НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
“___”_____20__ р

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп’ютерна інженерія»

на тему: «Система оптимізації трафіку в транспортних мобільних мережах»

Виконав:
студент _IV_курсу, групи__ІО-64__

___Череватенко Роман Валерійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _асистент Калюжний Олександр Олегович _ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант __н. контроль_ _доц. д.т.н. Сімоненко В.П._ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _доц. каф. СПіСКС к.т.н.,доц Марія ОРЛОВА_ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
"___"_____20__ р.

ЗАВДАННЯ

на дипломний проект студенту

Череватенку Роману Валерійовичу

1. Тема проекту «*Система оптимізації трафіку в транспортних мобільних мережах»*

керівник проекту Калюжний Олександр Олегович, асистент, затверджені наказом по університету від «*07*» *травня* 2020р. №__1081-с__

2. Термін здачі студентом закінченої роботи_____*2020р.*

3. Вихідні дані до проекту *технічне завдання, теоретичні дані.*

4. Зміст пояснювальної записки: *опис предметної області, дослідження модифікованого алгоритму балансування трафіку, програма, що виконує оптимізацію навантаження на канали у мережі.*

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

6. Дата видачі завдання *01.09 .2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	*Затвердження теми роботи*	01.09.2019	
2.	*Вивчення та аналіз завдання*	15.09.2019	
3.	*Розробка архітектури та загальної структури систем*	04.10.2019	
4.	*Розробка структур окремих підсистем*	13.12.2019	
5.	*Програмна реалізація системи*	03.02.2020	
6.	*Оформлення пояснювальної записки*	04.04.2020	
7.	*Передзахист*	26.05.2020	
8.	*Захист*	25.06.2020	

Студент Роман ЧЕРЕВАТЕНКО _____

(підпис)

Керівник Олександр КАЛЮЖНИЙ _____

(підпис)

Анотація

В бакалаврському дипломному проекті розроблено модифікований алгоритм балансування трафіку з використанням контролера SDN.

Запропонований алгоритм може бути використаний для балансування трафіку у мобільних мережах різного розміру. Продукт був створений за допомогою використання програмно-конфігурованих мереж та протоколів маршрутизації: ORR та PRL, на мові Python.

Annotation

In the bachelor's diploma project the modified algorithm of balancing of traffic with use of the SDN controller is developed.

The proposed algorithm can be used to balance traffic in mobile networks of different sizes. The product was created using software-configured networks and routing protocols: ORR and PRL, in Python.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ.467200.001 ВП	Відомість проекту	1	
3	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	57	
5	А4	ІАЛЦ.467200.004 Д1	Принципова схема апаратного забезпечення системи	1	
6	А4	ІАЛЦ.467200.005 Д2	Структурна схема пристрою	1	
7	А4	ІАЛЦ.467200.006 Д3	Функціональна схема методу визначення ям на дорогах	1	
8	А4	ІАЛЦ.467200.007 Д4	Текст програми	3	

Зм.		№ документа	Підпис	Дата	ІАЛЦ.467200.001 ВП		
Розробив		ЧереватенкоР,В			Відомість дипломного проекту	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Калюжний О.О					
Н. Контр.		Сімоненко В.П.				НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-64	
Затвердив		Стіренко С.Г.					

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи
освітньо-кваліфікаційного рівня бакалавр

на тему: «Система оптимізації трафіку в транспортних мобільних мережах»

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на дипломний проект бакалавру за темою «Система оптимізації трафіку в транспортних мобільних мережах».

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка модифікованого алгоритму балансування трафіку в транспортних мобільних мережах .

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, бакалаврські роботи інших студентів, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

- Система може обчислювати один шлях одночасно.
- Система не повинна оброблювати данні, якщо вони не відповідають вимогам.

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows 7 та новіші версії.
- Наявність на комп’ютері Python не нижче версії 3.0.

5.3. Вимоги до апаратної частини

- Оперативної пам’яті не менше 512 Мбайт.
- Вільне місце на жорсткому диску не менше 200 Мбайт.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	28.03.2020
Складання і узгодження технічного завдання	03.04.2020
Створення модулів системи, що розробляється	15.04.2020
Тестування окремих модулів системи	25.04.2020
Допрацювання, налагодження і виправлення помилок	01.05.2020
Оформлення документації дипломної роботи	15.05.2020

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проекту
на тему: «Система оптимізації трафіку в транспортних мобільних мережах»

Київ – 2020 року

ЗМІСТ

					ІАЛЦ.467200.003 ПЗ		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*ЧереватенкоР.В*			Система оптимізації трафіку в транспортних мобільних мережах Пояснювальна Записка	*Літ*	*Арк.* 2 / *Аркушів* 57
Перевірив		*Калюжний О.О*					
Н. Контр.		*Сімоненко В.П*				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-64*	
Затвердив		*Стіренко С.Г*					

СПИСОК СКОРОЧЕНЬ

TE – (англ. Traffic engineering) конструювання трафіку.

SDN – (англ. Software-defined network) програмно-конфігурована мережа.

QoS – (англ. Quality of Service) якість обслуговування.

VPN – (англ. Virtual Private Network), або віртуальна приватна мережа.

WSN – (англ. Wireless sensor network) — це розподілена, що самоорганізується мережу.

ORR – опортуністичний протокол маршрутизації.

RPL – протокол маршрутизації для бездротових мереж з низьким енергоспоживанням.

API – (англ. Application programming interface) – це інтерфейс взаємодії між сайтом і серверами.

CLI – інтерфейс командного рядка.

DCN – (англ. Data communication network) — мережеві центри даних.

DNS – (англ. Domain Name System) — це система доменних імен.

LLDP – (англ. Link Layer Discovery Protocol) — протокол канального рівня.

LLDP – (англ. Link Layer Discovery Protocol) — протокол канального рівня.

Control plane – керуючий рівнеь.

Data plane – передаваючий рівень.

Softswitch – програмний комутатор.

Datapath – шлях по якому передаються дані

ВСТУП

В даний час у зв'язку з появою нових мережевих технологій, таких як програмно-визначаючих мережі (SDN), існує необхідність в розробці нових і поліпшенні відомих методів управління мережевими ресурсами і побудовою трафіку.

Зі збільшенням розмірів мережі і значним енергоспоживанням завдання балансування трафіку стає актуальним. У зв'язку з цим основною метою даного проекту є дослідження і розробка методу балансування трафіку в мережах SDN, який забезпечує найбільш рівномірне завантаження каналів передачі інформації.

Аналіз особливостей організації та функціонування SDN для підвищення ефективності балансування навантаження.

Особливості структури SDN дозволяють підвищити ефективність конструювання трафіка. При розробці алгоритмів конструювання трафіка необхідно враховувати особливості будови структури SDN.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ АЛГОРИТМІВ ДЛЯ КОНСТРУЮВАННЯ ТРАФІКУ В МЕРЕЖІ SDN

1.1 Використання WSN мереж

Застарілі мережі досить непрості і недоцільні в їх використанні. Це обумовлено тим фактом, що такі мережі вертикально інтегровані, тобто дані і управління об'єднані разом. Виходячи з цього, можна зіткнутися з проблемою конфігурації великомасштабних мереж. Так як, мобільні технології мають швидкий розвиток, підхід до архітектури, експлуатація мережі, і управління самими мережами потребують внесення серйозних змін.

Проблема балансування навантаження в WSN вивчається вченими вже досить довго. Для вирішення проблеми погано збалансованості навантаження в бездротових мережах було запропоновано багато рішень. Наприклад: BasedonLEACH, animprovedLEACHprotocolin, WSNsisproposed.

Протокол LEACH працює як протокол адаптивної ієрархічної маршрутизації і є енерго економнішим, випадковим чином вибираючи заголовки кластера для рівномірного розподілу енергоспоживання в мережі. У WSN все вузли взаємозалежні при пересиланні пакетів від станції до станції завдяки алгоритму маршрутизації. Маршрути можуть відрізнятися між собою різними ступенями завантаженості, наприклад, в слідстві цього може виникнути незбалансовані конфлікти у різних маршрутів. Ця проблема була проаналізована з точки зору теорії ігор і було вироблено моделювання проблеми вибору шляху в WSN на основі еволюційної гри. Для продовження терміну служби мережі, так званим «перехоплювачів» необхідно вибрати обидві навантаження по збалансованим вузлів. Так само був запропонований опортуністичний протокол маршрутизації (ORR). Слідом був запропонований простий і ефективний RPL на основі використання черги, який здатний забезпечити балансування навантаження і поліпшити продуктивність

доставки пакетів в порівнянні з RPL. Черга призначена для кожного вузла, для вибору його батьківського вузла і для відстані між ними. Так само пропонувалася структура для маршрутизації при обліку балансування навантаження в WSN. Маркування шляхів в мережі використовується для моніторингу мережевого навантаження вузлів. Отже, вузли позначаються як перевантажені, недостатньо завантажені або збалансовані.

Алгоритм пом'якшення знаходить відповідних нових батьків для перемикання з перевантажених вузлів. Механізм маршрутизації дочірнього елемента перевантаженого вузла отримує вказівку переключити його батьківський вузол. Таке пересилання даних динамічної маршрутизації в цілому базується на даних локального вузла, яка легко потрапляє в локальний стимул, в той час як глобальний покращений алгоритм маршрутизації ускладнює розподіл і не підходить для великомасштабних мереж. З появою SDN, функції поділу управління і переадресації забезпечують більш ефективний шлях для алгоритмів маршрутизації WSN. Як показано на рис. 1.1, мережева архітектура на основі SDN буде більш гнучкою, а оновлення програм - більш простими.

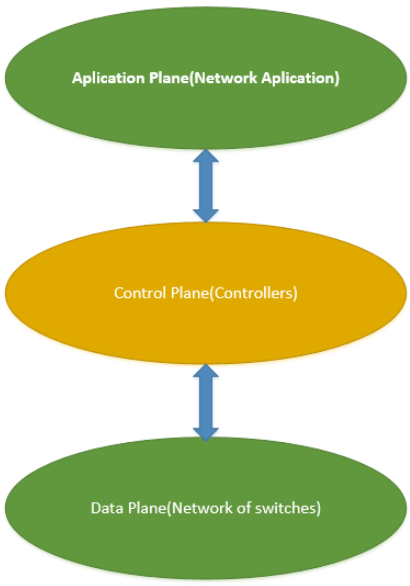


Рис.1.1 Загальна архітектура програмно-визначаючих мереж (SDN)

У комп'ютерних мереж нашого часу досить великі розмірності, а також велике різноманіття обладнання, що використовується ними. У тому числі і мобільні точки доступу. Це явно ускладнює управління таким типом мереж, а також конструювання самого трафіку.

Другим основним завданням на сьогоднішній день є збільшення пропускної спроможності і забезпечення надійності передачі інформації. У зв'язку з цим актуальним є розробка способів організації функціонування мережевих центрів даних і конструювання трафіка (TE) з урахуванням особливостей і переваг SDN.

При розгляді пристрою управління наших днів (Комутатор або роутер) «Шарами», містить в собі 3 компоненти див. рис 1.2.

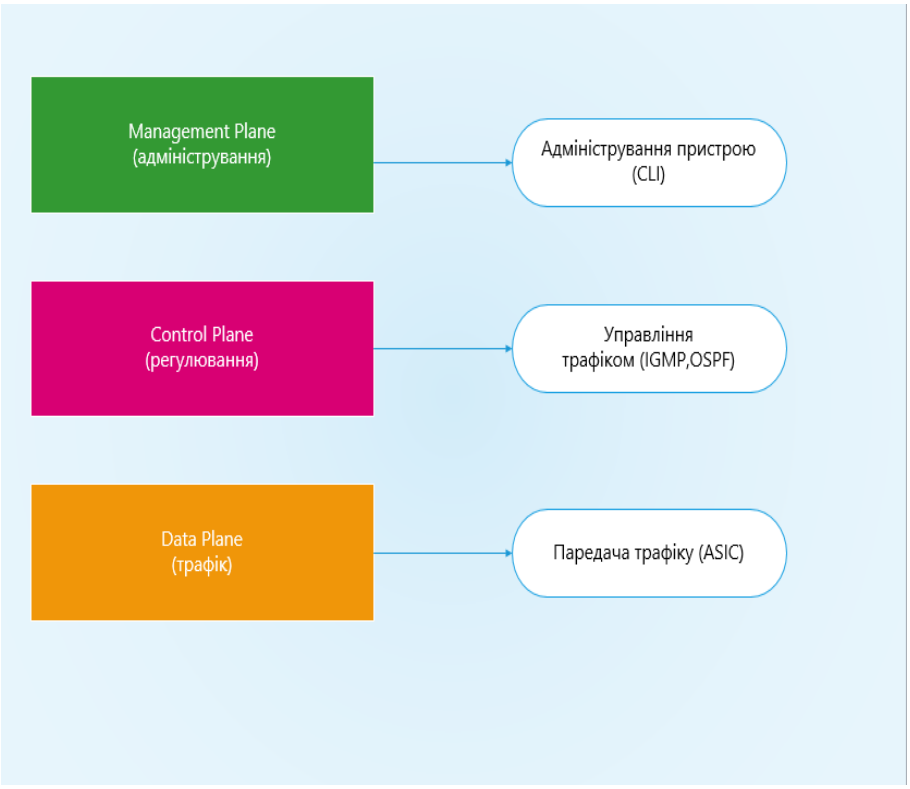


Рис. 1.2 Типовий мережевий пристрій

Рівень управління - це CLI, це API і протоколи управління або якийсь вбудований веб-сервер. Завдання цього етапу забезпечення керованості пристроєм.

1. Рівень управління трафіком - це якийсь інтелект пристрою. Його завдання полягає в автоматичному реагуванні на зміни трафіку. Це відбувається за рахунок роботи різних алгоритмів і самого функціоналу.

Передача трафіку - функціонал, що дозволяє здійснити фізичну передачу даних, що відбувається на рівні мережевих пакетів і мікросхем.

1.2 SDN додатки.

SDN додаток - це інтерфейс оптимізації мережі під певний бізнес додаток (наприклад, Microsoft Lynk) див. рис. 1.3, і його мета - зміна мережі в реальному часі під вимоги обслуговується програми.

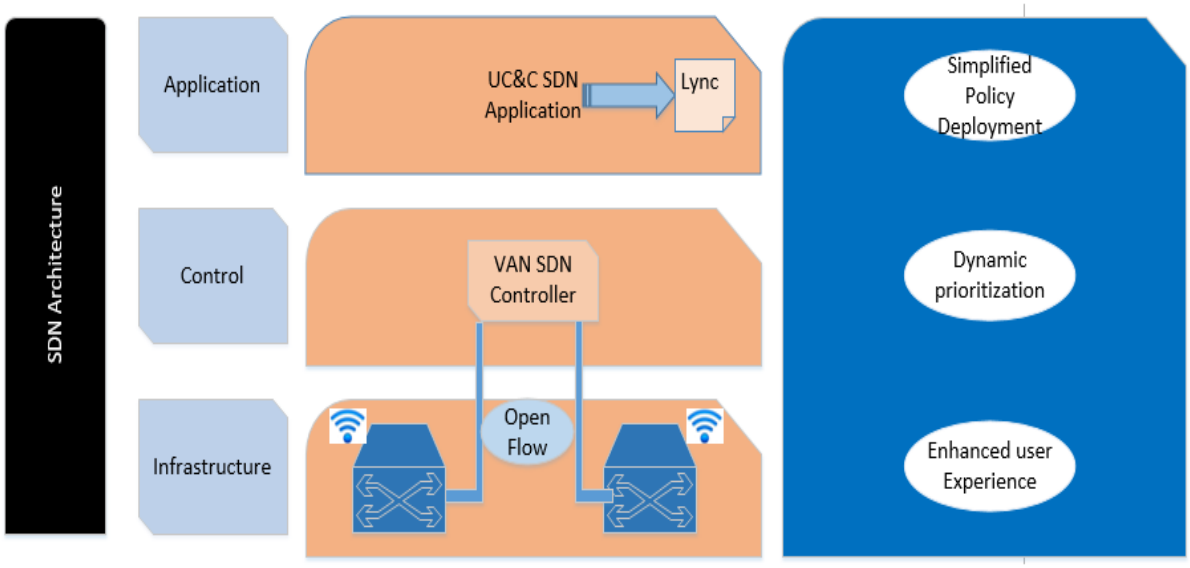


Рис. 1.3 SDN додаток для MS Lynk

В даному випадку (Microsoft Lynk) - як приклад візьмемо зміна QoS мережі між двома телефонними абонентами службовці для передачі відео дзвінка (HD) в системі реального часу без будь-яких перешкод, таких як затримок. Або використання сторонніх програм, наприклад, VPN тунелю між двома абонентам

Відповідно до головним принципам SDN, вся логіка управління знаходиться в контролері, які можуть відстежити роботу всієї мережі. Це не є відносно новою ідеєю. Наприклад, раніше зв'язківці використовували схожу логіку, яку намагалися реалізувати за допомогою модернізації телефонних мереж, саме в слідстві цього відбулася поява «інтелектуальних мереж», а пізніше і комутаторів класу Softswitch.

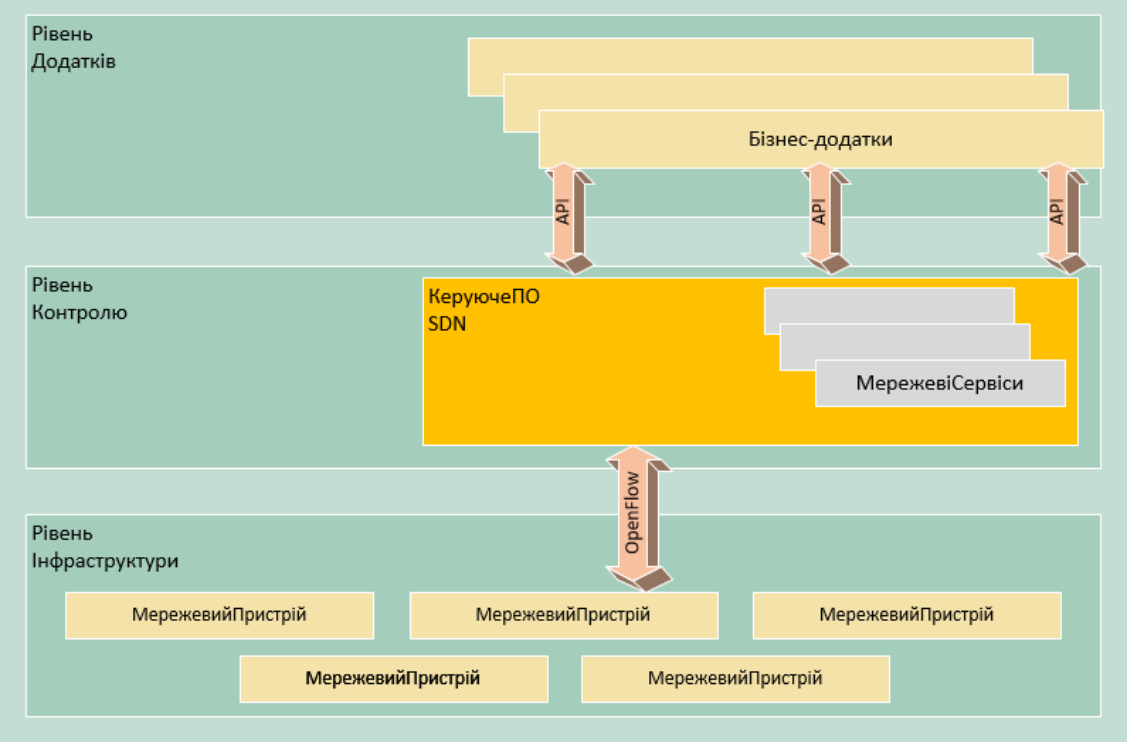


Рис. 1.4 Рівні роботи алгоритму передачі даних

На рис. 1.4 зображений алгоритм передачі даних якій містить в собі 3 рівні, а саме: рівень додатків, рівень контролю та рівень інфраструктури. Кожен з рівнів відповідає за свою частину передачі даних, та містить необхідні компоненти для цього.

У зв'язку зі зростаючими потребами в обчислювальної потужності та обсяги інформації актуальним завданням є підвищення ефективності комп'ютерних мереж. Сучасні комп'ютерні мережі характеризуються великою розмірністю та різноманітним складом обладнання, в тому числі мобільного зв'язку. У зв'язку з цим процес управління сучасними комп'ютерними

мережами ускладняється, зокрема, процедурою балансування трафіку. Для вирішення цих проблем використовується технологія програмно - конфігуруємих мереж (SDN). У SDN площину управління приділяється на окремий пристрій, зване контролером. Контролер є найбільш важливою і фундаментальною частиною архітектури SDN. В даний час, з появою нових технологій SDN, виникає необхідність в розробці нових методів балансування трафіку. Це підвищить ефективність процедури балансування трафіку, завдяки особливостям організації SDN. У порівнянні з традиційною мережею основною перевагою балансування навантаження в SDN є те, що балансування виконується централізовано в контролері SDN. Це дозволяє більш ефективну стратегію розподілу навантаження.

Контролер SDN оновлює інформацію про маршрутизації для комутаторів SDN, оновлюючи їх таблиці маршрутизації, щоб вибрати кращий маршрут з точки зору мінімізації енергоспоживання і перевантаження каналу. У порівнянні з розподіленими методами побудови трафіку і його балансування, централізований метод усуває необхідність обміну службовою інформацією між мережевими комутаторами.

Основна мета програмно-конфігуруються мереж (SDN) - це відділення логіки управління мережею від фізичного рівня (комутаторів і маршрутизаторів) вводячи при цьому можливість програмування мережі.

Використання SDN дає ряд переваг, таких як: підвищення ефективності мережевого обладнання, зниження експлуатаційні витрати, підвищення керованістю мережею. Головною перевагою програмно-конфігуруємих мереж є те, що організація і управління самою мережею відбувається на програмному рівні за допомогою комутаторів і основного контролера SDN.

1.3 Інтернет-Архітектура системи

У сучасній інтернет-архітектурі є скрутним динамічне перенастроювання великої кількості різних шляхів маршрутизації для кожного потоку. Наприклад, в той час, коли пакет надходить на маршрутизатор, здійснюється перевірка його записів таблиць маршрутизації і пересилання його відповідно певних правил, які задає адміністратор. OpenFlow пропонує нову парадигму, яка в більшій частині усуває цей недолік, надаючи нам можливість самим визначати правила маршрутизації, пов'язані з потоками даних, щоб потоки трафіку можна було миттєво змінити, як в середовищі SDN. Всі комутатори в мережі на основі SDN працюють по протоколу OpenFlow. Всі ці комутатори мають підключення до центрального контролера. Контролер здатний отримувати стан мережі в реальному часі з площини даних (сервер і клієнт).

Так само існує коефіцієнт втрати пакетів і доступна смуга пропускання відома як затримка.

Після чого відбудеться аналіз зібраних параметрів QoS і обчислення схеми потокового відео в залежності від смуги пропускання. Демонстрований контролер, який ви бачите на рис. 1.5, пропонує різні інтерфейси і функції, декілька з яких були частиною маршрутизатора в традиційній моделі Інтернету. Рис 1.5 зображує базову мережу підтримуючу протокол OpenFlow, який в свою чергу здебільшого складається з трьох частин: модуля маршрутизації, модуля вимірювання і модуля контролера. Модуль маршрутизації, заснований на контролері прожектора, здатний надавати дані про стан глобальної мережі, також є відповідальним за обчислення оптимального шляху і допустимої пропускної спроможності кожного потоку.

Він взаємодіє з модулем топології, який здатний отримувати і оновлювати топологію мережі в реальному часі. У разі, коли стан мережі не допустимо до вимог даних параметрів QoS, є необхідність в перерахунку маршрутизації модулем агрегації смуги пропускання. Так само, відбувається

генерація і відправка таблиць потоків контролером відповідно потрібного комутатора.

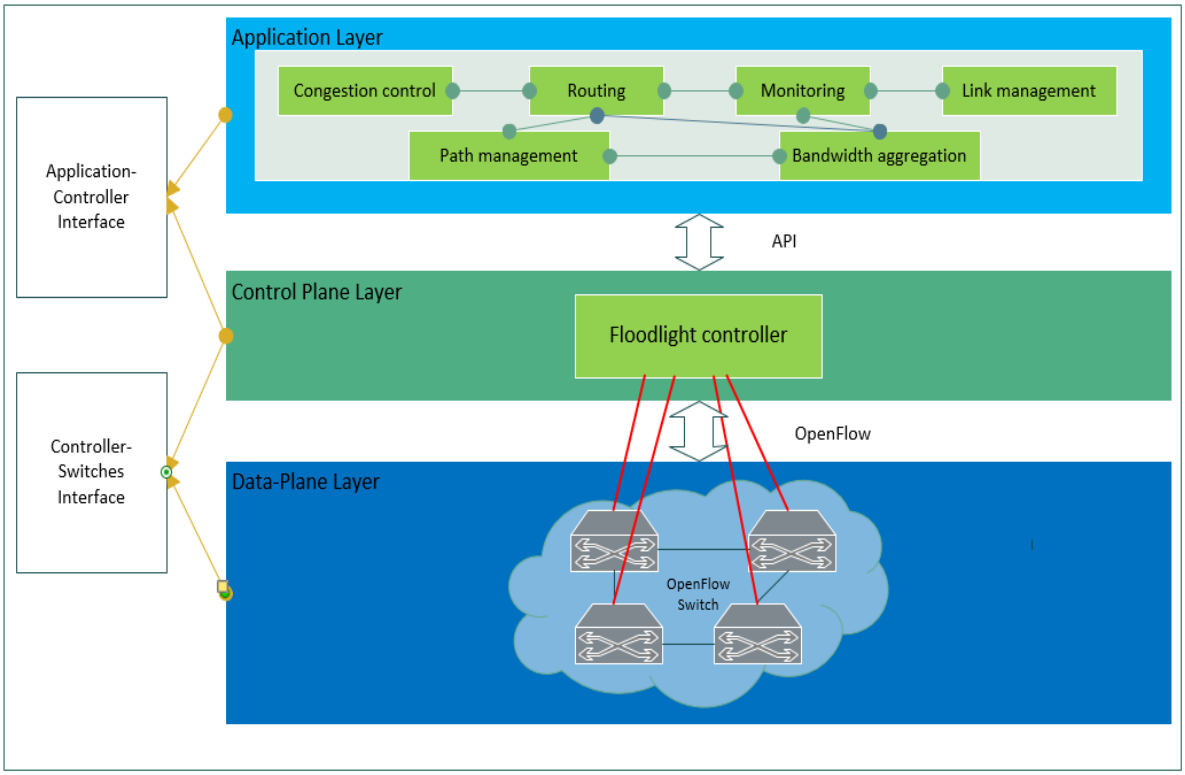


Рис. 1.5 Архітектура системи

Рис. 1.5 ілюструє 2 базових інтерфейси: інтерфейс «контролер-комутатор» і інтерфейс «контролер-додаток». Інтерфейс «контролер-комутатор» полягає в тому, що відбувається підключення до мережі комутаторів з використанням протоколу OpenFlow для відправки таблиць потоків, для визначення топології мережі, а також для отримання даних про стан трафіку. Останнє полягає в тому, що контролер надає відкритий безпечний інтерфейс для постачальників прикладних послуг з метою резервування нових розділів даних. Модуль контролера є відповідальним за стеженням і управлінням даних про стан глобальної мережі для більш вигідного використання мережевих ресурсів завдяки протоколу OpenFlow для здійснення зв'язку з площиною даних. Так само, він генерує і відправляє таблиці потоків на комутатор OpenFlow в залежності від повідомлення запиту,

відправленого клієнтом. Він має основні функціональні модулі, такі як: Перевірка поточного стану мережі з використанням протоколу виявлення канального рівня (LLDP) і підтримує глобальну інформацію про топології

Віртуальна машина з ексклюзивним шляхом маршрутизації для потоку активації.

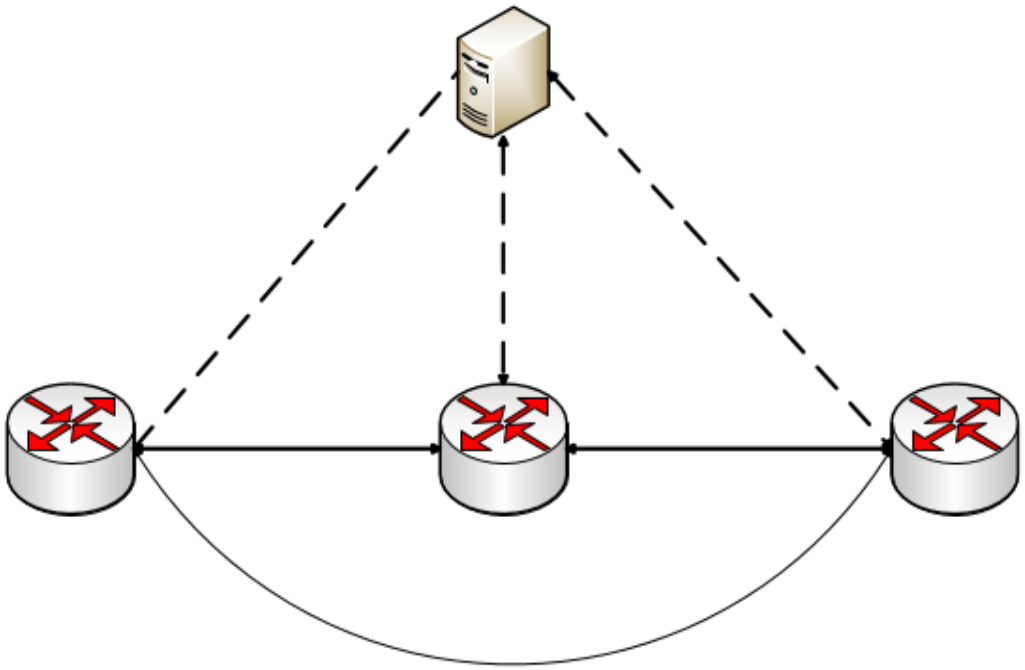


Рис. 1.6 Структура програмно-конфігуруємих мереж

Рис. 1.6 ілюструє базову структуру програмно-конфігуруємих мереж з використанням одного сервера та декількох вхідних приладів.

1.4 Конструювання трафіку і DNS.

Таким способом можна збільшити функціональні можливості конструювання трафіка (TE), а також отримати можливість, централізовано керувати мережевими ресурсами. При такій функціональності SDN контролер містить повну інформацію про структуру мережі та про кожен її вузол, це дозволяє здійснювати оптимізацію всієї мережі по заданих параметрах.

Централізоване управління на основі SDN (див. рис. 1.7) так само дозволяє зменшити час конструювання трафіка і підвищити рівень якості обслуговування трафіку.

DNS (Domain Name System - система доменних імен) - є розподіленою системою, призначеної для зберігання і обробки інформації в доменних зонах. Її початкова мета - це співвіднесення IP-адрес пристроїв в мережі, а також більш зрозумілих для людини символьних імен.

Надання інформації про IP-адреси хостів є не єдиним завданням DNS. Система здатна взаємодіяти з різними типами записів, завдяки яким можлива реалізація великого кола завдань: Балансування навантаження, прив'язки різних сервісів до домену (таких як пошта), переадресація між доменними іменами.

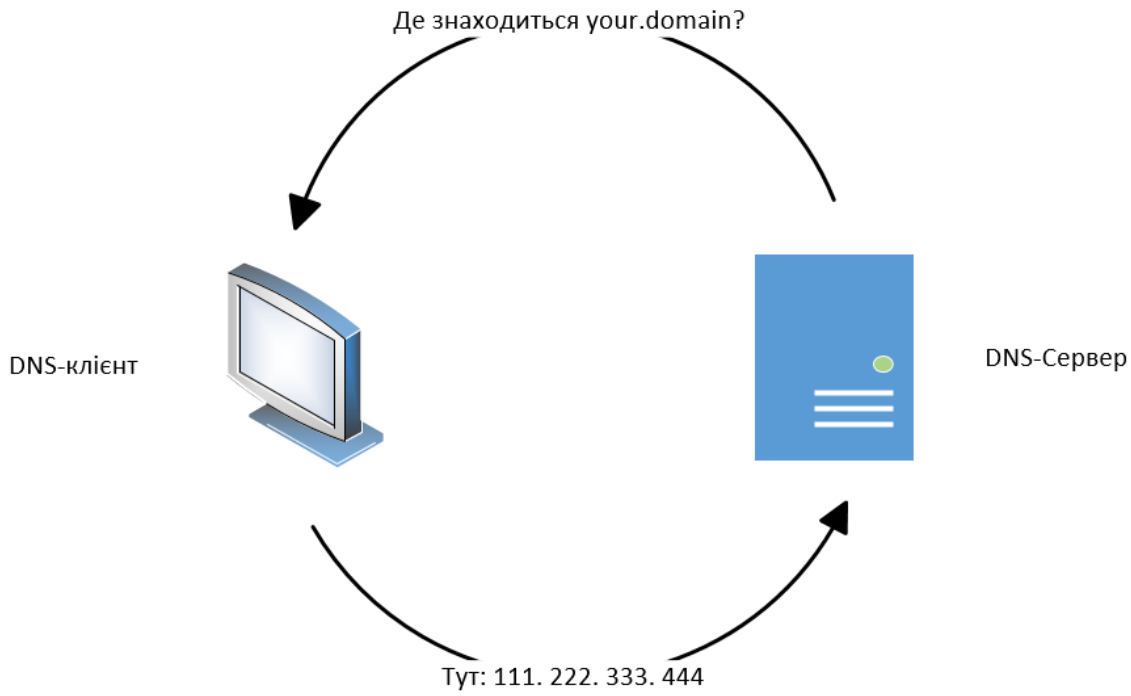


Рис. 1.7 DNS (Domain Name System - система доменних імен)

Основні характеристики DNS:

1. Кешування даних - у DNS-сервера є властивість тимчасово зберігати певну кількість інформації про не делегованих йому доменах з метою зниження загального рівня навантаження.
2. Ієрархічна структура - вузол, який відповідає за певну доменну зону, здатний самостійно управляти нижчими вузлами сервера.
3. Розподіленість зберігання і управління - кожен DNS - сервер повинен мати інформацію тільки по делегованих йому доменів.
4. Резервування - для того, щоб зберігати і обробляти інформацію про одних і тих самих вузлах необхідно кілька DNS - серверів, відокремлених як фізично, так і логічно. Такий підхід гарантує доступність інформації навіть при збої роботи декількох вузлів.

Процес передачі інформації:

Передача інформації здійснюється послідовно від передавального роутера R_i до суміжного роутера R_j у напрямку до роутера R_a.

Суміжний роутер R_j визначається по таблиці маршрутів $Tm(i)$ роутера R_i. Передача інформації здійснюється у відповідності з наступним алгоритмом:

Алгоритм передачі інформації:

1. Призначити початковий роутер R_s передавальним роутером R_i
2. Передавальний вузол по своїй таблиці маршрутів визначає наявність шляху з допустимим значенням QoS до приймаючого вузла.
3. При наявності допустимих шляхів в таблиці маршрутів Tm (i) здійснюється перехід до пункту 5.
4. При відсутності допустимого шляху передавальний вузол видає запит контролеру SDN про формування нових або реконфігурації існуючих шляхів.
5. Серед допустимих шляхів роутер R_i вибирає шлях з допустимою метрикою і мінімальним значенням завантаження шляху до роутера R_a.
6. Роутер R_i передає пакет даних до вибраного суміжному роутеру R_j

7. Якщо пакет не останній, то перехід до пункту 2.

Завершення процедури передачі інформації.

1.5 Процедура конструювання трафіка

Передавальний вузол по своїй таблиці маршрутів визначає наявність шляху з допустимим значенням QoS до приймаючого вузла.

При наявності допустимих шляхів перехід до пункту 4.

При відсутності допустимого шляху передавальний вузол видає запит контролеру SDN про формування нових або реконфігурації існуючих шляхів.

Серед допустимих шляхів передавальний вузол вибирає шлях з допустимою метрикою і мінімальним значенням Di.

Передавальний вузол повідомляє контролеру SDN інформацію про обраний шлях і починає процес передачі інформації.

Контролер SDN коригує метрики даного шляху і суміжних з ним шляхів.

Контролер SDN оновлює метрики в вузлах обраного шляху і суміжних з ним шляхів.

Процес передачі інформації завершується. Передавальний вузол інформує про це контролер SDN.

Контролер SDN коригує метрики даного шляху і суміжних з ним шляхів.

Контролер SDN оновлює метрики в вузлах обраного шляху і суміжних з ним шляхів.

Завершення процедури оркестровки трафіку.

Операція реконфігурації здійснюється в динамічному режимі.

Контролер SDN отримує від мережевих комутаторів інформацію про зміну стану каналів, змінюючи таблиці маршрутизації відповідних мережевих комутаторів. Це дозволяє оперативно реконфігурувати шляхи і оптимізувати завантаження каналів передачі інформації.

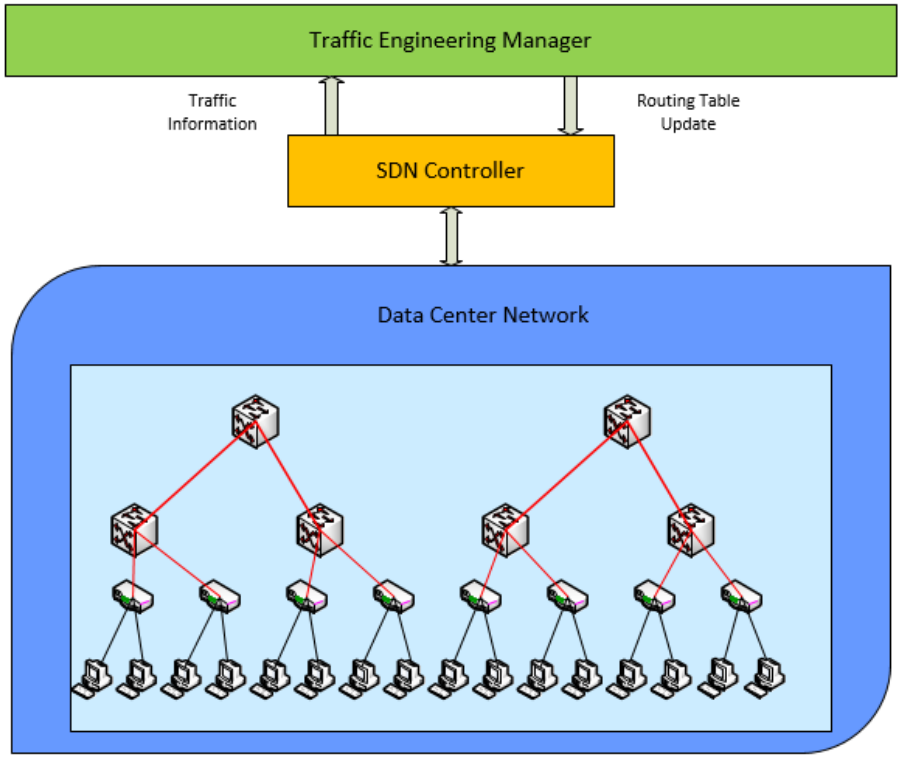


Рис. 1.8 Загальна структура системи Traffic Engineering (TE)

На рис. 1.8 Наведено структуру технічного трафіку (TE), Яка включає в себе центр обробки даних (DCN), а також контролер SDN та менеджера TE.

DCN займається складанням і передачею даних в SDN про завантаженість мережі, а також про стан каналу. Всю необхідну інформацію для конструювання трафіка менеджеру TE, забезпечує SDN. Після того, як контролер змінює інформацію про маршрутизації в SDN, відбувається перемикання за рахунок оновлення власних таблиць маршрутизації, для того, щоб знову вибрати найоптимальніший маршрут вже з урахуванням мінімізації енергоспоживання і перевантаження каналу.

Таким чином, ми можемо досягти економії енергоспоживання приблизно на 40%, а також збільшення максимального завантаження каналів на 60%, якщо порівнювати зі статичними способами TE.

На відміну від традиційних методів маршрутизації, в багатоколійні маршрутизації кілька шляхів між вузлами мережі формуються одночасно. Це дозволяє динамічно балансувати канали через зміну маршруту.

У зв'язку з цим виникає необхідність в дослідження і розробки методу балансування трафіку в мережах SDN на основі багатоколійні маршрутизації з метою підвищення ефективності функціонування комп'ютерних мереж.

Основним недоліком відомих методів управління трафіком є те, що більшість великих потоків, звані слоновими потоками, перенаправляються на один і той же шлях, що призводить до дисбалансу навантаження і втрати смуги пропускання. У зв'язку з цим при проектуванні трафіку і балансування необхідно враховувати розмір і характер навантаження каналів.

Відомі способи організації множинних шляхів мають високу тимчасову складність. Так само відомий покращений метод організації множинних шляхів, який є менш витратним з точки зору часової складності в порівнянні з іншими способами організації множинних шляхів. Було запропоновано покращений метод багатоколійні маршрутизації, який здатний враховувати характеристики організації SDN, завдяки наявності центрального контролера в мережі, таким чином, відбувається економія часу, необхідна для генерації множинного доступу маршруту до мережевих ресурсів. Найбільшим недоліком описаного раніше способу є те, що створення маршрутів кожним вузлом мережі не враховують шляхи, вже створені іншими вузлами. Тому відбувається реорганізація окремих ділянок вже створених шляхів. Тому розробка методів багатоколійні маршрутизацій і оркестровки трафіку в DCN важлива з урахуванням особливостей і переваг SDN.

Організація інформації про маршрут на основі аналізу відомих методів багатоколійні маршрутизації трафіку і з урахуванням особливостей організації роботи SDN. Цей алгоритм заснований на перевагах методів централізованої і децентралізованої маршрутизації. У контролері SDN на основі модифікованого алгоритму маршрутизації стану каналу створюється повний набір шляхів між різними вузлами з урахуванням топології DCN.

Але такий алгоритм є не доцільний у використанні з точки зору досить високої часовою складності даного алгоритму.

.

Таблиця 1.1

Таблиця порівняння алгоритмів .

	Швидкість	Точність	Простота алгоритму
Алгоритм пошуку у глибину	Середній	В окремих випадках, некоректні результаті завантаженості каналів	Легка
Модифікований алгоритм пошуку найкоротшого шляху	Швидкий	Точні результати завантаженості каналів	Середня

В табл. 1.1 продемонстрована порівняльна таблиця з основними характеристиками за для порівняння, а саме: швидкість, точність та простота алгоритму.

ВИСНОВКИ ДО ПЕРШОГО РОЗДІЛУ

На основі аналізу розглянутих робіт можна зробити висновок, для підвищення ефективності процесу (КТ) необхідно розробити більш модифікований алгоритм (КТ) з урахуванням особливостей технології SDN.

В даному розділі були розглянуті алгоритми маршрутизації пошуку множини, де формування шляхів походить від джерела інформації, при цьому присутні елементи випадкового перебору варіантів.

На відміну від цього алгоритму пропонується модифікований метод балансування трафіку, який враховує особливість транспортних мереж в мережах SDN на основі багатоколійні маршрутизації з метою підвищення ефективності функціонування комп'ютерних мереж.

Таким чином актуальною є проблема розробки системи оптимізації трафіку в транспортних мобільних мережах. Яка дозволить використовувати більш збалансоване навантаження даних на канали передачі інформації.

РОЗДІЛ 2
МОДИФІКОВАНИЙ АЛГОРИТМ ОПТИМІЗАЦІЇ ТРАФІКУ В ТРАНСПОРТНИХ МОБІЛЬНИХ МЕРЕЖАХ

Запропоновано метод балансування трафіку, який завдяки централізованому методу генерації інформації про маршрутизації в контролері SDN і використанні методу багатоколійні маршрутизації дозволяє підвищити ефективність процедури динамічної реконфігурації трафіку.

Інформація про маршрутизацію генерується з використанням модифікованого алгоритму хвильової маршрутизації. Цей алгоритм одночасно з формуванням заданого маршруту формує шляху від всіх проміжних вузлів до кінцевого вузла. Це скорочує час формування нової доріжки і виключає повторне формування інформації про маршрут для раніше сформованих ділянок доріжки.

Представлені результати моделювання процесу балансування трафіку при зміні навантаження на канали зв'язку. З результатів моделювання випливає, що пропонований спосіб балансування трафіку шляхом вибору шляху з мінімальним значенням критерію вибору шляху забезпечує рівномірне навантаження на мережу. Вибір шляху з мінімальною кількістю каналів викликає перерахунок метрики мінімальної кількості шляхів.

На відміну від традиційних методів маршрутизації, в багатоколійні маршрутизації кілька шляхів між мережевими вузлами формуються одночасно. Це дозволяє динамічно балансувати канали за рахунок перенаправлення.

У зв'язку з цим основною метою даного проекту є дослідження і розробка методу балансування трафіку в мережах SDN на основі багатоколійні маршрутизації з метою підвищення ефективності функціонування комп'ютерних мереж.

Для досягнення цієї мети в даній роботі представлений огляд літератури з відомим методам балансування трафіку. Наведено алгоритм формування безлічі мінімально навантажених непересічних шляхів. Запропоновано і обґрунтовано критерій вибору шляху, що забезпечує максимально можливу рівномірне навантаження на мережу. Представлений алгоритм балансування трафіку.

2.1 Огляд способів балансування трафіку в комп'ютерних мережах

У даній роботі вивчаються різні методи і підходи до вирішення проблем, балансуванню навантаження і управління перевантаженням. Основна увага приділяється таким питанням, як оптимізація трафіку даних в мережі з використанням SDN, QoS, балансування навантаження і контролю перевантаження.

В роботі запропонований спосіб балансування навантаження контролера і балансування навантаження каналу в SDN. Показано, що проблема розподілу навантаження для ліній і контролерів, які використовують відомі методи формування шляху, є NP-повним завданням.

Так само в роботі розглядаються питання балансування навантаження в топології Fat-Tree. в емуляторі Mininet. Результати моделювання в емуляторі Mininet показали, що середнє збільшення навантаження склало 50%, середній час затримки збільшилася на 41%, а також значно покращився з точки зору відгуку пинга, використання смуги пропускання і ширини смуги системи.

На додаток, в даному проекті запропоновано метод балансування трафіку, який завдяки централізованому методу генерації інформації про маршрутизації в контролері SDN і використанні багатоканальної маршрутизації спрощує реконфігурацію трафіку і забезпечує найбільш уніфіковане формування мережевого навантаження.

Формування інформації про маршрутизації з використанням алгоритму хвильової маршрутизації дозволяє одночасно генерувати шляху від всіх проміжних вузлів до кінцевого вузла. Це істотно зменшує тимчасову складність формування декількох шляхів.

Так само в проекті запропоновано алгоритм багатоколійної маршрутизації, який дозволяє підвищити продуктивність мережі на 10-15% за рахунок зменшення обсягу сервісних пакетів. Це дозволяє знизити енергоспоживання на 40% і збільшити максимальне використання каналів зв'язку на 60% в порівнянні з розподіленими методами балансування. Крім того, затримка пінга в мережі зменшується на 5-10%, а кількість контрольних пакетів в мережі зменшується на 60-70%.

У свою чергу, використання багатоколійної маршрутизації дозволяє швидко перенаправити маршрут після збоїв каналів і мережевих комутаторів.

Також запропоновано метод централізованого формування інформації про маршрутизації в розподілених центрах обробки даних на основі технології SDN, який виключає повторне формування маршрутів між проміжними ділянками маршруту.

Найбільшою проблемою вже відомих методів управління трафіком є те, що великі потоки виявляються спрямовані на єдиний шлях, в слідстві чого завантаженість каналів стає незбалансованою, і відбувається втрата смуги пропускання. Відповідно до вищесказаного необхідно брати до уваги характер завантаженості і розмірність самих каналів.

Спосіб балансування трафіку в SDN на основі багатоколійні маршрутизації з урахуванням характеру завантаження каналів:

На основі аналізу відомих методів балансування трафіку і врахування особливостей організації та функціонування програмно-конфігуруються мереж в даному проекті запропонована наступна послідовність балансування трафіку:

1. Формування множини непересічних шляхів між кінцевими вершинами шляху.
2. Серед згенерованих шляхів вибір шляху з мінімальним критерієм завантаження каналів шляху:
3. Регулювання шляху завантаження каналу.
4. Динамічна реконфігурація шляхів, прилеглих до цього шляху.

2.2 Формування множини непересічних шляхів між кінцевими вузлами шляху

Завантаження мережі залежить від довжини шляхів, за якими передається інформація. Наявність загальних каналів між різними шляхами призводить до перевантажень каналів і зниження ефективності комп'ютерних мереж. У зв'язку з цим при балансуванні трафіку необхідно формувати безліч мінімально завантажених непересічних шляхів. Для мереж SDN наступний модифікований хвильової алгоритм задовольняє цим вимогам.

Алгоритм формування безлічі мінімально навантажених непересічних шляхів

1. $D_h = 0$, $D_p = 0$, $d_p = 0$, $d_h = 0$,
2. if $L_{h,h} < L_{p,p}$ and …< $L_{n,n.}$ then go to W_h ; //Search for the transition to the
3. else go to W_p ; // second peak with less workload.
4. for i=1 step1 W_{n+1} Calculate d_n, D_n;
5. if $d_h < d_p ... d_n$ and < d_n then go to d_h ; // Choosing the least loaded
6. else go to d_p ; // channel to go to the next vertex.
7. if $d_h = d_p$;
8. if $D_h < D_p$ then go to d_h ; // Calculation of the minimum number of
9. else go to d_p ; // transitions to the final vertex at $d_h = d_p$.
10. if $W_p + 1 = \varnothing$ then go to 4; // Completion of the program in the absence

11.end ; // of the need for transition, due to reaching the final peak.

Де:

dp, dh - завантаження каналу.

Dp, Dh - кількість переходів.

W - вершини графа.

2 – Пошук переходу на другий пік з меншим навантаженням.

5 – Канал, для переходу до наступної вершини.

8 – Розрахунок мінімальної кількості.

9 – Переходи до кінцевої вершини.

10 – Завершення програми за відсутності потреби в переході через досягнення остаточного піку.

В результаті роботи цього алгоритму в контролері SDN формується набір мінімально завантажених непересічних шляхів для кожної пари вузлів. Найменший шлях вводиться в загальну таблицю шляхів між усіма вершинами.

Наприклад, для мережі, показаної на рис. 2.1, мінімальні шляху представлені в табл.2.1, виходячи з значень, для яких виконується балансування навантаження.

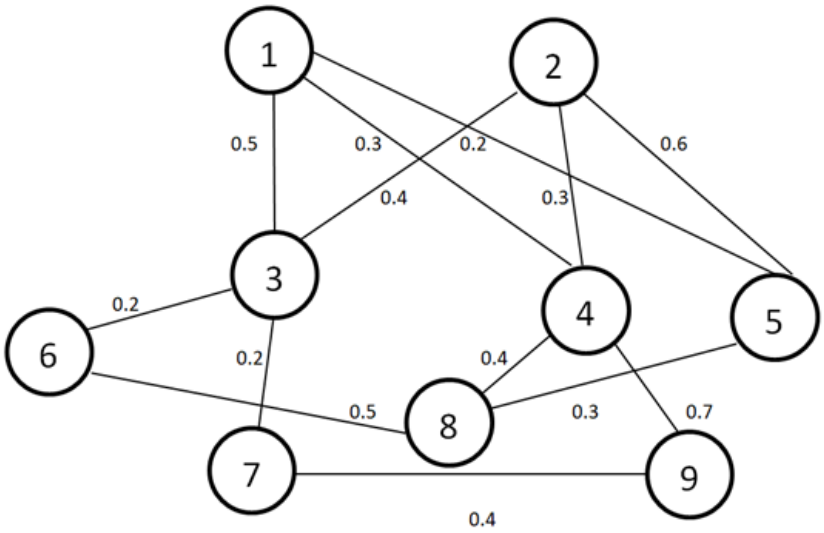


Рис. 2.1 Граф передачі даних між вершинами.

На рис. 2.1 представлений граф передачі даних між вершинами з використанням рандомної початкової ваги ребра.

Мінімальні шляхи у (Табл. 2.1) були отримані шляхом аналізу усіх можливих шляхів з рис. 2.1.

Таблиця 2.1

Таблиця мінімальних шляхів між вершинами графа

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
R_1	0	L1.4.2	L1.3	L1.4	L1.5	L1.3.6	L1.3.7	L1.5.8	L1.4.9
R_2	L1.4.2	0	L2.3	L2.4	L2.5	L2.3.6	L2.3.7	L2.4.8	L2.3.7.9
R_3	L1.3	L2.3	0	L3.2.4	L3.1.5	L3.6	L3.7	L3.6.8	L3.7.9
R_4	L1.4	L2.4	L3.2.4.	0	L4.1.5	L4.8.6	L4.2.3.7	L4.8	L4.9
R_5	L1.5	L2.5	L3.1.5	L4.1.5	0	L5.8.6	L5.1.3.7	L5.8	L5.1.4.9
R_6	L1.3.6	L2.3.6	L3.6	L4.8.6	L5.8.6	0	L6.3.7	L6.8	L6.3.7.9
R_7	L1.3.7	L2.3.7	L3.7	L4.2.3.7	L5.1.3.7	L6.3.7.	0	L7.3.6.8	L7.9
R_8	L1.5.8	L2.4.8	L3.6.8	L4.8	L5.8	L6.8	L7.3.6.8	0	L8.4.9
R_9	L1.4.9	L2.3.7.9	L3.7.9	L4.9	L5.1.4.9	L6.3.7.9	L7.9.	L8.4.9.	0

Динамічне балансування навантаження в залежності від характеру каналів навантаження.

2.3 Процедура балансування навантаження

У цьому проекті процедура балансування полягає у виборі серед безлічі доступних шляхів мінімально завантаженого шляху з відносно рівномірним завантаженням каналів даного шляху. Щоб забезпечити рівномірне навантаження, необхідно вибирати шлях з мінімальною середнім навантаженням і мінімальним середньоквадратичним відхиленням

навантаження на шлях від середнього значення. Для досягнення даної мети, пропонується використовувати критерій завантаження каналів шляхів:

$$D_i = \frac{n}{N}\left(d_i^0 + \frac{1}{n}\sum_{j=1}^{n}\left(d_j - d_i^0\right)^2\right), \quad (2.1)$$

де:

n – кількість зв'язків (каналів) шляху P_i між початковою і кінцевою вершинами V_e;

N – кількість зв'язків максимального шляху P_m між вершинами V_s и V_s;

d_i^0 – середнє значення навантаження каналів шляху P_i;

d_j – Завантаження каналу $L_j \in P_i$.

$$d_i^0 = \frac{1}{n}\sum_{j=1}^{n} d_j, \quad (2.2)$$

де:

Відношення n / N дозволяє враховувати відносну довжину шляху P_i.

Значення d_i^0 – визначає менш завантажений шлях.

Середньоквадратичне відхилення навантаження каналів шляху $\left(\left(d_j - d_i^0\right)^2\right)$ характеризує ступінь рівномірності навантаження каналів шляху.

Таким чином, коефіцієнт Di дозволяє вибрати найбільш оптимальний спосіб з точки зору балансування навантаження в мережі.

Процедура балансування навантаження:

Балансування навантаження на канали складається з вибору шляху передачі даних, який оптимізує завантаження мережевих каналів. У процесі балансування відбувається обмін службовою інформацією між контролером SDN і мережевими комутаторами рис.2.2.

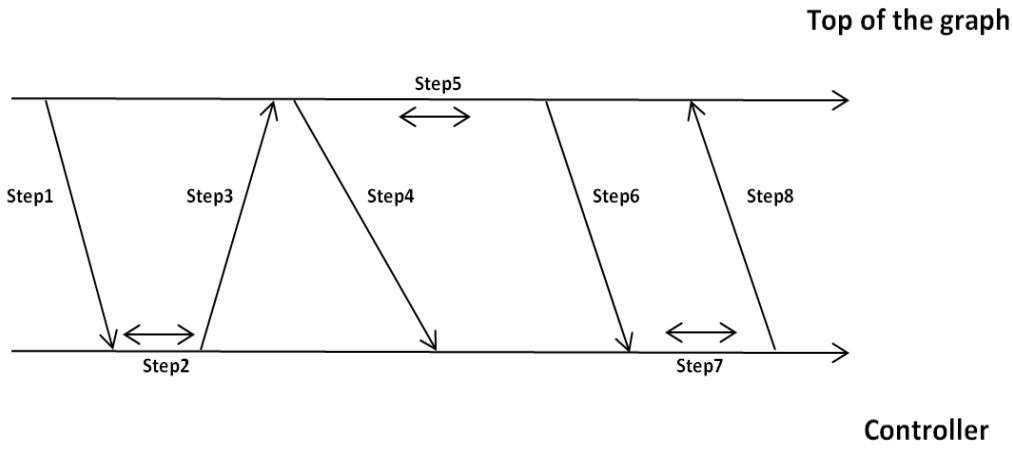


Рис. 2.2 Процес обміну службовою інформацією між контролером і комутатором.

Крок 1 – Запит маршруту.

Крок 2 – Формування маршруту.

Крок 3 – Перенесення маршруту.

Крок 4 – Номер маршруту.

Крок 5 – Передача інформації.

Крок 6 – Кінець передачі.

Крок 7 – Внесення виправлень в маршрут.

Крок 8 – Оновлення маршруту.

Алгоритм балансування навантаження:

1. Іноді на етапі 1 комутатор відправляє запит на маршрутизацію зі своєю адресою та адресою одержувача.
2. На кроці 2 контролер, згідно з таблицею мінімальних шляхів (Таб. 2.1), визначає наявність допустимих шляхів між відправником і отримувачем інформації. При відсутності допустимих шляхів SDN створює багато допустимих шляхів, використовуючи змінений протокол маршрутизації.
3. На кроці 3 маршрут передається контролером SDN верхньому відправнику відправника інформації. Серед доступних шляхів вибирається шлях з мінімальним завантаженням каналів, і його номер повідомляється контролера SDN.
4. На основі номера обраного маршруту контролер SDN коригує маршрути, суміжні з обраним маршрутом.
5. Потім інформація переноситься в обрану вершину.
6. Після завершення передачі інформації використовуються маршрути оновлюються.

2.4 Приклад балансування навантаження каналу.

Як приклад розглянемо моделювання процесу балансування навантаження при передачі інформації між вершинами W7 і W5 графіка, показаного на рис. 2.3 Використовуючи наведену вище формулу (2.1) для середнього значення навантаження каналів каналу. Ми розраховуємо завантаженість каналів для кожного з можливих шляхів, а саме:

1. W_7, W_3, W_1, W_5.
2. W_7, W_3, W_2, W_5.
3. W_7, W_9, W_4, W_8, W_5.
4. W_7, W_3, W_6, W_8, W_5.
5. W_7, W_9, W_4, W_2, W_5.

Знаходження середньої завантаженості шляху каналу:

1.1 W_7, W_3, W_1, W_5.

1.1.1 0.2 + 0.5 + 0.2 = 0.9

1.1.2 0.9 \ 3 = 0.3

Далі, використовуючи середнє навантаження (0.3), знаходимо відхилення значення навантаження каналу L від середнього навантаження каналу.

$P\Delta = P_i - P_{cp}$

1.1.3 0.3 – 0.2 = 0.1

1.1.4 0.3 – 0.5 = 0.2

1.1.5 0.3 – 0.2 = 0.1

На цьому етапі розраховується середньоквадратичне відхилення навантаження каналу.

1.1.6 $0.3+((0.1)^2 + (0.2)^2 + (0.1)^2)= 0.320$

Потім виконується множення знайденого вище середньоквадратичного відхилення шляхом ділення числа зв'язків на число зв'язків максимального шляху (n / N) між вершинами Vs і Vs;

1.1.7 (0.32*3) / 6= 0.160

Розрахуємо коефіцієнт завантаження для решти можливих шляхів:

2.2 W_7, W_3, W_2, W_5.

2.2.1 0.2 + 0.4 + 0.6 = 1.2/3 = 0.4

2.2.2 $0.4 +((0.2)^2+(0.2)^2) / 3 = 0.426$

2.2.3 (0.426 * 3) / 6 = 0.213

3.3 W_7, W_9, W_4, W_8, W_5.

3.3.1 0.4 + 0.7 + 0.4 + 0.3 = 1.8/4 = 0.45

3.3.2 $0.45 + ((0.05)^2+(0.25)^2 + (0.05)^2 + (0.15)^2)/ 4 = 0.472$

3.3.3 (0.4725*4) / 6 = 0.315

4.4 W_7, W_3, W_6, W_8, W_5.

4.4.1 0.2 + 0.2 + 0.5 +0.3 = 1.2 / 4 = 0.3

4.4.2 $0.3 + ((0.1)^2+(0.1)^2+(0.2)^2) / 4 = 0.315$

4.4.3 (0.315*4) / 6 = 0.210

5.5 W_7, W_9, W_4, W_2, W_5.

5.5.1 0.4 + 0.7+ 0.3 + 0.5 = 1.9/4 = 0.47

5.5.2 $0.47 + ((0.07)^2+(0.23)^2+(0.17)^2+(0.03)^2) / 4 = 0.419$

5.5.3 (0.4919*4) / 6= 0.327

1) W_7, W_3, W_1, W_5 = 0.160
2) W_7, W_3, W_2, W_5 = 0.213
3) W_7, W_9, W_4, W_8, W_5 = 0.315
4) W_7, W_3, W_6, W_8, W_5 = 0.210
5) W_7, W_9, W_4, W_2, W_5 = 0.327

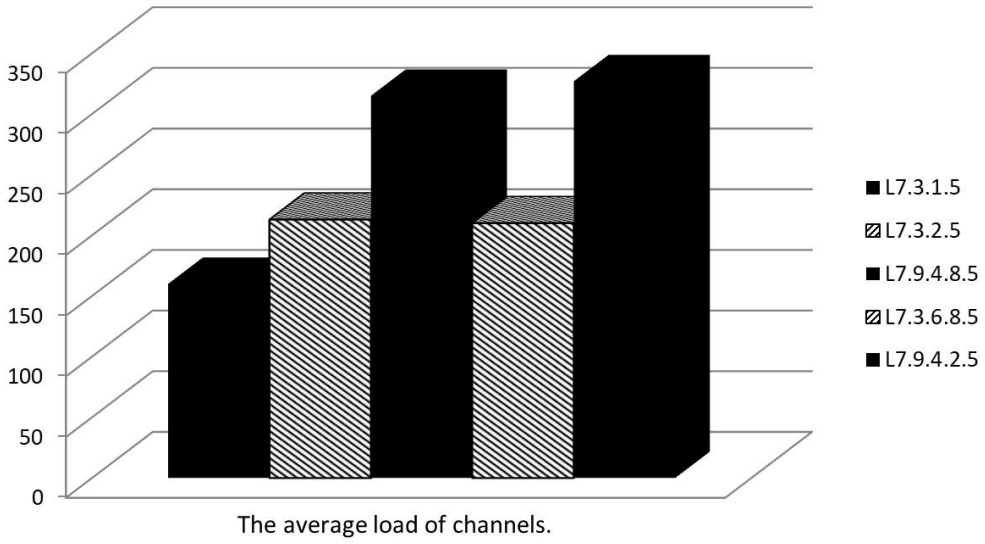


Рис. 2.3 Середнє завантаження каналів

Рис. 2.3 – це-діаграма результатів ефективності можливих шляхів W7 - W5.

Таким чином, для шляхів від вершини 7 до вершини 5 з найменшим коефіцієнтом навантаження, шлях 1. з D1 = 0.160.

Для кожного шляху, представленого в (Табл. 2.1), визначаються коефіцієнти завантаження каналу, значення яких представлено в (Табл. 2.2).

Таблиця 2.2

Шляхи завантаженості каналів

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
R_1	0	0.141	0.083	0.050	0.033	0.149	0.149	0.101	0.214
R_2	0.141	0	0.050	0.050	0.075	0.124	0.124`	0.141	0.179
R_3	0.083	0.050	0	0.141	0.149	0.033	0.033	0.149	0.124
R_4	0.050	0.050	0.141	0	0.101	0.181	0.137	0.066	0.077
R_5	0.033	0.075	0.149	0.101	0	0.117	0.137	0.050	0.223
R_6	0.149	0.124	0.033	0.181	0.117	0	0.044	0.071	0.129
R_7	0.149	0.124	0.033	0.137	0.137	0.044	0	0.240	0.057
R_8	0.101	0.141	0.149	0.066	0.050	0.071	0.240	0	0.229
R_9	0.179	0.215	0.124	0.077	0.223	0.129	0.057	0.229	0

Приклад порівняння завантаженості каналів після додаткового навантаження.

Коефіцієнти навантаження в (Табл. 2.2) були отримані шляхом розрахунку найкоротшого шляху за формулою (2.1)

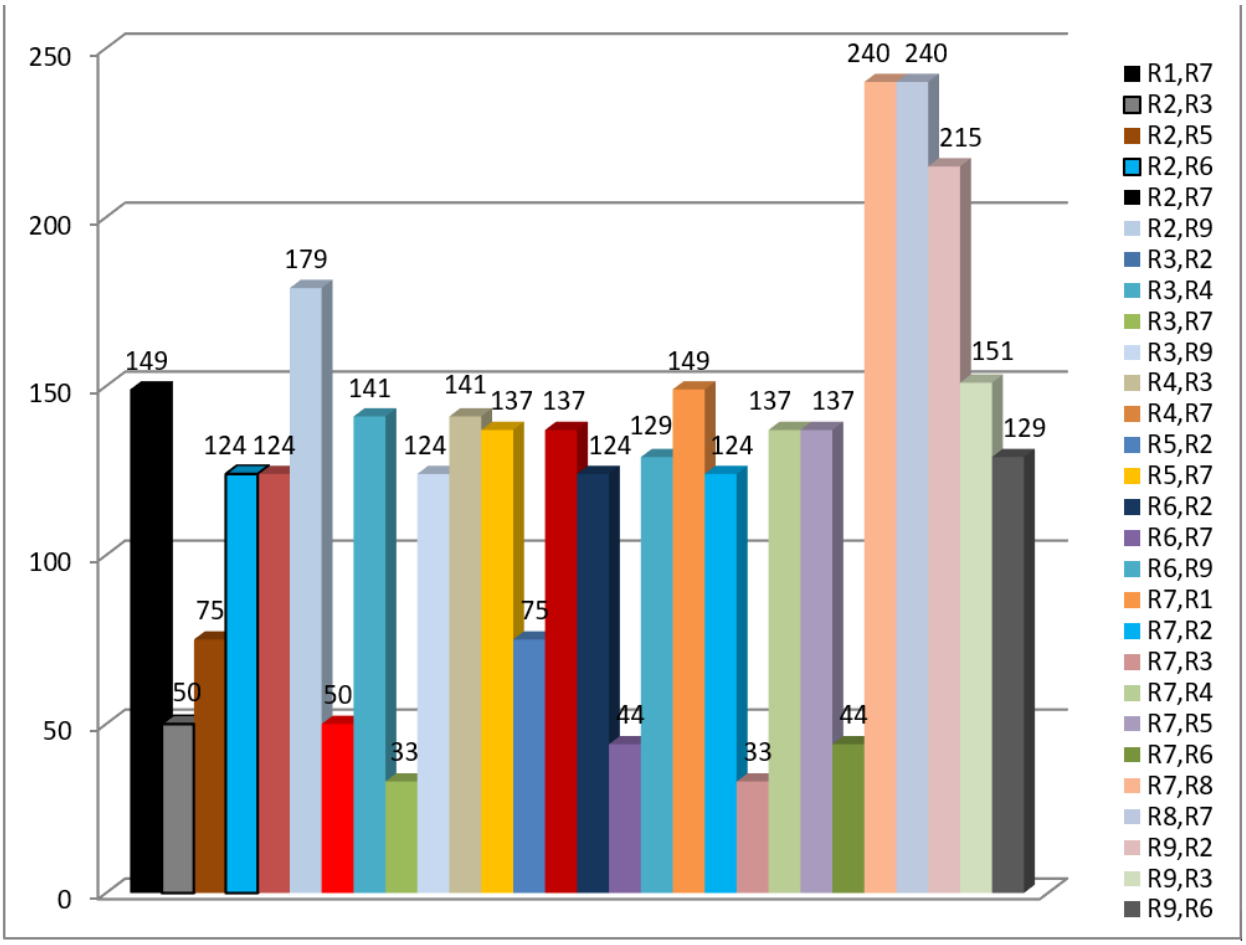


Рис. 2.4 Діаграма завантаженості каналів при звичайному навантаженні

На рис. 2.4 зображена діаграма завантаженості каналів при рівномірному розподіленню даними на кожен канал у запропонованому графі див. рис. 2.1.

2.5 Налаштування шляху завантаження каналу

Ми моделюємо ситуацію, коли потік даних між вузлами завантажує канал даних на (0.1) і буде йти по шляху W7, W3, W2, W5. Отже, навантаження на кожен перехід буде збільшуватися на (0.1.) Розрахуємо коефіцієнт ефективності завантаження каналу з урахуванням зміни ваги кожного переходу.

1. 0.3 + 0.5 + 0.7 = 1.5 / 3 = 0.5
2. $0.5 + ((0.2)^2 + (0.2)^2) / 3 = 0.5266$
3. (0.5266*3) / 7= 0.225

Кінцевий коефіцієнт ефективності завантаження при завантаженні потокової інформації (0.1).

В цьому випадку навантаження каналів, що мають сусідні шляхи з даними, також буде збільшуватися. Наприклад, при формуванні шляху (L7.3.2.5). Шлях L1.3 не змінений і його вага становить 0.5. Але шлях L3.7 вже використовувався для передачі даних. Отже, він важить на 0,1 більше, а саме L3,7 = 0.3. З урахуванням цих змін вагу шляху становить L1.3.7. зміниться. Розрахуємо коефіцієнт завантаження каналу з урахуванням зміни ваги переходу (ів).

L1.3.7

1. 0.5 + 0.3 = 0.8 / 2 = 0.4
2. $0.4 + ((0.1)^2+(0.1)^2) / 2 = 0.41$
3. (0.41*2) / 5= 0.164

В результаті новий коефіцієнт завантаження нашого каналу становить D1 = 0.164.В той час як минуле було D1 = 0.149

У цьому прикладі ми розраховуємо нову завантаженість каналу для кожного шляху, який має загальний ґрунт зі шляхом, по якому передається інформація.

В цьому випадку необхідно налаштувати 14 маршрутів. В результаті навантаження на окремі канали зросте.

У таблиці 2.3 наведені значення навантаження мережевих каналів при передачі інформації між вершинами W7 і W5. Шляхи виділені. Значення Di якого змінилося. Завантаження каналів в (Табл. 2.3) були отримані шляхом розрахунку значень за формулою (2.1) з урахуванням додаткового завантаження каналів між вершинами.

Таблиця 2.3

Канали завантаження для передачі інформації

	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
R_1	0	0.141	0.083	0.050	0.033	0.149	**0.164**	0.101	0.214
R_2	0.141	0	**0.062**	0.050	**0.087**	**0.145**	**0.162**	0.141	**0.202**
R_3	0.083	**0.062**	0	**0.205**	0.149	0.033	**0.042**	0.149	**0.141**
R_4	0.050	0.050	**0.205**	0	0.101	0.181	**0.179**	0.066	0.077
R_5	0.033	**0.087**	0.149	0.101	0	0.117	**0.156**	0.050	0.223
R_6	0.149	**0.145**	0.033	0.181	0.117	0	**0.072**	0.071	**0.153**
R_7	**0.164**	**0.162**	**0.042**	**0.179**	**0.156**	**0.072**	0	**0.274**	0.057
R_8	0.101	0.141	0.149	0.066	0.050	0.071	**0.274**	0	0.229
R_9	0.214	**0.202**	**0.141**	0.077	0.223	**0.153**	0.057	0.229	0

В табл. 2.3 продемонстровані результати обчислень завантаженості каналу при додатковому навантаженні даними, за формулою (2.1). Змодельована ситуація, при якій потік даних між вузлами завантажує канал даних на 0.1 і буде йти по шляху W7, W3, W2, W5. Алгоритм обчислення представлений вище.

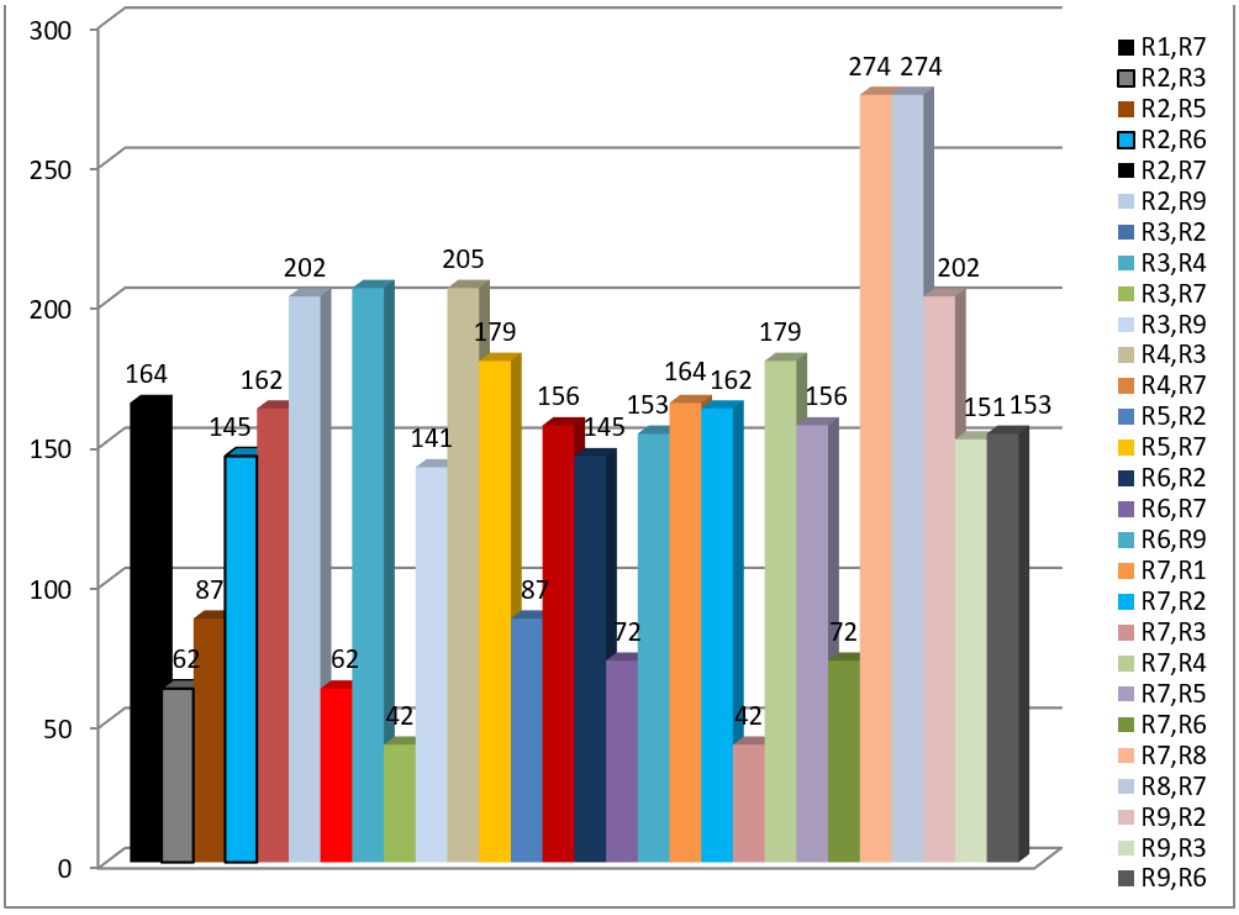


Рис. 2.5 Діаграма завантаженості каналів після додаткового навантаження даними

Рис. 2.5 демонструє оновлені результати завантаженості каналів які змінили свою вагу за рахунок додатковому навантаженні даними в (0.1).

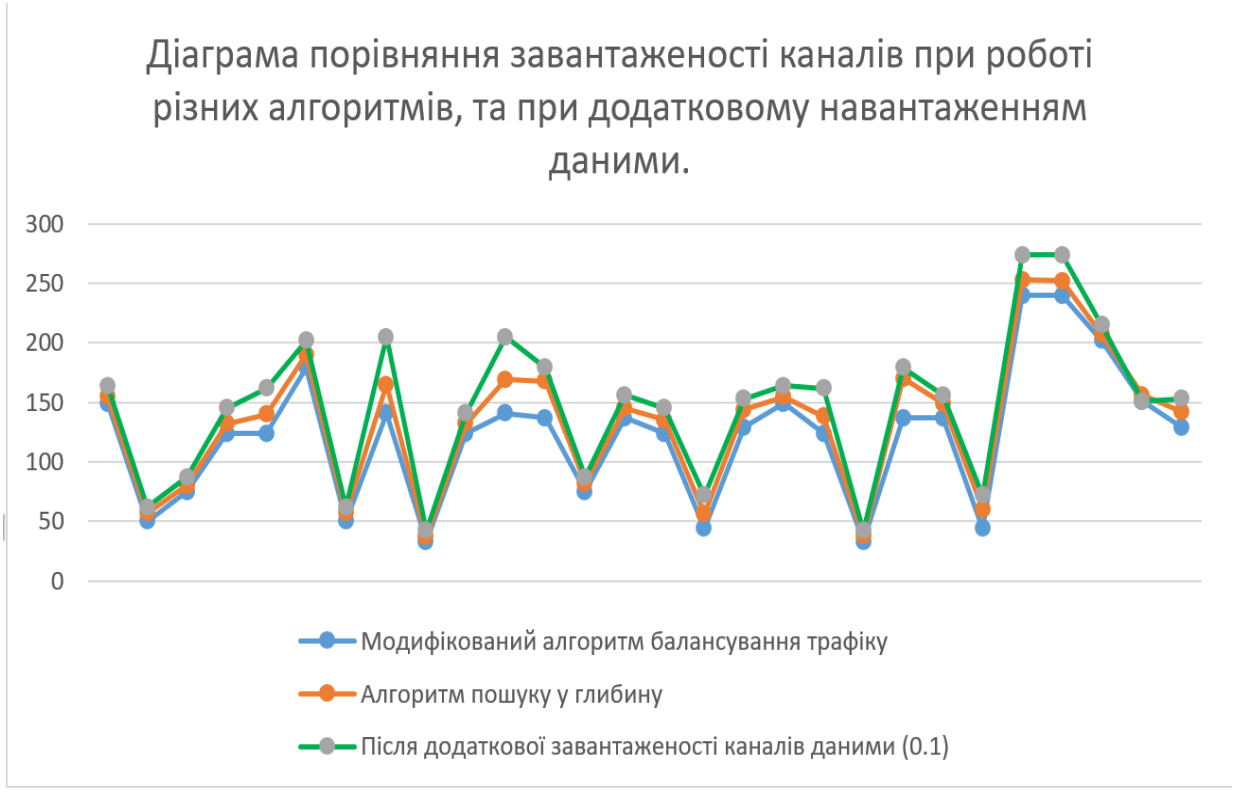


Рис. 2.6 Точкова порівняльна діаграма

Рис. 2.6 демонструє порівняння завантаженості каналів при роботі алгоритму пошуку у глибину, модифікованого алгоритму, та роботу модифікованого алгоритму при додатковому навантаженні даними в (0.1).

.

ВИСНОВКИ ДО ДРУГОГО РОЗДІЛУ

Запропоновано і обґрунтовано критерій вибору шляху з набору доступних шляхів, який дозволяє більш рівномірно завантажувати канали передачі інформації із заданими параметрами QoS. З практичної точки зору використання запропонованого в роботі методу скорочує час балансування трафіку за рахунок виключення процедури перерахунку маршрутів при зміні навантаження на окремі параметри.

Під час роботи мережі можуть динамічно змінювати навантаження на канали. Це може вплинути на узгодженість результатів моделювання. Цей метод балансування трафіку дозволяє надати більш рівномірне навантаження на мережу, що знижує споживання мережі.

РОЗДІЛ 3
МОДЕЛЮВАННЯ ЗАПРОПОНОВАНОГО АЛГОРИТМУ

3.1 Результати роботи програми

Запропонований метод балансування трафіку за рахунок одночасного формування безлічі найкоротших маршрутів між усіма вершинами, також є можливість робити маршрутизації в динамічному режимі без перерахунку всього шляху. Наприклад, вам все ще потрібно перейти від вершини 7 до вершини 5, але різниця в тому, що з кожним переходом контролер передає інформацію про будь-які зміни в кожну вершину шляху і динамічно змінює маршрут, якщо це необхідно.

Це дозволяє виключити процедуру повернення в точку початку руху, якщо який-небудь канал виявився перевантажений при передачі інформації. Розглянемо процес передачі інформації по маршруту L7.3; L3.2;

Припустимо, що під час передачі канал L2.5 був перевантажений. Для проведення експериментів. При відсутності альтернативного шляху проводиться повернення до найближчої останньої вершини (2). У режимі реального часу контролер змінює шлях до вершини, до якої ми повинні дістатися (W2, W4, W8, W5). Таким чином, немає необхідності повертатися до вершини руху (7), щоб планувати новий маршрут. Отже, існує економія на переходах і, отже, на завантаженості каналів всій мережі, що в нашому випадку більш ефективно в порівнянні з відомими методами балансування трафіку.

Якщо в мережі існує два типи трафіку: еластичний і нееластичний, перш за все необхідно сформувати шляхи для нееластичного трафіку, оскільки це пред'являє більш високі вимоги до якості обслуговування (QoS). В цьому випадку балансування навантаження в основному здійснюється за рахунок еластичного трафіку

3.2 Опис логіки програми

В даному проекті за для моделювання запропонованого алгоритму була використана високорівнева мова програмування – Python.

Даній вибір легко обґрунтувати за наступними пунктами:

1. Автоматичне управління пам'яттю. Що надає можливість програмісту не витрачати час щодо розподілення або звільнення пам'яті.
2. Велика кількість бібліотек, які можна використати задля досягнення необхідних рішень.
3. Наявність інтегрованих середовищ розробки, а саме: PyCharm від компанії JetBrains.
4. Використана мова є сучасною та все більш популярною у наші часи. Її використання є гарним вибором задля покращення навичок програміста.
5. Виконання операцій виконується на більш високому рівні абстракцій.

Для розробки програми було використано інтегроване середовище розробки PyCharm.

Для візуалізації було використана бібліотека matplotlib.

Рекомендовані характеристики комп’ютера при роботі з програмою:

1. оперативна пам’ять об’ємом не менше ніж 4 ГБ;
2. процесор з тактовою частотою не менше ніж 2 ГГц;
3. Версія PyCharm Edu не нижча ніж 3.0;

Інші характеристики не мають великого впливають на коректну роботу даної програми, тому ними можна знехтувати.

Задля підвищення ефективності конструювання трафіку та більш збалансованого навантаження на мережу є необхідність реалізувати модифікований алгоритм балансування трафіку з урахуванням усіх особливостей технології SDN. Реалізацію даного алгоритму можна поділити на декілька етапів.

Перш за все необхідно розглянути усі елементи які відповідають за формування графу.

Граф за своєю суттю – це набір вершин та ребер, які можуть мати вигляд або матриці суміжності, або як список суміжності. Пропоную розглянути список суміжностей реалізований в даній програмі :

Для початку визначимо метод, якій надає можливість змінювати структуру даних графа, а саме додавання вершин.

- i = 1

  while i <10:

  graph.add_node(i)

  i += 1

Цей метод приймає як вхідні данні число, далі створює нову вершину та ініціалізує її поля.

Також є аналогічна функція додавання ребер. Мета функції – це створення нових ребер та оновлення суміжних вершин.

- def add_edge(f_item=None, s_item=None, weight=None, graph=None, color = None):

graph.add_edge(f_item, s_item, weight=weight, color=color)

graph.add_edge(s_item, f_item, weight=weight, color=color)

Присутні такі 4 характеристики як:

1. З якої вершини у яку буде відбуватися додавання ребра.
2. У якій граф буде здійснено додавання
3. Колір ребра яке додається
4. Вага ребра.

Цей процес виконує додавання ребра до графа з вершини X до вершини Y.

Кінці ребра у даному графі є нічим іншим як цілими числами ,а також номерами вершин відповідно. Також слід зауважити що одна і та сама вершина може мати кілька ребер. У випадку коли буде здійснено перехід у вершину якої не існує, буде виведено повідомлення про те, що для переходу з вершини X у вершину Y спочатку треба її створити.

Після усіх необхідних дій по підготовці побудови графу необхідно описати функцію, яка зберігає всі шляхи графу від першої вершини до кінцевої. Збереження шляхів відбувається у вигляді списку. Список зберігає вершини які у свою чергу є простими числами.

```
 all_paths = [list(i) for i in map(nx.utils.pairwise, list_all_paths)]
for paths in all_paths:
  lenght = 0
  for path in paths:
    lenght += graph[path[0]][path[1]]['weight']
```

Для подальшого знаходження найменш завантаженого шляху необхідно знайти усі можливі шляхи із вершини X у вершину Y.

```
list_all_paths = nx.all_simple_paths(graph, source, target)
result_path = []
```

3.3 Алгоритм пошуку найменш завантаженого шляху

Далі можна приступити до описання самого алгоритму знаходження найменш завантаженого шляху по вже описаній формулі (2.1) у другому розділі.

Весь алгоритм можна розділити 3 етапи, а саме :

1. Перший етап полягає у знаходженні середньої завантаженості шляху каналу:

```
for path in paths:
  lenght += graph[path[0]][path[1]]['weight']
nagruzka = lenght / len(paths)
otklonenie = nagruzka
```

2. Далі необхідно знайти відхилення значень навантаження каналу L від середнього навантаження каналу:

```
for path in paths:
  x = nagruzka - graph[path[0]][path[1]]['weight']
  otklonenie += (abs(x)) ** 2
```

3. Потім виконується множення вже знайденого середньоквадратичного відхилення шляхом ділення числа зв'язків на число зв'язків максимального шляху (n / N) між вершинами Vs і Vs;

3.1. Для цього спочатку необхідно написати функцію яка має шукати число зв’язків максимального шляху:

```
def get_max_path(all_paths):
max_lenght = 0
for paths in all_paths:
if len(paths) > max_lenght:
max_lenght = len(paths)
return max_lenght
```

3.2. Частина програми яка рахує вже остаточній коефіцієнт завантаження каналу :

```
koef_zagruzki = (otklonenie * len(paths)) / get_max_path(all_paths)
txt = text1.get('1.0', END)
txt = txt + f'{paths} - {koef_zagruzki}'
text1.delete(1.0, END)
text1.insert(1.0, txt)
if koef_zagruzki < result_koef_zagruzki:
result_path = paths
result_koef_zagruzki = round(koef_zagruzki, 3)
```

Після кожного переходу з вершини X у вершину Y, кожне ребро задіяне в цьому переході збільшить свою вагу на 0.1. Таким чином один і той самий шлях не буде найменш завантаженим кожен раз.

Також слід помітити що у випадку коли при переході між вершинами графу одне з необхідних для здійснення передачі даних ребро стало не доступне з якоїсь причини, то немає необхідності здійснювати повернення у початкову вершину, тому що контролер SDN має всю необхідну інформацію про кожну вершину і у системі реального часу прокладе оновлений маршрут у необхідну вершину.

Кількість вершин та ребер для стартового графу задається програмно, отже немає необхідності у додаткових вікнах для задання кількості вершин та ребер. Також програмно задається початкова завантаженість кожного ребра.

Далі необхідно зберегти усі шляхи у проміжний список, якій буде використано у наступній дії такій як: розрахунок фактичної, середньої завантаженості каналів, результат буде виведено у вікні відображень усіх можливих шляхів.

ВИСНОВКИ ДО ТРЕТЬОГО РОЗДІЛУ

В даному розділі було проведено та розглянуто повний аналіз додатку, якій був розроблений в рамках дипломного проекту.

Було розглянуто безпосередньо логіку самої програми. Також було детально розглянуто запропонований алгоритм на програмному рівні.

Підкреслено та обґрунтовано вибір мови програмування Python. Виділено додаткові компоненти які були використані при розробці програми.

Описано та детально розглянуто основні функції та методи які були написані безпосередньо для роботи модифікованого алгоритму. Детальне описання модифікованого алгоритму балансування трафіку ще раз підкреслює усі переваги використання запропонованого алгоритму.

РОЗДІЛ 4
ГРАФІЧНЕ ПРЕДСТАВЛЕНЯ ПРОГРАМИ

4.1Опис програмного інтерфейсу

Програма направлена на роботу з модифікованим алгоритмом балансування трафіку з урахуванням якомога більшої кількості особливостей технології SDN, як вже були розглянуті у розділі 2.

Задля кращого розуміння роботи даної програми надається візуальне пояснення усіх можливих особливостей інтерфейсу, які продемонстровано на рис. 4.1.

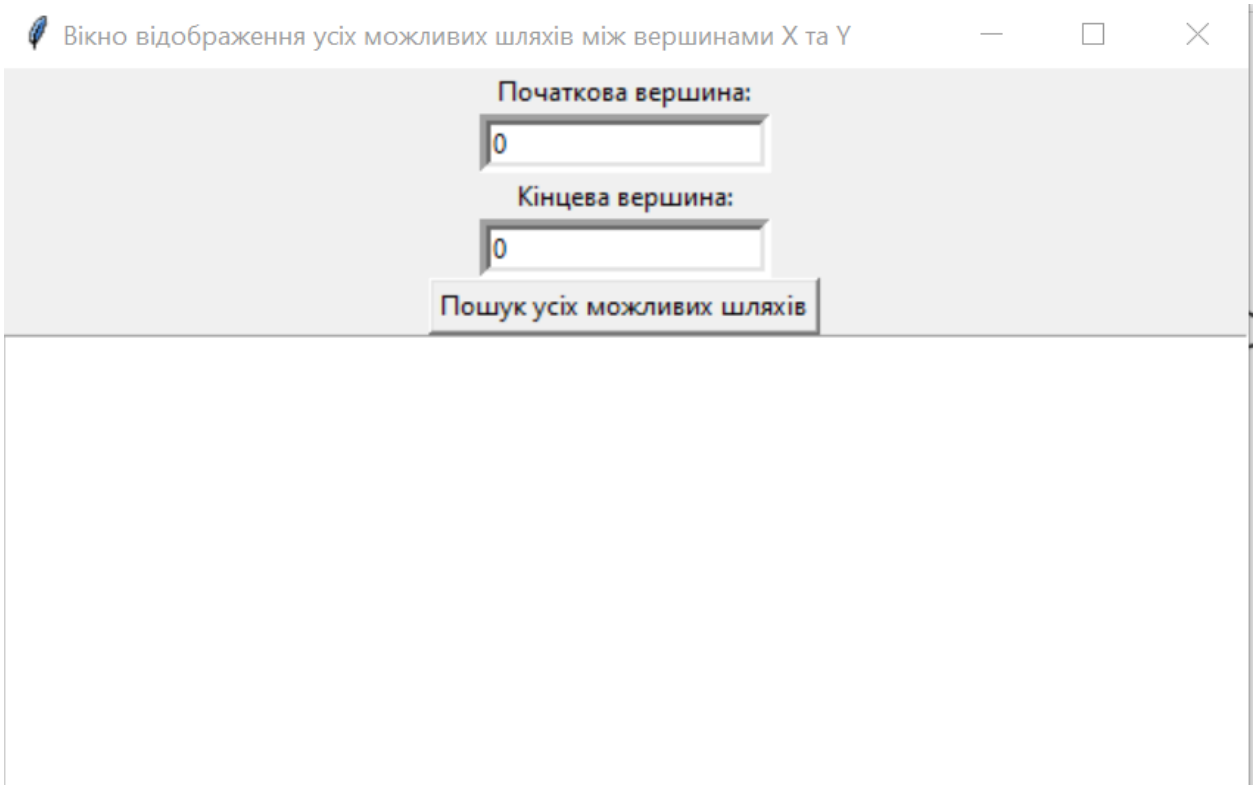


Рис.4.1 Інтерфейс програми

Дане вікно розподілене на декілька текстових ділянок (див. рис. 4.1) перше з яких – це введення номера вершини з якої необхідно здійснити перехід. Друге тестове поле відповідає за номер вершини у яку буде здійснено

перехід. Також нижче присутнє поле у якому буде виведено коефіцієнт завантаженості для усіх можливих шляхів з вершини X у вершину Y.

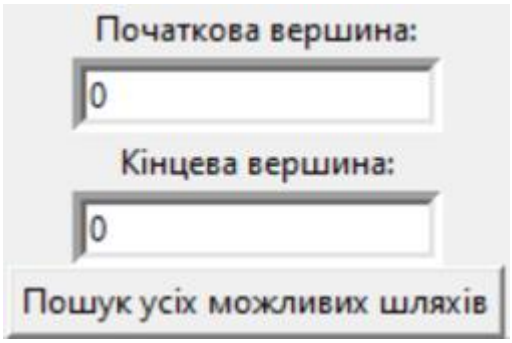


Рис. 4.2 Розташування текстових ділянок у вікні.

На рис 4.2 зображений приклад розташування текстових полів в запропонованій програмі.

4.2 Розгляд прикладу функціювання програми

Пропонується розгляд програми з використанням графу, якій продемонстрований на рис. 4.3. Насамперед необхідно програмно задати кількість вершин, кількість ребер, та назначити кожному ребру дві вершина, але одна вершина може мати декілька ребер, а також початковий коефіцієнт завантаженості кожного ребра. Користувач моє можливість визначити у відповідних полях початкову та кінцеву вершини між якими буде побудовано маршрут.

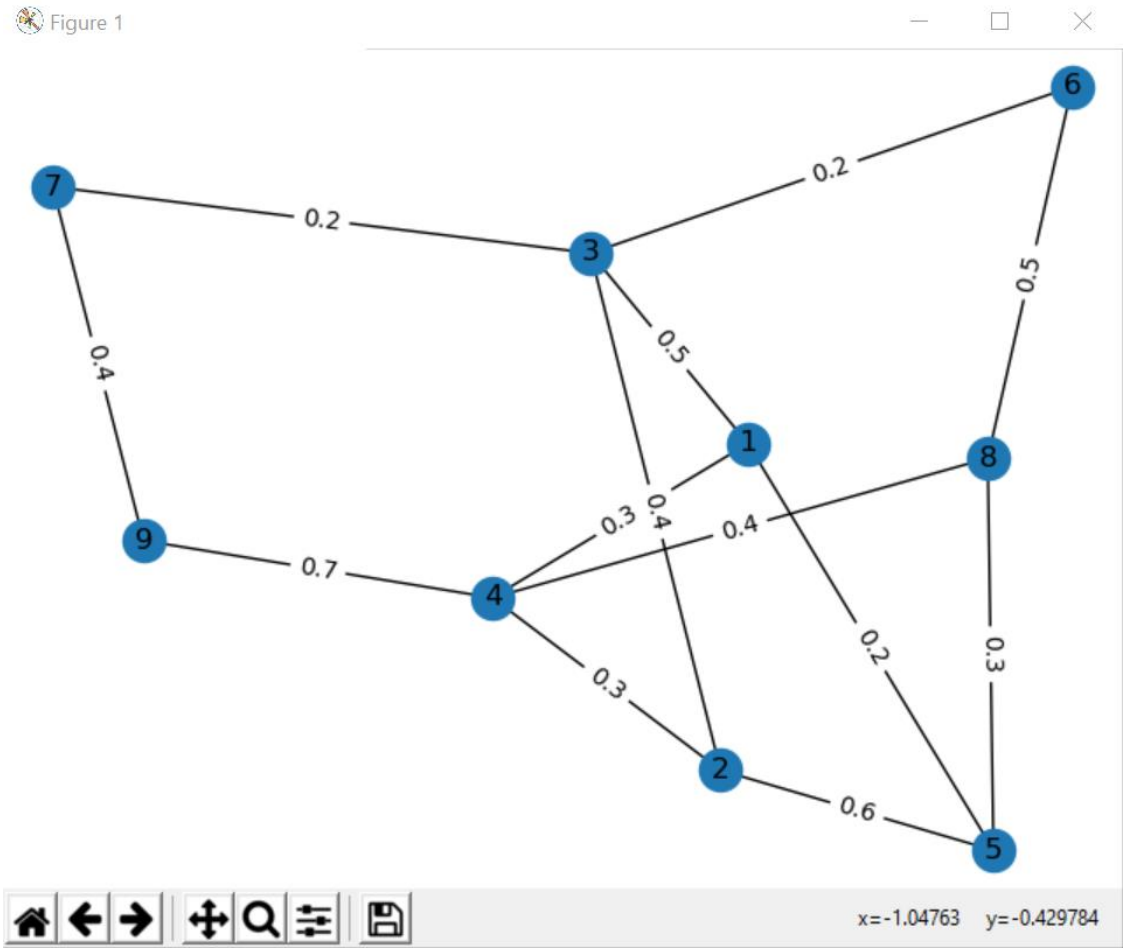


Рис. 4.3. Приклад запропонованого графу

Як приклад дані для вводу приведені на рис. 4.4

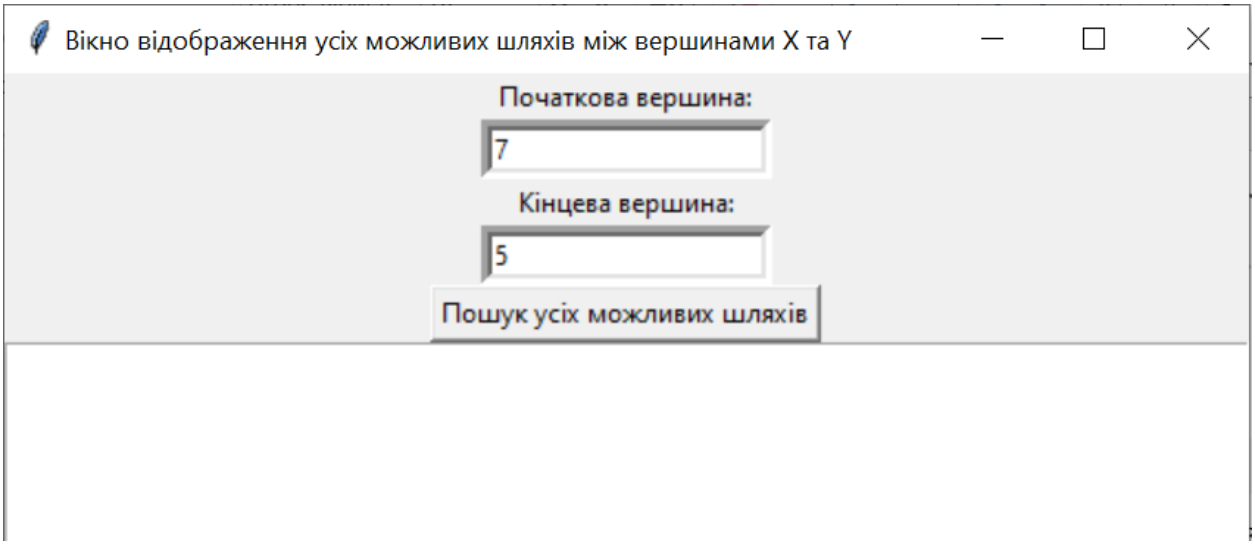


Рис. 4.4 Початкові данні вводу

Після натискання кнопки «Пошук усіх можливих шляхів» див. рис.4.4 алгоритм розпочне розрахування та у відповідному полі буде виведено коефіцієнти завантаженості каналів усіх можливих шляхів. Результат роботи алгоритму продемонстрований на рис. 4.5.

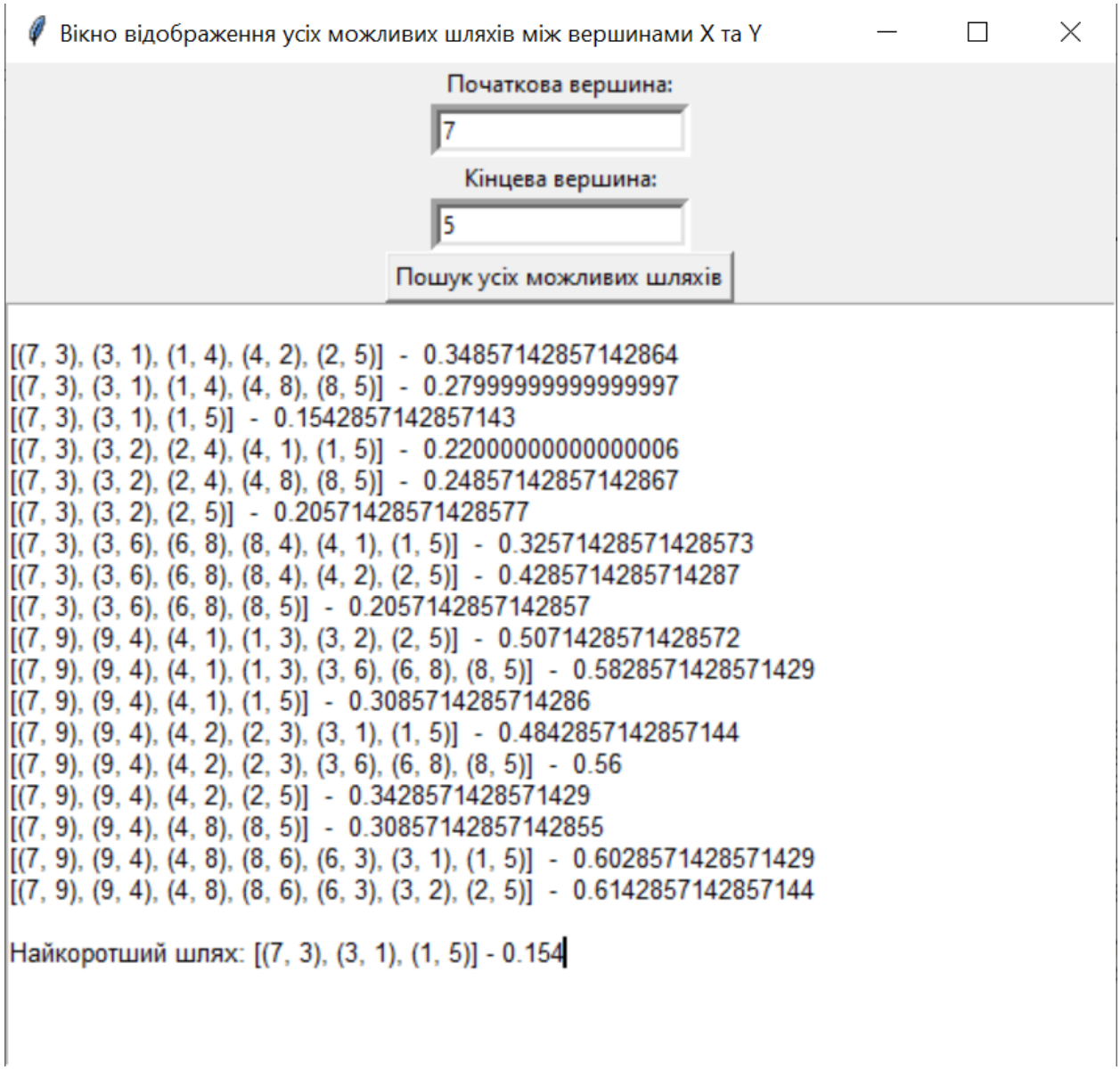


Рис. 4.5 Результат роботи програми

Як приклад роботи програми, заданий шлях з вершини 7 у вершину 5 див. рис. 4.5. У рамках моделювання, програмно задано що усі ребра які були задіяні у переході збільшать свою вагу на 0.1.

4.3 Візуалізація побудови шляху між вершинами у графі

Нова вага ребер відобразиться одразу після натискання кнопки «Пошук» на місці старої ваги.

Також весь шлях буде виділено червоним рис. 4.6, що надає можливість візуально побачити обраний, найкоротший шлях, та помітити зміну ваги ребер.

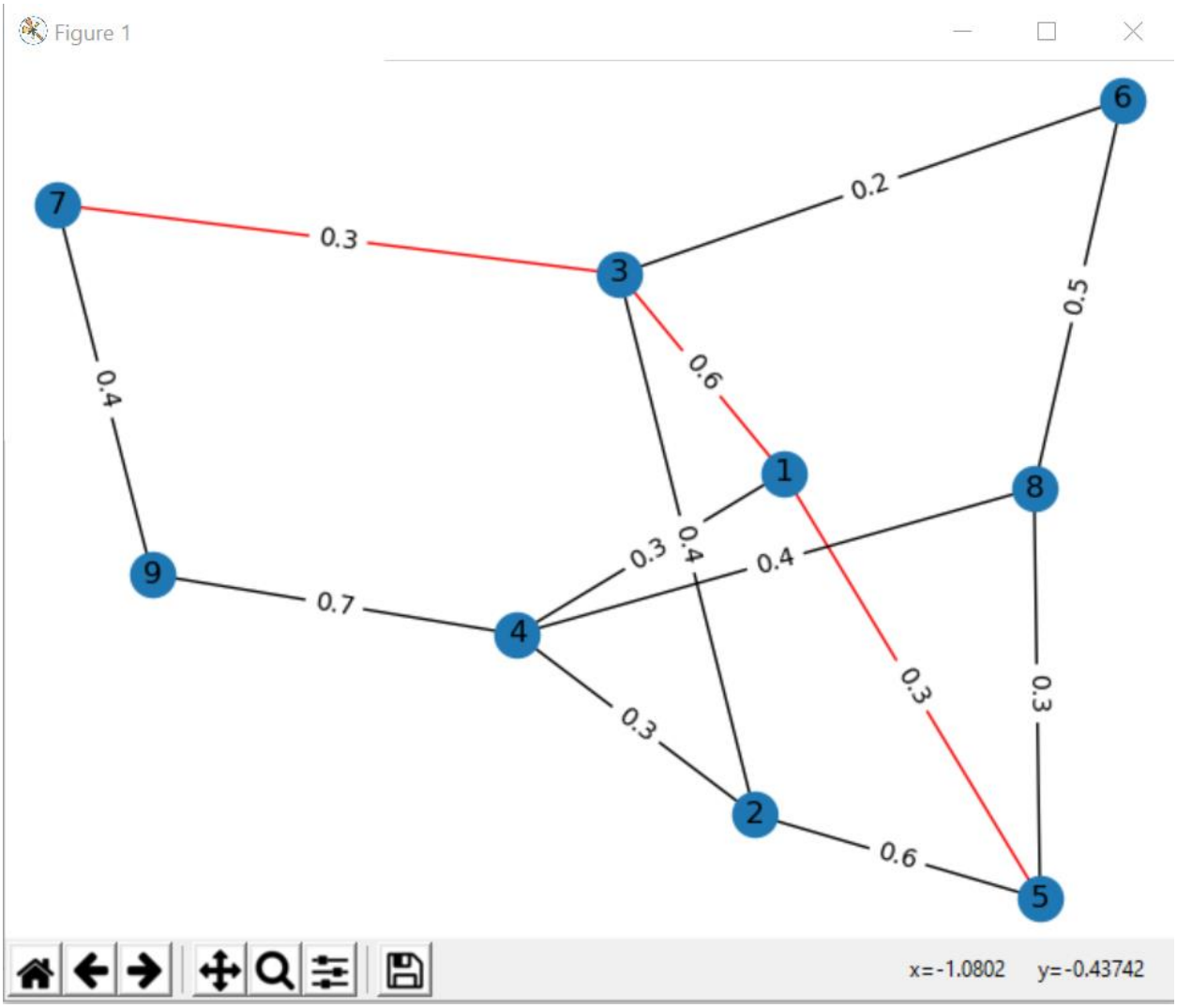


Рис4.6 Приклад графу з візуальним ефектом переходу між вершинами

Також весь шлях буде виділено червоним рис. 4.6, що надає можливість візуально побачити обраний, найкоротший шлях, та помітити зміну ваги ребер.

Слід виділити, що з кожним наступним переходом між вершинами будь який шлях якій буде мати хоча б одне ребро зі зміненою вагою, буде перераховувати коефіцієнт завантаженості відповідно оновленим даним про змінені ребра.

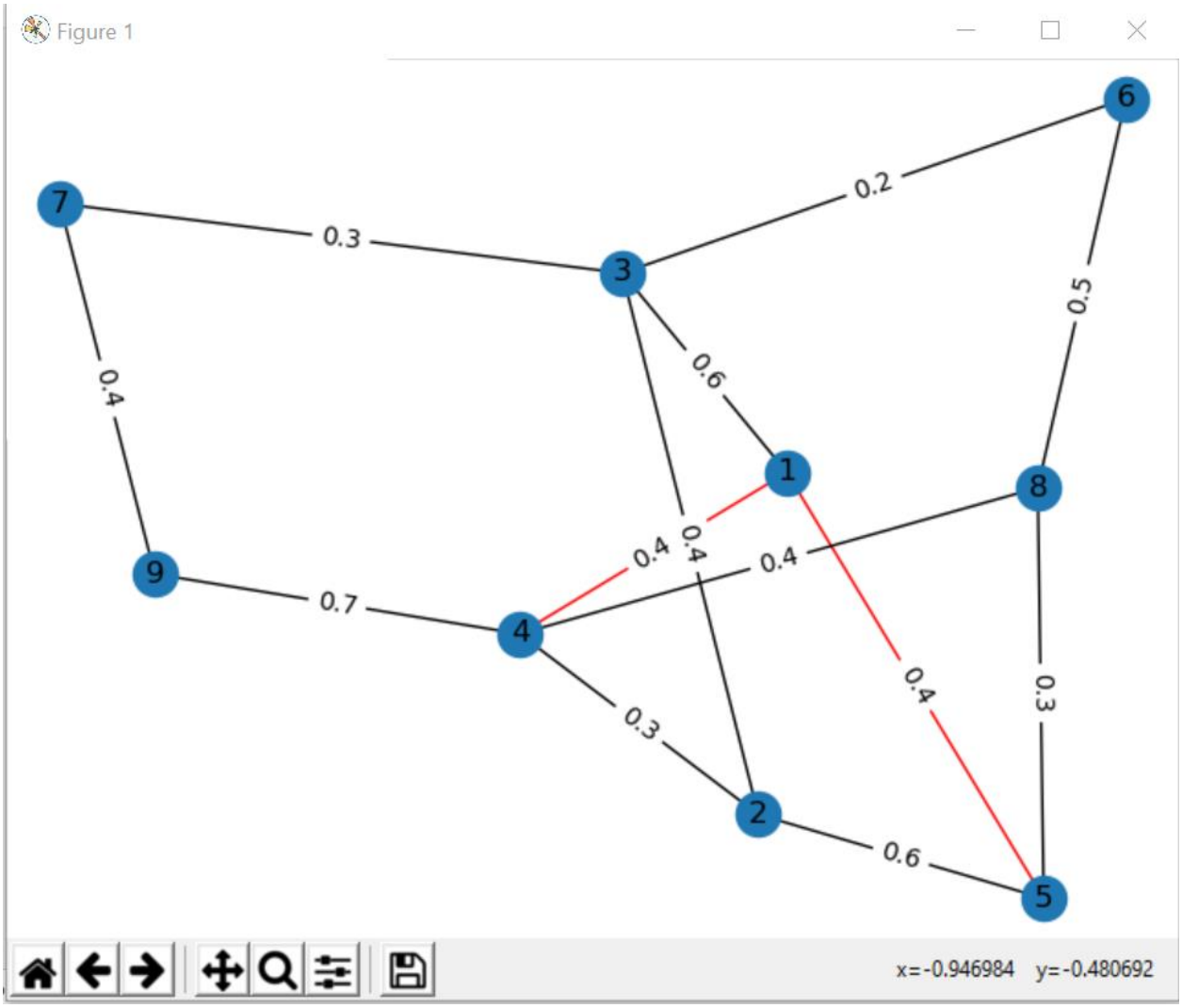


Рис 4.7. Оновлення ваг ребер задіяних у переході

Як можна бачити на рис. 4.7 перехід W4 і W1 збільшив вагу, та W1 і W5 знову збільшив свою вагу на 0.1, отже коефіцієнт завантаженості врахував усі зміни та обчислив новій коефіцієнт завантаженості для даного шляху.

Відповідно якщо перейти по одному і тому шляху двічі нова вага шляху з вершини 7 у вершину 5 буде складати вже не

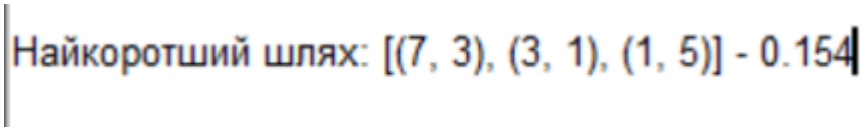


Рис. 4.8 Результат завантаженості шляху W7 – W5.

Буде демонструвати наступний результат завантаженості найкоротшого шляху:

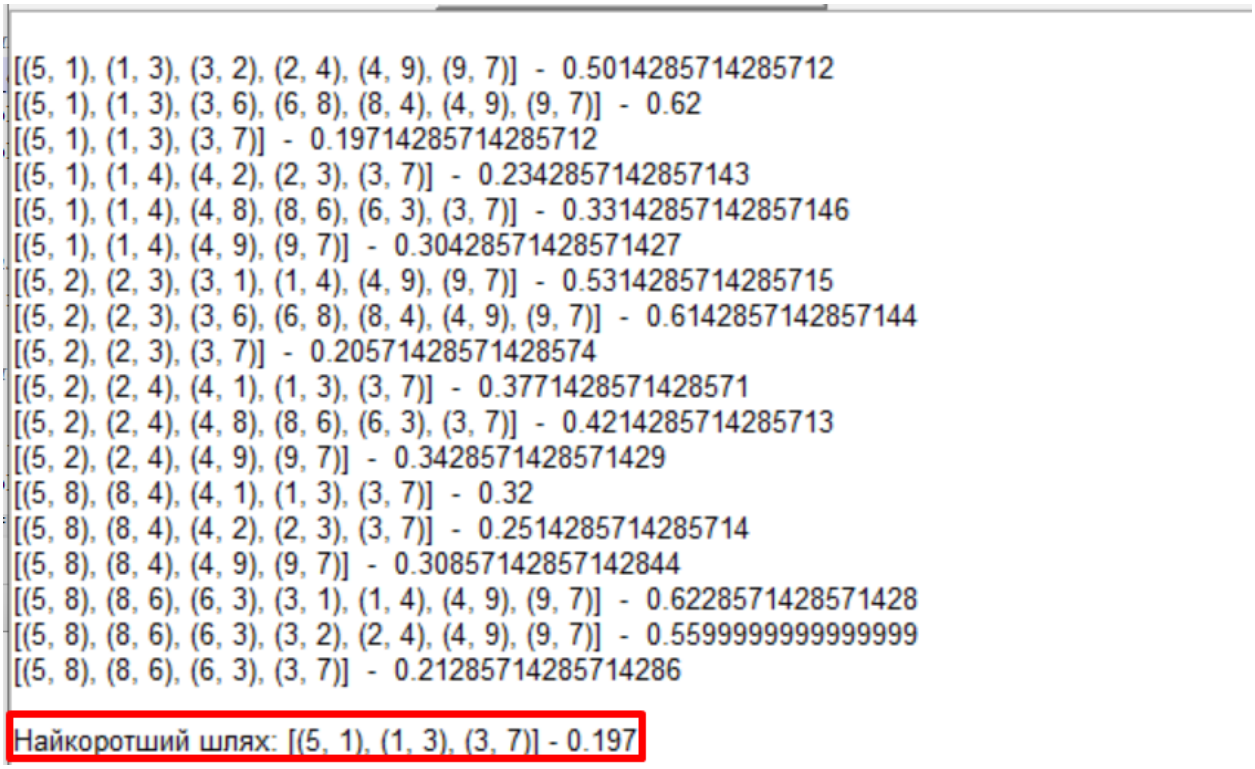


Рис. 4.9 оновлений результат завантаженості шляху W7 – W5.

Також на рис. 4.9 можна помітити що будуть перераховані значення усіх шляхів які мали хоча б одне спільне ребро задіяне у переході з вершини 5 у вершину 7.

ВИСНОВКИ ДО ЧЕТВЕРТОГО РОЗДІЛУ

У даному розділі було розглянуто візуальну частину програми. Було виділено основні переваги інтерфейсу даної програми. Також був налаштований зв’язок між логікою програми.

Була продемонстрована робота даного алгоритму на прикладі графу який був описаний у другій частині.

По завершенню виконання програми отримано бажані результати. Запропонована програма має зручний інтерфейс, непоганий функціонал та швидкодію.

ВИСНОВКИ

В даному проекті запропонований метод балансування трафіку, який за допомогою врахування особливостей організації SDN, зокрема, завдяки наявності в мережі центрального контролера, скорочують час, необхідний для формування декількох маршрутів і спрощення процесу балансування трафіку.

За рахунок безлічі маршрутів є можливість практично виключити затримку або втрату пакетів у випадку ремаршрутизації трафіку. Зокрема, чим більша кількість шляхів буде згенеровано у контролері SDN, тим ймовірність затримки або втрати пакетів буде менше.

Запропонований спосіб формування маршрутної інформації надає можливість забезпечити більш рівномірне завантаження каналів передачі інформації при заданих параметрах QoS.

Подальше вдосконалення методів балансування трафіку пов'язано із застосуванням методів теорії прийняття рішень при виборі шляху з метою прогнозування характеру змін навантаження в каналах зв'язку вчасно передача даних. Це зменшить частоту реконфігурації шляху і забезпечить більш рівномірне навантаження на мережу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Isong, B., Kgogo, T., Lugayizi, F.: Trust establishment in SDN: controller and applications. Int. J. Comput. Netw. Inf. Secur. (IJCNIS) 9(7), 20–28 (2017) . http://dx.doi.org/10.5815/ijcnis.2017.07.03 Published Online July 2017 in MECS (http://www.mecs-press.org/)

2 Sahoo, K.S., Mishra, S.K., Sahoo, S., Sahoo, B.: Software defined network: the next generation internet technology. Int. J. Wirel. Microwave Technol. (IJWMT) 7(2), 13–24 (2017). https://doi.org/10.5815/ijwmt.2017.02.02 Published Online March 2017 in MECS(http://www.mecs-press.net)

3 Mahmood Z. A., Nasir A. A., Fatima W. H.: Evaluating and Comparing the Performance of Using Multiple Controllers in Software Defined Networks. Int. I.J. Mod. Educ. and Comp. Science, (IJMECS) 8, 27-34 (2019). https://doi: 10.5815/ijmecs.2019.08.03. Published Online August 2019 in MECS (http://www.mecs-press.org/) .http://www.mecs-press.org .

4 Abdolhossein. F., Keihaneh K.,": A Centralized Controller as an Approach in Designing NoC", International Journal of Modern Education and Computer Science(IJMECS), Vol.9, No.1, pp.60-67, 2017.DOI:10.5815/ijmecs.2017.01.07. Published Online January 2017 in MECS (http://www.mecs-press.org/)

5 Abdolhossein. F., Keihaneh K.,": A Centralized Controller as an Approach in Designing NoC", International Journal of Modern Education and Computer Science(IJMECS), Vol.9, No.1, pp.60-67, 2017.DOI:10.5815/ijmecs.2017.01.07. Published Online January 2017 in MECS (http://www.mecs-press.org/)

6 Ashutosh K. S. , Naveen K., Shashank S.: PSO and TLBO based Reliable Placement of Controllers in SDN. Int. I. J. Comput. Netw. Inf. Secur.(IJCNIS), 2, 36-42 (2019). https:// doi. org/10.5815/ijcnis.2019.02.05. Published Online January 2017 in MECS http://www.mecs-press.org

7 Gamess E., Tovar D., Cavadia A.: Design and Implementation of a Benchmarking Tool for OpenFlow Controllers , International Journal of Information Technology and Computer Science(IJITCS), Vol.10, No.11, pp.1-13, 2018. DOI: 10.5815/ijitcs.2018.11.01 Published Online November 2018 in MECS (http://www.mecs-press.org/)

8 Gaurav Sharma, Manoj Singh, Prashant Sharma,"Modifying AODV to Reduce Load inMANETs", International Journal of Modern Education and Computer Science(IJMECS), Vol.8, No.10, pp.25-32, 2016.DOI: 10.5815/ijmecs.2016.10.04. Published Online October 2016 in MECS (http://www.mecs-press.org/)

9 Yasir A.,M., Chasib H., A., Zainab, Q., M. : Analyzing Methods and Opportunities in Software-Defined (SDN) Networks for Data Traffic Optimizations. Int. Inter. J. Rec. Inn. Trends Comp. Com. (IJRITCC) 6 (1), 75 – 82 (2018) http://www.ijritcc.org

10 Haibo. W., Hongli. X., Liusheng. H., Jianxin. W., Xuwei.Y. Load-balancing routing in software defined networks with multiple controllers. In Computer Networks Volume 141, 4 August 2018, Pages 82-91 https://doi.org/10.1016/j.comnet.2018.05.012

11 Sikandar Ejaz , Zeshan Iqbal , Peer Azmat Shah , Bilal Haider Bukhari , Armughan Ali , And Farhan Aadil Traffic Load Balancing Using Software Defined Networking (SDN) Controller as Virtualized Network Function Received February 8, 2019, accepted March 21, 2019, date of publication April 4, 2019, date of current version April 18, 2019. Digital Object Identifier 10.1109/ACCESS.2019.2909356

12 Sikandar Ejaz , Zeshan Iqbal , Peer Azmat Shah , Bilal Haider Bukhari , Armughan Ali , And Farhan Aadil Traffic Load Balancing Using Software Defined Networking (SDN) Controller as Virtualized Network Function Received February 8, 2019, accepted March 21, 2019, date of publication April 4, 2019, date of current version April 18, 2019. Digital Object Identifier 10.1109/ACCESS.2019.2909356

13 Kulakov, Y., Kogan, A.: The method of plurality generation of disjoint paths using horizontal exclusive scheduling. Adv. Sci. J. 10, 16–18 (2014). https://doi.org/10.15550/ASJ.2014.10. ISSN 2219-746X

14 Rajasekaran K., Balasubramanian K.: Energy Conscious based Multipath Routing Algorithm in WSN. Int. J. Comput. Netw. Inf. Secur. (IJCNIS), 1, 27-34 (2016). https://doi.org/10.5815/ijcnis.2016.01.04 in MECS (http://www.mecs-press.org/)

15 Kumar, P., Dutta, R., Dagdi, R., Sooda, K., Naik, A.: A programmable and Managed Software Defined Network. Int. J. Comput. Netw. Inf. Secur. (IJCNIS) 12, 11–17 (2017). https://doi.org/10.5815/ijcnis.2017.12.02. In MECS http://www.mecs-press.org/. Accessed Dec 2017.

16 Moza, M., Kumar, S.: Analyzing multiple routing configuration. Int. J. Comput. Netw. Inf. Secur. (IJCNIS) 5, 48–54 (2016). https://doi.org/10.5815/ijcnis.2016.05.07. In MECS http://www.mecs-press.org/. Accessed may 2016

17 Yurii Kulakov Traffic Orchestration in Data Center Network based on Software-Defined Networking Technology Yurii Kulakov, Alla Kohan, Sergii Kopychko International Conference on Computer Science, Engineering and Education Applications ICCSEEA 2019: Advances in Computer Science for Engineering and Education II pp 228-237

18 Shu, Z., Wan, J., Lin, J., Wang, S., Li, D., Rho, S., Yang, C.: Traffic engineering in softwaredefined networking: measurement and management. IEEE Access 4, 3246–3256 (2016). http://www.ieee.org/publications_standards/publications/rights/index.html

19 Abbasi, M.R., Guleria, A., Devi, M.S.: Traffic engineering in software defined networks: a survey. J. Telecommun. Inf. Technol. 4, 3–13 (2016).

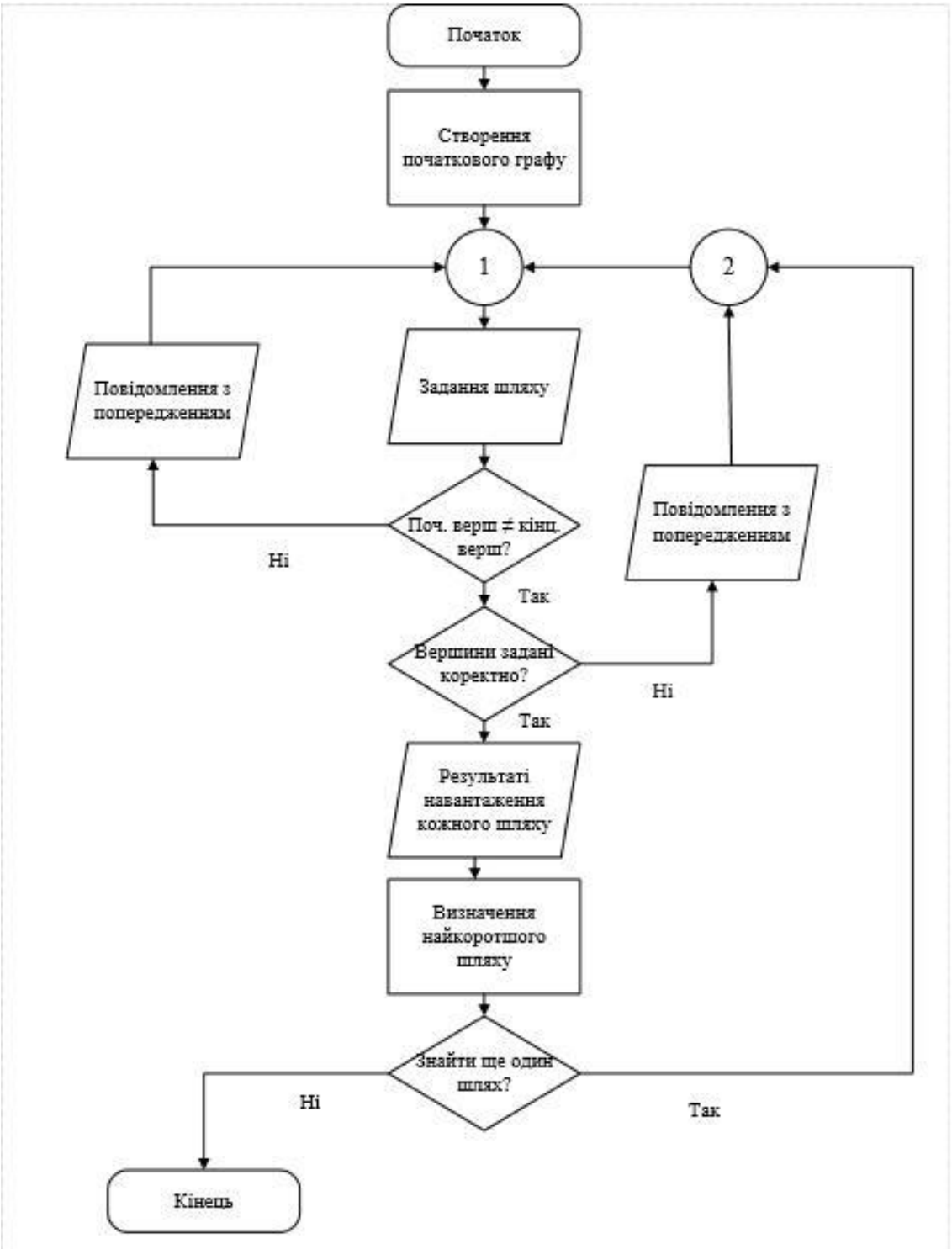
ДОДАТОК 1

Система оптимізації трафіку в транспортних мобільних мережах

Принципова схема апаратного забезпечення системи
ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2020 р.



					ІАЛЦ.467200.004 Д1		
Зм.		*№ докум.*	*Підпис*	*Дата*			
Розробив		*ЧеревatенкоР,В*			Система оптимізації трафіку в транспортних мобільних мережах Принципова схема	*Літ.*	*Аркуш*	*Аркушів*
Перевірив		Калюжний О.О					*1*	*1*
						НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-64		
Н. Контр.		*Сімоненко В.П.*						
Затвердив		*Стіренко* С.Г.						

ДОДАТОК 2

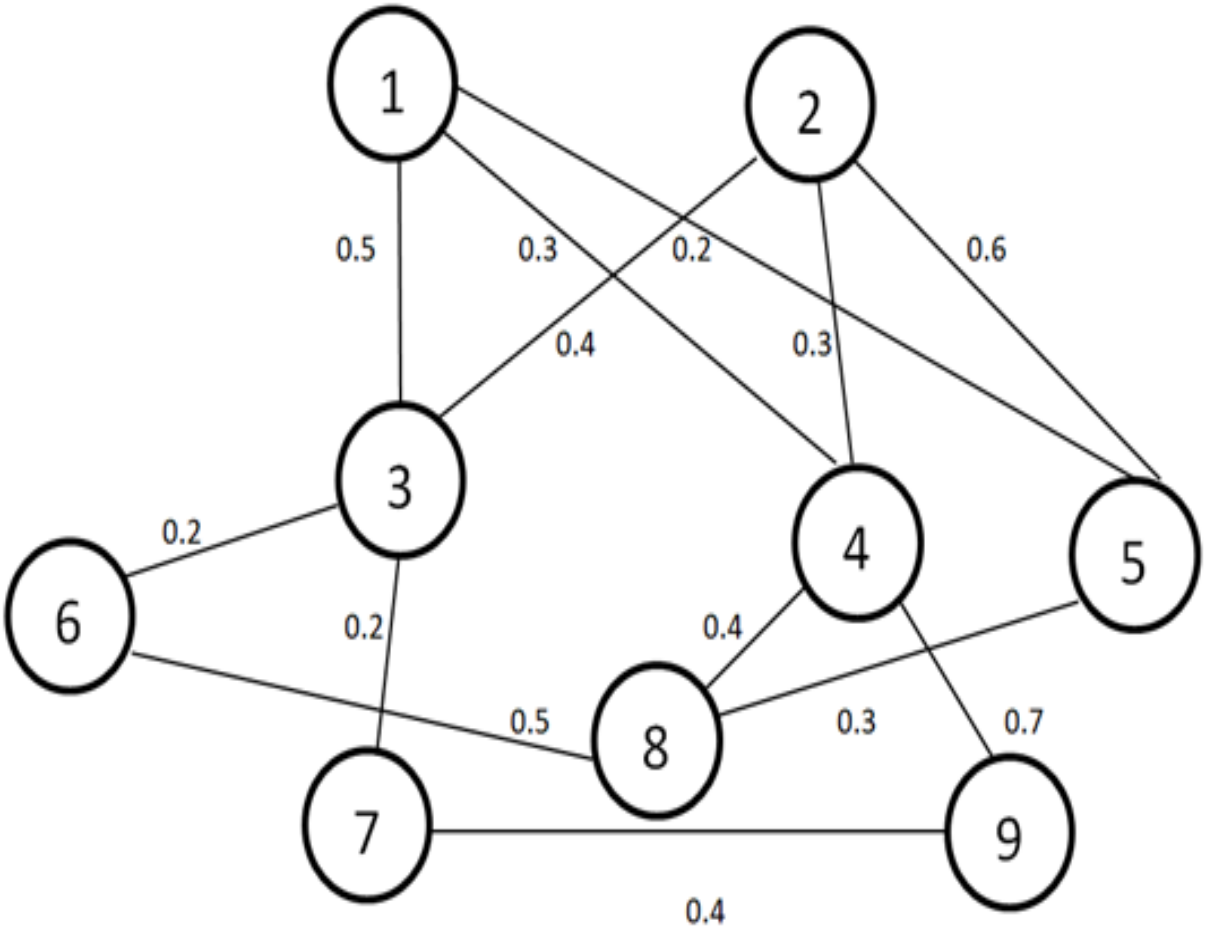
Система оптимізації трафіку в транспортних мобільних мережах

Структурна схема пристрою

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2020р.



					ІАЛЦ.467200.005 Д2	
Зм.		*№ документа*	*Підпис*	*Дата*		
Розробив		*ЧеревatенкоР,В*			Система оптимізації трафіку в транспортних мобільних мережах Структурна схема	*Літ.* | *Аркуш* | *Аркушів* *1* | *1*
Перевірив		Калюжний О.О				
Н. Контр.		*Сімоненко В.П.*				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-64*
Затвердив		*Стіренко* С.Г.				

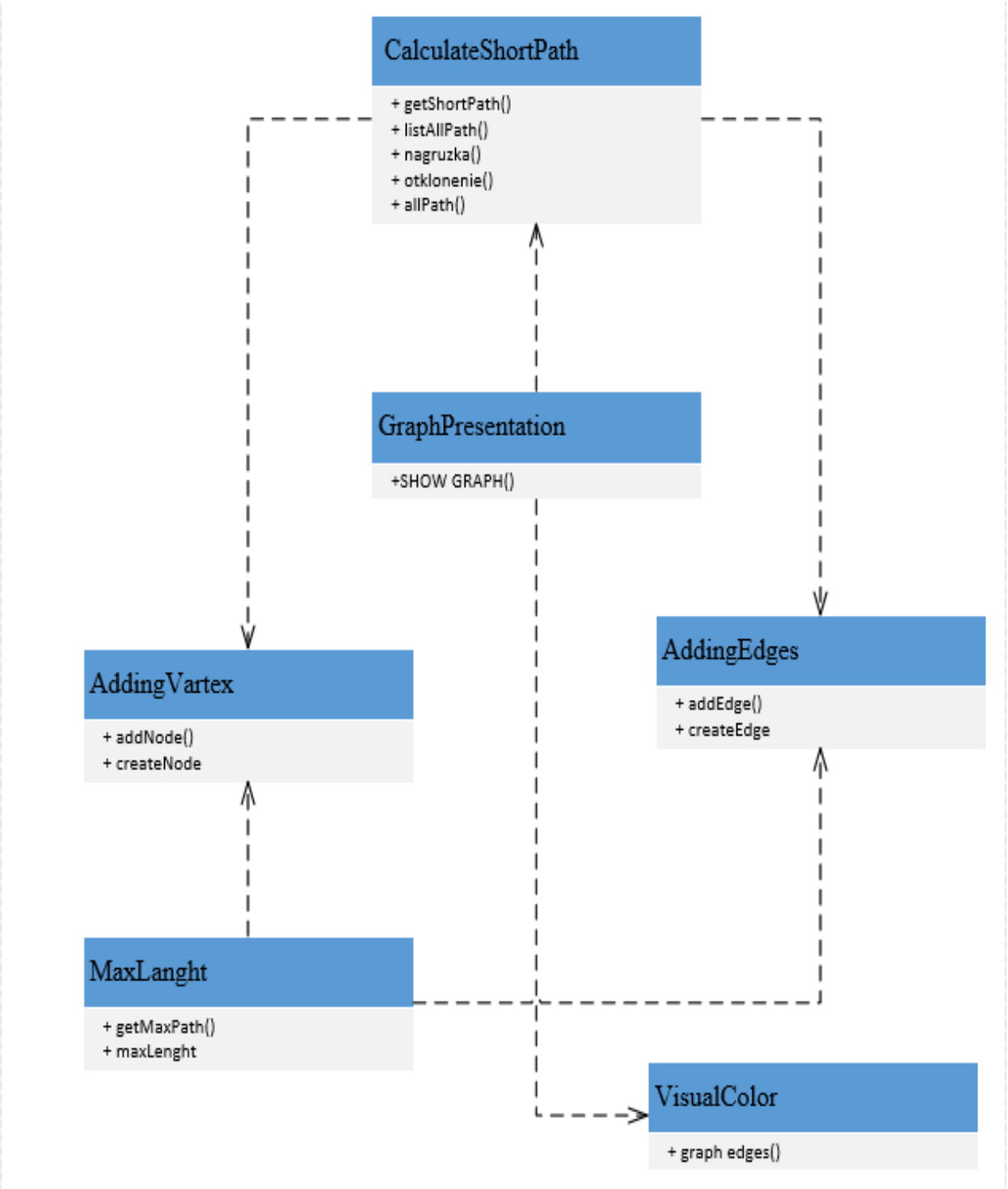
ДОДАТОК 3

Система оптимізації трафіку в транспортних мобільних мережах

Функціональна схема алгоритму балансування трафіку
ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2020 р.



					ІАЛЦ.467200.006 Д3	
Зм.		№ документа	Підпис	Дата		
Розробив		ЧереватенкоР,В			Система оптимізації трафіку в транспортних мобільних мережах Функціональна схема	Літ. / Аркуш: 1 / Аркушів: 1
Перевірив		Калюжний О.О				
Н. Контр.		Сімоненко В.П.				НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-64
Затвердив		Стіренко С.Г.				

ДОДАТОК 4

Система оптимізації трафіку в транспортних мобільних мережах

Текст програми

ІАЛЦ.467200.007 Д4

Аркушів 3

Київ 2020

```
import itertools
from tkinter import *
import networkx as nx
import numpy.random as rnd
import matplotlib.pyplot as plt


# функция добавления рёбер
def add_edge(f_item=None, s_item=None, weight=None, graph=None, color=None):
    graph.add_edge(f_item, s_item, weight=weight, color=color)
    graph.add_edge(s_item, f_item, weight=weight, color=color)


# инициализация графа
graph = nx.Graph()

# добавление вершин
i = 1
while i < 10:
    graph.add_node(i)
    i += 1
# добавления рёбер
add_edge(f_item=1, s_item=3, graph=graph, color='black',
         weight=0.5)  # с какой вершины, в какую, к какому графу, цвет ребра,
вес ребра
add_edge(f_item=1, s_item=4, graph=graph, color='black', weight=0.3)
add_edge(f_item=1, s_item=5, graph=graph, color='black', weight=0.2)
add_edge(f_item=2, s_item=3, graph=graph, color='black', weight=0.4)
add_edge(f_item=2, s_item=4, graph=graph, color='black', weight=0.3)
add_edge(f_item=2, s_item=5, graph=graph, color='black', weight=0.6)
add_edge(f_item=3, s_item=6, graph=graph, color='black', weight=0.2)
add_edge(f_item=3, s_item=7, graph=graph, color='black', weight=0.2)
add_edge(f_item=4, s_item=8, graph=graph, color='black', weight=0.4)
add_edge(f_item=4, s_item=9, graph=graph, color='black', weight=0.7)
add_edge(f_item=5, s_item=8, graph=graph, color='black', weight=0.3)
add_edge(f_item=6, s_item=8, graph=graph, color='black', weight=0.5)
add_edge(f_item=7, s_item=9, graph=graph, color='black', weight=0.4)

pos = nx.spring_layout(graph)  # чтобы позиция графа не менялась
root = Tk()
root.title('Вікно відображення усіх можливих шляхів між вершинами X та Y')
entry_start1 = IntVar()
entry_end1 = IntVar()
label1=Label(root,text='Початкова вершина:').pack()
entry_start = Entry(root, bd=5, textvariable=entry_start1).pack()  # поле
ввода начальной вершины
label2=Label(root,text='Кінцева вершина:').pack()
entry_end = Entry(root, bd=5, textvariable=entry_end1).pack()  # поле ввода
конечной вершины


# функция находит максимально длинный путь и считает количество его переходов
def get_max_path(all_paths):
    max_lenght = 0
    for paths in all_paths:
        if len(paths) > max_lenght:
            max_lenght = len(paths)
    return max_lenght


# функция нахождения кратчайшего пути
def get_short_path(graph, source, target):
    list_all_paths = nx.all_simple_paths(graph, source, target)  # находим
```

```
    все возможные пути
        result_path = []
        result_koef_zagruzki = 100000000000
        all_paths = [list(i) for i in map(nx.utils.pairwise, list_all_paths)]  #
    добавляем все возможные пути в список
        for paths in all_paths:
            lenght = 0
            for path in paths:
                lenght += graph[path[0]][path[1]]['weight']
            nagruzka = lenght / len(paths)  # Нахождение средней загруженности
    пути канала:
            otklonenie = nagruzka
            for path in paths:  # отклонение значения нагрузки канала L от
    средней нагрузки канала.
                x = nagruzka - graph[path[0]][path[1]]['weight']
                otklonenie += (abs(x)) ** 2
            koef_zagruzki = (otklonenie * len(paths)) / get_max_path(
                all_paths)  # среднеквадратичное отклонение нагрузки канала.
            txt = text1.get('1.0', END)
            txt = txt + f'{paths}  -  {koef_zagruzki}'
            text1.delete(1.0, END)
            text1.insert(1.0, txt)
            if koef_zagruzki < result_koef_zagruzki:
                result_path = paths
                result_koef_zagruzki = round(koef_zagruzki, 3)
        for e in graph.edges():
            graph[e[0]][e[1]]['color'] = 'black'  # делаем все ребра черными
        for p in result_path:
            n = round((graph[p[0]][p[1]]['weight'] + 0.1), 1)  # добавляем +0.1
    вес до всех ребер пути
            graph[p[0]][p[1]]['weight'] = n
            graph[p[0]][p[1]]['color'] = 'red'  # делаем их красными
        return result_path, result_koef_zagruzki


    # функция вывода графа
    def show_graph():
        plt.clf()
        edges = graph.edges()
        colors = [graph[u][v]['color'] for u, v in edges]
        nx.draw(graph, pos, with_labels=True, edge_color=colors)
        nx.draw_networkx_edge_labels(graph, pos,
    edge_labels=nx.get_edge_attributes(graph, 'weight'))
        plt.show()


    # функция для кнопки Поиск
    def click_button():
        text1.delete(1.0, END)
        if int(entry_start1.get()) == int(entry_end1.get()):
            txt = text1.get('1.0', END)
            txt = txt + '\nВи знаходитесь  у даній вершині'
            text1.delete(1.0, END)
            text1.insert(1.0, txt)
        elif int(entry_end1.get()) > 9 or int(entry_end1.get()) < 0:
            txt = text1.get('1.0', END)
            txt = txt + f'\nДля переходу в вершину {entry_end1.get()} необхідно
    насамперед її створити'
            text1.delete(1.0, END)
            text1.insert(1.0, txt)
```

```
 else:
        p, k = get_short_path(graph, int(entry_start1.get()),
int(entry_end1.get()))
        txt = text1.get('1.0', END)
        txt = txt + f'\nНайкоротший шлях: {p} - {k}'
        text1.delete(1.0, END)
        text1.insert(1.0, txt)
        show_graph()


button1 = Button(root, text="Пошук усіх можливих шляхів",
command=click_button)
button1.pack()
text1 = Text(root, font='Arial 10', wrap=WORD)
text1.pack()
show_graph()
root.mainloop()
```