" بسم الله الرحمن الرحيم "



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 1201176

Section: 1

# 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

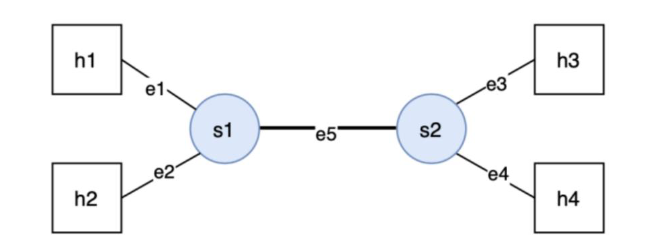
Part 3 guided through the Mininet Command-Line Interface (CLI) commands, starting with the display of options by initiating a minimal topology with `$ sudo mn`. The tutorial introduces the Python interpreter within the Mininet CLI, illustrating its utility for executing Python commands, exploring local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

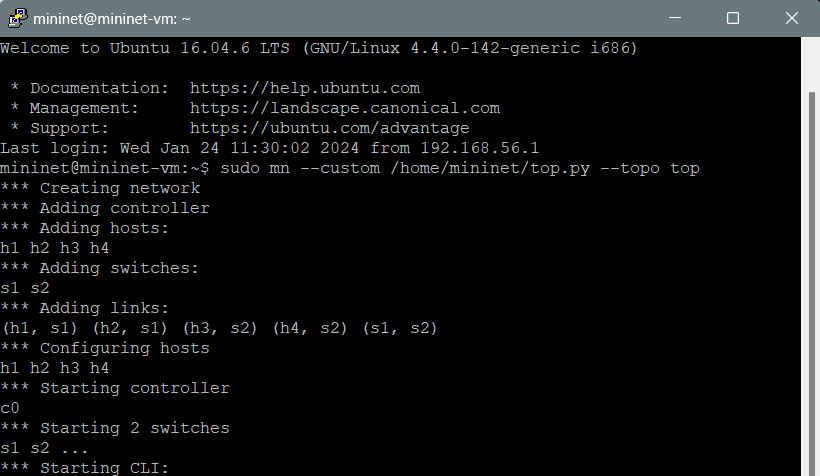
**Objective:** The task entailed constructing a network topology comprising two switches (s1, s2) and four hosts (h1 - h4), interconnected by specified links with designated bandwidth and delay parameters. The IP addresses of the hosts were to be determined based on a predetermined scheme involving student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



Figure

# 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

1. 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**

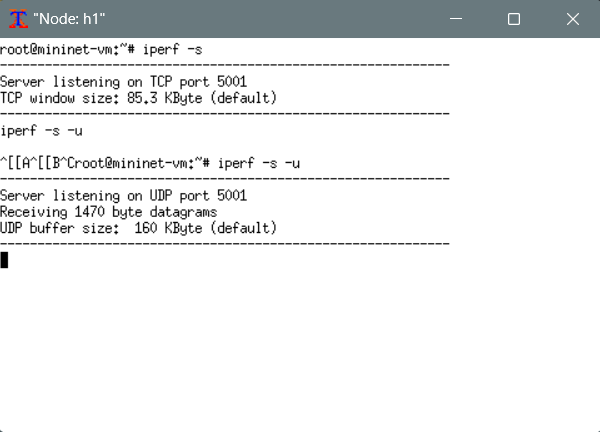


Figure 3

1. In the xterm window for host h3, we measured latency using the following command as shown in figure 4:

**h3$ ping -c 20 192.76.0.1**

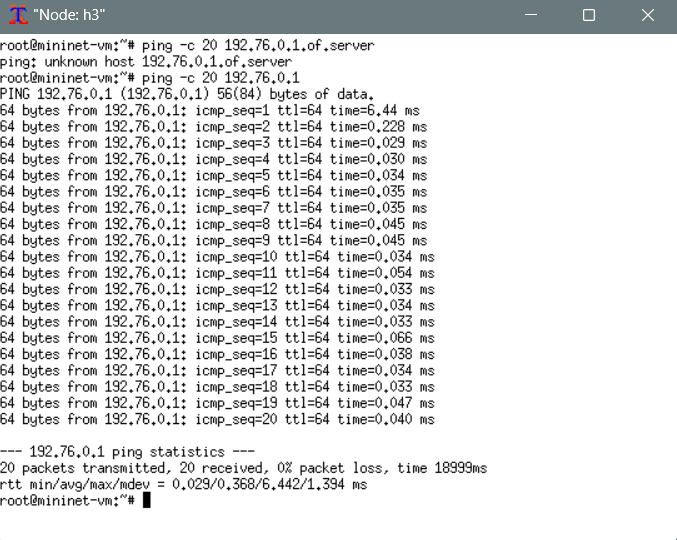


Figure 4

1. 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 5 & 6:

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

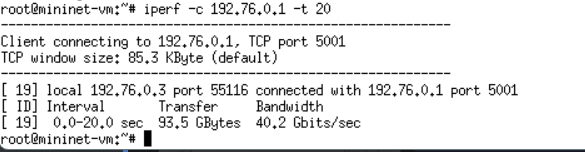


Figure 5

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

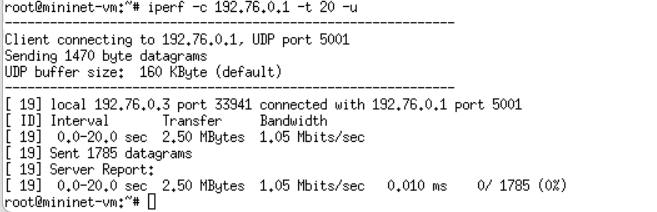


Figure 6

As shown in figure, the round-trip time (RTT) is the delay(latency) in communication between host h1 and h3, the minimum RTT is 0.029 msec, the average RTT is 0.368 msec, the maximum RTT is 6.442 msec, and the standard deviation is 1.394 msec. The term transfer in figures refers to the throughput or actual amount of data transfer rate between h1 and h3 which is 93.5 GBytes in TCP and 1.50 Mbytes in UDP. While the term Bandwidth refers to theoretical maximum capacity of the network to transmit data which is 40.2 Gbits/sec in TCP and 1.05 Mbits/sec in 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**

1. In the xterm window for host h3 and h4, we started iperf in server mode for TCP using the following commands as shown in figure 8.

**H3: h3$ iperf -s**

**H4: h4$ iperf -s**

1. In the xterm window for host h1 and h2, we measured latency using the following command as shown in figure 7:

**H1: h1$ ping -c 20 192.76.0.3**

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

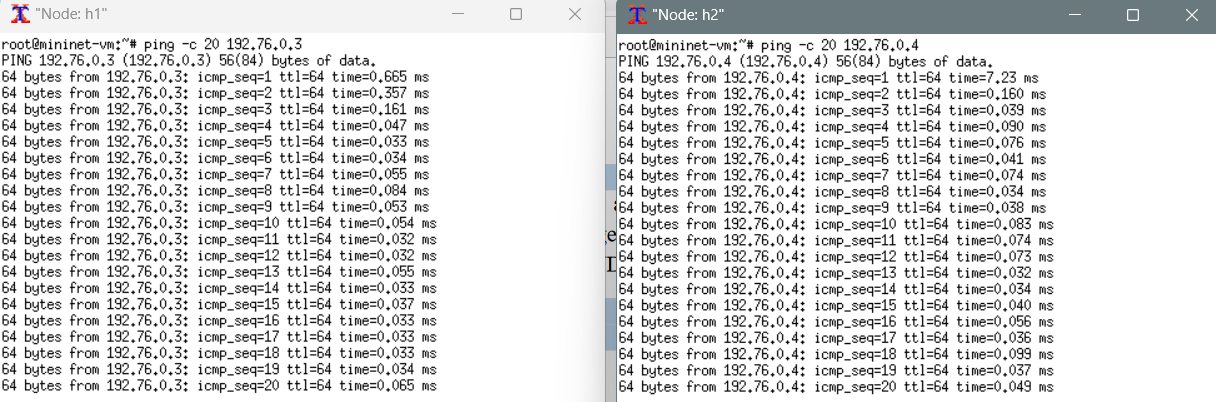


Figure 7

1. Then in the Xterm window of host h1 and h2, we measured throughput using the following command as shown in figure 8:

**H1: h1$ iperf -c 192.76.0.3 -t 20**

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

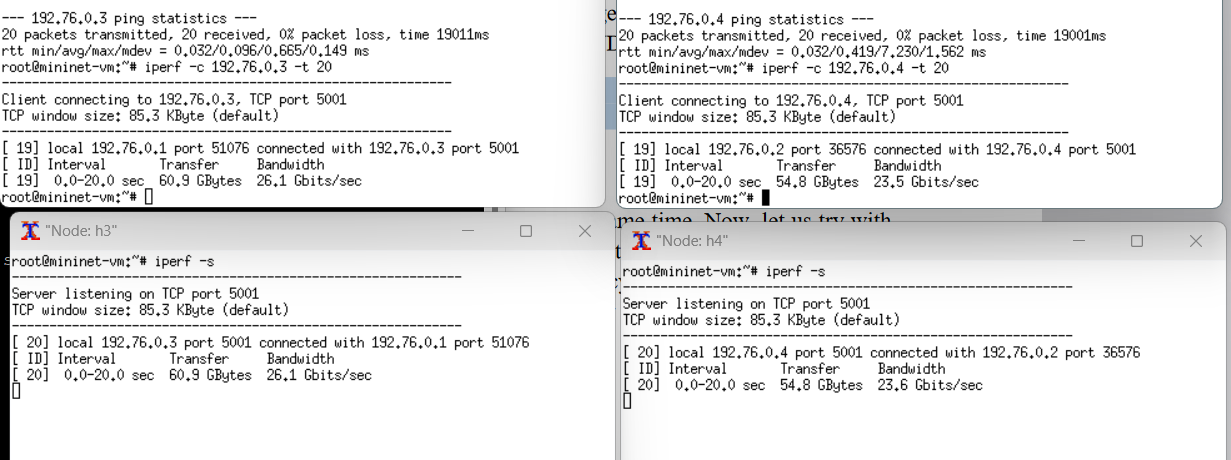
****

Figure 8

**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**

1. In the xterm window for host h4, we started iperf in server mode for TCP using the following commands as shown in figure 10.

**H4: h4$ iperf -s**

1. In the xterm window for host h1 and h2, we measured latency using the following command as shown in figure 9:

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

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

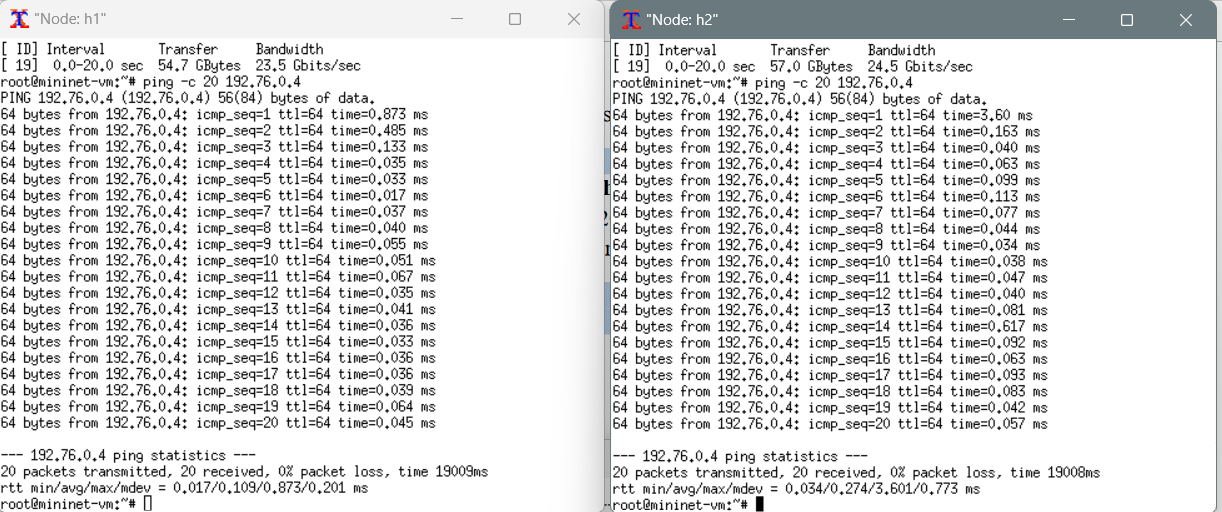
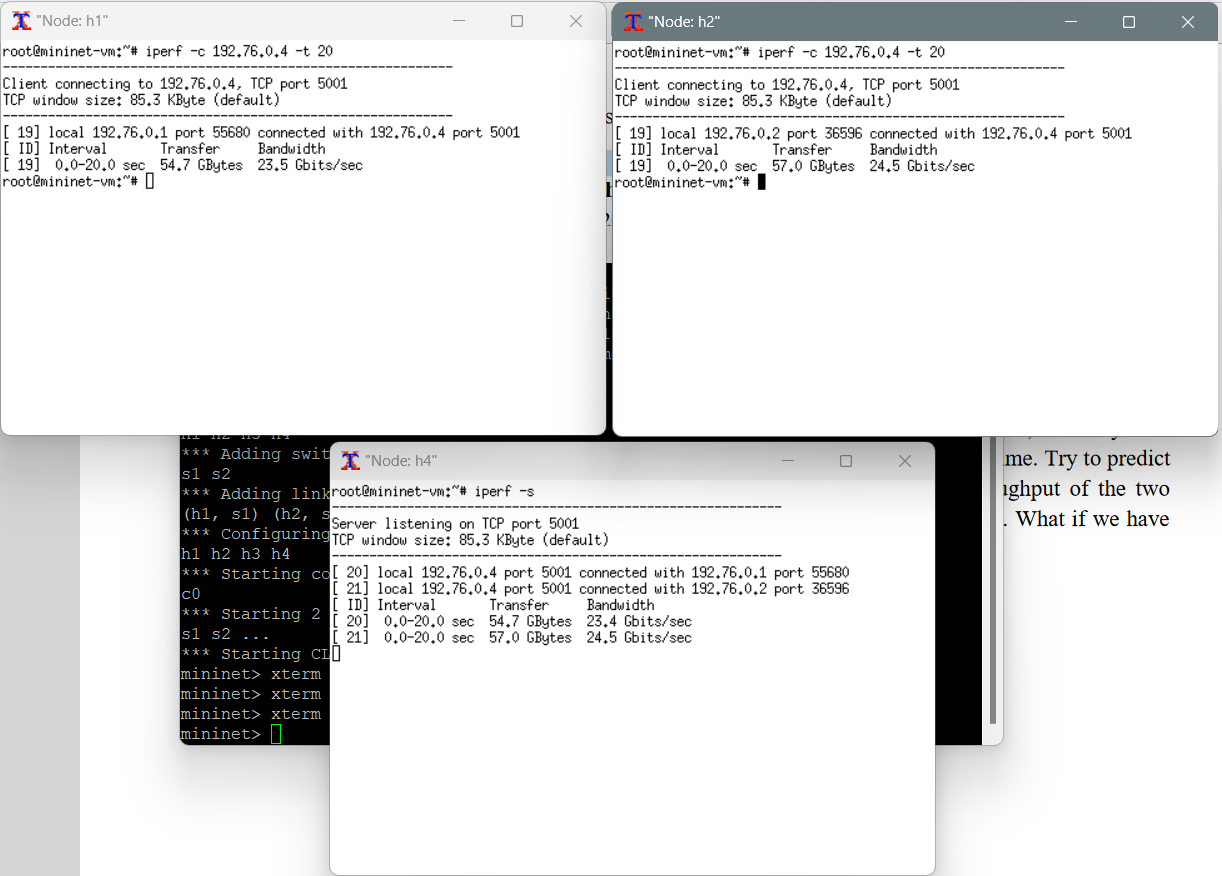
****

Figure 9

1. Then in the Xterm window of host h1 and h2, we measured throughput using the following command as shown in figure 10:

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

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

****

Figure