.00-2.00   sec   569 MBytes  4.77 Gbits/sec   32   12.4 MBytes
[ 27]   2.00-3.00   sec   570 MBytes  4.78 Gbits/sec    0   12.4 MBytes
[ 27]   3.00-4.00   sec   570 MBytes  4.78 Gbits/sec    0   12.4 MBytes
[ 27]   4.00-5.00   sec   569 MBytes  4.77 Gbits/sec    0   12.4 MBytes
[ 27]   5.00-6.00   sec   570 MBytes  4.78 Gbits/sec    0   12.4 MBytes
[ 27]   6.00-7.00   sec   568 MBytes  4.76 Gbits/sec    0   12.4 MBytes
[ 27]   7.00-8.00   sec   570 MBytes  4.78 Gbits/sec    0   12.4 MBytes
[ 27]   8.00-9.00   sec   570 MBytes  4.78 Gbits/sec    0   12.4 MBytes
[ 27]   9.00-10.00  sec   569 MBytes  4.77 Gbits/sec    0   12.4 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 27]   0.00-10.00  sec  5.56 GBytes  4.78 Gbits/sec   32             sender
[ 27]   0.00-10.04  sec  5.58 GBytes  4.78 Gbits/sec                  receiver

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

Figure 22. iPerf3 throughput test.

The figure above shows the iPerf3 test output report. The average achieved throughput is 4.78 Gbps (sender) and 4.78 Gbps (receiver), which is close to the assigned pacing value (5 Gbps).

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.

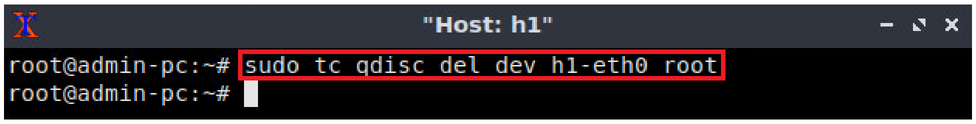
4 Enabling TCP pacing from application

An application can specify a maximum pacing rate using the SO_MAX_PACING_RATE setsockopt call. This packet scheduler adds delay between packets to respect rate limitation set on each socket. Application specific setting via SO_MAX_PACING_RATE is ignored only if it is larger than the `maxrate` value assigned with `fq` (if any).

In iPerf3, the option `--fq-rate` sets a rate to be used with fair-queueing based socket-level pacing, in bits per second.

Step 1. Remove previous *qdiscs* on host h1's *h1-eth0* interface.

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



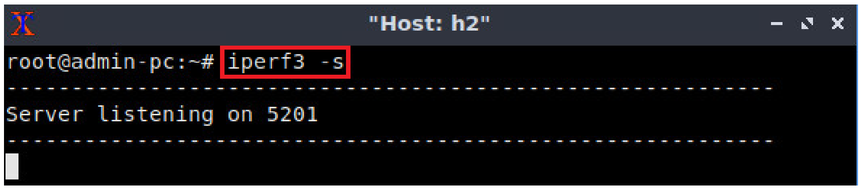
"Host: h1"

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

Figure 23. Removing *qdiscs* on host h1's *h1-eth0* interface.

Step 2. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal:

```
iperf3 -s
```



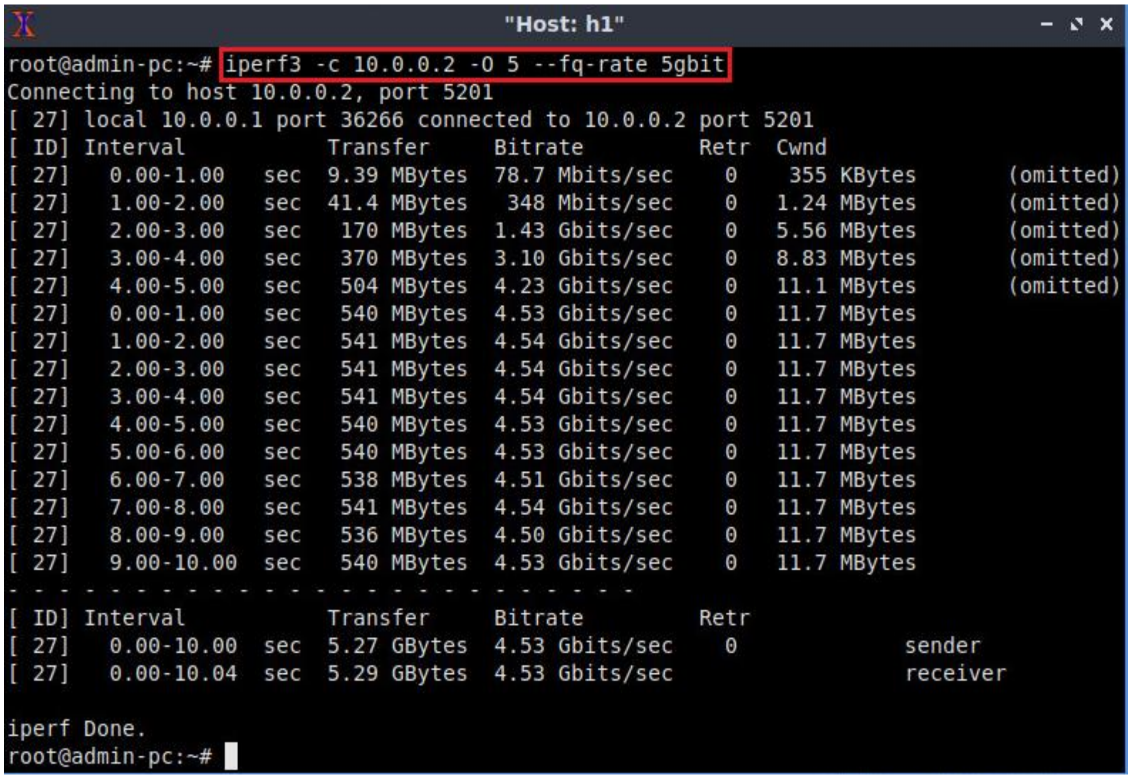
"Host: h2"

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

Figure 24. Host h2 running iPerf3 as server.

Step 3. Now launch iPerf3 in client mode by running the command `iperf3 -c 10.0.0.2 -O 5 --fq-rate 5gbit` in host h1's terminal. The `-O` option is used to specify the number of seconds to omit in the resulting report (5 seconds), and the `--fq-rate` is used to enable pacing through the SO_MAX_PACING_RATE setsockopt call.

```
iperf3 -c 10.0.0.2 -O 5 --fq-rate 5gbit
```



"Host: h1"

```
root@admin-pc:~# iperf3 -c 10.0.0.2 -O 5 --fq-rate 5gbit
Connecting to host 10.0.0.2, port 5201
[ 27] local 10.0.0.1 port 36266 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 27]   0.00-1.00   sec  9.39 MBytes  78.7 Mbits/sec    0    355 KBytes       (omitted)
[ 27]   1.00-2.00   sec  41.4 MBytes   348 Mbits/sec    0   1.24 MBytes       (omitted)
[ 27]   2.00-3.00   sec   170 MBytes  1.43 Gbits/sec    0   5.56 MBytes       (omitted)
[ 27]   3.00-4.00   sec   370 MBytes  3.10 Gbits/sec    0   8.83 MBytes       (omitted)
[ 27]   4.00-5.00   sec   504 MBytes  4.23 Gbits/sec    0   11.1 MBytes       (omitted)
[ 27]   0.00-1.00   sec   540 MBytes  4.53 Gbits/sec    0   11.7 MBytes
[ 27]   1.00-2.00   sec   541 MBytes  4.54 Gbits/sec    0   11.7 MBytes
[ 27]   2.00-3.00   sec   541 MBytes  4.54 Gbits/sec    0   11.7 MBytes
[ 27]   3.00-4.00   sec   541 MBytes  4.54 Gbits/sec    0   11.7 MBytes
[ 27]   4.00-5.00   sec   540 MBytes  4.53 Gbits/sec    0   11.7 MBytes
[ 27]   5.00-6.00   sec   540 MBytes  4.53 Gbits/sec    0   11.7 MBytes
[ 27]   6.00-7.00   sec   538 MBytes  4.51 Gbits/sec    0   11.7 MBytes
[ 27]   7.00-8.00   sec   541 MBytes  4.54 Gbits/sec    0   11.7 MBytes
[ 27]   8.00-9.00   sec   536 MBytes  4.50 Gbits/sec    0   11.7 MBytes
[ 27]   9.00-10.00  sec   540 MBytes  4.53 Gbits/sec    0   11.7 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 27]   0.00-10.00  sec  5.27 GBytes  4.53 Gbits/sec    0             sender
[ 27]   0.00-10.04  sec  5.29 GBytes  4.53 Gbits/sec                  receiver

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

Figure 25. iPerf3 throughput test with pacing enabled by iPerf3 application.

5 Concurrent transmission without pacing

In the previous section, we applied pacing on a single host (host h1) and we measured the average throughput. In this section we run a test where four clients (host h1, host h3, host h5, and host h7) are transmitting simultaneously to four servers (host h2, host h4, host h6, and host h8), while sharing the same bottleneck link (link connecting switch S1 to switch S2).

Since it is difficult to start the four clients at the same time, Client1's machine provides a script that automates this process.

Step 1. Close the terminals of host h1 and host h2.

Step 2. Go to Mininet's terminal, i.e., the one launched when MiniEdit was started.

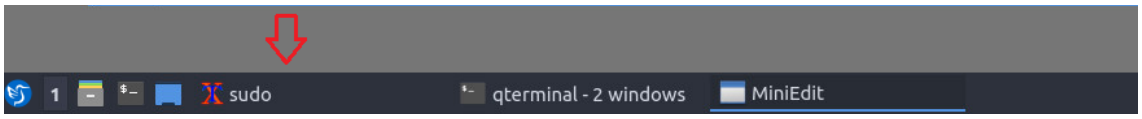
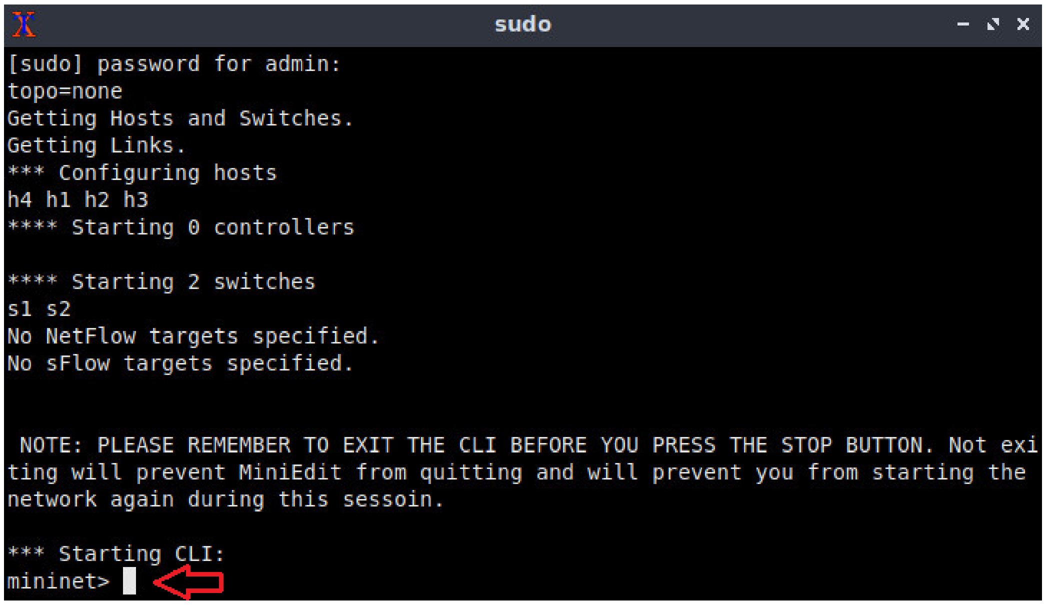


Figure 26. Opening Mininet's terminal.

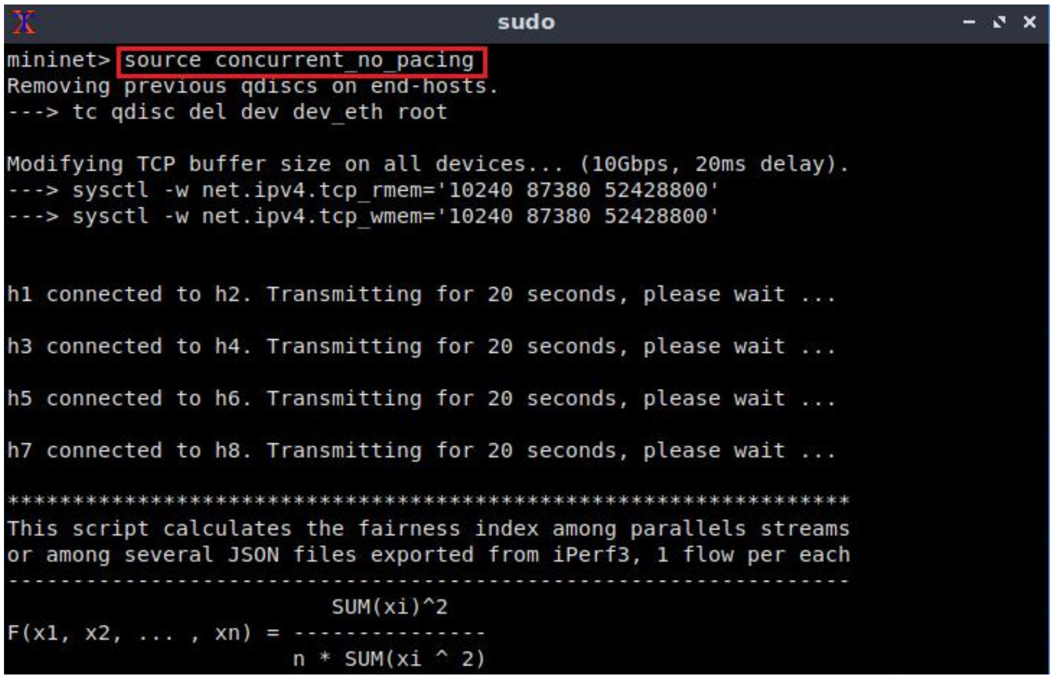


```
[sudo] password for admin:
topo=none
Getting Hosts and Switches.
Getting Links.
*** Configuring hosts
h4 h1 h2 h3
**** Starting 0 controllers

**** Starting 2 switches
s1 s2
No NetFlow targets specified.
No sFlow targets specified.


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exi
ting will prevent MiniEdit from quitting and will prevent you from starting the
network again during this sessoin.

*** Starting CLI:
mininet>
```

Figure 27. Mininet's terminal.

Step 3. Issue the following command on Mininet's terminal as shown in the figure below.

```
source concurrent_no_pacing
```



```
mininet> source concurrent_no_pacing
Removing previous qdiscs on end-hosts.
---> tc qdisc del dev dev_eth root

Modifying TCP buffer size on all devices... (10Gbps, 20ms delay).
---> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
---> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'


h1 connected to h2. Transmitting for 20 seconds, please wait ...

h3 connected to h4. Transmitting for 20 seconds, please wait ...

h5 connected to h6. Transmitting for 20 seconds, please wait ...

h7 connected to h8. Transmitting for 20 seconds, please wait ...

****************************************************************
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
----------------------------------------------------------------
                        SUM(xi)^2
F(x1, x2, ... , xn) = ---------------
                     n * SUM(xi ^ 2)
```

Figure 28. Running the tests simultaneously for 20 seconds without applying pacing.

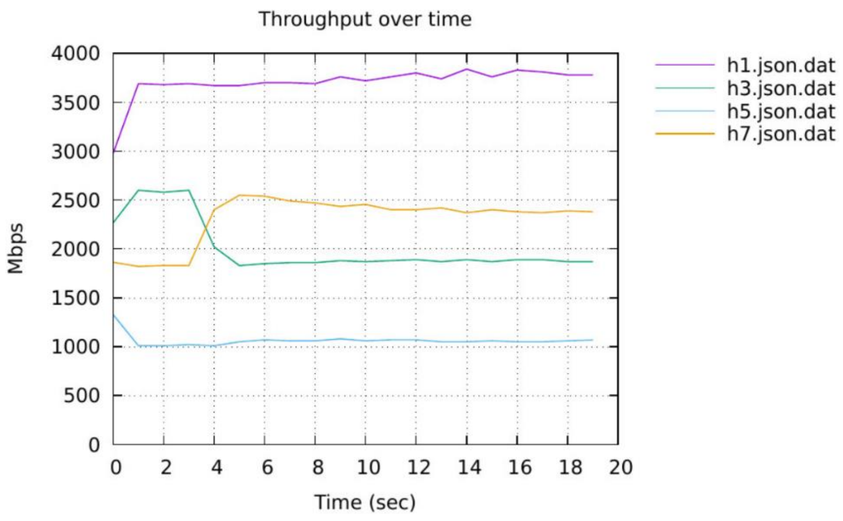
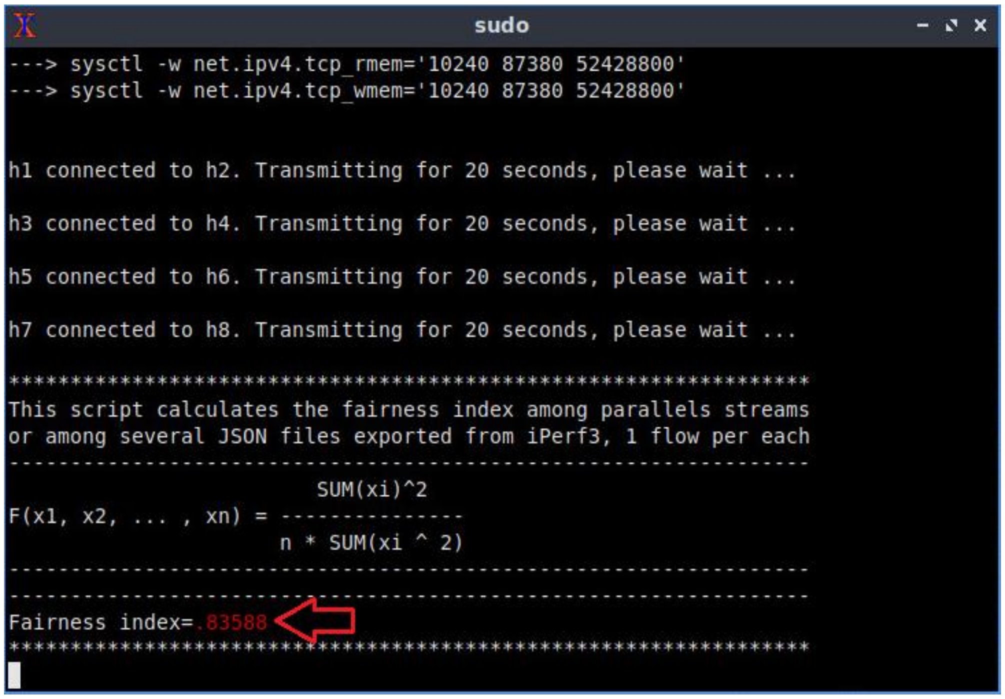


Figure 29. Throughput of host h1, host h3, host h5 and host h7.

The above graph shows that the throughput of host h1, host h3, host h5 and host h7. It is clear from the figure that there are variations in the flows. Moreover, the bottleneck bandwidth was not evenly shared among the hosts, which decreases the fairness index from 100%.

Step 4. Close the graph window and go back to Mininet's terminal. The fairness index is displayed at the end as shown in the figure below.



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


h1 connected to h2. Transmitting for 20 seconds, please wait ...

h3 connected to h4. Transmitting for 20 seconds, please wait ...

h5 connected to h6. Transmitting for 20 seconds, please wait ...

h7 connected to h8. Transmitting for 20 seconds, please wait ...

*****************************************************************
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----------------------------------------------------------------
                         SUM(xi)^2
F(x1, x2, ... , xn) = --------------
                      n * SUM(xi ^ 2)
-----------------------------------------------------------------
-----------------------------------------------------------------
Fairness index=.83588
*****************************************************************
```

Figure 30. Calculated fairness index.

The above figure shows a fairness index of .83588. This value indicates that the bottleneck bandwidth was approximately 83% evenly shared among host h1, host h3, host h5, and host h7.

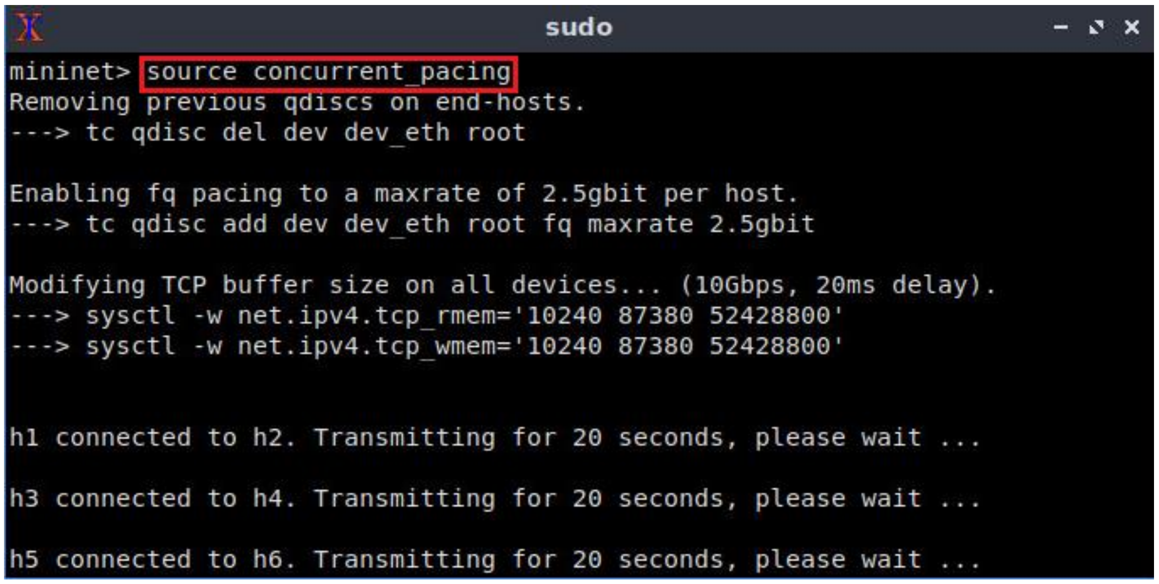
6 Concurrent transmission with pacing

In the previous section, we ran a test where four clients (host h1, host h3, host h5, and host h7) are transmitting simultaneously to four servers (host h2, host h4, host h6, and host h8), while sharing the same bottleneck link (link connecting switch S1 to switch S2) without applying pacing. In this section we repeat the same test, but with pacing enabled on host h1, host h3, host h5 and host h7.

Since it is difficult to start the four clients at the same time, Client1's machine provides a script that automates this process.

Step 1. Using same Mininet's terminal, issue the following command on Mininet's terminal as shown in the figure below.

```
source concurrent_pacing
```



```
mininet> source concurrent_pacing
Removing previous qdiscs on end-hosts.
---> tc qdisc del dev dev_eth root

Enabling fq pacing to a maxrate of 2.5gbit per host.
---> tc qdisc add dev dev_eth root fq maxrate 2.5gbit

Modifying TCP buffer size on all devices... (10Gbps, 20ms delay).
---> sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
---> sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'


h1 connected to h2. Transmitting for 20 seconds, please wait ...

h3 connected to h4. Transmitting for 20 seconds, please wait ...

h5 connected to h6. Transmitting for 20 seconds, please wait ...
```

Figure 31. Running the tests simultaneously for 20 seconds while applying pacing.

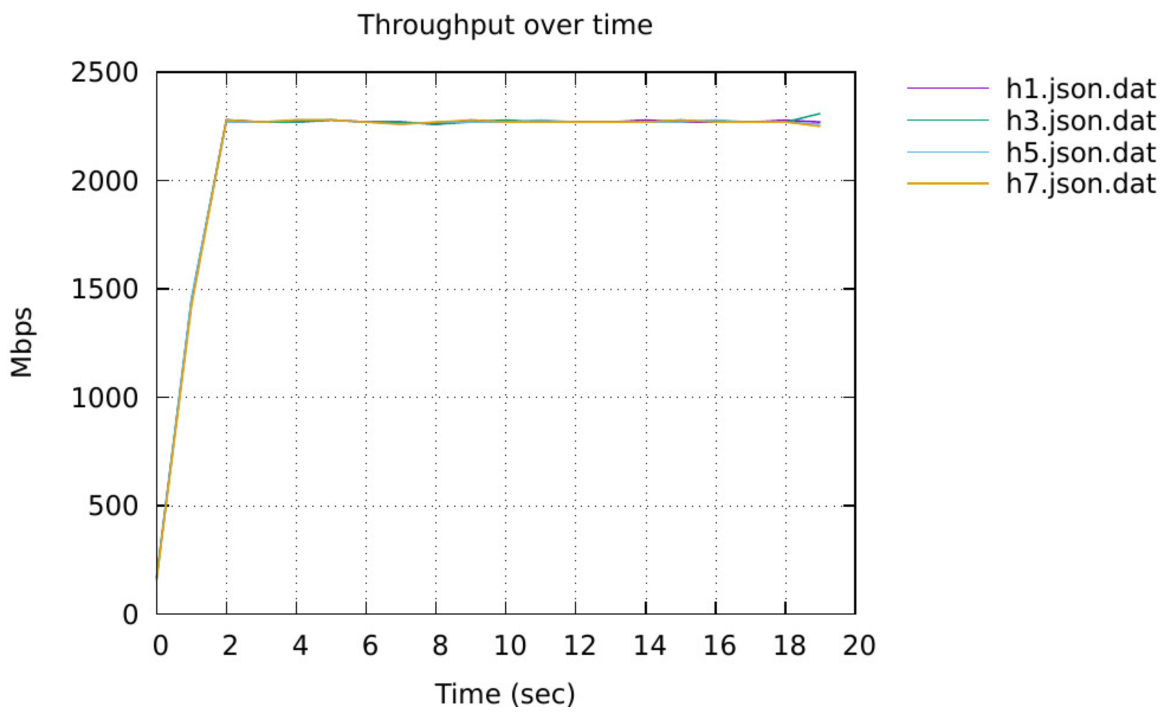
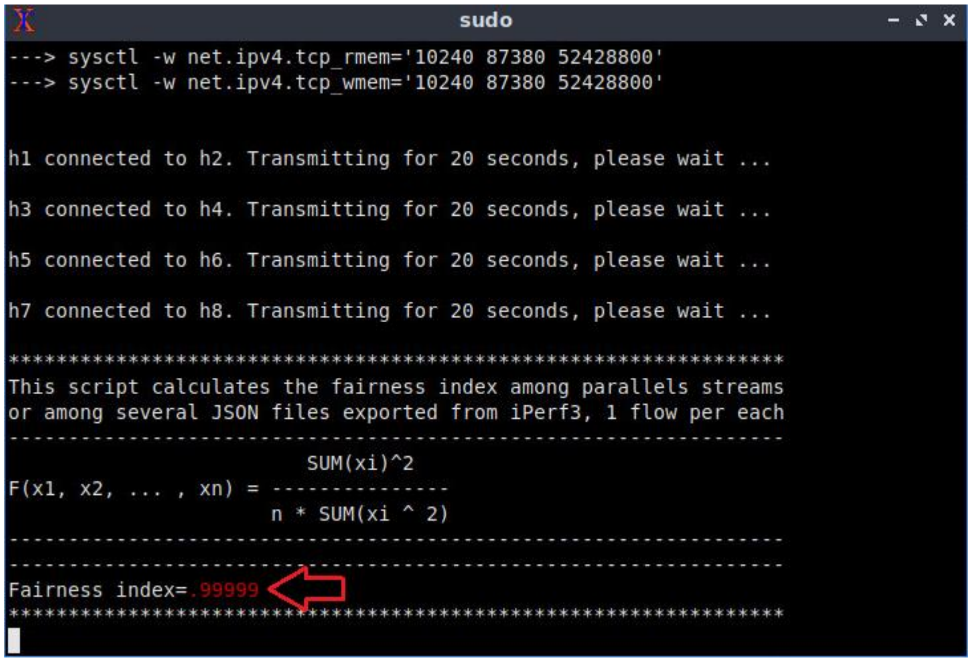


Figure 32. Throughput of host h1, host h3, host h5 and host h7 after applying pacing.

The above graph shows that the throughput of host h1, host h3, host h5 and host h7 with pacing enabled. It is clear from the figure that there are less variations in the flows compared to the non-paced flows. Moreover, the bottleneck bandwidth is now better shared among the hosts.

Step 2. Close the graph window and go back to Mininet's terminal. The fairness index is displayed at the end as shown in the figure below.



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


h1 connected to h2. Transmitting for 20 seconds, please wait ...

h3 connected to h4. Transmitting for 20 seconds, please wait ...

h5 connected to h6. Transmitting for 20 seconds, please wait ...

h7 connected to h8. Transmitting for 20 seconds, please wait ...

*****************************************************************
This script calculates the fairness index among parallels streams
or among several JSON files exported from iPerf3, 1 flow per each
-----------------------------------------------------------------
                         SUM(xi)^2
F(x1, x2, ... , xn) = --------------
                      n * SUM(xi ^ 2)
-----------------------------------------------------------------
-----------------------------------------------------------------
Fairness index=.99999
*****************************************************************
```

Figure 33. Calculated fairness index.

The above figure shows a fairness index of .99999. The fairness index here is better than the previous test .83588. Therefore, pacing generally improves fairness among transmitting hosts.

7 Parallel streams and without pacing

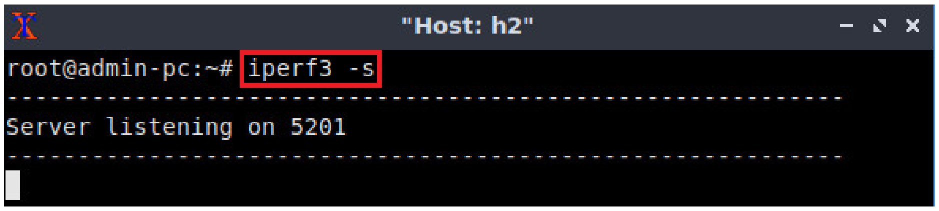
In the previous tests, four clients (host h1, host h3, host h5, and host h7) were transmitting simultaneously to four servers (host h2, host h4, host h6, and host h8), while sharing the same bottleneck link (link connecting switch S1 to switch S2). In this section only one client (host h1) is transmitting to one server (host h2) while using five parallel streams.

Step 1. In MiniEdit, hold the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

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

Step 3. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal:

```
iperf3 -s
```

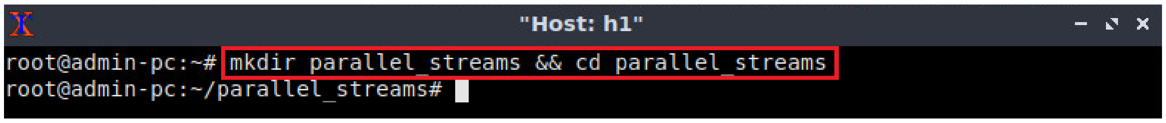


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 34. Host h2 running iPerf3 as server.

Step 4. Create and enter to a new directory *parallel_streams:*

```
mkdir parallel_streams && cd parallel_streams
```

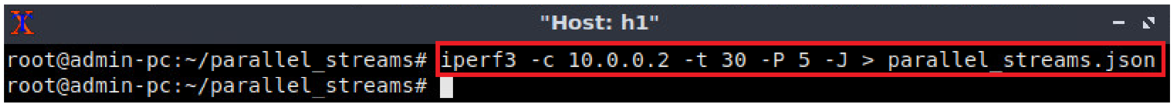

"Host: h1"

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

Figure 35. Creating and entering a new directory *parallel_streams*.

Step 5. Launch iPerf3 in client mode on host h1's terminal. The `-J` option is used to produce a JSON output and the redirection operator `>` to send the standard output to a file.

```
iperf3 -c 10.0.0.2 -t 30 -P 5 -J > parallel_streams.json
```

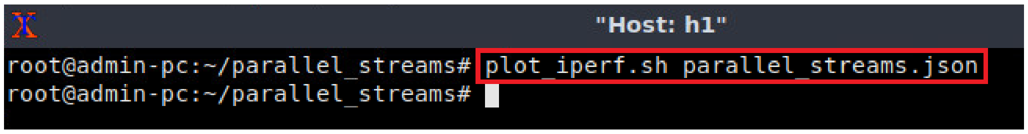

"Host: h1"

```
root@admin-pc:~/parallel_streams# iperf3 -c 10.0.0.2 -t 30 -P 5 -J > parallel_streams.json
root@admin-pc:~/parallel_streams#
```

Figure 36. Running iPerf3 client on host h1 with 5 parallel streams for 30 seconds, and redirecting the output to *parallel_streams.json*.

Step 6. Once the test is finished, in order to generate the output plots for iPerf3's JSON file run the following command:

```
plot_iperf.sh parallel_streams.json
```


"Host: h1"

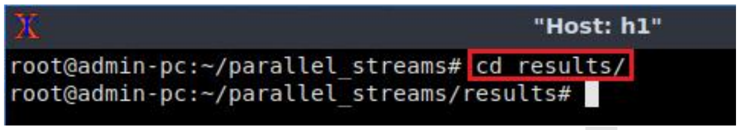
```
root@admin-pc:~/parallel_streams# plot_iperf.sh parallel_streams.json
root@admin-pc:~/parallel_streams#
```

Figure 37. `plot_iperf.sh` script generating output results.

This plotting script generates PDF files for the following fields: congestion window (*cwnd.pdf*), retransmits (*retransmits.pdf*), Round-Trip Time (*RTT.pdf*), throughput (*throughput.pdf*), maximum transmission unit (*MTU.pdf*), bytes transferred (*bytes.pdf*). These files are stored in a directory *results* created in the same directory where the script was executed.

Step 7. Navigate to the results folder using the `cd` command.

```
cd results/
```

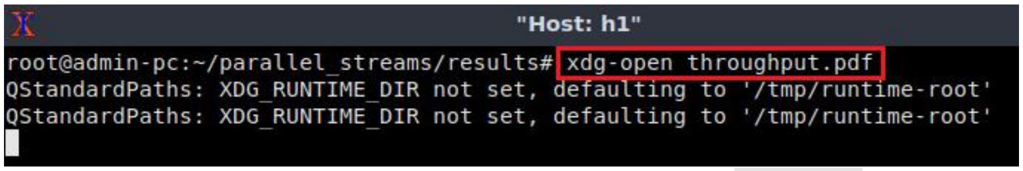

"Host: h1"

```
root@admin-pc:~/parallel_streams# cd results/
root@admin-pc:~/parallel_streams/results#
```

Figure 38. Entering the results directory using the `cd` command.

Step 8. Open the *throughput.pdf* file, use the following command:

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



```
root@admin-pc:~/parallel_streams/results# xdg-open throughput.pdf
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

Figure 39. Opening the *throughput.pdf* file using `xdg-open`.

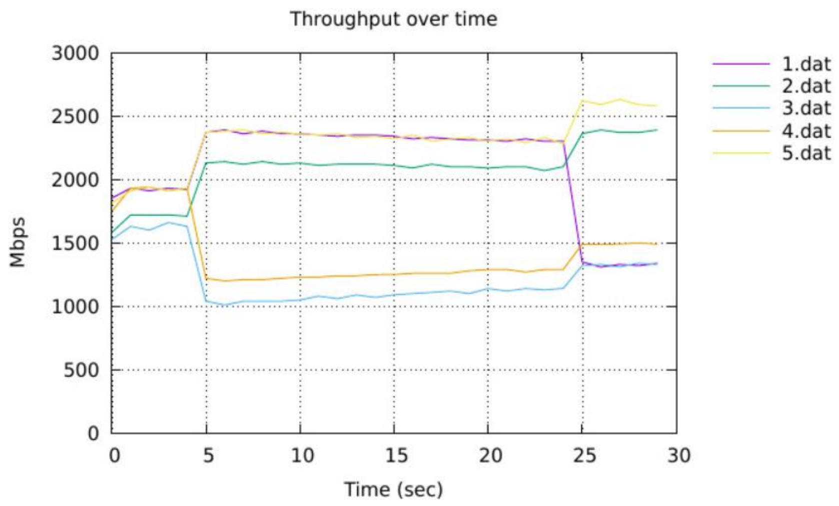
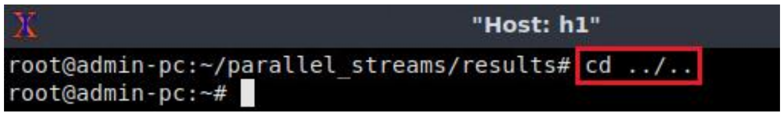


Figure 40. Throughput of 5 parallel streams initiated by host h1 without pacing.

Step 9. Close *throughput.pdf* file and stop the server by pressing `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

Step 10. Exit the *parallel_streams/results* directory by using the following command on host h1's terminal:

```
cd ../..
```



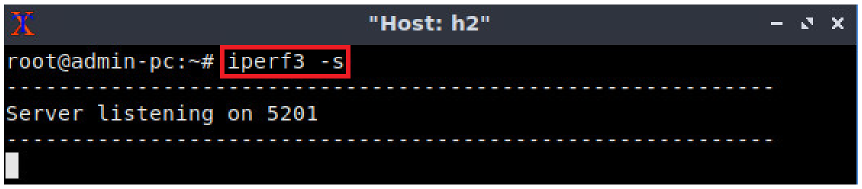
```
root@admin-pc:~/parallel_streams/results# cd ../..
root@admin-pc:~#
```

Figure 41. Exiting the *reno/results* directory.

8 Parallel streams and with pacing

Step 1. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal:

```
iperf3 -s
```

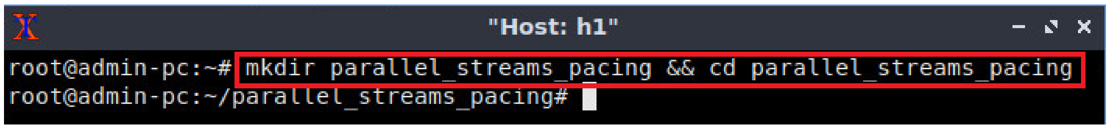


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 42. Host h2 running iPerf3 as server.

Step 2. Create and enter to a new directory *parallel_streams_pacing:*

```
mkdir parallel_streams_pacing && cd parallel_streams_pacing
```

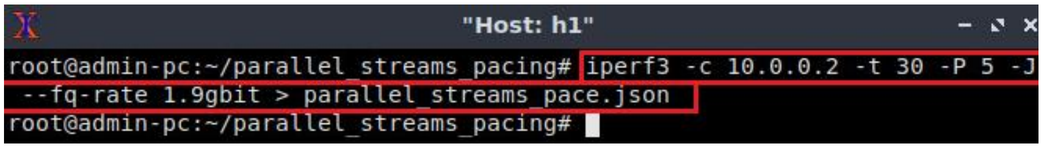


```
root@admin-pc:~# mkdir parallel_streams_pacing && cd parallel_streams_pacing
root@admin-pc:~/parallel_streams_pacing#
```

Figure 43. Creating and entering a new directory *parallel_streams_pacing*.

Step 3. Launch iPerf3 in client mode on host h1's terminal. The `-J` option is used to produce a JSON output and the redirection operator `>` to send the standard output to a file. The `-P` is used to specify the number of parallel streams, and the `--fq-rate` is used to enable pacing through the SO_MAX_PACING_RATE setsockopt call. In this test, pacing is applied to a maximum rate of 1.9 Gbps per stream, and 5 * 1.9 Gbps (9.5 Gbps) total for all streams. Note that assigning a pacing rate slightly less than the maximum bandwidth (10 Gbps in our case) reduces packet lost and the variations of flows.

```
iperf3 -c 10.0.0.2 -t 30 -P 5 -J --fq-rate 1.9gbit > parallel_streams_pace.json
```

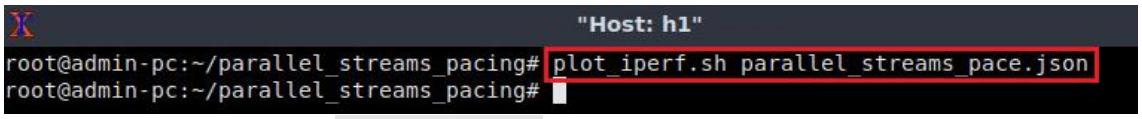


```
root@admin-pc:~/parallel_streams_pacing# iperf3 -c 10.0.0.2 -t 30 -P 5 -J
 --fq-rate 1.9gbit > parallel_streams_pace.json
root@admin-pc:~/parallel_streams_pacing#
```

Figure 44. Running iPerf3 client on host h1 with 5 parallel streams for 30 seconds with pacing enabled, and redirecting the output to *parallel_streams_pace.json*.

Step 4. Once the test is finished, type the command, to generate the output plots for iPerf3's JSON file run the following command:

```
plot_iperf.sh parallel_streams_pace.json
```



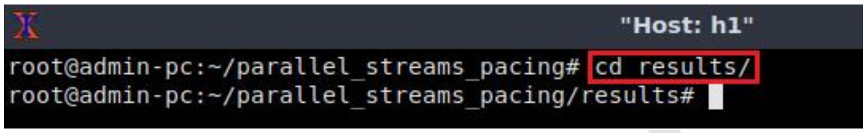
```
root@admin-pc:~/parallel_streams_pacing# plot_iperf.sh parallel_streams_pace.json
root@admin-pc:~/parallel_streams_pacing#
```

Figure 45. `plot_iperf.sh` script generating output results.

This plotting script generates PDF files for the following fields: congestion window (*cwnd.pdf*), retransmits (*retransmits.pdf*), Round-Trip Time (*RTT.pdf*), throughput (*throughput.pdf*), maximum transmission unit (*MTU.pdf*), bytes transferred (*bytes.pdf*). These files are stored in a directory *results* created in the same directory where the script was executed.

Step 5. Navigate to the results folder using the `cd` command.

```
cd results/
```

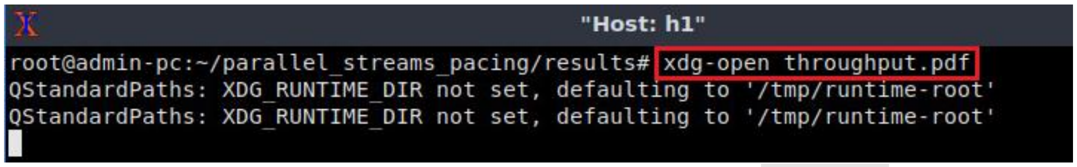


```
root@admin-pc:~/parallel_streams_pacing# cd results/
root@admin-pc:~/parallel_streams_pacing/results#
```

Figure 46. Entering the *results* directory using the `cd` command.

Step 6. Open the *throughput.pdf* file, use the following command:

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



```
root@admin-pc:~/parallel_streams_pacing/results# xdg-open throughput.pdf
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

Figure 47. Opening the *throughput.pdf* file using `xdg-open`.

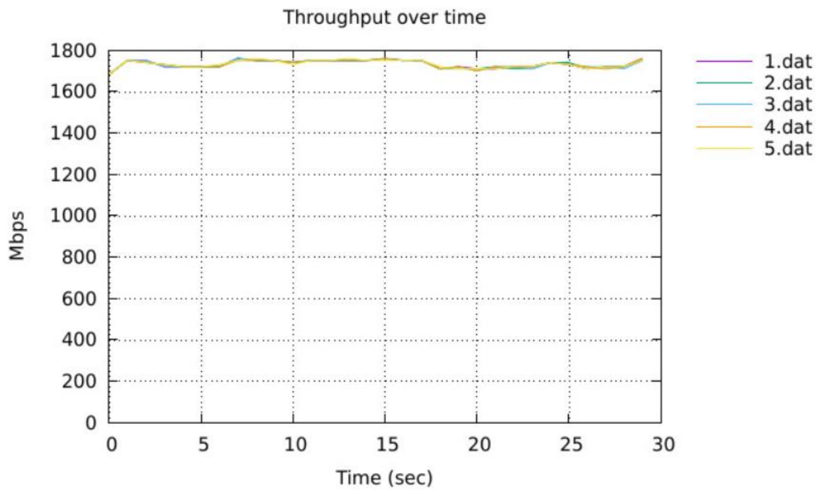


Figure 48. Throughput of 5 parallel streams initiated by host h1 with pacing applied to a maximum rate of 1.9 Gbps per stream.

The graph above shows how the advantages of applying pacing when using parallel streams. Compared to figure 40, the flows have less variations and the fairness among these flows is improved.

This concludes Lab 12. Stop the emulation and then exit out of MiniEdit.

References

1. A. Aggarwal, S. Savage, T. Anderson, "Understanding the performance of TCP pacing," in Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), Mar. 2000.

2. B. Tierney, N. Hanford, D. Ghosal, “Optimizing data transfer nodes using packet pacing: a journey of discovery,” in Workshop on Innovating the Network for Data-Intensive Science, Nov. 2015.
3. M. Ghobadi, Y. Ganjali, “TCP pacing in data center networks,” in IEEE Annual Symposium on High-Performance Interconnects (HOTI), Aug. 2013.
4. 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.
5. Fair Queue traffic policing. [Online]. Available: http://man7.org/linux/man-pages/man8/tc-fq.8.html
6. The centos project. [Online]. Available: https://www.centos.org
7. J. Corbet, “TSO sizing and the FQ scheduler,” LWN.net Online Magazine, Aug. 2013. [Online]. Available: https://lwn.net/Articles/564978
8. B. Tierney, “Improving performance of 40G/100G data transfer nodes,” in 2016 Technology Exchange Workshop, Sep. 2016. [Online]. Available: https://meetings.internet2.edu/2016-technologyexchange/detail/10004333/
9. I. Rhee, L. Xu, “CUBIC: a new TCP-friendly high-speed TCP variant,” ACM Special Interest Group on Operating Systems Operating System Review, vol. 42, issue 5, pp. 64-74, Jul. 2008.
10. J. Padhye, V. Firoiu, D. Towsley, J. Kurose, “Modeling TCP throughput: a simple model and its empirical validation,” in Proceedings of the ACM SIGCOMM ’98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp. 303-314, Sep. 1998.
11. A. Saeed, N. Dukkipati, V. Valancius, C. Contavalli, A. Vahdat, “Carousel: scalable traffic shaping at end hosts,” in Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pp. 404-417, Aug. 2017.
12. M. Shreedhar, G. Varghese, “Efficient fair queuing using deficit round-robin,” IEEE/ACM Transactions on Networking, vol. 4, issue 3, pp. 375-385, Jun. 1996.

NETWORK TOOLS AND PROTOCOLS

Lab 13: Impact of MSS on Throughput

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 MSS 3
1.1 Maximum transmission unit (MTU) 3
1.2 Maximum segment size (MSS) 4
2 Lab topology 5
2.1 Starting hosts h1 and h2 6
2.2 Emulating 10 Gbps WAN with packet loss 7
3 Modifying maximum transmission unit (MTU) 10
3.1 Identifying interface’s current MTU 10
3.2 Modifying MTU values on all interfaces 11
References 14

Overview

This lab introduces Maximum Transmission Unit (MTU), Maximum Segment Size (MSS), and their effect on network throughput in a high-bandwidth Wide Area Networks (WAN) with packet losses. Throughput measurements are conducted in this lab to compare the observed throughput while using a higher MSS against a normal MSS value.

Objectives

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

1. Understand Maximum Transmission Unit (MTU).
2. Define Maximum Segment Size (MSS).
3. Identify interfaces' default MTU value.
4. Modify the MTU of an interface.
5. Understand the benefit of using a high MSS value in a WAN that incurs packet losses.
6. Emulate WAN properties in Mininet and achieve full throughput with high MSS.

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 MSS.
2. Section 2: Lab topology.
3. Section 3: Modifying maximum transmission Unit (MTU) and analyzing results.

1 Introduction to MSS

1.1 Maximum transmission unit (MTU)

The *Maximum Transmission Unit (MTU)* specifies the largest packet size (in bytes), including headers and data payload, that can be transmitted by the link-layer technology[1]. Even though data rates have dramatically increased since Ethernet standardization, the MTU remains at 1500 bytes. A frame carrying more than 1500 bytes is referred to as a jumbo frame and can allow up to 9000 bytes.

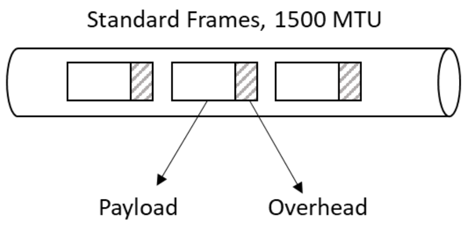


Figure 1. Standard Ethernet Frame's MTU

Figure 1 illustrates the standard Ethernet frame which has 1500 bytes MTU. Although most gigabit networks run with no impact while using the standard MTU, large numbers of frames increase CPU loads and overheads. In such cases jumbo frames can be used to mitigate excess overhead, as demonstrated in figure 2.

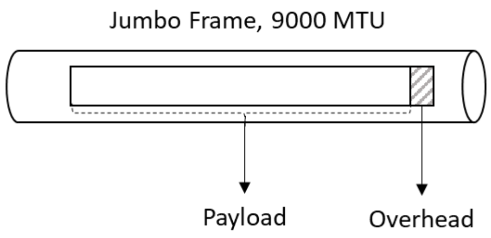


Figure 2. Jumbo Ethernet Frame's MTU

As shown in figure 2, jumbo frames impose lower overheads than normal frames (1500 MTU) by reducing the overall number of individual frames sent from source to destination. Not only does this reduce the number of headers needed to move the data, CPU load is also lessened due to a decrease in packet processing by routers and end devices.

1.2 Maximum segment size (MSS)

The Maximum Segment Size (MSS) is a parameter of the options field of the TCP header that specifies the largest amount of data, specified in bytes, that a computer or communications device can receive in a single TCP segment[3]. This value is specified in the TCP SYN packet during TCP's three-way handshake and is set permanently for the current session.

The MSS must be set to a value equal to the largest IP datagram (minus IP and TCP headers) that the host can handle in order to avoid fragmentation. Note that lowering the MSS will remove fragmentation, however it will impose larger overhead.

With highspeed networks, using half a dozen or so small probes to see how the network responds wastes a huge amount of bandwidth. Similarly, when packet loss is detected, the rate is decreased by a factor of two. TCP can only recover slowly from this rate reduction. The speed at which the recovery occurs is proportional to the MTU. Thus, it is recommended to use large frames.

In this lab, we show and compare the effect of jumbo frames versus standard frames in a WAN that incurs packet losses.

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.

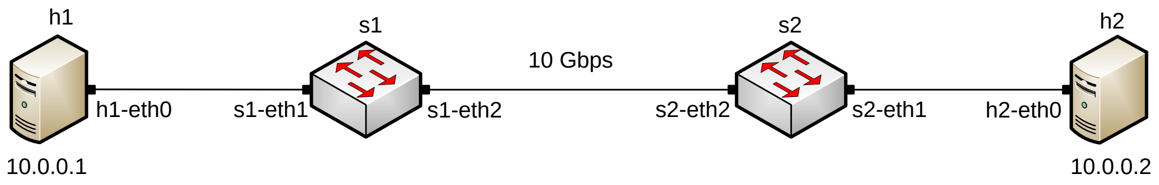


Figure 3. Lab topology.

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`.

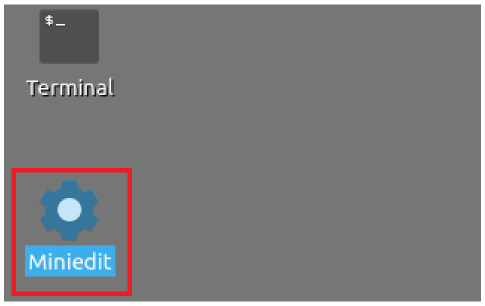


Figure 4. Miniedit shortcut.

Step 2. On Miniedit’s menu bar, click on *File* then *Open* to load the lab’s topology. Locate the *Lab 13.mn* topology file and click on *Open*.

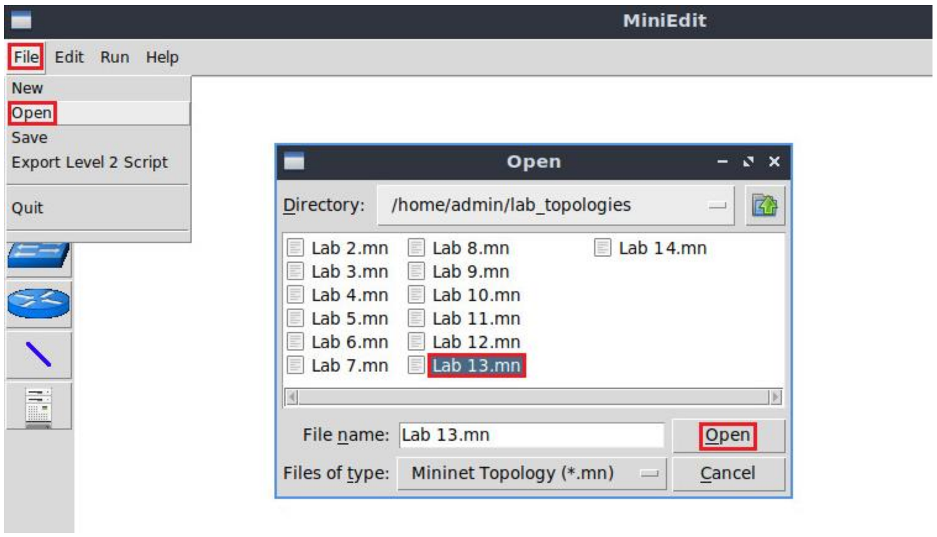


Figure 5. Miniedit's *Open* dialog.

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.

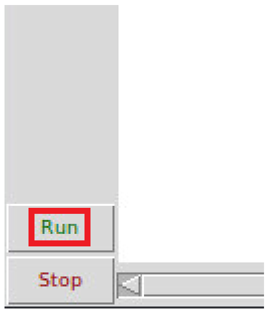


Figure 6. Running the emulation.

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

2.1 Starting hosts h1 and h2

Step 1. Hold the right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

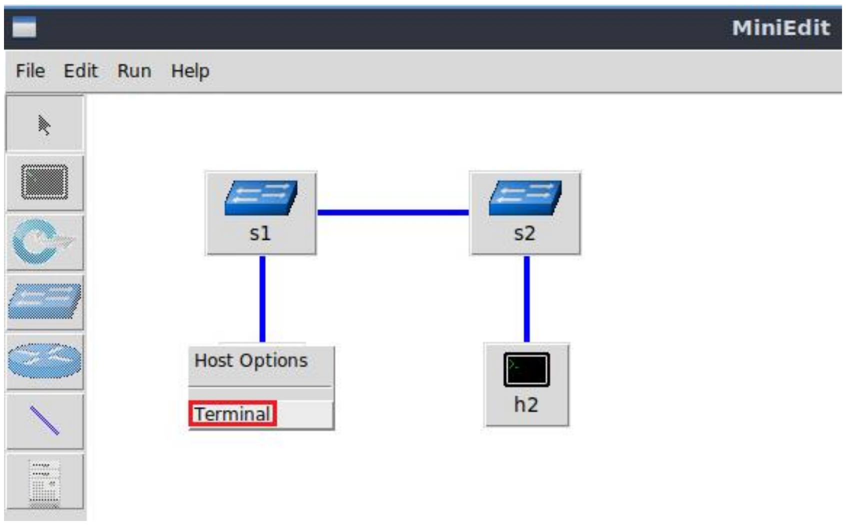
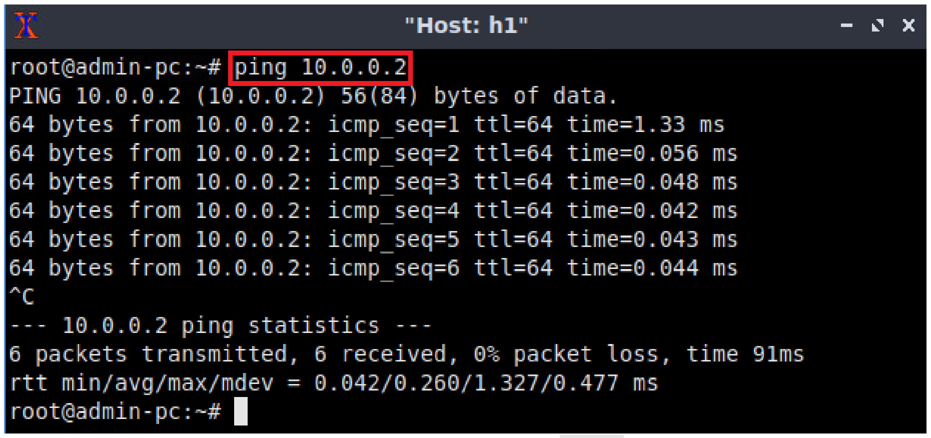


Figure 7. Opening a terminal on host h1.

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.



```
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.

2.2 Emulating 10 Gbps WAN with packet loss

This section emulates a WAN with packet loss. We will first set the bandwidth between host 1 and host h2 to 10 Gbps. Then, we will emulate a 1% packet loss and measure the throughput.

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

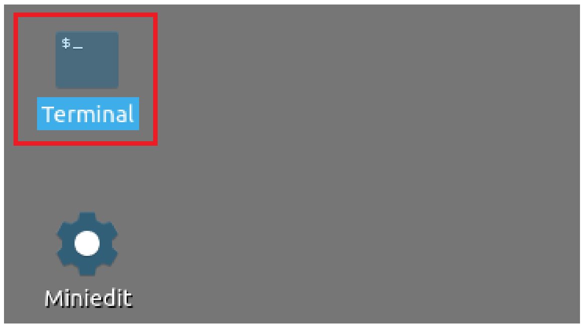
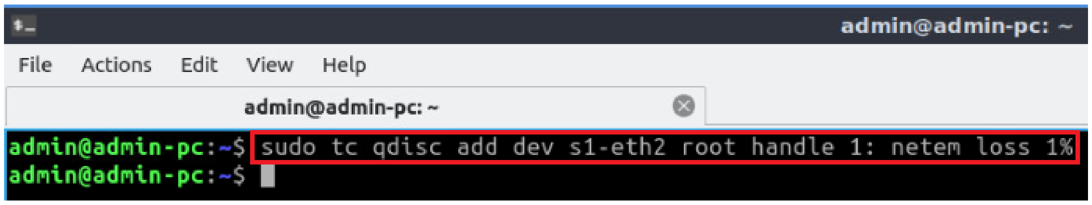


Figure 9. Shortcut to open a Linux terminal.

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 1% packet loss on switch S1's *s1-eth2* interface.

```
sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 1%
```



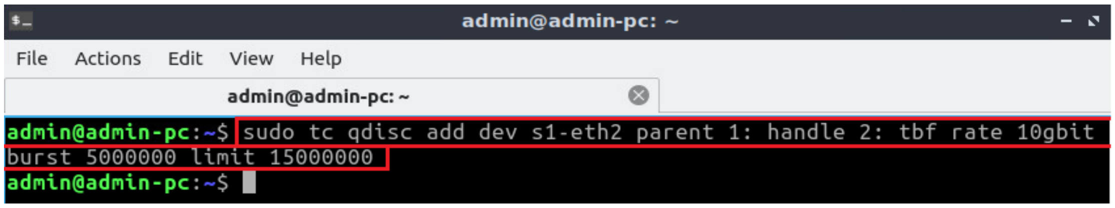
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 1%
admin@admin-pc:~$
```

Figure 10. Adding 1% packet loss to switch S1's *s1-eth2* interface.

Step 3. Modify the bandwidth of the link connecting the switch S1 and switch S2: on the same terminal, type the command below. This command sets the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface. The `tbf` parameters are the following:

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

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit burst 5000000
limit 15000000
```

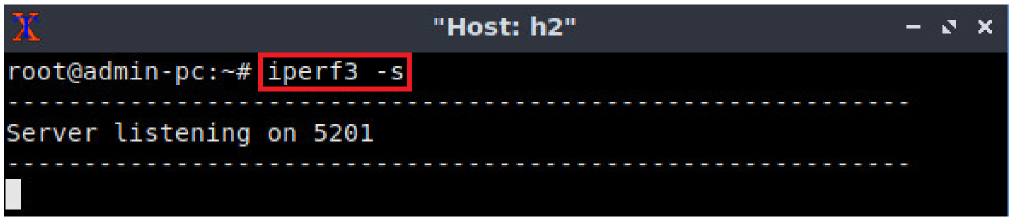


```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$
```

Figure 11. Limiting the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface.

Step 4. The user can now verify the rate limit 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:

```
iperf3 -s
```

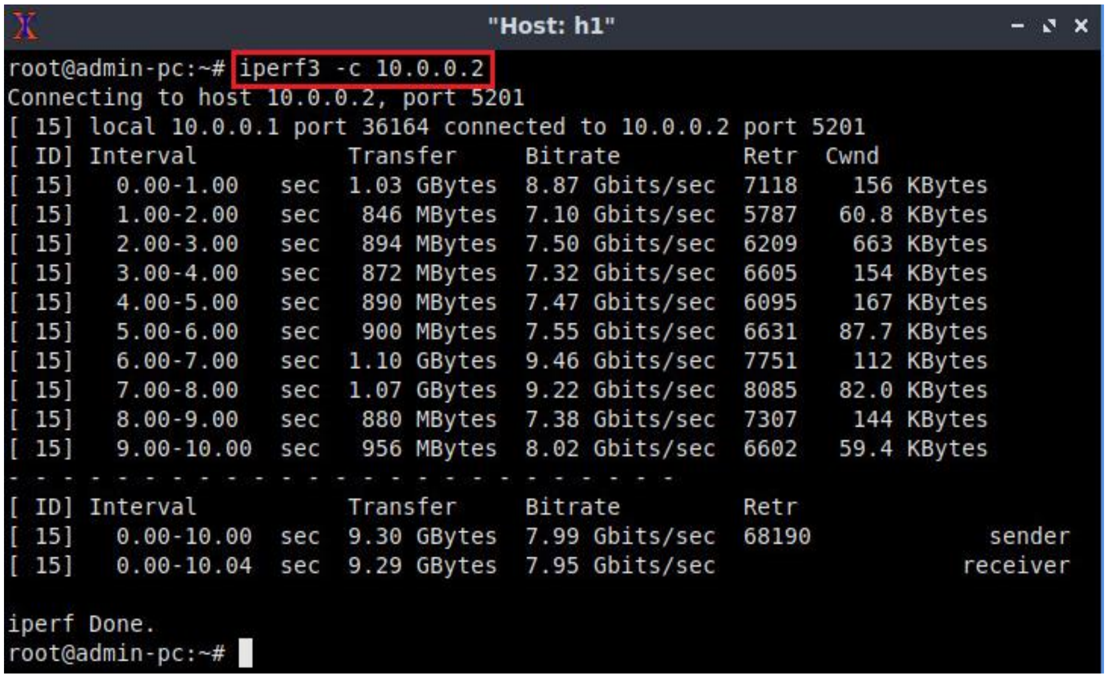


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 12. Host h2 running iPerf3 as server.

Step 5. Now to launch iPerf3 in client mode again by running the command `iperf3 -c 10.0.0.2` in host h1's terminal:

```
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 36164 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec  1.03 GBytes  8.87 Gbits/sec  7118    156 KBytes
[ 15]   1.00-2.00   sec   846 MBytes  7.10 Gbits/sec  5787   60.8 KBytes
[ 15]   2.00-3.00   sec   894 MBytes  7.50 Gbits/sec  6209    663 KBytes
[ 15]   3.00-4.00   sec   872 MBytes  7.32 Gbits/sec  6605    154 KBytes
[ 15]   4.00-5.00   sec   890 MBytes  7.47 Gbits/sec  6095    167 KBytes
[ 15]   5.00-6.00   sec   900 MBytes  7.55 Gbits/sec  6631   87.7 KBytes
[ 15]   6.00-7.00   sec  1.10 GBytes  9.46 Gbits/sec  7751    112 KBytes
[ 15]   7.00-8.00   sec  1.07 GBytes  9.22 Gbits/sec  8085   82.0 KBytes
[ 15]   8.00-9.00   sec   880 MBytes  7.38 Gbits/sec  7307    144 KBytes
[ 15]   9.00-10.00  sec   956 MBytes  8.02 Gbits/sec  6602   59.4 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  9.30 GBytes  7.99 Gbits/sec  68190             sender
[ 15]   0.00-10.04  sec  9.29 GBytes  7.95 Gbits/sec                  receiver

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

Figure 13. iPerf3 throughput test.

Note the measured throughput now is approximately 7.99 Gbps, which is different than the value assigned in the `tbf` rule (10 Gbps). In the next section, the test is repeated but using a higher MSS.

Step 6. 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. The summarized data on the server is similar to that of the client side's and must be interpreted in the same way.

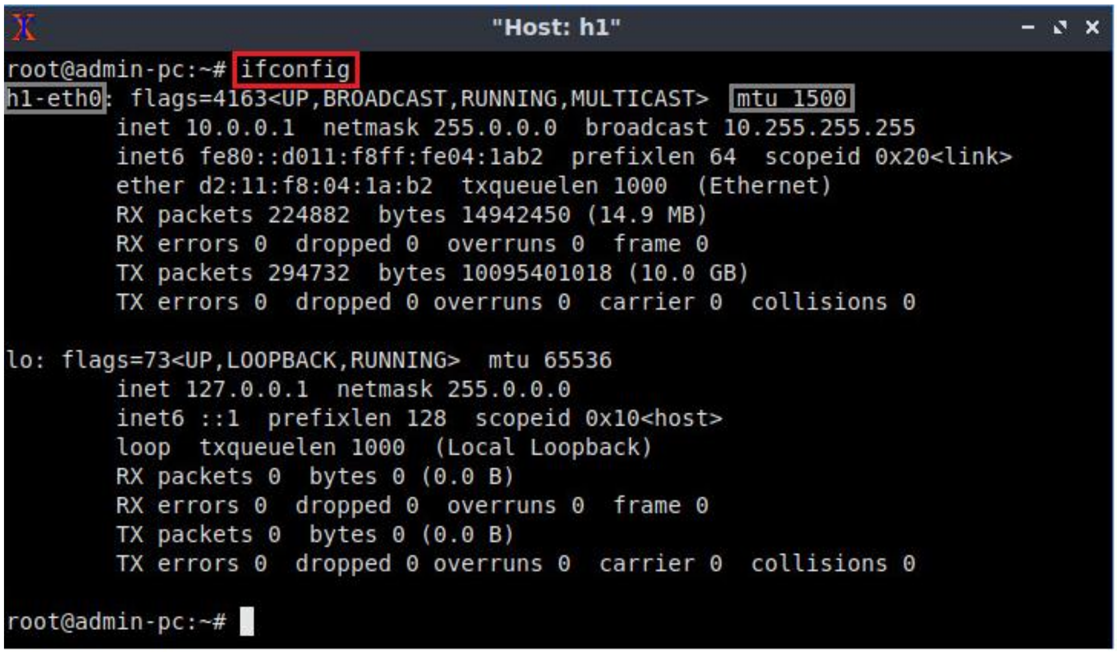
3 Modifying maximum transmission unit (MTU)

As explained previously, jumbo frames offer throughput improvements in networks incurring packet losses. In this section, the user will change the MTU of a network interface in Linux.

3.1 Identifying interface's current MTU

Step 1. To identify the MTU of a network interface of a device, the `ifconfig` is used. On host h1's terminal, type in the following command:

```
ifconfig
```



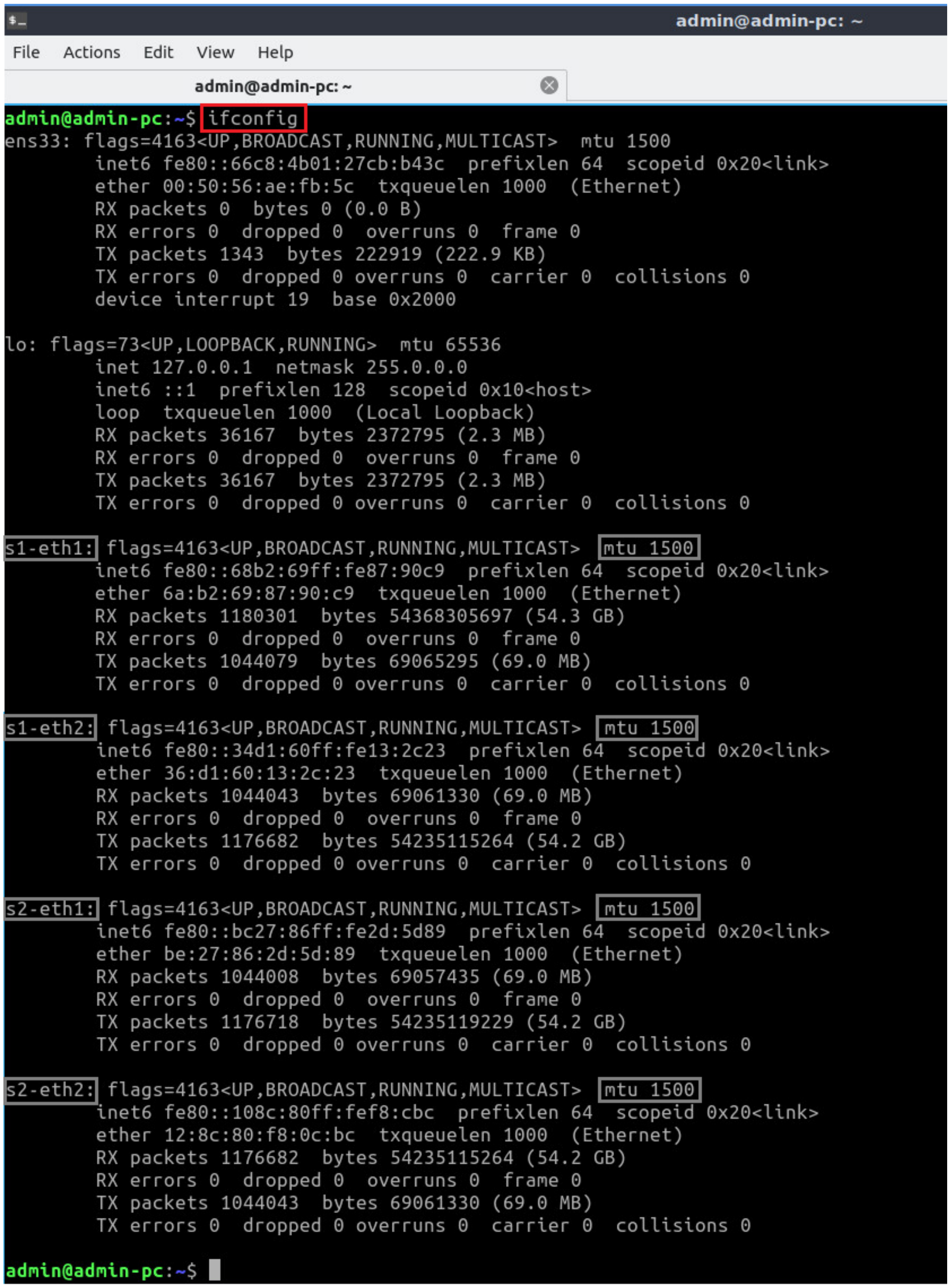
```
root@admin-pc:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::d011:f8ff:fe04:1ab2  prefixlen 64  scopeid 0x20<link>
        ether d2:11:f8:04:1a:b2  txqueuelen 1000  (Ethernet)
        RX packets 224882  bytes 14942450 (14.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 294732  bytes 10095401018 (10.0 GB)
        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 14. Identifying interface's MTU.

As shown in Figure 14, the interface *h1-eth0* has an MTU of 1500 bytes. The same steps can be performed on host h2's interface.

Step 2. In order to identify the MTU on the switches' interfaces, launch the Client's terminal located on the Desktop, and type in the following command:

```
ifconfig
```



```
admin@admin-pc:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::66c8:4b01:27cb:b43c  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:ae:fb:5c  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1343  bytes 222919 (222.9 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 36167  bytes 2372795 (2.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 36167  bytes 2372795 (2.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::68b2:69ff:fe87:90c9  prefixlen 64  scopeid 0x20<link>
        ether 6a:b2:69:87:90:c9  txqueuelen 1000  (Ethernet)
        RX packets 1180301  bytes 54368305697 (54.3 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1044079  bytes 69065295 (69.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::34d1:60ff:fe13:2c23  prefixlen 64  scopeid 0x20<link>
        ether 36:d1:60:13:2c:23  txqueuelen 1000  (Ethernet)
        RX packets 1044043  bytes 69061330 (69.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1176682  bytes 54235115264 (54.2 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s2-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::bc27:86ff:fe2d:5d89  prefixlen 64  scopeid 0x20<link>
        ether be:27:86:2d:5d:89  txqueuelen 1000  (Ethernet)
        RX packets 1044008  bytes 69057435 (69.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1176718  bytes 54235119229 (54.2 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s2-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::108c:80ff:fef8:cbc  prefixlen 64  scopeid 0x20<link>
        ether 12:8c:80:f8:0c:bc  txqueuelen 1000  (Ethernet)
        RX packets 1176682  bytes 54235115264 (54.2 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1044043  bytes 69061330 (69.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

admin@admin-pc:~$ 
```

Figure 15. Identifying switches' interfaces' MTU.

Each switch in the topology has two interfaces: switch S1 has *s1-eth1* and *s1-eth2*, switch S2 interfaces are *s2-eth1* and *s2-eth2*. The MTU value on all interfaces are 1500 bytes.

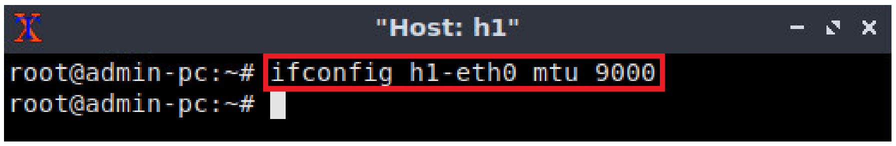
3.2 Modifying MTU values on all interfaces

To modify the MTU of a network interface use the following command:

```
ifconfig <iface> mtu <bytes>
```

Step 1. To change the MTU to 9000 bytes, on host h1's terminal, type in the following command:

```
ifconfig h1-eth0 mtu 9000
```

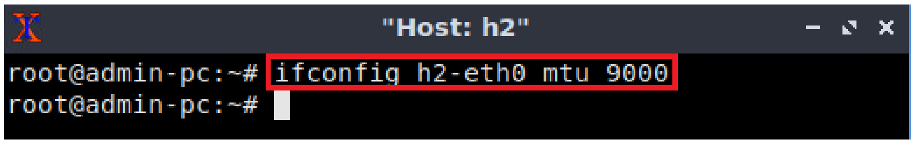


```
root@admin-pc:~# ifconfig h1-eth0 mtu 9000
root@admin-pc:~#
```

Figure 17. Changing host h1's interface MTU.

Step 2. To change the MTU to 9000 bytes, on host h2's terminal, type in the following command:

```
ifconfig h2-eth0 mtu 9000
```



```
root@admin-pc:~# ifconfig h2-eth0 mtu 9000
root@admin-pc:~#
```

Figure 18. Changing host h2's interface MTU.

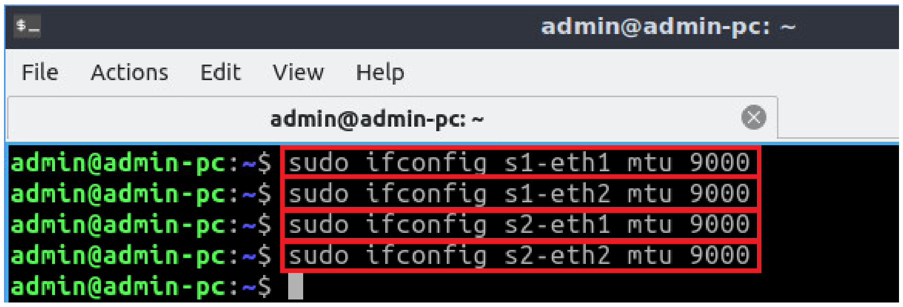
Step 3. Similarly, the MTU values of switch S1 and switch S2's interfaces must be changed to 9000 bytes. In order to modify the MTU values, type the following command on the Client's terminal. When prompted for a password, type `password` and hit *Enter*.

```
sudo ifconfig s1-eth1 mtu 9000
```

```
sudo ifconfig s1-eth2 mtu 9000
```

```
sudo ifconfig s2-eth1 mtu 9000
```

```
sudo ifconfig s2-eth2 mtu 9000
```

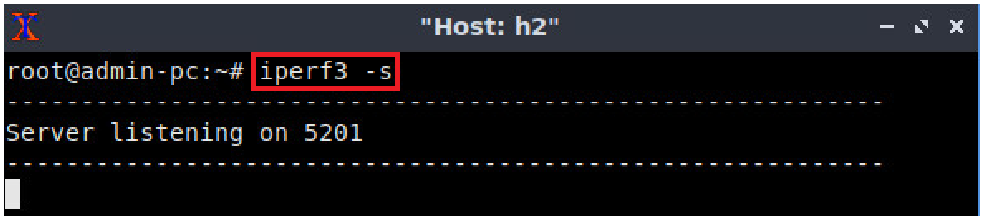


```
admin@admin-pc:~$ sudo ifconfig s1-eth1 mtu 9000
admin@admin-pc:~$ sudo ifconfig s1-eth2 mtu 9000
admin@admin-pc:~$ sudo ifconfig s2-eth1 mtu 9000
admin@admin-pc:~$ sudo ifconfig s2-eth2 mtu 9000
admin@admin-pc:~$
```

Figure 19. Changing MTU values on the switches.

Step 4. The user can now verify the effect of modifying the MTU values on the switches and the effect of MSS by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal:

```
iperf3 -s
```



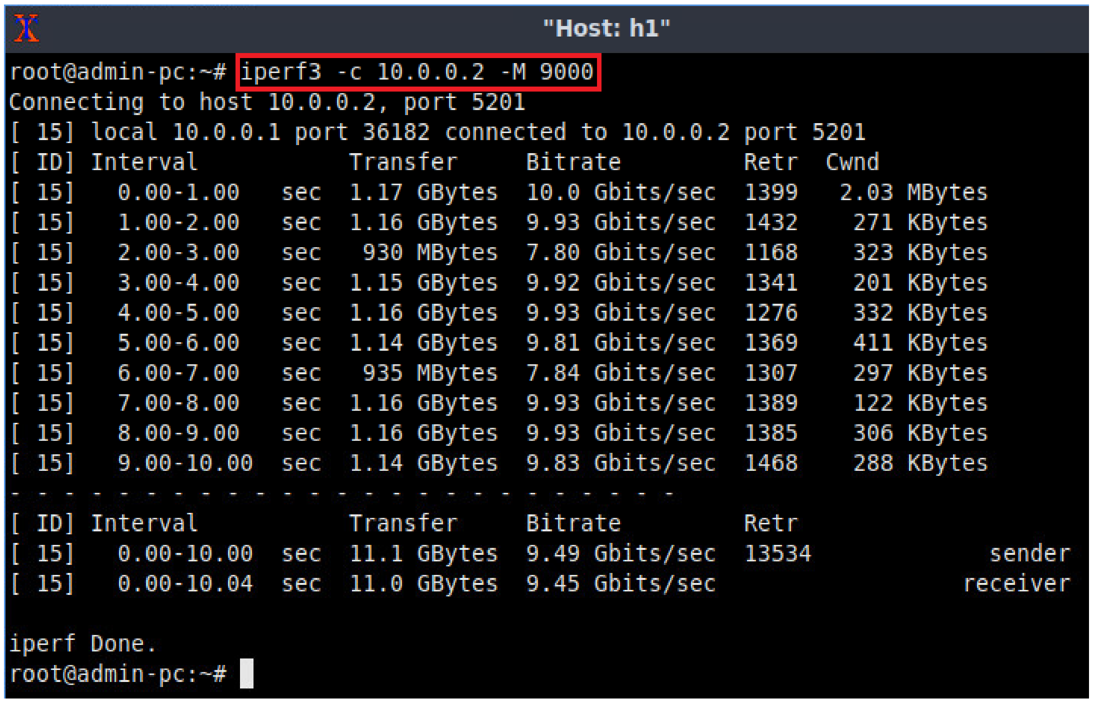
"Host: h2"

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

Figure 20. Host h2 running iPerf3 as server.

Step 5. To launch iPerf3 in client mode type the command below. The `-M` option is used to specify the MSS to be sent in the TCP handshake.

```
iperf3 -c 10.0.0.2 -M 9000
```



"Host: h1"

```
root@admin-pc:~# iperf3 -c 10.0.0.2 -M 9000
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 36182 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec  1.17 GBytes  10.0 Gbits/sec  1399   2.03 MBytes
[ 15]   1.00-2.00   sec  1.16 GBytes  9.93 Gbits/sec  1432    271 KBytes
[ 15]   2.00-3.00   sec   930 MBytes  7.80 Gbits/sec  1168    323 KBytes
[ 15]   3.00-4.00   sec  1.15 GBytes  9.92 Gbits/sec  1341    201 KBytes
[ 15]   4.00-5.00   sec  1.16 GBytes  9.93 Gbits/sec  1276    332 KBytes
[ 15]   5.00-6.00   sec  1.14 GBytes  9.81 Gbits/sec  1369    411 KBytes
[ 15]   6.00-7.00   sec   935 MBytes  7.84 Gbits/sec  1307    297 KBytes
[ 15]   7.00-8.00   sec  1.16 GBytes  9.93 Gbits/sec  1389    122 KBytes
[ 15]   8.00-9.00   sec  1.16 GBytes  9.93 Gbits/sec  1385    306 KBytes
[ 15]   9.00-10.00  sec  1.14 GBytes  9.83 Gbits/sec  1468    288 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  11.1 GBytes  9.49 Gbits/sec  13534             sender
[ 15]   0.00-10.04  sec  11.0 GBytes  9.45 Gbits/sec                  receiver

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

Figure 21. iPerf3 throughput test with a 9000 MSS value.

Notice the measured throughput now is approximately 10 Gbps, which is similar to the value assigned in the `tbf` rule (10 Gbps).

Step 6. 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. The summarized data on the server is similar to that of the client side's and must be interpreted in the same way.

This concludes Lab 13. Stop the emulation and then exit out of MiniEdit.

References

1. Huh, Eui-Nam, and Hyunseung Choo, “Performance enhancement of TCP in high-speed networks,” *Information Sciences* 178, no. 2 (2008), 352-362

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].

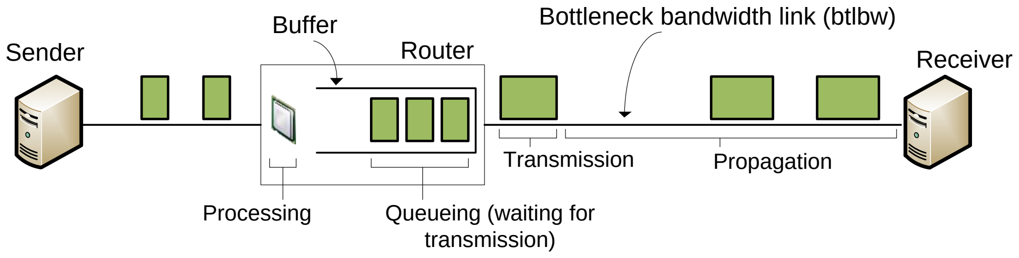

Sender
Buffer
Router
Bottleneck bandwidth link (btlbw)
Receiver
Transmission
Propagation
Processing
Queueing (waiting for transmission)

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

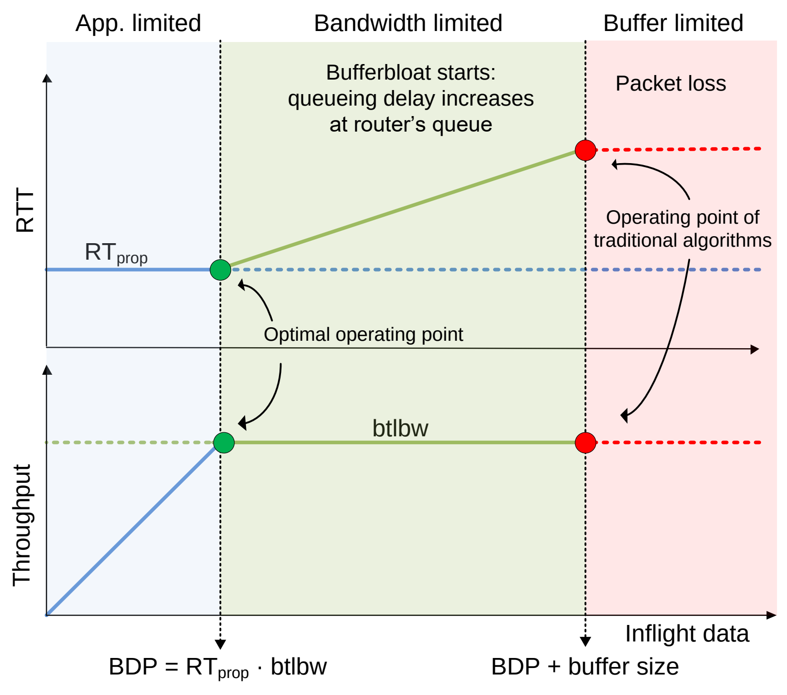
In modern networks composed of high-speed routers and switches, the processing and transmission delays may be negligible. The propagation delay can be considered as a constant (i.e., it has a fixed value). Finally, the dynamics of the queues in routers results in varying queueing delays. Ideally, this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

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

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.



● Optimal operating point
● Operating point of traditional congestion control algorithms

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.

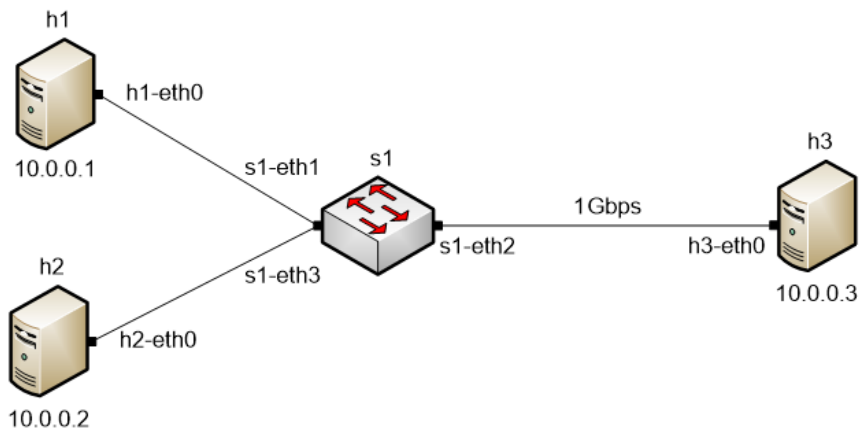


Figure 3. Lab topology.

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`.

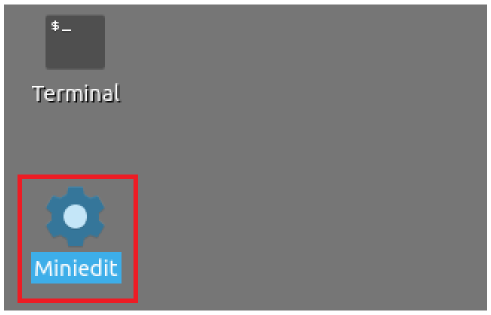


Figure 4. MiniEdit shortcut.

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*.

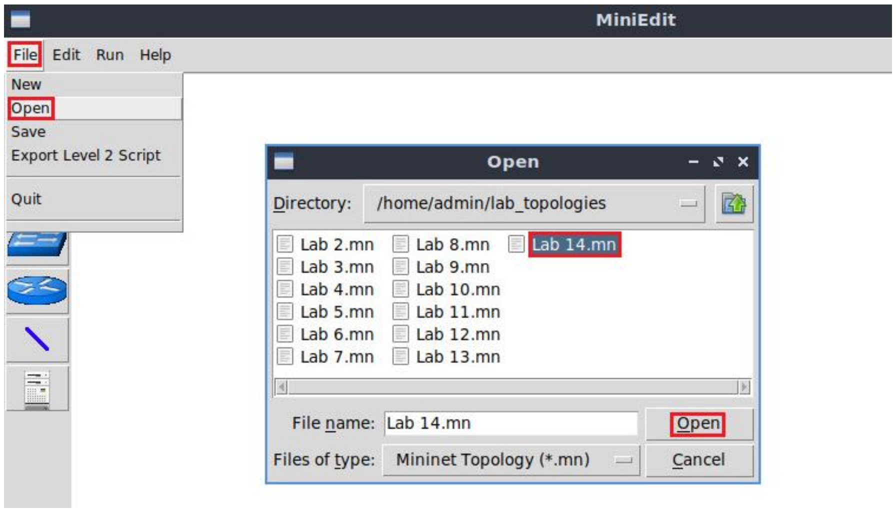


Figure 5. MiniEdit's *Open* dialog.

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.

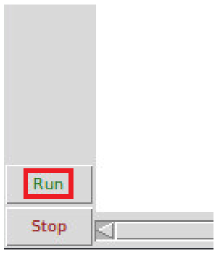


Figure 6. Running the emulation.

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.

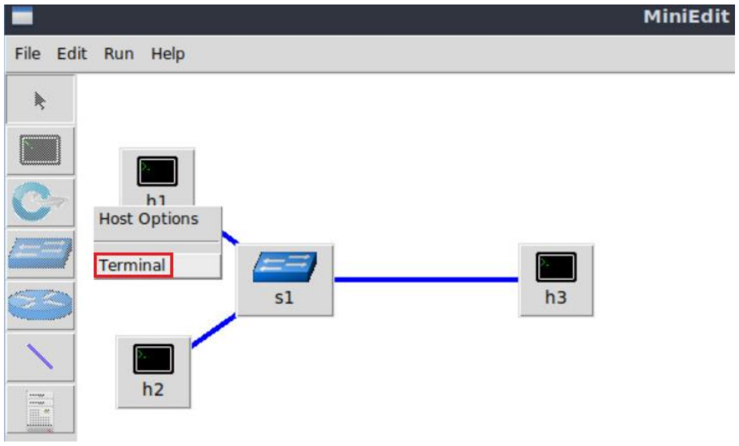
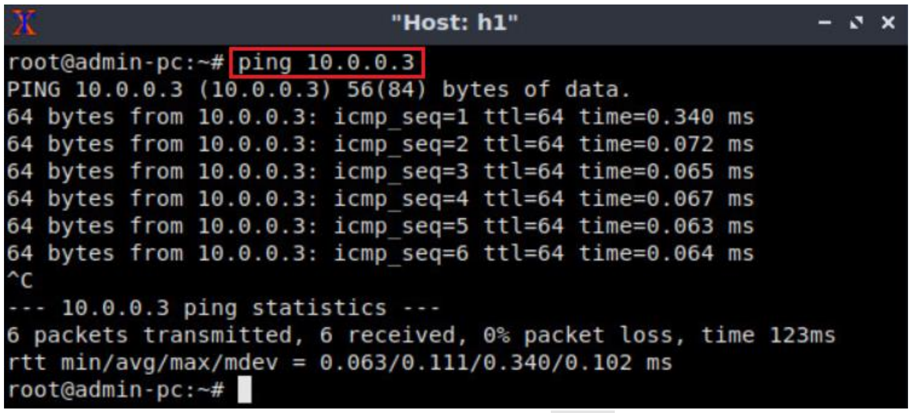


Figure 7. Opening a terminal on host h1.

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.

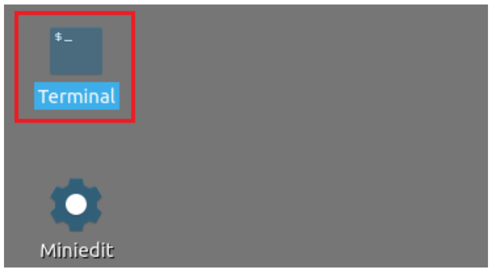
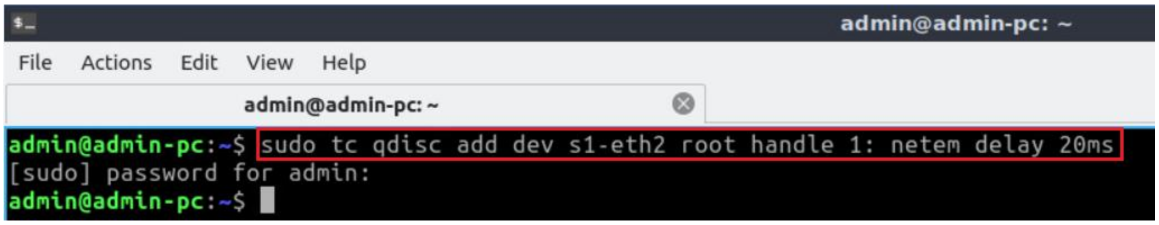


Figure 9. Shortcut to open a Linux terminal.

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
```



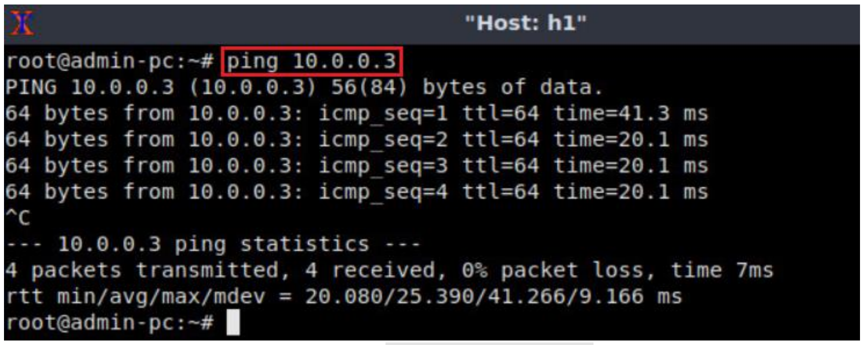
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$ 
```

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.

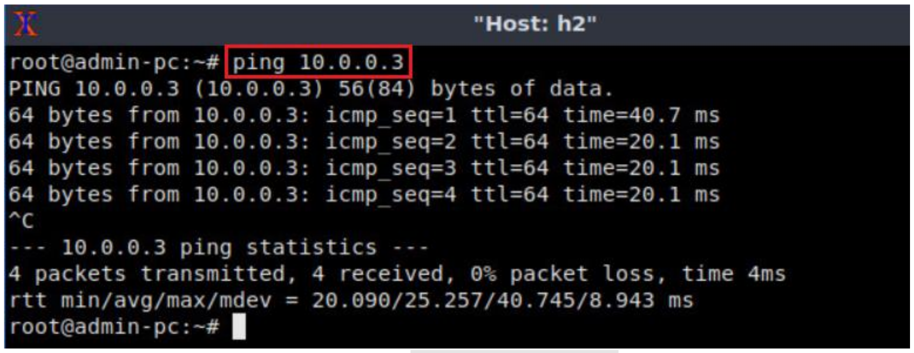


```
root@admin-pc:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=41.3 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=20.1 ms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



```
"Host: h2"
root@admin-pc:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=40.7 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 4ms
rtt min/avg/max/mdev = 20.090/25.257/40.745/8.943 ms
root@admin-pc:~#
```

Figure 12. Output of `ping 10.0.0.3` command.

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

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

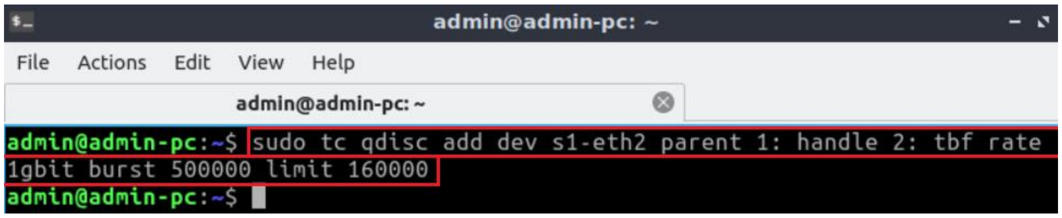
In this section, you are going to change the switch S1's buffer size to 100·MTU and emulate a 1 Gbps Wide Area Network (*WAN*) using the Token Bucket Filter (`tbf`). Then, you will test the throughput between host h1 and host h3. In this section, the MTU is 1600 bytes, thus the `tbf` limit value will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

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

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

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

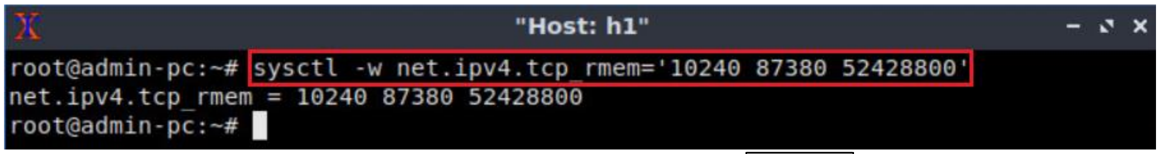
$$1 \text{ Mbyte} = 1024^2 \text{ bytes}$$

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

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’
```



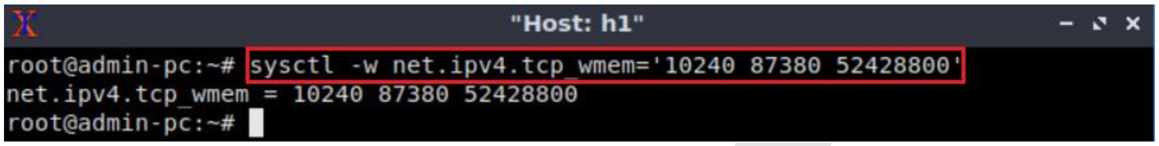
```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~# 
```

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'
```

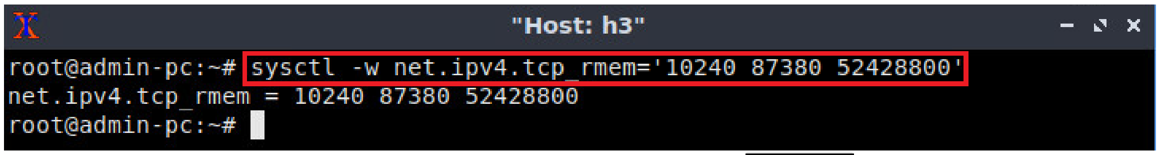


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
net.ipv4.tcp_wmem = 10240 87380 52428800
root@admin-pc:~#
```

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'
```

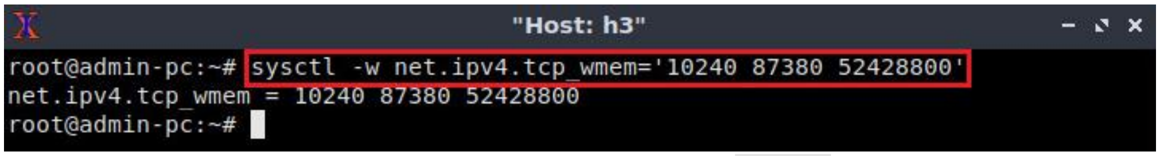


```
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
net.ipv4.tcp_rmem = 10240 87380 52428800
root@admin-pc:~#
```

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:~#
```

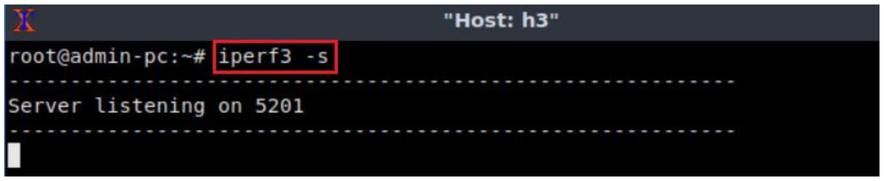
Figure 17. Send window change in `sysctl`.

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
```

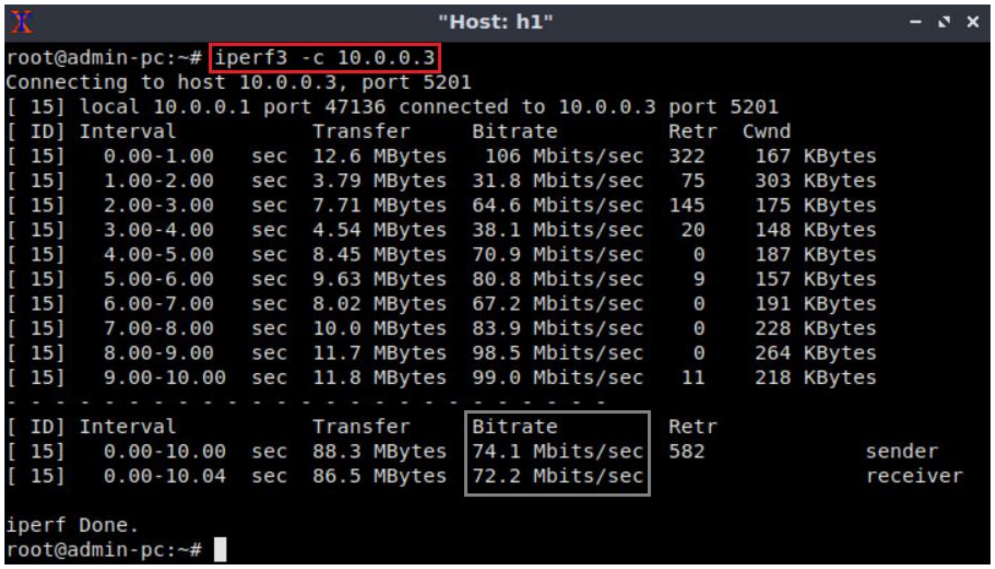


```
"Host: h3"
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

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
```



```
"Host: h1"
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

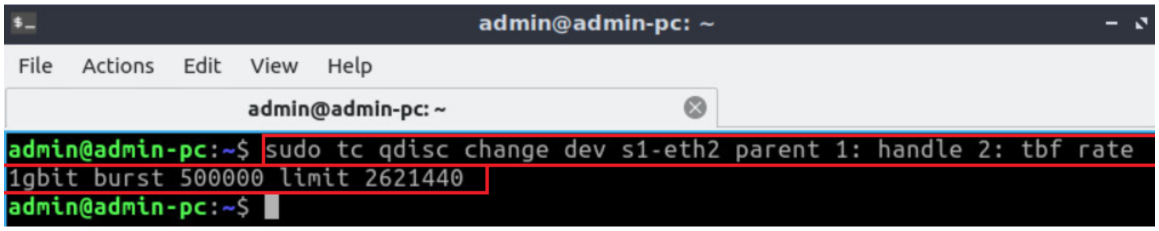
In this section, you are going to change the switch S1's buffer size to 1·BDP and emulate a 1 Gbps Wide Area Network (*WAN*) using the Token Bucket Filter (`tbf`). Then, you will test the throughput between host h1 and host h3. The BDP is 2,621,440 bytes, thus the `tbf` limit value will be set to 2,621,440.

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
```



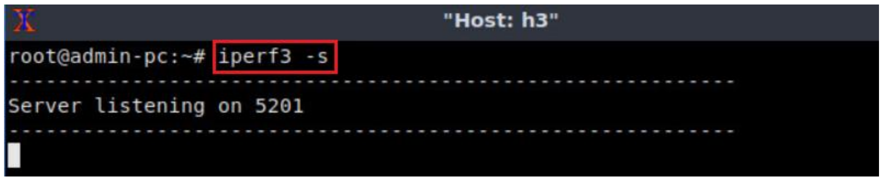
```
admin@admin-pc:~$ sudo tc qdisc change dev s1-eth2 parent 1: handle 2: tbf rate
1gbit burst 500000 limit 2621440
admin@admin-pc:~$
```

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
```

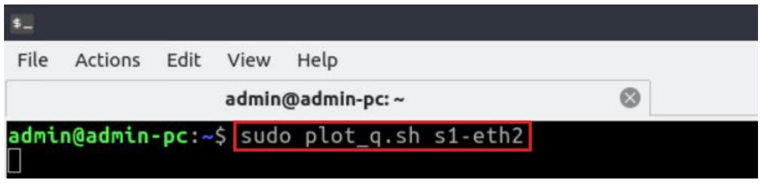


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

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.

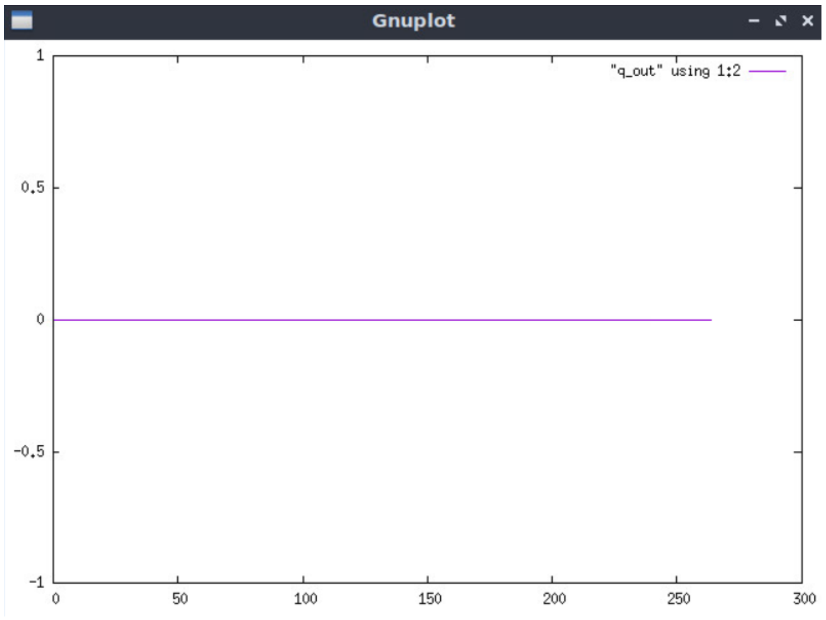
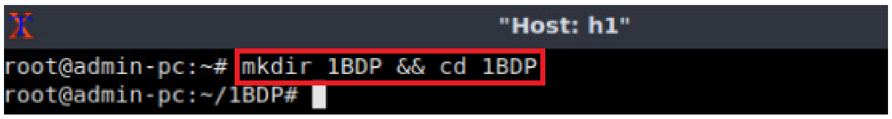


Figure 23. Queue occupancy on switch S1's *s1-eth2* interface.

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

```
mkdir 1BDP && cd 1BDP
```

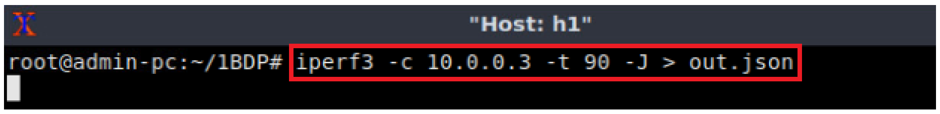


```
root@admin-pc:~# mkdir 1BDP && cd 1BDP
root@admin-pc:~/1BDP# 
```

Figure 24. 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
```

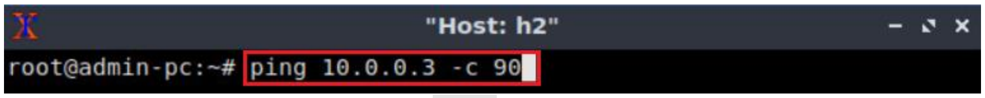


```
root@admin-pc:~/1BDP# iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

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.

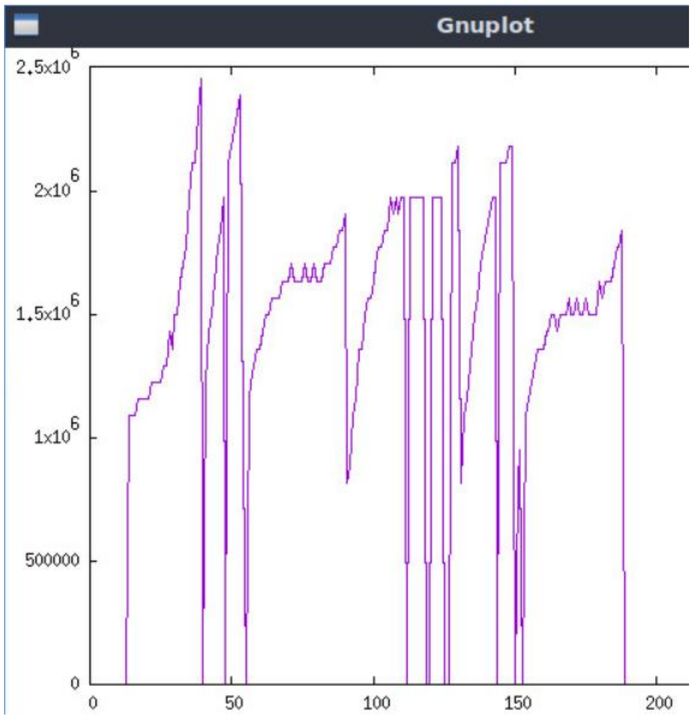


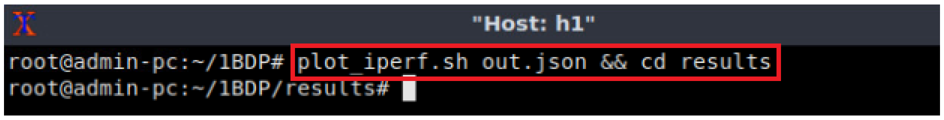
Figure 27. Queue occupancy on switch S1's *s1-eth2* interface.

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
```

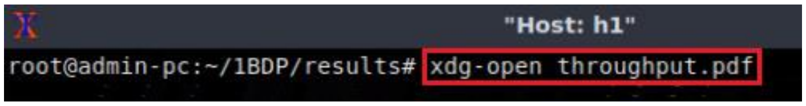


```
root@admin-pc:~/1BDP# plot_iperf.sh out.json && cd results
root@admin-pc:~/1BDP/results#
```

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
```



```
root@admin-pc:~/1BDP/results# xdg-open throughput.pdf
```

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

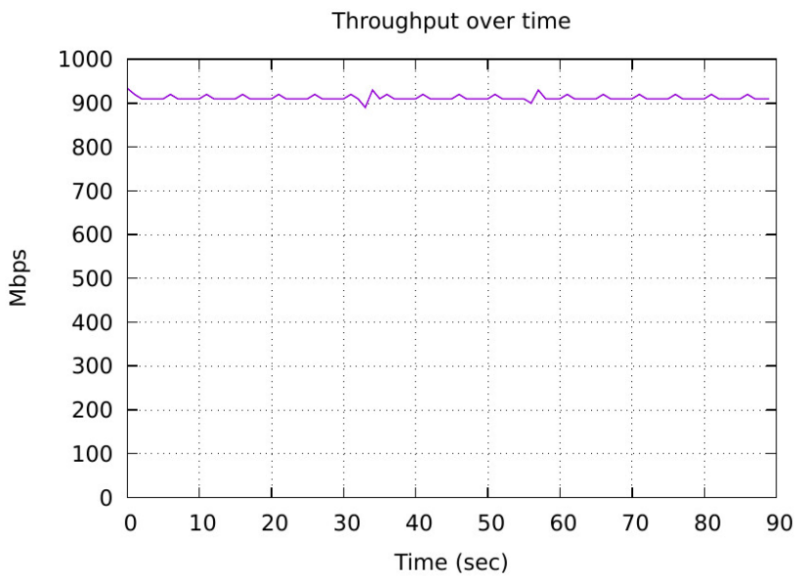
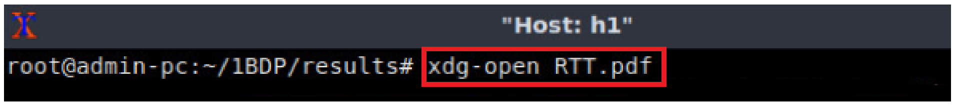


Figure 30. Measured throughput.

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.

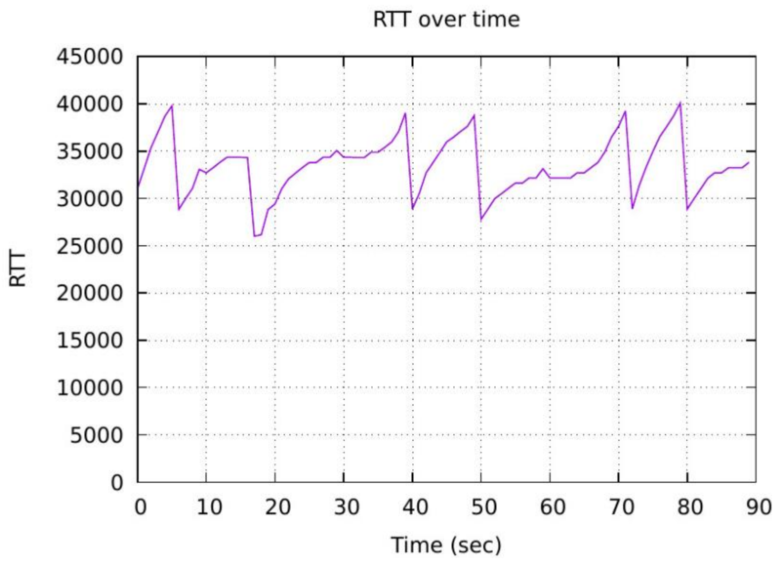
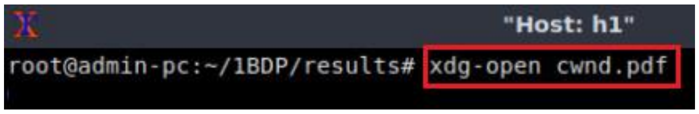


Figure 32. Measured round-trip time.

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.

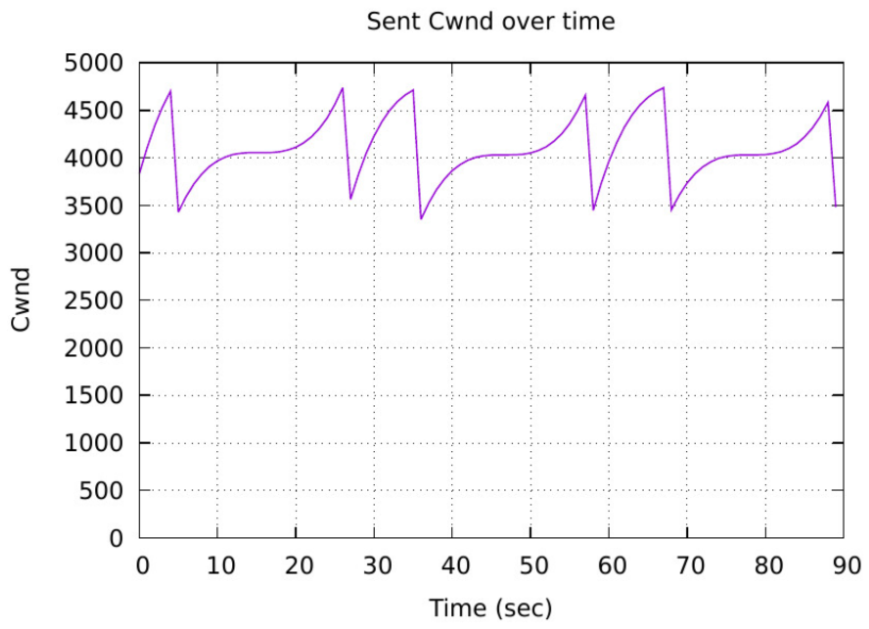
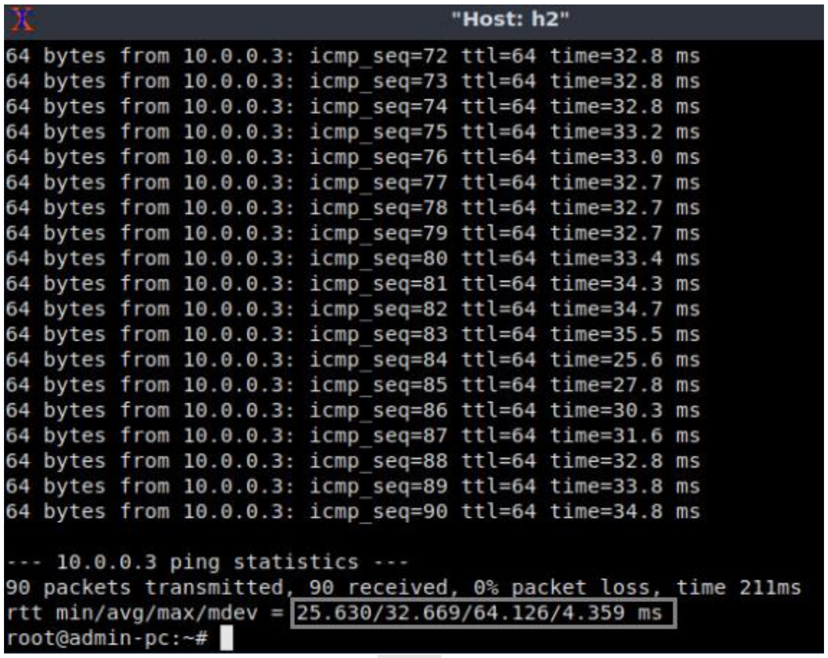


Figure 34. Congestion window evolution.

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

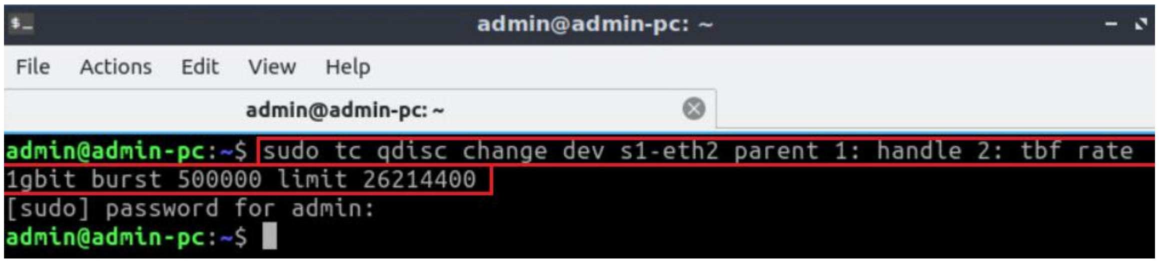
In this section, you are going to change the switch S1's buffer size to 10·BDP and emulate a 1 Gbps Wide Area Network (*WAN*) using the Token Bucket Filter (`tbf`). Then, you will test the throughput between host h1 and host h3. The BDP is 2,621,440 bytes, thus the `tbf` limit value will be set to 26,214,400.

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
```



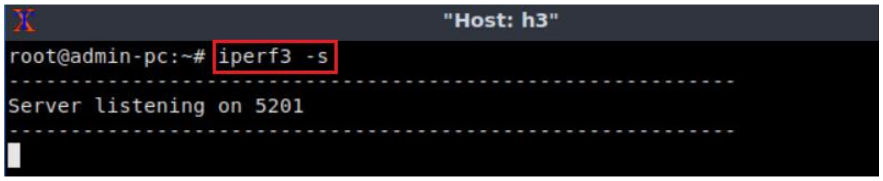
```
admin@admin-pc:~$ sudo tc qdisc change dev s1-eth2 parent 1: handle 2: tbf rate
1gbit burst 500000 limit 26214400
[sudo] password for admin:
admin@admin-pc:~$
```

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
```

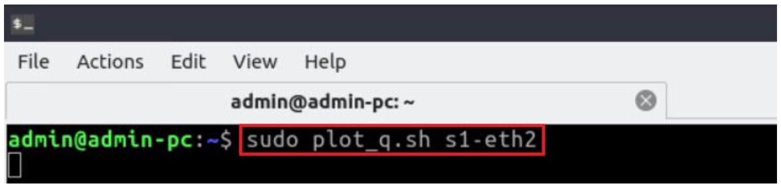


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

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.

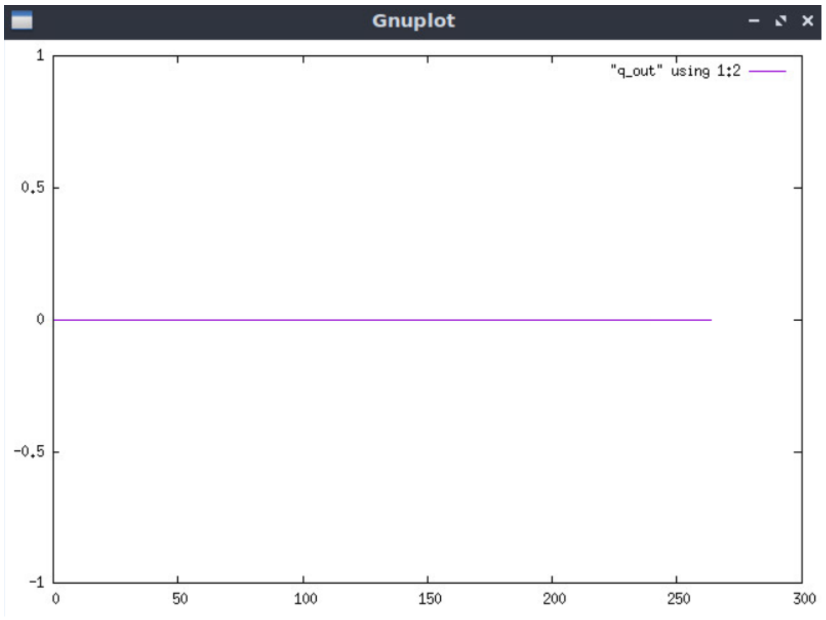
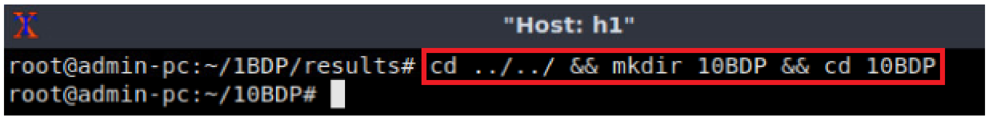


Figure 39. Queue occupancy on switch S1's *s1-eth2* interface.

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
```

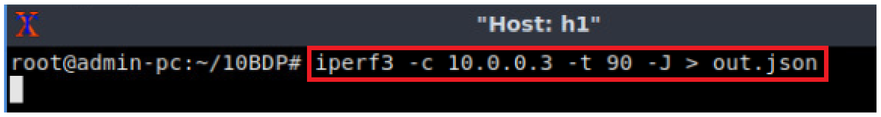


```
root@admin-pc:~/1BDP/results# cd ../../ && mkdir 10BDP && cd 10BDP
root@admin-pc:~/10BDP#
```

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
```

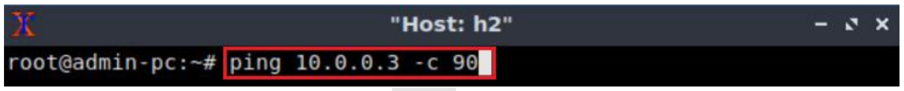


```
root@admin-pc:~/10BDP# iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

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.

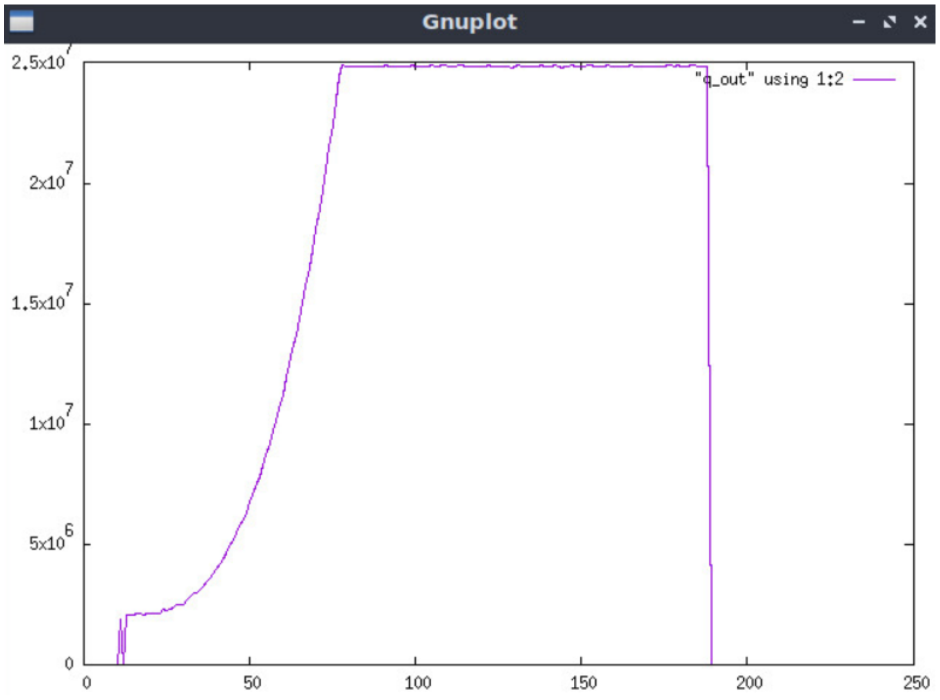


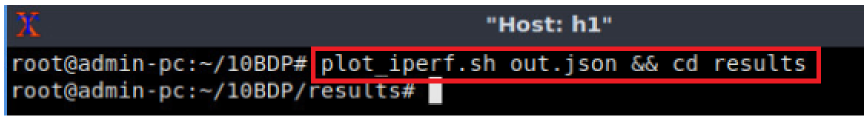
Figure 43. Queue occupancy on switch S1's *s1-eth2* interface.

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
```

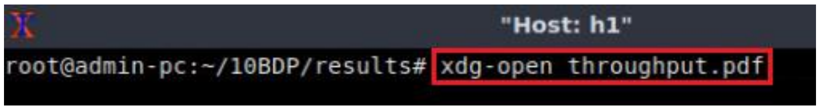


```
root@admin-pc:~/10BDP# plot_iperf.sh out.json && cd results
root@admin-pc:~/10BDP/results#
```

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.

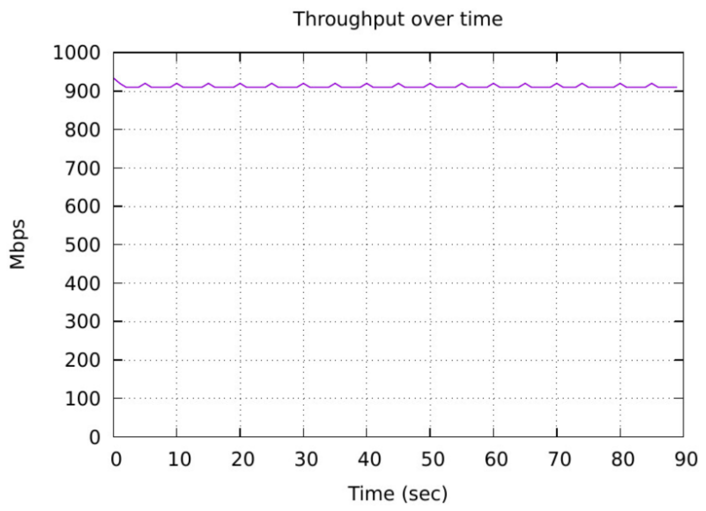
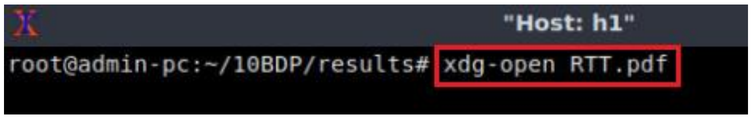


Figure 46. Measured throughput.

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
```



```
root@admin-pc:~/10BDP/results# xdg-open RTT.pdf
```

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

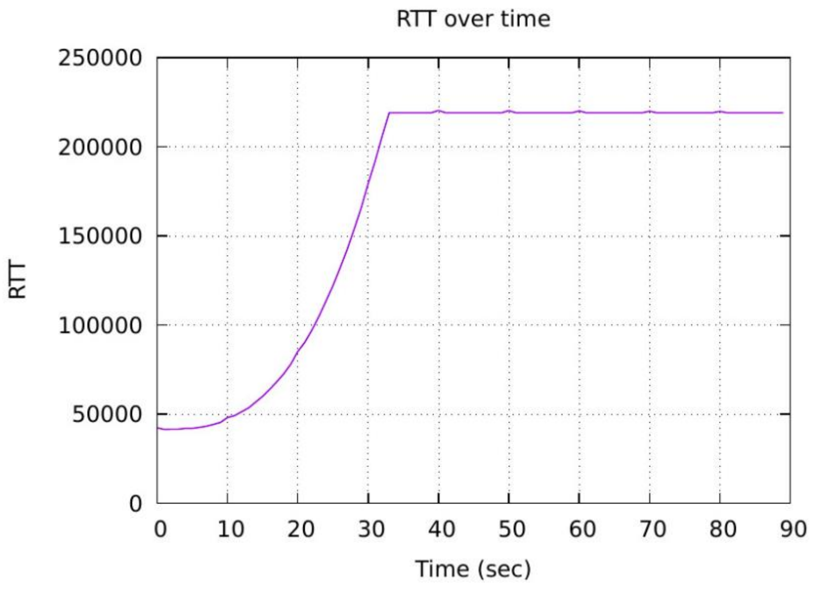
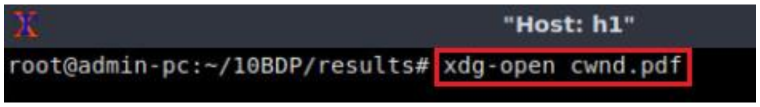


Figure 48. Measured Round-Trip Time.

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.

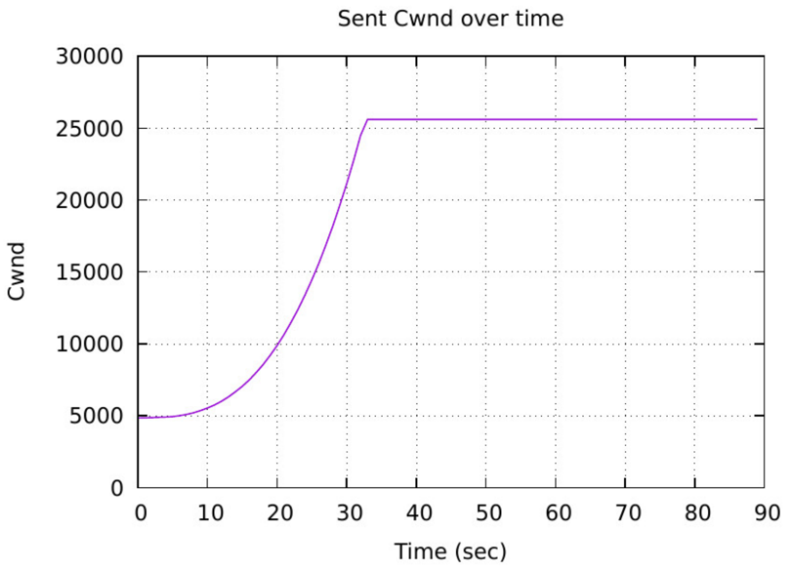
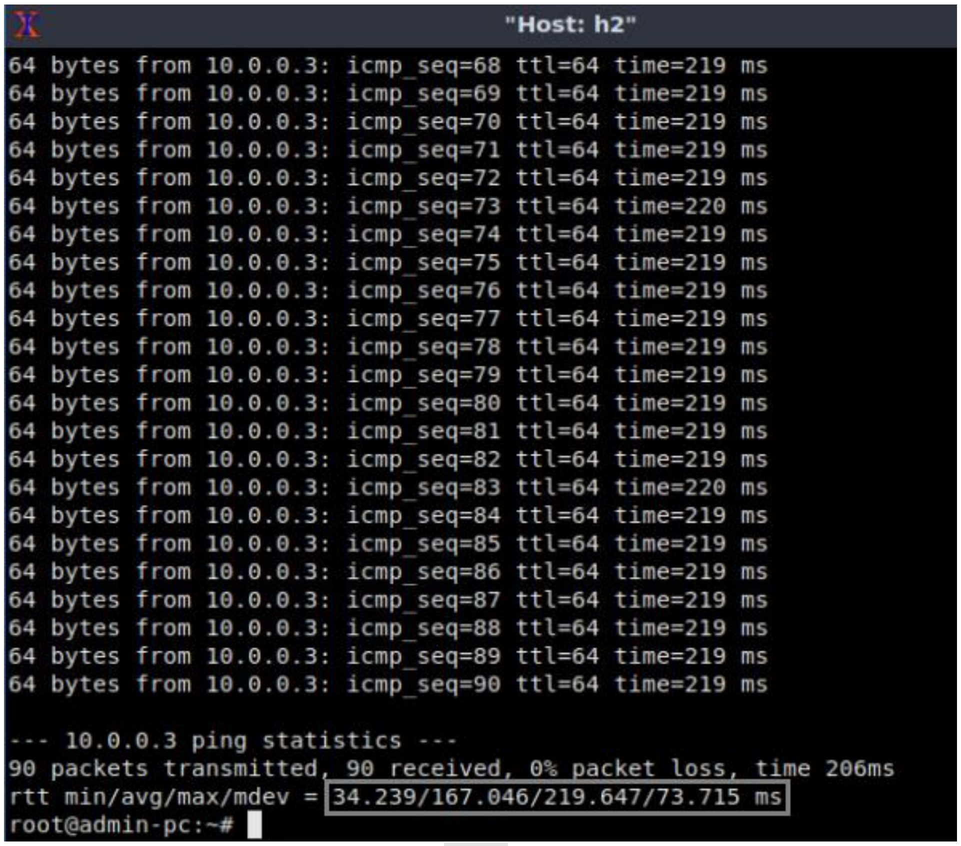


Figure 50. Congestion window evolution.

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.