NETWORK TOOLS AND PROTOCOLS
LAB SERIES

Book Version: 01-25-2021

Principal Investigator: Jorge Crichigno

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

Lab 1: Introduction to Mininet
Exercise 1: Building a Basic Topology
Lab 2: Introduction to iPerf
Lab 3: Emulating WAN with NETEM I Latency, Jitter
Lab 4: Emulating WAN with NETEM II Packet Loss, Duplication, Reordering, and Corruption
Lab 5: Setting WAN Bandwidth with Token Bucket Filter (TBF)
Exercise 2: Emulating a Wide Area Network (WAN)
Problem 1: Troubleshooting a WAN
Lab 6: Understanding Traditional TCP Congestion Control (HTCP, Cubic, Reno)
Lab 7: Understanding Rate-based TCP Congestion Control (BBR)
Lab 8: Bandwidth-delay Product and TCP Buffer Size
Exercise 3: Tuning TCP and Switch's Buffer Size
Exercise 4: Running Tests with Competing TCP Flows and Different Congestion Control Algorithms
Lab 9: Enhancing TCP Throughput with Parallel Streams
Exercise 5: Enhancing the Aggregate TCP Throughput with Parallel Streams
Problem 2: Enhancing TCP Throughput
Lab 10: Measuring TCP Fairness
Exercise 6: RTT Unfairness
Problem 3: Minimizing the Unfairness
Lab 11: Router’s Buffer Size
Lab 12: TCP Rate Control with Pacing
Exercise 7: Setting the Pacing Rate
Lab 13: Maximum Segment Size (MSS)
Lab 14: Router’s Bufferbloat
Exercise 8: Router’s Bufferbloat
Lab 15: Analyzing the Impact of Hardware Offloading on TCP Performance
Lab 16: Random Early Detection
Lab 17: Stochastic Fair Queueing
Lab 18: Controlled Delay (CoDel) Active Queue Management
Lab 19: Proportional Integral Controller-Enhanced (PIE)
Lab 20: Classifying TCP traffic using Hierarchical Token Bucket (HTB)
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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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. 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.

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\textsuperscript{2} and Software-Defined Networking (SDN)\textsuperscript{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\textsuperscript{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.

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.

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.

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.

![Mininet's default minimal topology](image1)

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's help command](image2)

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
```

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's net command](image1.png)

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`.

![Output of h1 ifconfig command](image2.png)
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.

![Figure 10. Connectivity test between host h1 and host h2.](image)

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

![Figure 11. Stopping the emulation using exit.](image)
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.](image)

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.](image)

MiniEdit will start, as illustrated below.

![Figure 14. MiniEdit Graphical User Interface (GUI).](image)
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.](image)

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.
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.

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

![Figure 19. Terminals at host h1 and host h2.](image1)

**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.

![Figure 20. Output of `ifconfig` command on host h1.](image2)

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.

![Figure 21. Connectivity test using ping command.](image)

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

![Figure 22. Stopping the emulation.](image)

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.
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.

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.
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.

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.](image)

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.](image)

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**

NETWORK TOOLS AND PROTOCOLS

Exercise 1: Building a Basic Topology

Document Version: 08-25-2021

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

In this exercise, you will build a topology and run Mininet commands to verify the configuration. Additionally, you will perform a connectivity test.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Deliverables

Follow the steps below to complete the exercise.

a) In MiniEdit, build the following topology.

![Topology Diagram]

b) Before running the emulation click on Edit->Preferences and set the start CLI box.

c) In the Mininet CLI, verify the name of the interfaces, links, and nodes in the topology.

d) Verify the IP and MAC addresses in the hosts.

e) Perform a connectivity test between host h1 and host h2.
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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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.

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.

**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`. 
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.

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.

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.

![Opening a terminal on host h1](image)

**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.

![Connectivity test using ping command](image)

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 ]
```
Lab 2: Introduction to Iperf3

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.](image)

**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
```

![Figure 9. Host h2 running iPerf3 server.](image)

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
```
Figure 10. Host h1 running iPerf3 as client.

The parameter `-c` in the 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 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 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.

```bash
iperf3 -s
```

Figure 11. Host h2 running iPerf3 as server.

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

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

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

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

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
```
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.

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

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

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
```

![Figure 19. Host h2 running iPerf3 as server on port 3250.](image)

**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
```

![Figure 20. Host h2 running on port 3250.](image)

**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).
Lab 2: Introduction to Iperf3

**Step 1.** Start the iPerf3 server on host h2.

```
iperf3 -s
```

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
```

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
```

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+d` 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.
Step 2. Start the iPerf3 client on host h1.

```bash
iperf3 -c 10.0.0.2
```

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.

```bash
iperf3 -s
```
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
```

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

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

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.

```
ls
```

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

```
cd results/
```
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
```

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

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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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.
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\textsuperscript{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.](image)

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 $2 \times 10^8$ meters/sec to $3 \times 10^8$ 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.

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.

**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`.

**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.
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.

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.
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.

![Connectivity test using ping command.](image)

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:

```bash
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.

![ifconfig output on h1](image)

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.
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.

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
```

![Figure 11. Verifying latency after emulating delay using `ping`.](image)

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
```

![Figure 12. Adding 100ms delay to the interface h2-eth0.](image)

**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
```
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
```

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
```

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
```
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
```

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
```

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

As a result, the `tc qdisc` 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
```

![Figure 19. Verifying latency after deleting all rules on both devices.](image)

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 (\( \mu \)), standard deviation (\( \sigma \)), and correlation. By default, NETEM uses a uniform distribution, so that the delay is within \( \mu \pm \sigma \).

#### 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 \( \pm 10 \) 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
```

![Figure 20. Add 100ms delay with \( \pm 10 \) millisecond.](image)
The new value added here represents jitter which defines the delay variation. Therefore, all packets leaving host h1 via interface \textit{h1-eth0} will experience a delay of 100ms, with a random variation of ± 10ms.

\textbf{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 \texttt{ping} command on host h1’s terminal:

```
ping 10.0.0.2
```

![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.

\textbf{Step 3.} In host h1’s terminal, type the following command to delete previous configurations:

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

![Figure 22. Deleting all rules on interface h1-eth0.]

6 \hspace{1cm} \textbf{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%
```

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
```

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
```

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.

![Figure 26. Adding normal distribution of delay.](image)

**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
```

![Figure 27. Verifying latency after using normal distribution.](image)

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

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.

Lab settings

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

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction to network emulators and NETEM.
2. Section 2: Lab topology.
4. Section 4: Adding packet corruption.
5. Section 5: Adding packet reordering.

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.

![Diagram of Delay, Loss, Reordering, Duplication, Corruption](image)

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.](image)

**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.](image)

**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*. 
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.

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.
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.

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`. 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.
• \texttt{[add | del | replace | change | show]}: this is the operation on qdisc. For example, to add delay on a specific interface, the operation will be \texttt{add}. To change or remove delay on the specific interface, the operation will be \texttt{change} or \texttt{del}.

• \texttt{dev id}: this parameter indicates the interface to be subject to emulation.

• \texttt{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 \texttt{ifconfig} to display information related to its network interfaces and their assigned IP addresses.

![Figure 8. Output of \texttt{ifconfig} command on host h1.](image)

The output of the \texttt{ifconfig} command indicates that host h1 has two interfaces: \texttt{h1-eth0} and \texttt{lo}. The interface \texttt{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 \texttt{tc} when emulating the WAN.

**Step 2.** In host h2, type the command \texttt{ifconfig} as well.
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 the `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%
```

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

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.

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%
```
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:

```bash
ping 10.0.0.2 -c 200
```

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.
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:

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

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

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

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:

```bash
ping 10.0.0.2
```
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%
```

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
```
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:

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

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:

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

The new value added here represents packet corruption percentage (0.01%).
Lab 4: Emulating WAN with NETEM II: Packet Loss, Duplication, Reordering, and Corruption

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
```

![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
```

![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
```
Step 5. In order to stop the server, press \[\text{Ctrl}+\text{c}\] 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 \texttt{reorder} option is used. Proceed with the steps below.

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

\begin{Verbatim}
sudo tc qdisc add dev h1-eth0 root netem delay 10ms reorder 25% 50%
\end{Verbatim}

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 \texttt{ping} command on host h1’s terminal, press \[\text{Ctrl}+\text{c}\] to stop the test:

\begin{Verbatim}
ping 10.0.0.2
\end{Verbatim}

Figure 27. \texttt{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
```

![Figure 28. Deleting all rules on interface h1-eth0.](image)

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, proceed 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%
```

![Figure 29. Adding packet duplication.](image)

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
```
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:

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

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

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

**References**

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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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.

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*. 
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.
- `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.
- `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.
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`.

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.

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.
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.

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.
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.

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 when emulating the network.

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

![Figure 10. Output of `ifconfig` command on host h2.](image)

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 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
```

![Figure 10. Limiting rate with TBF to 10 Gbps.](image)
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 command shown below.

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

![Figure 11. Retrieving system’s HZ.](image)

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

10 Gbps = 10,000,000,000 bps

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

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

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

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

```
iperf3 -s
```

![Figure 12. Host h2 running iPerf3 as server.](image)
**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
```

![iPerf3's report after limiting the rate on host h1 to 10 Gbps.](image)

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.

![Shortcut to open a Linux terminal.](image)

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

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
```

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

**Step 4.** Apply the 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
```

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 $\text{iperf3}$ tool to measure throughput. To launch iPerf3 in server mode, run the command $\text{iperf3 -s}$ in host h2’s terminal as shown in Figure 18:

```
iperf3 -s
```

![Figure 18. Host h2 running iPerf as server.](image)

Step 6. Now to launch iPerf3 in client mode, run the command $\text{iperf3 -c 10.0.0.2}$ in host h1’s terminal as shown in the figure below:

```
iperf3 -c 10.0.0.2
```

![Figure 19. iPerf3’s report after limiting the rate on switch S1 to 10 Gbps.](image)

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 $\text{Ctrl+c}$ 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.](image)

As shown in Figure 20, the first qdisc (qdisc1) is attached to the root label. Then, subsequent qdiscs can be attached to their parents by specifying the correct label. In this section, we will look at how to combine NETEM and TBF in order to have more properties emulated in our network. Specifically, we will introduce delay, jitter, and packet corruption, while specifying the rate on switch S1’s interface.

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

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

![Figure 21. Deleting all rules on switch S1’s s1-eth2.](image)

**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
```
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
```

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**: 2,500,000

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

Figure 25. Host h2 running iPerf 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
```

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**

NETWORK TOOLS AND PROTOCOLS

Exercise 2: Emulating a Wide Area Network (WAN)

Document Version: 08-25-2021

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

1   Exercise description ........................................................................................................... 3
1.1  Credentials ..................................................................................................................... 3
2   Setting the environment .................................................................................................... 3
3   Deliverables ...................................................................................................................... 4
Exercise 2: Emulating a Wide Area Network (WAN)

1 Exercise description

In this exercise, you emulate a WAN. This task requires setting the bottleneck link bandwidth and delay. Additionally, you will emulate packet losses and run a throughput test to verify the performance.

![Network Diagram]

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

Step 1. Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type `password`.

Step 2. Load the topology located at `/home/admin/NTP_Exercises/Exercise2/`. 
Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

3 Deliverables

Follow the steps below to complete the exercise.

a) Set the bottleneck link to 100 Mbps and add a 25ms latency to the interfaces s1-eth2 and s2-eth1.

b) Run an iPerf3 test for 120 seconds and explain the results. What is the throughput?

c) Add a 2% packet loss rate to the interface s1-eth2 and repeat part b).
NETWORK TOOLS AND PROTOCOLS

Problem 1: Troubleshooting a Wide Area Network (WAN)

Document Version: 09-02-2021

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

1 Problem description........................................................................................................................................3
  1.1 Credentials ........................................................................................................................................3
2 Setting the environment.........................................................................................................................3
1 Problem description

Find and troubleshoot the issue that is causing low throughput from host h1 to host h2. Verify that the interfaces are correctly configured to set the bottleneck bandwidth to 1 Gbps and the latency to 50ms. Once the problem is found, apply the changes that will guarantee the bottleneck link is fully utilized.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

**Step 1.** Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type `password`.

**Step 2.** Load the topology located at `~/NTP_Problems/Problem1/`. 
Problem 1: Troubleshooting a Wide Area Network (WAN)

Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.

Step 5. Navigate into the following directory ~/NTP_Problems/Problem1/ by issuing the following command:

```
admin@admin-pc:~$ cd ~/NTP_Problems/Problem1/
```

Step 6. Run the command below. If a password is required, type password.

```
sudo ./set_env.sh
```
Problem 1: Troubleshooting a Wide Area Network (WAN)

The script sets the bottleneck link to 1 Gbps and the latency to 50ms. Now, you can start solving the problem.

**Hint.** To see the configuration of each interface (e.g., end hosts and switches), consider the following command:

```
sudo tc -s qdisc show dev <interface_name>
```
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.
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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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.](image)

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 \textit{cwnd} (in bytes), imposes a constraint on the rate at which a TCP sender can send traffic. The \textit{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 \textit{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 \( \frac{\text{cwnd}}{\text{RTT}} \) bytes/sec. By adjusting the value of \textit{cwnd}, the sender can therefore adjust the rate at which it sends data into the connection.

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

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 \textit{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 \textit{cwnd} by half, and as long as no other packet loss is detected, TCP will slowly increase \textit{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\textsuperscript{4}, HTCP\textsuperscript{5}, and BBR\textsuperscript{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.

![Lab topology diagram](image)

**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`.

![MiniEdit shortcut](image)

**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*.

![MiniEdit open file](image)

**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.](image)

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.](image)

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.
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.

![Linux terminal icon](image)

**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%
```
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
```

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=0.875 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
--- 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
```

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. 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
```

![Figure 14. Listing all system parameters in Linux.](image)

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:
3.2 Write sysctl parameters

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

\begin{verbatim}
sysctl -w \texttt{<key>=<value>}
\end{verbatim}

**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:

\begin{verbatim}
sudo sysctl -w net.ipv4.ip_forward=1
\end{verbatim}

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

![Image](image16.png)

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 \texttt{/etc/sysctl.conf}.

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

\begin{verbatim}
sudo featherpad /etc/sysctl.conf
\end{verbatim}

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

```bash
net.ipv4.ip_forward=1
```
Step 3. To refresh the system with the new parameters, the `-p` switch is passed to the `sysctl` command as follows:

```bash
sudo sysctl -p
```

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

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
```

![Figure 20. Displaying the system’s available congestion control algorithms.](image)

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:

```bash
sudo modprobe tcp_bic
```

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.

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

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.

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

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
```

![Figure 24. Modifying the congestion control algorithm to Reno.](image)

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
```

![Figure 25. Current TCP congestion control algorithm after modifying to Reno.](image)

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
```

![Figure 26. Changing TCP congestion control algorithm to `reno` on host h1.](image)

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

```
iperf3 -s
```

![Figure 27. Starting iPerf3 server on host h2.](image)

**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
```
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
```
Figure 29. Changing TCP congestion control algorithm to \texttt{hTCP} on host h1.

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

```
iperf3 -s
```

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
```

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:

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

![Figure 32. Changing TCP congestion control algorithm to cubic on host h1.](image)

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

```bash
iperf3 -s
```

![Figure 33. Starting iPerf3 server on host h2.](image)

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

```bash
iperf3 -c 10.0.0.2 -t 20 -o 10
```
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
```
Lab 6: Understanding Traditional TCP Congestion Control

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'
```

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'`.

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

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
```

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

![Figure 39. Starting iPerf3 server on host h2.](image)

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

![Figure 40. Running iPerf3 client on host h1.](image)

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
```

![Figure 41. Changing TCP congestion control algorithm to htcp on host h1.](image)

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

```
iperf3 -s
```

![Figure 42. Starting iPerf3 server on host h2.](image)

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

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

![Figure 43. Running iPerf3 client on host h1.](image)
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
```

![Figure 44. Changing TCP congestion control algorithm to cubic on host h1.](image)

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

```
iperf3 -s
```

![Figure 45. Starting iPerf3 server on host h2.](image)

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

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


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.

Lab settings

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

Table 1. Credentials to access Client1 machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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\textsuperscript{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\textsuperscript{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).

![Image](image.png)

**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\(^4\).

![Throughput vs RTT](image)

**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 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. $RTT_{\text{min}}$ 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 in flight 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% of 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% of 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](image)

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.

![Lab topology diagram]

**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`.

![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*.

![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.

![Running the emulation.](image)

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.

![Opening a terminal on host h1.](image)

**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.
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.](image)

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%
```
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

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

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.
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
```

![Figure 15. Changing TCP congestion control algorithm to Reno on host h1.](image)

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

```
iperf3 -s
```

![Figure 16. Starting iPerf3 server on host h2.](image)
**Step 3.** Launch iPerf3 in client mode on host h1’s terminal. The \(-0\) option is used to specify the number of seconds to omit in the resulting report.

```
iperf3 -c 10.0.0.2 -t 20 -0 10
```

![Figure 17. Running iPerf3 client on host h1.](image)

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 \(\text{Ctrl} + \text{c}\) 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:
Lab 7: Understanding Rate-based TCP Congestion Control (BBR)

sysctl -w net.ipv4.tcp_congestion_control=bbr

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

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

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
```

![Figure 21. Injecting 30ms delay on switch S1’s s1-eth2 interface.](image)

**Step 2.** In host h1’s terminal, modify the TCP buffer size by typing the following commands: `sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'` and `sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'`. 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'
```

![Figure 22. Modifying the TCP buffer size on host h1.](image)

**Step 3.** In host h2’s terminal, also modify the TCP buffer size by typing the following commands: `sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'` and `sysctl -w net.ipv4.tcp_wmem='10240 87380 150000000'`.

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 150000000'
```
Lab 7: Understanding Rate-based TCP Congestion Control (BBR)

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

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

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

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

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
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
```

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/
```

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
```

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

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

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

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

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

Figure 29. Entering the results directory using the `cd` command.
Step 8. Close the *throughput.pdf* file and open the *cwnd.pdf* file using the following command:

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

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 ../..
```

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
```

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
```

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
```

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
```

![Figure 38](image)

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
```

![Figure 39](image)

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/
```

![Figure 40](image)

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
```

![Figure 41](image)

Opening the `throughput.pdf` file using `xdg-open`.
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:

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

![Opening the cwnd.pdf file using xdg-open](image)
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 ../..
```

It is clear from the above test that when introducing delay, BBR performs significantly better than Reno.

This concludes Lab 7. Stop the emulation and then exit out of MiniEdit.

References


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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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.

\[ T_{tx} = \frac{\# \text{bits}}{Bw} = \frac{8,388,608}{10 \cdot 10^9} = 0.84 \text{ milliseconds}. \]

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 \( t = 50 \text{ milliseconds} \). 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

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 with 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 \cdot 8 \text{ bits} = 8,388,608 \text{ bits}). With a bandwidth (Bw) of 10 Gbps, this number of bits is approximately transmitted in
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 \((1 \text{ Mbyte} = 1024^2 \text{ bytes})\) or Gbytes \((1 \text{ Gbyte} = 1024^3 \text{ bytes})\). 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.

![TCP receive buffer](figure.png)

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. 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 $t = \text{RTT}$, 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 $t = 2 \cdot \text{RTT}$. Thus, the throughput over the period $t = 0$ to $t = 2 \cdot \text{RTT}$ is reduced by half:

$$ \text{throughput} = \frac{\text{# bits transmitted}}{\text{period}} = \frac{\text{Bw} \cdot \text{RTT}}{2 \cdot \text{RTT}} = \frac{\text{Bw}}{2}.$$ 

![Figure 3. A scenario where a TCP receive buffer size of BDP cannot prevent throughput degradation.](image)

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 $2 \cdot \text{BDP}$ mitigates packet loss.

<table>
<thead>
<tr>
<th>High to moderate packet loss scenarios, using TCP BBR:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP send/receive buffer $\geq$ several BDPs (e.g., $4 \cdot \text{BDP}$)</td>
</tr>
</tbody>
</table>

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.

![Figure 4. A scenario where a TCP buffer size of 2\cdot\text{BDP} mitigates packet loss.](image)

1.4 \textbf{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\textsuperscript{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 \textit{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: 0x810 (ACK)
- Window size value: 2949
- [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.](image)

**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.](image)

**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.
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.

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.

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
```

![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**: 25,000,000

```
sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit burst 5000000 limit 25000000
```

![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 \texttt{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.

![Figure 15. Output of \texttt{ping 10.0.0.2} command.](image)

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 \texttt{iperf3} tool to measure throughput. To launch iPerf3 in server mode, run the command \texttt{iperf3 -s} in H2’s terminal:

\begin{verbatim}
iperf3 -s
\end{verbatim}

![Figure 16. Host h2 running \texttt{iperf3} as server.](image)

**Step 5.** Now to launch iPerf3 in client mode again by running the command \texttt{iperf3 -c} 10.0.0.2 in h1’s terminal:

\begin{verbatim}
iperf3 -c 10.0.0.2
\end{verbatim}
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
```
Lab 8: BDP and Setting TCP Buffer Size

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
```

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:

\[
\text{BDP} = (10 \cdot 10^9) \cdot (20 \cdot 10^{-3}) = 200,000,000 \text{ bits} = 25,000,000 \text{ bytes} \approx 25 \text{ Mbytes.}
\]

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.
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 $2 \cdot \text{BDP}$ (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 = 1024² bytes).

$$\text{BDP} = 25 \text{ Mbytes} = 25 \cdot 1024² \text{ bytes} = 26,214,400 \text{ bytes}$$

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'
```

![Figure 21. Receive window change in `sysctl`.](image)

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'
```

![Figure 22. Send window change in `sysctl`.](image)

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'
```

![Figure 23. Receive window change in sysctl.](image1)

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'
```

![Figure 24. Send window change in sysctl.](image2)

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
```

![Figure 25. Host h2 running iPerf3 as server.](image3)

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
```
Figure 26. iPerf3 throughput test.

Note the measured throughput now is approximately 10 Gbps, which is close to the value assigned in our rule (10 Gbps).

This concludes Lab 8. Stop the emulation and then exit out of MiniEdit and Linux terminal.

References

NETWORK TOOLS AND PROTOCOLS

Exercise 3: Tuning TCP and Switch’s Buffer Size

Document Version: 08-25-2021

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

1 Exercise description ........................................................................................................... 3
1.1 Credentials ..................................................................................................................... 3
2 Setting the environment ..................................................................................................... 3
3 Deliverables ....................................................................................................................... 5
Exercise 3: Tuning TCP and Switch’s Buffer Size

1 Exercise description

In this exercise, you will emulate a WAN and run throughput tests with different congestion control algorithms. Additionally, this task requires adding packet losses and verify the performance using Reno and then BBR.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

Table 1. Credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

Step 1. Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type password.

Step 2. Load the topology located at /home/admin/NTP_Exercises/Exercises3/.
Exercise 3: Tuning TCP and Switch’s Buffer Size

Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.

Step 5. Navigate into the following directory ~/NTP_Exercises/Exercise3/ by issuing the command below:

```
cd ~/NTP_Exercises/Exercise3/
```
Exercise 3: Tuning TCP and Switch’s Buffer Size

**Step 6.** Run the command below. If a password is required, type `password`.

```
sudo ./set_env.sh
```

3 Deliverables

Follow the steps below to complete the exercise.

- **a)** Set the sender’s congestion control to Reno.

- **b)** Run an iPerf3 test for 120 seconds and explain the results. What is the throughput?

- **c)** Inspect and modify the TCP and switch’s buffer sizes to fully utilize the link and maximize the throughput and repeat **b)**.

- **d)** Add a 0.01% packet loss rate to the interface `s1-eth2` and repeat **b)**.

- **e)** Set the sender’s congestion control to BBR and repeat **b)**.
NETWORK TOOLS AND PROTOCOLS

Exercise 4: Running Tests with Competing TCP Flows and Different Congestion Control Algorithms

Document Version: 08-25-2021

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

1 Exercise description ......................................................................................................................................... 3
  1.1 Credentials ................................................................................................................................................ 3
2 Setting the environment ..................................................................................................................................... 3
3 Deliverables...................................................................................................................................................... 5
1 Exercise description

In this exercise, you will run tests that involve competing TCP flows. The exercise requires tuning the TCP and switch’s buffer size. The tests will be performed first using CUBIC, then BBR.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

Step 1. Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type [password].
Step 2. Load the topology located at /home/admin/NTP_Exercises/Exercises4/.

Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.
**Step 5.** Navigate into the following directory ~/NTP_Exercises/Exercise4/ by issuing the command below.

```bash
cd ~/NTP_Exercises/Exercise4/
```

**Step 6.** Run the command below. If a password is required, type `password`.

```bash
sudo ./set_env.sh
```

The script sets the bottleneck link to 1 Gbps and the latency to 40ms. Now, you can start solving the problem.

### 3 Deliverables

Follow the steps below to complete the exercise.

- **a)** Set the senders’ congestion control to CUBIC.

- **b)** Inspect and modify the TCP and switch’s buffer sizes to fully utilize the link and maximize the throughput.

- **c)** Run two iPerf3 tests simultaneously for 120 seconds and explain the results. What is the throughput? Which one is the flow that utilizes more bandwidth?

- **d)** Change the senders’ congestion control to BBR and repeat c).

- **e)** Change the congestion control of a sender to CUBIC and repeat c).
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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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.

![Data transfer model with parallel streams.](image)

**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:

$$\text{Average throughput} \approx \frac{\text{MSS}}{\text{RTT} \cdot \sqrt{\text{L}}} \text{ bytes per second},$$

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:

$$\text{Aggregate average throughput} \approx \sum_{i=1}^{K} \frac{\text{MSS}}{\text{RTT} \cdot \sqrt{\text{L}}} = K \cdot \frac{\text{MSS}}{\text{RTT} \cdot \sqrt{\text{L}}} \text{ bytes per second}.$$ 

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.

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.
Lab 9: TCP Parallel Streams

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.

![Lab topology diagram](image)

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`.

![MiniEdit shortcut](image)

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.

![MiniEdit’s Open dialog](image)

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.
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.](image)

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
```
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 500000 limit 15000000
```

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.

```
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
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 28.080/32.784/40.883/9.006 ms
```

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
```

![Image of iPerf3 server on h2](image1.png)

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
```

![Image of iPerf3 client on h1](image2.png)

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
```

![Figure 16. Host h2 running iPerf3 as server.](image)

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
```

![Figure 17. iPerf3 throughput test with parallel streams.](image)
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.

![Figure 18. iPerf3 throughput test with parallel streams summary output.](image)

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.
Lab 9: TCP Parallel Streams

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

![Figure 19. Deleting previous rules on switch S1's s1-eth2 interface.](image)

**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%
```

![Figure 20. Adding 1% packet loss to switch S1's s1-eth2 interface.](image)

**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
```

![Figure 21. Limiting the bandwidth to 10 Gbps on switch S1's s1-eth2 interface.](image)

**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
```
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
```

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
```

Figure 24. Starting iPerf3 server on host h2.
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
```

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.
Note the measured throughput now is approximately 9.6 Gbps, which is close to the value assigned in our \texttt{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

NETWORK TOOLS AND PROTOCOLS

Exercise 5: Enhancing the Aggregate TCP Throughput with Parallel Streams

Document Version: 08-25-2021

Award 1829698
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”
Exercise 5: Enhancing the Aggregate TCP Throughput with Parallel Streams

Contents

1  Exercise description .................................................................................................................. 3
   1.1  Credentials ......................................................................................................................... 3
2  Setting the environment ............................................................................................................. 3
3  Deliverables ................................................................................................................................ 5
1 Exercise description

In this exercise, you will run tests using parallel streams. The task also requires verifying the aggregate throughput in the presence of packet losses.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

Step 1. Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required, type password.

Step 2. Load the topology located at /home/admin/NTP_Exercises/Exercises5/.
Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.

Step 5. Navigate into the following directory `~/NTP_Exercises/Exercise5/` by issuing the following command:

```
cd ~/NTP_Exercises/Exercise5/
```
Exercise 5: Enhancing the Aggregate TCP Throughput with Parallel Streams

**Step 6.** Run the command below. If a password is required, type `password`.

```
sudo ./set_env.sh
```

The script sets the bottleneck link to 100 Mbps and the latency to 20ms. Now, you can start solving the problem.

3 Deliverables

Follow the steps below to complete the exercise.

- **a)** Set a 0.2% packet loss rate to the interface `s2-eth2`.
- **b)** Run an iPerf3 test for 120 seconds and explain the results. What is the throughput?
- **c)** Run an iPerf3 test for 120 seconds using 5 parallel streams and explain the results. What is the aggregate throughput?
- **d)** Repeat **c)** with 20 parallel streams.
Problem 2: Enhancing TCP Throughput

Award 1829698
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”
Problem 2: Enhancing TCP Throughput

Contents

1 Problem description.................................................................................................................. 3
  1.1 Credentials ......................................................................................................................... 3
2 Setting the environment............................................................................................................ 3
1 Problem description

The interface s2-eth2 is experiencing packet losses. Provide a solution to ensure that the throughput from host h1 to host h2 achieves approximately 900 Mbps. Do not remove the emulated packet losses in the interface s1-eth2.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

**Step 1.** Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type \texttt{password}.

**Step 2.** Load the topology located at \texttt{/home/admin/NTP\_Problems/Problem2/}. 
Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.

Step 5. Navigate into the following directory ~/NTP_Problems/Problem2/ by issuing the following command:

```
cd ~/NTP_Problems/Problem2/
```
Problem 2: Enhancing TCP Throughput

**Step 6.** Run the command below. If a password is required, type `password`.

```
sudo ./set_env.sh
```

The script sets the bottleneck link to 1 Gbps, the latency to 20ms and the loss rate to 0.1%. Now, you can start solving the problem.
# 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
Lab 10: Measuring TCP Fairness

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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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
Lab 10: Measuring TCP Fairness

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](image1.png)

**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 the $x$-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 multiplicative 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 \( p_4 \), 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 \( \text{Opt} \ (R/2, R/2) \). 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.](image)

### 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:

\[
I = \frac{\left(\sum_{i=1}^{3} T_i\right)^2}{3 \sum_{i=1}^{3} T_i^2} = \frac{(5 \cdot 10^9 + 3 \cdot 10^9 + 1 \cdot 10^9)^2}{3 \cdot (5 \cdot 10^9)^2 + (3 \cdot 10^9)^2 + (1 \cdot 10^9)^2} = 0.77
\]

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.
Lab 10: Measuring TCP Fairness

**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`.

![MiniEdit shortcut](image)

**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*.

![MiniEdit’s Open dialog](image)

**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.

![Running the emulation](image)

**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.](image)

**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.

![Figure 9. Connectivity test using `ping` command.](image)

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.
Lab 10: Measuring TCP Fairness

**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.](image)

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
```

![Figure 11. Adding delay of 20ms to switch S1’s s1-eth1 interface.](image)

**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
```

![Figure 12. Limiting the bandwidth to 10 Gbps on switch S1’s s1-eth1 interface.](image)
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.

![Figure 13. Output of `ping 10.0.0.2` command.](image)

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
```

![Figure 14. Starting iPerf3 server on host h2.](image)

**Step 4.** Launch iPerf3 in client mode on host h1’s terminal.

```
iperf3 -c 10.0.0.2
```
Lab 10: Measuring TCP Fairness

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:

\[
BW = 10,000,000,000 \text{ bits/second}
\]

\[
RTT = 0.02 \text{ seconds}
\]

\[
BDP = 10,000,000,000 \times 0.02 = 200,000,000 \text{ bits} \\
= 25,000,000 \text{ bytes} \approx 25 \text{ Mbytes}
\]

The send and receive buffer sizes should be set to \(2 \cdot BDP\). We will use the 25 Mbytes value for the BDP instead of 25,000,000 bytes.

\[
1 \text{ Mbyte} = 1024^2 \text{ bytes}
\]

\[
BDP = 25 \text{ Mbytes} = 25 \times 1024^2 \text{ bytes} = 26,214,400 \text{ bytes}
\]

\[
2 \cdot BDP = 2 \cdot 26,214,400 \text{ bytes} = 52,428,800 \text{ bytes}
\]

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'
```
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'
```

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'
```

Next, the same commands must be configured on host h2.

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'
```

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
```
Lab 10: Measuring TCP Fairness

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
```

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
```
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
```

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
```

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.

![Opening Mininet’s terminal](image1.png)

**Step 3.** Issue the following command on Mininet’s terminal as shown in the figure below.

```
sudo source concurrent_same_algo
```
Lab 10: Measuring TCP Fairness

Figure 27. Running the tests simultaneously for 120 seconds. Both host h1 and host h3 are using Reno.

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.
Lab 10: Measuring TCP Fairness

The above figure shows a fairness index of 0.99595. 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.

Step 2. Issue the following command on Mininet’s terminal as shown in the figure below.

```
source concurrent_diff_algo
```
Lab 10: Measuring TCP Fairness

Figure 31. Running the tests simultaneously for 20 seconds. Host h1 is using Reno while host h3 is using BBR.

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.
The above figure shows a fairness index of 0.86036 (~ 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

NETWORK TOOLS AND PROTOCOLS

Exercise 6: RTT Unfairness

Document Version: 08-25-2021

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

1 Exercise description .............................................................................................................. 3
  1.1 Credentials .................................................................................................................. 3
2 Setting the environment ...................................................................................................... 3
3 Deliverables .......................................................................................................................... 5
1 Exercise description

In this exercise, run tests that involve competing TCP flows. The exercise requires tuning the TCP and switch’s buffer size. Additionally, you will create a scenario to observe RTT unfairness. Finally, you are going to run a test with two competing flows that have different congestion control.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

Table 1. Credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

Step 1. Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type password.
Exercise 6: RTT Unfairness

Step 2. Load the topology located at /home/admin/NTP_Exercises/Exercises6/.

Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.
Step 5. Navigate into the following directory `~/NTP_Exercises/Exercise6/` by issuing the command below.

```
cd ~/NTP_Exercises/Exercise6/
```

Step 6. Run the command below. If a password is required, type `password`.

```
sudo ./set_env.sh
```

The script sets the bottleneck link to 1 Gbps and the latency to 40ms.

3 Deliverables

Follow the steps below to complete the exercise.

- **a)** Set the senders’ congestion control to Reno.
- **b)** Add a 100ms delay to the interface `s2-eth2`.
- **c)** Inspect and modify the TCP and switch’s buffer sizes to fully utilize the link and maximize the throughput.
- **d)** Run two iPerf3 tests simultaneously (one TCP connection from host h1 to host h3, and another from host h2 to host h4) for 120 seconds and explain the results.
- **e)** Set the senders’ congestion control to BBR and repeat **d)**.
NETWORK TOOLS AND PROTOCOLS

Problem 3: Minimizing the Unfairness

Document Version: 08-22-2021

Award 1829698
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”
Problem 3: Minimizing the Unfairness

Contents

1 Problem description ............................................................................................................. 3
  1.1 Credentials .................................................................................................................... 3
2 Setting the environment ...................................................................................................... 3
1 Problem description

When host h1 and host h2 are transferring data simultaneously, the fairness index is low. Find the issue and provide a solution that guarantees a fairness index greater than 0.9. Additionally, ensure that the aggregate throughput is greater than 90 Mbps.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

**Step 1.** Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type [password].
Step 2. Load the topology located at /home/admin/NTP_Problems/Problem3/.

Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.
Step 5. Navigate into the following directory ~/NTP_Problems/Problem3/ by issuing the following command:

```
cd ~/NTP_Problems/Problem3/
```

Step 6. Run the command below. If a password is required, type password.

```
sudo ./set_env.sh
```

The script sets the bottleneck link to 100 Mbps and the latency to 40ms. Now, you can start solving the problem.
Lab 11: Router’s Buffer Size

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 ............................................. 8
   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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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.

Router queues/buffers absorb traffic fluctuations. Even in the absence of congestion, fluctuations are present, resulting mostly from coincident traffic bursts. 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. 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. 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.
Router’s buffer size = \( C \cdot \text{RTT [bits]} \) (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:

\[
\text{Router’s buffer size} = \frac{C \cdot \text{RTT}}{\sqrt{N}} \text{ [bits]} \quad \text{(large number of flows N)}
\]

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.

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.
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.

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.

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.

![Opening a terminal on host h1.](image)

**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.

![Connectivity test using ping command.](image)

**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.
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.

$$BW = 10,000,000,000 \text{ bits/second}$$

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

$$\text{BDP} = 10,000,000,000 \cdot 0.02 = 200,000,000 \text{ bits}$$
$$= 25,000,000 \text{ bytes} \approx 25 \text{ Mbytes}$$

The send and receive buffer sizes should be set to $5 \cdot \text{BDP}$. We will use the 25 Mbytes value for the BDP instead of 25,000,000 bytes.

1 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).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
```
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).

```bash
sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
```

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).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 131072000'
```

**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).

```bash
sysctl -w net.ipv4.tcp_wmem='10240 87380 131072000'
```
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'
```

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'
```

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'
```

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'
```

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.](image)

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
```

![Figure 18. Adding delay of 10ms to switch S1’s s1-eth1 interface.](image)

**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
```

![Figure 19. Adding delay of 10ms to switch S2’s s2-eth1 interface.](image)
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.

![Figure 20. Output of ping 10.0.0.2 command.](image)

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`.

![Figure 21. Output of ping 10.0.0.4 command.](image)

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 \cdot 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.

- \textbf{rate}: 10gbit
- \textbf{burst}: 5,000,000
- \textbf{limit}: 160,000

\begin{verbatim}
sudo tc qdisc add dev s1-eth1 parent 1: handle 2: tbf rate 10gbit burst 5000000 limit 160000
\end{verbatim}

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
```

Figure 23. Starting iPerf3 server on host h2.
Step 2. Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

![Figure 24. Starting iPerf3 server on host h4.](image)

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
```

![Figure 25. Typing iPerf3 client command on host h1.](image)

Step 4. Type the following iPerf3 command in host h3’s terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

![Figure 26. Typing iPerf3 client command on host h3.](image)

Step 5. Press Enter to execute the commands, first in host h1 terminal then, in host h3 terminal.
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
```

![Figure 28. Changing TCP congestion control algorithm to `reno` in host h1.](image)

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
```
Step 3. Launch iPerf3 in server mode on host h4’s terminal.

`iperf3 -s`

Step 4. Type the following iPerf3 command in host h1’s terminal without executing it.

`iperf3 -c 10.0.0.2 -t 90`

Step 5. Type the following iPerf3 command in host h3’s terminal without executing it.

`iperf3 -c 10.0.0.2 -t 90`

Step 6. Press Enter to execute the commands, first in host h1 terminal then, in host h3 terminal.
Figure 3.3. 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
```

Figure 3.4. 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
```
Step 3. Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

Figure 35. Starting iPerf3 server on host h2.

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
```

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
```

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.
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`
Lab 11: Router’s Buffer Size

- **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
```

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
```

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
```

Figure 42. Starting iPerf3 server on host h2.

**Step 3.** Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```
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
```

**Step 5.** Type the following iPerf3 command in host h3’s terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

**Step 6.** Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.
Lab 11: Router’s Buffer Size

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 avoidance 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
```

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
```

Figure 48. Starting iPerf3 server on host h2.

**Step 3.** Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

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
```

![Figure 50. Typing iPerf3 client command on host h1.](image)

**Step 5.** Type the following iPerf3 command in host h3’s terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

![Figure 51. Typing iPerf3 client command on host h3.](image)

**Step 6.** Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.

![Figure 52. Running iPerf3 client on host h1.](image)

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
```

![Figure 53. Changing TCP congestion control algorithm to BBR in host h1.](image)

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
```

![Figure 54. Starting iPerf3 server on host h2.](image)

**Step 3.** Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

![Figure 55. Starting iPerf3 server on host h4.](image)

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.
Step 5. Type the following iPerf3 command in host h3’s terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

Figure 56. Typing iPerf3 client command on host h1.

Step 6. Press Enter to execute the commands, first in host h1 terminal then, in host h3 terminal.

Figure 57. Typing iPerf3 client command on host h3.

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%
```

![Figure 59. Adding delay of 0.01% to switch S1’s s1-eth1 interface.](image)

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
```

![Figure 60. Changing TCP congestion control algorithm to cubic in host h1.](image)

**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
```

![Figure 61. Starting iPerf3 server on host h2.](image)
**Step 3.** Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

![Figure 62. Starting iPerf3 server on host h4.](image)

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
```

![Figure 63. Typing iPerf3 client command on host h1.](image)

**Step 5.** Type the following iPerf3 command in host h3’s terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

![Figure 64. Typing iPerf3 client command on host h3.](image)

**Step 6.** Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.
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
```

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
```
Step 3. Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

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
```

Step 5. Type the following iPerf3 command in host h3’s terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

Step 6. Press Enter to execute the commands, first in host h1 terminal then, in host h3 terminal.
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
```

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.
Step 3. Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

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
```

Step 5. Type the following iPerf3 command in host h3’s terminal without executing it.

```
iperf3 -c 10.0.0.2 -t 90
```

Step 6. Press *Enter* to execute the commands, first in host h1 terminal then, in host h3 terminal.
Figure 7.7. 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**


Lab 12: TCP Pacing

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.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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.](image)

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.

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. 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 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, 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.

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 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. 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). 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, 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.

![Figure 3. (a) FQ-pacing. (b) Deficit Round-Robin (DRR) algorithm.](image)
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.

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.

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.
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.

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.
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.

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.
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
```

![Figure 11. Adding delay of 20ms to switch S1’s *s1-eth1* interface.](image)

**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
```

![Figure 12. Limiting the bandwidth to 10 Gbps on switch S1’s *s1-eth1* interface.](image)

### 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.

![Figure 13. Output of `ping 10.0.0.2` command.](image)

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:

\[
BW = 10,000,000,000 \text{ bits/second} \\
RTT = 0.02 \text{ seconds} \\
BDP = 10,000,000,000 \cdot 0.02 = 200,000,000 \text{ bits} \\
= 25,000,000 \text{ bytes} \approx 25 \text{ Mbytes}
\]

The send and receive buffer sizes should be set to \(2 \cdot \text{BDP}\). We will use the 25 Mbytes value for the BDP instead of 25,000,000 bytes.

1 Mbyte = \(1024^2\) bytes

BDP = 25 Mbytes = \(25 \cdot 1024^2\) bytes = 26,214,400 bytes

TCP buffer size = \(2 \cdot \text{BDP} = 2 \cdot 26,214,400\) 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'
```
**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).

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

**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).

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

**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).

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

**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
```
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:

```bash
iperf3 -c 10.0.0.2
```

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:

```bash
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.
Lab 12: TCP Pacing

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.

[qdisc 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: 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 qdiscs.
- **maxrate 5gbit**: Maximum sending rate of a flow (default is unlimited). Enables pacing on a maximum rate of 5 Gbps.

![Figure 20. Enabling fair queuing pacing with a maximum rate of 5 Gbps to the interface h1-eth0 on host h1.](image)

**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
```

![Figure 21. Host h2 running iPerf3 as server.](image)
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
```

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.
Lab 12: TCP Pacing

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

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:

```bash
iperf3 -s
```

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.

```bash
iperf3 -c 10.0.0.2 -O 5 --fq-rate 5gbit
```

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.

Step 3. Issue the following command on Mininet’s terminal as shown in the figure below.

```
source concurrent_no_pacing
```
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.
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.
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.
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
```
Figure 34. Host h2 running iPerf3 as server.

**Step 4.** Create and enter to a new directory `parallel_streams`:

```bash
mkdir parallel_streams && cd 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.

```bash
iperf3 -c 10.0.0.2 -t 30 -P 5 -J > parallel_streams.json
```

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:

```bash
plot_iperf.sh parallel_streams.json
```

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.

```bash
cd 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
```

![Opening the throughput.pdf file using xdg-open](image)

Figure 39. Opening the *throughput.pdf* file using `xdg-open`.

![Throughput over time](image)

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 ../..
```

![Exiting the reno/results directory](image)

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
```
Lab 12: TCP Pacing

Figure 42. Host h2 running iPerf3 as server.

**Step 2.** Create and enter to a new directory `parallel_streams_pacing`:

```bash
mkdir parallel_streams_pacing && cd 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 `-c` 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.

```bash
iperf3 -c 10.0.0.2 -t 30 -P 5 -J --fq-rate 1.9gbit > parallel_streams_pace.json
```

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:

```bash
plot_iperf.sh parallel_streams_pace.json
```

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/
```

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
```

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


NETWORK TOOLS AND PROTOCOLS

Exercise 7: Setting the Pacing Rate

Document Version: 08-22-2021

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

1  Exercise description ........................................................................................................................................ 3
   1.1  Credentials ........................................................................................................................................... 3
2  Setting the environment .................................................................................................................................. 3
3  Deliverables.................................................................................................................................................. 5
1 Exercise description

In this exercise, you will build a topology and run tests that involve competing TCP flows. The exercise requires setting the senders’ pacing rate to ensure a fairness index of approximately 1.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

Step 1. Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type password.
Step 2. Load the topology located at `/home/admin/NTP_Exercises/Exercises7/`.

Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.
**Step 5.** Navigate into the following directory `~/NTP_Exercises/Exercise7/` by issuing the command below.

```bash
cd ~/NTP_Exercises/Exercise7/
```

**Step 6.** Run the command below. If a password is required, type `password`.

```
sudo ./set_env.sh
```

The script sets the bottleneck link to 100 Mbps and the latency to 40ms. Now, you can start with the exercise.

### 3 Deliverables

Follow the steps below to complete the exercise.

a) Inspect and modify the TCP and switch’s buffer sizes to fully utilize the link and maximize the throughput.

b) Run four iPerf3 tests simultaneously (one TCP connection from host h1 to host h5, another from host h2 to host h6, another from host h3 to host h7 and from host h4 to host h8) for 240 seconds and explain the results.

c) Set the senders’ pacing rate to ensure a fairness index greater than 0.92 and repeat b).

d) Run three iPerf3 tests simultaneously (one TCP connection from host h1 to host h5, another from host h2 to host h6, and from host h3 to host h7) for 240 seconds and explain the results. Is the bottleneck link fully utilized?
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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

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. 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. 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.

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.

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.
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.

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.
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.

![Connectivity test using ping command.](image)

### 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.
Lab 13: Impact of MSS on Throughput

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.

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

![Figure 10. Adding 1% packet loss to switch S1’s `s1-eth2` interface.](image)

**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

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

![Figure 11. Limiting the bandwidth to 10 Gbps on switch S1’s `s1-eth2` interface.](image)
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
```

![Figure 12. Host h2 running iPerf3 as server.](image)

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
```

![Figure 13. iPerf3 throughput test.](image)

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
```

![Figure 14. Identifying interface’s MTU.](image)

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
```
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
```

![Figure 17. Changing host h1’s interface MTU.](image)

**Step 2.** To change the MTU to 9000 bytes, on host h2’s terminal, type in the following command:

```
ifconfig h2-eth0 mtu 9000
```

![Figure 18. Changing host h2’s interface MTU.](image)

**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
```

![Figure 19. Changing MTU values on the switches.](image)
**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
```

![Figure 20. Host h2 running iPerf as server.](image)

**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
```

![Figure 21. iPerf3 throughput test with a 9000 MSS value.](image)

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

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”
# Lab 14: Router’s Bufferbloat

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>Objectives</td>
<td>3</td>
</tr>
<tr>
<td>Lab settings</td>
<td>3</td>
</tr>
<tr>
<td>Lab roadmap</td>
<td>3</td>
</tr>
<tr>
<td>1 Introduction to bufferbloat</td>
<td>3</td>
</tr>
<tr>
<td>1.1 Packet delays</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Bufferbloat</td>
<td>4</td>
</tr>
<tr>
<td>2 Lab topology</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Starting host h1, host h2, and host h3</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Emulating high-latency WAN</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Testing connection</td>
<td>9</td>
</tr>
<tr>
<td>3 Testing throughput on a network with a small buffer-size switch</td>
<td>10</td>
</tr>
<tr>
<td>3.1 Setting switch S1’s buffer size to 100-MTU</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Bandwidth-delay product (BDP) and hosts’ buffer size</td>
<td>11</td>
</tr>
<tr>
<td>3.3 Throughput test</td>
<td>11</td>
</tr>
<tr>
<td>4 Testing throughput on a network with a 1-BDP buffer-size switch</td>
<td>13</td>
</tr>
<tr>
<td>4.1 Setting switch S1’s buffer size to 1-BDP</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Throughput and latency tests</td>
<td>14</td>
</tr>
<tr>
<td>5 Testing throughput on a network with a large buffer-size switch</td>
<td>19</td>
</tr>
<tr>
<td>5.1 Setting switch S1’s buffer size to 10-BDP</td>
<td>19</td>
</tr>
<tr>
<td>5.2 Throughput and latency tests</td>
<td>20</td>
</tr>
<tr>
<td>References</td>
<td>25</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction to bufferbloat.
2. Section 2: Lab topology.
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).

![Figure 1. Delay components: processing, queueing, transmission, and propagation delays.](image)

- **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 $T = \frac{12,000}{10 \times 10^9} = 0.0012$ milliseconds or 1.2 microseconds.

- **Queueing delay**: The time a packet waits for transmission onto the link. The length of the queueing delay of a packet depends on the number of earlier-arriving packets that are queued and waiting for transmission onto the link. Queueing 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\textsuperscript{2,3}:

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

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\textsuperscript{4}. Consider Figure 2, where $RT_{\text{prop}}$ 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 $RT_{\text{prop}}$. $RT_{\text{prop}}$ 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.
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.
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.

![MiniEdit shortcut](image)

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.

![MiniEdit’s Open dialog](image)

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.

![Running the emulation](image)

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.](image)

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

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

![Figure 8. Connectivity test using ping command.](image)

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.
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
```

---

**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.
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`.

![Figure 12. Output of `ping 10.0.0.3` command.](image)

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
```
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.

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

\[
\text{RTT} = 0.02 \text{ seconds}
\]

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

\[
= 2,500,000 \text{ bytes} \approx 2.5 \text{ Mbytes}
\]

1 Mbyte = \(1024^2\) 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 \(\cdot\) 10-BDP) as Linux only allocates half of the assigned value.

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

Figure 14. Receive window change in \texttt{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'
```

![Figure 15. Send window change in `sysctl`.](image)

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

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

![Figure 16. Receive window change in `sysctl`.](image)

**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'
```

![Figure 17. Send window change in `sysctl`.](image)

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
```

![Figure 18. Starting iPerf3 server on host h3.](image)

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

```
iperf3 -c 10.0.0.3
```

![Figure 19. Running iPerf3 client on host h1.](image)

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
```

Figure 20. Limiting rate to 1 Gbps and setting the buffer size to 1\cdot 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
```

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
```

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

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

![Image of mkdir and cd commands](image1.png)

Figure 24. Creating and navigating into directory 1_BDP.

Step 4. Type the following iPerf3 command in host h1’s terminal without executing it. The `-J` option is used to display the output in JSON format. The redirection operator `>` is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

![Image of iPerf3 command](image2.png)

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
```

![Image of ping command](image3.png)

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.

![Queue occupancy on switch S1's s1-eth2 interface.](image)

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 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
```

![Generate plotting files and entering the results directory.](image)

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
```

![Opening the throughput.pdf file.](image)

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

![Opening RTT.pdf](image)

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

![Figure 33. Opening the cwnd.pdf file.](image)

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.
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\cdot 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\cdot 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
Lab 14: Router’s Bufferbloat

```bash
sudo tc qdisc change dev s1-eth2 parent 1: handle 2: tbf rate 1gbit burst 500000 limit 26214400
```

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.

```bash
iperf3 -s
```

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.

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

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

```bash
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.

```bash
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.

![Image of Gnuplot graph]

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 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
```

![Command output]

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
```

![Command output]

Figure 45. Opening the throughput.pdf file.
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 $10 \cdot \text{BDP}$.

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

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

Figure 46. Measured throughput.

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
```

![Figure 49. Opening the cwnd.pdf file.](image)

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.
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**

NETWORK TOOLS AND PROTOCOLS

Exercise 8: Router’s Bufferbloat

Document Version: 08-25-2021

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

1 Exercise description .................................................................................................................................................. 3
  1.1 Credentials ................................................................................................................................................... 3
2 Setting the environment ........................................................................................................................................... 3
3 Deliverables........................................................................................................................................................... 5
1 Exercise description

In this exercise, you will observe the router’s bufferbloat effects using different congestion controls.

1.1 Credentials

The information in Table 1 provides the credentials to access the Client’s virtual machine.

Table 1. Credentials to access the Client’s virtual machine.

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

2 Setting the environment

Follow the steps below to set the problem’s environment.

Step 1. Open MiniEdit by double-clicking the shortcut on the Desktop. If a password is required type password.
Exercise 8: Router’s Bufferbloat

Step 2. Load the topology located at `/home/admin/NTP_Exercises/Exercises8/`.

Step 3. Run the emulation by clicking on the button located on the lower left-hand side.

Step 4. Open the Linux terminal by clicking on the icon in the taskbar.
Step 5. Navigate into the following directory `~/NTP_Exercises/Exercise8/` by issuing the following command:

```
cd ~/NTP_Exercises/Exercise8/
```

Step 6. Run the command below. If a password is required, type `password`.

```
sudo ./set_env.sh
```

The script sets the bottleneck link to 1 Gbps and the latency to 50ms. Now, you can start solving the problem.

### 3 Deliverables

Follow the steps below to complete the exercise.

a) Set switch’s s1 buffer size to BDP.

b) Start a connectivity test from one of the senders using the ping command. Do not stop the test.

c) From another sender, run an iPerf3 test for 120 seconds and explain the results. Report the throughput and the RTT.

d) Change switch’s s1 buffer size to 10-BDP and repeat c).

e) Set the sender’s congestion control to BBR and repeat c).

f) Set the sender’s congestion control to Reno and repeat c).
Lab 15: Analyzing the Impact of Hardware Offloading on TCP Performance

Document Version: 04-11-2020

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 hardware offloading ...................................................................................... 3
  1.2 The ethtool command .............................................................................................................. 5
2 Lab topology ................................................................................................................................. 5
  2.1 Starting host h1 and host h2 ..................................................................................................... 7
3 Emulating a high-latency WAN ................................................................................................. 8
  3.1 Adding delay to switch s1 egress interface .............................................................................. 8
  3.2 Testing connection .................................................................................................................. 9
  3.3 Limiting the rate on switch S2 egress interface ..................................................................... 9
  3.4 Modifying end-hosts’ buffer size ......................................................................................... 10
  3.5 Performing a throughput test ............................................................................................... 12
4 Disabling hardware offloading ................................................................................................. 13
  4.1 Disabling TSO in the sender ................................................................................................. 13
  4.2 Disabling GRO in the receiver ............................................................................................. 13
  4.3 Performing a throughput test ............................................................................................... 13
References ..................................................................................................................................... 14
Overview

The lab aims to explain how to tune the parameters of a Network Interface Controller (NIC) in order to reduce CPU overhead of TCP flows. These parameters rely on the NIC to segment the data and then add the TCP, IP and data link layer protocol headers to each segment. During this lab the user will conduct throughput tests under different network conditions in order to evaluate the performance.

Objectives

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

1. Understand network drivers and hardware setting
2. Modify NIC parameters using `ethtool`.
3. Evaluate the impact of disabling TCP Segmentation Offload (TSO) in the sender and General Segmentation Offload (GRO) in the receiver.
4. Conduct evaluation tests under different network conditions.
5. Analyze the results after disabling hardware offloading parameters.

Lab settings

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

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Emulating a high-latency WAN.
4. Section 4: Disabling hardware optimization.

1  Introduction

1.1 Introduction to hardware offloading
Consider Figure 1. It illustrates two scenarios. Figure 1a presents a situation where hardware offloading is disabled in the sender host. Therefore, the application data is segmented in the transport layer, in this example in five segments then, these segments are individually encapsulated by the corresponding layers. Finally, the packets are sent. Similarly, when hardware offloading is disabled in receiver, the link layer (i.e. Ethernet), which is implemented in the Network Interface Controller, sends to the upper layers all the packets as it is received, that is, without grouping the frames in a larger set of data.

On the other hand, Figure 1b shows a scenario where hardware offloading is enabled, in this case, it is observed that the application data consists in a single data set all the way down up to the link layer. When the IP datagram reaches the link layer, the NIC performs data segmentation before they are sent to the destination.

In summary, when hardware offloading is disabled, the Operating System (OS) uses the CPU to segment TCP packets however, when hardware offloading is enabled, it allows the NIC to use its own processor to perform the segmentation. This saves on the CPU and importantly cuts down on the bus communications to/from the NIC. Nevertheless, offloading does not change what is sent over the network. In other words, offloading to the NIC can produce performance gains within the sender host, but not across the network\(^3\).
1.2 The `ethtool` command

The `ethtool` is a utility for configuration of Network Interface Controllers (NICs). This utility allows querying and changing settings such as speed, port, auto-negotiation, PCI locations and checksum offload on many network devices, especially ethernet devices. The syntax of the command is as follows:

```
ethtool <options> <interface> <parameters> on|off
```

- **ethtool**: query or control network driver and hardware settings.
- **options**: used to specify read or write operation.
- **interface**: specifies the network interface where `ethtool` should operate.
- **parameters**: specifies the parameter that will be enabled or disabled.

In this lab, we will use the `ethtool` to modify the sender and receiver NIC configuration.

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.
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.

Step 2. On MiniEdit’s menu bar, click on File then Open to load the lab’s topology. Locate the Lab 15.mn topology file and click on Open.
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.

![Run button](image)

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

![Terminal](image)

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.
Lab 15: Analyzing the Impact of Hardware Offloading on TCP Performance

Figure 7. Connectivity test using `ping` command.

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

3  Emulating a high-latency WAN

In this section, you are going to tune the network devices in order to emulate a Wide Area Network (WAN). Firstly, you will add 20ms latency to switch S1’s `s1-eth1` egress interface. Secondly you will set the bottleneck bandwidth to 1Gbps in switch S2’s `s2-eth2` egress interface.

Then, you will set the hosts’ TCP buffers to 8-BDP therefore, the bottleneck is not in the end-hosts. Finally, you will conduct throughput tests between host h1 and h2.

3.1  Adding delay to switch s1 egress interface

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.

The Linux terminal is a program that opens a window and permits you to interact with a command-line interface (CLI). A CLI is a program that takes commands from the keyboard and sends them to the operating system to perform.
**Step 2.** In the terminal, type the command below. When prompted for a password, type `password` and hit `Enter`. This command introduces 20ms delay to switch S1’s `s1-eth2` interface.

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

![Figure 9. Adding delay of 20ms to switch S1’s s1-eth2 interface.](image)

### 3.2 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.

![Figure 10. Output of `ping 10.0.0.2` command.](image)

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 20.087, 25.387, 41.244, and 9.155 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

### 3.3 Limiting the rate on switch S2 egress interface

**Step 1.** Apply `tbf` rate limiting rule on switch S2’s `s2-eth2` interface. In the client’s terminal, type the command depicted below. When prompted for a password, type `password` and hit `Enter`.

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

```
sudo tc qdisc add dev s2-eth2 root tbf rate 1gbit burst 500000 limit 2621440
```

Figure 11. Limiting rate to 1 Gbps and setting the buffer size to BDP on switch S2’s interface.

### 3.4 Modifying end-hosts’ buffer size

To perform the following calculation, the bottleneck bandwidth is considered as 1 Gbps, and the Round-trip time delay 20ms.

In order to have enough TCP buffer size, we will set the TCP sending and receiving buffer to $8 \cdot \text{BDP}$ in all hosts.

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

\[
\text{RTT} = 0.02 \text{ seconds}
\]

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

\[
= 2,500,000 \text{ bytes} \approx 2.5 \text{ Mbytes}
\]

The send and receive TCP buffer sizes should be set to $8 \cdot \text{BDP}$ to ensure the bottleneck is not in the end-hosts. For simplicity, we will use 2.5 Mbytes as the value for the BDP instead of 2,500,000 bytes.

\[
1 \text{ Mbyte} = 1024^2 \text{ bytes}
\]

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

\[
8 \cdot \text{BDP} = 8 \cdot 2,621,440 \text{ bytes} = 20,971,520 \text{ bytes}
\]

**Step 1.** At this point, 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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```
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. 20,971,520 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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

**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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```
3.5 Performing a throughput test

**Step 1.** The user can now verify the previous configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, in host h2’s terminal run the command shown below:

```
iperf3 -s
```

![Figure 16. Host h2 running iPerf3 as server.](image)

**Step 2.** Now to launch iPerf3 in client mode again by running the following command in host h1’s terminal.

```
iperf3 -c 10.0.0.2
```

![Figure 17. Performing a throughput test to verify the configuration.](image)

The figure above shows the iPerf3 test output report. The average achieved throughputs are 953 Mbps (sender) and 940 Mbps (receiver).

**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 Disabling hardware offloading

In this section, the user will disable the Generic Segmentation Offload (TSO) in the sender, and the Generic Receive Offload (GRO) in the receiver. Then, the user will conduct a throughput test in order to evaluate the impact on the performance.

4.1 Disabling TSO in the sender

Step 1. In host h1 terminal type the following command to disable TSO feature in the sender.

```
ethtool -K h1-eth0 tso off
```

Figure 18. Disabling TSO in the sender.

4.2 Disabling GRO in the receiver

Step 1. In host h2 terminal type the following command to disable GRO feature in the receiver.

```
ethtool -K h2-eth0 gro off
```

Figure 19. Disabling GRO in the receiver.

4.3 Performing a throughput test

Step 1. To launch iPerf3 in server mode, in host h2’s terminal run the command shown below:

```
iperf3 -s
```

Figure 20. Host h2 running iPerf3 as server.
Step 2. Now to launch iPerf3 in client mode again by running the following command in host h1’s terminal.

```
iperf3 -c 10.0.0.2
```

![iperf3 throughput test after disabling hardware offloading features.](image)

Figure 21. iPerf3 throughput test after disabling hardware offloading features.

The figure above shows the iPerf3 test output report. The average achieved throughputs are 467 Mbps (sender) and 456 Mbps (receiver). Results show a decrease in ~60% in the performance in comparison to the previous test.

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.

This concludes Lab 15. Stop the emulation and then exit out of MiniEdit.

References

NETWORK TOOLS AND PROTOCOLS

Lab 16: Random Early Detection (RED)

Document Version: 10-10-2019

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

Overview .................................................................................................................................................. 3
Objectives .............................................................................................................................................. 3
Lab settings ............................................................................................................................................ 3
Lab roadmap ........................................................................................................................................... 3
1 Introduction .......................................................................................................................................... 3
  1.1 Random Early Detection mechanism ......................................................................................... 4
2 Lab topology ....................................................................................................................................... 6
  2.1 Starting host h1, host h2, and host h3 ....................................................................................... 7
  2.1 Emulating high-latency WAN .................................................................................................... 8
  2.4 Testing connection ....................................................................................................................... 9
3 Testing throughput on a network using Drop Tail AQM algorithm ........................................... 10
  3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size ....................................................... 10
  3.2 Setting switch S2’s buffer size to $10 \cdot$ BDP ............................................................................ 12
  3.3 Throughput and latency tests ..................................................................................................... 13
4 Configuring RED on switch S2 ......................................................................................................... 18
  4.1 Setting RED parameter on switch S2’s egress interface ............................................................ 19
  4.2 Throughput and latency tests ..................................................................................................... 19
  4.3 Changing the bandwidth to 100Mbps ....................................................................................... 24
  4.4 Throughput and latency tests ..................................................................................................... 25
References ............................................................................................................................................. 30
Overview

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

Objectives

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

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

Lab settings

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

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
4. Section 4: Configuring RED on switch S2.

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

Random Early Detection (RED) was proposed by Floyd and Van Jacobson\textsuperscript{1} to address network congestion in a responsive rather than reactive manner. The main goal of RED is to provide congestion avoidance by controlling the average queue size. Other goals are the avoidance of global synchronization and introduce fairness to reduce the bias against bursty traffic. TCP global synchronization happens to a TCP flow during periods of congestion when each sender reduces and then increase their transmission rate at the same time due to packet loss.

1.1 Random Early Detection mechanism

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

```
```

- **tc**: Linux traffic control tool.
- **qdisc**: A queue discipline (qdisc) is a set of rules that determine the order in which packets arriving from the IP protocol output are served. The queue discipline is applied to a packet queue to decide when to send each packet.
- **[add | del | replace | change | show]**: This is the operation on qdisc. For example, to add the token bucket algorithm on a specific interface, the operation will be `add`. To change or remove it, the operation will be `change` or `del`, respectively.
- **dev [dev_id]**: This parameter indicates the interface is to be subject to emulation.
- **red**: This parameter specifies the Random Early Detection algorithm.
- **limit [BYTES]**: Hard limit on the real (not average) queue size in bytes. Further packets are dropped. Should be set higher than `max+burst`.
- **max [BYTES]**: This parameter specifies the maximum average queue size. After this value, the dropping probability is 100%. It is recommended to set this value to `limit/4`.
- **min [BYTES]**: This parameter specifies the minimum average queue size. Below this value, no packet is dropped. Above this threshold, the dropping probability is established by `probability` or it increases linearly if the parameter `adaptative` is set.
- **avpkt**: Used with burst to determine the time constant for average queue size calculations. It is suggested 1000 as good value.
• **burst [BYTES]**: Used for determining how fast the average queue size is influenced by the real queue size. Larger values make the calculation slower, allowing longer bursts of traffic before the marking or dropping phase starts. Empirical evaluations suggest the following guideline to set this value: 
\[(2 \cdot \text{min} + \text{max})/(3 \cdot \text{avpkt})\].

• **bandwidth [BPS]**: This value is optional and used to calculate the average queue size after any idle time. It should be set to the bandwidth of the interface. This parameter does not limit the rate. The default value is 10Mbps.

• **ecn**: This parameter enables RED to notify remote hosts that their rate exceeds the amount of bandwidth available. Non-ECN capable hosts can only be notified by dropping a packet.

• **probability**: This value specifies the dropping probability after the average queue length surpass the min threshold. It is specified as a floating point from 0.0 to 1.0. Suggested values are 0.01 or 0.02 (1% or 2% respectively).

• **adaptative**: This parameter sets a dynamic value to the dropping probability. This value varies from 1% to 50%.

In this lab, we will use the RED AQM algorithm to contain the queue size at the egress port of a router.

## 2 Lab topology

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

![Figure 2. Lab topology.](image)

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

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.

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.

![Terminal opening on host h1](image)

Figure 6. Opening a terminal on host h1.

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

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

![Successful ping test](image)

Figure 7. Connectivity test using ping command.

2.1 Emulating high-latency WAN

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

Step 1. Launch a Linux terminal by holding the Ctrl+Alt+T keys or by clicking on the Linux terminal icon.
The Linux terminal is a program that opens a window and permits you to interact with a command-line interface (CLI). A CLI is a program that takes commands from the keyboard and sends them to the operating system to perform.

**Step 2.** In the terminal, type the command below. When prompted for a password, type `password` and hit Enter. This command introduces 20ms delay to switch S1’s `s1-eth1` interface.

```
sudo tc qdisc add dev s1-eth1 root netem delay 20ms
```

![Figure 9. Adding delay of 20ms to switch S1’s s1-eth1 interface.](image)

### 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.

![Figure 10. Output of `ping 10.0.0.3` command.](image)
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`.

![Figure 11. Output of `ping 10.0.0.3` command.](image)

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

### 3 Testing throughput on a network using Drop Tail AQM algorithm

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

#### 3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size

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

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

\[
\text{RTT} = 0.02 \text{ seconds}
\]

\[
\text{BDP} = 1,000,000,000 \cdot 0.02 = 20,000,000 \text{ bits}
\]
\[ = 2,500,000 \text{ bytes} \approx 2.5 \text{ Mbytes} \]

1 Mbyte = \(1024^2\) bytes

BDP = 2.5 Mbytes = \(2.5 \cdot 1024^2\) bytes = 2,621,440 bytes

The default buffer size in Linux is 16 Mbytes, and only 8 Mbytes (half of the maximum buffer size) can be allocated. Since 8 Mbytes is greater than 2.5 Mbytes, then no need to tune the buffer sizes on end-hosts. However, in upcoming tests, we configure the buffer size on the switch to 10-BDP. In addition, to ensure that the bottleneck is not the hosts’ TCP buffers, we configure the buffers to 20-BDP (52,428,800).

**Step 1.** Now, we have calculated the maximum value of the TCP sending and receiving buffer size. In order to change the receiving buffer size, on host h1’s terminal type the command shown below. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum). The maximum value is doubled (2 \cdot 10\cdot BDp) as Linux only allocates half of the assigned value.

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

![Figure 12. Receive window change in `sysctl`]

The returned values are measured in bytes. 10,240 represents the minimum buffer size that is used by each TCP socket. 87,380 is the default buffer which is allocated when applications create a TCP socket. 52,428,800 is the maximum receive buffer that can be allocated for a TCP socket.

**Step 2.** To change the current send-window size value(s), use the following command on host h1’s terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum). The maximum value is doubled as Linux allocates only half of the assigned value.

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

![Figure 13. Send window change in `sysctl`]

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

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

**Figure 14.** Receive window change in `sysctl`

**Step 4.** To change the current send-window size value(s), use the following command on host h1’s terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum). The maximum value is doubled as Linux allocates only half of the assigned value.

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

**Figure 15.** Send window change in `sysctl`

The returned values are measured in bytes. 10,240 represents the minimum buffer size that is used by each TCP socket. 87,380 is the default buffer which is allocated when applications create a TCP socket. 52,428,800 is the maximum receive buffer that can be allocated for a TCP socket.

### 3.2 Setting switch S2’s buffer size to 10 ⋅ BDP

**Step 1.** Apply `tbf` rate limiting rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit `Enter`.

- `rate`: 1gbit
- `burst`: 500,000
- `limit`: 26,214,400

```bash
sudo tc qdisc add dev s2-eth2 root handle 1: tbf rate 1gbit burst 500000 limit 26214400
```

**Note:** The `limit` value is calculated as `10 × BDP`.
3.3 Throughput and latency tests

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

```bash
iperf3 -s
```

![Starting iPerf3 server on host h3.](image)

**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`.

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

![Plotting the queue occupancy on switch S2’s s2-eth2 interface.](image)

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through `s2-eth2` interface on switch S2, the queue occupancy is constantly 0.
Step 3. In host h1, create a directory called $Drop_Tail$ and navigate into it using the following command:

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

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

Step 4. Type the following iPerf3 command in host h1’s terminal without executing it. The $-J$ option is used to display the output in JSON format. The redirection operator $<$ is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

Figure 21. Running iPerf3 client on host h1.

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

```
ping 10.0.0.3 -c 90
```

Figure 19. Queue occupancy on switch S2’s s2-eth2 interface.
**Step 6.** Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal. Then, go back to the queue plotting window and observe the queue occupancy.

![Figure 23. Queue occupancy on switch S2’s s2-eth2 interface.](image)

The graph above shows that the queue occupancy peaked at $2.5 \cdot 10^7$, which is the maximum buffer size we configure on the switch.

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

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

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

![Figure 24. Generate plotting files and entering the results directory.](image)

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

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

![Figure 24. Generate plotting files and entering the results directory.](image)
The figure above shows the iPerf3 test output report for the last 90 seconds. The average achieved throughput is approximately 900 Mbps. We can see now that the maximum throughput was almost achieved (1 Gbps) when we set the switch’s buffer size to $10 \cdot \text{BDP}$.

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

```
xdg-open RTT.pdf
```
The graph above shows that the RTT was approximately 200,000 microseconds (200ms). The output shows that there is bufferbloat as the average latency is at least ten times greater than the configured delay (20ms).

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

The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 20.083, 192.823, 228.407, and 26.954 milliseconds, respectively. The
output also verifies that there is bufferbloat as the average latency (192.823) is significantly greater than the configured delay (20ms).

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

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

![Figure 30. Opening the cwnd.pdf file.](image)

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

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

### Configuring RED on switch S2

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

**Step 1.** Apply `tbf` rate limiting rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit Enter.

- **limit**: 26,214,400
- **max**: 8,738,133
- **min**: 2,184,533
- **burst**: 2185
- **avpkt**: 1000
- **bandwidth**: 1gbit
- **adaptive**

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

![Figure 32. Setting RED parameters on switch S2’s s2-eth2 interface.](image)

4.2 Throughput and latency tests

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

```
iperf3 -s
```

![Figure 33. Starting iPerf3 server on host h3.](image)

**Step 2.** In the Client’s terminal, type the command below to plot the switch’s queue in real-time. When prompted for a password, type `password` and hit Enter.

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

![image]
A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through $s2$-$eth2$ interface on switch S2, the queue occupancy is constantly 0.

![Figure 35. Queue occupancy on switch S2's s2-eth2 interface.](image)

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

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

![Figure 36. Creating and navigating into directory RED.](image)

**Step 4.** Type the following *iperf3* command in host h1’s terminal without executing it. The `-J` option is used to display the output in JSON format. The redirection operator `>` is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

![Figure 37. Running iPerf3 client on host h1.](image)

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

```
ping 10.0.0.3 -c 90
```
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.

The graph above shows that the queue occupancy peaked around $3.5 \cdot 10^6$ bytes, which is closer to a buffer of BDP size.

Step 7. In the queue plotting window, press the [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
```

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

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

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

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

**Step 11.** Close the RTT.pdf window then go back to h2’s terminal to see the ping output.
The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 26.833, 34.048, 38.824, and 3.311 milliseconds, respectively. The output also verifies that there is not bufferbloat as the average latency (34.048) is not significantly greater than the configured delay (20ms).

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

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

![Figure 46. Opening the cwnd.pdf file.](image)

![Figure 47. Evolution of the congestion window.](image)

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

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

### 4.3 Changing the bandwidth to 100Mbps

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

**Step 1.** Apply rate limiting rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit Enter.

- **rate**: 100mbit
- **burst**: 50,000
- **limit**: 26,214,400

```
sudo tc qdisc change dev s2-eth2 root handle 1: tbf rate 100mbit burst 50000 limit 2621440
```

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

### 4.4 Throughput and latency tests

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

```
iperf3 -s
```

Figure 49. Starting iPerf3 server on host h3.

**Step 2.** In the Client’s terminal, type the command below to plot the switch’s queue in real-time. When prompted for a password, type `password` and hit Enter.

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

Figure 50. Plotting the queue occupancy on switch S2’s s2-eth2 interface.
A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through s2-eth2 interface on switch S2, the queue occupancy is constantly 0.

![Figure 51. Queue occupancy on switch S2's s2-eth2 interface.](image)

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

```
cd ..
```

![Figure 52. Creating and navigating into directory 1BDP.](image)

**Step 4.** Type the following iPerf3 command in host h1’s terminal without executing it. The `-J` option is used to display the output in JSON format. The redirection operator `>` is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

![Figure 53. Running iPerf3 client on host h1.](image)

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

```
ping 10.0.0.3 -c 90
```
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.

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

Step 7. In the queue plotting window, press the ⏹️ 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
```

Step 9. Open the throughput file using the command below on host h1.
The figure above shows the iPerf3 test output report for the last 90 seconds. The average achieved throughput is 100 Mbps.

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

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

Figure 58. Measured throughput.

Figure 59. Opening the `RTT.pdf` file.
The graph above shows that the RTT increased from approximately ten times the default latency (20ms). The output above shows that there is a bufferbloat problem as the average latency is significantly greater. Since RED is configured to operate on a 1 Gbps link, for this test the point of operation changed therefore, unnecessary delay is observed.

**Step 11.** Close the *RTT.pdf* window then go back to h2’s terminal to see the `ping` output.
The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 148.914, 186.175, 468.728 and 33.481 milliseconds, respectively. The output also verifies that there is a bufferbloat problem as the average latency (186.175) is significantly greater than the configured delay (20ms).

**Step 12.** Close the \textit{RTT.pdf} window then open the congestion window (\textit{cwnd.pdf}) file using the command below.

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

![Figure 62. Opening the \textit{cwnd.pdf} file.](image)

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

**Step 13.** To stop \textit{iperf3} server in host h3 press \texttt{Ctrl+c}.

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

**References**

NETWORK TOOLS AND PROTOCOLS

Lab 17: Stochastic Fair Queueing (SFQ)

Document Version: 11-12-2019

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 FIFO queueing discipline ........................................................................................ 3
   1.2 SFQ queueing discipline ....................................................................................... 4
2 Lab topology .................................................................................................................. 5
   2.1 Starting host h1, host h2, host h3 and host h4 ....................................................... 7
   2.2 Emulating high-latency WAN .............................................................................. 8
   2.3 Testing connection ................................................................................................. 9
3 Testing the throughput of two competing TCP flows .................................................. 9
   3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size ............................... 10
   3.2 Modifying hosts’ buffer size ................................................................................. 10
   3.3 Changing congestion control algorithm in host h1 and host h2 ......................... 13
   3.4 Setting switch S2’s buffer size to BDP ................................................................. 14
   3.5 Throughput tests .................................................................................................. 14
4 Configuring SFQ on switch S2 ..................................................................................... 17
   4.1 Setting SFQ parameter on switch S2’s egress interface ......................................... 18
   4.2 Throughput tests .................................................................................................. 18
References ........................................................................................................................ 22
Overview

This lab introduces to Stochastic Fair Queuing (SFQ), which is a queueing discipline aimed to ensure fairness between TCP flows. The lab describes the steps to conduct throughput tests that shows the benefits of isolating the dynamic of competing TCP flows by applying SFQ rules to a router egress’ interface.

Objectives

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

1. Identify and describe the components of end-to-end latency.
2. Understand the scheduling process in a router.
3. Explain the impact of using SFQ to isolate the dynamic of competing TCP flows.
4. Visualize the interaction of competing TCP flows after SFQ is configured on a router’s interface.

Lab settings

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

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Testing the throughput of two competing TCP flows
4. Section 4: Configuring SFQ on switch S2.

1 Introduction

1.1 FIFO queueing discipline
First-in, first-out (FIFO) queuing is the most basic queue scheduling discipline. It is also the default queueing discipline in Linux and most of the routers. In FIFO queuing, all packets are treated equally by placing them into a single queue, and then delivering them in the same order that they were placed into the queue (see Figure 1).

![Figure 1. FIFO queuing discipline.](image)

FIFO queuing presents advantages for software-based routers due to its low computational load on the system when compared with other queueing disciplines. Additionally, the behavior of a FIFO queue is very predictable, which means that packets are not reordered, and the queue delay is determined by the maximum depth of the queue. If the queue depth remains short, FIFO queuing provides simple contention resolution for network resources without adding significantly delay to the link.

However, a single FIFO queue does not allow routers to classify packets or set priorities. If a router uses a single FIFO queue, it will impact all flows equally, this means that the average queuing delay for all flows increases as congestion increases. As a result, FIFO queuing can result in increased delay, jitter, and loss for real-time applications. Another limitation of FIFO queuing is that bursty TCP flows can consume the entire buffer space of a FIFO queue, and that causes all other flows to be denied service until after the burst is serviced. This can result in increased delay, jitter, and loss for the other coexistent TCP flows.

### 1.2 SFQ queueing discipline

Stochastic Fairness Queueing (SFQ) is a classless queueing discipline available for traffic control. SFQ does not shape traffic but only schedules the transmission of packets, based on classified flows. The goal of the algorithm is to ensure fairness so that each flow can send data in turn thus, preventing any single flow from drowning out the rest. This feature may in fact have some effect in mitigating a Denial of Service attempts. SFQ is work-conserving and therefore always delivers a packet if it has one available.

SFQ classifies flows in a fixed set of queues serviced in strict round-robin order. The maximum number of queues is configurable (1024 by default in the Linux implementation). In order to assign a queue to an ingress packet, a hash function is applied to its 5-tuple determined by the IP source and destination, layer 4 port source and destination and layer 4 protocol number. Packets with the same hash are assigned to the same queue. Because of the hash, multiple sessions might end up in the same bucket,
which would halve each session’s chance of sending a packet, thus halving the effective speed available. To prevent this situation from becoming noticeable, SFQ changes its hashing algorithm quite often so that any two colliding sessions will only do so for a small number of seconds.

![Figure 2. SFQ queuing discipline.](image)

The basic `tc qdisc` syntax used with `sfq` is as follows:

```
tc qdisc [add | ...] dev [dev_id] root sfq perturb [seconds] quantum
```

- **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.
- **sfq**: this parameter specifies the Stochastic Fair Queueing algorithm.
- **perturb**: It is used to specify the interval in seconds for queue algorithm perturbation. If the value is set to 0 indicate that no perturbation occurs. It is recommended avoiding low values to prevent packet reordering. Empirical evaluations recommend this value set to 10 seconds.
- **quantum**: Denotes the number of bytes which a flow can dequeue during a round of the round-robin process. It is recommended to set this value not less than the Maximum Transmission Unit (MTU).

## 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.
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`.

**Step 2.** On MiniEdit’s menu bar, click on *File* then *Open* to load the lab’s topology. Locate the `Lab 17.mn` topology file and click on *Open*. 
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.](image)

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 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.](image)

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.
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.](image)

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

**Step 2.** In the terminal, type the command below. When prompted for a password, type password and hit Enter. This command introduces 20ms delay to switch S1’s $s1$-eth1 interface.

```
sudo tc qdisc add dev s1-eth1 root netem delay 20ms
```

![Figure 10. Adding delay of 20ms to switch S1’s $s1$-eth1 interface.](image)
2.3 Testing connection

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

**Step 1.** On the terminal of host h1, type `ping 10.0.0.3`. 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.

![Figure 1](image1.png)

Figure 1. 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`.

![Figure 2](image2.png)

Figure 2. 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 the throughput of two competing TCP flows
In this section, you are going to tune the network devices in order to emulate a Wide Area Network (WAN). First, you will set the hosts’ TCP buffers to \( 8 \cdot \text{BDP} \) therefore, the bottleneck is not in the end-hosts. Then, you will add 20ms latency to switch S1’s s1-eth1 interface. Additionally, you will set the bottleneck bandwidth to 1Gbps in switch S2’s s2-eth2 interface. Finally, you will conduct throughput tests between two competing TCP flows which uses different congestion control algorithms (i.e. Cubic, BBR).

### 3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size

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

\[
\text{BW} = 1,000,000,000 \text{ bits/second} \\
\text{RTT} = 0.02 \text{ seconds} \\
\text{BDP} = 1,000,000,000 \cdot 0.02 = 20,000,000 \text{ bits} \\
\text{= } 2,500,000 \text{ bytes } \approx 2.5 \text{ Mbytes} \\
1 \text{ Mbyte } = 1024^2 \text{ bytes} \\
\text{BDP } = 2.5 \text{ Mbytes } = 2.5 \cdot 1024^2 \text{ bytes } = 2,621,440 \text{ bytes}
\]

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 BDP. In addition, to ensure that the bottleneck is not the hosts’ TCP buffers, we configure the buffers to \( 8 \cdot \text{BDP} \) (20,971,520).

### 3.2 Modifying hosts’ buffer size

For the following calculation, the bottleneck bandwidth is considered as 1 Gbps, and the round-trip time latency as 20ms.

\[
\text{BW} = 1,000,000,000 \text{ bits/second} \\
\text{RTT} = 0.02 \text{ seconds} \\
\text{BDP} = 1,000,000,000 \cdot 0.02 = 20,000,000 \text{ bits} \\
\text{= } 2,500,000 \text{ bytes } \approx 2.5 \text{ Mbytes}
\]

In order to have enough TCP buffer size, we will set the TCP sending and receiving buffer to \( 8 \cdot \text{BDP} \) in all hosts.
The send and receive TCP buffer sizes should be set to $8 \cdot \text{BDP}$ to ensure the bottleneck is not in the end-hosts. For simplicity, we will use 2.5 Mbytes as the value for the BDP instead of 2,500,000 bytes.

1 Mbyte = $1024^2$ bytes

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

$8 \cdot \text{BDP} = 8 \cdot 2,621,440$ bytes = 20,971,520 bytes

**Step 1.** At this point, 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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

![Figure 13. Receive window change in `sysctl`.](image)

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. 20,971,520 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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

![Figure 14. Send window change in `sysctl`.](image)

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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```
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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```
3.3 Changing congestion control algorithm in host h1 and host h2

In this part, you will set different congestion control algorithms in the host h1 and host h2 to cubic and BBR respectively. Consequently, you will have two TCP flows with different dynamic sharing the same bottleneck link.

The default congestion avoidance algorithm in the following test is cubic thus, there is no need to specify it manually.

**Step 1.** Verify that the congestion control algorithm is cubic by issuing the following command in host h1 terminal:

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

**Figure 21.** Verifying TCP congestion control algorithm in host h1.

**Step 2.** In host h2 terminal, type the following command to change the current TCP congestion control algorithm to BBR.

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

**Figure 22.** Changing TCP congestion control algorithm in host h2.
### 3.4 Setting switch S2’s buffer size to BDP

**Step 1.** Apply rate limiting rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit Enter.

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

```
sudo tc qdisc add dev s2-eth2 root: handle 1: tbf rate 1gbit burst 500000 limit 2621440
```

![Command output](image)

Figure 23. Limiting rate to 1 Gbps and setting the buffer size to BDP on switch S2’s interface.

### 3.5 Throughput tests

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

```
iperf3 -s
```

![Starting iPerf3 server on h3](image)

Figure 24. Starting iPerf3 server on host h3.

**Step 2.** Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

![Starting iPerf3 server on h4](image)

Figure 25. Starting iPerf3 server on host h4.
The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the iperf3 commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 4 and Step 7 without executing them next, in Step 8 you will press Enter in host h1 and host h2 to execute them.

**Step 3.** In host h1, create a directory called *h1_no_SFQ* and navigate into it using the following command:

```
mkdir h1_no_SFQ && h1_no_SFQ
```

![Figure 26. Creating and navigating into directory h1_no_SFQ.](image)

**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 60 -J > out.json
```

![Figure 27. Running iPerf3 client on host h1.](image)

**Step 5.** In host h2, create a directory called *h2_no_SFQ* and navigate into it using the following command:

```
mkdir h2_no_SFQ && h2_no_SFQ
```

![Figure 28. Creating and navigating into directory h2_no_SFQ.](image)

**Step 6.** Type the following iPerf3 command in host h2’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 10.0.0.4 -t 60 -J > out.json
```
Step 7. Press *Enter* to execute the commands shown in step 4 and step 6, first in host h1 terminal then, in host h3 terminal.

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

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

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

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

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is approximately 100 Mbps. It is observed that the Cubic flow collapses significantly since the link is not fairly shared with the other TCP flow.
Step 10. Close the *throughput.pdf* window then, in host h2, proceed similarly by typing the following command:

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

Figure 33. Generate plotting files and entering the *results* directory.

Step 11. In host h2 terminal, open the throughput file using the following command:

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

Figure 34. Opening the *throughput.pdf* file.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 850 Mbps. It is observed that the BBR flow takes over almost the full link which upholds that the link is not fairly shared with the other TCP flow.

Step 12. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

4 Configuring SFQ on switch S2
In this section, you are going to configure Stochastic Fair Queueing (SFQ) in switch S2’s s2-eth2 interface. Then, you will conduct throughput and latency measurements between host h1 and host h3.

4.1 Setting SFQ parameter on switch S2’s egress interface

**Step 1.** Apply sfq rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type password and hit Enter.

- `sudo tc qdisc add dev s2-eth2 parent 1: handle 2: sfq perturb 10`

![Figure 36. Setting SFQ parameters on switch S2’s s2-eth2 interface.](image)

4.2 Throughput tests

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

`iperf3 -s`

![Figure 37. Starting iPerf3 server on host h3.](image)

**Step 2.** Launch iPerf3 in server mode on host h4’s terminal.

`iperf3 -s`

![Figure 38. Starting iPerf3 server on host h4.](image)
The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the `iperf3` commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 4 and Step 7 without executing them next, in Step 8 you will press `Enter` in host h1 and host h2 to execute them.

**Step 3.** In host h1, create a directory called `h1_SFQ` and navigate into it using the following command:

```
cd ../.. && mkdir h1_SFQ && cd h1_SFQ
```

![Figure 39. Creating and navigating into directory h1_SFQ.](image)

**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 60 -J > out.json
```

![Figure 40. Running `iperf3` client on host h1.](image)

**Step 5.** In host h2, create a directory called `h2_SFQ` and navigate into it using the following command:

```
cd ../.. && mkdir h2_SFQ && cd h2_SFQ
```

![Figure 41. Creating and navigating into directory h2_SFQ.](image)

**Step 6.** Type the following `iperf3` command in host h2’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 10.0.0.4 -t 60 -J > out.json
```
Step 7. Press Enter to execute the commands shown in step 4 and step 6, first in host h1 terminal then, in host h3 terminal.

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

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

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

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

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is approximately 500 Mbps. It is observed that the Cubic flow uses
the half part of the link. Notice also that there are spikes every 10 seconds which is consistent with the rehashing time specified by the `perturb` parameter.

**Step 10.** In host h2, proceed similarly by typing the following command:

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

![Figure 46. Generate plotting files and entering the results directory.](image)

**Step 11.** In host h2 terminal, open the throughput file using the following command:

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

![Figure 47 Opening the throughput.pdf file.](image)

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is approximately 500 Mbps. It is observed that the BBR can fairly coexist with a Cubic flow since the dynamic of both flows do not interact as a result of SFQ being configured in switch S2’s `s2-eth2` interface.

**Step 12.** Close the `throughput.pdf` window then, to stop `iperf3` server in host h3 and host h4 press `Ctrl+c`.
This concludes Lab 17. Stop the emulation and then exit out of MiniEdit.

References

NETWORK TOOLS AND PROTOCOLS

Lab 18: Controlled Delay (CoDel) Active Queue Management

Document Version: 11-18-2019

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 ............................................................................................................................ 4  
1.1 CoDel active queue management .................................................................................... 4  
1.2 Fair queueing CoDel active queue management ............................................................... 6  
2 Lab topology ........................................................................................................................... 8  
2.1 Starting host h1, host h2, host h3 and host h4 ................................................................. 9  
2.2 Emulating high-latency WAN ....................................................................................... 10  
2.3 Testing connection ........................................................................................................... 11  
3 Testing the throughput of two competing TCP flows ........................................................ 12  
3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size .......................................... 12  
3.2 Modifying hosts’ buffer size ............................................................................................ 13  
3.3 Changing congestion control algorithm in host h1 and host h2 ................................... 16  
3.4 Setting switch S2’s buffer size to BDP .......................................................................... 16  
3.5 Throughput tests ............................................................................................................. 17  
4 Configuring CoDel on switch S2 .......................................................................................... 24  
4.1 Setting CoDel parameter on switch S2’s egress interface ............................................. 24  
4.2 Throughput tests ............................................................................................................. 25  
5 Configuring Fq_CoDel on switch S2 .................................................................................. 31  
5.1 Setting Fq_CoDel parameters on switch S2’s egress interface ...................................... 31  
5.2 Throughput test .............................................................................................................. 32  
6 Changing the bandwidth to 100Mbps ................................................................................. 39  
6.1 Throughput and latency tests ......................................................................................... 39  
References ............................................................................................................................... 46
Overview

This lab explains how Controlled Delay (CoDel) Active Queue Management (AQM) algorithm is used to manage outgoing TCP traffic in order to achieve low end-to-end latency and ensure fairness. Additionally, it is presented FQ_CoDel (Fair Queueing with Controlled Delay) AQM, a combination of fair queueing and CoDel algorithms which is aimed to mitigate bufferbloat and ensure fairness. Along this lab, throughput, latency and queue occupancy measurements are conducted in an emulated high-latency network showing the features of both algorithms.

Objectives

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

1. Identify and describe the components of end-to-end latency.
2. Understand the features of AQM algorithms.
3. Explain how CoDel algorithm contains the queue length in order to avoid bufferbloat.
4. Observe how FQ_CoDel ensures low end-to-end latency and fairness.
5. Visualize the benefits of isolating the dynamic of competing TCP flows.

Lab settings

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

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Testing the throughput of two competing TCP flows
4. Section 4: Configuring CoDel on switch S2.
5. Section 5: Configuring FQ_CoDel on switch S2.
6. Section 6: Changing the bandwidth to 100Mbps.
Introduction

The persistently problem of bufferbloat has been discussed since the early 80’s. Bufferbloat is caused by oversizing router’s queues that hold traffic that cannot be immediately forwarded consequently, unnecessary latency is experienced. The deployment of Active Queue Management (AQM) algorithms started being notorious in 1998. The aim of those algorithms is to solve the increasing problem of bufferbloat. Despite this awareness, the problem has only gotten worse as growth in memory density per Moore’s Law fueled an exponential increase in buffer pool size. Efforts to deploy AQM algorithms have been frustrated by difficult configuration and negative impact on network utilization. This bufferbloat problem has become more and more important throughout the Internet but particularly at the consumer edge. Queue management has become more critical due to increased consumer use of the Internet, mixing large video transactions with time-critical VoIP and gaming.1

Network buffers are designed to absorb the packet bursts that occur naturally in statistically multiplexed networks. Buffers helpfully absorb the queues created by reasonable packet network behavior such as short-term mismatches in traffic arrival and departure. Unfortunately, other less useful network behaviors can cause queues to fill, and their effects are not nearly as benign. Discussion of these issues and the reason why the solution is not simply creating smaller buffers. To understand queue management, it is critical to understand the difference between the necessary, useful "good" queue and the counterproductive "bad" queue. An effective AQM algorithm remediates bufferbloat at a bottleneck without affecting the hops where buffers are not bloated. However, the development and deployment of AQM algorithms are frequently subject to misconceptions about the cause of packet queues in network buffers.

1.1 CoDel active queue management

In order to mitigate the bufferbloat problem, it is presented Controlled Delay (CoDel), an innovative Active Queue Management (AQM) that adapts to changing links rates and it is suitable for deployment and experimentation in Linux-based routers. The goal of CoDel is to contain the queuing latency while maximizing the throughput. CoDel does not require any parameters tuning and it has been designed to work across a wide range of conditions with different links and round-trip times. Figure 1 illustrate a scenario where CoDel AQM is used to manage the queue. Firstly, a timestamp is added to every incoming packet. Then, by measuring the departure time of every packet in the queue, it is determined for how long a packet was waiting in the queue. Consequently, CoDel algorithm determines whether the enqueued packets are going to be dropped or not.
The algorithm depicted in Listing 1 shows in more detail how CoDel works: as mentioned before, timestamp is added to each packet at the ingress side of the queue in order to measure the packet departure time at the egress side when packets are dequeued. The departure time is then compared to a target delay (5ms by default), if it is below the target the packet is forwarded, otherwise the algorithm starts a timer and forwards the packet. When packets are dequeued, CoDel checks the departure time, and if it gets below the target, the timer is stopped. However, if the timer reaches the value of interval (100ms by default), CoDel enters the dropping state which is left when the departure time gets below the target delay. During this state, whenever the interval timer expires a packet is dropped, the timer is reset, and the next timer duration is reconfigured. Thus, the longer the departure time stays above the target delay, the higher the packet dropping frequency. Finally, as soon as the measured departure time of an outgoing packet gets below the target, the dropping state is left, and interval is restored to its default value.

```
Paket p
StatefulObject s
if(p.qdelay < TARGET)
    s.dropping = false
    count = 0
if(s.dropping == false)
    s.dropping = true
    s.drop_next_packet = now + INTERVAL
if(s.dropping && s.drop_next_packet >= now)
    drop()
    count++
    s.drop_next_packet = now + INTERVAL / sqrt(now)
```

Listing 1. Simplified CoDel pseudocode.

In summary, CoDel algorithm considers three scenarios:

- If the queueing delay is below \( \text{TARGET} \), a packet is never dropped.
- If \( \text{TARGET} \) is exceeded by more than \( \text{INTERVAL} \) time units, the first packet will be dropped.
- From \( \text{now} \), the interval between dropping packets is getting smaller, until \( \text{TARGET} \) delay is reached.

The complete algorithm considers other factors like the duration since the last dropping phase. The Linux implementation used in this lab is based on the full algorithm.
The basic **codel** syntax used with `tc` is as follows:

```
```

- **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.
- **codel**: this parameter enables the Controlled Delay (CoDel) algorithm.
- **limit**: this parameter specifies hard limit on the real queue size in packets. When this limit is reached, incoming packets are dropped. If the value is lowered, packets are dropped so that the new limit is met. The default value is 1000 packets.
- **target**: denotes the acceptable minimum standing/persistent queue delay. This minimum delay is identified by tracking the local minimum queue delay that packets experience. The default and recommended value is 5ms.
- **interval**: this parameter is intended to ensure that the measured minimum delay does not become too stale. The minimum delay must be experienced in the last epoch of length `interval`. It should be set on the order of the worst-case RTT through the bottleneck to give endpoints sufficient time to react. The default value is 100ms.
- **ecn|noecn**: it is used to mark packets instead of dropping them. If `ecn` has been enabled, `noecn` can be used to turn it off and vice-a-versa. By default, `ecn` is turned off.
- **ce_threshold**: sets a threshold above which all packets are marked with ECN Congestion Experienced. This is useful for DCTCP-style congestion control algorithms that require marking at very shallow queueing thresholds.

### 1.2 Fair queueing CoDel active queue management

FQ_CoDel (Fair Queuing Controlled Delay) is a queuing discipline that combines Fair Queuing with the CoDel AQM scheme. FQ_CoDel uses a stochastic model to classify incoming packets into different flows. It is aimed to provide a fair share of the bandwidth to all the flows using the queue.

Figure 2 shows that FQ_CoDel consists of two logical parts: 1) the scheduler, which selects which queue to dequeue a packet from, and 2) The CoDel AQM which works on each of the queues. Since FQ_CoDel mixes packets from multiple flows, it reduces the impact of bursty traffic. It also provides isolation for applications namely DNS, web and videoconferencing traffic. Additionally, it improves network utilization by keeping the
queue lengths short and it can be implemented in a memory and CPU efficient fashion across a wide range of hardware.

![Buffer managed by FQ_CoDel AQM.](image)

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

```
```

- **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.
- **fq_codel**: this parameter enables the Fair Queueing Controlled Delay (FQ_CoDel) algorithm.
- **limit**: this parameter specifies hard limit on the real queue size in packets. When this limit is reached, incoming packets are dropped. If the value is lowered, packets are dropped so that the new limit is met. The default value is 1000 packets.
- **target**: denotes the acceptable minimum standing/persistent queue delay. This minimum delay is identified by tracking the local minimum queue delay that packets experience. The default and recommended value is 5ms.
- **flows**: this parameter specifies the number of flows into which the incoming packets are classified. Due to the stochastic nature of hashing, multiple flows may end up being hashed into the same slot. Newer flows have priority over older ones. This parameter can be set only at load time since memory has to be allocated for the hash table. Default value is 1024.
- **interval**: this parameter is intended to ensure that the measured minimum delay does not become too stale. The minimum delay must be experienced in the last epoch of length `interval`. It should be set on the order of the worst-case
RTT through the bottleneck to give endpoints sufficient time to react. The default value is 100ms.

- **quantum**: denotes the number of bytes used as deficit in the fair queuing algorithm. The default value is 1514 bytes which corresponds to the Ethernet MTU plus the hardware header length of 14 bytes.
- **ecn|noecn**: it is used to mark packets instead of dropping them. If ecn has been enabled, noecn can be used to turn it off and vice-a-versa. By default, ecn is turned off.
- **ce_threshold**: sets a threshold above which all packets are marked with ECN Congestion Experienced. This is useful for DCTCP-style congestion control algorithms that require marking at very shallow queueing thresholds.

In this lab, we will use codel and fq_codel AQM algorithms to control the queue size at the egress port of a router.

## 2 Lab topology

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

![Lab topology](image)

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`.

![MiniEdit shortcut](image)

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 18.mn topology file and click on Open.

![Figure 5. MiniEdit’s Open dialog.](image)

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.](image)

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 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 host h3 and open their Terminals.

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

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

**Step 2.** In the terminal, type the command below. When prompted for a password, type `password` and hit `Enter`. This command introduces 20ms delay to Switch S1’s `s1-eth1` interface.

```
sudo tc qdisc add dev s1-eth1 root netem delay 20ms
```

2.3 Testing connection

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

**Step 1.** On the terminal of host h1, type `ping 10.0.0.3`. 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.
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`.

![Figure 12. Output of `ping 10.0.0.3` command.](image)

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 the throughput of two competing TCP flows

In this section, you are going to tune the network devices in order to emulate a Wide Area Network (WAN). First, you will set the hosts’ TCP buffers to $20 \cdot \text{BDP}$ therefore, the bottleneck is not in the end-hosts. Then, you will add 20ms latency to switch S1’s s1-eth1 interface. Additionally, you will set the bottleneck bandwidth to 1Gbps in switch S2’s s2-eth2 interface however, the buffer will be intentionally oversized to $20 \cdot \text{BDP}$ therefore, you should expect bufferbloat. Finally, you will conduct throughput tests between two competing TCP flows which uses different congestion control algorithms (i.e. Cubic, BBR).

#### 3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size

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

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

\[
\text{RTT} = 0.02 \text{ seconds}
\]

\[
\text{BDP} = 1,000,000,000 \cdot 0.02 = 20,000,000 \text{ bits}
\]
= 2,500,000 bytes ≈ 2.5 Mbytes

1 Mbyte = 1024^2 bytes

BDP = 2.5 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 BDP. In addition, to ensure that the bottleneck is not the hosts’ TCP buffers, we configure the buffers to 20·BDP (52,428,800).

### 3.2 Modifying hosts’ buffer size

For the following calculation, the bottleneck bandwidth is considered as 1 Gbps, and the round-trip time latency as 20ms.

In order to have enough TCP buffer size, we will set the TCP sending and receiving buffer to 20 · BDP in all hosts.

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

\[
\text{RTT} = 0.02 \text{ seconds}
\]

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

\[
= 2,500,000 \text{ bytes} ≈ 2.5 \text{ Mbytes}
\]

The send and receive TCP buffer sizes should be set to 20 · BDP to ensure the bottleneck is not in the end-hosts. For simplicity, we will use 2.5 Mbytes as the value for the BDP instead of 2,500,000 bytes.

1 Mbyte = 1024^2 bytes

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

20 · BDP = 20 \cdot 2,621,440 \text{ bytes} = 52,428,800 \text{ bytes}

**Step 1.** At this point, 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).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 52428800'
```
Controlled Delay (CoDel) Active Queue Management

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).

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

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 52,428,800 (maximum).

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

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 52,428,800 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 52428800'
```
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 52,428,800 (maximum).

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

Figure 17. 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 52,428,800 (maximum).

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

Figure 18. 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 52,428,800 (maximum).

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

Figure 19. 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 52,428,800 (maximum).

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

Figure 20. Send window change in `sysctl`.
3.3 Changing congestion control algorithm in host h1 and host h2

In this part, you will set different congestion control algorithms in the host h1 and host h2 to cubic and BBR respectively. Consequently, you will have two TCP flows with different dynamic sharing the same bottleneck link.

The default congestion avoidance algorithm in the following test is cubic thus, there is no need to specify it manually.

Step 1. Verify that the congestion control algorithm is cubic by issuing the following command in host h1 terminal:

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

![Figure 21. Verifying TCP congestion control algorithm in host h1.](image)

Step 2. In host h2 terminal, type the following command to change the current TCP congestion control algorithm to BBR.

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

![Figure 22. Changing TCP congestion control algorithm in host h2.](image)

3.4 Setting switch S2’s buffer size to BDP

Step 1. Apply \texttt{tbf} rate limiting rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type \texttt{password} and hit Enter.

- \texttt{rate}: 1gbit
- \texttt{burst}: 500,000
- \texttt{limit}: 26,214,400

```
sudo tc qdisc add dev s2-eth2 root: handle 1: tbf rate 1gbit burst 500000 limit 26214400
```

![Figure 23. Applying \texttt{tbf} rate limiting rule.](image)
3.5 Throughput tests

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

```bash
iperf3 -s
```

![Figure 24. Starting iPerf3 server on host h3.](image)

**Step 2.** Launch iPerf3 in server mode on host h4’s terminal.

```bash
iperf3 -s
```

![Figure 25. Starting iPerf3 server on host h4.](image)

The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the `iperf3` commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 4 and Step 6 without executing them next, in Step 7 you will press Enter in host h1 and host h2 to execute them.

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

```bash
mkdir h1.Drop_Tail && cd h1.Drop_Tail
```
Step 4. Type the following iPerf3 command in host h1’s terminal without executing it. The -J option is used to display the output in JSON format. The redirection operator > is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 60 -J > out.json
```

Figure 27. Running iPerf3 client on host h1.

Step 5. In host h2, create a directory `h1_Drop_Tail` and navigate into it using the following command:

```
mkdir h2_Drop_Tail && cd h2_Drop_Tail
```

Figure 28. Creating and navigating into directory `h2_no_SFQ`.

Step 6. Type the following iPerf3 command in host h2’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 10.0.0.4 -t 60 -J > out.json
```

Figure 29. Running iPerf3 client on host h2.

Step 7. In the Client’s terminal, type the command below to plot the switch’s queue in real-time. When prompted for a password, type `password` and hit Enter.

```
sudo plot_q.sh s2-eth2
```
A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through $s2$-eth2 interface on switch S2, the queue occupancy is constantly 0.

**Step 8.** Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal.
The graph above shows that the queue occupancy peaked at $2.5 \cdot 10^7$, which is the maximum buffer size we configure on the switch.

**Step 9.** In the queue plotting window, press the `Enter` key on your keyboard to stop plotting the queue.

**Step 10.** After the iPerf3 test finishes on host h1, enter the following command:

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

**Step 11.** Open the throughput file using the command below on host h1.

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

Figure 32. Queue occupancy on switch S2’s s2-eth2 interface.

Figure 33. Generate plotting files and entering the results directory.

Figure 34. Opening the throughput.pdf file.
The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 250 Mbps. It is observed that the Cubic flow collapses significantly since the link is not fairly shared with the other TCP flow.

**Step 12.** Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

![Figure 36. Opening the RTT.pdf file.](image)
The graph above shows that the RTT was approximately 200,000 microseconds (200ms). The output shows that there is bufferbloat as the average latency is at least ten times greater than the configured delay (20ms).

**Step 13.** Close the RTT.pdf window then, in host h2, proceed similarly by typing the following command:

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

**Step 14.** In host h2 terminal, open the throughput file using the following command:

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

**Figure 37.** Measured round-trip time.

**Figure 38.** Generate plotting files and entering the results directory.

**Figure 39.** Opening the throughput.pdf file.
The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 700 Mbps. It is observed that the BBR flow uses the major part of the available bandwidth which shows that the link is not fairly shared with the other TCP flow.

**Step 15.** Close the `throughput.pdf` window then open the Round-Trip Time (RTT) file using the command below.

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

Figure 41. Opening the `RTT.pdf` file.
The graph above shows that the RTT was approximately 200,000 microseconds (200ms) as well. The output shows that there is bufferbloat as the average latency is at least ten times greater than the configured delay (20ms).

**Step 16.** Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press Ctrl+c.

### 4 Configuring CoDel on switch S2

In the previous section you have observed bufferbloat, due to oversizing switch S2’s s2-eth2 interface. In this section, you are going to configure CoDel AQM in switch S2’s s2-eth2 interface. Then, you will conduct throughput and latency measurements between the hosts.

#### 4.1 Setting CoDel parameter on switch S2’s egress interface

**Step 1.** Apply `codel` rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit Enter.

```
sudo tc qdisc add dev s2-eth2 parent 1: handle 2: codel limit 17476 target 5ms interval 100ms
```

- **limit**: 17476
- **target**: 5ms
- **interval**: 100ms
4.2 Throughput tests

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

```
iperf3 -s
```

![Figure 44. Starting iPerf3 server on host h3.](image)

**Step 2.** Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```

![Figure 45. Starting iPerf3 server on host h4.](image)

The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the *iperf3* commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 4 and Step 6 without executing them next, in Step 7 you will press *Enter* in host h1 and host h2 to execute them.

**Step 3.** In host h1, exit from the previous folders then, create a directory called *h1_CoDel* and navigate into it using the following command:

```
cd ../.. && mkdir h1_CoDel && cd h1_CoDel
```
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.

```bash
type the following command in host h1’s terminal: iperf3 -c 10.0.0.3 -t 60 -J > out.json
```

Figure 47. Running iPerf3 client on host h1.

Step 5. In host h2, exit from the previous folders then, create a directory called h2_CoDel
and navigate into it using the following command:

```bash
cd ../.. && mkdir h2_CoDel && cd h2_CoDel
```

Figure 48. Creating and navigating into directory h2_CoDel.

Step 6. Type the following iPerf3 command in host h2’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.

```bash
type the following command in host h2’s terminal: iperf3 -c 10.0.0.4 -t 60 -J > out.json
```

Figure 49 Running iPerf3 client on host h2.

Step 7. 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.

```bash
sudo plot_q.sh s2-eth2
```
A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through s2-eth2 interface on switch S2, the queue occupancy is constantly 0.

**Step 8.** Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal.
The graph above shows that the queue occupancy peaked around 900,000 bytes, which is below the maximum buffer size we configure on the switch 26,214,400 bytes (10·BDP).

**Step 9.** In the queue plotting window, press the key on your keyboard to stop plotting the queue.

**Step 10.** After the iPerf3 test finishes on host h1, enter the following command:

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

**Step 11.** Open the throughput file using the command below on host h1.

```
xdg-open throughput.pdf
```
The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 400 Mbps. It is observed that the Cubic flow collapses significantly since the link is not fairly shared with the other TCP flow.

**Step 12.** Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

![Figure 56. Opening the RTT.pdf file.](image)

The figure over time shows the measured throughput.

![Throughput over time](image)

**Figure 55.** Measured throughput.

The measured round-trip time is shown in the following graph.

![RTT over time](image)

**Figure 57.** Measured round-trip time.
The graph above shows that the RTT was approximately 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not exceeding the configured delay (20ms).

**Step 13.** Close the *RTT.pdf* window then, in host h2, proceed similarly by typing the following command:

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

![Image of command output](image)

**Figure 58.** Generate plotting files and entering the *results* directory.

**Step 14.** In host h2 terminal, open the throughput file using the following command:

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

![Image of command output](image)

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

![Graph](image)

**Figure 60.** Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 400 Mbps. It is observed that the BBR flow takes the major part of the available bandwidth which shows that the link is not fairly shared with the other TCP flow.
Step 15. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

Figure 61. Opening the *RTT.pdf* file.

The graph above shows that the RTT was approximately 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not exceeding the configured delay (20ms).

Step 16. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

5 Configuring Fq_CoDel on switch S2

In the previous section, you have observed that despite getting low queue occupancy and, consequently mitigating bufferbloat, the link share was not fair between the two TCP flows. In this section, you are going to configure Fq_CoDel in switch S2’s `s2-eth2` interface to mitigate bufferbloat and at the same time to ensure fairness. Then, you will conduct throughput and latency measurements between the hosts.

5.1 Setting Fq_CoDel parameters on switch S2’s egress interface
Lab 18: Controlled Delay (CoDel) Active Queue Management

**Step 1.** Delete the previous `tc` rule by issuing the command shown below.

```bash
sudo tc qdisc del dev s2-eth2 parent 1: handle 2:
```

![Figure 63. Deleting previous `tc` rule on switch S2’s s2-eth2 interface.](image)

**Step 2.** Apply `fq_codel` rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit Enter.

- **limit**: 17476
- **target**: 5ms
- **interval**: 100ms
- **flows**: 2

```bash
sudo tc qdisc add dev s2-eth2 parent 1: handle 2: fq_codel limit 17476 target 5ms interval 100ms flows 2
```

![Figure 64. Setting Fq_CoDel parameters on switch S2’s s2-eth2 interface.](image)

5.2 **Throughput test**

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

```
iperf3 -s
```

![Figure 65. Starting iPerf3 server on host h3.](image)

**Step 2.** Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```
The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the `iperf3` commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 4 and Step 6 without executing them next, in Step 7 you will press `Enter` in host h1 and host h2 to execute them.

**Step 3.** In host h1, exit from the previous folders then, create a directory called `h1_CoDel` and navigate into it using the following command:

```
cd ../.. && mkdir h1_fq_CoDel && cd h1_fq_CoDel
```

**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 60 -J > out.json
```

**Step 5.** In host h2, exit from the previous folders then, create a directory called `h2_CoDel` and navigate into it using the following command:

```
cd ../.. && mkdir h2_fq_CoDel && cd h2_fq_CoDel
```

Figure 69. Creating and navigating into directory `h2_fq_CoDel`.
**Step 6.** Type the following iPerf3 command in host h2’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 10.0.0.4 -t 60 -J > out.json
```

![Figure 70. Running iPerf3 client on host h2.](image)

**Step 7.** In the Client’s terminal, type the command below to plot the switch’s queue in real-time. When prompted for a password, type `password` and hit Enter.

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

![Figure 71. Plotting the queue occupancy on switch S2’s s2-eth2 interface.](image)

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through `s2-eth2` interface on switch S2, the queue occupancy is constantly 0.

![Figure 72. Queue occupancy on switch S2’s s2-eth2 interface.](image)
Step 8. Press Enter to execute the commands, first in host h1 terminal then, in host h2 terminal.

![Graph showing queue occupancy on switch S2's s2-eth2 interface.](image)

Figure 73. Queue occupancy on switch S2’s s2-eth2 interface.

The graph above shows that the queue occupancy peaked around 1,200,000 bytes, which is below the maximum buffer size we configure on the switch 26,441,000 bytes (10*BDP).

Step 9. In the queue plotting window, press the **key on your keyboard to stop plotting the queue.

Step 10. After the iPerf3 test finishes on host h1, enter the following command:

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

![Command output showing generation of plotting files and entering the results directory.](image)

Figure 74. Generate plotting files and entering the results directory.

Step 11. Open the throughput file using the command below on host h1.

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

![Command output showing opening the throughput.pdf file.](image)

Figure 75. Opening the throughput.pdf file.
The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 450 Mbps. It is observed that the Cubic flow uses the half part of the available bandwidth.

**Step 12.** Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

![Opening the RTT.pdf file.](image)
The graph above shows that the RTT was approximately 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not considerably exceeding the configured delay (20ms).

**Step 13.** Close the `RTT.pdf` window then, in host h2, proceed similarly by typing the following command:

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

**Step 14.** In host h2 terminal, open the throughput file using the following command:

```
xdg-open throughput.pdf
```
Figure 81. Measured throughput.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 450 Mbps. It is observed that the BBR flow uses the half part of the available bandwidth thus, it is fairly sharing the link with the other flow.

**Step 15.** Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

Figure 82. Opening the *RTT.pdf* file.

Figure 83. Measured round-trip time.
The graph above shows that the RTT was around 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not considerably exceeding the configured delay (20ms).

**Step 16.** Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `Ctrl+c`.

### 6 Changing the bandwidth to 100Mbps

This section is aimed to analyze the impact of changing the bandwidth to 100 Mbps while *Fq_Codel* is configured to work considering the previous network condition. The results will show *Fq_CoDel* does not requires a reconfiguration if the network conditions changes (i.e, latency, bandwidth, loss rate). In this section, you will change the bandwidth to 100 Mbps then, you will observe the throughput, queue occupancy and, Round-Trip Time.

**Step 1.** Apply `tbf` rate limiting rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit `Enter`.

- `rate`: 100mbit
- `burst`: 50,000
- `limit`: 26,214,400

```
sudo tc qdisc change dev s2-eth2 root: handle 1: tbf rate 100mbit burst 50000
limit 26214400
```

Figure 84. Limiting rate to 100 Mbps and keeping the buffer size to 100⋅BDP on switch S2’s interface.

Notice that the link rate is changed to 100 Mbps however, the buffer size is maintained at 26,214,400 bytes which for this case corresponds to 100⋅BDP

### 6.1 Throughput and latency tests

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

```
iperf3 -s
```
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.

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

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through `s2-eth2` interface on switch S2, the queue occupancy is constantly 0.

The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the `iperf3` commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 4 and Step 6 without executing them next, in Step 7 you will press `Enter` in host h1 and host h2 to execute them.
Step 3. In host h1, exit from the previous folders then, create a directory called *h1_100_Mbps* and navigate into it using the following command:

```
    cd ../.. && mkdir h1_100_Mbps && cd h1_100_Mbps
```

![Figure 88. Creating and navigating into directory h1_100_Mbps.](image)

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 60 -J > out.json
```

![Figure 89. Running iPerf3 client on host h1.](image)

Step 5. In host h2, exit from the previous folders then, create a directory called *h2_100_Mbps* and navigate into it using the following command:

```
    cd ../.. && mkdir h2_100_Mbps && cd h2_100_Mbps
```

![Figure 90. Creating and navigating into directory h2_100_Mbps.](image)

Step 6. Type the following iPerf3 command in host h2’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 10.0.0.4 -t 60 -J > out.json
```

![Figure 91. Running iPerf3 client on host h2.](image)

Step 7. In the Client’s terminal, type the command below to plot the switch’s queue in real-time. When prompted for a password, type `password` and hit Enter.

```
    sudo plot_q.sh s2-eth2
```
A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through s2-eth2 interface on switch S2, the queue occupancy is constantly 0.

**Step 8.** Press *Enter* to execute the commands, first in host h1 terminal then, in host h2 terminal.
The graph above shows that the queue occupancy peaked at $1 \cdot 10^6$, which is below the maximum buffer size we configure on the switch 26,214,400 bytes.

**Step 9.** In the queue plotting window, press the `s` key on your keyboard to stop plotting the queue.

**Step 10.** After the iPerf3 test finishes on host h1, enter the following command:

```bash
plot_iPerf.sh out.json && cd results
```

**Step 11.** Open the throughput file using the command below on host h1.

```bash
xdg-open throughput.pdf
```
The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 50 Mbps. It is observed that the Cubic flow uses the half part of the available bandwidth.

**Step 12.** Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

![Opening the RTT.pdf file.](image)

Figure 98. Opening the *RTT.pdf* file.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 50 Mbps. It is observed that the Cubic flow uses the half part of the available bandwidth.

**Step 12.** Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

![Opening the RTT.pdf file.](image)

Figure 98. Opening the *RTT.pdf* file.

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 50 Mbps. It is observed that the Cubic flow uses the half part of the available bandwidth.

**Step 12.** Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

![Opening the RTT.pdf file.](image)

Figure 98. Opening the *RTT.pdf* file.
The graph above shows that the RTT was around 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not considerably exceeding the configured delay (20ms).

**Step 13.** Close the *RTT.pdf* window then, in host h2, proceed similarly by typing the following command:

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

![Figure 100. Generate plotting files and entering the results directory.](image)

**Step 14.** In host h2 terminal, open the throughput file using the following command:

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

![Figure 101. Opening the throughput.pdf file.](image)

![Figure 102. Measured throughput.](image)

The figure above shows the iPerf3 test output report for the last 60 seconds. The average achieved throughput is around 50 Mbps as well. It is observed that the BBR flow uses the half part of the available bandwidth thus, it is fairly sharing the link with the other flow. One of the most remarkable features of Fq_CoDel is that it works well even if the network condition changes.
Step 15. Close the *throughput.pdf* window then open the Round-Trip Time (RTT) file using the command below.

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

![Opening the RTT.pdf file.](image)

Figure 103. Opening the *RTT.pdf* file.

The graph above shows that the RTT was around 25,000 microseconds (25ms). The output shows that there was not bufferbloat as the average latency is not considerably exceeding the configured delay (20ms).

Step 16. Close the *throughput.pdf* window then, to stop *iperf3* server in host h3 and host h4 press `ctrl+c`.

This concludes Lab 18. Stop the emulation and then exit out of MiniEdit.

References

4. K. Nichols, V. Jacobson, A. McGregor, and A. Iyengar, “Controlled delay active queue management,” Internet engineering task force, Request for comments 8289, 2018
NETWORK TOOLS AND PROTOCOLS

Lab 19: Proportional Integral Controller-Enhanced (PIE)

Document Version: 11-23-2019

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

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

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

Objectives

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

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

Lab settings

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

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
4. Section 4: Configuring PIE on switch S2.

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

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

1.1 The PIE algorithm

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

![Figure 1. Buffer managed by PIE AQM.](image)

The basic `pie` syntax used with `tc` is as follows:
In this lab, we will use the PIE AQM algorithm to control the queue size at the egress port of a router.

2 Lab topology

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

**Step 2.** On MiniEdit’s menu bar, click on *File* then *Open* to load the lab’s topology. Locate the *Lab 19.mn* topology file and click on *Open*. 
Step 3. Before starting the measurements between end-hosts, the network must be started. Click on the Run button located at the bottom left of MiniEdit’s window to start the emulation.

![Figure 5. Running the emulation.](image)

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

2.1 Starting host h1, host h2, and host h3

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

![Figure 6. Opening a terminal on host h1.](image)

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.
2.1 Emulating high-latency WAN

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

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

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

**Step 2.** In the terminal, type the command below. When prompted for a password, type `password` and hit `Enter`. This command introduces 20ms delay to switch S1’s s1-eth1 interface.

```
sudo tc qdisc add dev s1-eth1 root netem delay 20ms
```

Figure 7. Connectivity test using `ping` command.

Figure 8. Shortcut to open a Linux terminal.

Figure 9. Adding delay of 20ms to switch S1’s s1-eth1 interface.
2.4 Testing connection

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

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

![Figure 10. Output of `ping 10.0.0.3` command.](image)

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`.

![Figure 11. Output of `ping 10.0.0.3` command.](image)

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

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

3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size

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

BW = 1,000,000,000 bits/second

RTT = 0.02 seconds

BDP = 1,000,000,000 · 0.02 = 20,000,000 bits
  = 2,500,000 bytes ≈ 2.5 Mbytes

1 Mbyte = 1024² bytes

BDP = 2.5 Mbytes = 2.5 · 1024² bytes = 2,621,440 bytes

The default buffer size in Linux is 16 Mbytes, and only 8 Mbytes (half of the maximum buffer size) can be allocated. Since 8 Mbytes is greater than 2.5 Mbytes, then no need to tune the buffer sizes on end-hosts. However, in upcoming tests, we configure the buffer size on the switch to 10-BDP. In addition, to ensure that the bottleneck is not the hosts’ TCP buffers, we configure the buffers to 20-BDP (52,428,800).

Step 1. Now, we have calculated the maximum value of the TCP sending and receiving buffer size. In order to change the receiving buffer size, on host h1’s terminal type the command shown below. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum). The maximum value is doubled (2 · 10-BDP) as Linux only allocates half of the assigned value.

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

![sysctl](image.png)

Figure 12. Receive window change in `sysctl`. 

The returned values are measured in bytes. 10,240 represents the minimum buffer size that is used by each TCP socket. 87,380 is the default buffer which is allocated when applications create a TCP socket. 52,428,800 is the maximum receive buffer that can be allocated for a TCP socket.

**Step 2.** To change the current send-window size value(s), use the following command on host h1’s terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum). The maximum value is doubled as Linux allocates only half of the assigned value.

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

![Figure 13. Send window change in `sysctl`.

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

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

![Figure 14. Receive window change in `sysctl`.

**Step 4.** To change the current send-window size value(s), use the following command on host h1’s terminal. The values set are: 10,240 (minimum), 87,380 (default), and 52,428,800 (maximum). The maximum value is doubled as Linux allocates only half of the assigned value.

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

![Figure 15. Send window change in `sysctl`.

The returned values are measured in bytes. 10,240 represents the minimum buffer size that is used by each TCP socket. 87,380 is the default buffer which is allocated when applications create a TCP socket. 52,428,800 is the maximum receive buffer that can be allocated for a TCP socket.
3.2 Setting switch S2’s buffer size to $10 \cdot BDP$

**Step 1.** Apply `tbf` rate limiting rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit *Enter*.

- **rate**: 1gbit
- **burst**: 500,000
- **limit**: 26,214,400

```bash
sudo tc qdisc add dev s2-eth2 root handle 1: tbf rate 1gbit burst 500000 limit 26214400
```

![Figure 16](image1.png)

Figure 16. Limiting rate to 1 Gbps and setting the buffer size to $10 \cdot BDP$ on switch S2’s interface.

3.3 Throughput and latency tests

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

```bash
iperf3 -s
```

![Figure 17](image2.png)

Figure 17. Starting iPerf3 server on host h3.

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

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

![Figure 18](image3.png)

Figure 18. Plotting the queue occupancy on switch S2’s s2-eth2 interface.
A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through \textit{s2-eth2} interface on switch S2, the queue occupancy is constantly 0.

![Queue occupancy plot](image)

**Figure 19.** Queue occupancy on switch S2’s \textit{s2-eth2} interface.

**Step 3.** In host \textit{h1}, create a directory called \textit{Drop\_Tail} and navigate into it using the following command:

```bash
mkdir Drop\_Tail && cd Drop\_Tail
```

![Directory creation](image)

**Figure 20.** Creating and navigating into directory \textit{Drop\_Tail}.

**Step 4.** Type the following \texttt{iPerf3} command in host \textit{h1}’s terminal without executing it. The \texttt{-J} option is used to display the output in JSON format. The redirection operator \texttt{>} is used to store the JSON output into a file.

```bash
iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

![iPerf3 command](image)

**Figure 21.** Running \texttt{iPerf3} client on host \textit{h1}.

**Step 5.** Type the following \texttt{ping} command in host \textit{h2}’s terminal without executing it.

```bash
ping 10.0.0.3 -c 90
```
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.

The graph above shows that the queue occupancy peaked at $2.5 \cdot 10^7$, which is the maximum buffer size we configure on the switch.

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

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

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

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

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

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

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

Figure 26. Measured throughput.

Figure 27. Opening the *RTT.pdf* file.
The graph above shows that the RTT was approximately 200,000 microseconds (200ms). The output shows that there is bufferbloat as the average latency is at least ten times greater than the configured delay (20ms).

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

The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 20.083, 192.823, 228.407, and 26.954 milliseconds, respectively. The
output also verifies that there is bufferbloat as the average latency (192.823) is significantly greater than the configured delay (20ms).

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

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

![Sent Cwnd over time](image)

Figure 30. Congestion window evolution.

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

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

4 Configuring PIE on switch S2

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

4.1 Setting PIE parameter on switch S2’s egress interface

**Step 1.** Apply `pie` rate limiting rule on switch S2’s s2-eth2 interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit `Enter`. 
Lab 19: Proportional Integral Controller-Enhanced (PIE)

- **Limit**: 17476
- **Target**: 22ms
- **TUpdate**: 5ms
- **Alpha**: 2
- **Beta**: 25

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

![Image](image1.png)

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

### 4.2 Throughput and latency tests

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

```bash
iperf3 -s
```

![Image](image2.png)

Figure 32. Starting iPerf3 server on host h3.

**Step 2.** In the Client’s terminal, type the command below to plot the switch’s queue in real-time. When prompted for a password, type **password** and hit Enter.

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

![Image](image3.png)

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

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through s2-eth2 interface on switch S2, the queue occupancy is constantly 0.
Step 3. Exit from Drop_Tail/results directory, then create a directory PIE and navigate into it using the following command.

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

![Figure 35. Creating and navigating into directory RED.](image)

Step 4. Type the following iPerf3 command in host h1’s terminal without executing it. The `-J` option is used to display the output in JSON format. The redirection operator `>` is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

![Figure 36. Running iPerf3 client on host h1.](image)

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

```
ping 10.0.0.3 -c 90
```
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.

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

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

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

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

Figure 39. Generate plotting files and entering the results directory.

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

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

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

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

![Figure 40. Opening the *throughput.pdf* file.](image)

![Figure 40. Measured throughput.](image)

![Figure 41. Opening the *RTT.pdf* file.](image)
The graph above shows that the RTT was contained between 30ms and 40ms which is not significantly greater than the configured delay (20ms) thus, there is not bufferbloat. Since the AQM algorithm configured on the switch is applying a dropping policy to prevent unnecessary delays.

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

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

--- 10.0.0.3 ping statistics ---
90 packets transmitted, 90 received, 0% packet loss, time 200ms
rtt min/avg/max/mdev = 28.878/64.165/217.367/79.417 ms
```

Figure 43. ping test result.

The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip
Time (RTT) were 20.078, 64.165, 217.367, and 70.417 milliseconds, respectively. The output also verifies that there is not bufferbloat as the average latency (34.048) is not significantly greater than the configured delay (20ms).

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

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

![Figure 44. Opening the cwnd.pdf file.](image)

![Figure 45. Evolution of the congestion window.](image)

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

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

### 4.3 Changing the bandwidth to 100Mbps

This section is aimed to analyze the impact of changing the bandwidth to 100 Mbps while RED is tuned to work with the previous network condition. The results will show that RED requires a reconfiguration if the network conditions changes (i.e., latency, bandwidth, loss rate). First, you will change the bandwidth to 100 Mbps then, you will observe the queue occupancy, RTT and congestion window in order to evaluate the performance of RED when the network condition changes.
**Step 1.** Apply rate limiting rule on switch $S_2$’s $s2$-$eth2$ interface. In the client’s terminal, type the command below. When prompted for a password, type `password` and hit Enter.

- **rate**: 100mbit
- **burst**: 50,000
- **limit**: 26,214,400

```
sudo tc qdisc change dev s2-eth2 root handle 1: tbf rate 100mbit burst 50000 limit 26214400
```

Figure 46. Limiting rate to 100 Mbps and keeping the buffer size to 10*BDP on switch $S_2$’s interface.

### 4.4 Throughput and latency tests

**Step 1.** Launch iPerf3 in server mode on host $h3$’s terminal.

```
iperf3 -s
```

Figure 47. Starting iPerf3 server on host $h3$.

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

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

Figure 48. Plotting the queue occupancy on switch $S_2$’s $s2$-$eth2$ interface.

A new window opens that plots the queue occupancy as shown in the figure below. Since there are no active flows passing through $s2$-$eth2$ interface on switch $S_2$, the queue occupancy is constantly 0.
Step 3. Exit from PIE/results directory, then create a directory PIE_100Mbps and navigate into it using the following command.

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

![Figure 50. Creating and navigating into directory PIE_100Mbps.](image)

Step 4. Type the following iPerf3 command in host h1’s terminal without executing it. The \(-J\) option is used to display the output in JSON format. The redirection operator \(>\) is used to store the JSON output into a file.

```
iperf3 -c 10.0.0.3 -t 90 -J > out.json
```

![Figure 51. Running iPerf3 client on host h1.](image)

Step 5. Type the following \(\text{ping}\) command in host h2’s terminal without executing it.

```
ping 10.0.0.3 -c 90
```

![Figure 52. Typing \(\text{ping}\) command on host h2.](image)
**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.

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

**Step 7.** In the queue plotting window, press the `Esc` 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
```

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

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

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

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

Figure 56. Opening the *RTT.pdf* file.
The graph above shows that the RTT peaked at 10 seconds up to 140ms then, the latency went down and stabilize around 40ms which is closer to the default latency (20ms). This output shows that there is not bufferbloat as the average latency is not significantly greater as in the previous section. Since PIE is configured to operate on a 1 Gbps link, for this test the point of operation changed therefore, this change does not affect the queue management policy.

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

The result above indicates that all 90 packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip
Time (RTT) were 22.073, 37.541, 139.705 and 16.828 milliseconds, respectively. The output also verifies that there is a bufferbloat problem as the average latency (186.175) is significantly greater than the configured delay (20ms).

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

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

![Figure 59. Opening the cwnd.pdf file.](image)

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

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

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

**References**

Lab 20: Classifying TCP traffic using Hierarchical Token Bucket (HTB)

Document Version: 11-28-2019
Lab 20: Hierarchical Token Bucket

Contents

Overview .......................................................................................................................... 3
Objectives ....................................................................................................................... 3
Lab settings ..................................................................................................................... 3
Lab roadmap ..................................................................................................................... 3
1 Introduction .................................................................................................................. 3
2 Lab topology ................................................................................................................ 6
   2.1 Starting host h1, host h2, host h3 and host h4 ....................................................... 7
   2.2 Emulating high-latency WAN ............................................................................. 8
   2.3 Testing connection ............................................................................................... 9
3 Emulating a high latency wide area network (WAN) ..................................................... 10
   3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size ................................. 10
   3.2 Modifying hosts’ buffer size ............................................................................... 11
   3.3 Setting switch S1’s buffer size to BDP ................................................................. 14
   3.4 Throughput tests of two TCP competing flows ................................................... 14
4 Setting HTB at switch s2 egress interface .................................................................... 17
   4.1 Defining classes .................................................................................................. 17
   4.2 Defining filters .................................................................................................... 18
   4.3 Throughput tests of two TCP competing flows using HTB ................................ 19
References ...................................................................................................................... 22
Overview

This lab is aimed to introduce the reader to Hierarchical Token Bucket (HTB). This queueing discipline controls the use of the outbound bandwidth on a given link by classifying different kinds of traffic into several slower links. Throughput tests are conducted to evaluate the impact of dividing a physical link according to a given policy.

Objectives

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

1. Understand the concept of link-sharing.
2. Define classes to allocate a maximum bandwidth to a TCP flow.
3. Associate a class to a specific flow using filters.
4. Evaluate the effects of HTB when two TCP are using the same link.

Lab settings

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

<table>
<thead>
<tr>
<th>Device</th>
<th>Account</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>admin</td>
<td>password</td>
</tr>
</tbody>
</table>

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Emulating a high latency wide area network (WAN).
4. Section 4: Setting HTB at switch s2 egress interface.

1 Introduction

On a network, the management of upstream link requires the implementation of a link-sharing mechanism. Without such mechanism the default behavior of the gateways routers does not necessarily lead to a fair Internet bandwidth sharing among the endpoints. The Internet is mostly based on TCP/IP which provides few features that allow network administrators to implement traffic control rules, TCP/IP does not
know the link capacity between two hosts so that, TCP protocols compete sending packets faster, when packets start getting lost it will slow down.

In summary, network resource management involves two types of services: services for real-time traffic and link-sharing services\(^1\). However, in a congested network, resource management mechanism is required at the gateway router. The main functions of the link-sharing mechanism are:

- Enable routers to control the distribution of traffic on local links according to the local demand therefore, each organization has a guaranteed bandwidth.
- Enable gateways to redistribute available bandwidth among organizations.
- Specify the bandwidth according the type of traffic.
- Accommodate the available bandwidth as new services are added.

All these requirements lead to the design of a hierarchical link-sharing structure. Figure 1 depicts the relationship between classes filters and queuing disciplines. In this structure, the traffic is classified according to a class. Classes and filters are tied together to the queuing discipline. A queuing discipline can be associated with several classes. Every class must have a queuing discipline associated it. Filters are used by the queuing discipline to assign incoming packets to one of its classes. Different types of filters can be employed, for example route classifiers and u32 filters. These filters usually classify the traffic based on the source IP, destination IP, source port, destination port, TOS byte and protocol. The universal 32bit (u32) filter allows to match arbitrary bitfields in the packet.

1.1 HTB algorithm

HTB controls the use of the outbound bandwidth on a given link by simulate several slower links. The user specifies how to divide the physical link into simulated links and how to decide which simulated link to use for a given packet to be sent. HTB shapes traffic based on the Token Bucket Filter (TBF) algorithm, which does not depend on interface characteristics and so does not need to know the underlying bandwidth of the outgoing interface.
Figure 2 illustrates a basic structure of HTB. The classes are configured as a tree according to relationships of traffic aggregations. Only leaf classes have queue to buffer the packets belong to the class. Children classes borrow bandwidth from their parents when their packet flow exceeds the configured rate. A child will continue to attempt to borrow bandwidth until it reaches ceil, which is the maximum bandwidth available for that class. Under each class, the user can specify other queueing disciplines namely Token Bucket Filter (TBF), Stochastic Fair Queueing (SFQ), Controlled Delay (CoDel), etc. It will depend on the service that such class is intended to provide. The default queueing discipline is First-in, First-out (FIFO).

![Hierarchical Token Bucket Structure](image)

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

```
tc qdisc [add | ...] dev [dev_id] root handle 1: htb default [DEFAULT-ID]
```

- `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.
- `htb`: enables Hierarchical Token Bucket queuing discipline.
- `default`: unclassified traffic will be enqueued under this class.

```
tc class [add | ...] dev [dev_id]
```

- `tc`: Linux traffic control tool.
- `class`: defines a class. Classes have a host of parameters to configure their operation.
- `[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.
- `rate`: specifies the maximum rate this class and all its children are guaranteed. This parameter is mandatory.
- **ceil**: determines the maximum rate at which a class can send, if its parent has available bandwidth. The default configuration is set to the configured rate, which implies no borrowing.
- **burst**: denotes the number of bytes that can be burst at ceil speed, in excess of the configured rate. It should be at least as high as the highest burst of all children.

In this lab, we will use the htb AQM algorithm to control the queue size at the egress port of a router.

## 2 Lab topology

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

![Lab topology figure](image)

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**.

![MiniEdit shortcut](image)

**Step 2.** On MiniEdit’s menu bar, click on File then Open to load the lab’s topology. Locate the Lab 20.mn topology file and click on Open.
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.

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 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 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.

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

**Step 2.** In the terminal, type the command below. When prompted for a password, type `password` and hit *Enter*. This command introduces 20ms delay to switch S1’s `s1-eth1` interface.

```
sudo tc qdisc add dev s1-eth1 root handle 1: netem delay 20ms
```

![Figure 10. Adding delay of 20ms to switch S1’s s1-eth1 interface.](image)

### 2.3 Testing connection

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

**Step 1.** On the terminal of host h1, type `ping 10.0.0.3`. 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.

![Figure 11. Output of ping 10.0.0.3 command.](image)
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.

![Figure 12. Output of `ping 10.0.0.3` command.](image)

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 Emulating a high latency wide area network (WAN)

In this section, you are going to tune the network devices in order to emulate a Wide Area Network (WAN). First, you will set the hosts’ TCP buffers to $8 \cdot \text{BDP}$ therefore, the bottleneck is not in the end-hosts. Then, you will set the bottleneck bandwidth to 1Gbps in switch S1’s $s1$-eth1 interface. Finally, you will conduct throughput tests between two competing TCP flows when Token Bucket Filter (TBF) is configured in switch S1 to limit the bandwidth.

#### 3.1 Bandwidth-delay product (BDP) and hosts’ TCP buffer size

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

$$
\begin{align*}
\text{BW} &= 1,000,000,000 \text{ bits/second} \\
\text{RTT} &= 0.02 \text{ seconds} \\
\text{BDP} &= 1,000,000,000 \cdot 0.02 = 20,000,000 \text{ bits} \\
&= 2,500,000 \text{ bytes} \approx 2.5 \text{ Mbytes}
\end{align*}
$$
1 Mbyte = $1024^2$ bytes

BDP = 2.5 Mbytes = $2.5 \times 1024^2$ bytes = 2,621,440 bytes

The default buffer size in Linux is 16 Mbytes, and only 8 Mbytes (half of the maximum buffer size) can be allocated. Since 8 Mbytes is greater than 2.5 Mbytes, then no need to tune the buffer sizes on end-hosts. However, in upcoming tests, we configure the buffer size on the switch to BDP. In addition, to ensure that the bottleneck is not the hosts’ TCP buffers, we configure the buffers to 8·BDP (20,971,520).

### 3.2 Modifying hosts’ buffer size

For the following calculation, the bottleneck bandwidth is considered as 1 Gbps, and the round-trip time latency as 20ms.

In order to have enough TCP buffer size, we will set the TCP sending and receiving buffer to 8·BDP in all hosts.

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

\[
\text{RTT} = 0.02 \text{ seconds}
\]

BDP = \(1,000,000,000 \times 0.02 = 20,000,000 \text{ bits}\)

\[
= 2,500,000 \text{ bytes} \approx 2.5 \text{ Mbytes}
\]

The send and receive TCP buffer sizes should be set to 8·BDP to ensure the bottleneck is not in the end-hosts. For simplicity, we will use 2.5 Mbytes as the value for the BDP instead of 2,500,000 bytes.

1 Mbyte = $1024^2$ bytes

BDP = 2.5 Mbytes = $2.5 \times 1024^2$ bytes = 2,621,440 bytes

8·BDP = \(8 \times 2,621,440 \text{ bytes} = 20,971,520 \text{ bytes}\)

**Step 1.** At this point, 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 20,971,520 (maximum).

\[\text{sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'}\]
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. 20,971,520 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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

**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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

**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 20,971,520 (maximum).

```bash
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

Next, the same commands must be configured on host h2, host h3, and host h4.
**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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

Figure 17. 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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

Figure 18. 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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
```

Figure 19. 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 20,971,520 (maximum).

```
sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
```

Figure 20. Send window change in `sysctl`.
3.3 Setting switch S1’s buffer size to BDP

In this section, you are going to set switch S1’s buffer size to BDP and emulate a 1 Gbps Wide Area Network (WAN) using the Token Bucket Filter. Then, you will set the TCP sending and receiving windows in all hosts. Finally, you will conduct a throughput test with two TCP competing flows.

**Step 1.** Apply rate limiting rule on switch S2’s s1-eth1 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 add dev s1-eth1 parent 1: handle 2: tbf rate 1gbit burst 500000 limit 2621440
```

![Figure 21. Limiting rate to 1 Gbps and setting the buffer size to BDP on switch S1’s interface.](image)

3.4 Throughput tests of two TCP competing flows

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

```
iperf3 -s
```

![Figure 22. Starting iPerf3 server on host h3.](image)

**Step 2.** Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```
The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the `iperf3` commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 3 and Step 4 without executing them next, in Step 5 you will press Enter in host h1 and host h2 to execute them.

**Step 3.** Type the following iPerf3 command in host h1’s terminal without executing it.

```
iperf3 -c 10.0.0.3 -t 60
```

**Step 4.** Type the following iPerf3 command in host h2’s terminal without executing it.

```
iperf3 -c 10.0.0.4 -t 60
```

**Step 5.** Press Enter to execute the commands shown in step 4 and step 6, first in host h1 terminal then, in host h3 terminal.

**Step 6.** Wait until the test finishes then, click on host h1 terminal to visualize the results. You will notice that host h1 uses approximately the half part of the link (~500Mbps).
Figure 25. Throughput report on host h1.

**Step 7.** To visualize the results in the other sender, click on host h2 terminal. You will notice that host h2 uses approximately the half part of the link (~500Mbps).

Figure 26. Throughput report on host h2.

**Step 8.** To stop *iperf3* servers in host h3 and host h4 press `Ctrl+c`. 
### 4 Setting HTB at switch s2 egress interface

In this section you will enable in switch S2’s s2-eth2 interface Hierarchical Token Bucket (HTB). First **htb** is defined as the root queueing discipline. Secondly, two classes are defined. These classes specify the bandwidth allocation for two TCP flows. Then, you will use filters to associate specific flows to the previously defined classes. Lastly, a throughput test is conducted to observe how HTB classifies TCP traffic. HTB ensures that the amount of service provided to each class is at least the minimum of the amount it requests, and the amount assigned to it.

**Step 1:** In order to enable **htb** in switch S2 egress interface, type the following command:

```bash
sudo tc qdisc add dev s2-eth2 root handle 1: htb
```

![Figure 26. Setting htb in switch S2’s s2-eth2 interface.](image)

#### 4.1 Defining classes

In this section, first you will define a root class which specifies **htb** as its parent. A root class, like other classes under an **htb** qdisc allows its children to borrow from each other, but one root class cannot borrow from another. Then, you will create two classes that will allocate 700Mbps and 300Mbps of bandwidth respectively. These classes borrow from the root the bandwidth they need.

**Step 1:** To define the root class type the following command in the Client’s terminal.

- **rate** 1gbit
- **ceil** 1gbit

```bash
sudo tc class add dev s2-eth2 parent 1:0 classid 1:1 htb rate 1gbit ceil 1gbit
```

![Figure 27. Defining a root class.](image)
Step 2: Define the following class by issuing the command shown below:

- **rate**: 700mbit
- **ceil**: 1gbit

```bash
sudo tc class add dev s2-eth2 parent 1:1 classid 1:10 htb rate 700mbit ceil 1gbit
```

![Figure 28. Defining a class rate and ceil values.](image)

Step 3: Define the next class by issuing the following command:

- **rate**: 300mbit
- **ceil**: 1gbit

```bash
sudo tc class add dev s2-eth2 parent 1:1 classid 1:20 htb rate 300mbit ceil 1gbit
```

![Figure 29. Defining a class rate and ceil values.](image)

### 4.2 Defining filters

In this section, you will specify the filters. The filters determine which class belong to each packet. In this case we use the source IP to match the flows. Note that the IP address of host h1 is 10.0.0.1 and the IP address of host h2 is 10.0.0.2.

**Step 1:** To define filter related to the first class that was defined in the previous section, in the Client’s terminal type the following command:

```bash
sudo tc filter add dev s2-eth2 protocol ip parent 1:0 prio 1 u32 match ip src 10.0.0.1 flowid 1:10
```
Step 2 To define filter for the first class type the following command in the Client’s terminal.

```
sudo tc filter add dev s2-eth2 protocol ip parent 1:0 prio 1 u32 match ip src 10.0.0.2 flowid 1:10
```

4.3  Throughput tests of two TCP competing flows using HTB

In this section, you will conduct a throughput test to verify the previous configuration.

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

```
iperf3 -s
```

Step 2. Launch iPerf3 in server mode on host h4’s terminal.

```
iperf3 -s
```
The following steps are aimed to replicate the case when two TCP flows are competing sharing the same link therefore, the \textit{iperf3} commands in host h1 and host h2 should be executed almost simultaneously. Hence, you will type the commands presented in Step 3 and Step 4 without executing them next, in Step 5 you will press \textit{Enter} in host h1 and host h2 to execute them.

**Step 3.** Type the following iPerf3 command in host h1’s terminal without executing it.

```
iperf3 -c 10.0.0.3 -t 60
```

![Figure 34. Running iPerf3 client on host h1.](image)

**Step 4.** Type the following iPerf3 command in host h2’s terminal without executing it.

```
iperf3 -c 10.0.0.4 -t 60
```

![Figure 35. Running iPerf3 client on host h2.](image)

**Step 5.** Press \textit{Enter} to execute the commands shown in step 4 and step 6, first in host h1 terminal then, in host h3 terminal.

**Step 6.** Wait until the test finishes then, click on host h1 terminal to visualize the results. You will notice that host h1 uses the bandwidth that was specified by the rate in the first class, which is approximately 700Mbps.
Step 7. Click on host h2 terminal to visualize the results. Notice that host h2 uses the bandwidth that was specified by the rate in the second class, which is around 300Mbps.

Step 8. To stop iperf3 servers in host h3 and host h4 press `Ctrl+c`.

This concludes Lab 20. Stop the emulation and then exit out of MiniEdit.
References


