**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

|  |  |
| --- | --- |
| «На правах рукопису»  УДК \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | До захисту допущено:  Завідувач кафедри  \_\_\_\_\_\_\_\_ Сергій СТІРЕНКО  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2021 р. |

**Магістерська дисертація**

**на здобуття ступеня магістра**

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

**зі спеціальності 123 «Комп'ютерна інженерія»**

**на тему: «Спосіб конструювання трафіка в SDN мережах на основі лямбда-архітектури»**

Виконав:

студент VІ курсу, групи ІО-01мп

Бабко Дмитро Сергійович \_\_\_\_\_\_\_\_\_\_

Керівник:

ст.викл., к.т.н.,

Куц Володимир Юрійович                                                        \_\_\_\_\_\_\_\_\_\_

Консультант з нормоконтролю:

проф., д.т.н., проф.,

Кулаков Юрій Олексійович \_\_\_\_\_\_\_\_\_\_

Рецензент:

доц. каф. СПіСКС, к.т.н., доц.,

Орлова Марія Миколаївна \_\_\_\_\_\_\_\_\_\_

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2021 року

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

Факультет *Інформатики та обчислювальної техніки*

(повна назва)

Кафедра *Обчислювальної техніки* (повна назва)

Рівень вищої освіти – другий (магістерський)

Спеціальність *123.  Комп’ютерна інженерія*

(код і назва)

Освітньо-професійна програма *Комп’ютерні системи та мережі*

(код і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_ Сергій СТІРЕНКО

(підпис)

« » 2021 р.

**ЗАВДАННЯ**

**на магістерську дисертацію студенту**

Бабку Дмитру Сергійовичу

(прізвище, ім’я, по батькові)

1. Тема дисертації Спосіб конструювання трафіка в SDN мережах на основі лямбда-архітектури

Науковий керівник дисертації Куц Володимир Юрійович, старший викладач, кандидат технічних наук

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» жовтня 2021 р. № 3587-с

2. Строк подання студентом дисертації  24.11.2021

3. Об’єкт дослідження процес конструювання трафіка в SDN мережах

4. Предмет дослідження процес збору та обробки мережевих даних для побудови матриці трафіка на основі лямбда-архітектури

5. Перелік завдань, які потрібно розробити: узагальнення та аналіз способів конструювання трафіка, дослідження та опис технології роботи лямбда-архітектури, розробка способу конструювання трафіка на основі лямбда-архітектури, реалізація моделі конструювання трафіка та аналіз отриманих результатів. \_\_\_\_\_\_\_

6. Консультанти розділів дисертації:

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
| завдання видав | завдання прийняв |
| 1. | ст.викл., Куц В.Ю. |  |  |
| 2. | ст.викл., Куц В.Ю. |  |  |
| 3. | ст.викл., Куц В.Ю. |  |  |
| 4. | ст.викл., Куц В.Ю. |  |  |
|  |  |  |  |

7. Дата видачі завдання 1.09.2021 \_\_\_\_\_\_\_\_\_\_\_\_\_

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів виконання  магістерської дисертації | Строк виконання  етапів дисертації | Примітка |
| 1. | Затвердження теми дослідження. Визначення предмету дослідження | 1.09.21-7.09.21 |  |
| 2. | Дослідження існуючих рішень та проблем конструювання трафіка в досліджуваній області | 8.09.21-21.09.21 |  |
| 3. | Побудова та обґрунтування архітектурного рішення | 22.09.21-28.09.21 |  |
| 4. | Моделювання генерації трафіка в SDN мережі | 29.09.21-05.10.21 |  |
| 5. | Реалізація системи безперервної потокової передачі повідомлень | 06.10.21-19.10.21 |  |
| 5. | Розробка процесу обробки, агрегації та акумуляції мережевих даних | 20.10.21-09.11.21 |  |
| 6. | Тестування моделі та аналіз її ефективності. | 10.11.21-13.11.21 |  |
| 7. | Розробка стартап-проекту | 14.11.21-18.11.21 |  |
| 8. | Оформлення магістерської дисертації. | 19.11.21-24.11.21 |  |

Студент                         \_\_\_\_\_Дмитро БАБКО\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(підпис)

Науковий керівник дисертації  \_\_\_Володимир КУЦ\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(підпис)

**РЕФЕРАТ**

**на магістерську дисертацію**

виконану на тему: Спосіб конструювання трафіка в SDN мережах на основі лямбда-архітектури

студентом: Бабком Дмитром Сергійовичем

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 90 аркушів основного тексту, 21 ілюстрація, 23 таблиць. При підготовці використовувалася література з 41 різного джерела.

**Актуальність.** На сьогоднішній день спостерігається стрімкий розвиток програмно-конфігурованих мереж. Збільшення розмірів мереж та різноманітність контенту, що продукується, призводить до ускладнення контролю над якістю та надійністю мережевої комунікації. Традиційні способи конструювання трафіка не адаптовані для оперативної роботи у високонавантаженому середовищі, що формує проблему для пошуку нових рішень конструювання.

Одна із проблем конструювання трафіка в SDN мережах, що має бути вирішена - це забезпечення швидкодії та стабільності при активному рості розмірів мереж. Процес конструювання трафіка, заснований на сучасних методах роботи з великими обсягами даних, дозволить оперативно виявляти та реагувати на проблемні ситуації в мережі та підвищити якість і надійність впроваджених мережевих рішень, а отже і покращити клієнтський досвід та посилити позиції SDN для впровадження у різних середовищах.

Методи роботи з Big Data, зокрема лямбда-архітектура, демонструють здатність ефективної роботи з великими масивами та успішно впроваджується у різних сферах хмарних обчислень та при роботі з Інтернетом речей, тому такий підхід може стати вирішенням проблеми конструювання мережевого трафіка.

**Мета і завдання дослідження.** Метою магістерської роботи є покращення швидкісних та надійнісних характеристик процесу конструювання трафіка шляхом застосування лямбда-архітектури для процесів збору та аналізу мережевих даних.

Для досягнення мети дослідження поставлено і вирішено такі завдання:

* узагальнення та аналіз способів конструювання трафіка;
* дослідження та опис технології роботи лямбда-архітектури;
* розробка способу конструювання трафіка на основі лямбда-архітектури;
* реалізація моделі конструювання трафіка та аналіз отриманих результатів.

**Об’єкт дослідження**– процес конструювання трафіка в SDN мережах.

**Предмет дослідження** – процес збору та обробки мережевих даних для побудови матриці трафіка на основі лямбда-архітектури.

**Методи досліджень*.*** Для досягнення поставлених в магістерській роботі задач використано методи теорії графів, методи імітаційного моделювання, методи роботи з великими даними.

**Наукова новизна** одержаних результатів роботи полягає у наступному:

* запропоновано спосіб конструювання трафіка, що дозволяє покращити швидкісні та надійнісні характеристики процесів збору та аналізу мережевих даних за рахунок застосування методу роботи з великими даними;
* розроблено кросс-платформену модель способу конструювання трафіка на основі лямбда-архітектури, що може бути інтегрована в існуючі архітектури SDN мереж.

Проведене дослідження дає змогу використання розробленої моделі конструювання трафіка в мережах SDN та її застосування для покращення процесів збору та аналізу мережевих даних.

**Особистий внесок здобувача.** Магістерське дослідження є самостійно виконаною роботою, в якій відображено особистий авторський підхід та особисто отримані теоретичні та прикладні результати, що відносяться до вирішення задачі конструювання трафіка в SDN. Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

**Практична цінність.** Отримані результати можуть використовуватися у майбутніх дослідженнях за напрямками:

* вдосконалення методів конструювання трафіка;
* розробка методів прогнозування трафіка;
* вирішення задачі балансування трафіка в SDN мережах;
* розробка аналітичних підходів до задачі конструювання трафіка.

**Ключові слова**

Конструювання трафіка, програмно-конфігуровані мережі, SDN, Big Data, лямбда-архітектура.

**ЗМІСТ**

[**ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ** 9](#_Toc88401282)

[**ВСТУП** 10](#_Toc88401283)

[**РОЗДІЛ 1. СПОСОБИ КОНСТРУЮВАННЯ ТРАФІКА В МЕРЕЖАХ SDN** 12](#_Toc88401284)

[**1.1.** **Технологічний огляд програмно-конфігурованих мереж** 12](#_Toc88401286)

[1.1.1. Структура програмно-конфігурованих мереж 12](#_Toc88401287)

[1.1.2. OpenFlow протокол 17](#_Toc88401288)

[**1.2.** **Концептуальний огляд методів Big Data** 21](#_Toc88401289)

[1.2.1. Теоретичні аспекти Big Data 21](#_Toc88401290)

[1.2.2. Пакетна обробка даних та MapReduce 24](#_Toc88401291)

[1.2.3. Обробка даних в реальному часі та лямбда-архітектура 26](#_Toc88401292)

[**1.3. Огляд способів конструювання трафіка в програмно-конфігурованих мережах** 29](#_Toc88401294)

[1.3.1. Види та задачі конструювання трафіка в SDN 29](#_Toc88401295)

[1.3.2. Огляд існуючих рішень для конструювання трафіка 33](#_Toc88401296)

[**Висновки до розділу 1** 37](#_Toc88401297)

[**РОЗДІЛ 2. РОЗРОБКА СПОСОБУ КОНСТРУЮВАННЯ ТРАФІКА НА ОСНОВІ ЛЯМБДА-АРХІТЕКТУРИ** 38](#_Toc88401298)

[**2.1.** **Спосіб конструювання трафіка на основі мережевих даних** 38](#_Toc88401300)

[**2.2.** **Формування повідомлень** 46](#_Toc88401301)

[**2.3.** **Лямбда-архітектура у задачі конструювання трафіка** 49](#_Toc88401302)

[**Висновки до розділу 2** 56](#_Toc88401303)

[**РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОДЕЛІ СПОСОБУ КОНСТРУЮВАННЯ ТРАФІКА** 57](#_Toc88401304)

[**3.1.** **Огляд засобів реалізації** 57](#_Toc88401306)

[**3.2.** **Структура програмного додатку** 59](#_Toc88401307)

[**3.3.** **Моделювання способу конструювання трафіка** 67](#_Toc88401308)

[**Висновки до розділу 3** 76](#_Toc88401309)

[**РОЗДІЛ 4. РОЗРОБКА СТАРТАП ПРОЕКТУ** 77](#_Toc88401310)

[**4.1.** **Маркетинговий аналіз стартап-проекту** 77](#_Toc88401312)

[**4.2.** **Технологічний аудит ідеї проекту** 80](#_Toc88401313)

[**4.3.** **Аналіз ринкових можливостей запуску стартап-проекту** 81](#_Toc88401314)

[**4.4.** **Розробка ринкової стратегії стартап-проекту** 91](#_Toc88401315)

[**4.5.** **Розробка маркетингової програми стартап-проекту** 95](#_Toc88401316)

[**Висновки до розділу 4** 99](#_Toc88401317)

[**ВИСНОВКИ** 100](#_Toc88401318)

[**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ** 102](#_Toc88401319)

# **ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ**

API (Address Resolution Protocol) Прикладний програмний інтерфейс

BR (Bytes Received) Байт отримано

BRT (Bytes Received Throughput) Пропускна здатність по отриманих байтах

BT (Bytes Transmitted) Байт передано

BTr (Bytes Throughput) Пропускна здатність по байтах

BTT (Bytes Transmitted Throughput) Пропускна здатність по переданих байтах

ER (Entity-Relationship) Модель «сутність-відношення»

IoT (Internet of Things) Інтернет речей

IP (Internet Protocol) Інтернет протокол

OD (Origin-Destination) старт – призначення

FT (Flow Table) Таблиця потоків

PR (Bytes Received) Пакетів отримано

PRT (Bytes Received Throughput) Пропускна здатність по отриманих пакетах

PT (Bytes Transmitted) Пакетів передано

PTr (Bytes Throughput) Пропускна здатність по пакетах

PTT (Bytes Transmitted Throughput) Пропускна здатність по переданих пакет ах

SDN (Software-defined Networks) Програмно-конфігуровані мережі

TE (Traffic Engineering) Конструювання трафіка

TM (Traffic Matrix) Матриця трафіка

QoS (Quality of Service) Якість обслуговування мережі

# **ВСТУП**

Реалії сучасного цифрового суспільства такі, що обсяги різноманітних електронних послуг стрімко зростають. Хмарні та розподілені обчислення, Інтернет речей, блокчейн та урізноманітнення і збагачення цифрового контенту потребують від мережевих рішень високої якості, гнучкості та стабільності. Велика кількість сервісів, обсяги використання яких мають тенденцію часто змінюватись, впливає на розмір експлуатованих мереж та на неоднорідність мережевого обладнання від різних виробників. Ці особливості суттєво ускладнюють можливість виконання процесу traffic engineering (конструювання трафіка, TE), що має на меті покращення стабільності роботи мережі і є особливо актуальним для топологій великих масштабів.

Традиційні мережеві рішення наразі розвиваються у вільному темпі в не завжди відповідають вимогам сервісів сучасності. У зв’язку з цим останні роки спостерігається зростання популярності технології програмно-конфігурованих мереж (SDN). Цей підхід виділяє певний рівень абстракції для комутаційного обладнання і дозволяє розділити менеджмент від безпосередньо передачі даних за допомогою централізованого керування на заміну звичного розподіленого менеджменту. Інша особливість таких мереж – це можливість програмної конфігурації при використанні відкриті та уніфіковані програмні інтерфейси. також дозволяє мережі бути програмно-конфігурованою, використовуючи відкриті і програмовані інтерфейси.

Швидка зміна активності сервісів, потенційні проблеми перевантаженості глобальної мережі та зростання обсягу даних, що передаються, ускладнює проблему конструювання трафіка, що особливо характерно для традиційних методів моніторингу та аналізу. Такі підходи є вразливими для опрацювання даних в режимі реального часу.

Метою магістерської роботи є підвищення швидкодії та надійності конструювання трафіка в високонавантажених SDN мережах.

Для досягнення мети поставлено наступні завдання:

* аналіз існуючих підходів до конструювання трафіка в традиційних та SDN мережах;
* дослідження сучасних концептів оперативного опрацювання великих обсягів даних згідно з принципами Big Data;
* розробка способу конструювання трафіка в SDN мережах;
* комп’ютерне моделювання системи конструювання трафіку на основі способу.

Об’єктом дослідження є процес конструювання трафіку в високонавантажених програмно-конфігурованих мережах. Предметом дослідження є архітектурне рішення Big Data системи моніторингу та аналізу трафіка.

Для виконання поставлених задач використано методи теорії графів, методи імітаційного моделювання, методи абстрагування.

Наукова новизна отриманих результатів полягає в наступному:

* запропоновано спосіб конструювання трафіку, що дозволяє пришвидшити отримання актуальної статистичної інформації за рахунок використання потокового опрацювання даних;
* запропоновано спосіб побудови архітектури, що забезпечує надійність та цілісність історичних даних у випадку непередбачуваних збоїв.

# **РОЗДІЛ 1**

# **СПОСОБИ КОСТРУЮВАННЯ ТРАФІКА В МЕРЕЖАХ SDN**

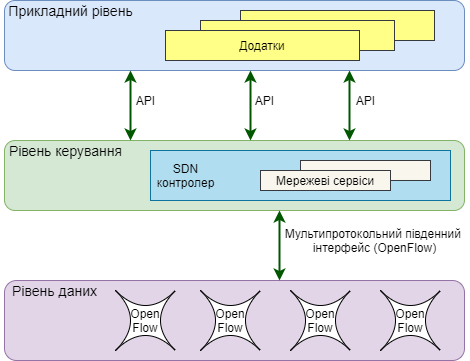
## **Технологічний огляд програмно-конфігурованих мереж**

### *Структура програмно-конфігурованих мереж*

Програмно-конфігуровані мережі (Software-defined networks, SDN) мають модель архітектури мережі, що дозволяє програмне керування, контроль та оптимізацію мережевих ресурсів. SDN відокремлює конфігурацію мережі та конструювання трафіка від базової апаратної інфраструктури задля забезпечення повного і послідовного контролю над мережею за допомогою відкритих прикладних програмних інтерфейсів (API). Існують способи використання відкритих протоколів, таких як OpenFlow, які можуть застосовувати глобально відоме програмне забезпечення для доступу до мережевих комутаторів та маршрутизаторів, що зазвичай мають передвстановлене закрите та пропрієтарне програмне забезпечення [1].

Основою інфраструктури традиційних мереж є пристрої пересилки, такі як комутатори та маршрутизатори. Вони мають два логічних компоненти для керування трафіком: рівень керування (що вирішує, як опрацьовувати мережевий трафік) і рівень даних (що направляє трафік на основі рішення рівня керування) [2]. При такому підході керування потоками трафіка є досить ресурсномісткою задачею, особливо для великомасштабних навантажених мереж, так як будь-яка нова конфігурація має бути реалізована вручну на кожному з пристроїв. Конфігурація пристроїв пересилки також суттєво залежить від виробника: різні виробники мають власні набори правил для налаштування своїх пристроїв, що також негативно впливає на доступність процесу керування.

У зв’язку з цим і була запропонована нова мережева архітектура для подолання проблем і обмежень традиційних мереж. Програмно-конфігурована мережа (SDN*)* – мережа передачі даних, у якій рівень керування відділений від пристроїв передачі даних і реалізується програмно. SDN є динамічною, керованою, економічною і адаптованою архітектурою, що робить її ідеальною для високошвидкісної, динамічної природи сучасних додатків. Ця архітектура дозволяє напряму керувати програмуванням мережі і абстрагувати базову інфраструктуру для додатків і мережевих служб [3]. Архітектура SDN наведена на рис. 1.1 [4].



*Рис. 1.1. Структура SDN*

Архітектура SDN зазвичай має три компоненти або групи функціональних можливостей [5]:

1. *Прикладний рівень* включає в себе різні додатки, що мають доступ до керування та модифікації роботи контролера. Додатки SDN — це певні програми, що передають поведінку та необхідні ресурси з контролером SDN прикладні програмні інтерфейси. Крім того, програми можуть створювати абстрактне уявлення про мережу, збираючи інформацію від контролера для прийняття рішень,. Ці програми можуть включати управління мережами, аналітику або бізнес-додатки, які використовуються для роботи великих центрів обробки даних. Наприклад, аналітичну програму можна створити для розпізнавання підозрілої активності в мережі з метою безпеки, або для визначення системи конструювання трафіку у високонавантаженій мережі
2. Контроль над мережею винесено окремо на *рівень керування,* що включає в себе контролер або множину контролерів. Контролер SDN - це логічний об'єкт, який отримує інструкції або вимоги від прикладного рівня SDN і передає їх мережевим компонентам. Контролер також витягує інформацію про мережу з апаратних пристроїв і передає назад на прикладний рівень SDN з абстрактним уявленням про мережу, включаючи статистику та події про те, що відбувається.
3. *Рівень інфраструктури*, або *рівень даних*, має на меті взаємопов’язані пристрої пересилки. Такі пристрої мають визначений набір базових функцій, що визначають послідовності дій, які мають бути виконані для будь-яких вхідних пакетів даних (наприклад, пересилка на потрібні порти, пересилка до контролеру, тимчасове чи постійне видалення).

API архітектури програмно-конфігурованих мереж часто називають *інтерфейсами північного та південного напрямків*, що визначають зв’язок між програмами, контролерами та мережевими системами. Північний інтерфейс визначається як з’єднання між контролером і програмами, тоді як інтерфейс південного напряму – це з’єднання між контролером і фізичним мережевим обладнанням. Оскільки SDN є накладенням віртуальної мережі, ці елементи не повинні бути фізично розташовані в одному місці.

Додатки виграють від розподілення між рівнями керування та даних. Вони можуть використовувати північний програмний інтерфейс для реалізації спеціальних операцій, таких як маршрутизація, моніторинг та конструювання трафіка.

Можна виділити три важливі відмінності між традиційною мережевою архітектурою та програмно-конфігурованими мережами [6]:

1. Контролер SDN має так званий «північний інтерфейс», що тримає зв’язок з додатками через виділені прикладні програмні інтерфейси. Це дозволяє розробникам програмних додатків безпосередньо конфігурувати мережу. У той самий час традиційні мережі працюють за допомогою визначених протоколів.
2. SDN є програмною мережею, що дозволяє користувачам керувати розподілом віртуальних ресурсів за допомогою відділеного рівня керування, а також дозволяє визначати мережеві шляхи та проактивно налаштовувати мережеві послуги. Традиційні мережі покладаються на фізичну інфраструктуру (таку, як комутатори та маршрутизатори) для встановлення з’єднань та належної роботи.
3. Програмно-конфігуровані мережі мають більше можливостей для зв’язку з пристроями по всій мережі на відміну від традиційних мереж. SDN дозволяє надавати ресурси з централізованого розташування та надає адміністраторам можливість та право на контролювання потоку трафіка з централізованого інтерфейсу користувача шляхом ретельного менеджменту. Такий підхід віртуалізує усю мережу та надає користувачам більше контролю над можливостями мережі. У традиційних рішеннях рівень керування знаходиться безпосередньо в комутаційному обладнанні, що не є зручним – адміністратори мережі не можуть легко отримати доступ до нього для виконання конструювання трафіка.

Отже, якщо порівнювати програмно-конфігуровані мережі із традиційними рішеннями, можна виділити наступні переваги [7]:

* Контролер має глобальне уявлення про топологію мережі, її стан і вимоги до додатків.
* Рівень даних може бути запрограмовано динамічно для покращення розподілу мережевих ресурсів.
* Зв’язок між контролером та комутаторами і роутерами не залежить від виробника пристрою (вендора).

Контролер має власну мережеву операційну систему, що забезпечує моніторинг, доступ та можливість керування усіма доступними ресурсами мережі. Мережева операційна система надає прикладний програмний інтерфейс, за допомогою якого здійснюється управління мережевими пристроями та постійне відстеження стану мережі. Контролер обов’язково централізований логічно, і необов’язково фізично [7]. В залежності від обраних метрик (наприклад, надійності або масштабування тощо) логічно централізований SDN-контролер може бути представлений декількома фізичними контролерами, які реалізують керування мережею сумісно.

Контролер об'єднує під рівнем керування всі мережеві пристрої – комутатори, маршрутизатори, NAT, брандмауери тощо – і дає можливість автоматизованого керування всією мережею з однієї точки доступу. Рівень керування відповідає за прийняття рішень про те, яким чином пакети повинні пересилатися одним чи декількома мережевими пристроями, та передає інструкції щодо цього на рівень інфраструктури (тобто відповідним мережевим пристроям). Таким чином, основними функціями рівня керування є моніторинг, конфігурація та підтримка мережевих пристроїв [8].

Потік в загальних рисах є послідовністю пакетів, що проходять через мережу і спільно використовують множину значень полів заголовка, таких як однакові IP-адреси джерела та призначення, один і той самий ідентифікатор VLAN і одну й ту саму MAC-адресу [9]. Кожен новий потік потребує дозволу від контролера, що перевіряє потік на відповідність до мережевої політики. Контролер SDN визначає, які потоки можуть проходити через інфраструктурний рівень. Тобто, перший пакет кожного нового потоку надсилається рівнем даних до контролера для отримання дозволу на продовження в мережі, а також для отримання його маршруту по усім встановленим пристроям пересилки.

Але у впровадженні SDN є і недоліки, пов’язані з тим, що технологія покладається на централізоване керування мережею. Серед основних можливих проблем у впровадженні програмно-конфігурованої мережі можна виділити наступні [11]:

1. *Затримка*. Кожен пристрій, що використовується в мережі, займає в ній місце. Швидкість взаємодії між пристроями та мережею залежить від кількості віртуалізованих ресурсів. Для забезпечення потреб у швидкості, є можливість збільшення ємкості ресурсів для віртуалізації, що може призвести до значної затримки.

2. *Технічне обслуговування.* Підтримка і обслуговування є дуже важливим аспектом мережі для виконання її основних функцій. SDN мережа є складно підтримуваною. Це робить практично неможливим керування реальними пристроями, особливо під час розширення мережі.

3. *Складність.* Досі немає жодних стандартизованих протоколів безпеки для SDN. Незважаючи на те, що існують деякі сторонні постачальники послуг, проблеми з безпекою наразі є актуальними. Тільки кваліфіковані адміністратори, які мають досвід роботи з системами SDN, можуть запобігти серйозним атакам.

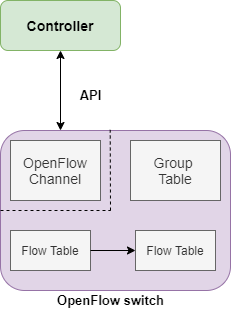
4. *Конфігурація.* Реконфігурація програмно-конфігурованої мережі є нетривіальною задачею, оскільки це пов’язано з великими витратами. Зокрема, реалізувати додаткові протоколи та змінити конфігурацію контролера SDN неможливо шляхом налаштування кожного з них, для цього необхідно переналаштовувати всю мережу.

### 1.1.2. *OpenFlow протокол*

OpenFlow – протокол, розроблений спеціально для програмно-конфігурованих мереж. Це відкритий протокол, що надає загальну специфікацію для реалізації мережевих пристроїв з підтримкою OpenFlow, а також для каналу зв'язку між пристроями для передачі даних і рівнем керування (між комутаторами та контролером. Основними критеріями, яким має відповідати протокол для SDN та які реалізовані в OpenFlow є наступні [11]:

1. Підтримка уніфікованої логічної архітектури, що має бути присутня в усіх мережевих пристроях, що керуються контролером. Навіть при різній реалізації логіки різними вендорами, контролер має бути доступний набір стандартних функцій.
2. Реалізація стандартного протоколу безпеки для зв’язку контролера та підлеглими пристроями.

OpenFlow надає специфікації логічного формату функцій комутації мережі, а також є протоколом зв’язку між контролером та пристроями. Архітектура протоколу складається з комутаційного обладнання, керованого контролером і визначається групою Open Networking Foundation (ONF) у специфікації комутаторів OpenFlow. Модель, за якою працюють комутатори за стандартами OpenFlow представлена на рис. 1.2 [15].



*Рис. 1.2. Стандарт взаємодії контролера та комутатора OpenFlow*

У протоколі OpenFlow використовується концепція *потоків* для опису трафіка. Таблиці потоків схожі на таблицю MAC/CAM традиційного комутатора, в якій зберігаються апаратні адреси хостів. У таблицях потоків зберігаються записи потоків або потоки, які надають інформацію комутатору SDN, що робити з пакетом, коли це стосується вхідного порту [16].

Усі комутатори, що підтримують даний протокол, мають таблицю потоків, що складається з трьох частин: поля відповідності, лічильники і інструкції відносно пакетів. Поля відповідності допомагають у визначенні, до якого потоку відноситься кожний пакет, що проходить через комутатор. Поля лічильників дозволяють реалізувати моніторинг і отримувати статистику мережі в реальному часі [17]. У випадку, коли існує більше ніж одна таблиця, вони реалізовуються у вигляді конвеєру. Інструкції змінюють операцію множини конвеєрної обробки. Лічильники оновлюються, коли пакети збігаються.

Із використанням протоколу комутаторів OpenFlow контролер може додавати, оновлювати та видаляти записи потоків у відповідних таблицях потоків (flow tables, FT) як реактивно, тобто у відповідь на відповідні пакети, так і проактивно [18].

*Реактивні* записи потоків створюються, коли контролер отримає динамічну інформацію, де знаходяться пристрої в топології і повинен оновити відповідні FT на цих пристроях, щоб здійснити наскрізне з’єднання. Наприклад, так як комутатори в чистому середовищі OpenFlow лежать на інфраструктурному рівні і є просто пристроями для переадресації трафіка, уся раціональна логіка спочатку має бути продиктована та запрограмована контролером. Отже, якщо певному хосту А потрібно створити з’єднання з відповідним зкомутованим пристроєм В, на контролер буде надіслано відповідне повідомлення, щоб дізнатися маршрут до цільового хоста. Контролер має розглянути таблиці MAC-адрес хостів комутаторів і спосіб їх підключення, програмуючи логіку в таблицях потоків кожного комутатора. Це вхід реактивного потоку.

*Проактивні* записи потоку програмуються ще до прибуття трафіку. Якщо вже є інформація, що два пристрої мають або не мають з’єднання, контролер може запрограмувати відповідні записи потоків на кінцевих точках з’єднання OpenFlow заздалегідь.

Лічильники є важливою складовою таблиці потоків. Вони доступні для кожної таблиці потоків, запису потоку, порту, черги, групи тощо. Специфікація OpenFlow 1.4 визначає набір лічильників, перелік наведено в додатку А [15].

З точки зору окремого комутатору, потік – це послідовність пакетів, котра відповідає певному запису в таблиці потоків.

Протокол OpenFlow підтримує три типи повідомлень [15]:

1. *Контролер-комутатор.* Такі повідомлення ініціюються контролером та необхідні для безпосереднього керування та перевірки стану комутаторів. Вони дозволяють контролеру керувати логічним станом комутатора, у тому числі його конфігурацією і контролювати деталі запису потоку і групової таблиці.
2. *Асиметричні.* Повідомлення надсилаються без запиту від контролера. Комутатори надсилають такі повідомлення для позначення прибуття пакету, зміни стану комутатора або у випадку помилки.
3. *Симетричні.* Повідомлення надсилаються без запиту в будь-якому випадку, тобто ініціюються або контролером, або комутатором. До них належать: hello-повідомлення (при встановленні першого з’єднання між контролером і комутатором), echo-запити (для вимірювання затримки чи пропускної здатності каналу) та vendor-повідомлення.

Основною перевагою у використанні OpenFlow протоколу є можливість безпечного та зручного переходу між мережевим обладнанням різних вендорів, так як протокол розроблявся з метою забезпечення сумісності різних апаратних рішень, а також суттєва підтримка та розробка нових специфікацій. Як альтернативу до OpenFlow протоколу в SDN можна виділити використання REST API, що є значно менш вживаним з причини необхідності додаткової розробки взаємодії між контролерами та комутаторами з різними архітектурами.

## **Концептуальний огляд методів Big Data**

### *Теоретичні аспекти Big Data*

Термін Big Data є загальним для визначення нетрадиційних стратегій та технологій, що необхідні для збору, організації, обробки та аналізу інформації з великих обсягів даних. Хоча проблема роботи з даними, що перевищують обчислювальну складність або місткість однієї обчислювальної машини не є новою, за останні роки поширеність, масштаби та цінність такого типу обчислень значно розширились [19].

Точного визначення поняття «великих даних» не надає жоден науковець, адже проекти, постачальники та бізнес-фахівці використовують його по-різному. Але загальний концепт Big Data полягає у методах, що застосовуються до різних типів великих і неструктурованих наборів даних, які не можуть бути опрацьовані за допомогою звичайних систем обробки даних в необхідні терміни. В такому контексті «великий набір даних» означає такий обсяг, який є надто великим для обґрунтованої обробки або зберігання за допомогою традиційних інструментів на одному пристрої. Це також означає, що загальний масштаб великих наборів даних може змінюватись залежно від використання [20] .

Основні вимоги для роботи з великими даними такі ж, які існують для роботами з даними будь-якого розміру. Однак масштаби, швидкість прийому та обробки, а також характеристика самих даних, з якими необхідно мати справу на кожному етапі процесу, створюють нові проблеми при розробці рішень. Мета більшості систем великих даних полягає в тому, щоб виявити інформацію та зв’язки з великих обсягів різнорідних даних, які були б неможливими за допомогою звичайних методів.

Розмір даних є першою характеристика, що використовується для розуміння Big Data, але не єдина. У 2001 році Дагом Лейні було запропоновано три виміри для визначення великих даних: об’єм, різноманітність і швидкість (англ. Three V’s: Volume, Variety, Velocity) [21].

*Об’єм* відноситься до великої кількості даних, що створюються додатками. Організації збирають дані з різних джерел, включаючи бізнес-транзакції, Інтернет речей (Internet of Things, IoT), промислове обладнання, відео, соціальні мережі тощо. У минулому зберігання таких даних було б проблемою, але створення доступніших сховищ даних, таких як «озера даних» (data lakes), полегшило процеси.

*Різноманітність* визначає наявність різних форматів даних у певному наборі. Різні технології забезпечують структуровані дані (електронні таблиці, реляційні бази даних), неструктуровані дані (текст, зображення, аудіо, відео) і напівструктуровані (наприклад, XML). Поняття швидкості відноситься до швидкості генерації та аналізу даних.

Поняття *швидкості* відноситься безпосередньо до швидкості генерації та аналізу даних. З розвитком IoT швидкість генерації та необхідної обробки даних зростає, а наявність деяких датчиків, такий як RFID-мітки та розумні лічильники викликають потребу працювати з цими потоками в режимі реального часу.

Згодом в доповнення до вищеназваних характеристик пропонують виділяти також інші величини, пов’язані з великими даними: достовірність, мінливість і цінність (Veracity, Variability, Value). Різноманітність джерел і складність обробки даних можуть призвести до проблем оцінки якості даних, а тому до проблеми оцінки якості результату обробки та аналізу даних. *Варіативність* також призводить до великої різниці в якості. Для виявлення, обробки або фільтрації низькоякісних даних можуть знадобитися додаткові ресурси. Найголовнішим завданням великих даних є надання *цінності.* Іноді наявні системи та процеси є настільки складними, що отримання фактичної цінності від обробки та аналізу даних також може бути ускладнено [22].

Основна мета використання методів Big Data полягає у трьох наступних пунктах:

1. Зберігання, обробка та аналіз великих, порівняно зі стандартними обсягами, даних
2. Можливість здійснення швидкого процесу обробки в умовах динамічного росту обсягів даних.
3. Можливість паралельної обробки даних різної структури та в різних цілях.

Отже, основними принципами застосування методів Big Data на відміну від традиційних рішень є [23]:

*Горизонтальна масштабованість.* Базовий принцип обробки великих обсягів даних полягає в горизонтальному масштабуванні. При динамічному зростанні кількості даних необхідно оперативно збільшувати кількість обчислювальних вузлів, які мають працювати над задачею опрацювання цих даних, при цьому важливий той факт, що обробка має виконуватися без зменшення продуктивності системи.

*Відмовостійкість.* Виходячи з принципу горизонтальної масштабованості, не менш важливою характеристикою системи є відмовостійкість. Обчислювальний кластер може містити в собі досить велику кількість вузлів, або нод, і згідно з принципом горизонтальної масштабованості, їх кількість може динамічно збільшуватись. Така система є потенційно більш вразливою з точки зору надійності так як зростає і ймовірність виходу машин з ладу. Технології для роботи з великими даними можуть враховувати можливість таких ситуацій і вживати превентивні засоби.

*Локальність даних.* Зазвичай для забезпечення відмовостійкості застосовуються розподілені системи збереження даних, що означає розподілення самих даних по великій кількості вузлів. Залежно від топології обчислювального кластера, витрати на пересилку даних з однієї машини на іншу можуть бути невиправдано великими. З цього виходить принцип локальності даних, який передбачає механізми обробки даних на тій самій машині, на якій вони зберігаються.

Різниця між принципами для роботи з невеликими, традиційними, однорідними даними та Big Data спонукає до використання нових технологічних підходів.

*Пакетна обробка* і *обробка в реальному часі* – два різні типи обробки великих даних. Вибір відповідної опції залежить в першу чергу від типу та джерел даних, регламентованого часту на обробку та необхідність оперативного використання результатів обробки та аналізу даних. Відповідно, пакетна обробка забезпечує більшу стабільність та надійність при обробці якісно структурованих даних, але потребує більше часу та обчислювальних ресурсів у момент часу. Використання обробки в реальному часі має на увазі безперервний ввід, обробку та вивід даних та має бути застосована у випадках, коли необхідно отримати актуальну інформацію одразу [24].

### ‘*Пакетна обробка даних та MapReduce*

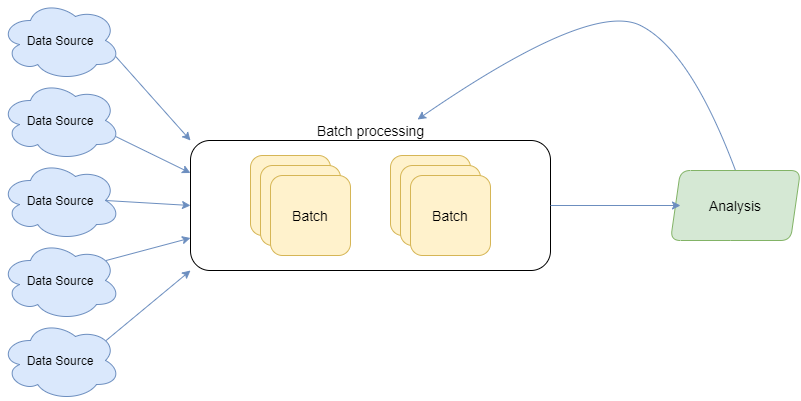
Пакетна обробка зазвичай використовується при роботі з надзвичайно великими обсягами даних або коли джерелами даних є застарілі системи, що не здатні продукувати дані в потоковому режимі.

Такий тип обробки доцільно використовувати у випадку, коли немає необхідності в отриманні результатів аналітики в реальному часі або коли важливіше обробляти великі обсяги інформації, ніж отримувати швидкі результати аналітики. Схематичне зображення процесу пакетної обробки даних зображено на рис. 1.3. [25].

MapReduce – модель розподілених обчислень, що використовується при обробці надзвичайно великих обсягів даних за допомогою обчислювальних кластерів, а також це фреймворк для виконання розподілених обчислень на вузлах кластера.

Головна ідея цієї моделі полягає в розділені інформаційного масиву на частини, паралельної обробки кожної частини на окремому вузлі та наступному об’єднанні усіх результатів. Виходячи з назви, є два основні етапи в даній моделі обчислень:

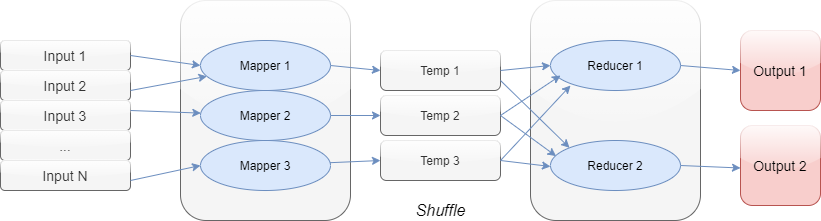
* *Map.* На першому етапі обробники працюють з блоками даних і приводять їх до вигляду пар «ключ-значення». Також цей етап передбачає можливість створення проміжних пар «ключ-значення» за потреби.
* *Reduce.* На другому етапі інші обробники збирають створені пари «ключ-значення» і агрегують їх або групують за ключем, продукуючи кінцевий результат.



*Рис. 1.3. Процес пакетної обробки*

Модель MapReduce представлена на рис. 1.4. [25].

MapReduce здатна вирішувати значну кількість реальних задач. Складна задача може включати в себе декілька операцій MapReduce і може потребувати для виконання багато часу та ресурсів. Для вирішення цієї проблеми доступні два варіанти реалізації MapReduce: Hadoop і Spark. Ці інструменти відрізняються один від одного не тільки своїми операторами потоків даних, а і підходом до проміжних файлів. Hadoop зберігає проміжні файли на диску, тоді як Spark розміщує їх в пам’яті. Через цю відмінність пропонується використовувати Hadoop для пакетної обробки і Spark для обробки в реальному часі.



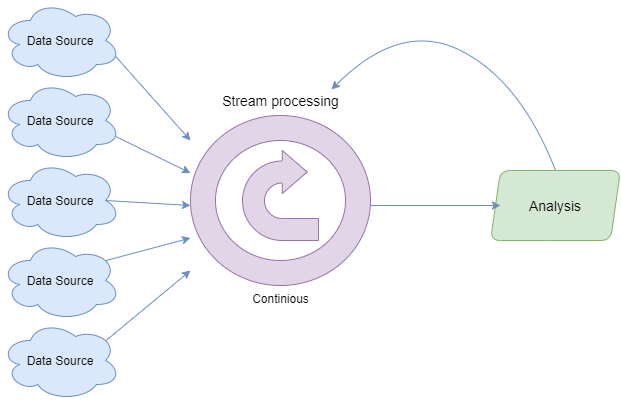
*Рис.1.4. Модель розподілених обчислень MapReduce*

### *Обробка даних в реальному часі та лямбда-архітектура*

Потокова обробка є ключовим елементом конвеєру даних, якщо є необхідність отримання аналітики в реальному часі. За цією моделлю обробка даних виконується щойно вони надходять на рівень зберігання, що часто також має бути близько за часом до моменту їх продукування, але це не обов’язково. Обробка потоків займає досить мало часу, що для кінцевого користувача або системи це можна описати як обробка в реальному часі. Такі операції зазвичай не мають статусу або можуть зберігати лише тимчасовий стан, тому зазвичай передбачають відносно прості трансформації та обчислення.

Структура потокової обробки спрощує необхідність у паралельному обладнанні та програмному забезпеченні, обмежуючи продуктивність паралельних обчислень. Конвеєрні функції ядра застосовуються до кожного елемента в потоці даних, використовуючи повторне використання вбудованої пам'яті, щоб мінімізувати втрати пропускної здатності. Інструменти та технології обробки потоків доступні в різних форматах: розподілені системи обміну повідомленнями для публікації та підписки, такі як Kafka, розподілені системи обчислення в реальному часі, такі як Storm або Spark, і механізми потокового передачі даних, такі як Flink.

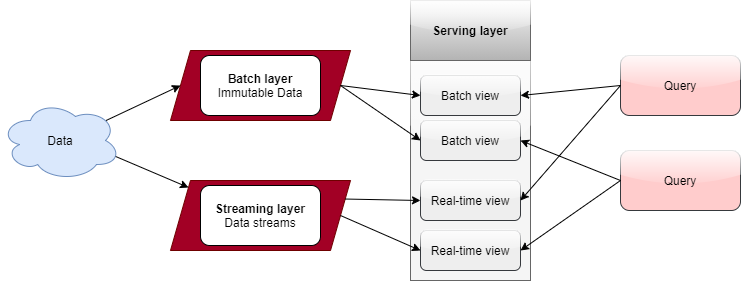
Структура процесу потокової обробки даних зображена на рис. 1.5. [26].



*Рис. 1.5. Модель потокової обробки даних*

У той час як пакетна обробка включає пакети даних, які вже зберігалися протягом певного періоду часу та виконуються в регулярний розклад або за потребою, потокова обробка дає змогу користувачеві передавати дані в інструменти аналітики, одразу як тільки вони згенеруються., що полегшує обробку даних у реальному часі та миттєві результати аналітики. Серед переваг можна виділити оперативне виявлення умов або аномалій за дуже короткий проміжок часу, що корисно для таких завдань, як антифрод-системи або конструювання трафіка. Пакетна обробка більш корисна в ситуаціях, коли важливіше обробляти великі обсяги даних, ніж збирати результати аналітики в реальному часі [27].

Пакетна та потокова моделі роботи з великими даними мають свої переваги та недоліки. Для виконання задачі конструювання трафіку в програмно-конфігурованих мережах важливо швидке отримання результатів для можливості оперативного реагування на позаштатні ситуації або ж для можливості збільшення надійності та якості мережі. Це можна забезпечити завдяки використанню потокової моделі. Але шлях потокової обробки, по-перше, може бути вразливим до переривань у роботі системи, і, по-друге, не забезпечує зберігання історичної інформації, що не менш важливо у задачі роботи з трафіком і що можна отримати завдяки пакетному рівню.

Лямбда-архітектура (англ. Lambda Architecture) – ще одна модель обчислень для роботи з великими даними. Головна мети лямбда-архітектури полягає в створенні системи, що може обробляти та створювати представлення даних в реальному часі, а також забезпечує зберігання та обробку історичної інформації. Система, побудована на основі лямбда-архітектури, може опрацьовувати як потокові, так і пакетні дані. Для цього обробка даних фактично розділяється на два шляхи (рис. 1.6.) [28]:

*Рис.1.6. Модель лямбда-архітектури*

*«Холодний» шлях* – пакетний рівень, де всі вхідні дані зберігаються в неопрацьованому вигляді і обробляються в пакетному режимі. *«Гарячий» шлях* – швидкісний рівень, або рівень прискорення, де дані аналізуються в режимі реального часу. Цей рівень забезпечує мінімальну затримку обробки з деякою можливою втратою точності. Також в цій архітектурі виділяють *сервісний рівень,* який індексує пакетне представлення – результати роботи пакетного рівня для ефективного виконання запитів.

Пакетний рівень віддає перевагу алгоритмам повторного обчислення над додатковими алгоритмами. Проблема інкрементних алгоритмів полягає у неможливості вирішити проблеми, пов’язані з людськими помилками. Повторний обчислювальний характер пакетного рівня створює прості пакетні представлення, оскільки ця складність вирішується під час попереднього обчислення. Крім того, відповідальність пакетного рівня полягає в історичній обробці даних з високою точністю [29].

Основна проблема пакетного шару - висока затримка. Пакетні завдання повинні виконуватися на всьому основному наборі даних і забирають багато часу. Наприклад, може бути певний розклад робіт, які виконуються через кожні дві години. Ці завдання можуть обробляти дані, які можуть бути відносно старими, оскільки не можуть встигати за надходженням нових потокових даних. Це серйозне обмеження для обробки даних у режимі реального часу. Для його подолання введено архітектуру швидкісного рівня.

Загалом, обробка даних у режимі реального часу здійснюється завдяки наявності швидкісного рівня. Потоки даних обробляються в режимі реального часу без урахування повноти та виправлень чи виконання переобчислень. Швидкісний рівень забезпечує найновіші результати запитів та компенсує високу затримку пакетного шару. Призначення цього шару - заповнення в проміжку, викликаному трудомістким пакетним шаром. Для того, щоб створювати представлення останніх даних у режимі реального часу, цей рівень нехтує пропускною здатністю та істотно зменшує затримку. Перегляди в реальному часі генеруються відразу після отримання даних, але вони можуть бути не такі повні або точні, як пакетний рівень.

Ідея цього дизайну полягає в тому, що точні результати пакетного шару замінюють подання в режимі реального часу, як тільки проходять обчислення. Такий розподіл обов’язків для різних рівнів і визначає ефективність лямбда-архітектури.

## **1.3. Огляд способів конструювання трафіка в програмно-конфігурованих мережах**

### 1.3.1. *Види та задачі конструювання трафіка в SDN*

Конструювання трафіка (англ. Traffic Engineering, TE) є важливим мережевим елементом, що включає в себе вимірювання та управління мережевим трафіком таким чином, щоб спрямовувати потоки інформації для покращення використання мережевих ресурсів та кращого задоволення вимог щодо якості обслуговування мережі (англ. Quality of Service, QoS).

У порівнянні з традиційними мережами, SDN має багато переваг для підтримки TE завдяки відмінним характеристикам, таким як ізоляція контролю та пересилки, глобальне централізоване керування та програмованість поведінки мережі [30].

Характеристики SDN корисні для вирішення поточних проблем конструювання мережевого трафіку можна узагальнити до наступних пунктів [31]:

1. *Вимірювання трафіку*. SDN завдяки наявності прикладного рівня надає можливість гнучко розгортати масштабовані глобальні задачі вимірювання в SDN, що можуть збирати інформацію про стан мережі в режимі реального часу, а також відстежувати й аналізувати трафік централізовано в контролері.

2. *Планування та керування трафіком*. Вимоги до застосування трафіку можна розглядати як глобальні для забезпечення можливості гнучкого, детального планування трафіку.

3. *Гнучкість роботи комутаторів OpenFlow.* Комутатор OpenFlow має кілька конвеєрів обробки таблиць потоків, що робить управління потоками більш гнучким та ефективним.

Хоча SDN забезпечує масштабну підтримку конструювання трафіку, наразі існує велика кількість відкритих питань, які необхідно вирішити. Досі ведуться дослідження, які мали б показати, що традиційні концепції TE можуть бути повністю сумісними з програмно-конфігурованими мережами. Інша проблема полягає в тому, що традиційні IP-мережі та SDN будуть співіснувати протягом тривалого часу і також досліджується питання щодо технологій конструювання трафіка у гібридних IP/SDN мережах [31]. Тому дослідження технологій TE на основі SDN мають велике значення для подальшого розповсюдження та застосування програмно-конфігурованих мереж.

Основна мета конструювання трафіка — задовольнити вимоги до продуктивності, економно використовуючи ресурси мережі. Показники, які використовуються для вимірювання ефективності трафіку, зазвичай включають затримку, втрату пакетів і пропускну здатність. Інші фактори, які враховуються - це надійність і відмовостійкість [32].

У багатьох випадках рішення щодо маршрутизації статичні. У разі адаптивної маршрутизації оцінка маршруту може бути змінена через зміну навантаження на маршрут. Щоб використовувати механізм адаптивної маршрутизації, між джерелом і призначенням має існувати кілька шляхів, і трафік має бути перенаправлений зі старого шляху на новий. Реалізація такого підходу вимагає знання всіх мережевих шляхів і загального навантаження мережі зі зібраними мережевими показниками, тобто швидкість втрати пакетів. У разі існування кількох шляхів є дві можливості [32]:

* Найкращий за обраною метрикою маршрут використовується для пересилання трафіку, тоді як решта маршрутів є резервними і використовуються лише у разі збою мережі. Більшість IP-мереж сьогодні використовують цей підхід.
* Трафік розподіляється між існуючими маршрутами. Існує кілька способів такого розподілу, наприклад за пакетами або потоками. Перший підхід забезпечує більш високу деталізацію розподілу трафіку ціною відсутності впорядкованості пакетів, тоді як другий варіант є простішим, але менш гнучким. Однак розподіл за пакетами не є сумісним з TCP-з'єднанням.

Надійність мережі заснована на можливості швидкого відновлення у випадку збоїв в мережевих компонентах (контролерах, комутаторах, каналах зв’язку) [33]. Швидке відновлення після позаштатної ситуації може бути складним і часомістким процесом через те, що контролер має розрахувати нові маршрути і передати їх усім працюючим комутаторам. Окрім цього, необхідно враховувати обмежені ресурси таблиці потоків на комутаторах, оскільки нові записи будуть додані тільки в таблиці потоків працюючих комутаторів.

Мережеві методи конструювання трафіку в SDN залежать від архітектури мережі, характеру трафіка та наявних мережевих ресурсів. Загалом виділяють наступні показники, в залежності від яких відрізняється методологія TE [33]:

*Зміни трафіка або стану мережі, обумовлені часом.* При змінах характеру трафіка з часом TE відбувається з урахуванням періодичності. Така концепція загалом не має на меті адаптацію до тимчасових або раптових змін у стані мережі. Основний пункт, на якому базуються такі способи, є прогнозування трафіка на певний період часу на основі принципів самоподібності. Для цього необхідна якісна історична інформація про навантаження мережі на різних вузлах в різні моменти часу, що дозволяє застосувати систему прогнозування трафіка на визначений період.

З іншого боку, конструювання трафіку може базуватися на зміну стану мережі з часом. Такий спосіб дозволяє оцінити поточний стан мережі у прогнозуванні трафіку, що робить можливим врахувати такі можливі ситуації, як збої або проблеми з обладнанням. Поточний стан мережі може бути описаний рядом метрик, наприклад, пропускна здатність, затримка тощо. Конструювання трафіка в залежності від стану мережі сильно залежить від своєчасного та точного збору цих даних, оскільки стан мережі постійно змінюється, тому такий підхід дозволяє підвищити відмовостійкість мережі.

*Централізовані та децентралізовані методи*. Централізовані методи конструювання трафіку можна охарактеризувати наявністю певного центрального контролера, що відповідає за конструювання трафіку для кожного пристрою інфраструктурного рівня. Таким чином центральний контролер має зібрати дані про стан усіх маршрутизаторів і здійснити процес керування. Децентралізовані способи конструювання трафіка притаманні традиційним IP-мережам. Для отримання інформації про стан мережі звичайні маршрутизатори покладаються або на сканування, або періодичний обмін інформацією з іншими маршрутизаторами.

### *1.3.2. Огляд існуючих рішень для конструювання трафіка*

Google представила програмно-визначену архітектуру глобальної мережі під назвою B4 для підключення своїх центрів обробки даних по всьому світу [34]. B4 розроблено для вирішення проблем глобальної мережі (WAN), таких як надійність, відмовостійкість і продуктивність. Однією з запроваджених функцій зі змін шаблонів трафіка механізм конструювання трафіку, що дозволяє додаткам гнучко адаптуватись у відповідь на зміни стану мережі. Ця архітектура використовує маршрутизацію та інженерію трафіку як окремі послуги. TE реалізовано як надбудова над протоколами маршрутизації, що дає змогу мережі використовувати резервні стратегії.

Hedera [35] представлена ​​для ефективного використання пропускної здатності в центрі обробки даних. Hedera виявляє так звані “elephant flow” на крайніх комутаторах. Така реалізація має на увазі періодичне витягування інформації, так як необхідно збирати статистичні дані кожні п’ять секунд для виявлення великих потоків. У цій системі спочатку комутатори надсилають новий потік, використовуючи свої правила зіставлення потоків за замовчуванням на одному зі своїх шляхів рівної ваги, доки розмір потоку не зросте і не буде відповідати пороговому значенню, а тоді потік позначається як «слоновий». У цей момент центральний планувальник Hedera використовує свій глобальний погляд на мережу та розраховує кращий шлях для потоку та маршрутизації трафіку. Щоб ефективно використовувати пропускну здатність, планувальник обчислює шлях таким чином, щоб він був неконфліктним, і він може врахувати потік. Цей метод може покращити використання пропускної здатності, але оскільки він використовує періодичне витягування, він може спричинити високе використання ресурсів та накладні витрати.

Основною метою проектування DevoFlow [36] є покращення масштабованості та продуктивності мережі за рахунок максимального збереження потоків на рівні даних без втрати централізованого уявлення про мережу. Це зменшує взаємодію між площиною керування та площиною даних. DevoFlow використовує агресивне використання символів підстановки, щоб зменшити взаємодію контролера та перемикачів. Таким чином, комутатори приймають рішення про маршрутизацію локально, тоді як контролер керує загальним станом мережі та бере участь у маршрутизації великих потоків (elephant flows). Ця система використовує такі методи, як вибірка пакетів, для збору статистики комутаторів і виявлення великих потоків. Спочатку DevoFlow пересилає трафік за допомогою правила багатошляхового підстановки. Коли буде виявлено великий потік, контролер обчислить шлях, що є найменш перевантаженим, і виконає перенаправлення трафіка на цей шлях.

У роботі [37] представлений механізм конструювання трафіку для мережі центрів обробки даних під назвою MicroTE, який використовує виявлення elephant flows кінцевих хостів. У системі використовується короткострокове передбачення та швидка адаптація до змін структури маршрутів. Щоб ефективно обробляти мережеве навантаження, він використовує переваги кількох шляхів у мережі та координує планування трафіку за допомогою глобального перегляду трафіку по доступних мережевих шляхах.

У роботі [38] була спроба дослідити спосіб, як інтегрувати інформованість про програми в мережі на основі SDN і як класифікувати трафік з високою точністю. Представлено фреймворк під назвою Atlas, який здатний класифікувати трафік в мережі та застосовувати політику вищого рівня. Представлений фреймворк використовує інструмент ML під назвою C5.0 для класифікації потоків на основі типів додатків, що показує до 94% точності. Статистика перемикача OpenFlow розширюється, щоб зберігати перші n розмірів пакетів кожного потоку та повідомляти про це контролеру. Контролер збирає такі функції потоку разом з інформацією, надісланою агентами для навчання інструменту ML.

Значна кількість рішень було запропоновано для задачі моніторингу та аналізу трафіка. OpenNetMon [39] розроблений як система моніторингу, що здатна визначити факт, чи задовольняються потреби скрізного QoS. З іншого боку, така система також може бути використана і для того, щоб додатки TE могли обчислювати відповідні шляхи.. Система реалізована як додатковий модуль контролера SDN і може здійснювати опитування крайніх комутаторів для збору статистики потоків з адаптивною швидкістю, щоб визначити основні метрики потоків, такі як пропускна здатність та затримка.

Існують різні системи аналізу трафіка, в яких зазвичай виконується оцінка і прогнозування матриці трафіка (TM) на основі процедур моделювання та оптимізації. Для моделювання використовуються математичні моделі оцінки TM. Так, iSTAMP [40] має на увазі спосіб розбиття таблиці потоків на дві частини: перша для загального аналізу і друга для моніторингу elephant flows. При використанні обох частин відбувається оцінка згенерованої матриці трафіка.

Дослідження [41] використовує підхід до моніторингу та аналізу трафіка з використанням методів Big Data. У дослідженні використовується статистична інформація таблиць потоків усіх доступних комутаторів для генерації матриці трафіку. Застосовується підхід MapReduce для виконання, збору, агрегації та акумуляції статистичних даних для визначення характеристик трафіку та подальшого процесу конструювання трафіка. Такий підхід можливий завдяки специфіці роботи OpenFlow для опитування портів комутаторів.

Більшість зазначений способів конструювання трафіка в SDN базуються на різних елементах таблиці потоків, а також приймають рішення, засновуючись на поточному стані мережі. Окрім цього, подібні методи є малоефективними в умовах динамічної зміни стану високонавантаженої мережі. Робота, що включає в себе елементи роботи з великими даними, частково вирішує цю проблему, але такий підхід має свої недоліки, наприклад, неможливість оперативно вплинути на зміну таблиць потоків через відносно довгий час обробки інформації.

Для уникнення цієї проблеми, пропонується розробити спосіб конструювання трафіка, що матиме можливість миттєвого реагування на зміну стана мережі та дозволить застосування підходу до TE у високонавантажених мережах. Такий спосіб також має реагувати не тільки на миттєві зміни характеру трафіка, але й враховувати історичну статистичну інформацію про мережу. Виходячи з проведеного аналізу, завданнями розробки способу конструювання трафіка в SDN є використання сучасних методів роботи з великими даними, а саме лямбда-архітектури, а також моделювання обробки та збереження статистичної мережевої інформації з метою подальшого аналізу.

## **Висновки до розділу 1**

У даному розділі було розглянуто структуру програмно-конфігурованих мереж, зазначено принципові відмінності порівняно з традиційними IP-мережами, а також розглянуто основні принципи роботи найбільш вживаного протоколу в SDN OpenFlow. Виконано концептуальний огляд методології Big Data, розглянуто основні підходи пакетної та потокової обробки даних. Виявлено основні проблеми існуючих підходів до конструювання трафіка в SDN та наведено огляд існуючих рішень для систем моніторингу та аналізу трафіка в програмно-конфігурованих мережах.

Зокрема, було виконано дослідження та аналіз основні принципи роботи SDN та її інфраструктурних елементів, а також прикладних програмних протоколів взаємодії. Зазначено головні відмінності програмно-конфігурованих мереж від традиційних та виявлено ситуацій, в яких використання SDN є пріорітетним.

Розглянуто теоретичні та практичні аспекти роботи з великими даними, досліджено способи пакетної обробки даних та роботи з даними в режимі реального часу. Виконано огляд основних архітектурних підходів до пакетної то потокової обробки даних, таких як MapReduce, каппа-архітектура та лямбда-архітектура.

Також у розділі досліджено перелік існуючих рішень для виконання задачі конструювання трафіка в SDN. Спираючись на дані аналізу, сформульовано основні задачі розробки способу конструювання трафіка в високонавантажених SDN, що дозволятиме швидко реагувати на зміни стану мережі в режимі реального часу, а також забезпечуватиме збереження та обробку історичних статистичних даних. Запропонований спосіб забезпечуватиме отримання актуальної інформації про мережевий трафіка, а також матиме механізми реалізації надійності та відмовостійкості.

# **РОЗДІЛ 2**

# **РОЗРОБКА СПОСОБУ КОНСТРУЮВАННЯ ТРАФІКА НА ОСНОВІ ЛЯМБДА-АРХІТЕКТУРИ**

У даній роботі пропонується спосіб конструювання трафіка на основі стану мережі за допомогою реалізації лямбда-архітектури. Для забезпечення отримання інформації про мережу планується використовувати можливості OpenFlow зі збору даних з лічильників портів. Отримана інформація має одразу передаватися в систему роботи з повідомленнями для подальшого процесу збору, агрегації та акумуляції даних. Після цього вже оброблені актуальні дані використовуються для реалізації конструювання трафіка.

Метою є виявлення переваг наявності пакетного та потокового шляхів обробки даних для, по-перше, отримання актуальної інформації про стан мережі та можливості швидкого реагування на зміни та, по-друге, збереження усієї статистичної інформації для подальшої обробки та використання в цілях аналізу історичних даних.

Для спрощення вважатимемо, що статистичні дані з портів комутаторів отримуються з визначеною адміністраторами мережі періодичністю, а також вважатимемо, що контролер завжди володіє актуальною інформацією про мережу. Такі спрощення є можливими виходячи з того, що основною метою роботи є саме розробка способу роботи з отриманими даними та налаштування відповідних процесів.

## **Спосіб конструювання трафіка на основі мережевих даних**

Головна задача отримання опрацьованих інвентарних даних та даних з лічильників полягає у формуванні *матриці трафіка*. Статистична інформація з портів є агрегованим представленням усього трафіка, що проходить по різних маршрутах. Для виконання завдання маршрутизації, необхідно сформувати матрицю трафіка на основі статистичних даних таким чином, щоб створити чітку відповідність між певним маршрутом та відповідними значеннями лічильників.

Умовно весь процес визначення матриці трафіка можна розділити на такі етапи:

1. Формування множини маршрутів, базуючись на інвентарних даних.
2. Встановлення відповідності між шляхом та статистичними мережевими даними.

До інвентарних даних належать:

* Множина , що містить усі доступні вузли мережі.
* Множина , що містить хости мережі.
* Множина містить усі комутатори в мережі. Кожен комутатор має певну кількість портів, що позначаються .
* Множина визначає з’єднання між хостами і комутаторами.
* Множина визначає з’єднання комутатор-комутатор.
* Множина визначає всі наявні з’єднання.

Мережа визначається графом Для отримання матриці трафіка необхідно визначити також додаткові сутності, що допоможуть для ініціалізації можливих маршрутів мережі:

* Множина визначає канали між хостами і портами комутаторів.
* Множина визначає канали між двома комутаторами (їх портами).
* .
* визначає комбінацію усіх пар хостів (або пар origin-destination, OD) в мережі.

Для кожного наявного маршруту визначимо таке *P,* що включатиме в себе усі канали, що належать до певного маршруту:. Виходячи з цього, визначимо множину маршрутів: }.

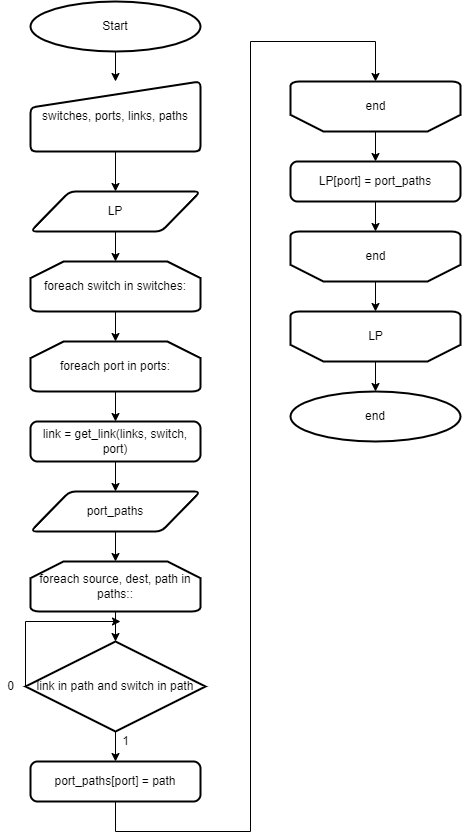
Множина маршрутів містить інформацію, а точніше список пар OD, що проходять через конкретний порт , тобто для який порт є частиною маршруту. Відповідно, для отримання цієї множини необхідно спершу отримати множину всіх пар OD. Далі для кожної пари має бути знайдений P, що по суті являє собою маршрут. Кожен такий шлях між хостами складається з трьох типів з’єднань: стартовий хост – стартовий комутатор, міжкомутаторне з’єднання (міжлінк) та кінцевий комутатор – кінцевий хост. Для кожного типу пари виконується пошук по множині L. Для з’єднання першого та третього типу повертаємо порт , а для з’єднань другого типу повертаємо пару портів На цьому етапі вже маємо множину портів, через які пролягає маршрут P. Після цього для кожного порту з множини формуємо пару *(port, (O, D))* і додаємо її до відповідного порту множини маршрутів.

Отже, формування LP-множини можливе за виконання наступних умов:

1. Маємо множини *H, S,* і граф *G.*
2. Побудована множина *HC* використовуючи функцію Величина HC задається комбінацією усіх елементів *H(n)* по два елементи.
3. Отримана множина *L.*

Блок-схема алгоритму формування LP-множини представлена на рисунку 2.1.

Для встановлення відповідності між мережевою інформацією та множиною маршрутів дані мають бути опрацьовані декількома шарами трансформацій. Інформація у повідомлені розподілена за часом, за точкою топології мережі та за інтерфейсом. Для виконання завдання створення відповідності маршрутів та даних з лічильників, спочатку необхідно здійснити декілька суттєвих маніпуляцій, що згодом також можуть бути використані у якості історичної інформації для додаткового аналізу.



*Рис. 2.1. Алгоритм формування LP-множини*

Окрім збору аналітичних даних з лічильників комутаторів OpenFlow, для задачі конструювання трафіку також важливо оперувати визначеними метриками. Тому для кожної множини повідомлення пропонується виконувати не тільки операцію агрегації, але й обчислення важливих статистичних даних, наприклад пропускної здатності. Перелік полів, що мають бути обчислені додатково, наведений у таблиці 2.1.

*Обчислення статистики для інтерфейсу.* Перша дія, яку необхідно виконати – це агрегація даних за інтерфейсом. Отримані повідомлення розпаковуються та формуються пари ключ-значення, ключем яких виступає назва поля, а значенням – мережеві дані, отримані та надіслані контролером. Для отримання статистичної інформації для інтерфейсу необхідно виконати агрегацію даних за ключами порту, комутатора та часової мітки для певної множини повідомлень за обраний час та просумувати їх. Окрім цього, на даному етапі виконується обчислення додаткової інформації по пропускній здатності.

*Таблиця 2.1.*

*Розрахована пропускна здатність для порту*

|  |  |
| --- | --- |
| **Розрахована пропускна здатність** | **Визначення**  (2.1)  (2.2)  (2.3)  (2.5)  (2.4) |
| Packet Received Throughput (PRT) |  |
| Packet Transmitted Throughput (PTT) |  |
| Packet Throughput (PTr) |  |
| Bytes Received Throughput (BRT) |  |
| Bytes Transmitted Throughput (BTT)  (2.6) |  |
| Bytes Throughput (BTr) |  |

*Обчислення статистики для комутатора.* Отримана на попередньому кроці статистична інформація може бути масштабована на комутатори. Тепер, коли відомо аналітичні дані для кожного порту кожного комутатора, необхідно виконати агрегацію за ключем комутатора та часової мітки. Додатково обчислювати пропускну здатність на цьому етапі немає необхідності, так як агрегація та сума пропускної здатності по портах має розповсюджуватись і на її значення для комутаторів.

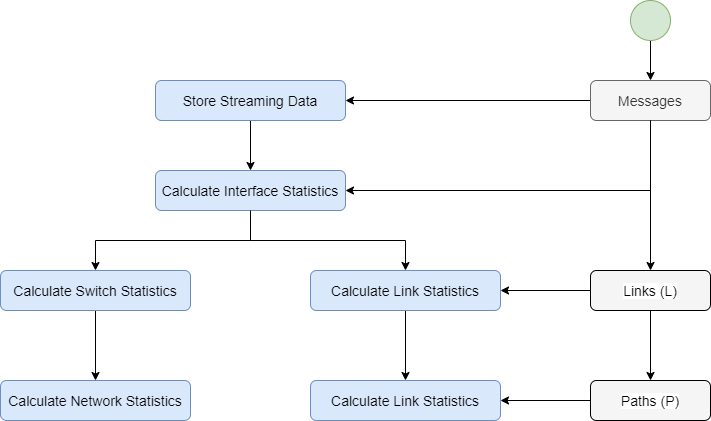
*Обчислення мережевої статистики.* Виходячи з статистики комутаторів, досить легко отримати статистику і для усієї мережі, виконавши операцію агрегації за часовою міткою та отримавши детальну інформацію про стан мережі в конкретне часове значення, що також є досить корисним для подальшої задачі аналізу трафіка.

*Обчислення статистики по каналах.* Кожен канал мережі складається з двох портів, що належать різним комутаторам і не є підключеними до жодного з хостів, в іншому випадку каналом виступає порт комутатора, до якого під’єднаний хост. Множина *L* описує усі активні канали мережі. Для обчислення статистики по кожному каналу виконується операція об’єднання між статистикою по інтерфейсах та множиною каналів.

При здійсненні операції об’єднання для двох портів виникатимуть конфлікти. Наприклад, поле Packets Received одного боку каналу має відповідати полю Packets Transmitted іншого каналу, але в реальних умовах це не завжди так. Пакети можуть бути втрачені або промарковані як такі, що містять пошкоджену інформацію та бути відхиленими комутатором. Але навіть, якщо фізичні або логічні проблеми на конкретній ділянці мережі відсутні, для повної відповідності цих полів дані з лічильників мають бути зібрані точно одночасно, що на практиці не є можливим. Для вирішення цієї проблеми при операції об’єднання обирається порт із максимальним значенням .

*Обчислення статистики по маршрутах.* Ціль процесу опрацювання маршрутної статистики – це створення матриці трафіка для всіх пар OD. Для цього має бути виконана операція об’єднання множини LP та згенерованою статистикою по каналах. У такому випадку буде отримана таблиця відповідності маршруту до статистичної мережевої інформації.

Загальний вигляд процесу обробки даних для отримання статистичної інформації по усіх вищенаведених пунктах представлений на рис. 2.2.

**

*Рис. 2.2. Процес отримання статистики трафіка*

Отже, отримані дані можна використовувати для конструювання трафіка у визначених цілях. Запропонований спосіб надає можливість застосовувати більшість аналітичних рішень з підвищення QoS, оптимізації маршрутів тощо. Одним із можливих варіантів є застосування z-оцінки зібраної статистичної інформації.

Для цього для кожної із зібраних величин (наприклад, Bytes Received або Packets Throughput) необхідно обчислити наступні математичні характеристики:

*Середнє значення.* Обчислюємо середнє значення для кожної величини обраного елементу (інтерфейсу, комутатора або маршрута), використовуючи формулу для середнього арифметичного:

(2.7)

,

де – середнє значення, – кількість записів у таблиці, – отримане значення характеристики.

*Середньоквадратичне відхилення.* Відхилення для кожного елемента обчислюється за формулою:

,

де – середньоквадратичне відхилення, – значення характеристики у певному записі таблиці, – середнє значення характеристики для комутатора, обчислене за формулою 2.7, *n* – кількість різних значень.

(2.8)

*Z-value.* Окрім середнього значення та середньоквадратичного відхилення, необхідно ввести оцінку, яка б характеризувала трафік відносно його розподілу. Такою оцінкою може бути z-value, значення якої відображає, скільки та які характеристики відповідають очікуваному розподілу.

Z-value обчислюється за наступною формулою:

(2.9)

,

де – z-оцінка, – значення певного лічильника, – середнє по певному значенню, – середньоквадратичне відхилення по певному лічильнику.

Відповідно до формули 2.9, додатня оцінка вказує на те, що розглянуте значення характеристики трафіка знаходиться вище за середнє, а від’ємна – на те, що розглянуте значення є нижчим за середнє. Така оцінка використовує властивість самоподібності трафіка та вважає його нормально розподіленим. Згідно з цим твердженням, значення, що знаходяться поза діапазоном (-3;3), тобто лежать поза межами є понаднормальними та можуть бути охарактеризовані як викиди.

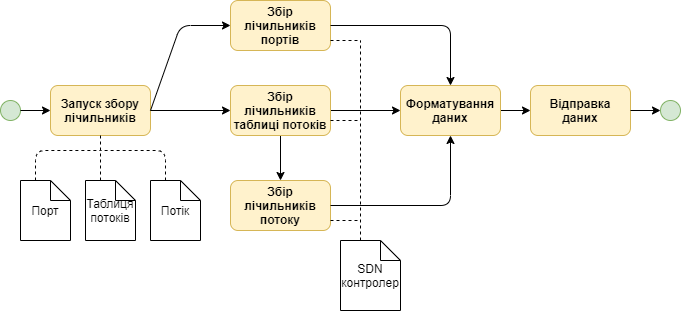
Поведінка трафіка в будь-якій з розглянутих множин може бути проаналізована шляхом обчислення z-value наступних параметрів: отримані пакети (PR), надіслані пакети (PT), отримані байти (BT), надіслані байти (BR), пропускна здатність по отриманих пакетах (PTR), пропускна здатність по надісланих пакетах (PTT), загальна пропускна здатність по пакетах (PTr), пропускна здатність по отриманих байтах (BTR), пропускна здатність по надісланих байтах (BTT), загальна пропускна здатність по байтах (BTr),

## **Формування повідомлень**

Протоколом OpenFlow передбачено можливість підтримки лічильників, що відображають актуальну інформацію на кожному порту комутатора. Дані лічильників оновлюються одразу при проходження пакету через відповідний інтерфейс. У запропонованому способі контролер періодично опитує комутатори для отримання актуальних даних лічильників портів. Згідно зі специфікацією OpenFlow комутаторів, лічильники підтримуються для декількох ресурсів, але в рамках задачі конструювання трафіка достатньо отримати дані з портів комутаторів та таблиць потоків.

Процес формування повідомлень складається з декількох основних етапів (рис. 2.3.):

1. *Збір даних.* Відбувається завантаження інвентарних даних про мережу та даних лічильників з рівнів керування та інфраструктури функціоналом контролера SDN.
2. *Форматування даних.* Формує отримані дані у вигляді, передбаченому системою обміну повідомленнями.
3. *Відправка даних*. З агрегованих даних формуються повідомлення та надсилаються до системи обміну повідомленнями.



*Рис. 2.3. Процес обробки повідомлень*

*Збір даних.* Фактично, дана операція необхідна для здійснення процесу опитування контролера на предмет сформованих даних. Згідно з налаштуванням мережі, контролер виконує опитування інфраструктурного рівня на предмет отримання даних лічильників, а також формує актуальну інформацію про поточну топологію та стан мережі.

Після того, як інформація буде сформована, контролер надсилає дані на прикладний рівень до відповідного додатку, що здійснює агрегацію та формує і надсилає повідомлення до системи конструювання трафіка.

Потокові дані, що безпосередньо засновуються на лічильниках, оновлюються з кожним пакетом, що проходить через відповідний інтерфейс, тому актуальність даних втрачається одразу після здійснення процесу опитування та швидкість зміни таких даних більша за швидкість зміни даних про мережу. При використанні даних з їх значення мають періодично збиратися та одразу надсилатися для подальшого опрацювання.

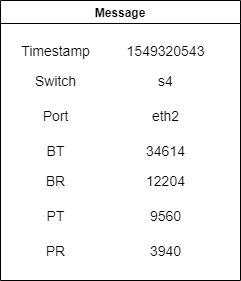
Нижче наведено базовий перелік лічильників, що можуть бути враховані у задачі конструювання трафіку в SDN:

* Packets Received (PR)– кількість отриманих пакетів;
* Packets Transmitted (PT) – кількість переданих пакетів;
* Bytes Received (BR) – кількість отриманих байт;
* Bytes Transmitted (BT) – кількість переданих байт;
* Collision Count (CC) – кількість зіткнень;
* Over RunError Received (ORER) – кількість пакетів з переповненням RX;
* Drops Transmitted (DT) – кількість пакетів, відкинутих TX;
* Drops Received (DR)– кількість пакетів, відкинутих RX;
* Frame Error Received (FER)– кількість помилок вирівнювання фреймів;
* CRC Error Received (CRC) – кількість помилок перевірки контрольної суми;
* Seconds – кількість секунд, протягом якого часу порт активний;
* Nanoseconds – кількість наносекунд в Seconds.

В рамках розробки способу конструювання трафіку будуть використані наступні з лічильників: PR, PT, BR та BT. Решта лічильників можуть бути додані за необхідності, так як дане рішення приймає до уваги можливість масштабування та зміну потреб кінцевого користувача.

Окрім існуючих лічильників, при передачі даних з рівня керування на прикладний рівень контролер лишає часову мітку, коли інформація була відправлена, а також дані, з якого саме інтерфейсу було отримано статистику. Цей етап є дуже важливим в рамках досліджуваної задачі, так як дозволяє структурувати дані про трафік у часі та в топології й оцінити їх актуальність.

*Форматування даних.* Після отримання додатком даних від контролера, відбувається процес форматування. Дані базово структуруються та додається часова мітка, що була отримана разом з даними від контролера. Приклад відформатованого повідомлення наведено на рис. 2.4.



*Рис. 2.4. Структура відформатованого повідомлення*

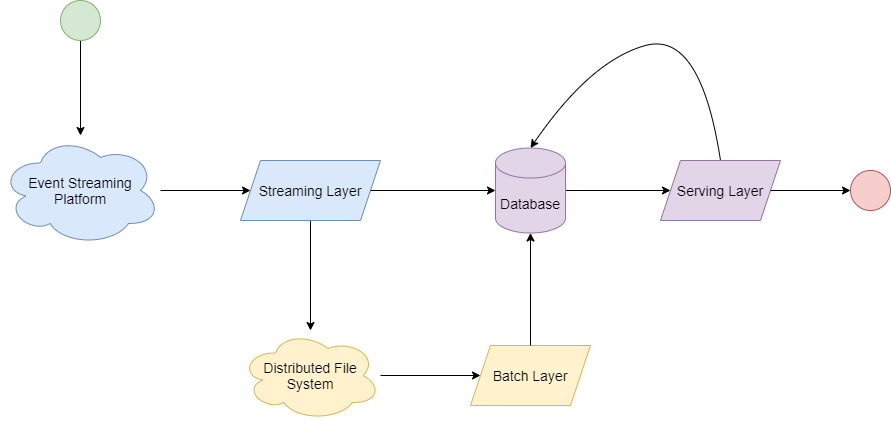
*Відправка даних.* Одразу як тільки дані були відформатовані, вони мають бути відправлені до системи обробки повідомлень. Такий підхід забезпечує своєчасне отримання актуальної інформації від мережі системою конструювання трафіку.

## **Лямбда-архітектура у задачі конструювання трафіка**

Після того, як повідомлення надходять до системи обробки повідомлень, відбувається процес формування черги. Цей процес складається з трьох основних етапів:

1. *Отримання* *відформатованого повідомлення* від відповідного процесу збору даних на порту та за необхідності інвентарних даних,
2. *Збереження повідомлення.* Повідомлення зберігаються у відповідному розділізгідно заголовку. Кожне повідомлення має бути збережене.
3. *Пересилка повідомлення* до процесу, що надсилає необхідний запит по вказаному заголовку.

Після того, як повідомлення збережені у системі обробки повідомлень із відповідним заголовком, вони можуть бути отримані відповідними рівнями лямбда-архітектури. Основним концептом лямбда-архітектури для обробки даних є застосування двох рівнів обробки: потокового та пакетного. Модель лямбда-архітектури для виконання задачі конструювання трафіка зображена на рис. 2.5.

****

*Рис. 2.5. Модель лямбда-архітектури для обробки даних*

Потоковий рівень забезпечує простий та швидкий процес трансформації отриманих даних до структурованого вигляду. Дані обираються з системи обробки повідомлень у якості міні-пакетів (mini-batch). При потраплянні до системи обробки повідомлень кожному повідомленню надається певне значення часового зсуву. Під час запуску потокового рівня конфігурується значення часового зсуву, починаючи з якого дані будуть формуватися у міні-пакети. Значення, при яких повідомлення будуть згруповані для обробки, можуть конфігуруватися залежно від масштабів системи та кінцевих вимог.

Першочергова ціль потокового рівня – надати швидке опрацьоване та структуроване представлення даних. Для кожного з міні-пакетів застосовується певний набір функцій трансформації та дані готуються до запису в сховище виду ключ-значення.

Окрім виконання безпосередньо функції трансформації даних, на потоковому рівні також лежить важлива задача збереження попередньо опрацьованих повідомлень до розподіленої файлової системи. Таким чином усі дані, що проходитимуть крізь потоковий рівень, будуть структуровано збережені для подальшої обробки пакетним рівнем. Таке рішення дозволить зменшити навантаження на систему обробки повідомлень та забезпечить постійність середовища даних.

Згідно з описаним у підрозділі 2.2. способом конструювання трафіка на основі отриманої інвентарної інформації та статистичної інформації про мережу на потоковому рівні спершу відбувається задача *Calculate Interface Statistics.* Процес запитує дані з системи обробки повідомлень, формує міні-пакети, виконує операції групування за інтерфейсом та підсумовує значення лічильників. Обчислена статистика може бути агрегована за найбільшим та найменшим значенням часової мітки та виражати середні значення лічильників за охоплений проміжок часу. Так як потоковий рівень працює тільки з новими повідомленнями, на ньому немає доступу до попередніх значень досліджуваних характеристик, тому питання обчислення аналітичних даних лишається на пакетний рівень.

Далі можливе паралельне виконання наступних двох гілок: калькуляція трафіка по комутаторах і по мережі та калькуляція трафіка за каналами й згодом – за маршрутами.

*Calculate Switch Statistics.* Процес виконує агрегацію обчислених даних за інтерфейсами та підсумовує відповідні характеристики для отримання статистики по комутаторах.

*Calculate Network Statistics.* Процес виконує агрегацію обчислених даних за комутаторами та підсумовує відповідні характеристики для отримання статистики по мережі.

Обидві задачі виконуються для збору, агрегації та акумуляції статистичної інформації з метою подальшого аналізу та можливості покращення якісних характеристик мережі або ж з метою виконання широкомасштабного конструювання трафіка.

*Calculate Link Statistics.* Використовує отриману інвентарну інформацію по каналах для агрегації даних таким чином, щоб об’єднати дані з відповідних інтерфейсів, що складають канал зв’язку. Процес виконує обчислення даних по трафіку, що протікає кожним конкретним каналом. Для виконується операція об’єднання статистики по інтерфейсах та множини L.

*Calculate Path Statistics.* Найголовніша умова для виконання конструювання трафіка – наявність інформації по трафіку відповідно до наявних маршрутів Для обчислення статистики використовується агрегована статистика по каналах та об’єднується з LP-множиною у даному процесі.

Після того, як усі задачі виконані, відбувається з’єднання зі сховищем даних для збереження та додавання нової інформації по кожному з процесів у власний простір ключів.

Пакетний рівень у даній реалізації лямбда-архітектури має дві основні задачі: формування аналітичної інформації на основі даних, опрацьованих потоковим рівнем, та задача пакетної обробки повідомлень, збережених потоковим рівнем до розподіленої файлової системи.

*Формування аналітичної інформації.* Даний процес працює з даними, збереженими до сховища потоковим рівнем. Основною задачею є формування середнього, відхилення та z-value по історичних даних. Структуровані дані завантажуються до системи та опрацьовуються за допомогою методолоії MapReduce.

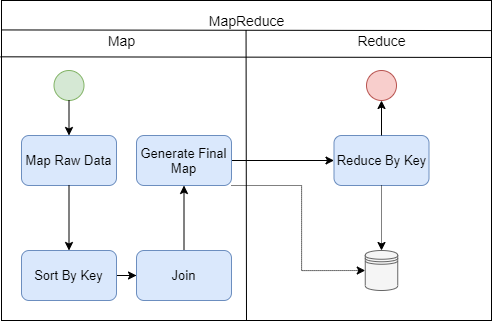
Функція *Map Streaming Data* отримує на вхід дані зі сховища і пакетно генерує асоціативний масив у вигляді ключ-значення, де ключем є інтерфейс, а значення – кількість записів у пакеті та сума значень обраної характеристики, наприклад, пропускної здатності.

Функція *Reduce Streaming Data* підсумовує значення кількості записів та сум значень характеристики, та виконує ділення суми на кількість. Таким чином, знаходиться середнє значення характеристики. Аналогічно обчислюються і середньоквадратичне відхилення, і z-value.

Зазначений пакетний процес є опціональним і може бути застосований, якщо аналітична інформація має оновлюватися частіше, аніж налаштований розклад пакетної обробки повідомлень. Загалом, основний процес розрахунку статистичних характеристик трафіка має виконуватися під час пакетного рівня обробки повідомлень.

Переваги потокового рівня у швидкості отримання інформації перекриваються недоліками у точності: можливе дублювання повідомлень (тобто, є вірогідність потрапляння одного й того самого повідомлення в декілька міні-пакетів) та їх відсутність (ймовірність втрати повідомлення між міні-пакетами). Для цього усі повідомлення, що надходять до потокового рівня, мають бути збережені до розподіленої системи даних і подальшої обробки.

Пакетна обробка повідомлень зазвичай є значно більш трудомісткою, повільною, але досить точною. Процес пакетної обробки здебільшого не виконується постійно, а відбувається за розкладом з метою синхронізації даних між рівнями. Для виконання пакетного рівня також зручно використовувати парадигму MapReduce (рис. 2.6.).



*Рис. 2.6. Модель виконання обчислень MapReduce*

Попередньо відформатований масив повідомлень знаходиться в розподіленій файловій системі. Структура повідомлень співпадає з тою, що застосовуються в потоковому рівні. Отже, задачі конструювання трафіка фактично включають у себе виконання наступних кроків:

*Calculate Interface Statistics.* Процес отримує на вхід збережені повідомлення у форматі розподіленої файлової системи і розбиває їх на пакети. Для кожного пакету створюється асоціативний масив у вигляді ключ-значення, де ключем виступає інтерфейс та часова мітка, а значеннями – відповідні числові значення лічильників. На цьому ж етапі до значень додається кількість записів у пакеті для подальшого обчислення аналітичних ознак. На етапі сортування дані сортуються за інтерфейсами та передаються на етап розрідження. У процесі розрідження дані групуються за ключами та виконується підсумовування відповідних значень (характеристик), окремими полями цей процес має повернути суми, розділені на сумарну кількість. Отриманий набір даних об’єднується з інформацією, що надійшла з потокового рівня, як основний індекс при цьому використовується дані пакетної обробки в себе аналогічні кроки й на пакетному рівні.

Далі знову можливе паралельне виконання двох гілок: обчислення мережевої статистики та статистики по маршрутах.

*Calculate Switch Statistics.* Опрацьовані дані за інтерфейсами розбиваються на пакети. Для кожного з пакетів створюється асоціативний масив, де ключами стають часові мітки та ідентифікатори комутаторів, а значення – відповідні лічильникові характеристики, а також кількість записів у пакеті. Далі пакети сортуються за ідентифікатором комутатора. На етапі розрідження пакети групуються за ключами та підсумовуються значення.

*Calculate Network Statistics.* У цьому процесі асоціативний масив ключ-значення має за ключ часові мітки, а значення – значення лічильників, отримані на етапі обчислення статистики комутаторів. Дані сортуються за часовою міткою та відправляються на етап розрідження. Тоді знову ж відбувається групування за ключами та значення підсумовуються.

Після кожного з етапів відбувається збереження даних як в розподілену файлову систему для забезпечення можливості резервного відновлення, так і співставлення отриманої інформації з потоковим набором даних.

*Calculate Link Statistics.*  На цьому етапі маємо дві паралельні однакові задачі створення асоціативного масиву. Дані беруться зі статистики інтерфейсів, де ключами виступають інтерфейси та часові мітки, а значеннями – поля лічильників. Після того, як масиви будуть сформовані, відбувається об’єднання двох задач *Map* в одну, де попередні задачі групуються за ключами відповідно до отриманої інвентарної інформації з множини *L.* Після цього дані сортуються за часовою міткою та розріджуються за ключами і підсумовують значення. Вирішення конфліктних ситуацій з різними можливими значеннями для каналів було описано у підрозділі 2.2.

*Calculate Path Statistics.* Даний процес є аналогічним попередньому, з тою лише різницею, що за вхідні дані приймаються опрацьовані дані по мережевих каналах, а в якості інвентарних даних використовується множина *LP.*

Після того, як обидва процеси будуть завершені, дані зберігаються в розподіленій файловій системи за відбувається співставлення з набором даних, отриманих в потоковому режимі. Усі конфліктні ситуації вирішуються в бік даних з пакетного рівня обробки.

Як і в інших способах конструювання трафіка в мережах SDN, ефективність запропонованого способу основною мірою залежить від надійного стабільного зв'язку всередині мережі, а саме між рівнем керування та інфраструктури, від обчислювальних можливостей контролера та розмірів мережі. У решті показників, особливо для високонавантажених мереж, пропонований підхід дозволяє швидко опрацьовувати великі обсяги статистичної інформації, а також забезпечити механізми резервного відновлення на випадок непередбачуваних збоїв системи. Окрім цього, такий підхід дозволяє гнучко налаштувати розклад виконання пакетної обробки для отримання аналітичної інформації та передбачує можливість налаштування систем, що будуть використовувати зібрану інформацію для додаткового аналізу та прогнозування трафіка.

## **Висновки до розділу 2**

У даному розділі було запропоновано спосіб конструювання трафіка високонавантажених системах SDN на основі сучасних підходів для роботи з великими даними. Враховано головні особливості мережевих технологій SDN, такі як централізований контролер, можливості протоколу OpenFlow для формування статистичної мережевої та інвентарної інформації.

Описано спосіб формування повідомлень, засновуючись на мережевих даних, що може збирати та постачати контролер SDN, розглянуто основні аспекти збору, формування та відправки даних для подальшої обробки.

Запропоновано спосіб конструювання трафіка, що враховує особливості потокової роботи з повідомленнями та заснований на стані мережі. Для цього пропонується використання інформації з лічильників та інвентарної інформації про мережу. Розглянуто спосіб формування необхідних множин для створення матриці трафіка та процеси обробки статистичної інформації, на основі яких можливо вирішити задачу конструювання трафіка.

Розглянуто ключові етапи побудови лямбда-архітектури для вирішення задачі обробки зібраних даних для конструювання трафіка. Зазначено переваги та недоліка потокового та пакетного рівнів та необхідність їх взаємодії в рамках зазначених цілей. Запропоновано вирішення можливих проблем з резервним відновленням та можливості розвитку системи прогнозування трафіка.

# **РОЗДІЛ 3**

# **РЕАЛІЗАЦІЯ МОДЕЛІ СПОСОБУ КОНСТРУЮВАННЯ ТРАФІКА**

Моделювання способу конструювання трафіка в SDN на основі лямбда-архітектури полягає в здійсненні наступних кроків:

* визначення топології мережі;
* виконання процесу генерації даних про трафік;
* реалізація системи обробки даних на основі лямбда-архітектури;
* реалізація інтерфейсу для візуалізації результатів моделі.

## **Огляд засобів реалізації**

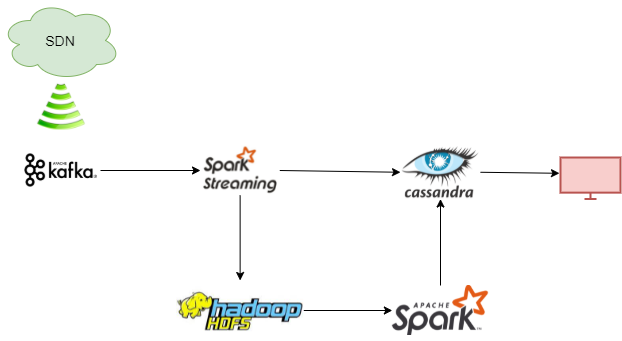
Для виконання моделювання програмного додатку способу конструювання трафіку в програмно-конфігурованих мереж було обрано мову програмування Python 3.7. Вибір заснований на широких можливостях моделювання мережі, наявності готових бібліотек для виконання наукових обчислень та для роботи з графами, наявності програмних рішень для роботи з великими даними, простотою реалізації веб-інтерфейсу. Окрім цього, більшість контролерів SDN мають підтримку програмних додатків, виконаних мовою Python Зокрема, при виконання моделі були використані наступні програмні засоби Python:

* бібліотека networkx для виконання роботи з графами та топологією мережі;
* бібліотеки numpy та scipy для виконання наукових обчислень;
* бібліотеки kafka-python, pyspark для реалізації лямбда-архітектури;
* бібліотека dash для створення веб-інтерфейсу.

Розробка програмного додатку була виконана у інтегрованому середовищі розробки PyCharm компанії JetBrains.

Для реалізації лямбда-архітектури пропонується використовувати наступний стек технологій (рис. 3.1.):

* Apache Kafka в якості рішення імплементації системи потокової обробки та збереження повідомлень:
* Spark Streaming для реалізації потокового рівня лямбда-архітектури – потокової обробки повідомлень, збереження повідомлень до розподіленої файлової системи та взаємодії з базою даних;
* Hadoop HDFS у якості розподіленої файлової системи;
* Apache Spark для реалізації пакетного рівня лямбда-архітектури – пакетної обробки даних, взаємодії з розподіленою файловою системою та взаємодії з базою даних;
* Apache Cassandra – NoSQL база даних для збереження даних, отриманих від потокового та пакетного рівнів лямбда-архітектури.



*Рис. 3.1. Програмні засоби реалізації лямбда-архітектури*

Для забезпечення кросс-платформеності та незалежності від середовища виконання, а також цілісності системи було прийнято рішення використовувати технології контейнеризації Docker та Docker Compose. Також однією з причин використання контейнеризації є постачання базових технологічних компонент у якості образів Docker.

При роботі з програмою рекомендуються такі мінімальні технічні характеристики комп’ютерної системи:

* процесор з тактовою частотою не менше 2 ГГц;
* об’єм оперативної пам’яті не менше ніж 8 ГБ;
* вільний об’єм накопичувача не менше 16 ГБ;
* наявність встановленого Docker у системі.

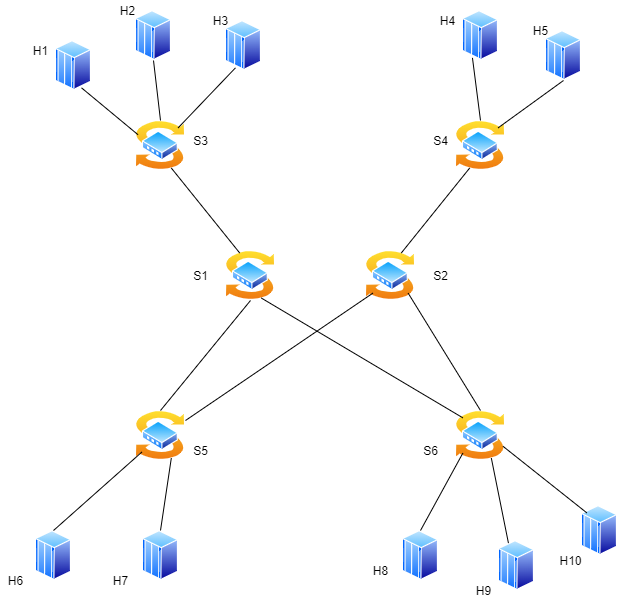
## **Структура програмного додатку**

Для моделювання була створена топологія мережі, що складається з десяти хостів, шести комутаторів OpenFlow та одного контролера. Схема топології наведена на рис. 3.2.

Аналогічно до інвентарних даних, що надсилаються контролером, мережа задається двома множинами: L1 (канали між хостами та комутаторами) та L2 (канали між комутаторами, міжлінки). Ці множини містять інформацію про всі наявні хости, комутатори, активні інтерфейси комутаторів та їх зв'язки. На основі вказаних множин у середовищі networkx будується граф мережі. З цього графу маємо можливість отримати будь-які мережеві характеристики, наприклад, список хостів чи список комутаторів. Додатково створюємо декілька методів, що будуть відповідати за визначення каналів у мережі відносно інтерфейсів комутаторів, а також для визначення усіх маршрутів мережі та генерації *LP-множини*.

Генерація множини *LP* виконується за алгоритмом, описаним у розділі 2.1. У модельованій мережі для створення цієї множини були визначені множини *H, S, L, P,* які задаються інформацією про топологію модельованої мережі.

Метод *get\_path()* приймає на вхід пару хостів source/target і повертає активний маршрут для цієї пари. Метод *get\_ports()* приймає на вхід маршрут і повертає список інтерфейсів, через які проходить даний маршрут. Результатом роботи алгоритму є асоціативний масив, у якому ключем є інтерфейс, а значенням – список маршрутів, що проходять через даний інтерфейс.



*Рис. 3.2. Топологія мережі*

Відомо, що за допомогою особливостей OpenFlow контролер SDN має можливість отримувати та пересилати інвентарні дані та дані з лічильників комутаторів. За відсутності підключення до існуючої мережі SDN та реальних даних з лічильників виконаємо моделювання генерації цих даних для демонстрації розробленого методу. Так, за обраною топологією для кожного з активних інтерфейсів створимо метод для генерації випадкових даних, що будуть імітувати реальні процеси при проходженні пакетів через комутатори OpenFlow. Обрані поля Bytes Received, Bytes Transmitted, Packets Received та Packets Transmitted мають випадково генеруватись у заданому діапазоні з певною періодичністю, поле Timestamp буде визначатись залежно від поточного часу в системі, а поля Switch та Port відповідно заповнюватимуться згідно з інвентарною інформацією. Згенеровані дані оформлюються як словник, у якому ім'я поля виступає ключам, а отримані дані – значенням.

Для передачі отриманих лічильникових та мережевих даних в систему обробки повідомлень ініціалізується продюсер Kafka, що буде потоково транслювати повідомлення у відповідний попередньо заданий топік. Для того, щоб виконати умову подібності даних, створені під час генерації словники трансформуються у форматі JSON та серіалізуються. Обрані бібліотеки kafka-python та PySpark мають власні реалізації серіалізації та десеріалізації даних, тому немає необхідності додатково реалізовувати цей крок. Продуковані таким чином повідомлення зберігаються у потоковій системі збереження та обробки повідомлень та можуть бути отримані за допомогою консьюмера при передачі таких параметрів, як назва топіку та адреса й порт сервісу.

Процес генерації мережевих даних запускається як окремий модуль і транслює дані, що отримуються продюсером із визначеним ім’ям топіка до попередньо запущеного сервісу Kafka. Так як засобом реалізації Kafka було обрано готовий образ, залишається лише виконати мережеві налаштування та Port Forwarding. За замовчуванням сервіси Kafka використовують порт 9092 для обміну інформацією.

Після того, як було запущено процес генерації мережевих даних, повідомлення одразу зберігається в Kafka. Згідно з концепцією, описаною в розділі 2, ці повідомлення мають бути зчитані, опрацьовані та збережені потоковим рівнем лямбда-архітектури, що імплементується за допомогою PySpark, а саме Spark Streaming. Для імплементації створюється окремий модуль, що буде запущено в docker-образі Spark за допомогою spark-submit.

Обробка повідомлень, їх збереження а також додавання опрацьованих даних в базу даних відбувається у одному модулі Spark Streaming. Спершу за допомогою програмних методів PySpark ініціалізується Kafka консьюмер, що буде під’єднуватись до системи роботи з повідомленнями та отримувати інформацію про надходження та вміст повідомлень згідно з вказаним топіком. Після того, як новий міні-пакет повідомлень буде отриманий, відбувається їх десеріалізація та формується міні-пакет JSON-об’єктів. Кожен з цих об’єктів має стандартну структуру та має бути розпакований як запис до PySpark датафрейму, що містить колонки, аналогічні до ключів словника даних, що генеруються. Відповідно, датафрейм заповнюється значеннями отриманого асоціативного масиву з JSON-об’єкту.

Після того, як початковий датафрейм було сформовано, може бути розпочато виконання задач конструювання трафіка, що описані у підрозділі 2.3, а також може бути виконано збереження даних до розподіленої файлової системи. Основні задачі можна розбити на чотири етапи:

1. Збереження початкового датафрейму до HDFS
2. Розрахувати з початкового датафрейму статистику для інтерфейсів
3. Виконати трансформації для обчислення статистики комутаторів та мережі
4. Виконати трансформації для обчислення статистики каналів мережі та маршрутів

Процес збереження до розподіленої файлової системи досить простий. Програмні засоби PySpark дозволяють одразу взаємодіяти із запущеним середовищем з HDFS, вказавши адресу та порт сервісу, директорію розподіленої файлової системи, з якою буде проводитись робота, та запровадити схему датафрейму, що буде зчитано або збережено. В нашому випадку дані зберігаються у форматі Parquet — це стовпчастий формат, який підтримується багатьма системами обробки даних. Spark забезпечує підтримку як читання, так і запису файлів Parquet, що автоматично зберігає схему вихідних даних.

Окрім цього, також окремо налаштовується процес резервного копіювання: до розподіленої файлової системи зберігаються файли контрольних точок, що містять у собі службову інформацію про те, які дані були опрацьовані, із яким часовим зсувом тощо. Цей процес є необхідним на випадок непередбачуваних зупинок або збоїв системи і дозволяє швидко відновити процес потокової обробки даних з, наприклад, останнього часового зсуву, що було оброблено.

Задача *Calculate Interface Statistics* має на увазі сукупність запитів SQL до початкового датафрейму для отримання статистики по інтерфейсах. Спершу дані групуються за полями *timestamp, switch* і *port.* Так як поле timestamp містить інформацію з точністю до мілісекунд, необхідно попередньо виконати апроксимацію до секудних значень для можливості виконання групування. Даліобчислюється мінімальне та максимальне значення часової мітки, різниця між якими виражає час S активності портів, що охоплює міні-пакет. Також обчислюється сума відповідних стовпців лічильників, отримані значення діляться на S для отримання середніх по часу, що охоплюється міні-пакетом. Зазвичай проміжки часу, що входять до міні-пакету, обмежуються до 5-10 секунд, що дає можливість одразу отримати середні характеристики за проміжком. Окрім цього, виконуються обчислення додаткових характеристик трафіка, описаних у розділі 2.2., таких як BTR, BTT, PTR, PTT, BTr, PTr.

Для забезпечення консистентності даних у часі застосовуються методики обробки даних, що запізнилися та віконні функції. У реальній системі можуть бути випадки, коли, наприклад, повідомлення, надіслане о 15:00, надходить до програми о 15:10. У цьому випадку модуль має використовувати час 15:00 замість 15:10, щоб оновити попередні підрахунки для вікна 14:58 – 15:08. Це відбувається природньо завдяки наявним програмним реалізаціям групування на основі вікон – в такому випадку структурований потік може підтримувати проміжний стан для часткової агрегації протягом тривалого періоду часу.

Результатом цих трансформацій є датафрейм зі стовпцями, що відображають статистику за період часу, охоплений міні-пакетом, та відповідними полями.

Далі фактично необхідно запустити два паралельні процеси – обчислення мережевої статистики та статистики по маршрутах.

Задачі *Calculate Switch Statistics* та *Calculate Network Statistics* виконуються послідовно одна за одною і приймають на вхід *Interface DataFrame і Switch DataFrame* відповідно. На цих етапах датафрейми групуються віконним способом за стовпцями *timestamp* та *switch* для статистики за комутаторами та стовпцем *timestamp* для мережевої статистики. Принципи агрегації та підсумовування лишаються такими самим, як і в попередній задачі з обчислення статистики за інтерфейсами.

Процеси *Calculate Link Statistics* та *Calculate Path Statistics* також виконуються послідовно один за одним. Для виконання операції обчислення статистики каналів дані, що були отримані після агрегації статистики інтерфейсів мають бути акумульовані з інвентарними даними щодо каналів. Для цього виконується операція *Join,* що проходить по всіх каналах мережі, виконує вибір записів інтерфейсів, які формують конкретний канал та об’єднує значення за принципами, описаними в розділі 2.2. Так, результатом виконання об’єднання є датафрейм, у якому для кожного каналу мережі зібрана відповідна статистична інформація, усереднена за часом охоплення міні-батчу. Аналогічно використовується інвентарна інформація про активні маршрути, яка знаходиться у LP-множині для обчислення статистики по машрутах, але процес обчислення має суттєві відмінності. Так, для конкретного маршруту виконується вибір записів з датафрему каналів за наявністю каналу в маршруті та підсумовуються значення по кожному з стовпців, операція виконується для кожного маршруту з LP-множини. Після цього маємо результуючий датафрейм з актуальними даними по кожному з маршрутів мережі.

Усі отримані дані на потоковому рівні мають бути збережені до бази даних Cassandra. Для цього попередньо під час запуску системи створюється або ж задається схема, у якій зазначено, який простір ключів використовується, які таблиці будуть заповнені та для кожної таблиці вказується набір полів з очікуваними типами значень. У випадку розробленої системи використовується простір ключів Lambda Architecture з таблицями SwitchPort Statistics, Switch Statistics, Network Statistics, Link Statistics і Path Statistics.

Для збереження даних до сховища Cassandra використовуються стандартний програмний конектор PySpark, який ініціалізується на початку виконання модуля і виконує і закриває з’єднання одразу по запиту. Для налаштування роботи конектора було вказано бібліотеку конектора та конфігураційні й мережеві дані для доступу до бази даних. Збереження даних відбувається щоразу, як тільки виконується певна задача до відповідної таблиці у режимі додавання даних в кінець таблиці.

Конфігурація пакетного рівня виконана інакшим чином, щоб задовольнити потреби в достовірності та коректності даних, а також для досягнення цілі аналізу історичних даних.

Для імплементації пакетного рівня створюється окремий модуль, що реалізовується за допомогою програмних засобів Spark. Задачі лишаються ті ж самі, що й для потокового рівня, але обчислюватись вони будуть для інших, історичних даних. Ці дані зберігаються в конкретній директорії у форматі Parquet-файлів, за їх збір, як було описано вище, відповідає потоковий рівень.

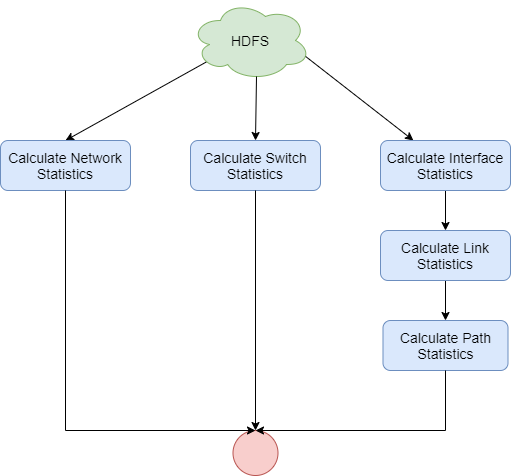
Ієрархія задач дещо відрізняється від потокового рівня: процес конструювання трафіка може бути розбитий на більшу кількість паралельних задач, як показано на рис. 3.3. Так як при реалізації пакетного рівня використовується методологія MapReduce, задачі обчислення статистики за інтерфейсами, комутаторами та цілком за мережею можуть бути виконані паралельно з одного масиву даних. Ключовою різницею між цим задачами є схема, за якою обираються ключі:

* У задачі *Calculate Interface Statistics* в якості ключів виступають поля (*port, switch, timestamp);*
* У задачі *Calculate Switch Statistics* в якості ключів виступають поля (*switch, timestamp);*
* У задачі *Calculate Network Statistics* ключем є поле *timestamp.*

Для всіх перерахованих задач як значення обираються відповідні значення полів. Окрім цього, завдяки паралельному розподіленню цих задач, для кожної з характеристик мережевого трафіку середні значення, відхилення та z-оцінки рахуватимуться окремо, що дозволяє відокремити аналіз кожної з компонент.

Виконання обчислень задач *Calculate Link Statistics* та *Calculate Path Statistics* відбувається згідно з логікою, описаною в розділі 2.3.

Після виконання кожної з задач відбувається з’єднання з базою даних за допомогою конектора та програмних засобів PySpark для об’єднання результатів потокового рівня та пакетного рівня. Кожна з отриманих під час пакетного рівня таблиць об’єднується з існуючою з вищим пріорітетом нових табличних даних.



*Рис. 3.3. Ієрархія задач пакетного рівня*

Створений модуль пакетного рівня PySpark запускається окремо на новій ноді застосунку Spark за допомогою spark-submit із вказуванням усіх конфігураційних файлів та конекторів, що використовуються.

На цьому етапі розробки програмної моделі маємо повний конвеєр даних, від надходження статистичних повідомлень до системи до оброблених, структурованих та синхронізованих аналітичних даних, що зберігаються у сховищі Cassandra.

Для візуалізації результатів було створено окремий сервіс за допомогою бібліотеки Dash, що дозволяє оперативно розробити та розгорнути веб-додаток для перегляду аналітичної інформації процесу конструювання трафіка, що зберігається в базі даних. Для цього застосовується окремий конектор, що періодично виконує запити до бази даних та відображає отриману інформацію на інтерактивному дашборді. Цей застосунок дозволяє перевірити коректну роботу запропонованого способу конструювання трафіка та може бути використаний в демонстративних цілях на етапі впровадження стартап-моделі.

Окрім цього, для забезпечення можливості запуску на будь-якій платформі, незалежно від операційної системи чи налаштованого середовища, було прийнято рішення виконати задачу контейнеризації програмної моделі за допомогою інструментів Docker та docker-compose. Так як більшість програмних компонент постачаються у вигляді docker-образів, основною задачею залишається написання конфігураційного файлу, що має об’єднати усі компоненти системи в одну мережу, а також зазначатиме налаштування середовища, що має бути розгорнуто при новому запуску системи. Окрім цього, такий спосіб оформлення програмного додатку також може бути корисним у демонстраційних цілях.

## **Моделювання способу конструювання трафіка**

Для демонстрації роботи способу конструювання трафіка на основі лямбда-архітектури необхідно розгорнути середовище та виконати запуск програмних компонент.

Для розгортання середовища було створено конфігуративний файл docker-compose, в якому об'єднані усі елементи лямбда-архітектури в одну мережу, вказано необхідні конфігурації та залежності. Модулі, що запускає docker-compose, включають в себе Zookeeper, Kafka, Cassandra, Spark (а точніше дві ноди spark-master і spark-worker) та HDFS.

Після того, як усі сервіси будуть запущені та відправлять повідомлення в логбук про свою готовність, можна розпочати роботу з системою. Для оперативного запуску системи обробки повідомлень, створення схеми даних у базі даних, а також роботу з каталогами розподіленої файлової системи було створено окремий shell-скрипт, при запуску якого відбуваються вищеописані операції.

Як тільки скрипт буде виконано, наступною запускається задача обробки даних потокового рівня StreamingProcessor за допомогою вбудованої утиліти Spark-ноди spark-submit. При запуску цього модуля на кластері Spark виконується компіляція, планування та розгортання потокових задач, описаних у розділі 3.2. Одразу після розгортання модуль запитує дані від системи обробки повідомлень із вказаним у конфігурації топіком та із визначеним часовим зсувом. При першому запуску жодних даних у Kafka немає, тому необхідно окремо запустити генератор мережевих даних, що у циклі створюватиме, форматуватиме та надсилатиме до Kafka повідомлення з даними про трафік, які будуть отримуватись, оброблятись та зберігатись запущеною Spark-задачею.

Після того, як дані почали генеруватись, в потоковому рівні відбувається процес трансформації даних, збереження повідомлень до розподіленої файлової системи, а також збереження опрацьованих даних у відповідні таблиці простору ключів Cassandra. Перші опрацьовані дані про трафік будуть доступні вже через декілька секунд після запуску процесу генерації.

Також після того, як було запущено генерацію трафіка, локально запускається модуль сервісного рівня, тобто веб-додаток, що виконує запити до Cassandra та відображає аналітичну інформацію у заданому дашборді. Дані почнуть відображатися одразу, як тільки потоковий рівень виконає необхідні записи до бази даних.

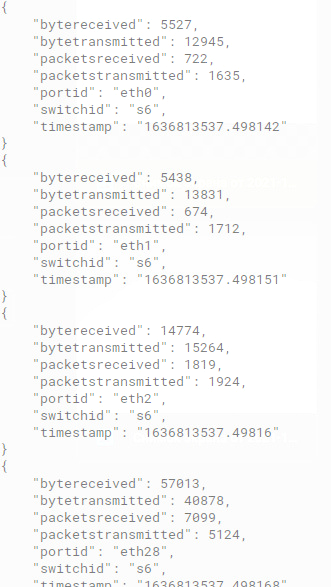
Запуск модуля пакетного рівня BatchProcessor концептуально має виконуватись за розкладом, але в демонстраційних цілях можливий і запуск одразу після старту решти сервісів. Для цього також виконується утиліта spark-submit. Після передачі до утиліти модуля пакетної обробки даних, на кластері Spark відбувається компіляція, планування та виконання задач пакетного рівня, описаних у розділі 3.2. Після цього отримані дані порівнюються з тими, що вже наявні в базі даних після роботи потокового рівня та зберігаються відповідно до нової актуальної інформації. В цьому випадку, коли сервісний рівень виконає новий запит з бази даних, вже будуть отримані дані, що були розміщені в таблицях пакетним рівнем. Ці дані не обов'язково будуть відрізнятися від попередніх, так як головна ідея пакетної обробки у консистентності та надійності, але також після виконання цієї задачі будуть актуалізовані поля, що відображають середнє значення, середньоквадратичне відхилення та z-оцінку різних сутностей, на основі якої відбувається процес конструювання трафіка.

Інвентарні дані про мережу попередньо задаються конфігураційним файлом JSON-формату, який містить інформацію про L1 та L2 множини. Ці дані містять інформацію про хости та комутатори мережі та їх канальні з’єднання. Відповідно, за наданою інформацією будується граф системи та множини каналів та маршрутів.

Також, ця інформація використовується для генерації трафіка за інтерфейсами та відправка його до системи обробки повідомлень Kafka. Для цього було запускається модуль sdn\_kafka\_producer.py, що зчитує інвентарні дані, ініціалізує Kafka-продюсер та ітеративно для кожного з інтерфейсів генерує мережеві дані. Після цього дані форматуються, серіалізуються та надсилається продюсером до системи обробки повідомлень.

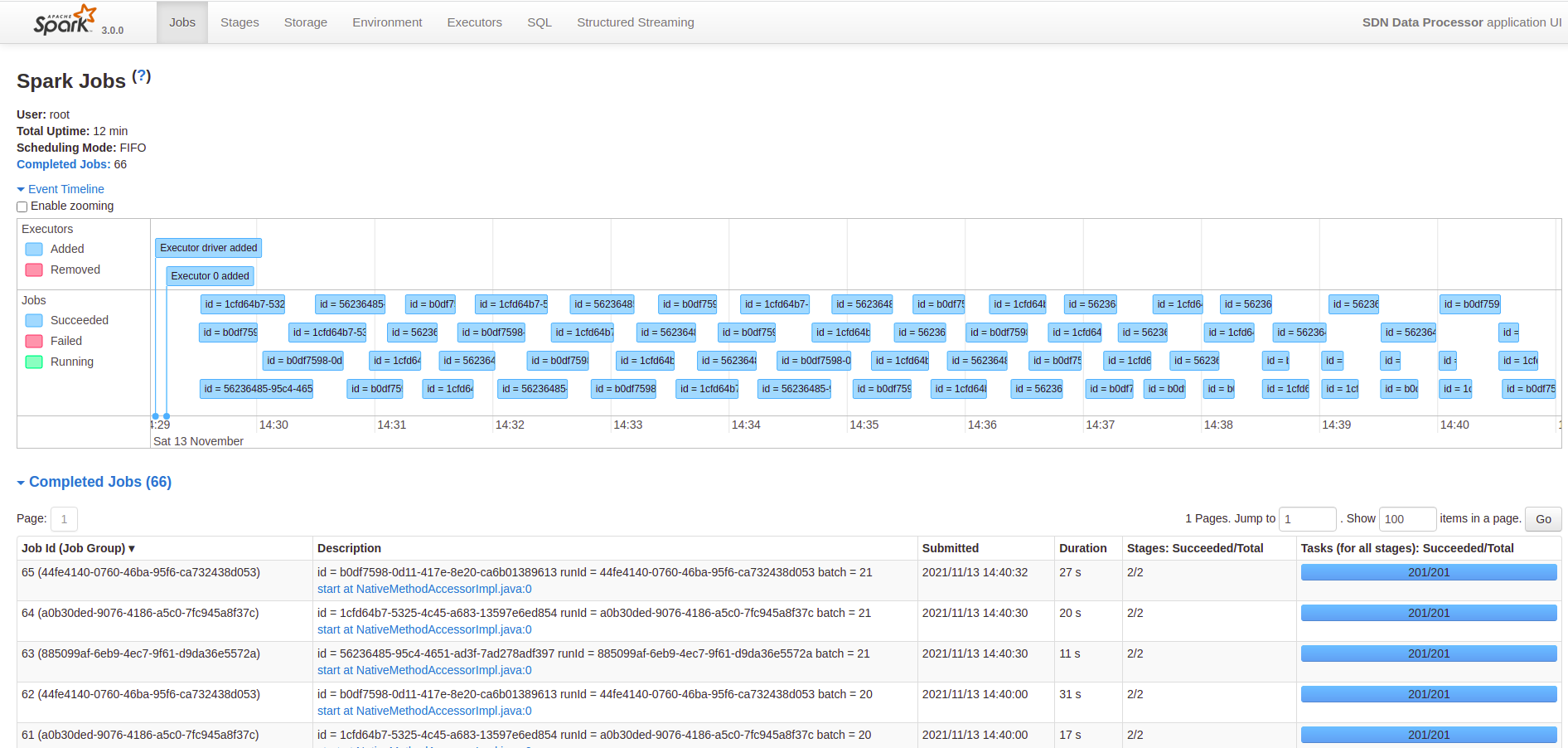
В рамках демонстрації роботи системи, під'єднаємося до працюючого кластеру Kafka та запустимо процес консьюмера, що слідкуватиме за топіком мережевих даних, в нашому випадку це sdn-data-event. Нижче на рисунку 3.4 наведено частина логів системи обробки повідомлень, що слідкують за обраним топіком. На цьому рисунку зображено структуровані повідомлення, що вже знаходяться в Kafka. Поля *bytesreceived, bytestransmitted, packetsreceived* та *packetstransmitted* являють собою згенеровані мережеві дані; поля *portid* та *switched –* дані про місце у топології мережі, та поле *timestamp –* інформація про часову мітку, у який час було згенеровано дані.

Після того, як дані були отримані Kafka, запустимо StreamingProcessor модуль на кластері Spark. Стандартні засоби програмного застосунку Spark дозволяють виконувати моніторинг по виконаних задачах не тільки за допомогою логів, але й використовуючи веб-інтерфейс. Під’єднаємося до Spark User Application та перевіримо, як виконується процес потокової обробки повідомлень.



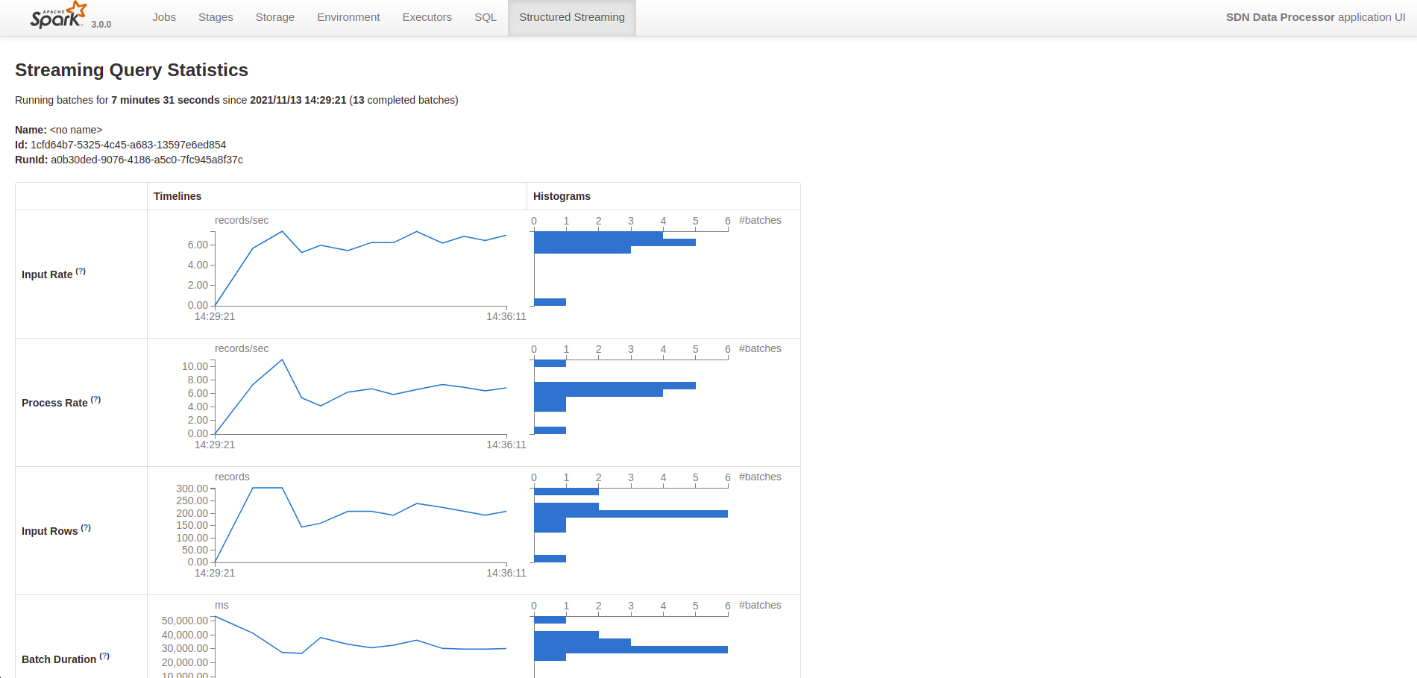
*Рис. 3.4. Вивід консьюмера Kafka за топіком sdn-data-event*

На рисунку 3.5. зображено вивід інтерфейсу користувача SDN Data Processor, отриманий за допомогою Spark User Application. У цьому інтерфейсі є можливість відслідкувати, як саме було заплановано виконання описаних у розділі 3.2. задач. Як бачимо, для потокового рівня задачі виконуються з певною, заданою в конфігурації, періодичністю, у даному випадку мінімальний проміжок між задачами складає 5 секунд. Задачі виконуються паралельно згідно з заданим конвеєром даних да особливостями Spark-додатку. Також на рис. 3.5. відображається інформація про те, скільки часу займає виконання кожної окремої задачі з використанням доступних апаратних ресурсів.



*Рис. 3.5. Інтерфейс моніторингу Spark задачі SDN Data Processor*

Детальну інформацію про ефективність виконання задач можна отримати у розділі Structured Streaming. На рисунку 3.6. зображено вивід статистики виконання задачі потокового рівня лямбда-архітектури. Така інформація допомагає створити оптимальну конфігурацію потокового рівня для роботи з наявним обсягом даних та апаратними ресурсами. У даному випадку, так як процес було запущено на локальній машині, опрацювання кожного міні-пакету відбувається довше, ніж те саме рішення, розгорнуте на спеціальному сервері.



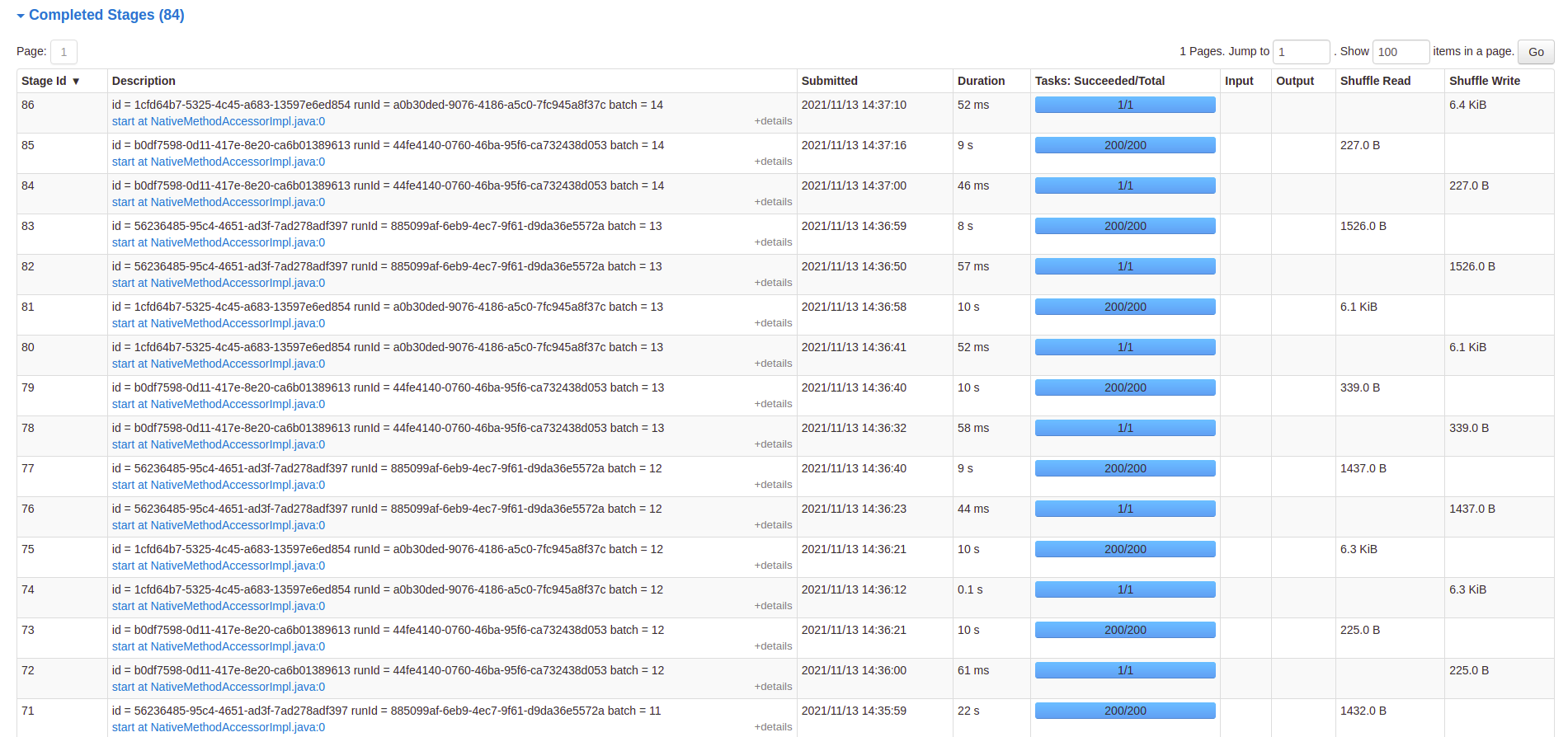
*Рис. 3.6. Статистична інформація Structured Streaming задачі SDN Data Processor*

Також додатково розглянути статистику виконання кожної з підзадач модуля StreamingProcessor можна у розділі Stages. На рис. 3.7. зображена статистика виконання задач обробки та збереження даних, також задач збереження повідомлень до розподіленої файлової системи на кластері Hadoop. Тут відображається інформація щодо часу виконання підзадач для кожного з міні-пакетів даних, а також інформація про обсяг даних, які було збережено до сховища.

Аналогічна інформація доступна для перегляду та аналізу після запуску задачі пакетного рівня BatchProcessor на кластері Spark.

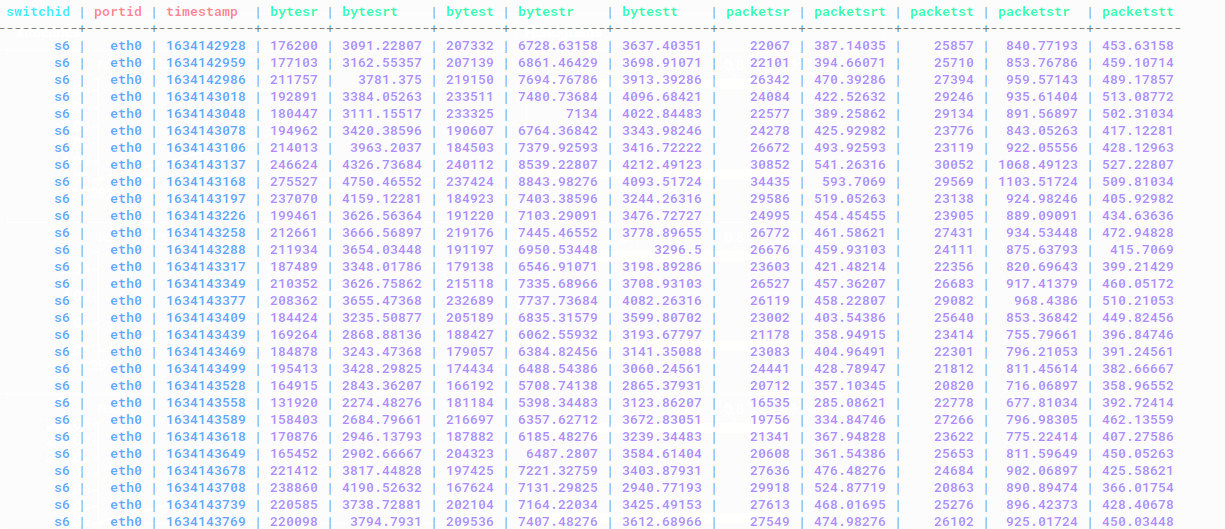
Основні задачі зі створення матриці трафіка покладені саме на вищенаведені процеси. Після того, обидва процеси будуть запущені, перевіримо збереження даних у сховищі Cassandra.

Схема простору ключів Cassandra створюється згідно з потребами системи на етапі ініціалізації. Вона включає в себе 5 таблиць: *Switch Port Statistics, Switch Statistics, Network Statistics, Link Statistics* та *Path Statistics.* Кожна з цих таблиць заповнюється та оновлюється після виконання певної підзадачі модулів StreamingProcessor та BatchProcessor.

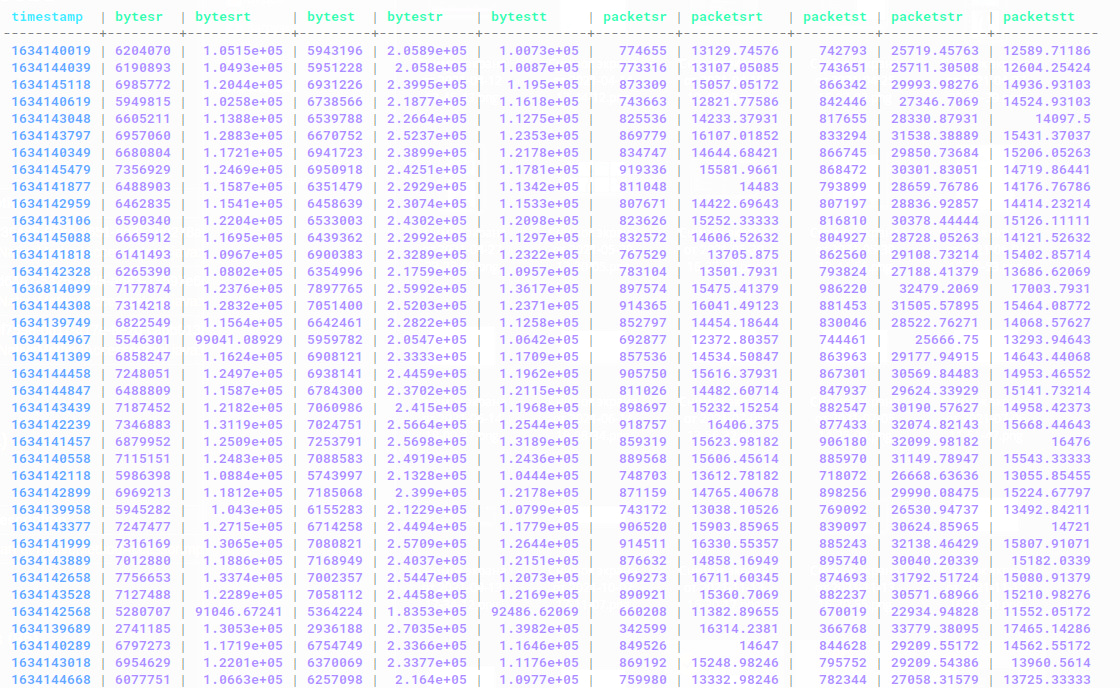


*Рис. 3.7. Статистична інформація Structured Streaming задачі SDN Data Processor*

Так, на рисунку 3.8. наведено структура та частина опрацьованих даних у вигляді матриці трафіка за інтерфейсами мережі, на рисунку 3.9. наведено матриця усього мережевого трафіка за певний проміжок часу. Аналогічна інформація міститься і в таблицях зі статистикою по каналах та маршрутах. Дана інформація може бути використана в рамках моніторингу мережі SDN, аналізу поведінки трафіка, а також для виконання задачі конструювання трафіка.



*Рис. 3.8. Матриця трафіка для інтерфейсів мережі*

**

*Рис. 3.9. Мережева матриця трафіка*

Таким чином, за допомогою розробленого програмного додатку було отримано аналітичні дані, за якими виконується конструювання трафіка на основі стану мережі. Такі дані містять повну інформацію, включаючи статистичні оцінки трафіка, на основі яких можливо запровадити системи предиктивної аналітики, такі як прогнозування трафіка, а також одразу застосувати будь-який з алгоритмів, заснованих на стані мережі для впливу на якісні показники SDN.

Подальші дослідження можуть включати в себе оцінки ефективності даного способу для гібридних мереж, SDN мереж з декількома контролерами, а також застосування більш детальної інформації про стан мережі, наприклад оцінки таблиць потоків комутаторів SDN. Серед наступних векторів розвитку системи також можна виокремити більш досконалий спосіб статистичного аналізу мережевих даних, аніж z-оцінка.

Окрім цього, запропоноване рішення підтримує подальше масштабування і можливість розробки систем прогнозування трафіка в розробленій екосистемі – наявність застосованих компонентів підтримує швидку імплементацію систем машинного навчання та запроваджує готовий конвеєр структурованих даних для їх тренування та здійснення процесу логічного виводу.

## **Висновки до розділу 3**

У даному розділі було виконано розробку програмного продукту для моделювання системи конструювання трафіка в мережах SDN, який з'єднується із контролером та включає в себе застосування лямбда-архітектури для збору, агрегації та обробки статистичних даних про стан мережі.

Виконано огляд засобів реалізації програмного додатку, обґрунтовано та наведено перелік необхідних сервісів для роботи додатку. Зазначено мінімальні системні вимоги для коректної роботи розробленої системи, обґрунтовано можливості кросс-платформеності запропонованого рішення, а також визначено основні завдання, що мають бути вирішені програмно.

Розглянуто структуру програмного додатку, описано основні функціональні рішення, застосовані при розробці. Надана коротка змістовна інструкція по роботі з системою. Визначені основні кроки роботи розробленої системи, сформовані та пов’язані між собою головні компоненти програмного рішення. Описано процеси генерації даних, їх обробки, збереження та резервного копіювання, а також взаємодії за допомогою веб-інтерфейсу.

Виконано моделювання системи конструювання трафіка, продемонстровано роботу основних рівнів програмного рішення. Тестування системи показало ефективну роботу описаних процесів, збору та агрегації аналітичної інформації та коректне фінальне створення матриці трафіка.

Зазначено та підсумовано основні цілі та методи подальших досліджень, зокрема більш досконала система статистичної оцінки стану мережі, а також дослідження в напрямку прогнозування трафіка.

# **РОЗДІЛ 4**

# **РОЗРОБКА СТАРТАП-ПРОЕКТУ**

Ринок SDN додатків не є новим і по віку приблизно дорівнює виходу самого концепту програмно-конфігурованих мереж. Приклади систем, що працюють над задачею конструювання трафіка в SDN було наведено у розділі 1.3., але цей список не є вичерпним. Головна відмінність та ідея розробленого програмного продукту полягає в застосування новітніх технологій та підходів для вирішення задачі конструювання трафіка, особливий акцент при цьому виконується на високонавантажені SDN мережі, тому на даному етапі доцільно спланувати основні положення подальшого розвитку та спланувати стратегію дій для створення високоефективної конкурентоспроможної системи. Далі в цьому розділі будуть наведені основні маркетингові, організаційні, фінансово-економічні та комерційні аспекти запропонованої системи.

## **Маркетинговий аналіз стартап-проекту**

У рамках цього проекту основна ідея проекту: додаток прикладного рівня мережі SDN, що матиме підключення до рівня керування та реалізовуватиме збір, агрегацію, акумуляцію та аналіз інформації про стан мережі. На основі отриманої інформації виконується процес конструювання трафіка, що дозволяє підвищити надійність мережі. Додаток може бути сконфігурований для різних обсягів даних, апаратних можливостей та потреб кінцевого користувача.

* *Таблиця 4.1.*
* *Опис ідеї стартап-проекту*

|  |  |  |
| --- | --- | --- |
| Зміст ідеї | Напрямки застосування | Вигоди для користувача |
| Програмний додаток для конструювання трафіка в SDN, підтримка високонавантажених середовищ, можливість конфігурації під кінцевого користувача | Конструювання потоків трафіку в SDN мережі | Можливість якісно оцінити стан мережі в SDN мережах і визначити проблемні мережеві сегменти, що потребують змін |
| Моделювання критичних ситуацій | Моделювання можливих збоїв під час роботи мережі, дослідження роботи мережі у випадку невідкладних збоїв та визначення альтернативних шляхів руху трафіку у таких випадках |
| Прогнозування потоків трафіку | Прогнозування ділянок мережі з підвищеним навантаженням для покращення QoS мережі |

Проведемо аналіз потенційних техніко-економічних переваг ідеї та визначимо, чим ідея відрізняється від існуючих аналогів та замінників. А саме:

* визначимо перелік техніко-економічних властивостей та характеристик ідеї;
* визначимо попередні кола проектів-конкурентів, що вже існують на ринку та проведемо збір інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів;
* проведемо порівняльний аналіз показників: для власної ідеї визначимо показники, що мають а) гірші значення (W, слабкі); б) аналогічні (N, нейтральні) значення; в) кращі значення (S, сильні).

*Таблиця 4.2.*

*Визначення сильних, слабких та нейтральних характеристик ідеї проекту*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Техніко-економічні характеристики ідеї | Проекти | | | | W | N | S |
| Мій | 1 | 2 | 3 |
| 1. | Підтримка SDN | + | + | + | - |  | **+** |  |
| 2. | Підтримка високонавантажених мереж | + | - | - | - |  |  | **+** |
| 3. | Застосування  на різних  платформах | + | + | - | + |  | **+** |  |
| 4. | Масштабованість | + | + | - | + |  | **+** |  |
| 5. | Віртуалізація вузлів | - | + | + | + | **+** |  |  |
| 6. | Можливість  динамічної ремаршрутизації | - | - | + | - |  | **+** |  |
| 7. | Підтримка  прогнозування  трафіка | + | + | - | - |  |  | **+** |
| 8. | Забезпечення  цілісності даних та надійності системи | + | - | - | + |  |  | **+** |
| 9. | Високий рівень  безпеки | - | - | + | + | **+** |  |  |

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності. Згідно з таблицею 4.2, серед основних переваг розробленого продукту є підтримка роботи з високонаваантаженими мережами, підтримка модулів прогнозування трафіка та цілісність і надійність даних, на що і варто зробити акцент. Порівняно з існуючими рішеннями, головними недоліками системи є недостатній безпековий рівень системи та відсутність віртуалізації вузлів, що варто врахувати в майбутньому.

## **Технологічний аудит ідеї проекту**

*Таблиця 4.3.*

*Технологічна здійсненність ідеї проекту*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Ідея проекту | Технології її реалізації | Наявність технологій | Доступність технологій |
| 1. | Програмний  додаток SDN | Мова програмування  Python | Є в наявності | Доступна безкоштовно |
| 2. | Конструювання трафіка  за допомогою лямбда-  архітектури | Система обробки повідомлень Kafka | Є в наявності | Доступна безкоштовно |
| 3. | Сервіс обробки даних  Spark | Є в наявності | Доступна безкошнтовно |
| 4. | Розподілена файлова система HDFS | Є в наявності | Доступна безкоштовно |
| 5. | Сховище даних NoSQL Cassandra | Є в наявності | Доступна безкоштовно |
| 6. | Веб-інтерфейс | Сервіс Dash | Є в наявності | Доступна безкоштовно |
| 7. | Система прогнозування трафіка | Моделі машинного навчання для  прогнозування трафіка SDN | Немає  в наявності | Не доступні |

Згідно з описаними технологіями, необхідними для реалізації проекту, створення і розробка системи конструювання трафіка в SDN за допомого лямбда-архітектури є можливим. Усі необхідні технології, окрім системи прогнозування трафіка є в наявності та доступні безкоштовно. Система прогнозування трафіка має бути розроблена додатково і доступність варіюється в залежності від наявності даних, наявності апаратних ресурсів для тренування моделей машинного навчання та профільних спеціалістів.

## **Аналіз ринкових можливостей запуску стартап-проекту**

Здійснимо визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту. Цей етап дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів. У таблиці 4.4 наведена попередня характеристика потенційного ринку для розроблюваного стартап-проекту.

*Таблиця 4.4.*

*Попередня характеристика потенційного ринку стартап-проекту*

|  |  |  |
| --- | --- | --- |
| № | Показники стану ринку (найменування) | Характеристика |
| 1. | Кількість головних гравців | >3 |
| 2. | Загальний обсяг продаж, грн/ум.од. | 780 тис. |
| 3. | Динаміка ринку (якісна оцінка) | Зростає 35% (до 2026 р.) |
| 4. | Наявність обмежень для входу  (характер обмежень) | Фінансові затрати, наявність кваліфікованих спеціалістів |
| 5. | Специфічні вимоги до стандартизації  та сертифікації | Наявність патенту,  стандартизація від  організацій IETF та ONF |
| 6. | Середня норма рентабельності в галузі  або по ринку, % | 40 |

За результатами аналізу таблиці ринок є привабливим для входження по причинах достатнього рівня рентабельності, невеликої кількості конурентів та умови зростання ринку по прогнозах до 2026 року.

Подальший аналіз ринкових можливостей потребує характеристики потенційних груп клієнтів. Характеристика та орієнтовний перелік вимог до продукту стартап-проекту представлені в таблиці 4.5.

*Таблиця 4.5.*

*Характеристика потенційних клієнтів стартап-проекту*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Потреба, що  формує ринок | Цільова аудиторія | Відмінності  у поведінці  різних  потенційних  цільових груп  клієнтів | Вимоги до споживачів продукту |
| 1. | Стрімке впровадження SDN мереж | Провайдери  інтернет послуг, приватні компанії, що впроваджують SDN у корпоративних мережах, дата-центри,  провайдери мобільних мереж | Розміри  мереж, різні стандарти мереж  в різних сегментах, безпекові стандарти | До продукції:   * Надійність; * Безпека; * Досуп до результатів роботи системи   До компанії-постачальника:   * доступ до службових даних мережі * допомога у підтримці продукту * надання зворотнього зв’язку |
| 2. | Підвищення  надійності  SDN мереж |
| 3. | Можливість керування трафіком |
| 4. | Здатність до  роботи  з високо-  навантаженими мережами |
| 5. | Гнучкість у впровадженні |
| 6. | Масштабо-  ваність |
| 7. | Збір та обробка даних SDN | Інформаційні служби, приватні підприємства | Стандарти налашту-  вання  мережі,  варіатив-  ність даних |

Проведемо аналіз ринкового середовища, а саме: складемо таблиці факторів, що сприяють ринковому впровадженню проекту (таблиця 4.7.), та факторів, що йому перешкоджають (таблиця 4.6.).

*Таблиця 4.6.*

*Фактори загроз*

|  |  |  |  |
| --- | --- | --- | --- |
| № | Фактор | Зміст загрози | Можлива реакція компанії |
| 1. | Недостатній  рівень попиту на міжнародному ринку | Усі потенційні конкуренти та  більшість потенційних клієнтів розташовані закордоном | Проведення  маркетингової компанії закордоном  з урахуванням особливостей ринку |
| 2. | Незацікавленість потенційних клієнтів  у продукті | Недовіра до технологічної  складової продукту та автоматизації  процесів, що досі відбувалися вручну | Проведення демонстрацій, надання тестового  доступу до продукту клієнтам  на перших кроках |
| 3. | Недостатній  рівень  конкуренто-спроможності | Рішення конкурентів задовольняють базові потреби клієнтів | Вдосконалення цінової політики за покращення якості надання послуг, проведення маркетингової компанії за ключовими перевагами продукту |
| 4. | Неоднозначна  економічна  ситуація | Відсутність фінансування, недостатня кількість інвестицій | Пошук додаткових  шляхів надходження коштів, заощадження витрат |
| 5. | Вузький спектр функціоналу | Недостатній  функціонал для забезпечення потреб клієнтів | Побудова ітеративних процесів розробки, введення чітких систем планування із врахуванням ринкових потреб |
| 6. | Проблеми з отриманням патентів та стандартизації | Масштабний бюрократичний прошарок  для отримання юридичного захисту продукту | Отримання  кваліфікованої  юридичної допомоги від спеціалістів |

*Таблиця 4.7.*

*Фактори можливостей*

|  |  |  |  |
| --- | --- | --- | --- |
| № | Фактор | Зміст можливості | Можлива реакція компанії |
| 1. | Залучення інвестицій | Наявність чіткого бізнес-плану, результатів аналізу ринку та конкуруючих рішень приваблює інвесторів та дозволяє пришвидшити вихід на ринок | Збільшення штату співробітників, пришвидшення темпів розробки та прискорення виходу на самоокупність |
| 2. | Розширення клієнтської бази | Детальне дослідження ринку та проведення маркетингової  кампанії потенційним клієнтам | Використання коштів клієнтів на покращення продукту та використання позитивного клієнтського досвіду в подальших маркетингових рішеннях |
| 3. | Розширення функціоналу | Розширення функціоналу продукту для охоплення  більшого ринкового сегменту | Ітеративний аналіз ринку для пошуку нових можливостей та приваблення клієнтів |
| 4. | Партнертсво з великими  гравцями ринку  послуг IaaS і PaaS | Надання послуг лідерами серед провайдерів послуг  SDN | Збільшення клієнтської бази, використання позитивного клієнтського досвіду, додаткове залучення коштів, отримання інсайтів щодо ринкових трендів |
| 5. | Вихід на міжнародний  ринок | Надання доступнішого продукту для представників міжнародного ринку | Забезпечення лідерських позицій у ринковому сегменті, використання позитивного  клієнтського досвіду, розширення функціоналу згідно з потребами міжнародного ринку |

Далі проведемо аналіз пропозиції: визначемо риси конкуренції на ринку (таблиця 4.8.).

*Таблиця 4.8.*

*Ступеневий аналіз конкуренції на ринку*

|  |  |  |
| --- | --- | --- |
| Особливості конкурентного середовища | В чому проявляється дана характеристика | Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною) |
| Тип конкуренції:  *олігополія* | На ринку присутні декілька компаній, що надають подібні рішення | Пропозиція вигідніших умов впровадження та користування, розробка ключового функціоналу, відсутнього у конкурентів, проведення маркетингових кампаній з акцентами на ключових відмінностях продукту, надання тестового періоду користування |
| За рівнем конкурентної боротьби:  *глобальний* | Існуючі рішення можуть бути впроваджені по всьому світу | Одразу організувати вихід на міжнародну арену та запровадити стратегії відповідно до критеріїв світового ринку |
| За галузевою ознакою:  *внутрішньо-*  *галузева* | Конкуренція в галузі мереж SDN | Розробка продукту яка вирішує обмежену кількість задач, але краще, ніж інші рішення |
| *Товарно-родова*  конкуренція | Продукти-замінники можуть виконувати подібні функції | Створення більш якісної та кращої продукції |
| *Нецінова*  конкуренція | Конкуренція створюється за допомогою вдосконалення якості продукції | Продукт має постійно вдосконалюватись та покращувати свої характеристики |
| *Не марочна*  інтенсивність | Споживачів цікавить більше характеристики продукту, ніж під яким брендом випущено продукт | Покращити якісні характеристики продукту порівняно з конкурентними |

Проведемо більш детальний аналіз умов конкуренції в галузі за моделлю 5 сил М. Портера.

М. Портер вирізняє п’ять основних факторів, що впливають на привабливість вибору ринку з огляду на характер конкуренції. Це:

* конкуренти, що вже є в галузі;
* потенційні конкуренти;
* наявність товарів-замінників;
* постачальники, що конкурують за ринкову владу;
* споживачі (аналогічно).

Сильні позиції продукту за кожним з факторів означають її можливість забезпечити необхідні темпи обороту капіталу та її здатність впливати на інших агентів ринку, диктуючи їм власні умови співпраці. Характеристики факторів моделі відрізняються для різних галузей та змінюються з часом.

*Таблиця 4.9.*

*Аналіз конкуренції в галузі за М. Портером*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Складові аналізу | Прямі  конкуренти  в галузі | Потенційні конкуренти  в галузі | Постачаль-  ники | Клієнти | Продукти-замінники |
| Canopus  Networks,  COSGrid  Networks,  flexiWAN | Zeeta  Networks,  Trustgrid | Cisco,  Juniper, VMWare,  IVM | Провайдери послуг,  приватні компанії,  дата-центри | SDN-TE,  Packet  Designs  Explorer |
| Висновки | Присутня конкуренція на міжнародному ринку. Конкуренція незначна, кожна з компаній зосереджується на конкретному ринковому сегменті. | Компанії займаються розробкою  та  підтримкою рішень  у сфері SDN  і за потреби можуть  випустити схожий продукт | Постачаль  никами є міжнародними лідерами у  сфері мережевого обладнання,  показник залучення є низьким | Більшість клієнтів давно присутні на ринку і мають контакти з постачальниками та можуть бути зацікавлені у продукті | Є декілька подібних рішення для конструювання трафіка, які є платними та покривають частину функціоналу |

За результатами таблиці робимо висновок, що робота на ринку з огляду на ситуацію є можливою, конкуренція на даному етапі є розрідженою, постачальники не зацікавлені у наданні подібних рішень, а існуючі проекти покривають лише частину запропонованого функціоналу.

На основі аналізу конкуренції (таблиця 4.9.) а також із урахуванням характеристик ідеї проекту (таблиця 4.2.), вимог споживачів до товару (таблиця 4.5.) та факторів маркетингового середовища (таблиці 4.6.-4.7.) визначимо та обґрунтуємо перелік факторів конкурентоспроможності (таблиця 4.10.).

*Таблиця 4.10.*

*Обґрунтування факторів конкурентоспроможності*

|  |  |  |
| --- | --- | --- |
| № | Фактор конкурентоспроможності | Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим) |
| 1. | Унікальність функціоналу | Надання вичерпного переліку унікальних функціональних можливостей для ви2рішення основних задач конструювання трафіка |
| 2. | Кроссплатформеність | Незалежність від архітектурних характеристик мережі; можливість швидко розгорнути систему на будь-якій наявній інфраструктурі, враховуючи мінімальні системні вимоги |
| 3. | Надійність та масштабованість | Продукт є надійним та дозволяє масштабуватись при зростанні топології мережі, що є особливо вигідним для клієнтів, які розпочинають побудову власної мережі для зменшення time-to-market. |

За визначеними факторами конкурентоспроможності проведемо аналіз сильних та слабких сторін стартап-проекту.

*Таблиця 4.11.*

*Порівняльний аналіз сильних та слабких сторін стартап-проекту*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Фактор конкуретноспроможності | Бали  1-20 | Рейтинг товарів-конкурентів у порівнянні з продуктом | | | | | | |
| -3 | -2 | -1 | 0 | +1 | +2 | +3 |
| 1. | Унікальність функціоналу | 17 |  | ✓ |  |  |  |  |  |
| 2. | Кроссплатформеність | 15 |  |  |  | ✓ |  |  |  |
| 3. | Надійність та масштабованість | 15 |  |  | ✓ |  |  |  |  |

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (таблиця 4.12.) на основі виділених ринкових загроз та можливостей, а також сильних та слабких сторін.

*Таблиця 4.12.*

*SWOT- аналіз стартап-проекту*

|  |  |
| --- | --- |
| **Сильні сторони:**   1. Унікальність функціоналу 2. Надійність та масштабованість 3. Сучасні технологічні підходи 4. Орієнтація на міжнародний ринок 5. Вирішення чітко сформованої клієнтської потреби | **Слабкі сторони:**   1. Наявність вже впроваджених конкуруючих рішень 2. Є потреба у часі для виходу на ринок 3. Потреба у постійному надходженні коштів для підтримки та вдосконалення продукту |
| **Можливості:**   1. Партнерство зі світовими мережевими лідерами 2. Приваблення інвестицій 3. Вдосконалення продукту на основі клієнтського досвіду 4. Розширення клієнтської бази | **Загрози:**   1. Швидший або успішніший вихід на ринок нового конкуруючого рішення 2. Відсутність попиту на продукт |

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок (таблиця 4.13.)

*Таблиця 4.13.*

*Альтернативи ринковоговпровадження стартап-проекту*

|  |  |  |  |
| --- | --- | --- | --- |
| № | Альтернатива ринкової поведінки | Ймовірність отримання ресурсів | Терміни реалізації |
| 1. | Відкриття проекту та розповсюдження в рамках opensource ліцензії | Низька. Є можливість продажу компанії у випадку зацікавленості з боку великих компаній | Одразу після розробки |
| 2. | Розповсюдження проекту за допомогою науково-публічної роботи: конференції, виступи і т.п. | Висока, оскільки публічними заходами можна поширити ідею проекту та завести знайомства з потенційними інвесторами та клієнтами | 1 рік |
| 3. | Участь у тендерах і укладення договорів з державними установами | Середня, так як на це впливають економічні, політичні, особисті мотиви і не можна гарантувати абсолютну прозорість та добросовісну конкуренцію за таких умов | * 1. – 2 роки |

## **Розробка ринкової стратегії стартап-проекту**

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 4.14).

*Таблиця 4.14.*

*Вибір цільових груп потенційних споживачів*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Опис профілю цільової групи потенційних клієнтів | Готовність споживачів сприйняти продукт | Орієнтовний попит в межах цільової групи | Інтенсивність конкуренції в сегменті | Простота входу в сегмент |
| 1. | Провайдери інтернет-послуг | Не готові сприйняти продукт | Низький | Середня | Проста |
| 2. | Приватні компанії | Готові сприйняти продукт | Середній | Висока | Проста |
| 3. | Провайдери мобільних послуг | Не готові сприйняти продукт | Низький | Низька | Складна |
| 4. | Дата-центри | Готові сприйняти продукт | Високий | Середня | Проста |
| 5. | Інформцційні служби | Готові сприйняти продукт | Низька | Низька | Проста |
| Обрані цільові групи: дата-центри, приватні компанії. Має бути використана стратегія диференційованого маркетингу. | | | | | |

Згідно з аналізом потенційних комерційних груп споживачів на запропонований продукт, було обрано стратегію диференційованого маркетингу, оскільки компанія працює водночас із кількома сегментами, з розробкою окремих програм для ринкового впливу.

*Таблиця 4.15.*

*Визначення базової стратегії розвитку*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Обрана альтернатива розвитку проекту | Стратегія охоплення ринку | Ключові конкурентоспроможі позиції відповідно до обраної альтернативи | Базова стратегія розвитку |
| 1. | Стратегія зростання | Вибірковий розподіл | Нарощування функціоналу, внутрішнє розширення, виробництво нових продуктів | Стратегія диференційованого маркетингу |

* *Таблиця 4.16.*

*Визначення базової стратегії конкурентної поведінки*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Чи є проект «першопрохідцем» на ринку? | Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів? | Чи буде компанія копіювати основні характеристики товару конкурента, і які? | Стратегія конкурентної поведінки |
| 1. | Ні | Комібінований підхід: є ніші, які не використовують подібні продукти та завдяки наявним перевагам можна забирати існуючих клієнтів | Ні, компанія може використовувати наявний функціонал і вдосконалювати його згідно з клієнтським досвідом | Стратегія виклику лідера |

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та вимог до продукту (табл. 4.5.), а також в залежності від обраної базової стратегії розвитку (табл. 4.15) та стратегії конкурентної поведінки (табл. 4.16) розробляється стратегія позиціонування (табл. 4.17). що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

*Таблиця 4.17.*

*Визначення стратегії позиціонування*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Вимоги до товару цільової аудиторії | Базова стратегія розвитку | Ключові конкуренто-спроможні позиції стартап-проекту | Вибір асоціацій, які мають сформувати комплексну позицію власного проекту |
| 1. | Вирішення проблеми конструювання трафіка, можливість прогнозування трафіка, надійність, стабільність, підтримка  та зворотній зв’язок | Стратегія диференційова-ного маркетингу | Унікальність функціоналу,  кросс  платформеність,  надійність,  масштабованість | Унікальність  Надійність  Клієнто-орієнтованість |

У результаті отримали узгоджену систему рішень щодо ринкової поведінки стартап-компанії, що визначатиме напрями роботи стартап-компанії на ринку.

## **Розробка маркетингової програми стартап-проекту**

Першим кроком є формування маркетингової концепції продукту, який отримає споживач (таблиця 4.18.).

*Таблиця 4.18.*

*Визначення ключових переваг концепції потенційного товару*

|  |  |  |  |
| --- | --- | --- | --- |
| № | Потреба | Вигода, яку пропонує товар | Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити) |
| 1. | Вирішення проблеми конструювання трафіка | Кроссплатформене, архітектурно-незалежне рішення з підтримкою високонавантажених мереж | Використання сучасних технологій та підходів, робота з високонавантаженими мережами, надійність, безпека. |
| 2. | Можливість прогнозування трафіка | Можливість інтеграції системи прогнозування трафіка із запропонованим рішенням | Інтеграція системи прогнозування трафіка, швидкість налаштування та розгортання |
| 3. | Підтримка та зворотній зв’язок | Простота у використанні та налаштуванні, менші витрати часу на підтримку | Більш дружній інтерфейс, допомога у налаштуванні та підтримці сервісу |

Далі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання.

*Таблиця 4.19.*

*Опис трьох рівнів моделі товару*

|  |  |  |
| --- | --- | --- |
| *Рівні товару* | *Сутність та складові* | |
| І. Товар за задумом | Кроссплатформений програмний додаток для конструювання трафіка в програмно-конфігурованих мережах з можливістю інтеграції систем прогнозування трафіка та їх конфігурації. | |
| ІІ. Товар у реальному виконанні | Властивості/характеристики | Розмір |
| 1. Сервіс обробки повідомлень | 400MB |
| 1. Сервіс обробки та аналізу даних | 1GB |
| 1. Розподілена файлова система | 1GB |
| 1. Сховище даних | 225+MB |
| Якість: внутрішнє тестування, логування виконання, система підтримки | |
| Пакування: SaaS-концепти, створення API та дашбордів для клієнтів. | |
| Марка: назва організації-розробника + назва товару | |
| ІІІ. Товар із підкріпленням | До продажу: ліцензія на використання | |
| Після продажу: надання доступу до API | |
| За рахунок чого потенційний товар буде захищено від копіювання:  Комплексне поєднання властивостей і характеристик, закладене на другому та третьому рівнях товару. | | |

*Таблиця 4.20.*

*Визначення меж встановлення ціни*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Рівень цін на товари-замінники | Рівень цін на товари-аналоги | Рівень доходів цільової групи споживачів | Верхня та нижня межі становлення ціни на товар/послугу |
| 1. | 1000$ - 5000$/міс. | 1000$ - 5000$/міс. | >50000$/міс. | 1000$-2000$/міс. |

*Таблиця 4.21.*

*Формування системи збуту*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Специфіка закупівельної поведінки цільових клієнтів | Функції збуту, які має виконувати постачальник товару | Глибина каналу збуту | Оптимальна система збуту |
| 1. | Покупка ліцензії на продукт або договір про надання послуг без передачі ПО до замовника | Надання доступу до API, розгортання системи на архітектурі клієнта | Канал нульового рівня | Інтернет, тендерні торги |

*Таблиця 4.22.*

*Концепція маркетингових комунікацій*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Специфіка поведінки цільових клієнтів | Канали комунікацій, якими користуються цільові клієнти | Ключові позиції, обрані для позиціонування | Завдання рекламного повідомлення | Концепція рекламного звернення |
| 1. | Купують товар на вимогу | Наукові кола, Інтернет, соціальні мережі, форуми | Інтернет-маркетинг | Презентація товару, унікальності і якості. | Пояснення функцій та можливо-стей продукту. |

**Висновки до розділу 4**

Четвертий розділ присвячено розробці стартап-проекту на основі розробленої системи, що реалізує спосіб конструювання трафіка в програмно-конфігурованих мережах на основі лямбда-архітектури.

Для дослідження в рамках розробки стартап-проекту було виконано наступні завдання:

1. Наведено загальний опис ідеї стартап-проекту, описано функціонал, визначено основних конкурентів та порівняно функціонал існуючих продуктів для визначення переваг розроблюваного продукту над конкурентами.
2. Проаналізовано сучасні ринкові можливості для запуску проекту та успішного виходу на ринок. Виявлено можливості та загрози стартап проекту та сплановані дії щодо їх запобігання.
3. Порівняно перспективи запуску в порівнянні з конкурентами, встановлено план та ринкову стратегію розвитку продукту, визначено цільові групи та способи просування проекту.
4. Розроблено маркетингову програму для просування продукту на ринку, описано способи та основні канали збуту, визначно пріорітетні цільові групи та маркетингові повідомлення для розширення клієнтської бази.

В підсумку, отримано стартап-проект для запуску розробленого програмного продукту на базі дипломного проекту для виходу на ринок.

# **ВИСНОВКИ**

Магістерська робота присвячена розробці способу конструювання трафіка в SDN мережах на основі лямбда-архітектури.

У роботі було виконано дослідження сучасного стану програмно-конфігурованих мереж, визначено основні аспекти та вектори розвитку, а також виконано технологічний огляд протоколу OpenFlow. Розглянуто головні види та відкриті задачі конструювання трафіка в програмно-конфігурованих мережах, виконано огляд існуючих рішень. Виявлено проблему, що особливо актуальна для високонавантажених мереж: продукується значна кількість нових даних, стан мережі має тенденцію постійно змінюватись, тому процес конструювання трафіка потребує швидкої реакції на події, натомість традиційні рішення демонструють занизькі показники швидкості роботи та є слабкомасштабованими. Запропоновано спосіб вирішення даної проблеми із застосуванням сучасних підходів Big Data, а саме імплементації лямбда-архітектури, а також розглянуто альтернативні підходи.

Сформовано основні завдання, які має вирішувати спосіб конструювання трафіка. Сформульовано основні етапи створення матриці трафіка. Визначено процес формування множин, які мають бути отримані з інвентарних даних, та процесів обробки даних для обчислення матриці трафіка. Зазначено аналітичні підходи для опрацювання отриманої статистики та реалізації способу конструювання трафіка. Окрім цього, описано основні етапи конвеєру інвентарних даних та даних про стан мережі. Виокремлено та сформовано процес формування повідомлень після отримання даних інфраструктурного рівня контролером. Розроблено спосіб обробки та агрегації даних, заснований на існуючих концепціях лямбда-архітектури, а саме потоковий та пакетний шляхи обробки, зберігання даних до сховища та резервного копіювання з використанням розподіленої файлової системи.

За проведеною розробкою реалізовано модель способу конструювання трафіка в SDN. Виконано технологічний огляд засобів реалізації, обґрунтовано вибір мови програмування, архітектури та сервісів. Обрано топологію, за якою реалізовано модель мережі. Створено процес генерації мережевих даних для емуляції роботи системи. Виконано опис структури програмного додатку та реалізації його компонент. Надано проміжні результати процесів формування повідомлень, їх збереження, обробки повідомлень швидкісним та пакетним шарами лямбда-архітектури, описано призначення та методи задач, що реалізуються та виконуються на кожному етапі. Також надано опис візуального інтерфейсу системи обробки даних та показано результуючу матрицю трафіка.

Завдяки технологічним та концептуальним особливостям даний спосіб може бути застосований до мереж як з високою інтенсивністю трафіка, так і до збалансованих. Також запропоноване рішення є масштабованим, тому з практичної точки зору, спосіб є достатньо універсальним та стабільним для застосування у виробничому середовищі. Окрім цього, методи роботи з великими даними дозволяють отримувати очікувану матрицю трафіка в режимі реального часу, що з практичної точки зору дозволяє вирішувати задачу конструювання трафіка в мережах SDN більш ефективно.

# **СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. H. Farhady, H. Lee, and A. Nakao, “Software-Defined Networking: A survey,” Computer Networks, vol. 81, , 2015.
2. D. Kreutz, F. M. V. Ramos, P. E. Ver ́ısssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking : A Comprehensive Survey,” Proceedings of the IEEE, vol. 103, 2015.
3. Software-Defined Networking (SDN) Definition [Електронний ресурс] // Open Networking Foundation – Режим доступу до ресурсу: https://www.opennetworking.org/sdn-definition/
4. I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, “A roadmap for traffic engineering in software defined networks,” Computer Networks, vol. 71, 2014.
5. Understanding the SDN Architecture - SDN Control Plane & SDN Data Plane [Електронний ресурс] – Режим доступу до ресурсу: https://www.sdxcentral.com/networking/sdn/definitions/inside-sdn-architecture/
6. SDN vs Traditional Networking: Which Leads the Way? [Електронний ресурс] – Режим доступу до ресурсу: http://www.chinacablesbuy.com/sdn-vs-traditional-networking-which-leads-the-way
7. Концепция и преимущества программно-определяемых сетей SDN [Електронний ресурс] // Infocom, 2016. – Режим доступу до ресурсу: http://infocom.uz/2016/10/27/koncepciya-i-preimushhestva-programmno-opredelyaemyx-setej-sdn/
8. SDN Architecture Overview [Електронний ресурс] // Open Networking Foundation – Режим доступу до ресурсу: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf
9. Mekky, Hesham, Fang Hao, Sarit Mukherjee, Zhi-Li Zhang, and T. V. Lakshman. "Application-aware data plane processing in SDN." In Proceedings of the third workshop on Hot topics in software defined networking, pp. 13-18. 2014.
10. Benamrane, Fouad & Ben Mamoun, Mouad & Redouane, Benaini. Performances of OpenFlow-Based Software-Defined Networks: An overview. Journal of Networks. 10. 329-337. 10.4304/jnw.10.6.329-337, 2015.
11. Advantages and Disadvantages of SDN. Drawbacks & Benefits of SDN [Електронний ресурс] – Режим доступу до ресурсу: https://www.hitechwhizz.com/2021/06/5-advantages-and-disadvantages-drawbacks-benefits-of-sdn.html
12. OpenFlow Switch Specification // Open Networking Foundation [Електронний ресурс] – Режим доступу до ресурсу: https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf
13. SDN Architecture Overview [Електронний ресурс] // Open Networking Foundation – Режим доступу до ресурсу: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf
14. A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, “Software-defined networking: Challenges and research opportunities for future internet,” Computer Networks,vol. 75, 2014.
15. OpenFlow Switch Specification [Електронний ресурс] // Open Networking Foundation – Режим доступу до ресурсу: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.4.0.pdf
16. Лихачев В.А. Программно-конфигурируемые сети на основе протокола OpenFlow [Електронний ресурс] // Вістник ВГУ, Серія: Системний аналіз і інформаційні технології №1, 2014. – Режим доступу до ресурсу: http://www.vestnik.vsu.ru/pdf/analiz/2014/01/2014-01-10.pdf
17. P. Goransson, C. Black, and T. Culver, Software Defined Networks, Second Edition: A Comprehensive Approach, 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.
18. OpenFlow – Basic Concepts and Theory [Електронний ресурс] // Overlaid, 2017. - Режим доступу до ресурсу: https://overlaid.net/2017/02/15/openflow-basic-concepts-and-theory/
19. An Introduction to Big Data Concepts [Електронний ресурс] // Sagara Technology Idea Lab, 2020. - Режим доступу до ресурсу: https://sagaratechnology.medium.com/ig-an-introduction-to-big-data-concepts-c4cac70a7f03
20. S. J. Samuel, K. Rvp, K. Sashidhar, and C. R. Bharathi, “a Survey on Big Data and Its Research Challenges,” ARPN Journal of Engineering and Applied Sciences, vol. 10, no. 8, 2015.
21. D. Laney, “3D Data Management: Controlling Data Volume, Velocity, and Variety,” Application Delivery Strategies by META Group Inc., vol. 949, p. 4, 2001.
22. Big Data [Електронний ресурс] // IT Enterprise, 2018. - Режим доступу до ресурсу: https://www.it.ua/ru/knowledge-base/technology-innovation/big-data-bolshie-dannye
23. Chebbi, Imen & Boulila, Wadii & Farah, Imed. Big Data: Concepts, Challenges and Applications. 9330. 10.1007/978-3-319-24306-1\_62, 2015.
24. A. D. Meshram, A. S. Kulkarni, and S. S. Hippargi, “Big Data Analytics using Real-Time Architecture,” International Journal of Latest Trendes in Engineering and Technology(IJLTET), vol. 6, no. 4, 2016.
25. K. Grolinger, M. Hayes, W. A. Higashino, A. L’Heureux, D. S. Allison, and M. A. Capretz, “Challenges for MapReduce in Big Data,” in 2014 IEEE World Congress on Services, pp. 182–189, IEEE, Jun. 2014.
26. J. A. Stankovic, S. H. Son, and J. Hansson, “Misconceptions About Real-Time Databases,” Computer, vol. 32, 1999.
27. N. Marz and J. Warren, Big Data - Principles and Best Practices of Scalable Real-Time Data Systems. Manning Publications Co., 2015.
28. Kiran, Mariam & Murphy, Peter & Monga, Inder & Dugan, Jon & Baveja, Sartaj. Lambda architecture for cost-effective batch and speed big data processing. 2785-2792. 10.1109/BigData.2015.7364082, 2015.
29. Hasani, Zirije, Margita Kon-Popovska, and Goran Velinov. "Lambda architecture for real time big data analytic." ICT Innovations: 133-143p., 2014.
30. Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, “Traffic engineering in software-defined networking: Measurement and management,” IEEE Access, vol. 4, 2016.
31. I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, ‘‘A roadmap for traffic engineering in SDN-OpenFlow networks,’’ Comput. Netw., vol. 71, pp. 1–30, 2014.
32. Khoa Truong Dinh, Sławomir Kukliński, Tomasz Osiński & Jacek Wytrębowicz, “Heuristic traffic engineering for SDN”, Journal of Information and Telecommunication, p.251-266, 2020.
33. D.Awduche, A. Chiu, A.Elwalid, I.Widjaja “Overview and Principles of Internet Traffic Engineering”, RFC 3784, 2002.
34. Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart and Amin Vahdat “B4: Experience with a globally-deployed software defined WAN”, ACM SIGCOMM Comp. Commun. Rev., vol. 43, no. 4, pp. 3–14, 2013.
35. M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks”, in Proc. 7th USENIX Symp. on Netw. Syst. Design & Implemen, vol. 10, pp. 19–19, 2010.
36. A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, “DevoFlow: scaling flow management for highperformance networks”, ACM SIGCOMM Comp. Commun. Rev., vol. 41, no. 4, pp. 254–265, 2011.
37. T. Benson, A. Anand, A. Akella, M. Zhang, “MicroTE: Fine grained traffic engineering for data centers”, in Proc. 7th Conf. on Emerg. Networking Experim. & Technol. Co-NEXT’11, Tokyo, Japan, p.8, 2011.
38. Z. A. Qazi et al., “Application-awareness in SDN”, ACM SIGCOMM Comp. Commun. Rev., vol. 43, no. 4, pp. 487–488, 2013.
39. N. L. M. Van Adrichem, C. Doerr, and F. A. Kuipers, “OpenNetMon: Network monitoring in OpenFlow software-defined networks,” IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, 2014.
40. M. Malboubi, S.-M. Peng, P. Sharma, and C.-N. Chuah, “A learning-based measurement framework for traffic matrix inference in software defined networks,” Computers & Electrical Engineering, vol. 66, Feb. 2018.
41. W. Queiroz, M. A. Capretz, and M. Dantas, “An approach for SDN traffic monitoring based on big data techniques,” Journal of Network and Computer Applications, vol. 131, Apr. 2019.

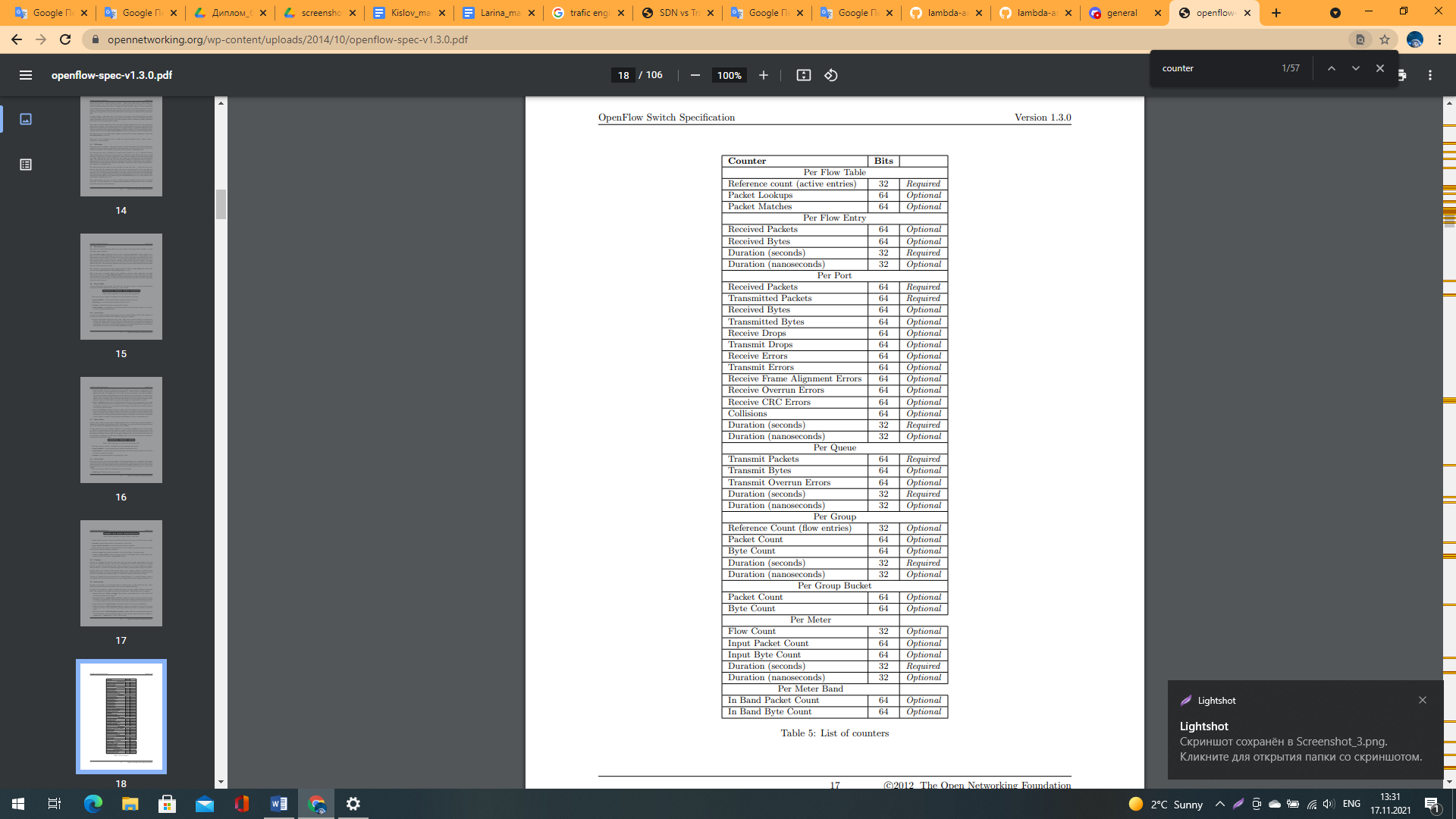
﻿ДОДАТОК А

Спосіб конструювання трафіка в SDN мережах на основі лямбда-архітектури

Список лічильників комутаторів OpenFlow

Аркушів 1

Київ – 2021



ДОДАТОК Б

Спосіб конструювання трафіка в SDN мережах на основі лямбда-архітектури

ЛІСТИНГ ПРОГРАМНГО КОДУ

Аркушів 15

Київ – 2021

**import** **networkx** **as** **nx**

**import** **numpy** **as** **np**

**from** **json** **import** load

**class** **NetworkGenerator**:

**def** \_\_init\_\_(self, host\_to\_switch\_links, switch\_to\_switch\_links):

self.\_host\_to\_switch\_links = host\_to\_switch\_links

self.\_switch\_to\_host\_links = self.\_revert\_l1\_links()

self.\_switch\_to\_switch\_links = switch\_to\_switch\_links

self.\_network = self.\_generate\_network\_graph()

self.\_paths = self.\_generate\_paths()

self.\_ports = self.\_generate\_ports()

self.\_lp = self.\_generate\_lp()

@property

**def** network(self):

*"""Getter network"""*

**return** self.\_network

@property

**def** hosts(self):

*"""Getter hosts"""*

**return** list(self.\_host\_to\_switch\_links.keys())

@property

**def** paths(self):

*"""Getter paths"""*

**return** self.\_paths

@property

**def** ports(self):

*"""Getter ports"""*

**return** self.\_ports

@property

**def** switches(self):

*"""Getter ports"""*

**return** list(self.\_switch\_to\_host\_links.keys())

@property

**def** lp(self):

**return** self.\_lp

**def** \_revert\_l1\_links(self):

switch\_to\_host\_links = {}

**for** h **in** self.hosts:

new\_key = self.\_host\_to\_switch\_links[h]

**if** new\_key **in** switch\_to\_host\_links:

**if** **not** isinstance(switch\_to\_host\_links[new\_key], list):

switch\_to\_host\_links[new\_key] = [switch\_to\_host\_links[new\_key]]

switch\_to\_host\_links[new\_key].append(h)

**else**:

switch\_to\_host\_links[new\_key] = h

**return** switch\_to\_host\_links

**def** \_generate\_ports(self):

ports = {}

**for** switch **in** self.\_switch\_to\_host\_links.keys():

ports[switch] = {}

**for** i, host **in** enumerate(self.\_switch\_to\_host\_links[switch]):

ports[switch][f"eth{i}"] = host

**for** i, uplink **in** enumerate(self.\_switch\_to\_switch\_links[switch]):

ports[switch][f"eth{28-i}"] = uplink

**return** ports

**def** \_generate\_network\_graph(self):

network = nx.Graph()

network.add\_nodes\_from(self.\_host\_to\_switch\_links.keys())

network.add\_nodes\_from(self.\_switch\_to\_switch\_links.keys())

network.add\_edges\_from(self.\_host\_to\_switch\_links.items())

**for** switch, links **in** self.\_switch\_to\_switch\_links.items():

**if** isinstance(links, list):

**for** link **in** links:

network.add\_edge(switch, link)

**else**:

network.add\_edge(switch, links)

**for** switch, links **in** self.\_switch\_to\_host\_links.items():

**if** isinstance(links, list):

**for** link **in** links:

network.add\_edge(switch, link)

**else**:

network.add\_edge(switch, links)

**return** network

**def** \_generate\_paths(self):

P = {}

**for** s **in** self.hosts:

target\_path = {}

**for** t **in** self.hosts:

**if** s == t:

**continue**

paths = nx.all\_simple\_paths(self.network, s, t)

target\_path[t] = tuple(paths)

P[s] = target\_path

**return** P

**def** \_generate\_lp(self):

LP = {}

**for** switch **in** self.switches:

**for** port **in** self.ports[switch]:

*# Get link for this port*

link = self.ports[switch][port]

port\_paths = []

**for** source, paths **in** self.paths.items():

**for** dest, paths **in** paths.items():

**for** path **in** paths:

**if** link **in** path **and** switch **in** path:

port\_paths.append(path)

*# Clear duplicates*

clear\_paths = []

**for** path **in** port\_paths:

**if** path **in** clear\_paths **or** path[::-1] **in** clear\_paths:

**continue**

**else**: clear\_paths.append(path)

LP[f"{switch}:{port}"] = clear\_paths

**return** LP

**def** get\_random\_host(self, seed=0):

**if** seed:

np.random.seed(seed)

**return** np.random.choice(self.hosts)

**def** get\_random\_path(self, source, target=None, seed=0):

**if** seed:

np.random.seed(seed)

targets = self.paths[source]

**if** target:

**if** target == source:

**raise** **ValueError**(f'Target and source values must be different. Given {source} and {target}')

**elif** target **not** **in** self.hosts:

**raise** **ValueError**(f'Unknown target value passed')

**else**:

possible\_targets = self.hosts.copy()

possible\_targets.remove(source)

target = np.random.choice(possible\_targets)

paths = targets[target]

rand\_ind = np.random.randint(len(paths))

**return** source, target, paths[rand\_ind]

**from** **datetime** **import** datetime

**import** **json**

**class** **SDNData**:

**def** \_\_init\_\_(self,

switch\_id : str,

port\_id : str,

timestamp : str,

byte\_transmitted : int,

byte\_received : int,

packets\_transmitted : int,

packets\_received : int,

):

self.\_switch\_id = switch\_id

self.\_port\_id = port\_id

self.\_byte\_transmitted = byte\_transmitted

self.\_byte\_received = byte\_received

self.\_packets\_transmitted = packets\_transmitted

self.\_packets\_received = packets\_received

self.\_timestamp = timestamp

@property

**def** switch\_id(self):

*# Getter switch\_id*

**return** self.\_switch\_id

@property

**def** port\_id(self):

*# Getter switch\_id*

**return** self.\_port\_id

@property

**def** timestamp(self):

*# Getter timestamp*

**return** self.\_timestamp

@property

**def** bt(self):

*# Getter byte transmitted*

**return** self.\_byte\_transmitted

@property

**def** br(self):

*# Getter byte transmitted*

**return** self.\_byte\_received

@property

**def** pt(self):

*# Getter packets transmitted*

**return** self.\_packets\_transmitted

@property

**def** pr(self):

*# Getter packets received*

**return** self.\_packets\_received

**def** serialize(self):

data = {

"timestamp" : self.timestamp,

"switchid" : self.switch\_id,

"portid" : self.port\_id,

"bytetransmitted" : self.bt,

"bytereceived" : self.br,

"packetstransmitted" : self.pt,

"packetsreceived" : self.pr

}

**return** json.dumps(data, indent=4, sort\_keys=True, default=str)

**def** \_\_str\_\_(self):

**return** self.serialize()

**from** **time** **import** sleep

**from** **json** **import** load, dump

**from** **kafka** **import** KafkaProducer

**import** **configparser**

**import** **logging**

**import** **random**

**from** **datetime** **import** datetime

**import** **sys**

**from** **network** **import** NetworkGenerator

**from** **sdn\_data** **import** SDNData

**class** **DataProducer**:

**def** \_\_init\_\_(self, config, network\_config ):

bootstrap\_servers = config.get('Kafka', 'metadata.broker.list')

self.network\_generator = NetworkGenerator(network\_config['l1'], network\_config['l2'])

self.producer = KafkaProducer(

bootstrap\_servers=bootstrap\_servers,

*# value\_serializer=lambda x:dumps(x).encode('utf-8')*

)

self.topic = config.get('Kafka', 'kafka.topic')

**def** send\_events(self, logger):

**while** True:

events = self.\_generate\_events()

**for** event **in** events:

logger.debug("Message to be sent: **%s**", event)

self.producer.send(self.topic, event.serialize().encode('utf-8'), )

sleep(random.random()\*3+1)

**def** \_generate\_events(self):

event\_list = []

**for** switch **in** self.network\_generator.switches:

**for** port **in** self.network\_generator.ports[switch]:

timestamp = str(datetime.timestamp(datetime.now()))

**if** int(port[3:]) > 24: *# uplinks:*

byte\_transmitted = random.randint(2 \*\* 10, 2 \*\* 16)

byte\_received = random.randint(2 \*\* 10, 2 \*\* 16)

**else**:

byte\_transmitted = random.randint(2 \*\* 1, 2 \*\* 14)

byte\_received = random.randint(2 \*\* 1, 2 \*\* 14)

packets\_transmitted = byte\_transmitted // 2 \*\* 3 + random.randint(-2 \*\* 5, 2 \*\* 5)

packets\_received = byte\_received // 2 \*\* 3 + random.randint(-2 \*\* 5, 2 \*\* 5)

event = SDNData(

switch,

port,

timestamp,

byte\_transmitted,

byte\_received,

packets\_transmitted,

packets\_received

)

event\_list.append(event)

**return** event\_list

**def** save\_network\_inf(self, output\_path):

paths = self.network\_generator.lp

**with** open(output\_path, "w") **as** output:

dump(paths, output)

**if** \_\_name\_\_ == '\_\_main\_\_':

PATH\_OUTPUT = "../sdn\_spark\_processor/network\_paths.json"

logger = logging.getLogger('sdn-kafka')

logger.setLevel(logging.INFO)

handler = logging.StreamHandler(sys.stdout)

handler.setLevel(logging.DEBUG)

formatter = logging.Formatter('**%(asctime)s** - **%(name)s** - **%(levelname)s** - **%(message)s**')

handler.setFormatter(formatter)

logger.addHandler(handler)

NETWORK\_CONFIG\_PATH = 'network\_conf.json'

PROPERTIES\_PATH = 'sdn-kafka.properties'

config = configparser.RawConfigParser()

config.read(PROPERTIES\_PATH)

**with** open(NETWORK\_CONFIG\_PATH, 'r') **as** f:

network\_config = load(f)

producer = DataProducer(config, network\_config)

producer.save\_network\_inf(PATH\_OUTPUT)

logger.info('Sending events...')

producer.send\_events(logger)

**from** **json** **import** load

**from** **pyspark.sql** **import** SparkSession, DataFrame

**from** **pyspark.sql.types** **import** \*

**from** **pyspark.sql.functions** **import** \*

**class** **StreamProcessor**:

**def** \_\_init\_\_(self, config):

self.\_config = config

@staticmethod

**def** \_read\_kafka\_config(config):

brokerlist = config.get('Kafka', 'brokerlist')

topic = config.get('Kafka', 'topic')

reset\_type = config.get('Kafka', 'resetType')

**return** brokerlist, topic, reset\_type

@staticmethod

**def** \_read\_spark\_config(config):

app\_name = config.get('Spark', 'app.name')

master = config.get('Spark', 'master')

checkpoint\_dir = config.get('Spark', 'checkpoint.dir')

log\_level = config.get("Spark", "log.level")

**return** app\_name, master, checkpoint\_dir, log\_level

@staticmethod

**def** \_read\_cassandra\_config(config):

host = config.get("Cassandra", "host")

port = config.get("Cassandra", "port")

username = config.get("Cassandra", "username")

password = config.get("Cassandra", "password")

**return** host, port, username, password

**def** \_create\_session(self, app\_name, master, checkpoint\_location, log\_level, cassandra\_host, cassandra\_port, username, password):

self.spark = SparkSession \

.builder \

.appName(app\_name) \

.master(master) \

.config("spark.cassandra.connection.host", cassandra\_host) \

.config("spark.cassandra.connection.port", cassandra\_port) \

.config("spark.cassandra.auth.username", username) \

.config("spark.cassandra.auth.password", password) \

.config("spark.sql.streaming.checkpointLocation", checkpoint\_location) \

.getOrCreate()

self.spark.sparkContext.setLogLevel(log\_level)

**return** self.spark

**def** \_consume\_kafka(self, session, brokerlist, topic, reset\_type):

df = session \

.readStream \

.format('kafka') \

.option('kafka.bootstrap.servers', brokerlist) \

.option("subscribe", topic) \

.option('startingOffsets', reset\_type) \

.load()

**return** df

**def** \_process\_kafka\_df(self, df, verbose=1):

schema = StructType() \

.add("timestamp", StringType()) \

.add("switchid", StringType()) \

.add("portid", StringType()) \

.add("bytetransmitted", IntegerType()) \

.add("bytereceived", IntegerType()) \

.add("packetstransmitted", IntegerType()) \

.add("packetsreceived", IntegerType())

df1 = df.selectExpr("CAST(value AS STRING) as message", "timestamp AS handle\_timestamp")

df2 = df1.select(from\_json(col("message"), schema).alias("jsonData"), "handle\_timestamp") \

.select("jsonData.\*", "handle\_timestamp") \

.selectExpr("CAST(SUBSTRING(timestamp, 1, 10) as INT) as cut\_timestamp", "\*") \

.drop("timestamp") \

.select("\*")

**return** df2

**def** \_get\_switchport\_stats(self, source\_df : DataFrame) -> DataFrame:

df = source\_df.withWatermark("handle\_timestamp", "1 minute") \

.groupBy("switchid", "portid", window("handle\_timestamp", "1 minute", "30 seconds")) \

.agg(sum("bytetransmitted").alias("bytest"),

sum("bytereceived").alias("bytesr"),

sum("packetstransmitted").alias("packetst"),

sum("packetsreceived").alias("packetsr"),

max("cut\_timestamp").alias("timestamp"),

min("cut\_timestamp").alias("MinT"))

switchport\_stats = df.select("\*",

(col("bytest") / (col("timestamp") - col("MinT"))).alias("bytestt"),

(col("bytesr") / (col("timestamp") - col("MinT"))).alias("bytesrt"),

(col("packetst") / (col("timestamp") - col("MinT"))).alias("packetstt"),

(col("packetsr") / (col("timestamp") - col("MinT"))).alias("packetsrt"),

((col("bytest") + col("bytesr")) / (col("timestamp") - col("MinT"))).alias(

"bytestr"),

((col("packetst") + col("packetsr")) / (col("timestamp") - col("MinT"))).alias(

"packetstr"),

).drop("MinT", "handle\_timestamp", "window")

**return** switchport\_stats

**def** \_get\_switch\_stats(self, source\_df):

df = source\_df.withWatermark("handle\_timestamp", "1 minute") \

.groupBy("switchid", window("handle\_timestamp", "1 minute", "30 seconds")) \

.agg(sum("bytetransmitted").alias("bytest"),

sum("bytereceived").alias("bytesr"),

sum("packetstransmitted").alias("packetst"),

sum("packetsreceived").alias("packetsr"),

max("cut\_timestamp").alias("timestamp"),

min("cut\_timestamp").alias("MinT"))

switch\_stats = df.select("\*",

(col("bytest") / (col("timestamp") - col("MinT"))).alias("bytestt"),

(col("bytesr") / (col("timestamp") - col("MinT"))).alias("bytesrt"),

(col("packetst") / (col("timestamp") - col("MinT"))).alias("packetstt"),

(col("packetsr") / (col("timestamp") - col("MinT"))).alias("packetsrt"),

((col("bytest") + col("bytesr")) / (col("timestamp") - col("MinT"))).alias(

"bytestr"),

((col("packetst") + col("packetsr")) / (col("timestamp") - col("MinT"))).alias(

"packetstr"),

).drop("MinT", "handle\_timestamp", "window")

**return** switch\_stats

**def** \_get\_network\_stats(self, source\_df):

df = source\_df.withWatermark("handle\_timestamp", "1 minute") \

.groupBy(window("handle\_timestamp", "1 minute", "30 seconds")) \

.agg(sum("bytetransmitted").alias("bytest"),

sum("bytereceived").alias("bytesr"),

sum("packetstransmitted").alias("packetst"),

sum("packetsreceived").alias("packetsr"),

max("cut\_timestamp").alias("timestamp"),

min("cut\_timestamp").alias("MinT"))

network\_stats = df.select("\*",

(col("bytest") / (col("timestamp") - col("MinT"))).alias("bytestt"),

(col("bytesr") / (col("timestamp") - col("MinT"))).alias("bytesrt"),

(col("packetst") / (col("timestamp") - col("MinT"))).alias("packetstt"),

(col("packetsr") / (col("timestamp") - col("MinT"))).alias("packetsrt"),

((col("bytest") + col("bytesr")) / (col("timestamp") - col("MinT"))).alias(

"bytestr"),

((col("packetst") + col("packetsr")) / (col("timestamp") - col("MinT"))).alias(

"packetstr"),

).drop("MinT","window")

**return** network\_stats

**def** write\_to\_console(self, df):

writer = df.writeStream \

.trigger(processingTime="5 seconds") \

.format("console") \

.outputMode("update") \

.start()

**return** writer

**def** write\_to\_cassandra(self, df, keyspace, table):

writer = df.select("\*").writeStream \

.trigger(processingTime="30 seconds") \

.format("org.apache.spark.sql.cassandra") \

.outputMode("append") \

.options(table=table, keyspace=keyspace) \

.start()

**return** writer

**def** start(self, logger, verbose=5):

logger.info("-" \* 60)

logger.info("Creating or getting session...")

logger.info("-" \* 60)

session = self.\_create\_session(\*self.\_read\_spark\_config(self.\_config),

\*self.\_read\_cassandra\_config(self.\_config))

logger.info("-"\*60)

logger.info("Start Stream Processing...")

logger.info("-"\*60)

*# Start consuming Kafka topic*

kafka\_df = self.\_consume\_kafka(session, \*self.\_read\_kafka\_config(self.\_config))

*# Process Kafka values to target schema*

df = self.\_process\_kafka\_df(kafka\_df)

**if** verbose:

logger.info("-" \* 60)

logger.info("Processed DataFrame schema:")

df.printSchema()

logger.info("-" \* 60)

*# Switchport stats*

switchport\_stats = self.\_get\_switchport\_stats(df)

**if** verbose:

logger.info("-" \* 60)

logger.info("SwitchPort stats schema:")

switchport\_stats.printSchema()

logger.info("-" \* 60)

*# def lp\_map\_fill(df):*

*# with open("network\_paths.json", "r") as f:*

*# lp\_map = load(f)*

**if** verbose:

logger.info("-" \* 60)

logger.info("Switch stats schema:")

logger.info(switchport\_stats.rdd.keys())

logger.info("-" \* 60)

switch\_stats = self.\_get\_switch\_stats(df)

**if** verbose:

logger.info("-" \* 60)

logger.info("Switch stats schema:")

switch\_stats.printSchema()

logger.info("-" \* 60)

network\_stats = self.\_get\_network\_stats(df)

**if** verbose:

logger.info("-" \* 60)

logger.info("Network stats schema:")

network\_stats.printSchema()

logger.info("-" \* 60)

switchport\_cassandra\_writer = self.write\_to\_cassandra(

switchport\_stats,

"traffickeyspace",

"switch\_port\_statistics",

)

switch\_cassandra\_writer = self.write\_to\_cassandra(

switch\_stats,

"traffickeyspace",

"switch\_statistics",

)

network\_cassandra\_writer = self.write\_to\_cassandra(

network\_stats,

"traffickeyspace",

"network\_statistics",

)

network\_cassandra\_writer.awaitTermination()

**from** **pyspark.sql.session** **import** SparkSession

**from** **pyspark.sql.functions** **import** \*

**from** **pyspark.sql.types** **import** \*

**import** **configparser**

**import** **logging**

**import** **sys**

**import** **os**

**from** **stream\_processor** **import** StreamProcessor

config = configparser.RawConfigParser()

CONFIG\_PATH = os.getenv("SPARK\_PROPERTIES")

config.read(CONFIG\_PATH)

APP\_NAME = config.get('Spark', 'app.name')

MASTER = config.get('Spark', 'master')

BROKERLIST = config.get('Kafka', 'brokerlist')

TOPIC = config.get('Kafka', 'topic')

RESET\_TYPE = config.get('Kafka', 'resetType')

**if** \_\_name\_\_ == '\_\_main\_\_':

*# conf = SparkConf().setAppName(app\_name).setMaster(master)*

*# sc = SparkContext(conf=conf)*

logger = logging.getLogger('sdn-spark')

logger.setLevel(logging.INFO)

handler = logging.StreamHandler(sys.stdout)

handler.setLevel(logging.INFO)

formatter = logging.Formatter('**%(asctime)s** - **%(name)s** - **%(levelname)s** - **%(message)s**')

handler.setFormatter(formatter)

logger.addHandler(handler)

stream = StreamProcessor(config)

stream.start(logger)

set -ex

# Create casandra schema

docker exec cassandra-sdn cqlsh --username cassandra --password cassandra -f /schema.cql

# Create Kafka topic "sdn-data-event"

docker exec kafka-sdn kafka-topics --create --topic sdn-data-event --partitions 1 --replication-factor 1 --if-not-exists --zookeeper zookeeper:2181

# Create our folders on Hadoop file system and total permission to those

docker exec namenode hdfs dfs -rm /lambda-arch

docker exec namenode hdfs dfs -mkdir /lambda-arch

docker exec namenode hdfs dfs -mkdir /lambda-arch/checkpoint

docker exec namenode hdfs dfs -chmod -R 777 /lambda-arch

docker exec namenode hdfs dfs -chmod -R 777 /lambda-arch/checkpoint

version: "3.3"

networks:

net:

driver: bridge

ipam:

config:

- subnet: 172.22.0.0/24

services:

zookeeper:

image: confluentinc/cp-zookeeper:5.1.0

hostname: zookeeper

container\_name: zookeeper-sdn

ports:

- 2181:2181

networks:

- net

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

kafka:

image: confluentinc/cp-kafka:5.1.0

ports:

- 9092:9092

- 29092:29092

depends\_on:

- zookeeper

environment:

KAFKA\_ZOOKEEPER\_CONNECT: "zookeeper:2181"

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT\_HOST://localhost:29092

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT

KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

volumes:

- /var/run/docker.sock:/var/run/docker.sock

hostname: kafka

container\_name: kafka-sdn

networks:

- net

restart: always

cassandra:

image: 'bitnami/cassandra:latest'

hostname: cassandra

networks:

net:

ipv4\_address: 172.22.0.6

ports:

- "9042:9042"

environment:

- "MAX\_HEAP\_SIZE=256M"

- "HEAP\_NEWSIZE=128M"

container\_name: cassandra-sdn

volumes:

- ./sdn-spark-processor/data/SDNData.cql:/schema.cql

spark-master:

image: bde2020/spark-master:3.0.0-hadoop3.2-java11

container\_name: spark-master

hostname: spark-master

healthcheck:

interval: 5s

retries: 100

ports:

- "8080:8080"

- "7077:7077"

- "4040:4040"

- "4041:4041"

environment:

- INIT\_DAEMON\_STEP=false

- SPARK\_DRIVER\_HOST=192.168.1.5

volumes:

- ./sdn-spark-processor/target:/opt/spark-data

networks:

- net

spark-worker-1:

image: bde2020/spark-worker:3.0.0-hadoop3.2-java11

container\_name: spark-worker-1

hostname: spark-worker-1

depends\_on:

- spark-master

ports:

- "8081:8081"

environment:

- "SPARK\_MASTER=spark://spark-master:7077"

volumes:

- ./data/spark/:/opt/spark-data

networks:

- net

namenode:

image: bde2020/hadoop-namenode:1.1.0-hadoop2.8-java8

container\_name: namenode

hostname: namenode

volumes:

- ./data/namenode:/hadoop/dfs/name

environment:

- CLUSTER\_NAME=test

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

Heal5thcheck:

interval: 5s

retries: 100

networks:

- net

ports:

- 50070:50070

- 8020:8020

datanode:

image: bde2020/hadoop-datanode:1.1.0-hadoop2.8-java8

container\_name: datanode

hostname: datanode

volumes:

- ./data/datanode:/hadoop/dfs/data

environment:

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

depends\_on:

- namenode

healthcheck:

interval: 5s

retries: 100

networks:

- net

ports:

- 50075:50075

- 50010:50010