

Lab Report – Mininet and OpenFlow

Boubacar DIALLO

October 6, 2025

Contents

1	Prerequisites	2
2	Useful Commands	2
3	Part A: Setting up the Virtual Machine	2
3.1	Importing the VM	2
3.2	Initial Access	2
3.3	Adding a Graphical Interface	2
4	Part B: Experimenting with Mininet	2
4.1	Displaying Help	2
4.1.1	Initial Traffic Observation	3
5	Part B.1 – Interacting with Hosts and Switches	3
5.1	Basic Topology	3
5.1.1	Exploring the Topology	5
5.1.2	Visible Processes	6
5.1.3	Connectivity Test	7
5.2	Topologies <code>single,3</code> and <code>linear,4</code>	8
5.3	Custom Topology with Controller	8
5.4	POX Controller	9
5.5	Controller Results	9
5.6	Fixed MAC Addresses	9
5.7	Link Parameter Variation (Part B.5)	10
5.8	Topology Code B.5	11
5.9	B.6 – Verbosity Levels and Xterms	11
5.10	B.7 – Python Interpreter	11
5.11	Reading Port Counters with <code>ovs-ofctl</code>	12
6	References	12

1 Prerequisites

- A host machine running **VMware Workstation**.
- The **Mininet 2.3.0 (minimal)** image downloaded from GitHub: <https://github.com/mininet/mininet/releases/tag/2.3.0>
- Internet connection to install additional packages.
- **VMware** for virtualization.
- **Overleaf** for writing and formatting this report.

2 Useful Commands

- `Ctrl + L` : clear the terminal.

3 Part A: Setting up the Virtual Machine

3.1 Importing the VM

- Import the Mininet OVA image into VMware.
- Configure the network mode: either NAT or Bridged depending on your setup.

3.2 Initial Access

- Default credentials: `mininet / mininet`.
- Access is provided through the text-based shell console.

3.3 Adding a Graphical Interface

```
sudo apt-get update && sudo apt-get upgrade -y
sudo apt-get install -y xinit lxde open-vm-tools open-vm-tools-desktop
```

After reboot, the LXDE desktop environment becomes available.

4 Part B: Experimenting with Mininet

4.1 Displaying Help

The following command displays Mininet's startup options (topology, controller, switch type, tests, verbosity, etc.):

```
sudo mn -h
```

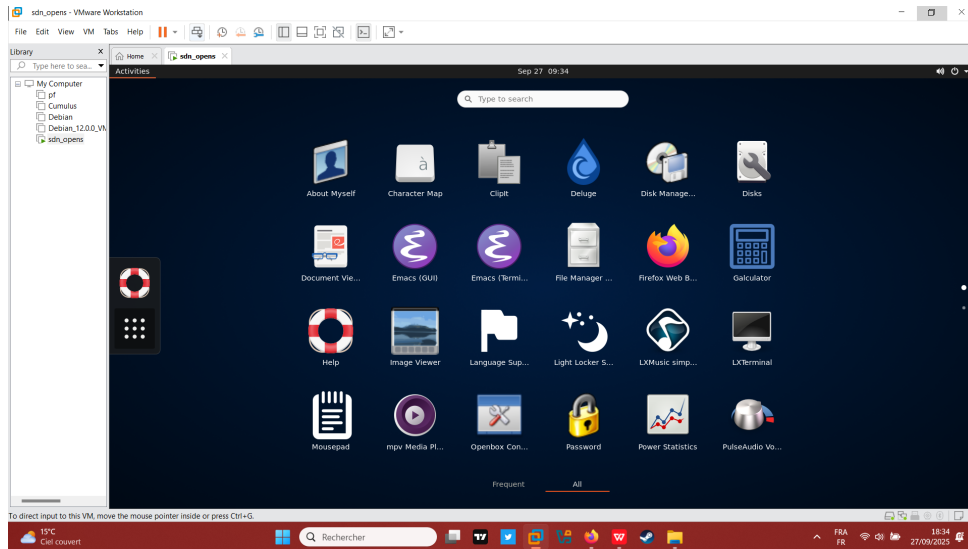


Figure 1: Mininet environment with LXDE desktop on VMware

4.1.1 Initial Traffic Observation

No OpenFlow packets appear before Mininet starts. However, other network protocols are visible:

- **ARP**: Resolves IP to MAC addresses.
- **NTP**: Synchronizes time with a time server.
- **mDNS**: Local service discovery protocol.

These packets come from the host system, not Mininet itself.

5 Part B.1 – Interacting with Hosts and Switches

5.1 Basic Topology

```
sudo mn
```

```

Home x sdn_opens x
Activities LXTerminal v
File Edit Tabs Help
mininet@mininet-vm:~$ sudo mn -h
usage: mn [options]
type mn -h for details)

The mn utility creates Mininet network from the command line. It can create
parametrized topologies, invoke the Mininet CLI, and run tests.

Options:
-h, --help                show this help message and exit
--switch=SWITCH           default|ivs|lxbr|ovs|ovsbr|ovsk|user[,param=value...]
                        user=UserSwitch ovs=OVSSwitch ovsbr=OVSBridge
                        ovsk=OVSSwitch ivs=IVSSwitch lxbr=LinuxBridge
                        default=OVSSwitch
--host=HOST               cfs|proc|rt[,param=value...] proc=Host
                        rt=CPULimitedHost{'sched': 'rt'}
                        cfs=CPULimitedHost{'sched': 'cfs'}
--controller=CONTROLLER  default|none|nox|ovsc|ref|remote|ryu[,param=value...]
                        ref=Controller ovsc=OVSController nox=NOX
                        remote=RemoteController ryu=Ryu
                        default=DefaultController none=NULLController
--link=LINK               default|ovs|tc|tcu[,param=value...] default=Link
                        tc=TCLink tcu=TCULink ovs=OVSLink
--topo=TOPO               linear|minimal|reversed|single|torus|tree[,param=value
                        ...] minimal=MinimalTopo linear=LinearTopo
                        reversed=SingleSwitchReversedTopo
                        single=SingleSwitchTopo tree=TreeTopo torus=TorusTopo
-c, --clean              clean and exit
--custom=CUSTOM          read custom classes or params from .py file(s)
--test=TEST              pingall|pingpair|iperf|iperfudp|all|none|build
-X, --xterms             spawn xterms for each node
-i IPBASE, --ibase=IPBASE
                        base IP address for hosts
--mac                    automatically set host MACs
--arp                    set all-pairs ARP entries
-v VERBOSITY, --verbosity=VERBOSITY
                        debug|info|output|warning|warn|error|critical
--innamespace            sw and ctrl in namespace?
--listenport=LISTENPORT
                        base port for passive switch listening
--nolistenport           don't use passive listening port
--pre=PRE                CLI script to run before tests
--post=POST              CLI script to run after tests
--pin                    pin hosts to CPU cores (requires --host cfs or --host
                        rt)
--nat                    [option=val...] adds a NAT to the topology that

```

Figure 2: Mininet help options

```

mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

```

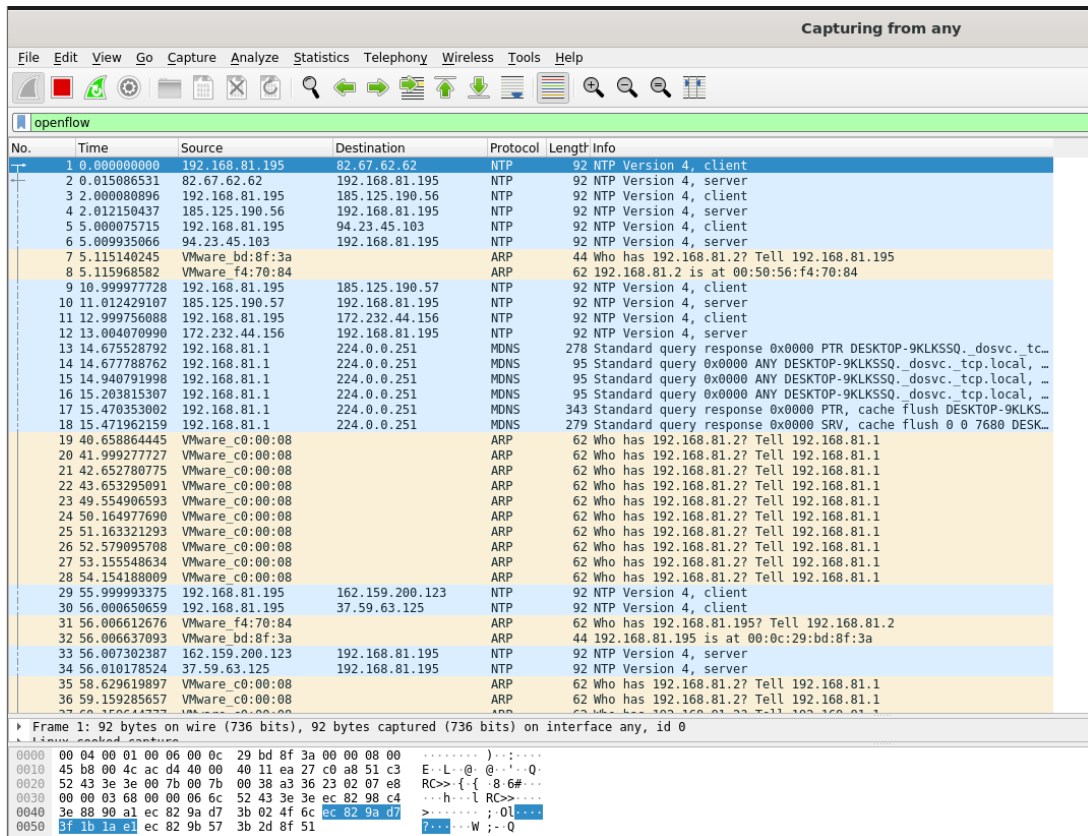


Figure 3: Wireshark capture filtered with openflow

5.1.1 Exploring the Topology

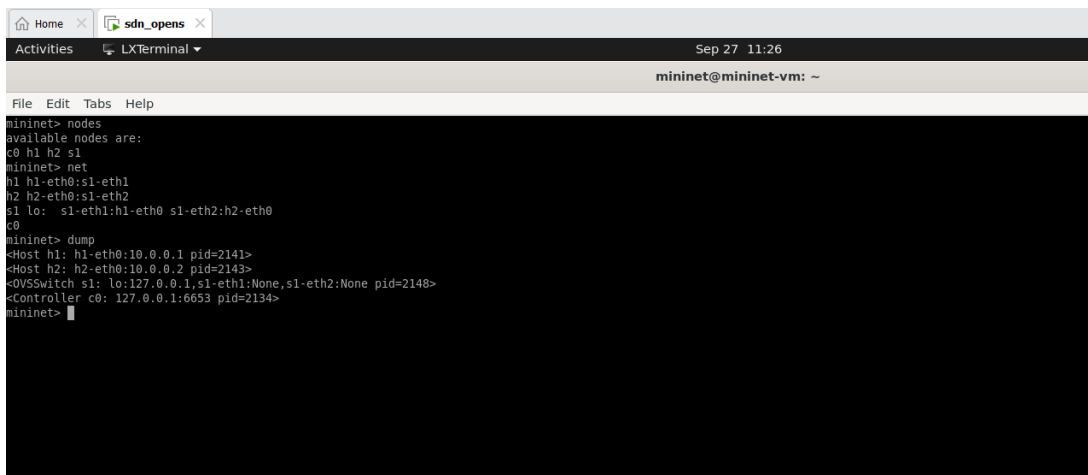
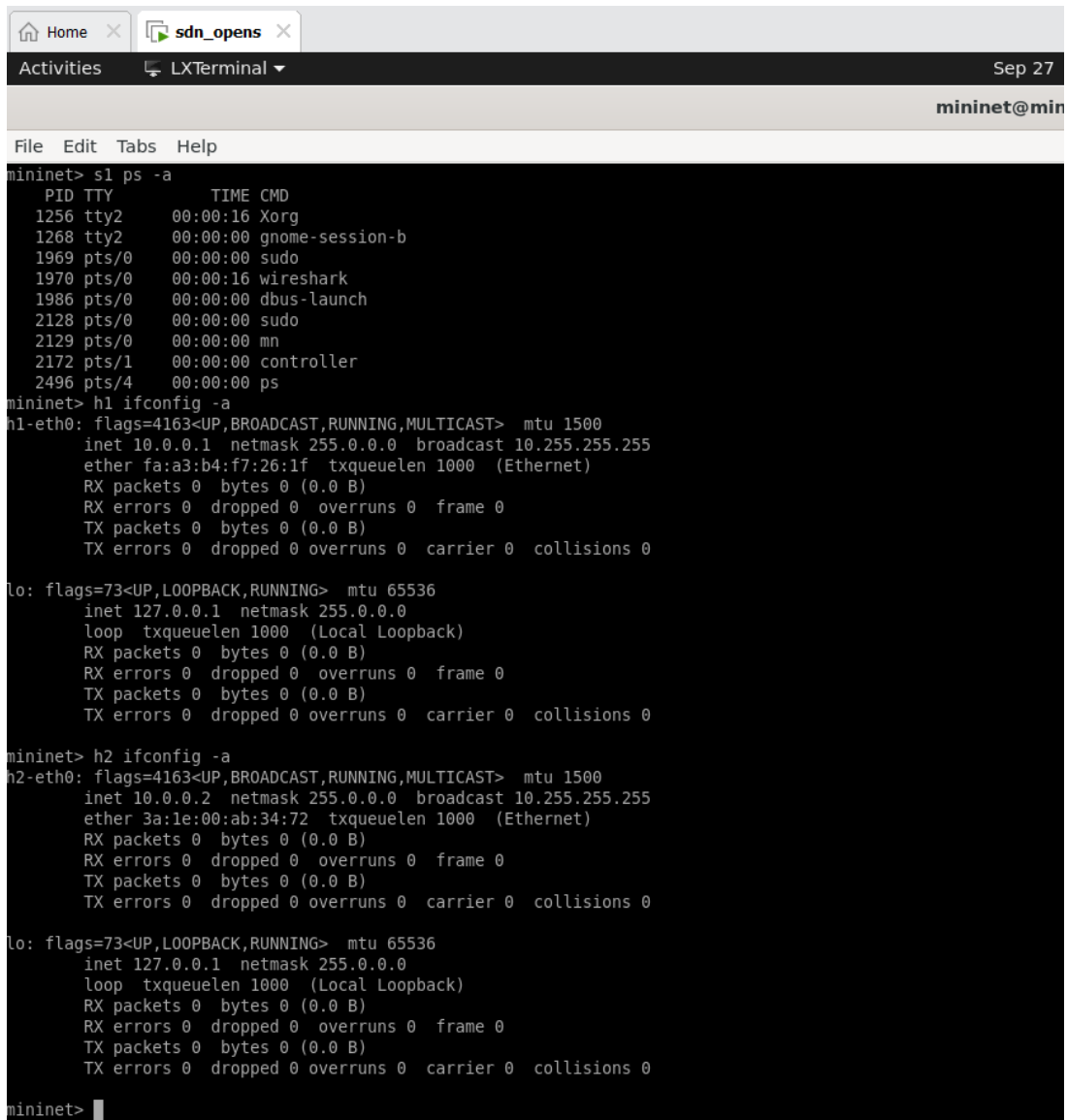


Figure 5: Results of nodes, net, and dump commands in Mininet

nodes lists all entities (hosts, switches, controller). **net** displays the links between nodes. **dump** provides details about each node (interfaces, IP, MAC). Each host has its own isolated network namespace.

5.1.2 Visible Processes



```
mininet> s1 ps -a
  PID TTY          TIME CMD
 1256 tty2      00:00:16 Xorg
 1268 tty2      00:00:00 gnome-session-b
1969 pts/0      00:00:00 sudo
1970 pts/0      00:00:16 wireshark
1986 pts/0      00:00:00 dbus-launch
2128 pts/0      00:00:00 sudo
2129 pts/0      00:00:00 mn
2172 pts/1      00:00:00 controller
2496 pts/4      00:00:00 ps

mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
        ether fa:a3:b4:f7:26:1f txqueuelen 1000 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

mininet> h2 ifconfig -a
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.2 netmask 255.0.0.0 broadcast 10.255.255.255
        ether 3a:1e:00:ab:34:72 txqueuelen 1000 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

mininet> █
```

Figure 6: Results of `ps -a` from `h1` and `s1`

The same process list appears in both `h1` and `s1`, meaning that only the network is virtualized; processes and the filesystem are shared among all nodes.

5.1.3 Connectivity Test

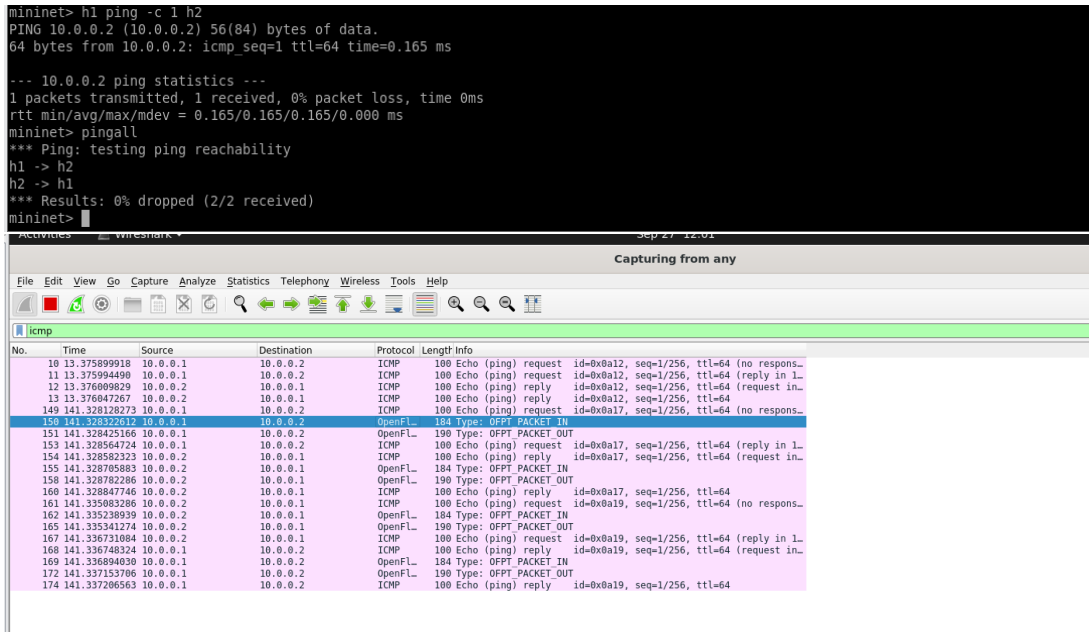


Figure 7: Connectivity test using ping and pingall

The first ping is slower because the controller installs OpenFlow rules (Packet_In / Packet_Out). The second one is faster since flows are already in place. pingall confirms full connectivity between hosts.

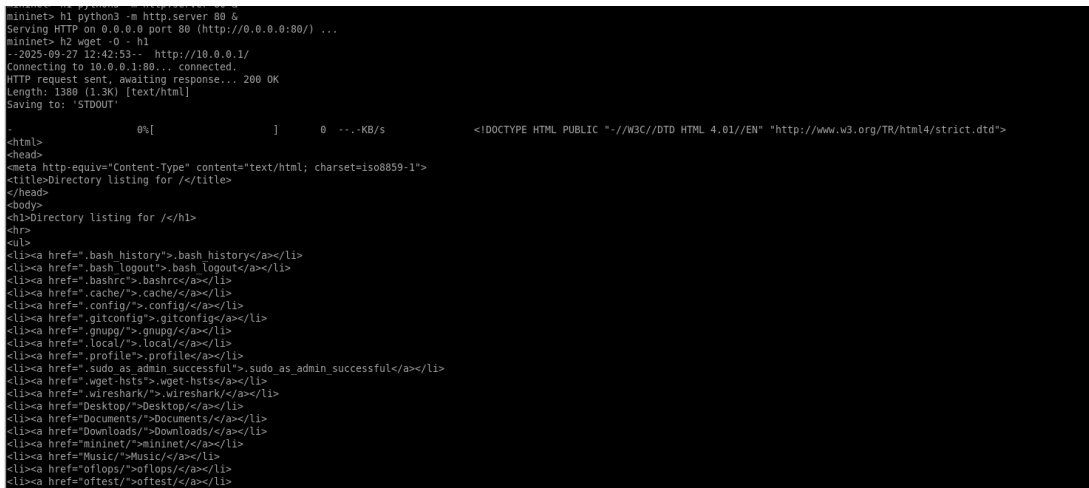


Figure 8: Application test: HTTP server on h1 and request from h2

This demonstrates that Mininet hosts can run real Linux applications. A simple HTTP server was launched on h1 (python3 -m http.server 80), and h2 retrieved data with wget. The server was stopped using h1 kill %python.

5.2 Topologies single,3 and linear,4

```
mininet@mininet-vm:~$ sudo mn --test pingall --topo single,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 1 switches
s1
*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 6.314 seconds
mininet@mininet-vm:~$
```

Figure 9: Single switch topology with 3 hosts (single,3)

```
mininet@mininet-vm:~$ sudo mn --test pingall --topo linear,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Waiting for switches to connect
s1 s2 s3 s4
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 1 controllers
c0
*** Stopping 7 links
...
*** Stopping 4 switches
s1 s2 s3 s4
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 7.124 seconds
mininet@mininet-vm:~$
```

Figure 10: Linear topology with 4 switches (linear,4)

Mininet's built-in topologies include `minimal`, `single,n`, `linear,n`, and `tree,depth,fanout`. With the `-test pingall` option, Mininet creates the topology, runs the test, and automatically cleans up.

5.3 Custom Topology with Controller

This topology defines hosts (`h3`, `h4`, `h5`) connected to two switches (`s1`, `s2`) linked together and managed by a controller.

```

mininet@...  A  mininet@...  A
GNU nano 4.8                                     custom3.py
from mininet.topo import Topo

class MyTopo(Topo):
    def build(self):
        # Hôtes avec IP selon le schéma
        h3 = self.addHost('h3', ip='10.0.1.2/24')
        h4 = self.addHost('h4', ip='10.0.1.3/24')
        h5 = self.addHost('h5', ip='10.0.2.2/24')

        # Switches (L2, pas d'IP nécessaire)
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')

        # Liens conformes au schéma
        self.addLink(h3, s1)
        self.addLink(h4, s1)
        self.addLink(s1, s2)
        self.addLink(h5, s2)

# La topologie à mn --custom
topos = { 'mytopo': (Lambda: MyTopo()) }

```

Figure 11: Custom topology script custom.py

5.4 POX Controller

The following command starts the POX controller with layer-2 learning and OpenFlow 1.0 support:

```
./pox.py log.level --DEBUG forwarding.l2_learning openflow.of_01 --port=6633
```

- `log.level -DEBUG`: enables detailed logs.
- `forwarding.l2_learning`: *simplelearningswitchbehavior*.
- `openflow.of_01`: *OpenFlow1.0protocolsupport*.
- `-port=6633`: TCP port used by switches to connect.

The controller installs flow rules automatically, allowing communication between hosts (e.g. h3 ping h4).

5.5 Controller Results

```

mininet> h3 ping h4
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data:
64 bytes from 10.0.1.3: icmp_seq=1 ttl=64 time=15.0 ms
64 bytes from 10.0.1.3: icmp_seq=2 ttl=64 time=0.708 ms
64 bytes from 10.0.1.3: icmp_seq=3 ttl=64 time=0.099 ms
64 bytes from 10.0.1.3: icmp_seq=4 ttl=64 time=0.135 ms
64 bytes from 10.0.1.3: icmp_seq=5 ttl=64 time=0.140 ms
64 bytes from 10.0.1.3: icmp_seq=6 ttl=64 time=0.143 ms
64 bytes from 10.0.1.3: icmp_seq=7 ttl=64 time=0.098 ms
64 bytes from 10.0.1.3: icmp_seq=8 ttl=64 time=0.100 ms
64 bytes from 10.0.1.3: icmp_seq=9 ttl=64 time=0.097 ms
64 bytes from 10.0.1.3: icmp_seq=10 ttl=64 time=0.099 ms
64 bytes from 10.0.1.3: icmp_seq=11 ttl=64 time=0.104 ms
64 bytes from 10.0.1.3: icmp_seq=12 ttl=64 time=0.104 ms
64 bytes from 10.0.1.3: icmp_seq=13 ttl=64 time=0.134 ms
^C
-- 10.0.1.3 ping statistics --

```

Figure 12: Successful ping between h3 and h4 under POX controller

5.6 Fixed MAC Addresses

By default, hosts receive random MAC addresses, which complicates debugging. The `-mac` option assigns deterministic addresses (e.g., h1 = 00:00:00:00:00:01, h2 = 00:00:00:00:00:02).

```

Completed in 19.1726 seconds
mininet@mininet-vm:~$ sudo mn --mac
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
    ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 13: Deterministic MAC addresses with `--mac`

5.7 Link Parameter Variation (Part B.5)

The `b5` topology includes two hosts connected to a switch with constraints: `h1` limited to 20% CPU and `h2` connected via a 10 Mbps link with 50 ms delay.

```

mininet@mininet-vm:~$ sudo mn --custom topob5.py --topo b5 --link tc --host cfs
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (10.00Mbit 50ms delay) (10.00Mbit 50ms delay) (h2, s1)
*** Configuring hosts
h1 (cfs 20000/100000us) h2 (cfs -1/100000us)
*** Starting controller
c0
*** Starting 1 switches
s1 ..(10.00Mbit 50ms delay)
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet> dump
<CPULimitedHost{'sched': 'cfs'} h1: h1-eth0:10.0.0.1 pid=2706>
<CPULimitedHost{'sched': 'cfs'} h2: h2-eth0:10.0.0.2 pid=2713>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=2720>
<Controller c0: 127.0.0.1:6653 pid=2699>
mininet> h1 iperf -s &
mininet> h2 iperf -c 10.0.0.1
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 3] local 10.0.0.2 port 55512 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-0.1 sec   14.8 MBytes 12.3 Mbits/sec

```

Figure 14: `iperf` result: bandwidth 10 Mbps

Bandwidth is limited to 10 Mbps, and the 50 ms link delay results in an RTT around 100 ms.

```

mininet> mininet> h1 ping -c10 h2
*** Unknown command: mininet> h1 ping -c10 h2
mininet> h1 ping -c10 h2
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 4] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 55512
[ ID] Interval      Transfer    Bandwidth
[ 4] 0.0-13.2 sec  14.8 MBytes  9.40 Mbits/sec
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=101 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=102 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=100 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 100.227/100.610/101.681/0.421 ms
mininet>

```

Figure 15: ping result: average RTT 100 ms

5.8 Topology Code B.5

```

-----
GNU nano 4.8 topob5.py
from mininet.topo import Topo
from mininet.link import TCLink

class B5Topo(Topo):
    def build(self):
        h1 = self.addHost('h1', ip='10.0.0.1/24', cpu=0.2)
        h2 = self.addHost('h2', ip='10.0.0.2/24')

        s1 = self.addSwitch('s1')

        self.addLink(h1, s1)

        self.addLink(h2, s1, cls=TCLink, bw=10, delay='50ms')

topos = {'b5': (lambda: B5Topo())}

```

Figure 16: Python script topob5.py

5.9 B.6 – Verbosity Levels and Xterms

```

sudo mn -v debug    # detailed output
sudo mn -v output  # CLI output only
sudo mn -x         # open xterm for each host and switch

```

Before the ping, the flow table is empty. After communication starts, new entries appear.

5.10 B.7 – Python Interpreter

Commands starting with py are evaluated as Python code:

```

sudo mn
mininet> py 'hello ' + 'world'
mininet> py locals()
mininet> py dir(s1)
mininet> py help(h1)
mininet> py h1.IP()

```

```

"switch: s1" (root) x
root@mininet-vm:/home/mininet# ovs-ofctl dump-flows s1
root@mininet-vm:/home/mininet# ovs-ofctl dump-flows s1
cookie=0x0, duration=6.278s, table=0, n_packets=0, n_bytes=0, idle_timeout=60,
priority=65535,arp,in_port="s1-eth2",vlan_tci=0x0000,dl_src=52:3e:da:3d:92:0c,dl
dst=92:57:81:db:1a:94,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2 actions=output
"s1-eth1"
cookie=0x0, duration=1.225s, table=0, n_packets=0, n_bytes=0, idle_timeout=60,
priority=65535,arp,in_port="s1-eth2",vlan_tci=0x0000,dl_src=52:3e:da:3d:92:0c,dl
dst=92:57:81:db:1a:94,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=1 actions=output
"s1-eth1"
cookie=0x0, duration=1.224s, table=0, n_packets=0, n_bytes=0, idle_timeout=60,
priority=65535,arp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=92:57:81:db:1a:94,dl
dst=52:3e:da:3d:92:0c,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,arp_op=2 actions=output
"s1-eth2"
cookie=0x0, duration=6.277s, table=0, n_packets=5, n_bytes=490, idle_timeout=60
, priority=65535,icmp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=92:57:81:db:1a:94
,dl_dst=52:3e:da:3d:92:0c,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,i
cmp_code=0 actions=output:"s1-eth2"
cookie=0x0, duration=6.277s, table=0, n_packets=5, n_bytes=490, idle_timeout=60
, priority=65535,icmp,in_port="s1-eth2",vlan_tci=0x0000,dl_src=52:3e:da:3d:92:0c
,dl_dst=92:57:81:db:1a:94,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,i
cmp_code=0 actions=output:"s1-eth1"
root@mininet-vm:/home/mininet#

"host: h1" x
root@mininet-vm:/home/mininet# 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=1.78 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.763 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.115 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.128 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.070 ms
--- 10.0.0.2 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11229ms
rtt min/avg/max/mdev = 0.069/0.283/1.784/0.487 ms
root@mininet-vm:/home/mininet#

```

Figure 17: Flow table of s1 before and after pinging between h1 and h2

```

mininet> py 'hello + 'world'
hello world
mininet> py locals
{'built-in function locals':
mininet> py locals(s1)
{'net': <mininet.net.Mininet object at 0x7eff4d20be80>, 'h1': <Host h1: h1-eth0:10.0.0.1 pid=4691>, 'h2': <Host h2: h2-eth0:10.0.0.2 pid=4693>, 's1': <OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=4689>
Bs', 'c0': <Controller c0: 127.0.0.1:6653 pid=4684>}
mininet> py dir(s1)
['_IP', 'MAC', 'OVVersion', 'CReapply', 'class', 'delattr', 'dict', 'dir', 'doc', 'eq', 'format', 'ge', 'getattr', 'getattribute', 'gt', 'hash', 'init', 'init_subclass', 'le', 'lt', 'module', 'ne', 'new', 'reduce', 'reduce_ex', 'repr', 'setattr', 'sizeof', 'str', 'subclasshook', 'weakref', 'weakref__popen', 'uids', 'addIntrf', 'argmax', 'attach', 'batch', 'batchshutdown', 'batchstartup', 'bridgeports', 'cleanup', 'cmd', 'cmdPrntty', 'cmds', 'commands', 'config', 'configdefault', 'connected', 'connections0', 'controlIntf', 'controller
UIDs', 'datapath', 'decoder', 'defaultIntf', 'defaultIntf', 'delattr', 'deleteIntfs', 'detach', 'dpctl', 'dpid', 'dpidlen', 'exceed', 'failMode', 'fotoNode', 'imageName', 'inbnd', 'intf', 'intfslup
', 'intfList', 'intfNames', 'intfOpts', 'intfs', 'isOldOVS', 'isSetup', 'lastCmd', 'lastPid', 'linkTo', 'listenPort', 'master', 'monitor', 'mountPrivateDirs', 'name', 'nameIntf', 'newPort', 'opts', 'outToNode',
'params', 'pexec', 'pid', 'pollOut', 'popen', 'portBase', 'ports', 'privateDirs', 'protocols', 'read', 'readbuf', 'readline', 'reconnects', 'sendCmd', 'sendIntf', 'setARP', 'setDefaultRoute', 'setHostRoute', 'setI
P', 'setMAC', 'setParam', 'setup', 'shell', 'slave', 'start', 'startShell', 'stdin', 'stdout', 'stop', 'stp', 'terminate', 'umountPrivateDirs', 'vscil', 'waitExited', 'waitOutput', 'waitReadable', 'waiting', 'wr
ite']
mininet> py h1.IP()
10.0.0.1
mininet> py h1.MAC()
9e:9b:c6:e7:7b:46
mininet> py h1.config()
mininet> py h1.cmd('ifconfig -a')
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
ether 9e:9b:c6:e7:7b:46 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
loop txqueuelen 1000 (local loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
mininet>

```

Figure 18: Examples of Python evaluation inside Mininet CLI

5.11 Reading Port Counters with ovs-ofctl

ovs-ofctl dump-ports s1 shows activity on each switch port:

- **LOCAL**: internal management interface (drops here are normal).
- **s1-eth1, s1-eth2**: host interfaces. RX/TX counters increase when traffic flows.

Verification procedure:

```

mininet> net
mininet> h1 ping -c3 10.0.0.2
$ sudo ovs-ofctl dump-ports s1

```

Simulated failure:

```

mininet> link s1 h1 down
mininet> pingall
$ sudo ovs-ofctl dump-ports s1
mininet> link s1 h1 up

```

6 References

- Mininet releases: <https://github.com/mininet/mininet/releases>

- Official Mininet documentation: <http://mininet.org/>
- VMware Tools guide: <https://kb.vmware.com/s/article/1022525>
- Mininet custom topologies (GitHub): <https://github.com/mininet/mininet/tree/master/custom>
- Mininet API documentation: <http://mininet.org/api/annotated.html>