Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Ю.В. Кравченко
«_____»_____2022 року

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технологій»

на тему:

МОДЕЛЬ ВІРТУАЛІЗАЦІЇ МЕРЕЖНИХ ФУНКЦІЙ ДЛЯ ВПРОВАДЖЕННЯ МЕРЕЖНИХ ТЕХНОЛОГІЙ МАЙБУТНЬОГО

Виконав: студент групи МІТм-21

Нікольчев Кирил Степанович
(прізвище ім'я по-батькові) (підпис)

Керівник: доцент кафедри мережевих та інтернет технологій

к.т.н., Герасименко К.В.
(посада, прізвище ім'я по-батькові) (підпис)

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Ю.В. Кравченко

«_____»_____2022 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Нікольчеву Кирилу Степановичу
(прізвище, ім'я, по батькові)

1. Тема роботи:

Модель віртуалізації мережних функцій для впровадження мережних технологій майбутнього

затверджена на засіданні кафедри МІТ «__31__» ___серпня_____ 2022 р. протокол № 1
2. Термін здачі закінченої роботи «5» грудня 2022 р
3. Вихідні дані до проекту (роботи)

Програмно-конфігурована мережа, маршрутизація, мова програмування Python.

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 70-80 стор.)

Вступ

1. Огляд підходів до симуляції мережевих пристроїв.
   - 1.1 Розвиток мережних пристроїв
   - 1.2 Функції мережних пристроїв
   - 1.3 Аналіз статичної маршрутизації
   - 1.4 Аналіз динамічної маршрутизації
   - 1.5 Централізована маршрутизація
   - 1.6 Децентралізована маршрутизація
   - 1.7 Гібридна маршрутизація
   - 1.8 Аналіз методів одношляхової та багатошляхової маршрутизації
   - 1.9 Протокол маршрутизації OSPF
   - 1.10 Протокол маршрутизації EIGRP
   - 1.11 Протокол маршрутизації BGP
   - 1.12 Інші класифікації методів маршрутизації
   - 1.13 SDN як рішення сучасних мережних проблем
   - 1.14 Концепція тестового середовища
2. Склад симуляції
3. Випробування роботи симуляції
   - 3.1. Опис топології
   - 3.2. Опис взаємодії маршрутизаторів
   - 3.3. Опис маршрутизації
   - 3.4. Маршрутизація трафіку

Висновки

5. Перелік графічного матеріалу 8-10 слайдів

1. Актуальність роботи
2. Функції мережевих пристроїв
3. SDN як рішення проблем сучасних мереж
4. Склад симуляції
5. Випробування роботи симуляції
6. Висновки

Дата видачі завдання

Керівник роботи Асист. Герасименко К.В. В.

(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання Нікольчев Кирил Степанович

(підпис) (прізвище, ім'я, по батькові)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	24.10.2022	
2	Розділ 1	01.11.2022	
3	Розділ 2	15.11.2022	
4	Розділ 3	01.12.2022	
5	Доповідь та слайди	05.15.2022	
6	Пояснювальна записка	05.15.2022	

Здобувач вищої освіти _____ Нікольчев Кирил Степанович

(підпис) (прізвище, ім'я, по батькові)

Керівник _____ Герасименко К.В.

(підпис) (прізвище, ім'я, по батькові)

РЕФЕРАТ

Пояснювальна записка: 61 с., 40 рис., 26 джерело.

Об’єкт дослідження: моделювання поведінки будь-якої кількості маршрутизаторів на пристрої.
Мета роботи (проекту): розробка корисних інструментів для реалізації нових функцій маршрутизатора та розробки нових мережевих протоколів.
Методи дослідження: У дослідженні використовувалися методи аналізу, методи класифікації та методи імітаційного моделювання.
Актуальність: у зв'язку зі швидким розвитком мережевих технологій у всьому світі впроваджуються нові функції, що дозволяє мережам працювати швидше та краще.
Загальна характеристика роботи: робота присвячена актуальній темі розширення та покращення існуючого функціоналу мережних пристроїв, а надалі гарантовано загальне покращення якості послуг, що надаються мережею. Розроблена система моделювання також дозволяє проводити детальніші експериментальні дослідження різних технічних рішень, що є важливим з наукового погляду.
Ключові слова: мережні пристрої, протокол динамічної маршрутизації, тестове середовище, Python, SDN, симуляція мережних функцій.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕЬ, СИМВОЛІВ,ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ETSI – European Telecommunications Standards Institute
SDN – Software-defined Networking
MAC – Media Access Control
FDDI – Fiber Distributed Data Interface
ATM – Asynchronous Transfer Mode
IP – Internet Protocol
DNS – Domain Name System
IPS – Image Packaging System
VPN – Virtual Private Network
ЦП – центральний процесор
DDOS – Distributed Denial of Service
RIP – Routing Information Protocol
OSPF – Open Shortest Path First
EIGRP – Enhanced Interior Gateway Routing Protocol
IGP – Interior Gateway Protocol
BGP – Border Gateway Protocol
MBGP – Multiprotocol BGP
ASN – Autonomous System Number
TCP – Transmission Control Protocol
WAN – Wide Area Network
TTM – Time To Maarket
ROI – Return on Investment
API – Application Programming Interface

ВСТУП

Мережеві технології розвиваються дуже швидко. Це відбувається через широке використання Інтернету в різноманітних сферах людської діяльності та дозвілля. Традиційно для виконання відповідних функцій на мережних пристроях реалізуються стандартні (щонайбільше створені виробниками пристроїв) протоколи та технології.[1] Але через деякий час було виявлено їх недоліки та невідповідності сучасним мережевим стандартам (збільшення кількості користувачів, пристроїв, сервісів і каналів зв’язку, підвищення вимог до мережі).[2] Існують також нові тенденції розвитку мережі, а саме реалізація концепції програмованої мережі SDN та віртуалізації мережних функцій. Тому постало завдання вдосконалення існуючих протоколів і технологій і створення фундаментально нових для повного задоволення потреб майбутніх інформаційних комунікаційних мереж і систем.[3][4]

Перехід до нових мультисервісних технологій змінює саму концепцію надання послуг, коли якість гарантується не тільки на рівні договірних угод з постачальником послуг і вимог дотримання стандартів, але і на рівні технологій і операторських мереж. Таким чином, корпоративні мережі об’єднують велику кількість локальних мереж і комп’ютерних систем із різними вимогами до якості обміну інформацією. Через це побудова корпоративної мережі на основі деревоподібної структури зв'язків між її абонентами неефективна, оскільки призводить до низького сумарного навантаження мережних каналів.

У роботі розглядаються існуючі методи маршрутизації, їх переваги та недоліки. Було виявлено, що для усунення недоліків методів маршрутизації необхідно використовувати концепцію віртуалізації мережевих функцій (ETSI ISG NFV). Також були показані можливості програмування майбутніх методів маршрутизації.

Таким чином за мету ставилась розробка системи симуляції для можливості реалізувати різні мережні функції та протоколи. У перспективі –

проведення експериментальних досліджень, а в подальшому – реалізація в контексті SDN розроблених технологій та протоколів.[5][6]

1. ОГЛЯД ПІДХОДІВ ДО СИМУЛЯЦІЇ МЕРЕЖЕВИХ ПРИСТРОЇВ

1.1 Розвиток мережних пристроїв

У той час, коли мережні технології були в початковому стані, кожен кінцевий пристрій у мережі отримував усі передані дані, і один справжній пристрій розумів, що дані призначені для нього, а інші просто відкидали їх.

Сьогодні зрозуміло, що такий план є небезпечним і неефективним у сьогоднішній атмосфері недовіри. Наприклад, ви не можете надсилати дані двосторонньої автентифікації на всі пристрої в мережі.

Технічно кажучи, вищевказане рішення відноситься до епохи концентраторів, а не інтелектуальних мережних пристроїв. Вони направляють трафік від одного інтерфейсу до всіх інших.

Оскільки мережеві технології стають все більш складними, розумні пристрої можуть ідентифікувати правильного кінцевого отримувача та відповідно направляти трафік до місця призначення. Це стосується комутаторів і маршрутизаторів, які зараз часто об'єднуються в одному пристрої.

1.2 Функції мережних пристроїв

Сучасні мережі використовують різні мережеві пристрої. Кожен мережний пристрій виконує певну функцію. Тепер поговоримо про основні види пристроїв та їх функції.

Рівень доступу використовує комутатори Ethernet. Можуть бути підключені кілька вузлів до мережі та надсилати повідомлення на певні вузли. Коли вузол надсилає повідомлення через комутатор іншому вузлу, він отримує кадр, декодує його та зчитує фізичну (MAC) адресу повідомлення.

Таблиця комутаторів, звана таблицею MAC-адрес, містить список активних портів і MAC-адрес підключених до них вузлів. Коли вузли обмінюються повідомленнями, комутатор перевіряє, чи містить таблиця MAC-адресу. У цьому випадку комутатор встановлює тимчасове з'єднання, яке називається каналом, між вихідним і цільовим портами. Цей новий канал є виділеним каналом, яким два вузла обмінюються даними. Інші вузли, підключені до комутатора, працюють на іншій пропускній спроможності каналу і не прийматимуть адресовані їм коментарі. Новий канал створюється для кожного нового з'єднання між вузлами. Такі окремі канали дозволяють одночасно встановлювати кілька з'єднань без колізій.

Оскільки перемикання виконується апаратно, воно здійснюється набагато швидше ніж аналогічні функції, що виконуються в мостах з використанням програмного забезпечення. Кожен порт комутатора забезпечує повну смугу пропускання середовища передачі кожної робочої станції. Цей процес називається мікросегментацією.

Мікросегментація (Microsegmentation) дозволяє створювати окремі або виділені сегменти тільки з однією робочою станцією. Кожна така станція має негайний доступ до всієї смуги пропускання та не повинна конкурувати з іншими станціями за доступ до середовища передачі. На дуплексному комутаторі колізій не відбувається, оскільки до кожного порту комутатора підключено лише один пристрій.

Перемикає перенаправлення широкомовних пакетів у всі сегменти мережі. Отже, мережа, що комутується, повинна розглядати всі сегменти як один широкомовний домен.

Деякі комутатори, переважно сучасні комутатори корпоративного рівня, можуть працювати у багаторівневому режимі. Наприклад, пристрої Cisco серій 6500 та 8500 виконують деякі функції рівня 3.

У деяких випадках до порту комутатора може бути підключений інший мережний пристрій, наприклад, концентратор. Це збільшує кількість вузлів, які можна підключити до мережі. Коли концентратор підключений до порту комутатора, MAC-адреси всіх вузлів, підключених до концентратора,

прив'язуються до одного порту. Один вузол на підключеному концентраторі може надіслати повідомлення іншому вузлу на тому самому пристрої. У цьому випадку комутатор приймає кадр і перевіряє розташування вузла призначення за таблицею. Якщо вихідний вузол і вузол призначення підключені до одного порту, комутатор відхиляє повідомлення.

Маршрутизатор — це мережний пристрій, який пересилає пакети між мережами на основі адрес третього рівня. Маршрутизатори можуть вибирати найкращий шлях для вихідних даних у мережі. Працюючи на рівні 3, маршрутизатори можуть приймати рішення на основі мережних адрес замість використання окремих MAC-адрес рівня 2. Маршрутизатори також можуть з'єднувати мережі з використанням різних технологій рівня 2, таких як Ethernet, Token Ring і оптоволоконний інтерфейс розподілених даних (FDDI — розподілений інтерфейс). Маршрутизатори зазвичай з'єднують мережі, використовуючи технологію асинхронної передачі даних ATM та послідовні з'єднання. Маршрутизатори, здатні пересилати пакети з урахуванням рівня 3, є основою глобальної мережі Інтернет і використовують протокол IP.

Завдання маршрутизатора - перевіряти вхідні пакети (тобто дані третього рівня), вибирати найкращий шлях у мережі та перемикатися на відповідний вихідний порт. У великих мережах маршрутизатори є основними пристроями, які регулюють рух потоків даних по мережі. У принципі, маршрутизатори дозволяють обмінюватися інформацією всім видів комп'ютерів.

Як маршрутизатори вирішують, чи надсилати дані в іншу мережу. Пакет містить IP-адреси джерела та одержувача, а також дані повідомлення, що передається. Маршрутизатори зчитують мережну частину IP-адреси призначення та використовують її, щоб визначити, яка з підключених мереж є найкращим способом пересилання повідомлення одержувачу.

Якщо мережна частина IP-адреси джерела та IP-адреси призначення не співпадають, для пересилання повідомлення необхідно використовувати маршрутизатор. Коли вузлу мережі 1.1.1.0 необхідно надіслати повідомлення вузлу мережі 5.5.5.0, повідомлення пересилається на маршрутизатор. Отримуючи повідомлення, воно розархівовується та зчитується IP-адреса

призначення. Потім вирішується, куди надіслати повідомлення. Далі маршрутизатор повторно інкапсулює пакет у кадр і пересилає адресату.

Отже, маршрутизатори виконують такі завдання:

- Підключають локальні мереж до глобальної.
- Сегментують мережі на окремі широкомовні домени, таким чином підвищуючи продуктивність, керованість та безпеку мережі.
- Створюють таблиці маршрутизації для пошуку найкращих маршрутів доставки пакетів через мережу. Вони використовують різні типи протоколів динамічної маршрутизації для пошуку найкращого маршруту до кінцевого отримувача на основі багатьох параметрів.
- Маршрутизатори використовуються як різні сервери, наприклад DNS або DHCP, для створення легкодоступної мережної інфраструктури.
- Маршрутизатори можуть створювати зашифровані тунелі для безпечної передачі даних віддалених мереж(VPN).
- Міжмережевий екран та система запобігання вторгненням(IPS).

1.3 Аналіз статичної маршрутизації

Звичайно, динамічна маршрутизація має кілька переваг перед статичною маршрутизацією. Однак статична маршрутизація все ще використовується в мережах. Насправді мережі зазвичай використовують комбінацію статичної та динамічної маршрутизації.

Статична маршрутизація в основному застосовується коли необхідне:

- Забезпечення простоти обслуговування таблиці маршрутизації в невеликих мережах, в яких не очікуються значні зростання;
- Маршрутизація до та з мереж-тупиків;
- Використання єдиного маршруту за замовчуванням, який використовується для представлення шляху до будь-якої мережі, яка

більше не має певної відповідності з іншим маршрутом у таблиці маршрутизації.

Переваги статичної маршрутизації включають мінімальну обробку ЦП, легкість розуміння та налаштування.

Серед недоліків можна відзначити тривале налаштування та обслуговування, можливі помилки конфігурації у великих мережах, необхідність втручання адміністратора для зміни інформації про маршрутизацію, погана масштабованість у мережах, що ростуть, громіздке обслуговування, необхідність знати всю мережу для правильного налаштування.

1.4 Аналіз динамічної маршрутизації

Для вирішення завдання підвищення якості передачі трафіку в мережі необхідно проаналізувати методи динамічної маршрутизації, їх функції, основні переваги і недоліки, а також можливі негативні явища в мережі, та методи впливу на них. Аналіз методів маршрутизації показує, що протоколи одноканальної маршрутизації, що використовують для пошуку найкоротшого шляху класичні алгоритми Дейкстри, Беллмана-Форда та Шурбале, не можуть бути використані як метод балансування навантаження (трафіку) у мережі. Тому що їх специфіка полягає в тому, щоб перенаправляти трафік лише за найкращим маршрутом. При цьому здебільшого шляхи вибираються без урахування поточного навантаження на інші мережеві ресурси. Пакети все одно відправляються таким способом, навіть якщо найкоротший шлях вже перевантажений.

За способом маршрутизації мережі можуть використовувати централізовану, децентралізовану та гібридну маршрутизацію.

1.5 Централізована маршрутизація

Централізовані мережі побудовані навколо одного централізованого сервера або головного вузла, який обробляє всі основні дані та зберігає дані користувача та інформацію, до якої інші користувачі можуть отримати доступ.

Звідси клієнтські вузли можуть бути підключені до головного сервера та надсилати запити на дані замість того, щоб виконувати їх безпосередньо.

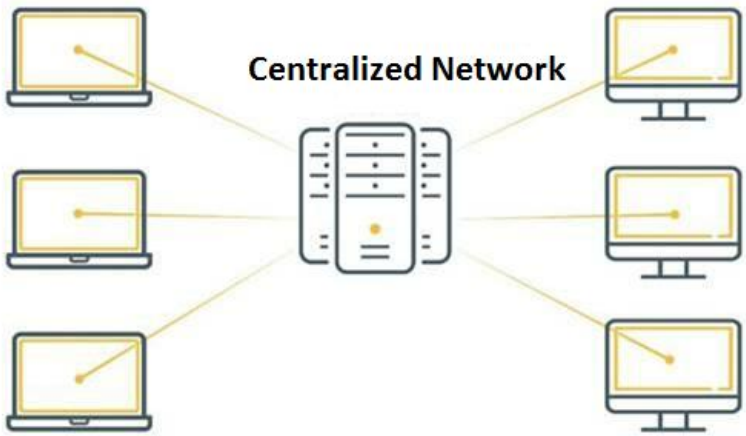

Рис. 1.1 Модель централізованої мережі.

Централізована маршрутизація реалізується за принципом вибору напрямку руху для кожного пакета центром управління мережею, а вузли мережі лише сприймають і реалізують результати вирішення завдання маршрутизації. Перевагою цього методу є можливість вибору простих за структурою вузлів, оскільки вони беруть мінімальну участь у процесі маршрутизації. Однак зі збільшенням кількості вузлів зростає складність організації централізованого управління мережею передачі даних.

Оскільки ланцюжки команд чітко визначені в централізованих мережах, делегування всередині мережі відносно просте, і на різних рівнях авторизації потрібен менший обмін повідомленнями. Також легко додавати та видаляти клієнтські вузли з мережі, створюючи або видаляючи з’єднання між

клієнтським вузлом і головним сервером. Однак це не збільшує обчислювальну потужність мережі.

Централізовані мережі, як правило, є найбільш економічно ефективним варіантом для невеликих систем і потребують менше ресурсів для налаштування та обслуговування. Крім того, коли адміністратору мережі потрібно виправити або оновити мережу, потрібно оновити лише центральний сервер. Це зменшує час і накладні витрати, необхідні для підтримки мережі в актуальному стані.

Враховуючи низхідний характер централізованих мереж, легше стандартизувати взаємодію між основним сервером і клієнтськими вузлами. Це може призвести до більш послідовної та оптимізованої роботи кінцевого користувача. Крім того, оскільки відносно легко відстежувати та збирати дані в режимі онлайн, суттєвим недоліком централізованого контролю є пряма залежність якості маршрутизації від надійності його центру управління, яка має тенденцію до зниження зі збільшенням складності останнього. Крім того, центр управління мережею повинен мати оперативну інформацію про стан мережі, оскільки вихід з ладу вузла або його перевантаження може призвести до виходу з ладу всієї мережі.

Оскільки централізовані мережі мають єдину точку збою, якщо основний сервер виходить з ладу, вся мережа, ймовірно, перестане працювати. Таким чином, клієнтські вузли не зможуть надсилати, отримувати або обробляти запити користувачів самостійно. Крім того, технічне обслуговування сервера може передбачати тимчасове відключення основного сервера, що, ймовірно, призведе до перебоїв у роботі та, як наслідок, до незручностей / зниження надійності з точки зору користувача. Наявність єдиної точки відмови також збільшує шанси на порушення безпеки або збої через загрози кібербезпеці, такі як атаки DDOS, оскільки існує лише одна ціль, яку можна зламати. Крім того, оскільки існує лише одне центральне сховище для даних користувачів, централізовані мережі завжди будуть нести ризики конфіденційності. Якщо основний сервер пошкоджений або не працює,

Централізовані мережі може бути складно масштабувати за межі певної точки, оскільки єдиний спосіб зробити це — додати більше пам’яті або обчислювальної потужності до центрального сервера. Крім того, якщо в мережі спостерігаються спалахи трафіку, які перевищують той, який мережа була розроблена для обробки, можуть виникати інформаційні вузькі місця, коли користувачі, віддалені від центрального сервера, відчувають підвищену затримку.

1.6 Децентралізована маршрутизація

Децентралізована мережа розподіляє навантаження з обробки інформації між кількома пристроями замість того, щоб покладатися на один центральний сервер. Кожен із цих окремих пристроїв служить міні-центральним блоком, який незалежно спілкується з іншими вузлами. У результаті, навіть якщо один із головних вузлів виходить з ладу або зламається, інші сервери можуть продовжувати надавати користувачам доступ до даних, а вся мережа продовжуватиме працювати з обмеженими перебоями або без них. Децентралізовані мережі стали можливими завдяки останнім технологічним досягненням, які забезпечили комп’ютери та інші пристрої значною обчислювальною потужністю, їх можна синхронізувати та використовувати для розподіленої обробки.

Розподілена або децентралізована маршрутизація здійснюється шляхом розподілу функцій управління мережею між її вузлами. На основі збереженої керуючої інформації кожен вузол самостійно визначає напрямок передачі пакетів. Це збільшує структурну складність вузлів, але мережа відзначається високим рівнем доступності, оскільки вихід з ладу будь-якого вузла не впливає на роботу мережі в цілому.

Оскільки децентралізовані мережі не мають єдиної точки відмови, вони можуть продовжувати працювати, навіть якщо головний вузол скомпрометовано або вимкнено. Крім того, децентралізовані мережі легко

масштабуються, оскільки ви можете просто додати більше пристроїв до мережі, щоб збільшити її обчислювальну потужність, а обслуговування мережі зазвичай не вимагає повного відключення мережі.

Запити користувачів часто надходять швидше за використання децентралізованої мережі, оскільки мережеві адміністратори можуть створювати головні вузли в регіонах з високою активністю користувачів, на відміну від маршрутизації з'єднань через великі території до одного централізованого сервера.

Децентралізовані мережі забезпечують більший рівень конфіденційності користувачів, оскільки інформація, що зберігається в мережі, розподіляється в кількох місцях, а не в одному місці. Це ускладнює моніторинг потоку даних у мережі та усуває ризик того, що зловмисники матимуть лише одну ціль.

Однак децентралізована маршрутизація має кілька недоліків. Децентралізовані мережі більш стійкі, ніж централізовані. Це зазвичай робить обслуговування цих мереж дорожчим і займає більше часу.

Оскільки децентралізована мережа використовує кілька пристроїв для підтримки системи, це накладає відповідне навантаження на IT-ресурси організації. У результаті децентралізовані системи часто не підходять для організацій, яким потрібна лише невелика система, оскільки співвідношення витрат і вигод за цих умов є несприятливим.

Оскільки головні вузли в децентралізованій мережі працюють незалежно і можуть не взаємодіяти один з одним, великі організації можуть зіткнутися з проблемами координації та важкістю керуванням та виконанням колективних завдань. Хоча це навмисна функція децентралізованих мереж, це означає, що не всі бізнес-моделі та організаційні структури обов'язково виграють від використання децентралізованої мережі.

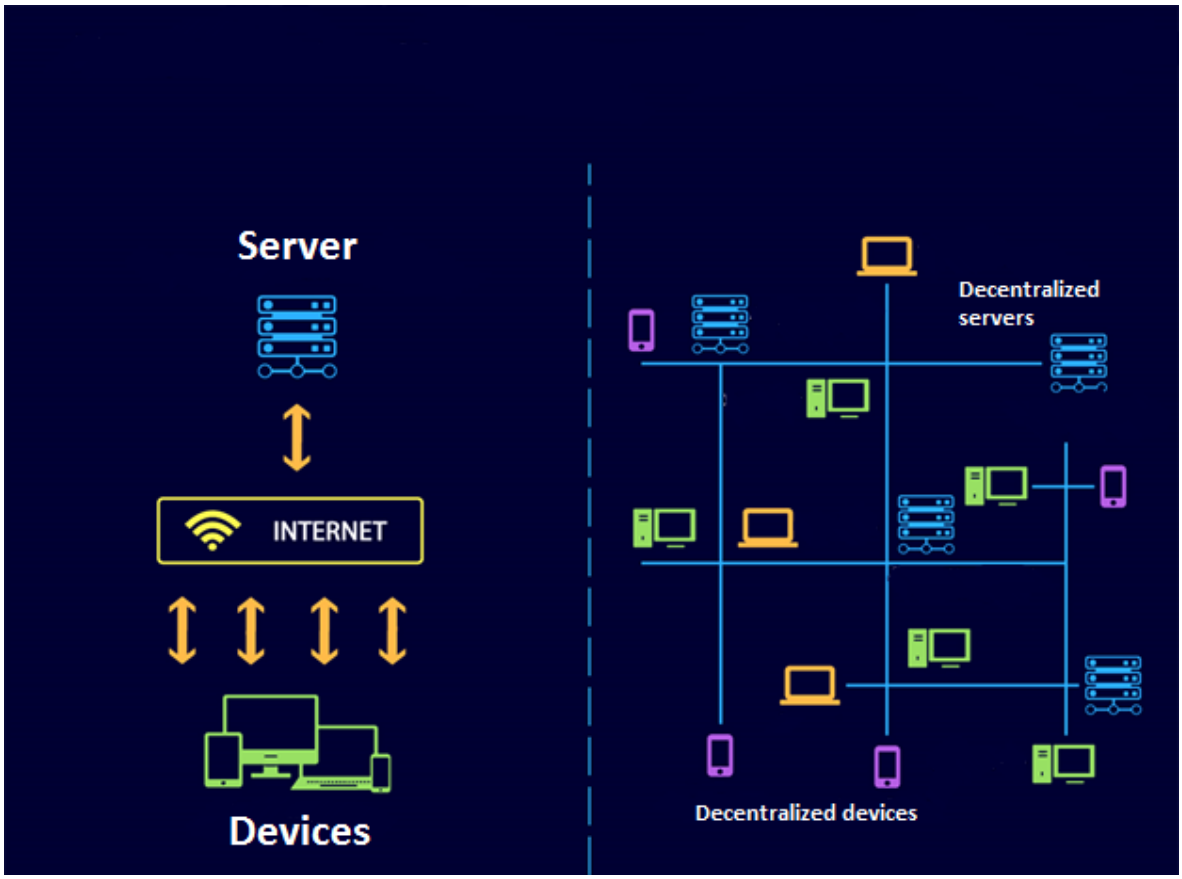

Рис. 1.2 Порівняння централізованих і децентралізованих мереж.

1.7 Гібридна маршрутизація

Гібридна маршрутизація характеризується застосуванням принципів централізованої та розподіленої маршрутизації (наприклад, гібридна адаптивна маршрутизація). Адаптивна маршрутизація передбачає адаптацію алгоритму маршрутизації до реального стану мережі. Недоліком методів адаптивної маршрутизації є складність прогнозування стану мережі.

1.8 Аналіз методів одношляхової та багатошляхової маршрутизації

За кількістю конкретних маршрутів до одного пункту призначення протоколи маршрутизації поділяються на одношляхові та багатошляхові. Протоколи з одним шляхом вводять інформацію про один оптимальний маршрут у таблиці маршрутизації. Очевидним недоліком є нерівномірне

завантаження мережі через максимальне завантаження оптимального маршруту. Багатошляхові протоколи відрізняються визначенням кількох оптимальних шляхів. Це дозволяє розпаралелювати передачу трафіку і, як наслідок, підвищити надійність передачі даних і ефективність використання каналів зв'язку. Незважаючи на очевидні переваги багатошляхових протоколів сьогодні, в сучасних мережах використовуються одношляхові протоколи, найвідоміші з яких RIP, OSPF і EIGRP.

1.9 Протокол маршрутизації OSPF

OSPF — це широко використовуваний протокол внутрішнього шлюзу (IGP), заснований на технології стану зв’язку та пошуку найкоротшого шляху. Цей протокол здійснює маршрутизацію пакетів, збираючи інформацію про стан каналів з сусідніх маршрутизаторів і на основі отриманої інформації будує карту мережі. Маршрутизатори OSPF надсилають багато типів службових повідомлень, включаючи повідомлення привітання, запити статусу зв’язку, оновлення та описи бази даних. Пошук найкоротшого шляху здійснюється за алгоритмом Дейкстри. OSPF використовує метрику (вартість) для вибору найкращого маршруту, який за замовчуванням розраховується на основі пропускної здатності каналу.

Перевага транспортування трафіку при використанні OSPF полягає в тому, що зміни топології мережі обробляються дуже швидко. Основним недоліком протоколу OSPF є те, що за допомогою алгоритму Дейкстри визначається один найкращий маршрут, по якому направляється весь трафік. Це може призвести до перевантаження IP-мережі та потребує впровадження додаткових методів.

1.10 Протокол маршрутизації EIGRP

EIGRP, протокол динамічної маршрутизації з дистанційним вектором, було оптимізовано для зменшення нестабільності протоколу після зміни

топології мережі, уникнення проблем з петлею маршруту та більш ефективного та економного використання пропускної здатності маршрутизатора та пропускної здатності. Композитна метрика, яка використовується для пошуку оптимального шляху, обчислюється на основі пропускної здатності, навантаження, затримки та надійності. Це покращує якість вибору оптимального маршруту.

Основними перевагами EIGRP є низьке споживання мережевих ресурсів при відсутності змін топології (передаються тільки пакети привітання), коли відбуваються зміни, по мережі передається тільки інформація про зміни, що відбулися, що дозволяє знижує навантаження на мережу і забезпечує короткий час конвергенції (в роздільній конвергенція забезпечується практично миттєво).

Поряд з перевагами сучасних протоколів динамічної маршрутизації слід зазначити, що всі вони шукають один найкращий маршрут з мінімальною метрикою, тобто односторонній, або балансують маршрути в мережі з однаковою метрикою, що зумовлює максимальне використання знайденого найкращого або альтернативного шляху та його перевантаження. при цьому інші вузли (ресурси) мережі не будуть задіяні в процесі передачі трафіку. Такий підхід не дає можливості досягти стану повної рівноваги, збалансованого розподілу навантаження між усіма можливими альтернативними шляхами.

EIGRP забезпечує механізми реалізації багатошляхової маршрутизації, зокрема за допомогою техніки балансування навантаження за нерівною вартістю, але вона рідко використовується, оскільки ускладнює процес налаштування. Крім того, динамічні параметри зв'язку, такі як надійність і використання, не використовуються за замовчуванням при розрахунку показників в EIGRP, оскільки їх використання призводить до постійних змін показників і, як наслідок, перебудови маршруту.

Виправляти ситуацію шляхом внесення змін до конкретного протоколу недоцільно, оскільки ця проблема спостерігається у всіх протоколах динамічної маршрутизації, тому більш ефективним рішенням буде модифікація процесу маршрутизації без внесення змін до конкретного протоколу маршрутизації. Такий варіант впливу дозволить зменшити затримку в передачі трафіку і

збалансувати навантаження на мережу, універсально для всіх протоколів динамічної маршрутизації.

1.11 Протокол маршрутизації BGP

BGP з 1994 року єдиний протокол маршрутизації між автономними системами в глобальній мережі Інтернет, а його розширена версія MBGP використовується в MPLS-мережах IT-провайдерів.

BGP є протоколом міждоменної маршрутизації та належить до класу дистанційно-векторних протоколів. Як протокол міждоменної маршрутизації вико- ристовується усіма інтернет-провайдерами, а також великими компаніями та організаціями, які мають власні публічні номери автономних систем (ASN) та користуються послугами більш ніж одного інтернет-провайдера (мултіхомінг) або мають прямі IP-з'єднання з багатьма іншими великими компаніям, що також мають власні публічні номери автономних систем, без використання послуг інтернет-провайдерів.

Разом з тим немає ніяких обмежень на використання BGP в локальних мережах крім рекомендацій про приватні ASN, але використання BGP в якості протоколу внутрішньодоменної маршрутизації є недоцільним через значний час конвергенції у порівнянні з іншими протоколами маршрутизації, що закладено в його дизайні.

1.12 Інші класифікації методів маршрутизації

Однорівневі або ієрархічні алгоритми відрізняються тим, як вони взаємодіють один з одним. У одноранговій системі маршрутизації всі маршрутизатори є рівними по відношенню один до одного. В ієрархічній системі маршрутизації пакети даних переміщуються від маршрутизаторів нижчого рівня до основних, які виконують базову маршрутизацію. Коли пакети

досягають загальної області призначення, вони чергуються вниз по ієрархії до хоста призначення.

У вихідних системах маршрутизації маршрутизатори діють просто як пристрої зберігання та пересилання пакету, надсилаючи його до наступної зупинки без будь-яких коливань, вони припускають, що відправник розраховує та визначає весь маршрут сам. Інші алгоритми припускають, що хост відправника нічого не знає про маршрути. За допомогою такого алгоритму маршрутизатори визначають маршрут через мережу на основі власних розрахунків.

Внутрішньодоменні або міждоменні алгоритми. Деякі алгоритми маршрутизації працюють лише в межах доменів; інші, як усередині, так і між доменами.

Алгоритми стану зв’язку направляють потоки інформації про маршрутизацію до всіх вузлів мережі. Кожен маршрутизатор надсилає лише ту частину відомої йому інформації, яка описує стан його власних каналів, але до всіх вузлів маршрутизації. Вектори відстані вимагають, щоб кожен маршрутизатор пересилав всю або частину своєї таблиці, але лише сусідам.

Розподіл пропускної здатності трактів передачі мережі може здійснюватися шляхом нормування швидкості TCP (TCP rate shaping), яке полягає у перехопленні та маніпулюванні розмірами TCP-вікна, або за допомогою механізмів управління чергами, а точніше - організації та обслуговування черг на мережних вузлах. Механізм обслуговування черг шляхом регулювання порядку обслуговування пакетів певного потоку (класу) трафіка дозволяє варіювати частоту їхньої обробки й у такий спосіб виділяти певну пропускну здатність даному потоку (класу). Черги та засоби їхньої обробки є інструментами також управління перевантаженнями, коли мережний пристрій не може передати пакети на вихідний інтерфейс в тому темпі, у якому вони надходять.

1.13 SDN як рішення сучасних мережних проблем

Нині можна назвати два напрями подальшого розвитку мережі:

- Оптимізація існуючих протоколів, технологій, алгоритмів та механізмів відповідно до сучасних мережних вимог.
- Автоматизоване керування мереженими пристроями.
- Віртуалізація функцій мережі.

Як бачимо, серед існуючих алгоритмів маршрутизації немає методів обліку втрат.

Динамічна маршрутизація реагує лише на грубі зміни, і погано реагує на зміни перевантаженості каналів, затримки не враховуються, а пріоритет типу трафіку не враховується навіть в EIGRP.

Враховуючи низку проблем, а також обмеження в можливості вирішення цих проблем через неможливість змінити стандарти, виникає потреба в інструментах, щоб обійти ці обмеження.

Одним з найбільш перспективних способів вирішення сучасних проблем розвитку мереж є модель програмно-визначуваної мережі (SDN), яка передбачає поділ функцій передачі трафіку та функцій керування, включаючи контроль як самого трафіку, так і пристроїв, які його передають. Вона включає поділ функцій відправки і управління трафіком, включаючи контроль як над самим трафіком, так і над пристроями, що його надсилають. Згідно з концепцією SDN, вся логіка управління працює по концепції потоків, в складі яких є контролери, що керують поведінкою усієї мережі за допомогою спеціальних протоколів (таких як OpenFlow), і які можуть з їх допомогою виконувати деякі дії (дозвіл, заборона, редагування, перенаправлення полів у пакетах і т.д.) Переваги програмно-визначуваної мережі полягають у централізованому управлінні, спрощеному обслуговуванні мережі та модернізації.

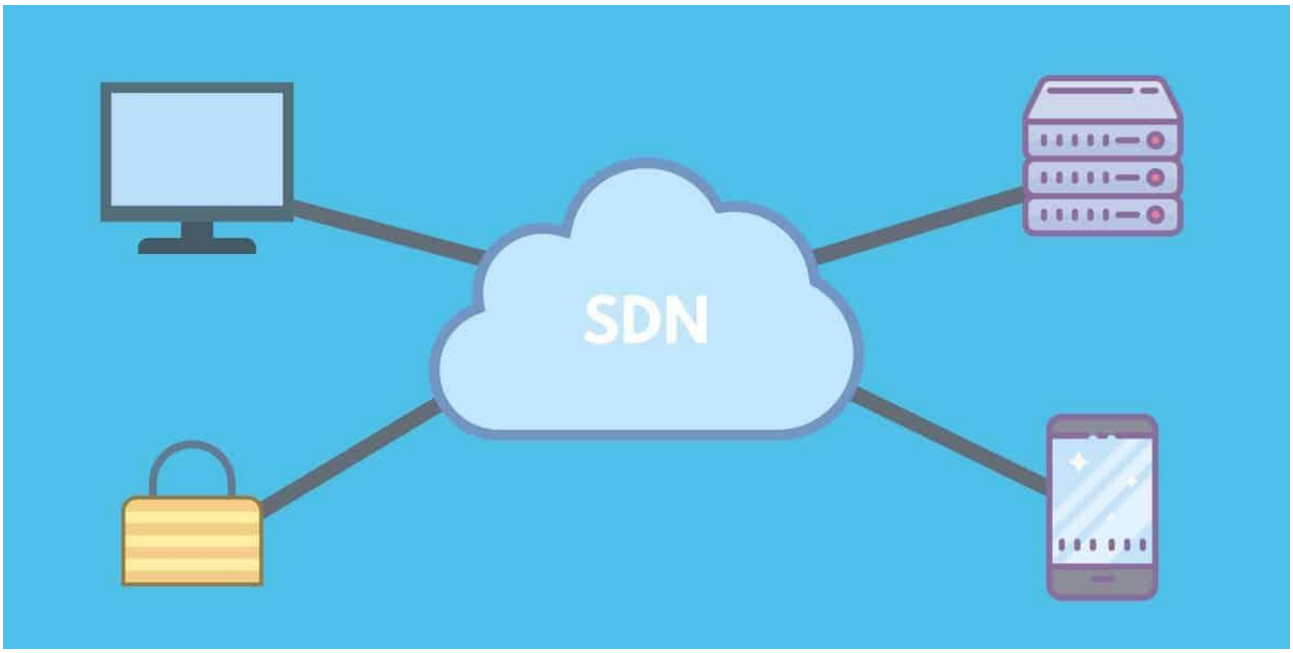

Рис. 1.3 Архітектурний каркас ETSI NFV.

Програмно-визначена мережа стала одним із найпопулярніших способів для організацій розгортати програми. Ця технологія допомогла організаціям швидше розгортати програми та зменшити загальну вартість розгортання. SDN дає мережевим адміністраторам можливість керувати та надавати мережеві послуги з централізованого місця.

Однією з причин популярності SDN була кількість проблем, пов’язаних із підтримкою традиційної застарілої мережі. За останні кілька років потреби сучасних підприємств експоненціально зросли, і фізична інфраструктура не справляється з ними. Саме в цьому середовищі віртуалізовані рішення, такі як SDN, почали розвиватися.

Ні для кого не секрет, що сучасні технології випередили апаратне забезпечення, налаштоване вручну. Традиційні мережі просто не можуть відповідати вимогам сучасних корпоративних користувачів. SDN пропонує організаціям бажану альтернативу, де вони можуть розширити свою мережеву інфраструктуру з мінімальними збоями. Сьогодні ми починаємо спостерігати, як компанії розгортають такі рішення SDN, як Cisco Open SDN Controller, Beacon, Brocade SDN Controller і Juniper Contrail.

Традиційні мережі покладаються на фізичну інфраструктуру, таку як комутатори та маршрутизатори, для встановлення з’єднань і належної роботи.

Навпаки, програмна мережа дозволяє користувачеві контролювати розподіл ресурсів на рівні віртуальної мережі через площину керування. Замість того, щоб взаємодіяти з фізичною інфраструктурою, користувач взаємодіє з програмним забезпеченням для створення нових пристроїв.

З цієї точки зору адміністратор може визначати мережеві шляхи та активно налаштовувати мережеві служби. SDN також має більше можливостей для зв'язку з апаратними пристроями по всій мережі, ніж традиційний комутатор. Основну різницю між ними можна описати як віртуалізацію. SDN віртуалізує всю вашу мережу. Віртуалізація створює абстрактну версію вашої фізичної мережі, яка дозволяє надавати ресурси з централізованого розташування.

У традиційній мережі площина даних повідомляє вашим даним, куди їм потрібно йти. Подібним чином у традиційній моделі мережі контрольна площина розташована всередині комутатора або маршрутизатора. Розташування площини керування особливо незручне, оскільки мережеві адміністратори не мають легкого доступу, щоб диктувати потік трафіку (особливо в порівнянні з SDN).

Під SDN площина управління стає програмною, і до неї можна отримати доступ через підключений пристрій. Це означає, що адміністратор може контролювати потік мережевого трафіку з централізованого інтерфейсу користувача з більшою ретельністю. Це дає користувачам більше контролю над тим, як працює їх мережа. Ви також можете змінити налаштування конфігурації мережі, не виходячи з централізованого концентратора. Управління конфігураціями таким чином є особливо корисним для сегментації мережі, оскільки користувач може швидко обробляти багато конфігурацій.

Причина, чому SDN стала альтернативою, полягає в тому, що вона дозволяє адміністраторам миттєво надавати ресурси та пропускну здатність. Це робиться при цьому, усуваючи вимогу інвестувати в більше фізичної інфраструктури. Навпаки, традиційна мережа потребуватиме нового обладнання, якщо її пропускна здатність мережі збільшиться. Традиційна модель — купувати більше обладнання, а не натискати кнопку на екрані.

Програмно-визначені мережі — це підхід до віртуалізації та контейнеризації мереж, який допомагає оптимізувати мережеві ресурси та швидко адаптувати мережі до мінливих потреб бізнесу, сфер застосування та трафіку. Вони працюють шляхом поділу площин управління та обробки даних мережі для створення програмованої інфраструктури.

За допомогою програмно-визначуваної мережі функції мережевої оркестрації, управління, аналітики та автоматизації стають завданням контролерів SDN. Контролери можуть використовувати масштабування, продуктивність та доступність сучасних ресурсів хмарних обчислень та сховищ. Контролери SDN все частіше будуються на платформах з відкритими стандартами та API, що дозволяє їм організовувати та контролювати мережеве обладнання від різних постачальників та керувати ним.

SDN забезпечує багато переваг для бізнесу. Поділ шарів управління та транспортування даних підвищує гнучкість та прискорює процес виходу на ринок нових додатків. Прискорене реагування на проблеми та збої підвищує доступність мережі. Нарешті, програмування дозволяє IT-організаціям легше автоматизувати мережеві функції та знизити експлуатаційні витрати.

Однією з головних переваг SDN є можливість централізованого керування мережею. У двох словах, SDN віртуалізує як дані, так і площину керування мережею, дозволяючи користувачеві надавати фізичні та віртуальні елементи з одного місця. Це надзвичайно корисно, оскільки моніторинг традиційної інфраструктури може бути складним, особливо якщо є багато розрізнених систем, якими потрібно керувати окремо. SDN усуває цей бар’єр і дозволяє адміністратору за бажанням деталізувати дані.

Хорошим побічним ефектом централізованого надання є те, що SDN дає користувачеві більше масштабованості. Маючи можливість надавати ресурси за бажанням, ви можете миттєво змінити свою мережеву інфраструктуру. Різниця в масштабованості є вражаючою, якщо порівнювати її з традиційною мережею, де ресурси потрібно купувати та налаштовувати вручну.

Незважаючи на те, що рух до віртуалізації ускладнив мережевим адміністраторам захист своїх мереж від зовнішніх загроз, це принесло з собою

величезну перевагу. Контролер SDN забезпечує централізоване розташування для адміністратора, щоб контролювати всю безпеку мережі. Хоча це відбувається за рахунок того, що контролер SDN стає мішенню, він надає користувачам чітке уявлення про свою інфраструктуру, за допомогою якої вони можуть керувати безпекою всієї своєї мережі.

Розгортання SDN дозволяє адміністратору оптимізувати використання апаратного забезпечення та працювати ефективніше. Користувач може за бажанням призначити активне обладнання новому призначенню. Це означає, що ресурси можна ділитися відносно легко. Це перевершує застарілу мережу, де апаратне забезпечення обмежується однією метою.

Однією з проблем віртуалізації будь-якої інфраструктури є затримка, яка виникає в результаті. Швидкість вашої взаємодії з пристроєм залежить від кількості доступних віртуалізованих ресурсів. Ваш сервіс залежить від того, як гіпервізор розподіляє ваше використання (що може додати затримку). Кожен активний пристрій у мережі впливає на доступність мережі. Ситуація буде посилюватися в майбутньому, оскільки все більше пристроїв Інтернету речей (IoT) вийдуть на ринок і почнуть включатися в суміш.

Навіть якщо ви можете керувати службами пристроїв у всій мережі, ви не можете керувати самими пристроями. Хоча на перший погляд це може здатися незначною деталлю, вона дуже важлива для масштабування мережі. Усі ці пристрої потрібно часто контролювати, виправляти та оновлювати, щоб залишатися в робочому стані. Як наслідок, важливо мати на увазі, що залишається велика кількість вимог до обслуговування, які SDN не враховує.

Хоча традиційні мережі можуть мати свої обмеження, існує стандартизований консенсус щодо загроз безпеці та процедур. На даний момент такого консенсусу щодо SDN не існує. Хоча існує багато постачальників рішень SDN, для багатьох адміністраторів питання безпеки SDN є незвіданою територією. Таким чином, може бути дуже важко підтримувати цілісність служби SDN від зовнішніх загроз, якщо ви не маєте необхідних знань для захисту системи.

Програмно-визначена мережа добре поєднується з іншою технологією – віртуалізацією мережевих функцій (NFV). Віртуалізація мережних функцій (NFV) — це архітектурна структура, яка була створена Європейським інститутом стандартів телекомунікацій(ETSI), що відокремлює мережні функції від власних апаратних пристроїв, дозволяючи їм працювати у програмному забезпеченні на стандартних серверах x86. NFV надає можливість віртуалізації мережевих функцій на основі пристроїв, таких як міжмережеві екрани, пристрої балансувальння навантаження та прискорювачі мережі WAN. Централізований контроль, який забезпечує SDN, може ефективно організовувати ці віртуальні функції мережі (VNF) та керувати ними за допомогою NFV. Деякі переваги NFV аналогічні перевагам віртуалізації серверів та хмарного середовища.

- Зниження капітальних (capex) та операційних витрат (opex) за рахунок нижчої вартості обладнання та меншого використання простору, електроенергії та охолодження.
- Швидший час виведення на ринок (TTM), оскільки віртуальні машини та контейнери легше та простіше розгортати, ніж апаратне забезпечення.
- Підвищення рентабельності інвестицій (ROI) за допомогою нових послуг.
- Можливість у короткі сроки збільшувати та зменшувати ємність при необхідності (еластичність).
- Відкритість ринку віртуальних пристроїв і постачальників чистого програмного забезпечення.
- Можливість тестування та впровадження нових та інноваційних послуг віртуально та зменшуючи ризики.

Головним фактором є економія коштів порівняно з використанням спеціалізованих апаратних компонентів, таких як NPU, FPGA та ASIC. На першому етапі розгортання NFV не використовуватимуться загальні сервери, натомість використовуватимуться сервери з чіпсетами співпроцесорів для розвантаження таких функцій, як шифрування та стиснення даних. Очікується, що зі збільшенням пропускної здатності мережі спеціалізовані апаратні ресурси

продовжуватимуть використовуватися для задоволення вимог щодо продуктивності лінійної швидкості. У деяких випадках ці ресурси будуть вбудовані в набори мікросхем або в самі процесори. Але це може збільшити витрати на процесори, які використовуються в багатьох інших програмах, де NFV не потрібен. Одним із рішень є використання гнучкої архітектури стелажного масштабування, яку ми обговорювали раніше в цій главі. Використовуючи ту саму структуру з низькою затримкою та високою пропускною здатністю, необхідну для дезагрегації пам’яті всередині стійки, можна використовувати спеціалізовані модулі спільної обробки NFV відповідно до вимог робочого навантаження. Це дозволить мережевим адміністраторам націлити ресурси NFV на такі програми, як безпека та балансування навантаження, де це необхідно в центрі обробки даних. Ці модулі разом з іншими модулями, які ми описали раніше, можуть бути динамічно перерозподілені в міру зміни робочого навантаження з часом за допомогою рівня оркестровки програмного забезпечення центру обробки даних.

Структура архітектури NFV, розробленої ETSI, показано на рис.1.1.

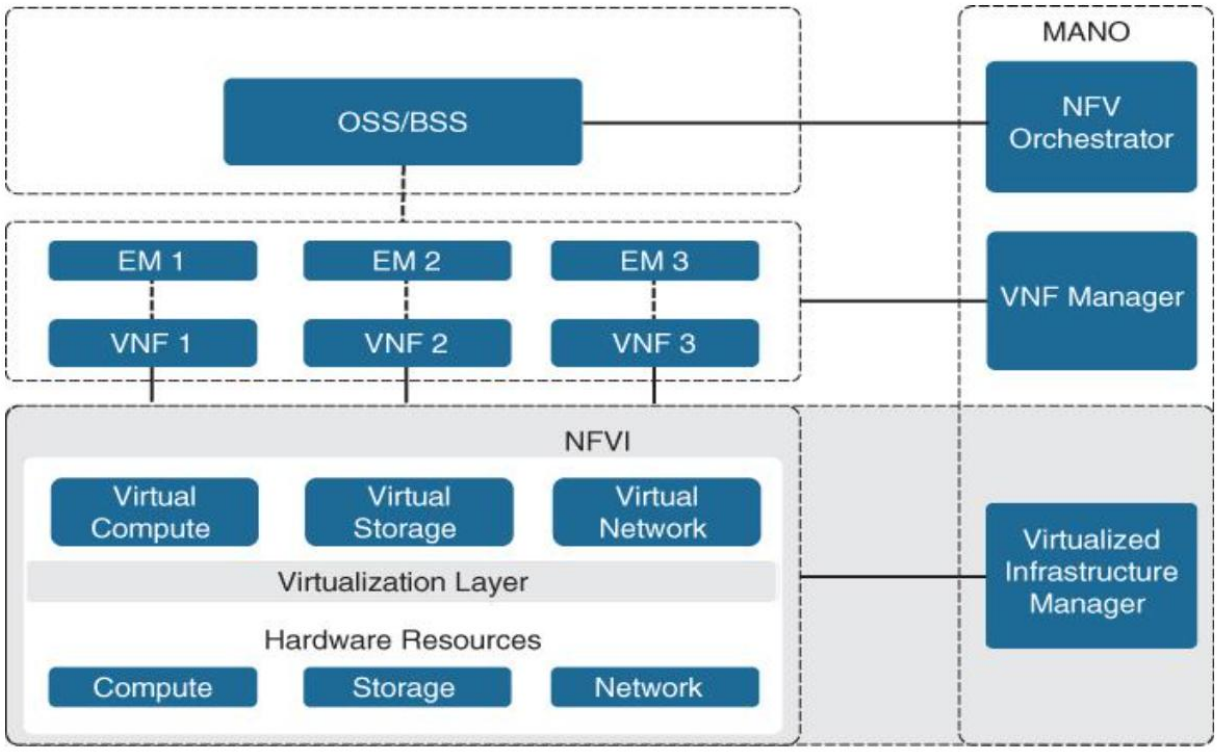

Рис. 1.4 Архітектурний каркас ETSI NFV.

SDN полегшує управління сильно розподіленими фізичними мережевими інфраструктурами за допомогою програмованості та централізованого управління. IT-організації можуть керувати наскрізною мережею у програмному забезпеченні з централізованої точки та автоматизувати деякі аспекти мережевих операцій. Таким чином, SDN прискорюють розгортання мережі, зміну конфігурації та усунення неполадок, що знижує експлуатаційну складність та вартість. Вони також прискорюють вирішення проблем та збоїв у мережі для покращення часу роботи та взаємодії з користувачем.

Програмне забезпечення для керування площиною SDN від мережевого обладнання, що дозволяє мережним адміністраторам використовувати API для програмованого масштабування та віддаленого керування підключеними пристроями. SDN забезпечують загальний інтерфейс для керування всією розподіленою мережею, що дозволяє IT-командам відмовитися від необхідності модернізації, усунення несправностей та керування кожним пристроєм окремо.

1.14 Концепція тестового середовища

Немає простого випробувального середовища для створення нових протоколів динамічної маршрутизації. Наразі є кілька проектів для імітації мереж на звичайних комп'ютерах, але вони не вносять кардинальних змін до протоколу та навантажують систему речами, які не потрібні під час тестування маршрутизації (наприклад, канальний рівень). Більшість існуючих проектів використовують контейнеризацію або потребують використання віртуалізації для розгортання на різних фізичних пристроях чи віртуальних системах.

Через це було прийнято рішення створити просту систему для того, щоб мати можливість тестувати протоколи динамічної маршрутизації. Ця система працює на одному комп'ютері і з її допомогою з’являється можливість швидкого та легкого створення та внесення змін у різних протоколах маршрутизації з подальшою перевіркою. Пізніше можна реалізувати розроблені методи маршрутизації, що будуть ефективніші за вже існуючі.

2 СКЛАД СИМУЛЯЦІЇ

Ця робота була заснована на класичному мережному пристрої – маршрутизаторі. Первинну функцію маршрутизатора можна розділити на дві області: створення мережевих карток і маршрутизація пакетів між мережами. Для побудови карт мережі маршрутизатори зазвичай використовують протоколи статичної чи динамічної маршрутизації. Завдяки протоколам динамічної маршрутизації маршрутизатори надають іншим мережевим пристроям інформацію не тільки про топологію мережі, а й про її зміни[7]. Статична маршрутизація не адаптується до змін у мережі. Обидві моделі виконують завдання побудови карти мережі як таблиці маршрутизації.

У процесі маршрутизації маршрутизатори розглядають кілька альтернативних шляхів досягнення одного пункту призначення. Ці альтернативи виникають через вбудовану надмірність у більшості мережних схем. Потребується кілька шляхів, тому що у разі збою одного шляху до даних стають доступними і інші альтернативні шляхи. Для розробки системи симуляції були використані:

- Мова програмування Python
- Модулі threading, socket, json, tabulate, ipaddress для Python

Симуляція маршрутизатора та його функцій складається з безлічі об'єктів, які взаємодіють між собою [8].

Об'єкти симуляції:

— Об'єкт **Simulation** – це об'єкт, що потрібен для зберігання інших об'єктів, з яких складається маршрутизатор. Його можна використовувати для запуску всіх маршрутизаторів разом методом *start_routers*, та щоб вивести таблицю інтерфейсів зі статусами та іншою інформацією про об'єкти інтерфейсів *print(simulation)*.

```python
class Simulation:

    def __init__(self):
        self.routers = {}

    def add_router(self, router_name, router_interfaces):
        self.routers[router_name] = Router(router_name, router_interfaces)

    def start_routers(self):
        for router in self.routers:
            self.routers[router].start()

    def __str__(self):
        data = {"Router": [], "Number": [], "Port": [], "Broadcast": [], "IP": [], "Status": []}
        for router in self.routers:
            for interface in self.routers[router].interfaces:
                data["Router"].append(interface.hostname)
                data["Number"].append(interface.number)
                data["Port"].append(interface.local_port)
                data["Broadcast"].append(interface.broadcast_port)
                data["IP"].append(interface.get_ip())
                data["Status"].append(interface.status)

        return tabulate(data, headers='keys', tablefmt='grid')
```

Рис. 2.1 Клас Simulation

— Об'єкт **Router** є основою цієї симуляції. Він містить інтерфейси, необхідні маршрутизаторам для обміну інформацією. Об'єкт інтерфейсу має бути переданий під час ініціалізації (створення об'єкта маршрутизатора).

Цей об’єкт також зберігає таблицю маршрутизації та інші дані маршрутизаторів.

```python
class Router:

    def __init__(self, hostname, interfaces):

        self.hostname = hostname

        self.interfaces = interfaces

        self.ip_list = []

        for interface in self.interfaces:
            self.ip_list.append(str(interface.get_ip()))
            interface.hostname = self.hostname
            interface.routing_function = self.__parse_interface_data

        self.__create_broadcast()

        self.data_packet = {"src_ip": "ip", "dst_ip": "ip", "data": "message"}
        self.broadcast_packet = {"src_ip": "ip", "src_port": "port", "data": "message"}

        self.threads = {}

        # self.routing_table = {"network": {"gateway": "0.0.0.0", "interface": 0, "metric": 20}}
        self.routing_table = {}

        self.dynamic_protocol = RIP(router=self)
```

Рис. 2.2 Клас Router

```python
class Router:

    def __init__(self, hostname, interfaces):

        self.hostname = hostname

        self.interfaces = interfaces
        self.ip_list = [str(self.interfaces[i]["interface"].ip) for i in interfaces]

        self.__create_broadcast()

        self.data_packet = {"src_ip": "ip", "dst_ip": "ip", "data": "message"}
        self.broadcast_packet = {"src_ip": "ip", "data": "message"}

        self.connections = {}
        self.threads = {}

        self.routing_table = {ip_network('192.168.0.0/28')}

    def __create_broadcast(self):
        for interface in self.interfaces:
            broadcast_port = self.interfaces[interface]["br_port"]
            broadcast_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            broadcast_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
            broadcast_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            broadcast_socket.bind(('localhost', broadcast_port))
            self.interfaces[interface]["br_socket"] = broadcast_socket

    def connect_to_router(self, ip, l_port):

        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind(('localhost', l_port))
        sock.listen(1)
        self.connections[l_port] = {"remote_ip": ip, "remote_port": l_port, "connection": sock}
        waiter = threading.Thread(target=self.wait_connection, args=(l_port,))
        waiter.start()
        self.threads[f"{l_port}_waiter"] = waiter

    def wait_connection(self, port):
        connection = self.connections[port]
        sock = connection["connection"]
        conn, addr = sock.accept()
```

Рис. 2.3 Сокети

Об'єкт **Router** містить такі загальнодоступні методи:

- *set_protocol*, який потрібний для налаштування об'єкта протоколу динамічної маршрутизації. Цей протокол працює на маршрутизаторі.

```python
def set_protocol(self, protocol):
    if not isinstance(protocol, RoutingProtocol):
        print("You must inherit 'RoutingProtocol' class")
        return 0

    print(f"{self.hostname} now has {protocol.name}")
```

Рис. 2.4 Метод set_protocol

- *has_ip*, який потрібен для того, щоб перевірити чи маршрутизатор має задану IP-адресу, передану методу.

```
def has_ip(self, message_ip):
    for interface in self.interfaces:
        if interface.get_ip() == message_ip:
            return True
    return False
```

Рис. 2.5 Метод has_ip

- *message_to_interface* потрібен для надсилання повідомлень з інтерфейсу.[9] Передається саме повідомлення, а також номер інтерфейсу, з якого воно надсилається.

```
def message_to_interface(self, message, interface_number, src_ip=None, dst_ip=None):
    connection = self.interfaces[interface_number].get_conn()
    conn = connection["connection"]
    packet = self.data_packet
    packet["data"] = message
    if src_ip:
        packet["src_ip"] = src_ip
    else:
        packet["src_ip"] = str(self.interfaces[interface_number].get_ip())
    if dst_ip:
        packet["dst_ip"] = dst_ip
    else:
        packet["dst_ip"] = str(connection["remote_ip"])
    packet = json.dumps(packet)
    conn.send(bytes(packet, "utf-8"))

def message_to_ip(self, message, ip):
    for interface in self.interfaces:
        if ip == interface.connection["remote_ip"]:
            self.message_to_interface(message, interface.number)
            return 0

    interface_number = self.__find_route(ip)
    self.message_to_interface(message,
                              interface_number,
                              str(self.interfaces[interface_number].get_ip()),
                              ip)
```

Рис. 2.6 Метод message_to_interface

- *message_to_ip* потрібен для надсилання повідомлення на певну IP-адресу. Для відправки необхідно передати IP-адресу та

саме повідомлення. Потім маршрутизатор буде намагатися знайти шлях для надсилання в таблиці маршрутизації або у підключених інтерфейсах.

```
def message_to_ip(self, message, ip):
    for interface in self.interfaces:
        if ip == interface.connection["remote_ip"]:
            self.message_to_interface(message, interface.number)
            return 0

    interface_number = self.__find_route(ip)
    self.message_to_interface(message,
                              interface_number,
                              str(self.interfaces[interface_number].get_ip()),
                              ip)
```

Рис. 2.7 Метод message_to_ip

- *message_to_broadcast* потрібен для надсилання повідомлення на широкомовний порт. Таке повідомлення отримають усі інтерфейси з однаковою широкомовною адресою.

```
def message_to_broadcast(self, message, interface="All"):
    def send_data(interface):
        sock = interface.broadcast_socket
        port = interface.broadcast_port
        packet = self.broadcast_packet
        packet["src_ip"] = str(interface.get_ip())
        packet["src_port"] = interface.local_port
        packet["data"] = message
        packet = json.dumps(packet)
        sock.sendto(bytes(packet, "utf-8"), ('<broadcast>', port))

    if interface != "All":
        send_data(self.interfaces[interface])
    else:
        for interface in self.interfaces:
            send_data(interface)
```

Рис. 2.8 Метод message_to_broadcast

- *set_default_route*, який потрібен щоб встановити маршрут за замовчуванням. Необхідно передати номер інтерфейсу та IP-адресу з доступом до мережі, а також метрику.

Метрики потрібні, щоб знайти найшвидший спосіб надсилання повідомлення.

```
def set_default_route(self, gateway, interface, metric=20):
    self.routing_table['0.0.0.0/0'] = {"gateway": gateway,
                                       "interface": interface,
                                       "metric": metric}
```

Рис. 2.9 Метод set_default_route

- *add_route* потрібний для прокладення маршруту до певної мережі.[10][11] Необхідно передати IP-адресу та маску підмережі віддаленої мережі, а також номер та IP-адресу інтерфейсу, що має доступ до віддаленої мережі, і метрику.

```
def add_route(self, network, netmask, gateway, interface, metric):
    self.routing_table[f'{network}/{netmask}'] = {"gateway": gateway,
                                                  "interface": interface,
                                                  "metric": metric}
```

Рис. 2.10 Метод add_route

- *start* потрібний для запуску самого маршрутизатора. Після того, як ця команда виконана, протокол динамічної маршрутизації почнуть працювати, а маршрутизатор почне прослуховувати широкомовний порт.

```
def start(self):
    self.__listen_broadcast()
    self.dynamic_protocol.start()
    print(f"Router {self.hostname} have started")
```

Рис. 2.11 Метод start

— Об'єкт **Interface** – це інтерфейс на маршрутизаторі, який взаємодіє з іншими інтерфейсами, обмінюючись даними.

```python
class Interface:

    def __init__(self, number, local_port, broadcast_port, interface):
        self.number = number
        self.local_port = local_port
        self.broadcast_port = broadcast_port
        self.interface = interface
        self.status = "Unused"

        self.hostname = None
        self.routing_function = None
        self.broadcast_socket = None
        self.listener = None

        self.connection = {}
```

Рис. 2.12 Клас Interface

Він має такі методи:

- *get_ip* для повернення IP-адреси інтерфейсу.

```python
    def get_ip(self):
        return self.interface.ip
```

Рис. 2.13 Метод get_ip

- *get_conn,* який повертає об'єкт підключення двох інтерфейсів.

```python
    def get_conn(self):
        return self.connection
```

Рис. 2.14 Метод get_conn

- *connect_to_router*, який ініціює певні дії щоб встановити зв’язок з іншим інтерфейсом.

```
def connect_to_router(self, r_ip):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('localhost', self.local_port))
    sock.listen(1)
    self.connection = {"remote_ip": r_ip, "connection": sock}
    waiter = threading.Thread(target=self.wait_connection)
    waiter.start()
    self.listener = waiter
```

Рис. 2.15 Метод connect_to_router

- *wait_connection* для початку очікування відповіді інших інтерфейсів для підтвердження встановленого з'єднання.

```
def wait_connection(self):
    connection = self.connection
    sock = connection["connection"]
    conn, address = sock.accept()
    print(f"Router {self.hostname} CONNECTED to {connection['remote_ip']}")
    self.connection["connection"] = conn
    # print(conn) #
    self.status = "Connected"
    listener = threading.Thread(target=self.listen_conn)
    listener.start()
    self.listener = listener
```

Рис. 2.16 Метод wait_connection

- *accept_connection* приймає запит на підключення до іншого інтерфейсу.

```
def accept_connection(self, r_port, ip):
    connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connection.connect(('localhost', r_port))
    self.connection = {"remote_ip": ip, "remote_port": r_port, "connection": connection}
    # print(connection) #
    self.status = "Connected"
    listener = threading.Thread(target=self.listen_conn)
    listener.start()
    self.listener = listener
```

Рис. 2.17 Метод accept_connection

- *listen_conn* очікує на встановлення з'єднання з іншим інтерфейсом.

```python
def listen_conn(self):
    connection = self.connection["connection"]
    while True:
        message = connection.recv(1024)
        message = message.decode("utf-8")
        if not message:
            break

        self.routing_function(message, self)
```

Рис. 2.18 Метод listen_conn

Щоб встановити з'єднання між двома інтерфейсами, потрібно виконати наступні дії:

1. Викликати метод інтерфейсу *connect_to_router* на першому інтерфейсі, а далі передати IP-адресу віддаленого інтерфейсу, на якому буде створено з'єднання. [12]

Потім перший інтерфейс очікує на встановлення з'єднання між двома інтерфейсами.

2. Викликати *accept_connection* на іншому інтерфейсі, з яким встановлює з’єднання перший інтерфейс, передавши адресу першого інтерфейсу для підключення та порт (*local_port*) для підключення інтерфейсу. [13]

Потім інтерфейс очікує на встановлення з'єднання між двома інтерфейсами.

— Клас **RoutingProtocol** є абстрактним класом для того, щоб створювати протоколи динамічної маршрутизації та визначати основні методи, які мають бути у всіх протоколів динамічної маршрутизації.

```python
class RoutingProtocol:

    def __init__(self, name):
        self.name = name

    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def new_message(self, message, interface):
        pass
```

Рис. 2.19 Клас RoutingProtocol

— Об'єкт **RIP** — це об'єкт, необхідний для того, щоб продемонструвати створення та роботу протоколів динамічної маршрутизації з маршрутизаторами під час симуляції.

```python
class RIP(RoutingProtocol):

    def __init__(self, router):
        RoutingProtocol.__init__(self, "RIP")
        self.router = router
        self.rip_packet = {'type': 'RIP', 'routing_table': None}
```

Рис. 2.20 *Клас RIP*

Він має такі методи:

- *new_message*, який потрібен для обробки вхідних повідомлень, призначених для протоколу динамічної маршрутизації. У цьому випадку це повідомлення, що містить таблицю маршрутизації іншого маршрутизатора.

```
def new_message(self, message, interface):
    routing_table = message['routing_table']
    for network in routing_table:
        metric = routing_table[network]["metric"]
        if network not in self.router.routing_table:
            print(1)
            self.router.routing_table[network] = {
                                    "gateway": interface.get_ip(),
                                    "interface": interface.number,
                                    "metric": metric}
            print(f"Update Routing Table {self.router.hostname} "
                  f"From RIP/ {self.router.routing_table}")
        elif metric < self.router.routing_table[network]["metric"]:
            print(2)
            self.router.routing_table[network] = {
                                    "gateway": interface.get_ip(),
                                    "interface": interface.number,
                                    "metric": metric}
            print(f"Update Routing Table {self.router.hostname} "
                  f"From RIP/ {self.router.routing_table}")
```

Рис. 2.21 Метод new_message

- *get_routing_table_to_send*, що потрібен для обробки таблиці маршрутизації маршрутизатора та надсилання її іншим маршрутизаторам.

```
def get_routing_table_to_send(self):
    r_table = self.router.routing_table
    for network in r_table:
        r_table[network]['metric'] = r_table[network]['metric']+1
        r_table[network]['interface'] = None
        r_table[network]['gateway'] = None
    return r_table
```

Рис. 2.22 Метод get_routing_table_to_send

- *send_route*, який потрібен для того, щоб відправляти підготовлену методом *get_routing_table_to_send* таблицю маршрутизації через усі інтерфейси маршрутизатора (на всіх сусідів у цій мережі).

```
def send_route(self):
    for interface in self.router.interfaces:
        packet = self.rip_packet
        packet['routing_table'] = self.get_routing_table_to_send()
        if packet['routing_table']:
            self.router.message_to_interface(packet, interface.number)
```

Рис. 2.23 *Метод* send_route

- *runner*, надсилає повідомлення кожні *n* секунд.

```
def runner(self):
    while True:
        time.sleep(5)
        self.send_route()
```

Рис. 2.24 Метод runner

- *start*, який потрібен для того, щоб мати можливість запускати протокол. Після того, як виконується запуск циклу з використанням методу *runner* для надсилання таблиці маршрутизації всім сусідам маршрутизаторів у мережі кожні *n* секунд.

```
def start(self):
    runner = threading.Thread(target=self.runner, )
    runner.start()
```

Рис. 2.25 Метод start

3 ВИПРОБУВАННЯ РОБОТИ СИМУЛЯЦІЇ

3.1 Опис топології

Щоб розпочати симуляцію, перш за все потрібно створити усі необхідні об'єкти.

Спочатку створюються списки з об'єктів, що належать інтерфейсу. Для цього потрібно передати наступні параметри:

- Порт інтерфейсу.
- Номер інтерфейсу.
- Широкомовний порт інтерфейсу. Для полегшення роботи в мережі можна використовувати широкомовні порти для встановлення з'єднань між інтерфейсами [15]. Для цього два інтерфейси, які підключаються один до одного, повинні мати однаковий широкомовний порт, що буде унікальним для цих інтерфейсів.
- Об'єкт **IPv4Interface** для зберігання IP-адрес.

```
sim = Simulation()

r1_interfaces = [Interface(0, 9101, 9910, IPv4Interface('10.1.1.1/24')),
                 Interface(1, 9102, 9920, IPv4Interface('10.1.2.1/24')),
                 ]

r2_interfaces = [Interface(0, 9201, 9910, IPv4Interface('10.2.1.1/24')),
                 ]

r3_interfaces = [Interface(0, 9301, 9920, IPv4Interface('10.3.1.1/24')),
                 Interface(1, 9302, 9930, IPv4Interface('10.3.2.1/24')),
                 ]

r4_interfaces = [Interface(0, 9401, 9930, IPv4Interface('10.4.1.1/24')),
                 ]
```

Рис. 3.1 Створення списків інтерфейсів

Потім потрібно викликати методи об'єкта симуляції. Це створить маршрутизатор і дасть змогу додати його до симуляції. Для цього необхідно передати ім'я маршрутизатора та інтерфейси, які було створено на

попередньому етапі. [16][17] Можна створювати скільки завгодно маршрутизаторів, але якщо між інтерфейсами повторюються якісь дані, крім широкомовних портів, то симуляція працюватиме некоректно, тому потрібно продумати правильне створення інтерфейсів.

```
sim.add_router("router1", r1_interfaces)
sim.add_router("router2", r2_interfaces)
sim.add_router("router3", r3_interfaces)
sim.add_router("router4", r4_interfaces)
```

Рис. 3.2 Створення об’єктів чотирьох маршрутизаторів

Для автоматичного з’єднання інтерфейсів з однаковим широкомовним портом між собою потрібно запустити усі створені маршрутизатори та викликати метод *message_to_broadcast("Hello, Lets connect")* на них.

```
sim.start_routers()

time.sleep(1)

sim.routers["router1"].message_to_broadcast("Hello, lets connect")
time.sleep(1)
sim.routers["router2"].message_to_broadcast("Hello, lets connect")
time.sleep(1)
sim.routers["router3"].message_to_broadcast("Hello, lets connect")
time.sleep(1)
print(sim)
```

Рис. 3.3 автоматичне з’єднання при запуску всіх маршрутизаторів

Це формує таку топологію:

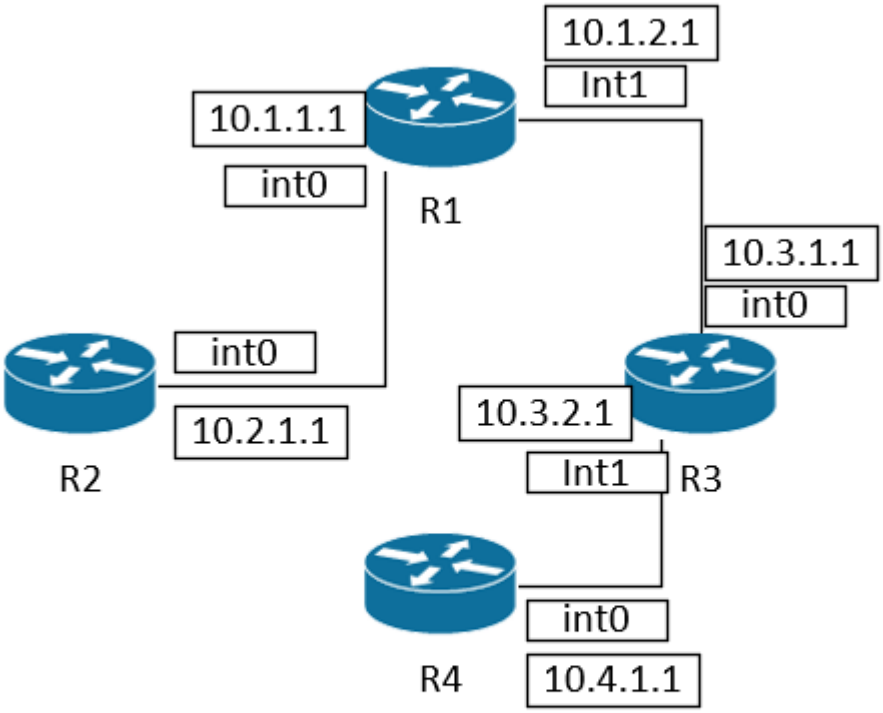

Рис. 3.4 Топологія, створена при симуляції

Побудову топології було виконано за допомогою наступних широкомовних портів

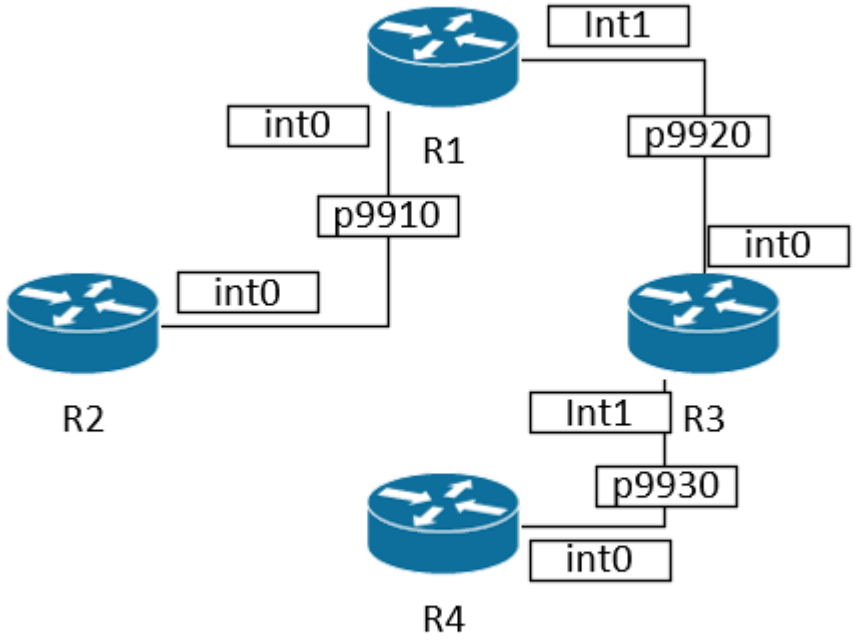

Рис. 3.5 Широкомовні порти під час побудови топології

3.2 Опис взаємодії маршрутизаторів

Такі об'єкти, як сокети, використовуються для обміну інформацією між маршрутизаторами у симуляції. [18]

Сокет – це програмний інтерфейс, що забезпечує обмін даними між процесами. Процеси, пов'язані з такими обмінами, можуть виконуватися як на одному пристрої, так і різних пристроях, об'єднаних мережею. [19][20]

Для того, щоб сокети працювали, необхідні мережні порти, тому вони мають бути вказані для кожного інтерфейсу симуляції. [21]

У симуляції сокети використовуються як альтернатива другому рівню моделі OSI, дозволяючи створювати з'єднання сокетів між двома програмними об'єктами та обмінюватися даними між ними.

Об'єкт інтерфейсу має широкомовні сокети, які прослуховують та надсилають дані до одного порту.

При створенні з'єднання між двома інтерфейсами, створюється сокет між портами цих інтерфейсів. Це дозволяє надсилати дані через один із цих двох інтерфейсів і отримувати їх через інший.

Процес підключення інтерфейсу до сокету – підключення виглядає так:

1. По-перше, є кілька маршрутизаторів з деякими інтерфейсами, що використовують один і той самий широкомовний порт. Як приклад на наступному рисунку.

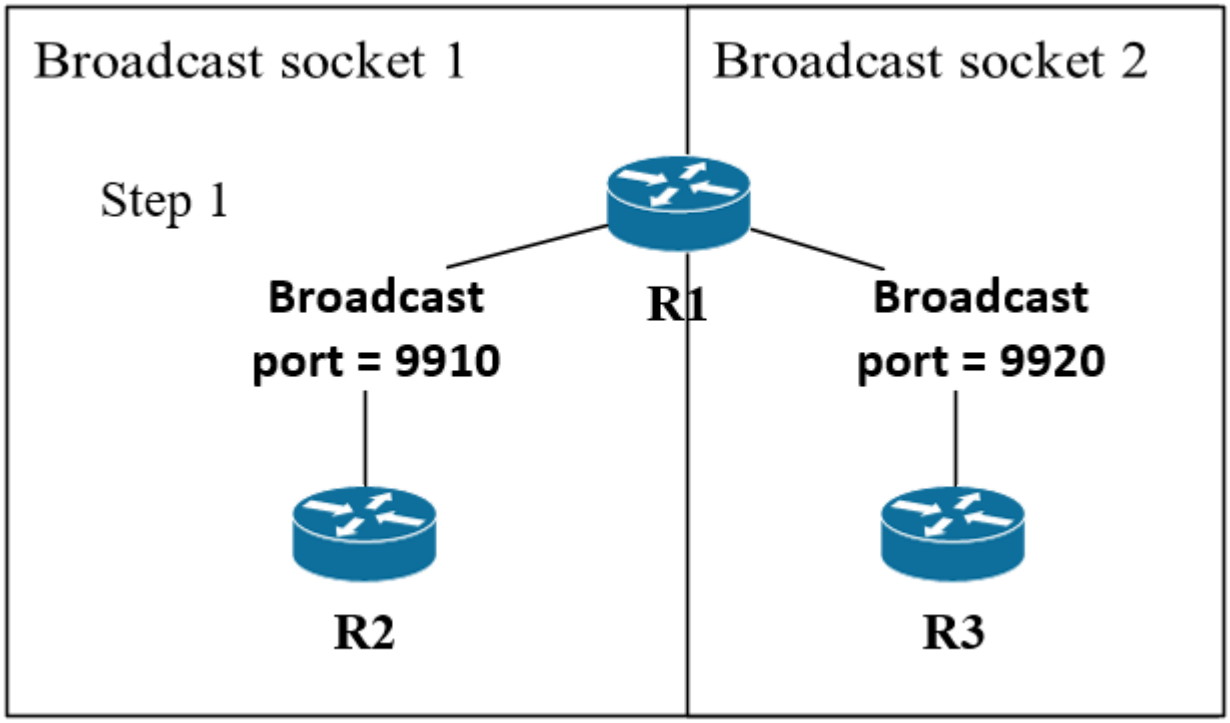

Рис. 3.6 Стан на початку формуванням сокету – з'єднання

2. Потім один із інтерфейсів надсилає повідомлення про встановлення з'єднання з широкомовним сокетом.

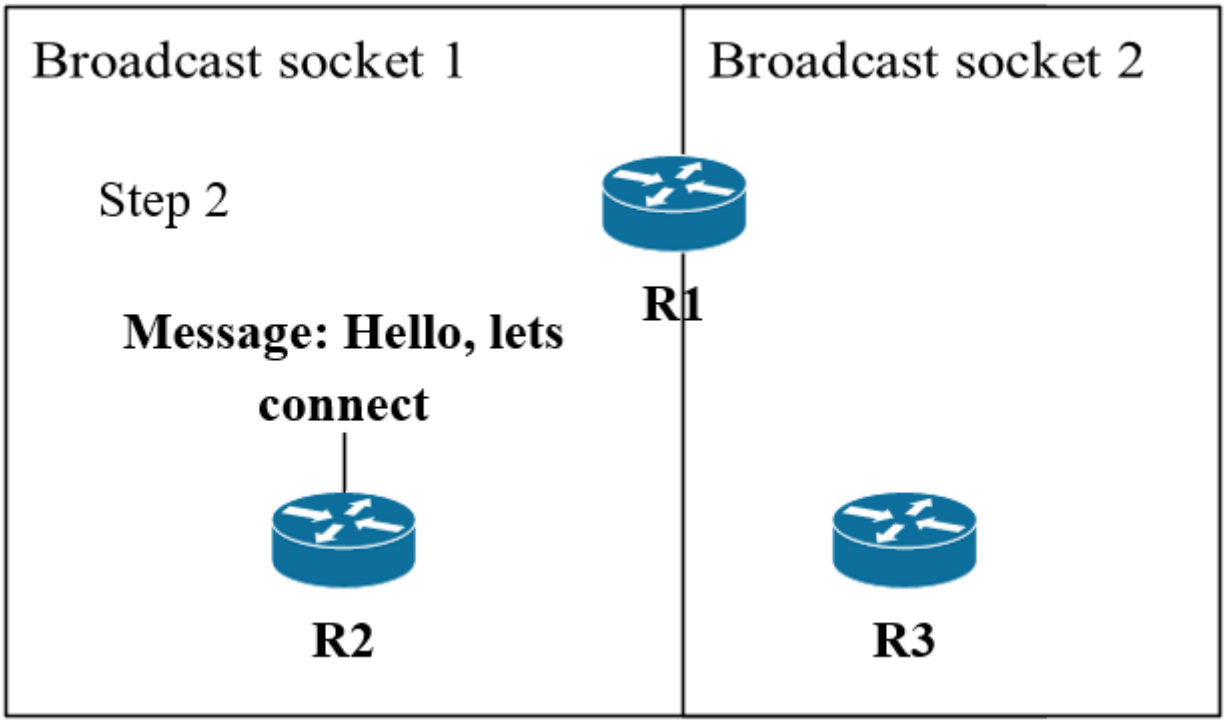

Рис. 3.7 Формування сокетів – другий крок підключення

3. Потім інший інтерфейс отримує повідомлення і, якщо вільний, тобто ще не підключений до іншого інтерфейсу, він відповідає згодою.

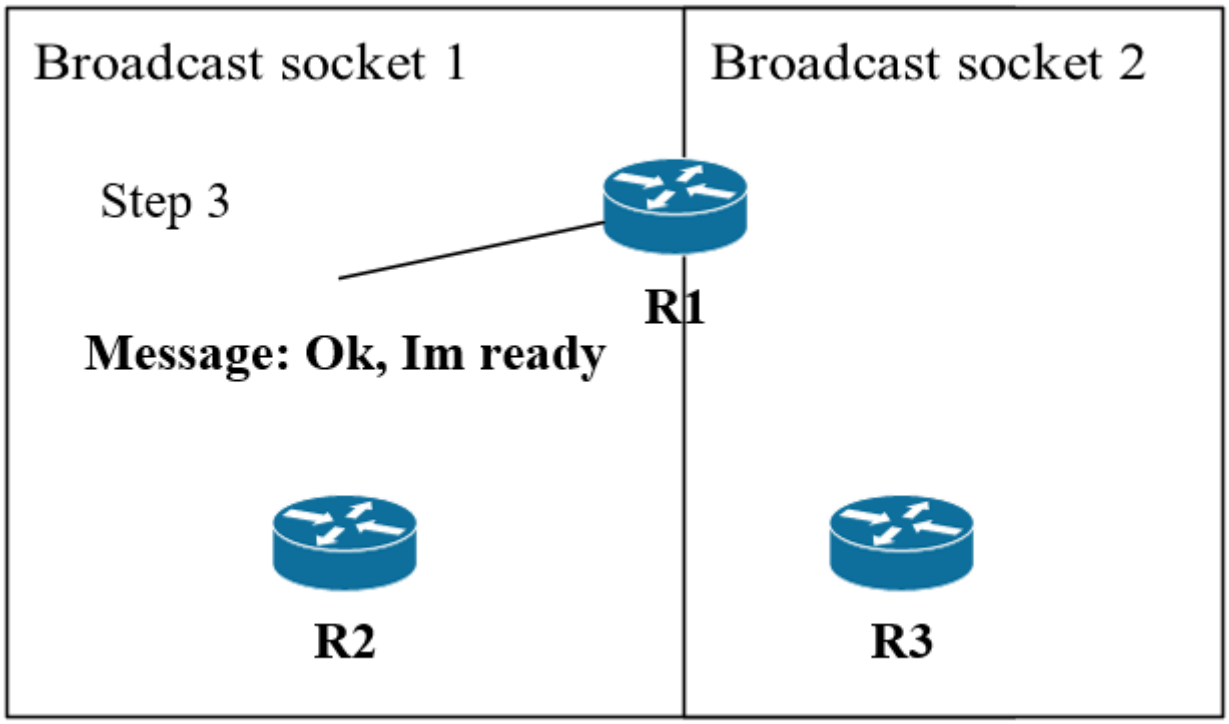

Рис. 3.8 Формування сокетів – третій крок підключення

4. Далі створюються сокети, що є з'єднаннями для кожного інтерфейсу. Тепер ці інтерфейси можна з'єднувати один з одним та обмінюватися даними через створене з'єднання сокетів.

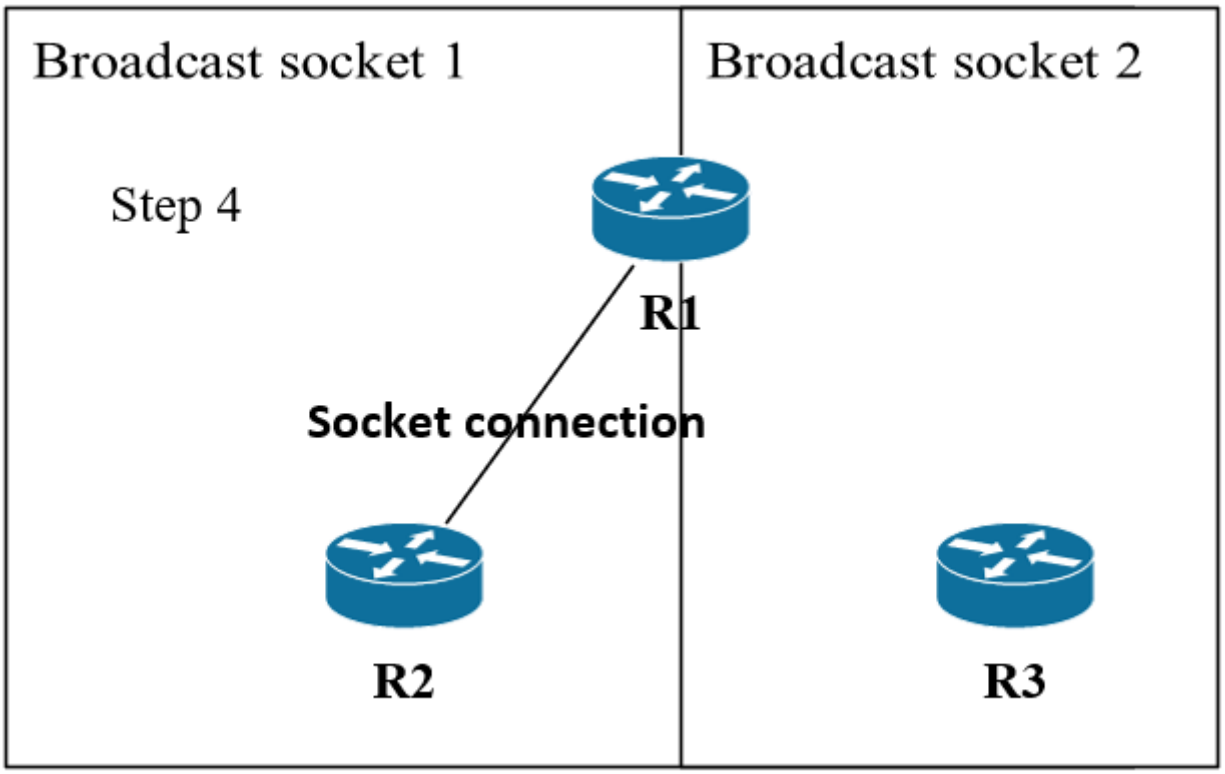

Рис. 3.9 Утворене з’єднання сокетів.

Всі передані дані інкапсулюються у версію пакетів, які виглядають наступним чином:

- Пакети звичайних повідомлень через з’єднання сокетів виглядають так:

data_packet = {"src_ip": "ip", "dst_ip": "ip", "data": "message"}

- Пакет повідомленню, надісланий на широкомовний порт, має наступний вигляд:

broadcast_packet = {"src_ip": "ip", "src_port": "port", "data": "message"}

Крім того, протоколи динамічної маршрутизації можуть мати власні шаблони пакетів для інкапсуляції даних. У симуляції, для прикладу, використовується протокол динамічної маршрутизації RIP. У цьому протоколі дані упаковані у такі пакети:

rip_packet = {'type': 'RIP', 'routing_table': None}

3.3 Опис маршрутизації

Після встановлення з'єднання між двома інтерфейсами повідомлення можуть надсилатися з конкретного інтерфейсу. Це повідомлення приймається іншими інтерфейсами та обробляється за необхідності. [22]

Крім того, коли на певну IP-адресу надсилається повідомлення, маршрутизатор спочатку намагається знайти IP-адресу призначення у списку інтерфейсів, безпосередньо підключених до інтерфейсу маршрутизатора.

При надсиланні даних на IP-адресу маршрутизатор слідує наступною логікою дій:

1. Якщо інтерфейс з IP-адресою призначення безпосередньо підключений до маршрутизатора, він буде відправляти повідомлення з віддаленого інтерфейсу, підключеного до інтерфейсу, якому призначене повідомлення.
2. Якщо IP-адреса призначення не знайдена у списку інтерфейсів, безпосередньо підключених до інтерфейсів маршрутизатора, він намагатиметься знайти мережу призначення у таблиці маршрутизації.
3. Якщо мережа, в якій знаходиться IP-адреса призначення, знаходиться в таблиці маршрутизації, маршрутизатор відправить повідомлення через порт із номером, присутнім у таблиці маршрутизації.
4. Якщо в таблиці маршрутизації немає доступної мережі для одержувача, маршрутизатор повідомляє, що не може знайти спосіб надіслати пакет.

Якщо отримані дані, не призначені для маршрутизатора, тобто IP-адреса одержувача в пакеті не збігається з IP-адресою в таблиці всіх IP-адрес інтерфейсів маршрутизатора, дані переупаковуються і відправляються за

допомогою методу відправки даних на IP-адресу з такою самою послідовністю дій.

Маршрутизатор має такі бази даних (таблиці) щоб зберігати важливу для роботи маршрутизатора інформації:

1. Таблиця IP-адрес для всіх інтерфейсів маршрутизатора. Адреси додаються автоматично під час ініціалізації маршрутизатора.

```
self.ip_list = []

for interface in self.interfaces:
    self.ip_list.append(str(interface.get_ip()))
    interface.hostname = self.hostname
    interface.routing_function = self.__parse_interface_data
```

Рис. 3.10 Таблиця ip-адрес маршрутизатора

2. Таблиця потоків обробки. Це не пов'язано з функціоналом самого маршрутизатора, але є також таблиця, що зберігає всі ініційовані маршрутизатором цикли.

3. Таблиця маршрутизації. Найважливіша база даних для маршрутизаторів, що дає змогу дізнатися, як отримати доступ до інших пристроїв у мережі.

```
# self.routing_table = {"network": {"gateway": "0.0.0.0", "interface": 0, "metric": 20}}
self.routing_table = {}
```

Рис. 3.11 Таблиця маршрутизації

3.4 Маршрутизація трафіку

Щоб вирішити, куди відправляти пакети, маршрутизатори використовують таблиці маршрутизації. Алгоритм дій щодо вибору маршруту було розглянуто у попередньому пункті. [23]

Є два способи налаштування важливих баз даних, таких як таблиці маршрутизації:

1. Статична маршрутизація

Наступні методи викликаються для заповнення таблиці маршрутизації маршрутизатора:

- *set_default_route* має пересилати IP-адресу та номер інтерфейсу, через який маршрутизатор може отримати доступ до будь-якої віддаленої мережі та метрики. Якщо в таблиці маршрутизації є дані, краще буде відправити вищу, ніж зазвичай, метрику, щоб маршрутизатори спочатку спробували надіслати дані до конкретної мережі.
- *add_route*, якому потрібно передавати IP-адресу та маску підмережі віддаленої мережі, а також IP-адресу та номер інтерфейсу, що здійснює доступ до віддаленої мережі, та метрику.

2. Динамічна маршрутизація з використанням протоколів динамічної маршрутизації.

Основною метою створення програми є імітаційне тестування нових протоколів динамічної маршрутизації.

В даний час немає програм, що дозволяють писати алгоритми динамічної маршрутизації мовою програмування Python.

Щоб підключити свій код до симуляції як протокол динамічної маршрутизації, повинні бути виконані такі умови:

- Згенеровані класи динамічної маршрутизації повинні успадковуватися від абстрактного класу RoutingProtocol.
- Створений клас повинен мати два методи, визначені абстрактно в класі RoutingProtocol.

Крім цих умов, в теорії можна зробити будь-що для поліпшення запису в таблицю маршрутизації.

Можна використовувати створений клас RIP як приклад і створити протокол на його основі.

ВИСНОВКИ

Не існує простого тестового середовища для створення нових протоколів динамічної маршрутизації. Існує кілька проектів для моделювання мережі на звичайному комп’ютері, але вони не передбачають значних змін протоколу та завантажують систему речами, які не потрібні під час тестування маршрутизації, наприклад рівень канального рівня.

Враховуючи проблеми сучасних методів і алгоритмів маршрутизації та обмеження у зміні існуючих стандартів, найкращим рішенням є використання NFV.

Архітектура SDN відокремлює керування пересиланням рівня даних від процесів прийняття рішень, пов’язаних із розподіленими програмами та застосуванням політики. Цей підхід дозволяє адаптувати отримані інтерфейси до конкретних потреб цих додатків та інформації, яка має передаватися між ними. Результатом може стати більш ефективна реалізація цільової функціональності на кожному рівні.

Тому було вирішено створити просту систему виключно для тестування протоколів динамічної маршрутизації. Система працює на одному комп'ютері і дозволяє швидко і легко створювати і модифікувати різні протоколи маршрутизації з подальшим їх тестуванням, а також в подальшому реалізовувати розроблені методи маршрутизації, більш ефективні за існуючі.

Емуляція мережевих пристроїв та їх функцій дозволяє розробляти і модифікувати власні стандарти, забезпечуючи велику гнучкість під час проектування як корпоративних, так і світових мереж[24]. У цій роботі було розглянуто створення методу розробки нового мережного функціоналу і модифікації вже існуючих. Також було протестоване середовище для тестування протоколів динамічної маршрутизації, створених самостійно з використанням Python і сокетів, моделюючи дуже просту мережу з одним сервером і кількома клієнтами.

У цій роботі повністю описана послідовність роботи симуляції. Це дозволяє протестувати протоколи динамічної маршрутизації, написані з використанням мови програмування Python, у будь-якій мережі та з будь-якою кількістю маршрутизаторів та з'єднань між ними.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Network routing: algorithms, protocols, and architectures / Medhi D., Ramasamy K. San Francisco: Kaufmann Publishers is an imprint of Elsevier, 2007.- 824 p.

[2] ISO/IEC 10589 Information technology — Telecommunications and information exchange between systems — Intermediate System to Intermediate System intra-domain routing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473), Second edition 2002-11-15.

[3] RFC 2328 OSPF Version 2, April 1998.

[4] Software-defined networking (SDN): a survey. [Електронний ресурс] - Режим доступу: https://onlinelibrary.wiley.com/doi/epdf/10.1002/sec.1737

[5] Network Functions Virtualisation (NFV). [Електронний ресурс] - Режим доступу: https://www.etsi.org/technologies/nfv

[6] Use Containerlab to emulate open-source routers. [Електронний ресурс] - Режим доступу:
https://www.brianlinkletter.com/2021/05/use-containerlab-to-emulate-open-source-routers/

[7] Creating a simple router simulation using Python and sockets. [Електронний ресурс] - Режим доступу:
https://medium.com/swlh/creating-a-simple-router-simulation-using-python-and-sockets-d6017b441c09

[8] Exploring the Functions of Routing. [Електронний ресурс] - Режим доступу: https://www.learncisco.net/courses/icnd-1/lan-connections/functions-of-routing.html

[9] Use Containerlab to emulate open-source routers. [Електронний ресурс] - Режим доступу: https://www.brianlinkletter.com/2021/05/use-containerlab-to-emulate-open-source-routers/

[10] Kravchenko, Y., Dakhno, N., Leshchenko, O., Tolstokorova, A. “Machine learning algorithms for predicting the results of COVID-19 coronavirus infection”, International conference Information Technology and Interactions, IT&I-2020, CEUR Workshop Proceedings, 2021, 2845, pp. 371–381.

[11] Kravchenko, Y., Afanasyeva, O., Tyshchenko, M., Mykus, S. “Intellectualisation of Decision Support Systems For Computer Networks: Production-Logical F-Inference”, International conference Information Technology and Interactions, IT&I-2020, CEUR Workshop Proceedings, 2021, 2845, pp. 117–126.

[12] Mashkov, V.Task allocation among agents of restricted alliance. In Proc. of the 8th IASTED International Conference on Intelligent Systems and Control, ISC, pp. 13-18, 2005.

[13] Mashkov, V. Restricted alliance and coalitions formation. In Proc. of IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2004, pp. 329-332, 2004.

[14] Yudin, O., Suprun, O., Ziubina, R.,Buchyk, S., Frolov, O., Barannik, N. Efficiency Assessment of the Steganographic Coding Method with Indirect Integration of Critical Information: Proceeding of the International Conference on Advanced Trends in Information Theory (ATIT 2019), Kyiv, Ukraine, pp.36-40.

[15] Savchenko, V., Akhramovych, V., Tushych, A., Sribna, I., Vlasov, I. Analysis of Social Network Parameters and the Likelihood of its Constraction. International Journal of Emerging Trends in Engineering Research. Volume 8 No. 2. 2020. pp. 271–276.

[16] Ivanova, D., Starkova, O. and Herasymenko, K. Realization of the Remote Power Management System Based on the Concept of Internet of Things. In Proceedings of the IEEE Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T), 2016, Kharkov: IEEE Ukraine Section, pp. 96-98.

[17] Polianytsia, A., Starkova, O. and Herasymenko, K. Survey of Hardware IoT platforms. In Proceedings of the IEEE 4th International Scientific-Practical

Conference Problems of Infocommunications Science and Technology, (PIC S and T 2017), Kharkov: IEEE Ukraine Section, 2017, p. 369-371.

[18] Starkova, O., Herasymenko, K. and Babailova, Y. Remote Control Systems of Household Appliances. In Proceedings of the IEEE 4th International Scientific-Practical Conference Problems of Infocommunications Science and Technology, (PIC S and T 2017), Kharkov: IEEE Ukraine Section, 2017, pp. 585-588.

[19] Pliushch, O.G. “Gradient Signal Processing Algorithm for Adaptive Antenna Arrays Obviating Reference Signal Presence,” presented at the IEEE International Scientific-Practical Conference PIC S&T, Kyiv, Ukraine, October 8–11, 2019, p. 190.

[20] Dudnik, A., Daria, P., Kobylchuk, M., Domkiv, T., Dahno, N., Leshchenko, O. (2020, November). Intrusion and Fire Detection Method by Wireless Sensor Network. In 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT)pp. 211-215.

[21] Leshchenko, O., Trush, O., Dahno, N., Dudnik, A., Kazintseva, K., & Kovalenko, O. (2020, November). Methods for Predicting Adjustments to the Rates of Modern “Digital Money”. In 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT), pp. 222-226).

[22] V. Bondarenko, “Subjective-probability approach to design an expert system for assessment of states of complex systems in conditions of non-regular destructive influences”, 2019 IEEE International Conference on Advanced Trends in Information Theory. 18.12.2019-20.12.2019. Kiev Ukraine. pp. 183-186.

[23] K.Park, K.Lee, S.Park, H.Lee, “Telecommunication node clustering with node compatibility and network survivability requirements” Management Science, vol. 46(3), 2000, pp.363-374.

[24] Телекомунікаційні мережі й технології: Учеб. посібник / В.Г. Кривуца, С.Н. Скляренко, Т31 А.П. Улеев і ін. : Під ред. В.Г.Крівуци. – Харків: ТОВ “Компанія СМІТ”, 2007. – 324 с.

[25] О.В. Лемешко, В.А. Лошаков, В.В. Поповський та ін. Багатоканальний електрозв’язок та телекомунікаційні технології: підручник у 2-х частин. Ч.1 /

О.В. Лемешко, В.А. Лошаков, В.В. Поповський та ін .; за заг. ред. проф. Поповського В.В. – Х .: ТОВ “Компанія СМІТ”, 2010. – 470 с.

[26] Поповський В.В, Лемешко О.В. ті ін. Педагогічний програмний засіб (ППЗ) «Телекомунікаційні системи та мережі». Друге видання. Виправлено та доповнено. 2018. [Електронний ресурс] / Режим доступу: https://www.znanius.com/3533.html