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

This lab introduces TCP parallel streams in Wide Area Networks (WANs) and explains how they are used to achieve higher throughput. Then, throughput tests using parallel streams are conducted.

Objectives

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

1. Understand TCP parallel streams.
2. Describe the advantages of TCP parallel streams.
3. Specify the number of parallel streams in an iPerf3 test.
4. Conduct tests and measure performance of parallel streams on an emulated WAN.

Lab settings

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

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

Device	Account	Password
Client1	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction to TCP parallel streams.
2. Section 2: Lab topology.
3. Section 3: Parallel streams in a high-latency high-bandwidth WAN.
4. Section 4: Parallel streams with packet loss.

1 Introduction to TCP parallel streams

1.1 Parallel stream fundamentals

Parallel Streams are multiple TCP connections opened by an application to increase performance and maximize the throughput between communicating hosts. With parallel streams, data blocks for a single file transmitted from a sender to a receiver are distributed over the multiple streams. Figure 1 shows the basic model. A control channel is established between the sender and the receiver to coordinate the data transfer. The actual transfer occurs over the parallel streams, collectively referred to as data channels. In this context, the term stream is a synonym of flow and connection.

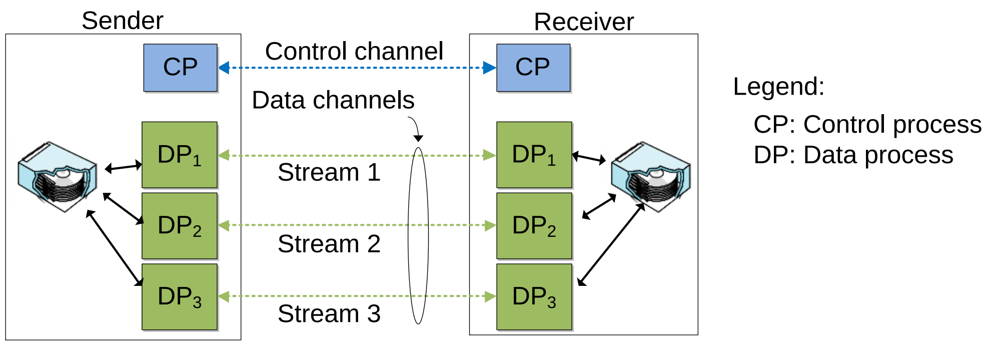


Figure 1. Data transfer model with parallel streams.

1.2 Advantages of parallel streams

Transferring large files over high-latency WANs with parallel streams have multiple benefits, as describe next.

Combat random packet loss not due congestion: assume that packet loss occurs randomly rather than due congestion. In steady state, the average throughput of a single TCP stream is given by[1]:

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

where MSS is the maximum segment size and L is the packet loss rate. The above equation indicates that the throughput is directly proportional to the MSS and inversely proportional to RTT and the square root of L. When an application uses K parallel streams and if RTT, packet loss, and MSS are the same in each stream, the aggregate average throughput is the aggregation of the K single stream throughputs[2]:

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

Thus, an application opening K parallel connections essentially creates a large virtual MSS on the aggregate connection that is K times the MSS of a single connection[2].

The TCP throughput follows the additive increase multiplicative decrease (AIMD) rule: TCP continuously probes for more bandwidth and increases the throughput of a connection by approximately 1 MSS per RTT as long as no packet loss occurs (additive increase phase). When a packet loss occurs, the throughput is reduced by half (multiplicative decrease event). Figure 2 illustrates the AIMD behavior for two connections with different MSSs. The MSS of the green connection is six than the MSS of the red connection. Since during the additive increase phase TCP increases the throughput by one MSS every RTT, the speed at which the throughput increases is proportional to the MSS (i.e., the larger the MSS the faster the recovery after a packet loss).

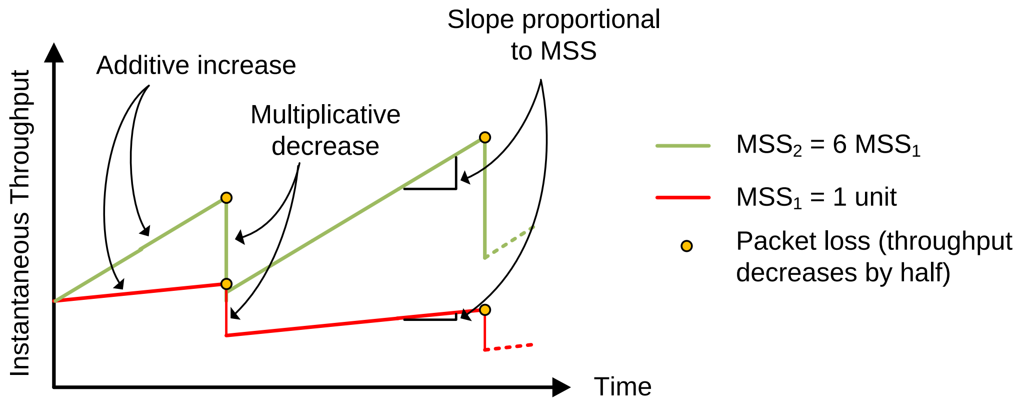


Figure 2. Additive increase multiplicative decrease (AIMD) behavior. The green curve corresponds to the throughput when the MSS is six times that of the red curve.

Mitigate TCP round-trip time (RTT) bias*:* when different flows with different RTTs share a given bottleneck link, TCP's throughput is inversely proportional to the RTT[3]. This is also noted in the equations discussed above. Hence, low-RTT flows get a higher share of the bandwidth than high-RTT flows. Thus, for transfers across high-latency WANs, one approach to combat the higher (unfair) bandwidth allocated to low-latency connections is by using parallel streams. By doing so, even if each high-latency stream receives less amount of bandwidth than low-latency flows, the aggregate throughput of the parallel streams can be high.

Overcome TCP buffer limitation: TCP receives data from the application layer and places it in the TCP buffer, as shown in Figure 3. TCP implements flow control by requiring the receiver indicate how much spare room is available in the TCP receive buffer. For a full utilization of the path, the TCP send and receive buffers must be greater than or equal to the bandwidth-delay product (BDP). This buffer size value is the maximum number of bits that can be outstanding (in-flight) if the sender continuously sends segments. If the buffer size is less than the bandwidth-delay product, then throughput will not be maximized. One solution to overcome small TCP buffer size situations is by using parallel streams. Essentially, an application opening K parallel connections creates a large buffer size on the aggregate connection that is K times the buffer size of a single connection.

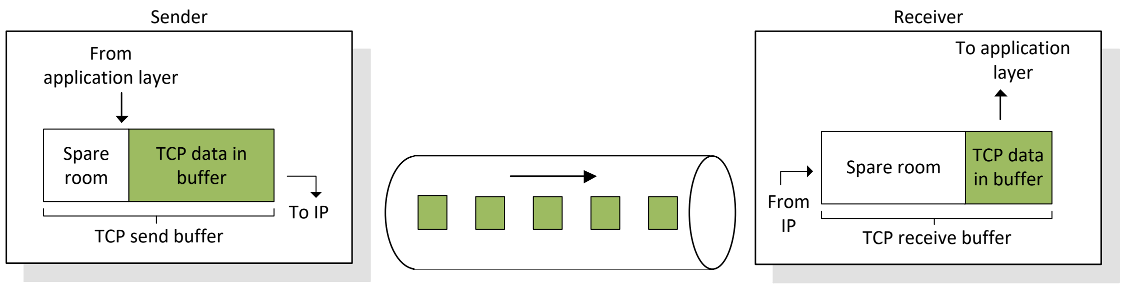


Figure 3. TCP send and receive buffers.

In this lab, we will explore the use of parallel streams to overcome TCP buffer limitation and to mitigate random packet loss.

2 Lab topology

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

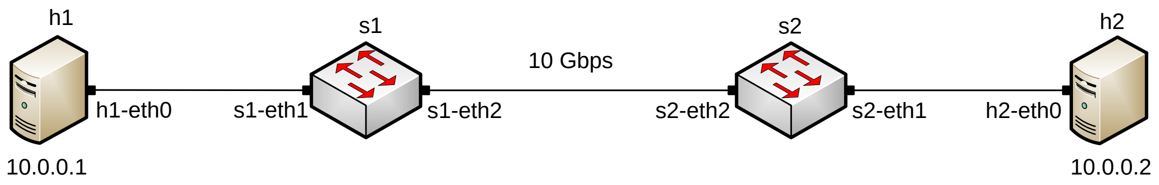


Figure 4. Lab topology.

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

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

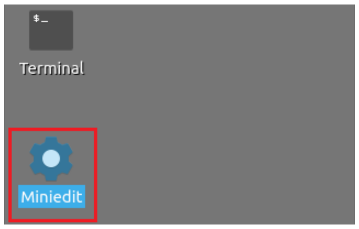


Figure 5. MiniEdit shortcut.

Step 2. On MiniEdit's menu bar, click on *File* then *Open* to load the lab's topology. Locate the *Lab 9.mn* topology file and click on *Open*.

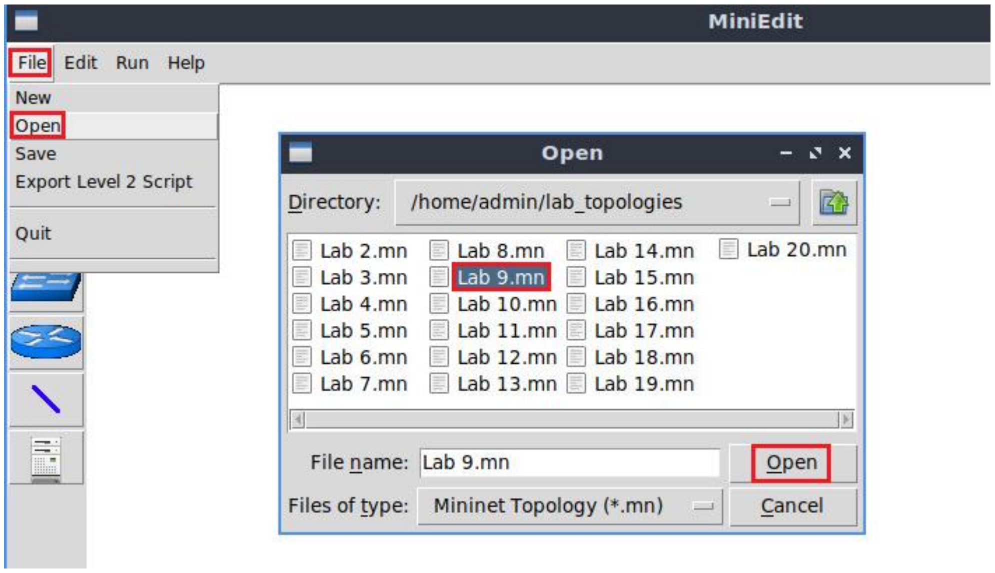


Figure 6. MiniEdit's *Open* dialog.

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

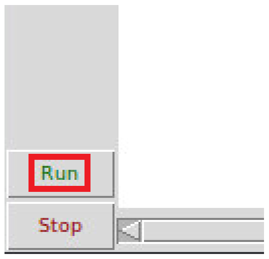


Figure 7. Running the emulation.

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

2.1 Starting host h1 and host h2

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

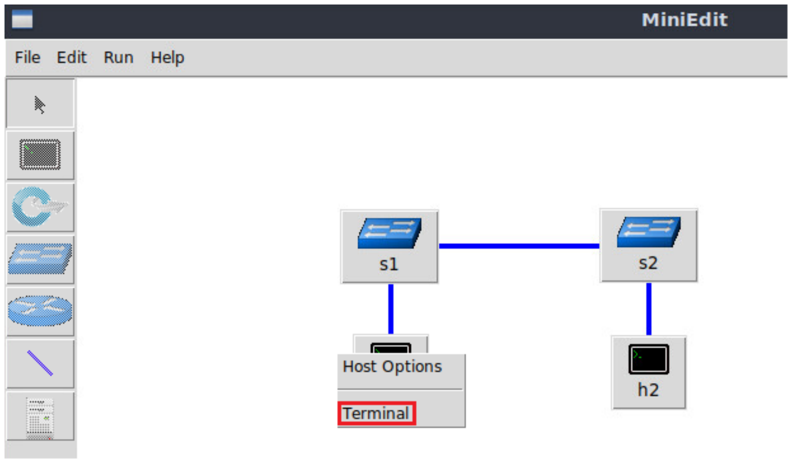
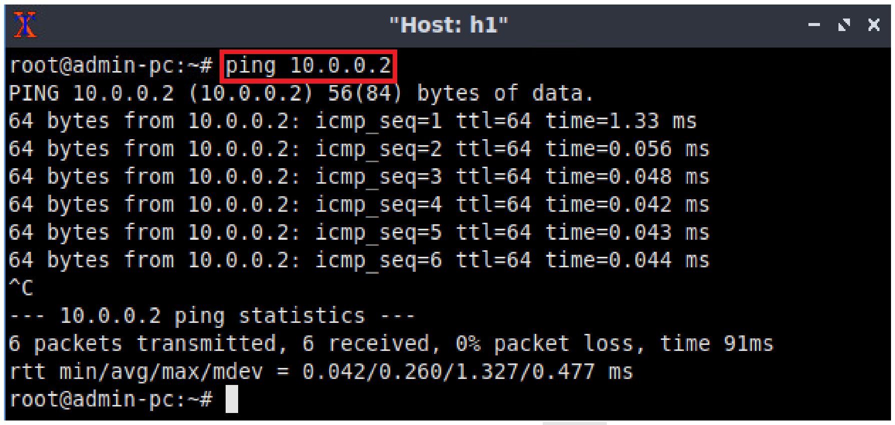


Figure 8. Opening a terminal on host h1.

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

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



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

Figure 9. Connectivity test using `ping` command.

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

2.2 Emulating 10 Gbps high-latency WAN

This section emulates a high-latency WAN. We will first emulate 20ms delay between switch S1 and switch S2 to measure the throughput. Then, we will set the bandwidth between host h1 and host h2 to 10 Gbps.

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

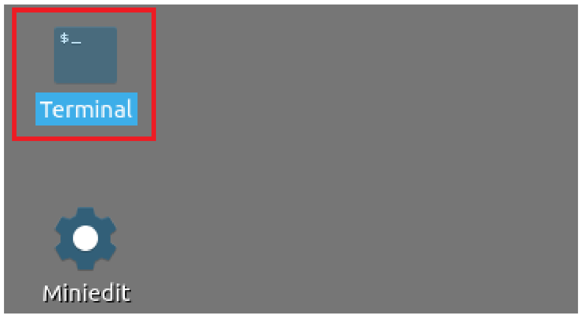
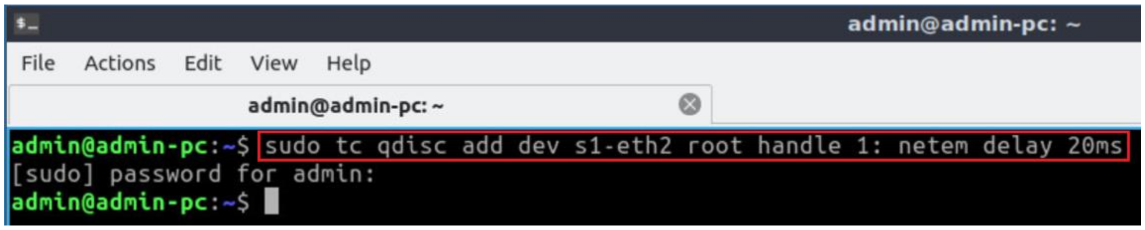


Figure 10. Shortcut to open a Linux terminal.

The Linux terminal is a program that opens a window and permits you to interact with a Command-Line Interface (CLI). A CLI is a program that takes commands from the keyboard and sends them to the operating system for execution.

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

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



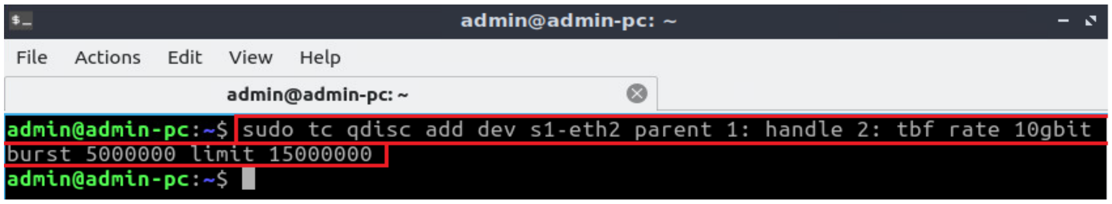
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem delay 20ms
[sudo] password for admin:
admin@admin-pc:~$
```

Figure 11. Adding delay of 20ms to switch S1's *s1-eth2* interface.

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

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

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



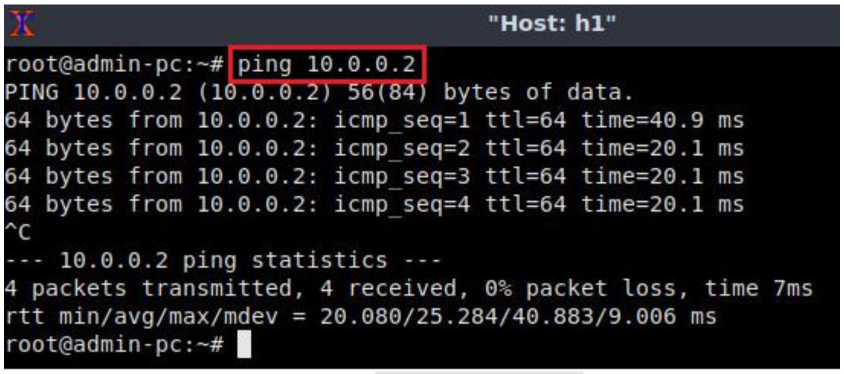
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$
```

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

2.3 Testing connection

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

Step 1. On the terminal of host h1, type `ping 10.0.0.2`. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test. Host h1 (10.0.0.1) sent four packets to host h2 (10.0.0.2), successfully receiving responses back.



```
root@admin-pc:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=40.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.1 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 7ms
rtt min/avg/max/mdev = 20.080/25.284/40.883/9.006 ms
root@admin-pc:~#
```

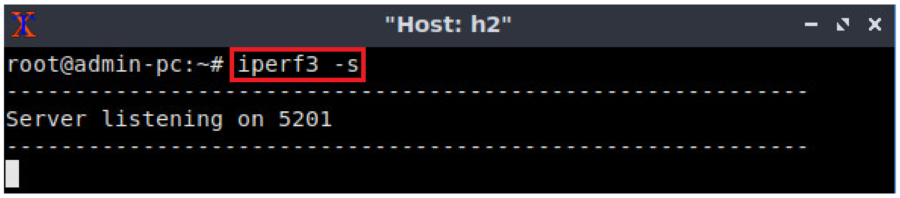
Figure 13. Output of `ping 10.0.0.2` command.

The result above indicates that all four packets were received successfully (0% packet loss) and that the minimum, average, maximum, and standard deviation of the Round-Trip Time (RTT) were 20.080, 25.284, 40.883, and 9.006 milliseconds, respectively. The output above verifies that delay was injected successfully, as the RTT is approximately 20ms.

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

Step 3. Launch iPerf3 in server mode on host h2's terminal.

```
iperf3 -s
```

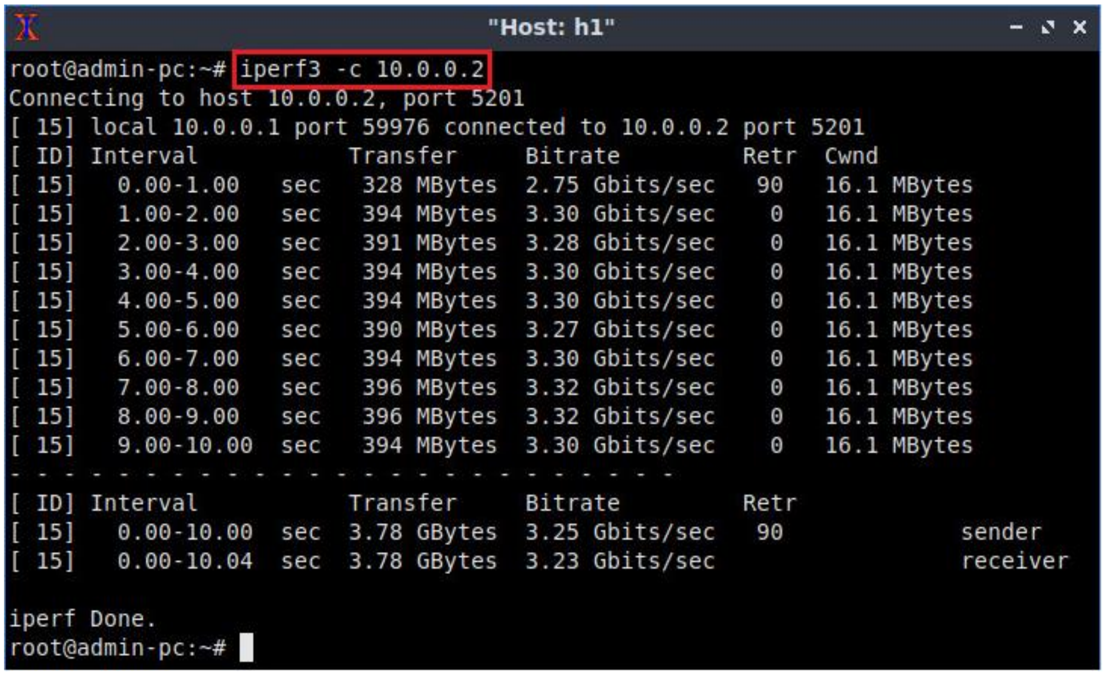


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 14. Starting iPerf3 server on host h2.

Step 4. Launch iPerf3 in client mode on host h1 's terminal. To stop the test, press `Ctrl+c`.

```
iperf3 -c 10.0.0.2
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 59976 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   328 MBytes  2.75 Gbits/sec   90   16.1 MBytes
[ 15]   1.00-2.00   sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
[ 15]   2.00-3.00   sec   391 MBytes  3.28 Gbits/sec    0   16.1 MBytes
[ 15]   3.00-4.00   sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
[ 15]   4.00-5.00   sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
[ 15]   5.00-6.00   sec   390 MBytes  3.27 Gbits/sec    0   16.1 MBytes
[ 15]   6.00-7.00   sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
[ 15]   7.00-8.00   sec   396 MBytes  3.32 Gbits/sec    0   16.1 MBytes
[ 15]   8.00-9.00   sec   396 MBytes  3.32 Gbits/sec    0   16.1 MBytes
[ 15]   9.00-10.00  sec   394 MBytes  3.30 Gbits/sec    0   16.1 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  3.78 GBytes  3.25 Gbits/sec   90             sender
[ 15]   0.00-10.04  sec  3.78 GBytes  3.23 Gbits/sec                  receiver

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

Figure 15. Running iPerf3 client on host h1.

Although the link was configured to 10 Gbps, the test results show that the achieved throughput is 3.22 Gbps. This is because the TCP buffer size is less than the bandwidth-delay product. In the upcoming section, we run a throughput test without modifying the TCP buffer size, but with multiple parallel streams.

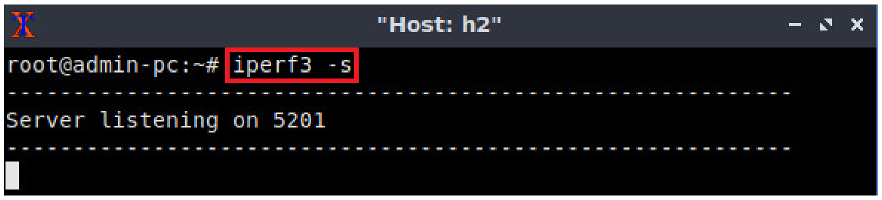
Step 5. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

3 Parallel streams to overcome TCP buffer limitation

In this section, parallel streams are specified by the client when executing the throughput test in iPerf3. The iPerf3 server should start as usual, without specifying any additional options or parameters.

Step 1. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown the figure below:

```
iperf3 -s
```

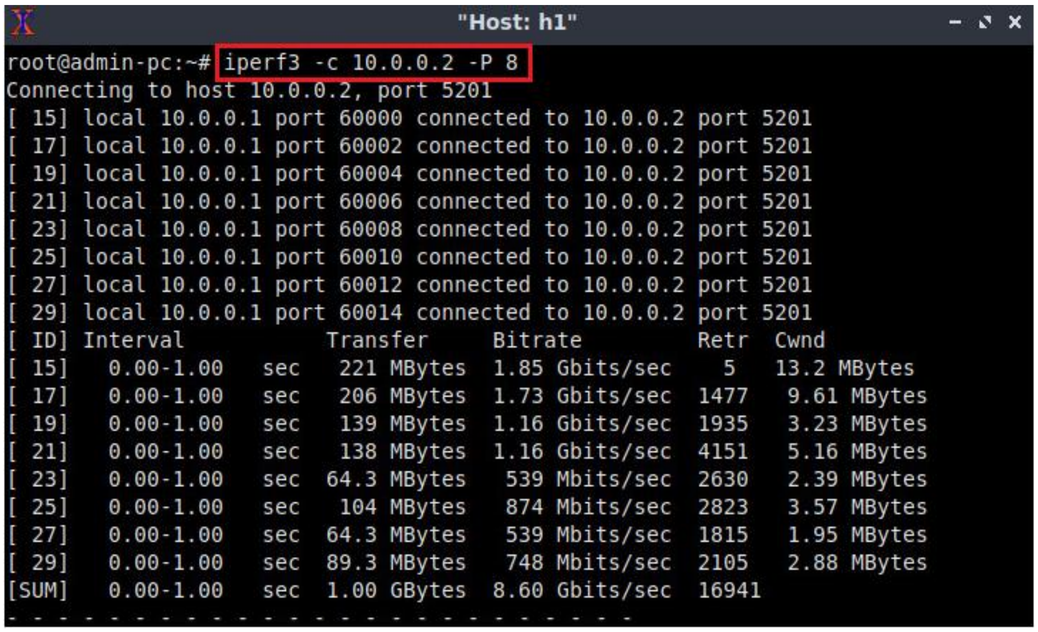


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 16. Host h2 running iPerf3 as server.

Step 2. Now the iPerf3 client should be launched with the `-P` option specified (not to be confused with the `-p` option which specifies the listening port number). This option specifies the number of parallel streams. Run the following command in host h1's terminal:

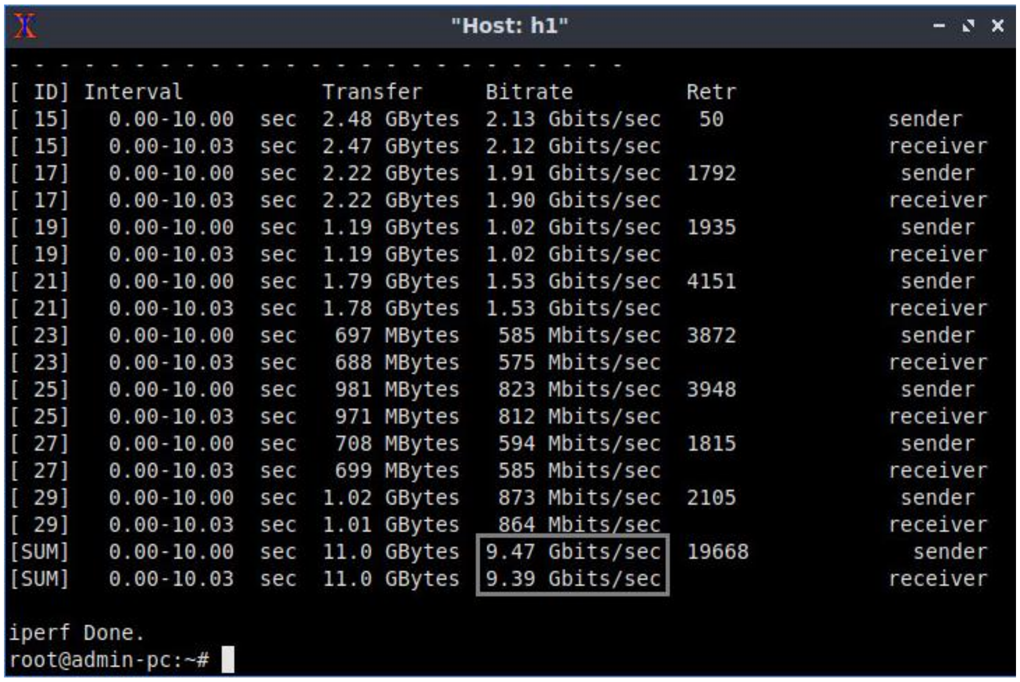
```
iperf3 -c 10.0.0.2 -P 8
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2 -P 8
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 60000 connected to 10.0.0.2 port 5201
[ 17] local 10.0.0.1 port 60002 connected to 10.0.0.2 port 5201
[ 19] local 10.0.0.1 port 60004 connected to 10.0.0.2 port 5201
[ 21] local 10.0.0.1 port 60006 connected to 10.0.0.2 port 5201
[ 23] local 10.0.0.1 port 60008 connected to 10.0.0.2 port 5201
[ 25] local 10.0.0.1 port 60010 connected to 10.0.0.2 port 5201
[ 27] local 10.0.0.1 port 60012 connected to 10.0.0.2 port 5201
[ 29] local 10.0.0.1 port 60014 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   221 MBytes  1.85 Gbits/sec    5   13.2 MBytes
[ 17]   0.00-1.00   sec   206 MBytes  1.73 Gbits/sec  1477    9.61 MBytes
[ 19]   0.00-1.00   sec   139 MBytes  1.16 Gbits/sec  1935    3.23 MBytes
[ 21]   0.00-1.00   sec   138 MBytes  1.16 Gbits/sec  4151    5.16 MBytes
[ 23]   0.00-1.00   sec  64.3 MBytes   539 Mbits/sec  2630    2.39 MBytes
[ 25]   0.00-1.00   sec   104 MBytes   874 Mbits/sec  2823    3.57 MBytes
[ 27]   0.00-1.00   sec  64.3 MBytes   539 Mbits/sec  1815    1.95 MBytes
[ 29]   0.00-1.00   sec  89.3 MBytes   748 Mbits/sec  2105    2.88 MBytes
[SUM]   0.00-1.00   sec  1.00 GBytes  8.60 Gbits/sec  16941
- - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 17. iPerf3 throughput test with parallel streams.

The above command uses 8 parallel streams. Note that 8 sockets are now opened on different local ports, and their streams are connected to the server, ready for transmitting data and performing the throughput test.



```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  2.48 GBytes  2.13 Gbits/sec   50             sender
[ 15]   0.00-10.03  sec  2.47 GBytes  2.12 Gbits/sec                  receiver
[ 17]   0.00-10.00  sec  2.22 GBytes  1.91 Gbits/sec  1792             sender
[ 17]   0.00-10.03  sec  2.22 GBytes  1.90 Gbits/sec                  receiver
[ 19]   0.00-10.00  sec  1.19 GBytes  1.02 Gbits/sec  1935             sender
[ 19]   0.00-10.03  sec  1.19 GBytes  1.02 Gbits/sec                  receiver
[ 21]   0.00-10.00  sec  1.79 GBytes  1.53 Gbits/sec  4151             sender
[ 21]   0.00-10.03  sec  1.78 GBytes  1.53 Gbits/sec                  receiver
[ 23]   0.00-10.00  sec   697 MBytes   585 Mbits/sec  3872             sender
[ 23]   0.00-10.03  sec   688 MBytes   575 Mbits/sec                  receiver
[ 25]   0.00-10.00  sec   981 MBytes   823 Mbits/sec  3948             sender
[ 25]   0.00-10.03  sec   971 MBytes   812 Mbits/sec                  receiver
[ 27]   0.00-10.00  sec   708 MBytes   594 Mbits/sec  1815             sender
[ 27]   0.00-10.03  sec   699 MBytes   585 Mbits/sec                  receiver
[ 29]   0.00-10.00  sec  1.02 GBytes   873 Mbits/sec  2105             sender
[ 29]   0.00-10.03  sec  1.01 GBytes   864 Mbits/sec                  receiver
[SUM]   0.00-10.00  sec  11.0 GBytes  9.47 Gbits/sec  19668             sender
[SUM]   0.00-10.03  sec  11.0 GBytes  9.39 Gbits/sec                  receiver

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

Figure 18. iPerf3 throughput test with parallel streams summary output.

Note the measured throughput now is approximately 9.5 Gbps, which is close to the value assigned in the `tbf` rule (10 Gbps).

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

4 Parallel streams to combat packet loss

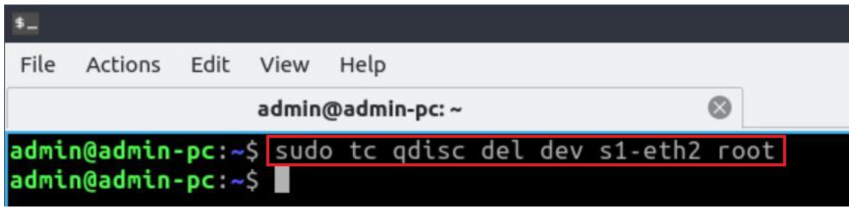
Packet loss is inevitable in real-world networks. This section explores the use of parallel streams to mitigate packet loss not due congestion (i.e., random packet loss), and compares the performance of single and parallel streams.

4.1 Limit rate and add packet loss on switch S1's s1-eth2 interface

In this topology, rate limiting is applied on switch S1's interface which connects it to switch S2 (*s1-eth2*) and 1% packet loss is introduced.

Step 1. Before applying any additional configuration, the previous rules assigned on the switch's interface must be deleted. To remove these, type the following command on the Client's terminal. When prompted for a password, type `password` and hit enter.

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

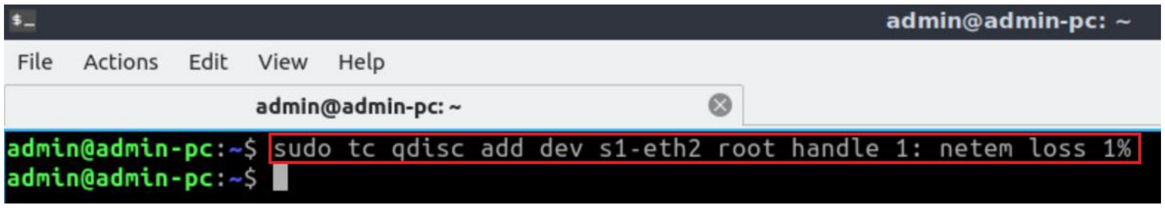


```
admin@admin-pc:~$ sudo tc qdisc del dev s1-eth2 root
admin@admin-pc:~$
```

Figure 19. Deleting previous rules on switch S1's *s1-eth2* interface.

Step 2. On the same terminal, type the below command to add 1% packet loss.

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



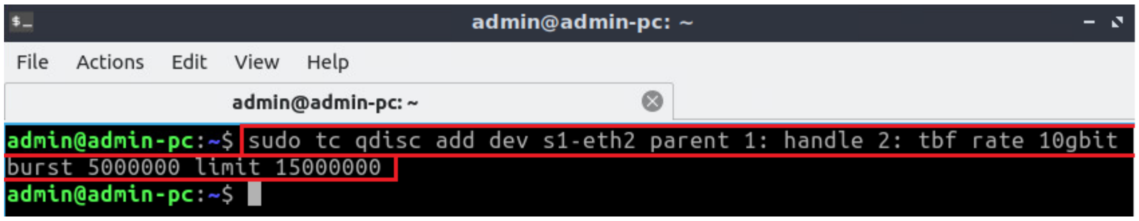
```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 root handle 1: netem loss 1%
admin@admin-pc:~$
```

Figure 20. Adding 1% packet loss to switch S1's *s1-eth2* interface.

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

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

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

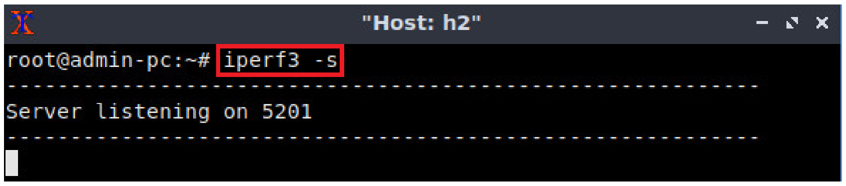


```
admin@admin-pc:~$ sudo tc qdisc add dev s1-eth2 parent 1: handle 2: tbf rate 10gbit
burst 5000000 limit 15000000
admin@admin-pc:~$
```

Figure 21. Limiting the bandwidth to 10 Gbps on switch S1's *s1-eth2* interface.

Step 3. The user can now verify the rate limit configuration by using the `iperf3` tool to measure throughput. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown the figure below:

```
iperf3 -s
```

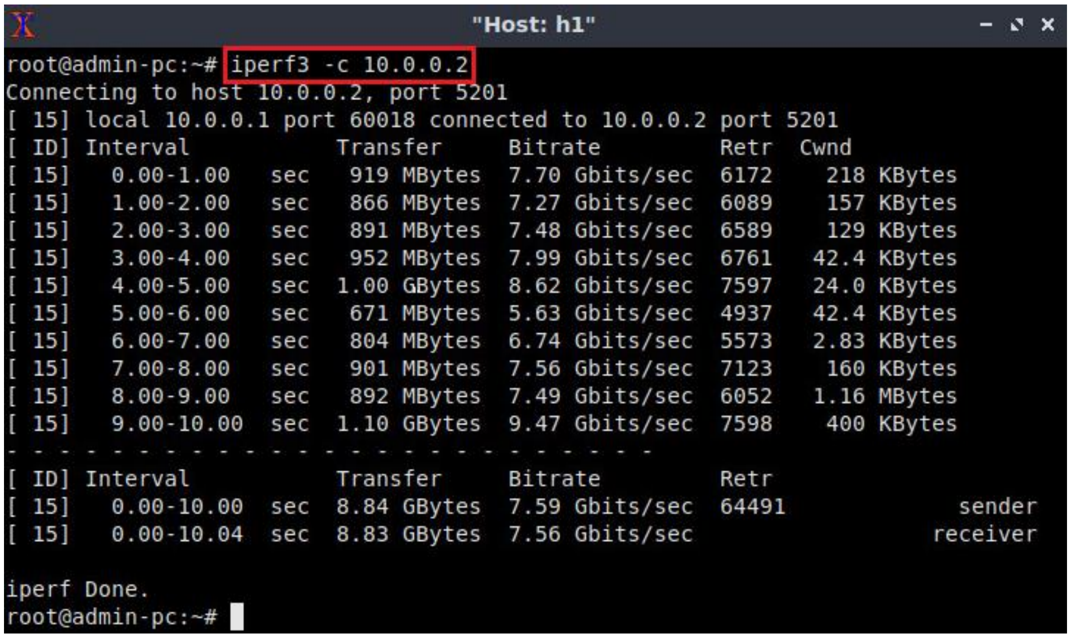


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 22. Starting iPerf3 server on host h2.

Step 4. Launch iPerf3 in client mode on host h1 's terminal. To stop the test, press `Ctrl+c`.

```
iperf3 -c 10.0.0.2
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 60018 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   919 MBytes  7.70 Gbits/sec  6172    218 KBytes
[ 15]   1.00-2.00   sec   866 MBytes  7.27 Gbits/sec  6089    157 KBytes
[ 15]   2.00-3.00   sec   891 MBytes  7.48 Gbits/sec  6589    129 KBytes
[ 15]   3.00-4.00   sec   952 MBytes  7.99 Gbits/sec  6761   42.4 KBytes
[ 15]   4.00-5.00   sec  1.00 GBytes  8.62 Gbits/sec  7597   24.0 KBytes
[ 15]   5.00-6.00   sec   671 MBytes  5.63 Gbits/sec  4937   42.4 KBytes
[ 15]   6.00-7.00   sec   804 MBytes  6.74 Gbits/sec  5573   2.83 KBytes
[ 15]   7.00-8.00   sec   901 MBytes  7.56 Gbits/sec  7123    160 KBytes
[ 15]   8.00-9.00   sec   892 MBytes  7.49 Gbits/sec  6052   1.16 MBytes
[ 15]   9.00-10.00  sec  1.10 GBytes  9.47 Gbits/sec  7598    400 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  8.84 GBytes  7.59 Gbits/sec  64491             sender
[ 15]   0.00-10.04  sec  8.83 GBytes  7.56 Gbits/sec                  receiver

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

Figure 23. Running iPerf3 client on host h1.

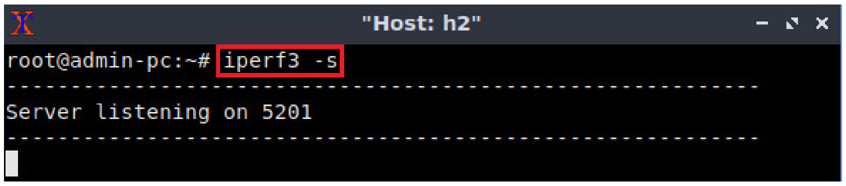
Note the measured throughput now is approximately 7.6 Gbps, which is different than the value assigned in the `tbf` rule (10 Gbps).

Step 5. In order to stop the server, press `Ctrl+c` in host h2's terminal. The user can see the throughput results in the server side too.

4.2 Test with parallel streams

Step 1. Now the test is repeated while using parallel streams. To launch iPerf3 in server mode, run the command `iperf3 -s` in host h2's terminal as shown in Figure 24:

```
iperf3 -s
```

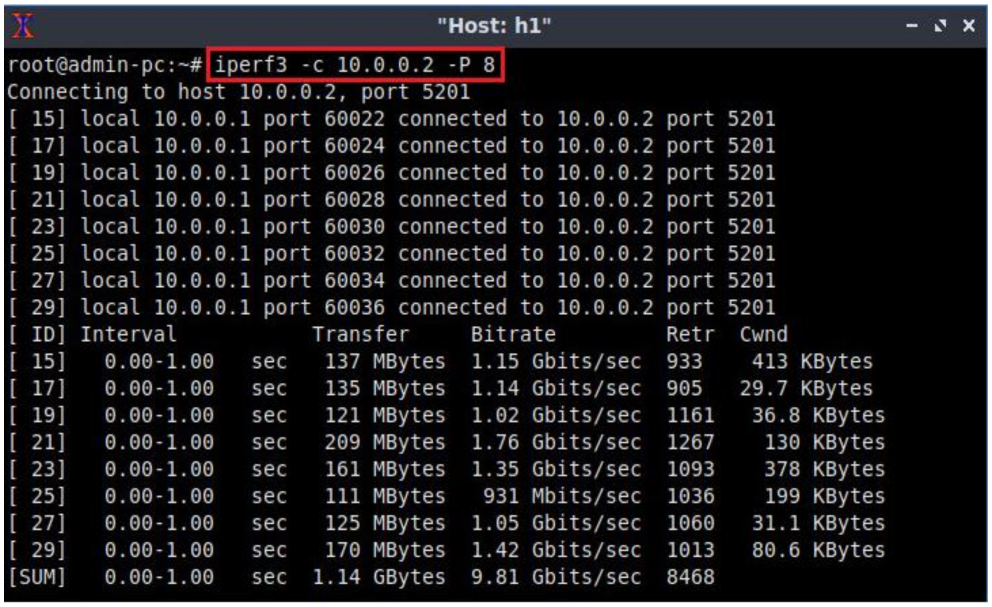


```
root@admin-pc:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 24. Host h2 running iPerf3 as server.

Step 2. Now the iPerf3 client should be launched with the `-P` option specified (not to be confused with the `-p` option which specifies the listening port number). This option specifies the number of parallel streams. Run the following command in host h1's terminal:

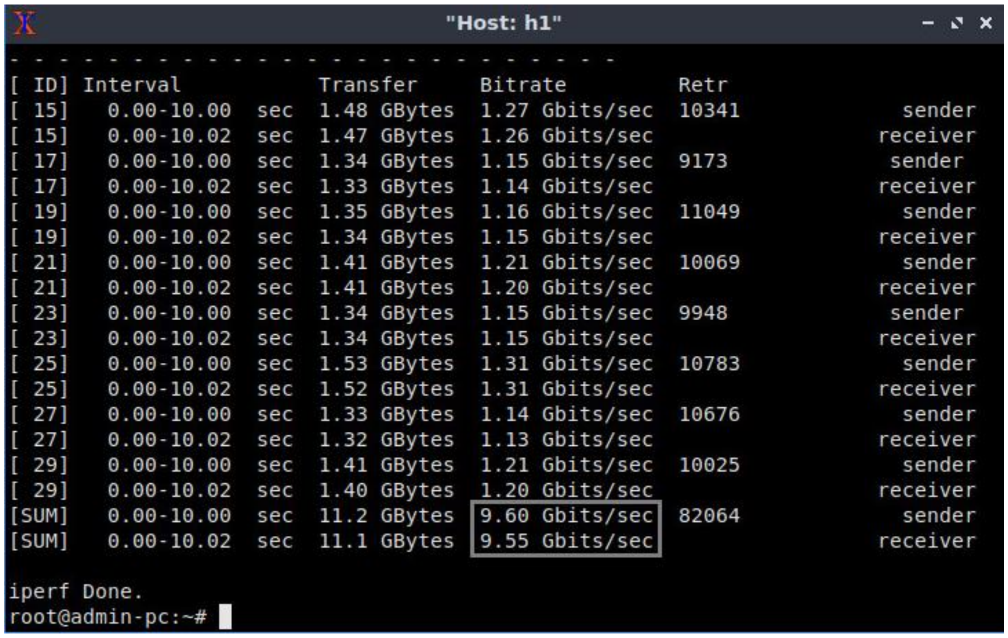
```
iperf3 -c 10.0.0.2 -P 8
```



```
root@admin-pc:~# iperf3 -c 10.0.0.2 -P 8
Connecting to host 10.0.0.2, port 5201
[ 15] local 10.0.0.1 port 60022 connected to 10.0.0.2 port 5201
[ 17] local 10.0.0.1 port 60024 connected to 10.0.0.2 port 5201
[ 19] local 10.0.0.1 port 60026 connected to 10.0.0.2 port 5201
[ 21] local 10.0.0.1 port 60028 connected to 10.0.0.2 port 5201
[ 23] local 10.0.0.1 port 60030 connected to 10.0.0.2 port 5201
[ 25] local 10.0.0.1 port 60032 connected to 10.0.0.2 port 5201
[ 27] local 10.0.0.1 port 60034 connected to 10.0.0.2 port 5201
[ 29] local 10.0.0.1 port 60036 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[ 15]   0.00-1.00   sec   137 MBytes  1.15 Gbits/sec  933    413 KBytes
[ 17]   0.00-1.00   sec   135 MBytes  1.14 Gbits/sec  905   29.7 KBytes
[ 19]   0.00-1.00   sec   121 MBytes  1.02 Gbits/sec  1161   36.8 KBytes
[ 21]   0.00-1.00   sec   209 MBytes  1.76 Gbits/sec  1267    130 KBytes
[ 23]   0.00-1.00   sec   161 MBytes  1.35 Gbits/sec  1093    378 KBytes
[ 25]   0.00-1.00   sec   111 MBytes   931 Mbits/sec  1036    199 KBytes
[ 27]   0.00-1.00   sec   125 MBytes  1.05 Gbits/sec  1060   31.1 KBytes
[ 29]   0.00-1.00   sec   170 MBytes  1.42 Gbits/sec  1013   80.6 KBytes
[SUM]   0.00-1.00   sec  1.14 GBytes  9.81 Gbits/sec  8468
```

Figure 25. Host h1 running iPerf3 as client with 8 parallel streams.

The above command uses 8 parallel streams. Note that 8 sockets are now opened on different local ports, and their streams are connected to the server, ready for transmitting data and performing the throughput test.



```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[ 15]   0.00-10.00  sec  1.48 GBytes  1.27 Gbits/sec  10341             sender
[ 15]   0.00-10.02  sec  1.47 GBytes  1.26 Gbits/sec                  receiver
[ 17]   0.00-10.00  sec  1.34 GBytes  1.15 Gbits/sec  9173             sender
[ 17]   0.00-10.02  sec  1.33 GBytes  1.14 Gbits/sec                  receiver
[ 19]   0.00-10.00  sec  1.35 GBytes  1.16 Gbits/sec  11049             sender
[ 19]   0.00-10.02  sec  1.34 GBytes  1.15 Gbits/sec                  receiver
[ 21]   0.00-10.00  sec  1.41 GBytes  1.21 Gbits/sec  10069             sender
[ 21]   0.00-10.02  sec  1.41 GBytes  1.20 Gbits/sec                  receiver
[ 23]   0.00-10.00  sec  1.34 GBytes  1.15 Gbits/sec  9948             sender
[ 23]   0.00-10.02  sec  1.34 GBytes  1.15 Gbits/sec                  receiver
[ 25]   0.00-10.00  sec  1.53 GBytes  1.31 Gbits/sec  10783             sender
[ 25]   0.00-10.02  sec  1.52 GBytes  1.31 Gbits/sec                  receiver
[ 27]   0.00-10.00  sec  1.33 GBytes  1.14 Gbits/sec  10676             sender
[ 27]   0.00-10.02  sec  1.32 GBytes  1.13 Gbits/sec                  receiver
[ 29]   0.00-10.00  sec  1.41 GBytes  1.21 Gbits/sec  10025             sender
[ 29]   0.00-10.02  sec  1.40 GBytes  1.20 Gbits/sec                  receiver
[SUM]   0.00-10.00  sec  11.2 GBytes  9.60 Gbits/sec  82064             sender
[SUM]   0.00-10.02  sec  11.1 GBytes  9.55 Gbits/sec                  receiver

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

Figure 26. iPerf3 throughput test with parallel streams summary output.

Note the measured throughput now is approximately 9.6 Gbps, which is close to the value assigned in our `tbf` rule (10 Gbps). In conclusion, parallel streams are beneficial when the packet loss rate is high. As shown in the previous test, when using parallel streams, the host was able to achieve the maximum theoretical bandwidth.

This concludes Lab 9. Stop the emulation and then exit out of MiniEdit.

References

1. M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," ACM Computer Communication Review, vol. 27, no 3, pp. 67-82, Jul. 1997.
2. T. Hacker, B. Athey, B. Noble, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," in Proceedings of the Parallel and Distributed Processing Symposium, Apr. 2001.
3. J. Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, pp. 303-314, Sep. 1998.