НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Кафедра комп'ютерних систем і мереж

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни
«Протоколи передачі даних в IoT системах»

для студентів спеціальності 123 «Комп'ютерна інженерія»

всіх форм навчання

Частина 1

2019

Методичні вказівки до лабораторних робіт з дисципліни "Протоколи передачі даних в IoT системах" для студентів спеціальності 123 "Комп'ютерна інженерія" всіх форм навчання – Частина 1 / Укл.: В.В. Шкарупило. – Київ: НУБіП, 2019. – 38 с.

Укладачі: В.В. Шкарупило, к.т.н., доцент, доцент каф. комп'ютерних систем і мереж НУБіП України

Рецензенти: М.Д. Місюра, к.т.н., доцент каф. комп'ютерних систем і мереж НУБіП України, В.О. Панкратьєв, ст. викладач каф. комп'ютерних наук НУБіП України

Відповідальний за випуск: В.В. Шкарупило, к.т.н., доцент, доцент каф. комп'ютерних систем і мереж НУБіП України

Затверджено:
на засіданні кафедри
"Комп'ютерні системи і мережі"
Протокол № 3 від 31.10.2019 р.
Рекомендовано до видання:
Вченою радою факультету ІТ
Протокол № 4 від 18.11.2019 р.

ЗМІСТ

ВСТУП

Реалії сучасного світу інформаційних технологій тісно пов'язані із концепцією Інтернету речей (IoT, Internet of Things). Ця концепція полягає в наступному: тисячі (мільйони) «розумних» пристроїв взаємодіють один із одним без участі людини з метою автоматизації (поліпшення) бізнес-процесів різноманітних предметних областей: «розумні міста», «розумні оселі» тощо. Створення систем згідно названої концепції можливе на основі технології програмно-конфігурованих мереж (*SDN, Software Defined Networking*). При цьому вводиться поняття «контролера» – засобу централізованого координуванні взаємодії компонентів системи. Для забезпечення узгодженої взаємодії останніх використовуються різноманітні протоколи передачі даних. Ключову роль при цьому відіграє протокол *OpenFlow* - протокол взаємодії контролера і комутаторів. Типовим шляхом організації процесу проектування систем на основі програмно-конфігурованих мереж є використання спеціалізованих емуляторів. Найпоширенішим таким засобом є емулятор *Mininet*. На використанні названого засобу і будується запропонований комплекс лабораторних робіт.

Комплекс лабораторних робіт охоплює наступні питання: Лабораторна робота №1 – «Встановлення та налаштування середовища *Mininet*» – розглядаються аспекти налаштування та використання середовища *Mininet*; Лабораторна робота №2 – «Створення базових топологій» – розглядаються питання створення та тестування базових топологій програмно-конфігурованих мереж, з використанням протоколу *OpenFlow*; Лабораторна робота №3 – «Робота з графічним інтерфейсом *MiniEdit*» – охоплюються питання конфігурування і тестування мереж за допомогою графічного інтерфейсу; Лабораторна робота №4 – «Автоматизація синтезу мереж» – розглядаються аспекти програмування мереж з використанням *Python API*.

ЛАБОРАТОРНА РОБОТА №1
1 ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ СЕРЕДОВИЩА MININET

Мета роботи – набути практичних навичок встановлення, налаштування та використання емулятору *Mininet*, зокрема протоколу *OpenFlow*.

1.1 Теоретичні відомості

Поточний стан розвитку глобальної мережі можна охарактеризувати наступним чином: існує нагальна потреба поліпшення засобів керування, моніторингу, конфігурування такої системи. Рішенням може слугувати використання програмно-конфігурованих мереж (*SDN, Software Defined Networking*). Ось деякі з ключових позицій [1]: виокремлення рівнів керування (*Control Plane*) та даних (*Data Plane*) [2], залучення уніфікованих комутаторів для перенаправлення потоків даних, застосування засобу централізованого координування таких комутаторів – контролеру (*controller*), слідування регламенту протоколу *OpenFlow* [3, 4]. Специфікація *OpenFlow* надає відкритий інтерфейс взаємодії компонентів *SDN*-мережі. Ключовими компонентами при цьому є контролер, комутатори (*switches*) та хости (*hosts*).

По причині відносної новизни *SDN* безпосередньо робота із відповідною мережею заданої топології не завжди є можливою. Рішенням може слугувати використання різноманітних засобів емуляції – комплексів програмно-апаратних засобів для відтворення *SDN*-мережі у межах віртуального середовища. Програмне забезпечення *SDN*-мереж базується на *Linux*-платформі. Ось приклади таких емуляторів: *Mininet* [5, 6], *EstiNet* [7], *OpenNet* [8], *ns-3* [9]. Кожне із рішень має певні переваги та недоліки, проте важко не відзначити, що *Mininet* часто фігурує у якості зразку, з яким порівнюють інші емулятори. Через це будемо використовувати його у нашому циклі лабораторних робіт.

Середовище *Mininet* призначене слугувати засобом емуляції *SDN*-мережі, зокрема створювати віртуальні хости, комутатори, контролери та зв’язки між ними. Названі компоненти та зв’язки між ними разом формують топологію мережі.

Архітектуру *SDN* подано на рис. 1.1 [10].

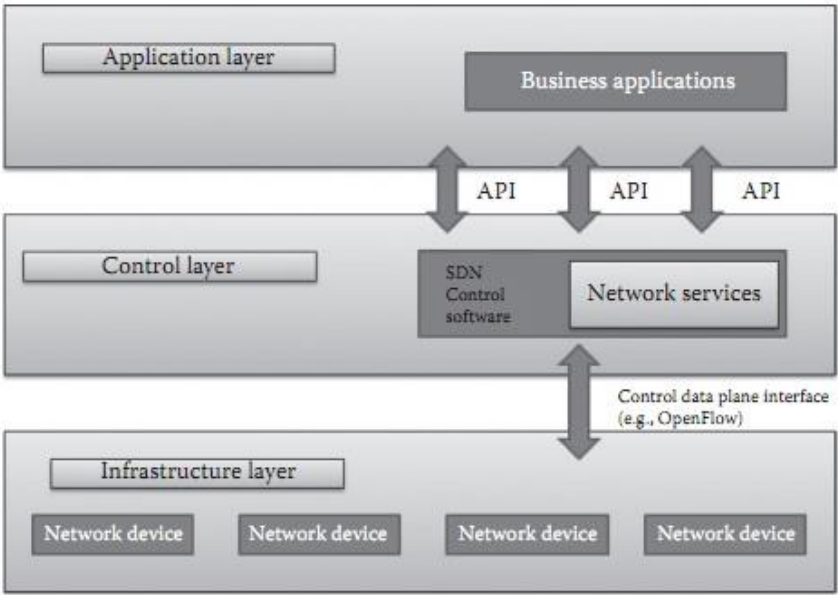

Рисунок 1.1 – Архітектура мережі *SDN*

Середовище *Mininet* надає засоби для здійснення розробки *SDN*-систем, проведення відповідних досліджень, тестувань, налаштувань програмної складової тощо.

Mininet, зокрема, забезпечує наступні можливості:

– дозволяє слугувати стендом для розробки *SDN*-додатків;

– надає підтримку одночасної роботи декількох розробників над спільною топологією;

– включає засобів комплексного тестування топології;

– надає спеціалізований *API* (*Application Programming Interface*), орієнтований на використання мови програмування *Python*.

У порівнянні із типовими підходами до віртуалізації, використання *Mininet* має наступні переваги:

– зручність інсталяції;

– більш швидкий час завантаження середовища;

– зручність реконфігурування системи.

У якості певного недоліку можна відзначити певні труднощі при роботі із графічною оболонкою емулятора із *Windows*-середовища, а також певну лімітованість конфігурації мережі можливостями апаратного забезпечення базової платформи [6].

Виконання роботи полягає в наступному:

– інсталяція на *Windows*-платформі *Linux*-середовища *Mininet* у межах засобу віртуалізації *VirtualBox*;

– конфігурування мережевих інтерфейсів розгорнутого віртуального середовища;

– ознайомлення із базовими консольними командами *Mininet*, зокрема створення мереж із заданими топологіями.

Представлення виконаних завдань на перевірку викладачеві має здійснюватися одним з наступних шляхів:

– у порядку виконання завдань;

– за результатами виконання усіх завдань.

За результатами виконання завдання має бути підготовлено та захищено звіт.

1.2 Виконання роботи

Зауваження: варіант обирається за номером комп'ютера. Для непарних номерів виконується перший варіант, для парних – другий.

1.2.1 Виконати інсталяцію середовища *Mininet*.

Для інсталяції емулятору *Mininet* має використовуватися наступне програмне забезпечення:

– у якості засобу віртуалізації встановити та використати програмний продукт *VirtualBox-4.3.10* – файл *VirtualBox-4.3.10-93012-Win.exe*, призначений для інсталяції на 32-бітну *Windows*-платформу;

– у якості Mininet-середовища – безпосередньо емулятору – версію 2.2.1 – вміст архіву *mininet-2.2.1-150420-ubuntu-14.04-server-i386.zip*, тобто цей файл слід попередньо розархівувати. Із назви файлу бачимо, що віртуалізації підлягає 32-бітне *Linux*-середовище *ubuntu-14.04-server*.

Після того, як засіб віртуалізації *VirtualBox-4.3.10* буде встановлено, вміст архіву – файл *mininet-vm-i386.vmdk* – слід використати у якості жорсткого диску створюваної віртуальної системи.

Демонстрацією успішно виконаних дій має бути система вигляду, подібного до такого, поданого на рис. 1.2.

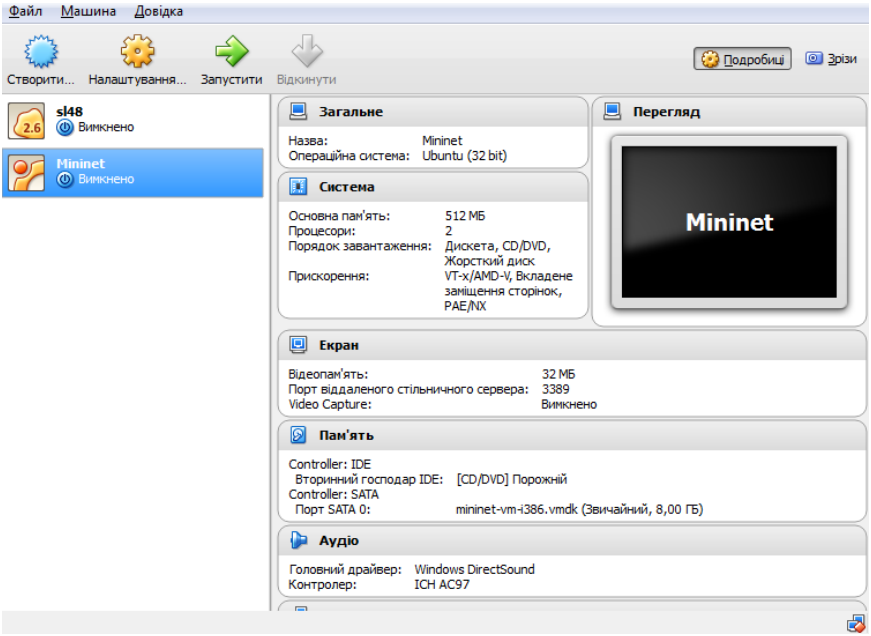

Рисунок 1.2 – Вікно інформації про створюване віртуальне середовище

Варто відзначити, що під створювану систему слід виділяти не менше 512 МБ оперативної пам'яті.

1.2.2 Здійснити налаштування мережевих інтерфейсів віртуального середовища.

Для налаштування мережевих інтерфейсів віртуального середовища слід обрати опцію "Налаштування", та у пункті "Мережа" сконфігурувати "Адаптер 1" та "Адаптер 2" (рис. 1.3, 1.4).

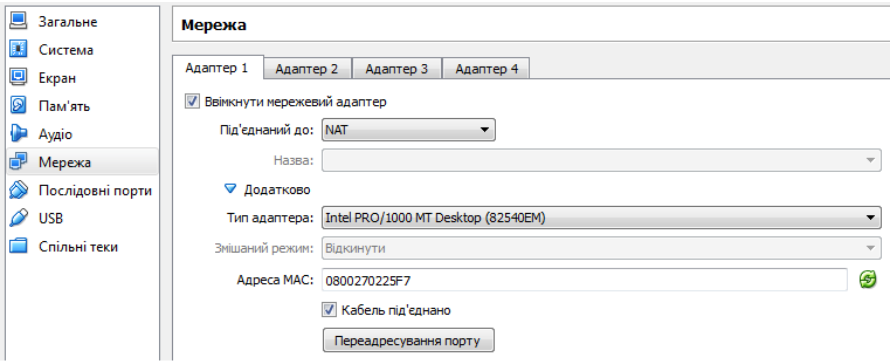

Рисунок 1.3 – Налаштування мережі ("Адаптер 1")

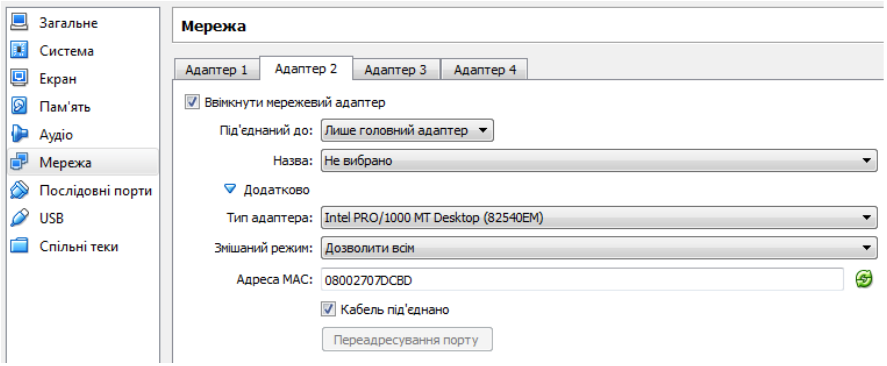

Рисунок 1.4 – Налаштування мережі ("Адаптер 2")

1.2.3 Запустити середовище *Mininet*.

Натиснути на елемент керування "Запустити".

Через деякий час після запуску буде запропоновано ввести логін для входу в систему:

– у рядку *mininet-vm login:* слід ввести логін – *mininet*;

– у рядку *password:* ввести пароль – *mininet*.

У результаті успішних дій активується консоль системи – рядок

```
mininet@mininet-vm:~$
```

1.2.4 Набути навичок роботи із базовими командами середовища *Mininet*.

1.2.4.1 Переглянути конфігурації мережевих інтерфейсів:

```
> sudo ifconfig
```

У результаті виконання названої команди перевірити *IP*-адресу інтерфейсу *eth0*. Нехай, наприклад, маємо адресу 10.0.2.15.

1.2.4.2 Створити топологію мінімальної конфігурації – один контролер (*c0*), один комутатор (*s1*) та два хости (*h1, h2*):

```
> sudo mn
```

Після виведеної інформації про компоненти та зв'язки створеної мережі з'явиться безпосередньо *Mininet*-консоль.

1.2.4.3 Переглянути інформацію про консольні команди *Mininet*:

```
> help
```

1.2.4.4 Переглянути інформацію про вузли мережі (має бути чотири вузли – комутатор (*s1*), два хости (*h1, h2*) та контролер (*c0*)):

```
> nodes
```

1.2.4.5 Перевірити зв'язки між вузлами, із зазначенням залучених портів:

```
> links
```

1.2.4.6 Одержати дані про *IP*-адресу та відповідну маску підмережі для зазначеного інтерфейсу хосту. Наприклад, у випадку інтерфейсу *eth0* вузла *h1* матимемо наступну команду:

```
> h1 ip addr show | grep eth0
```

1.2.4.7 Протестувати пропускну спроможність каналу зв'язку між зазначеними хостами за допомогою команди *iperf*. Наприклад, для вузлів *h1* та *h2* матимемо наступне:

```
> iperf h1 h2
```

Виконання команди займе певний час. При цьому кожний новий результат буде дещо відрізнятись від попереднього.

1.2.4.8* Переглянути інформацію про інтерфейси вузлів:

```
> net
```

1.2.4.9 Переглянути інформацію про конфігурації вузлів:

```
> dump
```

1.2.4.10 Вивести список мережевих інтерфейсів зазначеного вузла, виконавши команду *ifconfig*. Наприклад, у випадку вузла *h1* матимемо наступне:

```
> h1 ifconfig -a
```

1.2.4.11 Переглянути інформацію про процеси, що виконуються на вузлах. Наприклад, для вузла *s1* слід виконати наступну команду:

```
> s1 ps -a
```

1.2.4.12 Перевірити зв’язки між хостами.
Між заданими хостами:

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

Між усіма парами хостів:

```
> pingall
```

1.2.4.13 Запустити веб-сервер та відповідний клієнт на хостах.
Веб-сервер запустимо на вузлі *h1*, а його клієнт – на вузлі *h2*:

```
> h1 python -m SimpleHTTPServer 80 &
> h2 wget -O - h1
```

У результаті виконання останньої команди у консолі буде виведено *HTML*-код одержаної клієнтом веб-сторінки.

1.2.4.14 Завершити роботу веб-серверу:

```
> h1 kill %python
```

1.2.4.15 Продемонструвати викладачеві набуті навички.

1.2.4.16 Вийти із командного середовища *Mininet* до консолі віртуального *Linux*-середовища:

```
> exit
```

1.2.4.17 Очистити супутні конфігураційні дані, пов'язані із створеною топологією, виконавши команду *mn* із прапорцем *-c*:

```
> sudo mn -c
```

1.2.4.18 Оформити звіт за результатами виконаної роботи.

1.3 Зміст звіту

1.3.1 Титульний лист.
1.3.2 Мета роботи.
1.3.4 Відповіді на контрольні питання.

1.4 Контрольні питання

1.4.1 Технологія програмно-конфігурованих мереж *SDN*. Призначення, переваги та недоліки.
1.4.2 Дати визначення поняттю емуляції. Приклади засобів емуляції *SDN*-мереж.
1.4.3 Призначення протоколу *OpenFlow*.
1.4.4 Інсталяція та налаштування *Mininet*. Етапи виконання.

1.4.5 Кроки створення *SDN*-мережі з мінімальною топологією (за замовчанням) – один комутатор та два хости.

1.4.6 Навести та прокоментувати команди взаємодії з хостами та комутаторами.

1.4.7 Навести та прокоментувати команди перевірки зв’язків між хостами.

1.4.8 Навести та прокоментувати команди запуску веб-сервера та відповідного клієнта.

1.4.9 Навести та прокоментувати команду завершення роботи веб-сервера.

1.4.10 Навести та прокоментувати команду очистки конфігураційних даних.

ЛАБОРАТОРНА РОБОТА №2
2 СТВОРЕННЯ БАЗОВИХ ТОПОЛОГІЙ

Мета роботи – набути навичок створення та тестування базових топологій програмно-конфігурованих мереж, функціонування яких базується на використанні протоколу *OpenFlow*.

2.1 Теоретичні відомості

Створення топології програмно-конфігурованої мережі з заданими характеристиками ґрунтується на використанні команди `mn` з відповідними параметрами. Деякі з таких параметрів прокоментовано нижче.

Параметр `--topo tree` задає деревовидну топологію. Атрибут `depth` задає кількість рівнів комутаторів (у нашому випадку – один – як вершина дерева комутаторів; на другому рівні вже буде два комутатори, на третьому – чотири, і т.д.). Атрибут `fanout` регламентує кількість розгалужень від (підключень до) кожного із комутаторів. У нашому випадку `fanout=2`, тобто маємо два підключення (хости) по відношенню до єдиного комутатора.

Наприклад, якщо б ми мали `depth=2`, то топологія би містила два рівні комутаторів. Перший рівень був би представлений єдиним комутатором, до якого би були підключені два комутатори з наступного рівня, а до кожного із них, у свою чергу, було би підключено по два хости. Тобто усього мали б три комутатори та чотири хости.

Параметр `--test pingall` означає, що після того як мережу заданої конфігурації буде створено, необхідно, щоб кожний із хостів виконав команду `ping` по відношенню до кожного із інших хостів.

2.2 Виконання роботи

2.2.1 Створення програмно-конфігурованої мережі із заданими характеристиками комунікаційних каналів.

2.2.1.1 Виконати завдання згідно варіанту, де створити програмно-конфігуровану мережу із заданими значеннями пропускної спроможності (параметр *bw* – *bandwidth*) та затримки (параметр *delay*) на комунікаційних каналах.

Варіант 1. Пропускна спроможність – 10 Мб/с, затримка – 20 мс:

```
> sudo mn --link tc,bw=10,delay=20ms
```

Варіант 2. Пропускна спроможність – 100 Мб/с, затримка – 40 мс:

```
> sudo mn --link tc,bw=100,delay=40ms
```

Для кожного із варіантів виконати наступне:

– заміряти пропускну спроможність каналу зв'язку між хостами п'ять разів, виконавши команду *iperf*. Середнє арифметичне значення замірів занести до звіту;

– визначити мінімальне значення параметру *rtt* (*Round Trip Time*), виконавши команду *ping*.

Зауваження: параметр *rtt* охоплює час, що витрачається на відправку пакета до вузла-одержувача та на одержання вузлом-відправником підтвердження про отримання пакета.

Для кожного із варіантів пояснити одержані результати, а саме – чому заміряні значення пропускної спроможності дещо менші заданого значення (параметр *bw*), а мінімальне значення покажчика *rtt* у середньому в чотири рази більше за встановлене значення затримки *delay* – пояснити, чому саме у чотири.

2.2.2 Створення програмно-конфігурованої мережі із лінійною топологією.

2.2.2.1 Поекспериментувати із зміною розміру створюваної топології:

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

У результаті буде створено конфігурацію, що включатиме вже 3, а не 2 хости.

2.2.2.2 Виконати завдання 1.2.4.4 – 1.2.4.14 (Лабораторна робота №1) по відношенню до створеної мережі.

Занести до звіту результати тестування створеної топології.

2.2.2.3 Створити лінійну топологію із чотирма комутаторами та чотирма вузлами:

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

Така топологія вже включатиме сім зв'язків.

Протестувати мережу. Одержані результати занести до звіту.

2.2.3 Створення програмно-конфігурованої мережі із деревовидною топологією.

2.2.3.1 Створити мережу деревовидної топології мінімальної конфігурації (один контролер, один комутатор, два хости) та протестувати її командою `pingall`:

```
> sudo mn --topo tree,depth=1,fanout=2 --test pingall
```

Зауваження:

– процедура створення та тестування такої мережі доволі ресурсоємна, і триватиме близько 5 с, про що сповіщатиметься у консолі.

2.2.3.2 Поекспериментувати із створенням та тестуванням мереж із різними значеннями параметрів `depth` та `fanout`.

2.3 Зміст звіту

2.3.1 Титульний лист.

2.3.2 Мета роботи.

2.3.3 Пояснення до результатів виконання завдання 2.2.1.1, результати виконання завдання 2.2.2.2.

2.3.4 Відповіді на контрольні питання.

2.4 Контрольні питання

2.4.1 Прокоментувати переваги та недоліки програмно-конфігурованих мереж на основі деревовидної і лінійної топологій.

2.4.2 Команди завдання затримок на комунікаційних зв'язках.

2.4.3 Параметр `rtt`. Визначення. Прокоментувати залежність його значення від значення параметру `delay`.

2.4.4 Засоби зміни конфігурації мережі.

2.4.5 Призначення параметрів `depth` та `fanout` при створенні мережі деревовидної топології. Охарактеризувати їх вплив на кількість вузлів мережі.

ЛАБОРАТОРНА РОБОТА №3
З РОБОТА З ГРАФІЧНИМ ІНТЕРФЕЙСОМ MINIEDIT

Мета роботи – набути практичних навичок налаштування та використання графічного інтерфейсу *MiniEdit* середовища *Mininet* для створення і тестування програмно-конфігурованих мереж, з використанням протоколів *OpenFlow, SSH.*

3.1 Теоретичні відомості

Середовище *Mininet*, з метою поліпшення зручності та інтуїтивності його використання, має також і графічний інтерфейс – *MiniEdit.*

Для виконання роботи знадобиться наступне програмне забезпечення (ПЗ):

– встановлений засіб віртуалізації *VirtualBox* – збірка *VirtualBox-4.3.10-93012-Win.exe*;

– розгорнута віртуальна машина *Mininet* – вміст архіву mininet-*2.2.1-150420-ubuntu-14.04-server-i386.zip* використано у якості жорсткого диску віртуальної машини;

– утиліта *PuTTY* – виконуваний файл *putty.exe* – вільно розповсюджуване клієнтське ПЗ із підтримкою різноманітних протоколів віддаленого доступу, зокрема *Telnet* та *SSH* (*Secure Shell*). *SSH* – мережевий протокол прикладного рівня, що дозволяє здійснювати віддалене керування операційною системою. На відміну від *Telnet*, усі передавані дані шифруються;

– сервер *Xming Server* – збірка *xming-x-server-6.9.0.38.exe* – реалізація віконної системи *X Window System* для забезпечення стандартних інструментів і протоколів побудови графічного інтерфейсу користувача. У роботі використовуватимемо названий сервер у зв'язку із тим, що віртуальна машина, на якій встановлено *Mininet*, не містить графічної оболонки. Сервер надасть можливість візуалізувати у *Windows*-середовищі графічний інтерфейс *MiniEdit* для *Mininet*, що функціонує на віртуальній машині у *Linux*-середовищі [11].

Перші два зазначені пункти вже було пропрацьовано у попередній лабораторній роботі.

Безпосередньо виконання роботи полягатиме у здійсненні наступних кроків:

– інсталяція сервера *Xming Server* – для роботи у *Windows*-середовищі із графічним інтерфейсом *MiniEdit* для *Mininet* на основі віртуального *Linux*-середовища;

– запуск та конфігурування віртуального середовища *Mininet*;

– налаштування *SSH*-клієнту *PuTTY* – для захищеного підключення до віртуального *Linux*-середовища, на якому встановлено емулятор *Mininet*;

– підключення до віртуального *Linux*-середовища через інтерфейс *PuTTY*.

3.2 Виконання роботи

3.2.1 Встановити сервер *Xming*.

Виконати інсталяцію сервера на диск *E*. При цьому слід обрати опцію "*Full installation*" та встановити прапорець навпроти опції "*Normal PuTTY Link SSH client*".

У результаті успішної інсталяції сервер має бути запущено автоматично (рис. 3.1).

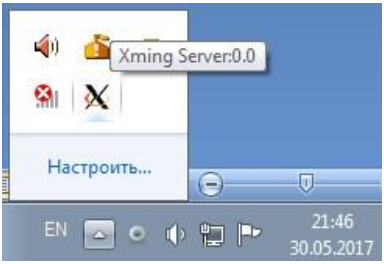

Рисунок 3.1 – Запущений сервер *Xming*

Для ручного запуску серверу необхідно використовувати утиліту *XLaunch.exe*, що знаходиться у директорії *Xming* інсталяції.

3.2.2 Налаштувати віртуальне середовище *Mininet*.

Запустити на виконання віртуальне середовище *Mininet* через засіб віртуалізації *VirtualBox*. При цьому ввести логін і пароль (*mininet, mininet*) – як при виконанні попередньої роботи.

Потім слід виконати наступну послідовність кроків:

– (1-й крок) – спробувати одразу запустити графічне середовище *MiniEdit* у віртуальному середовищі *Mininet*, виконавши наступну команду:

```
> sudo python ./mininet/examples/miniedit.py
```

У результаті буде виведено повідомлення про помилку на кшталт наступного: "*no display name and no $DISPLAY environment variable*". Аби позбутися його, необхідно виконати наступні кроки;

– (2-й крок) – переглянути конфігурацію мережевих інтерфейсів:

```
> ifconfig -a
```

Для роботи використовуватимемо інтерфейс *eth1*. На даний момент зазначений інтерфейс ще не має *IP*-адреси у межах під мережі. Необхідно його вручну сконфігурувати у якості клієнту *DHCP* наступною командою:

```
> sudo dhclient eth1
```

Знову переглянути конфігурацію мережевих інтерфейсів.

У результаті виконання команди зафіксувати *IP*-адресу для мережевого інтерфейсу *eth1*. Має бути адреса, подібна до наступної: 192.168.56.101. Вона буде використовуватися для зовнішнього підключення до віртуального середовища *Mininet* із *Windows*-середовища з використанням утиліти *PuTTY*.

3.2.3 Налаштувати *SSH*-підключення через *PuTTY*.

Для того, щоб *PuTTY*-клієнт мав можливість одержувати доступ до локально запущеного сервера *Xming*, необхідно виконати наступні кроки:

– відзначити прапорцем опцію "*Enable X11 forwarding*" у категорії "*Connection -> SSH -> X11*" налаштувань *PuTTY* (рис. 2.2);

– підключитися до віртуального *Mininet*-середовища у категорії *Session*, ввівши попередньо зафіксовану *IP*-адресу 192.168.56.101 та відкривши сесію, натиснувши на елемент управлянні *Open*.

Зауваження: порт підключення залишити за замовчанням (22). У якості типу підключення прапорцем має бути зазначена опція *SSH*.

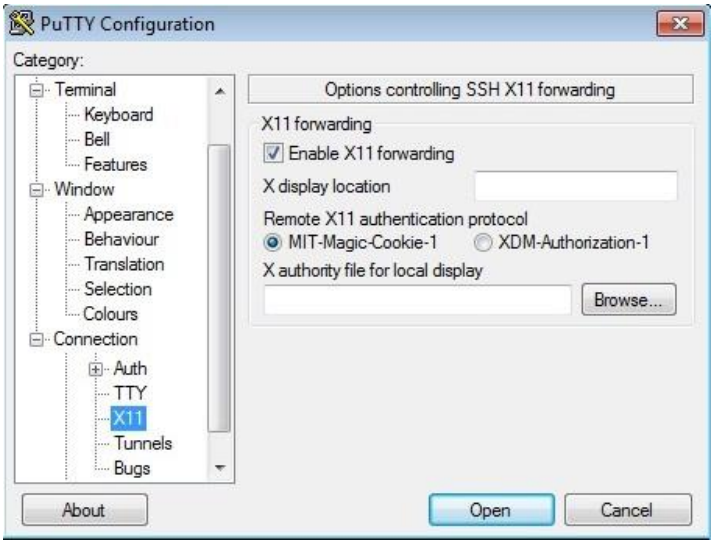

Рисунок 3.2 – Конфігурування SSH-клієнта

3.2.4 Підключитися до консолі *Mininet* через клієнтське програмне забезпечення *PuTTY*.

Виконати наступні кроки:

– у *PuTTY*-консолі, що відкрилася, ввести логін та пароль для входу у *Mininet*-середовище (*mininet*, *mininet*);

– встановити *SSH*-з'єднання, зазначивши зафіксовану *IP*-адресу інтерфейсу *eth1*:

```
> ssh -Y mininet@192.168.56.101
```

– на запит введення паролю ввести пароль *mininet*;

– знову виконати 1-й крок завдання 3.2.2, ввівши через консоль *PuTTY* відповідну команду.

У результаті правильного виконання зазначених кроків у вигляді окремого вікна має з'явитися середовище графічної оболонки *MiniEdit* (рис. 3.3). Це можливо за рахунок роботи серверу *Xming Server*.

Ключова перевага від використання серверу *Xming Server* – для роботи у *Windows*-середовищі не потрібно залучати додаткові бібліотеки, на відміну від серверу *Cygwin* [12].

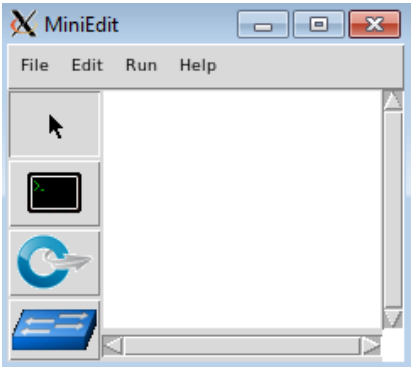

Рисунок 3.3 – Робоча область *MiniEdit*

3.2.5 Створити *SDN*-мережу мінімальної топології засобами графічного інтерфейсу *MiniEdit*.

Результуюча мережа повинна мати вигляд, поданий на рис. 3.4.

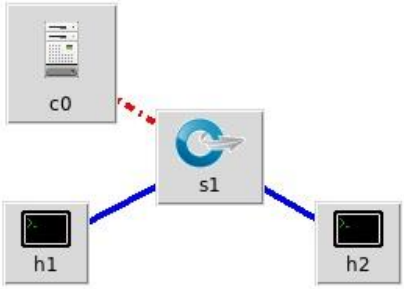

Рисунок 3.4 – Мережа, створена засобами *MiniEdit*

Кожний із вузлів такої мережі можна сконфігурувати потрібним чином (наприклад, задати потрібну *IP*-адресу, змінити назву тощо), натиснувши на відповідному елементі правою клавішею миші та обравши опцію *Properties*. Наприклад, у випадку контролеру *c0* матимемо налаштування, наведені на рис. 3.5.

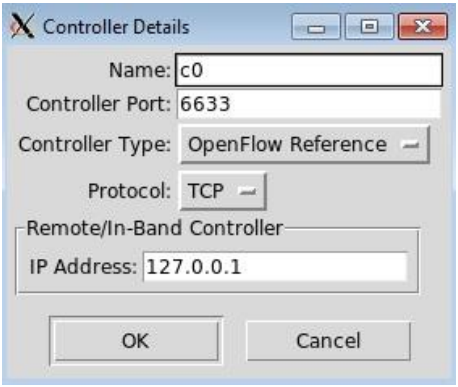

Рисунок 3.5 – Налаштування контролеру

На рис. 3.5 порт 6633 встановлюється за замовчанням.

У випадку, якщо контролерів декілька, їх порти мають відрізнятися. Наприклад, якщо маємо два контролери, номери портів можуть бути 6633 та 6634.

Для перевірки та встановлення налаштувань мережі у середовищі *MiniEdit* використовується елемент панелі інструментів *Edit -> Preferences* (рис. 3.6) [13].

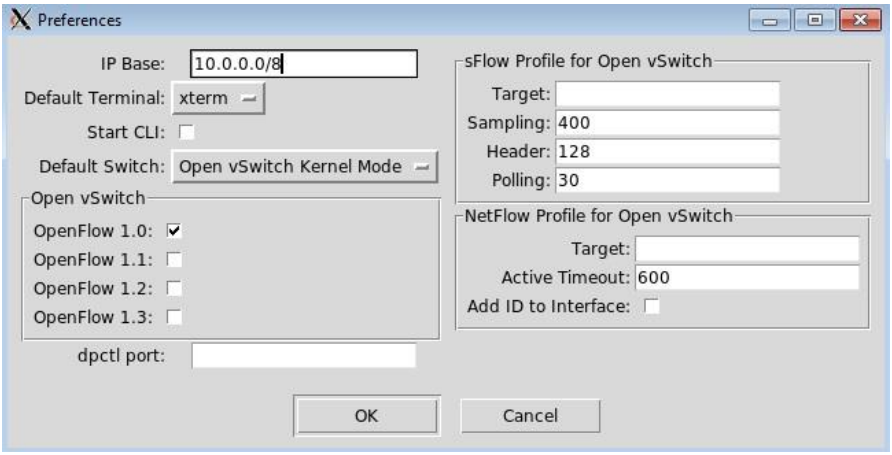

Рисунок 3.6 – Налаштування мережі

На рис. 3.6 видно, зокрема, що у якості протоколу *OpenFlow* за замовчанням використовується версія 1.0. Це є типовою картиною для поточного рівня впровадження мереж *SDN*. Ці налаштування справедливі лише для створеної топології (рис. 3.4).

3.2.6 Зберегти створену топологію у файлі із назвою *topo1.mn* за адресою за замовчанням */home/mininet* (файл топології обов'язково повинен мати розширення **.mn*). Потім створену топологію можна буде за потреби завантажувати у середовище *MiniEdit*.

Для збереження топології виконати команду *File -> Save*, для завантаження – *File -> Open*.

3.2.7 Перевірка створеної мережі.

3.2.7.1 На панелі керування *Edit -> Preferences* встановити прапорець навпроти опції *Start CLI*. Це дозволить працювати з командним рядком при перевірці мережі шляхом імітаційного моделювання.

3.2.7.2 Привласнити вузлам мережі *IP*-адреси.

Маска підмережі в рубриці *Edit -> Preferences* встановлена за замовчанням 10.0.0.0/8.

3.2.7.3 Перевірити зв'язок між хостами, запустивши процес імітаційного моделювання (*Run -> Run*) та, відкривши термінал вузла

h1 (натиск правої клавіші миші на вузлі -> *Terminal*), виконати команду `ping`, вказавши *IP*-адресу цільового вузла, наприклад, наступним чином:

```
> ping 10.0.0.4
```

Зауваження: перед закриттям терміналу ввести команду `exit`.

3.2.7.4 Перевірити схему підключення комутатора. Для цього потрібно переконатися, що імітаційне моделювання виконується, та натиснути *Run -> Show OVS Summary*.

3.2.8 Провести експерименти зі створеною мережею.

3.2.8.1 При запущеному процесі симуляції (імітаційного моделювання), відкрити термінал вузла `h1` і виконати наступну команду:

```
> wireshark &
```

Це запустить на зазначеному хості `h1` утиліту *wireshark* – програму-аналізатор трафіку.

3.2.8.2 У головному вікні програми – у полі *Capture* – у лівій нижній його частині виділити інтерфейс `h1-eth0` та натиснути на елемент керування *Start*. Це дозволить налаштувати програму моніторингу на порт `eth0` вузла `h1`.

3.2.8.3 У терміналі хоста `h1` знову виконати команду `ping`:

```
> ping 10.0.0.4
```

У результаті виконання команди у робочій області програми відображатиметься інформація про передані з порту `h1-eth0` пакети.

3.2.9 Створити мережу деревовидної топології. Конфігураційні дані задати згідно варіанту.

Зауваження:

– варіант обирається за номером по списку: перший варіант – для непарних номерів, другий – для парних.

Варіант 1: Мережа має включати 2 контролери, 7 комутаторів та 8 хостів. Сконфігурувати створену мережу згідно нижченаведених рекомендацій.

Варіант 2: Мережа має включати 3 контролери, 8 комутаторів та 10 хостів. Сконфігурувати створену мережу згідно нижченаведених рекомендацій.

Рекомендації до конфігурування та збереження мережі:

– для контролерів – задати унікальні номери портів;

– у панелі інструментів *Edit -> Preferences* встановити прапорець "*Start CLI*", що дасть змогу використовувати командний рядок;

– зберегти створену мережу у файлі з розширенням "**.mn*", виконавши *File -> Save*;

– зберегти відповідний *Python*-скрипт – у файлі з розширенням "**.py*". Для цього слід обрати опцію *File -> Export Level 2 Script*.

Зауваження: перш, ніж зупинити процес імітаційного моделювання, потрібно ввести команду *exit* у консолі, а потім вже натиснути елемент керування *Stop* на панелі керування.

3.2.10 Виконати дії, зазначені у завданнях 3.2.7, 3.2.8 по відношенню до мережі, створеної згідно варіанту.

3.3 Зміст звіту

3.3.1 Титульний лист.
3.3.2 Мета роботи.
3.3.3 Відповіді на контрольні питання.

3.4 Контрольні питання

3.4.1 Послідовність кроків для організації роботи із графічною оболонкою *MiniEdit* з *Windows*-середовища.

3.4.2 Призначення утиліти *PuTTY*.

3.4.3 Призначення серверу *Xming Server*.

3.4.4 Переваги використання серверу *Xming Server*.

3.4.5 Переваги та недоліки використання для створення *SDN*-мережі заданої топології засобів командного рядку та графічної оболонки *MiniEdit*.

3.4.6 Призначення утиліти *wireshark*.

3.4.7 Засоби тестування мережі у середовищі *MiniEdit*.

3.4.8 Призначення команди *Show OVS Summary*.

3.4.9 З урахуванням деревовидної топології *SDN*-мережі, пояснити залежності між кількостями контролерів, комутаторів і хостів.

3.4.10 Пояснити вплив скороченого формату запису маски підмережі на потенційну кількість хостів.

ЛАБОРАТОРНА РОБОТА №4
4 АВТОМАТИЗАЦІЯ СИНТЕЗУ МЕРЕЖ

Мета роботи – набути практичних навичок використання *Python API* з метою автоматизації процесу створення, конфігурування і дослідження *SDN*-мережі.

4.1 Теоретичні відомості

Середовище *Mininet*, з метою поліпшення зручності та інтуїтивності його використання, має також і графічний інтерфейс – *MiniEdit*.

Робота присвячена напрацюванню навичок побудови *SDN*-мереж з різними топологіями у середовищі емулятору *Mininet* з використанням *Python API*.

Також в роботі висвітлюються аспекти конфігурування таких параметрів як пропускна спроможність, затримки, втрати пакетів, розмір черги по відношенню до окремих з'єднань мережі.

Окрема увага приділяється питанням оцінки продуктивності мережі заданої топології з використанням команд `ping` та `iperf`.

При виконанні базового завдання створюватиметься та конфігуруватиметься мережа з лінійною топологією. Представлення лінійної топології на основі протоколу *OpenFlow* подане на рис. 4.1 [14].

Середовище *Mininet* підтримує параметризовані топології, що дозволяє за рахунок використання коду на мові *Python* створювати гнучкі топології, конфігурація яких залежатиме від параметрів, що передаються.

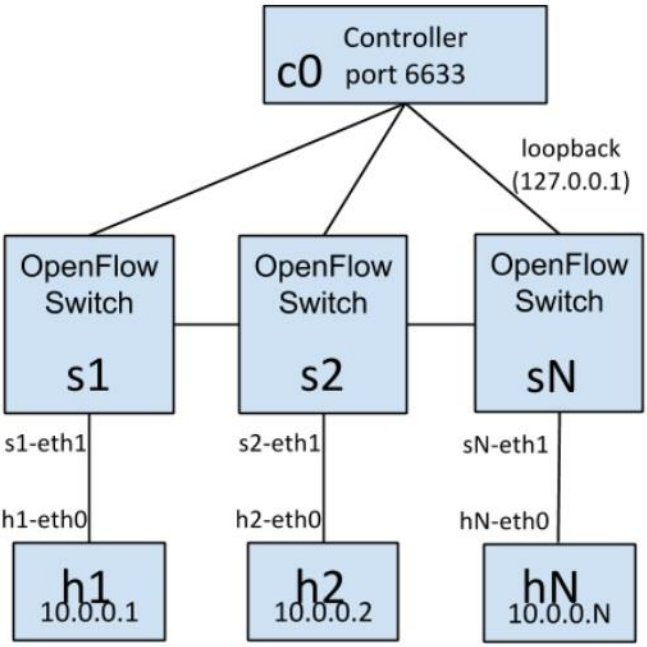

Рисунок 4.1 – *SDN*-мережа з лінійною топологією

У лістингу 4.1 подано приклад програми на мові *Python*, що створює топологію із заданим числом хостів, під'єднаних до відповідних індивідуальних комутаторів [14].

Лістинг 4.1 – Приклад створення і конфігурування *SDN*-мережі з лінійною топологією

```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import irange,dumpNodeConnections
from mininet.log import setLogLevel
class LinearTopo(Topo):
      "Linear topology of k switches, with one host per
switch."

      def __init__(self, k=2, **opts):
```

```
            """Init.
                  k: number of switches (and hosts)
                  hconf: host configuration options
                  lconf: link configuration options"""

            super(LinearTopo, self).__init__(**opts)
            self.k = k

            lastSwitch = None
            for i in irange(1, k):
                  host = self.addHost('h%s' % i)
                  switch = self.addSwitch('s%s' % i)
                  self.addLink( host, switch)
                  if lastSwitch:
                    self.addLink( switch, lastSwitch)
                  lastSwitch = switch

def simpleTest():
      "Create and test a simple network"
      topo = LinearTopo(k=4)
      net = Mininet(topo)
      net.start()
      print "Dumping host connections"
      dumpNodeConnections(net.hosts)
      print "Testing network connectivity"
      net.pingAll()
      net.stop()

if __name__ == '__main__':
      # Tell mininet to print useful information
      setLogLevel('info')
      simpleTest()
```

Коментарі до змінних, класів, методів та функцій лістингу 4.1 подано у табл. 4.1, де представлені основні засоби, використані для автоматизації процесу синтезу програмно-конфігурованої мережі лінійної топології.

Таблиця 4.1 – Пояснення до лістингу 4.1

№ п/п	Концепт	Призначення
1	`Topo`	Основний клас топологій *Mininet*;
2	`Mininet`	Основний клас створення та конфігурування мережі;
3	`addSwitch()`	Метод додавання комутатора до складу топології; повертає назву доданого комутатору;
4	`addHost()`	Метод додавання хоста до складу топології; повертає назву доданого хоста;
5	`addLink()`	Метод додавання двонапрямленого з'єднання між зазначеними вузлами;
6	`start()`	Запуск емуляції мережі;
7	`pingAll()`	Перевірити зв'язки між вузлами;
8	`stop()`	Зупинити процес емуляції мережі;
9	`net.hosts()`	Усі хости мережі;
10	`dumpNodeConnections()`	Скинути зв'язки між вузлами;
11	`setLogLevel( 'info' | 'debug' | 'output' )`	Задати режим виведення інформації. Рекомендується задати `'info'`, оскільки він надає найбільш повну інформацію.

Встановлення параметрів продуктивності мережі. Для цього використовуються класи `CPULimitedHost` і `TCLink`. Приклад відповідного конфігураційного файлу подано у лістингу 4.2 [14]. Назва однойменного файлу – `LinearTopo.py`.

Лістинг 4.2 – Встановлення параметрів продуктивності мережі

```
#!/usr/bin/python
from mininet.topo import Topo
```

```
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from              mininet.util              import
irange,dumpNodeConnections
from mininet.log import setLogLevel

class LinearTopo(Topo):
     "Linear topology of k switches, with one host
per switch."
     def __init__(self, k=2, **opts):
     """Init.
          k: number of switches (and hosts)
          hconf: host configuration options
          lconf: link configuration options"""

     super(LinearTopo, self).__init__(**opts)

     self.k = k
     lastSwitch = None

     for i in irange(1, k):
          host    =    self.addHost('h%s'    %    i,
     cpu=.5/k)
          switch = self.addSwitch('s%s' % i)
          # 10 Mbps, 5ms delay, 1% loss, 1000
     packet queue
          self.addLink(   host,   switch,   bw=10,
     delay='5ms', loss=1,
     max_queue_size=1000, use_htb=True)
          if lastSwitch:
               self.addLink(switch,   lastSwitch,
          bw=10, delay='5ms', loss=1,
     max_queue_size=1000, use_htb=True)
          lastSwitch = switch

def perfTest():
```

```
      "Create network and run simple performance
test"
      topo = LinearTopo(k=4)
      net = Mininet(topo=topo,
            host=CPULimitedHost, link=TCLink)
      net.start()

      print "Dumping host connections"
      dumpNodeConnections(net.hosts)
      print "Testing network connectivity"
      net.pingAll()
      print "Testing bandwidth between h1 and h4"
      h1, h4 = net.get('h1', 'h4')
      net.iperf((h1, h4))
      net.stop()

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

Призначення деяких методів лістингу 4.2:

– `self.addHost(name, cpu=f)` – додати до мережі хост, зазначивши, яка частка системних ресурсів буде виділена під віртуальний хост (`cpu=f`).

– `self.addLink( node1, node2, bw=10, delay='5ms', max_queue_size=1000, loss=1, use_htb=True)` – встановити двонапрямлене з'єднання між зазначеними вузлами із заданими пропускною спроможністю, затримкою, відсотком втрат пакетів, максимальним розміром черги.

4.2 Виконання роботи

4.2.1 Використання *Python API*.

4.2.1.1 Змінити тип створеного файлу `LinearTopo.py` на виконавчий:

```
> chmod u+x LinearTopo.py
```

4.2.1.2 Запустити скрипт (лістинг 4.2) на виконання:

```
> sudo ./LinearTopo.py
```

У результаті на консоль буде виведено інформацію про створення, конфігурування і тестування мережі.

4.2.2 Створення мережі деревовидної топології.

4.2.2.1 Створити конфігураційний файл, подібний до лістингу 4.2, у якому б вже створювалася і тестувалася мережа деревовидної, а не лінійної, топології. Архітектуру мережі, яку потрібно створити, подано на рис. 4.2.

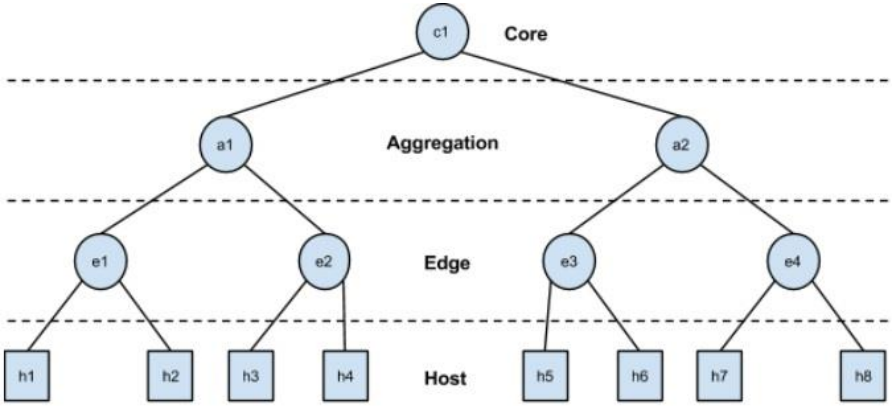

Рисунок 4.2 – Архітектура мережі деревовидної топології

На рис. 4.2 зображено архітектуру, що задається параметрами `depth=3` і `fanout=2`.

Топологія мережі, подана на рис. 4.2, є типовою для сучасних центрів обробки даних (ЦОД).

Каркас скрипта, що потрібно створити, наведено у лістингу 4.3.

Лістинг 4.3 – Каркас скрипта для створення і тестування мережі деревовидної топології

```
'''
Course:
- Programming Assignment
'''

from mininet.topo import Topo

class CustomTopo(Topo):
    "Simple Data Center Topology"

    "linkopts - (1:core, 2:aggregation, 3: edge)
parameters"
    "fanout - number of child switch per parent
switch"
    def __init__(self, linkopts1, linkopts2,
linkopts3, fanout=2, **opts):
        # Initialize topology and default options
        Topo.__init__(self, **opts)

        # Add your logic here ...



topos = { 'custom': ( lambda: CustomTopo() ) }
```

Зауваження 1:

– створений скрипт має дозволяти переглядати параметри bw і delay для кожного з'єднання мережі.

Зауваження 2. У лістингу 4.3 параметри функції `__init__` мають наступне призначення (рис. 4.2):

– `linkopts1` – для встановлення параметрів з'єднань між комутаторами рівнів *Core* і *Aggregation*;

– `linkopts2` – для встановлення параметрів з'єднань між комутаторами рівнів *Aggregation* і *Edge*;

– `linkopts3` – для встановлення параметрів з'єднань між комутаторами рівня *Edge* і хостами;

– `fanout` – кількість підключень до вузла.

4.3 Зміст звіту

4.3.1 Титульний лист.
4.3.2 Мета роботи.
4.3.3 Код програми, створеної на основі лістингу 4.3.
4.3.4 Відповіді на контрольні питання.

3.4 Контрольні питання

4.4.1 Призначення *Python API* у середовищі *Mininet*.
4.4.2 Призначення команд `ping` та `iperf`.
4.4.3 Призначення класу `Topo`.
4.4.4 Призначення методів `addSwitch()` та `addHost()`.
4.4.5 Призначення параметрів `hconf` і `lconf` лістингу 4.2.
4.4.6 Призначення команд `net.start()` та `net.stop()`.
4.4.7 Параметри методу `addLink()`.
4.4.8 Призначення команди `chmod`.
4.4.9 Призначення параметрів функції `__init__`.
4.4.10 Призначення параметрів `depth` і `fanout`.

ЛІТЕРАТУРА

1. Feamster, N. The road to SDN: an intellectual history of programmable networks [Text] / N. Feamster, J. Rexford, E. Zegura // *ACM SIGCOMM Computer Communication Review*. – 2014. – 44(2). – P. 87–98. doi: 10.1145/2602204.2602219

2. 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

3. OpenFlow [Electronic resource]. – Access mode: https://www.opennetworking.org/sdn-resources/openflow. – Title from the screen.

4. Goransson, P. Software Defined Networks: A Comprehensive Approach [Text] / P. Goransson, C. Black, T. Culver. – [2nd ed.]. – Waltham, MA, USA : *Elsevier*, 2016. – 436 p. ISBN: 9780128045558

5. Mininet: An Instant Virtual Network on your Laptop (or other PC) [Electronic resource]. – Access mode: http://mininet.org. – Title from the screen.

6. Keti F. Emulation of Software Defined Networks Using Mininet in Different Simulation Environments / F. Keti, S. Askar // Proc. of 2015 6th International Conference on Intelligent Systems, Modelling and Simulation (Kuala Lumpur, Malaysia, February 9–12, 2015). – P. 205–210. doi: 10.1109/ISMS.2015.46

7. Wang, S-Y. Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet [Text] / S-Y. Wang // Proc. of the *2014 IEEE Symposium on Computers and Communication* (Funchal, Portugal, June 23–26, 2014). doi: 10.1109/ISCC.2014.6912609

8. Chan, M-C. OpenNet: A simulator for software-defined wireless local area network [Text] / M-C. Chan, C. Chen, J-X. Huang [et al.] // Proc. of the *2014 IEEE Wireless Communications and Networking Conference* (Istanbul, Turkey, April 6–9, 2014). doi: 10.1109/WCNC.2014.6953088

9. Ivey, J. Comparing a Scalable SDN Simulation Framework Built on ns-3 and DCE with Existing SDN Simulators and Emulators [Text] / J. Ivey, H. Yang, C. Zhang, G. Riley // Proc. of the *2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation* (Banff, Alberta, Canada, May 15–18, 2016). – P. 153–164. doi: 10.1145/2901378.2901391

10. Hu, F. Network Innovation through OpenFlow and SDN: Principles and Design [Text] / F. Hu. – Boca Raton, FL, USA : *CRC Press*, 2014. – 520 p. ISBN 9781466572096

11. Xming X Server for Windows [Electronic resource]. – Access mode: https://sourceforge.net/projects/xming/. – Title from the screen.

12. Cygwin [Electronic resource]. – Access mode: http://cygwin.com. – Title from the screen.

13. How to use MiniEdit, Mininet's graphical user interface [Electronic resource]. – Access mode: http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/. – Title from the screen.

14. Assignment2_2 - Programming Assignment 2 Using Mininet and... [Electronic resource]. – Access mode: https://www.coursehero.com/file/22722303/Assignment2-2/. – Title from the screen.