

"بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ"



Faculty of Engineering and Technology.

Computer Engineering Department.

ENCS5321

ADVANCED COMPUTER NETWORKS

Task #1

**Mininet**

Instructor: Dr. Ibrahim Nemer.

Students: Maisam Alaa 1200650

Adam Nassan 1202076

Section: 1

## Table of Contents

Abstract .....	3
Introduction .....	4
Task Overview .....	4
Customized Network Topology .....	4
Performance Measurement .....	4
Effect of Multiplexing.....	4
Tools Utilized .....	5
Procedure .....	6
➤ Mininet.....	6
Part 1: Everyday Mininet Usage .....	6
Part 2: Advanced Startup Options .....	6
Part 3: Mininet Command-Line Interface (CLI) Commands.....	6
Part 4: Python API Examples.....	7
➤ Build a Customized Network Topology.....	7
➤ Measure the Performance of the Network .....	9
➤ Effect of Multiplexing.....	12
Results and Discussion.....	18
Summary .....	21
References .....	22
Appendix .....	23

## Table of figures

Figure 1 The custom Topology .....	7
Figure 2 implementation of the topology on mininet .....	8
Figure 3 open TCP&UDP connection on h1 Xterm window.....	10
Figure 4 Measure latency on h3 Xterm window for TCP .....	11
Figure 5 Measure latency on h3 Xterm window for UDP .....	11
Figure 6 measure throughput on h3 Xterm window for TCP .....	12
Figure 7 measure throughput on h3 Xterm window for UDP.....	12
Figure 8 measure latency on h3 and h4 Xterm window .....	14
Figure 9 measure throughput on h3 and h4 Xterm window .....	15
Figure 10 measure latency on h1 and h2 Xterm window .....	16
Figure 11 measure throughput on h1 and h2 Xterm window.....	17

## Table of tables

Table 1 Comparison for the first scenario .....	19
Table 2 Comparison for secound scenario .....	19

## Abstract

The main goal of this Task is to familiarize ourselves with mininet, and expound the ability of making a custom network topology using Mininet APIs. With the provided diagram and considering of the hosts (h1 h2 h3 h4), switches (s1 s2) and the bandwidth and delay specified for each connection, we are required to set up the topology. Then, using Xterm windows latency and throughput are measured between them using ping and iperf, respectively, for both TCP and UDP connections. Finally, the effect of multiplexing is explored by establishing simultaneous connections between h1-h3 and h2-h4, predicting and measuring the latency and throughput of the two connections.

## Introduction

This task explores the practical applications of network emulation and performance measurement using Mininet, a powerful network emulator, in conjunction with tools like PuTTY, Virtual Machine, and Xming Server. The project tasks delve into network topology construction, performance measurement, and the impact of multiplexing on network latency and throughput.

## Task Overview

The task comprises several tasks aimed at understanding and analyzing network performance within a simulated environment. Beginning with the Mininet walkthrough, participants familiarize themselves with the Mininet environment, gaining insights into network emulation techniques and operational functionalities.

## Customized Network Topology

A key task involves building a customized network topology required, which consists of hosts (h1 - h4) and switches (s1 and s2) interconnected with specified link properties. Hosts are assigned IP addresses based on a predetermined scheme derived from student identification numbers.

## Performance Measurement

Using Xterm terminals, participants measure latency and throughput between designated hosts (h1 and h3) using ping and iperf utilities, exploring both TCP and UDP connections. Through systematic measurements, participants gain insights into network performance under varying traffic conditions.

## Effect of Multiplexing

The task further explores the effect of multiplexing on network performance by initiating simultaneous connections between multiple hosts. Through predictive analysis and

empirical measurements using ping and iperf, participants evaluate latency and throughput implications, considering scenarios with both concurrent and single-destination connections.

### Tools Utilized

The task leverages a suite of tools to facilitate network experimentation and analysis. We utilized PuTTY for secure shell access to the Mininet virtual machine environment hosted on a Virtual Machine platform. The Xming Server enables graphical interface capabilities, enhancing user experience and facilitating visualizations within the Mininet environment.

In summary, the task provides a comprehensive exploration of network emulation, performance measurement, and the impact of multiplexing on network dynamics, utilizing a blend of practical experimentation and analytical insights.

## Procedure

### ➤ Mininet

#### Part 1: Everyday Mininet Usage

Part 1 focuses on everyday usage, starting with command syntax explanations. It guides through displaying startup options, initiating Wireshark for OpenFlow traffic monitoring, and addressing potential installation or configuration issues. The tutorial then delves into interacting with hosts and switches within a minimal topology, demonstrating Mininet CLI commands such as `help`, `nodes`, and `net`. Emphasis is placed on testing connectivity between hosts, revealing the flow of OpenFlow control traffic during communication.

#### Part 2: Advanced Startup Options

In Part 2 advanced startup options and features are explored. The tutorial introduces regression tests, demonstrating the ability to run self-contained tests using commands like ``$ sudo mn --test pingpair`` and ``$ sudo mn --test iperf``. It also covers changing the topology size and type using the ``--topo`` option, providing examples such as ``--topo single,3`` and ``--topo linear,4``. The importance of assigning specific MAC addresses to hosts for easier debugging is highlighted with the ``--mac`` option. Additionally, the tutorial covers the use of XTerm for debugging and interactive commands, as well as the exploration of different switch types, including the user-space switch and Open vSwitch (OVS).

#### Part 3: Mininet Command-Line Interface (CLI) Commands

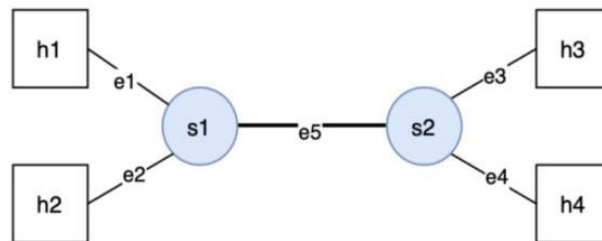
Part 3 guided through the Mininet Command-Line Interface (CLI) commands, starting with the display of options by initiating a minimal topology with ``$ sudo mn``. The tutorial introduces the Python interpreter within the Mininet CLI, illustrating its utility for executing Python commands, exploring local variables, and inspecting methods and properties of nodes.

## Part 4: Python API Examples

Part 4 of the Mininet walkthrough delves into Python API examples available in the Mininet source tree's examples directory. The tutorial highlights the existence of examples showcasing how to utilize Mininet's Python API, offering potentially valuable code that hasn't been integrated into the main code base. One specific example is presented, demonstrating the execution of an SSH daemon on every host using the command ``$ sudo ~/mininet/examples/sshd.py``.

### ➤ Build a Customized Network Topology

We detailed the process of building a customized network topology using Mininet APIs, as per the requirements outlined in the task. The primary objective was to replicate a specific network topology depicted in Figure 1, configuring hosts and switches with assigned IP addresses and link properties.



*Figure 1 The custom Topology*

**Objective:** The task entailed constructing a network topology comprising two switches (s1, s2) and four hosts (h1 - h4), interconnected by specified links with designated bandwidth and delay parameters. The IP addresses of the hosts were to be determined based on a predetermined scheme involving student identification numbers.

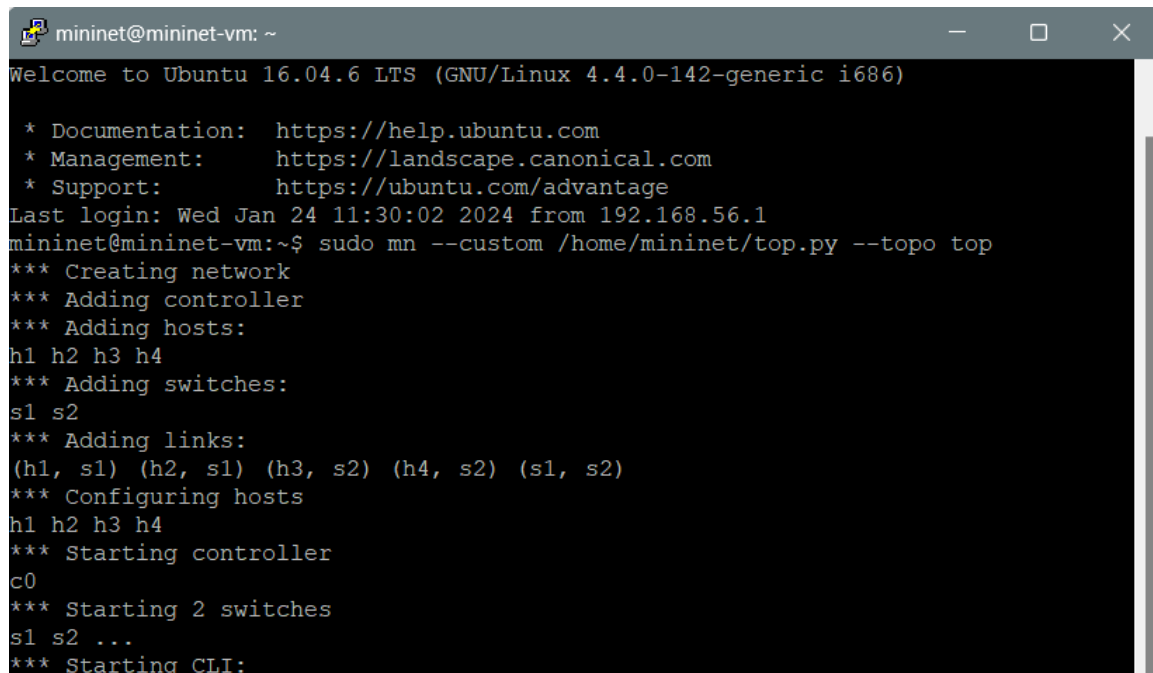
**Techniques:** To achieve the prescribed network configuration, Mininet, a popular network emulator, was utilized due to its ability to create virtual networks swiftly. We employed Mininet's Python APIs to define the topology, incorporating switches, hosts, and links while specifying properties such as bandwidth and delay.



**Implementation:** The topology was implemented using a Python script named **top.py**. The script defined a custom topology class, **CustTop**, inheriting from Mininet's **Topo** class. Within the **CustTop** class, switches (**s1**, **s2**) and hosts (**h1** - **h4**) were instantiated and connected according to the prescribed topology. The specified link properties, including bandwidth and delay, were assigned using Mininet's API functions.

**Validation:** Upon completing the topology definition, the script was executed within the Mininet environment to give us superuser privileges to perform administrative tasks in the mininet to validate its functionality. The success of the implementation was confirmed by verifying the connectivity between hosts and switches, as well as the adherence to the specified link properties.

```
sudo mn --custom /home/mininet/top.py --topo top
```



```
mininet@mininet-vm: ~  
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-142-generic i686)  
  
 * Documentation:  https://help.ubuntu.com  
 * Management:    https://landscape.canonical.com  
 * Support:       https://ubuntu.com/advantage  
Last login: Wed Jan 24 11:30:02 2024 from 192.168.56.1  
mininet@mininet-vm:~$ sudo mn --custom /home/mininet/top.py --topo top  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2 h3 h4  
*** Adding switches:  
s1 s2  
*** Adding links:  
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2)  
*** Configuring hosts  
h1 h2 h3 h4  
*** Starting controller  
c0  
*** Starting 2 switches  
s1 s2 ...  
*** Starting CLI:
```

*Figure 2 implementation of the topology on mininet*

### ➤ Measure the Performance of the Network

Based on the custom topology we did in the previous part, we evaluate the network performance between hosts h1 and h3 through ping and iperf tests. This includes measuring latency with ping, assessing round-trip time for data transmission, and using iperf to gauge both TCP and UDP throughput. The goal is to gain insights into the network's responsiveness, identify potential bottlenecks, and understand the impact of different transport layer protocols on data transfer capacity.

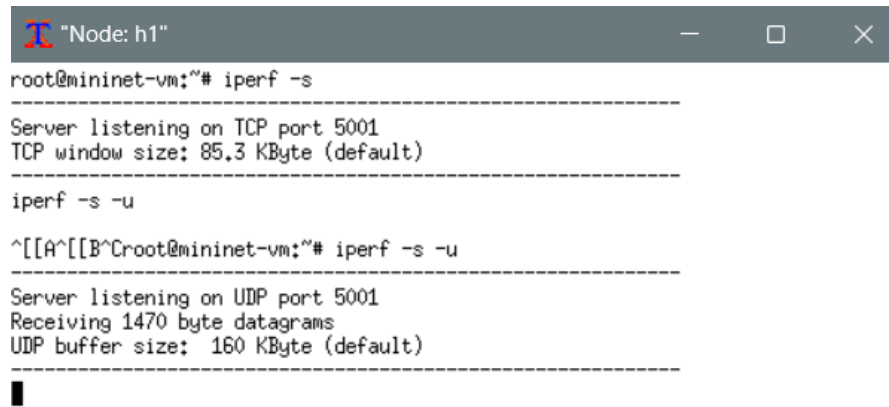
**Steps: we followed the following steps to measure the performance:**

- 1- We opened two xterm terminals for hosts h1 and h3 in the Mininet CLI using the following command:  

```
mininet> xterm h1 h3
```
- 2- In the xterm window for host h1, we started iperf in server mode (for both TCP and UDP) using the following commands as shown in figure 3.

**TCP: h1\$ iperf -s**

**UDP: h1\$ iperf -s -u**



```
Node: h1
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
iperf -s -u
^[[A^[[B^Croot@mininet-vm:~# iperf -s -u
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 160 KByte (default)
-----
```

*Figure 3 open TCP&UDP connection on h1 Xterm window*

- 3- In the xterm window for host h3, we measured latency (for both TCP and UDP) using the following command as shown in figure 4 & 5

**TCP: h3\$ ping -c 20 192.76.0.1**

**UDP: h3\$ ping -c 20 -i 1 -w 20 192.76.0.1**

```
"Node: h3"
root@mininet-vm:~# ping -c 20 192.76.0.1
PING 192.76.0.1 (192.76.0.1) 56(84) bytes of data.
64 bytes from 192.76.0.1: icmp_seq=1 ttl=64 time=266 ms
64 bytes from 192.76.0.1: icmp_seq=2 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=3 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=4 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=5 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=6 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=7 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=8 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=9 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=10 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=11 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=12 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=13 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=14 ttl=64 time=130 ms
64 bytes from 192.76.0.1: icmp_seq=15 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=16 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=17 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=18 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=19 ttl=64 time=130 ms
64 bytes from 192.76.0.1: icmp_seq=20 ttl=64 time=132 ms

--- 192.76.0.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19026ms
rtt min/avg/max/mdev = 130.607/138.789/266.420/29.291 ms
```

Figure 4 Measure latency on h3 Xterm window for TCP

```
"Node: h3"
[-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
[-w deadline] [-W timeout] [hop1 ...] destination
root@mininet-vm:~# ping -c 20 -i 1 -w 20 192.76.0.1
PING 192.76.0.1 (192.76.0.1) 56(84) bytes of data.
64 bytes from 192.76.0.1: icmp_seq=1 ttl=64 time=135 ms
64 bytes from 192.76.0.1: icmp_seq=2 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=3 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=4 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=5 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=6 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=7 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=8 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=9 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=10 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=11 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=12 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=13 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=14 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=15 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=16 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=17 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=18 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=19 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=20 ttl=64 time=133 ms

--- 192.76.0.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19036ms
rtt min/avg/max/mdev = 131.166/132.670/135.832/1.172 ms
```

Figure 5 Measure latency on h3 Xterm window for UDP

- 4- In the xterm window for host h3, we measured throughput using iperf for both TCP and UDP using the following command as shown in figures 6 & 7:

**TCP: h3\$ iperf -c 192.76.0.1 -t 20**

```
"Node: h3"
rtt min/avg/max/mdev = 130.607/138.789/266.420/29.291 ms
root@mininet-vm:~# iperf -c 192.76.0.1 -t 20

-----
Client connecting to 192.76.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 19] local 192.76.0.3 port 45414 connected with 192.76.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-21.4 sec  41.0 MBytes 16.1 Mbits/sec
```

*Figure 6 measure throughput on h3 Xterm window for TCP*

**UDP: h3\$ iperf -c 192.76.0.1 -t 20 -u**

```
"Node: h3"

-----
[ 19] local 192.76.0.3 port 45414 connected with 192.76.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-21.4 sec  41.0 MBytes 16.1 Mbits/sec
root@mininet-vm:~# iperf -c 192.76.0.1 -t 20 -u
-----
Client connecting to 192.76.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 160 KByte (default)
-----
[ 19] local 192.76.0.3 port 60316 connected with 192.76.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-20.0 sec  2.50 MBytes 1.05 Mbits/sec
[ 19] Sent 1785 datagrams
[ 19] Server Report:
[ 19] 0.0-20.0 sec  2.50 MBytes 1.05 Mbits/sec  0.513 ms  0/ 1785 (0%)
```

*Figure 7 measure throughput on h3 Xterm window for UDP*

### ➤ Effect of Multiplexing

The objective of this step is to explore the impact of simultaneous connections on latency and throughput within the network. By initiating concurrent communication between hosts (e.g., h1 talking to h3 and h2 talking to h4), the goal is to observe how multiplexing affects

both latency and throughput. Additionally, the second scenario where two connections (h1 & h2) are directed to the same destination (h4) to analyze potential variations in performance.

### Steps:

- 1- By using the same commands in the pervious part, we opened four xterm terminals for hosts h1,h2, h3 and h4 in the Mininet CLI using the following command:

```
mininet> xterm h1 h2 h3 h4
```

- 2- In the xterm window for host h1 and h2, we started iperf in server mode for TCP using the following commands as shown in figure 9.

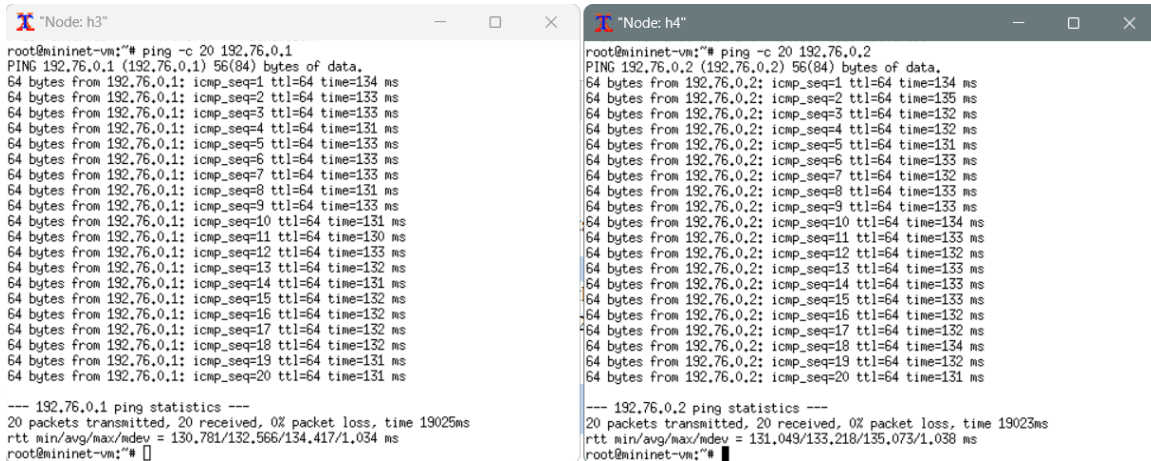
```
H1: h1$ iperf -s
```

```
H2: h2$ iperf -s
```

- 3- In the xterm window for host h3 and h4, we measured latency using the following command as shown in figure 8:

```
H3: h3$ ping -c 20 192.76.0.1
```

```
H4: h4$ ping -c 20 192.76.0.2
```



The image shows two Xterm windows side-by-side. The left window is titled "Node: h3" and shows the output of a ping command from root@mininet-vm to 192.76.0.1. It displays 20 successful ping results with times ranging from 131ms to 135ms. The right window is titled "Node: h4" and shows the output of a ping command from root@mininet-vm to 192.76.0.2. It also displays 20 successful ping results with times ranging from 131ms to 135ms. Both windows include a summary of ping statistics at the bottom.

```
root@mininet-vm:~# ping -c 20 192.76.0.1
PING 192.76.0.1 (192.76.0.1) 56(84) bytes of data:
64 bytes from 192.76.0.1: icmp_seq=1 ttl=64 time=134 ms
64 bytes from 192.76.0.1: icmp_seq=2 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=3 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=4 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=5 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=6 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=7 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=8 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=9 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=10 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=11 ttl=64 time=130 ms
64 bytes from 192.76.0.1: icmp_seq=12 ttl=64 time=133 ms
64 bytes from 192.76.0.1: icmp_seq=13 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=14 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=15 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=16 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=17 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=18 ttl=64 time=132 ms
64 bytes from 192.76.0.1: icmp_seq=19 ttl=64 time=131 ms
64 bytes from 192.76.0.1: icmp_seq=20 ttl=64 time=131 ms

--- 192.76.0.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19025ms
rtt min/avg/max/mdev = 130.781/132.566/134.417/1.034 ms
root@mininet-vm:~#
```

```
root@mininet-vm:~# ping -c 20 192.76.0.2
PING 192.76.0.2 (192.76.0.2) 56(84) bytes of data:
64 bytes from 192.76.0.2: icmp_seq=1 ttl=64 time=134 ms
64 bytes from 192.76.0.2: icmp_seq=2 ttl=64 time=135 ms
64 bytes from 192.76.0.2: icmp_seq=3 ttl=64 time=132 ms
64 bytes from 192.76.0.2: icmp_seq=4 ttl=64 time=132 ms
64 bytes from 192.76.0.2: icmp_seq=5 ttl=64 time=131 ms
64 bytes from 192.76.0.2: icmp_seq=6 ttl=64 time=133 ms
64 bytes from 192.76.0.2: icmp_seq=7 ttl=64 time=132 ms
64 bytes from 192.76.0.2: icmp_seq=8 ttl=64 time=133 ms
64 bytes from 192.76.0.2: icmp_seq=9 ttl=64 time=133 ms
64 bytes from 192.76.0.2: icmp_seq=10 ttl=64 time=134 ms
64 bytes from 192.76.0.2: icmp_seq=11 ttl=64 time=133 ms
64 bytes from 192.76.0.2: icmp_seq=12 ttl=64 time=132 ms
64 bytes from 192.76.0.2: icmp_seq=13 ttl=64 time=133 ms
64 bytes from 192.76.0.2: icmp_seq=14 ttl=64 time=133 ms
64 bytes from 192.76.0.2: icmp_seq=15 ttl=64 time=133 ms
64 bytes from 192.76.0.2: icmp_seq=16 ttl=64 time=132 ms
64 bytes from 192.76.0.2: icmp_seq=17 ttl=64 time=132 ms
64 bytes from 192.76.0.2: icmp_seq=18 ttl=64 time=134 ms
64 bytes from 192.76.0.2: icmp_seq=19 ttl=64 time=132 ms
64 bytes from 192.76.0.2: icmp_seq=20 ttl=64 time=131 ms

--- 192.76.0.2 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19023ms
rtt min/avg/max/mdev = 131.049/133.218/135.073/1.038 ms
root@mininet-vm:~#
```

Figure 8 measure latency on h3 and h4 Xterm window

Then in the Xterm window of host h3 and h4, we measured throughput using the following command as shown in figure 9:

**H3: h3\$ iperf -c 192.76.0.1 -t 20**

**H4: h4\$ iperf -c 192.76.0.2 -t 20**

The figure displays four terminal windows arranged in a 2x2 grid, each showing the output of the iperf command. The top-left window (h1) shows server listening on port 5001 and a client connection from 192.76.0.1 at port 45432, achieving a bandwidth of 11.8 Mbits/sec. The top-right window (h2) shows server listening on port 5001 and a client connection from 192.76.0.2 at port 58502, achieving a bandwidth of 7.49 Mbits/sec. The bottom-left window (h3) shows client connecting to 192.76.0.1 on port 5001 and a local connection to 192.76.0.3 at port 45432, achieving a bandwidth of 13.3 Mbits/sec. The bottom-right window (h4) shows client connecting to 192.76.0.2 on port 5001 and a local connection to 192.76.0.4 at port 58502, achieving a bandwidth of 8.28 Mbits/sec. Each window includes a table with columns for ID, Interval, Transfer, and Bandwidth.

```
"Node: h1"
root@mininet-vx:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 20] local 192.76.0.1 port 5001 connected with 192.76.0.3 port 45432
[ ID] Interval      Transfer     Bandwidth
[ 20] 0.0-23.2 sec  32.8 MBytes  11.8 Mbits/sec

"Node: h2"
root@mininet-vx:~# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 20] local 192.76.0.2 port 5001 connected with 192.76.0.4 port 58502
[ ID] Interval      Transfer     Bandwidth
[ 20] 0.0-22.8 sec  20.4 MBytes  7.49 Mbits/sec

"Node: h3"
root@mininet-vx:~# iperf -c 192.76.0.1 -t 20
Client connecting to 192.76.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 19] local 192.76.0.3 port 45432 connected with 192.76.0.1 port 5001
[ ID] Interval      Transfer     Bandwidth
[ 19] 0.0-20.6 sec  32.8 MBytes  13.3 Mbits/sec

"Node: h4"
root@mininet-vx:~# iperf -c 192.76.0.2 -t 20
Client connecting to 192.76.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 19] local 192.76.0.4 port 58502 connected with 192.76.0.2 port 5001
[ ID] Interval      Transfer     Bandwidth
[ 19] 0.0-20.6 sec  20.4 MBytes  8.28 Mbits/sec
```

Figure 9 measure throughput on h3 and h4 Xterm window

## Second scenario steps:

- 1- By using the same commands in the pervious part, we opened three xterm terminals for hosts h1, h2 and h4 in the Mininet CLI using the following command:

```
mininet> xterm h1 h2 h4
```

- 2- In the xterm window for host h4, we started iperf in server mode for TCP using the following commands as shown in figure 11.

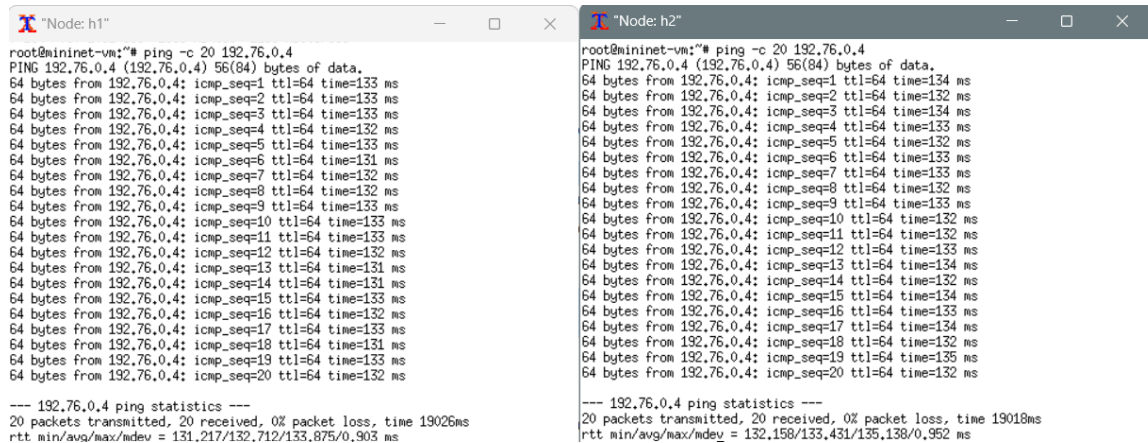
```
H4: h4$ iperf -s
```



- 3- In the xterm window for host h1 and h2, we measured latency using the following command as shown in figure 10:

**H1: h1\$ ping -c 20 192.76.0.4**

**H2: h2\$ ping -c 20 192.76.0.4**



The image shows two side-by-side xterm windows. The left window is titled "Node: h1" and the right window is titled "Node: h2". Both windows show the output of the command `ping -c 20 192.76.0.4`. The output for h1 shows a total time of 19026ms and an average rtt of 131.217ms. The output for h2 shows a total time of 19018ms and an average rtt of 132.158ms. Both windows show 20 packets transmitted and received with 0% packet loss.

```
root@mininet-vm:~# ping -c 20 192.76.0.4
PING 192.76.0.4 (192.76.0.4) 56(84) bytes of data:
64 bytes from 192.76.0.4: icmp_seq=1 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=2 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=3 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=4 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=5 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=6 ttl=64 time=131 ms
64 bytes from 192.76.0.4: icmp_seq=7 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=8 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=9 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=10 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=11 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=12 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=13 ttl=64 time=131 ms
64 bytes from 192.76.0.4: icmp_seq=14 ttl=64 time=131 ms
64 bytes from 192.76.0.4: icmp_seq=15 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=16 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=17 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=18 ttl=64 time=131 ms
64 bytes from 192.76.0.4: icmp_seq=19 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=20 ttl=64 time=132 ms

--- 192.76.0.4 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19026ms
rtt min/avg/max/mdev = 131.217/132.712/133.875/0.903 ms

root@mininet-vm:~# ping -c 20 192.76.0.4
PING 192.76.0.4 (192.76.0.4) 56(84) bytes of data:
64 bytes from 192.76.0.4: icmp_seq=1 ttl=64 time=134 ms
64 bytes from 192.76.0.4: icmp_seq=2 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=3 ttl=64 time=134 ms
64 bytes from 192.76.0.4: icmp_seq=4 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=5 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=6 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=7 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=8 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=9 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=10 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=11 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=12 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=13 ttl=64 time=134 ms
64 bytes from 192.76.0.4: icmp_seq=14 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=15 ttl=64 time=134 ms
64 bytes from 192.76.0.4: icmp_seq=16 ttl=64 time=133 ms
64 bytes from 192.76.0.4: icmp_seq=17 ttl=64 time=134 ms
64 bytes from 192.76.0.4: icmp_seq=18 ttl=64 time=132 ms
64 bytes from 192.76.0.4: icmp_seq=19 ttl=64 time=135 ms
64 bytes from 192.76.0.4: icmp_seq=20 ttl=64 time=132 ms

--- 192.76.0.4 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19018ms
rtt min/avg/max/mdev = 132.158/133.431/135.138/0.952 ms
```

Figure 10 measure latency on h1 and h2 Xterm window

- 4- Then in the Xterm window of host h1 and h2, we measured throughput using the following command as shown in figure 11:

**H1: h1\$ iperf -c 192.76.0.4 -t 20**

**H2: h2\$ iperf -c 192.76.0.4 -t 20**

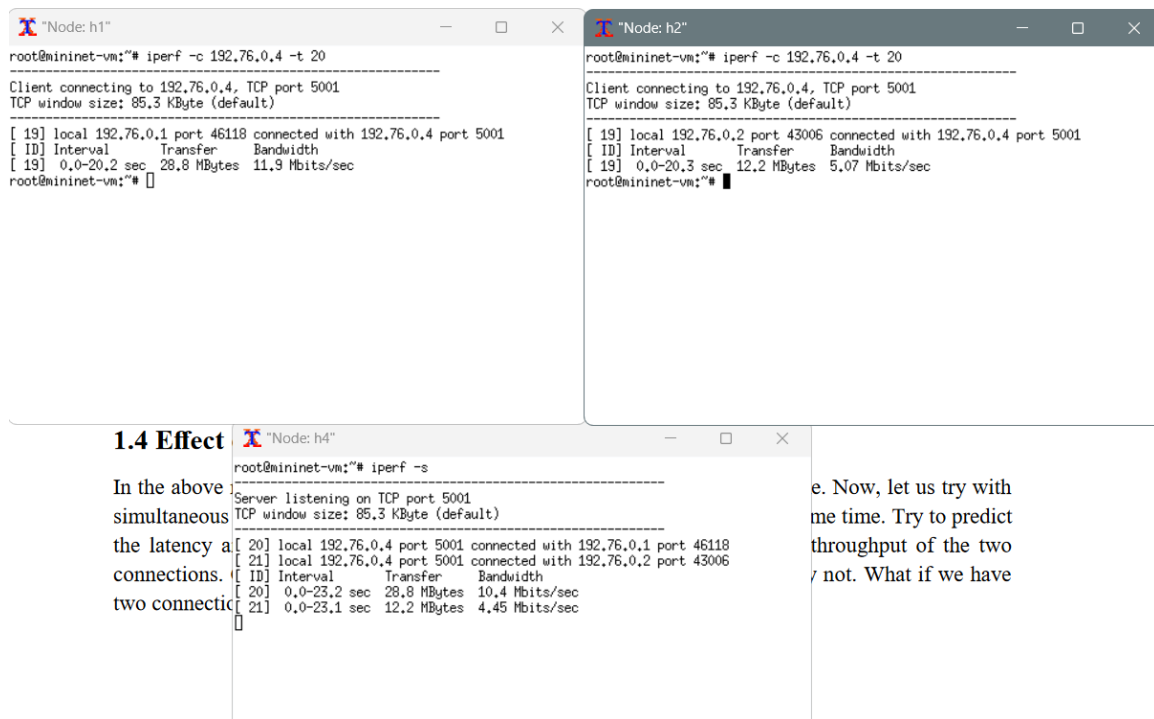


Figure 11 measure throughput on h1 and h2 Xterm window

## Results and Discussion

We measured the performance for both TCP and UDP connection between hosts h1 and h3. As shown in figure 4 and 5, the average round-trip time (RTT) is the delay(latency) in TCP communication between host h1 and h3 is 138.789 msec. while in UDP the delay between host h1 and h3 is 132.670 msec. The term Bandwidth refers to theoretical maximum capacity of the network to transmit data which is 16.1 Mbits/sec in TCP and 1.05 Mbits/sec in UDP.

In the first scenario in part 1.4, the latency (RTT) between h1 and h3 is 132.566 msec, while the latency (RTT) between h2 and h4 is 133.218 msec, as shown in figures 8 & 9. Additionally, the bandwidth between h1 and h3 is 13.3 Mbits/sec, and between h2 and h4 is 8.28 Mbits/sec.

While in the second scenario, we measured the latency and throughput between h1 & h4, h2 & h4 at the same time, the latency between h1 and h4 hosts is 132.712 msec, the latency between h2 and h4 hosts is 133.431 msec. while the bandwidth between h1 and h4 is 11.9 Mbits/sec, and the bandwidth between h2 and h4 is 5.07 Mbits/sec.

To measure the theoretical performance and compare the results with the above two scenarios, we need to find the latency and throughput.

$\text{Total\_delay} = 10 + 45 + 10 = 65 \text{ msec.}$

$\text{Latency} = 2 * \text{Total\_delay} = 2 * 65 = 130 \text{ msec.}$

$\text{Throughput} = \text{min bandwidth} / 2 = 15 / 2 = 7.5 \text{ Mbps.}$

Table 1 shows the comparison between the theoretical and practical performance for the first scenario.

*Table 1 Comparison for the first scenario*

First scenario	Theoretical	Practical
Latency	130 msec	h1&h3 = 132.566 msec h2 & h4 = 133.218 msec
Throughput	7.5 Mbps	h1 & h3=6.65 Mbps h2 & h4 =4.14 Mbps

Table 2 shows the comparison between the theoretical and practical performance for the second scenario.

*Table 2 Comparison for second scenario*

Secound scenario	Theoretical	Practical
Latency	130 msec	h1&h4 = 132.712 msec h2 & h4 = 133.431 msec
Throughput	7.5 Mbps	h1 & h4= 5.95 Mbps h2 & h4 = 2.535 Mbps

Note:

- The above numbers are from the original topology.
- The equations above are learned earlier.
- The latency increases in the practical part for both scenarios.
- The throughput decreases in the practical part for both scenarios.
- In the second scenario the latency is higher than the first scenario's latency.
- The first scenario has a higher throughput than the second scenario.

## Summary

In summary, this report will present the achieved steps in this project involving setting up a customized network topology using Mininet APIs, performance measurements, and multiplexing analysis. It will offer insights into network behavior under different configurations and provide a deeper understanding of network performance principles. After successfully completed this task, a few challenges were encountered during the design, the link properties to match the given specifications required attention to detail and interpreting and predicting the impact of multiplexing on latency and throughput involved.

When multiple hosts send data simultaneously over the same link to different destinations, latency increases and throughput decreases due to several factors. Firstly, contention for bandwidth arises as the hosts compete for transmission opportunities, this leads to delay in data delivery. Additionally, collisions may occur, resulting in corrupted packets and retransmission delays, further exacerbating latency issues. Moreover, the presence of buffering delays occurs when network devices cannot handle the incoming data rate efficiently, leading to packet loss and subsequent retransmissions, which further degrade throughput. Lastly, backpressure mechanisms may be activated to slow down transmission rates to prevent network congestion, which reduces throughput.

When two hosts send data simultaneously to the same destination over a shared link, latency increases and throughput decreases compared to when they send to different destinations. This occurs due to contention and potential congestion at the destination, causing processing overhead, queuing delays, and resource bottlenecks, hindering the efficient handling of incoming data streams as was shown in the cases above.

## References

- [1] [Mininet Walkthrough - Mininet](#)
- [2] <https://mininet.org/>
- [3] <https://mininet.org/api/annotated.html>
- [4] <https://opennetworking.org/mininet/>

## Appendix

### Pyhton code :

```
from mininet.topo import Topo

from mininet.net import Mininet

from mininet.link import TCLink

class CustTop(Topo):

    def build(self):

        # Add switches

        s1 = self.addSwitch('s1')

        s2 = self.addSwitch('s2')

        # Add hosts

        h1 = self.addHost('h1', ip='192.76.0.1/24')

        h2 = self.addHost('h2', ip='192.76.0.2/24')

        h3 = self.addHost('h3', ip='192.76.0.3/24')

        h4 = self.addHost('h4', ip='192.76.0.4/24')

        # Add links with specified properties

        self.addLink(h1, s1, bw=15, delay='10ms')

        self.addLink(h2, s1, bw=15, delay='10ms')
```



```
self.addLink(h3, s2, bw=15, delay='10ms')

self.addLink(h4, s2, bw=15, delay='10ms')

self.addLink(s1, s2, bw=20, delay='45ms')

# Create an instance of the topology

topo=CustTop()

# Create Mininet network with custom topology and link settings

net = Mininet(topo=topo, link=TCLink)

# Start the network

net.start()

# Open Mininet CLI for interaction

net.interact()
```