НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

«На правах рукопису»
УДК _____

«До захисту допущено»
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
“___” _____ 2019р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 113 «Прикладна математика»
(код і назва)

на тему: Моделі та методи підвищення спостережуємості сучасних програмно визначених мереж

Виконав (-ла): студент (-ка) 2 курсу, групи ФІ-83мп
(шифр групи)

Малишко Антон Сергійович
(прізвище, ім’я, по батькові) (підпис)

Керівник Кудін Антон Михайлович, д.т.н., професор
науковий ступінь, вчене звання, прізвище та ініціали) _____ (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»
Фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти: другий (магістерський) за освітньо–професійною програмою

Спеціальність: 113 «Прикладна математика»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)

«___»_____2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Малишко Антон Сергійович
(прізвище, ім'я, по батькові)

1. Тема дисертації

Моделі та методи підвищення спостережуємості сучасних програмно визначених мереж,
науковий керівник дисертації Кудін Антон Михайлович, д.т.н, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від _____р. № _____

2. Термін подання студентом дисертації _____

3. Об'єкт дослідження Програмно-визначена мережа_____
_____

4. Предмет дослідження (Вхідні дані – для магістерської дисертації за освітньо–професійною програмою)
_____
_____
_____

5. Перелік завдань, які потрібно розробити _____
_____
_____
_____
_____

6. Орієнтовний перелік ілюстративного матеріалу _____

_____

_____

7. Орієнтовний перелік публікацій _____

_____

_____

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент _____ (підпис) _____ (ініціали, прізвище)

Науковий керівник дисертації _____ (підпис) _____ (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

Кваліфікаційна робота містить: 62 стор., 11 рисунків, 22 джерел.

Метою виконання даної роботи є підвищення спостережуємості мереж, побудованих на основі технології програмно-конфігурованої мережі.

Об'єктом дослідження є програмно-конфігурована мережа.

Предметом дослідження є методи маршрутизації трафіку та варіювання топології транспортної мережі для забезпечення можливості повного аналізу трафіку.

Розглянуто проблему оптимальної маршрутизації трафіку через системи аналізу трафіку різних видів. Побудовано моделі програмно-конфігурованої мережі з різними підходами до збору інформації про трафік, обґрунтовано доцільність їх використання у різноманітних ситуаціях. Побудовано методи маршрутизації трафіку відповідно до визначених вимог та методи оптимізації маршрутів в залежності від стану та призначення мережі.

ТРАНСПОРТНА МЕРЕЖА, ПРОГРАМНО-КОНФІГУРОВАНА МЕРЕЖА, АНАЛІЗ ТРАФІКУ, МАРШРУТИЗАЦІЯ

РЕФЕРАТ

Квалификационная работа содержит: 62 стр., 11 рисунков, 22 источников.

Целью выполнения данной работы является повышение наблюдаемости сетей, построенных на основе технологии программно-определяемой сети.

Объектом исследования является программно-определяемая сеть.

Предметом исследования являются методы маршрутизации трафика и варьирование топологии транспортной сети для обеспечения возможности полного анализа трафика.

Рассмотрена проблема оптимальной маршрутизации трафика через системы анализа трафика разных видов. Построены модели программно-определяемой сети с разными подходами к сбору информации про трафик, обоснована целесообразность их использования в различных ситуациях. Построены методы маршрутизации трафикав соответствии с обозначеными требованиями и методы оптимизации маршрутов в зависимости от состояния и предназначения сети.

ТРАНСПОРТНАЯ СЕТЬ, ПРОГРАМНО КОНФИГУРИРУЕМАЯ СЕТЬ, АНАЛИЗ ТРАФИКА, МАРШРУТИЗАЦИЯ

ABSTRACT

Qualification work contains: 62 pages, 11 images, 22 sources.

The purpose of this work is to increase visibility of software-defined networks.

The object of the research is software-defined networking.

The subject of the research are traffic routing methods and variation of flow network topoogy in purpose of creating full traffic analysis possibility.

The problem of optimal routing through traffic analysis systems of different types is considered. Software-defined network models with different approaches of traffic information gathering have been constructed, there use cases have been substantiated. Traffic routing methods have been created according to defined requirements, and path optimization methods that depend on state and purpose of a network have been built.

FLOW NETWORK, SOFTWARE DEFINED NETWORK, TRAFFIC ANALISYS, ROUTING

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SDN – Software Defined Network

API – Application Programming Interface

ЦОД – центр обробки даних

REST – Representational State Transfer

NOS – Network Operation System

XMPP – Extensible Messaging and Presence Protocol

VLAN – Virtual Local Area Network

ВСТУП

Актуальність дослідження полягає у необхідності створення моделі програмно конфігурованої мережі, яка на рівні даних забезпечує спостережуємість та контроль за усім трафіком між додатками. Зокрема, розглядаються різні моделі маршрутизації, направленої на повний аналіз трафіку, і можливості оптимізації: максимізація пропускної здатності, балансування навантаження та зменшення довжини маршрутів.

Метою дослідження є вирішення задачі підвищення спостережуємості у програмно конфігурованій мережі. Для досягнення мети необхідно розв'язати задачу побудови моделі роботи програмно конфігурованої мережі та методів, що забезпечують виконання вимог щодо аналізу трафіку та оптимальної маршрутизації.

Для розв'язання задачі необхідно виконати наступні завдання:

1) провести огляд опублікованих джерел за тематикою дослідження;

2) проаналізувати особливості програмно конфігурованих мереж та можливості, які вони надають для виконання поставленої задачі;

3) побудувати моделі та методи, які реалізують можливості, які надає програмно конфігурована мережа.

При розв'язанні поставлених завдань було використано наступні *методи дослідження*: модель транспортної мережі, теорія графів, лінійне програмування

Наукова новизна полягає у модифікації моделей програмно конфігурованих мереж з оглядом на аналіз трафіку та побудові спеціальних алгоритмів маршрутизації, направлених на оптимальний аналіз трафіку на розподілених по мережі ресурсах.

Практична значимість отриманих результатів полягає у реалізації можливостей програмно конфігурованих мереж для оптимізації маршрутизації з забезпеченням умов повного спостереження за трафіком.

1 SDN ТА АНАЛІЗ ТЕЛЕМЕТРИЧНИХ ДАНИХ

Програмно-конфігурована мережа – це концепція, яка дозволяє зняти деякі обмеження сучасних мережевих структур. По-перше, вона розбиває вертикальну інтеграцію, тобто відокремлює логіку керування від маршрутизаторів і комутаторів на більш низькому рівні, які направляють трафік. По-друге, з поділом на ці рівні, комутатори стають простими маршрутизуючими пристроями, а управління зосереджено в логічно централізованому контролері, що спрощує впровадження мережевих політик, реконфігурацію мережі та її еволюцію. Ми зосередимо увагу на можливостях, які з'являються при використанні SDN, у контексті тих задач збору інформації про трафік, які потребують аналізу трафіку із суттєвим навантаженням на пристрої, які проводять цей аналіз.

1.1 Програмно-конфігурована мережа (SDN, Software-Defined Network)

SDN – це одна із форм віртуалізації мережі, в якій рівень управління мережею реалізується програмно і відділений від пристроїв, що відповідають за передачу даних.

Мета SDN полягає в тому, щоб допомогти інженерам та адміністраторам мереж швидко реагувати на зміни бізнес-вимог через централізований пункт управління. SDN охоплює кілька видів мережевих технологій, створених для того, щоб зробити мережу більш гнучкою для підтримки віртуалізованих серверів та баз даних сучасного центру обробки даних. Програмно-конфігуровані мережі визначили підхід до проектування, побудови та управління мережами, при якому

відокремлюють мережеву політику SDN («мізки») від рівня передачі даних. Це дозволяє керувати мережею безпосередньо програмно, а також абстрагувати інфраструктуру від додатків.

1.1.1 Загальна схема SDN

У традиційних мережах рівень контролю (control plane), який вирішує, як керувати мережевим трафіком, і рівень передачі даних (forwarding plane, або data plane), який спрямовує рух за рішеннями рівня контролю, тісно пов'язані та обмежують гнучкість та еволюцію мережі. У програмно-конфігурованих мережах рівень контролю відокремлюється від рівня передачі даних і переміщується в зовнішній об'єкт, який називається контролером SDN. Рішення про маршрутизацію даних базуються на понятті потоків, і ці рішення можна по-різному запрограмувати. Програмування SDN виконується на базі мережевих операційних систем (NOS, Network Operation Systems), які реалізують концепцію програмної конфігурації.

Спостережуємість в SDN логічно централізована в програмних контроллерах, які мають повноваження на здобуття інформації із усіх складових мережі. В результаті, для прикладних програм на більш високому рівні мережа абстрагується до логічно єдного пристрою. Таким чином, мережа представляється як логічно цілісна абстрактна структура, яку простіше змінювати та використовувати. У той же час дизайн мережевих пристроїв, що лежать в основі контролерів SDN, спрощується, оскільки їм не потрібно розрізняти протоколи і політики: їм лише потрібно приймати та виконувати інструкції від контролерів.

Тому логічну структуру програмно-конфігурованої мережі можна розділити на три рівні:

1) прикладний рівень, на якому працюють додатки;

2) рівень контролю, який включає мережеву операційну систему, яка надає додаткам програмний інтерфейс для використання мережі;

3) рівень інфраструктури (або транспортний рівень), на якому відбувається передача даних через комутатори та маршрутизатори.

Рівень інфраструктури включає мережеві пристрої, які відповідають за ефективну передачу даних. На цьому рівні, як і в традиційній мережі, знаходиться мережеве обладнання. Однак відміна від традиційної мережі в тому, що в даному випадку пристрої – як фізичні, так і віртуальні – є лише засобами для пересилання пактів і не можуть приймати рішення. Здатність приймати рішення переноситься на рівень керування. Більш того, концептуально програмно-конфігуровані мережі будуються на відкритих і стандартних інтерфейсах (наприклад, OpenFlow). Відкриті інтерфейси дозволяють контролерам динамічно програмувати різнородні пристрої пересилання, що важко реалізувати в традиційних мережах через велике розмаїття закритих інтерфейсів і розподіленого характеру рівня керування: кожний роутер налаштовується окремо.

Площина управління використовує протоколи, за допомогою яких заповнюються транспортні таблиці в мережевих елементах рівня інфраструктури. Платформа управління включає в себе програмні служби, які використовуються для дистанційного контролю і налаштування керування. Мережева політика визначена на рівні керування, там вона реалізується, пересилаючи дані відповідно до цієї політики [13].

Поставники програмно-конфігурованих мереж пропонують широкий вибір конкуруючих архітектур, але усі програмно-конфігуровані мережеві рішення мають певний варіант контролера SDN, а також «південні» і «північні» API.

Контролери SDN у якості мозку мережі забезпечують централізований перегляд мережі і дозволяють мережевим адміністраторам безпосередньо вказувати мережевим пристроям, яким чином слід

обробляти мережевий трафік на транспортному рівні. Таким чином забезпечується зв'язок між мережевими пристроями та можливість контролю мережі [4].

Південні API використовують у програмно-конфігурованих мережах для передачі інформації про правила маршрутизації трафіку (і не тільки) комутаторам і маршрутизаторам на більш низьких рівнях. OpenFlow вважається першим стандартом SDN, був першим південним API та залишається одним із найбільш вживаних протоколів. Незважаючи на те, що деякі вважають, що OpenFlow і SDN – це одне і теж саме, OpenFlow насправді є лише реалізацією однієї частини. У південних API також здійснюється процес приєднання нових машин-хостів до мережі. Для цього в основному використовується протокол XMPP (extensible messaging and presence protocol) – протокол комунікації, призначений для обміну повідомленнями між клієнтами через централізований сервер [5]. XMPP-клієнт встановлюється на нову машину та повідомляє контроллер про свою присутність у мережі. Після цього клієнт отримує конфігураційні дані, необхідні для участі у передачі даних у мережі.

Північні API використовують для спілкування з додатками та бізнес-логікою на більш високому рівні. Це низькорівневий інтерфейс, який дозволяє додаткам спілкуватися з мережевими пристроями. Саме цей інтерфейс реалізує абстракцію топологій та допомагає адміністраторам мережі програмно видозмінювати трафік та розгортати служби. У якості північних API як правило використовують REST API або OSGi [6].

Однією з найбільших переваг SDN є те, що логіка керування мережею централізована в програмних контролерах. Для додатків мережа представляється рівнем контролю, якому вони можуть віддавати команди через північні API. Це значно спрощує проектування та

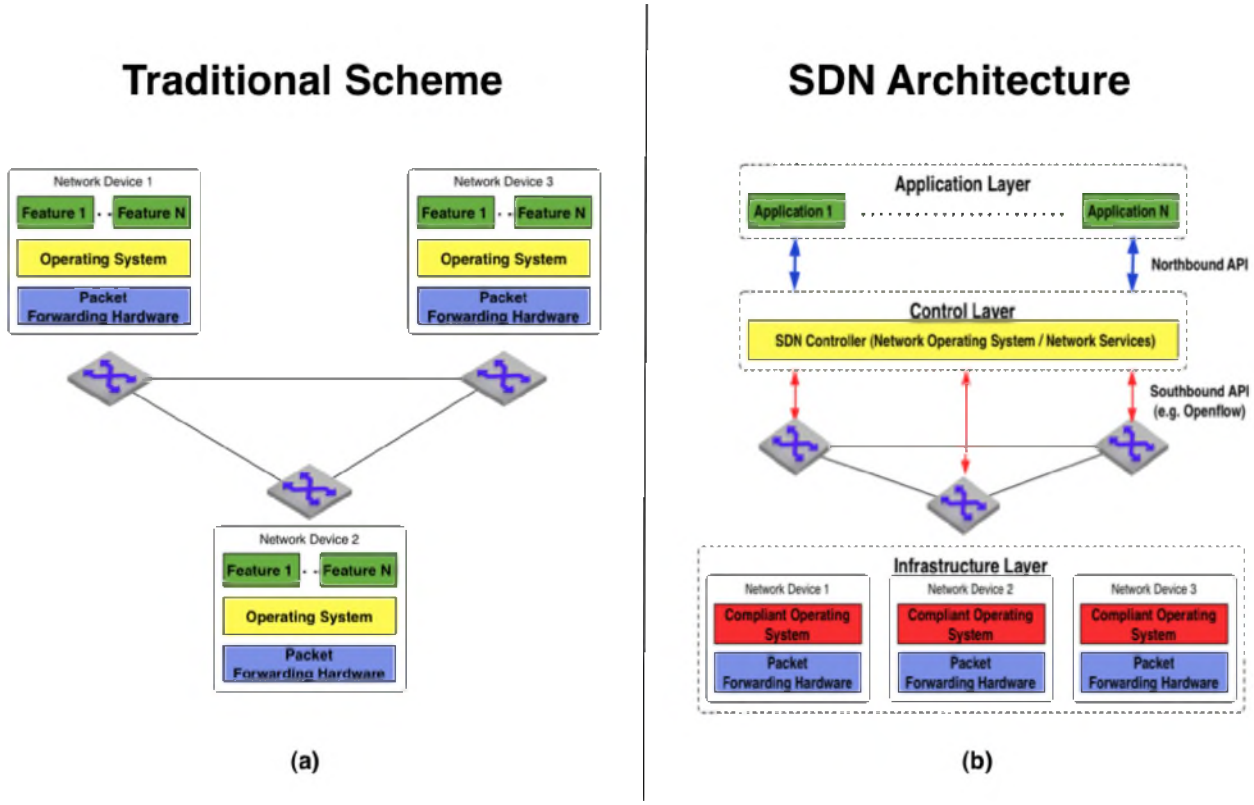

Рисунок 1.1 – Структура SDN

експлуатацію мережі. Крім того, централізований рівень управління полегшує адміністраторам мережі роботу по управлінню та оптимізації мережевих ресурсів. Логічна централізація SDN може бути досягнута використанням коптроллерів.

Контроллер SDN визначає потоки, які існують га транспортному. Кожен потік по мережі повинен спочатку бути дозволений контролером, який перевіряє, що потік не порушує мережеву політику. Якщо контролер дозволяє потік, він обчислює маршрут для потоку і додає запис для цього потоку в кожен комутатор на маршруті. На відміну від складних функцій, що входять до складу контроллера, комутатори просто перенаправляють пакети відповідно до таблиць потоків, записи в яких можуть бути заповнені лише контроллером. Зв'язок між контроллером і комутаторами реалізується через стандартизований протокол і API. Найчастіше для цього використовується протокол OpenFlow [13].

Архітектура SDN відрізняється винятковою гнучкістю: можлива

робота з різними типами комутаторів і на різних рівнях OSI. Контроклери і комутатори програмно-конфігурованих мереж можуть бути реалізовані для Ethernet-комутаторів (Layer 2), маршрутизаторів (Layer 3), транспортного (Layer 4) або маршрутизації на рівні додатків [13]. SDN спирається на загальні функції, наявні в мережевих пристроях, які в основному пов'язані з пересилкою пакетів на основі таблиць потоків.

В архітектурі SDN-комутатор виконує наступні функції:

– Комутатор інкапсулює і перенаправляє перший пакет потоку в контролер SDN, щоб контролер міг вирішити задачу маршрутизації.

– Комутатор перенаправляє вхідні пакети з відповідного порту на основі таблиці потоків. Таблиця потоків може включати інформацію про пріоритет, задану контроллером.

– Комутатор може відкидати пакети певного потоку, тимчасово або постійно, згідно з командами контроллера. Відкидання пакетів може використовуватися в цілях безпеки, подолання атак типу відмова в обслуговуванні (DoS) або вимог до управління трафіком.

Таким чином, контролер SDN управляє станом транспортних таблиць комутаторів в SDN. Це управління здійснюється за допомогою API, що дозволяє контролеру задовольняти найрізноманітніші вимоги без зміни будь-яких аспектів нижчого рівня мережі.

Завдяки розділення рівнів управління і транспорту, SDN дозволяє програмам працювати з одним абстрактним мережевим пристроєм, не піклуючись про деталі роботи пристрою. Мережеві додатки сприймають контролер як єдиний API. Таким чином, можна швидко створювати і розгортати нові додатки відповідно до конкретних корпоративних вимог продуктивності та/або безпеки.

Як правило, контроллер всього один, але це створює серйозні проблеми з масштабованістю через обмеженість ресурсів контролера при обробці великої кількості запитів. Один контролер SDN також має проблеми з надійністю як єдина точка відмови. Для подолання цих проблем можна використовувати розподілений контролер SDN [4]. Між

декількома контролерами можна рівномірно розподілити навантаження, можна передавати обов'язки одного контролера іншому у випадку збою одного чи декількох з них. Таким чином одразу вирішуються проблеми масштабованості та надійності.

Наявність кількох контролерів потребує введення нового API для спілкування між ними. Воно називається «західне/східне API».

У транспортній мережі, яку ми будемо використовувати для моделювання SDN, додатки та бізнес-логіка визначають впорядковані пари хостів, між якими треба передати дані, та передають ці пари до контроллеру SDN через північні API. Контроллер через південні API отримує інформацію з роутерів та інших мережевих пристроїв про стан мережі, на основі цих даних будує маршрут через мережу, і передає відповідні команди на маршрутизацію нового трафіку.

1.1.2 Маршрутизація у SDN

Особливістю SDN є маршрутизація на основі потоку. Потік визначається набором значень полів пакету, які використовуються комутаторами для ідентифікації потоку. В контексті SDN потік являє собою послідовність пакетів між відправником та отримувачем. Всі пакети потоку отримують ідентичні політики обслуговування на пристроях транспортного рівня, що дозволяє уніфікувати поведінку різних типів мережевих пристроїв, включаючи маршрутизатори, комутатори, мережеві екрани і інші проміжні мережеві пристрої. Гнучкість використання потоків обмежується лише можливостями реалізованих таблиць потоків [13].

Так звана таблиця потоків (flow table) є ключовою компонентою кожного комутатора SDN. Кожен рядок таблиці потоків складається з

трьох записів – поля заголовка, лічильника та дії. Коли пакет потрапляє на комутатор, заголовок пакета співставляється з полями заголовка записів таблиці потоку. Якщо існує запис, що відповідає такому заголовку, комутатор виконує відповідну дію: пересилає пакет далі, або відкидає. Поле лічильника використовується для відстеження статистичних даних про потік. Полів заголовка, що відстежуються, усього дванадцять [7]:

1) порт комутатора, з якого прийшов пакет;
2) ідентифікатор VLAN;
3) пріоритет VLAN;
4) MAC-адреса відправника;
5) MAC-адреса одержувача;
6) тип ethernet-кадру;
7) IP-адреса відправника;
8) IP-адреса одержувача;
9) IP-протокол;
10) біт IP ToS (тип послуги);
11) порт відправлення TCP / UDP;
12) порт призначення TCP / UDP.

Порівняння полів пакету з записами у таблиці зупиняється, коли знайдено перший запис, потік якого відповідає полям пакету. Кількість відповідних полів залежатиме від рівня відповідності. У разі повної відповідності всі дванадцять полів узгоджуються, при відповідності другого рівня узгоджуються лише полія другого рівня, третього рівня – поля заголовка третього рівня. Якщо пакет, що надходить, не відповідає жодному запису потоку, пристрій переадресації вважає це як пакет нового потоку, інкапсулює заголовок пакета у повідомленні «PACKET IN» і передає його контролеру для обробки як особливий випадок. Але в деяких випадках комутатор може в обов'язковому порядку пересилати пакет до контролера – наприклад, якщо це пакет управління протоколом маршрутизації.

Отримавши повідомлення «PACKET IN», контролер запускає процес пошуку маршруту, після завершення якого вимагатиме зміни таблиць переадресації. Контролер змінює таблиці переадресації комутаторів, на які опирається побудований маршрут, надсилаючи команду «FLOW MOD», а на джерело пакету відсилає повідомлення «PACKET OUT» [7].

1.2 Cisco Tetration

Однією із важливих характеристик мережі є спостережуємість – можливість відслідковувати процеси та події. На думку Cisco, на червень 2016 року не існує єдиного інструменту, призначеного для збору телеметричних даних по всьому ЦОД і масштабного аналізу великих обсягів таких даних в реальному часі. Організації виконують окремі завдання, використовуючи складні, повільні, розрізнені інструменти, і це виявляється досить затратним з точки зору грошей, часу і втраченої вигоди.

Cisco Tetration Analytics є платформою захисту робочих станцій на гібридній хмарі для центрів обробки даних (ЦОД), яка дозволяє детально відслідковувати усі процеси, що відбуваються у системі як локально так і на публічній хмарі. Програма отримує дані телеметрії від програмних та фізичних сенсорів, а потім обробляє їх за допомогою спеціальних методів машинного навчання, аналізу поведінки та алгоритмів.

Платформа орієнтована на реалізацію критичних для ЦОД завдань – забезпечення відповідності політикам, «криміналістичний» аналіз додатків (application forensics) і перехід до моделі інформаційної безпеки на основі білих списків. Tetration Analytics веде безперервний моніторинг та аналіз, що дозволяє IT-менеджерам глибше зрозуміти процеси, які відбуваються в ЦОД. Це спрощує механізм забезпечення

експлуатаційної надійності, виконання операцій по моделі нульової довіри (zero-trust operations) і міграцію додатків в хмару.

Платформа забезпечує:

– виявлення залежностей додатків один від одного в ЦОД і в хмарі;

– моделювання змін у політиках для оцінювання результатів до їх впровадження;

– здійснення пошуку в мільярдах потоків менш ніж за секунду за допомогою пошукового механізму Tetration і користувацького інтерфейсу;

– постійний моніторинг функціонування додатків для оперативного виявлення будь-яких відхилень.

1.2.1 Архітектура Tetration

Рисунок 1.2 – Архітектура Tetration

Платформа Cisco Tetration Analytics має три основних

функціональних рівня:

1) Рівень збору даних. Цей рівень формують переважно сенсори - очі і вуха аналітичної платформи. Сенсори можуть бути двох типів:

– Програмні (хостові). Їх встановлюють на будь-які сервери кінцевого хоста (віртуалізовані або без ПЗ).

– Апаратні сенсори. Вбудовані в комутатори Cisco Nexus 92160YC-X, Cisco Nexus 93180YC-EX і Cisco Nexus 93108TC-EX.

Ці сенсори збирають телеметричні дані трьох типів:

– Дані про потоки. Відомості про кінцеві пристрої, протоколи, порти, час запуску потоку, його тривалість тощо.

– Неоднорідність властивостей пакетів. Відмінності у властивостях пакетів всередині потоку. Як приклади можна назвати мінливість TTL, прапорів IP/TCP, розміру корисного навантаження тощо.

– Контекстна інформація. В заголовок пакета ці відомості не включені. Для програмного сенсора це можуть бути дані про процес, процес-джерело потоку, ідентифікатор процесу, користувач, асоційований з цим процесом тощо.

Сенсори не обробляють дані корисного навантаження і не здійснюють семплювання. Сенсори призначені безпосередньо для відстеження кожного пакета і кожного потоку. Крім сенсорів, на цьому рівні розташовуються сторонні джерела (балансувач навантаження, таблиці DNS-серверів тощо) для збору даних налаштувань. Конфігураційні дані доповнюють інформацію, що надходить від платформи аналітики.

2) Аналітичний рівень. Дані від сенсорів передаються платформі Cisco Tetration Analytics, яка грає роль мозкового центру і виконує власне аналітичні функції. Багатосерверна платформа на основі «великих даних» обробляє інформацію від сенсорів, і застосовуючи контрольоване і неконтрольоване машинне навчання, аналіз поведінки та інтелектуальні алгоритми, формує комплексну середу для виконання наступних завдань:

– Всесторонній моніторинг всієї інфраструктури центру обробки даних в режимі реального часу.

– Аналіз взаємодії програмних компонентів на основі поведінки цих компонентов.

– Автоматизоване групування подібних кінцевих пристроїв (наприклад, кластери веб-серверів, кластери баз даних тощо).

– Послідовна вироблення рекомендацій щодо політики білого списку для додатків, а також виявлення невідповідностей цій політиці з представленням результатів за лічені хвилини.

– Аналіз фактичного впливу політики і її тестування перед безпосереднім застосуванням в мережі.

– Довгострокове зберігання інформації для аналізу даних за тривалі проміжки часу без втрати деталізації.

– Детальна технічна експертиза із застосуванням природномовного пошуку та візуальних запитів.

3) Рівень візуалізації. Платформа підтримує доступ до цих даних через графічний веб-інтерфейс і API-інтерфейси REST. Крім того, платформа має інтерфейс повідомлень для низькорівневих систем, за допомогою якого такі системи можуть отримувати повідомлення про потоки трафіку, відповідність політикам тощо.

Можливість сегментації Cisco Tetration дозволяє автоматизувати впровадження чітко налагоджених політик для критично важливих для бізнес-моделі програм, що виконуються як у локальних центрах обробки даних так і в публічній хмарі. Застосування правильних політик для віртуальних і фізичних машин за допомогою цієї моделі дає можливість значно зменшити вразливість центру обробки даних, зменшуючи кількість можливих векторів атаки.

Підхід Cisco Tetration можна узагальнити так:

– Спочатку збираються дані телеметрії на основі кожного пакету кожного потоку та його контексту, що включає ідентифікатор процесу. Таким чином збирається найточніша інформація про мережевий потік та його учасників.

– Зібрані дані обробляються на платформі великих даних. За

допомогою технологій машинного навчання та штучного інтелекту на цьому етапі:

– Робочі навантаження та потоки даних пов'язуються з додатками;

– Виокремлюються рівні додатків за допомогою інтелектуальних алгоритмів кластеризації та пов'язуються між собою на різних рівнях;

– Виявляються зовнішні залежності;

– Створюються схеми, які демонструють усі внутрішні та зовнішні залежності.

– Рекомендована політика формується на основі автоматичного виявлених залежностей додатків. Ця політика може бути доповнена правилами, зумовленими специфічною бізнес-логікою замовника.

– Технології великих даних також використовуються коли клієнти проводять моделювання з використанням архівних даних у великому обсязі для розробки та вдосконалення політики мікросегментації додатків.

– Впровадження політики мікросегментації додатків автоматизоване та здатне поширюватись на платформи різних типів (хмари, центр обробки даних), контрольні пункти (мережевий брандмауер, мережева фабрика, хост) та типи хоста (фізичні або віртуальні машини).

– На стадії функціонування платформа слідкує за відхиленнями у поведінці програм та викликає відповідні робочі процеси для оновлення політик.

1.3 Cisco ACI (Application-Centric Infrastructure)

Cisco запропонувала свій підхід по побудови топології мережі, в центрі якого – ідея, що у бізнес-логіці головними є додатки. Application-Centric Infrastructure дозволяє конфігурувати усю мережу з

однієї точки за рахунок певним чином побудованої топології.

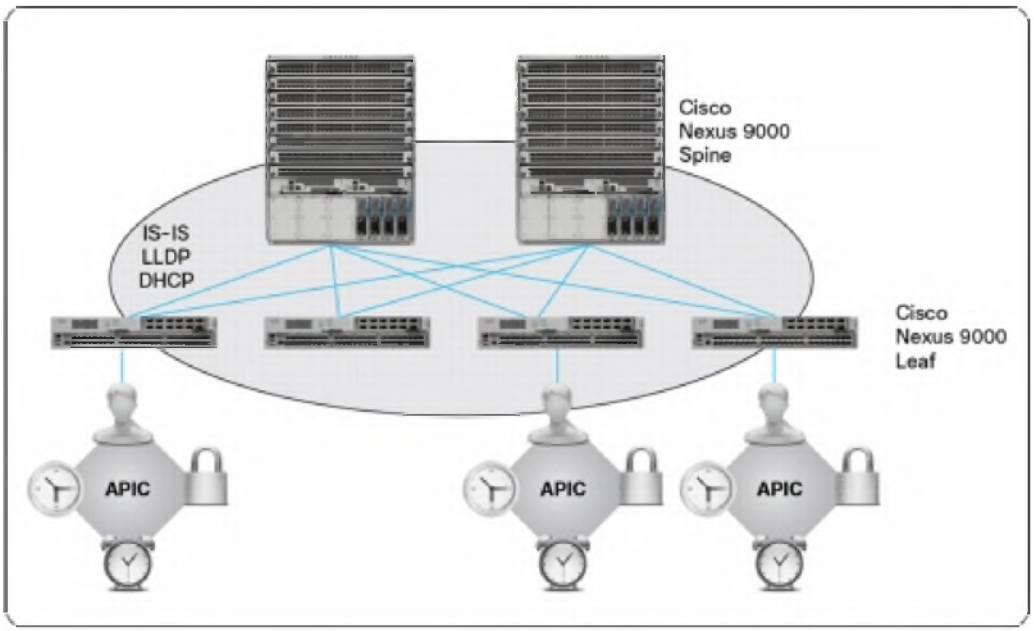

Рисунок 1.3 – Архітектура Cisco ACI

ACI включає два типи комутаторів – leaf та spine, при чому кожен spine зв'язаний з усіма leaf-ми. До leaf-комутаторів підключаються звичайні користувачі, а до spine – контроллери (APIC), які відповідають за керування та моніторинг.

Таким чином, легко вирішуються питання масштабування: якщо нам потрібно збільшити пропускну здатність фабрики, ми додаємо spine-комутатори, а якщо нам потрібно збільшити портову ємність - leaf.

Для обох рівнів використовуються комутатори серії Cisco Nexus 9000, які для Cisco є основним інструментом для побудови мереж ЦОД незалежно від їх архітектури. Для рівня spine використовуються комутатори Nexus 9300 або Nexus 9500, а для leaf – тільки Nexus 9300.

APIC-контролери являють собою спеціалізовані фізичні сервери, при цьому для невеликих мереж допускається використовувати кластер з одного фізичного APIC контролера і двох віртуальних. Контролери APIC здійснюють функції управління і моніторингу. Важливим є те, що контролери ніколи не беруть участь в передачі даних, тобто, якщо навіть

всі контролери кластера вийдуть з ладу, на стабільність роботи мережі це не вплине. Також необхідно відзначити, що за допомогою APIC-ів адміністратор керує всіма фізичними та логічними ресурсами фабрики, і для того, щоб внести будь-які зміни, не потрібно більше підключатися до того чи іншого пристрою, оскільки в ACI використовується єдина точка керування.

Логічною основою ACI є набір профілей додатків (Application Network Profile). Саме профілі додатків визначають політики взаємодії між усіма мережевими сегментами і описують безпосередньо самі мережеві сегменти. ANP дозволяє абстрагуватися від фізичного рівня і, по суті, уявити, як потрібно організувати взаємодію між різними сегментами мережі з точки зору програми.

Профіль додатку складається з груп підключень (End-point groups – EPG). Група підключень – це набір логічно згрупованих хостів (віртуальних машин, фізичних серверів, контейнерів тощо), які знаходяться в одному сегменті безпеки, тобто з точки зору безпеки мережі мають приблизно рівні права. Кінцеві хости, які відносяться до тієї чи іншої EPG, можуть визначатися великою кількістю критеріїв. Зазвичай використовуються такі:

– Фізичний порт
– Логічний порт (порт-група на віртуальному комутаторі)
– VLAN ID або VXLAN
– IP-адреса або IP-підмережа
– Атрибути сервера (ім'я, розташування, версія ОС і т.п.)

Для взаємодії різних EPG передбачена сутність, яка називається контрактом. Контракт визначає відносини між різними EPG. Іншими словами, контракт визначає яку послугу одна EPG надає іншій EPG. Наприклад, ми створюємо контракт, який дозволяє трафік по протоколу HTTPS. Далі ми поєднуємо цим контрактом, наприклад, EPG Web (група веб-серверів) і EPG App (група application-серверів), після цього ці дві групи можуть обмінюватися трафіком по протоколу HTTPS.

Централізація управління, автоматизація і моніторинг – ключові переваги ACI. ACI фабрика позбавляє адміністраторів від рутинної роботи по створенню великої кількості правил на різних комутаторах, маршрутизаторах і мережевих екранах (при цьому класичний ручний метод налаштування дозволений і може бути використаним). Налаштування профілів додатків та інших об'єктів ACI автоматично застосовуються по всій ACI фабриці. Навіть при фізичному переключенні серверів в інші порти комутаторів фабрики не буде потреби дублювати налаштування зі старих комутаторів на нові і видаляти непотрібні правила. Виходячи з критеріїв приналежності хоста до EPG, фабрика виконає ці дії автоматично і також автоматично видалить невикористовувані правила.

Інтегровані політики безпеки ACI реалізовані за принципом білих списків, тобто те, що явно не дозволено, за замовчуванням заборонено. У сукупності з автоматичним оновленням конфігурацій мережевого обладнання (видалення «забутих» невикористовуваних правил і дозволів) цей підхід значно підвищує загальний рівень безпеки мережі і звужує поверхню потенційної атаки.

ACI дозволяє організовувати мережеву взаємодію не тільки віртуальних машин і контейнерів, а й фізичних серверів, апаратних мережевих екранів і мережевого устаткування сторонніх виробників.

Для даної роботи найважливішим є те, що при такій побудові мережі можна здійснювати аналіз трафіку на spine-комутаторах, змусивши трафік кожної пари додатків проходити через spine-комутатор, навіть якщо обидва вони розташовані на рідні одного leaf-комутатора.

1.4 Транспортні мережі

Розглянемо орієнтований граф як мережу труб, по яких тече потік речовини (або інформації) від певного місця, де вона виробляється, до місця, де вона використовується. За вагу ребра графу приймемо пропускну спроможність труби.

Формально це можна записати так: орієнтований граф $G = (V, E)$, кожному ребру $e \in E$ якого поставлена у відповідність вага $c(e) \geq 0$, яка означає пропускну здатність труби. Виділимо також дві особливі вершини: джерело (source) s та витік (sink) t. Інформація рухається від джерела (де вона створюється з деякою постійною швидкістю) до витоку (де вона споживається з тією ж швидкістю).

Ми вважаємо, що на вершинах інформація не накопичується – скільки приходить, стільки і виходить (якщо вершина не є джерелом або стоком). Це властивість називається «законом збереження потоку» (flow conservation).

Таким чином побудована структура називається flow network, або транспортна (потокова) мережа [1].

1.4.1 Максимальний потік

Завдання про максимальний потік для транспортної мережі полягає в наступному: знайти максимально можливу швидкість виробництва (і поглинання) речовини, при якій її ще можна транспортувати від витоку до стоку при даних пропускних спроможностях труб. Ми розглянемо класичний метод Форда-Фалкерсона, а в другому розділі побудуємо його модифікацію для вирішення одного типу задач.

Потоком назвемо функцію

$$f : V \times V \to \mathbb{R}$$

яка задовольняє трьом умовам:
збереження потоку: $\forall\ u,\ v \in V$

$$\sum_{v \in V} f(u, v) = 0$$

кососиметричність: $\forall\ u,\ v \in V$

$$f(u, v) = -f(v, u)$$

обмеження пропускною здатністю: $\forall\ u,\ v \in V$

$$f(u, v) \leq c(u, v)$$

Величиною потоку називається

$$|f| = \sum_{v \in V} f(s, v)$$

Задача про максимальний потік полягає у пошуку потоку максимальної величини для заданого G з джерелом s та витоком t.

1.4.2 Багатопродуктовий потік

В реальній комп'ютерній мережі рідко буває тільки один потік, тому треба розглянути ситуацію, коли є декілька джерел та витоків, при чому потік даних із конкретного джерела повинен потрапити в конкретний витік, тобто вони розбиті на пари. Для кожної пари (s, t) також визначено кількість даних d, які потрібно транспортувати.

Для оптимального розподілення потоків між шляхами можна сформулювати задачу як набір лінійних обмежень, після чого вирішити задачу одним із широкого спектру методів лінійного програмування.

Позначимо $f_p(u, v) : \ V \times V \to [0, 1]$ – частина потоку p, що проходить через ребро e. Тоді обмеження мають таку форму:

$$\forall e \in E : \ \sum_{p \in P} f_p(e) \cdot d_p \leq c(e)$$

$$\forall u \neq s_i, t_i : \ \sum_{w \in V} f_p(u, w) - \sum_{w \in V} f_p(w, u) = 0$$

$$\sum_{w \in V} f_p(s_p, w) - \sum_{w \in V} f_p(w, s_p) = 1$$

$$\sum_{w \in V} f_p(w, t_p) - \sum_{w \in V} f_p(t_p, w) = 1$$

Оптимізаціями можуть бути балансування навантаження на аналітиків та/або ребра, максимізація потоку при невизначених d та інші залежні від ситуації варіанти.

Крім лінійного програмування, існують інші варіанти розв'язання проблеми багатопродуктового потоку, наприклад, [14]. Але цей алгоритм вимагає модифікації задачі – зміни значень пропускних здатностей ребер, а, значить, також і пропускної здатності платформи аналізу трафіку в рамках цієї роботи.

Висновки до розділу 1

Розробки Cisco у сфері підвищення спостережуваності мереж носять в основному практичний характер, а відомі теоретичні алгоритми пошуку маршрутів у графі потребують внесення змін для використання у рамках моделі програмно-конфігурованої мережі, побудованої у другому розділі.

2 ПОБУДОВА ПРОГРАМНО-КОНФІГУРОВАНОЇ МЕРЕЖІ З АКЦЕНТОМ НА АНАЛІЗ ТРАФІКУ

Підхід SDN дозволяє централізовано керувати маршрутизацією трафіку за допомогою контроллерів. При створенні нового потоку трафіку, контроллер обирає маршрут базуючись на основі топології мережі, інформації про її стан та інших критеріях, при чому ці критерії можна запрограмувати досить гнучко.

У технології хмарних обчислень використання контролера SDN для управління станом таблиці потоків комутатора є загальною особливістю всіх механізмів управління трафіком. Підхід SDN розроблений для гнучкого управління трафіком в мережі, тобто маршрутизувати трафік до кінцевої точки так, щоб збалансувати навантаження та іншим чином оптимізувати використання доступних мережевих ресурсів на шляху. За замовчуванням він не використовується для виявлення зловмисного трафіку, і оскільки ми хочемо впровадити цю функцію, ми вводимо поняття аналітика, якому делегується реалізація логіки виявлення зловмисних потоків.

Поширені захисні механізми, як правило, розроблені для зупинки зловмисного трафіку на кордоні мережі. У цій роботі ми розглядаємо ситуацію, коли зловмисний трафік створений всередині мережі, та для нього вже немає вузької точки, через яку він обов'язково пройде на шляху до хоста призначення.

Для підвищення спостережуваності мережі ми побудуємо модель, яка буде направляти трафік таким чином, щоб його було зручно відстежувати. Головною умовою, яка накладається на побудовану мережу, буде обов'язковий прохід кожного пакету через один із сенсорів системи безпеки.

2.1 Базова модель

Представимо комп'ютерну мережу у вигляді транспортної мережі. Розглянемо фізичну топологію комп'ютерної мережі як граф, вершинами $v \in V$ якого є пристрої, які беруть участь у генерації і маршрутизації трафіку, а ребрами $(u, v) \in E$ – канали зв'язку, які їх з'єднують: як правило, витi пари, оптоволоконні дроти або бездротове з'єднання. У каналів зв'язку є задокументована фізично обумовлена межа пропускної здатності $c(u, v) \geq 0$.

Кожен окремий канал зв'язку може працювати у одному з трьох режимів:

Симплексному, коли потік дозволено лише в одну сторону.

Дуплексному, коли потік дозволено в обидві сторони, при чому пропускна здатність протилежних каналів незалежна. У такому режимі зокрема працює вита пара, яка є найрозповсюдженішим каналом передачі даних у сучасних комп'ютерних мережах.

Напівдуплексному, коли в кожний момент часу з двох пристроїв на кінцях каналу один передає, інший приймає. Це можуть бути, наприклад, оптоволоконні дроти.

Дуплексний режим можна змоделювати в орієнтованому графі як пару протилежно направлених ребер з однаковими пропускними здатностями. У витій парі це два фізично розділених канали, і якщо у моделі потік іде по одному з протилежно направлених ребер, то очевидно, якому з каналів він відповідає.

Симплексний режим також легко змоделювати як направлене ребро орієнтованого графу з деякою пропускною здатністю.

Але напівдуплексний режим створює неоднозначність у виборі зваженого орієнтованого графу як основи для побудови моделі. В

залежності від того, скільки трафіку намагається пройти з різних боків, напівдуплексний канал перерозподіляє свою спільну пропускну здатність між ними. Можна змоделювати його як дуплексний з половиною пропускної здатності в кожну сторону, але тоді модель не буде адекватно відображати реальні можливості цього каналу. Це особливо важливо, коли оптоволоконне з'єднання спеціально встановлене для того, щоб пропускати великий обсяг трафіку в одну сторону, а в іншу – малий за потреби. Тут можна знову навести приклад з резервуванням бази даних, коли є дві локації – основна і резервна, і резервування іде тільки в одну сторону. Але якщо локації поміняються ролями, канал теж почне працювати в більшій мірі в іншу сторону. Модель при цьому покаже, що можна використовувати тільки половину реально доступної пропускної здатності напівдуплексного каналу.

Для напівдуплексного каналу краще підійде неорієнтований зважений граф, але він не підходить для дуплексного і симплексного режимів, симплексний є явно направленим, а в дуплексному протилежні канали не залежать один від одного.

За замовченням ми будемо використовувати модель орієнтованого графа, а напівдуплексний канал будемо моделювати як дуплексний з половиною пропускної здатності в кожну сторону, але у деяких випадках розглянемо можливі оптимізації моделі для врахування можливості перерозподілити пропускну здатність у різні сторони у напівдуплексному режимі.

Тепер розглянемо трафік.

Дані передаються в мережі у вигляді пакетів. Для кожного пакету існує пристрій, який його створив, і пристрій, якому він призначений. Ці пристрої відповідно назвемо джерелом s і витоком t для цього пакету.

Вершинами графу мережі будуть усі пристрої, які приймають участь у генерації та маршрутизації трафіку. Будь-яка вершина для пакету може бути джерелом, витоком або нейтральною, потенційно транзитною точкою.

Пакети групуються по потокам згідно з набором полів таблиць

маршрутизації, і уже для потоків ми будемо будувати маршрути. Перший пакет цього потоку ініціює процес побудови маршруту контроллером, після завершення якого запуск потік буде дозволено. Таким чином, у модель мережі замість джерел s і витоків t ми будемо оперувати потоками $p \in P$. Множина P у життєвому циклі мережі буде динамічно змінюватись при створенні нових потоків або їх завершенні.

2.1.1 Реалізація політик

Граф побудованої потокової мережі без врахування політик безпеки ми вважаємо зв'язним, тобто існує фізична можливість для будь-яких двох пристроїв побудувати маршрут, що їх з'єднує. Політики безпеки призначені для обмеження зв'язку додатків, вони визначаються бізнес-логікою на високому рівні.

Для моделювання впливу політик типу чорного списку на маршрутизацію трафіку будемо розглядати політику як функцію, яка на основі метаданих про потік буде виділяти з графу транспортної мережі підграф, у якому можна проводити пошук маршруту для цього потоку.

$$\text{policy: } V \times E \to V \times E \tag{2.1}$$

Далі при пошуку маршруту для потоку p будемо використовувати саме підграф транспортної мережі, отриманий накладанням політик, що стосуються p, на основний граф мережі.

Політики, які обмежують граф, належать до таких політик безпеки, які створюються на основі міркувань бізнес-логіки. З іншого боку, наприклад, у Cisco Tetration розглядається варіант автоматичного створення політик, при якому ведеться пошук залежностей між додатками, і на основі цих залежностей формується політика у формі

білого списку.

Якщо політика має вид білого списку, вона не накладає обмеження на граф мережі, а фактично сама є підграфом. Тобто чорний список визначає граф дозволених зв'язків, забороняючи деякі шляхи для деяких комунікацій, а білий список визначає дозволені шляхи, які самі по собі формують граф.

Тобто моделювання політик білого списку фактично також виділяє з основного графу підграф, у якому ми шукаємо шляхи.

2.1.2 Розширення базової моделі

Тепер модифікуємо базову модель з оглядом на можливості програмно визначених мереж. Головною з них для поставленої цілі є можливість обирати маршрут для кожного потоку.

Дані для аналізу трафіку можуть бути різних типів, основні з яких – це:

– необроблений трафік;

– події мережевих пристроїв;

– інформація про трафік, яку збирає потоковий протокол.

Для забезпечення захищеності мережі усі три типи даних грають важливі ролі, які доповнюють одна одну.

Система аналізу необробленого трафіку повинна зібрати дані і провести їх аналіз, при чому ключовим аспектом моделі є аналіз повністю усього трафіку. Здійснити це можливо за рахунок логіки маршрутизації, яка вимусить кожен потік проходити по маршруту, який включає в себе хоча б один сенсор, який збере дані про трафік і передасть їх на платформу аналізу – аналітика.

Таким чином, необхідно додати до моделі дві сутності: сенсор і

аналітика. Сенсор $se_i \in Se$ встановлюється на вершину графу транспортної мережі $v \in V$, збирає дані про потоки, які проходять через дану вершину, і передають зібрані дані на аналітика $a \in A$.

В залежності від топології та бізнес-логіки процесів мережі, можна по різному моделювати сенсори та аналітиків, а також використовувати різні моделі та методи для побудови маршрутів потоків.

Для аналітиків існують варіанти:

1) один сенсор - один аналітик;

2) багато сенсорів - один аналітик.

Модель «один сенсор - один аналітик» зручно використовувати, коли мережа складається з декількох кластерів, у яких по одній точці виходу назовні. Ця схема поширена у традиційних розподілених по декількох локаціях компаніях, де увесь трафік проходить через єдиний проксі-сервер, який визначає периметр.

Варіант, коли у одного аналітика багато сенсорів, доцільно буде використати, якщо топологія мережі схожа на сітку з багатьма зв'язками, і треба багато сенсорів, щоб була можливість ефективно використовувати усі шляхи. При цьому через кожен сенсор проходить відносно невелика кількість інформації.

Варіант «один сенсор - один аналітик» зводиться до варіанту з багатьма сенсорами, але ми розглядаємо варіант з одним сенсором на аналітика у деяких випадках окремо, оскільки він забезпечує додаткові можливості.

Для потоків:

1) один потік - один шлях;

2) розділення потоку по декількох шляхах (максимальний потік).

Якщо фізичні канали мережі мають високу пропускну здатність, або якщо бізнес-логіка не передбачає передачу великих даних через усю мережу, в ускладненні маршрутизації немає потреби. Розділяти потік доцільно коли, наприклад, треба зробити резервну копію бази даних, або провести міграцію віртуальної машини.

Для розподілення навантаження:

1) один потік - один аналітик;

2) обробка по частинах.

Аналіз трафіку може як потребувати інформацію з усіх пакетів потоку на одному аналітику, так і можливість аналізувати пакети по одному. В залежності від цього, можна обробляти потоки цілком або по частинах. Також обробка по частинах дає більші можливості при балансуванні навантаження та створення великої кількості нових екземплярів аналітиків на машинах з вільними обчислювальними ресурсами.

2.1.3 Стадії вирішення задачі спостережуваності

У процесі функціонування мережі на рівні прикладних програм згідно з бізнес-логікою постійно з'являються і зникають потреби у передачі даних між пристроями. Прикладний рівень абстрагується від процесу маршрутизації трафіку, який він створює, а лише віддає команди на передачу даних контролеру. І хоча пошук зловмисних процесів є важливим процесом, прикладні програми очікують певного рівня якості сервісу від транспортного рівня. Одним із важливих показників якості обслуговування є час, який проходить між запитом прикладного рівня до контроллеру щодо маршрутизації потоку, і відповіддю контроллера про те, що маршрут створено і можна запускати потік. Цей час безпосередньо залежить від того, яким чином буде підраховано маршрут для потоку.

З точки зору програми, яка створює запит на маршрутизацію, важливим є швидкість побудови маршруту, а з точки зору контроллеру важливішим є глобальний оптимум побудованих маршрутів, при якому аналітики завантажені рівномірно, а потоки розподілені рівномірно по

транспортним шляхам і не заважають один одному.

Побудова режиму роботи з компромісним підходом до цього конфлікту полягає у розділенні життєвого циклу мережі на два основні етапи, які будуть циклічно повторюватись.

На першому етапі ми приймаємо нові заявки на маршрутизацію. Ще не існує потоків, або ми розглядаємо граф транспортної мережі, з якої ми виключили усі потоки, зменшивши відповідно пропускну здатність ребер.

На другому етапі ми розглядаємо усі потоки та намагаємось перебудувати усі процеси у сукупності з метою оптимізації.

Тому підходи, які будемо використовувати для побудови маршрутів, належать до однієї із цих двох стадій:

1) при створенні нового потоку будуємо маршрут для нього, використовуючи ребра та аналітиків з вільною пропускною здатністю, не чіпаючи інші потоки;

2) перерозподіляємо потоки між аналітиками, оптимізуючи навантаження на них та на рівень передачі даних.

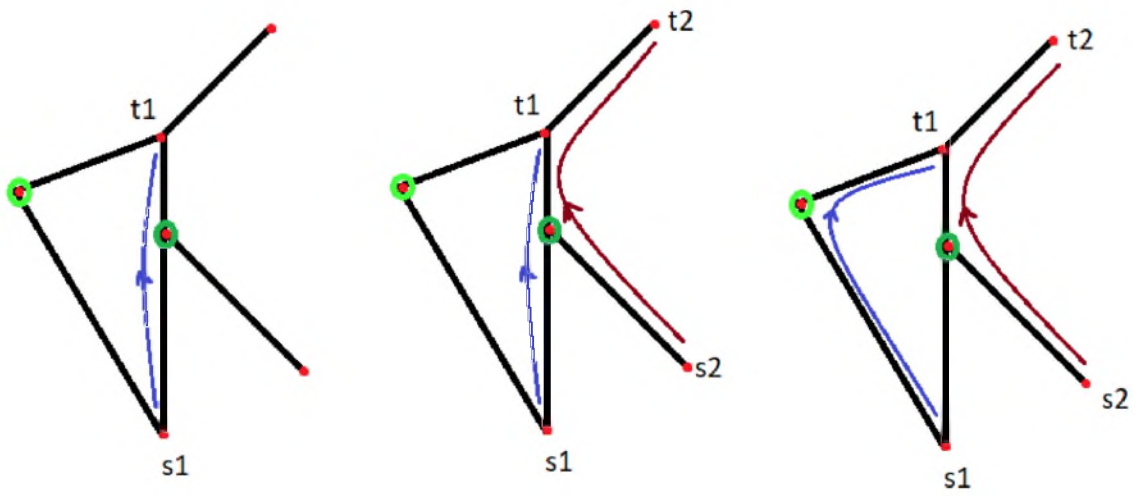

Рисунок 2.1 – Глобальна оптимізація для перерозподілення навантаження на аналітики і ребра

Ми розглянемо ці стадії послідовно. Спочатку розглянемо варіанти пошуку маршруту для нового потоку без перерозподілу інших потоків.

2.2 Варіант «один потік - один шлях»

Коли на контролер поступає новий запит на маршрутизацію між двома точками мережі, якщо невідомо, яка пропускна здатність йому потрібна, простіше за все побудувати один шлях у вигляді послідовності вершин графу, і пустити по них потік. Спочатку розглянемо цю ситуацію.

Кожному процесу $p \in P$ співставимо шлях $r_p = \{v_1 \dots v_n\} \in V^{|V|}$, $v_1 = s$, $v_n = t$ – впорядковану множину вершин, де кожна послідовна пара (з перетином) з'єднана ребром. Цей шлях визначається контроллером мережі, який задає шлях для даного процесу на основі набору запрограмованих правил після того, як перший пакет з потоку потрапить у мережу, а комутатор не знайде асоційованого з ним маршруту. Інший варіант – в явному вигляді створити запит на створення маршруту, у який можна записати також і мінімальні потреби у якості обслуговування та відомості про обсяг даних, які планується передати. Це дозволить контролеру менш часто проводити глобальну оптимізацію.

Умова на шлях може буде сформульована наступним чином: для будь-якого шляху r_{p_i} існує вершина $v \in r_{p_i}$ із сенсором se_i, підключеного до деякого аналітика a.

Таких вершин може бути декілька, і якщо їх сенсори належать до різних аналітиків, можливість обрати один із них дозволяє реалізувати балансування навантаження на аналітики. Введемо поняття навантаження на аналітика у деякий момент часу як відношення сум потоків, які обробляються цим аналітиком у цей момент часу, до максимального потоку, який може обробити аналітик. Позначимо навантаження: $q(a_i) \in [0, 1]$.

Аналогічно навантаження на ребро $e_i \in E$ графу позначимо $q_i \in [0, 1]$, де q – відношення кількості байт на секунду $abs(p)$, що передається потоком у деякий момент часу до максимальної кількості байт на секунду, що може

передати канал зв'язку $c(e_i)$, що відповідає ребру $(u, v)_i$ графу мережі.

Тоді при ініціації нового з'єднання побудуємо процедуру, яка знайде шлях r_0, для якого обов'язково повинна виконуватись наступна умова: $\exists se \in Se : se \in r_0$, тобто па шляху є хочу б один сенсор;

Крім необхідної умови наявності аналітика на шляху, бажано, щоб новий шлях:

1) не мав зовсім малу пропускну здатність;

2) проходив через одного з найменш завантажених аналітиків;

3) був коротким – тоді він не буде заважати іншим потокам, займаючи пропускну здатність, яку міг би використати хтось інший.

Числені значення умов якості обслуговування сильно залежать від бізнес-логіки процесу, який породив даний потік, тому ці дані визначаються на рівні додатків

При створенні нового потоку, контроллер, знаючи про стан мережі на цей момент, може розставити пріоритет цим оптимізаціям та використати відповідний алгоритм.

Наприклад, процедура, направлена на те, щоб рівномірно розподілити навантаження між усіма аналітиками, виглядає наступним чином:

1) в залишковому графі знайти аналітика з максимальною залишковою пропускною здатністю;

2) знайти найкоротший шлях від s до t, що проходять через один із сенсорів, що належать цьому аналітику;

3) якщо цей шлях не задовольняє заданим умовам якості обслуговування, обрати наступний по довжині шлях;

4) якщо таких шляхів декілька, обрати з них той, що має максимальну пропускну здатність.

Для пошуку найкоротшого шляху з обмеженням на прохід через одну із обраних вершин можна використати алгоритм Дейкстри для пошуку двох масивів найкоротших шляхів: один від s до сенсорів

обраного аналітика a:

$$l_s^a = \{dist(s, se_1^a), dist(s, se_2^a) \dots dist(s, se_n^a)\}$$

інший – від t до сенсорів обраного аналітика a:

$$l_t^a = \{dist(t, se_1^a), dist(t, se_2^a) \dots dist(t, se_n^a)\}$$

При цьому будемо зберігати шляхи, яким відповідають знайдені відстані.

Серед сенсорів аналітика оберемо той, для якого сумарна відстань $dist(s, se_i^a) + dist(t, se_i^a)$ буде найменшою, та пустимо потік через шлях, який йому відповідає.

Такий підхід до пошуку початкового шляху для новоствореного потоку дозволяє швидко знайти шлях, не зовсім оптимальний з точки зору комплексного підходу до оптимізації усіх процесів мережі. Модель SDN дозволяє нам отримати інформацію про стан усіх елементів мережі, що в свою чергу дає можливість знайти конфігурацію маршрутів, яка буде більш оптимальною з глобальної точки зору. Але на момент створення нового потоку, про нього ще недостатньо інформації, а маршрут прокладати треба одразу, тому на цей момент неможливо обрати маршрут, який згодом допоможе системі досягти загального оптимуму.

Закінчення етапу з пошуком початкових шляхів і запуск процесу глобальної оптимізації відбувається залежно від результатів пошуку маршрутів для нових процесів жадібним алгоритмом: якщо новий маршрут побудувати не вдалось або від не відповідає вимогам якості обслуговування, це є показником того, що маршрути можна оптимізувати.

2.2.1 Вплив наявності аналітика на модель транспортної мережі

На залишкову пропускну здатність шляху буде впливати і аналітик, оскільки через сенсор не буде пропущено пакети, які не було відправлено на аналіз. Тому будемо розглядати вершини, на які встановлено сенсори як вершини із обмеженою пропускною здатністю.

Ці вершини мають особливу властивість: оскільки кожен аналітик може мати декілька сенсорів, кожен новий потік, який проходить через вершину-сенсор буде зменшувати залишкову пропускну здатність не тільки тієї вершини, через яку він проходить, а і усіх вершин, які належать тому самому аналітику. Але при цьому вплив одного потоку враховується лише один раз одночасно для всіх вершин, і при прокладанні маршруту через ще одну чи більше таку вершину їх пропускні здатності вже не будуть зменшуватися.

Така особливість потребує внесення змін у звичні алгоритми пошуку маршруту у графі.

Для внесення необхідних змін спочатку перетворимо неорієнтований граф у орієнтований: кожне неорієнтоване ребро перетвориться на два протилежно направлених орієнтованих між тими самими вершинами. Пропускна здатність нових ребер буде такою самою, як і у початковому, оскільки у витій парі, яка зараз найширше використовується для прокладання комунікацій у мережах з топологією, яка нас цікавить, протилежні потоки є незалежними. Після цього кожну вершину v з пропускною здатністю c розділимо на дві вершини v_1 та v_2, між якими лежить орієнтоване ребро (v_1, v_2) із пропускною здатністю c. Усі ребра, що входили у вершину v тепер будуть входити у вершину v_1, а ті, що виходили з v тепер будуть виходити з v_2.

Тепер розглянемо модель пропускної здатності аналітика. Нехай

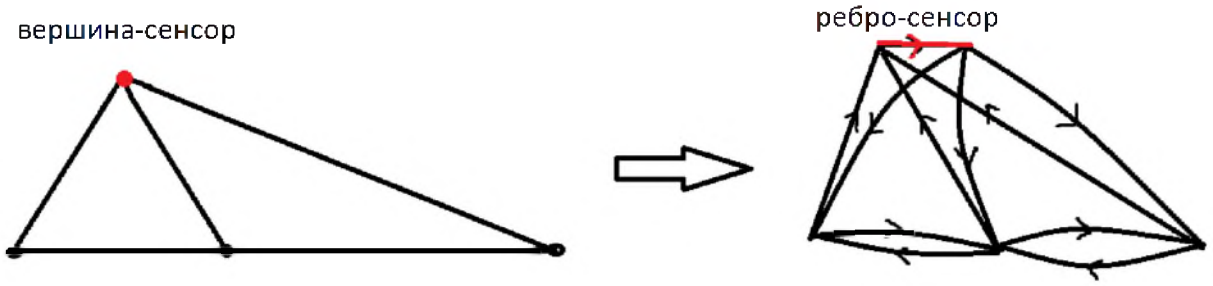

Рисунок 2.2 – Перетворення графу

контроллер прокладає новий маршрут у залишковому графі транспортної мережі, який буде проходити через декілька ребер-сенсорів, які належать різним аналітикам. Із усіх можливих маршрутів мати найбільшу пропускну здатність буде той, у якого мінімум пропускних здатностей ребер буде максимальним, але із усіх ребер-сенсорів враховується лише одне – те, що має найбільшу пропускну знатність, а пропускні здатності інших вважаються нескінченними, адже вони не затримують трафік.

Після того, як маршрут уже обраний, при підрахунку залишкового графу транспортної мережі, від пропускної здатності усіх ребер-сенсорів, які належать обраному аналізатору, одночасно буде відніматися потужність створеного потоку, оскільки пропускна здатність ребер-сенсорів показує, наскільки навантажений відповідний аналітик.

2.3 Один аналітик - багато сенсорів, один потік - багато шляхів

Як вже було сказано, рівень прикладних програм створює заявки на передачу даних та передає їх контроллеру. Ці заявки створюються для певних задач, які забезпечують виконання бізнес-логіки, для якої мережа, власне створювалась. Задачі можуть бути самими різними, і в деяких

випадках навіть сам додаток на момент створення заявки не знає, який обсяг даних йому треба передати, яка пропускна здатність каналу йому необхідна та наскільки швидко контролер повинен виконати задачу. Але в деяких випадках цю інформацію можна отримати. Наприклад, якщо планується резервування бази даних, обсяг цих даних є відомим, і контролер може отримати інформацію про нього. Якщо цей обсяг інформації досить великий, або задача потребує високої пропускної здатності каналу, може бути доцільним побудувати розгалужений маршрут для транспортування, забезпечивши таким чином високу пропускну здатність.

При цьому кожна частка потоку повинна проходити через сенсор. Спочатку ми розглянемо ситуацію, коли усі сенсори, котрі збирають дані про потік, повинні належати одному аналітику – тоді увесь потік буде оброблений в одній точці, і це мінімізує необхідність обміну інформацією між аналітиками.

Фактично це є задачею пошуку максимального потоку між двома точками графу. Із врахуванням умови на прохід усього трафіку через сенсори одного аналітика, постановку задачі можна сформулювати так:

Дано зважений орієнтований граф $(V,\ E)$, вага ребра $c(e) > 0$ – його пропускна здатність; дано дві особливі вершини $s \in V$ – джерело, та $t \in V$ – витік. Дано набір вершин $\{se_i\} \subset V$ – сенсори обраного для аналізу цього потоку аналітика.

Задача: знайти максимальний потік f із s в t, такий, що в його графі існує розріз – розбиття V на множини A та B, $s \in A$, $t \in B$, A та B не перетинаються, такий, для кожного ребра із A в B, хоча б одна з вершин, яка лежить на цьому ребрі, є сенсором (потік тільки через такі ребра може бути > 0).

Оскільки будь-який шлях із s в t обов'язково пройде через таке ребро, на усіх маршрутах буде сенсор, тобто такою постановкою задачі ми хочемо знайти підграф, через який ми зможемо пустити максимальний потік даних із s в t, і в цьому графі буде неможливо не пройти через хоча б один сенсор.

Інше формулювання задачі: знайти максимальний потік f із s в t, такий, що в його графі не існує шляхів із s в t, які не проходять через хоча б один сенсор.

Тоді зробимо таким чином: знайдемо граф максимального потоку із s в t, знайдемо у ньому усі шляхи, що не проходять через сенсори, і вилучимо їх. Для того, щоб знайти такі шляхи, вилучимо із графу максимального потоку усі сенсори se_i та знову знайдемо максимальний потік. Цей потік буде включати усі маршрути, які не проходять через сенсори.

Послідовно цей алгоритм можна записати так:

1) оберемо найменш завантаженого аналітика;

2) за допомогою алгоритму Форда-Фалкерсона знайдемо орієнтований граф максимального потоку між s та t, позначимо його G_1;

3) вилучимо з отриманого графу максимального потоку G_1 усі сенсори обраного аналітика, отримаємо граф G_2;

4) вилучимо із G_1 усі сенсори se_i – отримаємо граф G_2

5) у G_2 знайдемо максимальний потік із s в t – це буде граф G_3, який відображає маршрути, які не проходять через сенсори;

6) із G_1 вилучимо G_3 – отримаємо G_4;

7) вихід: G_4.

У G_4 немає шляхів, що не проходять через сенсор, усі такі шляхи ми вилучили. Цей граф є підграфом графу транспортної мережі, і по ньому ми будемо пропускати потік, для якого виконали алгоритм.

Тепер треба врахувати пропускну спроможність аналітика. Якщо максимальний потік, який ми знайшли, більше пропускної спроможності аналітика, ми маємо обмежити вихід потоку з s пропускною здатністю аналітика, інакше в цьому немає потреби – потік фізично не зможе перенавантажити аналітика, оскільки недостатньо пропускної спроможності самої мережі.

Для балансування навантаження на транспортну мережу, будемо розподіляти пакети прямо пропорційно значенням на ребрах графу G_3 та зворотно пропорційно довжині шляху. Тоді на всіх ребрах, що приймають

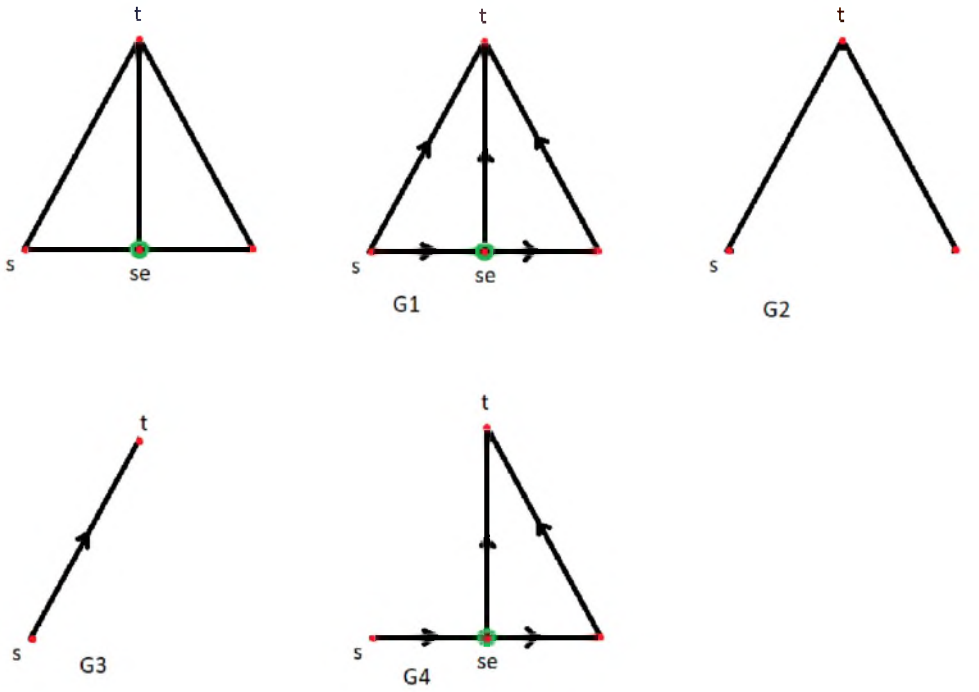

Рисунок 2.3 – Порядок пошуку маршрутів через сенсори

участь у передачі даних, залишиться вільна пропускна здатність, а довші маршрути будуть не так активно використовуватись, як коротші.

Є нюанс, пов'язаний з роботою алгоритму Форда-Фалкерсона. Якщо існує декілька варіантів побудови графу максимального потоку, серед них можуть бути такі, у яких різний сумарний потік проходить через сенсори, адже алгоритм пошуку максимального потоку ці випадки не відрізняє. Але у нашому випадку це важливо: ми хочемо обрати той варіант, у якому максимальний сумарний потік проходить через сенсори.

Тому, для коректності, на роботу алгоритму Форда-Фалкерсона накладемо уточнення: коли алгоритм шукає шляхи у залишковому графі, якщо таких шляхів декілька, він повинен з них спочатку обрати той, який проходить через сенсор і має максимальну пропускну здатність.

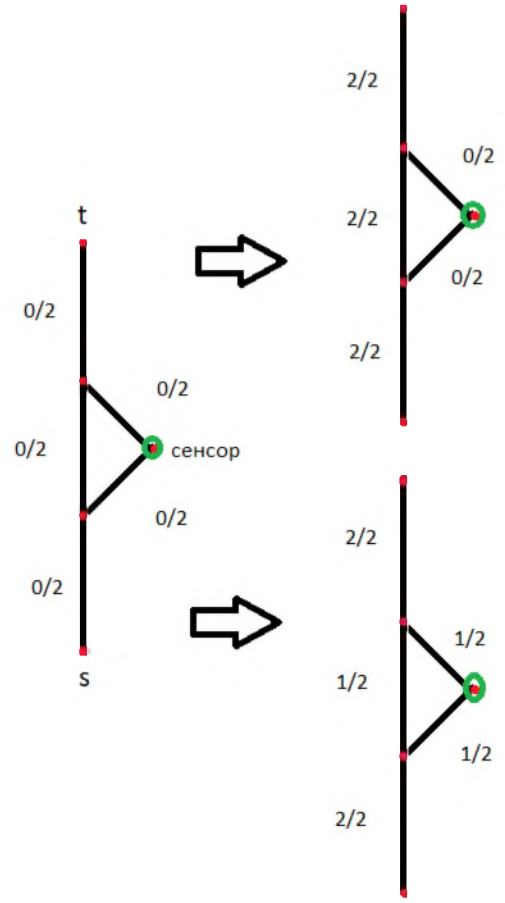

Рисунок 2.4 – Алгоритм Форда-Фалкерсона може обрати один з двох варіантів в залежності від того, який шлях буде розглянуто першим. Обидва ці варіанти нас не влаштовують.

2.4 Глобальна оптимізація

У попередніх секціях ми розглядали в основному роботу з залишковим графом транспортної мережі, тобто при побудові маршруту для нового потоку робили вигляд, що інших процесів не існує. Але після багатьох циклів створення-знищення потоків, доцільно буде переглянути увесь граф цілком. Цю задачу можна звести до задачі про багатопродуктовий потік з деякими змінами, зазначеними у попередній секції. Необхідність перегляду може виникнути у випадку

перенавантаження одного із аналітиків після появи нового потужного потоку: якщо усі аналітики навантажені приблизно порівну, новий потужний потік внесе значну нерівномірність у роботу аналітиків та транспортного рівня. Аналогічно може бути перенавантаженим одне із транспортних ребер. Процедура глобальної оптимізації покликана стабілізувати систему у такому чи подібних випадках.

Коли виникає необхідність у проведенні глобальної оптимізації, вже відомі приблизні потужності потоків, які циркулюють у мережі.

Для рішення задачі про багатопродуктовий потік необхідно сформулювати її як задачу лінійного програмування, після чого можна буде використати симплекс-метод для отримання точного рішення, або скомбінувати симплекс-метод з методом генерації стовпців для швидкого отримання приблизного результату.

Задача про багатопродуктовий потік може бути зведена до даної задачі, якщо кожну вершину покласти за сенсор, який веде до окремого аналітика з нескінченною пропускною здатністю – тоді будь який маршрут буде проходити через якогось аналітика.

2.4.1 Стійкість до відмов

На відміну від класичної схеми побудови мережі, де маршрутизація - це розподілений ітеративний процес, при якому робоча топологія мережі «обчислюється» спільно всіма пристроями, в SDN топологія як граф транспортної мережі буде зберігатися централізовано, у контроллері. Використання цієї моделі відкриває нові можливості для реакції на нештатні події, неможливі в рамках традиційного інжинірингу трафіку із використанням стандартних протоколів маршрутизації.

Розглянемо реконфігурацію мережі у випадку обриву транспортного

зв'язку між двома вузлами мережі, який відповідає деякому ребру у графі. У традиційній моделі, маршрутизатор, підключений до цього каналу, повідомить своїм сусідам про цю подію, і всі вони незалежно займуться розробкою нових маршрутів. При цьому вони будуть поширювати інформацію про зміну топології своїм сусідам, які будуть поширювати її далі. Зрештою цей ітераційний процес закінчиться, і мережа перейде в новий стан. В залежності від складності мережі і протоколів маршрутизації, що використовуються, цей процес може зайняти різний час, але враховуючи затримки при передачі інформації, цей процес може зайняти значну кількість часу. Більш того, він не є зовсім детермінований: інформація може доходити з різними затримками, що веде до різних рішень маршрутизаторів. Іншими словами, при повторному падінні того самого каналу, мережа може перейти в інший стан.

У програмно визначеній мережі цей процес є централізованим, а не розподіленим, тому розрахунок нової топології проводиться виходячи з знання про стан всієї мережі в цілому. Крім цього, оскільки створення нової топології - це чисто обчислювальний процес, він може бути виконаний значно швидше, адже тепер він не вимагає спілкування між маршрутизаторами.

Якщо було виявлено обрив каналу зв'язку, можна діяти по-різному.

1) Для усіх потоків або частин потоків, які проходили через дане ребро, знайти найкоротший маршрут між вершинами на кінцях цього ребра в обхід точки розриву. Цей варіант дозволяє швидко відновити зв'язок, не порушуючи умову проходження через сенсори, адже усі вершини, що належали маршруту до обриву ребра, будуть присутні у маршруті після відновлення зв'язку.

2) З іншого боку, таку подію як обрив зв'язку можна сприймати як один з тригерів глобальної оптимізації, перед проведенням якої ми вилучимо з графу мережі неробоче ребро. Це затримає відновлення зв'язку, але дозволить оптимально перерозподілити навантаження на

мережу.

2.4.2 Один аналітик - один сенсор, один потік - один шлях

Розглянемо спочатку ситуацію, коли у кожного аналітика є лише по одному сенсору.

Дано зважений орієнтований граф і набір потоків $\{s_i,\ t_i,\ d_i\} \in V \times V \times \mathbb{R}$, де s та t – витік та стік, d – пропускна здатність, яку потребує потік. Також дано підмножину ребер цього графу $\{se_i\} \in E$ – сенсори, через які повинні проходити потоки. Вага сенсорів – залишкова пропускна здатність відповідних аналітиків. Задача – обрати маршрути $r \in V^{|V|}$ між парами $(s_i,\ t_i)$, так, щоб кожен з цих маршрутів проходив хоча б через один сенсор, і аналітики були б завантажені найбільш рівномірно.

Якщо маршрут проходить через сенсори декількох аналітиків, враховується пропускна здатність лише одного із них. Для формалізації цієї умови введемо позначення $x_i^{se_j} \in \{0,\ 1\}$ – яка частка потоку p_i буде проаналізовано на аналітику a_j, якому відповідає сенсор se_j.

Тоді лінійний набір обмежень виглядає таким чином:

$$\forall i : \sum_j x_i^{se_j} = 1$$

, тобто увесь потік пройшов аналіз;

$$\forall se_j : \sum_i d_i x_{r_i}^{se_j} \leq c(se_j)$$

, тобто аналітики не перенавантажені;

$$\forall e \in E : \sum_{r_i:\ e \in r_i} d_i \leq c(e)$$

, тобто ребра не перенавантажені;

Цей набір обмежень допомагає визначити лінійну програму, яка може оптимізувати один із параметрів: можна мінімізувати максимальну завантаженість аналітика для розподілення навантаження, максимізувати суму пропускних здатностей обраних маршрутів, щоб реалізувати максимальний потік, або обрати інший варіант оптимізації в залежності від контексту задачі.

2.4.3 Один аналітик - багато сенсорів

Тепер розглянемо ситуацію, коли потік можна розділити. Для спрощення розглянемо варіант, коли один потік можуть обробляти різні аналітики – це спростить лінійну програму. Такий варіант актуальний для транспортної мережі, у якій максимальний потік між двома точками є великим, але маршрут виходить розгалужним, і кожна гілка має малу пропускну здатність у порівнянні з потужністю потоку, яку треба забезпечити.

Дано зважений неорієнтований граф і набір потоків $\{s_i, t_i, d_i\} \in V \times V \times \mathbb{R}$, де s_i та t_i – витік та стік, d_i – пропускна здатність, яку потребує потік. Визначено множину аналітиків A, кожному аналітику $a_j \in A$ співставлено набір вершин-сенсорів $Se^j \subset V$. У кожного аналітика є пропускна здатність $c(a_j) \geq 0$.

Маршрутом для потоку p_i тепер назвемо функцію $f_i : E \to \mathbb{R}$, яка для кожного ребра показує, яка частина цього потоку (в абсолютних одиницях) проходить через це ребро.

На f накладаються наступні обмеження:

$$\forall e \in E, \forall i : \ f_i(e) \geq 0$$

$$\forall i : \sum_{e \in out(s_i)} f_i(e) = d_i$$

$$\forall i : \sum_{e \in in(t_i)} f_i(e) = d_i$$

$$\forall i, \forall v \in V \setminus \{s_i, t_i\} : \sum_{e \in in(v)} f_i(e) = \sum_{e \in out(v)} f_i(e)$$

Тут $out(v)$ та $in(v)$, $v \in V$ – усі ребра, що відповідно виходять та входять в дану вершину;

Для того, щоб врахувати зобов'язаність трафіку повністю проходити через сенсори аналітиків із врахуванням їх обмеженої обчислювальної здатності мною був запропонований наступний підхід.

Аналогічно функції f_i введемо функцію $w_i : \ E \to \mathbb{R}$, яка показує, скільки (у абсолютних одиницях) необробленого потоку p_i проходить через ребро $e \in E$. Необроблепий потік спочатку (відразу після виходу з s) буде таким самим, як і f, але він буде зменшуватись кожен раз, коли буде проходити через сенсор, який його частину зчитає для обробки. Тобто якщо $\sum_{e \in in(se)} w_i(se) - \sum_{e \in out(se)} w_i(se) > 0$, тобто кількість необробленого трафіку зменшилась після проходу через деякий сенсор, і це показує, що цей сенсор se зчитав частину трафіку і передав своєму аналітику.

Тоді умову на відсутність перенавантаження аналітиків можна сформулювати так:

$$\forall a_j \in A : \sum_{se \in Se^j} \sum_i \left(\sum_{e \in in(se)} w_i(e) - \sum_{e \in out(se)} w_i(e) \right) \leq c(a_j)$$

Додаткові умови для коректності:

$$\forall i : \sum_{e \in out(s_i)} w_i(e) = d_i$$

$$\forall i : \sum_{e \in in(t_i)} w_i(e) = 0$$

$$\forall i, \forall e \in E : \ f_i(e) \geq w_i(e)$$

$$\forall i, \forall v \in V \setminus \{s_i, t_i\} : \sum_{e \in in(v)} w_i(e) \geq \sum_{e \in out(v)} w_i(e)$$

$$\forall i, \forall v \in V \setminus (\cup_j Se^j \cup \{s_i, t_i\}) : \sum_{e \in in(v)} w_i(e) = \sum_{e \in out(v)} w_i(e)$$

З точки зору оптимізації, важливіше за все збалансувати навантаження на аналітиків. Для балансування навантаження будемо мінімізувати навантаження на найбільш навантаженого аналітика:

$$\text{minimize} \max_{a_j} \left(\frac{\sum_{se \in Se^j} \sum_i \left(\sum_{e \in in(se)} w_i(e) - \sum_{e \in out(se)} w_i(e) \right)}{c(a)} \right)$$

Поставлена задача може бути вирішена методами лінійного програмування.

2.4.4 Локальна оптимізація

Глобальна оптимізація на усьому графі найкращі результати дасть тоді, коли існує багато можливостей для оптимізації, тобто досить довгий час оптимізація не проводилась. Вона дає можливість знайти оптимальний набір маршрутів для усіх потоків, який буде балансувати навантаження найкращим чином.

Але проблема у тому, що цей процес може займати багато часу для вирішення ситуації, у якій більшість потоків уже іде по оптимальному маршруту, тобто для них оптимізація не потрібна. Це, наприклад, може бути ситуація перенавантаження ребра, коли через одне ребро проходить багато маршрутів, і треба його розвантажити, але взагалом мережа

майже вільна.

Особливо гостро це питання встає тоді, коли мережа дійсно велика, і в ній проходять процеси, для яких призупинення їх роботи навіть на невеликий час є поганим: наприклад, дзвінки через IP-телефонію, відеоконференції та інші потокові передачі даних. В такому випадку нам потрібно якнайшвидше відновити зв'язок після падіння каналу, а глобальна оптимізація може виконуватись достатньо довго.

Тобто можна представити ситуації, коли ми хочемо оптимізувати одну невелику ділянку мережі, а проводити глобальну оптимізацію буде занадто довго.

Тому нам потрібна процедура, яке оптимізує лише невелику ділянку мережі. Назвемо цю процедуру **локальною оптимізацією**.

Метою процедури локальної оптимізації є балансування навантаження на ребра або аналітика або повне їх розвантаження при відмовах.

Оскільки локальна оптимізація є по суті глобальною оптимізацією на підграфі, спробуємо використати підхід глобальної оптимізації для вирішення цієї задачі.

Назвемо критичним елементом ребро, сенсор, аналітика або транзитну вершину (комутатор або машину-хоста), на яких трапилась подія, яка потребує негайної зміни маршрутів, що проходять через цей елемент. Якщо відбулась відмова елементу, цей елемент тимчасово викидається з графу транспортної мережі, а маршрути, що через нього проходять, перенаправляються іншим чином. Якщо відбулось перенавантаження, елемент з графу не видаляється.

Серед ситуацій можуть бути:

– Для ребра – обрив зв'язку або перенавантаження, коли потік потребує більшої пропускної спроможності.

– Для сенсора це відмова. При цьому вершина не викидається з графу, але вона перестає бути вершиною-сенсором і стає просто транзитною вершиною. Потоки, що проходили через цей сенсор і на ньому зчитувались, потребують зміни маршруту.

– Для аналітика це відмова або перенавантаження. Тут можна провести і глобальну оптимізацію, бо робота аналітиків є критичною для моделі, що розглядається в даній роботі, але часткова оптимізація теж актуальна, коли поруч є інші аналітики або глобальна оптимізація надто трудомістка. Відмову аналітика розглянемо окремо.

– Відмова транзитної вершини означає також відмову сенсорів, що їй належать, і вона виключається із графу.

Критичними точками для критичних елементів назвемо вершини графу, пов'язані з цим елементом. Для ребра це будуть вершини, між якими прокладено це ребро, для аналітика – вершини, на яких розміщені його сенсори.

Тепер виділимо з графу транспортної мережі підграф, у рамках якого будемо проводити локальну оптимізацію. Спочатку введемо міру цього підграфу, яка буде визначати його розмір.

Назвемо **порядком локальної оптимізації** натуральне число n. Вершина графу транспортної мережі буде належати підграфу локальної оптимізації тоді і тільки тоді, коли відстань між нею та найближчою критичною точкою не перевищує n.

У підграф також включаються усі ребра, що з'єднують включені вершини, і розглядаємо аналітиків, чиї сенсори потрапили у підграф. Згодом розглянемо підходи до вибору порядку, але загальний підхід – збільшувати поки нема достатньої кількості шляхів для задовільного перерозподілу навантаження.

Розглянемо цей підграф у контексті усієї транспортної мережі.

На момент визнання елементу, навколо якого будується підграф, критичним, у мережі визначена множина потоків P, і для кожного потоку $p_i \in P$ побудовано маршрут від s_i до t_i. Будемо вважати маршрути цих процесів нерозгалуженим, оскільки будь-який розгалужений маршрут можна розбити на композицію нерозгалужених.

Локальна оптимізація буде працювати швидше глобальної, оскільки нам потрібно розглянути лише ті процеси, маршрути яких проходять через

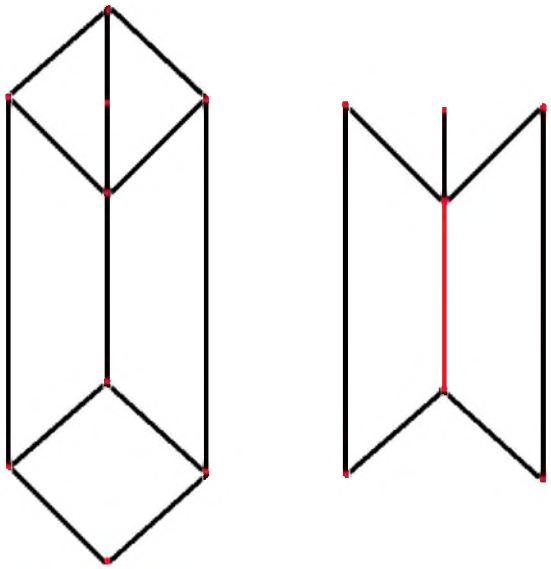

Рисунок 2.5 – Центральне ребро вийшло з ладу. Розглянемо локальну оптимізацію порядку 1

хоча-б одне ребро, включене у підграф. У кожного з таких маршрутів буде точка входу у підграф і точка виходу з нього.

Нехай $G_{cr} = (V_{cr}, E_{cr})$ – підграф навколо критичного елементу, який ми розглядаємо, $P_{cr} \subset P$ – множина процесів, у маршрутах яких існує хоча б одне ребро $e \in E_{cr}$, тобто маршрут цього потоку проходить через підграф.

Для потоку $p_i \in P_{cr}$ назвемо точкою входу $v_{p_i}^{in} \in V_{cr}$ таку вершину, яка лежить на маршруті, побудованому для p_i до появи критичного елемента, для якої наступне ребро маршруту вже належить G_{cr}, а попереднє ще ні.

Аналогічно, $v_{p_i}^{out} \in V_{cr}$ – точка виходу, якщо попереднє ребро лежить у G_{cr}, а наступне – ні.

Якщо маршрут перетинає підграф декілька разів, таких точок входу і виходу може бути декілька пар, ми будемо розглядати пари окремо.

Якщо для деякого потоку $p_i \in P$, витік t_i належить підграфу, ми будемо вважати t_i точкою виходу. Так само, якщо $s_i \in V_{ex}$, то будемо вважати, що $v_{p_i}^{in} = s_i$.

Аналогічно до того, яка ми моделювали частку необробленого потоку

$w_i(e)$у глобальній оптимізації, для кожного потоку із P_{cr} па момент входу у підграф та виходу з нього із усього трафіку певна частка буде вже проаналізована.

Тоді для певного потоку значення w у вхідній та вихідній точках відрізняються, це означає, що відповідна частина потоку була проаналізована, і під час локальної оптимізації цю частку потоку також треба провести через сенсор, аналітик якого обробить цю частку трафіку.

З кожного потоку $p_i \in P_{ex}$ можна виділити «підпотік», джерелом якого є $s = v_{p_i}^{in}$, а витоком – $t = v_{p_i}^{out}$. На відміну від звичайних потоків, у нього є компонента, яка вже проаналізована до потрапляння у G_{ex}, і компонента, яку планується проаналізувати після виходу із цього підграфу. Третя компонента повинна бути проаналізована усередині G_{ex}.

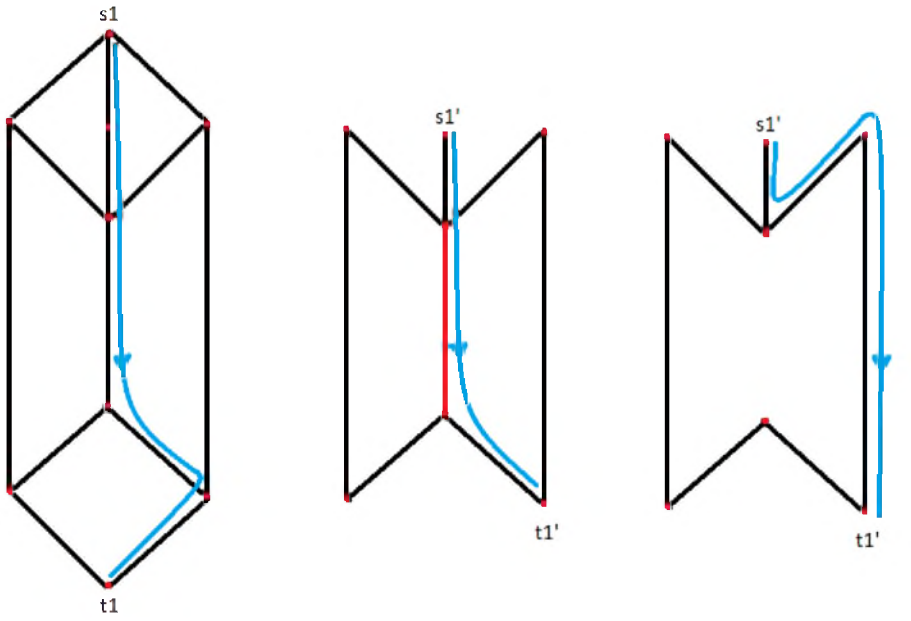

Рисунок 2.6 – Виділення підпотоку та пошук нового шляху при відмові ребра

Тому для кожного $p \in P_{ex}$ ми представимо ту частину його маршруту, яка проходить всередині графу G_{ex}, як композицію двох нових потоків: по одному іде оброблений трафік, який ми повинні провести через аналітика, а по іншому – необроблений. Розділивши кожен потік із P_{ex} на два, ми

отримаємо дві нові множини потоків: P^f_{ex} для компонент, які не потрібно аналізувати у G_{ex}, та P^w_{ex} – компоненти, які потрібно проаналізувати у цьому підграфі.

Тоді задача локальної оптимізації зводиться до двох задач: для P^w_{ex} потрібно вирішити задачу глобальної оптимізації па графі G_{ex}, а для потоків із P^f_{ex} – задачу багатотоварного потоку па тому самому графі.

Отже, ми звели задачу локальної оптимізації до вже відомих задач.

2.5 Перенесення та розділення аналітиків при віртуалізації мережі

Крім можливості вільної маршрутизації трафіку, SDN дозволяє швидше та з меншими втратами проводити міграцію віртуальних машин між фізичними машинами. Технологія живої міграції (Live Migration) дозволяє при цьому не втрачати мережеві з'єднання. Якщо помістити аналітиків у віртуальні машини, цю технологію можна використати для оптимізації розміщення аналітиків відносно сенсорів, підвищення стійкості до відмов, балансування навантаження тощо.

Нехай в нас є контейнер (або віртуальна машина), який містить програмне забезпечення аналітика. В залежності від обчислювальних ресурсів, які виділено цьому контейнеру, він має певну пропускну здатність як аналітик.

Оскільки контроллер може отримати інформацію про фізичні машини у мережі, ми можемо дізнатися, які обчислювальні ресурси вільні на кожній вершині графа транспортної мережі, а отже можемо підрахувати, яку потенційну пропускну здатність ця вершина може надати, якщо розмістити на ній контейнер з аналітиком.

Тоді можна розглядати кожну фізичну машину у мережі як потенційне

місце розміщення аналітика. Це доцільно для конкретної машини, якщо у неї майже постійно наявні значні вільні обчислювальні ресурси.

Тоді ми можемо звести задачу до задачі глобальної оптимізації, якщо будемо вважати, що кожна обчислювальна машина в мережі є вершиною-сенсором із своїм аналітиком, пропускна здатність якого визначається в залежності від вільних обчислювальних ресурсів цієї машини. Алгоритм підрахунку пропускної здатності залежить від реалізації аналітика.

Але якщо процес розгортання контейнерів виявиться занадто довгим, краще розглядати можливість обробки на такому контейнері як опцію на крайній випадок. Рішення про створення нового контейнера-аналітика приймається як спосіб розвантажити інших аналітиків, коли їх можливостей недостатньо для обробки нових запитів. Тоді при пошуку маршруту для нового потоку контроллером або глобальній оптимізації, з'являється нова опція – створення нового аналітика.

Тоді якщо створюється новий потік, і його потужність в сумі з потужностями потоків, які були створені раніше, більша за сумарну пропускну здатність аналітиків, для лінійної програми не буде рішення. Але ми можемо розмістити контейнери з аналітиками на фізичні машини, які стабільно мають невикористовувані обчислювальні ресурси, і за рахунок таких нових віртуалізованих аналітиків забезпечити достатню сумарну пропускну здатність.

Віртуалізація також підвищує стійкість до відмов. При відмові сенсора можна перенаправити на інший сенсор або іншого аналітика, при відмові аналітика - на іншого аналітика, або створити нового аналітика, якщо мережа віртуалізована/контейнеризована. Підходи до прийняття рішень при відмові залишаються такими самими, як і у класичних мережах, SDN лише дає можливість зробити їх більш гнучкими та дає більше інформації для прийняття рішень.

ВИСНОВКИ

В даній роботі було побудовано математичну модель програмно-конфігурованої мережі з можливістю аналізу трафіку та методи маршрутизації, направлені на підвищення спостережуваності мережі шляхом побудови маршрутів через сенсори аналітичної платформи.

У результаті виконання даної роботи були розглянуті можливості, що надають програмно конфігуровані мережі для підвищення спостережуємості.

Побудовано математичні моделі різних типів з орієнтацією на аналіз трафіку та методи маршрутизації, що забезпечують різні оптимізації. Проведено огляд ситуацій, в яких доцільно використовувати певні моделі та методи. Запропоновано методи підвищення надійності, засновані на особливостях програмно-конфігурованих мереж.

Запропоновані рішення не є єдиними можливими, існує багато алгоритмів, які можна модифікувати та використати для досягнення поставлених цілей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 3-е издание — М.: «Вильямс», 2013. — 1328 с. — ISBN 978-5-8459-1794-2.

2. Алгоритмы. Теория и практическое применение / Род Стивенс. – Москва : Издательство «Э», 2016. – 544с.

3. Сущенко С. П. Математические модели компьютерных сетей. – Томск : Издательский Дом Томского государственного университета, 2017. – 272 с.

4. Abha Kumari.A Survey of Controller Placement Problem in Software Defined Networks / Abha Kumari, Ashok Singh – arXiv:1905.04649 – 2019.

5. P. Saint-Andre. XMPP: the definitive guide. / P. Saint-Andre , K. Smith, R. Troncon – «O'Reilly Media, Inc.» – 2009.

6. W. Zhou. «Rest api design patterns for SDN northbound api» / W. Zhou, L. Li, M. Luo, W. Chou – Advanced Information Networking and Applications Workshops (WAINA) – 28th International Conference on. IEEE – 2014.

7. PaulGöransson. Software Defined Networks (Second Edition). A Comprehensive Approach / Paul Göransson, Chuck Black, Timothy Culver – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/B9780128045558000053.

8. Mehrnoosh Monshizadeh. Detection as a Service: An SDN Application / Mehrnoosh Monshizadeh, Vikramajeet Khatri, Raimo Kantola – 19th International Conference on Advanced Communication Technology (ICACT) – 2017.

9. Скороходов Александр. Cisco ACI. Инфраструктура, ориентированная на приложения – Cisco Connect – 2015 – Режим доступу: https://https://www.cisco.com/c/dam/assets/global/KZ/ciscoconnect/pdf/D1S1_ACI_part1_askorokh_final.pdf

10. Орлов Є. В. Програмно-конфігуровані мережі: архітектура, міжнародна стандартизація – Наукові записки Українського науково-дослідного інституту зв'язку – 2014 – №4(32)

11. Г.М. Жолткевич. Багатопродуктова транспортна задача континуального лінійного програмування – Системи обробка інформації – 2006.

12. Pengyuan Du. Traffic optimization in software defined naval network for satellite communications / Pengyuan Du, Fan Pang, Torsten Braun, Mario Gerla, Ceilidh Hoffmann, Jae H. Kim – 2017 IEEE Military Communications Conference (MILCOM) – 2017.

13. Семеновых А. А. Сравнительный анализ SDN-контроллеров / А. А. Семеновых, О.Р. Лапонина – International Journal of Open Information Technologies, vol. 6, no.7 – 2018.

14. T. Leighton. Fast Approximation Algorithms for Multicommodity Flow Problems / Leighton T., Makedon F., Plotkin S., Stein C., Tardos E., Tragoudas S. – Journal of Computer and System Sciences, volume 50, issue 2 – 1995.

15. Michal Pióro. Routing, Flow, and Capacity Design in Communication and Computer Networks / Michal Pióro, Deepankar Medhi – San Francisco : Morgan Kaufmann Publishers – 2004.

16. Мухин О. И. Моделирование систем [Електронний ресурс] / Олег Игоревич Мухин – Режим доступу до ресурсу: http://stratum.ac.ru/education/textbooks/modelir/contents.html.

17. Вентцель Е. С. Исследование операций. М., «Советское радио» – 1972 – Режим доступу: http://stu.sernam.ru/book_rop.php

18. Cynthia Barnhart. Multicommodity Flow Problems / Cynthia Barnhart, Niranjan Krishnan, Pamela H. Vance – Encyclopedia of Optimization. Springer, Boston, MA – 2008.

19. L. R. Foulds. A multi-commodity flow network design problem / Foulds L. R. – Transportation Research Part B: Methodological, Volume 15, Issue 4 – 1981. Volume 15, Issue 4

20. Системний аналіз [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: http://er.nau.edu.ua/bitstream/NAU/21000/23/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_CA_12.pdf

21. A. Moore. Descriminators for use in flow based Classification / A. Moore, D. Zuev and M. Crogan – Queen Marry University of London – 2005 – ISSN 1470-5559 – Режим доступу: https://pdfs.semanticscholar.org/16a8/118a338eed235406f2c6e0877345af6e20bc.pdf

22. Дэниел Гмах. Управление вычислительной мощностью и прогнозирование потребностей центров обработки данных. – Режим доступу: https://www.bytemag.ru/articles/detail.php?ID=11622