

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЗВІТ

з лабораторної роботи №2
з дисципліни ”Програмування комп’ютерних та віртуальних мереж”

Тема: Створення простої SDN мережі

Варіант №5

Виконав:
Студент 1 курсу, групи ІМ-51мн
Ковальов Олександр

Перевірів:
доцент, Долголенко Олександр Миколайович

Дата здачі: 16.10.2025

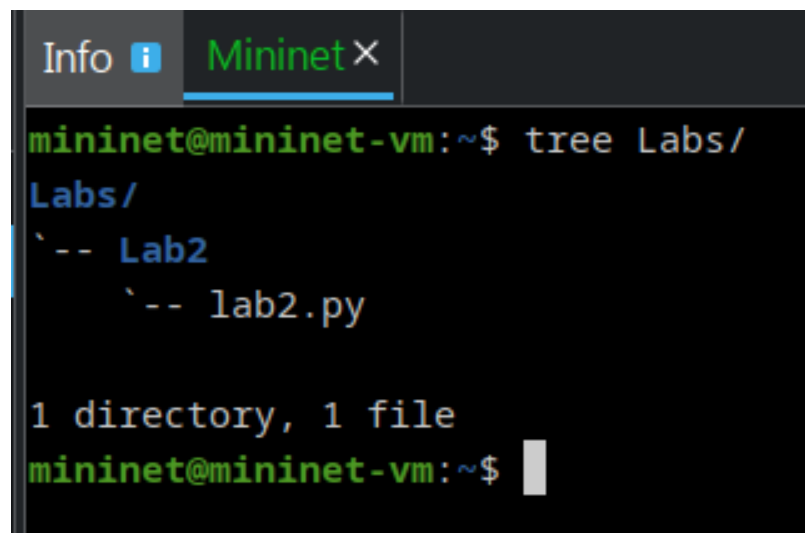
Мета роботи. Набути практичних навичок побудови програмно-конфігурованої мережі SDN.

Завдання:

1. Створити просту SDN мережу, що поєднує клієнта та два HTTP сервера;
2. Отримати доступ до серверів за допомогою клієнтського програмного забезпечення;
3. Продемонструвати можливість пінгування (ping) та відслідковування маршруту (traceroute) до працюючого OpenFlow комутатора та через нього.

Хід роботи.

Створимо мережу за допомогою Python скрипту. По-перше, було створено каталоги для збереження робіт та сам файл з кодом:



```
Info ⓘ Mininet ×
mininet@mininet-vm:~$ tree Labs/
Labs/
├── Lab2
│   └── lab2.py
└── 1 directory, 1 file
mininet@mininet-vm:~$
```

Спочатку імпортуємо основні класи, потрібні для роботи. Mininet – створює й керує емуляцією мережі (хости, свічі, лінки, контролери). Controller – базовий (локальний) OpenFlow-контролер, OVSSwitch – реалізація комутатора на базі Open vSwitch. CLI – інтерактивна командна оболонка Mininet. Функція setLogLevel потрібна для встановлення рівня логів (наприклад – info). Імпорт стандартного модуля os використовується для виконання системних команд у хостовій VM.

```
1 from mininet.net import Mininet
2 from mininet.node import Controller, OVSSwitch
3 from mininet.cli import CLI
4 from mininet.log import setLogLevel, info
5 import os
```

В головній функції `run` яка створює й запускає мережу, створюється об'єкт `net` типу `Mininet`. В параметрах вказується, що за замовчуванням використовується локальний контролер класу `Controller` та `Open vSwitch` як реалізація свічів.

```
1 def run():
2     net = Mininet(controller=Controller, switch=OVSSwitch)
```

Далі додамо контролер, свіч, хости (клієнти) та зв'язки між ними. Запустимо мережу.

```
1 c0 = net.addController('c0')
2 s1 = net.addSwitch('s1')
3 h1 = net.addHost('h1', ip='10.0.0.1/24')
4 h2 = net.addHost('h2', ip='10.0.0.2/24')
5 h3 = net.addHost('h3', ip='10.0.0.3/24')
6
7 net.addLink(h1, s1)
8 net.addLink(h2, s1)
9 net.addLink(h3, s1)
10
11 net.start()
```

За замовчуванням OVS це L2-пристрій без IP-адреси. Щоб мати можливість надіслати пінг до OpenFlow-комутатора, зробити `traceroute`, або запускати інші діагностичні інструменти – потрібно створити інтерфейс і дати йому IP.

```
1 info('*** Adding internal management port s1-mgmt with IP 10.0.0.254/24\n')
2 os.system('ovs-vsctl add-port s1 s1-mgmt -- set interface s1-mgmt type=internal')
3
4 os.system('ip addr add 10.0.0.254/24 dev s1-mgmt || true')
5 os.system('ip link set s1-mgmt up || true')
```

Запускаємо на другому і третьому клієнті HTTP сервери в фоні (щоб скрипт не чекав завершення процесу).

```
1 info('*** Starting HTTP servers on h2 and h3 (port 80)\n')
2 h2.cmd('python3 -m http.server 80 &')
3 h3.cmd('python3 -m http.server 80 &')
```

Запускаємо `Mininet CLI` – інтерактивний інтерфейс, де можна виконувати команди на хостах. Після виходу з CLI (тобто після `CLI(net)`), викликається `net.stop()`, яка зупиняє мережу: вимикає контролер, свічі, видаляє інтерфейси й очищає ресурси.

```
1 info('*** Setup complete - entering CLI\n')
2 CLI(net)
3 net.stop()
```

Наприкінці встановлюємо рівень логування Mininet на `info`, щоб бачити інформаційні повідомлення, статуси під час старту тощо, та викликаємо головну функцію `run()`, яка створює та керує мережею.

```
1  if __name__ == "__main__":
2      setLogLevel('info')
3      run()
```

Виконуємо скрипт, перевіряємо стан мережі:

```
mininet@mininet-vm:~/Labs/Lab2$ sudo -E python3 ./lab2.py
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Adding internal management port s1-mgmt with IP 10.0.0.254/24
*** Starting HTTP servers on h2 and h3 (port 80)
*** Setup complete - entering CLI
*** Starting CLI:
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
c0
mininet> nodes
available nodes are:
c0 h1 h2 h3 s1
```

Пінгуємо з основного клієнту (h1) сервери (h2, h3):

```
mininet> h1 ping -c 1 h2
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=1.28 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.275/1.275/1.275/0.000 ms
mininet> h1 ping -c 1 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=1.94 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.938/1.938/1.938/0.000 ms
mininet> █
```

Доступність HTML сторінок можна перевірити за допомогою утиліти curl:

```
mininet> h1 curl h2
<!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=iso8859-1">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="lab2.py">lab2.py</a></li>
</ul>
<hr>
</body>
</html>
mininet>
```

Якщо ж команда не працює бо утиліта відсутня, її треба встановити на самій віртуальній машині, поза mininet. Так як хости працюють за допомогою механізму неймспейсів, то пакет буде доступним і там. Також треба встановити утиліту traceroute для подальшої роботи.

```
mininet@mininet-vm:~/Labs/Lab2$ sudo apt install curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
curl is already the newest version (7.68.0-1ubuntu2.25).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
mininet@mininet-vm:~/Labs/Lab2$
```

Також, якщо додати прапорець "-I", можна дізнатися версію сервера, тощо. Це відбувається за допомогою перегляду заголовків пакету.

```
mininet> h1 curl -I h3
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.8.10
Date: Wed, 15 Oct 2025 21:06:58 GMT
Content-type: text/html; charset=iso8859-1
Content-Length: 340
```

Проведемо пінг та знаходження маршруту через комутатор.

```
mininet> h1 ping 10.0.0.254
PING 10.0.0.254 (10.0.0.254) 56(84) bytes of data.
64 bytes from 10.0.0.254: icmp_seq=1 ttl=64 time=2.43 ms
64 bytes from 10.0.0.254: icmp_seq=2 ttl=64 time=0.356 ms
64 bytes from 10.0.0.254: icmp_seq=3 ttl=64 time=0.131 ms
^C
--- 10.0.0.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2021ms
rtt min/avg/max/mdev = 0.131/0.971/2.427/1.033 ms
mininet> h1 traceroute 10.0.0.254
traceroute to 10.0.0.254 (10.0.0.254), 30 hops max, 60 byte packets
 1 10.0.0.254 (10.0.0.254) 5.345 ms 6.065 ms 6.314 ms
mininet> h1 traceroute h3
traceroute to 10.0.0.3 (10.0.0.3), 30 hops max, 60 byte packets
 1 10.0.0.3 (10.0.0.3) 1.974 ms 1.821 ms 1.815 ms
mininet>
```

Відсутність хопів можна пояснити тим, що це комутатор, а не маршрутизатор. Як бачимо, все працює.

Насамкінець, спробуємо пропінгувати та дістати веб-сторінку з самої віртуальної машини Mininet, а не в середині CLI. Для цього під'єднаємося ще раз, з першого підключення запустимо скрипт, а з другого виконаємо операції (або ж просто можна запустити скрипт у фоні):

```
mininet@mininet-vm:~$ ping -c 1 10.0.0.254
PING 10.0.0.254 (10.0.0.254) 56(84) bytes of data.
64 bytes from 10.0.0.254: icmp_seq=1 ttl=64 time=0.013 ms

--- 10.0.0.254 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.013/0.013/0.013/0.000 ms
mininet@mininet-vm:~$ ping -c 1 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=1.99 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.989/1.989/1.989/0.000 ms
mininet@mininet-vm:~$ curl -I 10.0.0.3
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.8.10
Date: Wed, 15 Oct 2025 21:24:40 GMT
Content-type: text/html; charset=iso8859-1
Content-Length: 340
```

Це спрацювало саме через те, що відповідний IP був назначений в самому скрипті. Мережева конфігурація віртуальної машини представлена на скріншоті, де видно, що прямого доступу до IP-адреси 10.0.0.3 немає.

```
mininet@mininet-vm:~$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:2d:19:50 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 metric 100 brd 192.168.56.255 scope global dynamic eth0
        valid_lft 520sec preferred_lft 520sec
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:9f:47:2b brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic eth1
        valid_lft 84032sec preferred_lft 84032sec
73: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether fe:32:f0:5e:d4:3d brd ff:ff:ff:ff:ff:ff
74: s1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 7e:78:69:6e:73:47 brd ff:ff:ff:ff:ff:ff
76: s1-eth1@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ovs-system state UP group default qlen 1000
    link/ether 16:c8:37:c7:4e:c4 brd ff:ff:ff:ff:ff:ff link-netnsid 0
77: s1-eth2@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ovs-system state UP group default qlen 1000
    link/ether 02:dd:08:f9:19:31 brd ff:ff:ff:ff:ff:ff link-netnsid 1
78: s1-eth3@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ovs-system state UP group default qlen 1000
    link/ether e2:0d:55:4c:53:0d brd ff:ff:ff:ff:ff:ff link-netnsid 2
79: s1-mgmt: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether da:5c:73:1a:79:55 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.254/24 scope global s1-mgmt
        valid_lft forever preferred_lft forever
mininet@mininet-vm:~$
```

Знаходження шляху також працює.

```
1 mininet@mininet-vm:~$ traceroute 10.0.0.2
2 traceroute to 10.0.0.2 (10.0.0.2), 30 hops max, 60 byte packets
3 1 10.0.0.2 (10.0.0.2) 1.435 ms 3.347 ms 3.371 ms
```

Додатково проведемо перевірку через утиліту tcpdump. На скріншоті можна побачити шлях пакету-запиту та пакету-відповіді. Це результат пінгу з консолі віртуальної машини до віртуального хосту h2, який має адресу 10.0.0.2.

```
mininet@mininet-vm:~$ sudo tcpdump -n -i s1-mgmt icmp or arp or tcp port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s1-mgmt, link-type EN10MB (Ethernet), capture size 262144 bytes
14:34:37.771389 IP 10.0.0.254 > 10.0.0.2: ICMP echo request, id 5, seq 1, length 64
14:34:37.771580 IP 10.0.0.2 > 10.0.0.254: ICMP echo reply, id 5, seq 1, length 64
```

Висновок. Під час виконання лабораторної роботи було успішно реалізовано просту SDN-мережу з одним клієнтом та двома HTTP-серверами за допомогою Mininet та Open vSwitch. Було написано і виконано Python-скрипт, який створює топологію, піднімає OVS-комутатор s1, підключає хости h1, h2, h3, додає internal-порт s1-mgmt з IP-адресою для менеджменту та запускає HTTP-сервери на h2 і h3. За допомогою інструментів ping, curl та tcpdump перевірено

доступність серверів і наявність трафіку. Також проведено експеримент з traceroute, який продемонстрував важливу відмінність між L2-комутатором і L3-маршрутизатором – OVS як L2-пристрій не декрементує TTL, тому traceroute у даній плоскій топології показує один хоп.

Результати роботи підтвердили виконання поставлених завдань і дозволили набути практичних навичок.

Лістинг скрипту.

```
1  #!/usr/bin/env python3
2  from mininet.net import Mininet
3  from mininet.node import Controller, OVSSwitch
4  from mininet.cli import CLI
5  from mininet.log import setLogLevel, info
6  import os
7
8  def run():
9      net = Mininet(controller=Controller, switch=OVSSwitch)
10     c0 = net.addController('c0')
11     s1 = net.addSwitch('s1')
12     h1 = net.addHost('h1', ip='10.0.0.1/24')
13     h2 = net.addHost('h2', ip='10.0.0.2/24')
14     h3 = net.addHost('h3', ip='10.0.0.3/24')
15
16     net.addLink(h1, s1)
17     net.addLink(h2, s1)
18     net.addLink(h3, s1)
19
20     net.start()
21
22     info('*** Adding internal management port s1-mgmt with IP 10.0.0.254/24\n')
23     os.system('ovs-vsctl add-port s1 s1-mgmt -- set interface s1-mgmt type=internal')
24
25     os.system('ip addr add 10.0.0.254/24 dev s1-mgmt || true')
26     os.system('ip link set s1-mgmt up || true')
27
28     info('*** Starting HTTP servers on h2 and h3 (port 80)\n')
29     h2.cmd('python3 -m http.server 80 &')
30     h3.cmd('python3 -m http.server 80 &')
31
32     info('*** Setup complete - entering CLI\n')
33     CLI(net)
34     net.stop()
35
36 if __name__ == "__main__":
37     setLogLevel('info')
38     run()
```