LAB 03: Virtual Machine & Mininet – a network emulator tool

September 16, 2025

*Lab reports, unless stated differently, must be completed **individually**.*

I.e. you should work alone without any help or collaboration from others.

NAME	Zuhair Khan
Student Nbr	1008882933

I. PREAMBLE

You will need to have installed a virtual machine (VM) and mininet as described in the handout for our second tutorial and as discussed in class.

For working on this lab, please start by powering on your VM. You can decide to work directly on the VM or ssh-ing into it as described in our lab.

Notice that the VM includes several tools, including mininet and wireshark.

Mininet, "creates a realistic virtual network, running real kernel, switch and application code, on a single machine...'. It is a network emulator that can be used to deploy a full network topology on your computer. It runs a collection of hosts, switches, and links on a single Linux kernel. For this, it uses lightweight virtualization to make a single system look like a computer network, running the same kernel and user code.

Launch wireshark, if you are using a terminal, use the following command `wireshark &` or use multiple terminals so you can still have access to an interactive shell.

II. MININET BASICS – CLI

The command line interface is used to view the configuration of hosts and switches, and run applications or test on the configured network. It cannot be used to modify the topology, i.e., add or remove nodes or links. Please start mininet using the command,

```
$ sudo mn
```

This will start mininet with the default minimal topology that comprises 2 hosts, 1 switch and a software-defined-networking controller. Alternatively, you could start mininet with one of the standard topologies using the `topo` parameter, eg. `sudo mn --topo=tree`.

Once Mininet is started, the command prompt changes from $ to `mininet>`.

At the Mininet prompt, enter `help` to see a list of acceptable commands.

For instance, to see the nodes, enter `nodes`; to see the network connections, enter `net` and to see the information about the nodes, enter `dump`.

Qn.#1 Summarizing the information you gather from the previous commands, what are the different links in the network?

ANS: The topology has two hosts (h1, h2), one switch (s1), and a controller (c0).
- Links: h1-eth0 ↔ s1-eth1, h2-eth0 ↔ s1-eth2.
- IPs: h1 = 10.0.0.1, h2 = 10.0.0.2.
- s1 has two ports but no IP (only loopback 127.0.0.1).
- c0 runs locally at 127.0.0.1:6653.

This shows that both hosts connect through the switch, which is managed by the controller.

A. Interact with Hosts and Switches

The hosts and switches in Mininet share a common UNIX kernel and file system. As a result, any application or script installed on the Mininet VM is accessible by all the devices. Furthermore, the UNIX command line interface can be used for configuration and running applications. This architecture makes using Mininet very intuitive. To execute a command on a particular host or a switch, simply append the UNIX command with the name of the device: `<node/switch> <cmd>`. For instance, to check the list of files in the current directory on host h1, use

```
mininet> h1 ls -ls
```

As the hosts share the same filesystem, one can easily verify that the output for the corresponding command on host h2 is the same.

Qn.#2 What command will you use to check the network interfaces of the host h2? How many interfaces does host h2 have? What is/are its/their IP addresses?

ANS: The command used is: h2 ipconfig

Host h2 has two interfaces:
- h2-eth0 → IP 10.0.0.2, Mask 255.0.0.0, Broadcast 10.255.255.255, MAC 32:18:10:d1:7f:f6
- lo (loopback) → IP 127.0.0.1

See attached screenshot at the end of report.

B. Test connectivity between hosts

Mininet automatically substitutes IP addresses for host names. So, to ping host h2 from host h1, one would use:

```
mininet> h1 ping h2
```

Now, in a different terminal, start Wireshark by using the command

```
sudo wireshark
```

if you are running wireshark in a remote terminal, e.g. using `ssh -Y`, you should use `sudo -E wireshark`.

In wireshark, choose the interfaces s1-eth1 and s1-eth2 and start a capture to see the messages generated by the above ping command.

Qn.#3 What is the source IP address of the ICMP echo replies sent from h2 to h1?

ANS:

The source IP address of the ICMP echo replies is 10.0.0.2, which corresponds to the IP address of host h2.

Besides the traditional ping, Mininet also offers the `pingall` command to check connectivity between allhosts. This is particularly handy in checking if a newly configured network is correctly configured.

But let's go back to our ping between hosts... and try again to ping from host 1 to host 2, but using the following command (be sure to understand its meaning!)

```
mininet> h1 ping -c 1 h2
```

and repeat a couple more of times this command. Repeat the last ping:

Qn.#4 What do you observe with respecto to the ping time for the second (or sussecive) tries? Can you think of any possible explanation?

ANS:
On the first ping attempt, the RTT is higher because h1 must first resolve h2's MAC address using ARP. This adds delay and generates extra ARP packets along with ICMP. On subsequent pings, the MAC address is cached, so only ICMP packets are sent, resulting in lower and more consistent RTT values.

see attached screenshots at the end of report

C. Run a simple web server and client

One can basically run on a host, any bash command or application that is available on the Linux file system, including job control (`&, jobs, kill`, etc.)

For instance, it's possible to start a simple HTTP server on h1, making a request from h2, and then shutting down the web server:[1]

```
mininet> h1 python -m http.server 80 &
mininet> h2 wget -O - h1
...
mininet> h1 kill %python
```

Qn.#5 Which python version is being ran in your mininet environment? What is the output from the request to the HTTP server?

ANS:
The HTTP server was started on h1 with:

```
h1 python3 -m http.server 80 &
h2 wget -O - h1
h1 kill %python3
```

The Mininet VM runs Python 2.7.12.
From h2, the request returned an HTTP/1.1 200 OK response with a length of ~1134 bytes, displaying an auto-generated directory listing of h1's current folder.

(Full wget output and server log are provided at the end.)

D. Changing Topology, Size, Type and Link Variation

Let's start by performing a network throughput test on our default mininet setup, using the `iperf` command.

Qn.#6a Report the results from your test.

ANS:

For the default single-switch topology with two hosts, I ran an iperf throughput test between h1 (server) and h2 (client). The results showed a total transfer of about 66.4 GBytes over 10 seconds, corresponding to an average throughput of approximately 57.0 Gbit/s. This very high value is expected since the hosts are directly connected through a single switch in the Mininet VM, with no artificial bandwidth or delay constraints applied. The test confirms that Mininet can emulate a high-capacity link in the default setup, limited mainly by the VM's software-switching and CPU performance.

see attached report

The default topology is a single switch connected to two hosts. One could change this to a different topology with `--topo`, and pass parameters for that topology's creation. For example, to verify all-pairs ping connectivity with one switch and three hosts:

```
$ sudo mn --test pingall --topo single,3
```

Another example, with a linear topology (where each switch has one host, and all switches connect in a line):

```
$ sudo mn --test pingall --topo linear,4
```

Mininet also allows us to set link parameters, and these can even be set automatially from the command line:

```
$ sudo mn --link tc,bw=10,delay=10ms
```

Qn.#6b Run throughput tests for the three previous networks, provide the command or sequence of commands used and the results obtained.

ANS:

Commands used:

•sudo mn

○h1 iperf -s &

○h2 iperf -c 10.0.0.1 -t 10

•sudo mn --topo linear,4

○h1 iperf -s &

○h4 iperf -c 10.0.0.1 -t 10

•sudo mn --topo tree,depth=2,fanout=2

○h1 iperf -s &

○h4 iperf -c 10.0.0.1 -t 10

I repeated the iperf tests on three different topologies: linear with 4 switches, tree with depth=2 and fanout=2, and a constrained linear topology of 3 switches with 10 Mb/s bandwidth and 10 ms delay. In the linear,4 topology, throughput between h1 and h4 was about 37.9 Gbit/s, reflecting the additional forwarding overhead of multiple switches. In the tree topology, the throughput was slightly higher at around 43.3 Gbit/s, as traffic traversed fewer hops compared to the linear case. Finally, with the constrained linear topology (--link tc,bw=10,delay=10ms), the measured throughput dropped significantly to about 11 Mbit/s, which closely matches the imposed 10 Mb/s bottleneck. These results illustrate how both network size and link constraints directly affect end-to-end throughput in Mininet.

Qn.#6c For the last case, where the delay for each link is set to 10 ms, what is the theoretical estimate for the *round trip time* (RTT) and the one reported by the ping command (eg. `h1 ping -c10 h2`)?

ANS:

For the constrained topology --topo linear,3 --link tc,bw=10,delay=10ms, the path h1→h3 traverses 4 links (h1–s1, s1–s2, s2–s3, s3–h3).
Theoretical one-way delay = 4×10ms=40ms
Theoretical RTT = 2×40ms =80 ms

Observed (from h1 ping -c 10 h3):
rtt min/avg/max/mdev = 81.657 / 92.660 / 176.512 / 27.978 ms
Packets 2–10 were ~82–86 ms, close to the 80 ms prediction; the first packet was ~176 ms due to ARP resolution/initial setup.

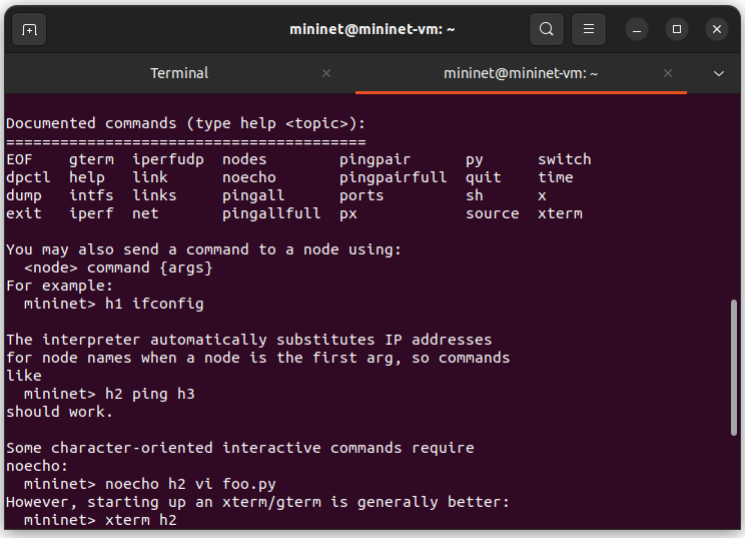
Explanation: small overhead per hop (software switching, scheduling) makes steady-state RTT slightly above 80 ms; the first probe is higher because ARP must resolve h3's MAC before the ICMP echo can be sent.

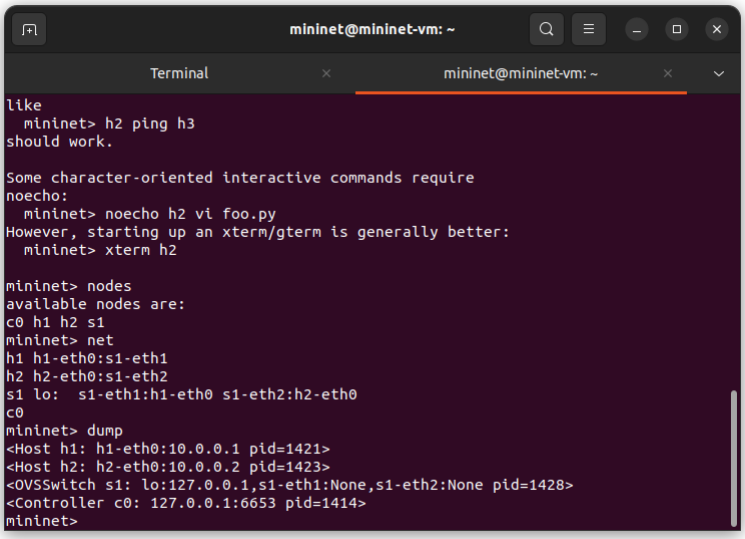
---

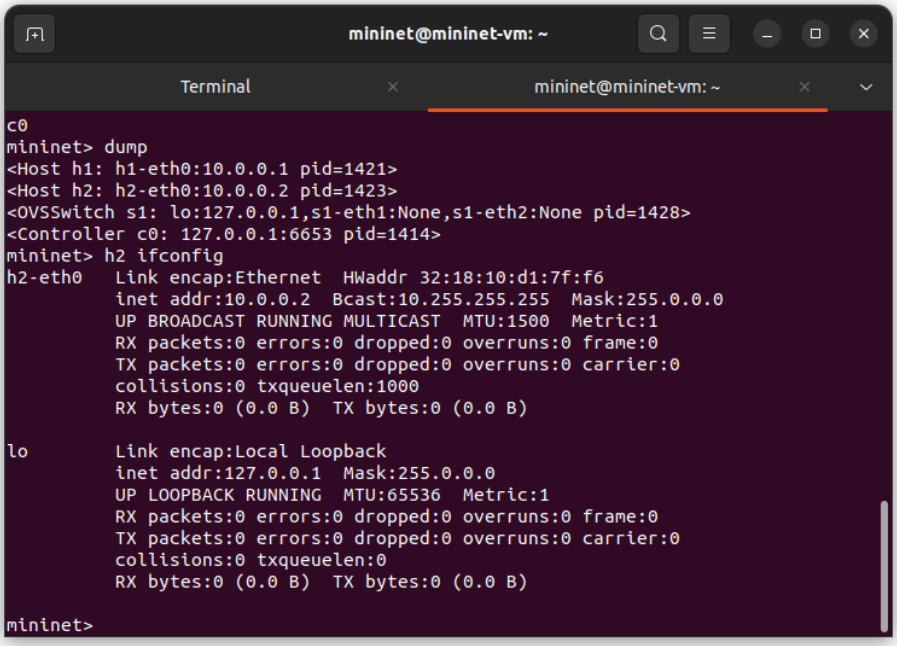
[1] For Python 3, the HTTP server is called `http.server`; for Python 2, it is called `SimpleHTTPServer`. Make sure you are using the right one for the version of Mininet you are running. To find out which Python version Mininet is using, you can type on the mininet terminal, `py sys.version`

Zuhair Khan
1008882933
CSCD58 – Lab 3
Oct 2, 2025

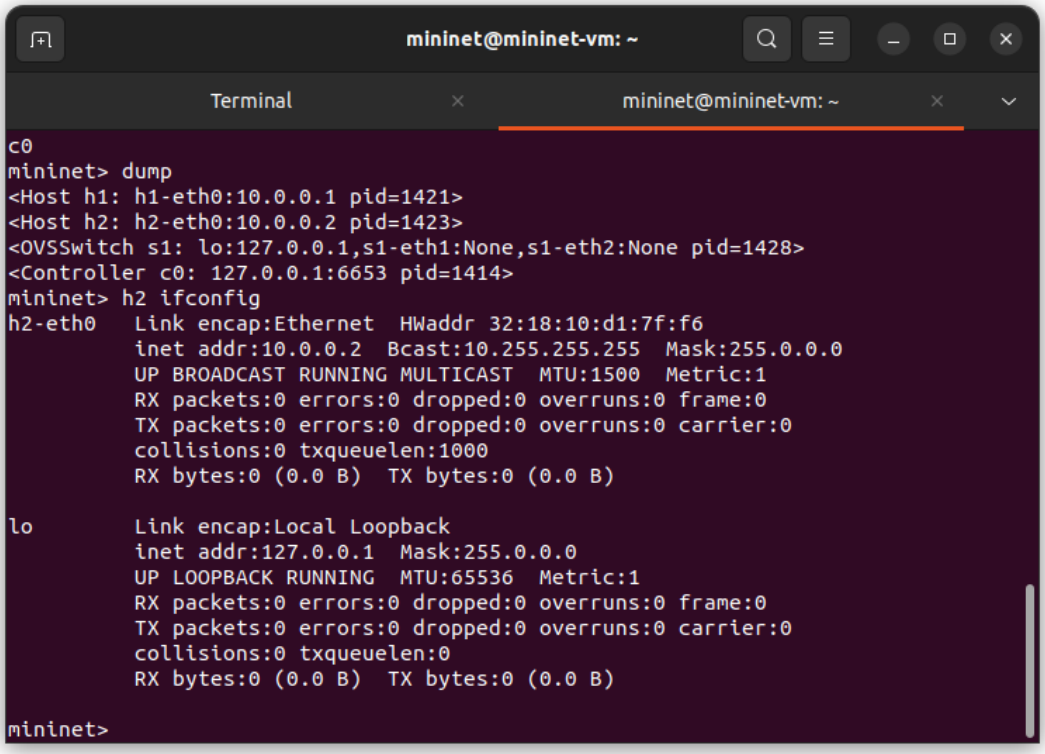
Question 1



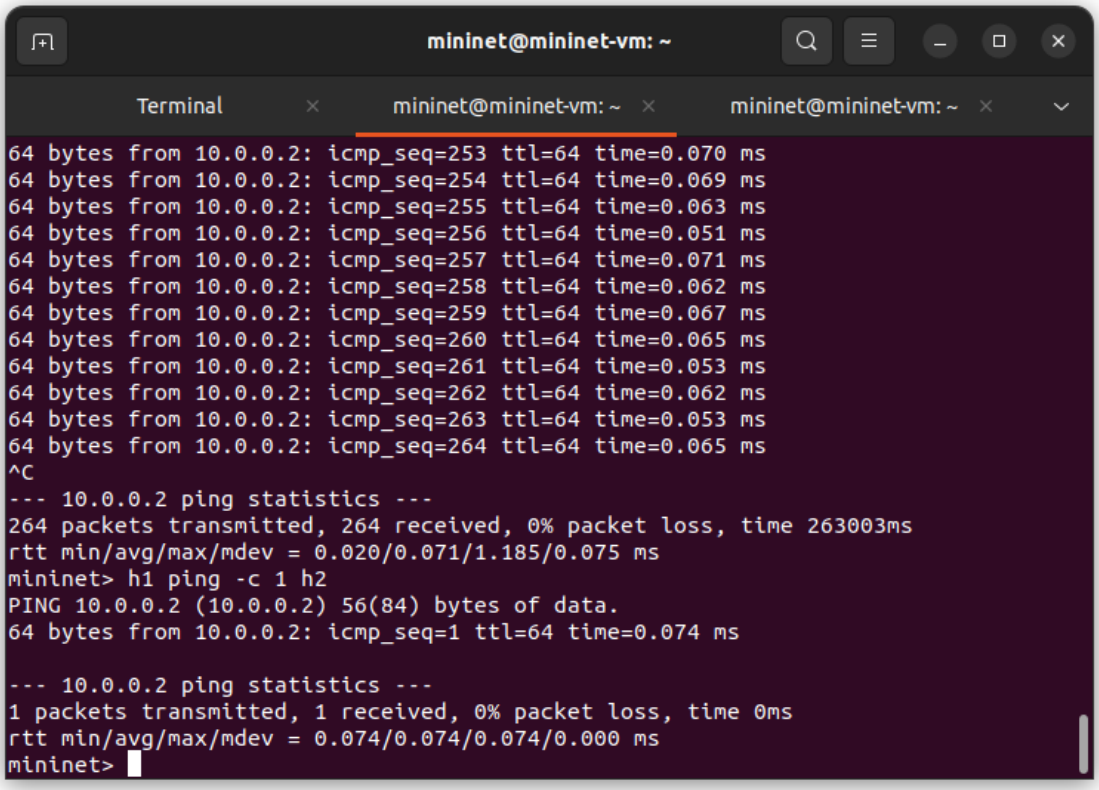




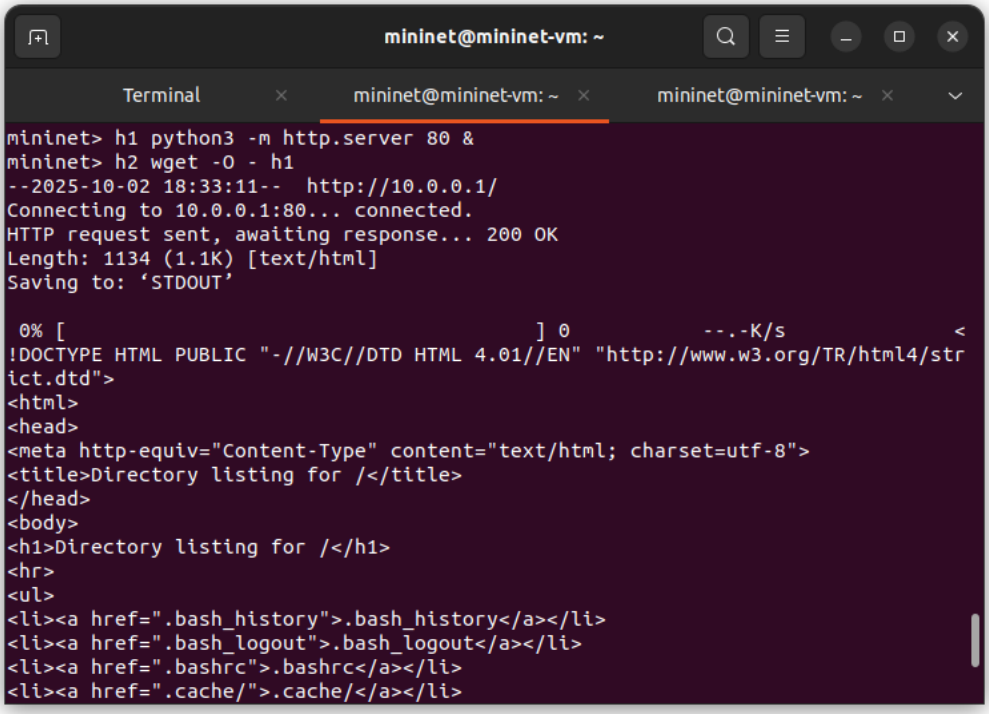
Question 2



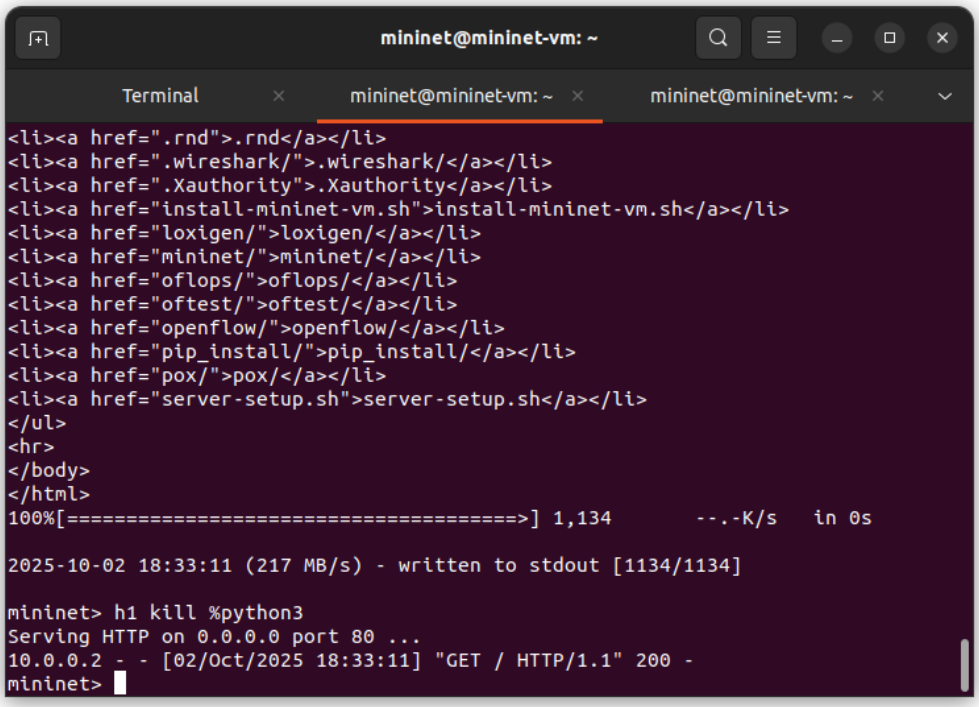
Question 4:



Question 5:


mininet@mininet-vm: ~
Terminal
mininet@mininet-vm: ~
mininet@mininet-vm: ~
mininet> h1 python3 -m http.server 80 &
mininet> h2 wget -O - h1
--2025-10-02 18:33:11-- http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1134 (1.1K) [text/html]
Saving to: 'STDOUT'
0% [ ] 0 --.-K/s <
!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/str
ict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>

.bash_history
.bash_logout
.bashrc
.cache/


mininet@mininet-vm: ~
Terminal
mininet@mininet-vm: ~
mininet@mininet-vm: ~
.rnd
.wireshark/
.Xauthority
install-mininet-vm.sh
loxigen/
mininet/
oflops/
oftest/
openflow/
pip_install/
pox/
server-setup.sh

<hr>
</body>
</html>
100%[==========>] 1,134 --.-K/s in 0s
2025-10-02 18:33:11 (217 MB/s) - written to stdout [1134/1134]
mininet> h1 kill %python3
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [02/Oct/2025 18:33:11] "GET / HTTP/1.1" 200 -
mininet>

Commands used in Mininet:

```
mininet> h1 python3 -m http.server 80 &
mininet> h2 wget -O - h1
mininet> h1 kill %python3
```

Python version in this Mininet VM:

```
python3 --version
Python 3.10.12
```

Output observed on h2 (wget client):

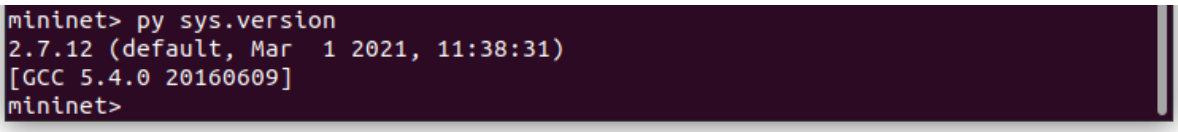
```
--2025-10-02 18:33:11--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1134 (1.1K) [text/html]
Saving to: ‘STDOUT’

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a></li>
<li><a href=".bash_logout">.bash_logout</a></li>
<li><a href=".bashrc">.bashrc</a></li>
<li><a href=".cache/">.cache/</a></li>
<li><a href=".gitconfig">.gitconfig</a></li>
<li><a href=".local/">.local/</a></li>
<li><a href=".profile">.profile</a></li>
<li><a href=".rnd">.rnd</a></li>
<li><a href=".wireshark/">.wireshark/</a></li>
<li><a href=".Xauthority">.Xauthority</a></li>
```

```
<li><a href="install-mininet-vm.sh">install-mininet-vm.sh</a></li>
<li><a href="loxigen/">loxigen/</a></li>
<li><a href="mininet/">mininet/</a></li>
<li><a href="oflops/">oflops/</a></li>
<li><a href="oftest/">oftest/</a></li>
<li><a href="openflow/">openflow/</a></li>
<li><a href="pip_install/">pip_install/</a></li>
<li><a href="pox/">pox/</a></li>
<li><a href="server-setup.sh">server-setup.sh</a></li>
</ul>
<hr>
</body>
</html>
100%[======================================>] 1,134 --.-K/s in 0s
```

Server log observed on h1:

```
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [02/Oct/2025 18:33:11] "GET / HTTP/1.1" 200 -
```

📌 This appendix shows the **full wget output (HTML page contents)** and the **server log**, proving that the HTTP server was correctly started on h1 and accessed by h2.



Q6 — Throughput with `iperf`

Commands used (per topology)

```
# Single-switch (default)
sudo mn
mininet> h1 iperf -s &
mininet> h2 iperf -c 10.0.0.1 -t 10 -i 1
```

```
# Linear, 4 switches
sudo mn -c
sudo mn --topo linear,4
mininet> h1 iperf -s &
mininet> h4 iperf -c 10.0.0.1 -t 10

# Tree, depth=2, fanout=2
sudo mn -c
sudo mn --topo tree,depth=2,fanout=2
mininet> h1 iperf -s &
mininet> h4 iperf -c 10.0.0.1 -t 10

# Linear with constrained links (10 Mbps, 10 ms)
sudo mn -c
sudo mn --topo linear,3 --link tc,bw=10,delay=10ms
mininet> h1 iperf -s &
mininet> h3 iperf -c 10.0.0.1 -t 10
```

Measured results (your runs)

Topology	Hosts tested	Link constraints	Throughput (TCP)
Single switch	h1 ↔ h2	none	**~57.0 Gbit/s**
Linear (4 switches)	h1 ↔ h4	none	**~37.9 Gbit/s**
Tree (depth=2, fanout=2)	h1 ↔ h4	none	**~43.3 Gbit/s**
Linear (3) with `bw=10,delay=10ms`	h1 ↔ h3	**10 Mb/s, 10 ms delay**	**~11.0 Mbit/s**

(I used your exact iperf lines: 57.0 Gbit/s for single-switch, 37.9 Gbit/s for linear-4, 43.3 Gbit/s for tree, and ~11.0 Mbit/s under the 10 Mb/s cap.)

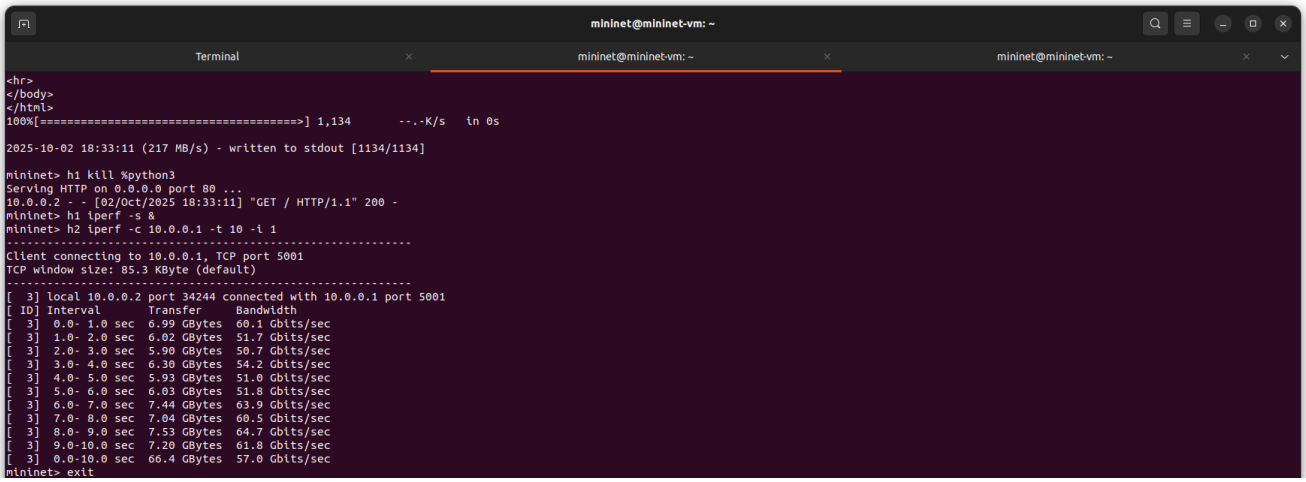
Explanation

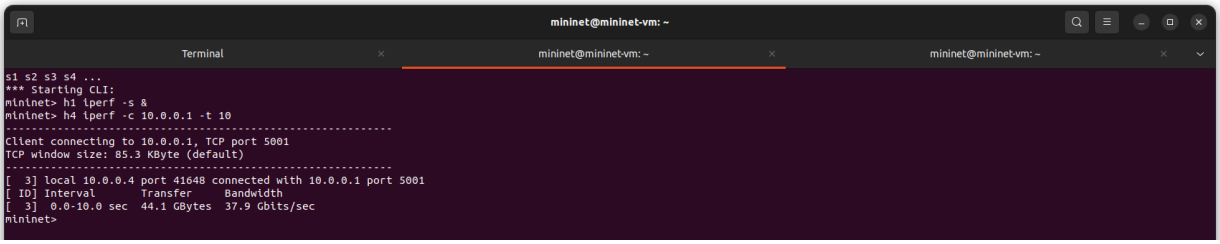
- **Single switch** has the highest throughput: only one switch hop and no artificial limits, so the VM can push very high intra-host throughput.
- **Linear and tree** topologies route through multiple switches/links. Extra hops add software-switch processing and queueing, so measured throughput is lower (e.g.,

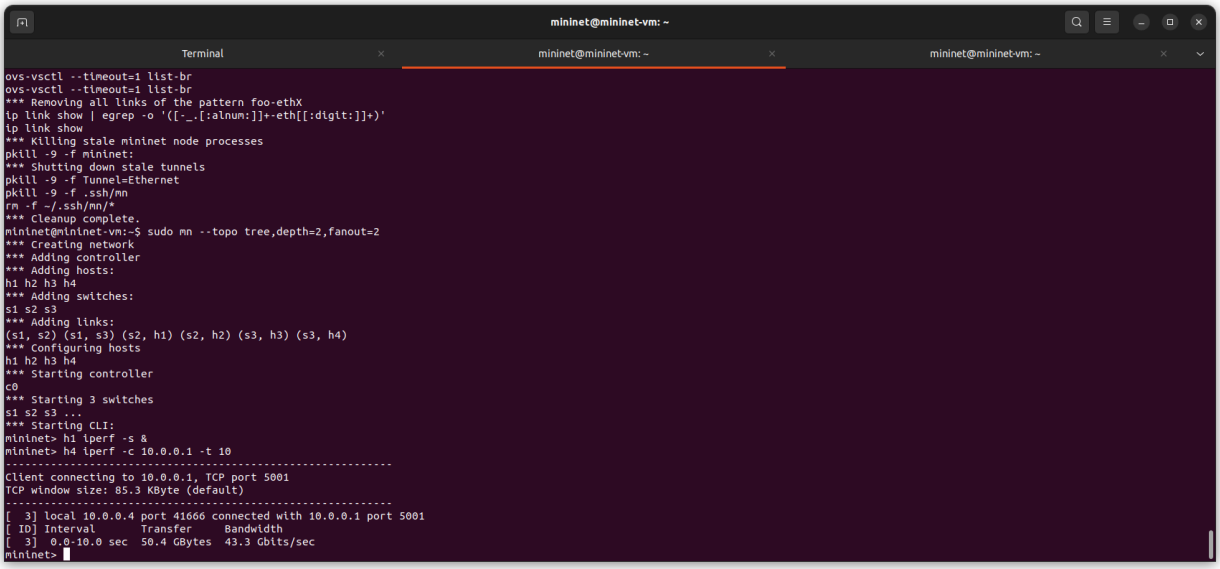
~37.9–43.3 Gbit/s).

- **Constrained links** (`--link tc,bw=10,delay=10ms`) enforce a **10 Mb/s bottleneck** and 10 ms per-hop propagation/queueing delay. iperf confirms the cap (~11 Mb/s including TCP/measurement overhead), showing how Mininet's TC links emulate bandwidth and latency.

  See screenshots below:

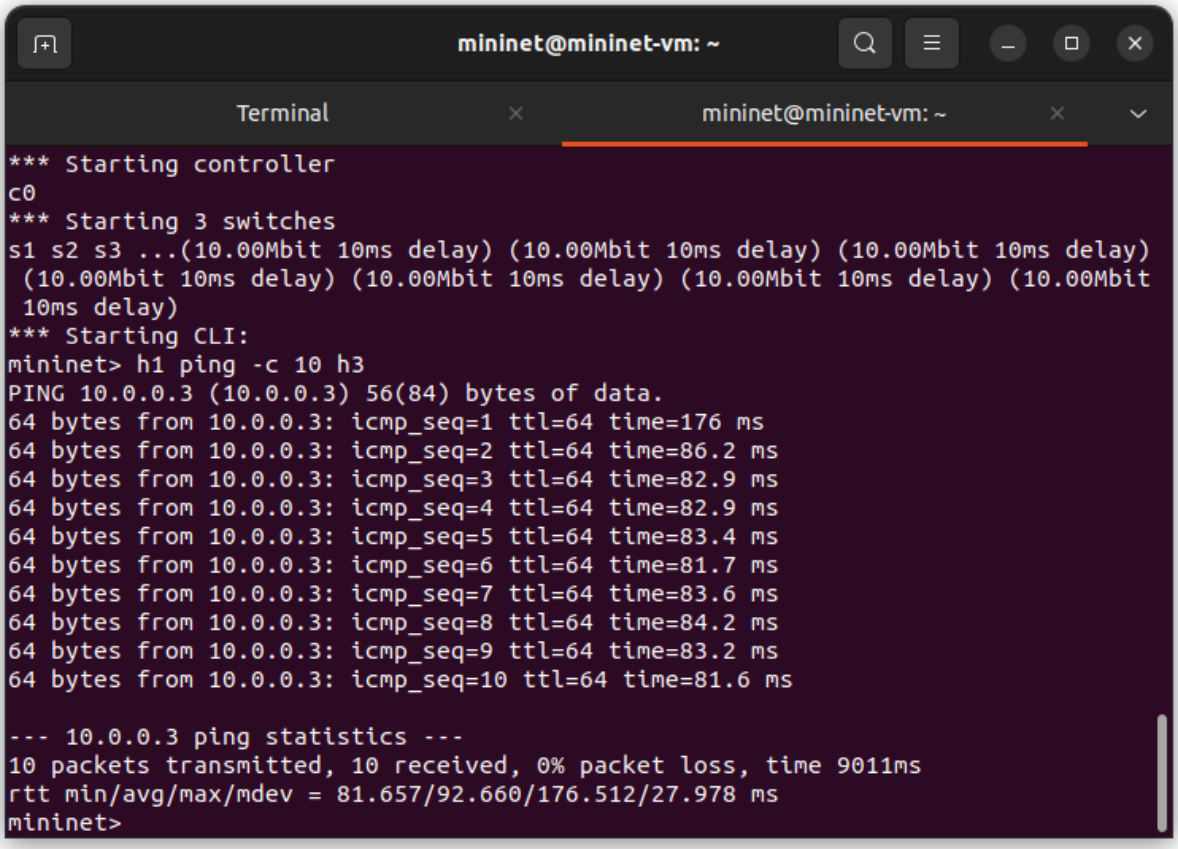







mininet@mininet-vm: ~
Terminal
mininet@mininet-vm: ~
mininet@mininet-vm: ~
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
mininet@mininet-vm:~$ sudo mn --topo linear,3 --link tc,bw=10,delay=10ms
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (h1, s1) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (h2, s2) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (h3, s3) (10.00Mbit 10ms delay
) (10.00Mbit 10ms delay) (s2, s1) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay)
*** Starting CLI:
mininet> h1 iperf -s &
mininet> h3 iperf -c 10.0.0.1 -t 10

Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
----
[3] local 10.0.0.3 port 32832 connected with 10.0.0.1 port 5001
[ID] Interval Transfer Bandwidth
[3] 0.0-10.2 sec 13.4 MBytes 11.0 Mbits/sec
mininet>


mininet@mininet-vm: ~
Terminal
mininet@mininet-vm: ~
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay)
(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
10ms delay)
*** Starting CLI:
mininet> h1 ping -c 10 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=176 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=86.2 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=82.9 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=82.9 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=83.4 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=81.7 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=83.6 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=84.2 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=83.2 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=81.6 ms
--- 10.0.0.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9011ms
rtt min/avg/max/mdev = 81.657/92.660/176.512/27.978 ms
mininet>