Міністерство освіти і науки України

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

# ПРОЄКТУВАННЯ SDN МЕРЕЖ

## Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра за освітньо-професійною та освітньо-науковою програмами
«Системне програмування та спеціалізовані комп'ютерні системи»
спеціальності F7 «Комп'ютерна інженерія»

Укладачі: Мартинова О.П., Крайносвіт А.А., Боярінова Ю.Є.

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2026

УДК 004.7

Укладачі: Мартинова Оксана Петрівна, канд. техн. наук, доц.
Крайносвіт Аркадій Артемович, асистент
Боярінова Юлія Євгеніївна, канд. техн. наук, доц.

Рецензент Олещенко Л.М., канд. техн. наук, доцент

Відповідальний редактор Тарасенко-Клятченко О.В., канд. техн. наук, доцент

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського*
*(протокол № 4 від 05.02.2026 р.)*
*за поданням вченої ради факультету програмних систем та прикладної математики*
*(протокол №8 від 26.01.2026 р.)*

Проєктування SDN мереж [Електронний ресурс] : лаб. практикум : навч. посіб. для здобувачів ступеня магістра за освіт.-проф. та освіт.-науков. програмами «Системне програмування та спеціалізовані комп'ютерні системи» спец. F7 «Комп'ютерна інженерія» / КПІ ім. Ігоря Сікорського ; уклад.: О. П. Мартинова, А. А. Крайносвіт, Ю. Є. Боярінова. – Електрон. текст. дані (1 файл: 9,86). – Київ : КПІ ім. Ігоря Сікорського, 2026. – 157 с.

Реєстр. № НП XX/XX-XXX. Обсяг 6 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

# ЗМІСТ

ВСТУП …….. 4

ЛАБОРАТОРНА РОБОТА 1. Основи технології програмно-керованих мереж. Моделювання роботи мережевого контролера в Cisco Packet Tracer ..…….... 5

ЛАБОРАТОРНА РОБОТА 2. Основи технології програмно-керованих мереж. Реалізація REST API за допомогою контролера SDN в Cisco Packet Tracer ……....33

ЛАБОРАТОРНА РОБОТА 3. Моделювання програмно-керованої мережі у симуляторі Mininet. Ручне керування потоками……61

ЛАБОРАТОРНА РОБОТА 4. Моделювання програмно-керованої мережі у симуляторі Mininet. Використання графічного редактора MiniEdit ..…….....91

ЛАБОРАТОРНА РОБОТА 5. Побудова брандмауера як сервісу в Mininet……110

ЛАБОРАТОРНА РОБОТА 6. Налаштування маршрутизації в Mininet……..…..121

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ……..……157

# ВСТУП

Навчальний посібник розроблений для допомоги студентам в оволодінні знаннями та практичними навичками з дисципліни «Проєктування SDN мереж». Навчальний посібник складається із вступу та 6 розділів, в яких містяться завдання на виконання лабораторних робіт за освітньою програмою «Системне програмування та спеціалізовані комп'ютерні системи» спеціальності F7 «Комп'ютерна інженерія».

Метою цієї збірки лабораторних робіт з дисципліни «Проєктування SDN мереж» є засвоєння та поглиблення знань щодо принципів організації та роботи програмно-керованих комп'ютерних мереж. Для досягнення цієї мети в лабораторних роботах досліджуються моделі комп'ютерних мереж, побудованих з використанням віртуального середовища моделювання Mininet та симулятора мережі Cisco Packet Tracer.

В середовищі Packet Tracer створюється модель мережі, до її складу додається контролер мережі, налаштовується і виконується ручне керування мережею безпосередньо адміністратором за допомогою інтерфейсу командного рядка (CLI) та демонструються можливості керування мережею за допомогою програмно-керованого контролера (SDN-контролера). Також наведені приклади створення і виконання сценаріїв автоматизованого керування мережею за допомогою програмного контролера мережі шляхом надсилання запитів керування REST API із застосуванням додатків Postman та Visual Studio Code.

В середовищі Mininet створюються та досліджуються робота типових складових програмно-керованої мережі (маршрутизатора, комутатора, брандмауера), структура та особливості налаштування програмованих комутаторів, що використовують протокол Open Flow для створення та оновлення таблиць потоків на комутаторах.

Кожна лабораторна робота призначена для вивчення конкретної теми та пов'язана з відповідними питаннями, що стосуються розгляду особливостей організації та функціонування окремих компонентів програмно-керованої комп'ютерної мережі.

В результаті виконання лабораторних робіт з дисципліни «Проєктування SDN мереж» студент повинен:
ЗНАТИ:

- особливості архітектури програмно-керованих комп'ютерних мереж, організацію взаємодії контролера SDN з мережевими пристроями;
- методику налаштування програмно-керованих мережевих пристроїв для виконання задач комутації та маршрутизації.

УМІТИ:

- формулювати вимоги до параметрів налаштування мережевих пристроїв залежно від структури комп'ютерної мережі;
- аналізувати мережевой трафік за допомогою різних засобів (Wireshark, Packet Tracer, tcpdump тощо);
- аналізувати і оцінювати результати виконаної роботи.

# Лабораторна робота 1

## Основи технології програмно-керованих мереж. Моделювання роботи мережевого контролера в Cisco Packet Tracer

**Мета роботи.** Побудувати та налагодити модель мережі, ввести до її складу контролер мережі та порівняти управління мережею за допомогою інтерфейсу командного рядка (CLI) з управлінням мережею за допомогою програмно-керованого мережевого контролера (SDN-контролера).

### 1. Теоретичні відомості

**Програмно-керована мережа** (**SDN** – Software Defined Networking) – мережа передачі даних, в якій рівень управління мережею відділений від пристроїв передачі даних і реалізується програмно.
Дані передаються відповідно до таблиць маршрутизації, що зберігаються на апаратних системах, як і раніше. Але ці таблиці централізовано керуються віддаленою системою, у зв'язку з чим адміністратору не потрібно змінювати таблиці на кожному комутаторі. В ідеальному випадку всі мережеві компоненти мають керуватися і налаштовуватися однією операцією. Спільна робота компонентів програмно-керованої мережі може бути організована згідно із стандартом OpenFlow.
Програмно-керовані мережі ефективні для побудови інфраструктурних хмарних сервісів, в умовах, коли за запитом від споживачів послуг необхідно автоматично і в найкоротші терміни створювати віртуальні вузли і виділяти віртуальні мережеві ресурси для них.
Також програмно-керовані мережі доцільні в умовах великих центрів обробки даних, дозволяючи скоротити витрати на супровід мережі за рахунок централізації управління на програмному контролері і підвищити відсоток використання ресурсів мережі завдяки динамічному управлінню.
Іншим перспективним застосуванням програмно-керованих мереж вважаються додатки в концепції «інтернету речей». Це концепція комунікаційної мережі фізичних або віртуальних об'єктів («речей»), які мають технології для взаємодії між собою та з оточуючим середовищем, а також можуть виконувати певні дії без втручання людини.
SDN пропонує новий спосіб управління маршрутизацією пакетів даних через централізований сервер.

Мережа SDN розділяється на три рівні, кожен з яких складається із різних компонентів.
**Рівень додатків** охоплює рішення, спрямовані на розширення мережевих послуг. Ці рішення в основному є програмними додатками, які взаємодіють з контролером.
**Рівень площини управління** включає логічно централізований контролер SDN, який підтримує глобальне представлення мережі. Він також приймає запити через чітко визначені API-інтерфейси з рівня додатків і виконує консолідуюче управління і моніторинг мережевими пристроями за допомогою стандартних протоколів.
**Рівень інфраструктури** або рівень даних задіює фізичне мережеве обладнання, включно з комутаторами та маршрутизаторами локальних мереж, надає програмоване та високошвидкісне обладнання та програмне забезпечення, що відповідає галузевим стандартам.

Ідея SDN полягає у відокремленні рівня програми та управління від самих передавальних пристроїв, таких як комутатори (рисунок 1). Це означає, що вся логіка, пов'язана з визначенням потоків даних, виноситься з мережевих пристроїв. Точкою управління є

додаток, який визначає, куди і як передавати трафік. Контролер і додатки запускаються на сервері і керують усіма пристроями мережі.
Контролер потрібен для трансляції результатів роботи додатків в мову правил потоків даних, яку розуміють пристрої передачі даних. Такі правила передаються за допомогою протоколу OpenFlow.

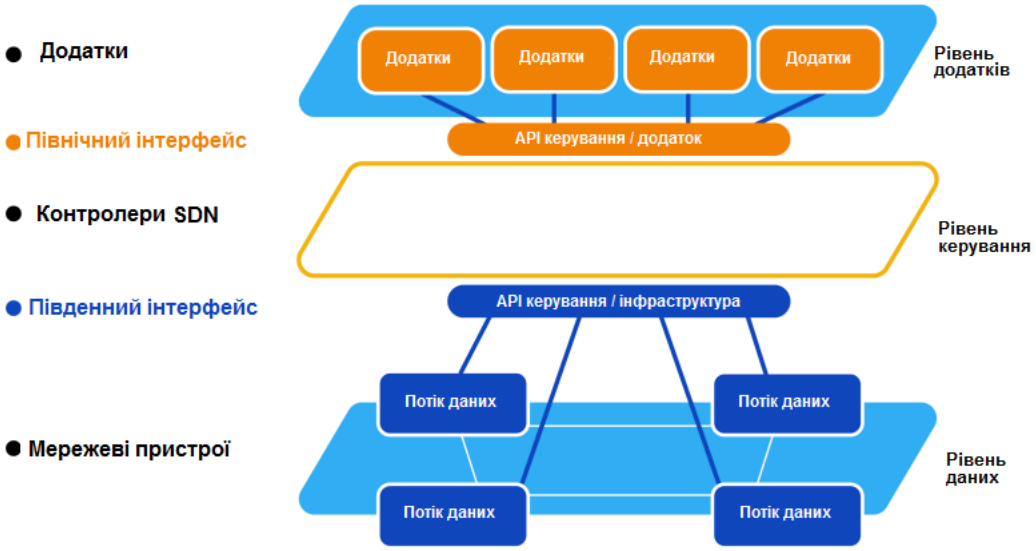



Рисунок 1– SDN відокремлює рівень керування від самих передавальних пристроїв

Різниця між традиційною розподіленою мережею та централізованою мережею з контролером SDN, як ядром мережі, полягає в наступному:

Традиційна мережа – повністю розподілена мережева структура, в якій кожен маршрутизатор незалежно опрацьовує записи маршрутизації. Кожен маршрутизатор у мережі збирає інформацію про стан каналу як матеріал для алгоритму маршрутизації, який обчислює найкоротший шлях до певного пункту призначення. Будь-які зміни топології мережі призведуть до того, що маршрутизатори будуть лавинно розсилати інформацію про стан нового каналу і незалежно обчислювати новий маршрут.
Отже, традиційна мережа може автоматично відновлювати роботу мережі у разі збою. Це дозволяє підтримувати високу надійність таких мереж.
У мережі SDN мережева відмовостійкість залежить від контролера SDN, оскільки контролер SDN є основним компонентом і централізовано керує мережевими пристроями. В результаті це може привести до виникнення єдиної точки відмови.

Переваги використання мереж SDN:

- зменшення вартості пристроїв за рахунок централізації функцій, що керують потоками даних;
- прискорення розгортання нових IT-послуг;
- зменшення витрат на обслуговування через централізований контроль та автоматизацію;
- розширення функціональності мережі.

Контролер SDN використовує два спеціальні інтерфейси – північний інтерфейс (NBI – Northbound Interface) і південний інтерфейс (SBI – Southbound Interface ).
**Південний інтерфейс (SBI)**

Контролер SDN повинен зв'язуватися з пристроями для програмування рівня даних. Це виконується через південний інтерфейс. Це не фізичний інтерфейс, а програмний інтерфейс API (Application Programming Interface – інтерфейс прикладного програмування). API дозволяє додатку відкривати іншу програму за допомогою попередньо визначених функцій і структур даних.

Деякі популярні південні інтерфейси:

- OpenFlow: це, мабуть, на сьогодні найпопулярніший SBI; це протокол з відкритим вихідним кодом від Open Networking Foundation. Існує досить багато мережевих пристроїв і контролерів SDN, які підтримують OpenFlow.
- Cisco OpFlex: це відповідь Cisco на OpenFlow. Це також протокол з відкритим вихідним кодом, який був представлений IETF для стандартизації.
- CLI: Cisco пропонує APIC-EM, який є рішенням SDN для поточного покоління маршрутизаторів і комутаторів. Він використовує протоколи, які доступні на обладнанні поточного покоління, такі як telnet, SSH і SNMP.

**Північний інтерфейс (NBI)**

Північний інтерфейс використовується для доступу до самого контролера SDN. Це дозволяє адміністратору отримати доступ до SDN, щоб налаштувати його або витягти з нього інформацію. Це можна зробити за допомогою графічного інтерфейсу, але він також пропонує API, який дозволяє іншим програмам отримувати доступ до контролера SDN.
Подібний підхід дозволяє використовувати це для написання сценаріїв і автоматизації мережевого адміністрування.

Ось деякі приклади:

- Вивести інформацію з усіх мережевих пристроїв у мережі.
- Показати стан усіх фізичних інтерфейсів у мережі.
- Додати нову VLAN на всіх комутаторах мережі.
- Показати топологію всієї мережі.
- Автоматичне налаштування IP-адрес, маршрутизації і списків доступу при створенні нової віртуальної машини.

Через API кілька додатків можуть отримати доступ до контролера SDN:

- Користувач, який використовує графічний інтерфейс для отримання інформації про мережу з контролера SDN. За лаштунками графічний інтерфейс використовує API.
- Скрипти, написані на Java або Python, можуть використовувати API для отримання інформації з контролера SDN або для налаштування мережі.
- Інші додатки можуть отримати доступ до контролера SDN. Наприклад, додаток, який автоматично налаштовує мережу після створення нової віртуальної машини на сервері VMware ESXi.

Контролери SDN зазвичай використовують REST API (Representational State Transfer).
REST API використовує HTTP-повідомлення для взаємодії з контролером SDN при надсиланні та отриманні інформації.
Він використовує ті ж HTTP-повідомлення:

- **HTTP GET**: використовується, коли потрібно отримати інформацію.
- **HTTP POST/PUT**: використовується, коли потрібно завантажити або оновити інформацію. Це схоже на перегляд вебсторінки, тільки цього разу запит виконується не до вебсторінки або зображення, а до конкретного об'єкта у контролері SDN, наприклад, до списку з усіма VLAN в мережі. Коли контролер SDN отримає запит HTTP GET, він надасть відповідь з потрібною інформацією. Ця інформація надається у загальному форматі даних.

Два найбільш часто використовуваних форматів даних:

- **JSON** (JavaScript Object Notation).
- **XML** (eXtensible Markup Language).

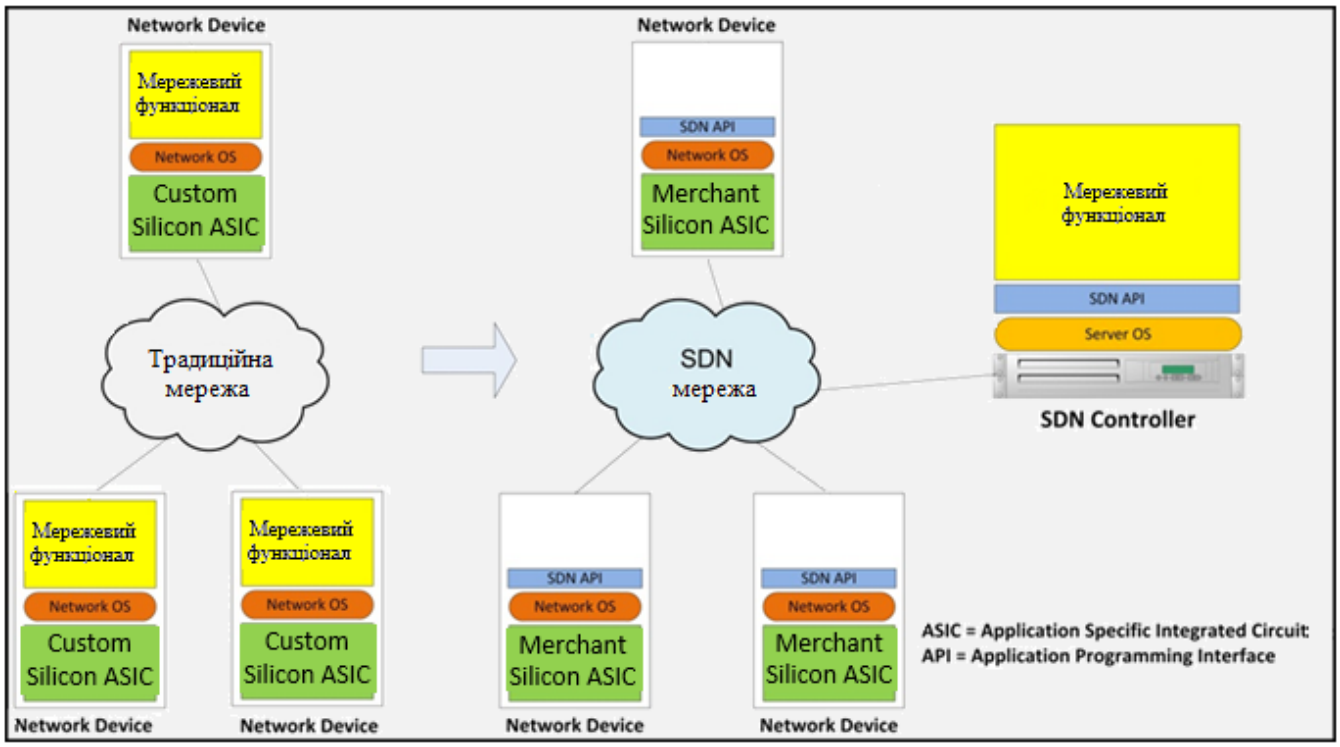



Рисунок 2– Порівняння структур традиційної мережі і мережі SDN

У SDN мережах завдання комутації трафіку і завдання управління чітко розділені. Вся логіка керування централізується і передається контролеру (рисунок 2). Комутатор у концепції SDN – досить примітивний пристрій, який відповідає тільки за перемикання пакетів на підставі дуже простих правил. Контролер SDN керує усіма комутаторами в мережі і програмує кожен з них для правильної передачі трафіку. Централізація логіки управління дозволяє програмувати мережу як єдине ціле і спростити операційну модель великих корпоративних мереж, які занадто статичні на даний момент і не відповідають сучасному бізнесу, з властивими йому мобільністю користувачів/пристроїв/додатків, а також розподілом додатків між віртуальними машинами і інтенсивним обміном даними. Незалежне від виробника керування усіма пристроями з єдиного центру істотно спрощує конфігурацію і експлуатацію мережі. Завдяки контролеру з розширеними API-інтерфейсами вся мережа стає подібною до одного великого логічного комутатора. Протокол OpenFlow – один з найбільш універсальних протоколів комунікації контролерів і комутаторів на сьогоднішній день. Він надає стандартний підхід до програмування таблиць комутації, в яких основним об'єктом є потік даних.

SDN має багато переваг перед традиційною мережею, включаючи наступні:

- Більш широкі можливості управління з підвищеними швидкістю і гнучкістю. Замість того, щоб вручну програмувати кілька апаратних пристроїв конкретних постачальників, розробники можуть управляти потоком трафіку в мережі, просто програмуючи стандартні контролери на базі програмного забезпечення з відкритим вихідним кодом. Крім того, адміністратори мережі мають більш широкий вибір мережевого обладнання, оскільки вони можуть вибирати протокол з відкритим кодом для зв'язку з будь-якою кількістю апаратних пристроїв через центральний контролер.

- Налаштовувана мережева інфраструктура. Завдяки SDN адміністратори можуть налаштовувати мережеві служби та виділяти віртуальні ресурси, щоб централізовано змінювати мережеву інфраструктуру в режимі реального часу. Це дозволяє адміністраторам мережі оптимізувати потік даних у мережі, надаючи пріоритет програмам, які потребують більш високого рівня доступності.
- Високий рівень безпеки. SDN забезпечує візуалізацію всієї мережі та забезпечує більш цілісне уявлення про загрози безпеці. Поширення інтелектуальних пристроїв з підключенням до Інтернет дає SDN перевагу в порівнянні з традиційними мережами. Розробники можуть створювати окремі зони для пристроїв, яким потрібні різні рівні безпеки, або швидко ізолювати зламані пристрої, щоб не піддавати ризику всю мережу.

## 2. Виконання роботи

Лабораторну роботу будемо виконувати в такій послідовності:

1. Налаштування початкової топології мережі (рисунок 3).
2. Використання інтерфейсу командного рядка для збору інформації про комутатори.
3. Налаштування контролера SDN.
4. Використання контролера SDN для визначення топології.
5. Використання контролера SDN для збору інформації.
6. Використання контролера SDN для налаштування параметрів мережі.

Для побудови моделі мережі використовуються  комутатори 3-го рівня Cisco 3650-24PS і модульні маршрутизатори з програмно-визначеними можливостями ISR4331.

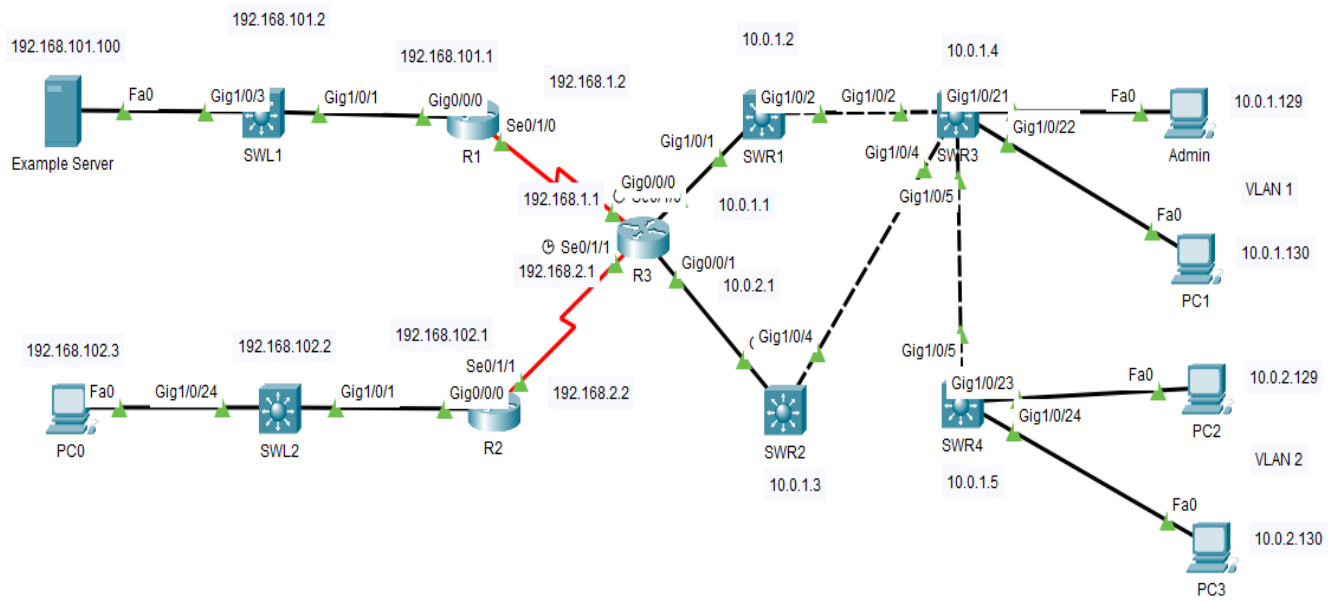



Рисунок 3 – Початкова топологія мережі

Мережу налаштуємо наступним чином:

- На маршрутизаторах працюватиме протокол маршрутизації OSPFv2.
- Протокол SSH буде задіяний на всіх пристроях з користувачем **cisco** та паролем **cisco123!**
- R3 буде сервером DHCPv4 для VLAN1 та VLAN2.
- Усі комутатори SWR# належатимуть до VLAN1.

**Таблиця IP-адрес**

| Пристрій | Інтерфейс | IP-адреса |
|---|---|---|
| R1 | Gig0/0/0 | 192.168.101.1 |
| | Se0/1/0 | 192.168.1.2 |
| R2 | Gig0/0/0 | 192.168.102.1 |
| | Se0/1/1 | 192.168.2.2 |
| R3 | Gig0/0/0 | 10.0.1.1 |
| | Gig0/0/1 | 10.0.2.1 |
| | Se0/1/0 | 192.168.1.1 |
| | Se0/1/1 | 192.168.2.1 |
| SWL1 | VLAN 1 | 192.168.101.2 |
| SWL2 | VLAN 1 | 192.168.102.2 |
| SWR1 | VLAN 1 | 10.0.1.2 |
| SWR2 | VLAN 1 | 10.0.1.3 |
| SWR3 | VLAN 1 | 10.0.1.4 |
| SWR4 | VLAN 1 | 10.0.1.5 |
| Admin | Fa0 | 10.0.1.129 |
| PC1 | Fa0 | 10.0.1.130 |
| PC2 | Fa0 | 10.0.2.129 |
| PC3 | Fa0 | 10.0.2.130 |
| PC0 | Fa0 | 192.168.102.3 |
| Example Server | Fa0 | 192.168.101.100 |
| PT Controller0* | Gig0 | 192.168.101.254 |

* Мережевий контролер PT-Controller0 буде доданий і налаштований в розділі 2.15 на сторінці 19.

Налаштування пристроїв мережі.

2.1. Виконуємо налаштування Example Server:

- Налаштування інтерфейсу

  IP Address 192.168.101.100
  Subnet Mask 255.255.255.0
  Default Gateway 192.168.101.1
  DNS Server 192.168.101.100

- В службі DNS створюємо запис типу A, який визначає IP-адресу 192.168.101.100 для доменного імені www.example.com; перемикач **DNS Service** в «**On**».
- Активуємо службу NTP; перемикач **NTP Service в** «**On**».

2.2. Виконуємо налаштування комутатора SWL1. Попередньо відкриваємо закладку Physical і вставляємо блок живлення у шасі пристрою (рисунок 4).

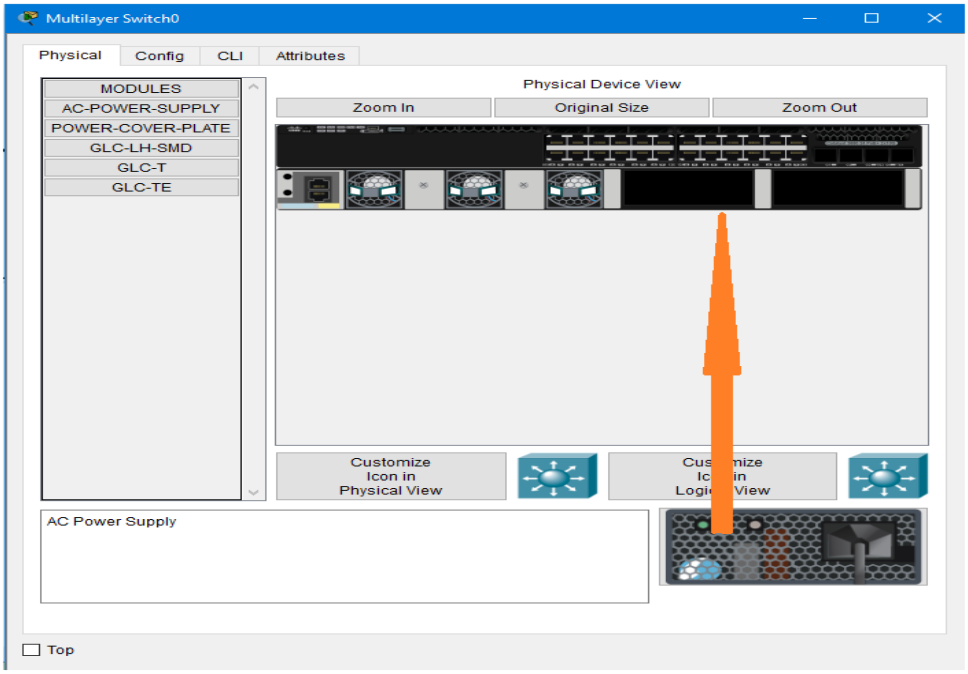



Рисунок 4 – Під'єднання блока живлення

Подальші налаштування виконуємо користуючись інтерфейсом командного рядка.

Switch>**enable**
Switch#**conf term**
Switch(config)#**hostname SWL1**
SWL1(config)#**ip domain-name example.com** ім'я домена для генерації ключа
SWL1(config)#**username cisco privilege 15 password 0 cisco123!**
SWL1(config)#**crypto key generate RSA general-keys modulus 768** ключ для шифрування SSH-трафіка

SWL1(config)#**ip ssh version 2**
SWL1(config)#**line vty 0 4** входимо в режим конфігурації термінальних ліній
SWL1(config-line)#**login local** автентифікуємо всі вхідні сеанси через локальну базу даних імен користувачів, створених за допомогою команди username
SWL1(config-line)#**exit**
SWL1(config)#**ip name-server 192.168.101.100**
SWL1(config)#**int vlan 1**
SWL1(config-if)#**no shutdown**
SWL1(config-if#)**ip address 192.168.101.2 255.255.255.0**
SWL1(config-if)#**exit**
SWL1(config)#**ip default-gateway 192.168.101.1**

SWL1(config)#**int gig1/0/1**
SWL1(config-if)#**no shutdown**
SWL1(config-if)#**switchport mode access**
SWL1(config-if)#**switchport nonegotiate**
SWL1(config-if)#**spanning-tree portfast**
SWL1(config)#**int gig1/0/3**
SWL1(config-if)#**no shutdown**
SWL1(config-if)#**switchport mode trunk**
SWL1(config-if)#**exit**

SWL1(config)#**no cdp run**
SWL1(config)#**logging 192.168.101.100**

SWL1(config)#**ntp server 192.18.101.100** для генерації ключа шифрування
SWL1(config)#**do write**

Пояснення 1. Команда **switchport nonegotiate** у комутаторах Cisco відключає **Динамічний Протокол Тракінгу (DTP)** , запобігаючи автоматичному узгодженню режиму порту (access або trunk) з сусіднім пристроєм, що змушує адміністратора налаштовувати режим порту **вручну** . Це корисно, коли потрібно, щоб порт залишився портом доступу (access) або транковим портом (trunk) **незалежно** від того, що намагається зробити сусідній комутатор, і щоб уникнути небажаних перемикань у режим trunk.

Пояснення 2. **Spanning-Tree PortFast –** це функція комутаторів Cisco, яка дозволяє порту **відразу переходити в стан пересилання (forwarding)** , минаючи звичайні затримки STP (прослуховування та навчання), щоб кінцеві пристрої (комп'ютери) могли негайно розпочати роботу. Це прискорює підключення, тому що порт не чекає, поки STP перевірить наявність петель, припускаючи, що до порту підключено лише кінцевий хост, а не інший комутатор.

Пояснення 3. Команда **no cdp run** у мережевому обладнанні Cisco означає глобальне відключення протоколу **Cisco Discovery Protocol (CDP)**, який використовується для виявлення сусідніх пристроїв Cisco та обміну інформацією про них, причому відключення відбувається відразу для всіх інтерфейсів пристрою.

2.3. Виконуємо налаштування комутатора SWL2. Попередньо відкриваємо закладку Physical і вставляємо блок живлення у шасі пристрою (рисунок 4).

Switch>**enable**
Switch#**conf term**
Switch(config)#**hostname SWL2**
SWL2(config)#**ip domain-name example.com**
SWL2(config)#**username cisco privilege 15 password 0 cisco123!**
SWL2(config)#**crypto key generate RSA general-keys modulus 768**

SWL2(config)#**ip ssh version 2**
SWL2(config)#**line vty 0 4**
SWL2(config-line)#**login local**
SWL2(config-line)#**exit**
SWL2(config)#**ip name-server 192.168.101.100** адреса сервера імен
SWL2(config)#**int vlan 1**
SWL2(config-if)#**no shutdown**
SWL2(config-if#)**ip address 192.168.102.2 255.255.255.0**
SWL2(config-if)#**exit**
SWL2(config)#**ip default-gateway 192.168.102.1**

SWL2(config)#**int gig1/0/24**
SWL2(config-if)#**no shutdown**
SWL2(config-if)#**switchport mode access**
SWL2(config-if)#**switchport nonegotiate**
SWL2(config-if)#**spanning-tree portfast**
SWL2(config)#**int gig1/0/1**
SWL2(config-if)#**no shutdown**
SWL2(config-if)#**switchport mode access**
SWL2(config-if)#**switchport nonegotiate**
SWL2(config-if)#**spanning-tree portfast**
SWL2(config-if)#**exit**

SWL2(config)#**no cdp run**
SWL2(config)#**logging 192.168.101.100**
SWL2(config)#**ntp server 192.18.101.100**
SWL2(config)#**do write**

2.4. Виконуємо налаштування інтерфейсу PC0:

IP Address 192.168.102.3
Subnet Mask 255.255.255.0

Default Gateway 192.168.102.1
DNS Server 192.168.101.100

2.5. Виконуємо налаштування маршрутизатора R1.

Router>**enable**
Router#**conf term**
Router(config)#**hostname R1**
R1(config)#**username cisco privilege 15 password 0 cisco123!**
R1(config)#**ip domain-name example.com**
R1(config)#**ip name-server 192.168.101.100**

R1(config)#**crypto key generate RSA general-keys modulus 768**
R1(config)#**ip ssh version 2**
R1config)#**line vty 0 4**
R1(config-line)#**login local**
R1(config-line)#**exit**

R1(config)#**int gig0/0/0**
R1(config-if)#**ip address 192.168.101.1 255.255.255.0**
R1(config-if)#**no shut**
R1(config-if)#**exit**
R1(config)#**int se0/1/0**
R1(config-if)#**ip address 192.168.1.2 255.255.255.0**
R1(config-if)#**no shut**
R1(config-if)#**exit**

R1(config)#**router ospf 1**
R1(config-router)#**network 0.0.0.0 255.255.255.255 area 0**
R1(config-router)#**exit**
R1(config)#**ip route 0.0.0.0 0.0.0.0 serial0/1/0**
R1(config)#**no cdp run**
R1(config)#**logging 192.168.101.100**
R1(config)#**ntp server 192.168.101.100**
R1(config)#**do write**

2.6. Виконуємо налаштування маршрутизатора R2.

Router>**enable**
Router#**conf term**
Router(config)#**hostname R2**
R2(config)#**username cisco privilege 15 password 0 cisco123!**
R2(config)#**ip domain-name example.com**
R2(config)#**ip name-server 192.168.101.100**

R2(config)#**crypto key generate RSA general-keys modulus 768**
R2(config)#**ip ssh version 2**
R2config)#**line vty 0 4**
R2(config-line)#**login local**
R2(config-line)#**exit**

R2(config)#**int gig0/0/0**
R2(config-if)#**ip address 192.168.102.1 255.255.255.0**
R2(config-if)#**no shut**
R2(config-if)#**exit**
R2(config)#**int se0/1/1**
R2(config-if)#**ip address 192.168.2.2 255.255.255.0**

R2(config-if)#**no shut**
R2(config-if)#**exit**

R2(config)#**router ospf 1**
R2(config-router)#**network 0.0.0.0 255.255.255.255 area 0**
R2(config-router)#**exit**
R2(config)#**ip route 0.0.0.0 0.0.0.0 serial0/1/1**
R2(config)#**no cdp run**
R2(config)#**logging 192.168.101.100**
R2(config)#**ntp server 192.168.101.100**
R2(config)#**do write**

2.7. Виконуємо налаштування маршрутизатора R3.

Router>**enable**
Router#**conf term**
Router(config)#**hostname R3**
R3(config)#**username cisco privilege 15 password 0 cisco123!**
R3(config)# **ip domain-name example.com**
R3(config)#**ip name-server 192.168.101.100**

R3(config)**#crypto key generate RSA general-keys modulus 768**
R3(config)#**ip ssh version 2**
R3config)#**line vty 0 4**
R3(config-line)#**login local**
R3(config-line)#**exit**

R3(config)#**int gig0/0/0**
R3(config-if)#**ip address 10.0.1.1 255.255.255.0**
R3(config-if)#**no shut**
R3(config-if)#**exit**
R3(config)#**int gig0/0/1**
R3(config-if)#**ip address 10.0.2.1 255.255.255.0**
R3(config-if)#**no shut**
R3(config-if)#**exit**
R3(config)#**int se0/1/0**
R3(config-if)#**ip address 192.168.1.1 255.255.255.0**
R3(config-if)#**no shut**
R3(config-if)#**exit**
R3(config)#**int se0/1/1**
R3(config-if)#**ip address 192.168.2.1 255.255.255.0**
R3(config-if)#**no shut**
R3(config-if)#**exit**

R3(config)#**router ospf 1**
R3(config-router)#**network 0.0.0.0 255.255.255.255 area 0**
R3(config-router)#**exit**
R3(config)#**ip route 0.0.0.0 0.0.0.0 serial0/1/1**
R3(config)#**no cdp run**
R3(config)#**logging 192.168.101.100**
R3(config)#**ntp server 192.168.101.100**
R3(config)#**do write**

R3(config)#**ip dhcp pool LAN1**
R3(dhcp-config) #**network 10.0.1.0 255.255.255.0**

R3(dhcp-config) #**default-router 10.0.1.1**
R3(dhcp-config) #**exit**
R3(config)#**ip dhcp pool LAN2**
R3(dhcp-config) #**network 10.0.2.0 255.255.255.0**
R3(dhcp-config) #**default-router 10.0.2.1**
R3(dhcp-config) #**exit**
R3(config)#**ip dhcp excluded-address 10.0.1.1 10.0.1.128**
R3(config)#**ip dhcp excluded-address 10.0.2.1 10.0.2.128**
R3(config)#**do write**

2.8. Виконуємо налаштування маршрутизатора SWR1. Попередньо відкриваємо закладку Physical і вставляємо блок живлення у шасі пристрою (рисунок 4).

Switch>**enable**
Switch#**conf term**
Switch(config)#**hostname SWR1**
SWR1(config)#**ip domain-name example.com**
SWR1(config)#**username cisco privilege 15 password 0 cisco123!**
SWR1(config)#**crypto key generate RSA general-keys modulus 768**
SWR1(config)#**ip ssh version 2**
SWR1config)#**line vty 0 4**
SWR1(config-line)#**login local**
SWR1(config-line)#**exit**

SWR1(config)#**ip name-server 192.168.101.100**
SWR1(config)#**int vlan 1**
SWR1(config-if)#**no shutdown**
SWR1(config-if)#**ip address 10.0.1.2 255.255.255.0**
SWR1(config-if)#**exit**
SWR1(config)#**ip default-gateway 10.0.1.1**

SWR1(config)#**int gig1/0/1**
SWR1(config-if)#**no shut**
SWR1(config-if)#**switchport mode access**
SWR1(config-if)#**switchport nonegotiate**
SWR1(config-if)#**spanning-tree portfast**
SWR1(config-if)#**exit**
SWR1(config)#**int gig1/0/2**
SWR1(config-if)#**no shut**
SWR1(config-if)#**switchport mode trunk**
SWR1(config)#**exit**

SWR1(config)#**no cdp run**
SWR1(config)#**logging 192.168.101.100**
SWR1(config)#**ntp server 192.168.101.100**
SWR1(config)#**do write**

2.9. Виконуємо налаштування маршрутизатора SWR2. Попередньо відкриваємо закладку Physical і вставляємо блок живлення у шасі пристрою (рисунок 4).

Switch>**enable**
Switch#**conf term**
Switch(config)#**hostname SWR2**
SWR2(config)#**ip domain-name example.com**
SWR2(config)#**username cisco privilege 15 password 0 cisco123!**

SWR2(config)#**crypto key generate RSA general-keys modulus 768**
SWR2(config)#**ip ssh version 2**
SWR2config)#**line vty 0 4**
SWR2(config-line)#**login local**
SWR2(config-line)#**exit**
SWR2(config)#**ip name-server 192.168.101.100**
SWR2(config)#**int vlan 1**
SWR2(config-if)#**no shutdown**
SWR2(config-if)#**ip address 10.0.1.3 255.255.255.0**
SWR2(config-if)#**exit**
SWR2(config)#**ip default-gateway 10.0.1.1**

SWR2(config)#**int gig1/0/1**
SWR2(config-if)#**no shut**
SWR2(config-if)#**switchport access vlan 2**
SWR2(config-if)#**switchport mode access**
SWR2(config-if)#**switchport nonegotiate**
SWR2(config-if)#**spanning-tree portfast**
SWR2(config-if)#**exit**
SWR2(config)#**int gig1/0/4**
SWR2(config-if)**no shutdown**
SWR2(config-if)#**switchport mode trunk**
SWR2(config-if)#**exit**

SWR2(config)#**no cdp run**
SWR2(config)#**logging 192.168.101.100**
SWR2(config)#**ntp server 192.168.101.100**
SWR2(config)#**do write**

2.10. Виконуємо налаштування маршрутизатора SWR3. Попередньо відкриваємо закладку Physical і вставляємо блок живлення у шасі пристрою (рисунок 4).

Switch>**enable**
Switch#**conf term**
Switch(config)#**hostname SWR3**
SWR3(config)#**username cisco privilege 15 password 0 cisco123!**
SWR3(config)#**ip domain-name example.com**
SWR3(config)#**crypto key generate RSA general-keys modulus 768**
SWR3(config)#**ip ssh version 2**
SWR3(config)#**line vty 0 4**
SWR3(config-line)#**login local**
SWR3(config-line)#**exit**

SWR3(config)#**ip name-server 192.168.101.100**
SWR3(config)#**int vlan 1**
SWR3(config-if)#**no shutdown**
SWR3(config-if)#**ip address 10.0.1.4 255.255.255.0**
SWR3(config-if)#**exit**
SWR3(config)#**ip default-gateway 10.0.1.1**

SWR3(config)#**int gig1/0/2**
SWR3(config-if)#**no shut**
SWR3(config-if)#**switchport mode trunk**
SWR3(config-if)#**exit**
SWR3(config)#**int gig1/0/4**
SWR3(config-if)#**no shut**

SWR3(config-if)#**switchport mode trunk**
SWR3(config-if)#**exit**
SWR3(config)#**int gig1/0/5**
SWR3(config-if)#**no shut**
SWR3(config-if)#**switchport mode trunk**
SWR3(config-if)#**exit**
SWR3(config)#**gig1/0/21**
SWR3(config-if)#**no shut**
SWR3(config-if)#**switchport mode access**
SWR3(config-if)#**switchport nonegotiate**
SWR3(config-if)#**spanning-tree portfast**
SWR3(config-if)#**exit**
SWR3(config)#**int gig1/0/22**
SWR3(config-if)**no shutdown**
SWR3(config-if)#**switchport mode access**
SWR3(config-if)#**switchport nonegotiate**
SWR3(config-if)#**spanning-tree portfast**
SWR3(config-if)#exit

SWR3(config)#**no cdp run**
SWR3(config)#**logging 192.168.101.100**
SWR3(config)#**ntp server 192.168.101.100**
SWR3(config)#**do write**

2.11. Виконуємо налаштування маршрутизатора SWR4. Попередньо відкриваємо закладку Physical і вставляємо блок живлення у шасі пристрою (рисунок 4).

Switch>**enable**
Switch#**conf term**
Switch(config)#**hostname SWR4**
SWR4(config)#**username cisco privilege 15 password 0 cisco123!**
SWR4(config)#**ip domain-name example.com**
SWR4(config)#**crypto key generate RSA general-keys modulus 768**
SWR4 (config)#**ip ssh version 2**
SWR4(config)#**line vty 0 4**
SWR4(config-line)#**login local**
SWR4(config-line)#**exit**

SWR4(config)#**ip name-server 192.168.101.100**
SWR4(config)#**int vlan 1**
SWR4(config-if)#**no shutdown**
SWR4(config-if)#**ip address 10.0.1.5 255.255.255.0**
SWR4(config-if)#**exit**
SWR4(config)#**ip default-gateway 10.0.1.1**

SWR4(config)#**int gig1/0/5**
SWR4(config-if)#**no shut**
SWR4(config-if)#**switchport mode trunk**
SWR4(config-if)#**exit**
SWR4(config)#**int gig1/0/23**
SWR4(config-if)**no shutdown**
SWR4(config-if)#**switchport access vlan 2**
SWR4(config-if)#**switchport mode access**
SWR4(config-if)#**switchport nonegotiate**
SWR4(config-if)#**spanning-tree portfast**
SWR4(config-if)#**exit**

SWR4(config)#**int gig1/0/24**
SWR4(config-if)**no shutdown**
SWR4(config-if)#**switchport access vlan 2**
SWR4(config-if)#**switchport mode access**
SWR4(config-if)#**switchport nonegotiate**
SWR4(config-if)#**spanning-tree portfast**
SWR4(config-if)#**exit**

SWR4(config)#**no cdp run**
SWR4(config)#**logging 192.168.101.100**
SWR4(config)#**ntp server 192.168.101.100**
SWR4(config)#**do write**

2.12. Налаштування інтерфейсів комп'ютерів Admin, PC1, PC2 та PC3 виконується за допомогою служб DHCP, що працюють на маршрутизаторі R3.

2.13. Після налаштування усіх пристроїв потрібно переконатися, що всі вони доступні один для одного. Для цього можна скористатися командним рядком на кожному пристрої або інструментом **Simple PDU** (P) – всі пристрої мають «пінгувати» один одного.

У наступних вправах ми порівняємо управління мережею за допомогою інтерфейсу командного рядка (CLI) з управлінням мережею з використанням програмно-керованого мережевого контролера (SDN-контролера).

2.14. Збирання інформації про пристрої. Тепер вручну, користуючись інтерфейсом командного рядка CLI, потрібно отримати доступ до кожного пристрою, щоб зібрати інформацію про версію програмного забезпечення. Для цього будемо використовувати раніше налаштований на мережевих пристроях протокол SSH.

Крок 1: На комп'ютері адміністратора **Admin** за допомогою протоколу SSH отримуємо доступ до комутатора SWR3.

а. Натискаємо **Admin** > **Desktop** > **Command Prompt**.

б. Вводимо команду **ssh -l cisco 10.0.1.4**. Опція -l – це літера «L».

в. Коли з 'явиться запит, вводимо пароль **cisco123!**. Тепер ми увійшли в систему комутатора SWR3.

Крок 2: Збираємо інформацію про програмне забезпечення SWR3.

а. Виконуємо команду **show version** з фільтром, щоб переглянути лише відомості про програмне забезпечення, встановлене на пристрої. Звертаємо увагу, що SWR3 працює під управлінням IOS 16.3.2 та Boot Loader 4.26.

SWR3#**show version | include RELEASE**

Cisco IOS Software [Denali], Catalyst L3 Switch Software (CAT3K_CAA-UNIVERSALK9-M), Version 16.3.2, RELEASE SOFTWARE (fc4)
BOOTLDR: CAT3K_CAA Boot Loader (CAT3K_CAA-HBOOT-M) Version 4.26, RELEASE SOFTWARE (P)

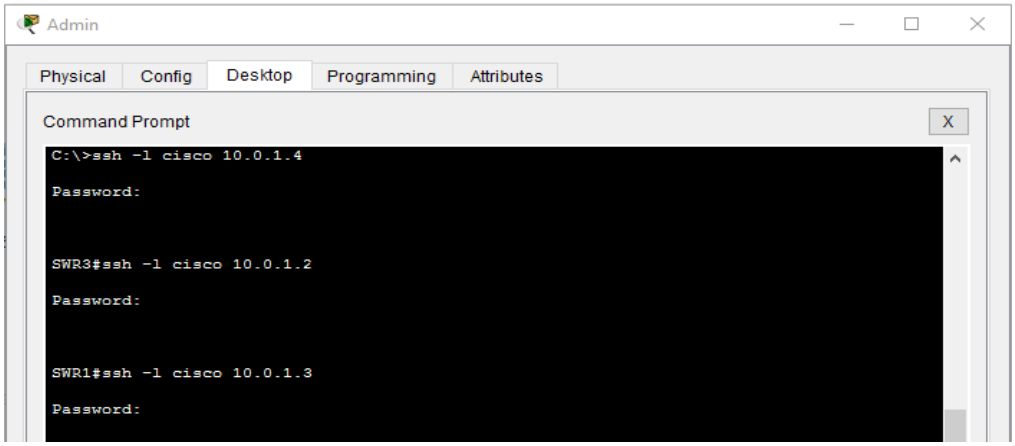
б. Копіюємо інформацію в буфер обміну.

в. Відкриваємо редактор текстових файлів і вставляємо інформацію в текстовий файл.

г. Зберігаємо файл як **software-versions.txt**.

Крок 3: Продовжуємо збирати інформацію про програмне забезпечення на решті мережевих пристроїв.

а. У командному рядку SWR3 за допомогою команди **ssh** отримуємо доступ до наступного мережевого пристрою, наприклад, SWR1 та повторюємо Крок 2.




б. Продовжуємо документувати версії програмного забезпечення, поки не завершимо роботу з усіма дев'ятьма мережевими пристроями: SWL1, SWL2, SWR1, SWR2, SWR3, SWR4, R1, R2 та R3.

в. Послідовно за допомогою команди **exit** виходимо з усіх сеансів SSH.

2.15. До налаштованої мережі додаємо мережевий контролер. Packet Tracer забезпечує простий PT-контролер для імітації контролера SDN.

Щоб дізнатися більше про реалізацію в Packet Tracer мережевого контролера, натискаємо меню **Help** (Довідка), а потім **Contents** (Зміст). У індексі ліворуч, знаходимо заголовок **Configuring Devices** (Налаштування пристроїв). Під цим заголовком знаходимо **Network Controllers** (Мережеві контролери). Тут можна знайти інформацію про налаштування мережевого контролера.

Крок 1: Додаємо мережевий контролер:

а. У нижньому лівому куті вікна Packet Tracer вибираємо панель **End Devices** і натискаємо на пристрій **Network Controller** (рисунок 5).


Рисунок 5 – Вибирання пристрою Network Controller

б. Додаємо мережевий контролер на вільне місце зліва від комутатора SWL1. Назва пристрою має бути PT-Controller0.

в. З'єднуємо прямим мідним кабелем інтерфейс GigabitEthernet0 контролера з інтерфейсом Gigabit Ethernet1/0/2 пристрою SWL1 (рисунок 6).

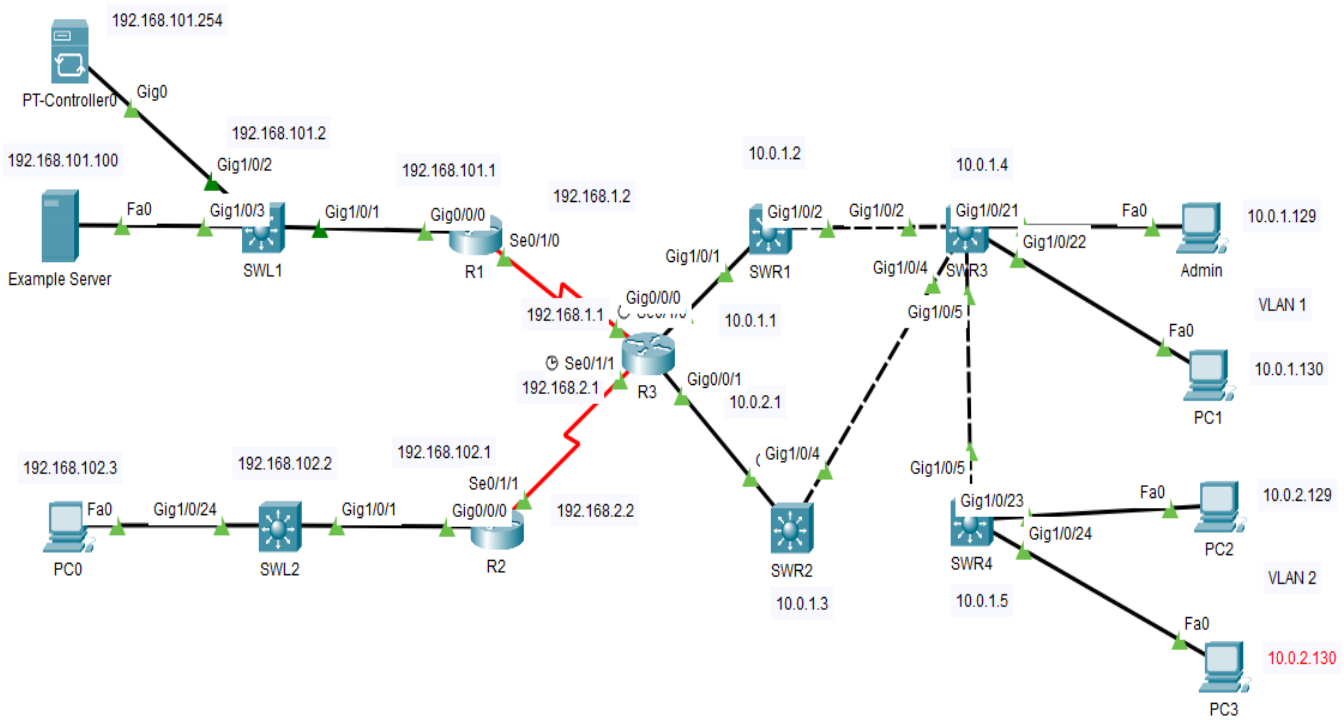
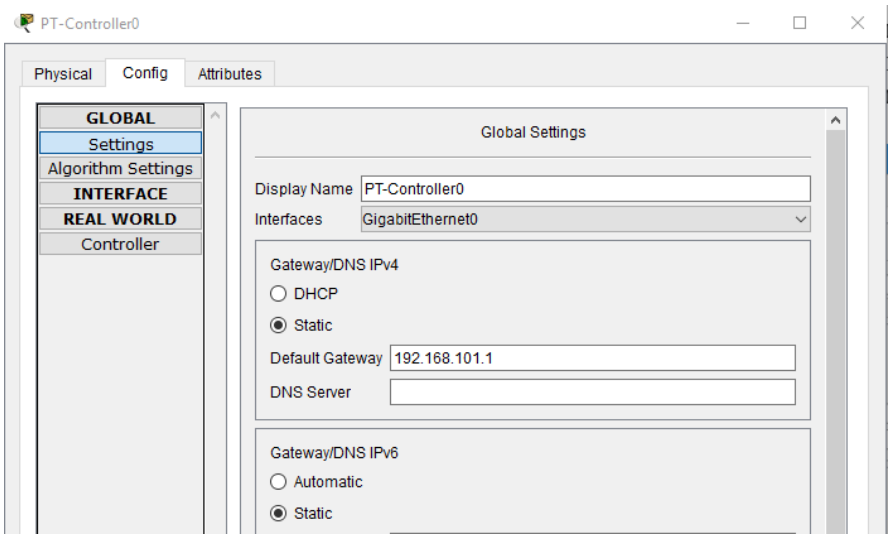


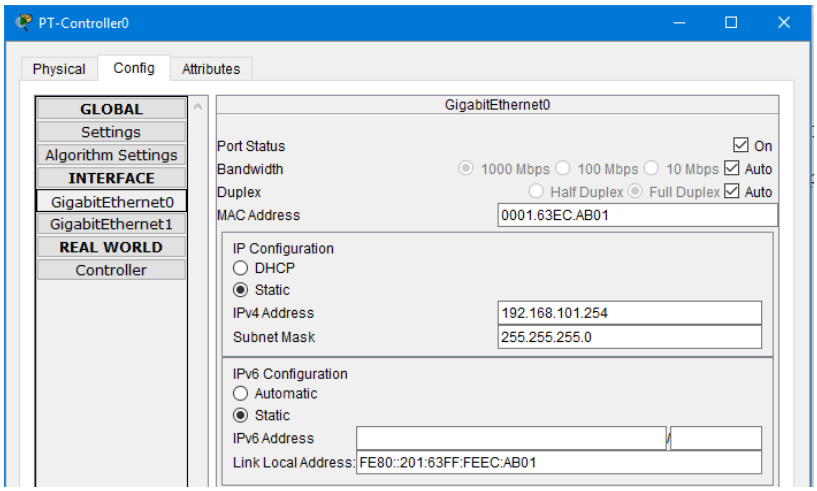

Рисунок 6 – Під’єднання пристрою Network Controller

Крок 2: Налаштуємо підключення для PT-Controller0.

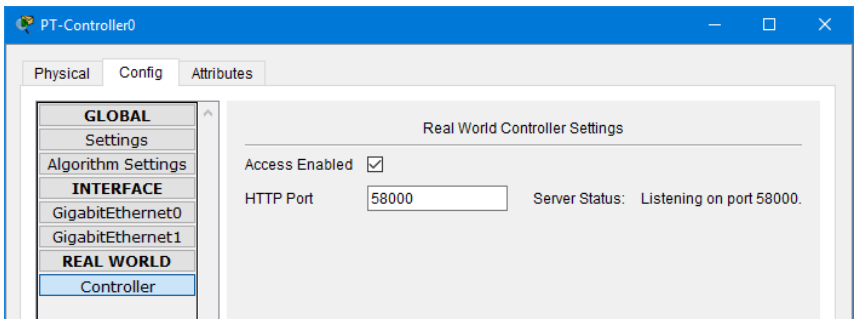
а. Клікаємо на PT-Controller0, відкриваємо закладку Config, переходимо в **Global** > **Settings**.
б. Призначаємо для **Default Gateway** IP-адресу 192.168.101.1.




в. Переходимо на **Interface** і вибираємо **GigabitEthernet0**.
г. Для конфігурування інтерфейсу вводимо IP-адресу 192.168.101.254 та маску підмережі 255.255.255.0

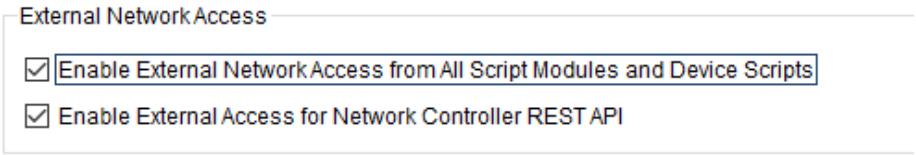



д. Переходимо на **REAL WORLD**, клікаємо **Controller.** Якщо **Server Status** в стані **Stop**, переходимо до підпункту є.




Якщо статус сервера в стані **Disabled in Preferences** (відключений у налаштуваннях), то в налаштуваннях Packet Tracer потрібно включити зовнішній доступ, дотримуючись таких інструкцій:

1) Вибираємо **Options** (параметри)> **Preferences** (налаштування) в меню Packet Tracer.
2) Клікаємо **Miscellaneous** (різне).
3) У розділі **External Network Access** вибираємо **Enable External Network Access** і **Enable External Access.**




4) Закриваємо налаштування та натискаємо **PT-Controller0> Config**, якщо це необхідно.
5) Зліва під **REAL WORLD**, клікаємо **Controller.**

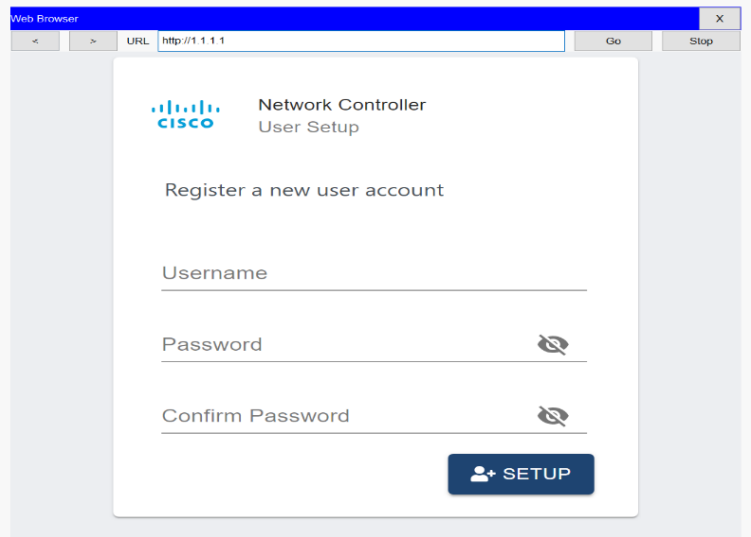
є. Стан сервера тепер **Stopped**. Клікаємо **Access Enabled**, щоб увімкнути його. Статус сервера змінюється на **Listening on port 58000 (**прослуховування на порту 58000). Це номер порту в скриптах Python.

Крок 3: З комп'ютера **Admin** перевіряємо підключення до PT-Controller0.

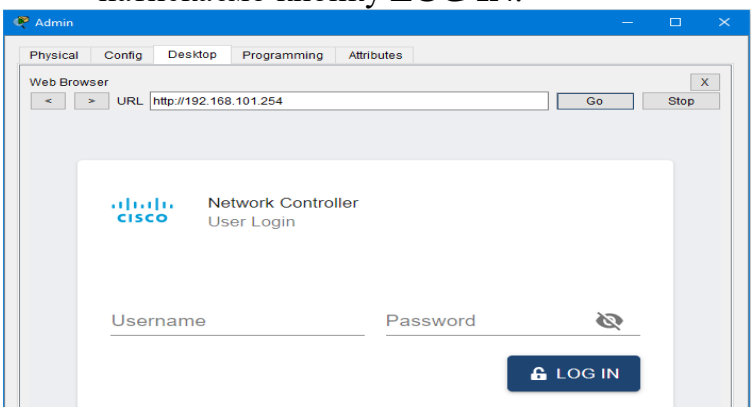
а. Клікаємо **Admin > Desktop > Web Browser**.

б. Вводимо адресу IPv4 192.168.101.254, щоб отримати доступ до User Setup в PT-Controller0.

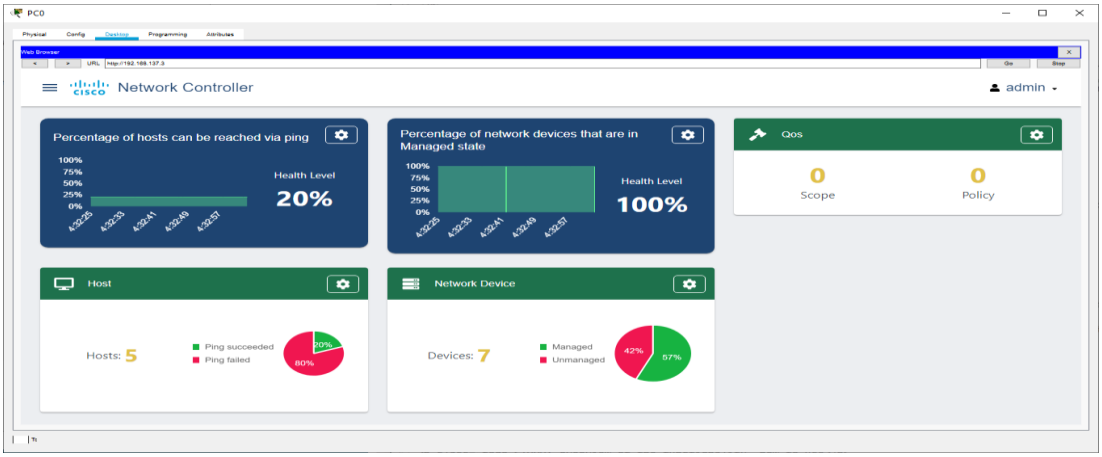
в. Вводимо **cisco** в полі **Username** та **cisco123!** у полях **Password** та **Confirm Password**, а потім натискаємо **Setup**.




г. На екрані входу **User Login** користувача вводимо наші облікові дані та натискаємо кнопку **LOG IN**.



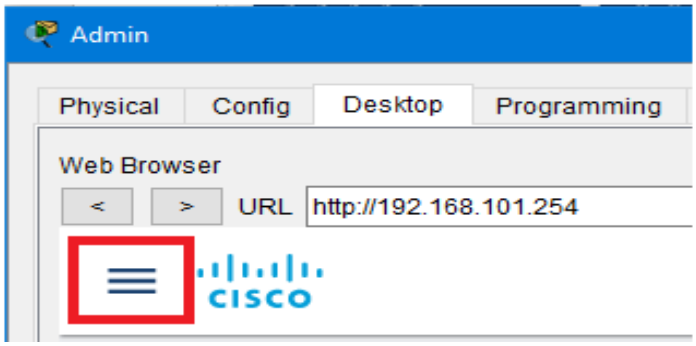

д. Зараз ми увійшли на приладову панель Dashboard для PT-Controller0. У цей момент може бути корисно розширити вікно, щоб можна було бачити всю приладову панель.



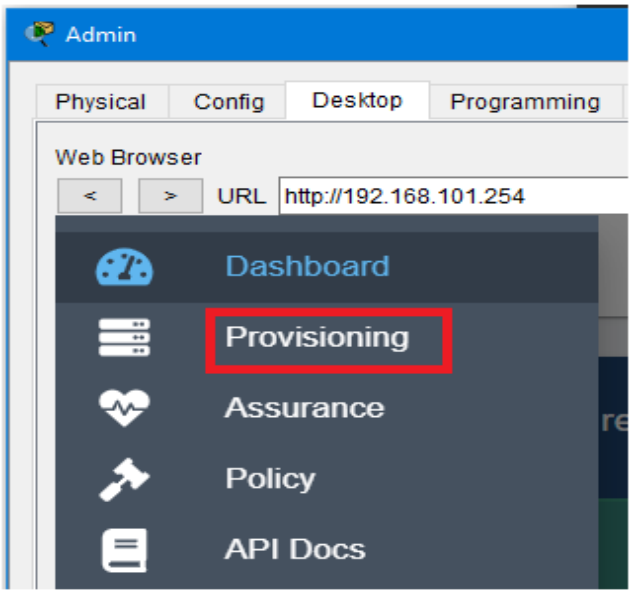

2.16. Використаємо контролер SDN, щоб визначити топологію нашої мережі. У цій частині ми налаштуємо PT-Controller0 для використання протоколу Cisco Discover Protocol (CDP) для автоматичного пошуку дев'яти мережевих пристроїв у нашій топології. PT-Controller0 також відкриє всі п’ять хост-пристроїв, приєднаних до мережі.

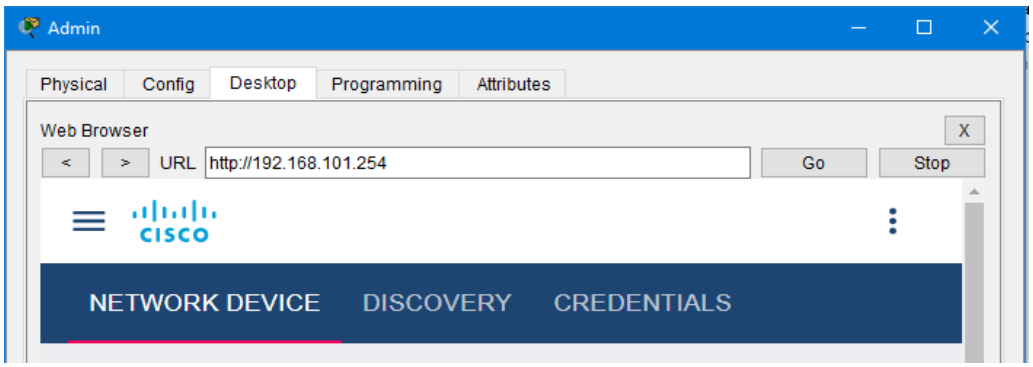
Крок 1: Додаємо облікові дані **Credentials** для доступу до всіх мережевих пристроїв у топології.

а. У графічному інтерфейсі мережевого контролера натискаємо кнопку меню зліва від логотипу Cisco.




б. Вибираємо **Provisioning**. У цьому розділі можна уручну додавати пристрої для контролю, виявляти нові пристрої та створювати глобальні облікові дані, які використовуються для доступу до цих пристроїв. Однак ми будемо використовувати протокол CDP для автоматичного виявлення пристроїв.

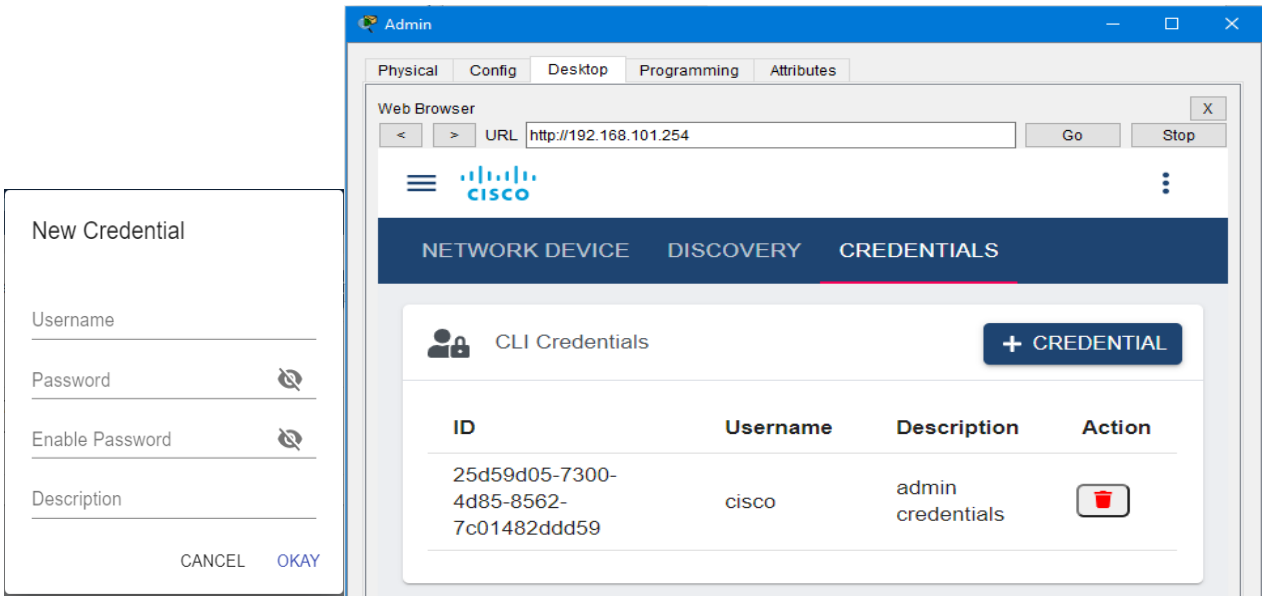



в. Клікаємо на **CREDENTIALS** (облікові дані), а потім натискаємо **+ CREDENTIAL**, щоб додати новий обліковий запис.

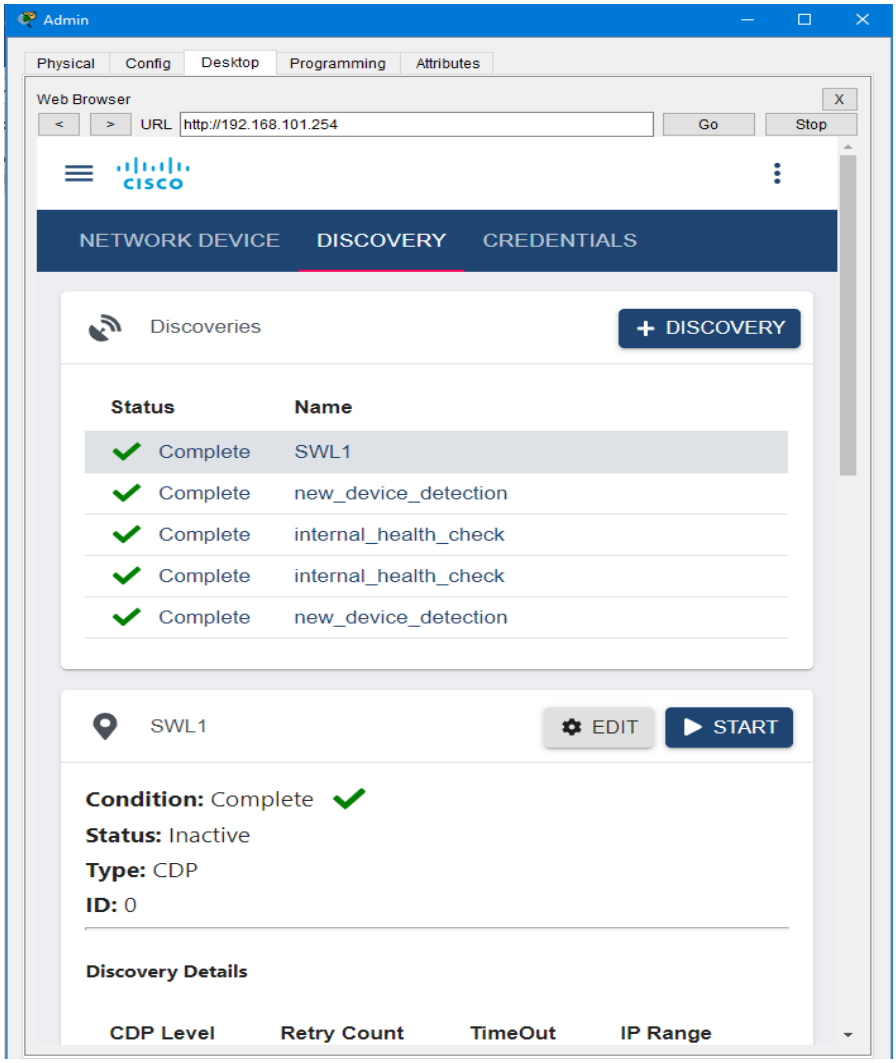
г. Для імені користувача вводимо **cisco,** для пароля вводимо **cisco123!**. Залишаємо **Enable Password** порожнім. Для опису вводимо **admin credentials**, а потім натискаємо **OKAY**.

д. Нові облікові дані тепер зберігаються на PT-Controller0 для використання в завданнях.

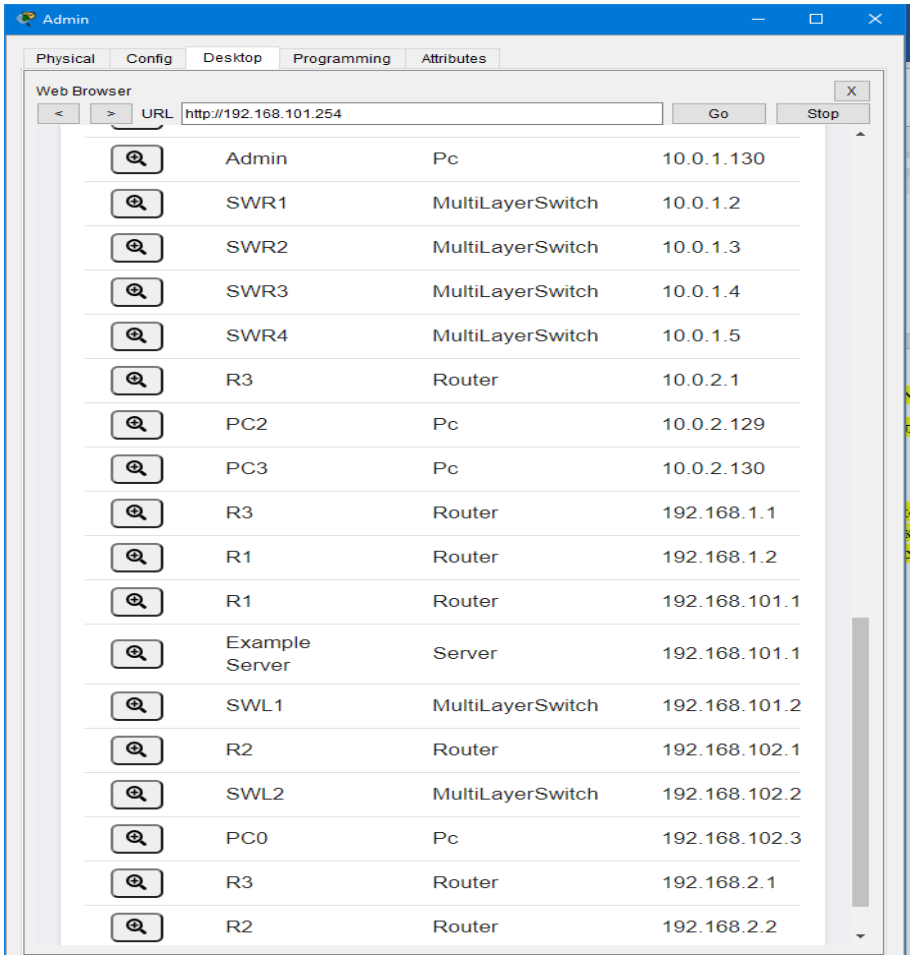



Крок 2: Використаємо протокол CDP, щоб визначити всі пристрої в мережі.

а. Клікаємо **Discovery**, потім **+ Discovery**, щоб створити новий пошук.
б. Для **Name** вводимо **SWL1**. Для IP-адреси вводимо 192.168.101.2. Для **Credential List** вибираємо **cisco - admin credentials**.
в. Клікаємо **ADD**
г. Тепер можемо спостерігати процес визначення пристроїв в мережі (процес триває деякий час).



(продовження)

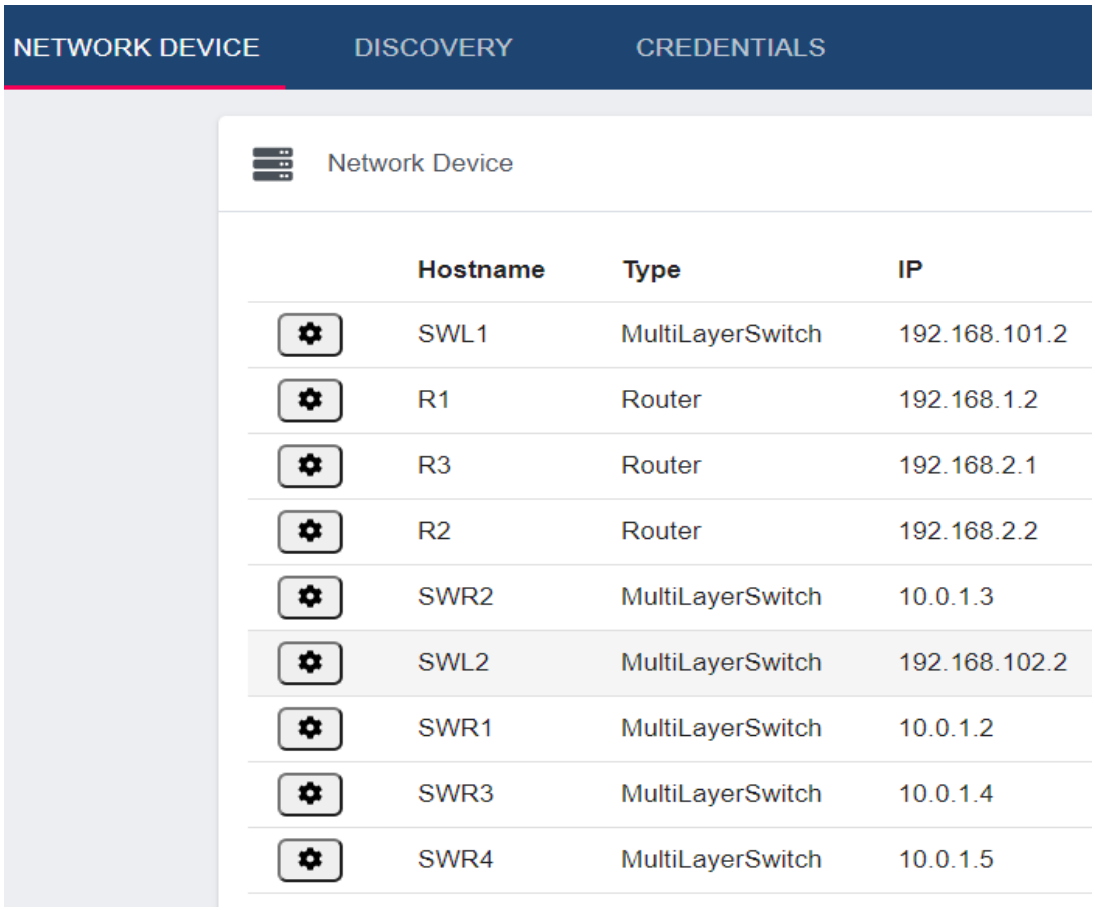



2.17. Тепер розглянемо використання контролера SDN для збору інформації. Будемо використовувати GUI PT-Controller0 для перегляду інформації про мережеві пристрої та хост-пристрої в топології. Також переглянемо топологію, створену контролером і переглянемо всю мережу.

Крок 1: Переглянемо список виявлених мережевих пристроїв.

а. Клікаємо **NETWORK DEVICE**. Тепер можна побачити всі виявлені дев'ять мережевих пристроїв.

б. Клікаємо значок поруч із іменем хоста будь-якого пристрою, щоб побачити інформацію, зібрану процесом **Discover**. Зверніть увагу, що в списку міститься версія програмного забезпечення, а також інша детальна інформація про пристрій.




Крок 2: Переглянемо список усіх виявлених пристроїв-хостів.

а. Повертаємося на приладову панель. Клікаємо меню поруч із логотипом Cisco, а потім натискаємо **Dashboard** (щоб повернутися на приладову панель з будь-якого місця можна просто натиснути на банер **Network Controller**).

б. На приладовій панелі можна побачити діаграми з кількістю хостів, яких можна дістатися за допомогою команди **ping** та кількість мережевих пристроїв, які керуються контролером. Обидва показники повинні мати значення 100%.

в. На приладовій панелі також можна бачити панелі для **QoS**, **Network Device** та **Host.** Клікаємо значок  на панелі **Host**. Відбувається перехід на вкладку **Hosts** для **ASSURANCE**.

У розділі **ASSURANCE** можна побачити стан мережевих пристроїв (включаючи кінцеві пристрої) та список усіх хостів. Цікавим є також представлення топології мережі (без кінцевих пристроїв), і, нарешті, можна застосувати трасування шляху між хостами мережі, щоб побачити деталі взаємодії між ними.

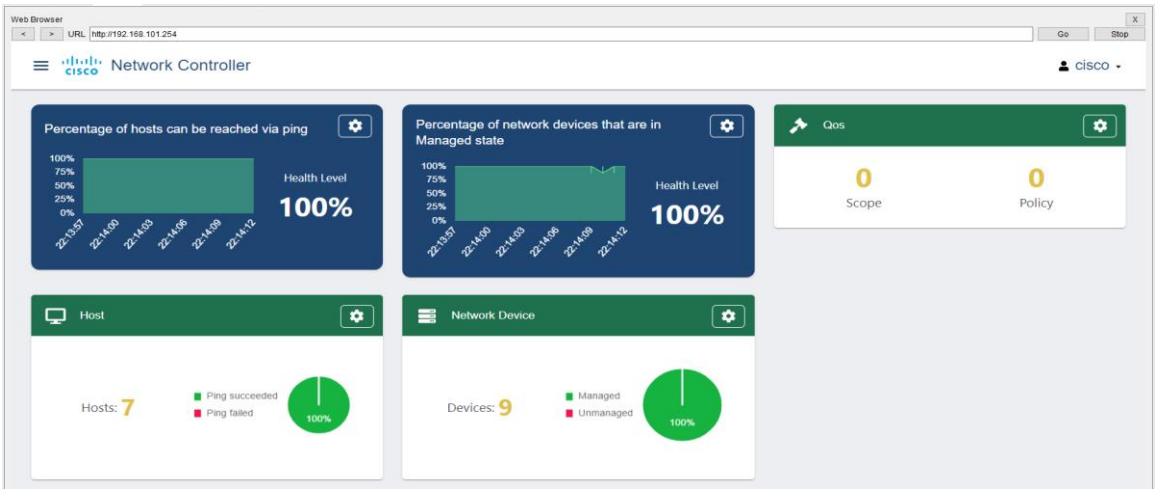
Натискаємо **HOSTS** щоб перейти до розділу **HOSTS**.

Host

| | Host Device | | | | Connected Network Device | | |
|---|---|---|---|---|---|---|---|
| | MAC | IP | Hostname | Type | IP | Hostname | Port |
| ⚙ | 000C.CF34.AC5E | 192.168.101.100 | Example Server | Server | 192.168.101.2 | SWL1 | GigabitEthernet1/0/3 |
| ⚙ | 0001.634D.27E8 | 192.168.102.3 | PC0 | Pc | 192.168.102.2 | SWL2 | GigabitEthernet1/0/24 |
| ⚙ | 0060.5C49.324E | 10.0.2.129 | PC2 | Pc | 10.0.1.5 | SWR4 | GigabitEthernet1/0/23 |
| ⚙ | 0090.217E.2C3E | 10.0.2.130 | PC3 | Pc | 10.0.1.5 | SWR4 | GigabitEthernet1/0/24 |
| ⚙ | 0090.2B4E.7B94 | 10.0.1.130 | Admin | Pc | 10.0.1.4 | SWR3 | GigabitEthernet1/0/21 |
| ⚙ | 0090.213C.B161 | 10.0.1.129 | PC1 | Pc | 10.0.1.4 | SWR3 | GigabitEthernet1/0/22 |

г. На цій сторінці можна переглянути всю інформацію про підключення рівня 2 та рівня 3 для кожного хоста, а також мережевий пристрій, до якого він підключений

д. Клікаємо значок ⚙ поруч із будь-яким хостом, щоб переглянути більш детальну інформацію.

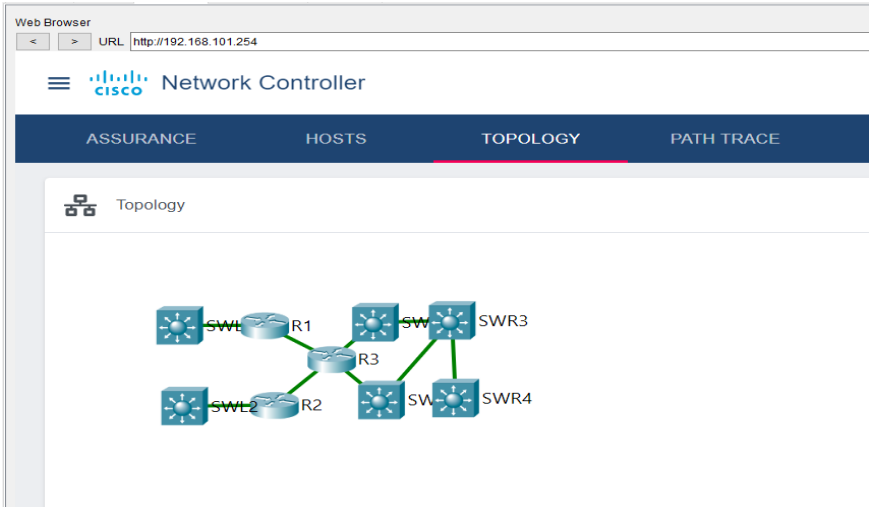



Крок 3: Переглянемо топологію, створену PT-Controller0.

а. Натискаємо на вкладку **TOPOLOGY**. Зверніть увагу, що PT-Controller динамічно створив ту ж топологію, яку можна бачити у головному вікні Packet Tracer.

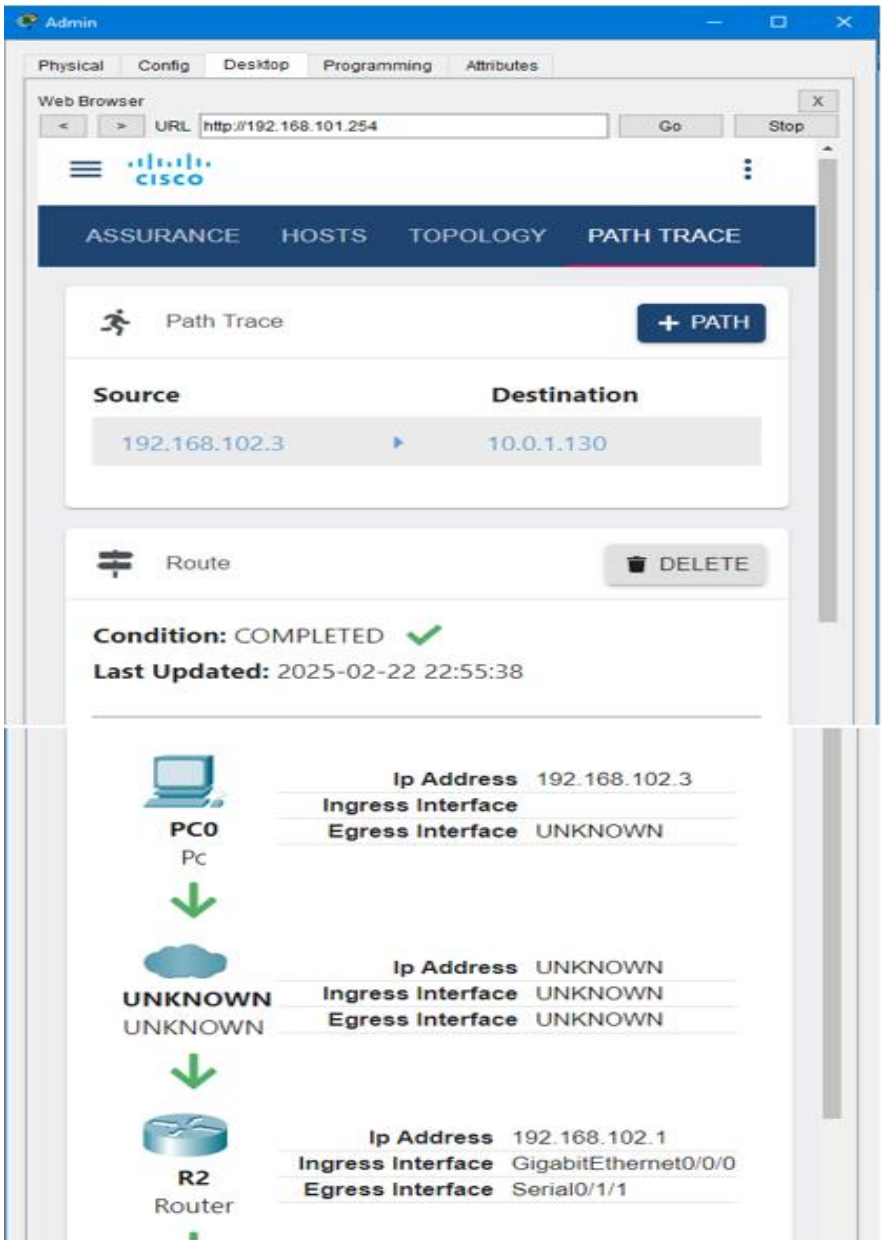
б. В цьому місці ми можемо натиснути на будь-який мережевий пристрій, щоб побачити детальні відомості про нього.

в. Також в цьому місці можна натиснути та перетягнути значки пристроїв, щоб змінити топологію. Однак такі зміни не будуть збережені, коли ми покинемо робочий простір **TOPOLOGY**.




Крок 4: Простежимо шлях від одного пристрою до іншого.

а. Натискаємо вкладку PATH TRACE.
б. Натискаємо + PATH, щоб додати новий шлях.
в. Простежимо шлях від одного кінця мережі до іншого. Наприклад, введемо IP-адреси для PC1 до PC0. Потім натискаємо OKAY і значок .

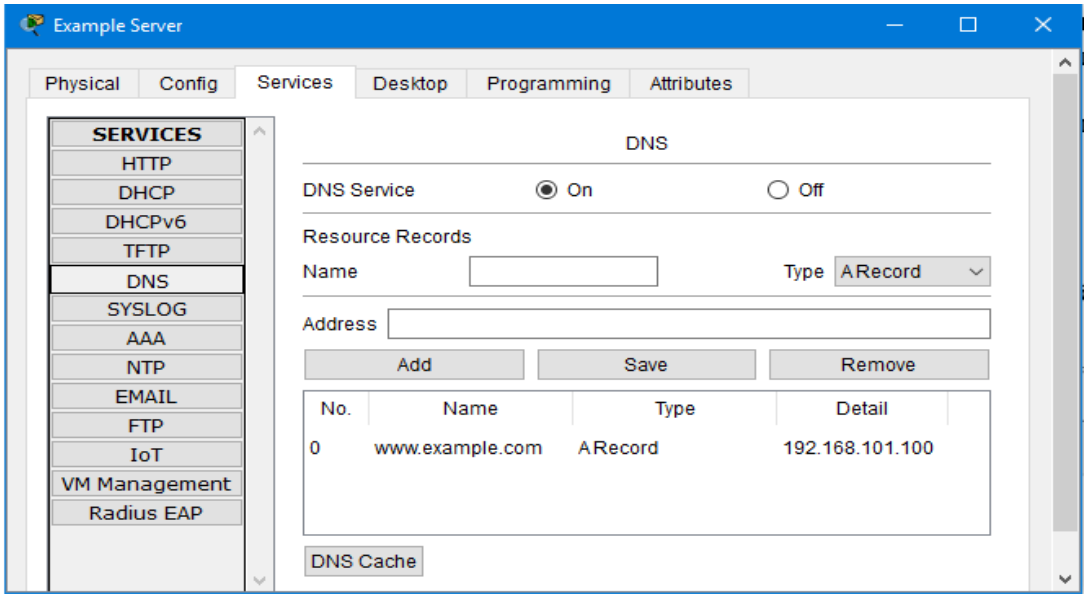


г. Виділяємо курсором новий шлях, який був доданий, щоб відобразити його трасування. Отримуємо звіт про маршрут, який показує всі переходи від джерела до призначення. Зверніть увагу, що вказана лише інформація про пристрої рівня 3. Комутатори відображаються як UNKNOWN (невідомий) пристрій. Це пов'язано з тим, що зараз вони працюють лише на рівні 2.

2.18. Розглянемо використання контролера SDN для налаштування мережевих параметрів. Основною перевагою автоматизації керування мережею за допомогою контролера є можливість налаштування глобальних мережевих параметрів і політик для всіх пристроїв, а потім застосування цього налаштування одним натисканням кнопки. У цій частині ми налаштуємо PT-Controller0 з мережевими параметрами для DNS, NTP та Syslog. Потім ми застосуємо це налаштування до підтримуваних мережевих пристроїв. Нарешті, ми перевіримо та протестуємо політику.

Крок 1: Перевірка роботи сервісів на пристрої **Example server**.

а. Натискаємо **Example server** > **Services**.
б. У розділі **Services** натискаємо DNS. Звертаємо увагу, щоб служба DNS була увімкнена і щоб був один запис типу A для www.example.com.




в. У розділі **Services** натискаємо **SYSLOG**. Звертаємо увагу, щоб служба **Syslog** була увімкнена.
г. У розділі **Services** натискаємо **NTP**. Звертаємо увагу, щоб служба NTP була увімкнена.

Крок 2: У розділі **Network Settings** можна налаштувати однакові глобальні параметри для кожного керованого пристрою. Наприклад, DNS, NTP та різні інші параметри можуть бути застосовані глобально до всіх керованих пристроїв. Налаштуємо глобальну політику для **DNS**, **SYSLOG** та **NTP**.

а. Натискаємо **Admin**. Відкриємо Web Browser, якщо він був закритий і під’єднаємося до PT-Controller0.
б. Натискаємо меню ліворуч від логотипу Cisco.
в. Натискаємо **Policy**.

г. На вкладці **QOS** звертаємо увагу, що є параметри для налаштування SCOPE та POLICY. Тепер ми налаштуємо **NETWORK SETTINGS**.
д. Натискаємо **NETWORK SETTINGS**.
є. Натискаємо **DNS**. Вводимо **example.com** як **Domain Name** та **192.168.101.100** як IP-адресу. Натискаємо **Save**.
ж. Натискаємо **NTP**.
і. Вводимо **192.168.101.100** як IP-адресу. Натискаємо **Save**.
к. Натискаємо **SYSLOG**.
л. Вводимо **192.168.101.100** як IP-адресу. Натискаємо **Save**
м. Натискаємо **DNS**, **NTP** та **SYSLOG** ще раз, щоб перевірити правильність інформації. Якщо ні, виправляємо інформацію, зберігаючи кожного разу.
н. Натискаємо **PUSH CONFIG**.
о. Відкривається діалогове вікно **Push All Network Settings**. Перевіряємо налаштування та натискаємо **OKAY**. На короткий час з'являється повідомлення "Saved Successfully".

Крок 3: Перевіримо та протестуємо мережеві налаштування, які були надіслані на пристрої.

У нижній частині вікна **NETWORK SETTINGS** є наступна примітка:

Примітка: Ця функціональність підтримується тільки на пристроях, що працюють під управлінням IOS-XE OS та Switch 2960-24TT. Це означає, що для цієї версії Packet Tracer наші глобальні налаштування були застосовані лише до маршрутизаторів та комутаторів.

а. Клікаємо на будь-якому із трьох маршрутизаторів, наприклад R1.
б. Вибираємо **CLI**.
в. У вікні, що з’явилося, натискаємо Enter, щоб отримати командний рядок.
г. Входимо у привілейований режим і перевіряємо налаштування DNS.

```
R1>enable
R1#show run | begin ip domain
ip domain-name example.com
ip name-server 192.168.101.100
!
```

д. Вводимо наступні команди, щоб перевірити налаштування NTP. Час на R1 має відповідати поточному часу. Packet Tracer’у потрібний певний час для поширення NTP-повідомлень.

R1# **show ntp associations**

```
R1#show ntp associations

address         ref clock       st   when     poll    reach  delay          offset
disp
*~192.168.101.100127.127.1.1     1    23       32      45     2.00           0.00
0.48
 * sys.peer, # selected, + candidate, - outlyer, x falseticker, ~ configured
```

R1# **show clock**

```
R1#sh clock
10:0:31.557 UTC Sat Mar 22 2025
R1#
```

є. Щоб перевірити, чи налаштовано ведення журналу, виконаємо наступну команду:

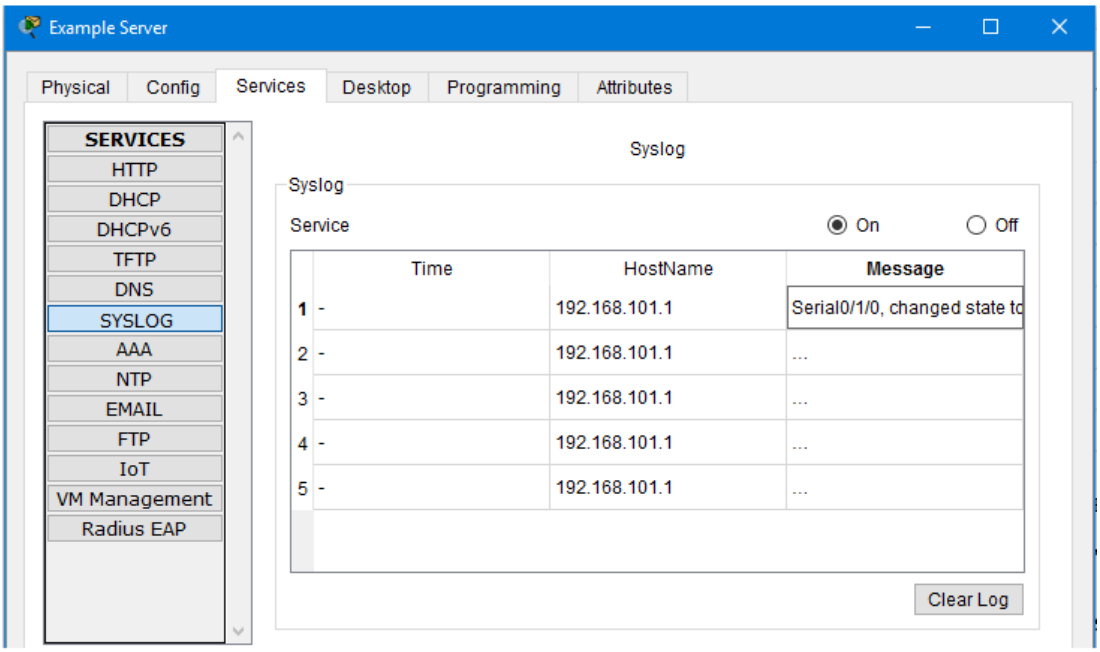
```
R1# show run | include logging
```

```
R1#show run | include logging
logging 192.168.101.100
R1#
```

ж. Щоб протестувати роботу журналювання, за допомогою CLI вимикаємо інтерфейс Serial0/1/0, а потім знову активуємо його.

```
R1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# interface s0/1/0
R1(config-if)# shutdown
%LINK-5-CHANGED: Interface Serial0/1/0, changed state to administratively down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1/0, changed state to down
15:36:37: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.2.1 on Serial0/1/0 from FULL to DOWN,
Neighbor Down: Interface down or detached
R1(config-if)# no shutdown
%LINK-5-CHANGED: Interface Serial0/1/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1/0, changed state to up
15:36:53: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.2.1 on Serial0/1/0 from LOADING to
FULL, Loading Done
R1(config-if)# end
R1#
```

і. Натискаємо **Example Server > Services > SYSLOG**. На сервері ми маємо побачити ті ж повідомлення syslog, які ми бачили в CLI. Двічі клікаємо на будь-якому запису, щоб переглянути текст повідомлення.




Отже, реєстрація мережевих подій в системному журналі відбувається.

Виконання лабораторної роботи завершене.

Контрольні запитання

1. Технологія програмно-керованих мереж SDN. Призначення, переваги та недоліки мереж SDN.
2. Переваги та недоліки керування SDN-мережею засобами командного рядка та засобами графічної оболонки SDN.

Звіт про виконання лабораторної роботи має містити:

1. Налаштовану в Packet Tracer модель мережі з SDN-контролером.
2. Файл з відомостями про програмне забезпечення мережевих пристроїв software-versions.txt.

Додаток 1

Приклад налаштування SSH в пристроях Cisco.

1. cisco> **enable**
2. cisco# **show clock**
   cisco# **clock set 17:10:00 28 Mar 2025**
3. cisco# **configure terminal**
4. cisco(config)# **ip domain name test.dom**
5. cisco(config)# **crypto key generate rsa general-keys modulus 768**
6. cisco(config)# **username user privilege 15 password 7 Pa$$w0rd**
7. cisco(config)# **line vty 0 4**
8. cisco(config-line)# **transport input ssh**
9. cisco(config-line)# **login local**
10. cisco(config-line)# **exec-timeout 60 0**
11. cisco(config-line)# **end**
12. cisco# **copy running-config**

Пояснення:

1. Входимо в привілейований режим.
2. Перевіряємо і за потреби встановлюємо точний час для генерації ключа.
3. Входимо в режим конфігурування.
4. Вказуємо ім’я домена (необхідне для генерації ключа).
5. Генеруємо RSA ключ (можна вибрати розмір ключа).
6. Вказуємо користувача з ім’ям user, паролем Pa$$w0rd і рівнем привілеїв 15.
7. Входимо в режим конфігурування термінальних ліній з 0 по 4.
8. Вказуємо спосіб доступу до мережі за замовчуванням SSH (за потреби).
9. Автентифікуємо всі вхідні сеанси через локальну базу даних імен користувачів, створених за допомогою команди **username**.
10. Вказуємо час таймауту для автоматичного закриття SSH сесії через 60 хвилин (за потреби).
11. Виходимо із режиму конфігурування термінальних ліній.
12. Зберігаємо конфігураційний файл в енергонезалежну пам’ять.

# Лабораторна робота 2

## Основи технології програмно-керованих мереж. Реалізація REST API за допомогою контролера SDN в Cisco Packet Tracer

**Мета роботи**. Ознайомитися з принципами створення і використання сценаріїв автоматизованого керування мережею за допомогою програмованого контролера мережі; дослідити побудовану на віртуальній машині DEVASC VM модель мережі з мережевим контролером шляхом надсилання запитів керування REST API із застосуванням додатків Postman та Visual Studio Code.

### 1. Теоретичні відомості

Програмно-керована мережа (Software Defined Networks - SDN) – мережа передачі даних, в якій рівень управління мережею відокремлений від пристроїв передачі даних і реалізується програмно. У ролі «мозку» мережі зазвичай виступає контролер. Він централізовано керує мережевими пристроями, збираючи з них різні телеметричні параметри (дані по трафіку, завантаженню обладнання, якості зв'язку тощо). Завдяки такій централізації контролер забезпечує автоматизацію в мережі, глибоку аналітику її роботи, безпеку, а також може реалізовувати досить складну логіку обробки мережевого трафіку.

З'єднання між контролером мережі та елементами мережі є двостороннім. Обидві сторони використовують інтерфейси програмування додатків (APIs), які дозволяють контролеру мережі та мережевим пристроям спілкуватися програмно. Контролер мережі взаємодіє з мережевими пристроями через південні APIs, а елементи мережі взаємодіють з контролером мережі через північні APIs.

Південні програмні інтерфейси (SDN southbound APIs) використовуються для зв'язку між контролером SDN та комутаторами і маршрутизаторами мережі. Вони можуть бути з відкритим кодом або пропрієтарними. Southbound APIs полегшують контроль за мережею та дозволяють контролеру SDN динамічно вносити зміни в мережу відповідно до вимог і потреб у реальному часі.

Північні програмні інтерфейси (SDN Northbound APIs) – це зазвичай RESTful APIs SDN, які використовуються для зв'язку між контролером SDN та службами і додатками, що працюють у мережі. Ці APIs можуть бути використані для сприяння ефективному автоматизованому керуванню в мережі, щоб відповідати потребам різних додатків. Northbound APIs виконують зв'язок між додатками та контролером SDN. Додатки можуть надавати мережі інформацію про дані, пропускну здатність тощо, а мережа може надати ці ресурси або повідомити про свої можливості.

Північний інтерфейс використовується для доступу до самого контролера SDN. Це дозволяє адміністратору отримати доступ до SDN, щоб налаштувати його або отримати від нього інформацію. Це можна зробити за допомогою графічного інтерфейсу, але він також пропонує API, який дозволяє іншим програмам отримувати доступ до контролера SDN.

Подібний підхід дозволяє використовувати APIs для написання сценаріїв і автоматизації мережевого адміністрування.

Ось деякі приклади:

- Вивести інформацію з усіх мережевих пристроїв у мережі.
- Показати стан всіх фізичних інтерфейсів в мережі.
- Додати нову VLAN на всіх комутаторах мережі.
- Показати топологію мережі.
- Автоматично налаштувати IP-адреси, маршрутизацію і списки доступу при створенні нової віртуальної машини.

За допомогою API додатки можуть отримати доступ до контролера SDN шляхом:

- Використання графічного інтерфейсу для отримання інформації про мережу з контролера SDN.
- Використання скриптів, написаних на Java або Python, для отримання інформації з контролера SDN або для налаштування мережі.
- Використання інших додатків для отримання доступу до контролера SDN.

Мережевий контролер, керований за допомогою вебінтерфейсу користувача або API, забезпечує централізовану інформаційну панель для перегляду стану мережі, що дозволяє адміністратору мережі швидко виявляти та усувати неполадки, а також надсилати зміни конфігурації на всі керовані пристрої одночасно.

Cisco Packet Tracer 8.2.2 SDN Controller надає REST APIs, які дозволяють програмувати мережу за допомогою мови Python. Оскільки доступ до APIs можливий з хоста, адміністратор мережі може скористатися можливостями реального середовища розробки та програмувати емульовану мережу Cisco Packet Tracer за допомогою Microsoft VS Code та Windows Subsystem Linux.

Для виконання сценаріїв керування мережею доступ до мережевого контролера можна отримати з реальних програм, які працюють на головному комп'ютері, таких як веб-браузер, VS Code, Python, curl, Postman. Зовнішній доступ до контролера SDN повинен бути включений в налаштуваннях Cisco Packet Tracer 8.2.2, перш ніж його буде включено на вкладках конфігурації пристрою SDN controller.

**REST API** — це спосіб взаємодії вебсайтів та вебдодатків із сервером. Термін складається з двох абревіатур, які розшифровуються наступним чином:

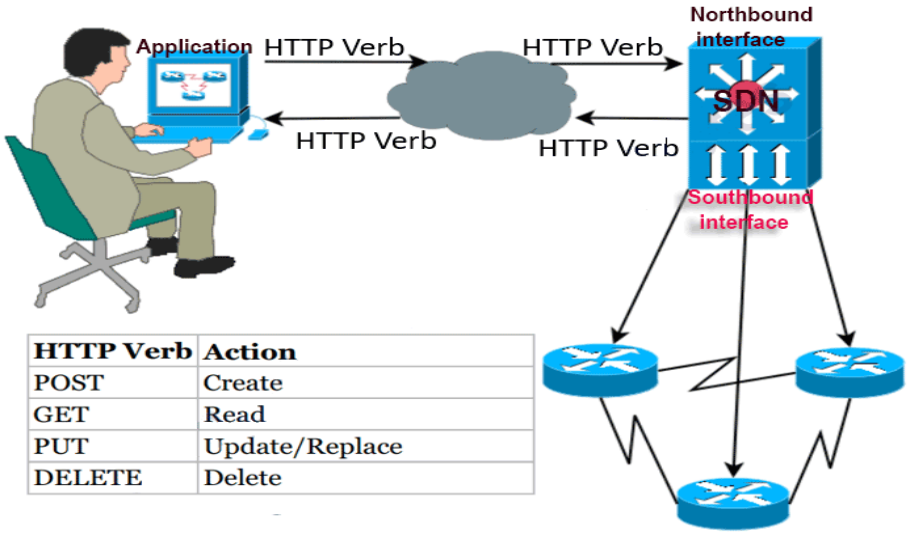
- **API** (Application Programming Interface) – це код, який дозволяє двом програмам обмінюватися даними з сервером.
- **REST** (Representational State Transfer) – технологія створення API за допомогою протоколу HTTP.

Технологію REST API застосовують скрізь, де користувачу сайту або вебдодатку потрібно надати дані із сервера. Сервіс, написаний з дотриманням всіх вимог REST, називають RESTful.

REST API оперують ресурсами. Ресурсом може бути будь-яка інформація, якій можна присвоїти ім'я, наприклад, документ або зображення, набір інших ресурсів, невіртуальний об'єкт тощо. У той же час REST використовує ідентифікатор ресурсу для розпізнавання конкретного ресурсу, що бере участь у взаємодії між компонентами. Існує чотири основні методи використання ресурсів, пов'язаних з REST API:

**GET –** дозволяє серверу знайти запитувані дані та надіслати їх у відповідь.

**PUT –** якщо виконати запит «PUT», сервер оновить запис у базі даних.
**POST –** цей метод дозволяє серверу створити новий запис у базі даних.
**DELETE –** цей метод дозволяє серверу видалити запис у базі даних.




| HTTP Verb | Action |
|---|---|
| POST | Create |
| GET | Read |
| PUT | Update/Replace |
| DELETE | Delete |

Коли клієнтський запит виконується через RESTful API, він передає звіт про стан ресурсу відправнику запиту або кінцевій точці. Цей звіт надається в одному з декількох форматів через HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP або звичайний текст.

**Інструменти REST**

**Postman** – одним із популярних клієнтів HTTP є інструмент під назвою Postman. Він має дуже простий інтерфейс користувача і дуже простий у використанні. Postman – це платформа для створення та використання API. Postman спрощує спільну роботу, дозволяючи швидко створювати досконалі API-інтерфейси.

**HTTPX** – це сучасний, кросплатформний HTTP-клієнт, написаний на Python. Він призначений для спрощення взаємодії з вебсервісами через CLI та забезпечення зручності для користувачів. Його прості HTTP-команди дозволяють користувачам надсилати HTTP-запити, використовуючи інтуїтивно зрозумілий синтаксис.

**Python Requests** можна використовувати для надсилання HTTP-запитів. Це зручна бібліотека з безліччю можливостей, починаючи від передачі параметрів в URL-адресах до надсилання власних заголовків і перевірки SSL. Бібліотека Requests – це дуже корисний інструмент, який можна використовувати щоразу, коли використовуються будь-які API.

**Visual Studio Code**, який зазвичай називають VS Code, – це інтегроване середовище розробки компанії Microsoft для Windows, Linux, macOS та веббраузерів. Функції включають підтримку редагування коду, підсвічування синтаксису, інтелектуального завершення коду, навігації, а також легку інтеграцію з існуючими інструментами. Microsoft VS Code також можна використовувати для спрощення програмування SDN API на Python 3.

Програмно-керований мережевий контролер PT-Controller, вбудований в Cisco Packet Tracer 8.2.2 для реалізації концепцій SDN, можна запрограмувати за допомогою REST API, доступ до якого можна отримати за допомогою мов програмування, наприклад, Python 3.

Попередження: код Python, наданий у прикладних файлах **pkt** Cisco Packet Tracer, не працює при використанні в реальному середовищі Python, оскільки деякі бібліотеки Python, емульовані в Packet Tracer, відсутні в Python 3.

Розширення Postman VS Code дозволяє розробляти та тестувати API в Postman безпосередньо з Visual Studio Code. Це спрощує робочий процес розробки, оскільки можна швидко протестувати нові API в тому ж додатку, який використовується для розробки. Це дозволяє контролювати кожен етап життєвого циклу API.

**JSON** (JavaScript Object Notation) – це структура даних, яка походить від мови програмування Java, але її можна використовувати як портативну структуру даних для будь-якої мови програмування. JSON широко використовується у вебсервісах і є одним з основних форматів даних, які потрібно знати, щоб взаємодіяти з інфраструктурою Cisco. Структура даних побудована на основі пар ключ/значення, які спрощують зіставлення і пошук даних.

**DEVASC** (The Developing Applications and Automating Workflows Using Cisco Core Platforms) – це спеціально створена віртуальна машина на базі Ubuntu 20.04 Linux, яка дозволяє досліджувати мережеві програми, використовуючи платформи Cisco як основу, впроваджувати елементи автоматизації в процес налаштування мережі та заходи по забезпеченню мережевої безпеки.

Робота в DEVASC дозволяє отримати практичний досвід вирішення реальних завдань з використанням інтерфейсів прикладного програмування Cisco (API) і сучасних засобів розробки програм керування мережею.

## 2. Виконання роботи

План виконання лабораторної роботи

Частина 1. Підготовка комп'ютера до віртуалізації – встановлення VirtualBox.
Частина 2. Запуск віртуальної машини DEVASC.
Частина 3. Знайомство з графічним інтерфейсом віртуальної машини DEVASC.
Частина 4. Перевірка зовнішнього підключення до Packet Tracer.
Частина 5. Запит маркера автентифікації за допомогою Postman.
Частина 6. Надсилання запитів REST за допомогою Postman.
Частина 7. Надсилання запитів REST за допомогою VS Code.
Частина 8. Надсилання запитів REST в середовищі Packet Tracer.

Необхідні ресурси:

- Комп'ютер з об'ємом пам'яті не менше 4 ГБ і 15 ГБ вільного місця на диску.
- Високошвидкісний доступ до Інтернет для завантаження Oracle VirtualBox та віртуальної машини DEVASC.

Частина 1. Підготовка комп'ютера до віртуалізації.

У цій частині ми завантажимо та встановимо програмне забезпечення для віртуалізації робочого столу та віртуальну машину DEVASC.

Примітка: наведені нижче інструкції призначені для Windows 10, що використовує VirtualBox версії 6.1.4. Незалежно від операційної системи або версії VirtualBox, обов'язково знайдіть і виберіть параметри, зазначені в наступних кроках.

Примітка: Якщо VirtualBox був встановлений раніше, можете перейти до кроку 2.

Крок 1: Завантажуємо та встановлюємо VirtualBox.

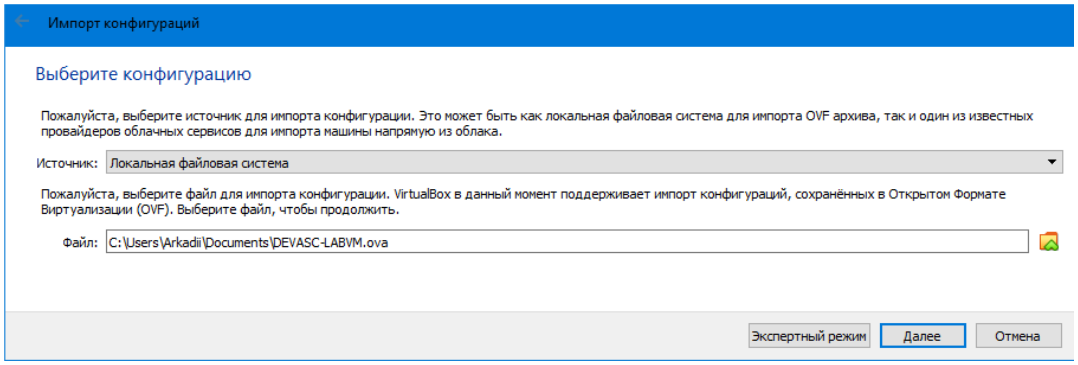
В лабораторній роботі будемо використовувати додаток Oracle VirtualBox – програму віртуалізації, яку потрібно завантажити та встановити для підтримки образу віртуальної машини DEVASC.

а. Переходимо за посиланням https://www.virtualbox.org/. Натискаємо на посилання для завантаження цієї сторінки.
б. Вибираємо і завантажуємо відповідний інсталяційний файл VirtualBox залежно від вашої операційної системи, наприклад, VirtualBox-6.1.4-136177-Win.
в. Запускаємо програму встановлення **VirtualBox** і приймаємо запропоновані за замовчуванням параметри.

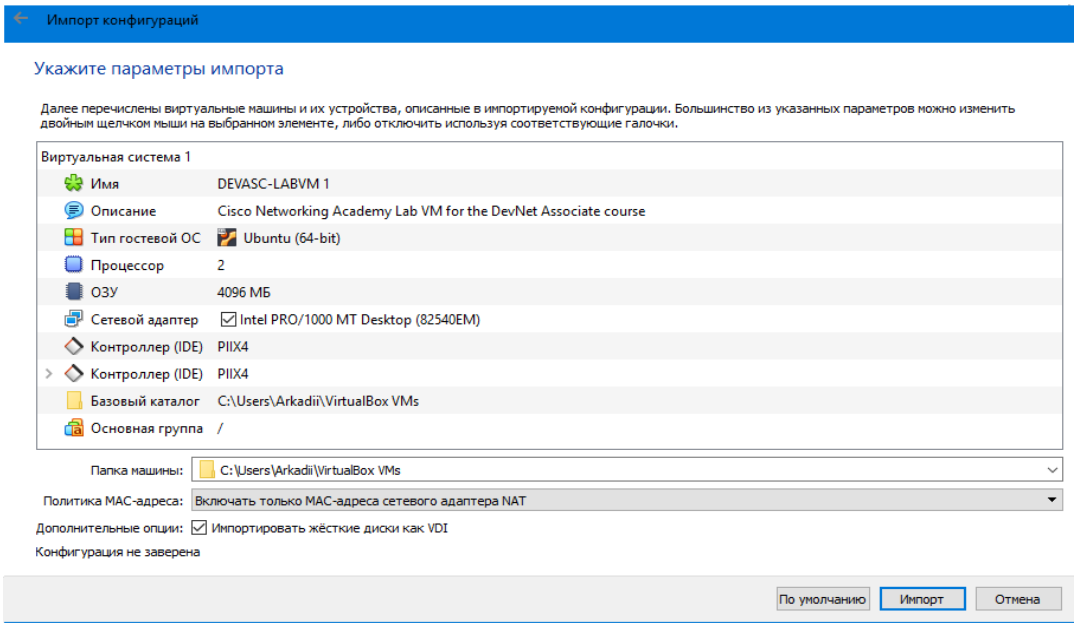
Частина 2: Імпортуємо і запускаємо віртуальну машину DEVASC.

а. Для отримання образу віртуальної машини **DEVASC_VM.OVA** переходимо за одним із трьох посилань і виконуємо завантаження:
https://drive.google.com/file/d/1pyt6lz3EoSRLCSjITs8PnelWHcRDJWw5/view?usp=sharing
https://drive.google.com/file/d/1pyt6lz3EoSRLCSjITs8PnelWHcRDJWw5/view?usp=drive_link
https://drive.labhub.eu.org/0:/OVA/

б. Запускаємо **Oracle VM VirtualBox** і створюємо віртуальну машину DEVASC-LABVM:
в. У VirtualBox вибираємо **Файл** > **Імпорт конфігурацій**.
г. Знаходимо місце розташування завантаженого образу віртуальної машини DEVASC і натискаємо **Далі**.

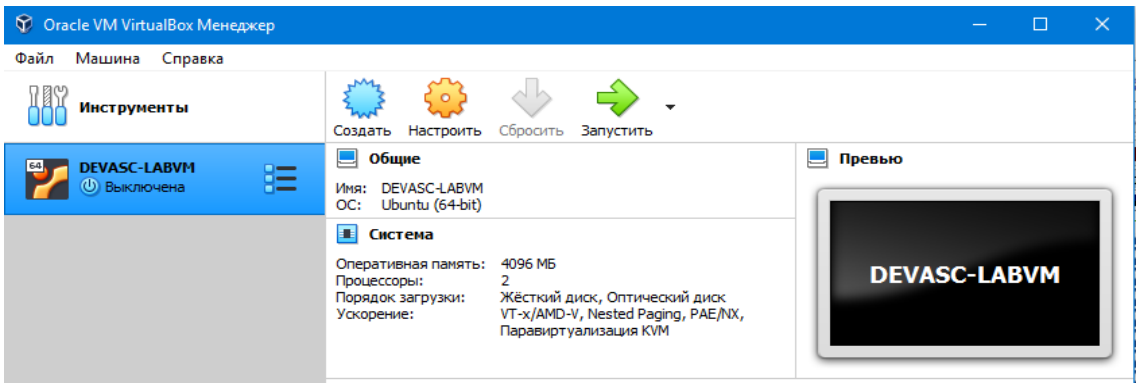



д. У наступному вікні натискаємо **Імпорт** , щоб продовжити.




Частина 3. Ознайомимося з графічним інтерфейсом віртуальної машини DEVASC.

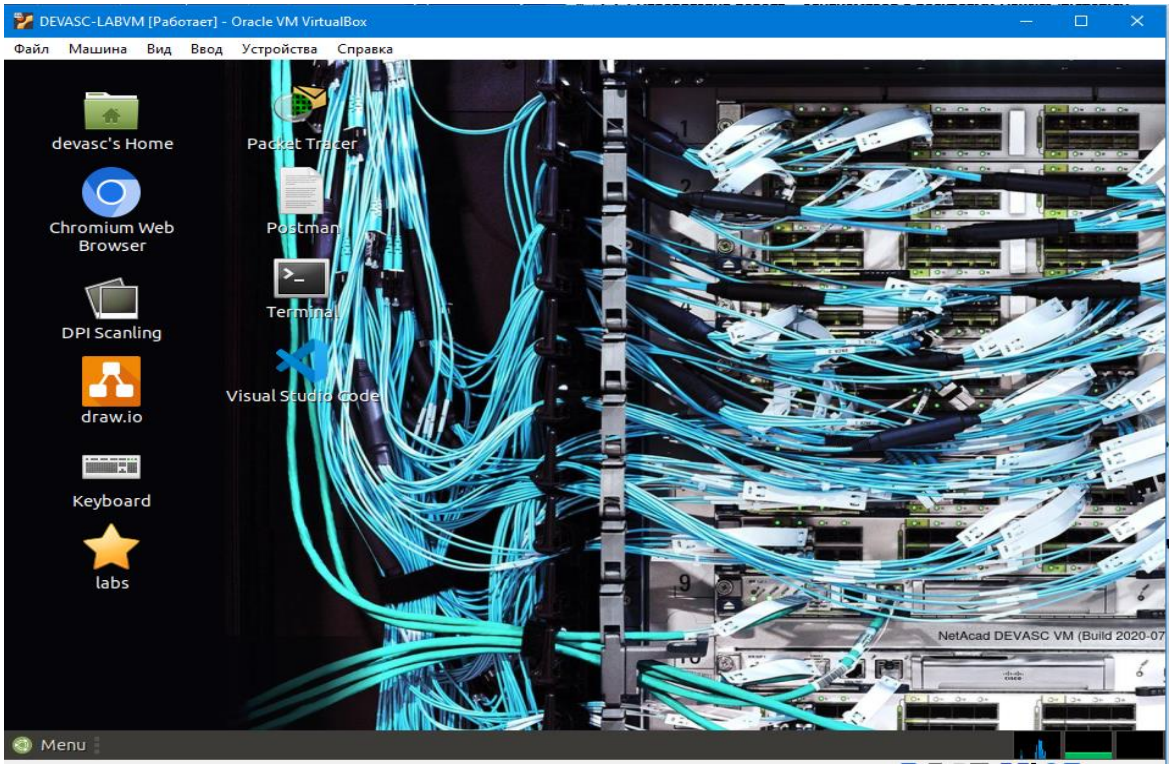
а. Запускаємо віртуальну машину DEVASC.




б. Віртуальна машина DEVASC містить інструмент Packet Tracer. Погоджуємося з ліцензійною угодою Cisco Packet Tracer щоб продовжити запуск віртуальної машини. Використовуємо клавіші із стрілками для прокрутки тексту. Після завершення ознайомлення натискаємо <**I Agree**> та <**OK**>.

в. Операційна система Ubuntu продовжуватиме завантажуватися. Закриваємо всі спливаючі повідомлення.

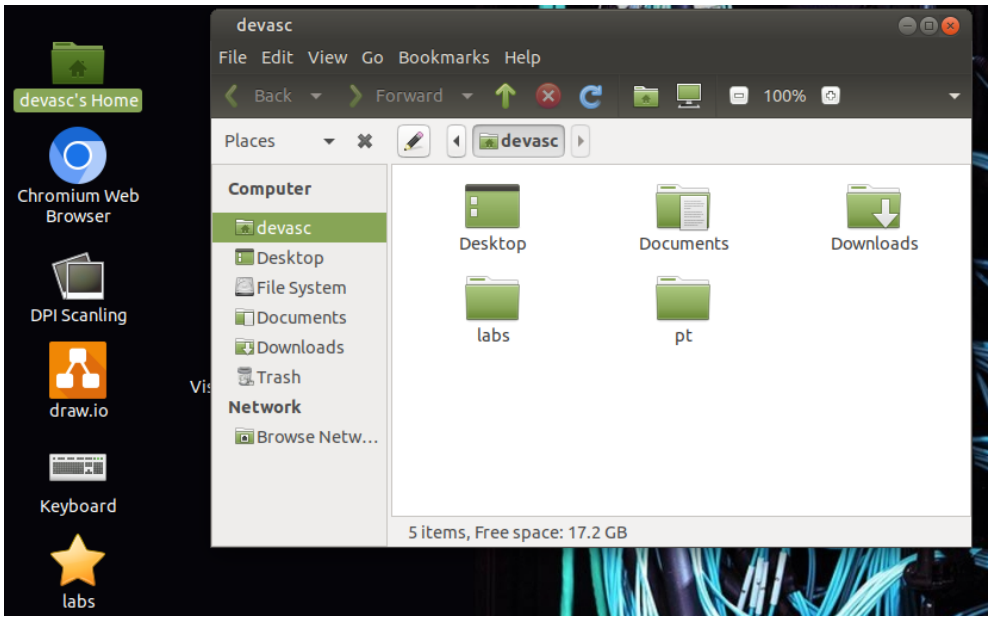
На запрошення **labvm login** можна не відповідати, або можна натиснути **Enter**.



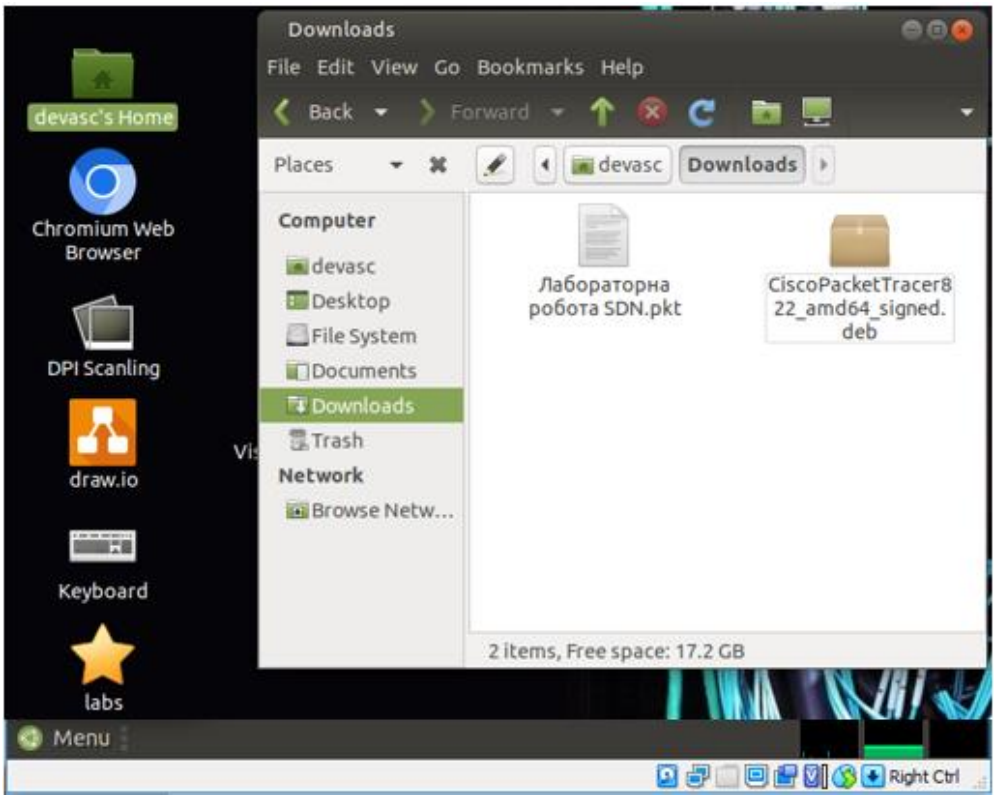

Після завантаження віртуальної машини потрібно виконати оновлення програми Cisco Packet Tracer. **Версія програми на віртуальній машині має відповідати версії програми на основній машині, де була створена модель мережі з мережевим контролером SDN.**

г. З сайту виробника за посиланням https://www.computernetworkingnotes.com/ccna-study-guide/download-packet-tracer-for-windows-and-linux.html#google_vignette завантажуємо потрібну нам версію програми CPT для операційної системи Ubuntu. На поточний момент це **CiscoPacketTracer822_amd64_signed.deb for Linux**.

д. У головному вікні віртуальної машини DEVASC клікаємо на піктограмі **devasc's Home.** У вікні, що з'явилося, відкриваємо папку **Downloads** і копіюємо в неї завантажений файл. Копіювання виконуємо методом перетягування із вікна основної машини у вікно віртуальної машини.



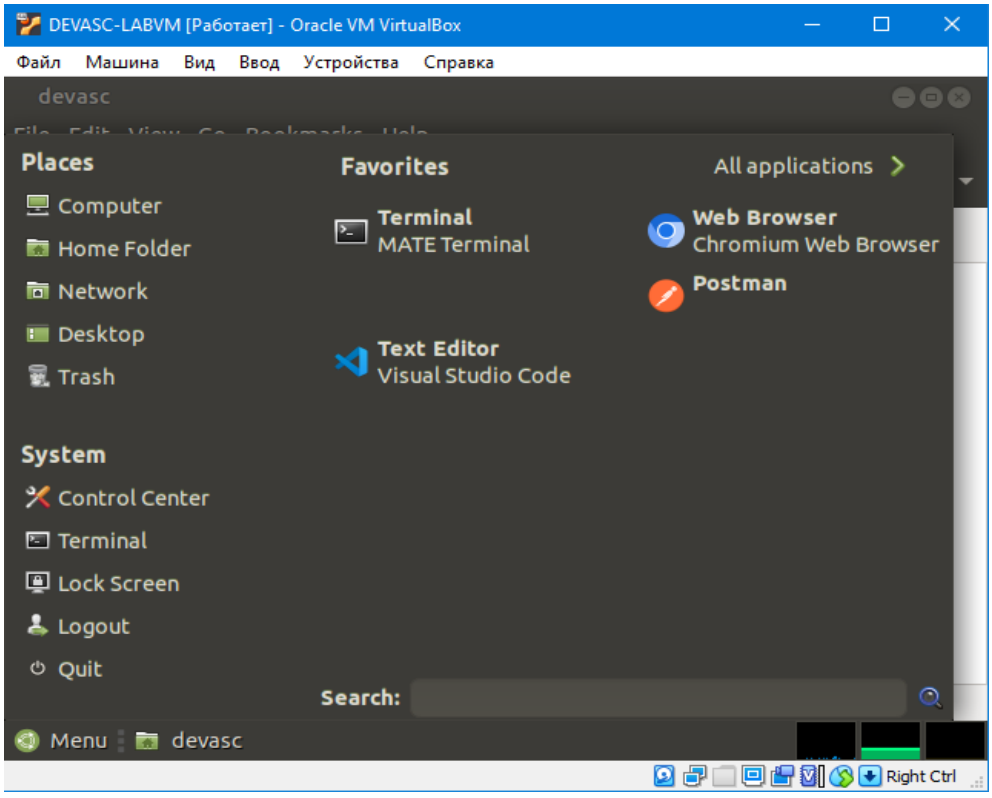
є. Методом перетягування копіюємо в папку **Downloads** також файл, що містить модель мережі з мережевим контролером (файл з розширенням **pkt**).




ж. Запускаємо установчий файл і встановлюємо нову версію програми Packet Tracer.

і. В лабораторній роботі будемо використовувати інструменти **Terminal**, **VS Code**, **Packet Tracer**, браузер **Chromium** і сервіс **Postman**. Відкриваємо ці програми та ознайомимося з ними.

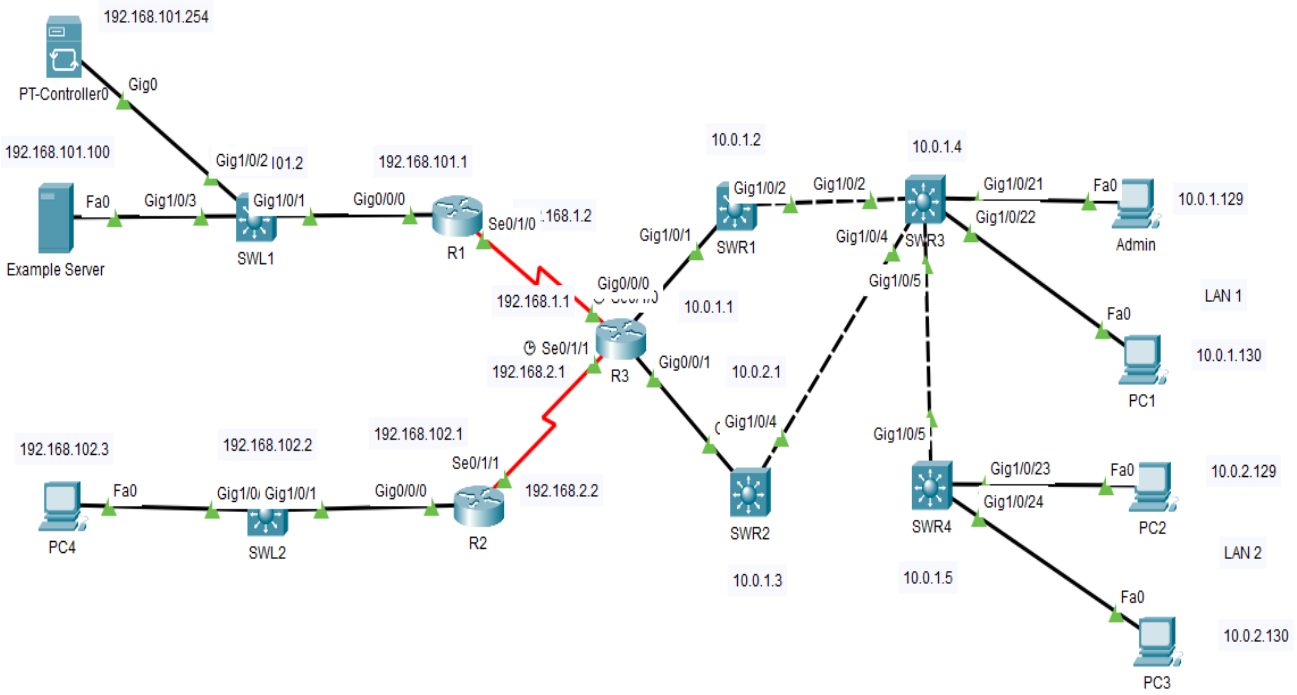
к. Натискаємо кнопку **Menu** та знайомимося з розділами **Places**, **System** та **ALL applications**.



Частина 4. Перевірка зовнішнього підключення до Packet Tracer.
У цій частині ми переконаємося, що до Packet Tracer можуть отримати доступ інші програми на віртуальній машині DEVASC. Ця дія має бути виконана повністю в середовищі віртуальної машини DEVASC.

Крок 1: Перевіряємо налаштування Packet Tracer для зовнішнього доступу.

а. Запускаємо Packet Tracer і відкриваємо файл з розширенням **pkt** з моделлю нашої мережі.

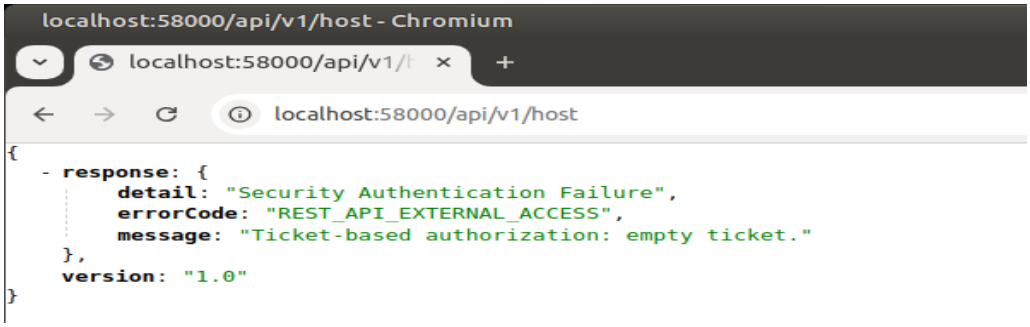



б. Вибираємо **Options** > **Preferences** > **Miscellaneous**. У розділі **External Network Access** переконуємося, що встановлено прапорець **Enable External Access for Network Controller REST API** (Дозволити зовнішній доступ для мережевого контролера REST API).
в. Закриваємо вікно налаштувань.
г. Вибираємо **PT-Controller0** > **Config**.
д. У розділі **REAL WORLD**, вибираємо **Controller**.
є. Перевіряємо, чи ввімкнений **Access Enabled**, і записуємо номер порту, який, швидше за все, має значення 58000. Це номер порту, який використовують скрипти Python, які будуть застосовані під час зовнішнього доступу до програми Packet Tracer із Chromium, VS Code та Postman пізніше у цьому завданні.


Крок 2: Переконаємося, що можна отримати доступ до Packet Tracer з іншої програми на віртуальній машині DEVASC.

Відкриваємо Chromium і переходимо за посиланням ***http://localhost:58000/api/v1/host***.

Маємо отримати наступну відповідь:




На цьому кроці перевіряється, що можна отримати зовнішній доступ до Packet Tracer та PT-Controller0. Звертаємо увагу, що для авторизації потрібний маркер автентифікації **ticket**, який поки-що відсутній (message: "Ticket-based authorization: empty ticket.") і буде отриманий у наступній частині. **Ticket** містить певну криптографічно захищену інформацію для перевірки особи та дозволів користувача.

Ключова інформація, що міститься в дійсному **ticket**, включає:

- Особистість користувача.
- Мітку часу створення **ticket**.
- Обмеження терміну придатності.
- Цифровий підпис або ключ шифрування, який сервіс використовує для перевірки автентичності **ticket**.

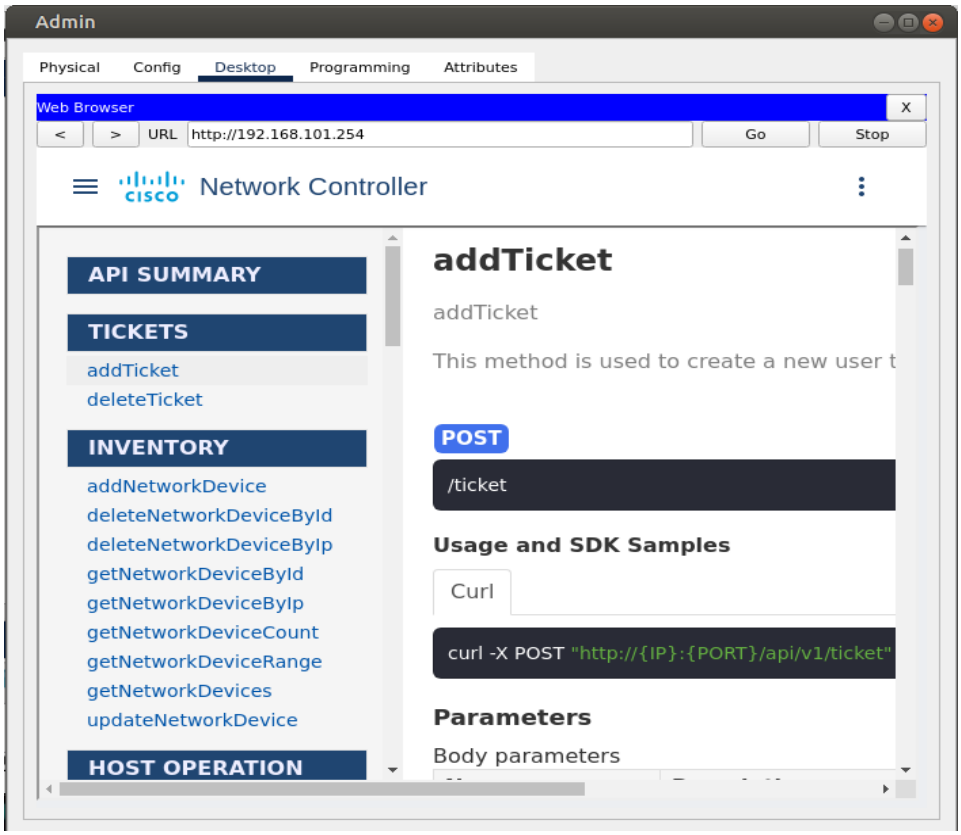
Частина 5. Запит маркера автентифікації (маркера безпеки) за допомогою Postman.

У цій частині ми вивчимо документацію REST API в Packet Tracer і за допомогою Postman виконаємо запит маркера автентифікації у PT-Controller0. Також це можна зробити у VS Code за допомогою сценарію Python.

Крок 1: Вивчаємо документацію REST API для мережевого контролера.

Щоб переглянути документацію REST API для PT-Controller0, на моделі мережі виконаємо такі дії:

а. Вибираємо **Admin > Desktop > Web Browser**.
б. Вводимо IP-адресу192.168.101.254.
в. Входимо до PT-Controller0 за допомогою імені користувача **cisco** та пароля **cisco123!**.
г. Клікаємо меню поруч із логотипом **Cisco** та вибираємо **API Docs**.
д. Доступ до цієї документації також можна отримати з меню help. Виберіть **Help > Contents**.
є. На панелі навігації ліворуч прокручуємо сторінку вниз та вибираємо **Network Controller API**. Тут представлена та ж сама документація, що і на PT-Controller0.
ж. У документації API натискаємо **add Ticket**. Будемо використовувати цю документацію на наступному кроці.



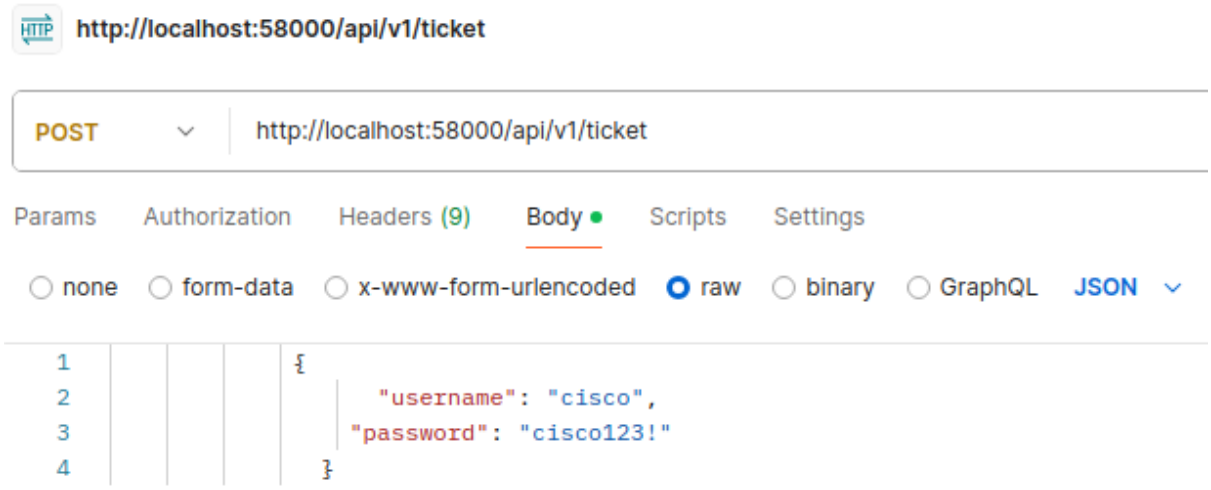

Крок 2: В POST створюємо і надсилаємо новий запит API для отримання коду автентифікації.

а. Ознайомившись з документацією по методу **add Ticket REST API**, відкриваємо Postman. В області запуску натискаємо на знак "плюс", щоб створити новий запит без назви.
б. Натискаємо стрілку вниз і змінюємо тип запиту з **GET** на **POST**.
в. Вводимо URL-адресу **http://localhost:58000/api/v1/ticket**.
г. Під полем **URL-адреса** натискаємо **Body**. Змінюємо тип на **raw**.
д. Натискаємо стрілку вниз поруч із **Text** і змінюємо його на **JSON**. Ця зміна також встановить **Content-type** заголовку HTTP на "application/json", який потрібен для цього виклику API.
є. Вставляємо наступний об'єкт JSON у поле **Body**. Переконуємося, що код правильно відформатований.

```
{
    "username" : "cisco",
    "password" : "cisco123!"
}
```

Крок 3: Надсилаємо запит POST.

а. Натискаємо **Send**, щоб надіслати запит POST до PT-Controller0.




Маємо отримати відповідь, подібну до наступної. Однак значення **serviceTicket** буде іншим.


```
{
    "response": {
        "idleTimeout": 900,
        "serviceTicket": "NC-2-470851aceb9e479185cb-nbi",
        "sessionTimeout": 3600
    },
    "version": "1.0"
}
```


б. **Скопіюйте** значення **service Ticket** у текстовий файл для подальшого використання.

Частина 6. Надсилання запитів REST за допомогою Postman

У цій частині будемо використовувати отриманий **Service Ticket** для надсилання двох запитів REST до PT-Controller0.

Крок 1: Створюємо новий запит **GET** для всіх мережевих пристроїв у мережі.

а. У Postman натискаємо на значок "плюс", щоб створити новий запит без назви.
б. Вводимо URL-адресу **http://localhost:58000/api/v1/network-device**.
в. Під полем URL-адреса вибираємо **Headers**.
г. Під останнім **Key** клікаємо поле Key і вводимо **X-Auth-Token**.
д. У полі **Value** вводимо значення раніше отриманого **Service Ticket**.

Крок 2: Надсилаємо запит **GET**.

Натискаємо **Send**, щоб надіслати запит **GET** на PT-Controller0.

Маємо отримати відповідь з відомостями, які зібрав контролер у дев'яти мережевих пристроїв в мережі. Зразок відповіді наведений нижче.


```
{
  "response": [
    {
      "collectionStatus": "Managed",
      "connectedInterfaceName": [
```

```
      "GigabitEthernet0/0/0",
      "GigabitEthernet0",
      "FastEthernet0"
    ],
    "connectedNetworkDeviceIpAddress": [
      "192.168.101.1",
      "192.168.101.254",
      "192.168.101.100"
    ],
    "connectedNetworkDeviceName": [
      "R1",
      "NetworkController",
      "Example Server"
    ],
    "errorDescription": "",
    "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
    "hostname": "SWL1",
    "id": "CAT1010UA1D-uuid",
    "interfaceCount": "29",
    "inventoryStatusDetail": "Managed",
    "lastUpdateTime": "1",
    "lastUpdated": "2025-02-22 19:54:23",
    "macAddress": "0007.EC5D.3AD6",
    "managementIpAddress": "192.168.101.2",
    "platformId": "3650",
    "productId": "3650-24PS",
    "reachabilityFailureReason": "",
    "reachabilityStatus": "Reachable",
    "serialNumber": "CAT1010UA1D-",
    "softwareVersion": "16.3.2",
    "type": "MultiLayerSwitch",
    "upTime": "1 hours, 41 minutes, 32 seconds"
  },
  {
    "collectionStatus": "Managed",
    "connectedInterfaceName": [
      "GigabitEthernet1/0/1",
      "Serial0/1/0"
    ],
    "connectedNetworkDeviceIpAddress": [
      "192.168.101.2",
      "192.168.1.1"
    ],
    "connectedNetworkDeviceName": [
      "SWL1",
      "R3"
    ],
    "errorDescription": "",
    "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
    "hostname": "R1",
    "id": "FDO13028O3K-uuid",
    "interfaceCount": "6",
    "inventoryStatusDetail": "Managed",
    "ipAddresses": [
      "192.168.101.1",
      "192.168.1.2"
    ],
    "lastUpdateTime": "1",
```

```
    "lastUpdated": "2025-02-22 19:54:23",
    "macAddress": "0004.9A71.5400",
    "managementIpAddress": "192.168.1.2",
    "platformId": "ISR4300",
    "productId": "ISR4331",
    "reachabilityFailureReason": "",
    "reachabilityStatus": "Reachable",
    "serialNumber": "FDO13028O3K-",
    "softwareVersion": "15.4",
    "type": "Router",
    "upTime": "1 hours, 41 minutes, 32 seconds"
  },
  {
    "collectionStatus": "Managed",
    "connectedInterfaceName": [
      "GigabitEthernet1/0/1",
      "GigabitEthernet1/0/1",
      "Serial0/1/0",
      "Serial0/1/1"
    ],
    "connectedNetworkDeviceIpAddress": [
      "10.0.1.2",
      "",
      "192.168.1.2",
      "192.168.2.2"
    ],
    "connectedNetworkDeviceName": [
      "SWR1",
      "SWR2",
      "R1",
      "R2"
    ],
    "errorDescription": "",
    "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
    "hostname": "R3",
    "id": "FDO13026ME3-uuid",
    "interfaceCount": "6",
    "inventoryStatusDetail": "Managed",
    "ipAddresses": [
      "10.0.1.1",
      "10.0.2.1",
      "192.168.1.1",
      "192.168.2.1"
    ],
    "lastUpdateTime": "1",
    "lastUpdated": "2025-02-22 19:54:23",
    "macAddress": "00E0.8FE6.5340",
    "managementIpAddress": "192.168.2.1",
    "platformId": "ISR4300",
    "productId": "ISR4331",
    "reachabilityFailureReason": "",
    "reachabilityStatus": "Reachable",
    "serialNumber": "FDO13026ME3-",
    "softwareVersion": "15.4",
    "type": "Router",
    "upTime": "1 hours, 41 minutes, 32 seconds"
  },
  {
```

```
    "collectionStatus": "Managed",
    "connectedInterfaceName": [
      "GigabitEthernet1/0/1",
      "Serial0/1/1"
    ],
    "connectedNetworkDeviceIpAddress": [
      "192.168.102.2",
      "192.168.2.1"
    ],
    "connectedNetworkDeviceName": [
      "SWL2",
      "R3"
    ],
    "errorDescription": "",
    "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
    "hostname": "R2",
    "id": "FDO13021WMI-uuid",
    "interfaceCount": "6",
    "inventoryStatusDetail": "Managed",
    "ipAddresses": [
      "192.168.102.1",
      "192.168.2.2"
    ],
    "lastUpdateTime": "0",
    "lastUpdated": "2025-02-22 19:54:24",
    "macAddress": "000B.BEDB.4D28",
    "managementIpAddress": "192.168.2.2",
    "platformId": "ISR4300",
    "productId": "ISR4331",
    "reachabilityFailureReason": "",
    "reachabilityStatus": "Reachable",
    "serialNumber": "FDO13021WMI-",
    "softwareVersion": "15.4",
    "type": "Router",
    "upTime": "1 hours, 41 minutes, 33 seconds"
  },
  {
    "collectionStatus": "Managed",
    "connectedInterfaceName": [
      "GigabitEthernet0/0/1",
      "GigabitEthernet1/0/4"
    ],
    "connectedNetworkDeviceIpAddress": [
      "10.0.2.1",
      "10.0.1.4"
    ],
    "connectedNetworkDeviceName": [
      "R3",
      "SWR3"
    ],
    "errorDescription": "",
    "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
    "hostname": "SWR2",
    "id": "CAT1010E2QZ-uuid",
    "interfaceCount": "29",
    "inventoryStatusDetail": "Managed",
    "lastUpdateTime": "0",
    "lastUpdated": "2025-02-22 19:54:24",
```

```
    "macAddress": "00E0.8F43.E278",
    "managementIpAddress": "10.0.1.3",
    "platformId": "3650",
    "productId": "3650-24PS",
    "reachabilityFailureReason": "",
    "reachabilityStatus": "Reachable",
    "serialNumber": "CAT1010E2QZ-",
    "softwareVersion": "16.3.2",
    "type": "MultiLayerSwitch",
    "upTime": "1 hours, 41 minutes, 33 seconds"
  },
  {
    "collectionStatus": "Managed",
    "connectedInterfaceName": [
      "GigabitEthernet0/0/0",
      "FastEthernet0"
    ],
    "connectedNetworkDeviceIpAddress": [
      "192.168.102.1",
      "192.168.102.3"
    ],
    "connectedNetworkDeviceName": [
      "R2",
      "PC0"
    ],
    "errorDescription": "",
    "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
    "hostname": "SWL2",
    "id": "CAT1010D778-uuid",
    "interfaceCount": "29",
    "inventoryStatusDetail": "Managed",
    "lastUpdateTime": "14",
    "lastUpdated": "2025-02-22 19:54:10",
    "macAddress": "00D0.FFE5.8436",
    "managementIpAddress": "192.168.102.2",
    "platformId": "3650",
    "productId": "3650-24PS",
    "reachabilityFailureReason": "",
    "reachabilityStatus": "Reachable",
    "serialNumber": "CAT1010D778-",
    "softwareVersion": "16.3.2",
    "type": "MultiLayerSwitch",
    "upTime": "1 hours, 41 minutes, 19 seconds"
  },
  {
    "collectionStatus": "Managed",
    "connectedInterfaceName": [
      "GigabitEthernet0/0/0",
      "GigabitEthernet1/0/2"
    ],
    "connectedNetworkDeviceIpAddress": [
      "10.0.1.1",
      "10.0.1.4"
    ],
    "connectedNetworkDeviceName": [
      "R3",
      "SWR3"
    ],
```

```
    "errorDescription": "",
    "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
    "hostname": "SWR1",
    "id": "CAT1010X4UB-uuid",
    "interfaceCount": "29",
    "inventoryStatusDetail": "Managed",
    "lastUpdateTime": "14",
    "lastUpdated": "2025-02-22 19:54:10",
    "macAddress": "0060.2FE4.5D8D",
    "managementIpAddress": "10.0.1.2",
    "platformId": "3650",
    "productId": "3650-24PS",
    "reachabilityFailureReason": "",
    "reachabilityStatus": "Reachable",
    "serialNumber": "CAT1010X4UB-",
    "softwareVersion": "16.3.2",
    "type": "MultiLayerSwitch",
    "upTime": "1 hours, 41 minutes, 19 seconds"
  },
  {
    "collectionStatus": "Managed",
    "connectedInterfaceName": [
      "GigabitEthernet1/0/2",
      "GigabitEthernet1/0/4",
      "GigabitEthernet1/0/5",
      "FastEthernet0",
      "FastEthernet0"
    ],
    "connectedNetworkDeviceIpAddress": [
      "10.0.1.2",
      "10.0.1.3",
      "10.0.1.5",
      "10.0.1.129",
      "10.0.1.130"
    ],
    "connectedNetworkDeviceName": [
      "SWR1",
      "SWR2",
      "SWR4",
      "Admin",
      "PC1"
    ],
    "errorDescription": "",
    "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
    "hostname": "SWR3",
    "id": "CAT1010UIL1-uuid",
    "interfaceCount": "29",
    "inventoryStatusDetail": "Managed",
    "lastUpdateTime": "14",
    "lastUpdated": "2025-02-22 19:54:10",
    "macAddress": "0060.3E97.07A2",
    "managementIpAddress": "10.0.1.4",
    "platformId": "3650",
    "productId": "3650-24PS",
    "reachabilityFailureReason": "",
    "reachabilityStatus": "Reachable",
    "serialNumber": "CAT1010UIL1-",
    "softwareVersion": "16.3.2",
```

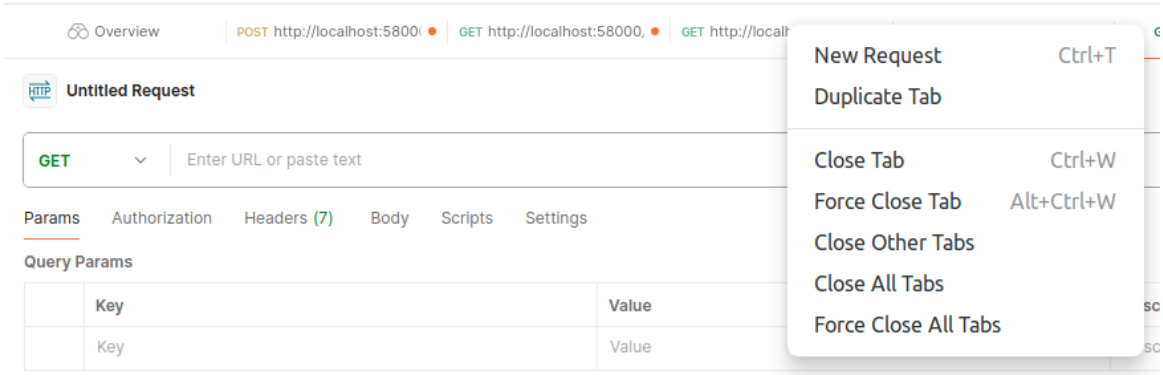
```
      "type": "MultiLayerSwitch",
      "upTime": "1 hours, 41 minutes, 19 seconds"
    },
    {
      "collectionStatus": "Managed",
      "connectedInterfaceName": [
        "GigabitEthernet1/0/5",
        "FastEthernet0",
        "FastEthernet0"
      ],
      "connectedNetworkDeviceIpAddress": [
        "10.0.1.4",
        "10.0.2.129",
        "10.0.2.130"
      ],
      "connectedNetworkDeviceName": [
        "SWR3",
        "PC2",
        "PC3"
      ],
      "errorDescription": "",
      "globalCredentialId": "25d59d05-7300-4d85-8562-7c01482ddd59",
      "hostname": "SWR4",
      "id": "CAT1010AVGH-uuid",
      "interfaceCount": "29",
      "inventoryStatusDetail": "Managed",
      "lastUpdateTime": "14",
      "lastUpdated": "2025-02-22 19:54:10",
      "macAddress": "0040.0B26.22B7",
      "managementIpAddress": "10.0.1.5",
      "platformId": "3650",
      "productId": "3650-24PS",
      "reachabilityFailureReason": "",
      "reachabilityStatus": "Reachable",
      "serialNumber": "CAT1010AVGH-",
      "softwareVersion": "16.3.2",
      "type": "MultiLayerSwitch",
      "upTime": "1 hours, 41 minutes, 19 seconds"
    }
  ],
  "version": "1.0"
}
```


Крок 3: Продублюємо запит і змінимо його для виконання запиту до контролера про всі хости у мережі.

а. У Postman клікаємо правою кнопкою миші вкладку запиту на отримання інформації про хости та вибираємо **Dublicate Tab**.

б. Вся інформація у запиті та ж сама, за винятком URL-адреси. Просто змінюємо «**network-device**» на «**host**»:

http://localhost:58000/api/v1/host.




Крок 4: Надсилаємо запит GET.

Натискаємо **Send**, щоб надіслати запит GET на PT-Controller0.

Маємо отримати відповідь із зазначенням відомостей, які зібрав контролер для шести хост-пристроїв в мережі.


```
{
  "response": [
    {
      "connectedAPMacAddress": "",
      "connectedAPName": "",
      "connectedInterfaceName": "GigabitEthernet1/0/3",
      "connectedNetworkDeviceIpAddress": "192.168.101.2",
      "connectedNetworkDeviceName": "SWL1",
      "hostIp": "192.168.101.100",
      "hostMac": "000C.CF34.AC5E",
      "hostName": "Example Server",
      "hostType": "Server",
      "id": "PTT0810E6O8-uuid",
      "lastUpdated": "2025-02-22 18:52:40",
      "pingStatus": "SUCCESS"
    },
    {
      "connectedAPMacAddress": "",
      "connectedAPName": "",
      "connectedInterfaceName": "GigabitEthernet1/0/24",
      "connectedNetworkDeviceIpAddress": "192.168.102.2",
      "connectedNetworkDeviceName": "SWL2",
      "hostIp": "192.168.102.3",
      "hostMac": "0001.634D.27E8",
      "hostName": "PC0",
      "hostType": "Pc",
      "id": "PTT08105JJZ-uuid",
      "lastUpdated": "2025-02-22 18:52:40",
      "pingStatus": "SUCCESS"
    },
    {
      "connectedAPMacAddress": "",
      "connectedAPName": "",
      "connectedInterfaceName": "GigabitEthernet1/0/21",
      "connectedNetworkDeviceIpAddress": "10.0.1.4",
      "connectedNetworkDeviceName": "SWR3",
      "hostIp": "10.0.1.130",
      "hostMac": "0090.2B4E.7B94",
      "hostName": "Admin",
```

```
      "hostType": "Pc",
      "id": "PTT08107KOH-uuid",
      "lastUpdated": "2025-02-22 18:52:40",
      "pingStatus": "SUCCESS"
    },
    {
      "connectedAPMacAddress": "",
      "connectedAPName": "",
      "connectedInterfaceName": "GigabitEthernet1/0/23",
      "connectedNetworkDeviceIpAddress": "10.0.1.5",
      "connectedNetworkDeviceName": "SWR4",
      "hostIp": "10.0.2.129",
      "hostMac": "0060.5C49.324E",
      "hostName": "PC2",
      "hostType": "Pc",
      "id": "PTT08107C8A-uuid",
      "lastUpdated": "2025-02-22 18:52:40",
      "pingStatus": "SUCCESS"
    },
    {
      "connectedAPMacAddress": "",
      "connectedAPName": "",
      "connectedInterfaceName": "GigabitEthernet1/0/24",
      "connectedNetworkDeviceIpAddress": "10.0.1.5",
      "connectedNetworkDeviceName": "SWR4",
      "hostIp": "10.0.2.130",
      "hostMac": "0090.217E.2C3E",
      "hostName": "PC3",
      "hostType": "Pc",
      "id": "PTT0810Q0JY-uuid",
      "lastUpdated": "2025-02-22 21:35:14",
      "pingStatus": "SUCCESS"
    },
    {
      "connectedAPMacAddress": "",
      "connectedAPName": "",
      "connectedInterfaceName": "GigabitEthernet1/0/22",
      "connectedNetworkDeviceIpAddress": "10.0.1.4",
      "connectedNetworkDeviceName": "SWR3",
      "hostIp": "10.0.1.129",
      "hostMac": "0090.213C.B161",
      "hostName": "PC1",
      "hostType": "Pc",
      "id": "PTT0810HTGU-uuid",
      "lastUpdated": "2025-02-22 18:52:40",
      "pingStatus": "SUCCESS"
    }
  ],
  "version": "1.0"
```


Крок 5: Закриваємо Postman, щоб звільнити пам'ять у віртуальній машині DEVASC.

Частина 7: Надсилаємо запити REST за допомогою програми VS Code

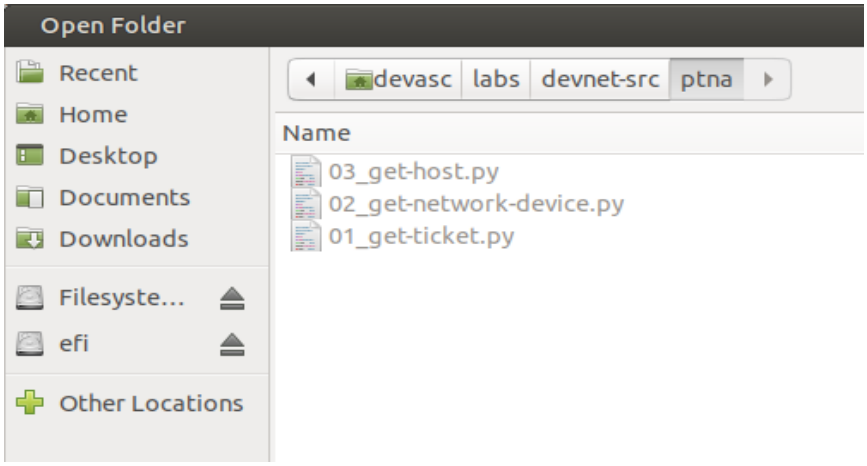
У цій частині ми будемо використовувати скрипт Python у VS Code для надсилання тих самих запитів API, які ми надсилали в програмі Postman. Однак ми також будемо

використовувати цикли Python **for** для аналізу JSON і відображення лише певних пар ключ-значення.

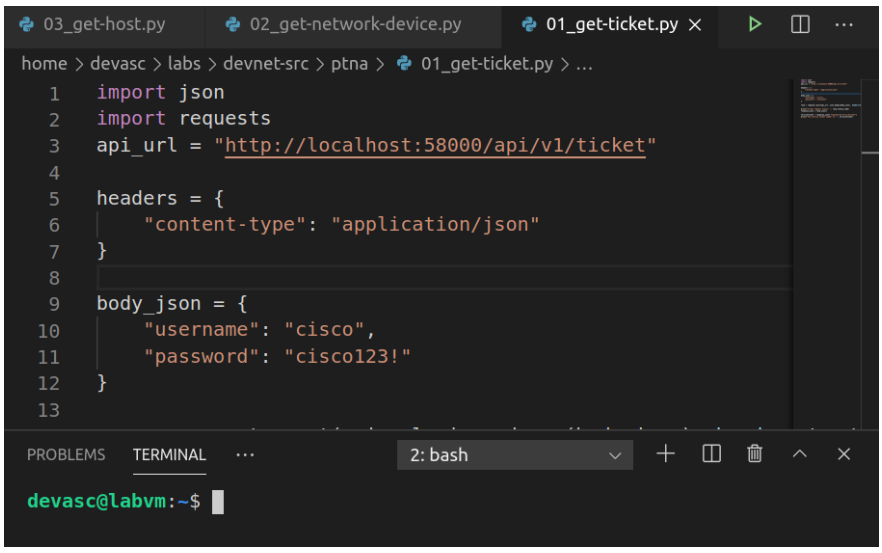
Крок 1: Використаємо скрипт для створення запиту на обслуговування.

а. Відкриваємо **VS Code**.
б. Вибираємо **File** > **Open Folder**...**Home/labs/** і переходимо до каталогу **devnet-src/ptna**.
в. Натискаємо **OK**.

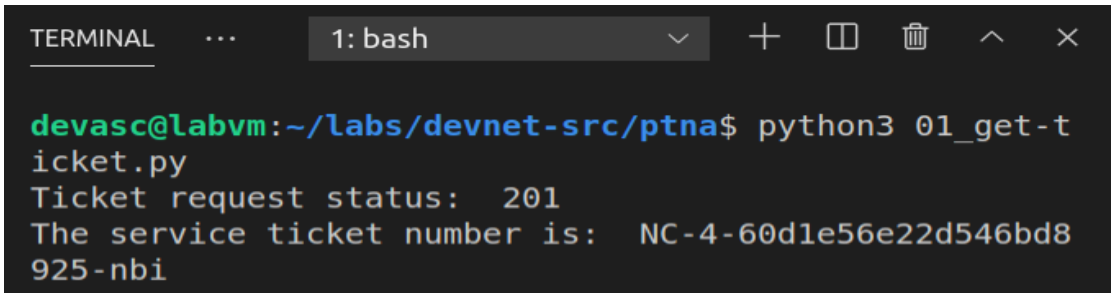
Зверніть увагу, що на панелі **EXPLORE** відображаються три файли-сценарії: **01_get-ticket.py**, **02_get-network-device.py і 03_get-host.py**. Перегляньте код для кожного з них. Зверніть увагу, що сценарії для мережевих пристроїв та хостів вимагають замінити значення «**serviceTicket**» на значення, визначене під час виконання запиту на отримання значення **ticket**. Виконайте новий запит на обслуговування за сценарієм **01_get-ticket.py**.




г. Відкриваємо вікно терміналу у VS Code: **Terminal > New Terminal**.




д. Запускаємо **01_get-ticket (**кнопка поруч). Маємо отримати результат, подібний до наступного. Результат містить значення ключа **Service Ticket**.




Замінюємо значення **Service Ticket** в скриптах **02_get-network-device.py** і **03_get-host.py** на отримане значення.

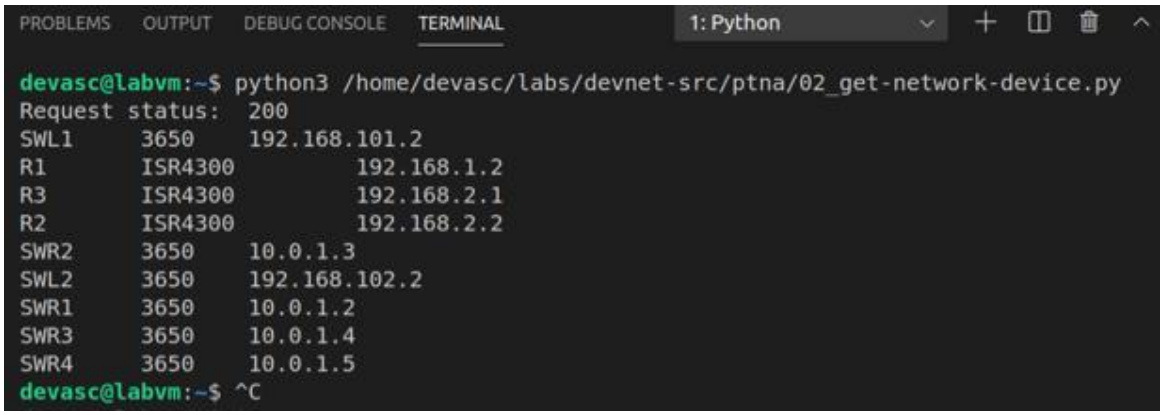
Крок 2: Використаємо скрипт-сценарій для запиту списку мережевих пристроїв.

Раніше в Postman виклик **network devices API** повернув список усіх дев'яти мережевих пристроїв та всю інформацію, доступну для кожного з них. Однак скрипт **02_get-network-device.py** виводить тільки ті значення ключів, які перелічені в скрипті: hostname, PlatformID і managementIpAddress.

У вікні терміналу запускаємо скрипт **02_get-network-device.py**.

devasc@labvm:~/labs/devnet-src/ptna$ **python3 02_get-network-device.py**

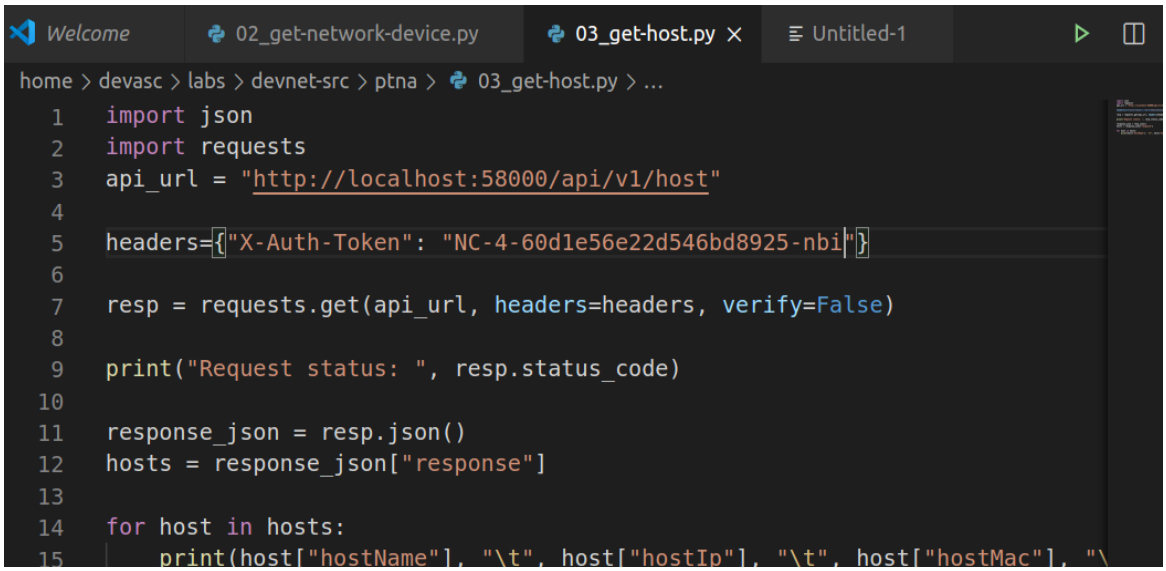
Маємо отримати такий результат:



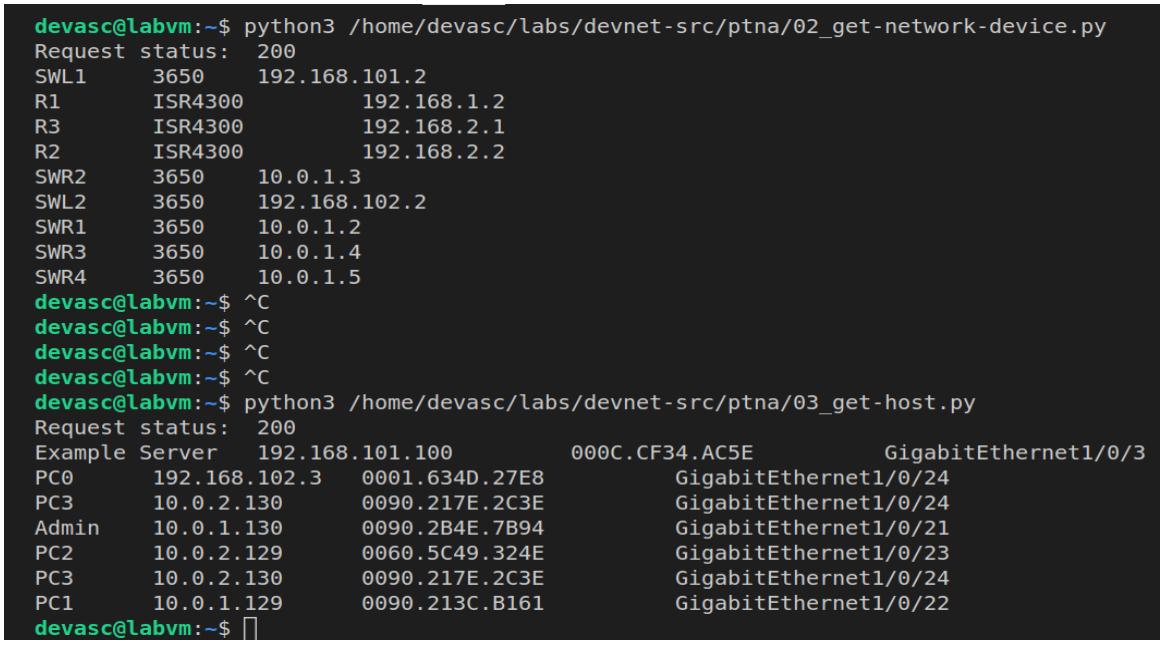

Крок 3: Використаємо скрипт-сценарій для запиту списку хост-пристроїв.

Аналогічно, в скрипті міститься список конкретної інформації, яку потрібно отримати для кожного із шести хост-пристроїв, під’єднаних до мережі.

У вікні терміналу запускаємо сценарій **03_get-host.py**.




Отримуємо результат

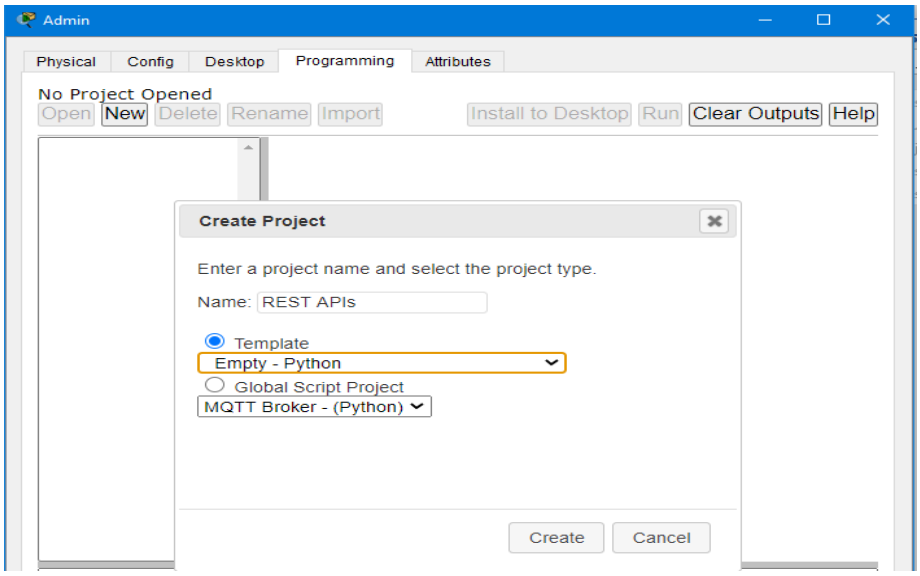



Частина 8: Надсилання запитів REST, створених в Packet Tracer.

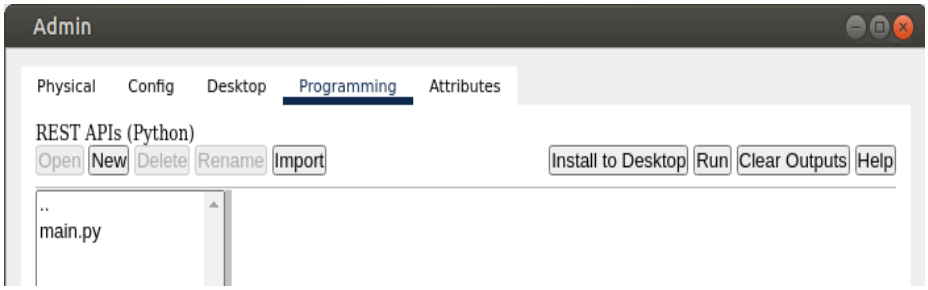
У цій частині ми будемо використовувати ті самі сценарії з однією невеликою відмінністю: будемо надсилати ті ж самі запити API, які ми надсилали з VS Code, створені безпосередньо в Packet Tracer.

Крок 1: Переходимо до нашої моделі в Packet Tracer.

а. У Packet Tracer вибираємо комп'ютер адміністратора **Admin PC**.
б. Переходимо на вкладку **Programming**.
в. На даний момент проєкту немає. Натискаємо **New**.
г. В поле **Name** вводимо **REST APIs** та вибираємо значення Template (шаблон) – **Empty – Python.**
д. Натискаємо **Create**.




Проєкт **REST APIs (Python)** тепер створений з порожнім скриптом **main.py**.



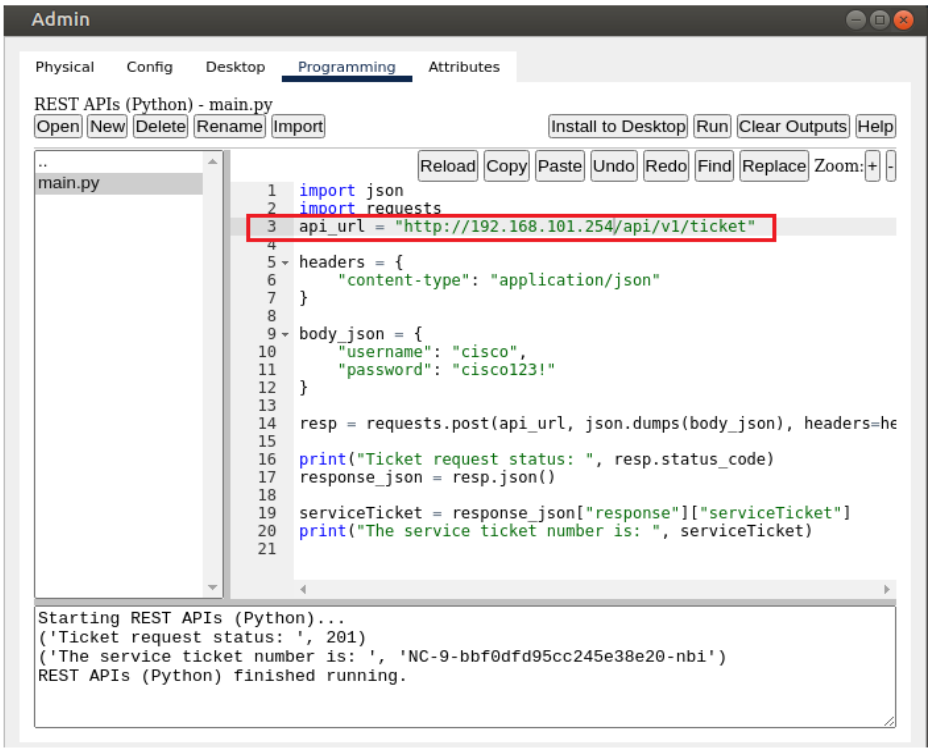

Крок 2: Змінюємо сценарії для запуску всередині Packet Tracer.

Для доступу з однієї програми до іншої на тій самій машині-хості потрібно вказати номер порту в URL-адресі. Однак Packet Tracer імітує реальну мережу. У реальному світі ми зазвичай не вказуємо номер порту під час надсилання запитів до API. Крім того, в URL-адресі ми можемо використовувати доменне ім'я або IP-адресу.

а. У вікні VS Code натискаємо **File > Open File,** переходимо по шляху **Home > labs > devnet-src > ptna** і знаходимо потрібні нам скрипти.
б. Відкриваємо, виділяємо і копіюємо (**Copy**) код скрипта **01_get-tiket.py**.
в. На комп'ютері **Admin** відкриваємо вкладку **Programming** і двічі клікаємо на папці **main.py**, щоб відкрити її.
г. Натискаємо **Paste** і вставляємо код скрипта у **main.py**.

```
import json
import requests
api_url = "http://localhost:58000/api/v1/ticket"
headers = {
   "content-type": "application/json"
}
body_json = {
   "username": "cisco",
   "password": "cisco123!"
}
resp = requests.post(api_url, json.dumps(body_json), headers=headers, verify=False)
print("Ticket request status: ", resp.status_code)
response_json = resp.json()
serviceTicket = response_json["response"]["serviceTicket"]
print("The service ticket number is: ", serviceTicket)
```

д. Змінюємо значення **api_url**. Замінюємо **localhost:58000/api/v1/tiket** на **192.168.101.254/api/v1/tiket**. Зміни будуть автоматично збережені.




є. Натискаємо **Run**. Отримане у полі **Outputs** значення **service tiket** не буде відповідати тому, що показано в прикладі. Однак ви маєте побачити результат, подібний тому, що показаний нижче. Збережіть **service tiket** для використання в наступних запитах.

```
Starting REST APIs (Python)...
('Ticket request status: ', 201)
('The service ticket number is: ', ' NC-9-bbf0dfd95cc245e38e20-nbi ')
REST APIs (Python) finished running.
```

ж. Повертаємося до VS Code. Натискаємо **File > Open File,** переходимо по шляху **Home > labs > devnet-src > ptna** і знаходимо потрібні нам скрипти.

і. Відкриваємо, виділяємо і копіюємо (**Copy**) код скрипта **03_get-host.py**.

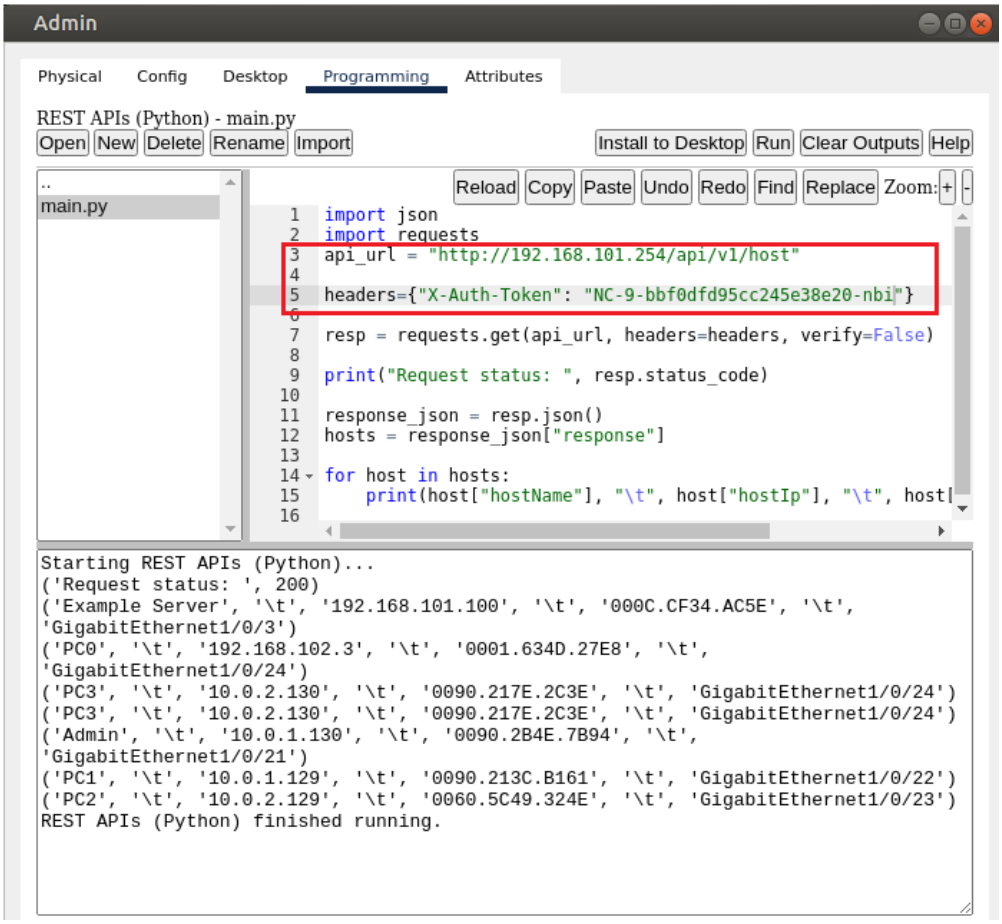
й. На комп'ютері **Admin** відкриваємо вкладку **Programming** і двічі клікаємо на папці **main.py**, щоб відкрити її.

к. Натискаємо **Paste** і вставляємо код скрипта у **main.py**.

л. Замінюємо **localhost:58000/api/v1/host** на **192.168.101.254/api/v1/host**.

м. Вставляємо в запит коректне значення **service ticket,** отримане при виконанні пункта «є». Зміни будуть автоматично збережені.

н. Натискаємо **Run**. У полі **Outputs** отримуємо результат виконання скрипта.




о. Знову повертаємося до VS Code. Натискаємо **File > Open File,** переходимо по шляху **Home > labs > devnet-src > ptna** і знаходимо потрібні нам скрипти.

п. Відкриваємо, виділяємо і копіюємо (**Copy**) код скрипта **02_get-network-device**.

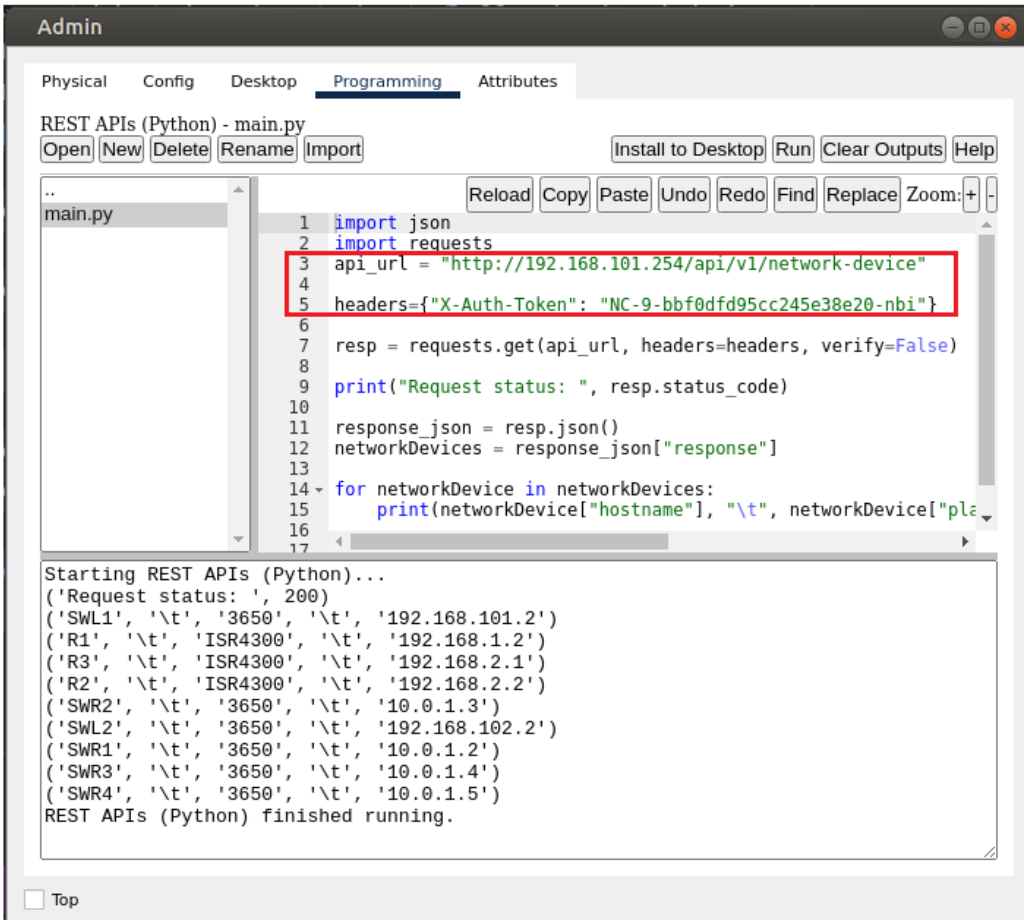
р. На комп'ютері **Admin** відкриваємо вкладку **Programming** і двічі клікаємо на папці **main.py**, щоб відкрити її.

с. Натискаємо **Paste** і вставляємо код скрипта у **main.py**.

т. Замінюємо **localhost:58000/api/v1/network-device** на **192.168.101.254/api/v1/network-device.**

у. Вставляємо в запит коректне значення **service ticket,** отримане при виконанні пункта «є». Зміни будуть автоматично збережені.

ф. Натискаємо **Run**. У вікні **Outputs** отримуємо результат виконання скрипта.




Виконання лабораторної роботи завершене.

**Завершення роботи  віртуальної машини.**

Коли ви закінчите роботу з віртуальною машиною, ви можете зберегти поточний стан машини для подальшого використання або вимкнути віртуальну машину. Закрийте віртуальну машину за допомогою графічного інтерфейсу:

а. У меню **Машина** Virtual Box виберіть Закрити.
б. Натисніть радіо-кнопку "Зберегти стан машини" та натисніть OK. Наступного разу, коли ви запустите віртуальну машину, ви зможете продовжити роботу в операційній системі із збереженого стану.
в. Інші два варіанти:
- Відправити сигнал вимкнення: Це імітує натискання кнопки живлення на фізичному комп'ютері.
- Вимкнути машину: Це імітує витягування шнура з розетки на фізичному комп'ютері.

Завершити роботу віртуальної машини можна за допомогою командного рядка CLI:

Щоб вимкнути VM за допомогою CLI, можна використати меню **Файл>Закрити** всередині віртуальної машини DEVASC-LABVM  або ввести команду **sudo shutdown -h now** у вікні терміналу.

Перезавантаження ВМ.

Якщо потрібно перезавантажити ВМ, можна скористатися параметрами меню всередині віртуальної машини DEVASC-LABVM або ввести команду **sudo reboot** у вікні терміналу.

Контрольні запитання

1. Які основні причин виникнення концепції програмованих мереж?
2. Які основні функціональні блоки мережі SDN?
3. Яка послідовність дій при під’єднанні до мережевого контролера в Packet Tracer?

# Лабораторна робота 3

## Моделювання програмно-керованої мережі у симуляторі Mininet. Ручне керування потоками.

**Мета:** Ознайомитися з мережевим емулятором Mininet, який дозволяє створювати на комп'ютері моделі віртуальних мереж, використовуючи реальні образи операційних систем, комутаторів та додатків, провести дослідження поведінки створених тестових мереж при виконанні сценаріїв керування мережевими комутаторами Open vSwitch.

### План роботи:

1. Ознайомлення з теоретичними відомостями до лабораторної роботи.
2. Завантаження та встановлення системи віртуалізації VMware Workstation Player.
3. Завантаження середовища Mininet VM.
4. Запуск Mininet VM за допомогою VMware Player.
5. Встановлення і запуск програми Xming.
6. Підключення до віртуальної машини Mininet за допомогою програми Putty.
7. Виконання різних сценаріїв в середовищі Mininet.
8. Робота в графічному редакторі MiniEdit.

## 1. Теоретичні відомості

Використання Mininet для створення віртуальної комутованої мережі

Mininet — це *мережевий емулятор*, який створює мережу віртуальних хостів, комутаторів, контролерів та з'єднань. Хости Mininet працюють під керуванням стандартного мережевого програмного забезпечення Linux, а його комутатори підтримують OpenFlow для високогнучкої маршрутизації при дослідженні моделей програмно-визначених мереж.

Mininet підтримує дослідження, розробку, навчання, створення прототипів, тестування, налагодження та будь-які інші завдання, які можуть бути корисними завдяки наявності повноцінної експериментальної мережі на ноутбуці або іншому ПК.

Команда **sudo mn** запускає середовище Mininet. Це віртуалізує мережу пристроїв і хостів, які працюють як реальні пристрої. Можна запускати програми на хост-пристроях або переглядати таблиці OpenFlow на комутаторах. Програми на хостах можуть надсилати пакети через інтерфейси комутатора.
Пакети обробляються так само, як і реальним комутатором Ethernet або маршрутизатором.
Топології різних розмірів можна створити простими командами, такими як *sudo mn* (один комутатор і два хоста) або *sudo mn --topo linear,4* (чотири комутатори та чотири хости). Це дозволяє швидко редагувати, запускати та тестувати різні конфігурації SDN.
Користувальницькі топології можуть також бути створені для моделювання дуже великих центрів обробки даних або інших великих мереж.
На хостах Mininet можуть також працювати реальні програми, такі як вебсервер, DNS-сервер, інструменти моніторингу tcpdump або Wireshark.
Все, що працює на Linux, можна запустити на Mininet.
Комутатори Mininet можна програмувати за допомогою протоколу OpenFlow. Mininet підтримує віддалені (встановлені на іншому сервері) та зовнішні контролери (зовнішні від Mininet). У цьому курсі ми будемо використовувати контролер HP VAN SDN для керування комутаторами Mininet.

Як і багато інших мережевих систем віртуалізації, доступних для фізичних маршрутизаторів і комутаторів, головною перевагою Mininet є те, що складні мережі можуть бути створені для демонстрації, вивчення та тестування.
Mininet взаємодіє з контролерами OpenFlow. Контролер може бути використаний для оновлення таблиць потоку на комутаторах Mininet.
OpenFlow - це стандартний протокол, що підтримує площину централізованого керування в окремому пристрої (контролері).
OpenFlow здійснює апаратну абстракцію, забезпечуючи контролерові мережі спосіб зв'язку з пристроями різних виробників та різноманітними апаратними засобами (маршрутизаторами, комутаторами, балансувальниками навантаження тощо) та використовує стандартний інтерфейс.
Протокол реалізує логіку контролю за виконанням переадресації пакетів за допомогою правил пакетів та вкладає ці правила в апаратну абстракцію, де за ними може слідувати індивідуальний мережевий пристрій.
Традиційна комутація здійснюється комутаторами у відповідності до MAC-адрес.
Комутатори вивчають MAC-адреси пристроїв, які до них підключені і будують таблицю комутації (рисунок 7).

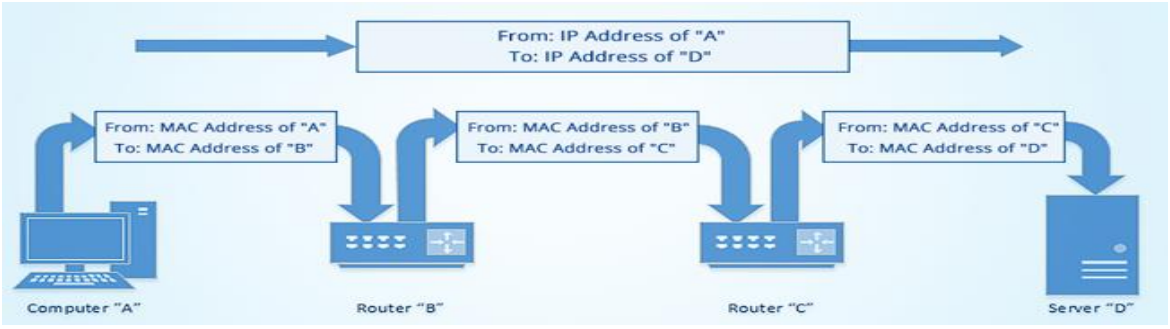



Рисунок 7– Традиційна комутація

У середовищі OpenFlow пристрої перш за все використовують таблиці потоків (flow tables), а не таблиці маршрутизації чи таблиці MAC-адрес.

Класичні мережеві комутатори підтримують таблицю переадресації, яка відображає адреси керування доступом до середовища (MAC) на фізичні порти. Коли комутатор отримує пакет, його MAC-адреса зіставляється з таблицею та надсилається з відповідного порту. Якщо MAC-адреса пакета не знайдена в таблиці переадресації, пакет транслюється на всі фізичні порти, по суті імітуючи роботу мережевого концентратора.
Коли комутатор вперше вмикається, таблиця переадресації порожня. Щоб дізнатися про підключені пристрої та заповнити таблицю, MAC-адреса джерела кожного отриманого пакета зберігається в таблиці переадресації та прив'язується до фізичного порту, на якому пакет було отримано.

Подібно до класичних мережевих комутаторів, OpenFlow-пристрій використовує узагальнену таблицю переадресації, яка називається таблицею потоків , щоб вирішити, що робити з вхідним пакетом. Таблиця потоків використовується для пошуку дій для кожного пакета, отриманого комутатором.
На відміну від класичних таблиць переадресації, ключі пошуку в таблиці потоків – це не лише MAC-адреси, а й узагальнені структури даних, які називаються ключами потоку , що містять загальні метадані про пакет, включаючи MAC-адреси, IP-адреси, порти TCP та UDP, теги VLAN тощо. Окрім ключа потоку, рядок у таблиці потоків також містить маску потоку – бітову маску, що вказує, які біти ключа потоку слід перевірити на збіг. Це

дозволяє реалізувати правила, що відповідають кільком пакетам із спільними значеннями заголовка.

Іншими словами, комутатор має конвеєр OpenFlow для обробки пакетів, а не традиційний процес, що використовує традиційні механізми комутації та маршрутизації (рисунок 8).

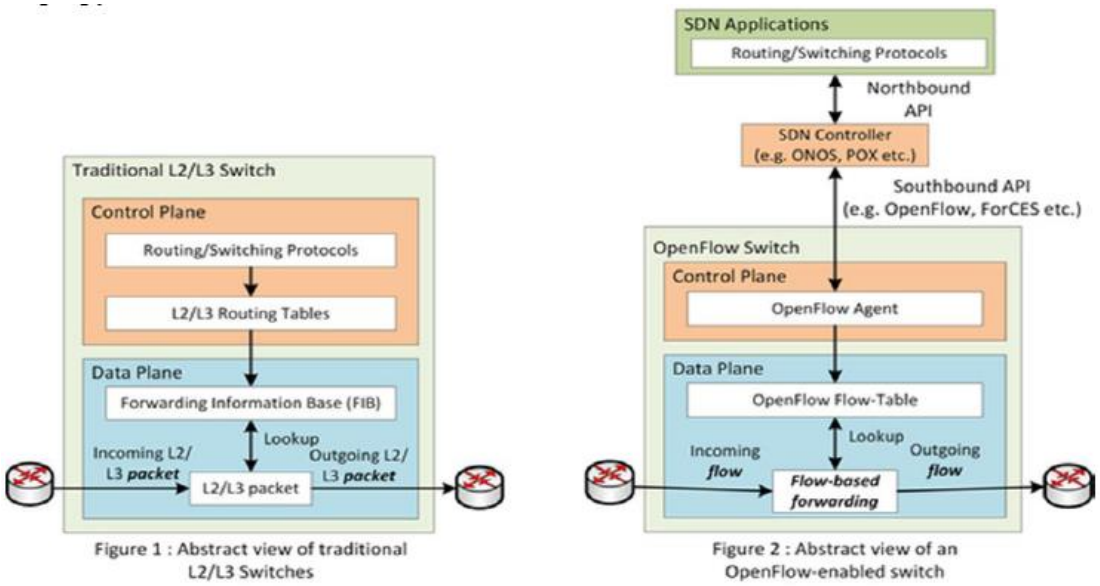



Рисунок 8 – Порівняння традиційної комутації з комутацією за допомогою таблиць потоків

Пристроєм OpenFlow може бути будь яке мережеве обладнання, яке підтримує протокол OpenFlow, наприклад комутатор. Кожний пристрій підтримує таблицю потоку (flow table), яка вказує на обробку, що застосовується до будь якого пакету певного потоку.
Протокол OpenFlow працює як інтерфейс між контролером та комутатором, налаштовуючи таблицю потоку.
Протокол OpenFlow реалізується на обох сторонах з'єднання мережевих пристроїв з мережевим контролером. OpenFlow використовує концепцію потоків для ідентифікації трафіку, базуючись на заздалегідь визначених правилах співставлення, які задаються статично або динамічно мережевим контролером. Таким чином можна контролювати, як трафік проходить через мережеві пристрої залежно від таких параметрів, як особливості використання мережі, навантаження, доступні ресурси хмарних і розподілених додатків, доступні ресурси зберігання даних. Відповідно, мережа програмується з визначенням окремих потоків.

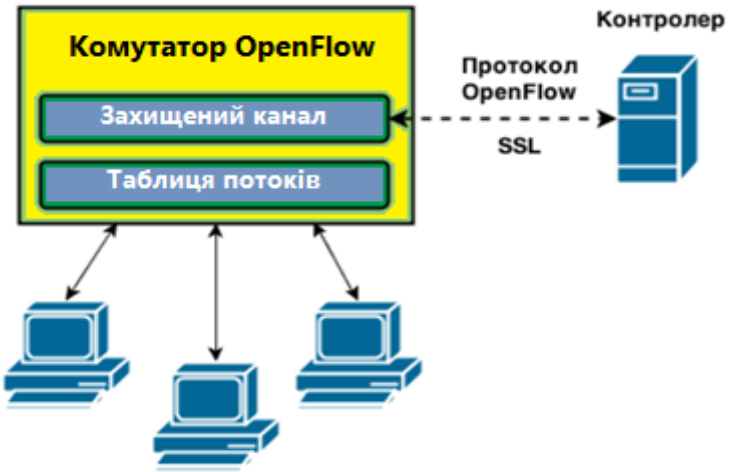



Рисунок 9– Комутатор OpenFlow. Таблиця потоків керується віддаленим контролером через захищений канал

Комутатор OpenFlow складається із однієї або декількох таблиць потоку (Flow Tables) та групової таблиці (Group Table), які здійснюють пошук і пересилання пакету, а також каналу OpenFlow до зовнішнього контролера. Комутатор здійснює комунікацію з контролером, а контролер керує комутатором через протокол OpenFlow (рисунок 9).

Таблиця потоку (Flow Table) оновлюється додаванням або вилученням потокових записів (flow entries) використовуючи протокол OpenFlow.
Таблиця потоку (Flow Table) містить деяку кількість потокових записів (flow entries), які асоціюються з діями для управління комутатором, який застосовує деякі дії (переадресація, відкидання або інкапсуляція) до певного потоку.
Потоковий запис в базовій таблиці потоку містить принаймі три поля:

- **Header field**: використовується для визначення умов точного збігу з потоком. Збіг потоку базується на визначених критеріях збігу.
- **Action** (instruction): дії які визначають, яким чином пакет повинен бути оброблений.
- **Counters** (statistics) відслідковує кількість пакетів і байтів для кожного потоку (наприклад, 100 пакетів, 8000 байт). Час з моменту останнього збігу потоку записується для видалення неактивних потоків. Це налаштовується в SDN-контролері.

Три дії, які повинні підтримуватися усіма виділеними комутаторами OpenFlow, є наступними:

- Forward
- Drop
- Redirect

**Forward**: Перший варіант – переадресація (пересилання) пакетів потоку на заданий порт (або набір портів). Це дозволяє здійснювати комутацію пакетів через мережу. У більшості комутаторів пересилання відбувається зі швидкостями лінії.

**Drop**: Другий варіант – відкидання пакетів потоку. Це може бути використано з міркувань безпеки, за рахунок чого блокується несанкціонований трафік, зупиняються DOS атаки або зменшується зайвий широкомовний трафік з кінцевих хостів.

**Redirect**: Третій варіант – інкапсуляція пакету та переадресація (пересилання) пакетів на контролер SDN. Контролер приймає рішення і пересилає пакет назад до комутатора. Як правило, цей метод використовується тільки для першого пакету нового потоку, завдяки чому контролер може вирішити, чи додавати цей потік до таблиці потоку.

Використовуючи протокол OpenFlow, контролер може додавати, оновлювати та видаляти потокові записи в таблиці потоків як реактивно (у відповідь на пакети) так і проактивно. Кожна таблиця потоку комутатора містить набір потокових записів; кожен потоковий запис місить поле збігу (match fields), лічильники (counters) та набір інструкцій, які необхідно застосувати до пакетів.
Процес перевірки збігу розпочинається з першої таблиці потоку і може закінчитись на додаткових таблицях. Потокові записи порівнюють на збігання пакети у пріоритетному порядку, починаючи з першого запису у кожній використаній таблиці (рисунок 10). Якщо знайдено запис, що збігається, то виконуються інструкції, які пов'язані з конкретним потоковим записом.
Якщо в таблиці потоку не знайдено жодного збігу, то результат залежить від конфігурації потокового запису в таблиці пропущених потоків (table-miss flow entry); наприклад, пакет може бути пересланим на контролер через канал OpenFlow, відкинутим, або може бути продовжена обробка у настуній таблиці потоку.

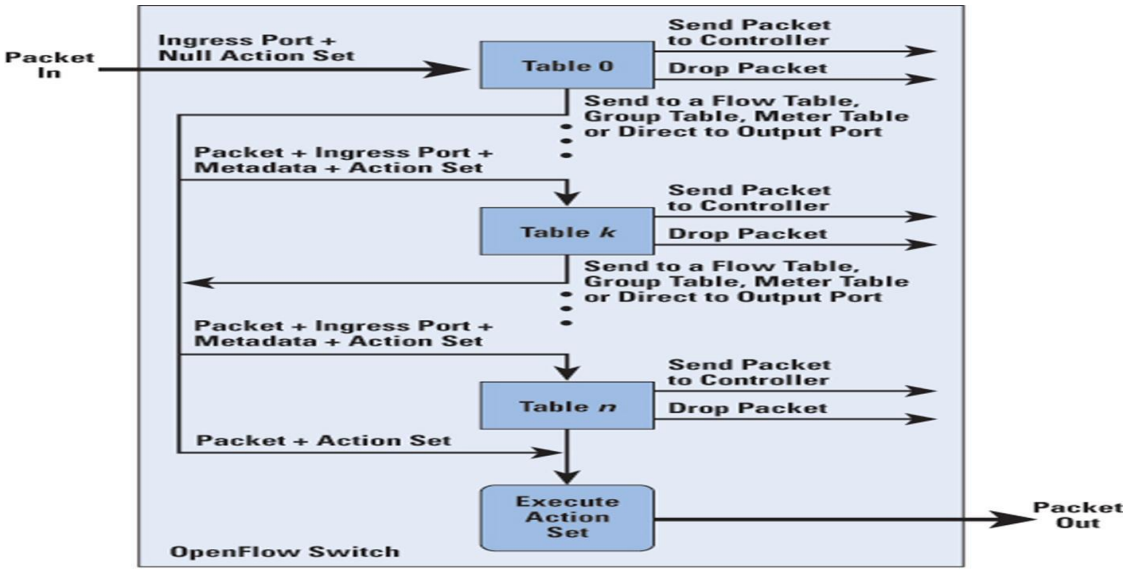



Рисунок 10 – Взаємодія таблиць потоків

Реактивна вставка потоку відбувається тоді, коли пакет потрапляє до комутатора OpenFlow без збігу потоку. Пакет відправляється контролеру, який його оцінює, додає відповідні потоки і дозволяє комутатору продовжувати переадресацію.
Альтернативно, потоки можуть бути вставлені контролером до комутатора проактивно, до прибуття пакетів.
**Компоненти комутатора на базі OpenFlow** Логічний комутатор OpenFlow складається з однієї або декількох таблиць потоків і таблиці груп, які виконують пошук і пересилання пакетів, і одного або декількох каналів OpenFlow на зовнішній контролер. Комутатор взаємодіє з контролером, і контролер керує комутатором по протоколу OpenFlow (рисунок 11).

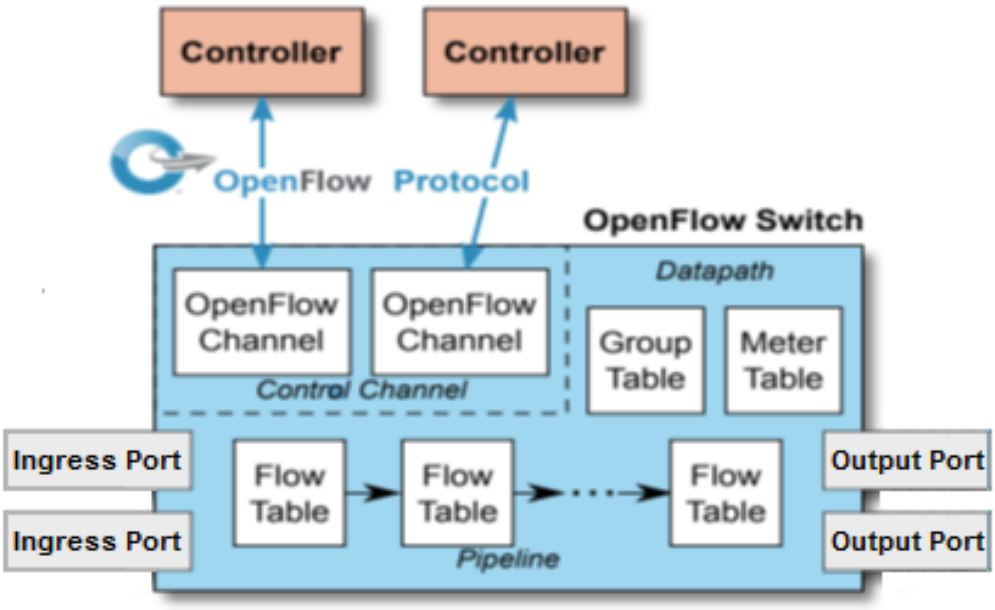



Рисунок 11 – Головні компоненти OpenFlow комутатора

OpenFlow надає наступні компоненти: таблиці потоків (flow tables), групові таблиці (group tables), таблиці метрик (meter tables), лічильники (counters). Ці інструменти дають можливість управляти потоками та розподіляти трафік за певними правилами в залежності від швидкості та інтенсивності надходження пакетів. Детальний розгляд даних компонентів описаний в наступних розділах.

**Flow table.** Таблиці потоків (flow tables) складаються з записів потоків (flow entries). Кожен з таких записів містить поля збігу, пріоритету, лічильника, інструкції для виконання, часу закінчення терміну валідності потоку, cookie, прапори. Запис таблиці потоку ідентифікується по полях відповідності та пріоритету: поля відповідності і пріоритет, взяті разом, ідентифікують унікальний запис потоку в певній таблиці потоку. Запис потоку може містити дії, які повинні бути виконані над пакетом в деякій точці конвеєра (рисунок 12).

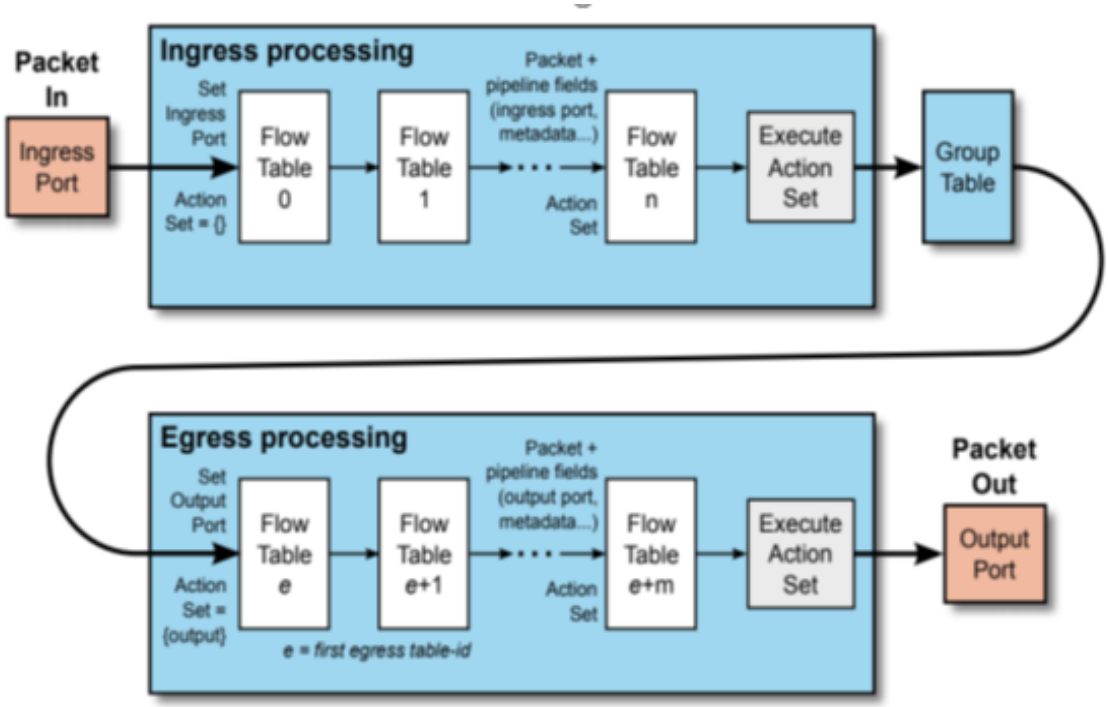



Рисунок 12 – Проходження пакетів через конвеєр обробки

Поля заголовка пакета витягуються з пакету, і витягуються поля конвеєра пакетів. Поля заголовка пакета, що використовуються для пошуку в таблиці, залежать від типу пакета і зазвичай включають різні поля заголовка протоколу, такі як адреса джерела Ethernet або адреса призначення IPv4. На додаток до заголовків пакетів, також можуть виконуватися зіставлення з вхідним портом, полем метаданих та іншими полями конвеєра. Кожна таблиця потоку повинна підтримувати запис потоку, що пропускає таблицю, щоб обробляти пропущені таблиці. Запис потоку з помилками в таблиці визначає, як обробляти пакети, несумісні з іншими записами потоку в таблиці потоків, і може, наприклад, посилати пакети контролеру, відкидати пакети або направляти пакети в наступну таблицю.

**Meter table.** Таблиці метрик складаються із записів, що визначають метрики кожного потоку. Поточні метрики дають можливість встановити обмеження швидкості, просту операцію QoS, що обмежує набір потоків для обраної смуги пропускання. Також поточні метрики дають можливість організувати більш складну обробку потоку з врахуванням QoS, наприклад, на основі вимірювання, яке може класифікувати набір пакетів по декількох категоріях на основі його швидкості (рисунок 13).

Пакети можуть проходити через декілька метрик при використанні метрик в послідовних таблицях потоків. Кожна полоса (meter band) складається з типу полоси, цільової швидкості для полоси тощо.

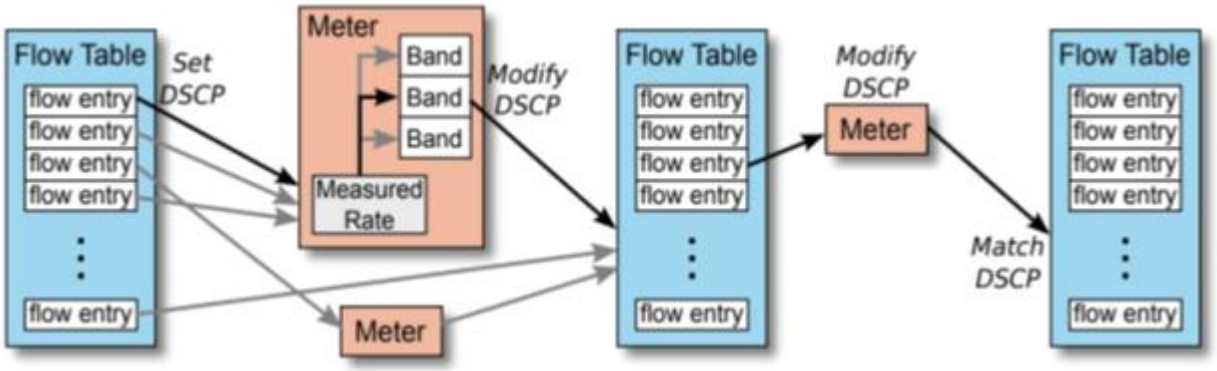


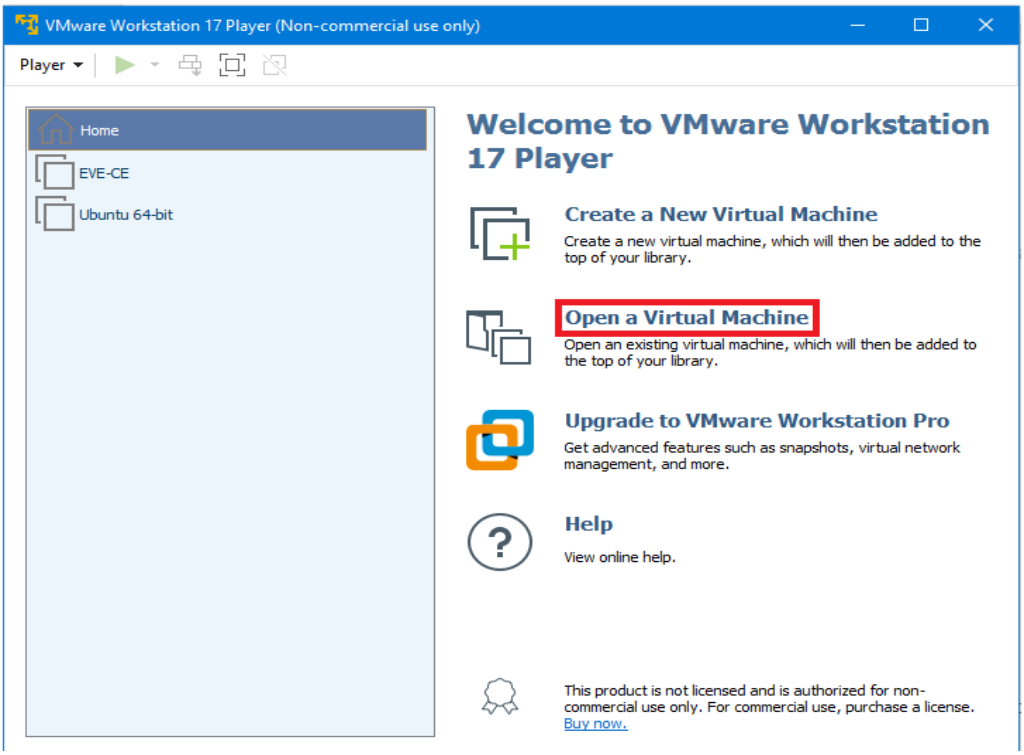

Рисунок 13 – Таблиці метрик

## 2. Виконання роботи

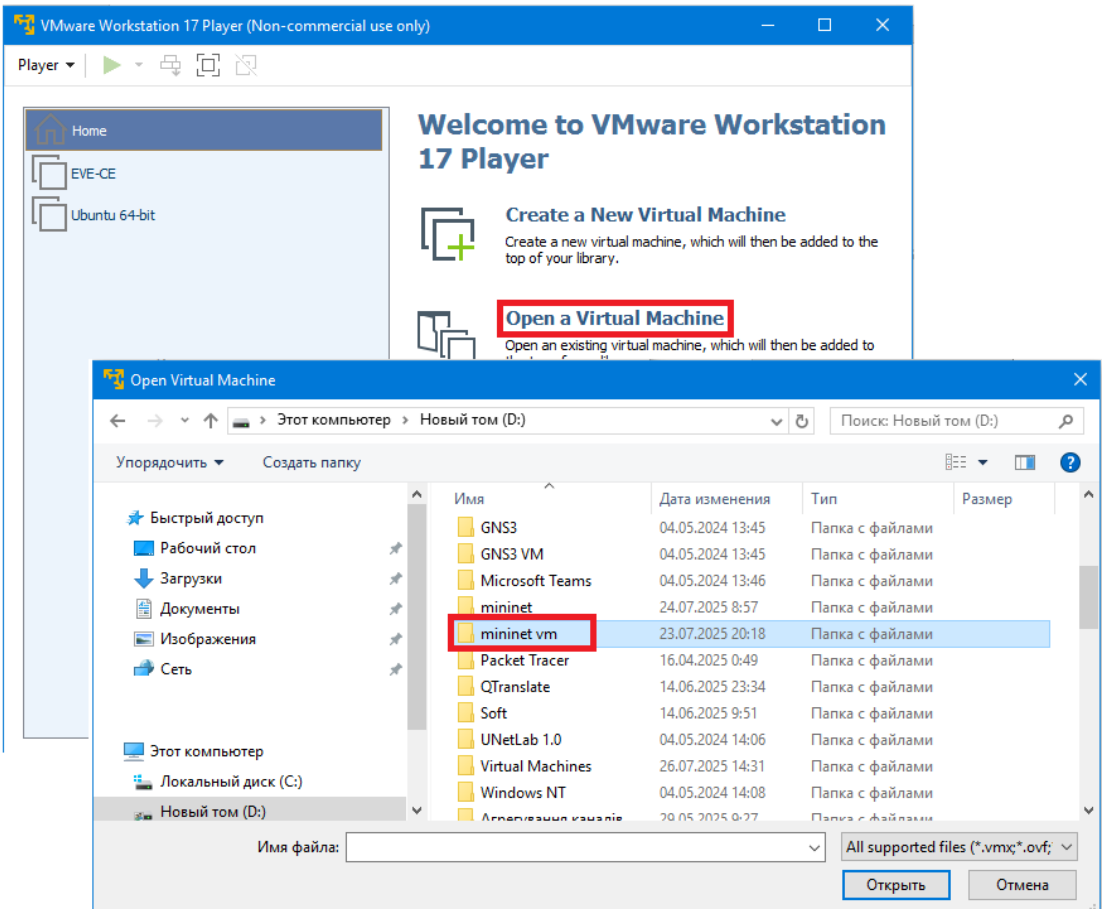
В лабораторній роботі використовуються такі програмні засоби:

- образ віртуальної машини  Mininet,
- система віртуалізації VMware Workstation Player,
- програма для віддаленого адміністрування Putty,
- графічний клієнт для Windows WinSCP.

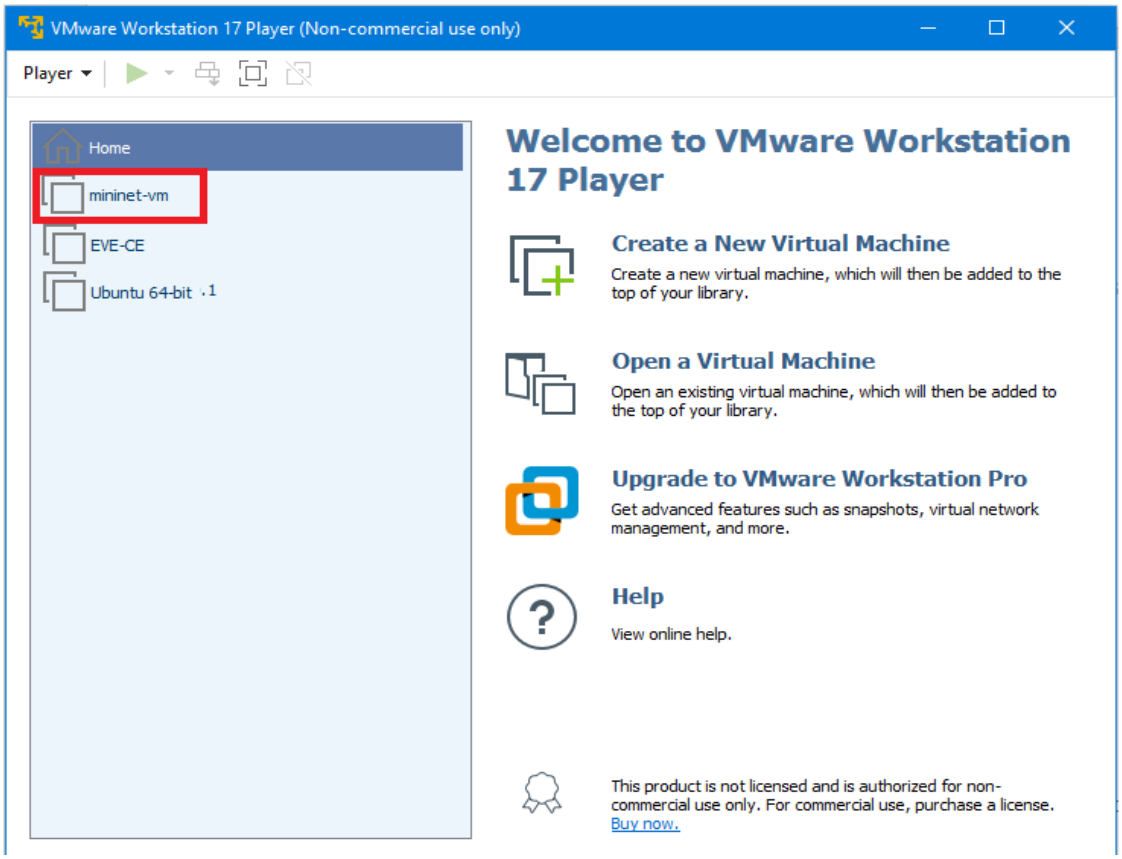
1. Завантажуємо образ віртуальної машини  Mininet / Ubuntu VM за посиланням https://github.com/mininet/mininet/releases/.  Ця VM включає сам Mininet, усі попередньо встановлені бінарні файли та інструменти OpenFlow. Ім’я архіву для завантаження *mininet-2.3.0-210211-ubuntu-18.04.5-server-amd64-ovf.zip*.
2. Завантажуємо та встановлюємо систему віртуалізації VMware Workstation Player.
3. Виконуємо розархівування завантаженого архіву, наприклад, в папку **mininet vm**.
4. Запускаємо VMware Workstation Player, імпортуємо завантажений образ, встановлюємо і запускаємо віртуальну машину Mininet.



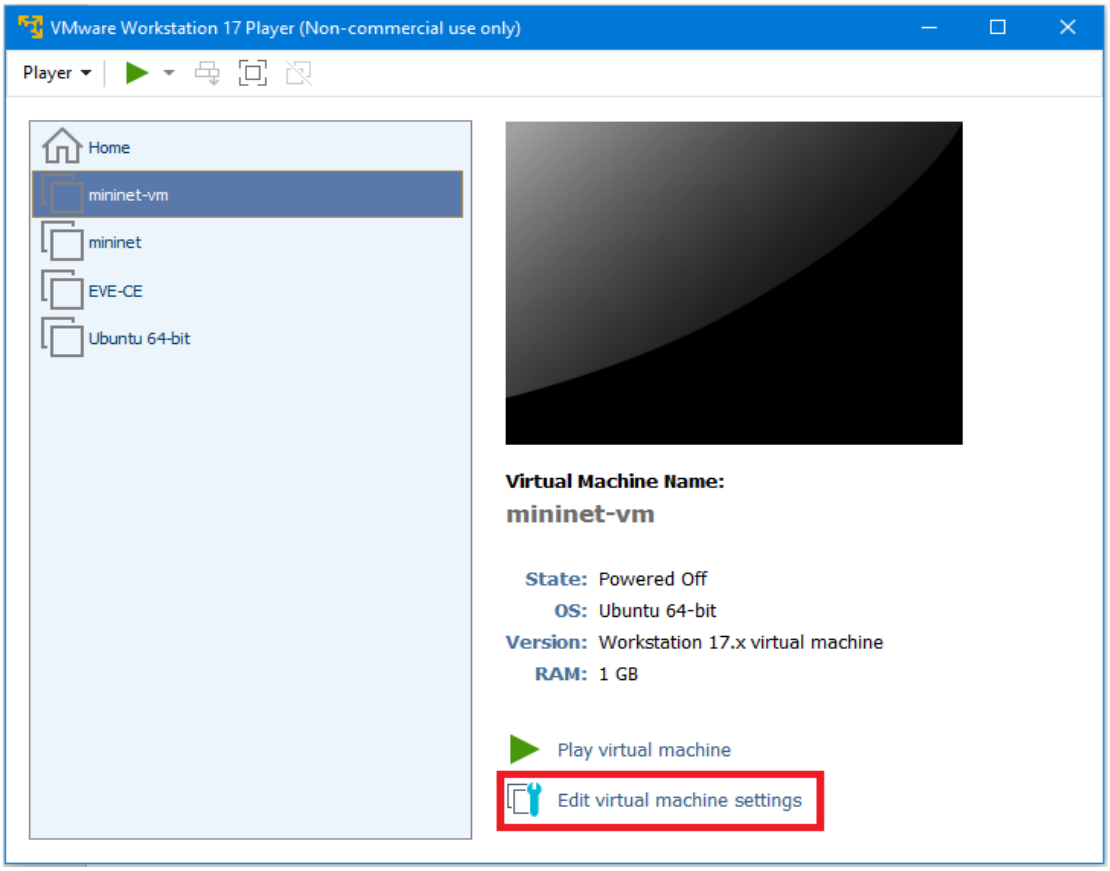
5. Відкриваємо образ віртуальної машини з каталогу **mininet vm**.

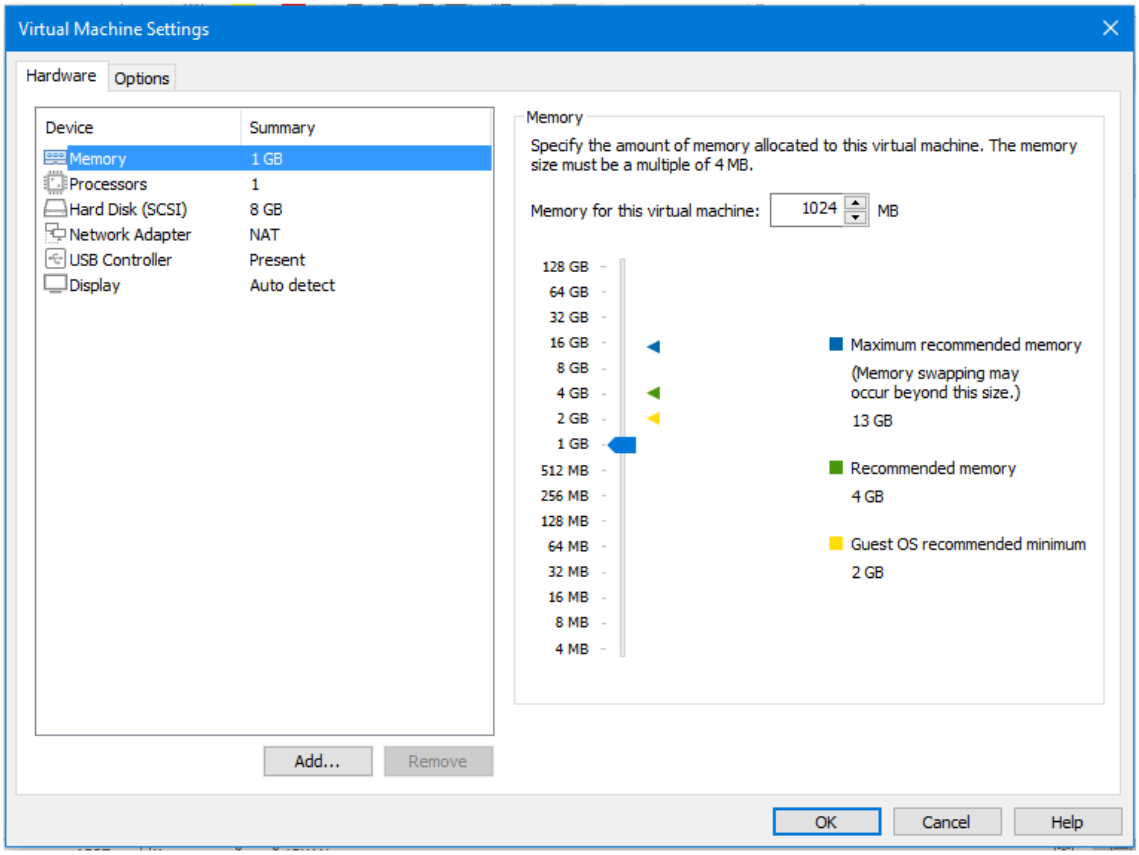



Після встановлення віртуальна машина **mininet-vm** готова до запуску.

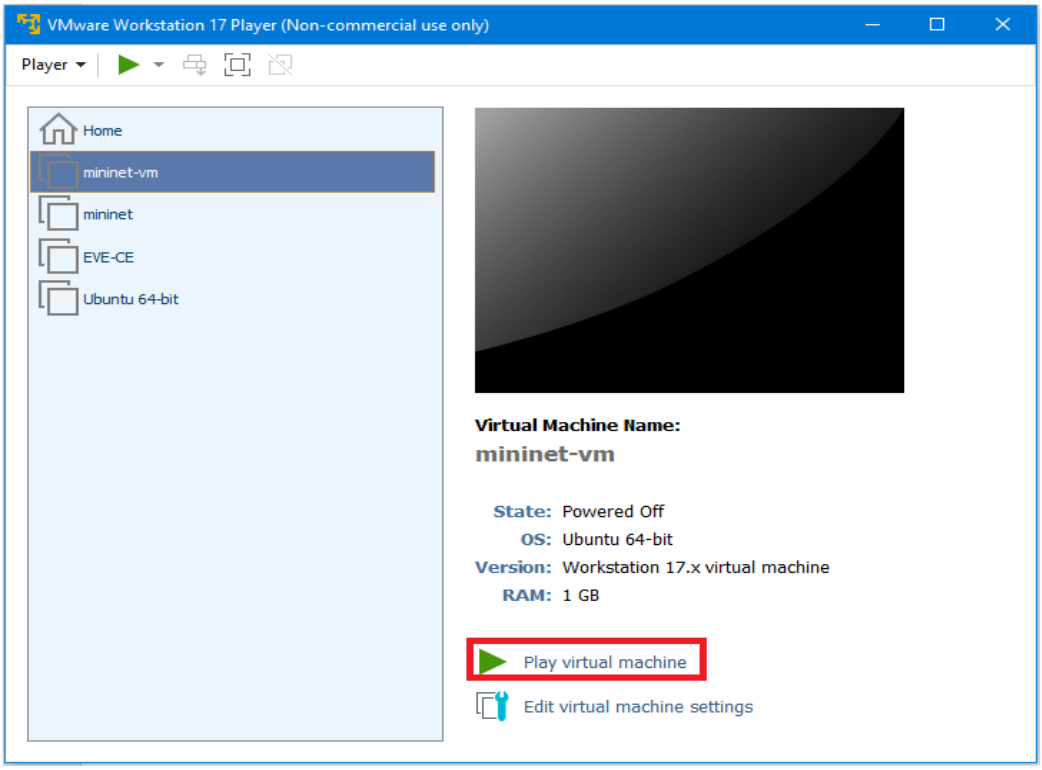


6. За потреби можна редагувати деякі параметри віртуальної машини. Залишаємо значення параметрів за замовчуванням.



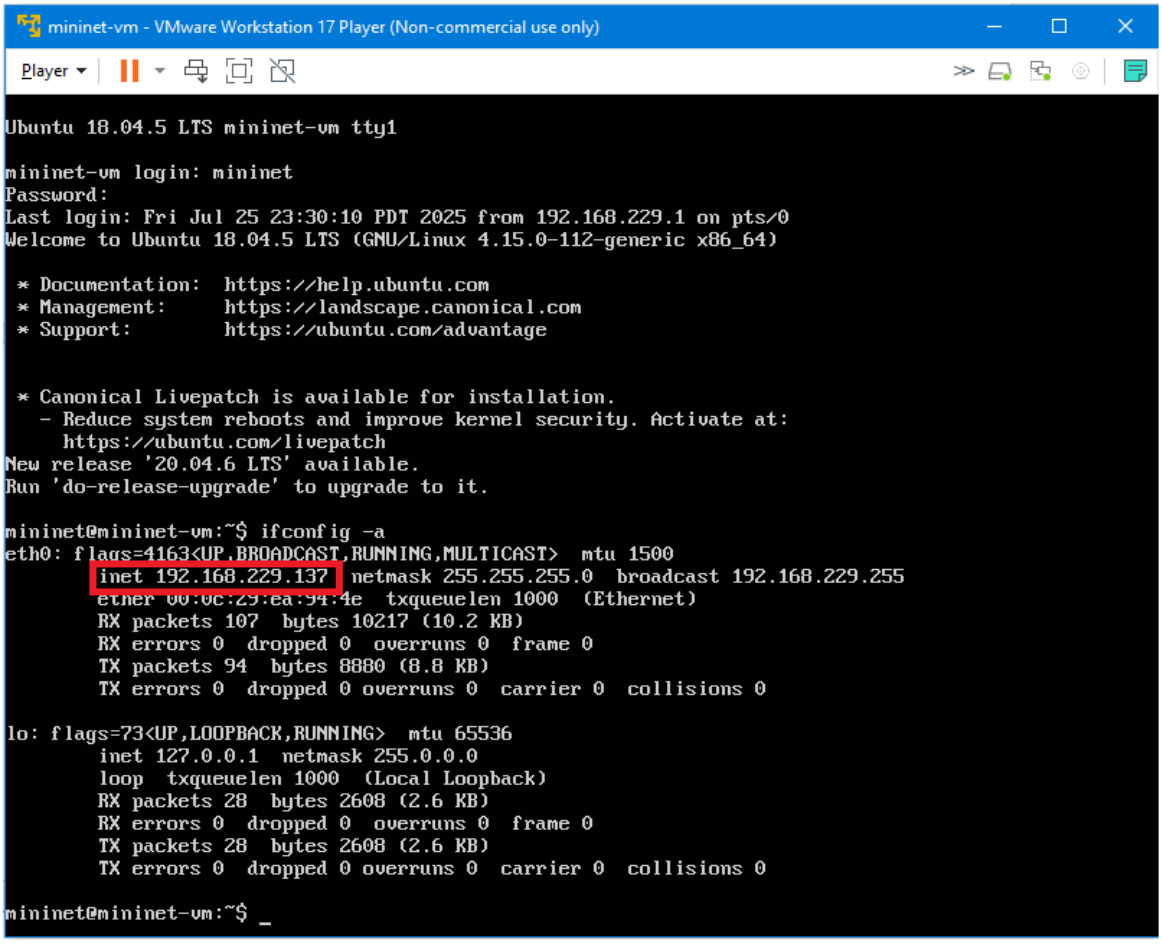




7. Вибираємо і запускаємо віртуальну машину Mininet.




У вікні, що відкрилося вводимо дані для автентифікації: login *mininet*, password *mininet*.

8. Визначаємо характеристики інтерфейсів віртуальної машини за допомогою команди *ifconfig -a* і запам'ятовуємо надану віртуальній машині Mininet IP-адресу:




Коректно завершувати роботу віртуальної машини Minitet потрібно за допомогою команди:

```
$ sudo shutdown -h now
```

9. Після встановлення Mininet потрібно виконати оновлення:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

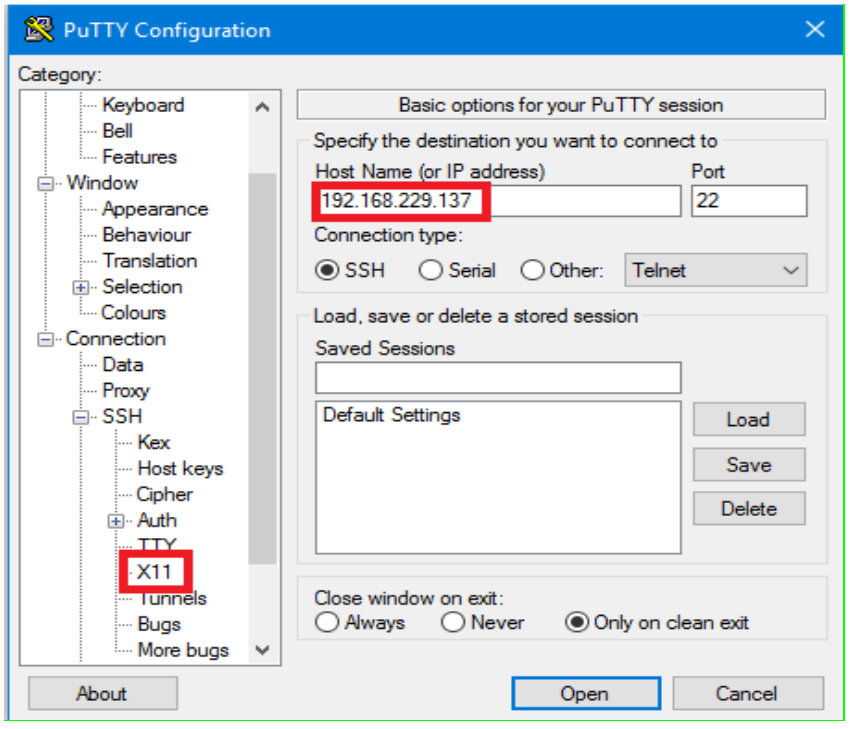
10. Встановлюємо і запускаємо ***Xming*** (або ***VcXsrv***) і ***Putty***.

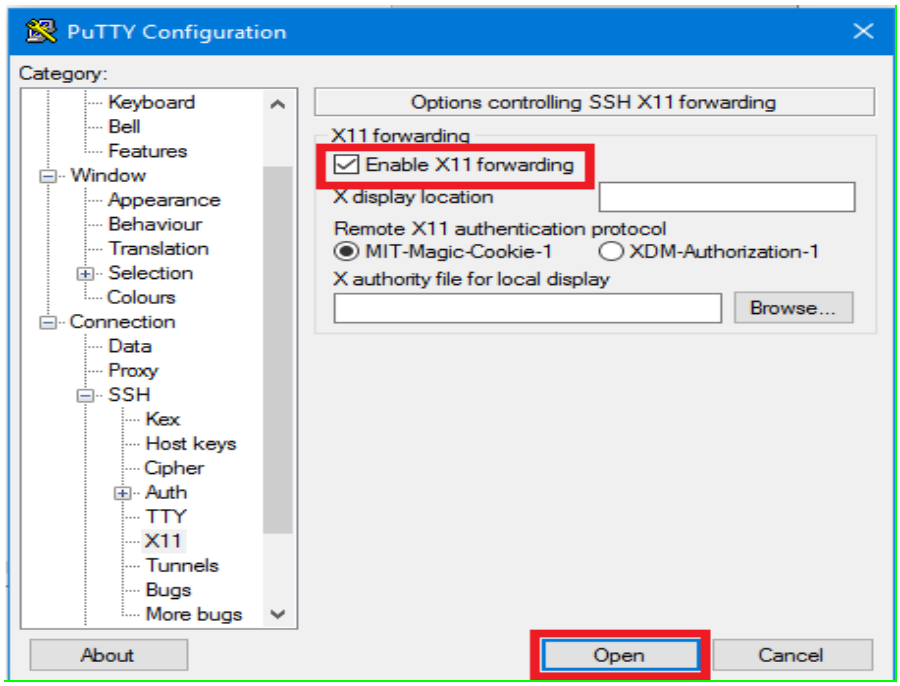
Xming - це безкоштовний X-сервер для Windows, який дозволяє відображати графічні застосунки, що працюють на віддаленому Linux-сервері, на комп'ютері з Windows. Встановлення і запуск Xming виконуємо в такій послідовності:

1. Встановлюємо Xming на Windows:

   • Завантажуємо та встановлюємо Xming з офіційного сайту або за допомогою менеджера пакетів. Завантажуємо, наприклад, файл Xming-6-9-0-31-setup.exe). При встановленні вибираємо опцію, яка включає встановлення XLaunch, майстра для налаштування Xming.
   • Після встановлення запускаємо XLaunch з меню "Пуск". У майстрі XLaunch вибираємо режим роботи "Multiple windows". В треї має з'явитися піктограма Xming.
2. Встановлюємо (якщо раніше не був встановлений) SSH-клієнт Putty для безпечного підключення до Linux-сервера.
3. При запуску Putty налаштовуємо SSH-тунель. Потрібно налаштувати SSH-тунель, щоб перенаправити трафік X11 через безпечне з'єднання:

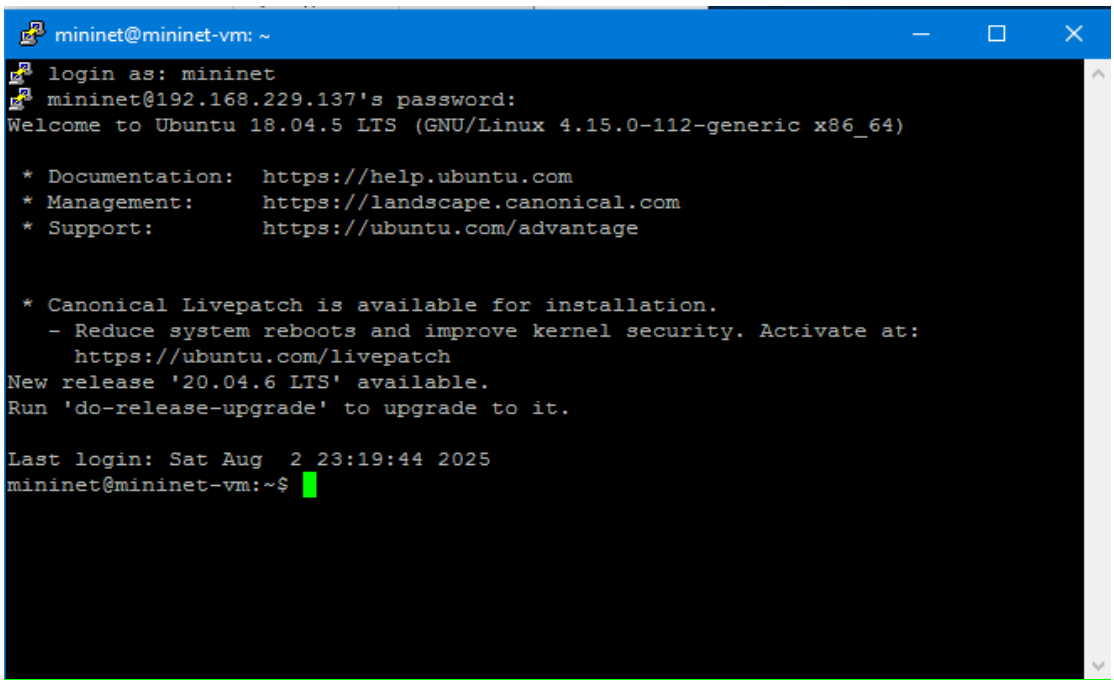
   - Переходимо до розділу "Connection -> SSH -> X11".
   - Встановлюємо прапорець "Enable X11 forwarding".
   - Зберігаємо налаштування сесії.
   - Запускаємо Putty. Адреса хоста має бути такою, яка була отримана при виконанні команди **ifconfig** в пункті 8.






Відбувається підключення до віртуальної машини Mininet за протоколом SSH.

11.У вікні, що відкрилося вводимо: login *minine*t, password *mininet*.




12. Продовжуємо виконання лабораторної роботи користуючись консоллю віртуальної машини Mininet.

**Зверніть увагу** на синтаксис команд, які використовуються у посібнику:

- «**$**» передує командам Linux, які слід вводити в командному рядку Putty,
- «**mininet**>» передує командам Mininet, які слід вводити в інтерфейсі командного рядка Mininet,
- «#» передує командам Linux, які вводяться у командному рядку терміналів емульованих хостів.

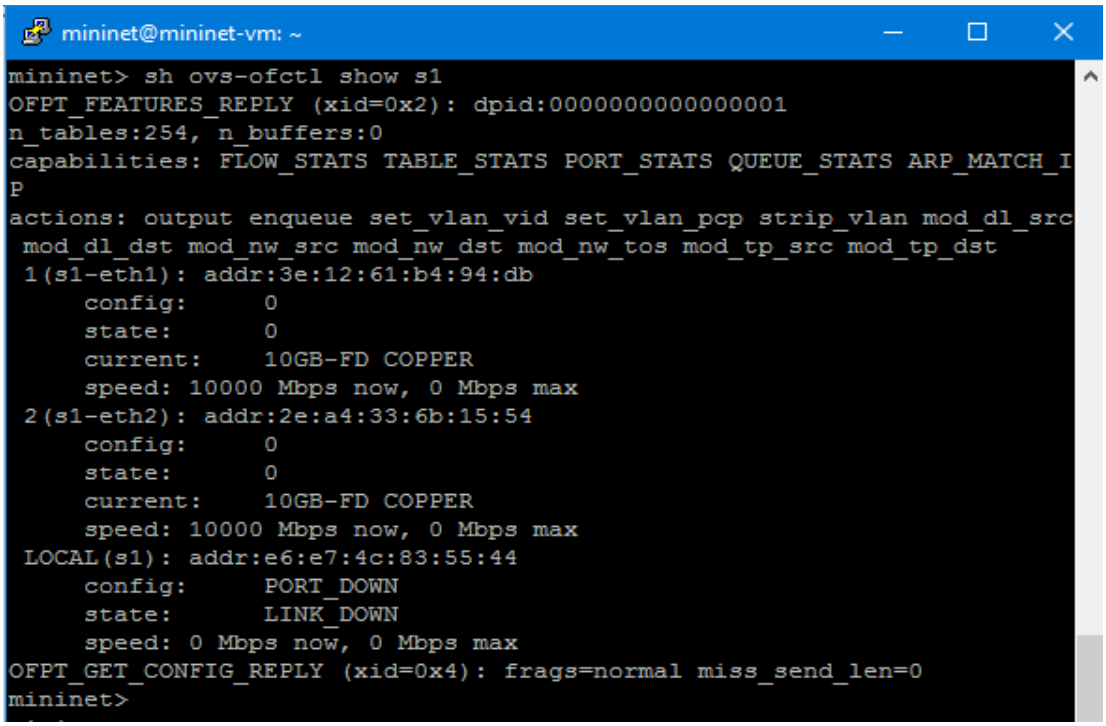
13. Ручне керування потоками. У цій частині роботи буде розглянуте керування записами потоку на Open vSwitch (OVS) уручну за допомогою утиліти **ovs-ofctl**. Записи потоку на комутаторі з підтримкою OpenFlow контролюють поведінку пакетів.

**ovs-ofctl** – це утиліта командного рядка для керування та моніторингу OpenFlow-комутаторів, зокрема Open vSwitch (OVS). Вона дозволяє переглядати та змінювати стан комутатора, включаючи його конфігурацію, таблиці потоків (flow tables) та порти. **ovs-ofctl** працює з протоколом OpenFlow, який є стандартом для управління мережевим обладнанням, такими як комутатори.

Утиліта ovs-ofctl дозволяє:

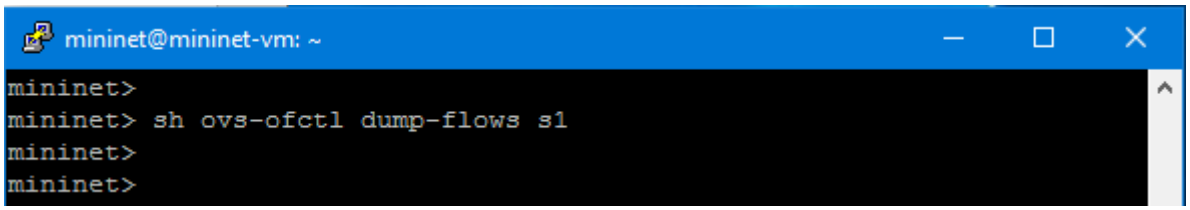
- Показати інформацію про комутатори, наприклад, *ovs-ofctl show s1* покаже інформацію про комутатор s1.
- Виконувати керування потоками: *ovs-ofctl* дозволяє додавати, видаляти та змінювати правила (потоки) у таблицях комутаторів, які визначають, як слід пересилати трафік.
- Переглядати статистики портів: *ovs-ofctl dump-flows* показує записи таблиці потоків OpenFlow.

Команда **sh ovs-ofctl show s1** відображає інформацію про комутатор OpenFlow з назвою "s1", яким керує Open vSwitch (OVS). Вона надає детальну інформацію про інтерфейси, порти, функції та іншу інформацію про конфігурацію комутатора. Ця команда корисна для перевірки поточного стану комутатора та перевірки його конфігурації в середовищі Mininet або загальних налаштуваннях OVS. Надається детальна інформація про кожний підключений до комутатора порт, зокрема номер порту, ім’я інтерфейсу, MAC-адреса, стан з’єднання (вгору/вниз) і позначки конфігурації порту.




Команда **sh ovs-ofctl dump-flows s1** використовується для відображення записів таблиці потоків OpenFlow для комутатора з іменем "s1" у Open vSwitch. Ця команда корисна для розуміння того, як мережевий трафік обробляється комутатором на основі налаштованих правил. Команда виводить список правил потоку, зокрема критерії відповідності (до якого

трафіку вони застосовуються) та дії (що робити з відповідним трафіком). Ця інформація має вирішальне значення для виявлення мережевих проблем, перевірки конфігурацій та розуміння поведінки мережі на базі OpenFlow.

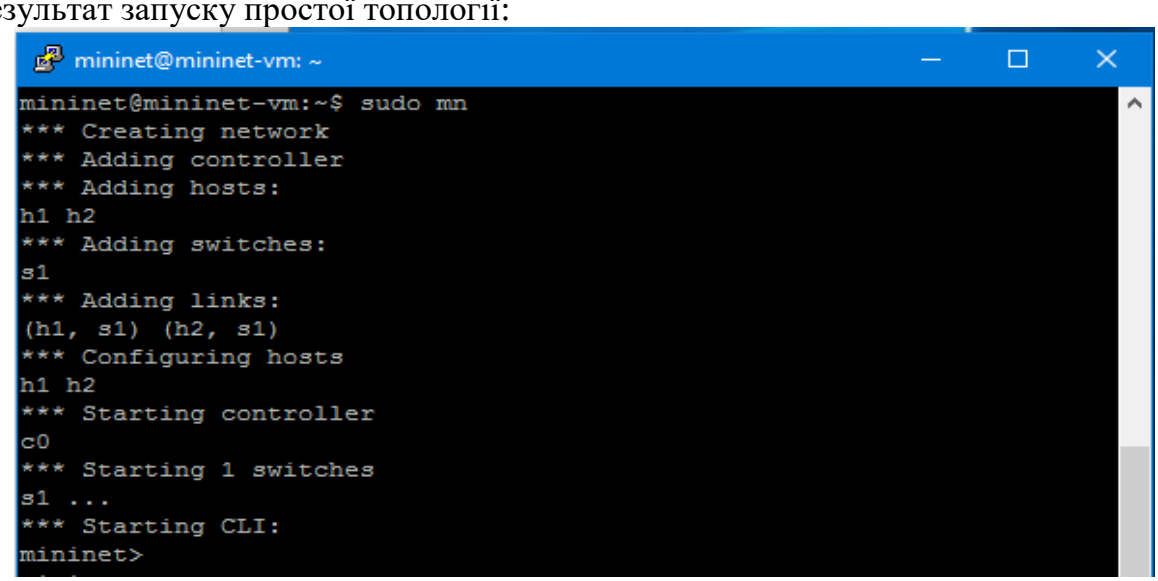



Потоків ще створено не було, тому зараз таблиця пуста. Таблиця заповниться, коли будуть створені потоки.

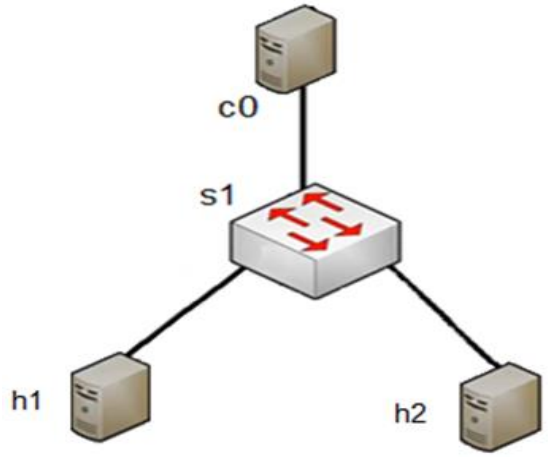
14. Якщо запускати Mininet без опцій, то за замовчуванням Mininet створює топологію, що складається з одного контролера ( ***c0*** ), одного комутатора ( ***s1*** ) та двох хостів ( ***h1*** та ***h2*** ).

$ **sudo mn**

Результат запуску простої топології:




Створену топологію графічно можна представити у такому вигляді:




Mininet підтримує різні топології для імітації різних мережевих конфігурацій. Ось список часто використовуваних топологій та інструкції з їх запуску:

- **Одинарна топологія**: один комутатор, підключений до кількох хостів.
  Приклад: **$ sudo mn --topo single,4** (створює один комутатор і чотири хости).

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 *** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
```

- **Лінійна топологія**: лінійний ланцюжок комутаторів, кожен з яких підключений до хоста.
  Приклад: **$ sudo mn --topo linear,4** (створює чотири комутатори, кожен з яких підключений до одного хоста, а комутатори з'єднані між собою).

```
*** 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) (s1, s2) (s2, s3) (s3, s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 4 switches
s1 s2 s3 s4
*** Starting CLI:
```

- **Деревоподібна топологія**: ієрархічна топологія з деревоподібною структурою.
  Приклад: **$ sudo mn --topo tree,depth=3,fanout=2** (створює дерево глибиною 3 та 2 гілками на вузол).

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6) (s5, s7) (s6, h5) (s6, h6) (s7, h7)
(s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
```

Для роботи з Mininet можна використовувати такі ключі запуску:

| Команда | Дія |
|---|---|
| -h, -help | Отримання довідки |
| -host= | Задає адресу запуску Mininet |
| -switch= | Вибір типу програмно-керованого комутатора. Варіанти: [kernel, user, ovsk] |
| -controller | Задає тип контролера. Варіанти: [nox_dump, none, ref, remote, nox_pysw] |
| -topo= | Задає параметри топології. Варіанти: [tree, reversed, single, linear, minimal] |
| -c, -clean | Закриває mininet и очищає всі віртуальні машини |
| -mac | Включає формування різних MAC-адрес для хостів |
| -x, -xterms | Запускає консолі в графічному режимі для кожного хоста |
| -arp | Встановлює arp-записи в хости в статичному режимі |
| -ip= | Задає IP-адресу контролера |
| -port= | Задає порт контролера |
| efixlen= | Довжина маски для автоматичного розподілу адрес |

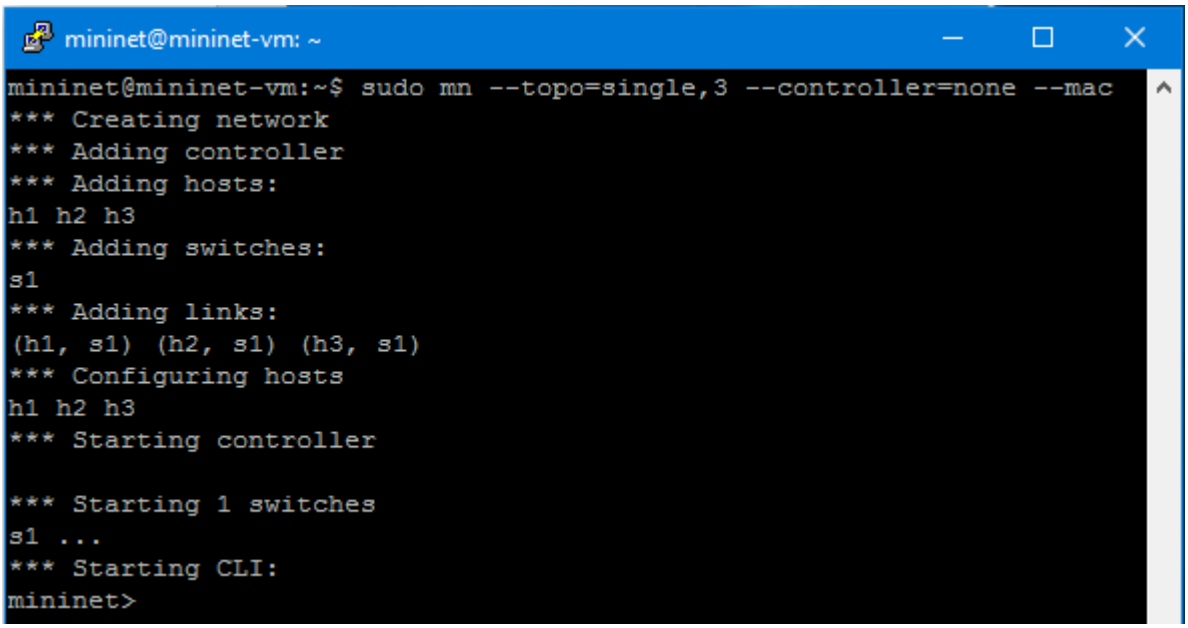
Виходимо із інтерфейсу командного рядка віртуальної машини Mininet.

mininet> **exit**

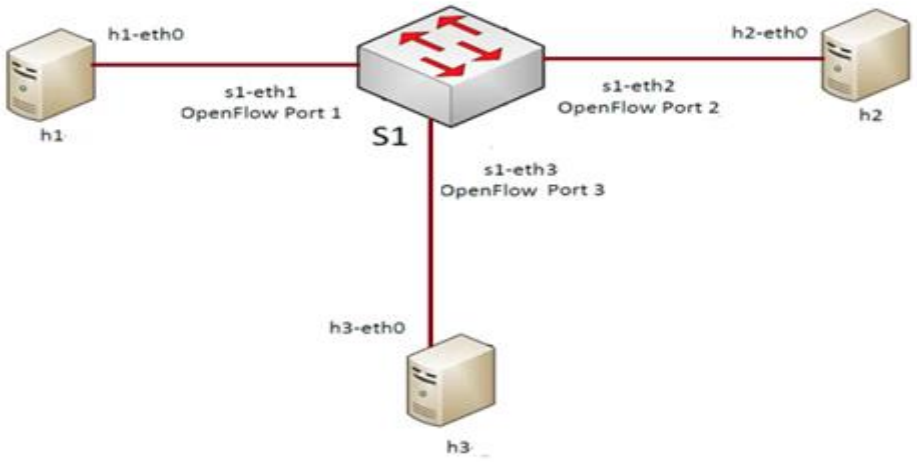
15. Для наступної вправи створимо віртуальну мережу з одним комутатором та трьома хостами, виконуючи команду з консолі Mininet, наведену нижче:

$ **sudo mn --topo=single,3 --controller=none --mac**

- `--controller=none` означає, що команди будуть надаватися вручну,
- `--mac` означає, що mac-адреси будуть спрощені.




Топологія створеної мережі має такий вигляд:




Тепер виконаємо команди *dump* та *net*, щоб перевірити створену топологію.

```
mininet@mininet-vm: ~
mininet>
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1462>
<Host h2: h2-eth0:10.0.0.2 pid=1464>
<Host h3: h3-eth0:10.0.0.3 pid=1466>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=
1471>
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
mininet>
```

Визначимо відповідність номерів портів s1 комутатора і номерів портів OpenFlow (OFP):

mininet> **sh ovs-ofctl show s1**

```
mininet@mininet-vm: ~
mininet> sh ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_I
P
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
 mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
 1(s1-eth1): addr:26:e5:af:6c:27:42
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:7a:ac:32:b8:0a:a9
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:c6:30:a5:5a:29:e1
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:c6:9e:27:18:ab:4a
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
mininet>
```

Можна побачити, що порту OFP **1** відповідає порт комутатора до s1-eth1, порту OFP **2** – s1-eth2, порту OFP **3** – s1-eth3.

Команда **sh** в інтерфейсі командного рядка (CLI) Mininet дозволяє виконувати зовнішні команди оболонки (Ubuntu).

mininet> **sh ovs-ofctl add-flow s1 action=normal**

тут **normal** означає традиційну поведінку комутатора, тобто класичні операції переадресації комутатора.

Команди оболонки можна виконати також з окремого термінала xterm за допомогою sudo, наприклад,

$ **sudo ovs-ofctl show s1**

Перевіримо наявність з'єднання між хостами за допомогою команди *pingall*:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Тепер перевіримо записи в таблиці потоків комутатора:

mininet> **sh ovs-ofctl dump-flows s1**

```
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=369.906s, table=0, n_packets=24, n_bytes=1680, actions=NORMAL
mininet>
```

Ось деякі з полів, на які слід звернути увагу:

- **duration** – кількість секунд, протягом яких запис знаходиться у таблиці,
- **table** – конкретна таблиця, у якій встановлено потік,
- **n_packets** – кількість пакетів, що відповідали запису,
- ***action=normal*** – вказує на дію, яку потрібно виконати, коли пакет відповідає запису потоку. Дія **n*ormal*** дозволяє пристрою виконувати нормальну обробку пакетів.

Видалимо створений потік за допомогою команди *del-flow*:

mininet> **sh ovs-ofctl del-flows s1**

Ця команда видаляє всі потоки в s1. Тепер перевіримо таблицю потоків ще раз:

mininet> **sh ovs-ofctl dump-flows s1**

```
mininet> sh ovs-ofctl del-flows s1
mininet> sh ovs-ofctl dump-flows s1
mininet>
```

Бачимо, що більше не визначено жодних потоків.

Тепер ще раз виконуємо команду pingall: спостерігаємо, що тепер хости недоступні один для одного.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet>
```

16. **Використання даних рівня 1.** У цій частині роботи будемо працювати на рівні фізичних портів. Запрограмуємо комутатор таким чином, щоб усе, що надходить на комутатор s1 з порту OpenFlow 1, надсилалося на порт OpenFlow 2, і навпаки. Потім створимо потік з дією, яка буде відкидати будь-який пакет, який досягає таблиці потоків і відповідає її (наразі порожнім) критеріям.

Отже, запишемо два потоки:

mininet> **sh ovs-ofctl add-flow s1 priority=500,in_port=1,actions=output:2**
mininet> **sh ovs-ofctl add-flow s1 priority=500,in_port=2,actions=output:1**

Ці дві команди вказують комутатору, що дані, які надходять з порту 1, мають бути перенаправлені на порт 2 і навпаки. Нам потрібні два правила для досягнення двонаправленого з'єднання (echo-запит та echo-відповіді):

Виконаємо команди:

mininet> **h1 ping -c2 h2**

```
mininet@mininet-vm: ~
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=1,actions=output:2
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=2,actions=output:1
mininet> h1 ping -c2 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=0.240 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.076 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.076/0.158/0.240/0.082 ms
mininet>
```

Хости h1 та h2 доступні один для одного.

Тепер виконаємо ще раз команду dump-flows:

mininet> **sh ovs-ofctl dump-flows s1**

```
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=274.055s, table=0, n_packets=4, n_bytes=280, priority=500,in_port="
s1-eth1" actions=output:"s1-eth2"
 cookie=0x0, duration=255.840s, table=0, n_packets=4, n_bytes=280, priority=500,in_port="
s1-eth2" actions=output:"s1-eth1"
mininet>
```

Можна побачити два новостворені потоки та інформацію про них. Значення пріоритету є важливим. Якщо пакет потрапляє на комутатор, і для нього діють різні правила, виконується лише те, яке має вище значення пріоритету.

Додаємо ще один потік з дією *drop*. Ця дія забороняє будь-які потоки і має вищий пріоритет, ніж у попередніх правил.

mininet> **sh ovs-ofctl add-flow s1 priority=32768,action=drop**

Спробуємо знову перевірити доступність хостів. Зараз хости недоступні один для одного.

```
mininet> sh ovs-ofctl add-flow s1 priority=32768,action=drop
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet>
```

17. Налаштування пріоритету потоків. У цьому розділі ми додамо потік, який має вищий пріоритет ніж пріоритет існуючого потоку протягом певного часу.

Пріоритет може мати значення від 0 до 65535 і значення за замовчуванням 32768. Коли одночасно задані два записи правил потоку, запис правил із вищим пріоритетом буде спрацьовувати раніше, ніж запис правила з нижчим пріоритетом.

- Перевіримо існуючі правила потоку:

  mininet> **sh ovs-ofctl dump-flows s1**

```
mininet@mininet-vm: ~/mininet/examples
mininet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=208.393s, table=0, n_packets=0, n_bytes=0, actions=drop
 cookie=0x0, duration=2107.634s, table=0, n_packets=12, n_bytes=784, priority=500,in_port="
s1-eth1" actions=output:"s1-eth2"
 cookie=0x0, duration=2094.910s, table=0, n_packets=12, n_bytes=784, priority=500,in_port="
s1-eth2" actions=output:"s1-eth1"
mininet>
```

Як бачимо, у таблиці потоків є двонаправлене правило, яке дозволяє обмінюватися пакетами хостам h1та h2 і правило із дією *drop*.

Видалимо потік з дією *drop*, який використовувався в попередньому завданні:

mininet> **sh ovs-ofctl del-flows s1 --strict "priority=32768"**

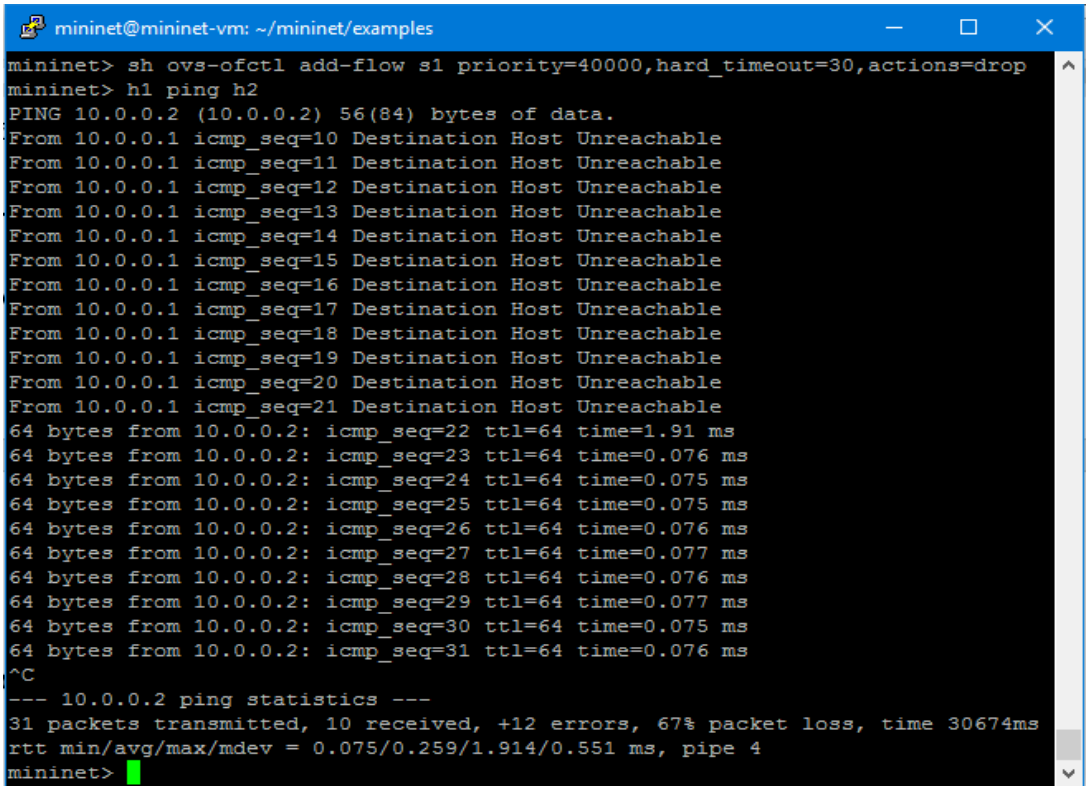
Прапорець «--strict» і фільтр «**priority=32768**» дозволяють видалити певне правило з таблиці потоків.

Створимо потік, щоб **тимчасово** припинити всі підключення:

mininet> **sh ovs-ofctl add-flow s1 priority=40000,hard_timeout=30,actions=drop**

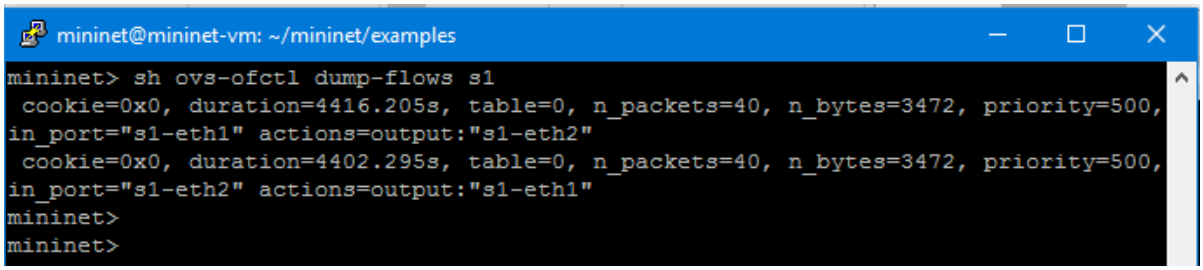
**priority** – Встановлюємо пріоритет більшим ніж визначений за замовчуванням (32768).
**hard_timeout** – Кількість секунд, протягом яких правило буде дійсним, перш ніж воно буде видалене.
**actions=drop** – Весь трафік, що проходить через s1, буде відхилений.

Згідно з цим визначенням потоку, ми очікуємо відсутність з'єднання між h1 і h2 протягом 30 секунд. Після цього періоду жорсткого тайм-ауту потік буде видалений, і з'єднання відновиться.
Запускаємо постійний *ping* між h1 і h2:




Як і очікувалося, нове правило потоку заборонило мережеву активність на s1 протягом вказаного часу *hard_timeout*. Після цього правило потоку було видалене, і комутатор знову встановив з'єднання.

Переглянемо потоки ще раз і звертаємо увагу на значення *n_packages*:




Спостерігаємо, що кількість пакетів для перших двох потоків збільшилася через пінг.

Виходимо із Mininet та видаляємо раніше створені потоки:

```
mininet> exit
```

Рекомендується після кожного дослідження очищати топології Mininet для видалення будь-яких несподіваних процесів за допомогою команди *sudo mn -c*, щоб в пам'яті не залишилося колишніх топологій:

```
$ sudo mn -c
```

Переглядаємо таблицю потоків. Записи в таблиці відсутні.

mininet> **sh ovs-ofctl del-flows s1**

```
mininet> sh ovs-ofctl del-flows s1
mininet> sh ovs-ofctl dump-flows s1
mininet>
```

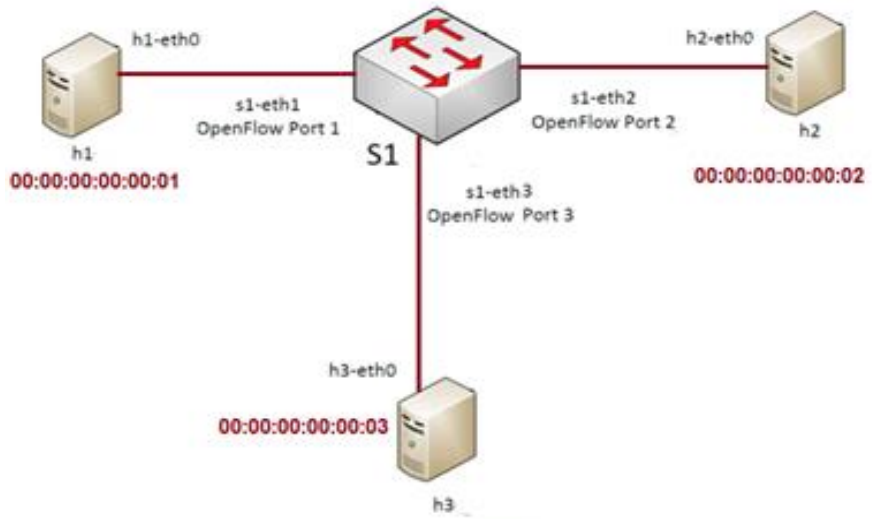
## 18. **Використання даних рівня 2.**

Знову створюємо мережу:

$ **sudo mn --topo=single,3 --controller=none --mac**

У цій частині роботи ми повторимо ті ж операції, але замість номерів портів будемо використовувати MAC-адреси хостів. Оскільки ми виконуємо Mininet з параметром ***--mac*** хости матимуть спрощені MAC-адреси.

Перевіримо конфігурацію інтерфейсів на хостах h1, h2 та h3. Нам потрібні IP- і MAC-адреси.




```
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 13  bytes 882 (882.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 33  bytes 1890 (1.8 KB)
        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 5  bytes 560 (560.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5  bytes 560 (560.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
mininet> h2 ifconfig
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 00:00:00:00:00:02  txqueuelen 1000  (Ethernet)
        RX packets 13  bytes 882 (882.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 1638 (1.6 KB)
        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 1  bytes 112 (112.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 112 (112.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

```
mininet> h3 ifconfig
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.3  netmask 255.0.0.0  broadcast 10.255.255.255
        ether 00:00:00:00:00:03  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 602 (602.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 25  bytes 1386 (1.3 KB)
        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 4  bytes 448 (448.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 448 (448.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet>
```

Додамо два потоки і перевіримо роботу хостів за допомогою команди *pingall*.

mininet> **sh ovs-ofctl add-flow s1**
**dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=output:2**
mininet> **sh ovs-ofctl add-flow s1**
**dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=output:1**

Увага! У «dl_src» літера «L»

```
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actio
ns=output:2
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actio
ns=output:1
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```

Як бачимо, «пінги» не проходять, оскільки *ping* працює на рівні IP, і перше, що він робить, це використовує ARP для з'ясування MAC-адрес різних хостів. ARP – це протокол широкомовлення, і комутатор, з урахуванням поточних правил, фільтрує цей тип трафіку. Тому потрібно додати ще одне правило:

mininet> **sh ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,action=flood**

Це правило додає потік, який направляє всі кадри Ethernet типу 0x806 (ARP) на всі порти комутатора.

Тут **nw_proto=1** – вказує на  ARP-запит . Оскільки відповіді в ARP є одноадресними, то у нас вже визначені всі потоки.

Після додавання нового правила потоку знову перевіряємо його за допомогою команди *pingall.*

```
mininet> sh ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,action=flood
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)
mininet>
```

Тепер h1 та h2 контактують, але h3 все ще від'єднаний, тому, що ми не додали жодного правила для хоста h3.

Додамо ще два потоки для h2 і h3 і перевіримо доступність хостів за допомогою команди *pingall.*

mininet> **sh ovs-ofctl add-flow s1**
**dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02,actions=output:2**
mininet> **sh ovs-ofctl add-flow s1**
**dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03,actions=output:3**

```
mininet> sh ovs-ofctl add-flow s1  dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:0
0:02,actions=output:2
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00
:03,actions=output:3
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 h3
h3 -> X h2
*** Results: 33% dropped (4/6 received)
mininet>
```

Тепер хости h2 і h3 доступні один для одного. Відбулося підключення трафіка через порт s1-eth3 комутатора.

Видаляємо усі потоки в комутаторі.

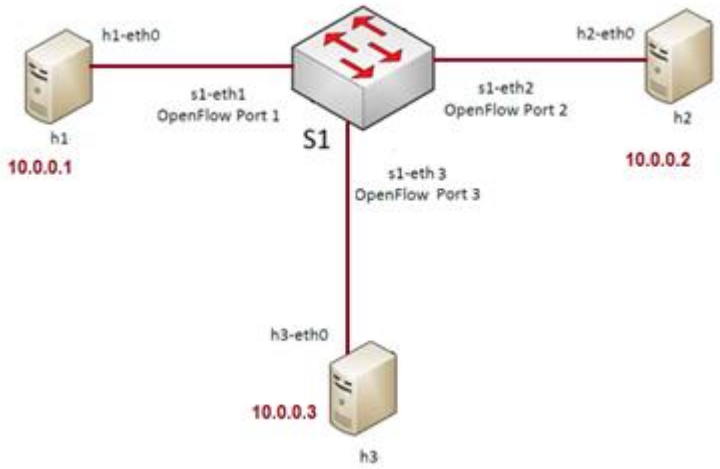
mininet> **sh ovs-ofctl del-flows s1**

## 19. **Використання даних рівня 3.**

У Openflow існують різні правила, які дозволяють **змінювати вміст пакетів** . У цій частині ми розглянемо **базовий приклад використання інформації рівня 3 (IP)** для створення потоків. Усі хости спілкуватимуться один з одним, а також ми надамо пріоритет даним, що надходять від h3, використовуючи DSCP (Differentiated Services Code Point), тобто використовуючи DiffServ.

DSCP – це 6-бітне поле у заголовку IP-пакета, яке використовується для класифікації та пріоритезації мережевого трафіку, забезпечуючи якість обслуговування (QoS), щоб маршрутизатори та комутатори могли обробляти чутливі до затримки дані (голос, відео) краще, ніж звичайний трафік. Простіше кажучи, це «мітка», яка вказує мережевим пристроям, наскільки важливий пакет, щоб вони могли застосовувати до нього відповідну політику, наприклад, відправляти його у пріоритетну чергу.

*DiffServ* (диференційовані послуги) – це архітектура комп'ютерних мереж, яка визначає простий механізм класифікації мережевого трафіку та забезпечення якості обслуговування (QoS) в IP-мережах. Він базується на використанні поля TOS (тип послуги) IP-пакетів.




Отже, спочатку ми починаємо з визначення потоків:

```
mininet> sh ovs-ofctl add-flow s1
priority=500,dl_type=0x800,nw_src=10.0.0.0/24,nw_dst=10.0.0.3/24,actions=normal
mininet> sh ovs-ofctl add-flow s1
priority=800,dl_type=0x800,nw_src=10.0.0.3,nw_dst=10.0.0.0/24,actions=mod_nw_tos:18
4,normal
```

Ми використовуємо значення $184_{10}$, оскільки, щоб вказати значення $46_{10}$ у полі IP TOS, нам потрібно зсунути його на 2 біти ліворуч відповідно до значення бітів поля TOS*.

* Прискорене пересилання EF (Expedited Forwarding) у DSCP (TOS) у двійковому вигляді це 101110, десяткове значення 46. Мета прискореного пересилання EF – помістити пакети в пріоритетну чергу, де вони будуть мати найвищий пріоритет, що гарантує мінімальну затримку та відсунтість втрат пакетів. Якщо в пріоритетній черзі з є пакети, вони будуть відправлені раніше за всі інші черги.

Тепер нам потрібно знову ввімкнути ARP. Ми зробимо це дещо іншим способом:

```
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.1,actions=output:1
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.2,actions=output:2
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.3,actions=output:3
```

Перевіримо роботу мережі за допомогою команди *pingall*.

```
mininet> sh ovs-ovctl add s1 priority=500,dl_type=0x800,nw_src=10.0.0.0/24,nw_dst=10.0.0.3/24,action
=normal
/bin/sh: 1: ovs-ovctl: not found
mininet> sh ovs-ofctl add s1 priority=500,dl_type=0x800,nw_src=10.0.0.0/24,nw_dst=10.0.0.3/24,action
=normal
ovs-ofctl: unknown command 'add'; use --help for help
mininet> sh ovs-ofctl add-flow s1 priority=500,dl_type=0x800,nw_src=10.0.0.0/24,nw_dst=10.0.0.3/24,a
ction=normal
mininet> sh ovs-ofctl add-flow s1 priority=800,dl_type=0x800,nw_src=10.0.0.3/24,nw_dst=10.0.0.0/24,a
ction=mod_nw_tos:184,normal
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.1,action=output:1
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.2,action=output:2
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.3,action=output:3
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> h3 tcpdump -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Тепер перевіримо, чи з'являються біти DSCP у IP-заголовку пакету ICMP. Виконаємо команду *pingall*. Перевіряємо за допомогою Wireshark, чи пакети від h3 дійсно змінені.

Захоплюємо трафік на інтерфейсі s1-eth1. Отримуємо значення поля EF у DSCP у $b8_{16,}$ або $184_{10.}$

Для того, щоб Wireshak використовував окреме вікно, з консолі віртуальної машини запускаємо утиліту **startx**:

```
$ startx
```

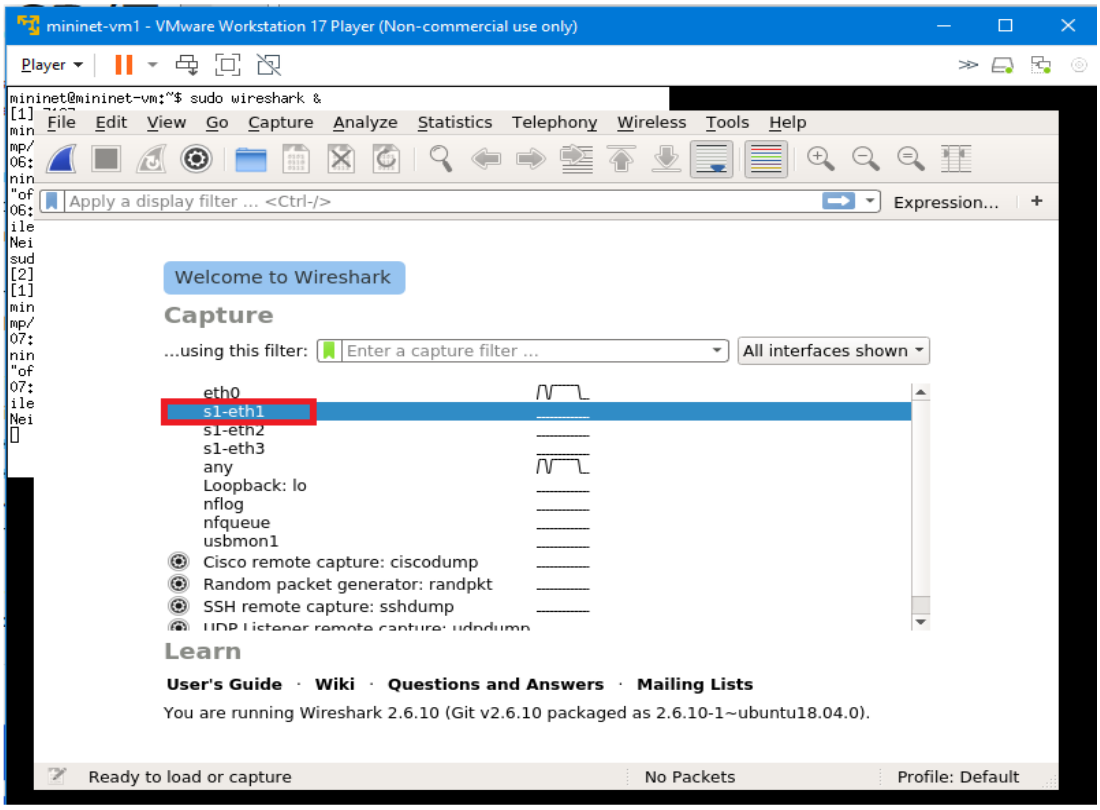
Утиліта *startx* – це сценарій, який використовується в операційних системах на основі Linux та інших систем, подібних до Unix, для запуску середовища робочого столу (X-Server) з командного рядка для ініціалізації графічного середовища для користувача.

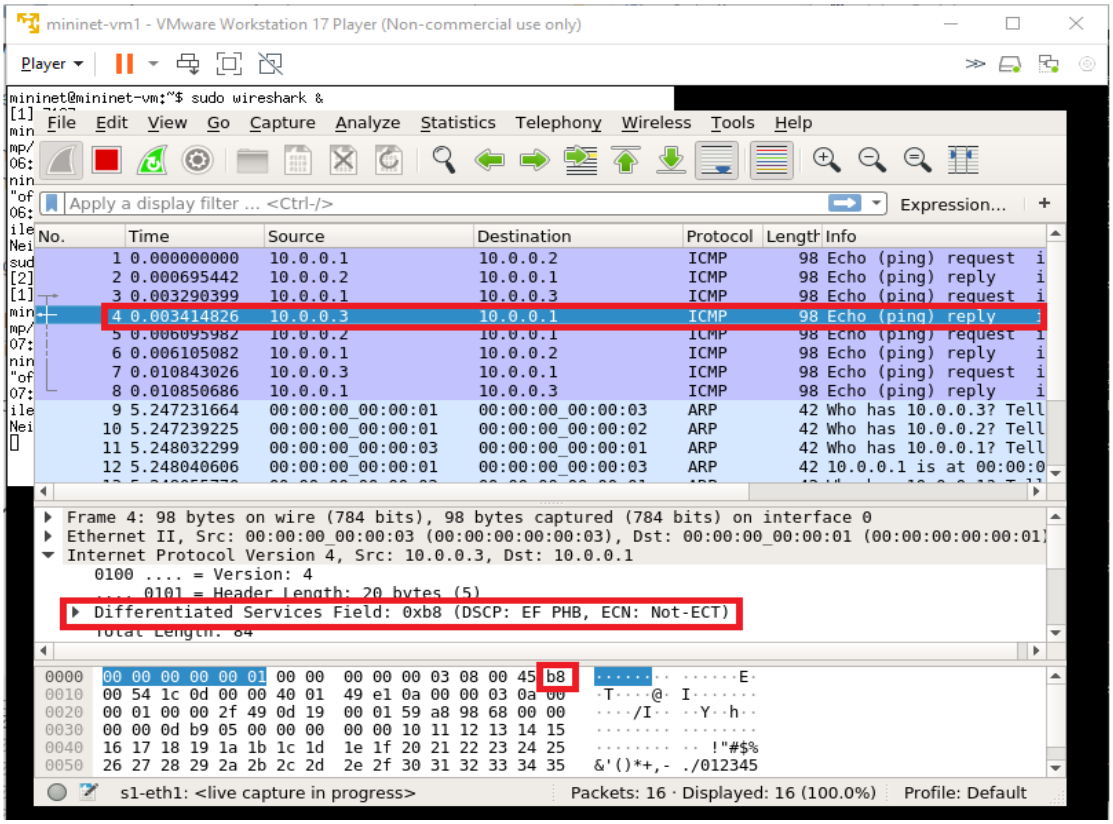
Якщо при спробі запустити графічне середовище з командного рядка з'являється помилка «команда startx не знайдена», то потрібно додатково встановити пакет **xinit**:

```
$ sudo apt update
$ sudo apt install xinit
```

Після встановлення xinit знову запускаємо *startx*. З'являється нове вікно. У цьому вікні запускаємо Wireshark у фоновому режимі.

```
$ sudo wireshark &
```




Бачимо, що, як і очікувалось, у IP-заголовку пакета з хоста h3 значення поля EF у DSCP дорівнює $b8_{16}$, або $184_{10}$.

Тепер видаляємо усі потоки в комутаторі:

```
mininet> sh ovs-ofctl del-flows s1
```

Завершуємо роботу Mininet і видаляємо виконані раніше налаштування:

```
mininet> exit
```

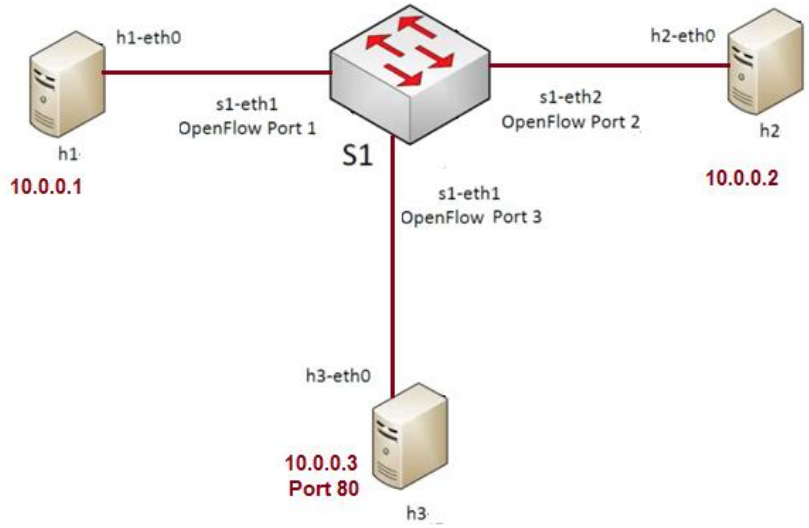
$ **mn -c**

Знову створюємо мережу:

$ **sudo mn --topo=single,3 --controller=none --mac**

## 20. **Використання даних рівня 4.**

На завершення продемонструємо, як теж саме можна зробити на рівні 4, рівні додатків. У цьому прикладі простий вебсервер Python буде запущений на хості h3, а хост h1 підключатиметься до цього сервера використовуючи порт 80.




Отже, запускаємо вебсервер на h3:

mininet> **h3 python2 -m SimpleHTTPServer 80 &**

Додаємо потік для ARP:

mininet> **sh ovs-ofctl add-flow s1 arp,actions=normal**

Додаємо правило, яке пересилає весь TCP-трафік (nw_proto=6) з портом призначення 80 на порт комутатора 3 (дозволяємо використовувати порт 80):

mininet> **sh ovs-ofctl add-flow s1**
**priority=500,dl_type=0x800,nw_proto=6,tp_dst=80,actions=output:3**

Це правило можна використовувати для перенаправлення всього трафіку даних на брандмауер, підключений до певного порту.

І, нарешті, додаємо таке правило:

mininet> **sh ovs-ofctl add-flow s1 priority=800,ip,nw_src=10.0.0.3,actions=normal**

Тут

**priority=800**: призначає рівень пріоритету запису потоку. Вищі значення пріоритету означають, що потік буде зіставлено раніше за потоки з нижчим пріоритетом, якщо пакету потенційно може відповідати кілька потоків.

**ip**: вказує, що запис потоку застосовується до IP-пакетів.

**nw_src=10.0.0.3**: поле збігу вказує IP-адресу джерела пакетів, до яких має застосовуватися цей запис потоку. Цьому потоку відповідатимуть лише пакети, що надходять з цієї IP-адреси – 10.0.0.3

**actions=normal**: визначає дію, яку потрібно виконати, коли пакет відповідає цьому запису потоку. Означає, що комутатор повинен обробляти пакет, використовуючи традиційну поведінку комутації Ethernet, включаючи вивчення MAC-адреси та пересилання на основі вивчених MAC-адрес.

Щоб перевірити, чи все працює як слід, виконуємо запит з хоста h1 до вебсервера на хості h3:

mininet> **h1 curl h3**

Якщо після цієї команди з'являється повідомлення про помилку

bash: curl: command not found

потрібно перевірити, чи встановлено пакет *curl*. Виконуємо таку команду:

$ **apt-cache policy curl**

Якщо curl в системі встановлений, маємо отримати подібний результат:

```
mininet@mininet-vm:~$ apt-cache policy curl
curl:
  Installed: 7.58.0-2ubuntu3.24
  Candidate: 7.58.0-2ubuntu3.24
  Version table:
 *** 7.58.0-2ubuntu3.24 500
        500 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages
        500 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages
        100 /var/lib/dpkg/status
     7.58.0-2ubuntu3 500
        500 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 Packages
mininet@mininet-vm:~$
```

Якщо *curl* в системі не встановлений, його потрібно встановити. Для цього з консолі віртуальної машини виконуємо такі команди:

$ **sudo apt update**
$ **sudo apt install curl**

Після встановлення curl знову виконуємо команду

mininet> **h1 curl h3**

Маємо отримати від вебсервера список каталогів, що містить імена файлів, розміри файлів тощо.

```
mininet> h3 python2 -m SimpleHTTPServer 80 &
mininet> sh ovs-ofctl add-flow s1 arp,actions=normal
mininet> sh ovs-ofctl add-flow s1 priority=500,dl_type=0x800,nw_proto=6,tp_dst=80,actions=output:3
mininet> sh ovs-ofctl add-flow s1 priority=800,ip,nw_src=10.0.0.3,actions=normal
mininet> h1 curl h3
bash: curl: command not found
mininet> h1 curl h3
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".gitconfig">.gitconfig</a>
<li><a href=".gnupg/">.gnupg/</a>
<li><a href=".mininet_history">.mininet_history</a>
<li><a href=".profile">.profile</a>
<li><a href=".ssh/">.ssh/</a>
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a>
<li><a href=".wget-hsts">.wget-hsts</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href=".Xauthority">.Xauthority</a>
<li><a href="mininet/">mininet/</a>
<li><a href="oflops/">oflops/</a>
<li><a href="oftest/">oftest/</a>
<li><a href="openflow/">openflow/</a>
<li><a href="pox/">pox/</a>
<li><a href="Work/">Work/</a>
</ul>
<hr>
</body>
</html>
mininet> h1 curl h2
^C
mininet> _
```

## Контрольні запитання

1. Технологія програмно-керованих мереж SDN. Призначення, переваги та недоліки.
2. Дати визначення поняттю емуляції. Приклади засобів емуляції SDN-мереж.
3. Емулятор Mininet. Призначення, особливості.
4. Інсталяція та налаштування Mininet. Етапи виконання.
5. Кроки створення SDN-мережі з мінімальною топологією (за замовчуванням) – один комутатор та два хости.
6. Команди взаємодії з хостами та комутаторами.
7. Команди перевірки з’єднаннь між хостами.
8. Команди запуску вебсервера та відповідного клієнта.
9. Засоби зміни конфігурації мережі.

# Лабораторна робота 4

## Моделювання програмно-керованої мережі у симуляторі Mininet. Використання графічного редактора MiniEdit.

**Мета:** Ознайомитися з графічним редактором MiniEdit мережевого емулятора Mininet, отримати практичні навички в створенні мережевих топологій для дослідження роботи програмно-керованої мережі.

### План роботи:

1. Ознайомлення з інструментами MiniEdit.
2. Налаштування параметрів MiniEdit.
3. Створення топології і налаштування мережевих пристроїв.
4. Створення таблиць потоків комутатора.
5. Ручне керування потоками. Налаштування пріоритету потоків.

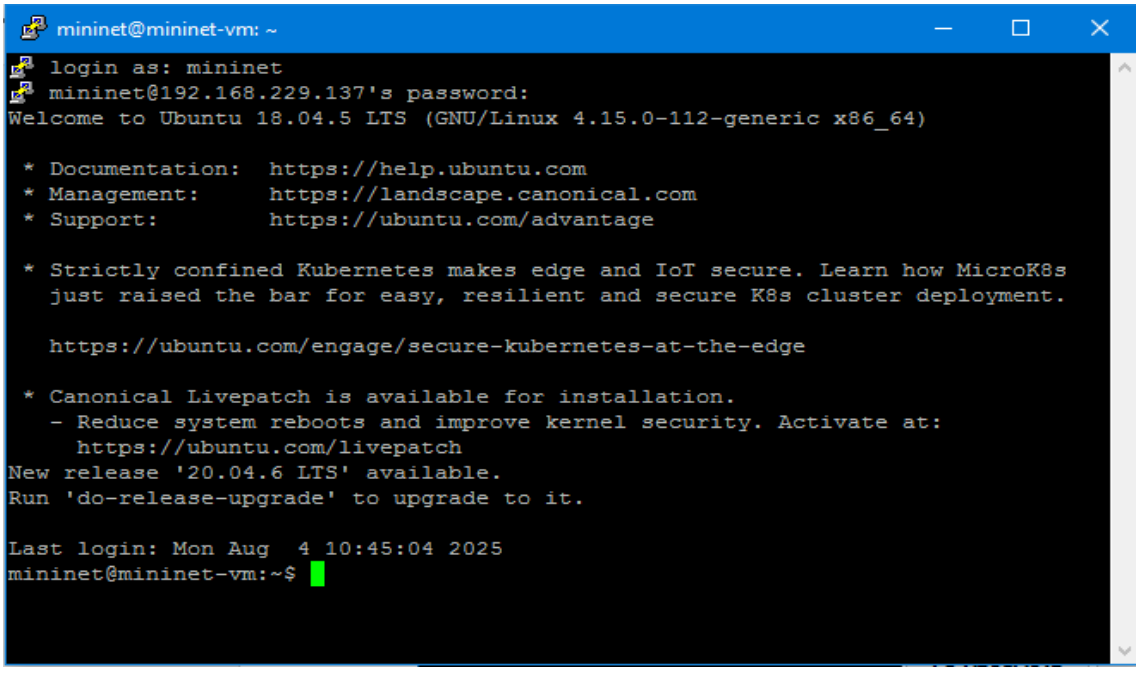
## 1. Теоретичні відомості

MiniEdit – це простий графічний редактор для Mininet, який використовується для експериментів та демонстрації можливостей розширення Mininet. Він має простий інтерфейс з полотном та рядком меню. На полотні розміщені піктограми інструментів зліва, а рядок меню знаходиться у верхній частині вікна.

MiniEdit – це частина Mininet, яка дозволяє візуалізувати та редагувати топології мережі. За допомогою MiniEdit можна створювати, видаляти та з'єднувати вузли мережі, а також налаштовувати їхні параметри.

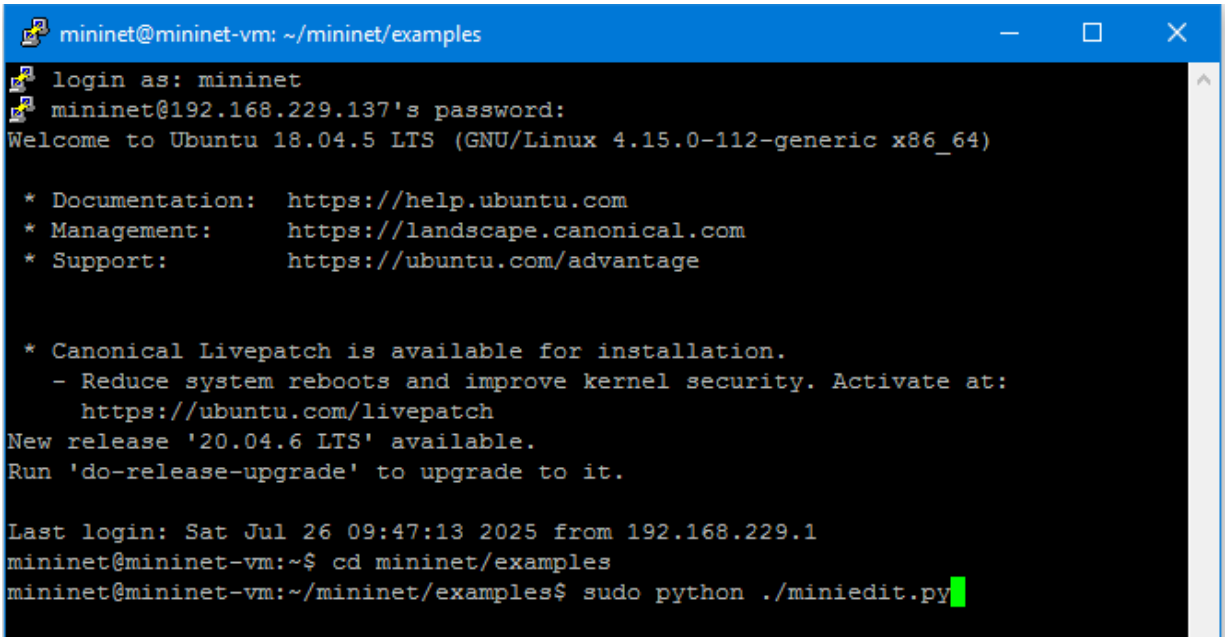
## 2. Виконання роботи

Виконуємо підключення до віртуальної машини Mininet за допомогою Putty, якщо цього не було зроблено раніше. У вікні, що відкрилося, вводимо: login *mininet*, password *mininet*.

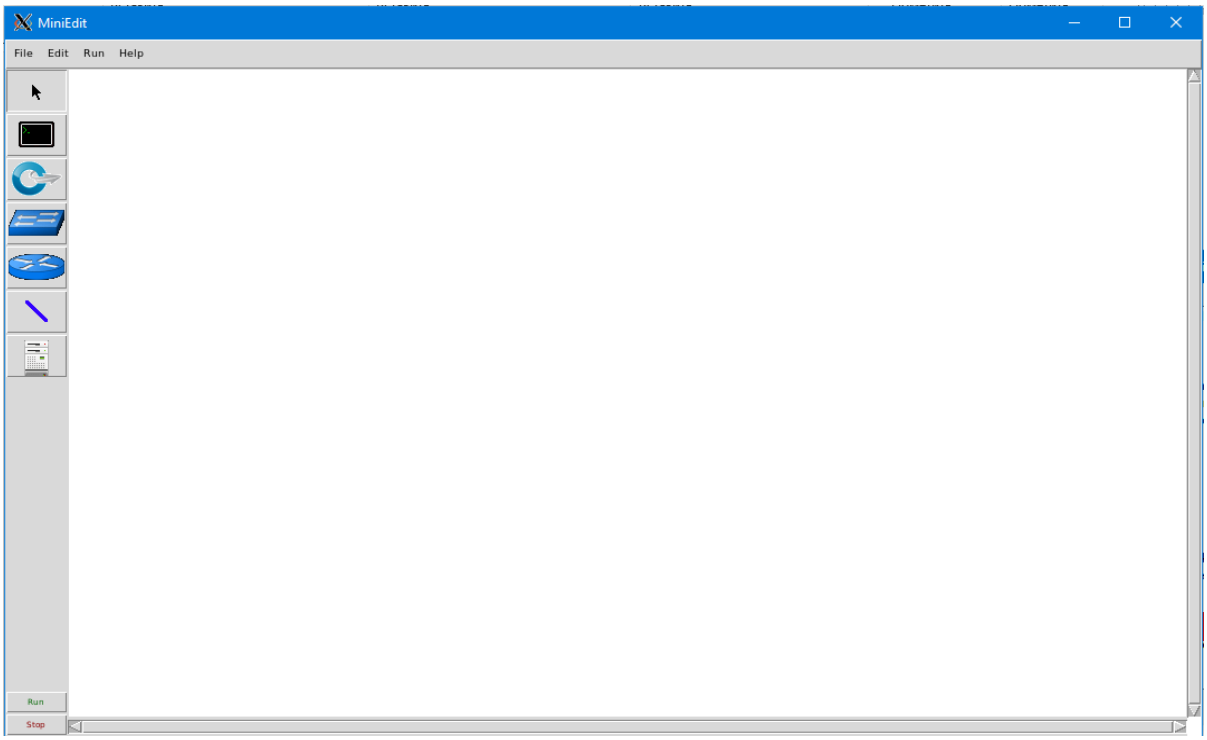


Переходимо до каталогу, у якому зберігається скрипт запуску MiniEdit. Запуск MiniEdit, виконуємо за допомогою команди *sudo*, щоб запустити з правами користувача *root*.

```
$ cd mininet/examples
$ sudo python ./miniedit.py
```




Після успішного запуску графічного редактора має відкритися вікно MiniEdit.




Mininet не має динамічного графічного інтерфейсу для візуалізації, але він тісно інтегрується з іншими інструментами, як-от Wireshark (для аналізу пакетів) та matplotlib (для побудови графіків), щоб забезпечити візуалізацію та моніторинг мережевого трафіку та стану мережі, що дозволяє отримати динамічне оновлення через сторонні інструменти.

Емулятор мережі Mininet містить MiniEdit, простий редактор графічного інтерфейсу користувача (GUI) для Mininet. MiniEdit призначений для графічного проєктування топології програмно-керованих мереж (SDN).
Він дозволяє користувачам:

- **Створювати та редагувати топології:** можна перетягувати хости, комутатори та контролери, а також з'єднувати їх для створення мережевих конфігурацій.
- **Налаштовувати мережеві елементи:** встановлювати парамети, такі як IP-адреси для хостів, та визначати деталі контролера.
- **Зберігати та завантажувати топології:** можна зберігати створені конфігурації мережі для подальшого використання та завантажувати існуючі.
- **Запускати емуляції:** можна запускати емуляцію Mininet безпосередньо з графічного інтерфейсу та взаємодіяти з емульованою мережею, зокрема відкривати термінали для хостів для виконання команд, таких як *ping*, *tcpdump* тощо.

## Інструменти в MiniEdit

MiniEdit має простий інтерфейс користувача, який являє собою полотно з рядком значків інструментів у лівій частині вікна та рядком меню у верхній частині вікна. Перелік інструментів, доступних у MiniEdit для створення мережевих моделей на основі програмно-керованих мереж:

**Select** – цей інструмент використовується для переміщення вузлів на полотні, а потім для перетягування будь-якого існуючого вузла.

**Host** – цей інструмент використовується для створення вузлів на полотні, які виконуватимуть функцію хост-комп'ютерів.

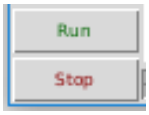
**Switch** – цей інструмент працює так само, як і інструмент Host, описаний вище. Очікується, що ці комутатори будуть підключені до контролера.

**Legacy Switch** – цей інструмент створює навчальний комутатор Ethernet із налаштуваннями за замовчуванням.

**Legacy Router** – цей інструмент створює базовий маршрутизатор, який працюватиме незалежно, без контролера.

**NetLink** – цей інструмент створює з'єднання між вузлами на полотні.

**Controller** – цей інструмент створює контролер. Можна додати кілька контролерів.

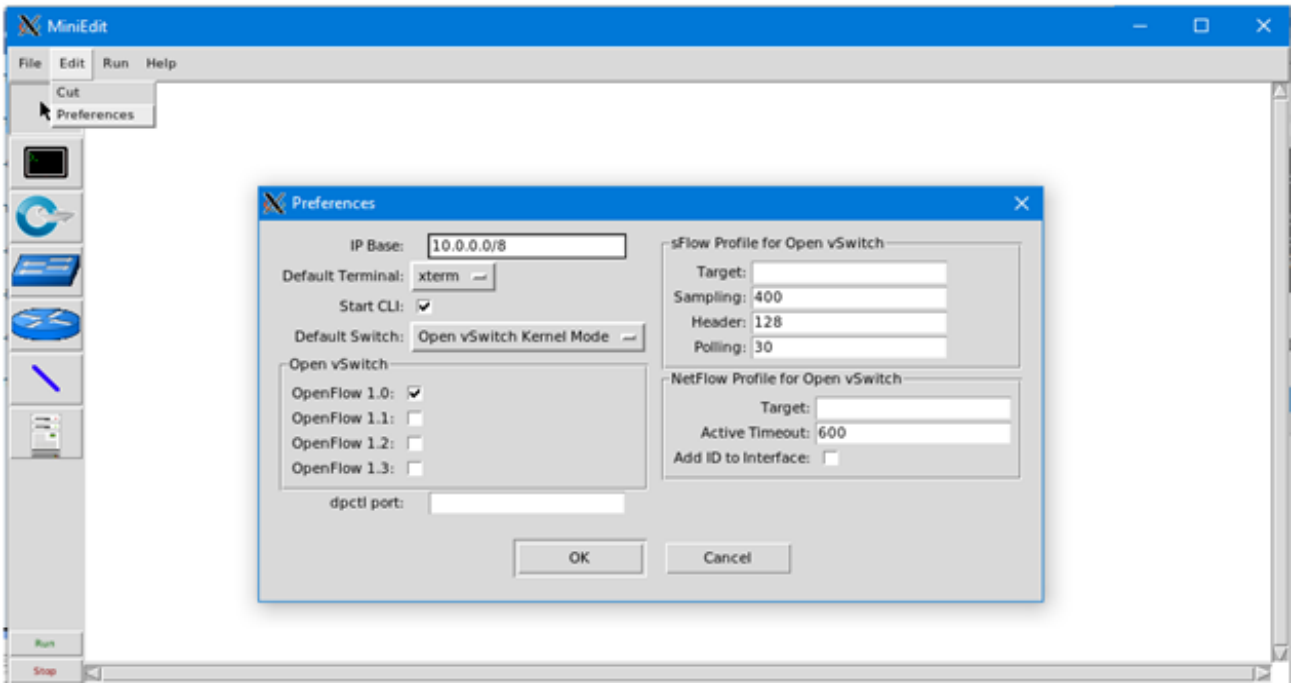



Ці кнопки запускають і зупиняють сценарій симуляції Mininet, який наразі відображається на полотні MiniEdit.

## Налаштування параметрів MiniEdit

Щоб налаштувати параметри MiniEdit, скористаємося командою меню MiniEdit: ***Edit*** → ***Preferences***. У діалоговому вікні, що з'явиться, вносимо потрібні зміни:

- Встановлюємо параметр **Start CLI** (*запустити командний рядок)*. За замовчуванням вікно консолі MiniEdit не надає користувачеві доступу до інтерфейсу командного рядка Mininet. Якщо є потреба використовувати інтерфейс командного рядка Mininet під час запуску симуляції, ставимо позначку в полі *Start CLI*. Також можна встановити версію OpenFlow, яка буде використовуватися.




## Створення власної топології мережі за допомогою MiniEdit

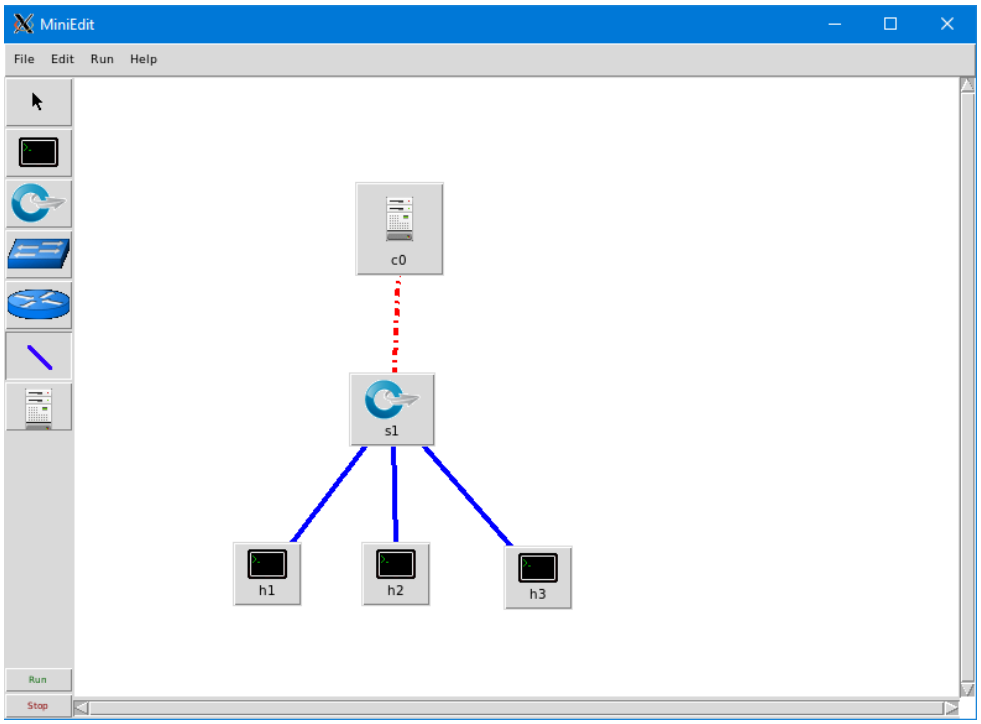
Додавання хостів, комутаторів та контролерів.
Спочатку додамо кілька хостів до мережевого сценарію. Клікаємо по значку *хоста*, потім переміщаємо вказівник у місце на полотні MiniEdit, де потрібно розмістити хост, а потім клікаємо ще раз. На полотні з'являється значок хоста.

Поки інструмент *Host* активний, можна додавати більше хостів. Продовжуємо натискати в кожному місці на полотні, де має бути розміщений хост. У цьому прикладі ми додаємо три хости.

Додаємо комутатор і контролер, використовуючи той самий метод: натискаємо на інструмент *Switch* і додаємо комутатор, потім натискаємо на інструмент *Controller* і додаємо контролер.

Далі додаємо з'єднання між вузлами на полотні. Натискаємо на інструмент *NetLink* , потім клікаємо на вузлі та перетягуємо з'єднання на інший вузол. Наприклад: підключаємо хост до комутатора або комутатор до контролера.

Завершена топологія мережі має бути схожою на показану на скріншоті нижче:



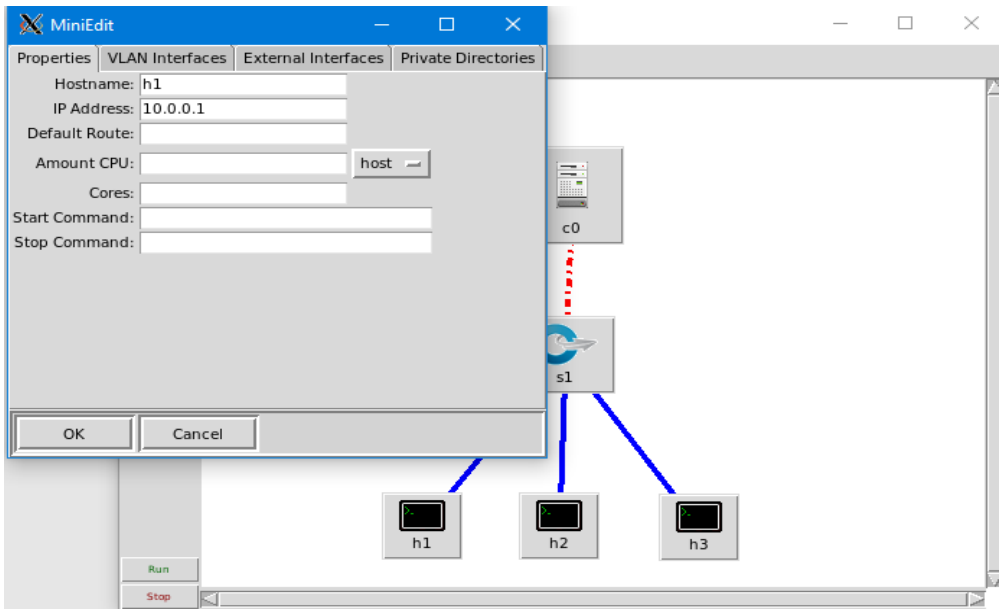

**Іменування в Mininet**

Для ефективного використання Mininet важливо розуміти схему іменування хостів, комутаторів та інтерфейсів. Зазвичай хости називаються **h1...hn** , а комутатори - **s1...sn** . Інтерфейси, що належать до вузла, іменуються, починаючи з імені вузла, наприклад, h1-eth0 – це інтерфейс за замовчуванням хоста h1, а s1-eth1 – це перший порт даних комутатора s1.

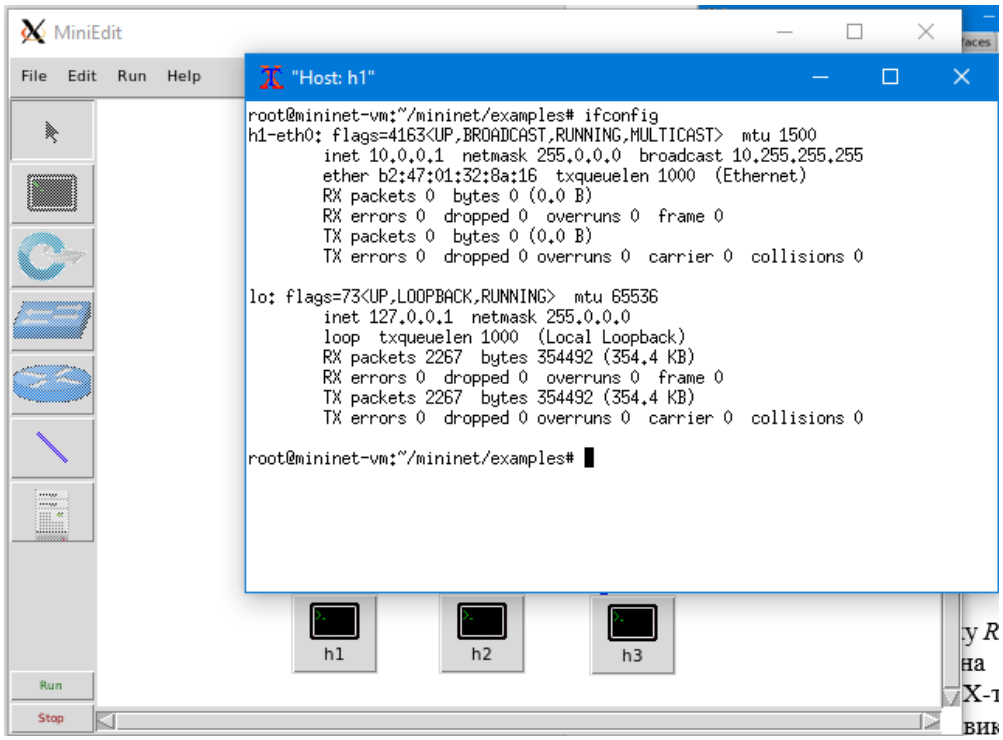
Інтерфейси хоста видно лише зсередини самого хоста, але порти комутатора видно в просторі імен "root" (їх можна побачити, ввівши *ifconfig* в додатковому вікні під час роботи Mininet).

Розглянемо приклад. Надамо IP-адресу хосту *h1*, запустимо на ньому X-термінал і перевіримо налаштування інтерфейсу:

- Клікаємо правою клавішею миші на хості і не відпускаючи клавіші вибираємо пункт *Properties*. У вікні, що з'явилося в полі *IP Address* записуємо **10.0.0.1**, натискаємо **OK**.



- Натискаємо кнопку *Run* в нижньому лівому кутку робочого полотна, щоб запустити сценарій MiniEdit. Клікаємо правою клавішею миші на хості і не відпускаючи клавіші вибираємо пункт *Terminal*. З'являється вікно X-термінала хоста h1.
- У вікні "Host: h1" виконуємо команду ifconfig, щоб дізнатися про інтерфейси хоста.

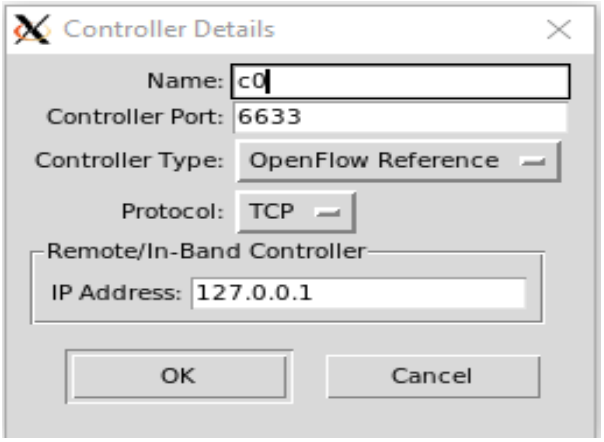



- Натискаємо кнопку *Stop* в нижньому лівому кутку робочого полотна, щоб зупинити сценарій MiniEdit.

**Налаштування контролера**

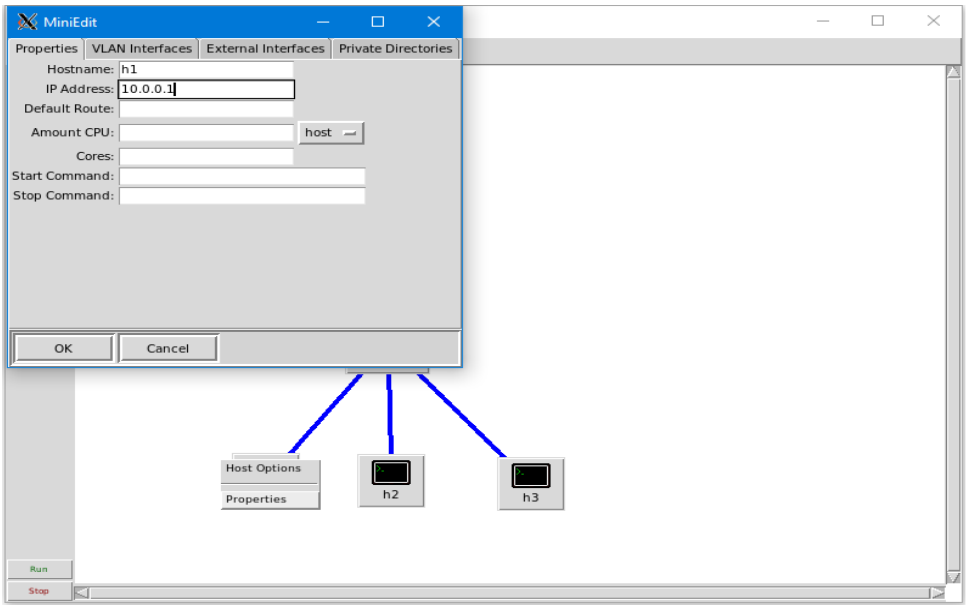
Відредагуємо параметри контролера, щоб він використовував унікальний номер порту.

Клікаємо правою кнопкою миші на контролері та вибираємо *Preferencies* в меню, що з'явиться.
Вибираємо тип контролера – стандартний контролер OpenFlow Reference, вбудований у Mininet. Номер порту за замовчуванням для контролера – 6633. Якщо у роботі застосовується кілька таких контролерів, потрібно номер порту змінювати так, щоб вони використовували окремі порти, наприклад, 6634, 6635 тощо.
.



### Налаштування хостів

Надамо IP-адреси хостам h1, h2 та h3, відповідно 10.0.0.1, 10.0.0.2 та 10.0.0.3. Для цього клікаємо правою клавішею миші на хості. У вікні, що випало, не відпускаючи правої клавіші миші, переміщаємо курсор на *Properties* і відпускаємо клавішу миші. Має відкритися вікно для налаштування хоста. У полі *IP Address* вказуємо відповідну IP-адресу. Натискаємо **OK**. Приклад – на рисунку.




### Збереження налаштувань

Параметри MiniEdit зберігаються у файлі топології MiniEdit для кожного сценарію, тому для кожного збереженого сценарію можуть бути різні налаштування.

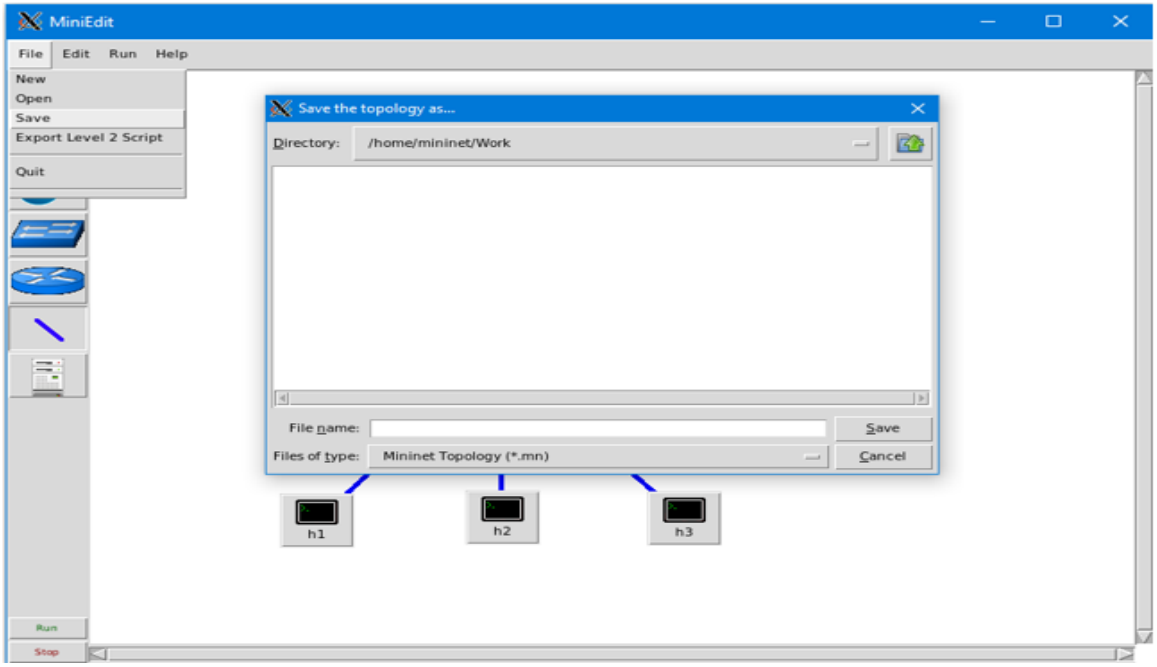
### Збереження конфігурації

Збережемо файл топології MiniEdit, щоб мати змогу завантажити цей сценарій у MiniEdit у майбутньому. Також експортуємо скрипт Mininet Python, який можна буде запустити у вікні терміналу для виконання сценарію.

### Зберігаємо файл топології

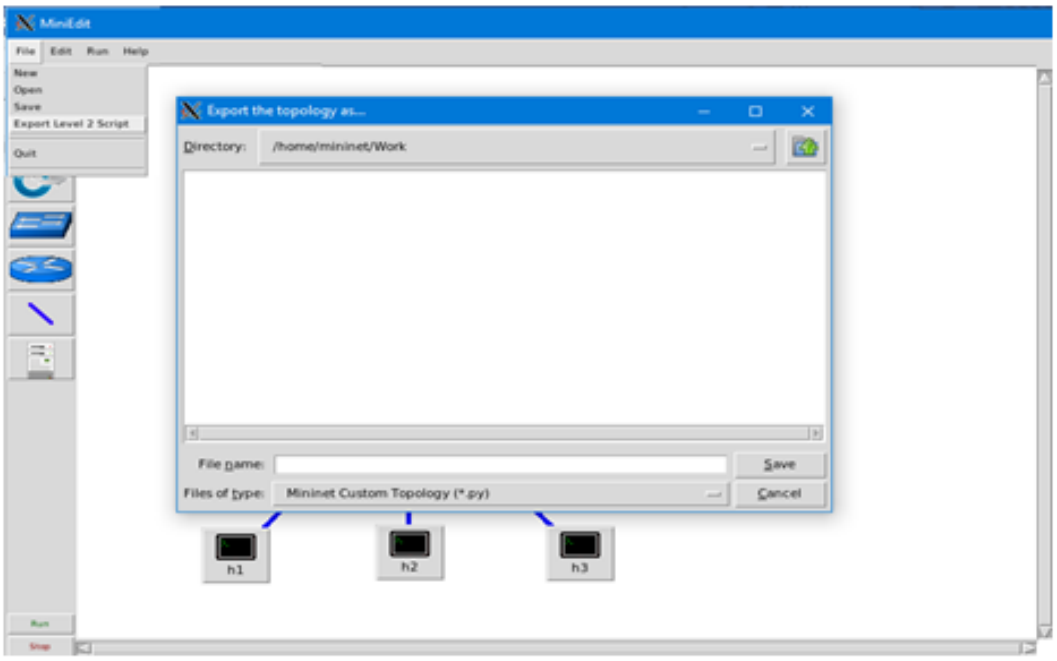
Для зберігання файлів топології і конфігурації рекомендується в каталозі home/mininet створити робочий каталог, наприклад Work. Це можна виконати, наприклад, за допомогою утиліти WinSCP.

Щоб зберегти файл топології Mininet (*.*mn*), натискаємо *File* у верхньому рядку меню та вибираємо *Save* у меню, що випадає. Вводимо ім'я файлу (наприклад, *mytest*) та зберігаємо його.

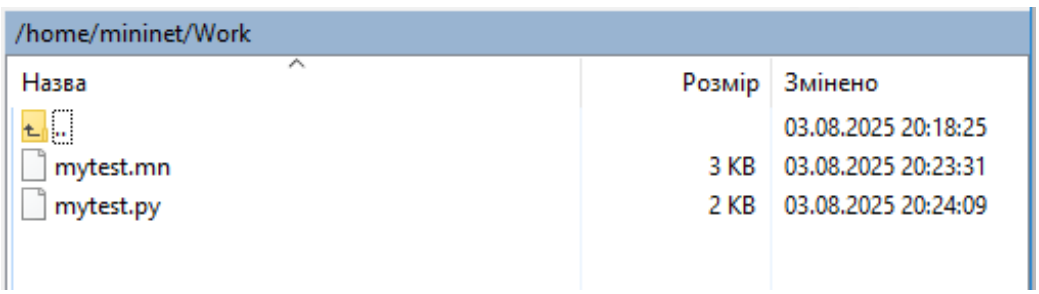



**Зберігаємо власний скрипт Mininet**

Щоб зберегти файл Mininet Custom Topology (**.py*), натискаємо *File* у верхньому рядку меню та вибираємо *Export Level 2 Script* у меню, що випадає. Вводимо ім'я файлу (наприклад, *mytest*) та зберігаємо його.




Маємо отримати збережені файли *mytest.mn* та *mytest.py*:



Альтернативою запуску симуляції безпосередньо в MiniEdit є запуск власного скрипта топології Mininet, створеного в MiniEdit. Це файл із розширенням *.py*, який ми раніше створили, коли використовували команду меню: File→ *Export Level 2 Script* .

Перевага запуску сценарію користувацької топології Mininet полягає в тому, що можна редагувати сценарій, спочатку створений у MiniEdit, для створення складніших сценаріїв та використання функцій Mininet, які не підтримуються MiniEdit.

Щоб запустити скрипт, створений раніше в MiniEdit, потрібно виконати такі дії:

```
$ sudo python2 mytest.py
```

Скрипт розгортає топологію мережі, з'являється командний рядок *mininet*>. Тепер можна перевірити сценарій за допомогою команди *ping* для перевірки з'єднань між хостами в мережі. Наприклад:

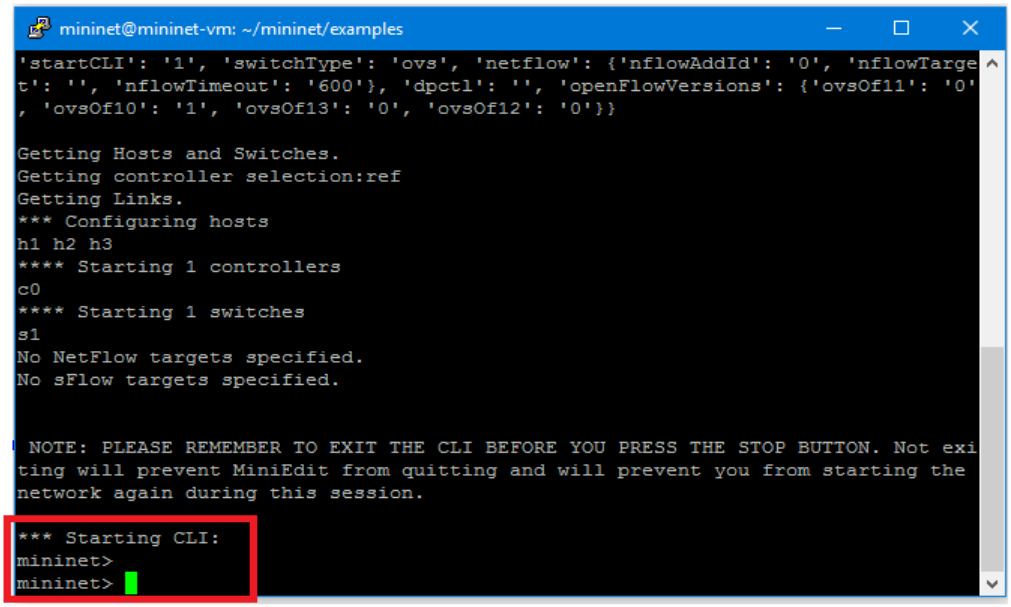
```
mininet> exit
mininet> net
```

Відкрийте файл mytest.py та опишіть розділи коду, які ви в ньому знайдете.

```
mininet@mininet-vm:~/Work$ python2 mytest.py
*** Mininet must run as root.
mininet@mininet-vm:~/Work$ sudo python2 mytest.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h2 h1 h3
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet> h1 ping -c 3 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=3.27 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.581 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.079 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.079/1.311/3.273/1.402 ms
mininet> net
h2 h2-eth0:s1-eth3
h1 h1-eth0:s1-eth2
h3 h3-eth0:s1-eth1
s1 lo:  s1-eth1:h3-eth0 s1-eth2:h1-eth0 s1-eth3:h2-eth0
c0
mininet>
```

### Запускаємо мережевий сценарій MiniEdit

Щоб запустити сценарій симуляції, натискаємо кнопку *Run* у графічному інтерфейсі MiniEdit. У вікні терміналу, з якого був запущений MiniEdit, бачимо кілька повідомлень, що показують хід запуску симуляції, а потім запрошення CLI (оскільки ми позначили поле *Start CLI* у вікні налаштувань MiniEdit).

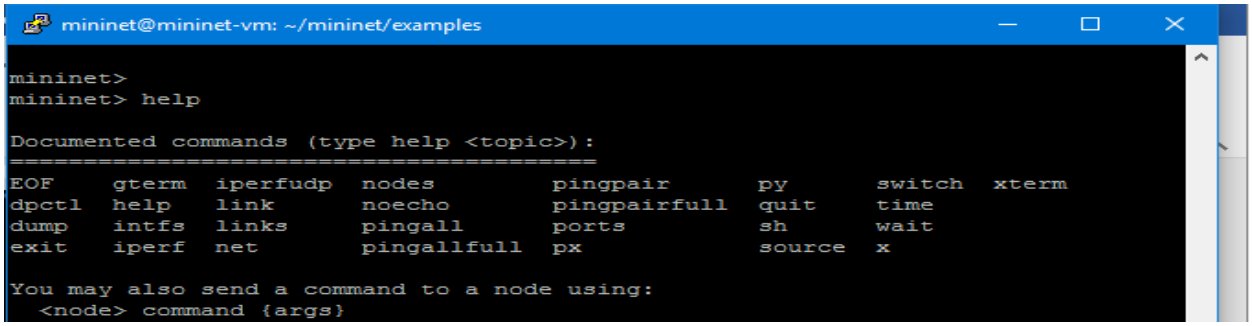



Консоль MiniEdit показує запрошення Mininet. Також можна побачити повідомлення про те, що таблиця, яка описує потоки даних через комутатор s1, не налаштована.
Зверніть увагу на попередження. Перш ніж зупинити симуляцію кнопкою *Stop*, потрібно вийти з інтерфейсу командного рядка, ввівши команду **Exit** в командному рядку *Mininet* у вікні консолі MiniEdit.

Щоб вийти з інтерфейсу командного рядка CLI Mininet:

mininet> **exit**

Запускаємо команду отримання довідки, щоб переглянути список доступних команд:




Розглянемо деякі команди командного рядка Mininet:

**net** – показує топологію та з'єднання між хостами та комутаторами в Mininet.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
mininet>
```

**links** – показує з'єднання між хостами та комутаторами, а також перевіряє їх.

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet>
mininet>
```

**nodes** – ця команда в Mininet відображує список усіх вузлів, таких як хости, комутатори, контролери тощо.

```
mininet>
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet>
mininet>
```

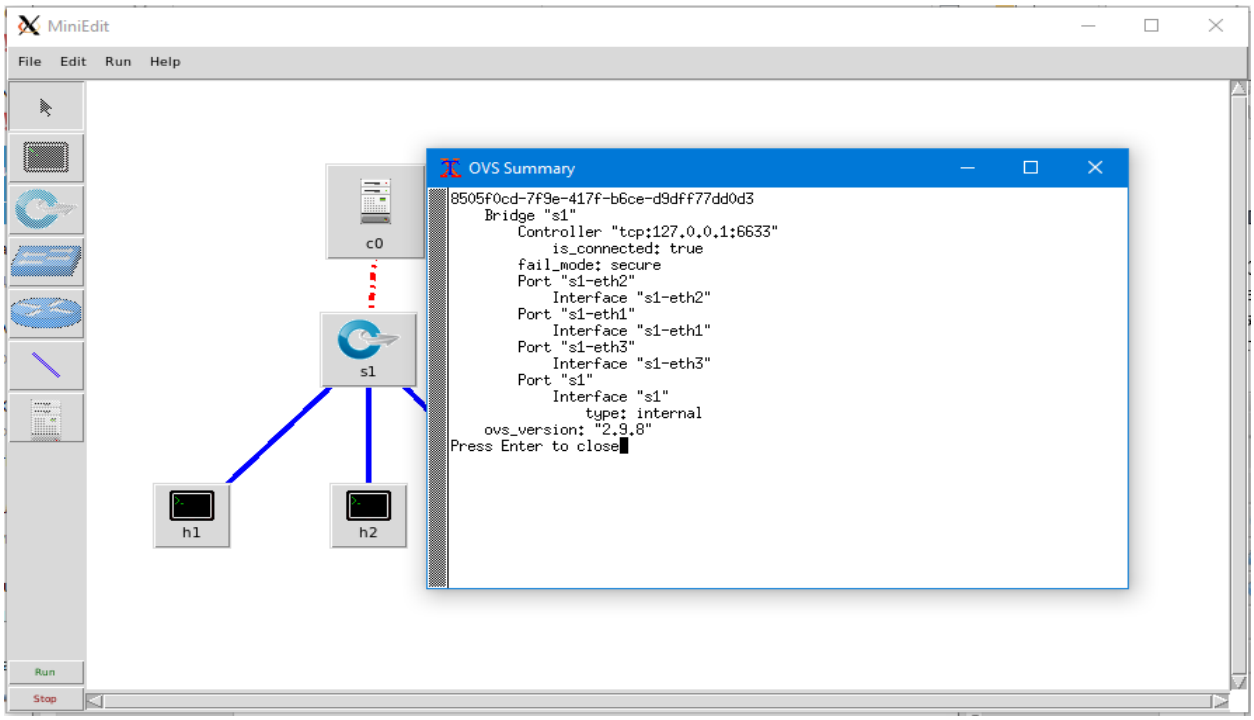
**dump**– виводить інформацію про всі вузли.

```
mininet> dump
<Host h3: h3-eth0:10.0.0.3 pid=2136>
<Host h2: h2-eth0:10.0.0.2 pid=2138>
<Host h1: h1-eth0:10.0.0.1 pid=2140>
<customOvs s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=2145>
<Controller c0: 127.0.0.1:6633 pid=2129>
```

Після запуску сценарію моделювання переглянемо стан різних елементів мережі, відкриємо вікна терміналів, запустимо мережевий трафік та запустимо програми на змодельованих хостах. Ці вправи продемонструють, як використовувати деякі функції MiniEdit.

### Переглянемо конфігурації Open vSwitch

- Спочатку перевіримо конфігурацію комутаторів у мережевому моделюванні, щоб переконатися, що все налаштовано правильно. Виконуємо команду меню MiniEdit: *Run → Show OVS Summary*, щоб переглянути список конфігурацій комутатора. У цьому випадку можна перевірити, чи прослуховує комутатор потрібний контролер на потрібному порту.



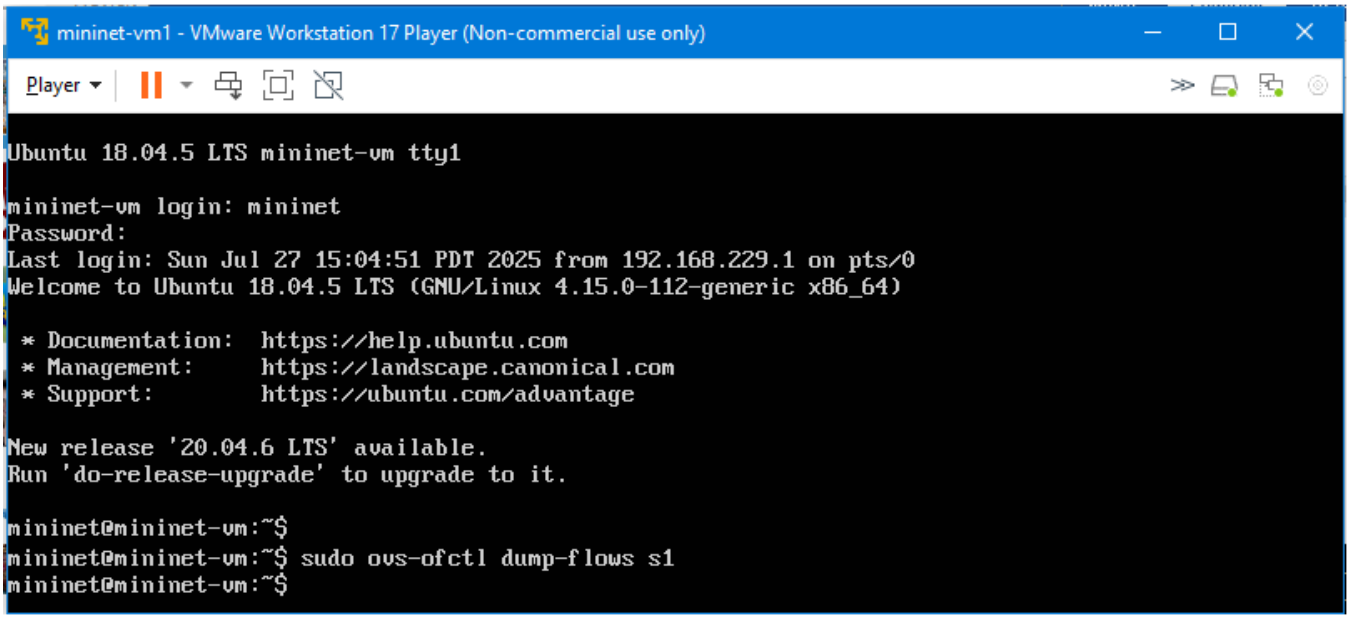

- Далі переглянемо таблицю потоків комутатора за допомогою команди **ovs-ofctl dump-flows s1**. Потрібно виконати цю команду на головному комп'ютері або на віртуальній машині, на якій запущено Mininet. Для цього потрібно використовувати вікно терміналу, підключене до комп'ютера (не до одного з вузлів у симуляції мережі). Можна

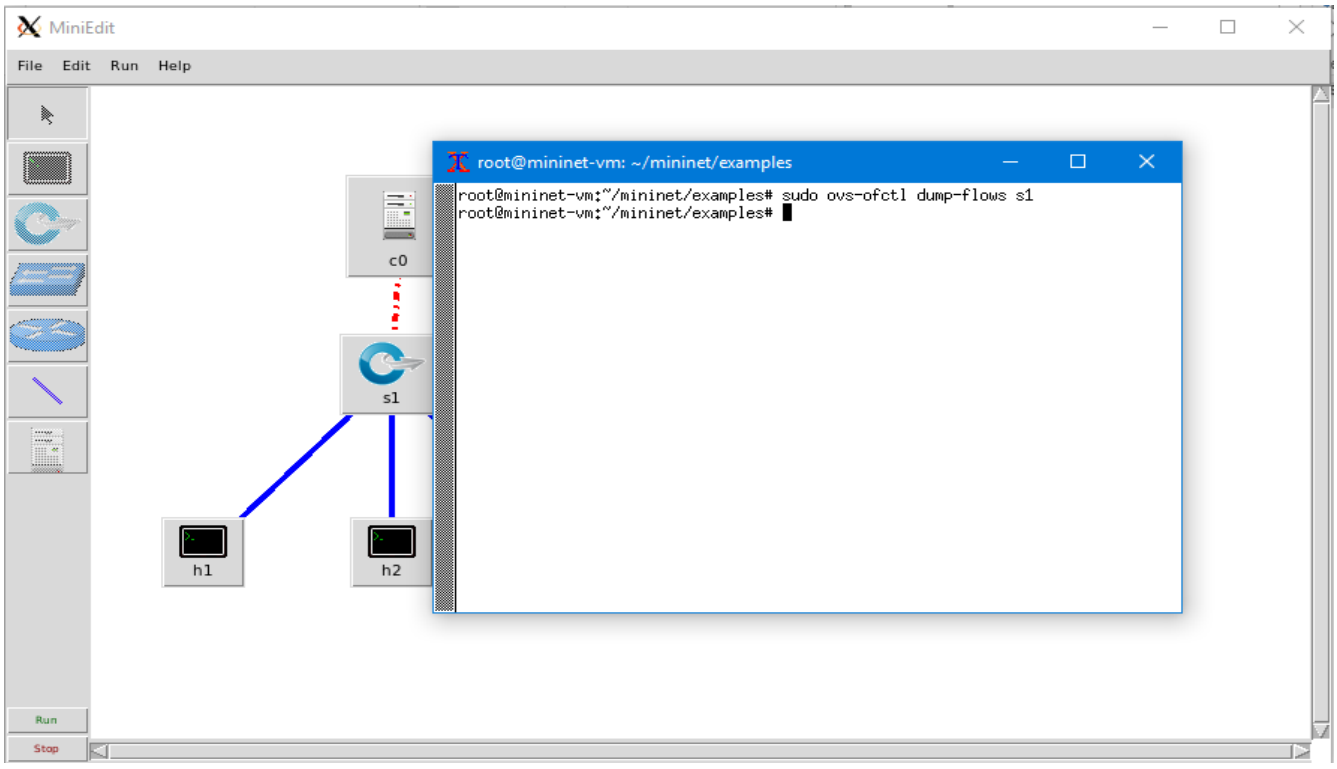
скористатися MiniEdit, щоб відкрити xterm, підключений до головного комп'ютера, за допомогою команди меню MiniEdit: *Run→ Root Terminal*.

Xterm – це стандартний емулятор терміналу для системи X-Window, що забезпечує інтерфейс командного рядка в графічному вікні. Він дозволяє користувачам запускати програми, що потребують оболонки або середовища командного рядка в Linux та інших Unix-подібних операційних системах.

Отже, двома способами перевіряємо таблицю потоків на комутаторі s1. Зараз вона має бути порожньою.

```
$ sudo ovs-ofctl dump-flows s1
```



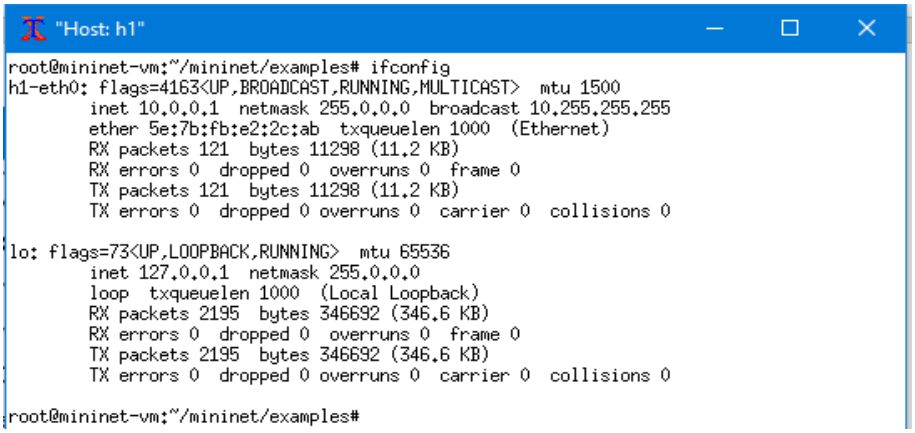



**Запустимо програми для генерування та моніторингу трафіку**

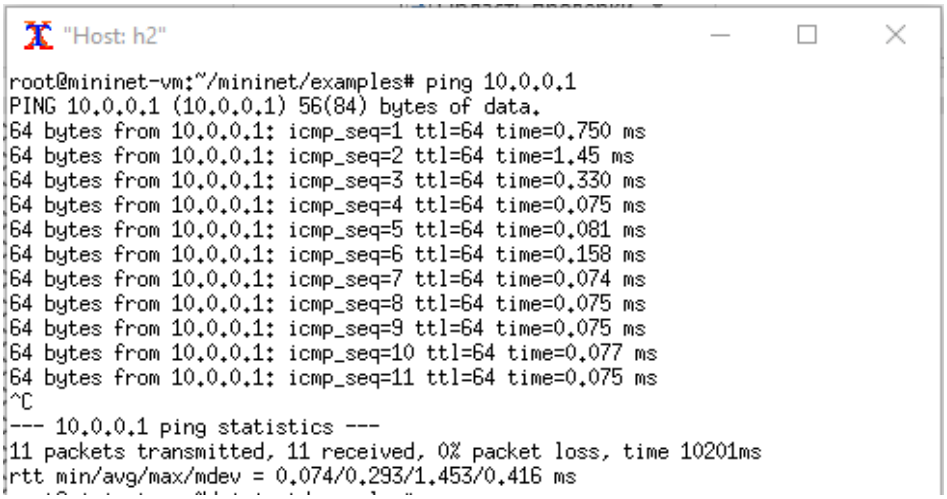
Відкриваємо вікна **xterm** на хостах *h1* та *h2*. Клікаємо правою кнопкою миші на кожному хості в графічному інтерфейсі MiniEdit та вибираємо *Terminal* у меню, що з'явиться:

- У вікні *h1* xterm виконуємо команду ifconfig, щоб дізнатися про інтерфейси хоста, а потім запускаємо утиліту tcpdump на працюючому інтерфейсі h1-eth0 . У вікні *h2* xterm виконуємо команду *ping* IP-адреси хоста h1 для створення трафіку між хостами *h1* та *h2*.

Зверніть увагу на IP-адресу та ім'я інтерфейсу h1-eth0. Вони додатково налаштовуються за допомогою параметрів Mininet.



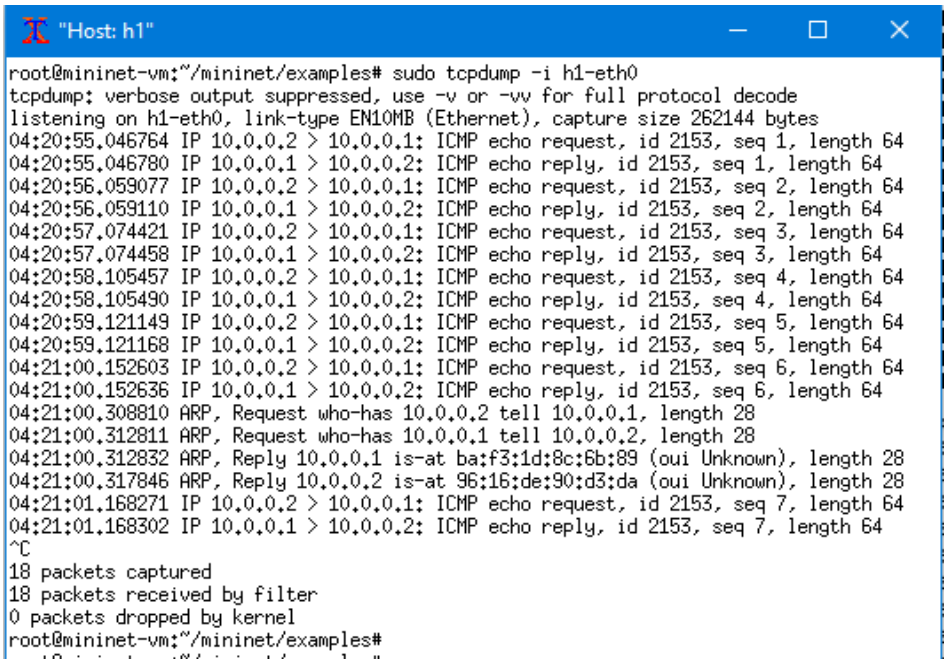

У вікні хоста h2 можна бачити результати виконання команди *ping*.



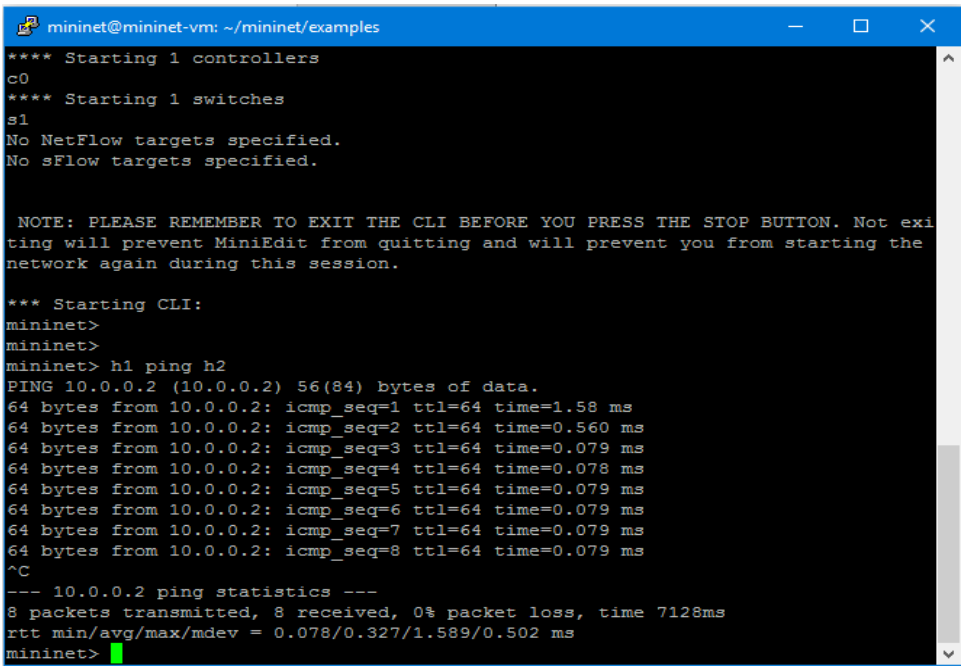

- Для перехоплення трафіку на хості h1 запускаємо утиліту *tcpdump*:

**$ sudo tcpdump -i h1-eth0** h1-eth0 – ім'я інтерфейсу

У вікні хоста h1, де запущений *tcpdump*, можна бачити успішно надіслані ICMP-пакети та отримані відповіді.




- З консолі MiniEdit виконуємо таку команду:
  mininet> **h1 ping h2**



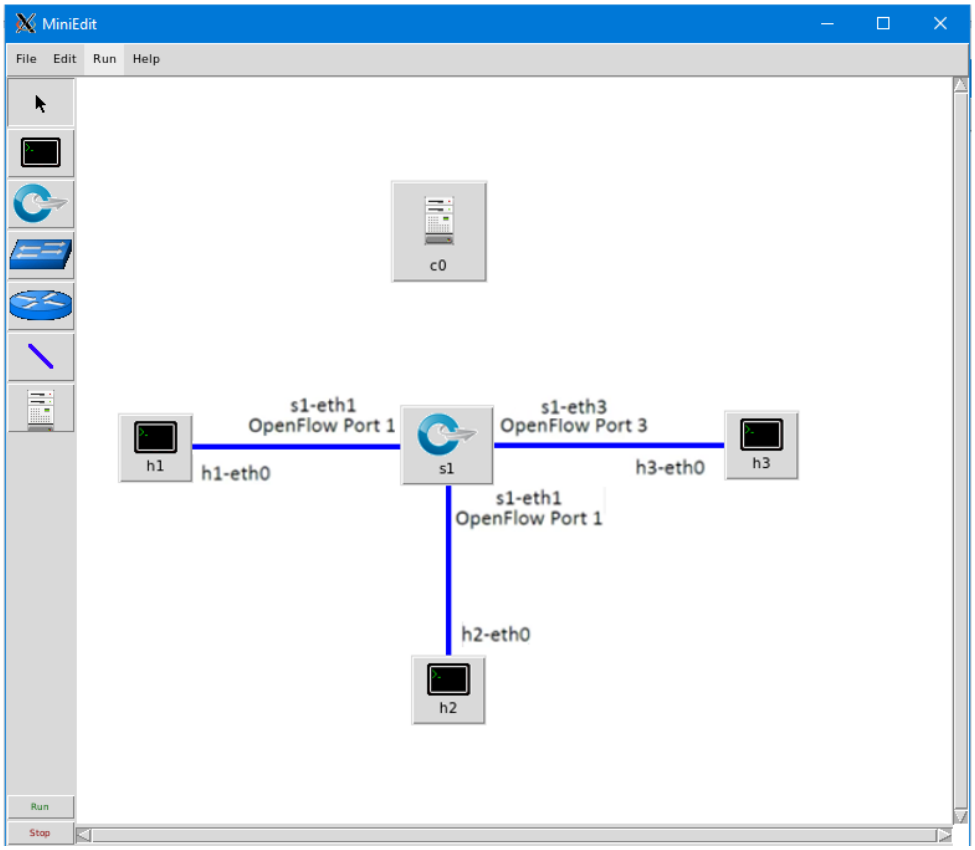

**Ручне керування потоками.**
**Переключення портів комутатора.**

У цьому розділі буде розглянуте керування записами потоку на Open vSwitch (OVS) уручну за допомогою команди *ovs-ofctl*. Зазвичай ці потоки (або правила) встановлюються динамічно за допомогою контролера SDN.

Контролер за замовчуванням змушує комутатор діяти як звичайний комутатор Ethernet. Комутатор у Mininet може працювати в автономному режимі. При отриманні пакета він

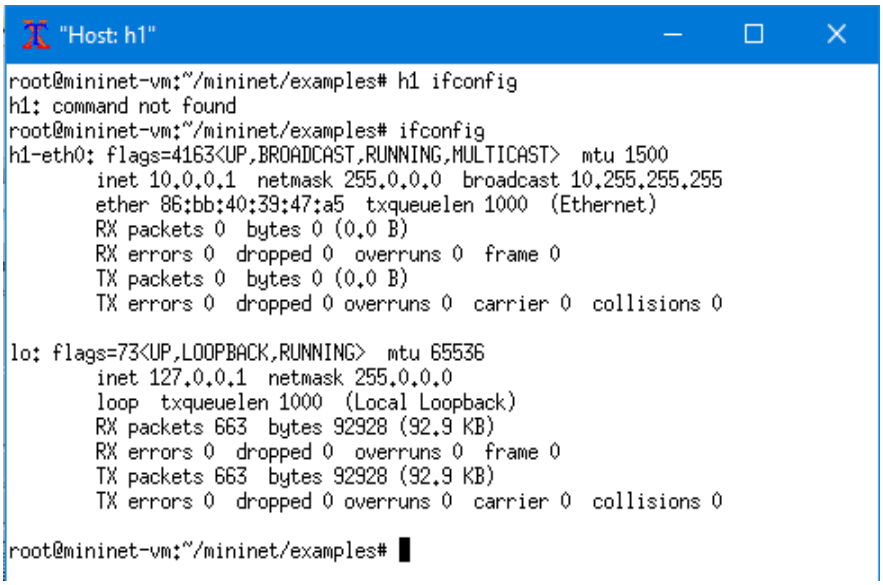
перевіряє, чи є MAC-адреса призначення у таблиці. Якщо є, то він пересилає пакет на відповідний порт. Якщо ні, він широкомовно розсилає пакет по всіх портах, крім вхідного. Внесемо зміни в топологію досліджуваної мережі – відключимо контролер мережі:

- Виходимо з інтерфейсу командного рядка, ввівши команду **Exit** вкомандному рядку *Mininet* у вікні консолі MiniEdit.
- Зупиняємо симуляцію кнопкою *Stop.*
- Від'єднуємо контролер від комутатора на топології. Для цього клікаємо лівою клавішею миші на лінку, що з'єднує контролер і комутатор, потім натискаємо *Edit* в меню і вибираємо *Cut*.

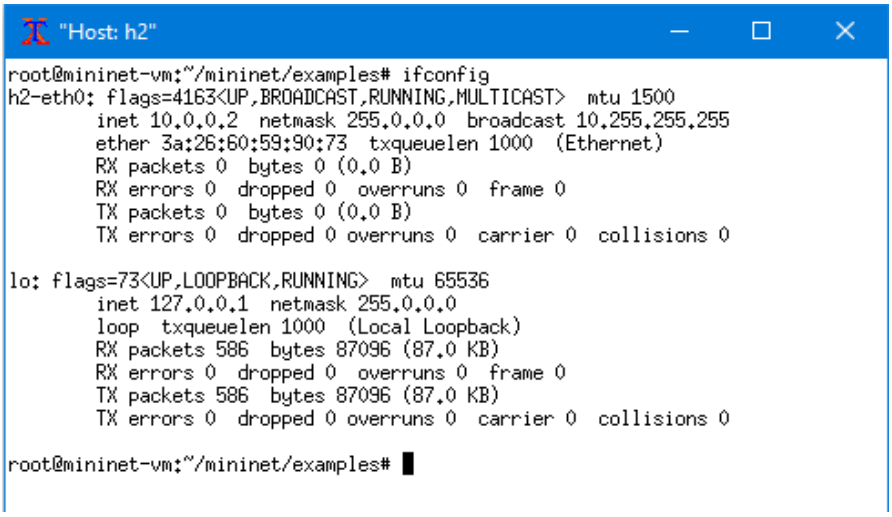



Тестування з'єднань.

- Перед тестуванням з'єднань між хостами h1, h2 і h3 необхідно запустити емуляцію.
- Натискаємо кнопку *Run*, щоб розпочати емуляцію. Починається емуляція і кнопки панелі MiniEdit стають сірими, що вказує на те, що вони зараз вимкнені.
- Відкриваємо термінал на хості h1, утримуючи праву кнопку миші на хості h1 та вибираючи *Terminal*. Це дозволяє виконувати команди на хості h1.
- Повторюємо процедуру на хості h2.
- На терміналі хоста h1 вводимо команду **ifconfig**, щоб відобразити надану йому IP-адресу. Інтерфейс h1-eth0 на хості h1 повинен бути налаштований з IP-адресою 10.0.0.1 та маскою підмережі 255.0.0.0.



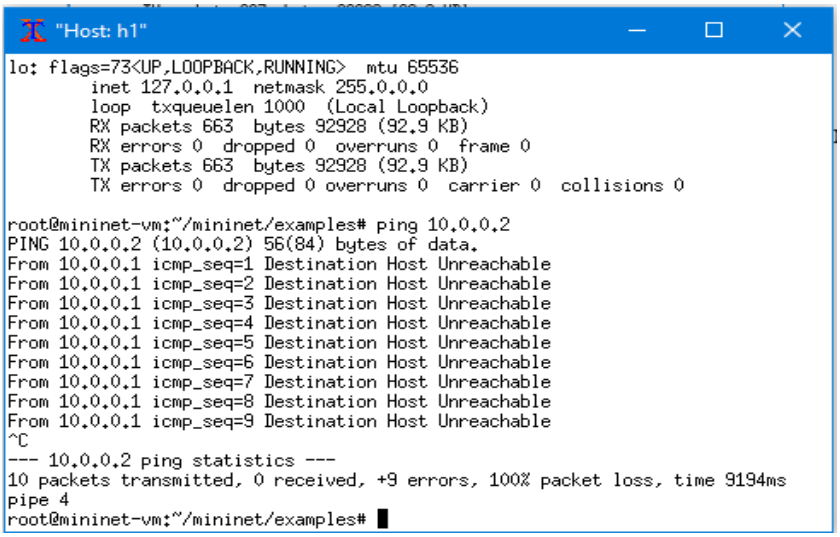

- Повторюємо ці дії на хості h2. Його інтерфейс h2-eth0 повинен бути налаштований з IP-адресою 10.0.0.2 та маскою підмережі 255.0.0.0.




У цьому завданні на комутаторі  спочатку створюються два потоки для з’єднання хостів h1 та h2, а потім ще два потоки  для з’єднання хостів h1 та h3. В результаті на комутаторі відбувається переключення з інтерфейсу s1-eth2 на інтерфейс т s1-eth3.

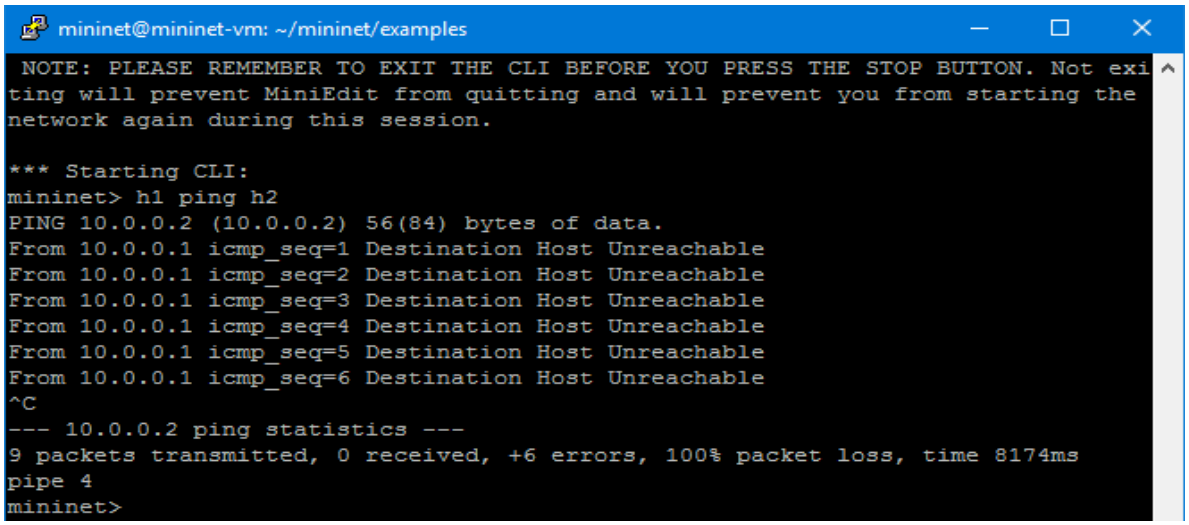
Перевіримо доступність хостів h1та h2 один до одного.
Перевірку виконуємо двома  способами: з терміналів хостів і з консолі Mininet:

- На хост-терміналі h1 вводимо *ping 10.0.0.2*. Ця команда перевіряє наявність з'єднання між host h1 та host h2. Щоб зупинити тест, натискаємо *Ctrl+C*.

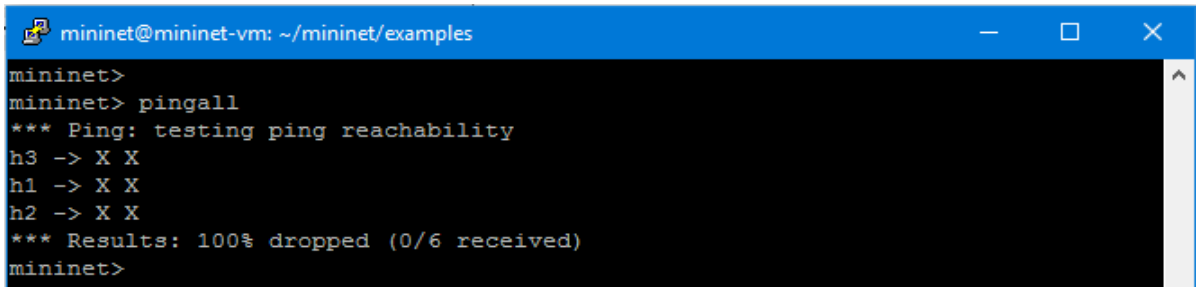



- Переходимо на консоль Mininet і перевіряємо наявність з'єднання між хостами h1 та h2. З'єднання відсутнє.

mininet > **h1 ping h2**



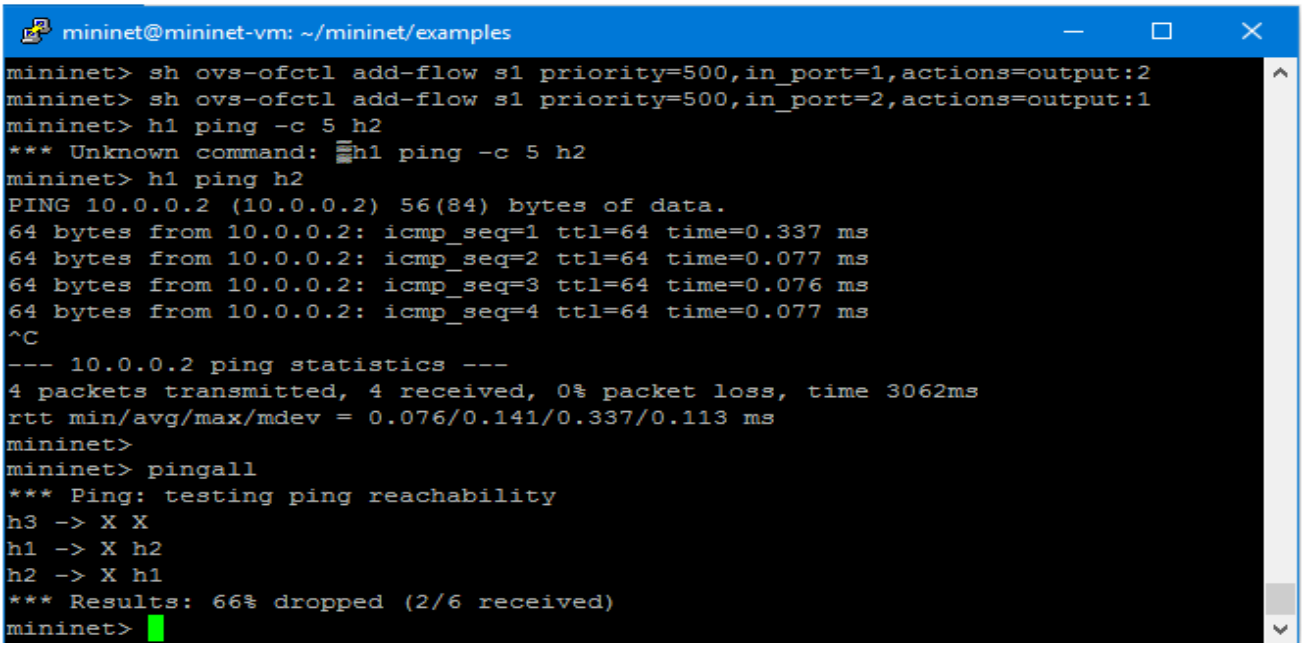

На консолі Mininet виконаємо команду *pingall*. З'єднання відсутні.




Один із способів забезпечити з'єднання комутатора з хостами — просто направити весь вхідний трафік з одного порту на інший порт. У цьому сценарії створюються правила для двох потоків: вхідного трафіка з порту s1-eth1 до s1-eth2 і вхідного трафіка з s1-eth2 до s1-eth1. Ці правила дозволять створити потрібні з'єднання.

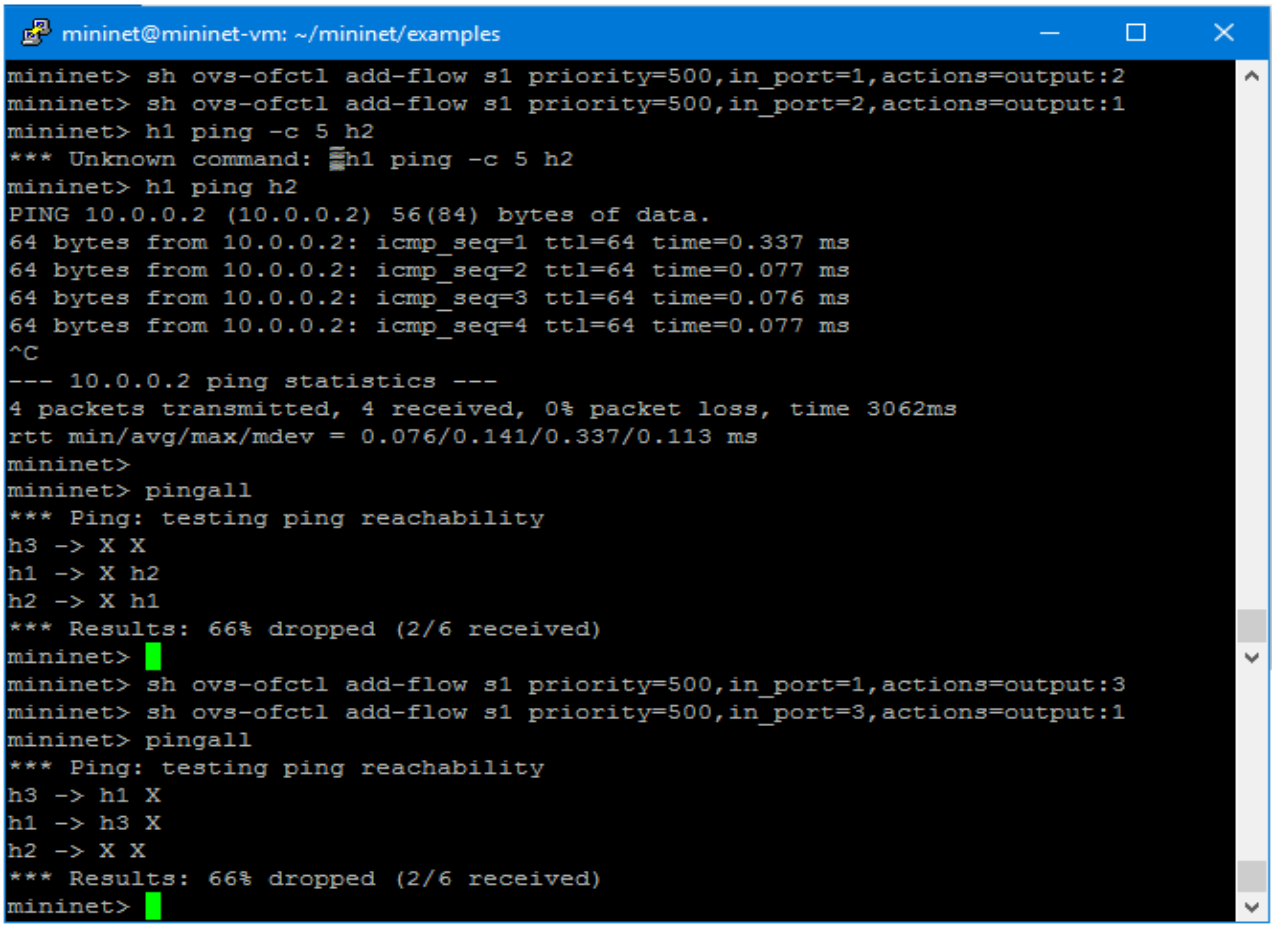
mininet> **sh ovs-ofctl add-flow s1 priority=500,in_port=1,action=output:2**
mininet> **sh ovs-ofctl add-flow s1 priority=500,in_port=2,action=output:1**

Перевіримо за допомогою ICMP наявність з’єднання між хостами h1 та h2. З’єднання встановлені.




Запишемо ще два правила, які дозволять з’єднати порти комутатора s1-eth1 до s1-eth3 і перевіримо наявність з’єднань між хостами h1 та h3:

mininet> **sh ovs-ofctl add-flow s1 priority=500,in_port=1,action=output:3**
mininet> **sh ovs-ofctl add-flow s1 priority=500,in_port=3,action=output:1**




Відбулося переключення трафіка з порту s1-eth2 на порт s1-eth3 комутатора.

Ми ознайомилися з ручним налаштуванням потоків Open vSwitch.

## Контрольні запитання

1. Послідовність кроків для організації роботи із графічною оболонкою MiniEdit з Windows-середовища.
2. Призначення утиліти PuTTY.
3. Призначення серверу Xming Server.
4. Переваги та недоліки використання засобів командного рядка та графічної оболонки MiniEdit для створення SDN мережі заданої топології.
5. Засоби тестування мережі у середовищі MiniEdit.

# Лабораторна робота 5

## Побудова брандмауера як сервісу в Mininet.

**Мета:** Ознайомитися з структурою файлу з кодом Python, в якому прописана певна мережева топологія. Провести дослідження поведінки створеної тестової мережі при виконанні різних сценаріїв керування мережевими комутаторами. Встановити простий брандмауер на хості.

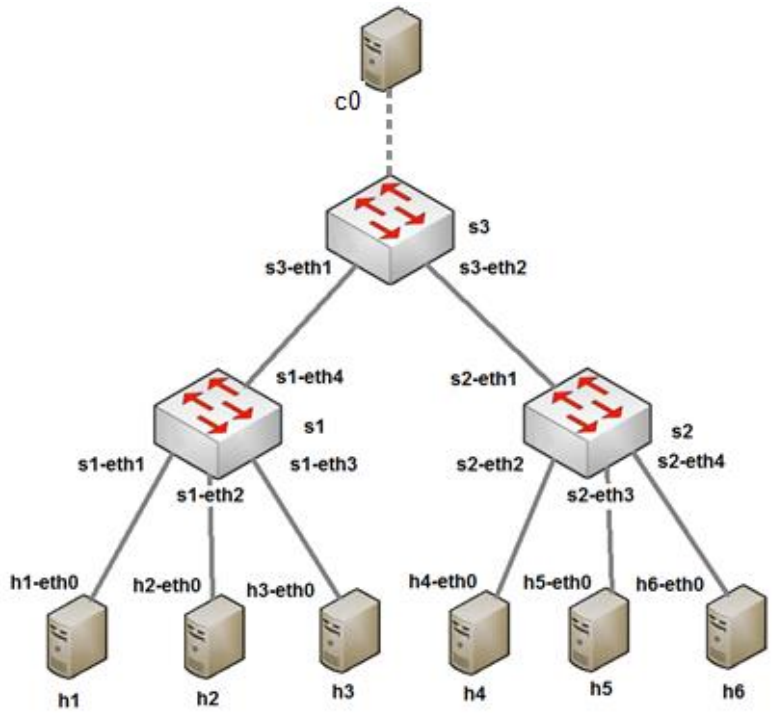
### План роботи:

1. Підготовка до виконання скрипта досліджуваної топології.
2. Запуск скрипта топології.
3. Створення потоків на комутаторах і виконання перевірок доступності хостів в мережі.
4. Перехоплення трафіку аналізатором мережевого трафіку Wireshark та утилітою *tcpdump*.
5. Налаштування простого брандмауера *iptables* на хості.

### 1. Виконання роботи

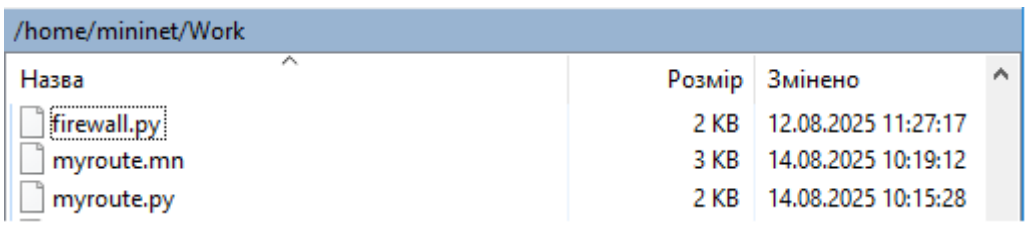
1. Створимо топологію тестової мережі за допомогою скрипта на Python. Цей скрипт описує топологію: 3 комутатори та 6 хостів, де 3 хости підключені до одного комутатора, а інші 3 три – підключені до іншого комутатора, третій комутатор з'єднує два комутатора.

Досліджувана топологія має такий графічний вигляд:




2. Копіюємо нижче наведений код скрипта у створюваний файл з ім’ям *firewall.py*. Файл із скриптом зберігаємо в папку Work на віртуальній машині Mininet за адресою /home/mininet/Work.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.cli import CLI
from mininet.link import Intf
from mininet.log import setLogLevel, info
from mininet.node import Controller, OVSKernelSwitch, RemoteController

def myNetwork():

    net = Mininet( topo=None,controller=RemoteController)

    info( '*** Adding controller\n' )
    c0 = net.addController('c0', controller=RemoteController, ip="127.0.0.1")


    info( '*** Add switches\n')
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')
    s3 = net.addSwitch('s3')

    info( '*** Add hosts\n')
    h1 = net.addHost('h1', ip="10.0.0.1")
    h2 = net.addHost('h2', ip="10.0.0.2")
    h3 = net.addHost('h3',ip="10.0.0.3")
    h4 = net.addHost('h4',ip="10.0.0.4")
    h5 = net.addHost('h5',ip="10.0.0.5")
    h6 = net.addHost('h6',ip="10.0.0.6")

    info( '*** Add links\n')
    net.addLink(h1, s1)
    net.addLink(h2, s1)
    net.addLink(h3, s1)
    net.addLink(s1, s3)
    net.addLink(s2, s3)
    net.addLink(h4, s2)
    net.addLink(h5, s2)
    net.addLink(h6, s2)

    info( '*** Starting network\n')
    net.start()
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```




3. Запускаємо збережений скрипт.

**$ sudo python firewall.py**

```
mininet@mininet-vm:~/Work$ sudo python firewall.py
  File "firewall.py", line 9
SyntaxError: Non-ASCII character '\xc4' in file firewall.py on line 9, but no encoding declared; see
 http://python.org/dev/peps/pep-0263/ for details
mininet@mininet-vm:~/Work$ sudo python firewall.py
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> _
```

4. Перевіряємо доступність хостів:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X
h2 -> X X X X X
h3 -> X X X X X
h4 -> X X X X X
h5 -> X X X X X
h6 -> X X X X X
*** Results: 100% dropped (0/30 received)
mininet> _
```

5. Створюємо потоки на комутаторах s1 і s2:

```
mininet> sh ovs-ofctl add-flow s1 action=normal
mininet> sh ovs-ofctl add-flow s2 action=normal
```

Це дозволить лише h1, h2, h3 і h4, h5, h6 мати доступ один до одного. Оскільки вони під'єднані до окремих комутаторів, хости h1, h2, h3 не зможуть підключитися до хостів h4, h5, h6, оскільки потоки не спрямовані на комутатор s3.

Ще раз перевіряємо доступність хостів:

```
mininet> sh ovs-ofctl add-flow s1 actions=normal
mininet> sh ovs-ofctl add-flow s2 actions=normal
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X X X
h2 -> h1 h3 X X X
h3 -> h1 h2 X X X
h4 -> X X X h5 h6
h5 -> X X X h4 h6
h6 -> X X X h4 h5
*** Results: 60% dropped (12/30 received)
mininet>
```

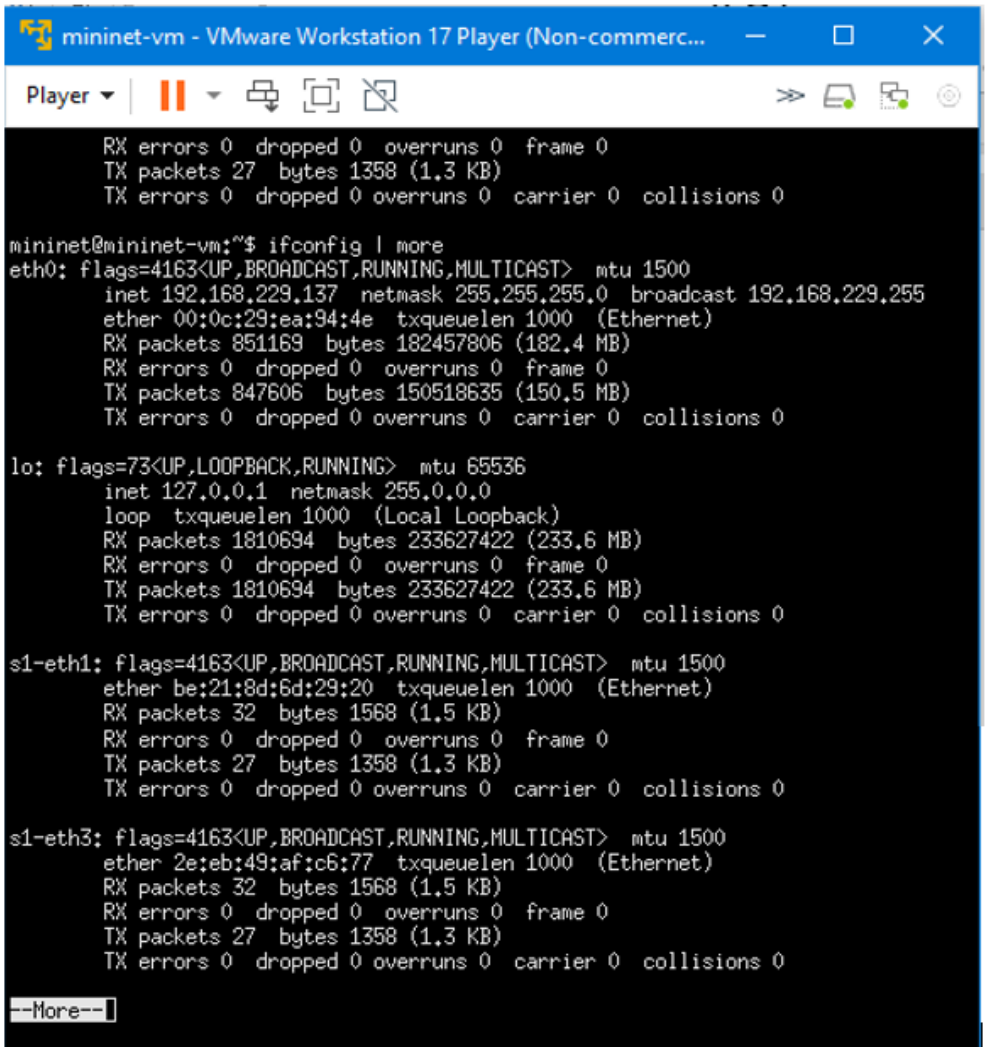
6. Переглянемо проходження пакетів через інтерфейс комутатора s1-eth3 за допомогою програми Wireshark. Для того, щоб Wireshak використовував окреме вікно, з консолі віртуальної машини запускаємо утиліту **startx**:

```
$ startx
```

Якщо при спробі запустити графічне середовище з командного рядка з'являється повідомлення «команда startx не знайдена», то потрібно додатково встановити пакет **xinit**:

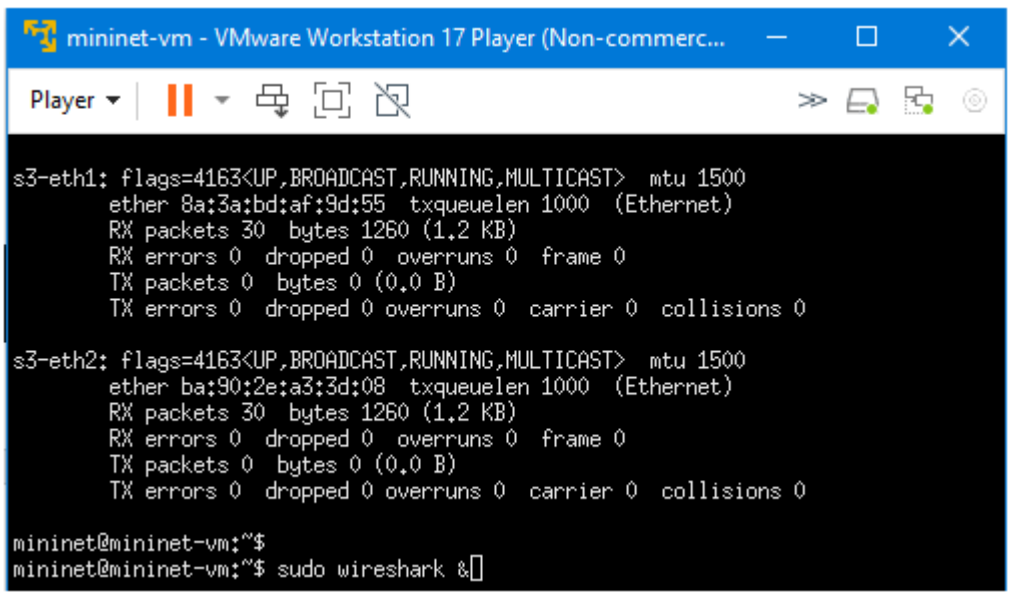
```
$ sudo apt update
$ sudo apt install xinit
```

Після встановлення *xinit* знову запускаємо *startx*. З'являється нове вікно.
У цьому вікні вводимо команду *ifconfig*, щоб отримати імена доступних інтерфейсів.

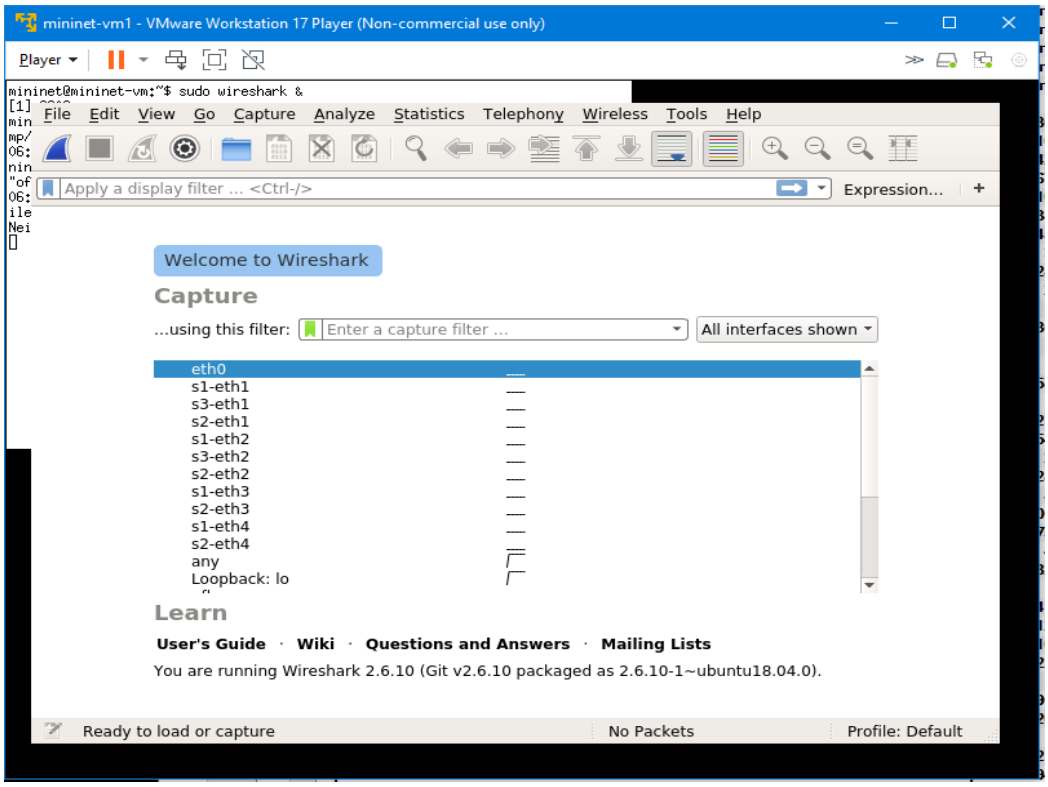



7. У цьому ж вікні запускаємо Wireshark у фоновому режимі.

```
$ sudo wireshark &
```

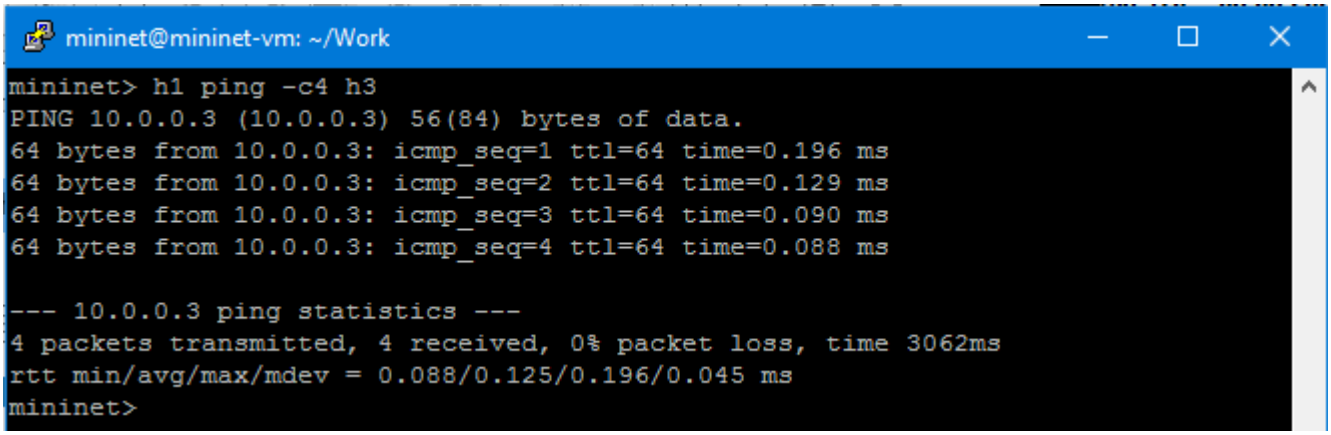


У списку доступних інтерфейсів вибираємо інтерфейс *s1-eth3* для захоплення трафіку (подвійний клік).

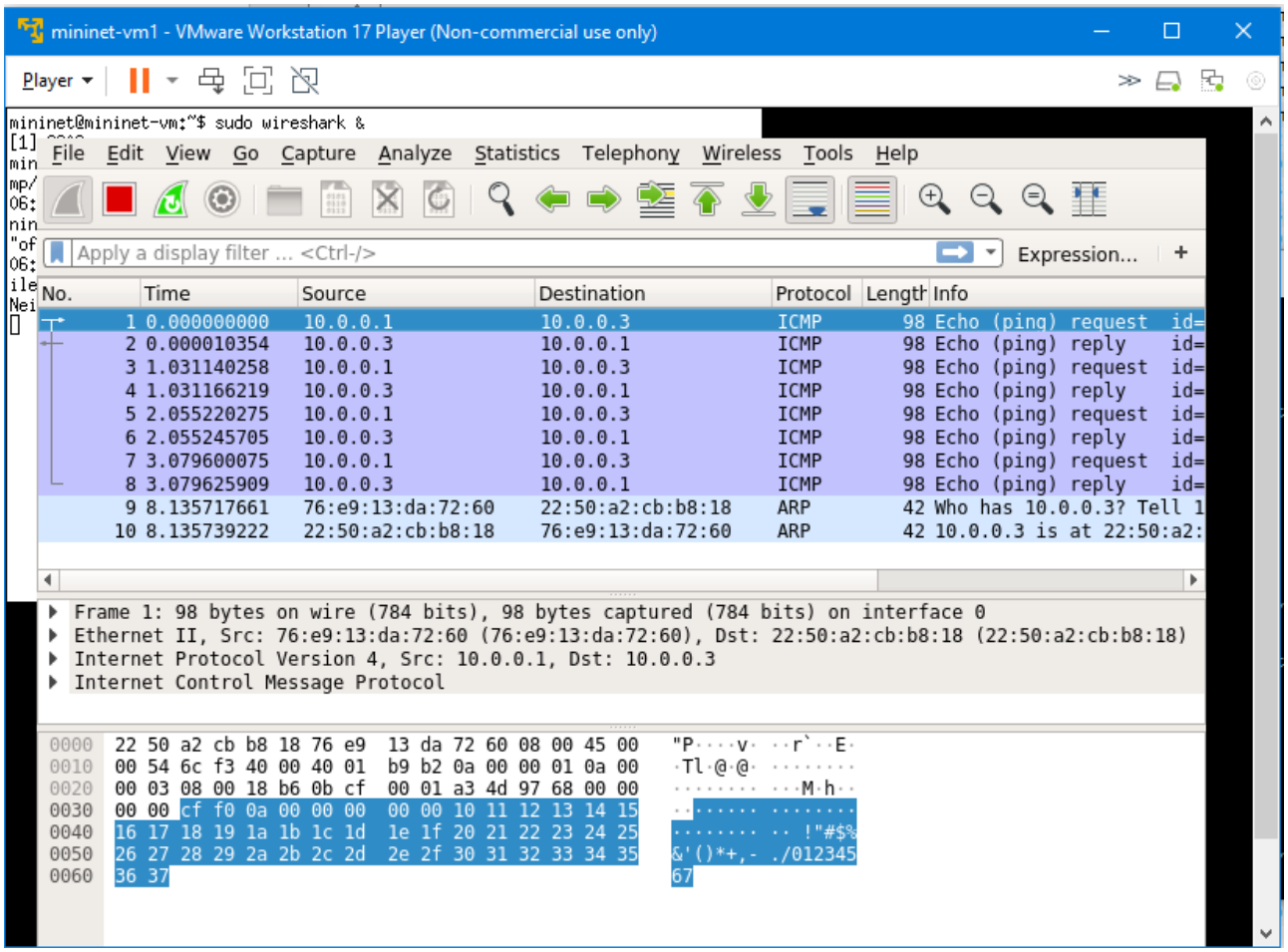



8. Переходимо на консоль віртуальної машини Mininet і перевіряємо з'єднання між h1 і h3:

   mininet> **h1 ping –c4 h3**




9. У вікні, де запущений Wireshark спостерігаємо проходження пакетів ICMP через інтерфейс комутатора s1-eth3.

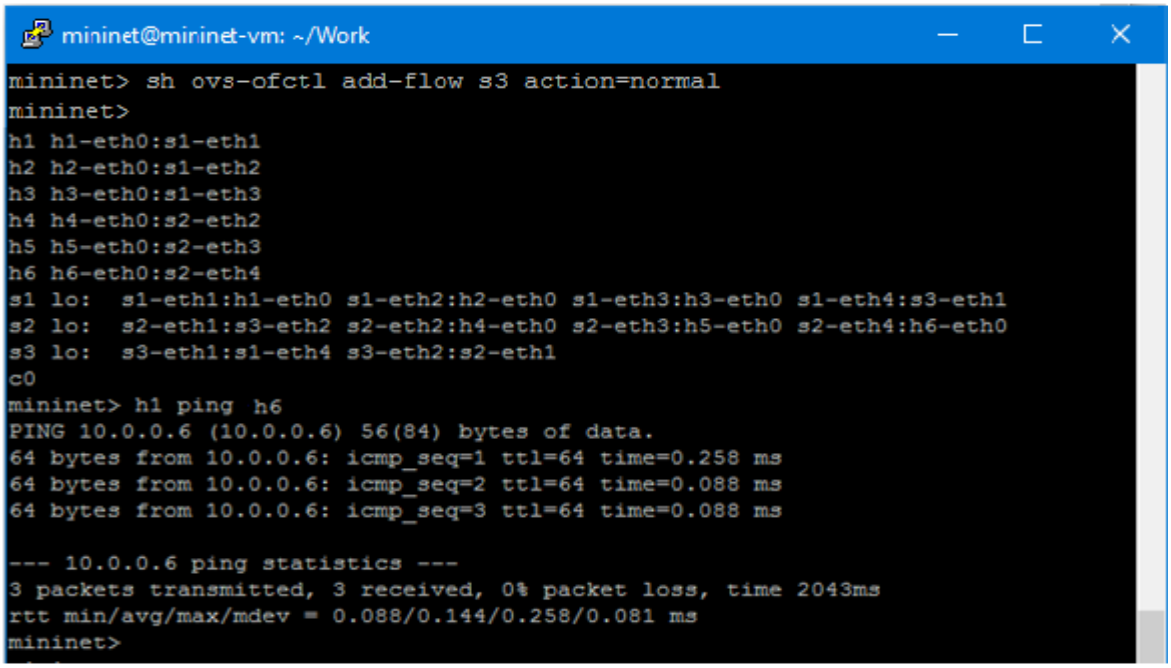



Завершуємо роботу програми Wireshark: *File – Quit*. Закриваємо вікно, в якому запускався Wireshark:

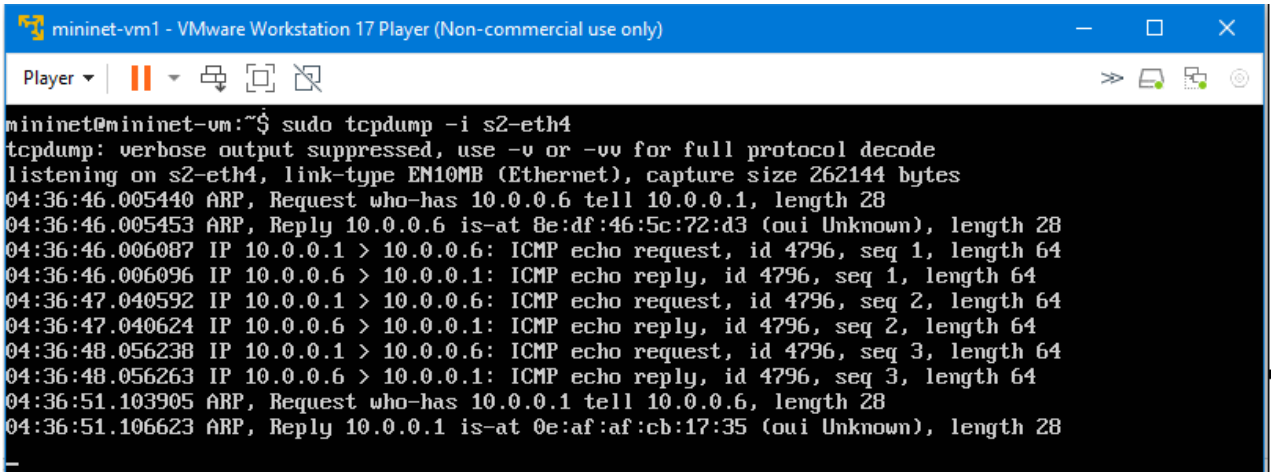
$ **exit**

10. З консолі Mininet створюємо потік на комутаторі s3 і перевіряємо наявність з'єднання між хостами h1 та h6.

mininet> **sh ovs-ofctl add-flow s3 action=normal**



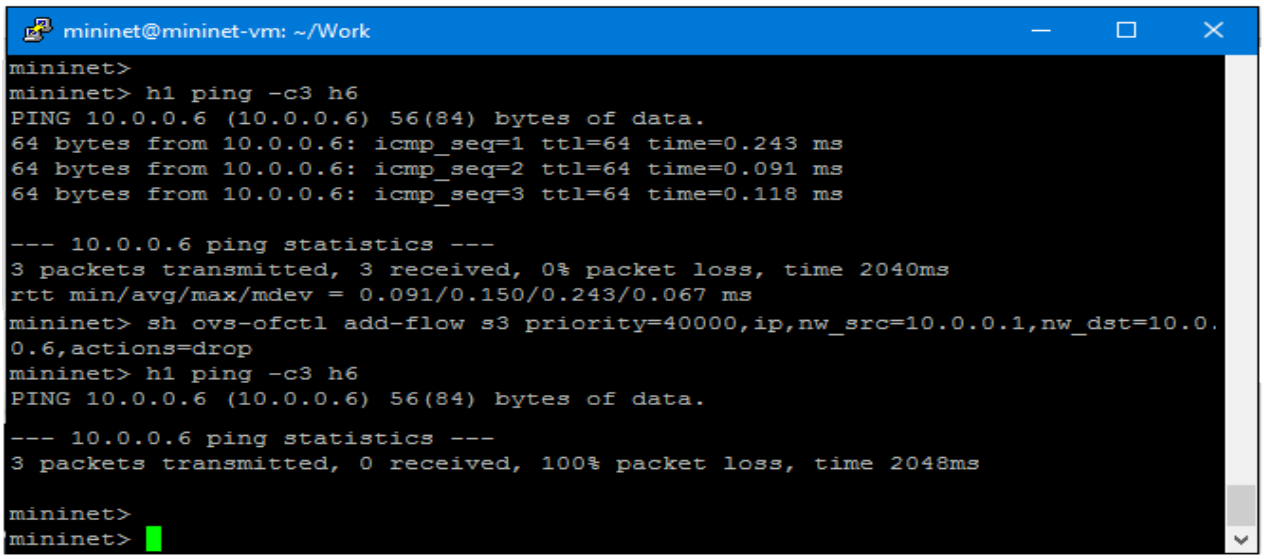
11. З консолі віртуальної машини запускаємо утиліту *tcpdump* для захоплення трафіку на інтерфейсі s2-eth4, до якого під'єднаний хост h6. Спостерігаємо надходження пакетів ICMP на цей інтерфейс.



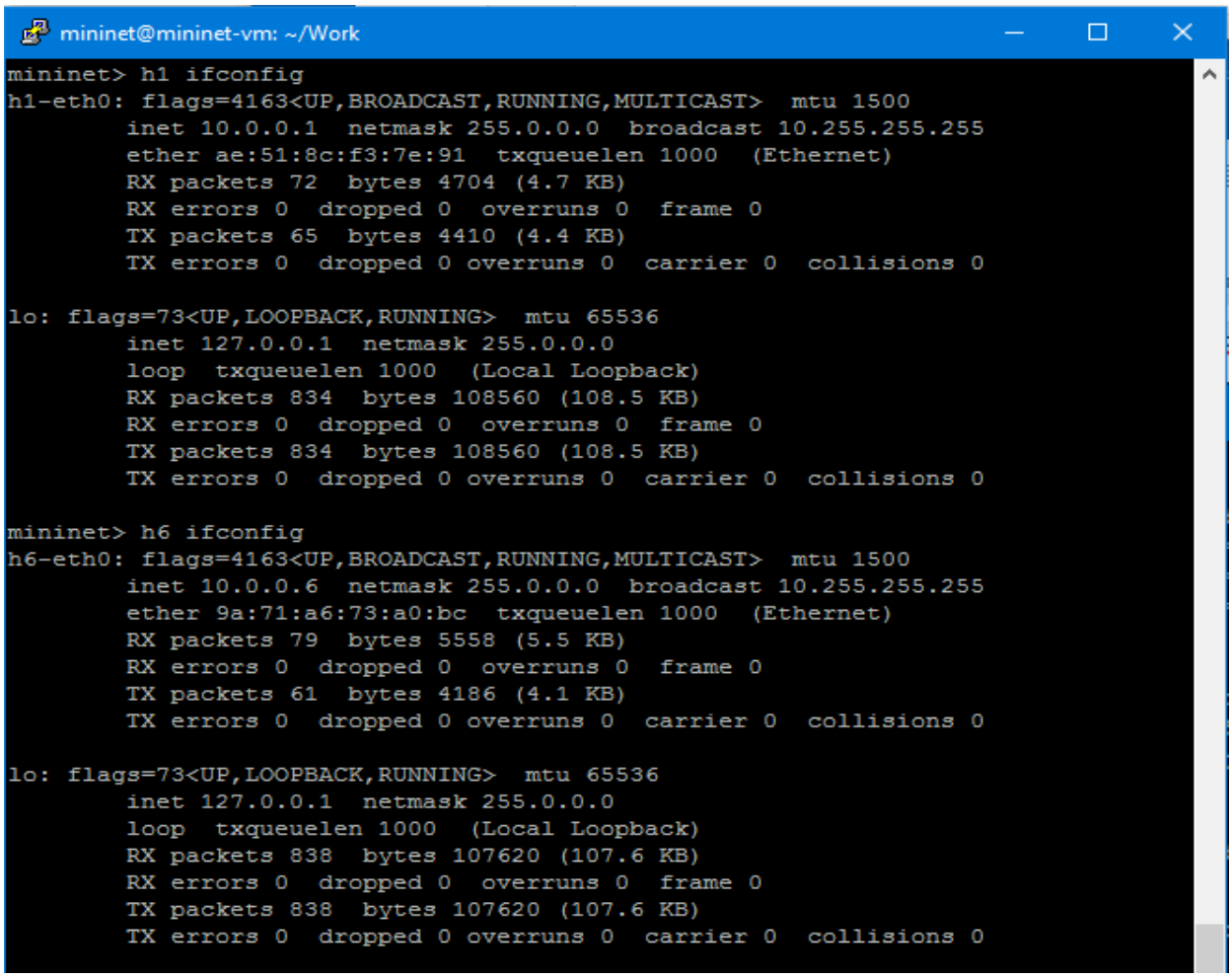

12. Виконаємо ще одне налаштування. На комутаторі s3 створимо правило потоку, яке буде забороняти проходження пакетів IP від хоста h1 до хоста h6. Адресація хостів – за допомогою IP-адрес. Пріоритет цього правила має бути вищим, ніж встановлений за замовчуванням. Отже, запишемо таке правило:

mininet> **sh ovs-ofctl add-flow s3 priority=40000, ip,nw_src=10.0.0.1,nw_dst=10.0.0.6,action=drop**

Знову перевіряємо наявність з'єднаня між хостами h1 та h6. З'єднання заборонене.



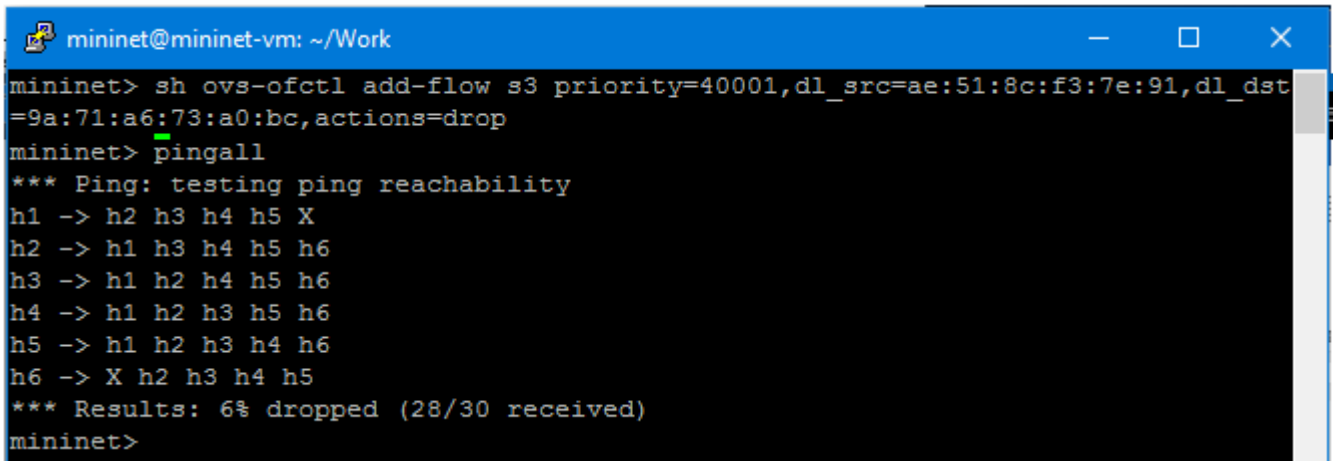

13. Аналогічне за дією правило можна записати використовуючи для адресації хостів MAC-адреси. Визначимо MAC-адреси хостів h1 і h6 за допомогою команди ifconfig, яку виконаємо з консолі Mininet.




У цьому випадку потік має такий вигляд:

mininet> **sh ovs-ofctl add-flow s3**
**priority=40001,dl_src=ae:51:8c:f3:7e:91,dl_dst=9a:71:a6:73:a0:bc,actions=drop**

Перевіримо доступність хостів:



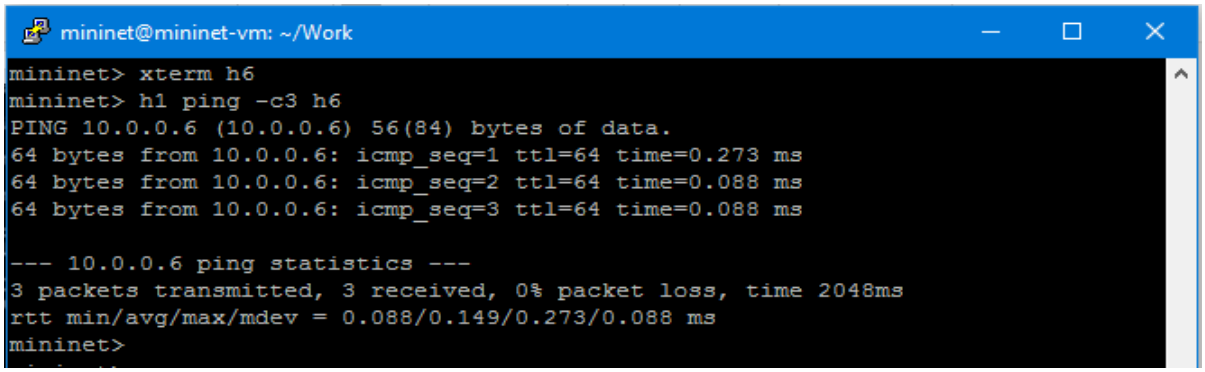

14. Тепер запускаємо консоль віртуального хоста h6. Для запуску консолі потрібно використовувати команду xterm. Вона запускає окреме вікно з консоллю, в якій знаходиться повноцінний Ubuntu на віртуальній машині. Весь набір команд Ubuntu працює в цій консолі, включаючи ifconfig, ping, tcpdump, iperf, vi, sh тощо.

Отже, відкриваємо консоль *xterm* на віртуальному хості h6. З консолі віртуальної машини h6 за допомогою команди *ifconfig* визначаємо назву інтерфейсу хоста h6. Запускаємо утиліту *tcpdump* для захоплення трафіку на цьому інтерфейсі.

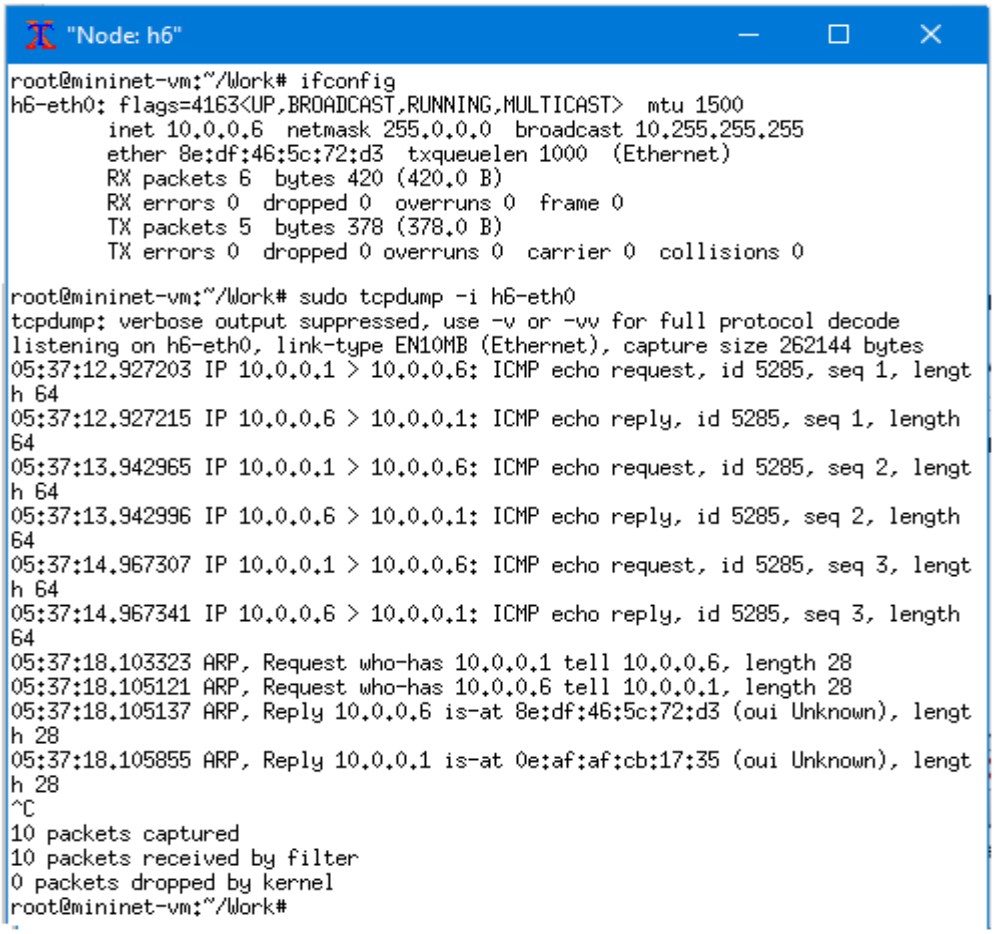
mininet> **xterm h6**
# **ifconfig**
# **tcpdump -i h6-eth0**

15. Повертаємося до консолі віртуальної машини mininet і перевіряємо з'єднання між хостами h1 та h6.

mininet> **h1 ping –c3 h6**




На консолі віртуальної машини h6 спостерігаємо проходження пакетів ICMP через інтерфейс хоста h6.



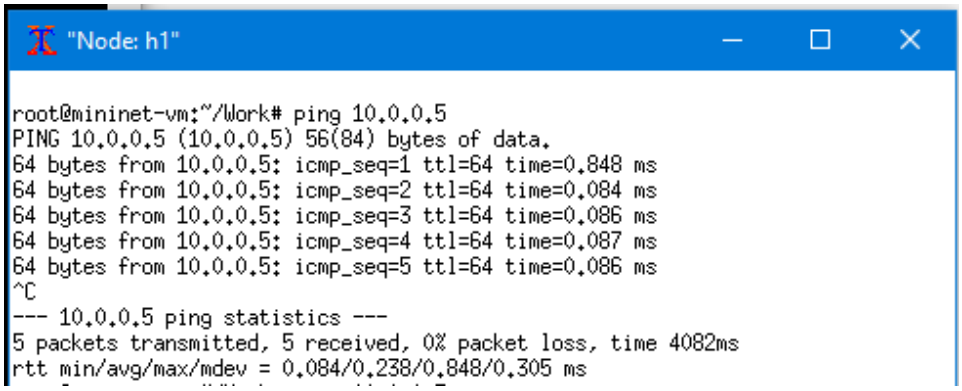
16. У Mininet для створення та налаштування простого брандмауера (firewall) використовується утиліта *iptables*. Вона дозволяє визначати правила фільтрації мережевого трафіку між вузлами в мережі, що моделюється в Mininet. Для цього потрібно використовувати командний рядок, де застосовуються команди *iptables* з відповідними параметрами. Розглянемо, як можна використовувати *iptables* у Mininet для блокування трафіку між двома вузлами.

Розглянемо приклад блокування трафіку між h1 та h5 за допомогою *iptables*:

Відкриваємо термінали на хостах h1 та h5:

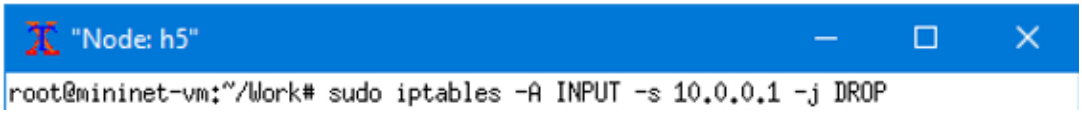
```
mininet> xterm h1
mininet> xterm h5
```

Перевіряємо наявність з'єднання між хостами h1 і h5:

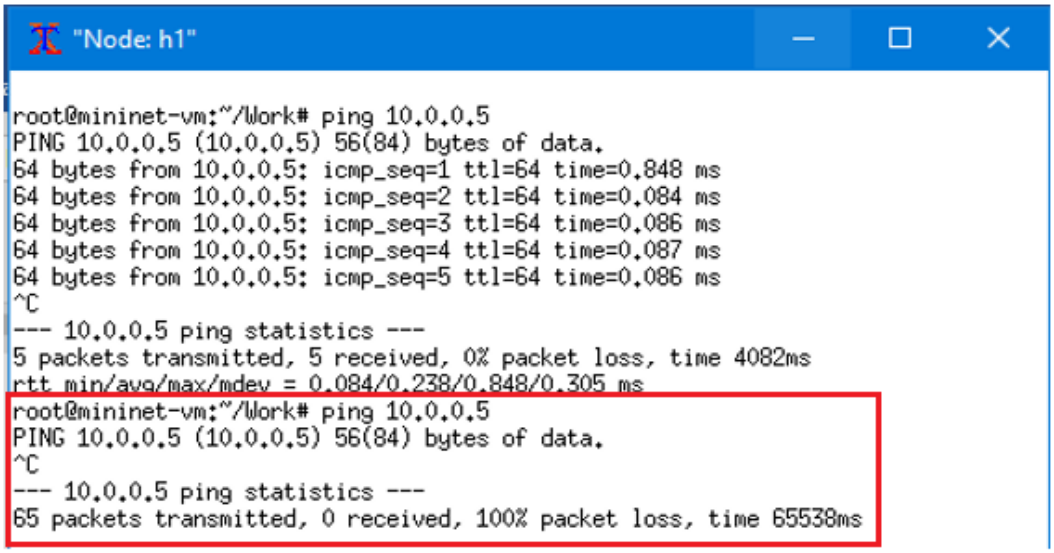



На хості h5 запишемо блокуюче правило *iptables*:

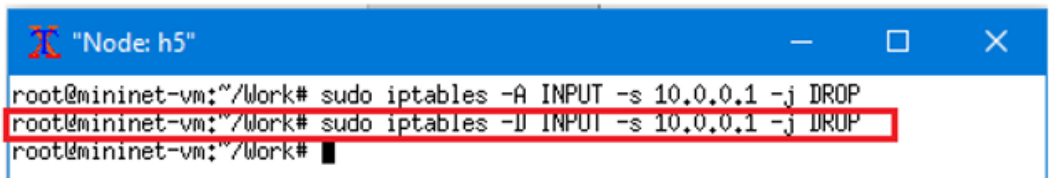
# **sudo iptables -A INPUT –s 10.0.0.1 –j DROP**



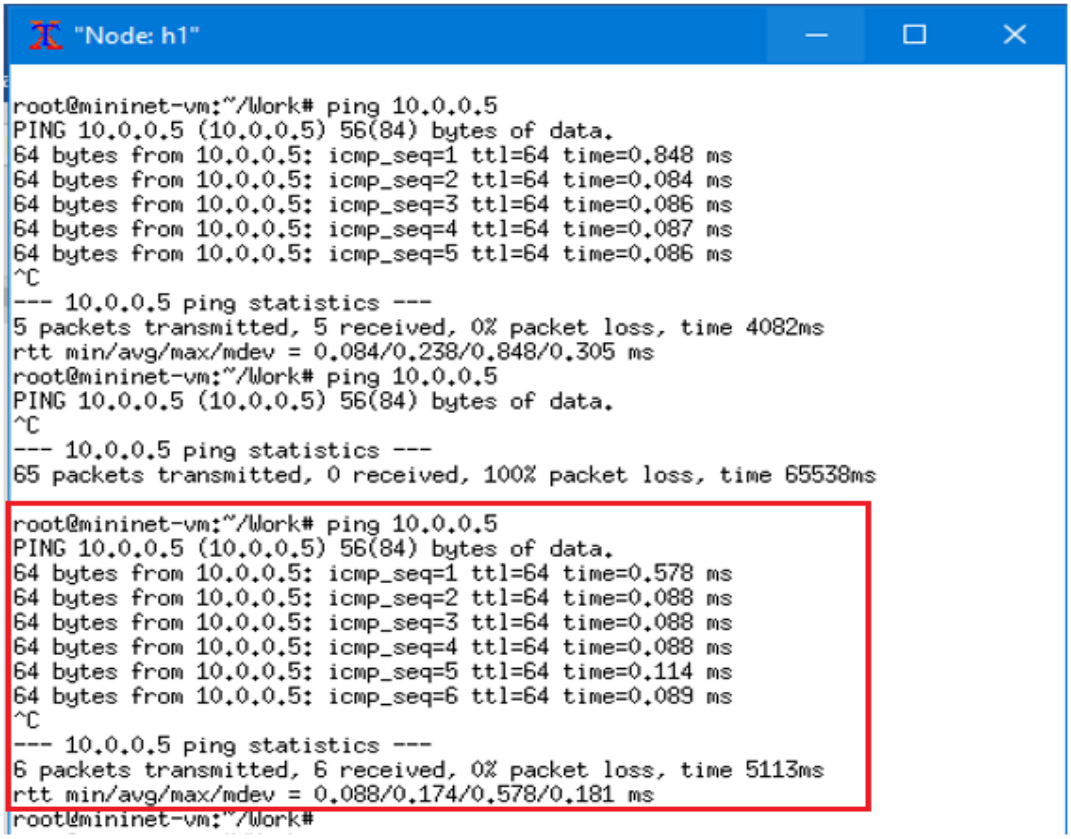

Повторно перевіримо наявність з'єднання між h1 і h5. З'єднання заблоковане правилом *iptables*.




Відміняємо блокування трафіку на хості h5:




Ще раз перевіряємо наявність з'єднання між h1 і h5. З'єднання розблоковане правилом *iptables*.




Виконання лабораторної роботи завершене.

**Контрольні запитання**

1. Як запустити простий вебсервер і відповідний клієнт на хості в Mininet?

# Лабораторна робота 6

## Налаштування маршрутизації в Mininet.

**Мета:** Ознайомитися з різними варіантами налаштування простого маршрутизатора у середовищі Mininet.

**План роботи:**

1. Створення графічної топології мережі в MiniEdit.
2. Отримання та аналіз скриптів початкового сценарію на Python.
3. Внесення змін і доповнень до початкового сценарію для створення правил комутації і маршрутизації.
4. Запуск та налаштування зміненого скрипта. Перевірка правильності виконаних налаштувань.

## 1. Теоретичні відомості

Програмно-визначена мережа (SDN) – це мережева архітектура, яка характеризується розділенням площини даних і площини управління.
В архітектурі SDN можна виділити три рівня (рисунок 1):

- інфраструктурний рівень (площина даних), що містить набір мережевих пристроїв (комутаторів і маршрутизаторів);
- рівень управління, що включає в себе мережеву операційну систему, яка забезпечує додаткам мережеві сервіси та програмний інтерфейс для управління мережевими пристроями та мережею;
- рівень мережевих додатків для гнучкого та ефективного управління мережею

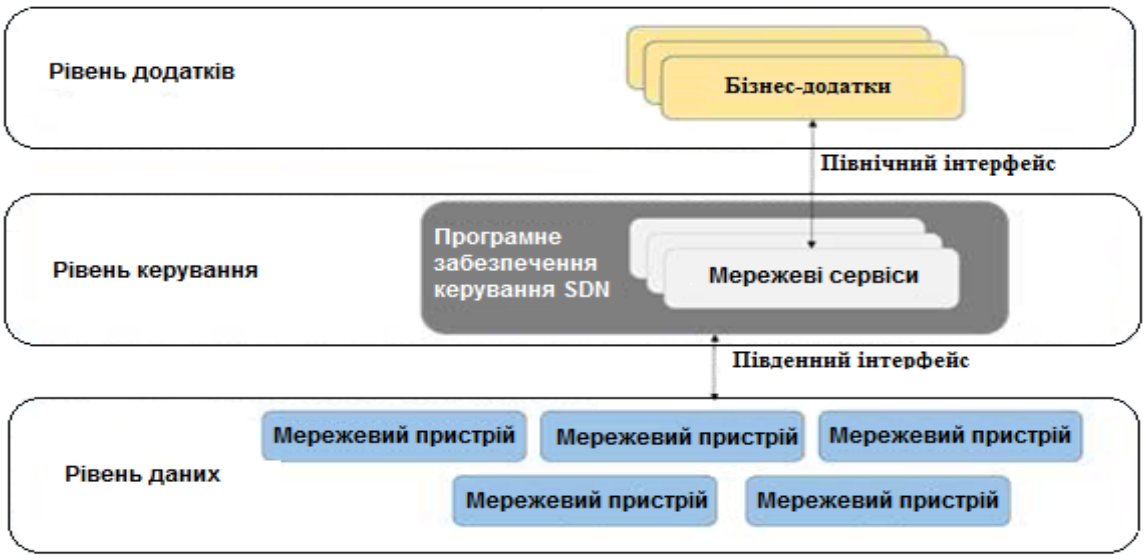



Рисунок 1 – Архітектура SDN

Переваги використання SDN досягаються за рахунок відокремлення площини даних від площини управління. Також із розділенням площин управління та даних мережеві комутатори стають простими пристроями пересилання, а логіка управління реалізується у логічно централізованому контролері (або мережевій операційній системі), спрощуючи застосування політик, налаштування та еволюцію мережі.

Площина даних включає мережеві пристрої, які відповідають за ефективну передачу даних. На цьому рівні знаходиться мережеве обладнання, як і в традиційній мережі. Однак відмінність від традиційної мережі полягає в тому, що в даному випадку пристрої, як фізичні, так і віртуальні, є лише засобами для пересилання потоку даних і не можуть приймати автономні рішення. Здатність приймати рішення переноситься на рівень управління.

Відмінною особливістю SDN є також пересилання на основі потоку, а не на основі адрес призначення. Потік у цілому визначається набором значень полів пакета, що діють як критерій при виборі відповідності (фільтра) та набору дій (інструкцій). У контексті SDN потік представляє собою послідовність пакетів між джерелом і пунктом призначенням. Всі пакети потоку отримують ідентичні політики обслуговування на пристроях пересилки, що дозволяє уніфікувати поведінку різних типів мережевих пристроїв, включаючи маршрутизатори, комутатори, міжмережеві екрани та інші проміжні мережеві пристрої. Гнучкість використання потоків обмежується лише можливостями реалізованих таблиць потоків.
Потік визначається за допомогою полів заголовка (match fields), таких як IP-адреси відправника та отримувача, порти TCP/UDP, VLAN ID або MAC-адреси.
Усі пакети, що належать до одного потоку, обробляються комутатором однаково згідно з інструкціями, записаними у таблиці потоків (Flow Table).
Для кожного потоку контролер задає певні дії, наприклад: переслати на конкретний порт, змінити заголовок, відкинути пакет (drop) або обмежити швидкість.
Потік може бути як дуже вузьким (наприклад, конкретне TCP-з’єднання між двома програмами), так і дуже широким (наприклад, увесь трафік між двома підмережами).
Це поняття лежить в основі роботи протоколу OpenFlow, де замість традиційної маршрутизації на основі лише адреси призначення використовується гнучке керування потоками на базі політик контролера.

Взаємодія площини управління та площини застосунків здійснюється за допомогою північного інтерфейсу (NorthBondAPI). Зазвичай кожен контролер має свій власний північний інтерфейс. Наприклад, такі контролери, як Floodlight, Trema, NOX, Onix і OpenDaylight використовують власні північні API.

Зв'язок між пристроями та контролером здійснюється через так званий південний інтерфейс SBI (SouthBondAPI). Він називається південним, тому що на діаграмах зазвичай контролер зображується зверху, а мережеві пристрої знизу, на півдні. Цей термін стосується не будь-якого фізичного інтерфейсу на пристрої, а швидше до програмного інтерфейсу, який використовується для забезпечення зв'язку контролера та мережевих пристроїв. SBI зазвичай складається з протоколу зв’язку та API, інтерфейсу прикладного програмування, який у цьому випадку в основному використовується для полегшення обміну даними між програмами.

Контролер — це програма в архітектурі SDN, яка зазвичай працює на сервері та використовує спеціальні протоколи для роботи з мережевими пристроями (комутаторами). Між контролером і мережевими пристроями відбувається обмін

даними. Наприклад, API на мережевих пристроях може дозволити контролеру отримувати доступ до інформації про них, керувати їх таблицями площини даних, які використовуються для пересилання пакетів тощо. API відіграють важливу роль у автоматизації мережевих завдань. Ось кілька південних інтерфейсів: OpenFlow, Cisco OpenFlex, Cisco onePK, NETCONF.

Функції контролера і комутатора. Контролер SDN визначає потоки, які існують у площині даних. Кожен потік по мережі повинен спочатку бути дозволений контролером, який перевіряє, що потік не порушує мережеву політику. Якщо контролер дозволяє потік, він вираховує маршрут для потоку і додає запис для цього потоку в кожен комутатор на шляху. На відміну від складних функцій, що входять до складу контролера, комутатори просто перенаправляють пакети згідно з таблицями потоків, записи в яких можуть бути заповнені лише контролером. Зв'язок між контролером і комутаторами відбувається за допомогою стандартизованого протоколу і API. Найчастіше для цього використовується протокол OpenFlow.
Архітектура SDN відрізняється винятковою гнучкістю, можлива робота з різними типами комутаторів і на різних рівнях протоколу. Контролери та комутатори SDN можуть бути реалізовані для Ethernet-комутаторів (Layer 2), маршрутизаторів (Layer 3), транспорту (Layer 4) або маршрутизації на рівні додатків.

В архітектурі SDN комутатор виконує наступні функції:

- Комутатор інкапсулює та перенаправляє перший пакет потоку до контролера SDN, щоб контролер міг вирішити, чи слід додати потік до таблиці потоків комутатора.
- Комутатор перенаправляє вхідні пакети з відповідного порту на основі таблиці потоків. Таблиця потоків може включати інформацію про пріоритет, задану контролером.
- Комутатор може відкидати пакети у визначеному потоці, тимчасово або постійно, як це продиктовано контролером. Відкидання пакетів може використовуватися для цілей безпеки, стримування атак типу відмова в обслуговуванні (DoS) або вимог до керування трафіком.

Таким чином, контролер SDN керує роботою комутаторів у SDN. Це керування здійснюється за допомогою API, що дозволяє контролеру задовольняти різноманітні вимоги додатків без зміни будь-яких аспектів нижчого рівня мережі.

Робота протоколу Openflow полягає у модифікації таблиці передачі пакетів комутатора (рисунок 2).

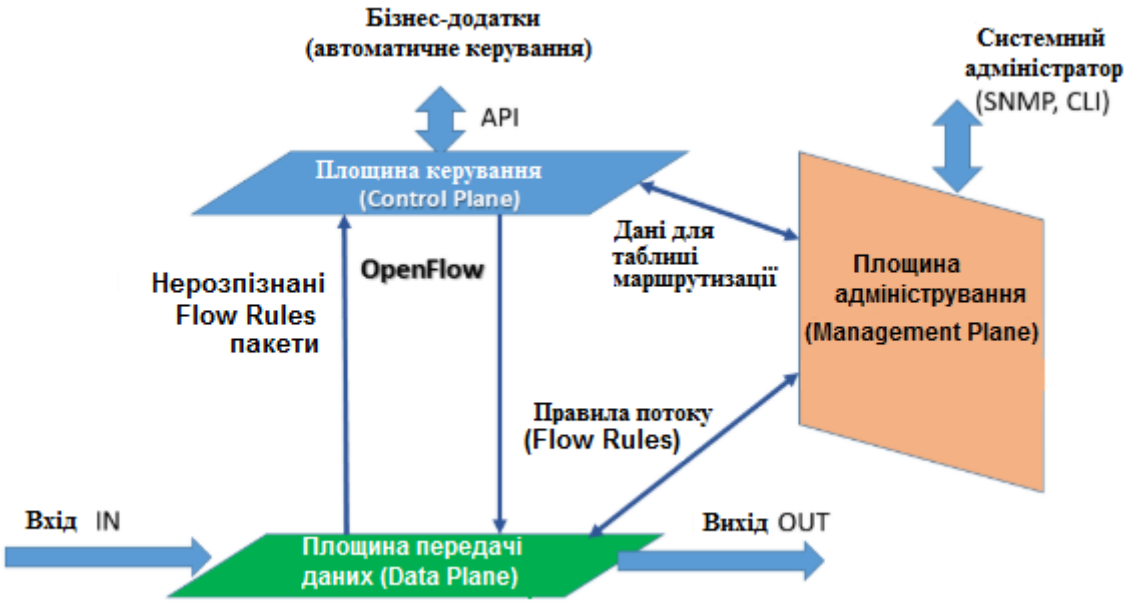



Рисунок 2 – Керування потоками в SDN

Площина адміністрування (Management Plane) встановлює режими роботи пристрою, модифікує версії системного програмного забезпечення, виконує команди простого протоколу адміністрування мережі SNMP (Simple Network Management Protocol) від зовнішньої системи адміністрування мережі NMS, а також команди зовнішньої конфігурації пристрою через інтерфейс командного рядка CLI (Command Line Interface). Площина передачі даних (Data Plane) передає пакети та кадри даних з входу пристрою (IN) на його вихід (OUT) відповідно до таблиці передачі (Forwarding Table). На площині керування (Control Plane) працюють протоколи маршрутизації та комутації.

OpenFlow визначає стандарт розсилки «правил потоку» (flow rules) у таблицю передачі (forwarding table) кожного мережевого пристрою таким чином, щоб площина керування могла керувати площиною передачі даних. Ці правила потоку містять налаштування для мережевих пристроїв, такі як MAC-адреса джерела та призначення (source & destination MAC), IP-протокол для джерела та призначення (source & destination IP), TCP-протокол для джерела та призначення (source and destination TCP), дані про віртуальну (логічну) локальну мережу VLAN у загальному пулі інфраструктури, мітки QoS і MPLS, та іншу інформацію. Правила потоку потім додаються до існуючих таблиць передачі пакетів (forwarding table) з входу на вихід кожного мережевого пристрою.

## 2. Виконання роботи

У цій лабораторній роботі роботі потрібно в середовищі Mininet вручну налаштувати простий маршрутизатор та комутатор(и) для обробки та пересилання пакетів. Завдання: на комутаторі створити правила пересилання, щоб вхідні пакети потрапляли на певний вихідний інтерфейс. Додавання потоків на комутаторі виконується за допомогою інструменту *ovs-ofctl*.

*ovs-ofctl* – це утиліта командного рядка для керування та моніторингу комутаторів *openvswitch* та будь-яких інших комутаторів, що підтримують протокол OpenFlow. За

допомогою *ovs-ofctl* можна переглядати стан OpenFlow комутатора, включаючи його можливості, конфігурацію та записи в таблицях потоків, а також виконувати адміністративні дії.

У програмно-визначених мережах (SDN) маршрутизація через CLI використовується для управління центральним контролером, який, у свою чергу, через API визначає правила передачі даних на мережевих пристроях. CLI застосовується для взаємодії з контролером (наприклад, для налаштування правил маршрутизації або управління потоками). В лабораторній роботі будемо застосовувати CLI для налаштування правил маршрутизації та управління потоками безпосередньо на маршрутизаторі та кожному комутаторі.

Хости, які підключаються до комутатора *openvswitch*, знаходяться в різних підмережах. За допомогою певних налаштувань маршрутизатора і комутатора(ів) хости різних підмереж мають бути доступні один для одного.

Побудова мережі в емуляторі Mininet починається з топології. Топологія визначає, скільки буде хостів, комутаторів, а також яким чином вони будуть з'єднані в мережі.

Основна функціональність Mininet реалізована на Python, лише деякі утиліти написані на мові C. Практично будь-яка довільна топологія може бути описана за допомогою спеціального синтаксису на Python.

Mininet підтримує простий API на Python для створення власних топологій мережі, і ми можемо створювати власні топології, написавши код на Python, який наведено нижче разом із його функціями.

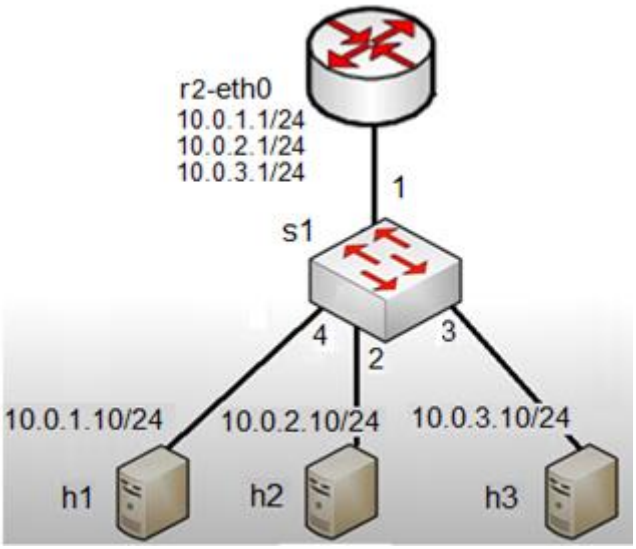
Щоб створити топологію, спочатку запускаємо MiniEdit. Щоб запустити MiniEdit: виконуємо команду:

```
mininet@mininet-vm:~$ cd mininet/examples
mininet@mininet-vm:~/mininet/examples$ sudo python ./miniedit.py
```

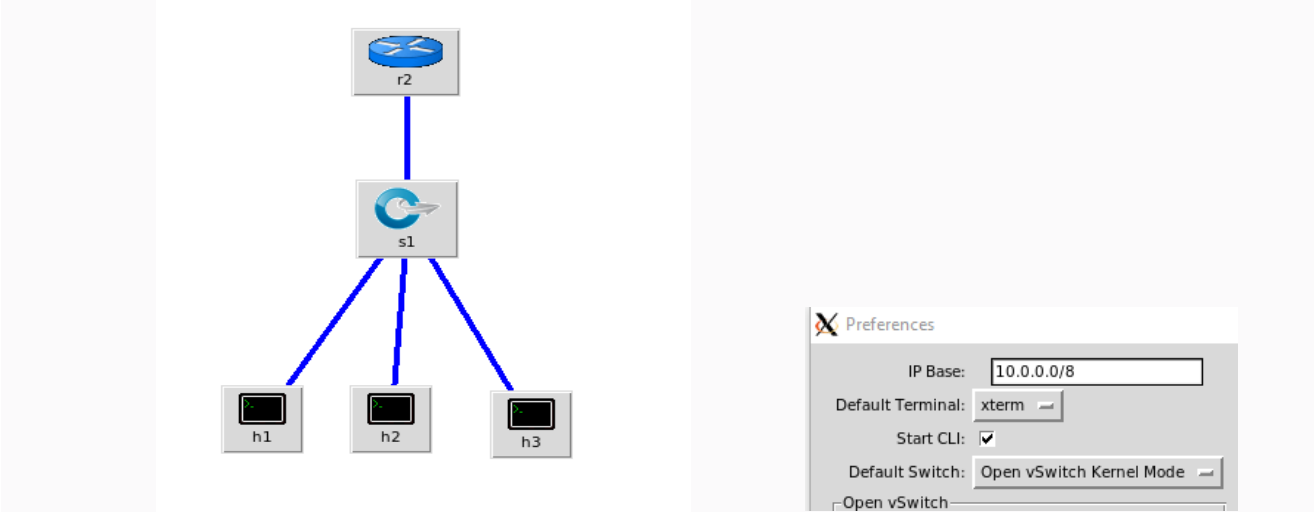
**Важливо**. Для виконання лабораторної роботи потрібно, щоб був запущений сервер **Xming.** Програму Putty слід запускати з SSH-тунелем (див. пояснення на сторінці 71).

**Варіант А**

У цьому завданні будемо використовувати MiniEdit на віртуальній машині Mininet для створення скрипта другого рівня на Python, який будує три хости, комутатор та маршрутизатор LegacyRouter.



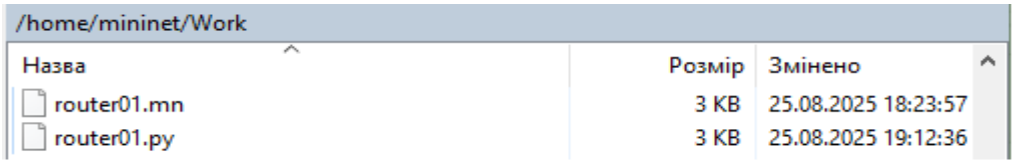

У цьому варіанті до одного комутатора *openvswitch* під'єднуються хости з різних підмереж, а саме 10.0.1.0/24, 10.0.2.0/24 і 10.0.3.0/24. Для налаштування доступності хостів, що знаходяться в різних підмережах, застосовується маршрутизатор r1. Зверніть увагу, що одному інтерфейсу маршрутизатора потрібно призначити кілька різних IP-адрес.




Встановлюємо параметр Start CLI, щоб в режимі симуляції можна було користуватися інтерфейсом командного рядка.

Для зберігання файла топології (.mn) і файла конфігурації рекомендується в каталозі home/mininet створити робочий каталог, наприклад Work. Це можна виконати, наприклад, за допомогою утиліти WinSCP.

Ми використали MiniEdit для створення початкової топології мережі.



Тепер потрібно відкрити збережений файл *router01.py* і ознайомитися із його структурою.

```
#!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    info( '*** Add switches\n')
    r2 = net.addHost('r2', cls=Node, ip='0.0.0.0')
    r2.cmd('sysctl -w net.ipv4.ip_forward=1')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(r2, s1)
    net.addLink(s1, h2)
    net.addLink(s1, h3)
    net.addLink(h1, s1)
```

Коротко розглянемо структуру цього файлу.

- Перші дев’ять рядків імпортують потрібні модулі на Python.
- Рядок коду net = Mininet( topo=None, build=False, ipBase='10.0.0.0/8') ініціалізує об'єкт Mininet із певними конфігураціями.
- Потім описуються вузли, які будуть використовуватися в цій мережі.
- Після того, як створюються вузли, створюються з'єднання між вузлами мережі.

Важливі класи, методи, функції та змінні у наведеному коді включають:

- topo: базовий клас для топологій Mininet.
- addSwitch(): додає комутатор до топології та повертає ім'я комутатора.
- addHost(): додає хост до топології та повертає ім'я хоста.
- addLink(): додає двонапрямне з'єднання до топології. З'єднання в Mininet є двонапрямними, якщо не зазначено інше.

- mininet: основний клас для створення та управління мережею.
- start(): запускає мережу.

Потрібно змінити скрипт другого рівня таким чином, щоб хости могли «пінгувати» один одного. Це вимагає розуміння адресації підмережі, функцій маршрутизатора та конфігурації статичних маршрутів. Маємо дотримуватися певних вимог цього завдання:

а. Хост h1 та інтерфейс маршрутизатора r2, підключений до s1, розташовані в мережі з приватною IPv4-адресою 10.0.1.0/24.
б. Хост h2 та інтерфейс маршрутизатора r2, підключений до s1, розташовані в мережі з приватною IPv4-адресою 10.0.2.0/24.
в. Хост h3 та інтерфейс маршрутизатора r2, підключений до s1, розташовані у мережі з приватною IPv4-адресою 10.0.3.0/24.

Отже внесемо необхідні зміни в файл *router01.py*. Для цього скористаємося редактором утиліти WinSCP.

Змінимо налаштування хостів. Виконаємо налаштування маршрутизатора. Створимо потрібні потоки на комутаторі.

1. Вводимо нові MAC-адреси та IP-адреси в код:

```
h1 : ip="10.0.1.10/24", mac="00:00:00:00:00:01"
h2 : ip="10.0.2.10/24", mac="00:00:00:00:00:02"
h3 : ip="10.0.3.10/24", mac="00:00:00:00:00:03"
```

2. Вводимо нове значення IP-адрес інтерфейсу маршрутизатора. Оскільки маршрутизатору вже призначена випадкова IP-адреса, її потрібно видалити. Це виконується за допомогою команди

```
r2.cmd("ifconfig r2-eth0 0")
```

Тепер призначаємо MAC-адресу і IP-адреси інтерфейсу маршрутизатора.

```
r2 = net.addHost( 'r2', mac="00:00:00:00:01:00" )
r2.cmd("ip addr add 10.0.1.1/24 brd + dev r2-eth0")
r2.cmd("ip addr add 10.0.2.1/24 brd + dev r2-eth0")
r2.cmd("ip addr add 10.0.3.1/24 brd + dev r2-eth0")
r2.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")
```

Команда ip addr add 10.0.1.1/24 brd + dev r2-eth0 додає адресу 10.0.1.1/24 з маскою підмережі 255.255.255.0 із стандартною широкомовною адресою і ім’ям r2-eth0.

Команда *echo 1 > /proc/sys/net/ipv4/ip_forward* дозволяє переадресацію IPv4 у системі Linux, дозволяючи хосту діяти як маршрутизатор, пересилаючи IP-пакети між

мережами. Використанням цієї команди встановлюємо прапорець у файлі переадресації IP як '1', щоб увімкнути переадресацію на маршрутизаторі.

3. Наступним кроком є додавання маршрутів за замовчуванням на хостах. Для цього використовується команда з таким синтаксисом:

```
ip route add default via <IP-адреса шлюзу>
```

Отже, маршрути за замовчуванням на хостах h1, h2 та h3 будуть такими:

```
h1.cmd("ip route add default via 10.0.1.1")
h2.cmd("ip route add default via 10.0.2.1")
h3.cmd("ip route add default via 10.0.3.1")
```

4. Останнім кроком буде додавання маршрутів потоків на комутаторі S1, щоб хости та маршрутизатор могли взаємодіяти один з одним. На комутаторі має бути створено 5 потоків.

```
s1.cmd("ovs-ofctl add-flow s1 priority=1,arp,actions=flood")
s1.cmd("ovs-ofctl add-flow s1 priority=65535,ip,dl_dst=00:00:00:00:01:00,actions=output:1")
s1.cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=10.0.1.0/24,actions=output:4")
s1.cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=10.0.2.0/24,actions=output:2")
s1.cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=10.0.3.0/24,actions=output:3")
```

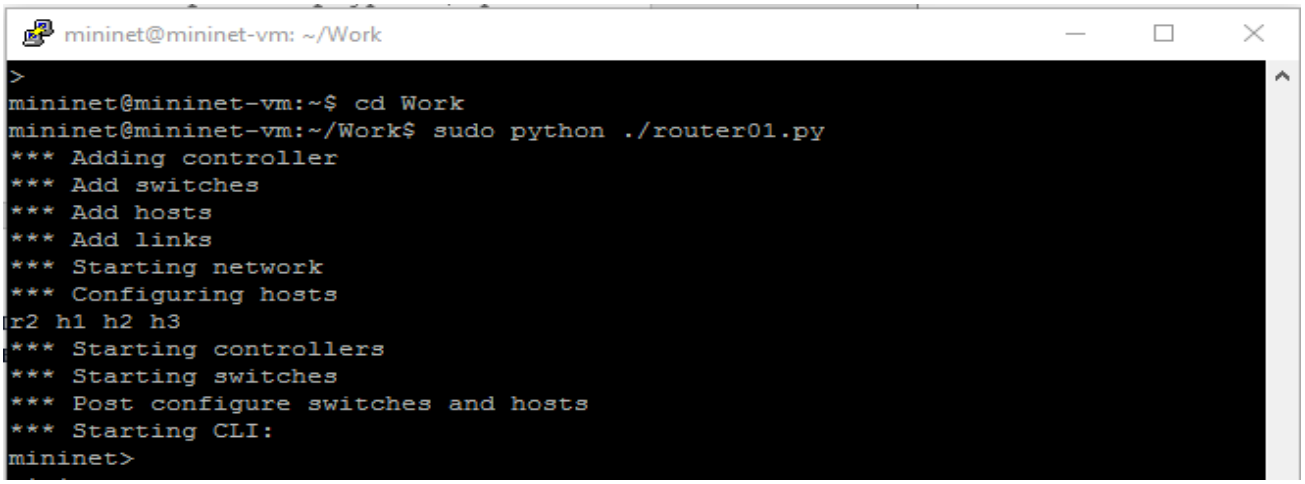
Після виконання усіх змін маємо отримати такий код:

```
#!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    info( '*** Add switches\n')
    r2 = net.addHost( 'r2', mac="00:00:00:00:01:00" )
    r2.cmd('sysctl -w net.ipv4.ip_forward=1')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h1 = net.addHost( 'h1', ip="10.0.1.10/24", mac="00:00:00:00:00:01" )
```

```
    h2 = net.addHost( 'h2', ip="10.0.2.10/24", mac="00:00:00:00:00:02" )

    h3 = net.addHost( 'h3', ip="10.0.3.10/24", mac="00:00:00:00:00:03" )

    info( '*** Add links\n')
    net.addLink(r2, s1)
    net.addLink(s1, h2)
    net.addLink(s1, h3)
    net.addLink(h1, s1)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([])

    info( '*** Post configure switches and hosts\n')

    r2.cmd("ifconfig r2-eth0 0")

    r2.cmd("ip addr add 10.0.1.1/24 brd + dev r2-eth0")

    r2.cmd("ip addr add 10.0.2.1/24 brd + dev r2-eth0")

    r2.cmd("ip addr add 10.0.3.1/24 brd + dev r2-eth0")

    r2.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")

    h1.cmd("ip route add default via 10.0.1.1")

    h2.cmd("ip route add default via 10.0.2.1")

    h3.cmd("ip route add default via 10.0.3.1")

    s1.cmd("ovs-ofctl add-flow s1 priority=1,arp,actions=flood")

    s1.cmd("ovs-ofctl add-flow s1
priority=65535,ip,dl_dst=00:00:00:00:01:00,actions=output:1")

    s1.cmd("ovs-ofctl add-flow s1
priority=10,ip,nw_dst=10.0.1.0/24,actions=output:4")

    s1.cmd("ovs-ofctl add-flow s1
priority=10,ip,nw_dst=10.0.2.0/24,actions=output:2")

    s1.cmd("ovs-ofctl add-flow s1
priority=10,ip,nw_dst=10.0.3.0/24,actions=output:3")


    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

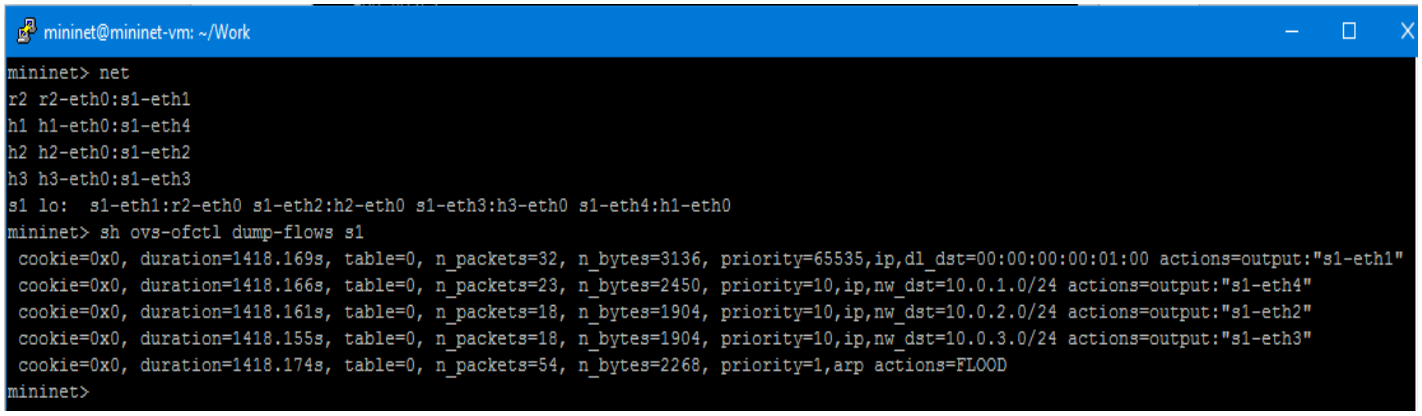
5. Копіюємо відредагований код скрипта у новий файл *router01.py* і зберігаємо файл в каталог Work.
6. Запускаємо збережений новий скрипт Python на віддаленому терміналі Putty:

mininet@mininet-vm:~$ **cd Work**
mininet@mininet-vm:~/Work$ **sudo python ./router01.py**




7. Для перевірки налаштувань виконаємо дві команди:

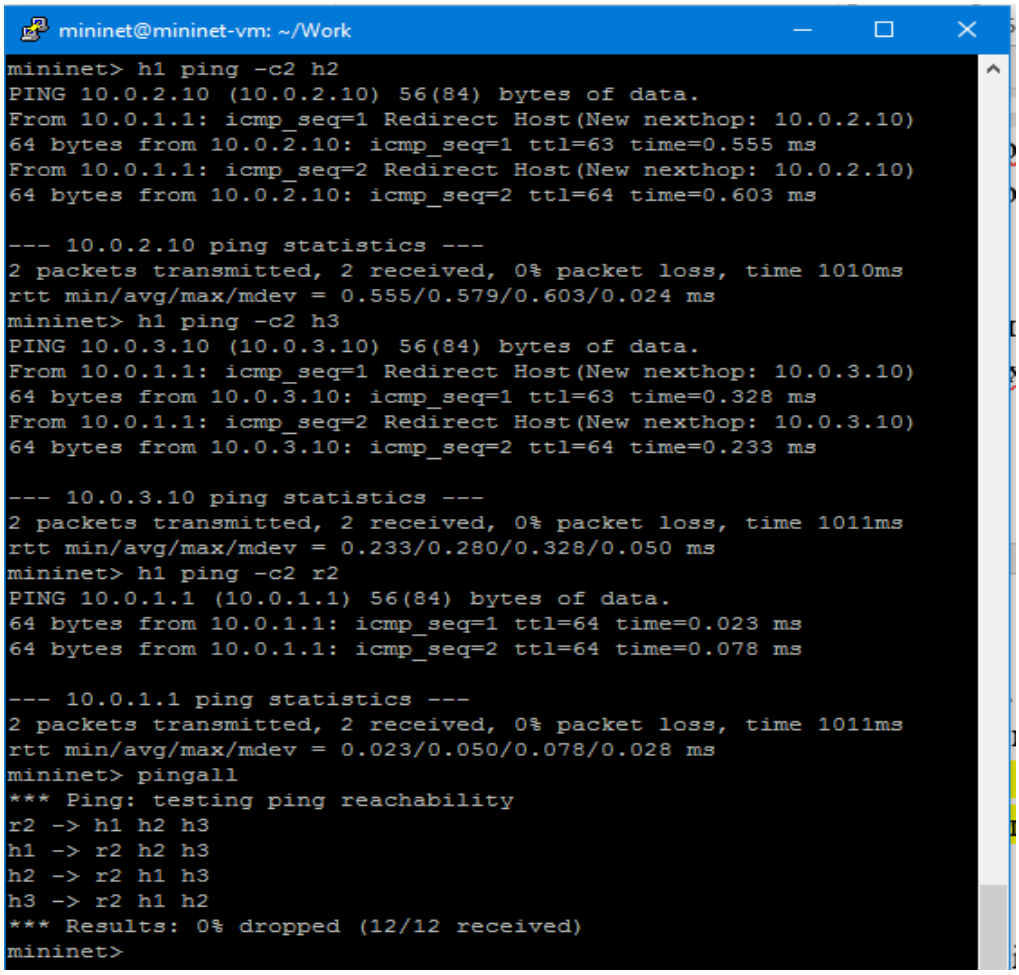
mininet> **net**
mininet> **sh ovs-ofctl dump-flows s1**



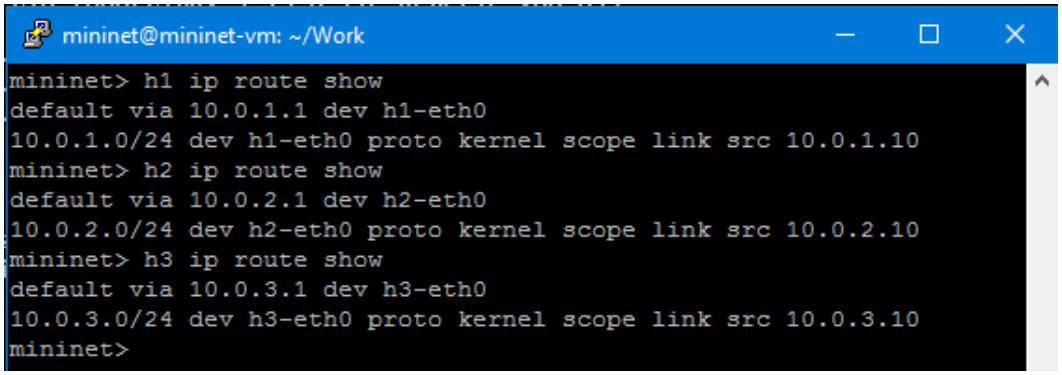

Аналізуючи результат виконання команди *net* бачимо, що до маршрутизатора r2 під'єднаний порт s1-eth1 комутатора, хост h1 під'єднаний до порту s1-eth4 комутатора, хост h2 під'єднаний до порту s1-eth2 комутатора, хост h3 під'єднаний до порту s1-eth3 комутатора.

Тепер поглянемо на результат виконання другої команди. Бачимо, що налаштування потоків виконане правильно. Пакети від хостів направляються на порт s1-eth1. Пакети для хоста h1 направляються на порт s1-eth4, пакети для хоста h2 направляються на порт s1-eth2, пакети для хоста h3 направляються на порт s1-eth3.

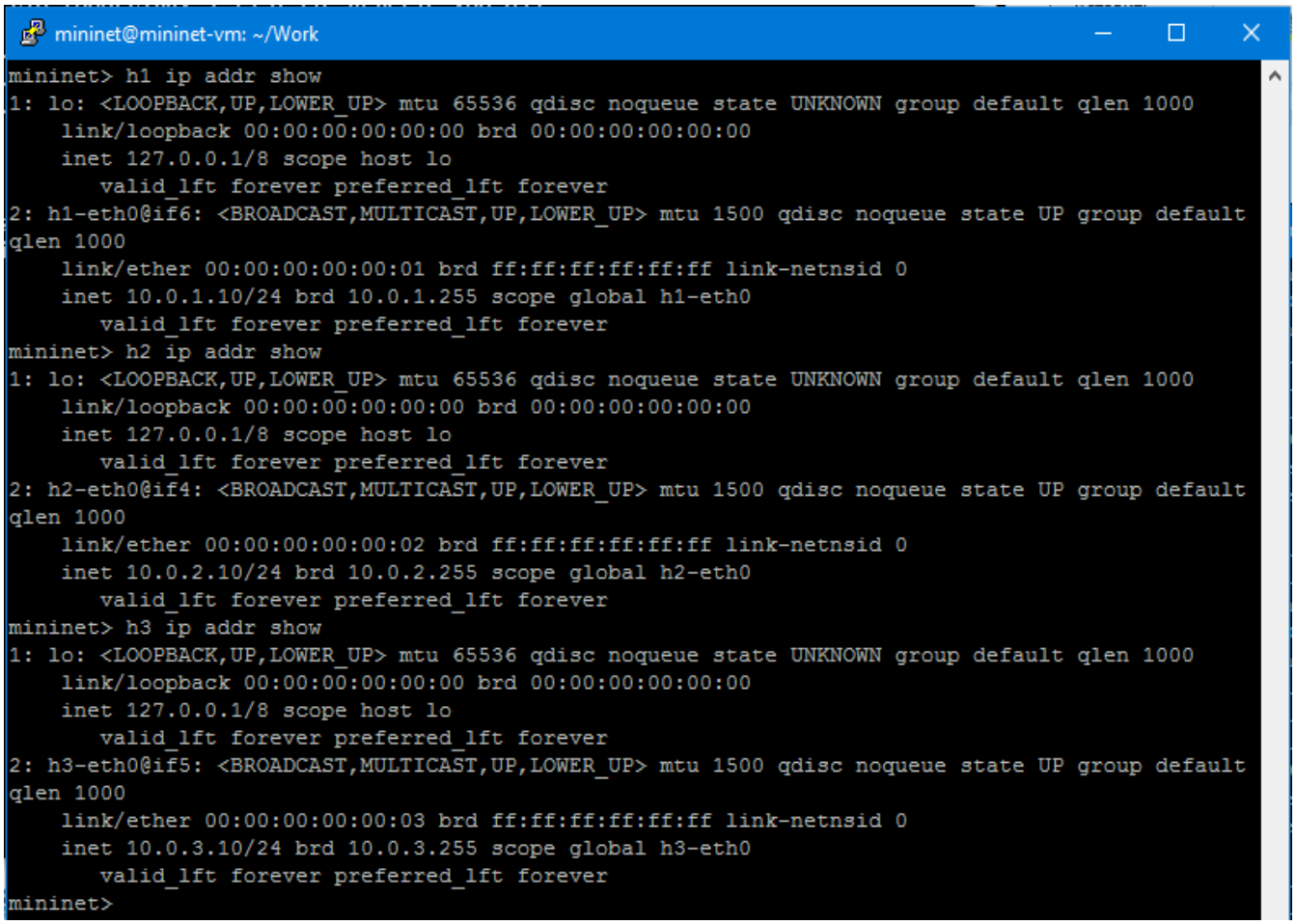
8. Перевіряємо наявність з'єднання між хостами, які розташовані в різних підмережах.



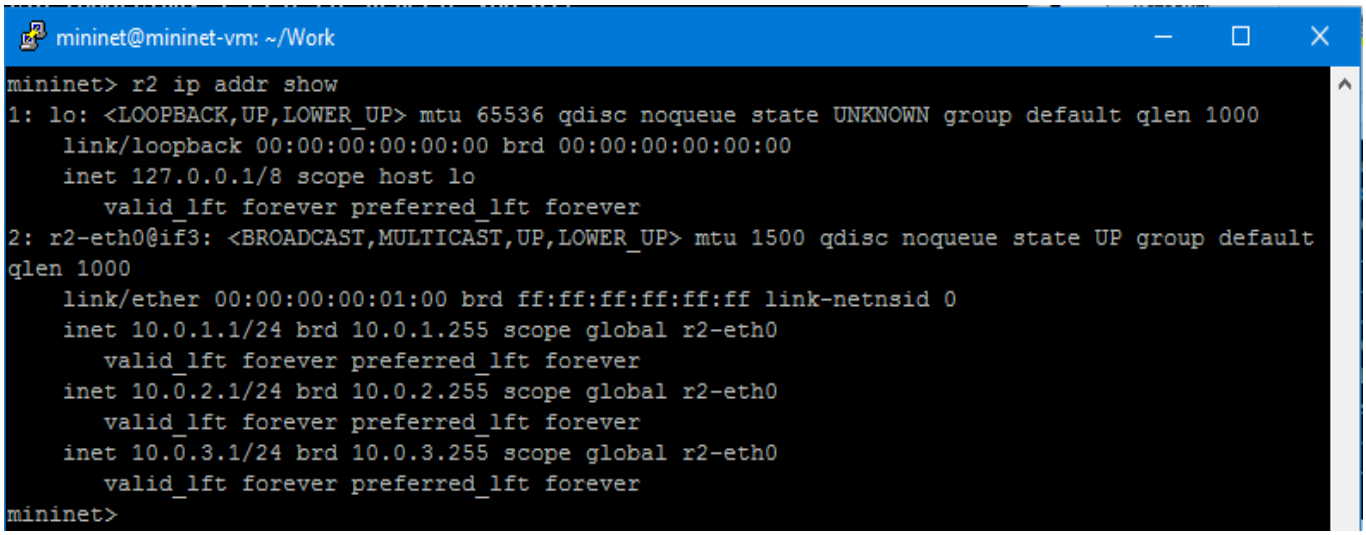

9. Переглянемо таблицю маршрутизації на хостах h1, h2 та h3. Бачимо, що записи маршрутизації для хостів визначають маршрути за замовчуванням.




10. Переглянемо налаштування інтерфейсів хостів.




11. Переглянемо інформацію про мережеві інтерфейси маршрутизатора. У виводі перераховані всі мережеві інтерфейси з призначеними їм IP-адресами і MAC-адресою.




12. Переглянемо проходження пакетів через інтерфейс комутатора s1-eth1 за допомогою програми Wireshark. Для того, щоб Wireshark використовував окреме вікно, з консолі віртуальної машини запускаємо утиліту **startx**:

```
$ startx
```

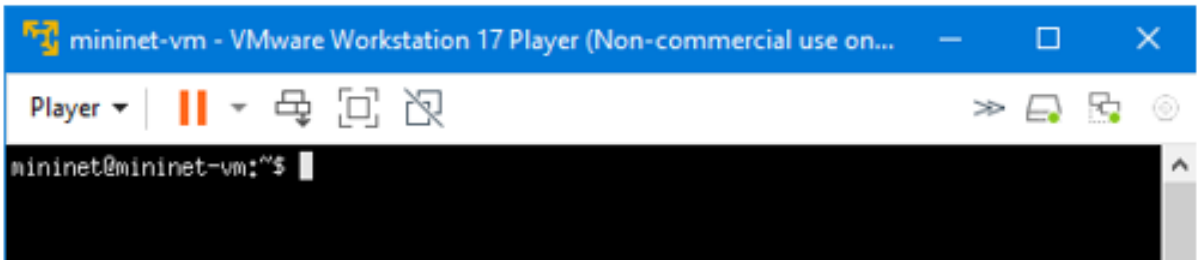
Утиліта *startx* – це сценарій, який використовується в операційних системах на основі Linux та інших систем, подібних до Unix, для запуску середовища робочого столу (X-Server) з командного рядка для  ініціалізації графічного середовища для користувача.

Якщо при спробі запустити графічне середовище з командного рядка з'являється помилка «команда startx не знайдена», то потрібно на віртуальну машину додатково встановити пакет **xinit**:

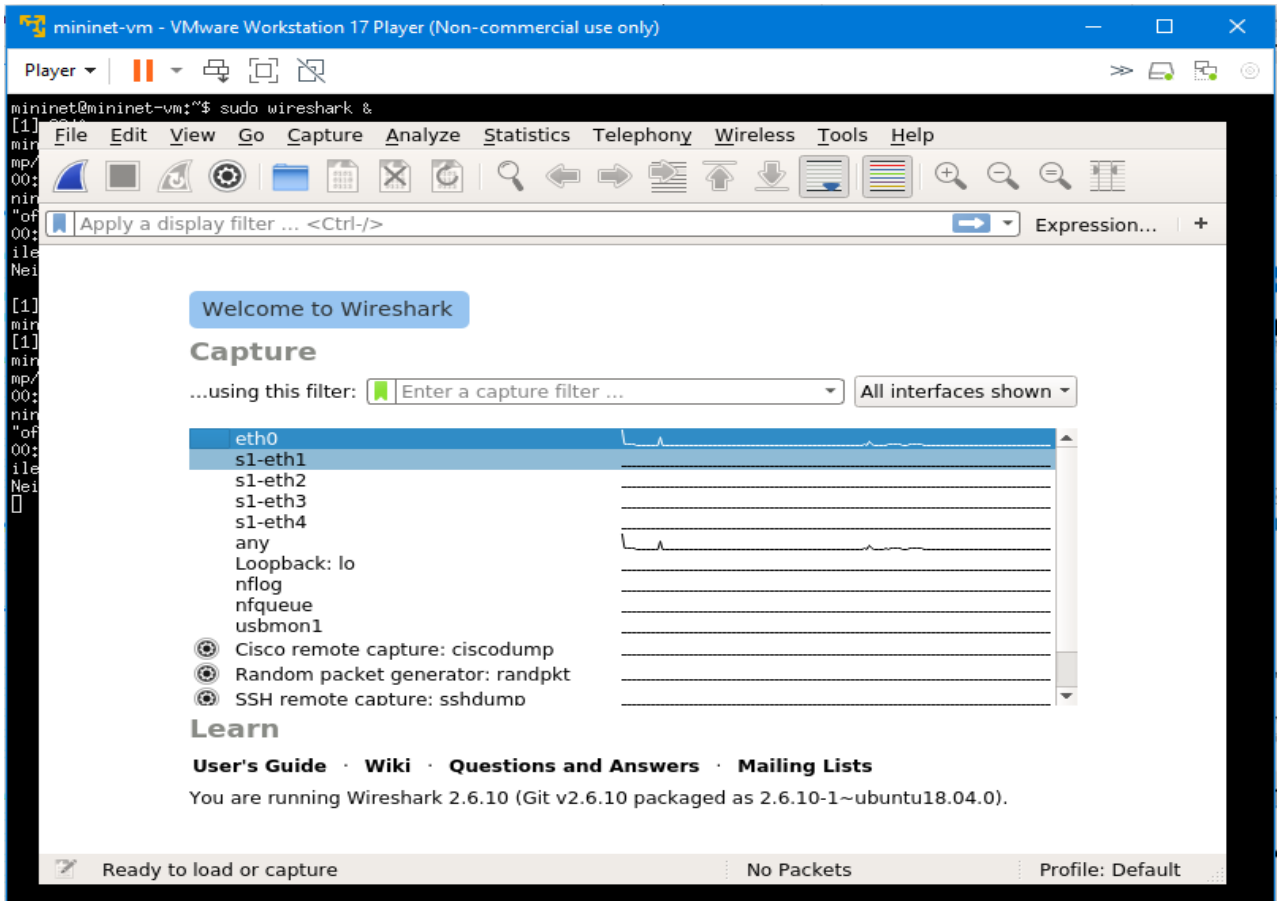
```
$ sudo apt update
$ sudo apt install xinit
```

Після встановлення *xinit* знову запускаємо *startx*. З'являється нове вікно. У цьому вікні запускаємо Wireshark у фоновому режимі.

```
$ sudo wireshark &
```




В списку доступних інтерфейсів вибираємо інтерфейс *s1-eth1* для захоплення трафіку (подвійний клік).

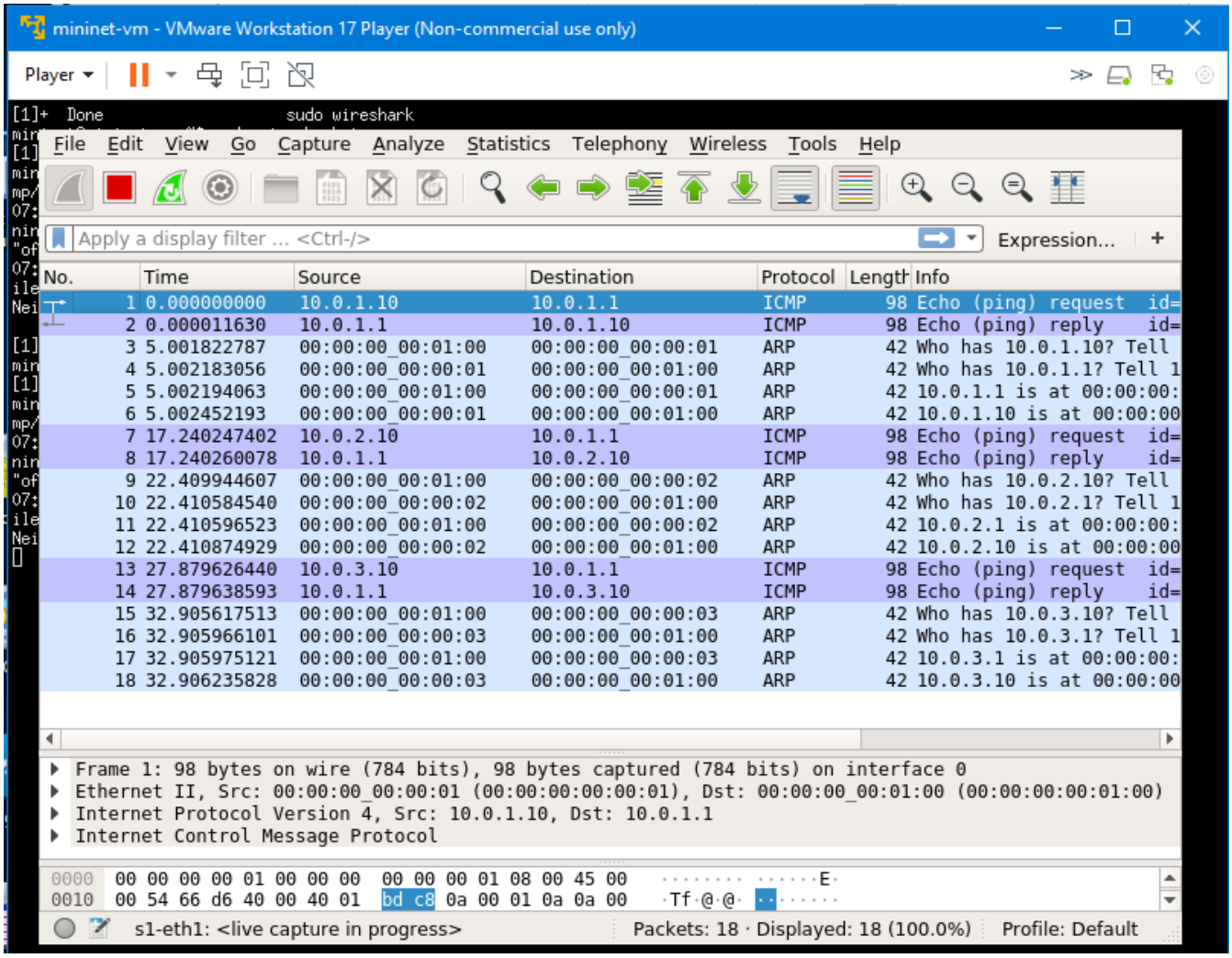



Переходимо на консоль віртуальної машини Mininet і перевіряємо наявність з'єднанн між хостами h1, h2, h3 та маршрутизатором r2.

```
mininet> h1 ping -c1 r2
mininet> h2 ping -c1 r2
mininet> h3 ping -c1 r2
```

```
mininet@mininet-vm: ~/Work
mininet> h1 ping -c1 r2
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.173 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.173/0.173/0.173/0.000 ms
mininet> h1 ping -c1 r2
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.192 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.192/0.192/0.192/0.000 ms
mininet> h2 ping -c1 r2
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.168 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.168/0.168/0.168/0.000 ms
mininet> h3 ping -c1 r2
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.151 ms

--- 10.0.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.151/0.151/0.151/0.000 ms
mininet>
```

У вікні, де запущений Wireshark, спостерігаємо проходження пакетів ICMP через інтерфейс комутатора s1-eth1 в прямому і зворотному напрямках.

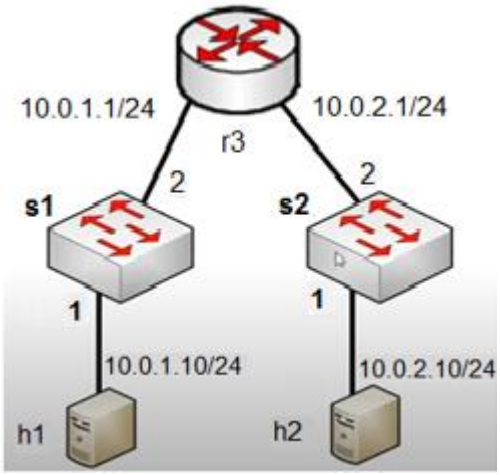


13. Завершуємо роботу програми Wireshark: *File – Quit*. Закриваємо вікно, в якому запускався Wireshark:

```
$ exit
```

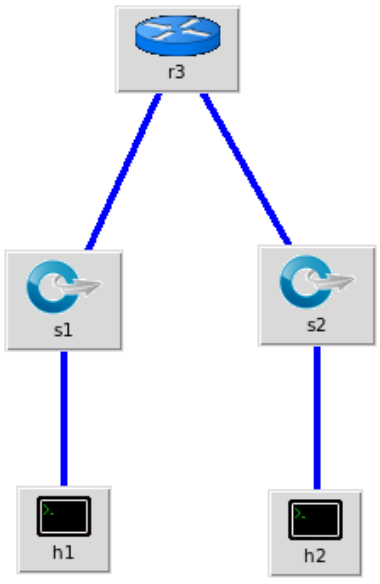
**Варіант Б**

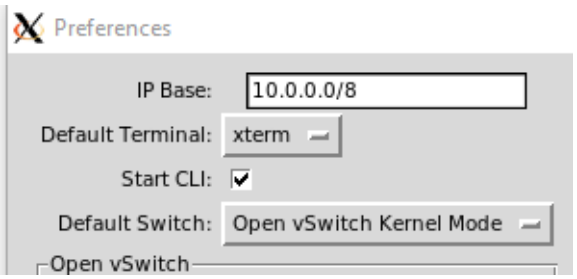
У цьому завданні, будемо використовувати MiniEdit на віртуальній машині Mininet для створення скрипта другого рівня на Python, який будує два хости, два комутатори, які з'єднані одним маршрутизатором LegacyRouter.




У цьому завданні є дві підмережі, тобто 10.0.1.0/24 і 10.0.2.0/24. Це підмережі, з'єднані одним маршрутизатором r3. У кожній підмережі є один комутатор *openvswitch*. Потрібно, щоб h1 міг спілкуватися з h2.

Підготуємо графічну топологію та створимо відповідний скрипт.



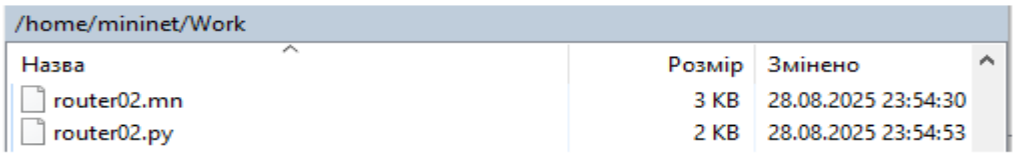




Встановлюємо параметр Start CLI, щоб в режимі симуляції можна було користуватися інтерфейсом командного рядка.

Після створення топології в MiniEdit зберігаємо файл топології (.mn) і файл конфігурації (скрипт Python) з ім’ям router02.py, наприклад, в каталог /home/mininet/Work.

Для зберігання файлів топології і конфігурації рекомендується в каталозі home/mininet створити робочий каталог, наприклад Work. Це можна виконати, наприклад, за допомогою утиліти WinSCP.

Зберігаємо скрипт з ім'ям *router02.py* за адресою /home/mininet/Work.




Тепер потрібно відкрити збережений файл *router02.py* і ознайомитися із його структурою. Для цього скористаємося редактором утиліти WinSCP.

```
  #!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    info( '*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    r3 = net.addHost('r3', cls=Node, ip='0.0.0.0')
    r3.cmd('sysctl -w net.ipv4.ip_forward=1')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(s2, h2)
    net.addLink(s1, h1)
    net.addLink(r3, s1)
    net.addLink(r3, s2)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([])
```

```
    net.get('s2').start([])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Коротко розглянемо структуру цього файлу.

- Перші дев’ять рядків імпортують потрібні модулі в Python.
- Рядок коду net = Mininet( topo=None, build=False, ipBase='10.0.0.0/8') ініціалізує об'єкт Mininet із певними конфігураціями.
- Потім описуються вузли, які будуть використовуватися в цій мережі.
- Після того, як створюються вузли, створюються з'єднання між вузлами мережі.

Важливі класи, методи, функції та змінні у наведеному коді включають:

- topo: базовий клас для топологій Mininet.
- addSwitch(): додає комутатор до топології та повертає ім'я комутатора.
- addHost(): додає хост до топології та повертає ім'я хоста.
- addLink(): додає двонапрямне з'єднання до топології. З'єднання в Mininet є двонапрямними, якщо не зазначено інше.
- mininet: основний клас для створення та управління мережею.
- start(): запускає мережу.

Потрібно змінити скрипт другого рівня таким чином, щоб хости могли «пінгувати» один одного. Це вимагає розуміння адресації підмережі, функцій маршрутизатора та конфігурації статичних маршрутів. Маємо дотримуватися певних вимог цього завдання:

а. Хост h1 та інтерфейс маршрутизатора r3, підключений до s1, розташовані в мережі з приватною IPv4-адресою 10.0.1.0/24.
б. Хост h2 та інтерфейс маршрутизатора r3, підключений до s2, розташовані в мережі з приватною IPv4-адресою 10.0.2.0/24.

Отже внесемо необхідні зміни в файл *router02.py*.

Змінимо налаштування хостів. Виконаємо налаштування маршрутизатора. Створимо потрібні потоки на комутаторі.

1. Вводимо нові MAC-адреси та IP-адреси в код:

   h1 : ip="10.0.1.10/24", mac="00:00:00:00:00:01"

h2 : ip="10.0.2.10/24", mac="00:00:00:00:00:02"

2. Змінимо налаштування маршрутизатора та двох комутаторів:

```
r3 = net.addHost( 'r1')
s1 = net.addSwitch( 's1')
s2 = net.addSwitch( 's2')
```

3. Вводимо нові значення IP-адрес інтерфейсу маршрутизатора. Оскільки маршрутизатору вже призначена випадкова IP-адреса, її потрібно видалити. Це виконується за допомогою команди

```
r3.cmd("ifconfig r3-eth0 0")
r3.cmd("ifconfig r3-eth1 0")
```

Тепер призначаємо IP-адреси і MAC-адреси інтерфейсам маршрутизатора.

```
r3.cmd("ifconfig r3-eth0 hw ether 00:00:00:00:01:01")
r3.cmd("ifconfig r3-eth1 hw ether 00:00:00:00:01:02")
r3.cmd("ip addr add 10.0.1.1/24 brd + dev r3-eth0")
r3.cmd("ip addr add 10.0.2.1/24 brd + dev r3-eth1")
r3.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")
```

Команда ip addr add 10.0.1.1/24 brd + dev r3-eth0 додає адресу 10.0.1.1/24 з маскою підмережі 255.255.255.0 із стандартною широкомовною адресою і ім’ям r3-eth0.

Команда `echo 1 > /proc/sys/net/ipv4/ip_forward` дозволяє переадресацію IPv4 у системі Linux, дозволяючи хосту діяти як маршрутизатор, пересилаючи IP-пакети між мережами. Використанням цієї команди встановлюємо прапорець у файлі переадресації IP як '1', щоб увімкнути переадресацію на маршрутизаторі.

4. Наступним кроком є додавання маршрутів за замовчуванням на хостах. Для цього використовується наступний синтаксис:

```
ip route add default via <IP-адреса шлюзу>
```

Отже, маршрути за замовчуванням на хостах h1, h2 та h3 будуть такими:

```
h1.cmd("ip route add default via 10.0.1.1")
h2.cmd("ip route add default via 10.0.2.1")
```

5. Останнім кроком буде додавання маршрутів потоків на комутаторах s1 та s2, щоб хости та маршрутизатор могли взаємодіяти один з одним. У кожного комутатора має бути 3 потоки.

```
s1.cmd("ovs-ofctl add-flow s1 priority=1,arp,actions=flood")
s1.cmd("ovs-ofctl add-flow s1 priority=65535,ip,dl_dst=00:00:00:00:01:01,actions=output:2")
s1.cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=10.0.1.10/24,actions=output:1")
```

```
s2.cmd("ovs-ofctl add-flow s2 priority=1,arp,actions=flood")
s2.cmd("ovs-ofctl add-flow s2 priority=65535,ip,dl_dst=00:00:00:00:01:02,actions=output:2")
s2.cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_dst=10.0.2.10/24,actions=output:1")
```

- Перший потік має бути найменш пріоритетний, він буде потрібний, якщо MAC-адреса призначення невідома.
- Другий потік створюється для того, щоб комутатор знав, куди слід надсилати пакети, отримані від хостів. Наприклад, пакет для MAC-адреси 00:00:00:00:01:01 буде переслано через порт Ethernet 2.
- Наступний потік допомагає комутатору доставляти пакети до відповідного хоста залежно від IP-адреси призначення в пакеті. Наприклад, пакети для підмережі 10.0.1.0 будуть адресовані через порт Ethernet 1.

Після виконання усіх змін маємо отримати такий код:

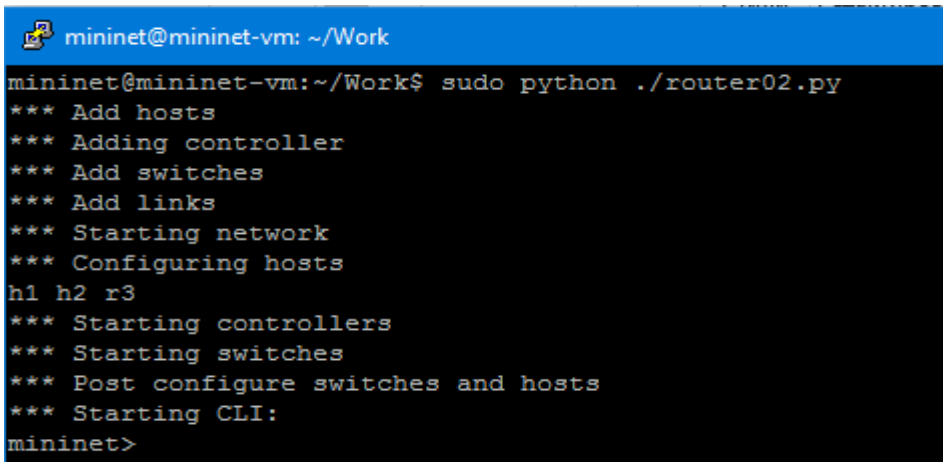
```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Add hosts\n')
    h1 = net.addHost('h1', ip="10.0.1.10/24", mac="00:00:00:00:00:01" )
    h2 = net.addHost('h2', ip="10.0.2.10/24", mac="00:00:00:00:00:02" )

    info( '*** Adding controller\n' )
    info( '*** Add switches\n')
    r3 = net.addHost('r3')
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')

    info( '*** Add links\n')
    net.addLink(s2, h2)
    net.addLink(s1, h1)
    net.addLink(r3, s1)
    net.addLink(r3, s2)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s1').start([])
```

```
    net.get('s2').start([])
    r3.cmd("ifconfig r3-eth0 0")
    r3.cmd("ifconfig r3-eth1 0")
    r3.cmd("ifconfig r3-eth0 hw ether 00:00:00:00:01:01")
    r3.cmd("ifconfig r3-eth1 hw ether 00:00:00:00:01:02")
    r3.cmd("ip addr add 10.0.1.1/24 brd + dev r3-eth0")
    r3.cmd("ip addr add 10.0.2.1/24 brd + dev r3-eth1")
    r3.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")
    h1.cmd("ip route add default via 10.0.1.1")
    h2.cmd("ip route add default via 10.0.2.1")
    s1.cmd("ovs-ofctl add-flow s1 priority=1,arp,actions=flood")
    s1.cmd("ovs-ofctl add-flow s1
priority=65535,ip,dl_dst=00:00:00:00:01:01,actions=output:2")
    s1.cmd("ovs-ofctl add-flow s1
priority=10,ip,nw_dst=10.0.1.10/24,actions=output:1")
    s2.cmd("ovs-ofctl add-flow s2 priority=1,arp,actions=flood")
    s2.cmd("ovs-ofctl add-flow s2
priority=65535,ip,dl_dst=00:00:00:00:01:02,actions=output:2")
    s2.cmd("ovs-ofctl add-flow s2
priority=10,ip,nw_dst=10.0.2.10/24,actions=output:1")
    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

6. Копіюємо відредагований код скрипта у новий файл *router02.py* і зберігаємо файл в каталог Work.

Запускаємо збережений новий скрипт Python користуючись віддаленим терміналом Putty:

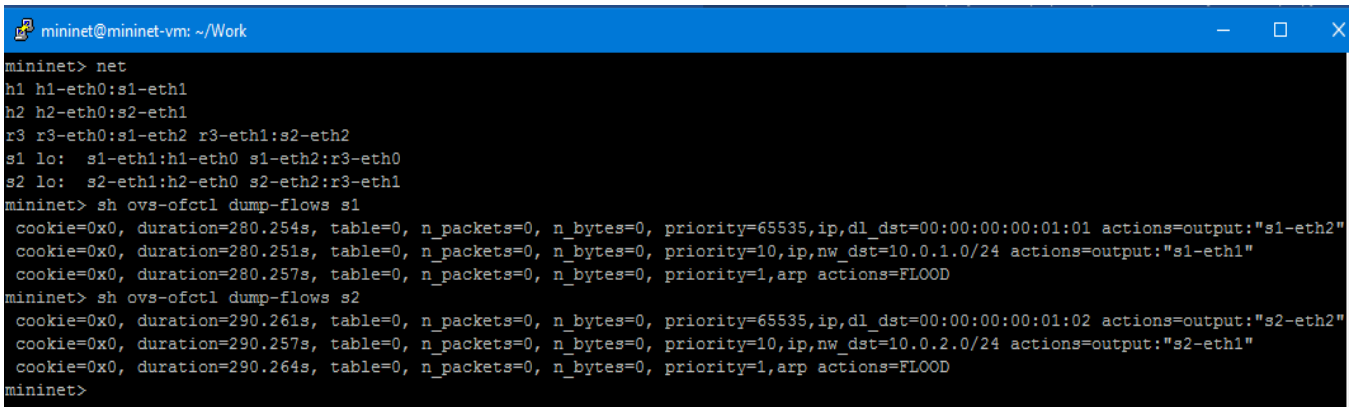
mininet@mininet-vm:~$ **cd Work**
mininet@mininet-vm:~/Work$ **sudo python ./router02.py**




7. Для перевірки налаштувань на комутаторах виконаємо три команди:

mininet> **net**

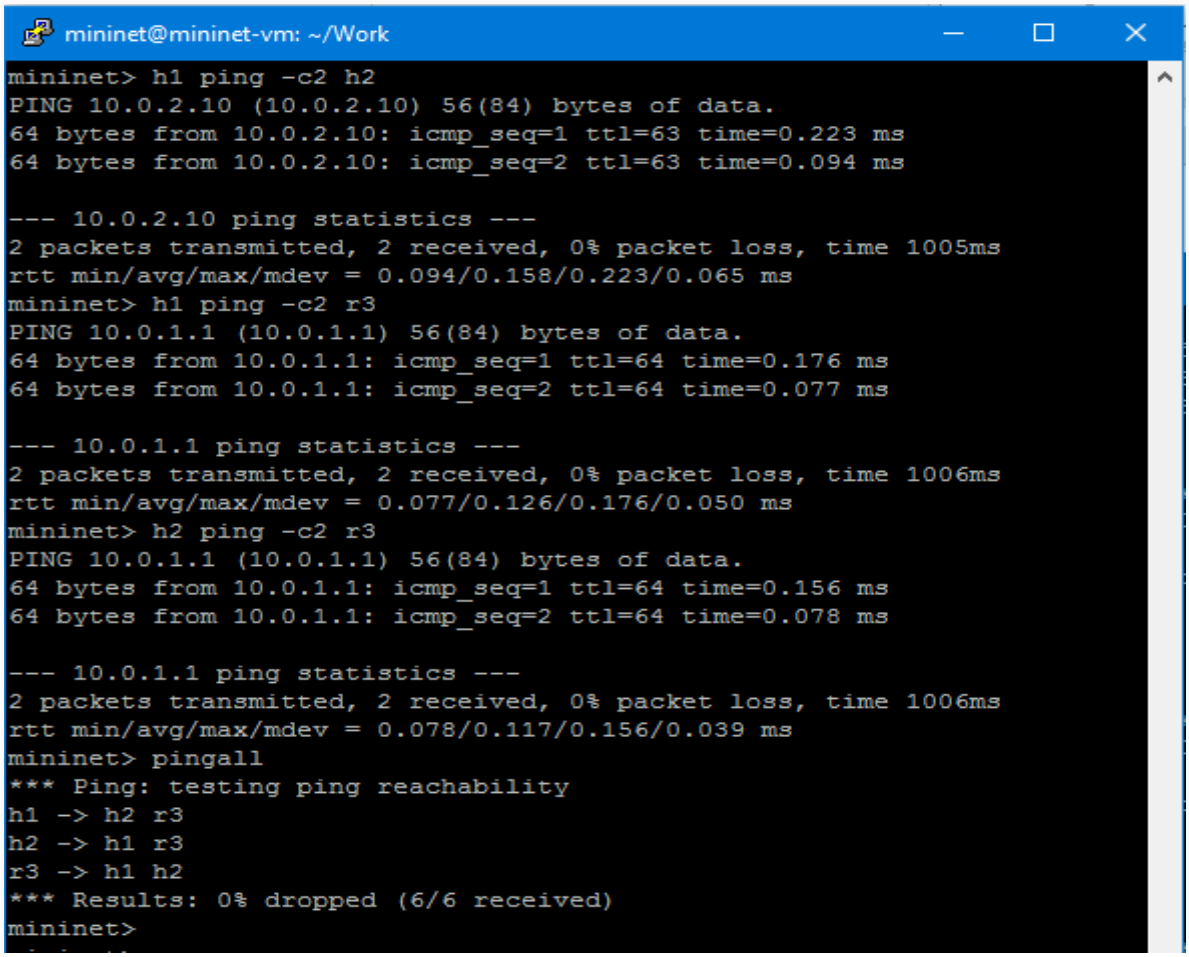
mininet> **sh ovs-ofctl dump-flows s1**
mininet> **sh ovs-ofctl dump-flows s2**



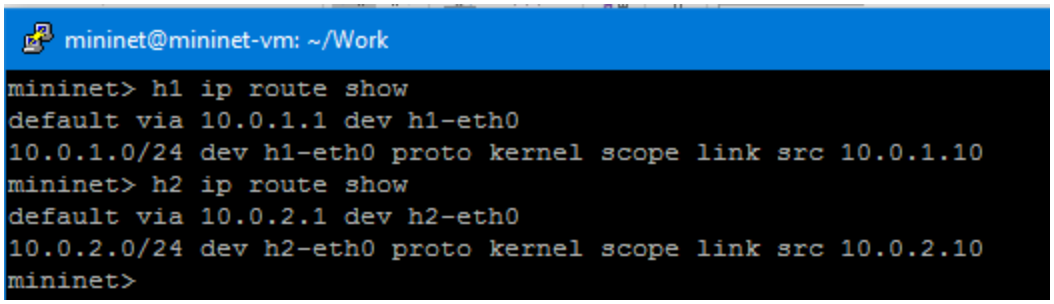

Аналізуючи вивід команди *net* бачимо, що до маршрутизатора r3 під'єднаний порт s1-eth2 комутатора s1 та порт s2-eth2 комутатора s2, хост h1 під'єднаний до порту s1-eth1 комутатора s1, хост h2 під'єднаний до порту s2-eth1 комутатора s2.

Тепер поглянемо на вивід другої та третьої команд. Бачимо, що налаштування потоків коректне. Пакети від хоста h1 направляються на порт s1-eth2, пакети для хоста h1 направляються на порт s1-eth1, пакети від хоста h2 направляються на порт s2-eth2, пакети для хоста h2 направляються на порт s2-eth1.

8. Перевіряємо наявність з'єднання між хостами, які розташовані в різних підмережах.



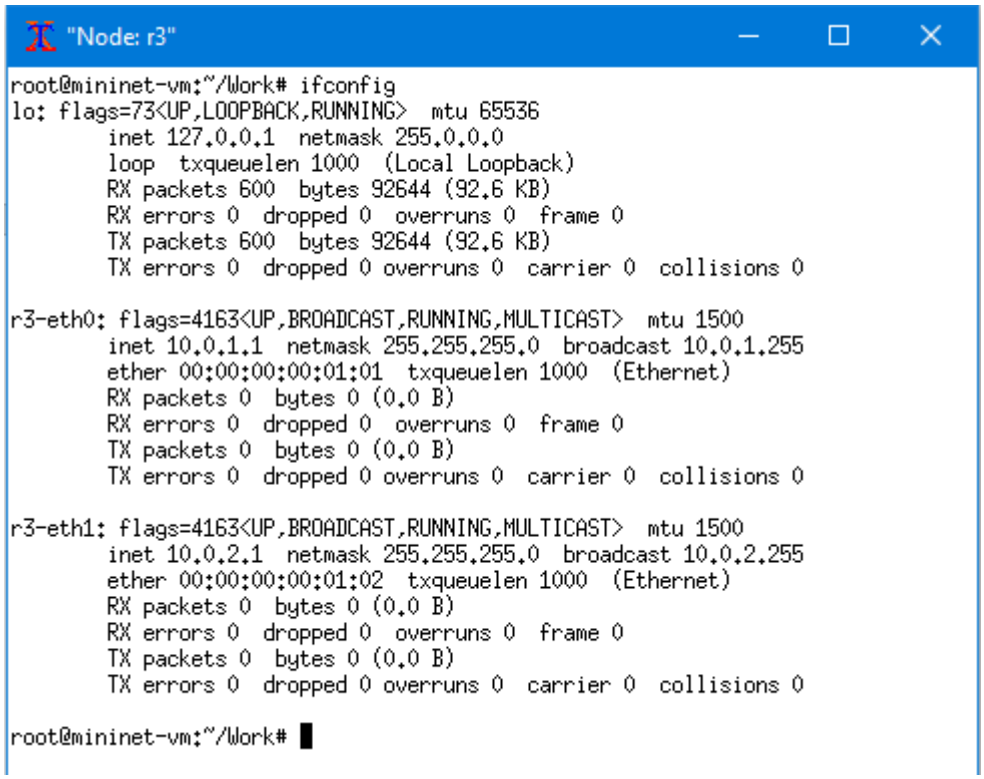
9. Перевіряємо налаштування маршрутів на хостах:




Отже, налаштування маршрутизації в мережі виконане вірно.

10. Переглянемо налаштування інтерфейсів на маршрутизаторі r3. Для цього створимо окремий термінал для r3 за допомогою команди **xterm** і виконаємо команду **ifconfig.**

mininet> **xterm r3**




11. Переглянемо проходження пакетів через інтерфейс комутатора s1-eth1 за допомогою програми Wireshark. Для того, щоб Wireshark використовував окреме вікно, з консолі віртуальної машини запускаємо утиліту **startx**:

$ **startx**

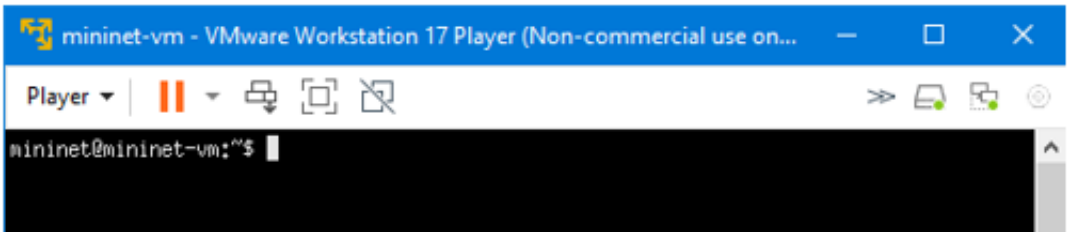
Утиліта *startx* – це сценарій, який використовується в операційних системах на основі Linux та інших систем, подібних до Unix, для запуску середовища робочого столу (X-Server) з командного рядка для ініціалізації графічного середовища для користувача.

Якщо при спробі запустити графічне середовище з командного рядка з’являється помилка «команда startx не знайдена», то потрібно на віртуальну машину додатково встановити пакет **xinit**:

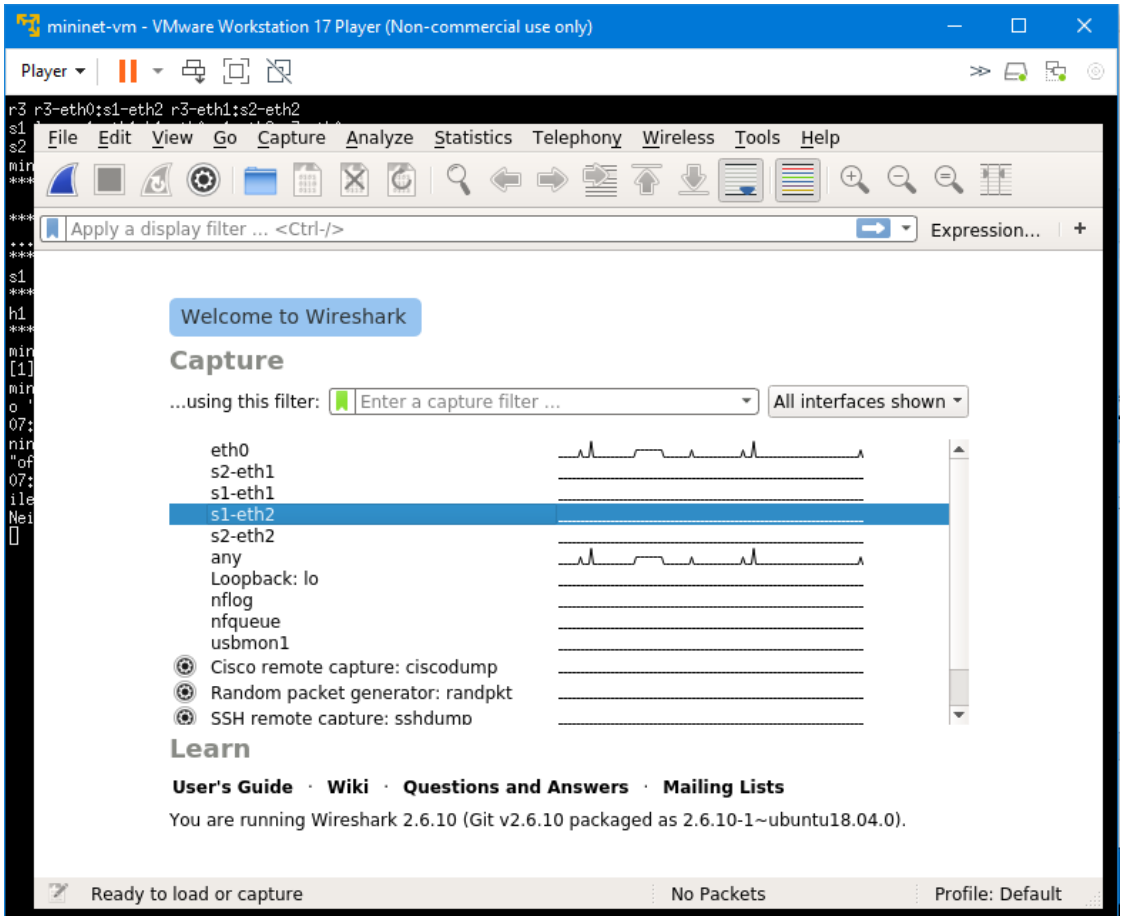
```
$ sudo apt update
$ sudo apt install xinit
```

12. Після встановлення *xinit* знову запускаємо *startx*. З'являється нове вікно. У цьому вікні запускаємо Wireshark.

```
$ sudo wireshark &
```

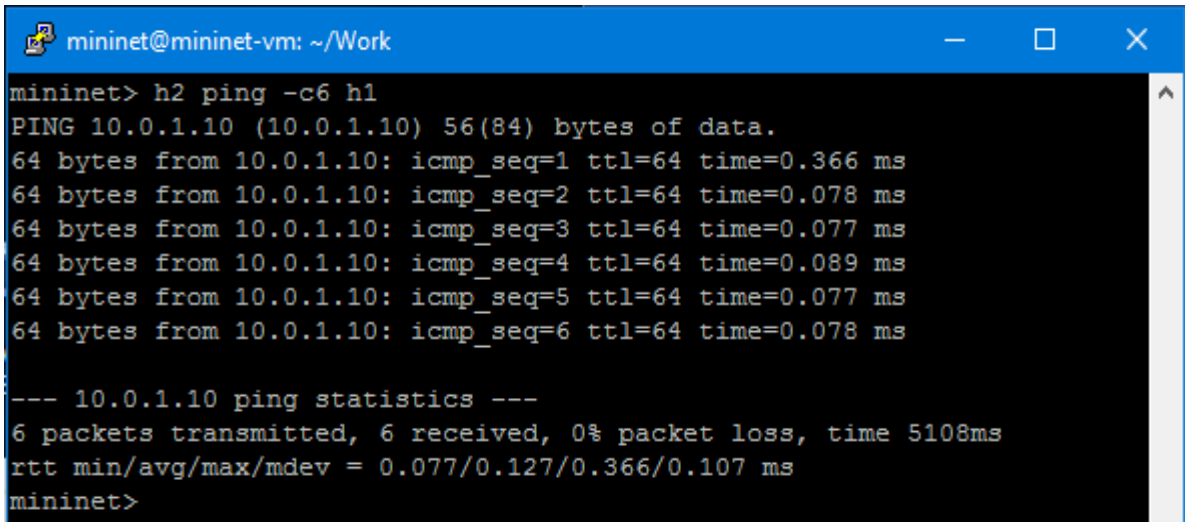



В списку доступних інтерфейсів вибираємо інтерфейс *s1-eth1* для захоплення трафіку (подвійний клік).

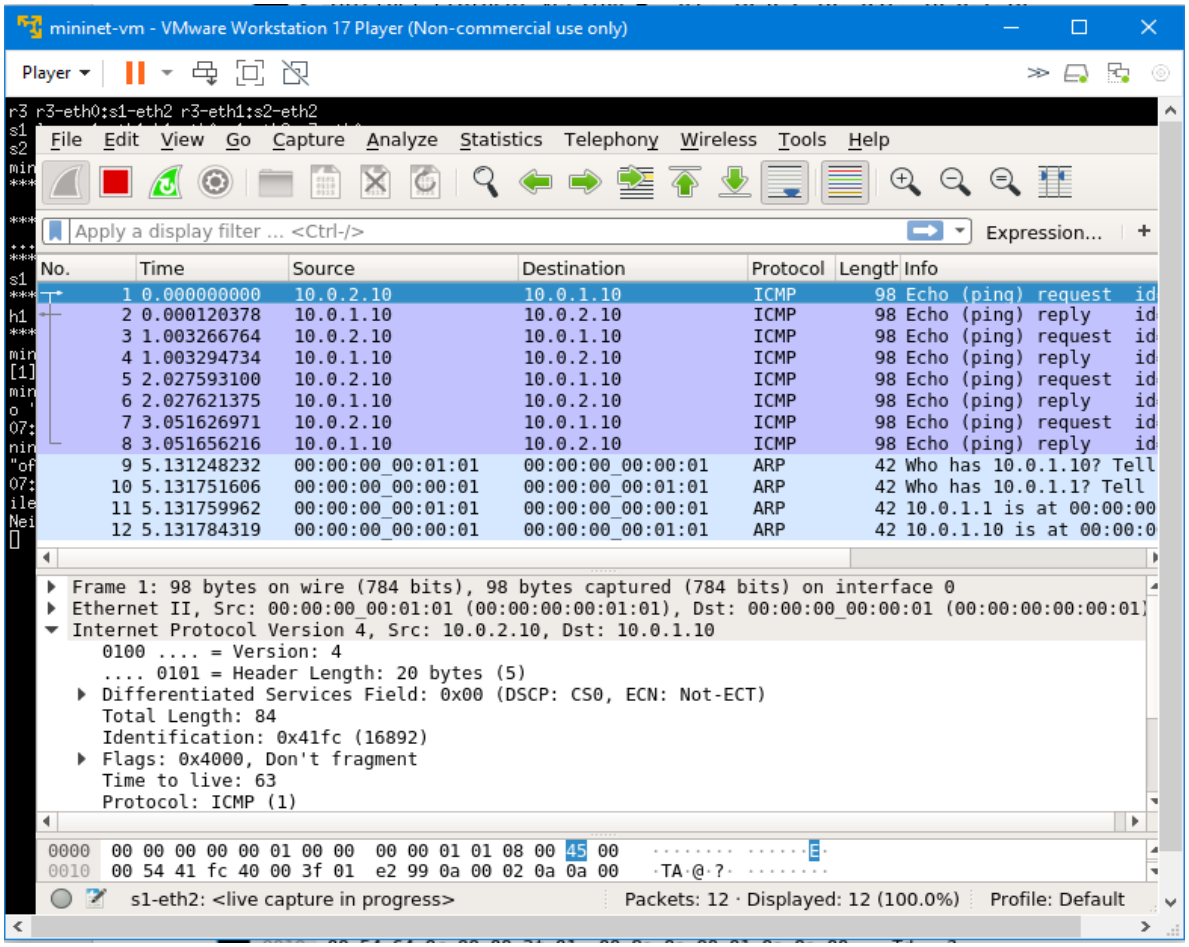



13. Переходимо на консоль віртуальної машини Mininet і перевіряємо наявність з'єднання між h2 і h1:

```
mininet>  h2 ping -c6 h1
```




У вікні, де запущений Wireshark спостерігаємо проходження пакетів ICMP через інтерфейс комутатора s1-eth1.

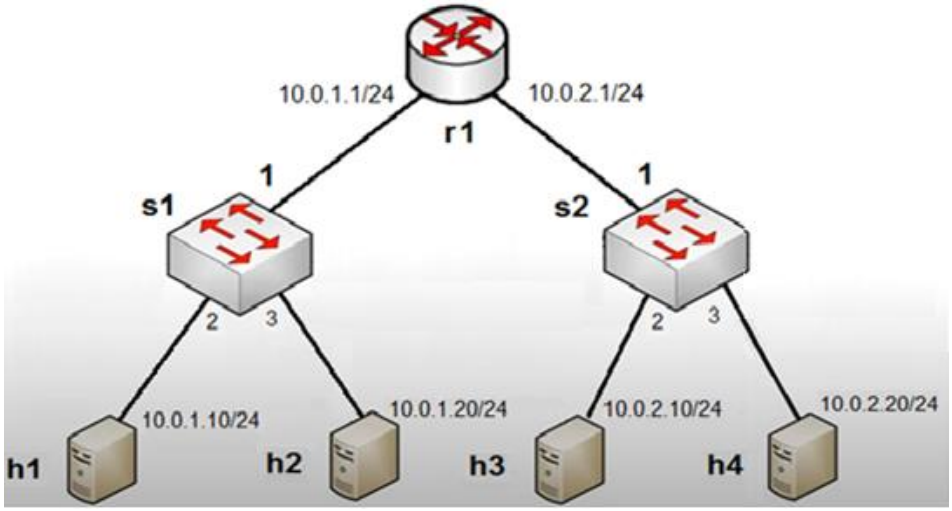



14. Завершуємо роботу програми Wireshark: *File – Quit*. Закриваємо вікно, в якому запускався Wireshark:

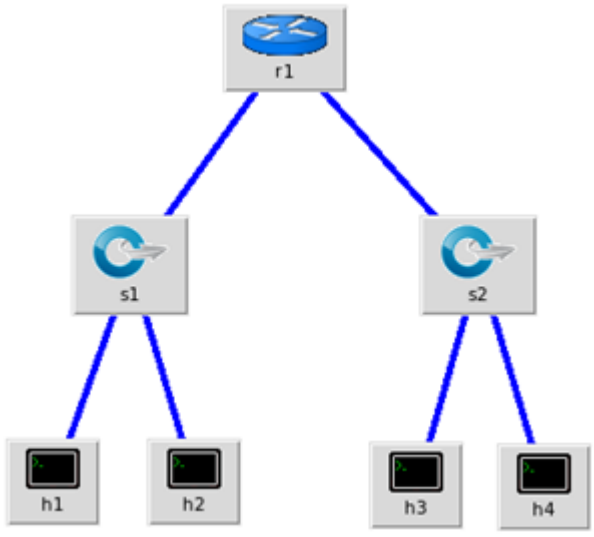
```
$ exit
```

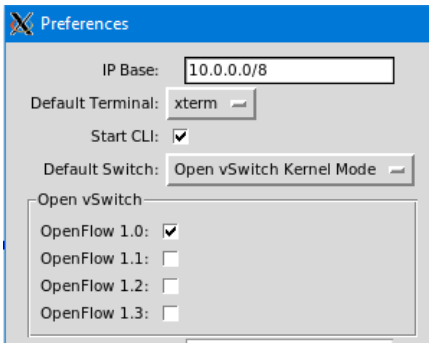
**Варіант В**

У третьому варіанті є дві підмережі, тобто 10.0.1.0/24 та 10.0.2.0/24. Це підмережі, які з'єднані одним маршрутизатором r1. У кожній підмережі є один комутатор *openvswitch*. Потрібно, щоб h1 міг спілкуватися з h2, h3 або h4.

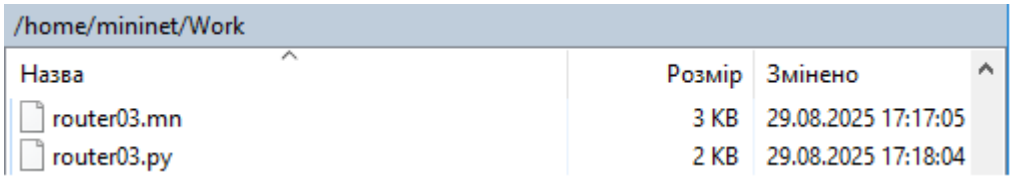



Підготуємо скрипт для створення топології з налаштованим маршрутизатором для виконання. Зберігаємо скрипт з ім'ям *router03.py* за адресою /home/mininet/Work.




Встановлюємо параметр Start CLI, щоб в режимі симуляції можна було користуватися інтерфейсом командного рядка.




Тепер потрібно відкрити збережений файл *router03.py* і ознайомитися із його структурою. Для цього скористаємося редактором утиліти WinSCP.

```
#!/usr/bin/env python

from mininet.net import Mininet
```

```
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    info( '*** Add switches\n')
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    r1 = net.addHost('r1', cls=Node, ip='0.0.0.0')
    r1.cmd('sysctl -w net.ipv4.ip_forward=1')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

    info( '*** Add hosts\n')
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
    h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)

    info( '*** Add links\n')
    net.addLink(h1, s1)
    net.addLink(h2, s1)
    net.addLink(h3, s2)
    net.addLink(h4, s2)
    net.addLink(s1, r1)
    net.addLink(s2, r1)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s2').start([])
    net.get('s1').start([])

    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Коротко розглянемо структуру цього файлу.

- Перші дев’ять рядків імпортують деякі модулі в Python.
- Рядок коду net = Mininet( topo=None, build=False, ipBase='10.0.0.0/8')ініціалізує об'єкт Mininet із певними конфігураціями.
- Потім описуються вузли, які будуть використовуватися в цій мережі.
- Після того, як створюються вузли, створюються з'єднання між вузлами мережі.

Важливі класи, методи, функції та змінні у наведеному коді включають:

- topo: базовий клас для топологій Mininet.
- addSwitch(): додає комутатор до топології та повертає ім'я комутатора.
- addHost(): додає хост до топології та повертає ім'я хоста.
- addLink(): додає двонапрямне з'єднання до топології. З'єднання в Mininet є двонапрямними, якщо не зазначено інше.
- mininet: основний клас для створення та управління мережею.
- start(): запускає мережу.

Потрібно змінити скрипт другого рівня таким чином, щоб хости могли «пінгувати» один одного. Це вимагає розуміння адресації підмережі, функцій маршрутизатора та конфігурації статичних маршрутів. Маємо дотримуватися певних вимог цього завдання:

а. Хости h1, h2 та інтерфейс маршрутизатора r1, підключений до s1, розташовані в мережі з приватною IPv4-адресою 10.0.1.0/24.
б. Хости h3, h4 та інтерфейс маршрутизатора r1, підключений до s2, розташовані в мережі з приватною IPv4-адресою 10.0.2.0/24.

Отже виконаємо необхідні зміни у файлі *router03.py*.

Змінимо налаштування хостів. Виконаємо налаштування маршрутизатора. Створимо потрібні потоки на комутаторі.

15. Вводимо нові MAC-адреси та IP-адреси в код:

```
h1 = net.addHost( 'h1', ip="10.0.1.10/24", mac="00:00:00:00:00:01" )
h2 = net.addHost( 'h2', ip="10.0.1.20/24", mac="00:00:00:00:00:02" )
h3 = net.addHost( 'h3', ip="10.0.2.10/24", mac="00:00:00:00:00:03" )
h4 = net.addHost( 'h4', ip="10.0.2.20/24", mac="00:00:00:00:00:04" )
```

16. Додаємо до топології маршрутизатор та два комутатори:

```
r1 = net.addHost( 'r1')
s1 = net.addSwitch( 's1')
s2 = net.addSwitch( 's2')
```

17. Вводимо нове значення IP-адрес інтерфейсу маршрутизатора. Оскільки інтерфейсам маршрутизатора вже призначені випадкові IP-адреси, їх потрібно видалити. Це виконується за допомогою команд

```
r3.cmd("ifconfig r1-eth0 0")
r3.cmd("ifconfig r1-eth1 0")
```

Тепер призначаємо нові значення IP-адрес і MAC-адрес інтерфейсам маршрутизатора.

```
r1.cmd("ifconfig r1-eth0 hw ether 00:00:00:00:01:01")
r1.cmd("ifconfig r1-eth1 hw ether 00:00:00:00:01:02")
r1.cmd("ip addr add 10.0.1.1/24 brd + dev r1-eth0")
r1.cmd("ip addr add 10.0.2.1/24 brd + dev r1-eth1")
r1.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")
```

Команда ip addr add 10.0.1.1/24 brd + dev r1-eth0 додає адресу 10.0.1.1/24 з маскою підмережі 255.255.255.0 із стандартною широкомовною адресою і ім'ям r1-eth0.

Команда *echo 1 > /proc/sys/net/ipv4/ip_forward* дозволяє переадресацію IPv4 у системі Linux, дозволяючи хосту діяти як маршрутизатор, пересилаючи IP-пакети між мережами. Використання цієї команди встановлюємо прапорець у файлі переадресації IP як '1', щоб увімкнути переадресацію на маршрутизаторі.

18. Наступним кроком є додавання маршрутів за замовчуванням на хостах. Для цього використовується наступний синтаксис:

```
ip route add default via <IP-адреса шлюзу>
```

Отже, маршрути за замовчуванням на хостах h1, h2 та h3 будуть:

```
h1.cmd("ip route add default via 10.0.1.1")
h2.cmd("ip route add default via 10.0.1.1")
h3.cmd("ip route add default via 10.0.2.1")
h4.cmd("ip route add default via 10.0.2.1")
```

19. Останнім кроком буде додавання маршрутів потоків на комутаторах s1 та s2, щоб хости та маршрутизатор могли взаємодіяти один з одним. У кожного комутатора має бути 4 потоки.

```
s1.cmd("ovs-ofctl add-flow s1 priority=1,arp,actions=flood")
s1.cmd("ovs-ofctl add-flow s1 priority=65535,ip,dl_dst=00:00:00:00:01:01,actions=output:1")
s1.cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=10.0.1.10,actions=output:2")
s1.cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=10.0.1.20,actions=output:3")
s2.cmd("ovs-ofctl add-flow s2 priority=1,arp,actions=flood")
s2.cmd("ovs-ofctl add-flow s2 priority=65535,ip,dl_dst=00:00:00:00:01:02,actions=output:1")
s2.cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_dst=10.0.2.10,actions=output:2")
s2.cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_dst=10.0.2.20,actions=output:3")
```

- Перший потік має бути найменш пріоритетний, він буде потрібний, якщо MAC-адреса призначення невідома.

- Другий потік створюється для того, щоб комутатор знав, куди слід надсилати пакети, коли він отримує їх від хостів. Наприклад, пакет для MAC-адреси 00:00:00:00:01:01 буде переслано через порт Ethernet 1.
- Наступні потоки допомагають комутатору доставляти пакети до відповідного хоста залежно від IP-адреси призначення в пакеті. Наприклад, хост 10.0.1.10 буде переадресовані через порт Ethernet 2, а хост 10.0.1.20 – через порт Ethernet 3.

20. Після виконання усіх змін маємо отримати такий код:

```
#!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    info( '*** Add switches\n')
    r1 = net.addHost( 'r1')

    s1 = net.addSwitch( 's1')

    s2 = net.addSwitch( 's2')

    info( '*** Add hosts\n')
    h1 = net.addHost( 'h1', ip="10.0.1.10/24", mac="00:00:00:00:00:01" )

    h2 = net.addHost( 'h2', ip="10.0.1.20/24", mac="00:00:00:00:00:02" )

    h3 = net.addHost( 'h3', ip="10.0.2.10/24", mac="00:00:00:00:00:03" )

    h4 = net.addHost( 'h4', ip="10.0.2.20/24", mac="00:00:00:00:00:04" )

    info( '*** Add links\n')
    net.addLink( r1, s1 )

    net.addLink( r1, s2 )

    net.addLink( h1, s1 )

    net.addLink( h2, s1 )

    net.addLink( h3, s2 )

    net.addLink( h4, s2 )

    info( '*** Starting network\n')
```

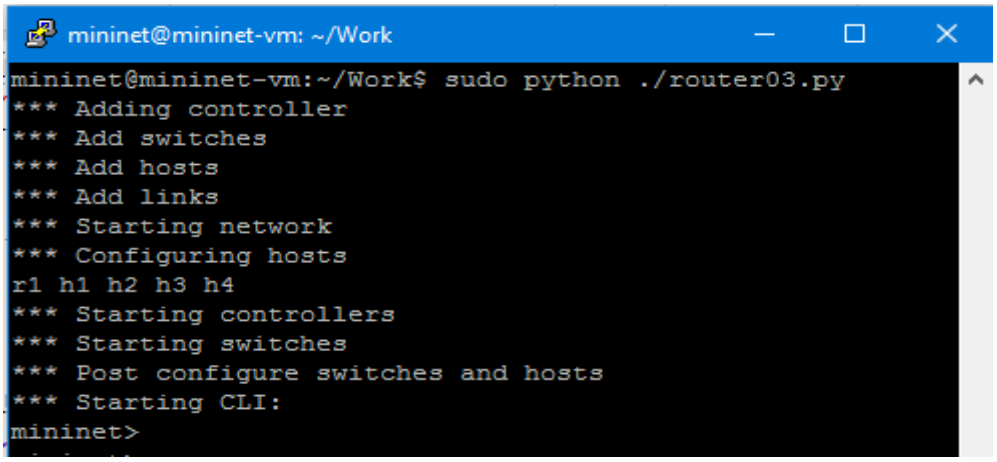
```
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('s2').start([])
    net.get('s1').start([])

    r1.cmd("ifconfig r1-eth0 0")

    r1.cmd("ifconfig r1-eth1 0")

    r1.cmd("ifconfig r1-eth0 hw ether 00:00:00:00:01:01")

    r1.cmd("ifconfig r1-eth1 hw ether 00:00:00:00:01:02")

    r1.cmd("ip addr add 10.0.1.1/24 brd + dev r1-eth0")

    r1.cmd("ip addr add 10.0.2.1/24 brd + dev r1-eth1")

    r1.cmd("echo 1 > /proc/sys/net/ipv4/ip_forward")

    h1.cmd("ip route add default via 10.0.1.1")

    h2.cmd("ip route add default via 10.0.1.1")

    h3.cmd("ip route add default via 10.0.2.1")

    h4.cmd("ip route add default via 10.0.2.1")

    s1.cmd("ovs-ofctl add-flow s1 priority=1,arp,actions=flood")

    s1.cmd("ovs-ofctl add-flow s1
priority=65535,ip,dl_dst=00:00:00:00:01:01,actions=output:1")

    s1.cmd("ovs-ofctl add-flow s1
priority=10,ip,nw_dst=10.0.1.10,actions=output:2")

    s1.cmd("ovs-ofctl add-flow s1
priority=10,ip,nw_dst=10.0.1.20,actions=output:3")

    s2.cmd("ovs-ofctl add-flow s2 priority=1,arp,actions=flood")

    s2.cmd("ovs-ofctl add-flow s2
priority=65535,ip,dl_dst=00:00:00:00:01:02,actions=output:1")

    s2.cmd("ovs-ofctl add-flow s2
priority=10,ip,nw_dst=10.0.2.10,actions=output:2")

    s2.cmd("ovs-ofctl add-flow s2
priority=10,ip,nw_dst=10.0.2.20,actions=output:3")


    info( '*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
```

```
    setLogLevel( 'info' )
    myNetwork()
```

21. Зберігаємо відредагований код в каталог Work.

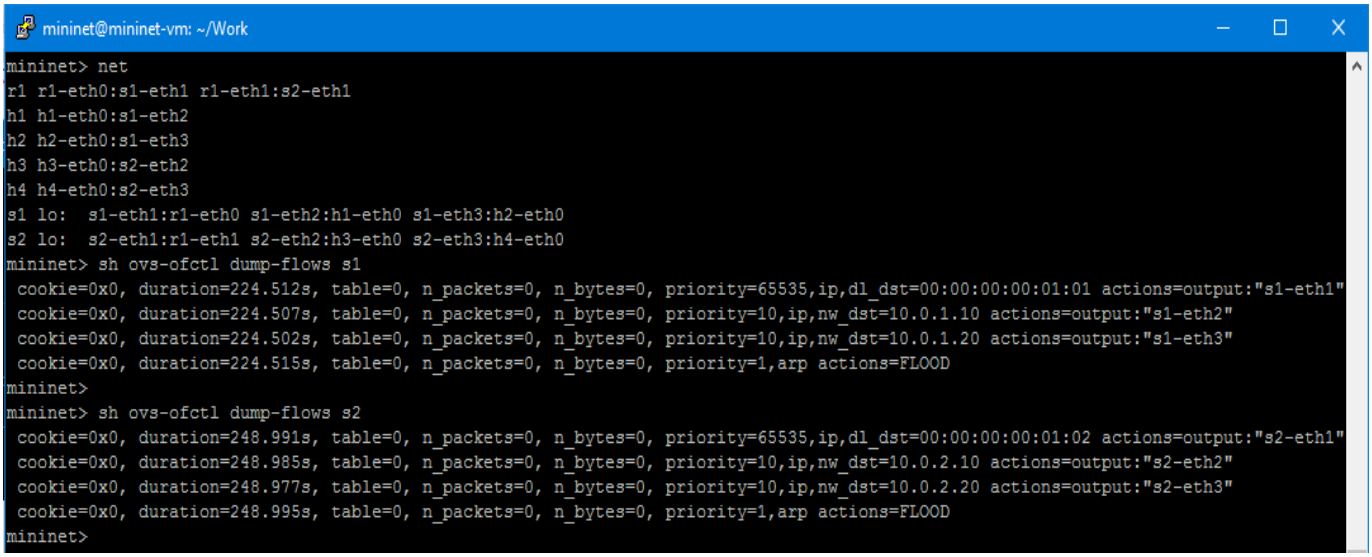
Запускаємо збережений новий скрипт Python на терміналі Putty:

mininet@mininet-vm:~$ **cd Work**
mininet@mininet-vm:~/Work$ **sudo python ./router03.py**




22. Для перевірки налаштувань виконаємо три команди:

mininet> **net**
mininet> **sh ovs-ofctl dump-flows s1**
mininet> **sh ovs-ofctl dump-flows s2**




Бачимо, що тепер потоки на комутаторах налаштовані вірно.

Виконаємо кілька перевірок:

mininet> **h1 ping -c3 h3**
mininet> **h2 ping -c3 h4**

mininet> **h1 ping -c3 r1**
mininet> **pingall**

```
mininet@mininet-vm: ~/Work
mininet> h1 ping -c3 h3
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=1 ttl=63 time=0.241 ms
64 bytes from 10.0.2.10: icmp_seq=2 ttl=63 time=0.099 ms
64 bytes from 10.0.2.10: icmp_seq=3 ttl=63 time=0.094 ms

--- 10.0.2.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 0.094/0.144/0.241/0.069 ms
mininet> h2 ping -c3 h4
PING 10.0.2.20 (10.0.2.20) 56(84) bytes of data.
64 bytes from 10.0.2.20: icmp_seq=1 ttl=63 time=0.226 ms
64 bytes from 10.0.2.20: icmp_seq=2 ttl=63 time=0.098 ms
64 bytes from 10.0.2.20: icmp_seq=3 ttl=63 time=0.095 ms

--- 10.0.2.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2042ms
rtt min/avg/max/mdev = 0.095/0.139/0.226/0.062 ms
mininet> h1 ping -c3 r1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.188 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.076 ms

--- 10.0.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.076/0.113/0.188/0.053 ms
mininet> pingall
*** Ping: testing ping reachability
r1 -> h1 h2 h3 h4
h1 -> r1 h2 h3 h4
h2 -> r1 h1 h3 h4
h3 -> r1 h1 h2 h4
h4 -> r1 h1 h2 h3
*** Results: 0% dropped (20/20 received)
mininet>
```

Бачимо, що налаштування маршрутизації в мережі виконане вірно.

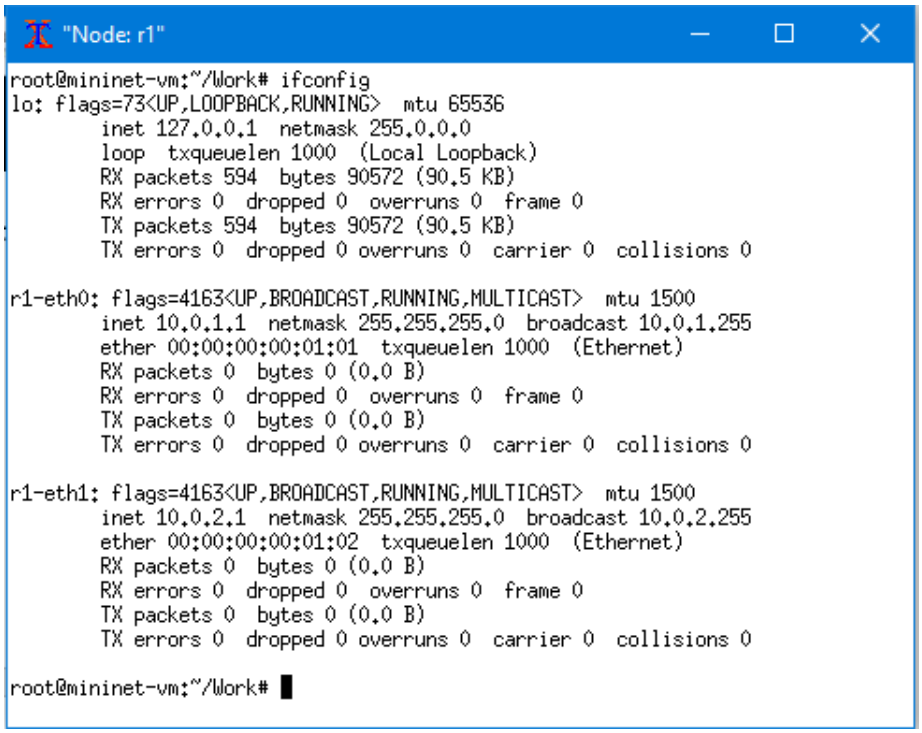
23. Переглянемо таблицю маршрутизації хостів h1 та h4 за допомогою команди *route*:

mininet> **h1 route**
mininet> **h4 route**

```
mininet> h1 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.1.1        0.0.0.0         UG    0      0        0 h1-eth0
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 h1-eth0
mininet> h4 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.1        0.0.0.0         UG    0      0        0 h4-eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 h4-eth0
mininet>
```

24. Переглянемо налаштування інтерфейсів на маршрутизаторі r1. Для цього створимо окремий термінал для r1за допомогою команди **xterm** і виконаємо команду **ifconfig.**

mininet> **xterm r1**




25. Переглянемо проходження пакетів через інтерфейс комутатора s1-eth3 за допомогою програми Wireshark. Для того, щоб Wireshark використовував окреме вікно, з консолі віртуальної машини запускаємо утиліту **startx**:

```
$ startx
```

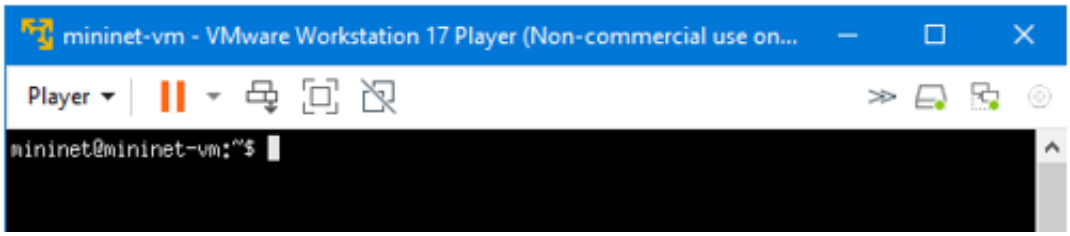
Утиліта *startx* – це сценарій, який використовується в операційних системах на основі Linux та інших систем, подібних до Unix, для запуску середовища робочого столу (X-Server) з командного рядка для ініціалізації графічного середовища для користувача.

Якщо при спробі запустити графічне середовище з командного рядка з'являється помилка «команда startx не знайдена», то потрібно на віртуальну машину додатково встановити пакет **xinit**:

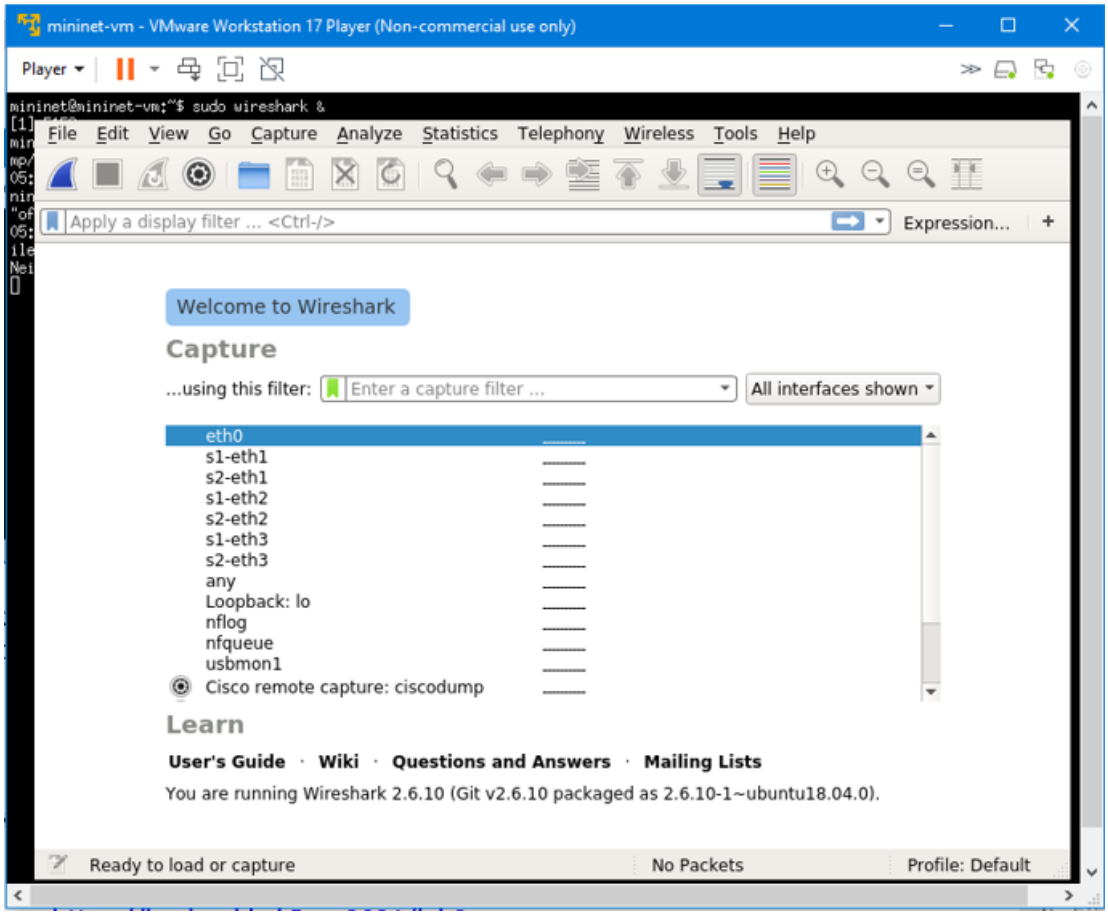
```
$ sudo apt update
$ sudo apt install xinit
```

Після встановлення *xinit* знову запускаємо *startx*. З'являється нове вікно. У цьому вікні запускаємо Wireshark.

```
$ sudo wireshark &
```

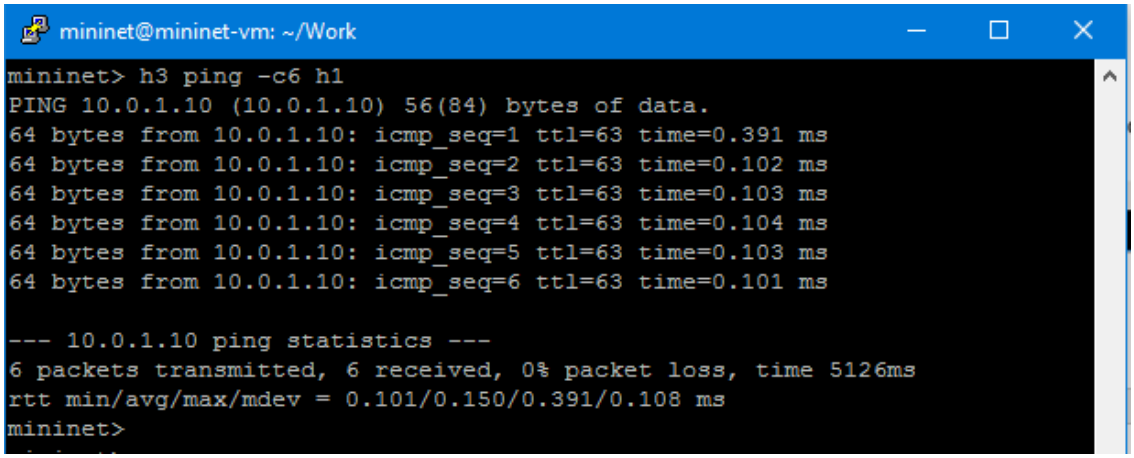



В списку доступних інтерфейсів вибираємо інтерфейс *s1-eth1* для захоплення трафіку (подвійний клік).

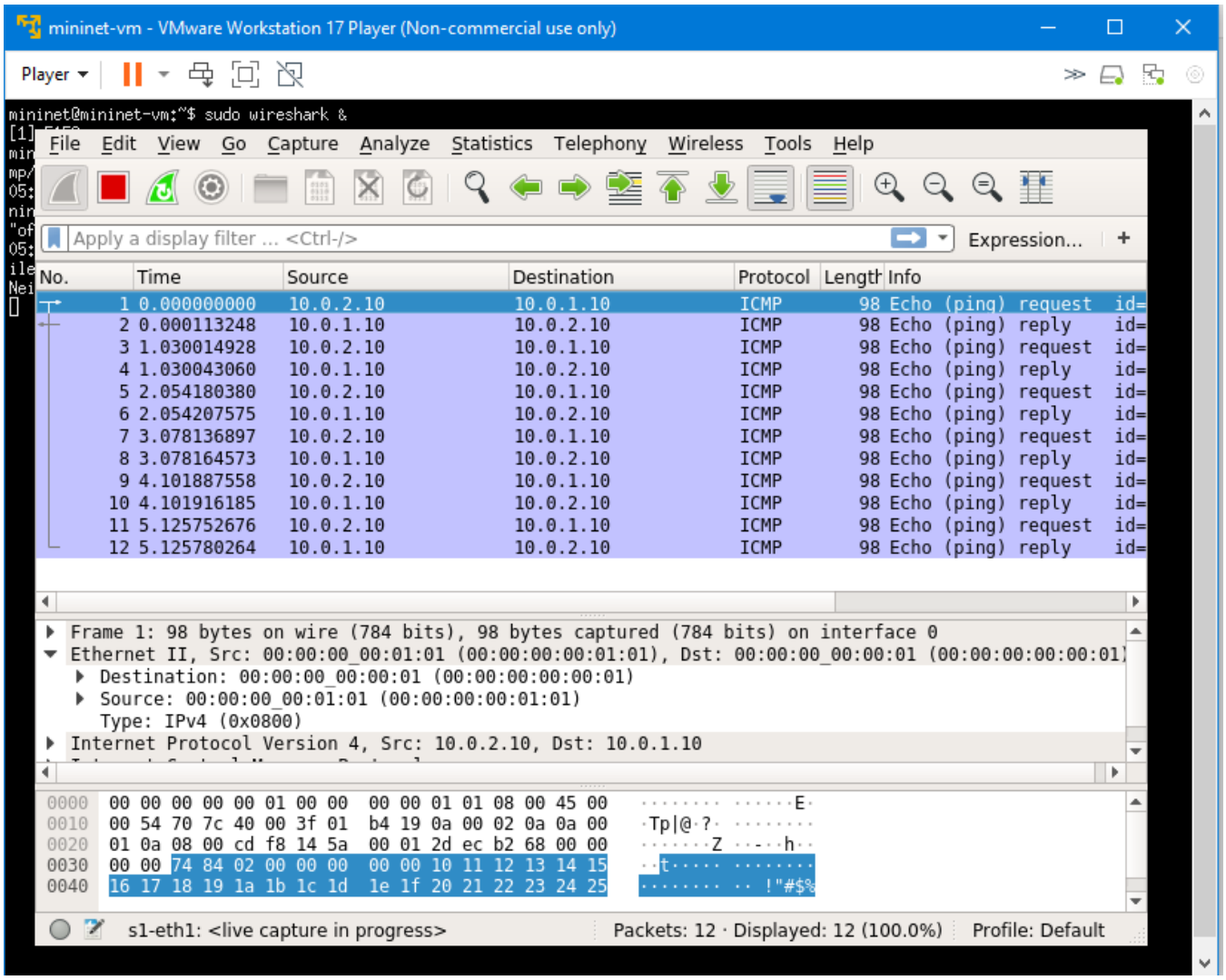



Переходимо на консоль віртуальної машини Mininet і перевіряємо наявність з'єднання між h3 і h1:

mininet> **h3 ping –c6 h1**




У вікні, де запущений Wireshark спостерігаємо проходження пакетів ICMP через інтерфейс комутатора s1-eth1.




Завершуємо роботу програми Wireshark: *File – Quit*. Закриваємо вікно, в якому запускався Wireshark:

```
$ exit
```

## Контрольні запитання

2. Встановлення та налаштування Mininet. Короткий опис кроків.
3. Основні команди управління Mininet.
4. Основні ключі запуску Mininet.
5. За допомогою яких команд можна створити мережу SDN мінімальної топології?
6. Як створити користувацьку топологію мережі в Mininet?
7. Яке призначення команд *xterm* та *startx*?
8. Як запустити простий вебсервер і відповідний клієнт на хості в Mininet?

**Перелік використаних джерел**


1. Використання розширення Visual Studio Code. Електронний ресурс https://learn.microsoft.com/uk-ua/power-pages/configure/vs-code-extension (дата звернення: 11.08.2025).
2. Встановлення віртуального лабораторного середовища https://www.scribd.com/document/690631340/1-1-2-lab-install-the-virtual-machine-lab-environment-uk-UA (дата звернення: 29.05.2025).
3. Гніденко М.П., Вишнівськтй В.В., Ільїн О.О. Побудова SDN мереж. Навчальний посібник. Міністерство освіти і науки України. Державний університет телекомунікацій. 2019.
4. Підготовка Visual Studio Code 2022 для веб розробки. Електронний ресурс https://www.youtube.com/watch?v=M-WeTRvLRtc (дата звернення: 11.06.2025).
5. Формат файлу JSON - Що таке файл JSON? Електронний ресурс https://docs.fileformat.com/uk/web/json/ (дата звернення: 12.06.2025).
6. Шкарупило В.В., Кудерметов Р.В., Мазур Д.С., Скарга-Бандурова І.С., Шумова Л. О., Великжанін А. Ю., Харченко В.С., Узун Д.Д., Узун Ю.О., Годованюк П.А.. Програмно-конфігуровані мережі та Інтернет Речей: Практикум / За ред. Кудерметова Р.К. – МОН України, Запорізький національний технічний університет, Східноукраїнський національний університет ім. В. Даля, Національний аерокосмічний університет ім. М. Є. Жуковського «ХАІ».2019.
7. Adding router and routes for end to end connectivity. Електронний ресурс https://www.chegg.com/homework-help/questions-and-answers/lab-4-adding-router-routes-end-end-connectivity-last-lab-manuallyadded-flows-switch-using--q116960374 (дата звернення: 24.08.2025).
8. Cisco Packet Tracer - Introduction to the Network Controller. Електронний ресурс https://www.youtube.com/watch?v=I-RNW5REdmo&t=18s (дата звернення: 10.06.2025).
9. Lab 2 - Mininet - SDN with Mininet. Електронний ресурс https://hackmd.io/@rse2021/lab2 (дата звернення: 14.08.2025).
10. Mininet_simple_router. Dr. Chih-Heng Ke , Department of Computer Science and Information Engineering, National Quemoy University, Kinmen, Taiwan. Електронний ресурс. https://nqucsie.myqnapcloud.com/smallko/sdn/mininet_simple_router.htm (дата звернення: 11.09.2025).
11. Mininet Walkthrough. Електронний ресурс. https://mininet.org/walkthrough/ (дата звернення: 26.08.2025).
12. Nadeau, T. D. SDN: Software Defined Networks [Text] / T. D. Nadeau, K. Gray. – Sebastopol, CA, USA : O'Reilly Media, 2013. – 384 p. ISBN: 978-1-449-34230-2
13. Network Controller in Cisco Packet Tracer: Starting the Centralized Managemen https://learningnetwork.cisco.com/s/blogs/a0D6e0000112GB4EAM/network-controller-in-cisco-packet-tracer-starting-the-centralized-management (дата звернення: 21.06.2025).
14. Packet Tracer – Compare CLI and SDN Controller Network Management. Електронний ресурс https://itexamanswers.net/8-8-2-packet-tracer-compare-cli-and-sdn-controller-network-management-answers.html (дата звернення: 11.06.2025).
15. Packet Tracer – Implement REST APIs with an SDN Controller. Електронний ресурс https://itexamanswers.net/8-8-3-packet-tracer-implement-rest-apis-with-an-sdn-controller-answers.html (дата звернення: 17.09.2025).