NETWORK TOOLS AND PROTOCOLS

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

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

Device	Account	Password
Client1	admin	password

Lab roadmap

This lab is organized as follows:

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

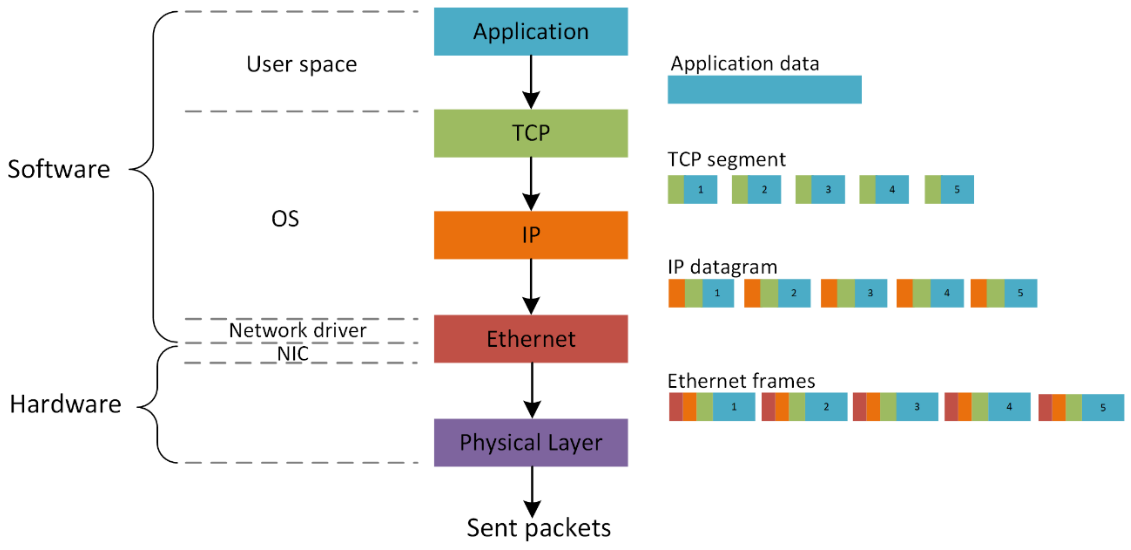
1 Introduction

1.1 Introduction to hardware offloading

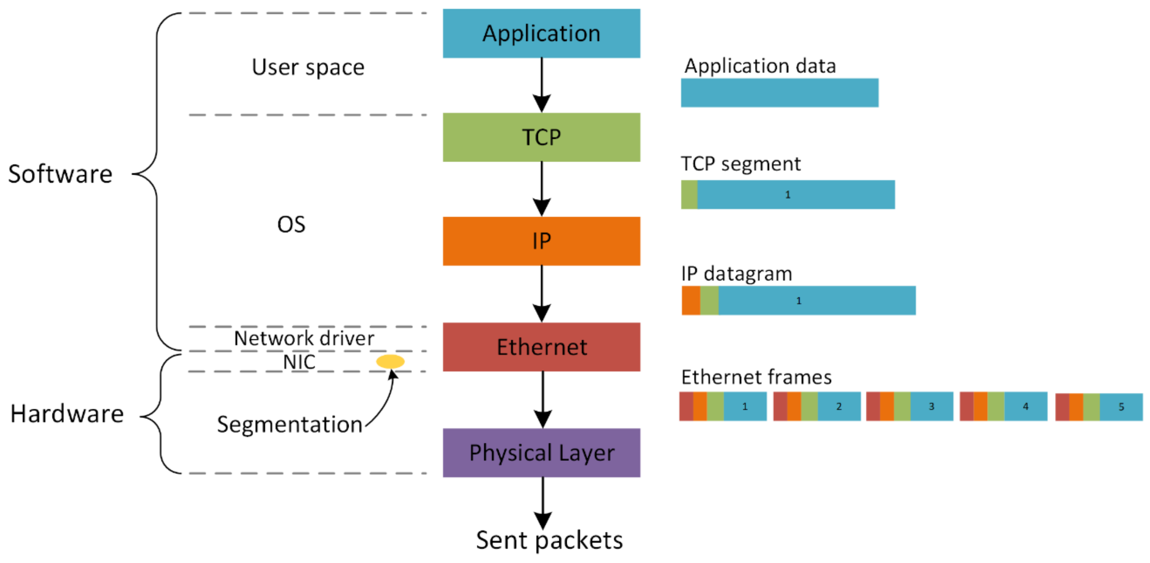
Consider Figure 1. It illustrates two scenarios. Figure 1a presents a situation where hardware offloading is disabled in the sender host. Therefore, the application data is segmented in the transport layer, in this example in five segments then, these segments are individually encapsulated by the corresponding 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].



(a)



(b)

Figure 1. Impact of TCP hardware offloading. (a) Hardware offloading disabled. (b) Hardware offloading enabled.

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.

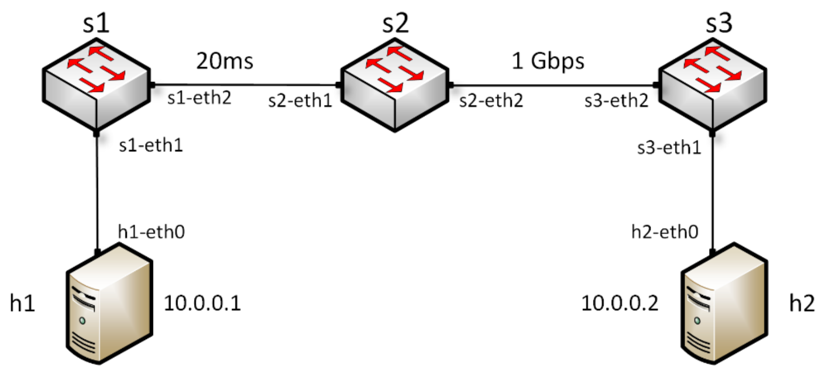


Figure 2. Lab topology.

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

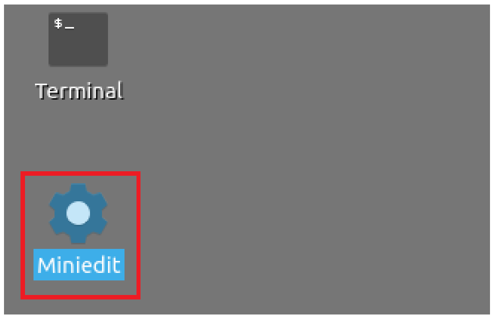


Figure 3. MiniEdit shortcut.

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

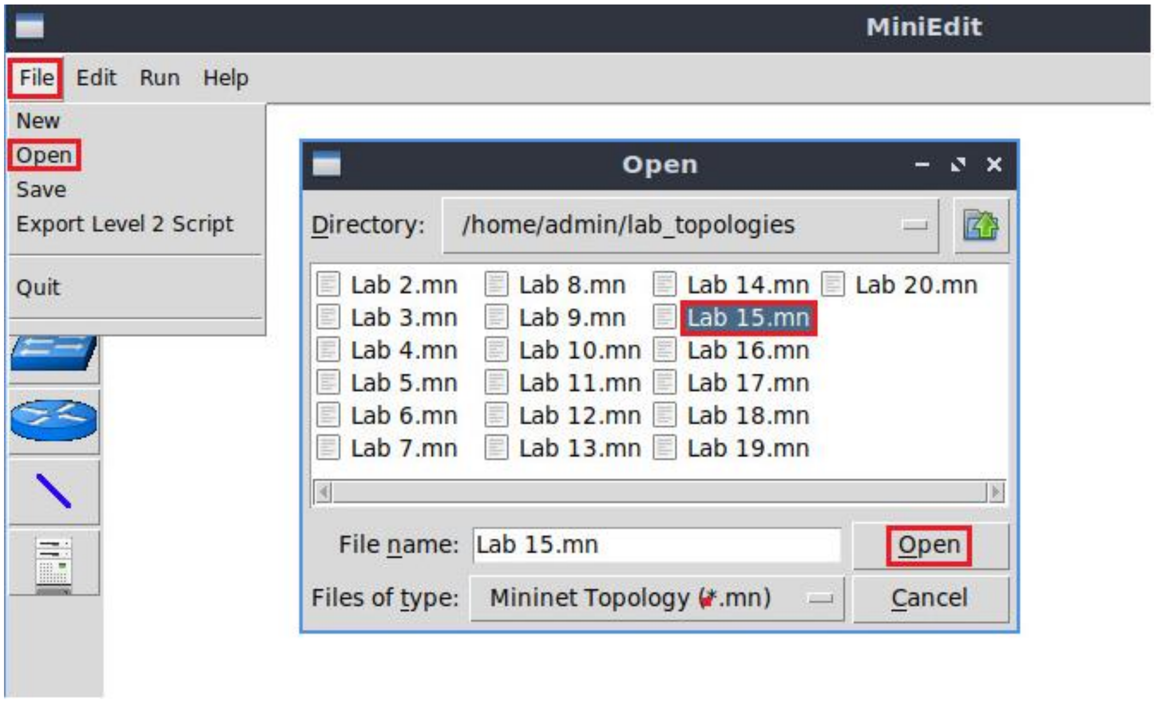


Figure 4. MiniEdit's *Open* dialog.

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

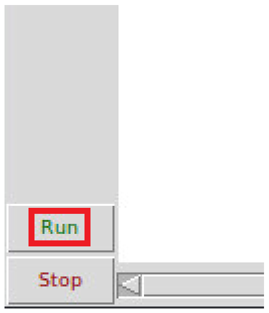


Figure 5. Running the emulation.

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

2.1 Starting host h1 and host h2

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

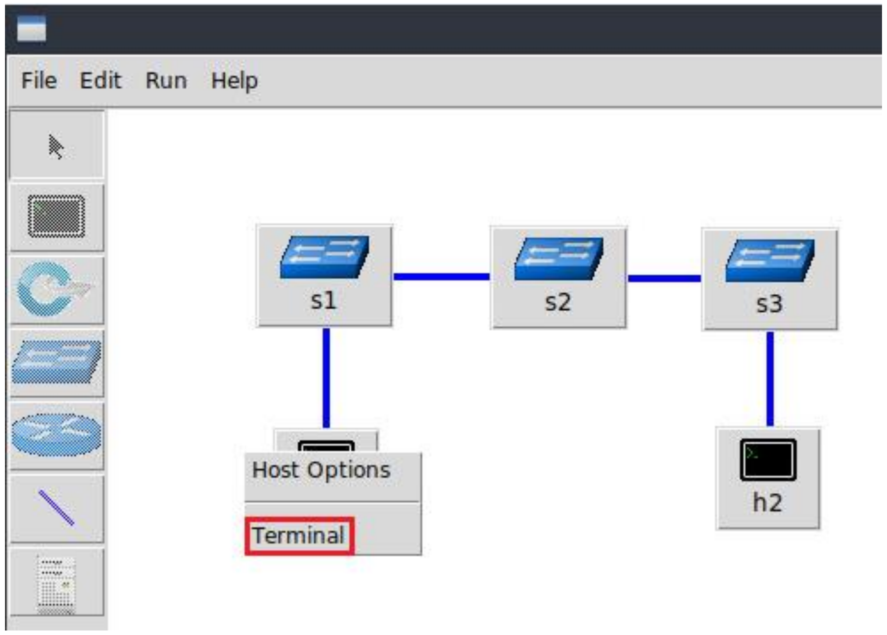
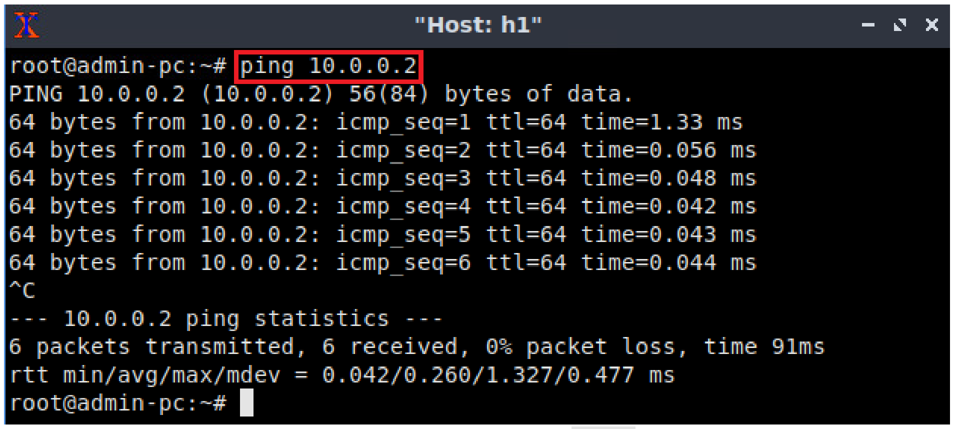


Figure 6. Opening a terminal on host h1.

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

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



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

Figure 7. Connectivity test using `ping` command.

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.

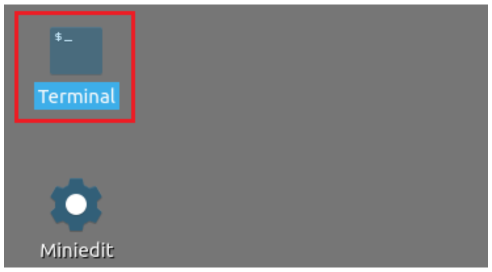
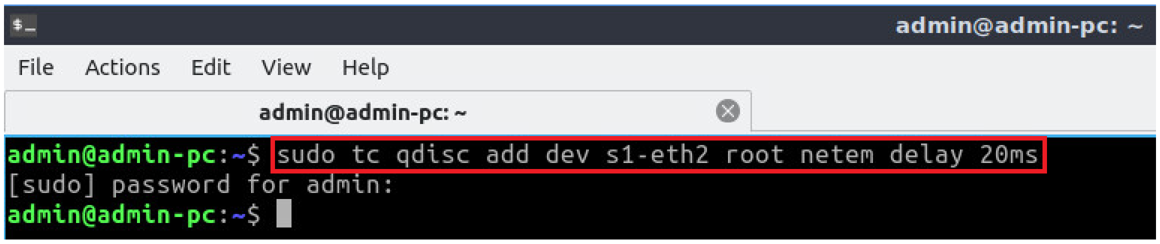


Figure 8. Shortcut to open a Linux terminal.

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

Step 2. In the terminal, type the command below. When prompted for a password, type `password` and hit *Enter*. This command introduces 20ms delay to switch S1's *s1-eth2* interface.

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



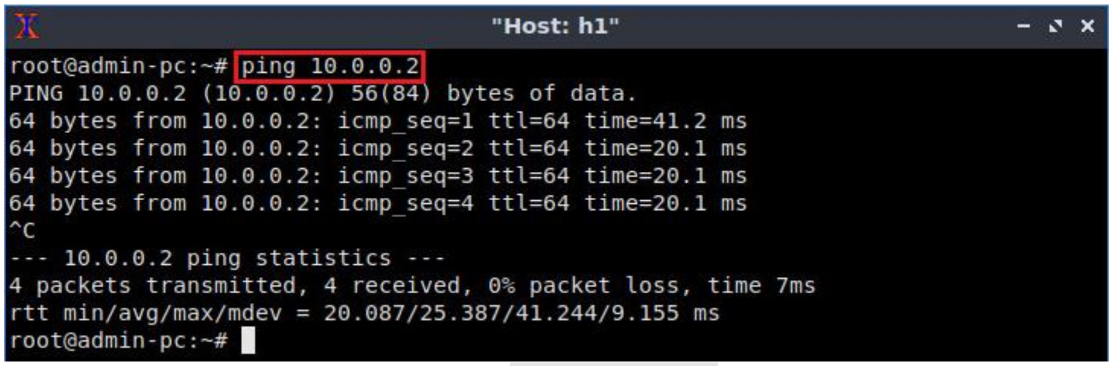
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 9. Adding delay of 20ms to switch S1's *s1-eth2* interface.

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.



```
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=41.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.087/25.387/41.244/9.155 ms
root@admin-pc:~#
```

Figure 10. 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 20.087, 25.387, 41.244, and 9.155 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

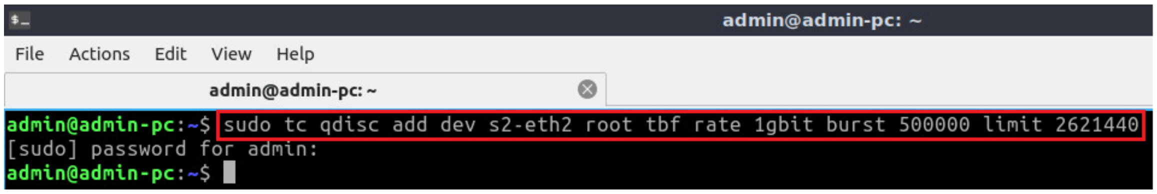
3.3 Limiting the rate on switch S2 egress interface

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
```



```
admin@admin-pc:~$ sudo tc qdisc add dev s2-eth2 root tbf rate 1gbit burst 500000 limit 2621440
[sudo] password for admin:
admin@admin-pc:~$ 
```

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 · 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 · 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.

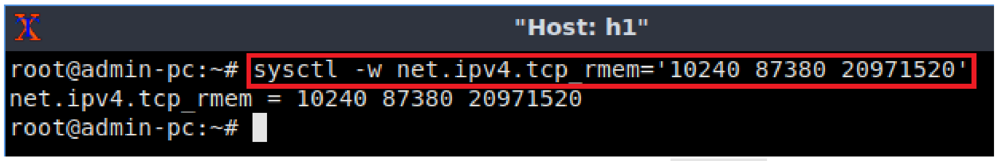
$$1 \text{ Mbyte} = 1024^2 \text{ bytes}$$

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

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

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'
```



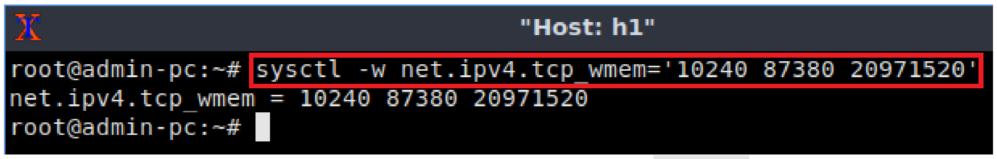
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
net.ipv4.tcp_rmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 12. Receive window change in `sysctl`.

The returned values are measured in bytes. 10,240 represents the minimum buffer size that is used by each TCP socket. 87,380 is the default buffer which is allocated when applications create a TCP socket. 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'
```



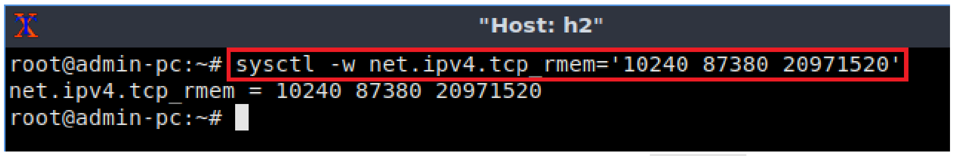
```
"Host: h1"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
net.ipv4.tcp_wmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 13. Send window change in `sysctl`.

Next, the same commands must be configured on host h2, host h3, and host h4.

Step 3. To change the current receiver-window size value(s), use the following command on host h2's terminal. The values set are: 10,240 (minimum), 87,380 (default), and 20,971,520 (maximum).

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

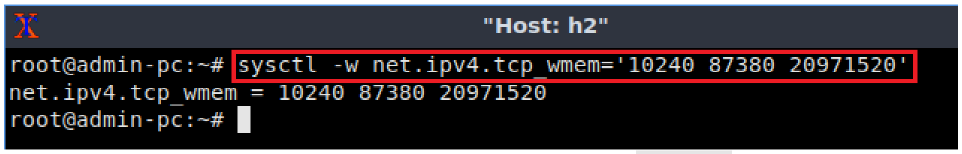


```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_rmem='10240 87380 20971520'
net.ipv4.tcp_rmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 14. Receive window change in `sysctl`.

Step 4. To change the current send-window size value(s), use the following command on host 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'
```



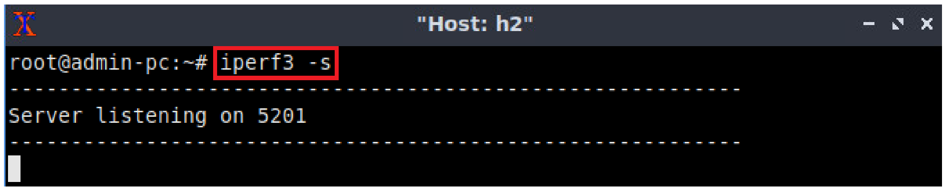
```
"Host: h2"
root@admin-pc:~# sysctl -w net.ipv4.tcp_wmem='10240 87380 20971520'
net.ipv4.tcp_wmem = 10240 87380 20971520
root@admin-pc:~#
```

Figure 15. Send window change in `sysctl`.

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
```

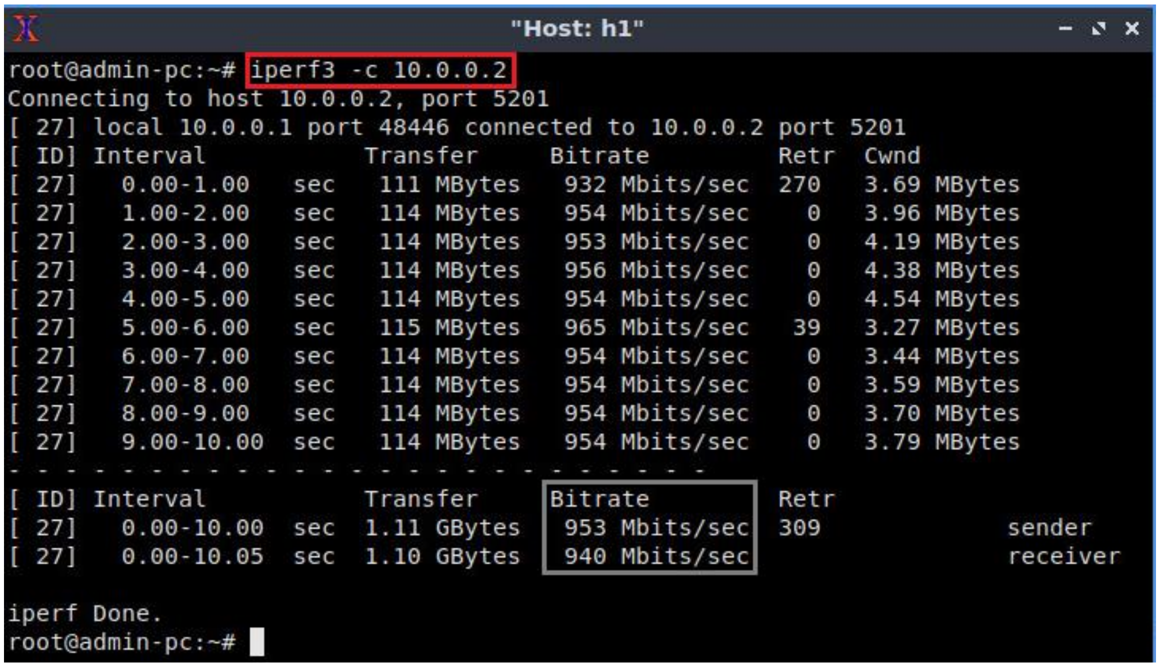


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 16. 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
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 27] local 10.0.0.1 port 48446 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 27]   0.00-1.00   sec   111 MBytes   932 Mbits/sec  270   3.69 MBytes
[ 27]   1.00-2.00   sec   114 MBytes   954 Mbits/sec    0   3.96 MBytes
[ 27]   2.00-3.00   sec   114 MBytes   953 Mbits/sec    0   4.19 MBytes
[ 27]   3.00-4.00   sec   114 MBytes   956 Mbits/sec    0   4.38 MBytes
[ 27]   4.00-5.00   sec   114 MBytes   954 Mbits/sec    0   4.54 MBytes
[ 27]   5.00-6.00   sec   115 MBytes   965 Mbits/sec   39   3.27 MBytes
[ 27]   6.00-7.00   sec   114 MBytes   954 Mbits/sec    0   3.44 MBytes
[ 27]   7.00-8.00   sec   114 MBytes   954 Mbits/sec    0   3.59 MBytes
[ 27]   8.00-9.00   sec   114 MBytes   954 Mbits/sec    0   3.70 MBytes
[ 27]   9.00-10.00  sec   114 MBytes   954 Mbits/sec    0   3.79 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 27]   0.00-10.00  sec  1.11 GBytes   953 Mbits/sec  309             sender
[ 27]   0.00-10.05  sec  1.10 GBytes   940 Mbits/sec                  receiver

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

Figure 17. Performing a throughput test to verify the configuration.

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.

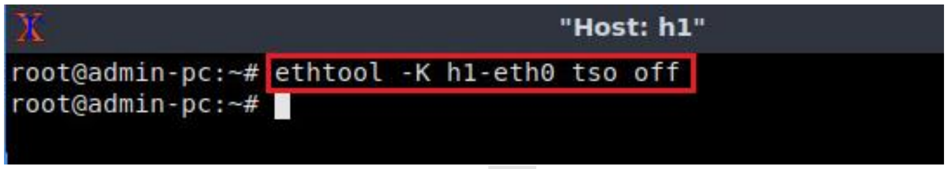
4 Disabling hardware offloading

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
```



"Host: h1"

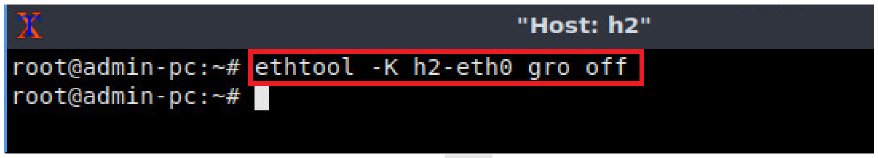
```
root@admin-pc:~# ethtool -K h1-eth0 tso off
root@admin-pc:~#
```

Figure 18. Disabling `gso` 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
```



"Host: h2"

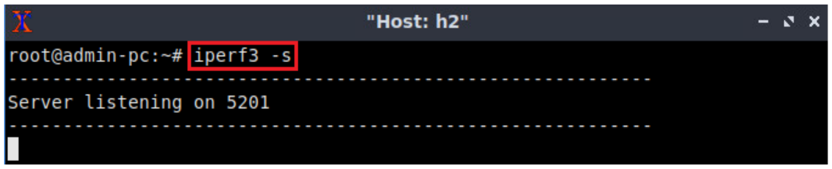
```
root@admin-pc:~# ethtool -K h2-eth0 gro off
root@admin-pc:~#
```

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
```



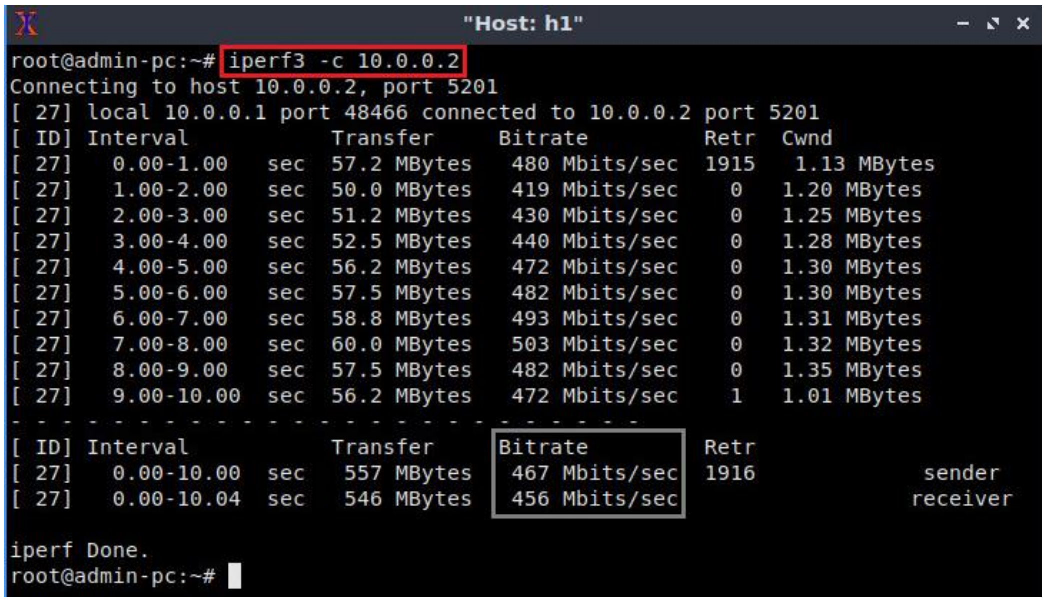
"Host: h2"

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

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
```



```
"Host: h1"
root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 27] local 10.0.0.1 port 48466 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 27]   0.00-1.00   sec  57.2 MBytes   480 Mbits/sec  1915   1.13 MBytes
[ 27]   1.00-2.00   sec  50.0 MBytes   419 Mbits/sec    0   1.20 MBytes
[ 27]   2.00-3.00   sec  51.2 MBytes   430 Mbits/sec    0   1.25 MBytes
[ 27]   3.00-4.00   sec  52.5 MBytes   440 Mbits/sec    0   1.28 MBytes
[ 27]   4.00-5.00   sec  56.2 MBytes   472 Mbits/sec    0   1.30 MBytes
[ 27]   5.00-6.00   sec  57.5 MBytes   482 Mbits/sec    0   1.30 MBytes
[ 27]   6.00-7.00   sec  58.8 MBytes   493 Mbits/sec    0   1.31 MBytes
[ 27]   7.00-8.00   sec  60.0 MBytes   503 Mbits/sec    0   1.32 MBytes
[ 27]   8.00-9.00   sec  57.5 MBytes   482 Mbits/sec    0   1.35 MBytes
[ 27]   9.00-10.00  sec  56.2 MBytes   472 Mbits/sec    1   1.01 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 27]   0.00-10.00  sec   557 MBytes   467 Mbits/sec  1916             sender
[ 27]   0.00-10.04  sec   546 MBytes   456 Mbits/sec                  receiver

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

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

1. Journey to the center of the linux kernel: traffic Control, shaping and QoS. [Online]. Available: http://wiki.linuxwall.info/doku.php/en:ressources:dossiers:networking:traffic_control.
2. How to use the linux traffic control panagiotis vouzis [Online]. Available: https://netbeez.net/blog/how-to-use-the-linux-traffic-control.
3. Segmentation and Checksum Offloading: Turning Off with ethtool. [Online]. Available: https://sandilands.info/sgordon/segmentation-offloading-with-wireshark-and-ethtool