**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ПРОПУСКНОЇ СПРОМОЖНОСТІ В МЕРЕЖАХ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ QUALITY OF SERVICE (QOS)»**

на здобуття освітнього ступеня магістр
за спеціальності 123 Комп'ютерна інженерія
(код,найменування спеціальності)
освітньо-професійної програми Комп'ютерні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій ГУМЕНЮК
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти гр.КСДМ-62
Андрій ГУМЕНЮК
(ім'я, ПРІЗВИЩЕ)

Керівник: В'ячеслав ЧЕРЕВИК
к.т.н., доцент *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
науковий ступінь, вчене звання *(ім'я, ПРІЗВИЩЕ)*

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерної інженерії
Ступінь вищої освіти «Магістр»

Спеціальність 123 Комп'ютерна інженерія
Освітньо-професійна прорама Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерної інженерії
Наталія ЛАЩЕВСЬКА
(ім'я, ПРІЗВИЩЕ)
"____" _____2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гуменюку Андрію Володимировичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Дослідження методів оптимізації пропускної спроможності в мережах з використанням технології Quality of Service (QoS)
керівник роботи В'ячеслав ЧЕРЕВИК к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "19" 10 2023 р. №145
2. Строк подання кваліфікаційної роботи
3. Вихідні дані кваліфікаційної роботи:
3.1. Технічна документація стосовно методів оптимізації пропускної спроможності в мережах.
3.2. Інтернет-ресурси стосовно програмно-конфігурованих мереж.
3.3. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
4.1. Аналіз існуючих методів оптимізації пропускної спроможності в мережах з використанням QoS.
4.2. Підвищення якості обслуговування потоків трафіку, що потребує пропускної здатності.
4.3. Забезпечення якості обслуговування потоків трафіку, чутливих до затримок.
5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання “19” жовтня 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	**Примітка**
1.	Підбір технічної літератури	.2023р. .2023р.	Виконано
2.	Аналіз існуючих методів оптимізації пропускної спроможності в мережах з використанням QoS	.2023р. .2023р.	Виконано
3.	Підвищення якості обслуговування потоків трафіку, що потребує пропускної здатності	.2023р. .2023р.	Виконано
4.	Забезпечення якості обслуговування потоків трафіку, чутливих до затримок	.2023р. .2023р.	Виконано
5.	Оформлення роботи, висновки	.2023р. .2023р.	Виконано
6.	Розробка демонстраційного матеріалу, доповідь	.2023р. .2023р.	Виконано

Здобувач вищої освіти Андрій ГУМЕНЮК
(підпис) *(ім’я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи В’ячеслав ЧЕРЕВИК
(підпис) *(ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступеня магістр: 95 стор., 3 табл., 30 рис., 21 джерел.

Мета роботи – підвищення ефективності використання мережевих ресурсів із забезпеченням заданих параметрів якості обслуговування за допомогою оптимізації пропускної спроможності в мережах.

Об'єкт дослідження – методи оптимізації пропускної спроможності в мережах з використанням QoS.

Предмет дослідження – програмно-конфігуровані мережі.

Короткий зміст роботи: В дипломній роботі було досліджено використання альтернативних шляхів маршрутизації для вибраних потоків, які мають вимоги QoS; використання контролера SDN для встановлення правил переадресації в комутаторах, які можуть досягти більшої пропускної здатності. Було розроблено нові механізми для гарантування затримки додатків залежно від своєчасної доставки даних датчиків і керуючих сигналів. Було розглянуто нові механізми передачі великих даних у середовищі хмарних обчислень. Замість того, щоб використовувати один шлях TCP для передавання даних, було досліджено, як дозволити програмі налаштувати кілька TCP-шляхів для однієї програми для досягнення більшої пропускної здатності. В роботі було оцінено ці нові механізми за допомогою експериментів і порівнено їх з існуючими підходами.

КЛЮЧОВІ СЛОВА: КОМП'ЮТЕРНІ МЕРЕЖІ, ПРОГРАМНО-КОНФІГУРОВАНІ МЕРЕЖІ, QOS, ПРОПУСКНА СПРОМОЖНІСТЬ, TCP, АРХІТЕКТУРА МЕРЕЖІ, ПРОТОКОЛ OPENFLOW, МАРШРУТИЗАЦІЯ

ABSTRACT

The text part of the qualification work for obtaining a master's degree: 95 pages, 30 figures, 21 sources.

The purpose of the work is increasing the efficiency of the use of network resources with the provision of the specified parameters of the quality of service by means of the optimization of bandwidth in networks.

The object of research is methods of bandwidth optimization in networks using QoS.

The subject of research is software-configured networks.

Summary of the work: The thesis investigated the use of alternative routing paths for selected flows that have QoS requirements; using an SDN controller to establish forwarding rules in switches that can achieve higher throughput. New mechanisms have been developed to guarantee application latency depending on the timely delivery of sensor data and control signals. New mechanisms for transferring big data in the cloud computing environment were considered. Instead of using a single TCP path for data transfer, it was explored how to allow an application to configure multiple TCP paths for a single application to achieve higher throughput. The work evaluated these new mechanisms with the help of experiments and compared them with existing approaches.

KEY WORDS: COMPUTER NETWORKS, SOFTWARE-CONFIGURED NETWORKS, QOS, BANDWIDTH CAPACITY, TSR, NETWORK ARCHITECTURE, OPENFLOW PROTOCOL, ROUTING

ЗМІСТ

ВСТУП

Незважаючи на величезний успіх і впровадження комп'ютерних мереж в останні десятиліття, традиційна мережева архітектура не відповідає деяким вимогам багатьох додатків. Одним з особливих недоліків є відсутність зручних методів надання гарантії якості обслуговування (QoS) різним мережевим додаткам. У цій магістерській роботі досліджується нові механізми програмно-визначених мереж (SDN) для забезпечення QoS цільових мережевих потоків.

Дане дослідження сприяє забезпеченню QoS-підтримки додатків у трьох аспектах. По-перше, було досліджено використання альтернативних шляхів маршрутизації для вибраних потоків, які відповідають вимогам QoS. Замість того, щоб використовувати найкоротший шлях за замовчуванням, який використовується поточними протоколами маршрутизації мережі, в роботі досліджується використання контролера SDN для встановлення правил переадресації в комутаторах, які можуть досягти більшої пропускної здатності.

По-друге, було розроблено нові механізми для гарантування затримки цих додатків залежно від своєчасної доставки даних датчиків і керуючих сигналів. Новий механізм попередньо виділяє черги з вищим пріоритетом у маршрутизаторах/комутаторах і резервує ці черги для трафіку керування/датчиків.

По-третє, було досліджено, як змусити додатки скористатися можливістю, наданою SDN. Зокрема, вивчаються нові механізми передачі великих даних у середовищі хмарних обчислень. Замість того, щоб використовувати один шлях TCP для передавання даних, було досліджено, як дозволити програмі налаштувати кілька TCP-шляхів для однієї програми для досягнення більшої пропускної здатності. В роботі було оцінено ці нові механізми за допомогою експериментів і порівнено їх з існуючими підходами.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ОПТИМІЗАЦІЇ ПРОПУСКНОЇ СПРОМОЖНОСТІ В МЕРЕЖАХ З ВИКОРИСТАННЯМ QOS

Передача даних відіграє вирішальну роль у багатьох комп'ютерних додатках і стала невід'ємною частиною нашого повсякденного життя, особливо завдяки повсюдному використанню мобільних пристроїв. Ці програми з комунікаційним компонентом мають широкий спектр вимог до базових комп'ютерних мереж. Традиційна передача файлів з низькою швидкістю передачі даних або інтерактивний віддалений вхід на загальний комп'ютер має дуже обмежені вимоги до мережі, за винятком надійності, яка обробляється протоколом управління передачею (TCP) шляхом повторної передачі втрачених пакетів. На противагу цьому, зовсім недавно такі додатки, як відео на вимогу, передача великих даних, кіберфізичні системи, хмарні обчислення і т.д., пред'являють більш жорсткі вимоги з точки зору часу, що вимірюються як затримка або час завершення передачі даних, відомі як вимоги до якості обслуговування (QoS). Наприклад, програми для телеконференцій вимагають, щоб аудіо/відео дані передавалися з низькою затримкою пакетів, щоб зберегти відчуття взаємодії в реальному часі. Потокове відео вимагає достатньої пропускної здатності для відтворення відео з невеликою буферизацією або без неї. В іншому випадку це призведе до переривання відтворення відео на приймальному кінці. Вони можуть мати інші вимоги до QoS, такі як низьке тремтіння (зміна затримки) і низький коефіцієнт втрати пакетів. Для додатків для передачі великих даних не мають жорстких термінів доставки пакетів, як у потоковому відео. Однак можна покластися на те, що мережі забезпечать достатню пропускну здатність, щоб передача могла завершитися протягом розумного проміжку часу. У кіберфізичних системах, таких як розумні мережі та домашні мережеві системи, може знадобитися контролювати стан обладнання та надсилати сигнали керування, щоб розпочати операцію. Ці дані датчиків і керуючі сигнали повинні вчасно змінювати свої пункти призначення, щоб замкнути цикл зворотного зв'язку, хоча обсяг даних може бути невеликим. Все це створює проблеми для базових мереж.

Традиційна мережева архітектура не забезпечує підтримку QoS для передачі даних. Навіть якщо загальна пропускна здатність мережі може відповідати вимогам мережевих додатків, якість послуг, що надаються для цих додатків, може бути незадовільною через найкращі зусилля поточної мережі. Було проведено досить багато досліджень, присвячених механізмам забезпечення QoS в традиційних мережах. Більшість з них є теоретичними дослідженнями і не були розгорнуті в Інтернеті в скільки-небудь значних масштабах.

Одна з причин полягає в тому, що пропрієтарні протоколи, розгорнуті в мережевому обладнанні вендорами, в основному фіксовані і не можуть бути змінені кінцевими користувачами. Виробники мережевих пристроїв зазвичай не хочуть виставляти свої внутрішні реалізації на загальний огляд і, як правило, тримають їх закритими, що ускладнює програмування мережі. Це обмежує гнучкість і ускладнює керування мережею. Мережеві адміністратори не можуть змінювати свою поведінку відповідно до вимог кінцевих програм. Мережеві дослідники не можуть модифікувати програмне забезпечення на мережевих маршрутизаторах і комутаторах, щоб експериментувати, впроваджувати і розгортати нові протоколи для забезпечення підтримки QoS, необхідної додаткам.

Software Defined Networking (SDN) – це нещодавно запропонована нова парадигма реалізації мережевих функцій. Він розділяє площину управління і площину даних. Площина даних відповідає за функції пересилання даних, тоді як площина керування визначає, як пересилаються дані. Функція площини управління централізована в SDN контролер, який встановлює правила переадресації, які будуть використовуватися комутаторами на площині даних. Відкритий стандарт дозволяє користувачам писати модулі контролерів для визначення поведінки площини даних мережі. Крім того, контролери надають інтерфейс API, щоб користувачі могли писати зовнішні прикладні програми, які взаємодіють з контролером і мають можливість вказувати, що комутатори пересилання даних повинні робити з пакетами даних.

SDN надає дослідникам нову можливість експериментувати з новими мережевими механізмами для надання послуг, необхідних додаткам. У цій роботі

досліджується нові механізми SDN для забезпечення QoS цільових мережевих потоків. З даного аналізу зрозуміло, що існують дві фундаментальні вимоги до критичних за часом додатків, тобто пропускна здатність і затримка. Для пропускної здатності, замість того, щоб використовувати алгоритм маршрутизації на основі призначення за замовчуванням, було досліджено альтернативні шляхи маршрутизації для вибраних потоків. За рахунок переваг гнучкості, що надається SDN, було встановлено правила в комутаторах пересилання даних з метою використання шляху пересилання з більш високою доступною пропускною здатністю, замість використання найкоротшого шляху за замовчуванням, який використовується поточними протоколами маршрутизації мережі. Для досягнення мети необхідно було стежити за станом мережі та розробити метод з'ясування поточної доступної пропускної здатності за відповідними посиланнями.

Інша проблема, — це затримка. Цікаві саме ті додатки, які потребують своєчасної передачі даних датчиків і керуючих сигналів, щоб вся система могла функціонувати належним чином. По суті, не можна вирішити проблему, викликану межею швидкості світла. Однак можна спостерігати, що одним із факторів, що сприяють цьому, є те, що ці пакети можуть чекати в черзі, коли є конкуруючий трафік, який використовує той самий шлях. В роботі було розроблено новий механізм, який попередньо виділяє черги з вищим пріоритетом у маршрутизаторах/комутаторах і резервує ці черги для трафіку керування/датчиків. Це еластичне резервування в тому сенсі, що якщо смуга пропускання не використовується зарезервованим трафіком, вона може бути використана іншими.

Однак зарезервований трафік має вищі пріоритети. Таким чином, це може гарантувати, що трафік керування/датчиків не затримується на перевантажених маршрутизаторах для досягнення мети утримання їх наскрізної затримки в межах ліміту.

Хоча мережа може бути оснащена механізмами для встановлення нових шляхів маршрутизації, в роботі також було досліджено, як змусити програми скористатися можливістю, наданою SDN. З точки зору застосування, було

вивчено нові механізми передачі великих даних у середовищі хмарних обчислень. Замість того, щоб використовувати один шлях TCP для передавання даних, було досліджено, як дозволити програмі налаштувати кілька шляхів TCP для однієї програми. Було диференційовано короткі та довгі потоки та адаптивно визначено, чи створювати підпотоки для TCP-з'єднання та скільки допоміжних підпотоків створювати. Володіючи знаннями про топологію мережі та доступні потужності, що надаються контролером SDN, було розроблено алгоритм, який покращує загальну пропускну здатність для довгих потоків, не штрафуючи ті короткі потоки, які не потребують використання багатопроменевого TCP (MPTCP).

У цій роботі запропоновано три механізми, які спрямовані на покращення надання QoS для обраних мережевих потоків. Було визначено вимоги до певного мережевого трафіку та представлено методи та системи для досягнення цілей їх продуктивності. Ці механізми засновані на SDN. Запропоновані механізми були розроблені та оцінені на випробувальних стендах. Основними досягненнями магістерської роботи є:

- підвищення якості обслуговування, що надається потокам трафіку, які мають вимоги до пропускної здатності. В роботі запропоновано рішення на основі SDN для безперервного моніторингу стану мережі та динамічного налаштування шляхів переадресації для потоків трафіку, що вимагають пропускної здатності;

- забезпечення якості обслуговування трафіку, чутливого до затримок. Запропоновано структуру для управління та перенаправлення потоків трафіку, які повинні передаватися з підвищеним пріоритетом для дотримання термінів. Ця структура вміщує різні класи потоків трафіку з різним рівнем вимог;

- максимізація пропускної здатності великих потоків за рахунок використання багатопроменевих TCP і SDN. Запропоновано нову архітектуру, яка дозволяє великим потокам трафіку досягати більш високої пропускної здатності за рахунок використання декількох шляхів. Даний підхід дозволяє додаткам динамічно створювати нові підпотоки, які пересилаються через найменш перевантажені шляхи контролером SDN.

1.1 Якість обслуговування

Забезпечення якості обслуговування не було однією з цілей при початковому проектуванні Інтернету. Однак інтернет-додатки (наприклад, потокове передавання мультимедіа, онлайн-ігри, телеконференції тощо) розвивалися з часом, і їхня потреба в гарантії QoS стала очевидною. Хтось може заперечити, що надмірне виділення мережевих ресурсів для задоволення вимог QoS економічно доцільніше, ніж заміна існуючої мережевої архітектури. Однак протягом багатьох років було докладено багато зусиль, спрямованих на забезпечення QoS. Інтегровані послуги (IntServ) і диференційовані послуги (DiffServ) були двома основними пропозиціями, хоча вони не були успішно розгорнуті у великих масштабах.

1.1.1 IntServ

IntServ надає гарантію якості обслуговування, резервуючи ресурси на кожному маршрутизаторі на шляху, пройденому пакетами потоку. Ця архітектура складається з двох частин. По-перше, специфікація потоку, яка описує транспортний потік і вимоги до нього. Потік визначається як «розрізняється потік пов'язаних дейтаграм, який є результатом однієї активності користувача і вимагає однакового QoS». По-друге, Resource Reservation Protocol (RSVP), який є протоколом сигналізації, що використовується між хостами та маршрутизаторами для запиту резервування ресурсів (наприклад, пропускної здатності). Для того, щоб забезпечити запитуваний QoS, маршрутизаторам необхідно реалізувати контроль трафіку. Архітектура IntServ визначає три компоненти контролю трафіку: планувальник пакетів, класифікатор і контроль допуску. Планувальник пакетів використовує набір черг для керування пересиланням різних пакетних потоків. Класифікатор відображає кожен вхідний пакет на деякий клас. Контроль допуску приймає або відхиляє новий запит QoS на потік трафіку.

Незважаючи на те, що IntServ надає гарантію QoS, він має деякі недоліки, які перешкоджають широкому впровадженню цієї архітектури. Для цього потрібно підтримувати інформацію про стан потоку, яка пропорційна кількості потоків. Це впливає на масштабованість, особливо у великих мережах, таких як Інтернет. Він також вимагає, щоб усі маршрутизатори на шляху підтримували три компоненти: контроль трафіку та протокол RSVP. Ці обмеження призводять до другої пропозиції, DiffServ.

1.1.2 DiffServ

DiffServ був запропонований для подолання труднощів при впровадженні IntServ. У ньому передбачені механізми агрегування транспортних потоків в класи. Класи грубозернистого трафіку покращують масштабованість, на відміну від дрібнозернистих потоків трафіку IntServs. Класифікація здійснюється за допомогою поля Differentiated Services Code Point (DSCP) у заголовках IPv4 та IPv6. DSCP було введено для заміни поля ToS в IPv4. Класифіковані пакети позначені таким чином, щоб їх могли ідентифікувати маршрутизатори та переслати відповідним чином. Всі пакети, які мають однакове значення DSCP, об'єднуються в один клас під назвою Behavior Aggregate і будуть однаково оброблятися всіма маршрутизаторами в домені. Класифікацію та маркування потрібно виконувати тільки на краю мережі. Однак усі маршрутизатори повинні впроваджувати поведінку Per-Hop Behaviors (PHB), яка описує властивості для переадресації класів трафіку (наприклад, мінімальну пропускну здатність).

Поведінка за перехопленням гарантує, що трафік із високим пріоритетом отримає сприятливе ставлення порівняно з іншими класами трафіку. Зазвичай це досягається шляхом впровадження черг з різним пріоритетом і формування трафіку (обмеження швидкості). Ця архітектура не надає твердої гарантії QoS, як IntServ.

1.1.3 Метрики QoS

Вимоги до якості обслуговування, як правило, вказуються в угодах про рівень обслуговування (SLA), які визначають гарантовану продуктивність мережі, яка повинна бути забезпечена для додатків клієнтів постачальниками послуг. Продуктивність мережі вимірюється за набором атрибутів, які включають:

- гарантована мінімальна пропускна здатність. На пропускну здатність, що досягається потоками трафіку, впливає кілька факторів, таких як пропускна здатність каналу зв'язку та перевантаженість мережі. Забезпечення гарантованої мінімальної пропускної здатності для певних потоків трафіку (наприклад, потокове відео в реальному часі) гарантує, що такі потоки доставлятимуть дані відповідно до потреб на приймальну сторону;

- гарантована максимальна затримка. Наскрізна затримка — це загальний час, необхідний для передачі одного пакета від вихідного хоста до вузла призначення. Він включає затримку передачі, затримку поширення, затримку в черзі та затримку обробки. Передача голосу через IP (VoIP), телеконференції та онлайн-ігри в Інтернеті є прикладами мережевих додатків, у яких підвищена затримка впливає на їх продуктивність;

- гарантований максимальний коефіцієнт втрати пакетів. Перевантаження мережі може призвести до збою доставки деяких пакетів. Це може статися, коли буфери в мережевих пристроях досягають максимальної ємності. В цьому випадку роутерам і комутаторам доведеться скидати деякі пакети. TCP, будучи надійним транспортним протоколом, забезпечує цілісність даних, що передаються, використовуючи підтвердження і ретрансляція втрачених пакетів. Однак пропущені пакети впливають на продуктивність протоколу TCP, оскільки він вважається сигналом перевантаження (визначається тайм-аутами повторної передачі та дублюванням підтверджень). У відповідь на сигнал перевантаження алгоритм контролю перевантаження TCP зменшить швидкість надсилання потоку TCP, щоб уникнути перевантаження;

- гарантоване максимальне тремтіння (зміна затримки). Умови мережі змінюються з часом, що може призвести до різної затримки для пакетів, що належать до одного потоку трафіку. Така зміна затримки не є бажаною і може вплинути на якість обслуговування багатьох програм, таких як потокове аудіо/відео та онлайн-ігри в Інтернеті.

Окрім надмірного виділення мережевих ресурсів (що не є економічно оптимальним рішенням), постачальники послуг можуть використовувати такі механізми, як резервування ресурсів на кожному вузлі на шляху, пройденому пакетами даних, і маршрутизація з урахуванням QoS. Резервування ресурсів може передбачати надання вищого пріоритету певним потокам трафіку над іншими. Маршрутизація з урахуванням QoS намагається спрямувати потоки трафіку через шляхи, які задовольняють вимогам QoS.

Перш ніж перейти до розгляду різних типів проблем маршрутизації QoS, описано представлення мережі та її властивості. Мережа представлена у вигляді орієнтованого графа G(V, E), де V - множина всіх вузлів мережі, а E - множина всіх зв'язків між вузлами. Кожна ланка $e \in E$ має пов'язані властивості: e_b є доступна пропускна здатність, e_d — затримка, e_l — співвідношення втрат пакетів (тобто відсоток втрачених пакетів до загальної кількості пакетів; співвідношення 0 означає відсутність втрачених пакетів). Точність цих атрибутів дуже важлива для продуктивності алгоритмів маршрутизації QoS. Стан мережі та вимірювання необхідно оновити, щоб алгоритми маршрутизації знайшли найбільш підходящі шляхи з достатніми ресурсами, які відповідають необхідним параметрам QoS.

Задача на пошук шляху повинна повертати шлях P від вихідного вузла s до вузла призначення d, який задовольняє необхідним параметрам QoS. Визначимо наступні функції для шляху P :

$$D(P) = \sum_{e \in P} e_d$$

$$B(P) = \min_{\forall_{e \in P}} e_b$$

$$L(P) = 1 - \prod_{e \in P} (1 - e_l)$$

Функція затримки є адитивною, тоді як функція смуги пропускання увігнута. Функція коефіцієнта втрат пакетів є мультиплікативною, але може бути замінена на адитивну, взявши логарифм відношення .

1.1.4 Маршрутизація QoS

Проблема пошуку шляху в QoS-маршрутизації може включати один або кілька параметрів QoS. Алгоритми QoS-маршрутизації широко вивчені в літературі. Кожен необхідний параметр може бути як задачею обмежень, так і задачею оптимізації. Наприклад, задача з обмеженням пропускної здатності визначається як знаходження шляху P такого, що кожна ланка на шляху e ∈ P має доступну ємність, більшу або рівну необхідному параметру пропускної здатності. Задача найширшого шляху, задача оптимізації пропускної здатності, визначається як знаходження шляху P, який має найбільшу доступну пропускну здатність (тобто максимізацію вузького місця відповідного шляху). У таблиці 1.1 наведено приблизний перелік задач маршрутизації QoS, які можуть бути розв'язані за поліноміальний час. У найпростішій формі для розв'язання задачі та знаходження відповідного шляху можуть бути використані один параметр QoS, варіації алгоритмів пошуку найкоротшого шляху (наприклад, алгоритм Дейкстри або алгоритм Беллмана-Форда). Деякі проблеми зі складеними параметрами (наприклад, обмежена пропускна здатність Задача найменшої затримки) може бути розв'язана за поліноміальний час, запустивши алгоритм найкоротшого шляху (ваги ребер графа є вимірюваннями затримки) і видаливши зв'язок, який має доступну ємність меншу за обмеження пропускної здатності.

Таблиця 1.1 - Приклад задач маршрутизації QoS, розв'язуваних за поліноміальний час

Атрибути	Параметри	Обмеження	Оптимізації	Нотатки
Пропускна здатність	B_{min}	$B(P) > B_{min}$		Обмежена пропускна здатність
Затримки	D_{max}	$D(P) \leq D_{max}$		Обмежені затримки
Втрата пакетів	L_{max}	$L(P) \leq L_{max}$		Обмежен Втрата пакетів
Пропускна здатність			$\arg\max_p (B(P))$	Найширший шлях
Затримки			$\arg\min_p (D(P))$	Найменша затримка
Втрата пакетів			$\arg\min_p (L(P))$	Найменше Втрата пакетів
Пропускна здатність, затримка	B_{min}	$B(P) > B_{min}$	$\arg\min_p (D(P))$	Обмежені-Пропускна здатність з найменшою затримкою
Пропускна здатність, втрата пакетів	B_{min}	$B(P) > B_{min}$	$\arg\min_p (L(P))$	Обмежені-Найменша втрата пакетів пропускної здатності
Затримка, пропускна здатність	D_{max}, B_{min}	$D(P) \leq D_{max}$ $B(P) > B_{min}$		Обмежені delay-пропускна здатність

Існують і інші задачі складено-метричної маршрутизації, які, як відомо, є NP-Повними. До таких задач можна віднести:

- проблеми з багатофункціональним шляхом (Multi-Constrained-Path MCP). Для заданої мережі G(V, E), де кожна ланка (u, v) ∈ E має m адитивних ваг $w_i(u, v) \leq 0$, i = 1, ..., m, і задано m обмежень адитивних параметрів C_i, i = 1, ..., m,

задача формулюється як знаходження шляху P ∈ P', де P' — множина всіх можливих шляхів від вихідного вузла s до вузла призначення d такий, що:

$$\forall p \in P', W_i(p) \leq C_i \text{ для } i = 1, ..., m$$

де:

$$W_i(p) = \sum_{(u,v)\in p} W_i(u,v)$$

Прикладом задач MCP є задача з обмеженою затримкою. Мета полягає в тому, щоб знайти шлях таким чином, щоб затримка була меншою за параметр обмеження затримки, а тремтіння була меншою за параметр обмеження джиттера. Слід зазначити, що можна мати кілька шляхів, які задовольняють усім обмеженням. Будь-який такий шлях вважається можливим вирішенням цієї проблеми.

- задачі з множинним обмеженням оптимального шляху (Multi-Constrained Optimal Path MCOP). Для заданої мережі G(V, E), де кожна ланка (u, v) ∈ E має m адитивних ваг $w_i(u, v) \leq 0$, i = 1, ..., m, а задано m обмежень адитивних параметрів C_i, i = 1, ..., m і параметр адитивної вартості W_k, задача формулюється як знаходження шляху P ∈ P', де P'- множина всіх можливих шляхів від вихідного вузла s до вузла призначення d такий, що:

(a) $\forall p \in P', W_i(p) \leq C_i$ для i = 1, ..., m

(b) $W_k(P)$ мінімізується над усіма можливими шляхами, що задовольняють (a).

де:

$$W_i(p) = \sum_{(u,v)\in p} W_i(u,v)$$

Прикладом задач MCOP є задача найменшого тремтіння з обмеженою затримкою. Мета полягає в тому, щоб знайти шлях такий, щоб затримка була меншою за параметр обмеження затримки, а тремтіння було зведено до мінімуму.

У таблиці 1.2 наведено приблизний перелік NP-повних задач маршрутизації QoS. Для розв'язання цих задач доводиться використовувати алгоритми евристики та апроксимації.

Таблиця 1.2 Приклад NP-повних задач маршрутизації QoS

Метрик	Параметри	Обмеження	Оптимізації	Нотатки
Затримки Коливання	$Dmax$, $Jmax$	$D(P) \leq Dmax$ $J(P) \leq Jmax$		Обмежені затримка-тремтіння
Затримки Втрата пакетів	$Dmax$, $Lmax$	$D(P) \leq Dmax$ $L(P) \leq Lmax$		Обмежені Втрата пакетів із затримкою
Затримки Коливання	$Dmax$	$D(P) \leq Dmax$	arg minP ($J(P)$)	Обмежені-Затримки Найменше - тремтіння
Коливання Втрата пакетів	$Jmax$	$J(P) \leq Jmax$	arg minP ($L(P)$)	Обмежені-Коливання Найменша втрата пакетів

1.2 Програмно-конфігуровані мережі (SDN)

1.2.1 Ранні спроби перед SDN

До того, як SDN став популярною темою досліджень у галузі мереж, було обговорено та запропоновано кілька спроб, які поділяють певні схожі ідеї, для вирішення проблем традиційної мережевої архітектури. Ці спроби внесли свій вклад в різні аспекти в популярні в даний час архітектури SDN. Ідеї та концепції варіюються від розв'язки площини управління та площини прямування до досягнення певного рівня програмованості в мережах.

Платформа управління маршрутизацією (The Routing Control Platform RCP) була запропонована для вирішення деяких проблем з існуючим механізмом маршрутизації в рамках автономних систем (АС). Внутрішня архітектура

протоколу прикордонного шлюзу (iBGP), що використовується в АС, вимагає повносітчастої конфігурації, яка не масштабується на великі мережі. Хоча використання ієрархії Routing Reflectors (RR) допомагає уникнути проблеми масштабованості, воно спричиняє такі проблеми, як коливання протоколу, постійні цикли та складність конфігурації. Ці проблеми ускладнюють управління автономними системами з точки зору зміни конфігурації, діагностика та усунення помилок переадресації. Ці проблеми виникають через те, що рішення щодо маршрутизації приймаються маршрутизаторами, які не мають повного огляду всієї мережі.

В архітектурі RCP процес прийняття рішень BGP реалізований на логічно централізованій платформі. Платформа управління маршрутизацією відокремлена від площини IP-переадресації. Основна мета цієї платформи полягає в тому, щоб централізовано виконувати рішення щодо вибору маршруту, а не змушувати маршрутизатори виконувати цю роботу. RCP дозволяє уникнути вищезгаданих проблем, обчислюючи рішення щодо маршрутизації на основі повного уявлення про всю топологію мережі та доступні маршрути. Ця інформація збирається за допомогою існуючих протоколів BGP та Interior Gateway Protocol (IGP). У цій архітектурі RCP є три модулі: IGP Viewer, BGP Engine і Route Control Server. IGP Viewer підтримує актуальне представлення топології IGP. BGP Engine відповідає за вивчення маршрутів BGP з кожного маршрутизатора. Сервер управління маршрутами використовує інформацію, отриману з двох інших модулів, для обчислення найкращого маршруту для кожного маршрутизатора. Після того, як процес вибору маршруту виконується централізовано, Route Control Server зв'язується з маршрутизаторами через механізм BGP і встановлює нові записи переадресації, які відповідають обраним маршрутам. Зв'язок між Routing Control Platform і маршрутизаторами здійснюється за допомогою існуючих стандартних протоколів (BGP і IGP) без будь-яких модифікацій або введення нових протоколів.

4D-проект був однією з перших спроб просунути ініціативу роз'єднання архітектури мережі в різні площини. Чистий аркуш 4D підхід до контролю та

управління мережею запропонував екстремальний принцип проектування, розділивши логіку рішень маршрутизації та пересилання пакетів. Це вважається крайнім проектним моментом, оскільки пересилання пакетів і контроль логіки було тісно пов'язана в існуючих мережевих пристроях. Проблеми з поточною архітектурою Інтернету викликані складністю площин управління. Це пов'язано з тим, що логіка управління і переадресація пакетів об'єднані в розподілені маршрутизатори і комутатори. Замість того, щоб доповнювати попередні будівельні блоки для вирішення поточних проблем, команда 4D-проекту запропонувала підхід з чистого аркуша, який забезпечує альтернативну перспективу поступової еволюції комп'ютерних мереж.

Провідні принципи цього підходу до проектування зосереджені на задоволенні цілей на рівні мережі, наявності загальномережевого вигляду топології та забезпеченні прямого контролю над роботою мережевих пристроїв. Запропонована 4D-архітектура з чистого аркуша розділяє мережеві функції на 4 площини: площини даних, виявлення, поширення та прийняття рішень. Площина даних призначена для обробки окремих пакетів на основі конфігурованих правил, продиктованих площиною рішень. Площина виявлення відповідає за збір інформації про топологію мережі та інші мережеві вимірювання. Площина розподілу служить надійним каналом зв'язку між площиною прийняття рішень і площиною даних. Вона використовується для встановлення правил на мережевих пристроях для керування процесом обробки пакетів. Площина прийняття рішень діє як «мозок» мережі і має на меті замінити площину управління в традиційній мережевій архітектурі. Вона складається з логічно централізованих контролерів, розміщених на декількох серверах. Основним призначенням цього є прийняття всіх рішень, які керують і контролюють мережу. Щоб зробити це ефективно, він використовує інформацію, зібрану площиною виявлення (глобальний вигляд топології мережі та мережеві вимірювання в реальному часі). Вихідні дані площини прийняття рішень надходять у вигляді станів обробки пакетів, які конфігуруються в мережевих пристроях на площині даних площиною розподілу

Ethane вважається прямим попередником OpenFlow. Проект продовжив попередню роботу SANE. SANE також був підходом до розробки з чистого аркуша. Однак SANE було складно розгорнути в реальному світі, оскільки це вимагає зміни всієї мережевої інфраструктури. Ethane пом'якшив цю проблему, підтримавши поступове розгортання. Це не вимагає зміни всієї інфраструктури, а комутатори Ethane можуть бути поступово розгорнуті в існуючій мережевій інфраструктурі.

Проектування Ethane наслідував приклад 4D-проекту. Він розв'язує площину керування та площину даних. Він також використовує логічно централізований контролер, який має доступ до всієї мережі. Основна увага цього централізованого контролера приділяється забезпеченню дотримання глобальних мережевих політик. Іншим компонентом є шар Ethane перемикачів. Ці комутатори дуже прості і містять лише таблицю потоків і захищений канал зв'язку з централізованим контролером. Ця проста конструкція комутаторів є основою комутаторів OpenFlow.

Перемикач Ethane пересилає пакети на основі відповідного запису таблиці потоку. Якщо пакет не був зіставлений з будь-яким записом в таблиці потоків, що є звичайним випадком для першого пакета будь-якого потоку в цій системі, то він пересилається централізованому контролеру. Контролер встановить відповідні записи в таблиці потоків на перемикачах Ethane. Встановлені записи таблиці потоків базуються на аналізі пакета контролером. Основною метою контролера є забезпечення дотримання глобальних мережевих політик. Цілі проектування Ethane (забезпечення дотримання політики корпоративного рівня) і розмір їх цільової інфраструктури (невеликі мережі кампусів) роблять реактивний режим розумним підходом до проектування. Однак це рішення не масштабується на великі мережі.

1.2.2 Модель SDN

Software-Defined Networking – це нова мережева архітектура, яка має на меті створити проривне вдосконалення способів проектування та управління комп'ютерними мережами. Основна концепція SDN полягає в поділі мережевих функцій на дві різні площини: площину управління і площину даних. Рішення щодо керування рухом приймаються прикладними програмами (так званими контролерами) у площині керування, тоді як переадресація трафіку здійснюється мережевими пристроями в площині даних. Зв'язок між площиною керування та площиною даних здійснюється за допомогою стандартизованих протоколів. OpenFlow, що є найбільш вдало реалізованим протоколом SDN. Однією з переваг централізованого мережевого контролера є наявність глобального представлення топології мережі та моніторингу її вимірювань. Це дозволяє контролеру приймати більш обґрунтовані рішення щодо маршрутизації. У традиційній мережевій архітектурі обробка трафіку базується на пакетах. Мережеві пристрої приймають рішення про переадресацію, використовуючи адресну інформацію в пакетах даних, але дані зазвичай надсилаються у вигляді потоку від хоста до хоста. Програмно-визначені мережі використовують потік як основну одиницю для обробки трафіку. На рисунку 2.1 показана спрощена архітектура SDN.

В епоху, що передувала SDN, рішення щодо маршрутизації QoS в основному оброблялися в мережевих пристроях. Запропоновані алгоритми можна згрупувати в три категорії: вихідна маршрутизація (яка виконується на вихідному вузлі), розподілена маршрутизація (обчислення шляху розподіляється між декількома вузлами) та ієрархічна маршрутизація (логіка маршрутизації та стан мережі розподіляються між кластерами вузлів). Переваги та недоліки кожного підходу обговорювалися в літературі. Однак поява SDN змінює спосіб обробки рішень щодо маршрутизації. Вузлам мережі не потрібно підтримувати глобальний стан мережі або обчислювати маршрути QoS. Ці завдання можуть бути делеговані площині управління.

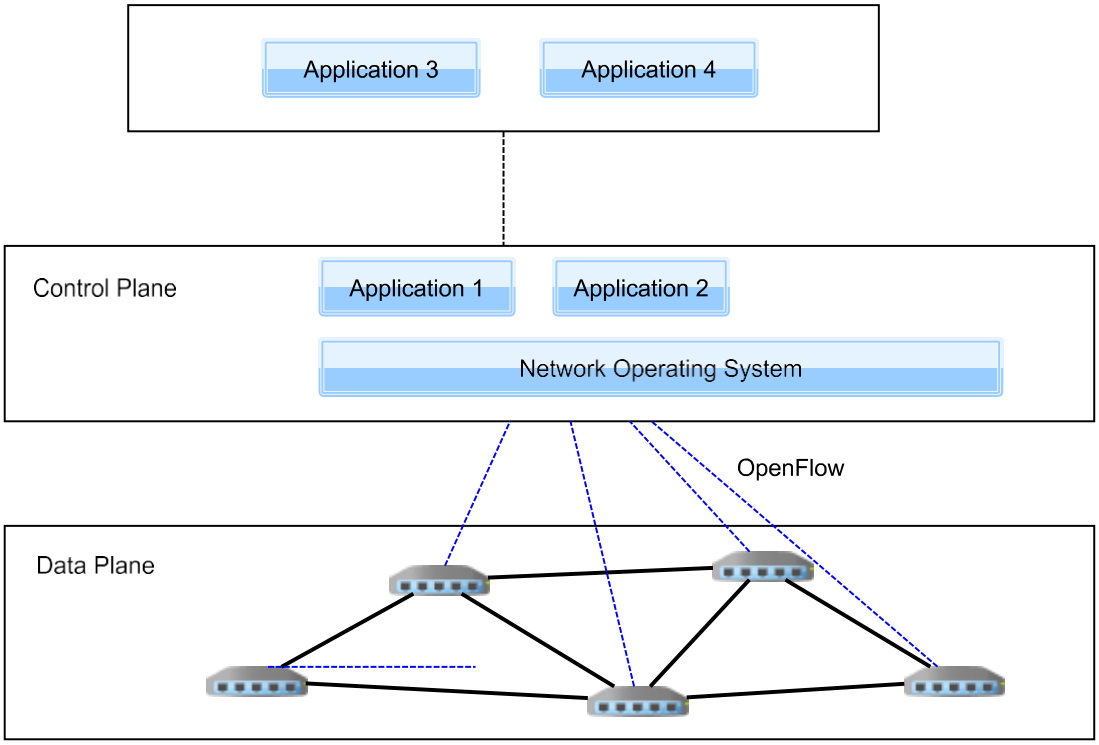


Рисунок 1.2 - Спрощена архітектура SDN

1.2.3 Характеристики SDN

Програмно-конфігурована мережа має деякі відмінні характеристики, які визначають, чим вона відрізняється від традиційної мережевої архітектури. До таких характеристик можна віднести:

- розв'язка функцій управління і прямої передачі. Хоча ці дві функції тісно пов'язані в традиційних мережевих пристроях, поділ між площиною керування та площиною пересилання даних є ключовою особливістю програмно-визначених мереж;

- логічно централізоване управління мережею. Інструменти та протоколи управління розподілені в традиційній мережевій архітектурі таким чином, що робить завдання управління дуже складним, а іноді і неефективним;

- відкриті стандарти. Для сприяння розвитку нових мережевих протоколів та інструментів, популярні архітектури програмно-визначених мереж базуються на відкритих стандарти, таких як протокол OpenFlow;

- програмована мережа. Здатність програмувати мережу була однією з важливих мотивацій програмно-визначених мереж. Традиційні мережеві архітектури забезпечують обмежену підтримку для досягнення обмеженого рівня мережевої програмованості;

- на основі потоку. У традиційній мережевій архітектурі обробка трафіку базується на пакетах. Мережеві маршрутизатори та комутатори приймають рішення щодо переадресації, використовуючи інформацію про адресу в пакетах даних, але дані зазвичай надсилаються у вигляді потоку від хоста до хоста. Програмно-визначені мережі використовують потік як основну одиницю для обробки трафіку.

1.2.4 Переваги SDN

Програмно-конфігуровані мережі обіцяють багато переваг для комп'ютерних мереж наступного покоління. До таких переваг можна віднести:

- простіше керування завдяки відокремленню логіки керування мережею від базових мережевих пристроїв (наприклад, комутаторів, маршрутизаторів, проміжних коробок) і надання централізованих інструментів керування для мережевих адміністраторів;

- інновації в нових мережевих протоколах та інструментах. Використання відкритих стандартних протоколів дозволяє створювати нові інноваційні продукти;

- гнучкість мережевих конфігурацій і застосування нових змін у відповідь на вимоги до трафіку;

- тестування та розгортання нових мережевих протоколів та алгоритмів набагато простіше, ніж в традиційний мережевій інфраструктурі, де виділена інфраструктура, як правило, потрібна лише для тестування;

- віртуалізація мережевих функцій. Багато мережевих функцій, які зазвичай реалізовані в спеціальних пристроях (наприклад, брандмауери, балансувальники

навантаження, системи виявлення вторгнень тощо), можуть бути реалізовані у віртуальних машинах.

1.2.5 OpenFlow

Архітектура OpenFlow дотримується принципу поділу між площиною управління та площиною пересилання даних. Спочатку OpenFlow був запропонований для того, щоб дослідники могли розробляти та тестувати нові мережеві протоколи та рішення в мережах кампусів. В даний час це найпопулярніша архітектура для програмно-визначених мереж. Деякі комерційні продукти забезпечують підтримку архітектури OpenFlow. Нижній рівень цієї архітектури, площина пересилання даних, складається з комутаторів OpenFlow. Комутатор OpenFlow містить як мінімум три основні компоненти. По-перше, одна або кілька таблиць потоку. По-друге, захищений канал зв'язку між комутатором і контролером. По-третє, підтримка протоколу OpenFlow.

Записи потоку в таблиці потоків визначають, як обробляється відповідний пакет. Наприклад, пакет може бути переспрямований на певний порт, інкапсульований і відправлений контролеру або відкинутий. Кожен запис ланцюжка зазвичай будується з таких полів:

- зіставити поля: для визначення пакетів правил, що належать ланцюжку. Ці правила зазвичай збігаються з інформацією, що міститься в заголовках пакетів або номері порту;

- інструкції: дія, пов'язана з правилом, яка визначає, як обробляється пакет;

- лічильники: статистика у вигляді лічильників потоку (наприклад, кількість прийнятих/переданих пакетів і байтів, тривалість потоку);

- Priority: для визначення відповідного пріоритету записів потоку;

- тайм-аути: перемикач, коли запис потоку закінчився. Існує два типи тайм-аутів: жорсткий тайм-аут (загальний час з моменту встановлення) і м'який тайм-аут (час простою);

- Cookie: елемент даних, обраний контролером. Файли cookie не впливають на обробку пакетів у площині даних. Контролер може використовувати файли cookie для фільтрації потоків на основі різних типів завдань або залежно від того, який модуль/додаток у площині керування їх встановив;

- flags: використовується для керування записами ланцюжка. Наприклад, прапорець OFPFF_SEND_FLOW_REM використовується для надсилання повідомлення контролеру при видаленні відповідного запису ланцюжка.

Коли пакет приймається комутатором з підтримкою OpenFlow, його властивості зіставляються з набором атрибутів, що зберігаються в записах таблиць потоків. Якщо знайдено збіг, то відповідні інструкції для запису таблиці додаються до списку дій. Якщо пакет відповідає більш ніж одному запису, до списку дій додаються інструкції запису з найвищим пріоритетом. Комутатор OpenFlow може мати кілька таблиць потоку, які обробляються як конвеєр. Можна мати інструкцію для явного спрямування пакета в іншу таблицю. Якщо пакет не збігся з жодним записом потоку, він називається table-miss. Застосовані дії table-miss залежать від конфігурації таблиці. Пакет може бути інкапсульований і переданий контролеру. Однак можна обробити пакет, що не збігається, за допомогою IP-переадресації, якщо комутатор підтримує як OpenFlow, так і переадресацію без OpenFlow.

Протокол OpenFlow визначає зв'язок між контролером і комутатором OpenFlow. Він містить набір протокольних повідомлень, якими обмінюються контролер і комутатори по захищеному каналу зв'язку. Використовуючи протокол OpenFlow, пульт дистанційного керування може встановлювати, оновлювати або видаляти записи потоку в таблиці потоків всередині комутатора OpenFlow. Це також дозволяє контролеру отримувати статистику. Багато продуктів та інструментів були розроблені з використанням архітектури OpenFlow.

Більшість ранніх систем програмно-визначених мереж були розроблені для роботи в реактивному режимі (наприклад, Ethane). У реактивному режимі перший пакет потоку передається контролеру, який встановлює нові записи потоку для

управління рештою пакетів цього потоку. У проактивному режимі системи не вимагають відправки першого пакета контролеру. Замість цього контролер змінює записи потоку на основі інших вхідних даних, таких як зміни топології або реагування на статистику. Архітектура OpenFlow підтримує як реактивний режим, так і проактивний режим. У реактивному режимі буде затримка продуктивності, так як перший пакет кожного потоку повинен буде відправитися в контролер для подальшої обробки. Хоча це зниження продуктивності може бути доступним у невеликих мережах кампусів, воно не забезпечує реалістичного рішення для великих виробничих мереж. Це також збільшує проблему масштабованості, оскільки контролер повинен обробляти більшу кількість пакетів у великих мережах.

1.2.6 Open vSwitch

Open vSwitch - це програмна реалізація мережевого комутатора, яка може бути використана у віртуалізованому середовищі. Вона забезпечує зв'язок між віртуальними машинами та фізичними мережевими інтерфейсами в гіпервізорі. Цей програмний комутатор забезпечує підтримку багатьох мережевих протоколів і стандартів, включаючи OpenFlow. Open vSwitch може працювати як базовий комутатор L2 або може бути інтегрований у віртуалізоване середовище. Для підтримки віртуалізованого розгортання Open vSwitch експортує інтерфейси для маніпулювання таблицями пересилання та керування станами конфігурації. Це дозволяє віддалені процеси для безпосереднього доступу до конфігурацій і пересилання таблиць і їх пересилання. Open vSwitch також експортує локальний інтерфейс керування підключенням, який дозволяє рівню віртуалізації маніпулювати своєю топологічною конфігурацією. Open vSwitch можна використовувати для створення одного логічного комутатора між кількома відкритими vSwitch, що працюють на окремих фізичних серверах. Це також корисно для подолання обмежень мобільності віртуальних машин між різними підмережами IP.

2 ПІДВИЩЕННЯ ЯКОСТІ ОБСЛУГОВУВАННЯ ПОТОКІВ ТРАФІКУ, ЩО ПОТРЕБУЄ ПРОПУСКНОЇ ЗДАТНОСТІ

Комунікаційні технології є одним із ключових компонентів поточних і майбутніх додатків, забезпечуючи надійні та ефективні можливості двостороннього зв'язку. Різні додатки генерують великі обсяги трафіку даних з різними вимогами до якості обслуговування. Вони можуть бути чутливими до затримок, пропускної здатності або можуть обслуговуватися службою найкращих зусиль. Управління в режимі реального часу в промислових системах може бути легко досягнуто за допомогою виділених мереж. Однак використання виділених мереж не є можливим рішенням у більш загальних налаштуваннях мережі. Мережева архітектура з комутацією пакетів обслуговує кілька додатків, де різні потоки трафіку даних співіснують один з одним. Нові проблеми, з якими стикається існуюча мережева інфраструктура та протоколи управління, включають жорсткі вимоги до часу, високу надійність та гнучкість керування.

У цій главі було розроблено структуру, засновану на програмно-визначених мережах для надання критично важливих послуг зв'язку. Архітектура SDN, як правило, вважається найбільш застосовною до тих доменів додатків, в яких адміністратор має повний контроль. Приклади включають центри обробки даних, домашні мережі, програми інтелектуальних мереж, автоматизацію розподілу та мікромережі.

Трафік даних можна розділити на дві категорії. Один – це трафік, який не має вимог до QoS (так званий трафік найкращих зусиль або потоки найкращих зусиль), а інший – це трафік, який має одну або кілька вимог до QoS, таких як пропускна здатність, затримка, тремтіння або коефіцієнт втрати пакетів (так званий критичний трафік або критичні потоки). Основна увага в цьому розділі приділяється наданню кращого обслуговування критично важливих потоків, які мають потребу в пропускній здатності, шляхом динамічного налаштування шляхів переадресації в площині даних. З цією метою керуюча програма буде стежити за станом мережі і направляти критичні потоки по кращому шляху,

встановлюючи правила OpenFlow на комутатори. Було розроблено алгоритм пошуку шляху і реалізовано його як модуль для контролера Floodlight. Оцінка продуктивності показує, що даний підхід може значно покращити пропускну здатність, отриману критичними потоками, порівняно з алгоритмом маршрутизації найкоротшим шляхом, який використовується в сучасних мережах.

Вивчення проблеми маршрутизації QoS для мультимедійних додатків можна датувати роботами Ванга і Кроукрофта. Вони довели, що знаходження шляху, що задовольняє декільком адитивним метрикам, є NP-повним. Потім вони запропонували кілька алгоритмів обчислення шляху, використовуючи або централізовану маршрутизацію джерела, або розподілену маршрутизацію стрибок за стрибком.

OpenQoS - це конструкція контролера, запропонована для забезпечення гарантії QoS для мультимедійних додатків. Мережевий трафік поділяється на мультимедійні потоки та потоки даних. OpenQoS реалізує динамічну маршрутизацію мультимедійного трафіку для розміщення його на маршрутах з гарантованим QoS. Потоки даних обробляються за допомогою традиційного алгоритму найкоротшого шляху. Динамічна маршрутизація QoS формується як задача з обмеженим найкоротшим шляхом (CSP). Вона формулюється як знаходження шляху, який мінімізує функцію витрат, якщо загальна затримка буде меншою або рівною заданому значенню Dmax (необхідному для мультимедійного потоку). Показником вартості посилання є показник затримки плюс показник перевантаження. Показник перевантаження для кожного каналу зв'язку залежить від використання пропускної здатності. Він встановлюється на 0, якщо використання пропускної здатності менше 70%. Однак у своїй пропозиції вони використовували кількість переходів як міру затримки (тобто міра затримки для всіх посилань встановлена на 1). Хоча у OpenFlow не передбачено підтримки вимірювання затримки збору даних, існують методи для досягнення цього завдання, наприклад, використання пакетів зондування. Оскільки задача CSP є NP-повною, вони запропонували використовувати алгоритм апроксимації під назвою алгоритм агрегованої вартості на основі релаксації Лагранжа (LARAC)

для знаходження хорошого маршруту. Вони реалізували маршрутизацію QoS поверх контролера Floodlight.

VSDN (Video over SDN) — це ще один фреймворк, який спрямований на забезпечення гарантії QoS для мультимедійних потоків у програмах потокового відео. Він розкриває деякі API QoS, які використовуються як відправником, так і одержувачем для запиту QoS для потокового відео. Централізований контролер розраховує можливий шлях на основі вимог QoS і продовжує стежити за мережевими ресурсами. Однак VSDN вимагає модифікації існуючих комутаторів OpenFlow для підтримки запропонованого гарантованого обслуговування.

2.1 Фреймворк на основі SDN для налаштування шляхів для потоків, що вимагають пропускної здатності

Представляємо мережу у вигляді орієнтованого графа G(V, E), де V - множина всіх комутаторів OpenFlow і кінцевих хостів в площині даних, а E - множина всіх зв'язків між цими вузлами. Зв'язки представлені у вигляді впорядкованих пар. Наприклад, (v_1, v_2) представляє єдину ланку в топології, де v_1 - вихідний вузол, а v_2 - вузол призначення. Кожному зв'язку присвоюється атрибут обчисленої доступної ємності.

Варто звернути увагу, що атрибути можуть відрізнятися для посилань (v_1, v_2) і (v_2, v_1) через асиметричні посилання.

Сформульовано задачу як знаходження найкоротшого шляху від вихідного вузла s до вузла призначення d так, щоб мінімальна доступна пропускна здатність шляхових зв'язків була більшою за необхідну пропускну здатність для критичного потоку трафіку.

В роботі розроблено два модулі (Status Monitoring та QoS-based Path Setup) для контролера Floodlight, як показано на рисунку 3.1. Floodlight пропонує гнучку систему завантаження модулів, яка дозволяє розширювати. Базові модулі прожектора взаємодіють з перемикачами OpenFlow. Модулі розширення можуть використовувати функціональні можливості, що надаються базовими модулями.

Першим модулем розширення є Status Monitoring, який збирає мережеві вимірювання як основу для другого модуля. Вони містять інформацію про використання, наприклад використану пропускну здатність, для кожного каналу. Другим модулем розширення є Path Setup на основі QoS, який обчислює шлях, використовуючи топологію мережевого графіка та зібрані вимірювання. Він знаходить шлях, який задовольняє певним вимогам до пропускної здатності, і встановлює правила OpenFlow на комутатори в площині даних.

2.1.1 Моніторинг стану

Специфікація комутатора OpenFlow визначає список лічильників, які повинні підтримуватися комутаторами з підтримкою OpenFlow. У список входять потокові лічильники і портові лічильники. Нас цікавлять портові лічильники, зокрема лічильники переданих байтів. Контролер може отримати значення цих лічильників, відправивши комутатору повідомлення про запит статистики та отримавши повідомлення у відповідь, яке містить кількість переданих байтів на момент запиту. Слід зазначити, що ці вимірювання не будуть на 100% точними до моменту їх отримання контролером, через затримку повідомлень запит-відповідь та затримку обробки у контролера. Тим не менш, вони дають хорошу оцінку використання посилань, що відстежуються.

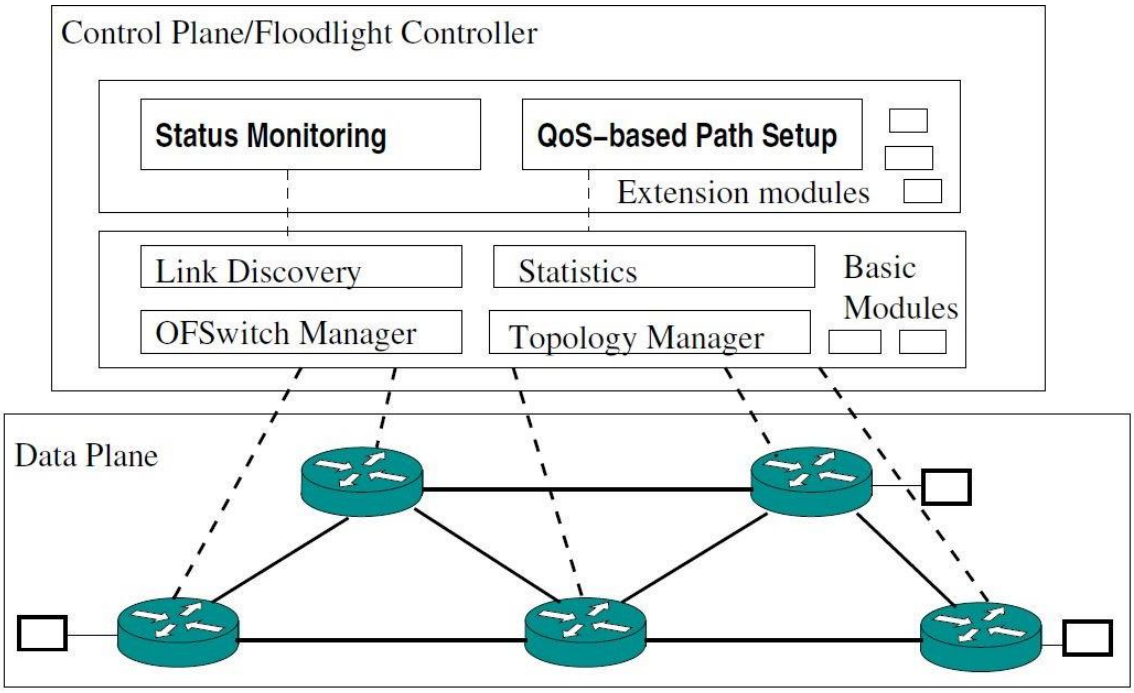


Рисунок 2.1 - Архітектура SDN для забезпечення підтримки QoS

Додаток контролера періодично збирає статистику вимірювань (лічильники переданих байтів). Кожна ланка в топології мережі має вихідний комутатор та/або порт комутатора призначення. Використовувана пропускна здатність каналу зв'язку виходить шляхом ділення кількості переданих байтів з вихідного комутатора-порту цього каналу за інтервал часу 1. Різниця між значеннями двох послідовних повідомлень у відповідь для лічильника комутатора-порту використовується для обчислення споживаної пропускної здатності відповідного мережевого каналу. Специфікація OpenFlow не містить часових позначок для відповідей на вимірювання. Отже, часовий інтервал задається на рівні площини управління як різниця між тимчасовими мітками відправки цих запитів на комутатор. Часовий проміжок слід вибирати ретельно. Він повинен бути досить маленьким, щоб розрахункова пропускна здатність була відносно актуальною. Він також не може бути занадто маленьким; в іншому випадку контролер і комутатори будуть обтяжені накладними витратами на обробку цих повідомлень. Інтервал від 5 до 10 секунд є розумним вибором.

Інформацію про топологію мережі отримуємо від диспетчера топології модуля Floodlight. Для кожної ланки ми віднімаємо обчислену пропускну

здатність від її максимальної ємності, щоб отримати доступну ємність, яка використовується в алгоритмі маршрутизації. Щоб отримати максимальну пропускну здатність зв'язку, контролер може відправити запит на комутатор з питанням про особливості порту (або декількох портів) в комутаторі. Заявлена пропускна здатність порту є однією з особливостей порту.

2.1.2 Налаштування шляху на основі QoS

Існує багато заходів для конкретизації різних аспектів вимог QoS. У цьому розділі розглянуто лише потоки трафіку, що вимагають вимог до пропускної здатності. Ці потоки, які називають критичними потоками, повинні бути розміщені на мережевих шляхах, які мають достатню пропускну здатність, тоді як некритичні потоки обробляються з максимальними зусиллями та маршрутизуються за алгоритмом найкоротшого шляху. Критичний трафік можна відрізнити від трафіку з найкращими зусиллями, зіставивши пакети з потоку з набором полів. OpenFlow визначає список полів відповідності, які можна використовувати для розрізнення ланцюжків. Список обов'язкових полів, які повинні підтримуватися комутаторами OpenFlow, включає порт Ingress, вихідний і цільовий MAC-адреси, тип MAC, IP-адреси джерела і призначення, тип протоколу і порти транспортного рівня (рис. 2.2). Є й інші поля, зазначені в специфікації OpenFlow, але комутатори не зобов'язані підтримувати їх усі. Ланцюжок можна ідентифікувати, встановивши правила зіставлення для будь-якої кількості цих полів. Наприклад, ми можемо визначити критичний трафік як увесь трафік TCP з певної IP-адреси або весь трафік TCP у межах попередньо визначеного діапазону вихідних портів.

Перший пакет кожного потоку інкапсулюється в повідомлення PACKET IN OpenFlow і передається контролеру. Це означає, що контролер обробляє перший пакет, а потім встановлює правила пересилання для обробки решти пакетів того ж потоку. Представлений в роботі модуль прослуховує повідомлення PACKET IN і обробляє пакет, інкапсульований в отримані повідомлення. Щоб визначити тип

потоку, пакет декапсулюється для вилучення IP-адрес або TCP-портів (якщо це TCP-пакет) залежно від того, що було визначено як критичні потоки. Потім вони порівнюються з попередньо визначеними адресами або діапазонами. Якщо це критичний трафік, модуль розрахунку маршруту намагається розмістити потік на маршруті, який задовольняє вимоги до пропускної здатності. В іншому випадку він вважається некритичним трафіком, який маршрутизується як трафік найкращих зусиль і обробляється модулем переадресації Floodlight, який використовує алгоритм Дейкстри для обчислення найкоротшого шляху.

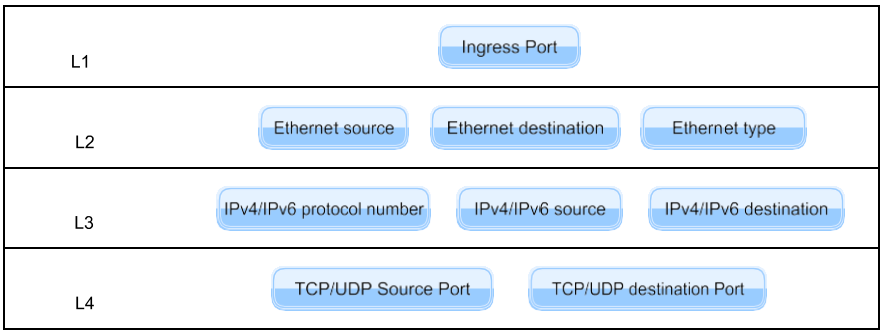


Рисунок 2.2 - Зіставлення полів в OpenFlow

Метод знаходження QoS-шляху для потоку описаний в Алгоритмі 1. Він обходить мережевий граф для пошуку вузла призначення. Попутно він обходить критичним потоком ланки, які мають доступну пропускну здатність менше необхідної. Доступна пропускна здатність розраховується та періодично оновлюється, щоб відобразити поточний стан мережі. При досягненні кінцевого вузла пошук зупиняється і будує шлях для критичного потоку. Слід зазначити, що цей шлях не обов'язково є найширшим (з найбільшою доступною пропускною здатністю), але це найкоротший шлях з достатньою доступною пропускною здатністю. Це рішення прийнято з огляду на зменшення кількості переходів уздовж шляху, що зменшить використання мережевих комутаторів і (у більшості випадків) зменшить затримку.

Після того, як шлях буде знайдено, відповідні правила OpenFlow будуть згенеровані та встановлені контролером Floodlight на комутатори вздовж шляху. Якщо під час пошуку не вдається знайти шлях, який задовольняє необхідну пропускну здатність, він повертає null. У цьому випадку критичний транспортний

потік не може бути розміщений на маршруті, який відповідає його вимогам. Замість цього він буде оброблятися як звичайний трафік і направлятися по найкоротшому шляху за допомогою алгоритму Дейкстри.

В роботі реалізовано алгоритм QoS-маршрутизації у вигляді модуля всередині контролера Floodlight. Floodlight — контролер на основі Java, який був відгалужений від Beacon, одного з перших контролерів OpenFlow. Floodlight пропонує гнучку систему завантаження модулів, яка дозволяє розширювати. Існує два методи написання додатків поверх контролера Floodlight. По-перше, додаток може бути написаний на Java як вбудований модуль всередині Floodlight. Він може взаємодіяти з іншими вбудованими модулями та споживає надані послуги безпосередньо. По-друге, додаток може бути написаний будь-якою мовою і спілкуватися з Floodlight за допомогою REST API. Він може отримувати інформацію та викликати служби, використовуючи відкриті API Floodlight REST. Реалізовано алгоритм маршрутизації QoS як модуль Java всередині Floodlight. Якщо хтось хоче розробити програму, яка є великою і виконує обчислювально дорогі дії, то було б краще написати її як окрему програму, яка може працювати на іншому сервері та взаємодіяти з Floodlight за допомогою його відкритих REST API.

Algorithm 1 FindQoSPath(source, destination, reqCapacity)

```
queue ← empty
visited ← empty
prevLink ← empty
queue.add(source)
visited.add(source)
dstFound ← false
while (queue is not empty) and (dstFound = false) do
    node ← queue.remove()
    for link ← topologyLinks.connectedto(node) do
        neighbor ← link.getDestination()
        if visited.contains(neighbor) then
            continue
        end if
        if getAvailableCapacity(link) < requiredCapacity then
            continue
        end if
        queue.add(neighbor)
        visited.add(neighbor)
        prevLink[neighbor] ← link
        if neighbor = destination then
            dstFound ← true
            break
        end if
    end for
end while
if dstFound = true then
    route ← empty
    node ← destination
    while (node ≠ source) do
        route.addFirst(prevLink[node])
        node ← prevLink[node].getSource()
    end while
    return route
else
    return null
end if
```

2.2 Експерименти

2.2.1 Налаштування експерименту «Перший експеримент»

Для тестування алгоритму маршрутизації QoS було використно Mininet з програмним забезпеченням Open vSwitch. Mininet — це інструмент емуляції, який забезпечує віртуалізоване середовище для створення прототипів та оцінки SDN-додатків. Він використовує легкі методи віртуалізації на рівні операційної системи для емуляції хостів, посилань, комутаторів і контролерів. Було написано

скрипти на Python з використанням Mininet Python API для створення топології мережі. Mininet можна налаштувати для роботи з програмними комутаторами. Було обрано Open vSwitch через його гнучкість та хорошу підтримку специфікації комутатора OpenFlow.

Для тестування представленого модуля QoS-маршрутизації було створено топологію мережі з 8 комутаторами (Open vSwitch) і 22 хостами в Mininet. Топологія показана на рисунку 3.3. Пропускна здатність зв'язків між комутаторами встановлена на 25 Мбіт/с, а пропускна здатність зв'язків між комутаторами та хостами – на 10 Мбіт/с. Mininet використовує команду контролю трафіку в Linux tc для визначення пропускної здатності. Після цього було згенеровано 14 потоків за допомогою iperf3. П'ять із цих потоків були критичними. У таблиці 3.1 наведено перелік потоків у цьому експерименті. Використовуючи скрипт Python, була запущена топологія Mininet, а потім згенерувані потоки в порядку, показаному в таблиці. Контролер Floodlight працював на іншому комп'ютері, підключеному до хоста Mininet.

Результати

По-перше, було проведено цей експеримент з вимкненим модулем маршрутизації QoS у Floodlight. Потім було проведено той самий експеримент з увімкненим модулем маршрутизації QoS. Оскільки канал зв'язку між комутаторами та хостами має пропускну здатність 10 Мбіт/с, максимальна швидкість, яку може надіслати критичний потік, становить 10 Мбіт/с. Показана виміряна пропускна здатність для кожного потоку в обох випадках на рисунку 2.3.

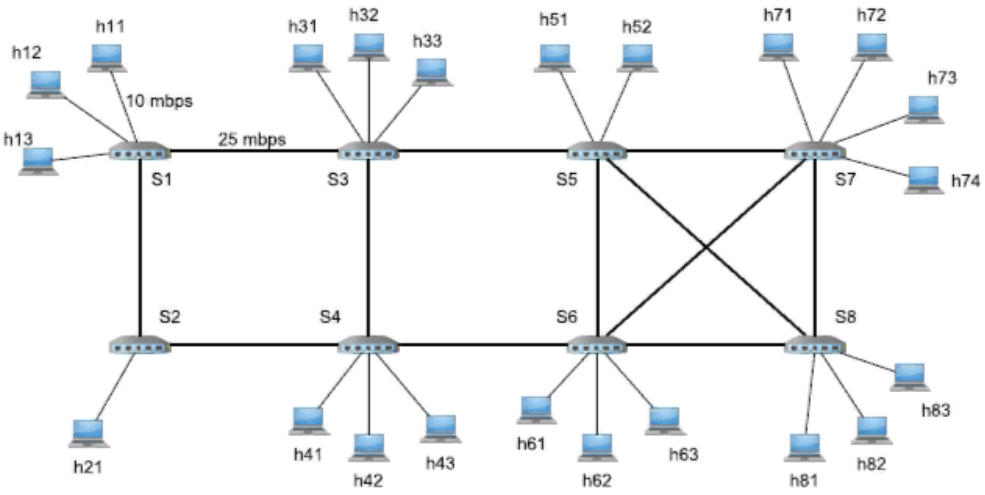


Рисунок 2.3 - Топологія першого експерименту в Mininet

Таблиця 2.1 - Транспортні потоки

Тексти	Target	Призначення	Час початку	Має вирішальне значення	Розмір
1	h31	h42	2	Ні	150 МБ
2	h41	h32	3	Ні	140 МБ
3	h32	h41	4	Ні	145 МБ
4	h42	h31	7	Ні	135 МБ
5	h11	h71	22	Так	135 МБ
6	h71	h61	23	Ні	140 МБ
7	h43	h33	25	Ні	120 МБ
8	h33	h81	41	Так	130 МБ
9	h74	h63	42	Ні	115 МБ
10	h12	h51	57	Так	125 МБ
11	h72	h62	59	Ні	125 МБ
12	h51	h83	76	Ні	100 МБ
13	h13	h82	91	Так	120 МБ
14	h73	H21	106	Так	110 МБ

Результати експерименту показують, що всі п'ять критичних потоків досягали набагато меншої пропускної здатності, коли контролер Floodlight працював без модуля маршрутизації QoS. Середні показники для критичних потоків становили 8046 кбіт/с (потік No5), 7248 кбіт/с (потік No 8), 7272 кбіт/с (потік No 10), 6838 кбіт/с (потік No 13) і 7024 кбіт/с (потік No 14). Вимірювання покращилися у випадку з увімкненим модулем маршрутизації QoS. Пропускна здатність для критичних потоків становила 9408 кбіт/с (потік No 5), 9374 кбіт/с

(потік No 8), 9446 кбіт/с (потік No 10), 9565 кбіт/с (потік No 13) та 9572 кбіт/с (потік No 14).

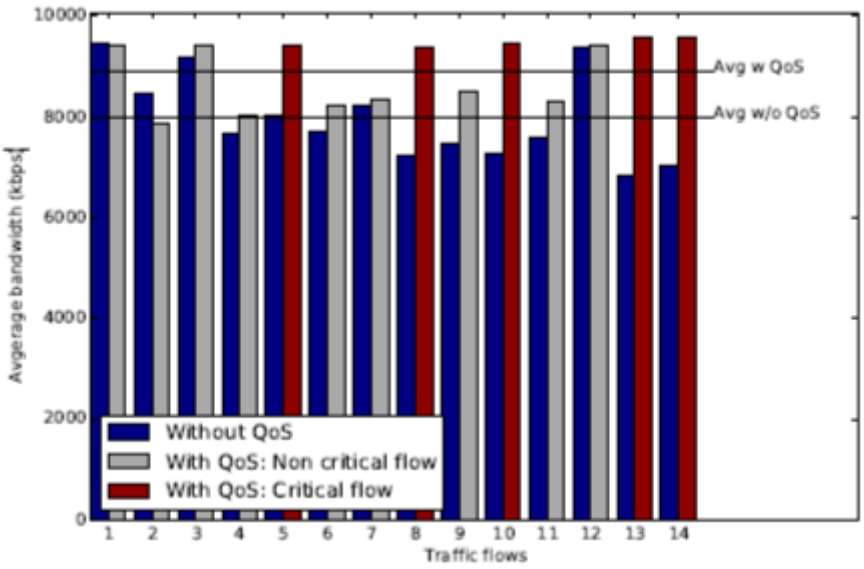


Рисунок 2.4 - Пропускна здатність з п'ятьма критичними потоками

На рисунках 2.5 і 2.6 показана виміряна пропускна здатність для кожного критичного потоку, як показує iperf3 з інтервалом в 10 секунд. Вісь X позначає час життя кожного потоку (його тривалість), а не час експерименту. У першому досліді потік No 10 (від h12 до h51) був прокладений через S1, S3 і S5. Link (S3, S 5) вже мав 2 потоки (No5 і No8), залишаючи його з пропускною здатністю близько 5 Мбіт/с. Використання цього маршруту призвело до перевантаження зв'язку. При включенні модуля маршрутизації QoS пропускна здатність всіх цих критичних потоків була поліпшена, як показано на рисунку 3.6. Модуль маршрутизації QoS розмістив потік No10 за іншим маршрутом (S1, S 2, S4, S6, S5). Таке розміщення дозволило всім трьом потокам мати кращу пропускну здатність. З рисунка ми бачимо, що різниця невелика, а пропускна здатність цих критичних потоків постійно тримається в районі 9,5 Мбіт/с.

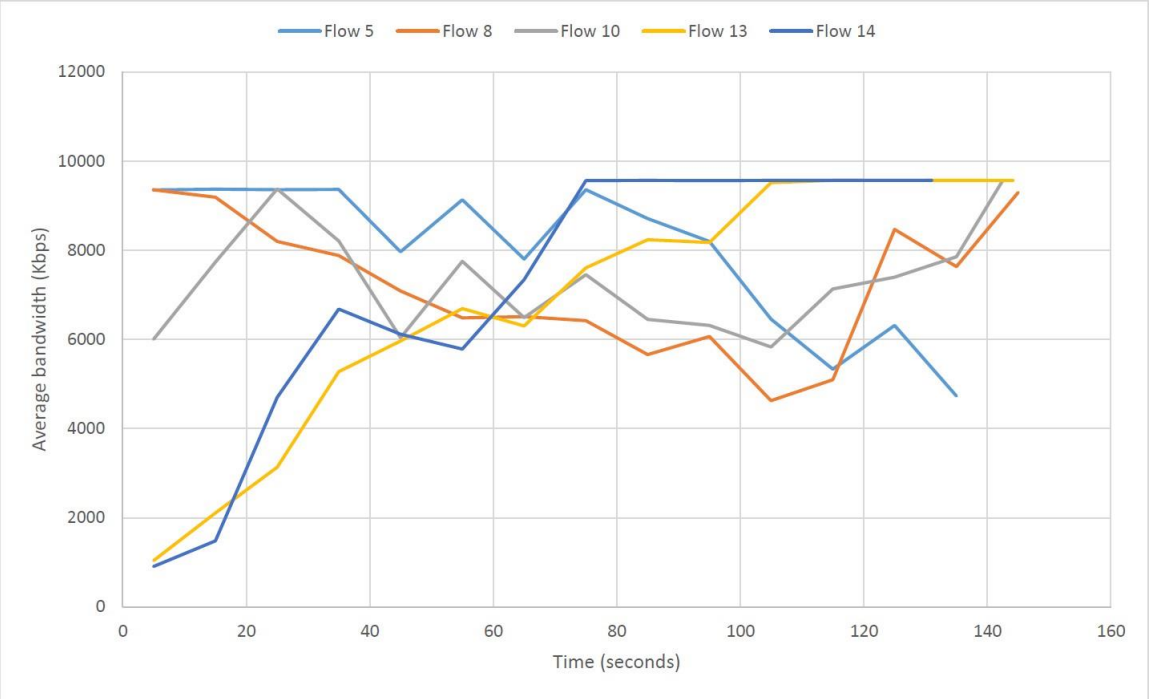


Рисунок 2.5 - Пропускна здатність для критичних потоків без модуля маршрутизації QoS

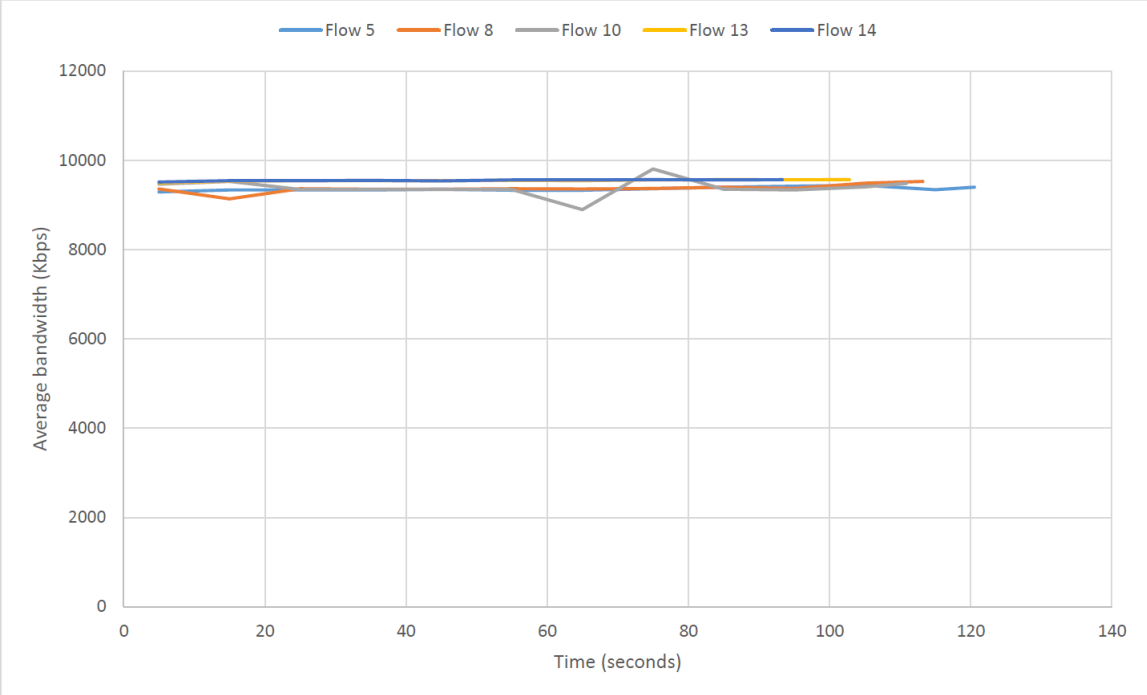


Рисунок 2.6 - Пропускна здатність для критичних потоків з модулем маршрутизації QoS

Щоб показати різницю поводження з потоком залежно від його типу, було проведено той самий експеримент після зміни потоку No 11 (з h72 на h62) на

критичний потік. У двох попередніх експериментах цей потік був розміщений на траєкторії (S7, S6) і його пропускна здатність становила близько 7600 кбіт/с. Після цієї зміни той самий потік був поміщений на шлях (S7, S8, S6). Виміряна пропускна здатність для цього потоку становила 9571 кбіт/с. Ця зміна також покращила пропускну здатність інших потоків, як показано на рисунку 2.7.

Переваги уникнення перевантажених посилань включають краще використання пропускної здатності всієї мережі. Це проілюстровано на рисунку 2.8. Результати були отримані в результаті трьох запусків експерименту, тобто без включеного модуля QoS, з включеним модулем QoS з 5 критичними потоками, а з модулем QoS з 6 критичними потоками. Він показує, що при включенні модуля QoS мережа досягла більш високого рівня використання.

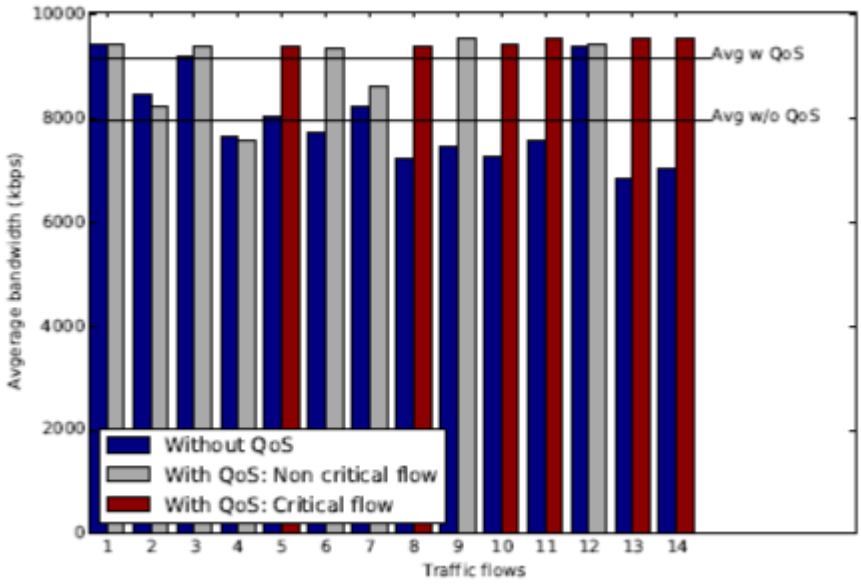


Рисунок 2.7 - Пропускна здатність з шістьма критичними потоками

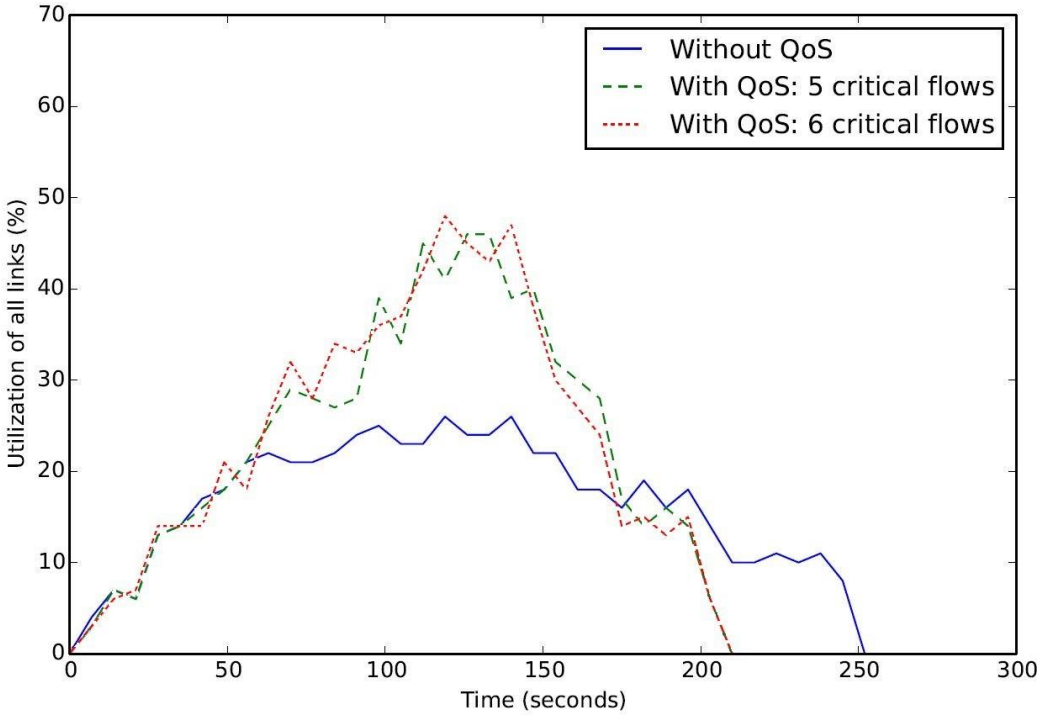


Рисунок 2.8 - Використання всієї мережі

2.4.2 Налаштування експерименту з другим експериментом

У попередньому експерименті модуль маршрутизації QoS зміг знайти можливий шлях для всіх критичних потоків. Це означає, що всі критичні потоки були прийняті модулем. Однак це не завжди так. Іноді мережа стає перевантаженою таким чином, що не дозволяє знайти шлях з достатньою пропускною здатністю. У другому експерименті заплановано виміряти загальну продуктивність мережі по відношенню до кількості прийнятих потоків. Було створено іншу топологію з 5 комутаторів і 20 хостів. Кожен комутатор підключений до чотирьох хостів. Пропускна здатність зв'язків між комутаторами встановлена на 15 Мбіт/с, а пропускна здатність зв'язків між комутаторами та хостами – на 10 Мбіт/с. Топологія показана на рисунку 2.9. Було згенеровано випадковий список з 20 потоків трафіку таким чином, що:

- Кожен хост буде відправляти і отримувати тільки один ланцюжок.
- Перемикачі джерела та призначення різні для кожного потоку.
- Час і тривалість початку є випадковими (в межах діапазону).

- Усі потоки критичні.

Результати

Цей експеримент повторювався 30 разів, щоразу з різним списком потоків. Для кожного разу виміряно сумарну середню пропускну здатність усіх потоків і середню пропускну здатність прийнятих потоків. На рисунку 2.10 представлені отримані результати. Можна чітко побачити, що зі збільшенням числа прийнятих потоків середня пропускна здатність всіх потоків зростає. При цьому середня пропускна здатність прийнятих потоків більше, ніж у всіх потоків.

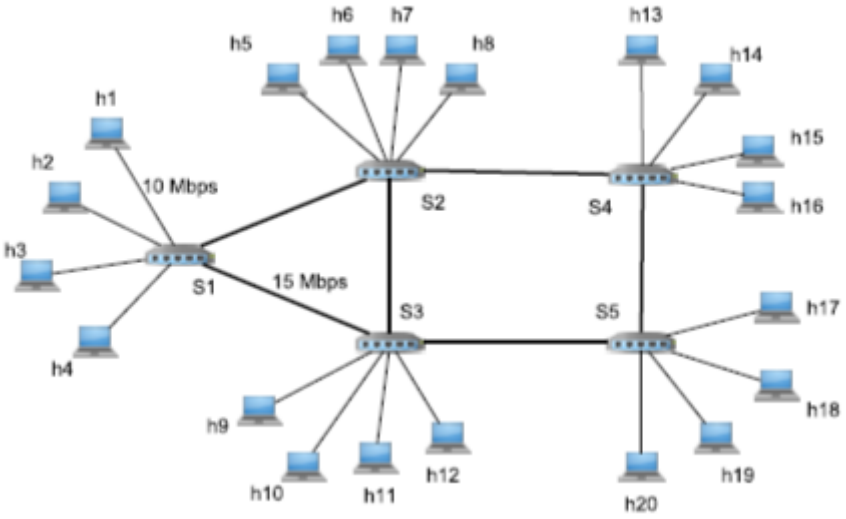


Рисунок 2.9 - Топологія другого експерименту в Mininet

У цьому розділі ми запропонували структуру на основі SDN для забезпечення кращого обслуговування критичних потоків. Ця структура зосереджена на задоволенні потреб критичних потоків у пропускній здатності. Вона скористалася парадигмою SDN, розробивши модулі в контролері для моніторингу мережі та налаштування шляхів QoS для потоків, що вимагають

пропускної здатності. Результати оцінки продемонстрували, що нові модулі можуть підвищити продуктивність для тих додатків, які використовують критичні потоки.

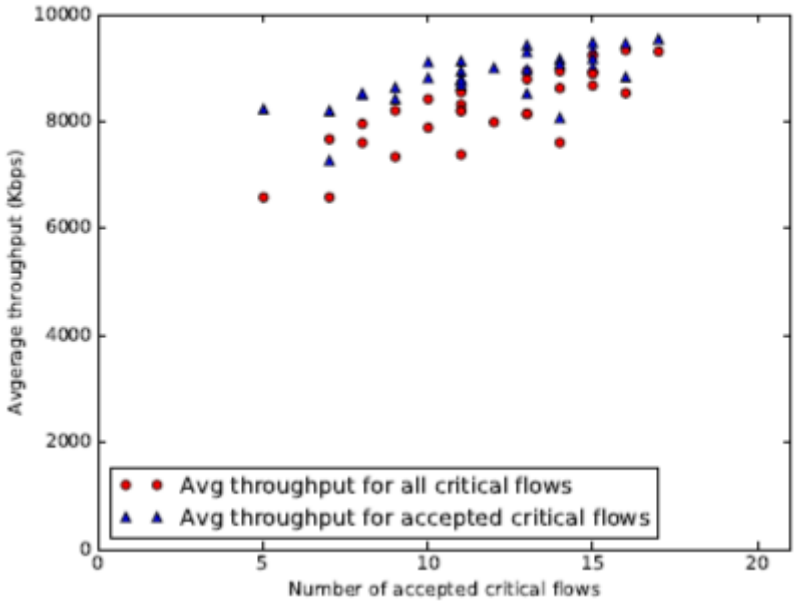


Рисунок 2.10 - Середня пропускна здатність до числа прийнятих потоків

3 ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ОБСЛУГОВУВАННЯ ПОТОКІВ ТРАФІКУ, ЧУТЛИВИХ ДО ЗАТРИМОК

3.1 Підвищення пропускної здатності великих потоків за допомогою багатопроменевого TCP

Сучасні комп'ютерні мережі вміщують різнорідні додатки, які мають різний рівень вимог до якості обслуговування. Деякі потоки мережевого трафіку мають жорсткі терміни, коли незначне збільшення затримки може вплинути на загальну продуктивність додатка. Такі потоки трафіку потрібно перенаправляти з вищим пріоритетом, ніж інші потоки трафіку. Планування та проектування мережевих систем, здатних ефективно забезпечувати гарантію затримки, залишається складним завданням. У цьому розділі запропоновано систему, призначену для забезпечення QoS для чутливих до затримок потоків трафіку з використанням підходу SDN. Ця система надає зручні механізми для визначення, управління та переадресації різних класів потоків трафіку з різним рівнем пріоритетів.

Останніми роками все частіше зростає інтерес до хмарних обчислень та віртуалізованих середовищ. Це зумовлено необхідністю ефективного використання обчислювальних ресурсів і зниження витрат. У такій інфраструктурі зазвичай розміщуються різного роду додатки для різних клієнтів. Кожен додаток/клієнт має власний набір вимог, які зазвичай визначаються в їхніх угодах про рівень обслуговування (SLA). Вимоги до якості обслуговування (QoS) включають наскрізну пропускну здатність і затримку серед інших атрибутів. Було докладено кілька зусиль для вирішення проблем забезпечення QoS різних типів мережевих додатків у різних середовищах за допомогою різних протоколів і методів. Забезпечення та моніторинг QoS у хмарних обчисленнях є ще складнішим через складність середовища спільної інфраструктури. Зазвичай багато постачальників послуг вдаються до надмірного виділення мережевих ресурсів для задоволення вимог QoS. Однак надмірне резервування не є оптимальним рішенням. Ефективне використання мережевих ресурсів вимагає

максимізації отриманої продуктивності при одночасному зниженні експлуатаційних і капітальних витрат.

Наскрізна затримка є важливим показником QoS для певних типів мережевих додатків. Наприклад, телеконференції, передача голосу через IP (VoIP) та онлайн-ігри в Інтернеті зазвичай чутливі до високої затримки. Вони вимагають, щоб дані, що передаються, мали наскрізну затримку нижче певного порогу. Якщо затримка перевищує цей поріг, це погіршує продуктивність і впливає на якість обслуговування (QoE) для кінцевого користувача (наприклад, переривання VoIP-дзвінків). Іншим прикладом, який може мати вищий пріоритет, є контрольний трафік, який повинен передаватися з найменшою затримкою в черзі. Різні мережеві додатки мають різні рівні пріоритетів, які повинні враховуватися мережевими пристроями при пересиланні пакетів. Крім того, вимоги до максимальної затримки різняться для різних програм. Такий мережевий трафік повинен бути ідентифікований і перенаправлений відповідно до відповідної політики. Однак інший мережевий трафік (наприклад, трафік FTP) може бути більш стійким до більшої затримки та періодичного зниження пропускної здатності. Такий трафік може бути перенаправлений з максимальною віддачею без помітного зниження продуктивності.

Вимоги QoS для трафіку, чутливого до затримки, відрізняються залежно від типу програми. ITU-T рекомендує при загальному плануванні мережі не перевищувати максимум 400 мс для односторонньої затримки. Однак вони зазначають, що на багато інтерактивних додатків (наприклад, голосові дзвінки, відеоконференції, інтерактивні програми для роботи з даними) впливає набагато менша затримка. Досвід роботи більшості додатків, як правило, вважається прийнятним, якщо затримка не перевищує 150 мс. Зі збільшенням затримки трафіку вплив на роботу додатків стає помітним. Коли затримка перевищує 400 мс, більшість програм зіткнуться з незадовільною продуктивністю.

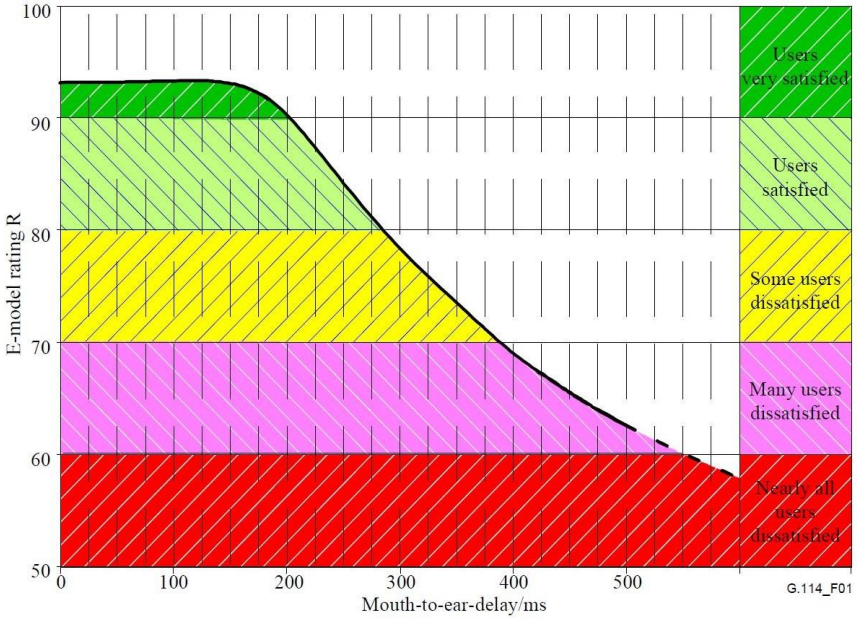


Рисунок 3.1 - Вплив загальної затримки на задоволеність користувачів

На наскрізну затримку пакетів даних, що передаються, впливає кілька факторів. До них відносяться:

- Затримка обробки. Час, необхідний мережевим пристроям для обробки пакета. В ідеальних ситуаціях затримка обробки незначна, якщо пакети обробляються з лінійною швидкістю. Однак обробка може включати перевірку та зміну деяких полів пакета (наприклад, позначення поля DSCP). Крім того, це може включати ще дорожчі завдання, такі як глибока перевірка пакетів (DPI). Такі обчислювально дорогі завдання призводять до значної затримки обробки.
- Затримка передачі. Час, необхідний маршрутизатору/комутатору, щоб перемістити всі біти пакетів у мережеве з'єднання. На затримку передачі впливає пропускна здатність каналу і розмір пакета.
- Затримка розмноження. Час, витрачений на один біт, щоб пройти через посилання від джерела до пункту призначення. На затримку поширення впливає тип і відстань з'єднання.
- Затримка в черзі. Сумарний час очікування перебування пакета в чергах маршрутизаторів і комутаторів. На затримку черги впливає розмір черги під час прибуття пакета. На це також впливає диференціальна обробка пакетів

даних (тобто трафік з вищим пріоритетом проводитиме менше часу в черзі порівняно з трафіком з нижчим пріоритетом).

Щоб зменшити затримку потоків трафіку, постачальники послуг можуть використовувати такі методи, як резервування ресурсів та/або маршрутизація з урахуванням QoS. Надання гарантованої мінімальної швидкості (пропускної здатності) для потоку гарантує, що всі пакети, які надходять з однаковою швидкістю (або нижче), будуть перенаправлені без затримок. Однак, якщо швидкість прибуття перевищує гарантований мінімальний тариф, то пакетам, можливо, доведеться чекати в черзі протягом певного періоду часу. Політики також можуть включати максимальну швидкість, щоб гарантувати, що потоки не перевищують певну пропускну здатність.

Зазначимо, що більшість схем трафіку мають розривний характер. Наприклад, сеанс HTTP починається із завантаження HTML-сторінки та пов'язаних з нею мультимедійних об'єктів (наприклад, таблиць стилів, зображень тощо), а потім стає неактивним протягом певного періоду часу, доки користувач не надішле запит на іншу HTML-сторінку. Неузгодженість рівнів трафіку дуже поширена навіть серед усталених довготривалих TCP-з'єднань. Причина полягає в тому, що контроль перевантаження TCP регулює швидкість надсилання на основі сигналів перевантаження, щоб адаптуватися до поточних умов мережі. Він дотримується механізму адитивного збільшення мультиплікативного зменшення (AIMD), який спрямований на збереження зручності та справедливості TCP щодо інших транспортних потоків. З цієї точки зору можна побачити, що якби таким потокам трафіку була присвоєна гарантована пропускна здатність (квота), то дуже ймовірно, що в даний момент часу залишиться невикористана резервна пропускна здатність з цієї квоти.

Схеми трафіку, чутливі до затримок, також можуть бути розривними. В роботі спрямовано увагу на потоки трафіку, які не становлять значної частини від загального трафіку в мережі. Такі потоки трафіку не передаються з стабільно високою швидкістю постійно. Тому їх гарантована мінімальна пропускна здатність (якщо вона зарезервована) не використовується під час простою. Однак,

коли такі потоки трафіку передаються, вони не терплять високої затримки. Їх потрібно пересилати з вищим пріоритетом на шляхи, які гарантують максимальну затримку, меншу або рівну вказаній їм затримці. З цього моменту можна усвідомити важливість управління пріоритетами, гарантованою мінімальною ставкою, максимальною ставкою та використанням невикористаної гарантованої ставки для різних класів потоків мережевого трафіку.

У цій главі запропоновано методику, яка призначена для надання та моніторингу якості обслуговування транспортних потоків, чутливих до затримок, за допомогою SDN-підходу. У системі передбачені зручні механізми визначення і управління класами руху в площині управління. Кожен клас має дві властивості:

- Визначальні характеристики: які містять унікальне ім'я класу та перелік усіх потоків, що належать до цього класу. Кожен потік представлений набором OpenFlow
- Атрибути QoS: які містять пріоритет класу та, необов'язково, мінімальну швидкість, максимальну швидкість та необхідну затримку.

3.1.2 Вимірювання затримки в SDN

Вимірювання латентності в SDN обговорювалося в літературі. Наприклад, OpenNetMon вставляє пакети зондування в вихідний комутатор кожного каналу зв'язку і встановлює правила, які змушують їх обходити канал, що підлягає вимірюванню. Комутатор призначення налаштований на надсилання пакетів зондування назад на площину керування. Процес також включає в себе оцінку затримки між контролером і кожним комутатором шляхом введення пакетів, які негайно повертаються назад контролеру, який використовує час проходження сигналу туди і назад (RTT) для визначення затримки. Додаток моніторингу в контролері обчислює розрахункову затримку кожного контрольованого каналу. Затримка каналу зв'язку обчислюється шляхом віднімання розрахункової затримки між контролером і комутатором від різниці між часом відправлення та прибуття.

$$Затримка = Time_{Arrival} - Time_{Sent} - 1/2\ (RTT_{source} + RTT_{destination})$$

Floodlight, популярний контролер SDN, використовує аналогічний механізм для обчислення розрахункового значення затримки для кожної ланки топології. Модуль виявлення зв'язків інкапсулює часові позначки з пакетами LLDP, використовуючи необов'язкову структуру тип-довжина-значення (TLV). Він періодично відправляє LLDP-пакети через всі доступні порти. Затримка зв'язку згладжується шляхом обчислення середнього значення певної кількості вимірювань затримки. Обчислена затримка зберігається як атрибут для кожної ланки у виявленій топології.

3.1.3 Дисципліни масового обслуговування

Дисципліна черги (також відома як qdisc) керує тим, як обробляються пакети під час очікування в черзі. Тип дисципліни масового обслуговування безпосередньо впливає на затримку, оскільки визначає, як довго пакету доведеться чекати, перш ніж його обслужать (передадуть). Обговорення даної роботи обмежене дисциплінами черги, які доступні в Linux. Найпростішим типом дисципліни масового обслуговування є qdisc first-in first-out (FIFO). Це гарантує, що перший пакет, який надійде, буде переданий перед наступними пакетами, незалежно від будь-яких інших міркувань. Незважаючи на те, що FIFO qdisc забезпечує просту і швидку реалізацію, він не забезпечує диференціальної обробки пакетів даних. Тому він не підходить для забезпечення QoS. Крім того, використання FIFO qdisc не забезпечує чесного обслуговування кількох потоків трафіку. Це пов'язано з тим, що деякі великі потоки можуть швидко заповнити буфер черги, що призведе до скидання пакетів інших потоків. Ця несправедлива схема обслуговування викликала потребу в різних дисциплінах черги, які забезпечують справедливе обслуговування кількох потоків.

Стохастична чесна черга (SFQ) qdisc є реалізацією чесної черги. Він спрямований на однакове обслуговування всіх транспортних потоків, створюючи кілька черг, а потім, використовуючи хешування, зіставляє пакети з чергами.

Рівень справедливості досягається шляхом передачі даних з кожної черги за круговою системою. І FIFO, і SFQ є безкласовими дисциплінами масового обслуговування. Тобто не можна надавати різні послуги для різних класів транспортних потоків.

Ієрархічний сегмент токенів (HTB) — це класовий qdisc, який був запропонований для заміни попередньої класової дисципліни масового обслуговування, яка називається класовою чергою (CBQ) qdisc у системах Linux. HTB дозволяє організувати різні класи потоків трафіку в ієрархічну структуру. У той час як HTB в основному використовується для гарантії пропускної здатності, наша система на основі SDN використовує його для забезпечення QoS, чутливого до затримок. Черги можна налаштувати за допомогою команди tc Linux. Програмний перемикач Open vSwitch забезпечує підтримку HTB qdisc, хоча і не охоплює всіх його можливостей. Відкритий vSwitch дозволяє налаштовувати черги HTB або за інтерфейсом командного рядка (CLI), або за протоколом OVSDB.

Ієрархічна крива добросовісного обслуговування (HFSC) є ще одним класовим qdisc, доступним у Linux. HFSC також надає можливість створення ієрархічної структури різних класів трафіку. HFSC має на меті забезпечити гарантоване обслуговування як для параметрів пропускної здатності, так і для параметрів затримки, підтримуючи криву обслуговування для кожного параметра. Open vSwitch дозволяє створювати черги на основі HFSC qdisc. Однак параметр затримки не підтримується Open vSwitch.

3.2 Архітектура на основі SDN для підтримки потоків, чутливих до затримок

У цьому розділі представлено запропонований нами підхід до QoS для трафіку, чутливого до затримок. Запропонована система орієнтована на пристосування різних класів транспортних потоків. Він використовує архітектуру SDN і дисципліну черги HTB. У даному підході потоки, чутливі до затримки,

згруповані в класи. Кожен клас містить інформацію, яка визначає, як пересилаються його потоки.

Система на базі SDN складається з наступних компонентів:

- Модуль контролю допуску: відповідає за обробку нових запитів QoS. Залежно від характеру вимог і наявних ресурсів, нові запити або приймаються, або відхиляються.
- Модуль забезпечення QoS: відповідає за налаштування черг та встановлення правил OpenFlow для надання необхідного сервісу для прийнятих QoS-запитів.
- Модуль моніторингу QoS: отримує та представляє статистику про активні класи QoS.
- Модуль агента OVSDB: перетворює команди налаштування черги на повідомлення протоколу OVSDB.

3.2.1 Вхідний контроль

Контроль допуску відповідає за обробку нових запитів QoS. Він доступний через REST API для авторизованих додатків. Крім того, системні адміністратори можуть отримати доступ до веб-інтерфейсу для додавання нових запитів. Очікується, що в новому запиті буде вказано пріоритет класу трафіку.

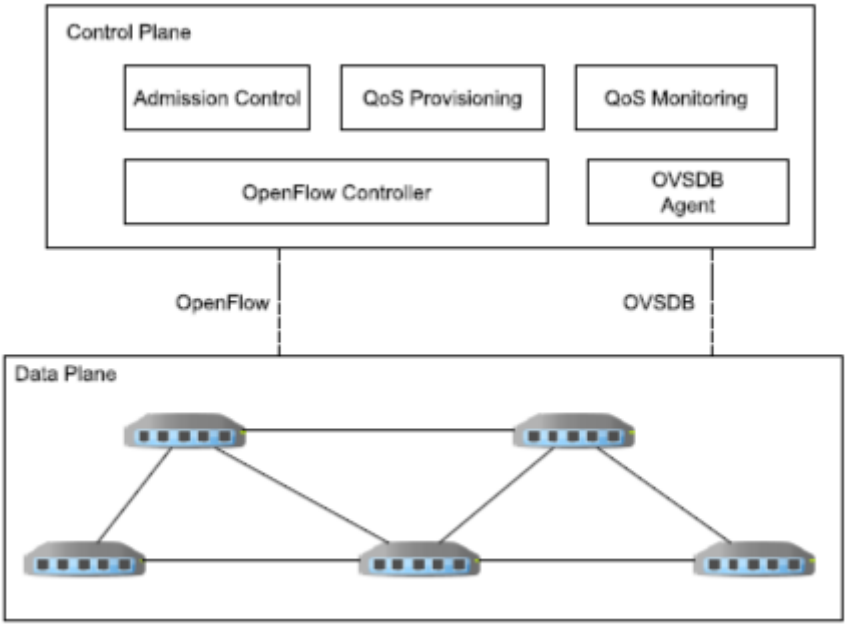


Рисунок 3.2 - Підготовка архітектури QoS до чутливих до затримки потоків

Пріоритет виражається цілим значенням від 0 (найвищий пріоритет) до 7 (найкращі зусилля). У запиті за бажанням можна вказати мінімальну ставку та максимальну ставку, яка буде зарезервована в кожному вузлі на шляху. Ці два атрибути не є обов'язковими, і якщо вони відсутні, їх значення будуть присвоєні з типових конфігурацій:

мінімальна ставка = квота ∗ пропускна здатність каналу

максимальна ставка = пропускна здатність каналу

Якщо новий запит містить конкретне значення необхідної затримки (у мс), то контроль допуску отримає виміряну затримку для найкоротшого шляху між вихідним і цільовим хостами та порівняє її з необхідною затримкою. Він відхиляє запит, якщо він не може бути виконаний (тобто, якщо виміряна затримка найкоротшого шляху більша за запитувану затримку). Однак не виключено, що запит не містить певну затримку. Скоріше, він просто вказує пріоритет класу. У цьому випадку запит приймається і додається до списку активних класів, якщо мінімальна ставка може бути зарезервована на кожному вузлі по шляху. В іншому випадку запит відхиляється.

Нижче наведено приклад нового QoS-запиту у форматі JSON:

```
{ "name": "VoIP",
  "min_rate": "2000000",
  "max_rate": "10000000",
  "latency": "150",
  "priority": "2",
  "flows": [{"ipv4_src":"10.0.0.2",
             "ipv4_dst":"10.0.0.4",
             "tcp_src":"5001"},
            {"ipv4_src":"10.0.0.4",
             "ipv4_dst":"10.0.0.2",
             "tcp_dst":"5001"}]
}
```

3.2.2 Налаштування черг

Кожен новий прийнятий запит QoS вимагає налаштування відповідних черг на площині даних. Ці конфігурації виконуються агентом OVSDB, який перетворює команди налаштування черги в повідомлення протоколу OVSDB. Дисципліну черги HTB потрібно заздалегідь налаштувати за допомогою кореневого класу. Цей етап виконується, як тільки комутатор підключається до контролера. Кожен клас налаштований з пріоритетом, мінімальною ставкою та максимальною ставкою. Модуль OVSDB надсилає метод транзакції RPC кожному комутатору у вибраному шляху. Метод транзакції містить ряд операцій для налаштування нової конфігурації черги в базі даних комутаторів.

3.2.3 Встановлення правил OpenFlow

Після налаштування черг за допомогою агента OVSDB, модуль забезпечення QoS встановить правила OpenFlow на кожному комутаторі вздовж шляху (або шляхів) для всіх потоків трафіку, які належать до цього класу.

Встановлені правила OpenFlow зіставляють кожен потік трафіку і направляють його пакети на ID черги, налаштований для класу.

3.2.4 Моніторинг та звітність

Моніторинг – важливе завдання, яке доповнює встановлення та налаштування QoS. Модуль періодично отримує статистику з комутаторів площини даних. Він представляє їх системним адміністраторам і авторизованим додаткам за допомогою REST API. Кожен звіт статистики черги включає:

- Кількість переданих байтів.
- Кількість переданих пакетів.
- Кількість скинутих пакетів.

3.3 Забезпечення QoS для чутливих до затримки потоків

HTB qdisc дозволяє впорядковувати класи трафіку в багаторівневе ієрархічне дерево. У системі представленій в роботі було використано два шари, де кореневий вузол (у першому шарі) представляє батьківський клас для всіх видів трафіку. Кореневий вузол налаштовується, як тільки комутатор підключається до контролера. Максимальна ставка і мінімальна ставка для кореневого класу дорівнюють швидкості з'єднання. Для кожного прийнятого QoS-запиту контроль допуску створює листовий вузол, пов'язаний з кореневим вузлом з властивостями, зазначеними в запиті. Двома основними властивостями є мінімальна ставка і пріоритет. Як правило, максимальна ставка для класу дорівнює root (швидкості посилання), якщо в запиті явно не вказано інше.

Мінімальний тариф - це гарантована пропускна здатність, надана класу. Однак клас може перевищити цю межу до максимальної ставки (або ceil). Це відбувається, коли листовий вузол може запозичувати токени (тобто споживати невикористану пропускну здатність) у свого батьківського класу. У будь-який момент часу листовий вузол може мати один з трьох станів:

- Пропускна здатність досягла максимальної швидкості. Він не може надсилати пакети або позичати токени у свого батька. Пакети класу залишатимуться в черзі до тих пір, поки токени не стануть доступними.
- Пропускна здатність досягла мінімальних показників. Він може спробувати запозичити токени у свого батька. Якщо токени доступні, то пакети будуть передані в кількості, що збігається з доступними токенами. В іншому випадку пакети залишаться в черзі до тих пір, поки токени не стануть доступними.
- Пропускна здатність менша за мінімальну швидкість. Він може відправляти пакети до тих пір, поки є достатня кількість токенів.

Жетони додаються до сегмента класу за вказаною мінімальною ставкою, доки не буде досягнуто розміру сегмента. Пакети, що належать до класу, надсилаються лише в тому випадку, якщо відповідний сегмент має доступні токени. В іншому випадку пакетам доведеться чекати в черзі класів, поки не з'являться нові токени. Типово, черга FIFO прикріплена до кожного класу листа. Пакети, що прибувають, скидаються, якщо черга класів заповнена. Збільшення розміру буфера черги зменшує коефіцієнт втрати пакетів. Однак великий розмір буфера може призвести до збільшення затримки для трафіку з нижчим пріоритетом.

За кожен відправлений пакет з сегмента вилучається токен (для простоти ми припускаємо, що один токен представляє один пакет). Якщо сегмент класу порожній, а черга класів містить пакети, він намагається запозичити токени у батьківського класу. Коли кілька класів намагаються запозичити у одного з батьків, першими подаються класи з найвищим пріоритетом. Якщо два або більше класів мають однаковий пріоритет, резервна пропускна здатність дорівнює розділені між ними. Пріоритет представлений як число від 0 до 7, де 0 є найвищим пріоритетом. Класу листків дозволено позичати та споживати токени, доки не буде досягнуто максимальної ставки.

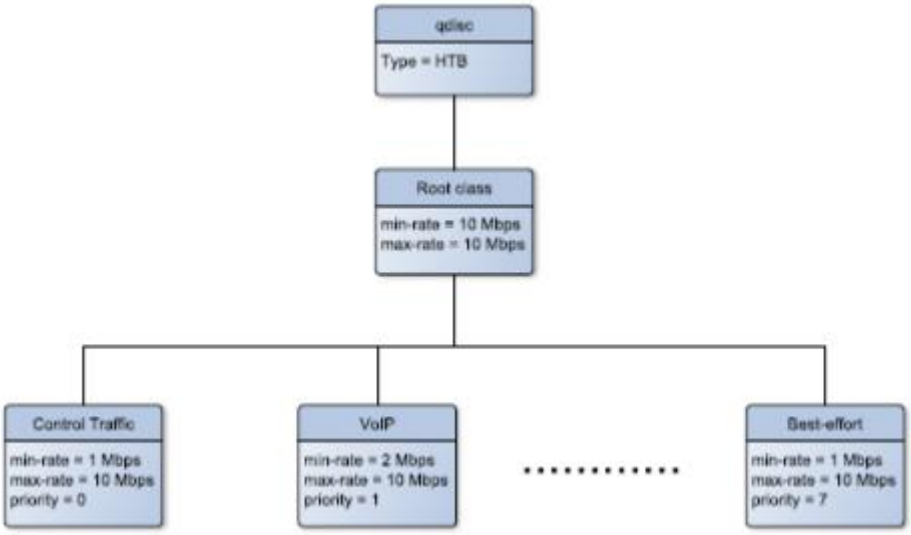


Рисунок 3.3 - Приклад різних класів трафіку

Хоча протокол OpenFlow можна використовувати для встановлення правил, які використовують існуючі черги, він не може бути використаний для створення нових черг. Конфігурації комутаторів, які включають черги серед інших атрибутів, визначаються поза областю видимості OpenFlow. Використання командного рядка-інтерфейсу (CLI) комутатора не є зручним методом, особливо для динамічного налаштування великої кількості комутаторів. Для конфігурації комутатора запропоновано два протоколи. OpenFlow Management and Configuration Protocol (OF-CONFIG) і Open vSwitch Database Management Protocol (OVSDB). Було вирішено реалізувати протокол OVSDB, оскільки він повністю підтримується OpenvSwitch.

Сам OVSDB заснований на протоколі JSON-RPC версії 1.0. Конфігурації комутаторів зберігаються в базі даних з визначеною схемою. Маніпулювання цією базою даних, використовуючи OVSDB, дозволяє програмам динамічно керувати комутатором і змінити його конфігурацію.

Модуль OVSDB встановлює канал зв'язку з кожним комутатором, підключеним до контролера SDN. Спочатку він отримує схему бази даних і конфігурації комутаторів. Потім він переконується, що кожен порт комутатора налаштований таким чином, щоб мати кореневий клас HTB qdisc. Після цього він

робить доступним для інших модулів отримання та зміну конфігурацій, включаючи створення нових черг. Модуль контролю прийому використовує цю функціональність спочатку для того, щоб перевірити, чи є вільні можливості для нового запиту, а потім для створення нових класів для прийнятих запитів.

3.4 Оцінка ефективності

Щоб оцінити представлені в даній роботі систему на основі SDN, було проведено експерименти з використанням тестового середовища Global Environment for Network Innovations (GENI). GENI надає інфраструктуру, яка дозволяє дослідникам проводити експерименти в галузі мереж і розподілених систем. У експериментах топологія лінійна, що складається з трьох вузлів перемикання. Було використано програмний комутатор OpenvSwitch. Пропускна здатність становить 10 Мбіт/с для всіх каналів. Топологія показана на рисунку 4.4. Щоб імітувати високопріоритетний контрольний трафік, розроблено інструмент, який обмінюється повідомленнями за протоколом TCP і вимірює продуктивність мережі. Був проведений один і той же експеримент двічі протягом 30 секунд, один з даною системою QoS, а інший без системи QoS. Високопріоритетний контрольний трафік був від VM1 до VM3. У той же час між VM2 і VM4 генерувався низькопріоритетний фоновий трафік. Було використано iperf для генерації UDP-трафіку. UDP підходить для імітації фонового трафіку, оскільки він не має контролю перевантажень, який знижує швидкість надсилання при перевантаженні мережі.

Результати обох експериментів показують величезну різницю у виміряній затримці контрольного трафіку. Без системи QoS затримка становила від 41 мс до 207 мс (рис. 4.5). Однак у даній системі QoS затримка коливалася від 3,9 мс до 5,1 мс.

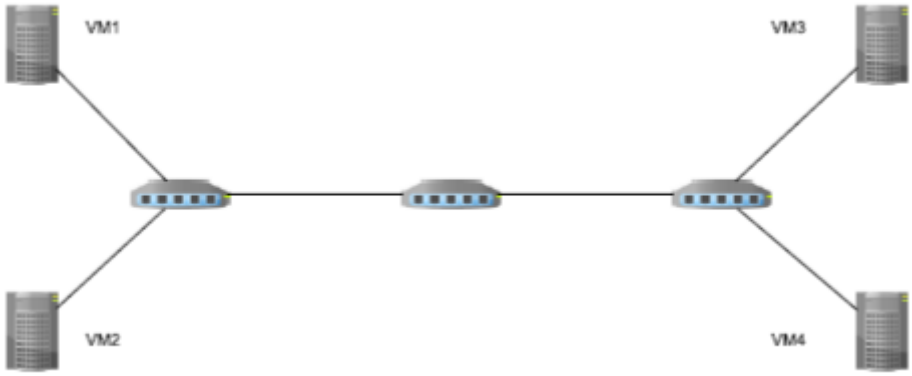


Рисунок 3.4 - Топологія експерименту

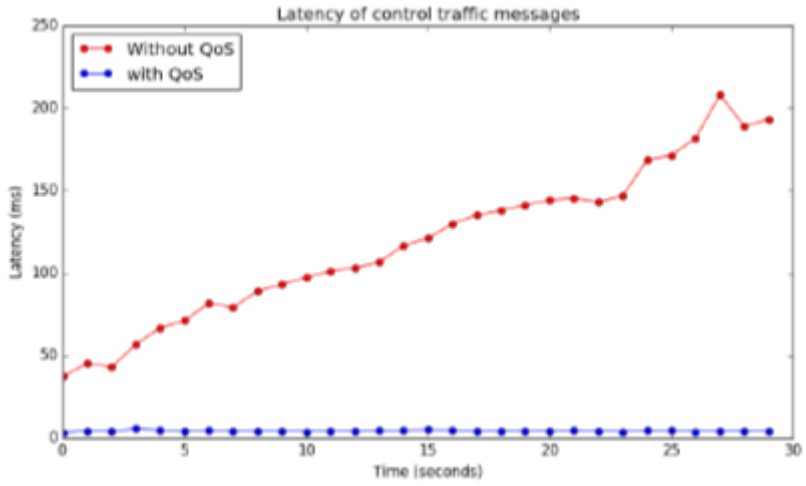


Рисунок 3.5 - Виміряна затримка повідомлень контрольного трафіку

Той самий експеримент повторили з фоновим трафіком TCP. Результати показують, що не було значного впливу на затримку керуючого трафіку навіть без запущеного модуля QoS. Причина полягає в тому, що алгоритм контролю перевантаження для TCP знижує швидкість надсилання, коли він виявляє перевантаження в мережі, підтримуючи так звану дружність до TCP. Крім того, надсилався один потік TCP на той самий шлях. Цей результат показує величезну різницю між двома протоколами.

У цьому розділі було запропоновно систему для вирішення проблеми забезпечення QoS трафіку, чутливого до затримок. Затримка в черзі становить значну частину від загальної затримки переданих пакетів. Використовуючи SDN, запропонована система динамічно розподіляє черги для різних класів трафіку. Він вміщує класи з різними пріоритетами та атрибутами. Результати оцінки показують, що затримка високопріоритетного трафіку зводиться до мінімуму навіть за наявності фонового трафіку UDP, який насичує мережевий тракт.

3.5 Підвищення пропускної здатності великих потоків за допомогою багатопроменевого TCP

Останні розробки в топологіях центрів обробки даних пропонують високу пропускну здатність і кілька шляхів між вузлами обробки. Збільшення використання додатків для хмарних обчислень створює проблеми для базової мережевої інфраструктури. Для вирішення цих завдань потрібне ефективне управління трафіком, яке може максимізувати використання багатошляхових мереж центрів обробки даних. Різноманіття шляхів не тільки дає можливість виконувати балансування навантаження та покращує відмовостійкість, але також може бути використане для виділення більшої пропускної здатності для великих потоків та підвищення продуктивності додатків, обмежених мережею.

У цьому розділі представлено архітектуру, засновану на багатошляховому TCP і програмно-визначеній мережі, щоб забезпечити кращий розподіл пропускної здатності для великих потоків. Контролер SDN, який володіє глобальними знаннями про топологію мережі та вимірювання трафіку, здатний приймати обґрунтовані рішення щодо маршрутизації. Запропонована архітектура SDN дозволяє додаткам досягати кращої пропускної здатності для великих потоків шляхом ініціювання нових підпотоків MPTCP. Щоб забезпечити таку можливість, ми модифікуємо реалізацію MPTCP у ядрі Linux, щоб дозволити програмам створювати додаткові підпотоки MPTCP на вимогу. Ці підпотоки

MPTCP розміщуються централізованим контролером на найменш завантажених шляхах. Результати оцінки, отримані в результаті проведення експериментів на випробувальному стенді GENI, показують значне поліпшення пропускної здатності великих потоків.

Центри обробки даних еволюціонували в останні роки, щоб забезпечити високу пропускну здатність і кілька шляхів між вузлами обробки. Це було зумовлено активнішим розгортанням різних додатків, таких як обробка великих даних і хмарні обчислення. Для максимального використання декількох шляхів існує потреба в ефективних методах маршрутизації. Багатошляхова система рівної вартості (ECMP) на основі хешу може бути використана для багатошляхової маршрутизації для розподілу потоків трафіку між рівними шляхами. Однак ECMP не може повною мірою використовувати наявну пропускну здатність кількох шляхів. В основному це пов'язано з хеш-зіткненням великих потоків, що може призвести до вузьких місць у мережі, тоді як інші шляхи використовуються недостатньо. На рисунку 3.6 показані два приклади, коли колізії хеш-хешів ECMP впливають на пропускну здатність довготривалих потоків. Два потоки мали один і той же зв'язок через зіткнення хешу на агрегатному шарі, а два інших потоки зіткнулися на основному шарі. Таке призначення шляху знижує пропускну здатність усіх чотирьох потоків на 50%.

MPTCP може полегшити цю проблему, створюючи кілька підпотоків, які можуть проходити різними шляхами. Контроль перевантажень MPTCP адаптується до виявлених перевантажених шляхів і збільшує трафік інших підпотоків. Він робить це, зберігаючи при цьому зручність TCP, гарантуючи, що MPTCP і звичайний TCP можуть співіснувати в одному середовищі. Однак використання ECMP з MPTCP не гарантує, що підпотоки одного і того ж з'єднання будуть проходити різними шляхами. Таким чином, покладання на ECMP для прямих транспортних потоків може обмежити потенціал приріст пропускної здатності при використанні MPTCP.

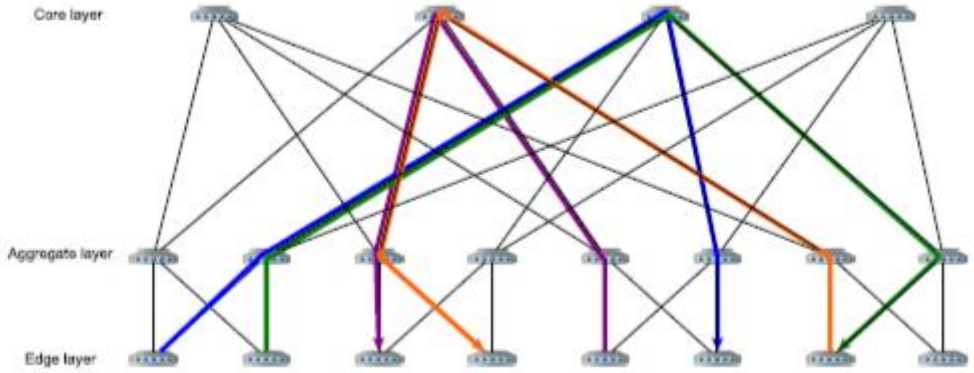


Рисунок 3.6 - Два приклади колізії хешів ECMP

Хоча корисно використовувати MPTCP для великих потоків, це може мати негативний вплив на короткі потоки. Кожен новий підпотік MPTCP тягне за собою додаткові накладні витрати і споживає ресурси хоста. Крім того, підтримка кількох підпотоків MPTCP, кожен з яких проходить різним шляхом, може призвести до збільшення затримки. Короткі потоки, як правило, чутливі до затримки. Тому MPTCP слід використовувати тільки для великих потоків.

Запропоновано архітектуру на основі SDN для підвищення пропускної здатності великих потоків в дата-центрах. У цій архітектурі програми створюють нові підпотоки MPTCP для тривалих з'єднань. Ці новостворені допоміжні потоки маршрутизуються централізованим планувальником потоку в контролері SDN. Слід зазначити, що перший підпотік кожного тривало працюючого з'єднання і короткі потоки не передаються на контролер. Таким чином, немає затримки планування для коротких потоків. Також підключення великого потоку не буде перервано, оскільки перший підпотік буде маршрутизований за допомогою правил балансування навантаження OpenFlow, завчасно встановлених контролером SDN. Перший підпотік залишиться на тому ж шляху до завершення передачі даних.

3.5.1 Топології ЦОД

Проектування мереж центрів обробки даних останнім часом було зосереджено на використанні більш дешевого обладнання, а не дорогих маршрутизаторів і комутаторів високого класу. Мотивація цієї тенденції полягає в побудові більш економічно ефективних мереж, які відповідають сучасним вимогам до обчислень. Кілька конструкцій вирішують такі проблеми, як вузькі місця та обмежена пропускна здатність традиційних мереж центрів обробки даних, з'єднуючи між собою кілька товарних комутаторів.

Топологія k-ary Fat Tree — це топологія Clos, яка розташовує k-портові перемикачі в багатокореневій деревоподібній структурі. Основний шар має перемикачі $(k/2)^2$, з'єднані з k капсулами. Кожен под містить перемикачі шарів агрегації k/2 та перемикачі k/2 крайових шарів. Топологія підтримує $(k^3)/4$ хости. На рисунку 3.7 показана така топологія, де k = 6. Інші конструкції включають BCube, Jellyfish і VL2.

Ці топології ставлять перед центрами обробки даних нові виклики, пов'язані з організацією трафіку. Архітектура SDN може зіграти значну роль у вирішенні цих завдань. Програми-контролери здатні відстежувати мінливі умови трафіку та застосовувати правила OpenFlow для адаптації до поточних вимог до трафіку.

3.6 Багатошляховий TCP

Multipath TCP – це розширення звичайного протоколу TCP, яке надає можливість одному TCP-з'єднанню працювати на декількох шляхах. Це розширення не вимагає будь-яких змін на прикладному рівні, оскільки транспортний рівень прозоро обробляє кілька з'єднань, які називаються підпотоками, які відображаються як єдине з'єднання з додатком. Він також прозорий для мережевого рівня, оскільки кожен підпотік виглядає як звичайне TCP-з'єднання. На рисунку 4.7 зображені MPTCP і TCP в стеку протоколів. Кожен підпотік MPTCP може проходити через різний шлях у мережі або навіть інший

мережевий інтерфейс, якщо хост має кілька мережевих інтерфейсів (багатодомний хост). Розширення MPTCP забезпечує більшу стійкість протоколу TCP, оскільки з'єднання не буде перервано, якщо одна ланка вийде з ладу, поки існує хоча б один підключений підпотік. Ще однією перевагою є надання додаткам більшої пропускної здатності шляхом агрегування кількох шляхів.

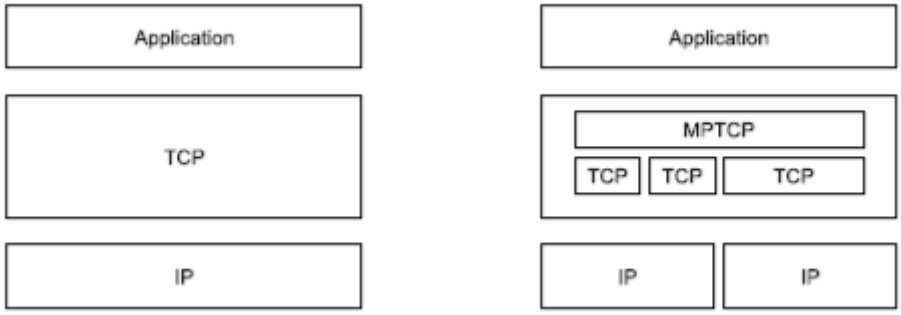


Рисунок 3.7 - Протоколи TCP і MPTCP

Багатошляховий сеанс TCP запускається так само, як і звичайний сеанс TCP. Різниця полягає в тому, що пакети SYN, SYN/ACK та ACK містять тип опції MPTCP з підтипом опції MP ABLE. Підтип параметра MP ABLE використовується першим підпотоком з'єднання MPTCP, щоб вказати, що хост-відправник підтримує MPTCP. Якщо вузол-відправник отримує SYN/ACK без MP ABLE, припускається, що інший вузол не підтримує MPTCP. Інша можливість, хоча і дуже малоймовірна, полягає в тому, що застарілий проміжний блок змінює параметри TCP, оскільки його налаштовано на видалення нерозпізнаних параметрів TCP. У цьому випадку з'єднання повертається до звичайного TCP, і два хости продовжують обмінюватися пакетами через єдиний потік TCP. Якщо обидва хости завершили тристороннє рукостискання за допомогою MP CAPABLE, то вони встановили багатопроменеве TCP-з'єднання, і будь-який з них може ініціювати додаткові підпотоки. Під час початкового тристороннього рукостискання два хости обмінюються ключами, які використовуються для зв'язку наступних підпотоків із цим з'єднанням. Ключі унікальні для кожного з'єднання. На рисунку 4.8 показаний процес встановлення багатопроменевого

TCP-з'єднання за допомогою MP ABLE і подальшого додавання одного додаткового підпотоку за допомогою MP JOIN.

Ключі, якими обмінюються в рукостисканні MP ABLE, використовуються для генерації криптографічних хешів, які називаються токенами. Після цього ключі не відправляються по мережі. Замість цього криптографічний хеш ключа, або токен, використовується для ідентифікації з'єднання. Токени використовуються в MP JOIN для зв'язування підпотоку з існуючим з'єднанням MPTCP. Мета використання токенів полягає в тому, щоб переконатися, що новий підпотік приєднається до правильного з'єднання MPTCP. Код автентифікації повідомлень на основі хешу (HMAC), згенерований з ключа та nonce (випадкове число), використовується для автентифікації іншого хоста.

Нові підпотоки MPTCP не обов'язково повинні знаходитися між різними парами IP. Можна мати кілька підпотоків з однієї пари IP, враховуючи, що вони працюють на різних портах TCP. Як правило, це вигідно там, де ECMP використовується для перенаправлення транспортних потоків, оскільки ці підпотоки можуть мати різні шляхи, що призведе до збільшення пропускної здатності програми. Однак, ECMP використовує хешування, і немає гарантії, що кілька підпотоків одного і того ж з'єднання виявляться в несумісних шляхах.

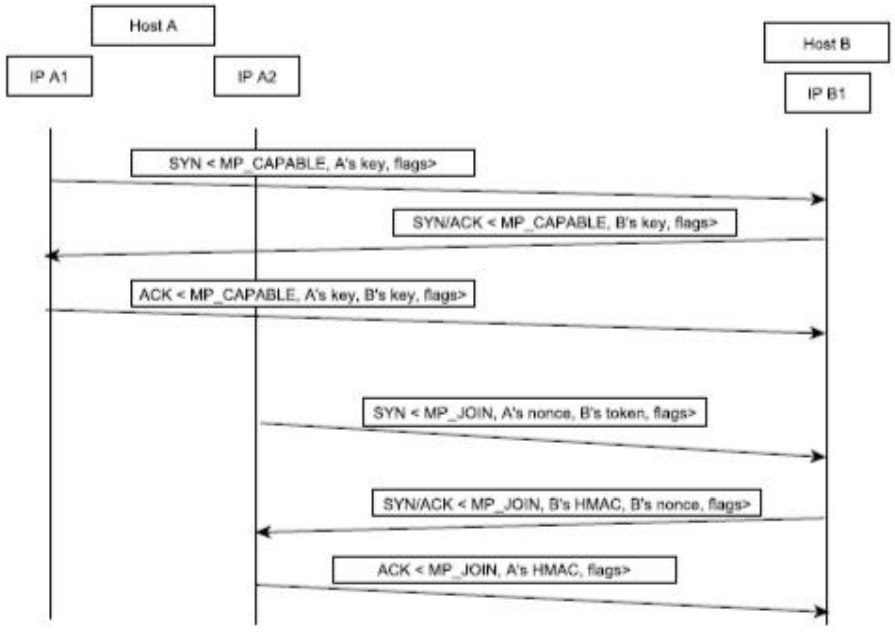


Рисунок 3.8 - Ініціювання підпотоків MPTCP

3.6.1 Контроль заторів

У звичайному TCP контроль перевантажень відіграє ключову роль в уникненні колапсу перевантаження в мережі. TCP підтримує так зване вікно перевантаження, щоб підтримувати швидкість передачі нижче рівня, який може спричинити перевантаження мережі. По суті, це обмежує кількість непідтверджених даних на вихідному хості. Розмір вікна перевантаження змінюється протягом усього терміну служби з'єднання, щоб адаптуватися до виявлених сигналів перевантаження. Хост призначення зазвичай надсилає підтвердження (ACK) до файлу вихідний хост для кожного отриманого пакета. Вихідний хост робить висновок, що сталася втрата пакета, коли він не отримує ACK до тайм-ауту. TCP обчислює цей тайм-аут, званий тайм-аутом ретрансляції (RTO), на основі згладженого значення розрахункового часу проходження туди й назад (RTT) між вихідним і кінцевим хостами. Крім RTO, сигналом про перевантаження вважаються дублікати підтверджень (DupACK). Коли хост отримує пакет, що вийшов з ладу, він надсилає ACK для останнього пакета замовлення, який він отримав до цього часу. Відправка DupACK означає, що одержувач все ще очікує наступний пакет. Однак, оскільки пакет може бути затриманий або перевпорядкований під час передачі, для виявлення втрати пакета необхідні три DupACK.

Коригування TCP для вікна перевантаження слідують за наближенням адитивного збільшення/мультиплікативного зменшення (AIMD). Це означає, що TCP збільшить швидкість передачі, додавши значення до вікна перевантаження, коли він отримає новий ACK, і зменшить швидкість передачі на мультиплікативний коефіцієнт (наприклад, поділити на 2), коли буде виявлено втрату пакета. Така консервативна поведінка має важливе значення для уникнення перевантаження мережі.

У MPTCP роль контролю перевантажень розширюється, включаючи проблеми, яких немає в одноконтурному TCP. Основними проблемами є

справедливість по відношенню до звичайного TCP і перенесення трафіку з перевантажених доріжок на інші шляхи, які мають менший трафік. Щоб пояснити проблему справедливості, врахуйте, що кожен підпотік MPTCP поводиться як звичайний TCP. Якщо одне TCP-з'єднання має спільне вузьке місце лише з одним MPTCP-з'єднанням, яке має два підпотоки, це призведе до того, що потік TCP отримає вдвічі меншу пропускну здатність, ніж MPTCP-з' єднання. Цей механізм відомий як контроль незв'язаних перевантажень. У цьому механізмі кожен підпотік MPTCP s підтримує свої власні вікна перевантаження Ws, і поведінка AIMD виглядає наступним чином:

- Кожен ACK на підпотоці s збільшить вікно W_s на $1/W_s$.
- Кожна виявлена втрата підпотоку s буде зменшуватися W_s на $W_s/2$.

Сумарна пропускна здатність з'єднання MPTCP надмірно велика. Однак цей виграш може бути за рахунок конкуруючих потоків TCP на спільних вузьких місцях. Така поведінка занадто агресивна по відношенню до однопотокового TCP і порушує принцип проектування «не нашкодь». Для того, щоб новий протокол був прийнятий, він повинен бути справедливим по відношенню до існуючих протоколів. Тому підпотоки MPTCP не повинні поводитися як звичайні потоки TCP.

Сполучений контроль перевантажень намагається вирішити цю проблему, змінюючи параметри AIMD. У зв'язаному управлінні перевантаженнями кожне з'єднання MPTCP підтримує вікно перевантаження W_{total}, яке дорівнює сумі всіх W, підтримуваних його підпотоками. Величина W_s обов'язково буде ≥ 1 . Поведінка AIMD виглядає наступним чином:

- Кожен ACK на підпотоці s збільшить вікно W_s на $1/ W_{total}$.
- Кожна виявлена втрата підпотоку s буде зменшуватися W_s на $W_{total}/2$.

Контроль зчеплених перевантажень досягає мети справедливості по відношенню до регулярного TCP, беручи до уваги сумарне вікно перевантаження всіх підпотоків. Крім того, він балансує трафік між підпотоками на основі сигналів про перевантаження. Переміщуючи трафік з підпотоків на перевантажених ланках на інші підпотоки на найменш завантажених шляхах,

пропускна здатність інших потоків на перевантажених вузьких місцях збільшиться. Отже, загальна мережа досягне кращого використання ресурсів і балансування навантаження. Однак сполучене управління заторами має недоліки. По-перше, він надає перевагу шляхам з меншим коефіцієнтом втрати пакетів, незалежно від їх значень RTT. Деякі маршрутизатори оснащені великими буферами для зменшення падіння пакетів, що в той же час може призвести до збільшення RTT. Однак менше значення RTT призведе до більшої пропускної здатності. Друга проблема полягає в тому, що трафік може бути повністю зміщений з перевантажені шляхи, немає способу вказати, що такі шляхи доступні, коли вони стають менш завантаженими. Стає зрозумілим, що на кожному підпотоці має бути достатній трафік, щоб зондувати шляхи, незалежно від того, наскільки вони перевантажені.

Алгоритм пов'язаних збільшень (LIA) був запропонований для усунення недоліків контролю зчеплених перевантажень. LIA має на меті досягнення наступних цілей:

1. Покращення пропускної здатності: з'єднання MPTCP має досягати пропускної здатності, що дорівнює або перевищує пропускну здатність, якої досяг би один потік TCP на найкращому шляху, доступному для MPTCP.

2. Не завдає шкоди: підпотоки MPTCP одного і того ж з'єднання, що спільно використовують мережевий ресурс, не повинні займати більше пропускної здатності, ніж один потік на тому ж шляху.

3. Балансування заторів: З'єднання MPTCP має максимально перемістити трафік із найбільш завантажених доріжок на найменш завантажені шляхи.

Поведінка AIMD в LIA виглядає наступним чином:

- Кожен ACK на підпотоках s збільшить вікно W_s на:

$$\min\left(\frac{\alpha * bytes\_acked * MSS_s}{W_{total}}, \frac{bytes\_acked * MSS_s}{W_s}\right)$$

Мінімальне значення приросту дорівнює 1. MSS_s - максимальний розмір сегмента для підпотоку s.

- Кожна виявлена втрата підпотоку s буде зменшуватися W_s на $W_s/2$.

α - параметр, який контролює агресивність MPTCP. Він розраховується на основі передбачуваного RTT наступним чином:

$$\alpha = W_{total} \frac{\max_s(\frac{W_s}{(RTT_s)^2})}{(\sum_s(\frac{W_s}{RTT_s}))^2}$$

Сумарна пропускна здатність MPTCP-з'єднання в LIA залежить від значень α, коефіцієнта втрати пакетів, RTT і MSS для кожного з його підпотоків. LIA є частиною реалізації MPTCP ядром Linux. Ми використовували цей контроль перевантажень у нашій демонстрації, оскільки він забезпечує кращий приріст пропускної здатності, не завдаючи шкоди звичайним потокам TCP. Крім того, для досягнення адаптивного балансування навантаження важливо переміщати трафік між підпотоками залежно від сигналів перевантаження.

3.7 Архітектура на основі SDN для підвищення пропускної здатності великих потоків

Запропонована архітектура спрямована на підвищення пропускної здатності великих потоків, не впливаючи при цьому на короткі потоки в мережах центрів обробки даних. Він поєднує в собі архітектуру SDN і протокол MPTCP. Транспортні протоколи, такі як MPTCP, не мають знань про топологію мережі і не можуть диктувати, як маршрутизуються пакети в мережі. Контролер SDN володіє глобальними знаннями топології мережі і вміє отримувати актуальну статистику трафіку кожного порту комутатора. Додаток контролера починається з налаштування правил OpenFlow для маршрутизації коротких потоків без пересилання контролеру. Потім він встановлює правила для перенаправлення допоміжних підпотоків до контролера, які будуть оброблятися модулем пошуку шляху.

Допоміжні підпотоки одного і того ж з'єднання не обов'язково розміщуються на роз'єднаних шляхах. Додаток контролера SDN виконує пошук

на основі поточної доступної ємності кожного каналу. Цей адаптивний пошук збільшує потенційний приріст пропускної здатності кожного додаткового підпотоку. Щоб проілюструвати різницю між пошуком за непере'єднаним шляхом і адаптивним пошуком шляху, розглянемо спрощений приклад, показаний на рисунку 3.9. Посилання показують доступну пропускну здатність у поточних умовах мережі. Якщо ми маємо два підпотоки і призначаємо їх найкоротшим непересічним шляхам, їх сумарна пропускна здатність становитиме 4 Мбіт/с. Однак сукупна пропускна здатність збільшується до 9 Мбіт/с при використанні нашої адаптивної переадресації SDN.

Першим кроком у цьому процесі є диференціація коротких і довгих потоків. Після того, як потік буде визначено як довгий, він буде розділений на кілька підпотоків, щоб отримати більшу пропускну здатність. З метою демонстрації ми вирішили не встановлювати жорстких порогових значень і делегувати це завдання прикладному рівню, щоб мати більше гнучкості в експериментах з різними рівнями порогових значень. Великі потоки будуть протікати на мутиплових допоміжних підпотоках на додаток до першого підпотоку. Допоміжні підпотоки матимуть позначене поле DSCP, яке запускатиме перемикачі для пересилання їх контролеру. Основними компонентами цієї архітектури є:

- Балансування навантаження на основі OpenFlow для коротких потоків за допомогою групових таблиць.
- Адаптивний планувальник потоків для маршрутизації допоміжних підпотоків великих потоків.
- Розширено програмний інтерфейс сокета для реалізації MPTCP ядром Linux.

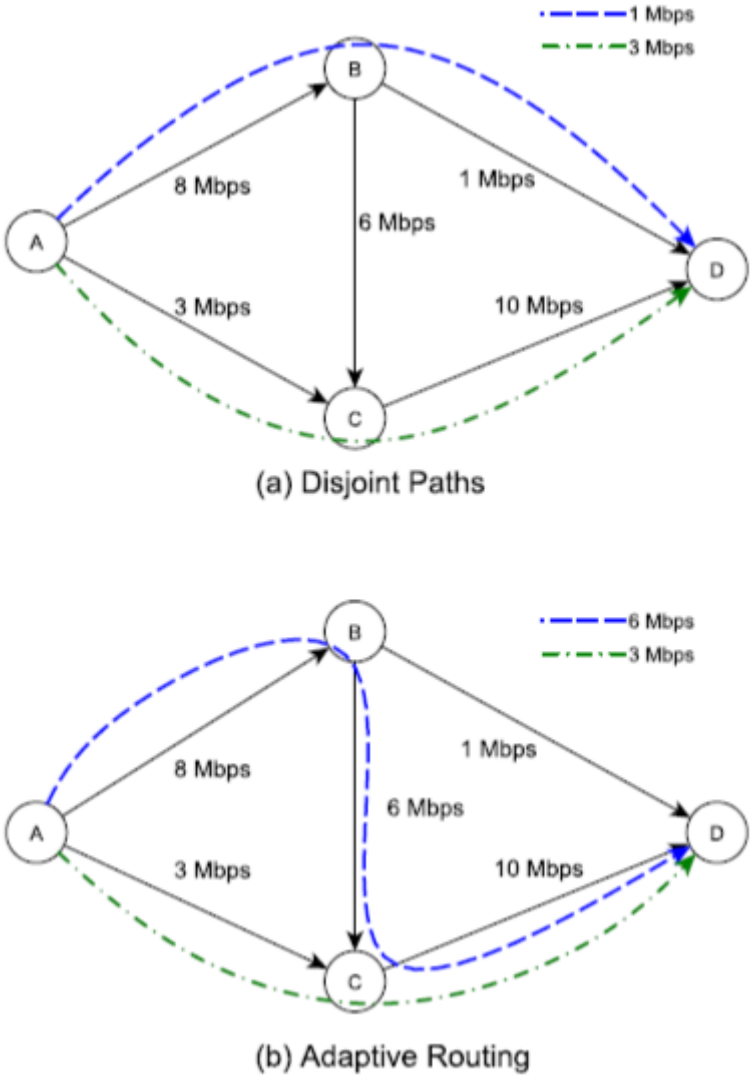


Рисунок 3.9 - Непересічні шляхи vs адаптивна маршрутизація

3.7.1 Балансування навантаження за допомогою групових таблиць OpenFlow

Перша частина запропонованої нами архітектури пов'язана з маршрутизацією коротких потоків і перших підпотоків великих потоків. Короткі потоки, як правило, чутливі до затримки і надходять у великій кількості. Короткі потоки, що передаються на контролер SDN для обробки

Вони реактивні не є практичними. Це не масштабоване рішення, оскільки воно збільшує кількість правил OpenFlow і створює значне навантаження на контролер. Крім того, затримка обробки в площині управління збільшує затримку. Тому краще обробляти їх тільки в площині даних. Контролер може завчасно встановлювати правила переадресації для переадресації трафіку. Для виконання балансування навантаження нам потрібен ефективний метод розділення транспортних потоків, які не маршрутизуються реактивно контролером.

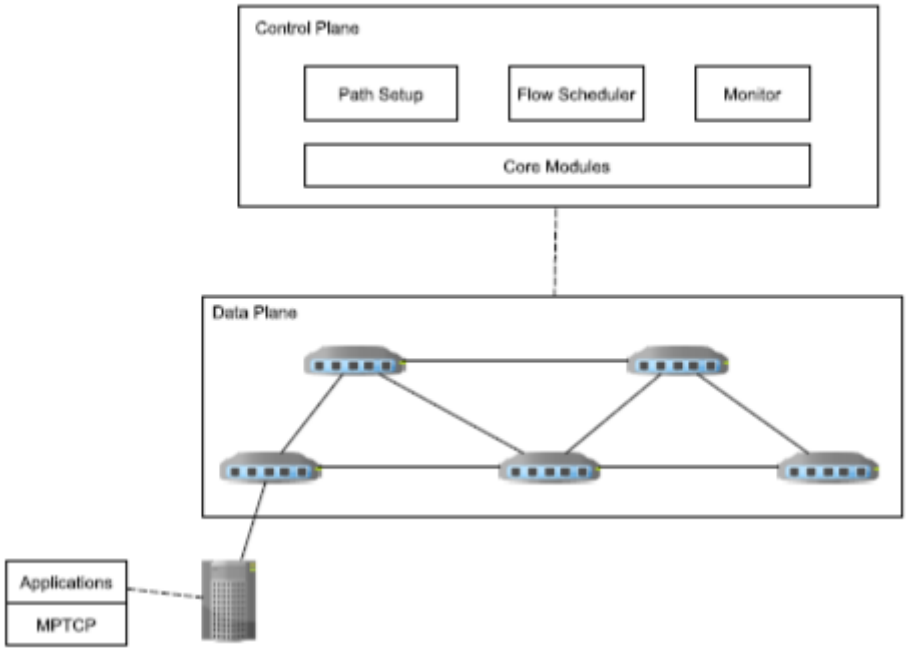


Рисунок 3.10 - Архітектура на основі SDN для підвищення пропускної здатності великих потоків

Групові таблиці були введені в OpenFlow версії 1.1. Вони забезпечують абстракцію для представлення набору портів як єдиного цілого. Кожна таблиця груп OpenFlow містить набір сегментів дій. Ці сегменти визначають дії, які будуть застосовані до пакетів із відповідних потоків. Існують різні типи груп, які можна використовувати для різних цілей. До групових типів належать:

1. Непрямий: містить лише одне відро. Зазвичай це корисно у випадку, коли кілька записів ланцюжка повинні вказувати на один загальний набір дій.

2. Усі: містить кілька відер. Усі сегменти виконуються для кожного відповідного пакета потоку. Це корисно для таких програм, як багатоадресна передача або трансляція.

3. Виберіть: Містить кілька відер. Тільки один сегмент виконується для кожного відповідного узгодженого потоку (рис. 3.6).

4. Швидке перемикання на відмову: містить упорядкований список сегментів. Кожен сегмент пов'язаний з портом/групою для відстеження його стану. Він виконує перше відро, статус якого активний.

Ми використовуємо групові таблиці OpenFlow для розподілу потоків між різними контурами. Потоки трафіку, які відповідають встановленим правилам OpenFlow, будуть перенаправлені до групових таблиць. Кожне відро має пов'язану вагу, яка визначає ймовірність вибору ковша для кожного відповідного потоку. Алгоритм вибору сегмента не визначений в специфікаціях OpenFlow і залишається для перемикання реалізації. Можливі реалізації включають зважені кругові алгоритми та алгоритми хешування. OpenvSwitch, який використовується в нашій оцінці, реалізує хешування для вибору сегмента.

У нашій конструкції контролер встановлює таблиці груп OpenFlow у кожному комутаторі та відповідні правила узгодження для рівномірного розподілу потоків трафіку між кількома шляхами. Правила відповідності встановлюються з найнижчим пріоритетом. Хоча можливо, що хешування може призвести до нерівномірного розподілу на початку, воно сходиться до рівномірного розподілу зі збільшенням кількості потоків.

3.7.2 Маршрутизація допоміжних потоків

Допоміжні потоки відрізняються від інших потоків трафіку своїм полем диференційованих послуг (DS). Поле DS містить 6-бітне поле Differentiated Services Codepoint (DSCP) і 2-розрядне поле Explicit Congestion Notification (ECN). Про це повідомляє DSCP спочатку використовується в диференційованих послугах (DiffServ) для класифікації трафіку на різні класи на основі вимог до

якості обслуговування (QoS). Ми використовуємо пул кодових точок, які зарезервовані для експериментального або локального використання. Цей пул має 16 різних значень у просторі кодових точок xxxx11.

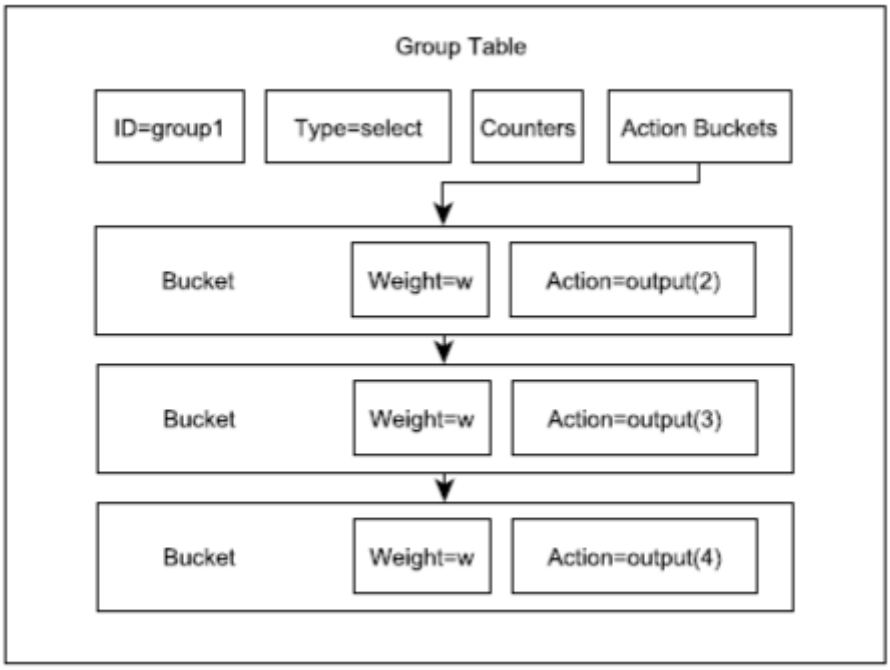


Рисунок 3.11 - Таблиця груп OpenFlow

Контролер встановлює правила OpenFlow в площині даних для перенаправлення транспортних потоків з позначеними DSCP (допоміжними підпотоками) на площину управління для обробки. Ці правила мають вищий пріоритет, ніж правила групової таблиці, які обробляють пересилання коротких потоків і перших підпотоків з'єднань MPTCP. Після того, як повідомлення PACKET IN надходить до контролера, воно обробляється нашим модулем раніше, ніж будь-який інший модуль контролера (наприклад, вбудований модуль пересилання). Ми виконуємо інспекцію пакетів інкапсульованого пакета в повідомленні PACKET IN, щоб витягти поле DSCP і параметри TCP. Якщо значення DSCP знаходиться в межах нашого попередньо визначеного діапазону, ми аналізуємо параметри TCP, щоб перевірити, чи є це пакетом MPTCP (тобто він належить до нового допоміжного підпотоку) або звичайним пакетом TCP.

Сегменти TCP можуть мати змінну кількість необов'язкових полів заголовка, які називаються параметрами TCP. Їх призначення - надати можливість створювати нові розширення, недоступні в оригінальних TCP-заголовках, не вимагаючи зміни існуючої структури. Кожен варіант TCP позначається обов'язковим видом опціону. Довжина виду опціону завжди становить один октет. Однак довжина параметрів TCP є змінною. Ці параметри підтримуються Управлінням з присвоєння номерів в Інтернеті (IANA). Два варіанти TCP не мають додаткової інформації, окрім типу параметра. Параметр END_OF_LIST (0x00), який вказує на кінець списку параметрів, і параметр NO_OPERATION (0x01), який використовується для відступів. Усі інші параметри мають два інші поля: довжину параметра та дані параметра. Довжина параметра визначає довжину всіх трьох полів (а не лише даних параметра). Різновидом параметра MPTCP є 30 (0x1e). У початковому пакеті SYN MP JOIN довжина параметра MPTCP дорівнює 12. Щоб витягти токен MPTCP, нам потрібно проаналізувати параметри TCP (алгоритм 2). Перші чотири біти даних опції містять MPTCP subType (список підтипів наведено в таблиці 3.1), за якими слідують чотири біти прапорців і один октет для ідентифікатора адреси. Наступні чотири октети є жетоном приймача. На рисунку 3.12 представлена структура опції MPTCP для пакета MP JOIN SYN.

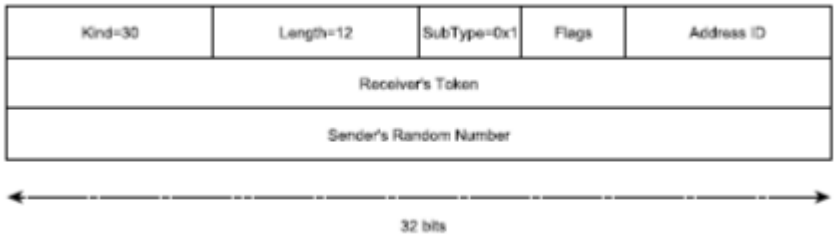


Рисунок 3.12 - Опція MPTCP для пакета MP JOIN SYN

Algorithm 2 extractMPTCPToken(packet)

```
tcpOptions ← getTCPOptions(packet)
i=0
while i < tcpOptions.length do
    opKind = tcpOptions[i]
    switch opKind do
        case 0x00:                                          ▷ End_Of_List
            return null
        case 0x01:                                          ▷ No_Operation
            i += 1
            continue
        default:
            i += 1
            opLen ← tcpOptions[i]
            if opKind==0x1e then                            ▷ MPTCP option
                i += 1
                subType ← tcpOptions[i] & 0xF0              ▷ subType is only 4 bits
                if subType==0x10 and opLen==12 then         ▷ MP_JOIN in SYN
                    token ← Copy(tcpOptions, i+2, i+6)
                    return token
                else                                        ▷ not MP_JOIN
                    return null
                end if
            else                                            ▷ other TCP option
                i += (opLen -1)
            end if
end while
return null
```

Для кожного нового допоміжного підпотоку наш модуль виконує алгоритм пошуку, щоб знайти список найкращих шляхів для цього з'єднання (алгоритм 3). Спочатку він витягує токен приймача MPTCP, пов'язаний із цим допоміжним підпотоком. З'єднання MPTCP ідентифікуються за унікальними токенами приймача. Якщо він збігається з кешованим потоком, то його шлях витягується та встановлюється. В іншому випадку це означає, що допоміжний підпотік належить новому з'єднанню MPTCP. Найкращим шляхом у нашому випадку є найширший шлях, отриманий модифікованою версією алгоритму Дейкстри (алгоритм 4). Пошук ґрунтується на поточній топології та оновлених вимірюваннях трафіку. Контролер періодично збирає статистику трафіку з кожного комутатора.

Якщо вхідне повідомлення PACKET IN призначене для першого допоміжного підпотоку MPTCP-з'єднання, список шляхів кешується для наступних допоміжних підпотоків цього з'єднання. Таким чином, алгоритм пошуку потрібно виконувати тільки один раз для кожного з'єднання, незалежно

від кількості його допоміжних потоків. Встановлені правила OpenFlow мають м'який тайм-аут і будуть видалені після того, як стануть неактивними. Перемикачі сповіщають контролер про прострочені правила OpenFlow, надсилаючи повідомлення FLOW REMOVED, що дозволить контролеру очистити відповідний запис у кешованих потоках. Використання цього методу не перериває трафік, що йде через перший підпотік, який маршрутизується за правилами групової таблиці OpenFlow без переходу до контролера. Це дозволяє лише великим потокам отримати більшу пропускну здатність і швидше закінчити за рахунок використання кількох підпотоків, різноманітності шляхів і SDN. Система контролю заторів для MPTCP адаптується до мінливих дорожніх умов і переміщує трафік з перевантажених каналів на інші підпотоки, не завдаючи шкоди існуючим TCP-з'єднанням.

Algorithm 3 getPath(packet)

```
token← extractMPTCPToken(packet)
if exists(cachedPaths[token]) then
    p← dequeue(cachedPaths[token])
    install(p)
    return
end if
linkCapacity← topology.getAvailableCapacity()
for i← 0 to maxPaths-1 do
    p← dijkstra(flow.src, flow.dst,linkCapacity)
    w← width(p)
    for l← p.links do
        linkCapacity[l]← linkCapacity[l]-w
    end for
    cachedPaths[token].add(p)
end for
p← dequeue(cachedPaths[token])
install(p))
```

```
Algorithm 4 Dijkstra(s,v,graph)
  q← empty
  visited← empty
  for node← graph.nodes do
      width[node]← 0
      previous[node]← null
  end for
  width[s]← ∞
  q.add(s)
  while q is not empty do
      node← q.dequeue()
      if visited.contains(node) then
          continue
      end if
      visited.add(node)
      for link← graph.connectedTo(node) do
          neighbor← link.getDestination()
          altWidth← max(width[neighbor],
              min(width[node],link.getWidth())
          if altWidth > width[neighbor]  then
              width[neighbor]← altWidth
              previous[neighbor]← node
              q.add(neighbor)
          end if
      end for
  end while
```

3.7.3 Socket API для створення допоміжних підпотоків

Для того, щоб додатки могли використовувати нашу архітектуру SDN, їм потрібна можливість створювати підпотоки MPTCP і позначати DSCP одночасно. Реалізація ядра Linux MPTCP не надає такої можливості. Дослідники запропонували розширений API сокета для MPTCP, який розширює реалізацію ядра Linux для надання додаткових можливостей. Розширений API сокетів надає програмам додаткові можливості, які недоступні в оригінальній реалізації ядра. Мета полягає в тому, щоб дозволити розробникам додатків мати більший контроль над роботою MPTCP. Список їхніх функцій API включає, серед іншого, отримання інформації про підпотоки, створення підпотоків, завершення підпотоків і налаштування поля DSCP.Функції.

Незважаючи на те, що розширений API сокетів забезпечує більший контроль над операціями MPTCP, вони не відповідають нашим вимогам. Причина

полягає в тому, що налаштування DSCP може відбуватися тільки після ініціації підпотоку, а це означає, що позначити DSCP для SYN-пакетів неможливо. У нашій архітектурі SDN SYN-пакети повинні мати певне значення DSCP, щоб їх можна було передати контролеру. Контролеру необхідно витягти токени приймача MPTCP з SYN-пакетів. Токени використовуються як унікальні ключі для ідентифікації MPTCP-з'єднань. У поточних специфікаціях OpenFlow неможливо створити правила відповідності OpenFlow параметрам TCP. Тому ми покладаємося на поле DSCP, щоб розрізняти допоміжні підпотоки та інші потоки. Токен MPTCP надсилається лише з пакетом MP JOIN SYN під час тристороннього рукостискання TCP. Підтримка токенів MPTCP дозволяє контролеру розміщувати допоміжні підпотоки одного і того ж з'єднання на кращих шляхах з найбільшою доступною пропускною здатністю. З цією метою ми розробили новий API сокетів, який дозволяє програмам динамічно створювати підпотоки MPTCP з позначкою DSCP для існуючого з'єднання MPTCP. Поле DSCP задається ініціацією потоку (тобто SYN-пакетом).

Коротко опишемо розширений API сокетів. Нові функціональні можливості були реалізовані над системними викликами getsockopt і setsockopt. Вони представили нові параметри сокетів для запиту інформації на рівні ядра та виконання дій над існуючими MPTCP-з'єднаннями. До варіантів розеток можна віднести наступне:

- MPTCP_GET_SUB_IDS: отримати поточний список ідентифікаторів підпотоків, переглянутий ядром.
- MPTCP_GET_SUB_TUPLE: отримати IP-адресу та номер порту певного підпотоку.
- MPTCP_OPEN_SUB_TUPLE: запит на створення нового підпотоку для існуючого MPTCP-з'єднання.
- MPTCP_CLOSE_SUB_ID: запит на закриття конкретного підпотоку.
- MPTCP_SUB_GETSOCKOPT: передайте параметр socket для getsockвибрати певний підпотік і повернути результат.

- MPTCP_SUB_SETSOCKOPT: передайте параметр socket, щоб setsockopt для певного підпотоку.

За допомогою MPTCP_SUB_SETSOCKOPT можна задати значення DSCP конкретного підпотоку. Однак, як ми обговорювали раніше, це відбувається тільки після того, як підпотік вже ініціалізований і обмін тристоронніми пакетами рукостискання. Ми представляємо нову опцію сокета MPTCP_OPEN_SUB_WITHTOS, яка створює новий підпотік і одночасно позначає DSCP. Тобто SYN-пакети матимуть задане значення DSCP. Додаток повинен передати mptcp_newsub_withtos структуру, яка визначається наступним чином:

```
struct mptcp_newsub_withtos {
char *tosval;          /* Значення DSCP */
struct mptcp_sub_tuple *sub;       /* sub_tuple нового підпотоку*/
}
```

Член tosval задає значення поля DSCP. Зауважте, що поле DSCP має довжину лише 6 бітів. Поряд з двобітним ECN-полем вони замінюють те, що раніше було відоме як поле ToS в IP-пакеті. Біти DSCP 6 є найбільш значущими бітами тосваля. Наприклад, значення DSCP (000111) представлено шістнадцятковим значенням (0x1C), оскільки два найменш значущих біта встановлені в нуль (00011100). Піделемент містить інформацію кортежу нового підпотоку. На рисунку 5.8 показаний приклад використання нової опції сокета MPTCP_OPEN_SUB_WITHTOS для створення одного додаткового підпотоку, який має десяткове значення DSCP 28 (0x1C). Новий підпотік використовує ту саму пару IP і той самий порт призначення існуючого підпотоку, який має ідентифікатор subflow_id. Ідентифікатор підпотоку можна отримати з ядра за допомогою MPTCP_GET_SUB_IDS, який надає список активних підпотоків з їхніми ідентифікаторами. Поточна реалізація обмежує кількість активних підпотоків для одного підключення MPTCP до 32, а підпотокам присвоюються ідентифікатори в діапазоні від 0 до 31. Вихідний порт нового підпотоку

вибирається ядром випадковим чином, але може бути вказаний, якщо потрібно. Інформація про mptcp_sub_tuple витягується за допомогою MPTCP_GET_SUB_TUPLE.

Наша модифікація ядра Linux дуже мінімальна. Він представляє нову функцію API сокетів, яка дозволяє програмам вказувати значення DSCP під час ініціації підпотоку. Наша робота базується на реалізації ядра Linux MPTCP v0.92. Він також розширює API сокета.

Сценарій використання цього розширеного API полягає в тому, що після певного порогу переданих даних додаток видає виклик для створення нового підпотоку. Це повинно відбуватися тільки при великих потоках, які вимагають багато часу для завершення перенесення даних. Вказівка порогу і кількості допоміжних підпотоків залишається за додатком.

```
struct mptcp_newsub_withtos *newsub;
struct sockaddr *sin;
struct sockaddr_in *sin4;
unsigned int optlen, newlen;
int DSCPvalue = 28;
newlen = 100;
newsub = malloc(newlen);
if (!newsub) {
  fprintf(stderr,"Error: malloc\n");
  return 0;
}
newsub->tosval= (char *) &DSCPvalue;
optlen = 100;
newsub->sub = malloc(optlen);
if (!newsub->sub) {
  fprintf(stderr,"Error: malloc\n");
  return -1;
}
optlen = 100;
newsub->sub->id = subflow_id; // from MPTCP_GET_SUB_IDS
error = getsockopt(sock, IPPROTO_TCP, MPTCP_GET_SUB_TUPLE, newsub->sub
   , &optlen);
if (error) {
  fprintf(stderr,"MPTCP_GET_SUB_TUPLE error: %d\n", error);
  free(newsub->sub);
  free(newsub);
  return -1;
}
sin = (struct sockaddr*) &newsub->sub->addrs[0];
if(sin->sa_family == AF_INET){
  sin4 = (struct sockaddr_in*) &newsub->sub->addrs[0];
  sin4->sin_port = htons(0); //source port for new flow
  error = getsockopt(sock, IPPROTO_TCP, MPTCP_OPEN_SUB_WITHTOS,newsub
   ,&optlen);
  if (error) {
    fprintf(stderr,"MPTCP_OPEN_SUB_WITHTOS error: %d\n", error);
    free(newsub->sub);
    free(newsub);
    return -1;
  }
}
```

Рисунок 3.13 - Створення нового підпотоку за допомогою DSCP

3.8 Налаштування експериментів

Щоб оцінити представлено архітектуру SDN, було проведено експерименти з використанням середовища тестового стенду GENI. Топологія мережі використовується k-арій Fat Tree. Було розгорнуто 6-портову мережу Fat Tree, яка містить 45 комутаторів і 54 хости. Топологія мережі показана на рисунку 4.14. Всі хост-вузли працювали під управлінням Linux Ubuntu з модифікованим ядром

MPTCP. Для вузлів комутатора використовується програмний комутатор OpenvSwitch. Всі комутатори підключаються до контролера Floodlight, на якому працюють наші модулі маршрутизації та моніторингу.

Для транспортних потоків зазначимо, що схеми трафіку дата-центрів розрізняються за розміром, часом прибуття та іншими властивостями. Дослідники вивчили сліди трафіку в десяти центрах обробки даних і виявили, що приблизно 80% всіх потоків мають невеликі розміри і закінчуються менш ніж за 11 секунд в більшості досліджуваних центрів обробки даних. Для емуляції трафіку в центрах обробки даних було згенеровано дві матриці трафіку, які слідують цьому спостереженню. У матриці трафіку TM1 80% потоків невеликі, а в матриці трафіку TM2 70% потоків невеликі. Схема трафіку — це випадкова перестановка, коли кожен хост надсилає потік одному випадковому хосту (в іншій стійці). Час між прибуттям вибирається випадковим чином між 500 мс і 1000 мс. Кожна матриця трафіку містить 540 потоків.

В роботі було розроблено невеликий додаток для генерації трафіку для проведених експериментів. Ця програма може створювати допоміжні підпотоки за допомогою нового API сокета. Допоміжні підпотоки ініціюються після того, як переданий трафік перевищує порогові значення. Метою цього додатка є тестування нового API сокета та програми контролера SDN. Поріг для створення кожного додаткового підланцю становить 1 МБ. Кожна матриця трафіку виконувалася п'ять разів наступним чином:

1. Регулярна маршрутизація TCP та ECMP.
2. MPTCP з 2 підпотоками та маршрутизацією ECMP.
3. MPTCP з 2 підпотоками та допоміжною маршрутизацією SDN.
4. MPTCP з 3 підпотоками та маршрутизацією ECMP.
5. MPTCP з 3 підпотоками та допоміжною маршрутизацією SDN.

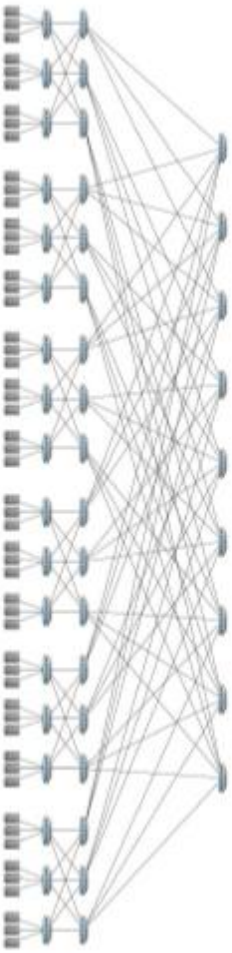


Рисунок 3.14 - Топологія мережі

3.8.1 Результати оцінювання

Результати, отримані в результаті проведення цих експериментів, показують значне поліпшення пропускної здатності великих потоків при використанні MPTCP. Це очікувано, оскільки кілька підпотоків здатні агрегувати пропускну здатність різних шляхів. На рисунку 3.15 показана середня пропускна здатність всіх великих потоків в кожному експерименті. У TM1 середня пропускна здатність для TCP склала 49,4% від максимальної пропускної здатності каналу. Використання MPTCP з ECMP збільшило пропускну здатність до 64,4% при 2 підпотоках і 72,1% при 3 субпотоках. Тоді як у TM2 середня пропускна здатність TCP становила 36,6%. MPTCP з ECMP покращив отриману пропускну здатність до 52,4% при 2 підпотоках і 56% при 3 підпотоках. Однак для всіх випадків

MPTCP наша допоміжна маршрутизація SDN показала себе набагато краще, ніж ECMP. У TM1 середня пропускна здатність MPTCP з маршрутизацією SDN склала 77,8% при 2 підпотоках і 78,5% при 3 підпотоках. У TM2 використання MPTCP з маршрутизацією SDN призвело до 63,8% з 2 субпотоками та 67,4% з 3 субпотоками. Покращення використання допоміжної маршрутизації SDN порівняно з ECMP коливалися від 6,4% до 13,4%.

На рисунках 3.11 і 3.12 показана CDF пропускної здатності великих потоків в TM1 і TM2 відповідно. В обох матрицях трафіку допоміжна маршрутизація SDN з 2 підпотоками призвела до більш високої середньої пропускної здатності, ніж ECMP з 3 підпотоками. Цей результат свідчить про важливість адаптивної маршрутизації, яку демонструє наша архітектура SDN. Він також показує, як хеш-колізія ECMP може вплинути на потенційні покращення використання MPTCP. На рисунках 3.18 і 3.19 показаний CDF часу завершення великих потоків в TM1 і TM2. Більш висока середня пропускна здатність призводить до скорочення часу завершення потоку.

Одне цікаве спостереження в TM1 полягає в тому, що поліпшення пропускної здатності становило менше 1% при використанні 3 субпотоків в порівнянні з 2 субпотоками при маршрутизації SDN. Однак покращення становило близько 4% у випадку TM2, де ми маємо більшу кількість великих потоків. Це приводить нас до висновку, що коли мережа менш перевантажена, ймовірно, кращим компромісом є використання меншої кількості субпотоків.

Є багато факторів, які слід враховувати при виборі кількості допоміжних підводів. Ці фактори включають топологію мережі та кількість шляхів, які можуть бути використані великими потоками. Існує також компроміс між потенційним приростом пропускної здатності та накладними витратами на створення додаткового субпотоку. Наша архітектура SDN демонструє більш високий приріст пропускної здатності при меншій кількості субпотоків у порівнянні з ECMP.

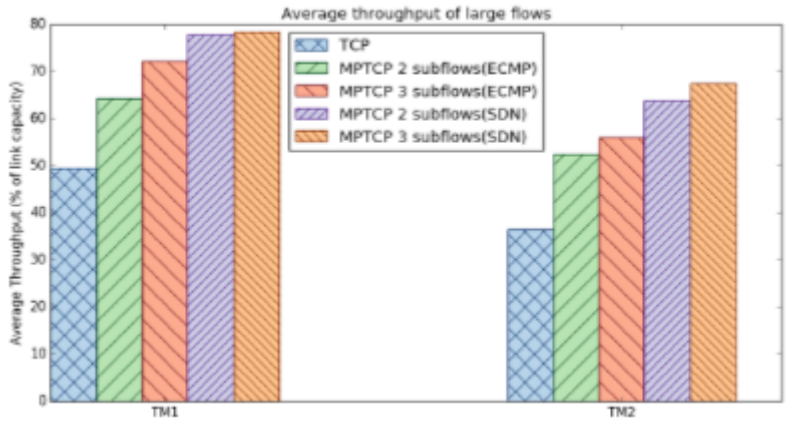


Рисунок 3.15 - Середня пропускна здатність великих потоків

Однією з головних проблем, з якою стикається будь-яка архітектура SDN, є питання масштабованості. Контролер SDN повинен вміти своєчасно обробляти вхідні повідомлення OpenFlow. Ключовою метою розробки нашої архітектури SDN є максимальне зменшення кількості повідомлень OpenFlow. Великі потоки складають більшу частину переданого трафіку в центрах обробки даних, в той час як переважна більшість потоків є короткими. Пересилаючи тільки допоміжні підпотоки великих потоків на контролер SDN, ми зменшуємо кількість повідомлень OpenFlow. Таким чином, навантаження на контролер SDN зводиться до мінімуму.

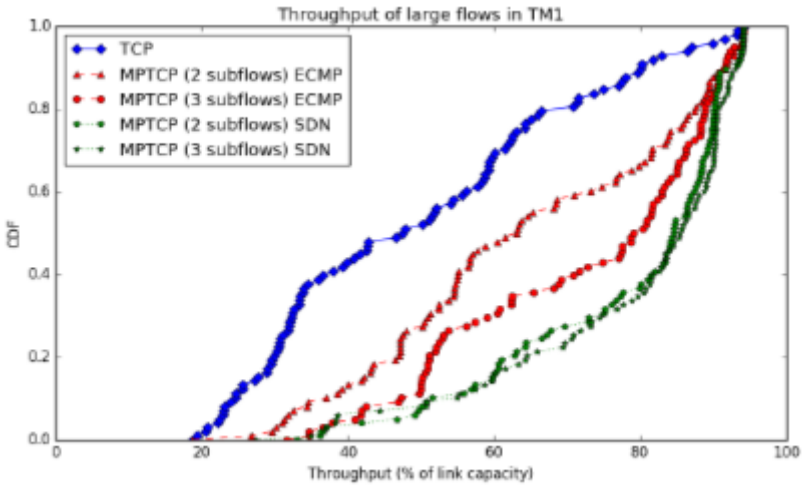


Рисунок 3.16 - Пропускна здатність великих потоків (TM1)

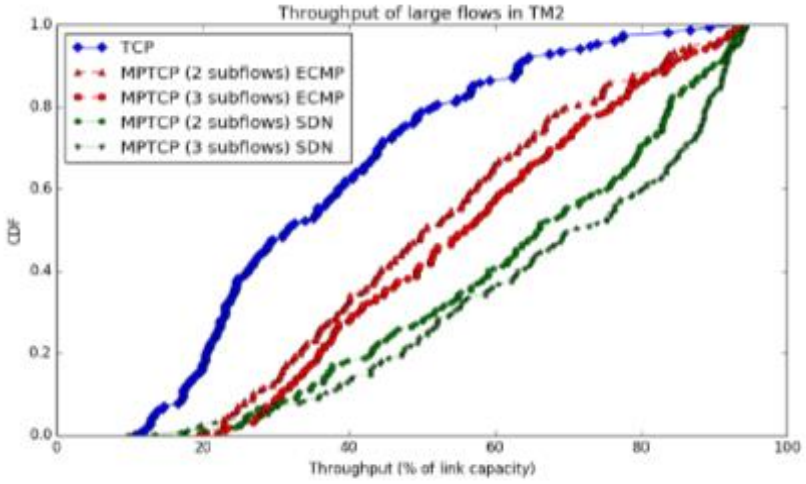


Рисунок 3.17 - Пропускна здатність великих потоків (TM2)

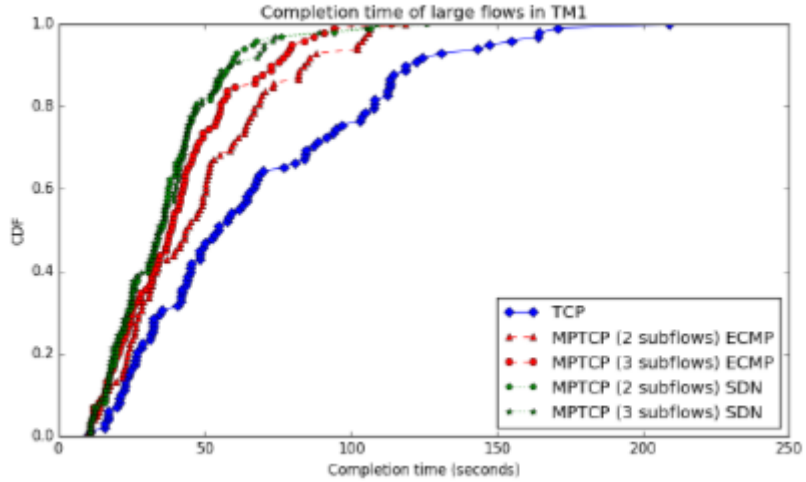


Рисунок 3.18 - Час завершення великих потоків (TM1)

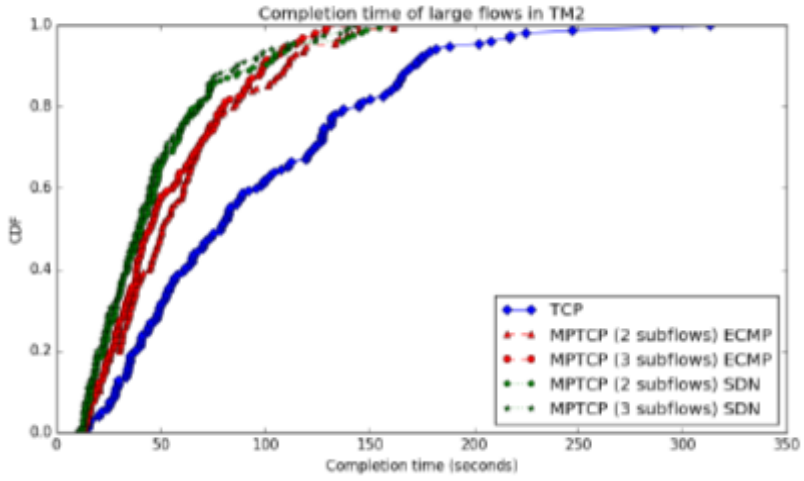


Рисунок 3.19 - Час завершення великих потоків (TM2)

У цій главі було запропонувано архітектуру на основі SDN для використання MPTCP в центрах обробки даних. У цій архітектурі додаткові підпотоки MPTCP створюються на вимогу за допомогою модифікованого ядра Linux. Експерименти проводилися на випробувальному стенді GENI середовище для оцінки використання MPTCP та OpenFlow. Показано, що використання централізованого контролера дозволяє підвищити пропускну здатність великих потоків, що призводить до кращого використання ресурсів мережі.

ВИСНОВКИ

У даній магістерській роботі було досліджено різні механізми, які спрямовані на підвищення якості послуг, що надаються обраним транспортним потокам. Основним напрямком даного дослідження є розробка зручних методів і систем, які мають можливість надавати різним мережевим додаткам гарантовану якість обслуговування. Дане дослідження зосереджене на певних типах потоків мережевого трафіку, які мають різні аспекти вимог QoS. Використовуючи програмно-визначену мережеву архітектуру, було розроблено та оцінено три механізми дослідження:

- Підхід до маршрутизації QoS на основі SDN для покращення надання якості послуг для потоків трафіку, що вимагають пропускної здатності. Маючи централізований контролер, який здійснює безперервний моніторинг мережевих вимірювань, програма маршрутизації QoS розміщує потоки трафіку на шляхах, які мають достатню доступну пропускну здатність. Результати оцінки показують, що такий підхід може значно покращити пропускну здатність потоків, що вимагають пропускної здатності, порівняно з алгоритмом маршрутизації найкоротшим шляхом, який використовується в існуючих мережах.

- Гнучка структура для забезпечення якості обслуговування трафіку, чутливого до затримок. Затримка є важливим вимірюванням QoS для різних мережевих додатків.

На це впливає затримка черги мережевих пристроїв, а також поточний стан пройденого шляху. Представлений в роботі фреймворк на основі SDN дозволяє своєчасно доставляти пакети даних, що належать до трафіку, чутливого до затримок. Цей фреймворк забезпечує QoS чутливий до затримок трафік з акцентом на визначенні пріоритетів для різних класів трафіку та механізмі призначення невикористаної ємності мережі.

- Архітектура на основі SDN для максимізації пропускної здатності великих потоків за допомогою Multipath TCP. Інженерія трафіку в сучасних мережах центрів обробки даних повинна використовувати наявні багатошляхові

топології. Використовуючи багатопроменеві TCP і SDN, було розроблено рішення, яке максимізує коефіцієнт приросту пропускної здатності великих потоків. Представлений в роботі підхід дозволяє додаткам досягати більш високої пропускної здатності для великих потоків шляхом динамічного створення декількох підпотоків MPTCP. Ці підпотоки MPTCP маршрутизуються централізованим контролером SDN через найменш перевантажені шляхи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Hoebeke R. MPLS: Adding Value to Networking. Alcatel Telecommunication Review / R. Hoebeke, M. Aissaoui, T. Nguyen – 3 Quarter 2020.

2. Xiao X. Traffic Engineering with MPLS in the Internet / X. Xiao, A. Hannan, B. Bailey, Ni L. – IEEE Network, March/April 2017.

3. Donner A. An MPLS Networking Concept for Satellite Constellations. ITC18 / Donner A., Beriolo M., Charzinski J., Werner M., Lehnert R. and Tran-Gia P. (Editors), Elsevier Science B.V., 2018

4. Zhou B. Formulation of the Traffic Engineering Problems in MPLS Based IP Networks / B. Zhou, J. Hu and M. K. Girish – Proceedings ISCC 2017. Fifth IEEE Symposium on Computers and Communications(ISCC), Antibes, France, 2017, pp. 214.

5. Cerav-Erbas S.K. Traffic engineering in MPLS networks with multiple objectives: modeling and optimization, Industrial and Systems Engineering, Manisa, Turkei. – 2014.

6. 3GPP TR 25.814 Physical layer aspects for evolved Universal Terrestrial Radio Access (UTRA), Release 7), V7.1.0, 2006. pp.64

7. Hyytia E and Virtamo J (2017). Random waypoint mobility model in cellular networks. Wireless Networks, 13(2): 177-188

8. Huang M, Feng S, and Chen J (2014). A practical approach for load balancing in LTE networks. Journal of Communications, 9(6): 490-497.

9. Li J and Sampalli S (2021). Cell mobility based admission control for wireless networks with link adaptation. In The IEEE International Conference on Communications, IEEE, Glasgow, UK: 5862-5867.

10. Li WY, Zhang X, Jia SC, Gu XY, Zhang L, Duan XY, and Lin JR (2022). A novel dynamic adjusting algorithm for load balancing and handover co-optimization in LTE SON. Journal of Computer Science and Technology, 28(3): 437-444.

11. Chahed T, Altman E, and Elayoubi SE (2008). Joint uplink and downlink

admission control to both streaming and elastic flows in CDMA/HSDPA systems. Performance Evaluation, 65(11-12): 869-882

12. Lobinger A, Stafanski S, Jansen T, and Balan I (2018). Load balancing in downlink LTE self-optimizing network. In the IEEE 71st Vehicular Technology Conference, IEEE, Taipei, Taiwan.

13. Hayat MS, Kazmi SIA, Hasan R, and Bhatti AH (2016). An architecture of future wireless network for smart cities by improving 4G LTE wireless network. In the 3rd MEC International Conference on Big Data and Smart City, IEEE, Muscat, Oman: 1-5.

14. Grondalen O and Osterbo O (2017). Benefits of self-organizing networks (SON) for mobile operators. Journal of Computer Networks and Communications, 2017: 1-16.

15. Yeo SH and Alwi S (2022). Evaluation des performances des techniques d'accès ofdma et sc-fdma dans la technologie lte. Ph.D. Dissertation, University Abou Bekr Belkaid, Tlemcen, Algeria.

16. Liu Q, Zhou S, and Giannakis GB (2015). Queuing with adaptive modulation and coding over wireless links: cross-layer analysis and design. IEEE Transactions on Wireless Communications, 4(3): 1142-1153

17. M. Huynh, and P. Mohapatra, ―Metropolitan Ethernet Network: A move from LAN to MAN‖, Computer Networks, vol. 51, no. 17, pp 4867- 4894, 2018.

18. P., Sager, Does circuit emulation in metro gigabit Ethernets require service priority?, Thesis, ETH Zurich,2015.

19. 23. Алгоритм зважених черг (WQ) [Електронний ресурс] – Режим доступу до ресурсу: https://opengl.org.ru/seti-peredachi-paketnykhdannykh/vzveshennoe-obsluzhivanie.html.

20. Типи QoS та критерії оцінки [Електронний ресурс] – Режим доступу до ресурсу: https://romi.center/ru/learning/glossary/quality-ofservice/ .

21. Алгоритми управління чергами-умови виникнення черг [Електронний ресурс] – Режим доступу до ресурсу: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwebonto.ru%2Fadsl

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

МАГІСТЕРСЬКА РОБОТА

«ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ПРОПУСКНОЇ СПРОМОЖНОСТІ В МЕРЕЖАХ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ QUALITY OF SERVICE (QOS)»

виконав студент: **Гуменюк А.В.**
керівник: **Черевик В.М.,** к.т.н., доцент

Мета магістерської роботи:

підвищення ефективності використання мережевих ресурсів із забезпеченням заданих параметрів якості обслуговування за допомогою оптимізації пропускної спроможності в мережах.

Об’єкт дослідження:

методи оптимізації пропускної спроможності в мережах з використанням QoS.

Предмет дослідження:

програмно-конфігуровані мережі.

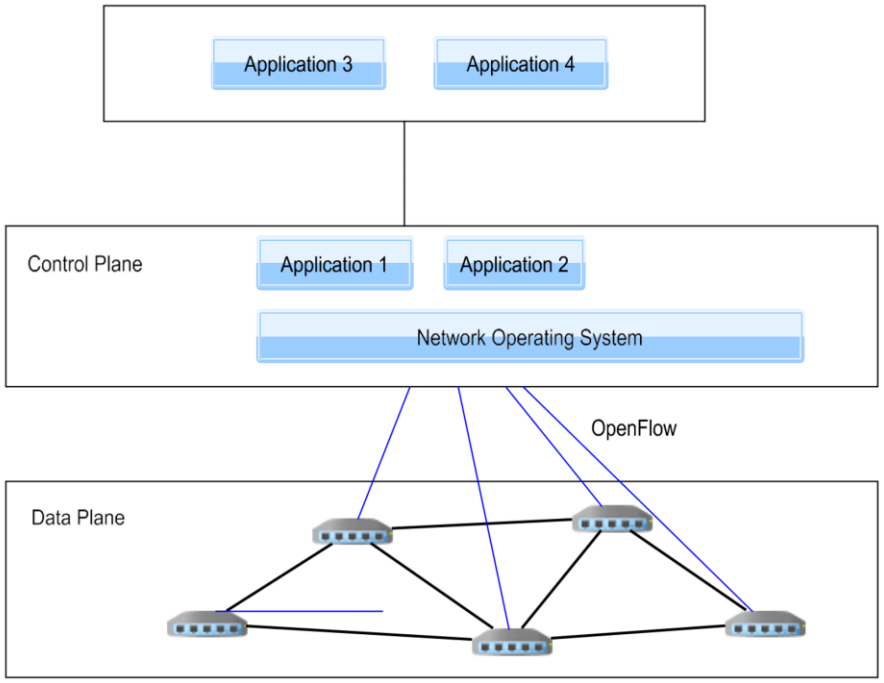
Актуальність:

Актуальність дослідження методів оптимізації пропускної спроможності в мережах з використанням технології Quality of Service (QoS) обумовлена наступними факторами:

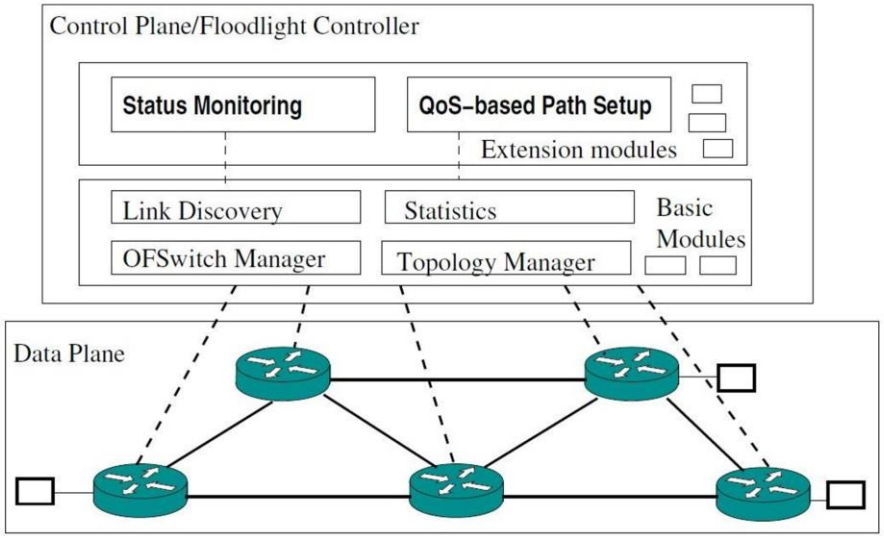
1. Зростання обсягу мережевого трафіку: з кожним роком кількість мережевих додатків і послуг збільшується, що призводить до збільшення обсягу передаваної інформації. Це ставить перед провайдерами і адміністраторами мереж вимогу забезпечити достатню пропускну спроможність для задоволення потреб користувачів.
2. Різноманітність мережевих додатків: різні типи додатків мають різні вимоги до якості обслуговування. Наприклад, для відеоконференцій необхідна низька затримка і мінімальна втрата пакетів, тоді як для завантаження файлів може бути важлива висока пропускна спроможність. Забезпечення гарантованої якості обслуговування для різних типів додатків є важливим завданням.
3. Використання програмно-визначених мереж (SDN): SDN технологія надає нові можливості для управління мережею і надання QoS. Вона дозволяє централізовано керувати мережевими ресурсами і призначати пріоритети різним мережевим потокам. Дослідження оптимальних методів використання SDN для забезпечення QoS є актуальним завданням.
4. Зростання вимог до якості обслуговування: користувачі стають все вимогливішими і очікують високої якості обслуговування в мережі. Забезпечення гарантованої якості обслуговування стає важливим фактором для задоволення потреб користувачів і збереження конкурентоспроможності провайдерів мережевих послуг.

Отже, дослідження методів оптимізації пропускної спроможності в мережах з використанням QoS є актуальним і важливим для покращення якості обслуговування мережевих додатків і ефективного використання мережевих ресурсів.

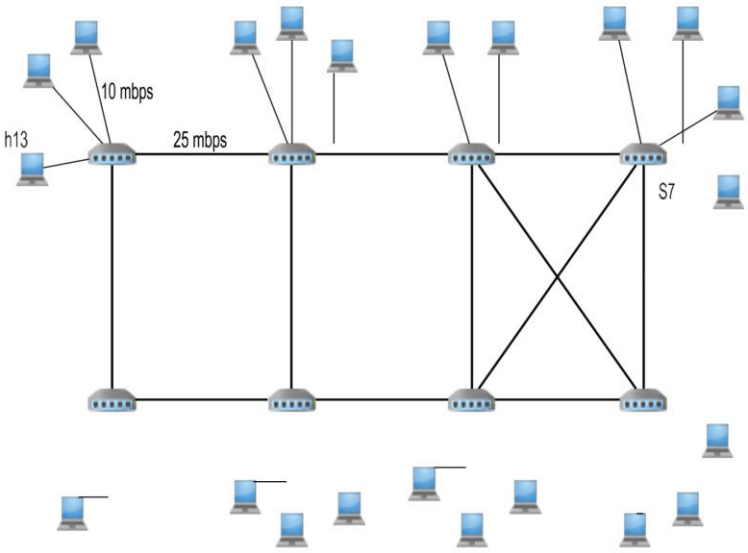
Спрощена архітектура SDN



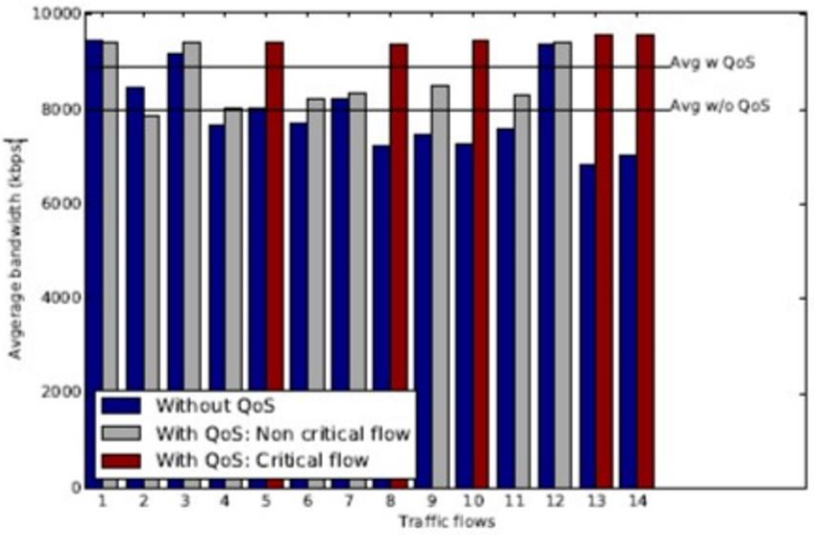
Архітектура SDN для забезпечення підтримки QoS



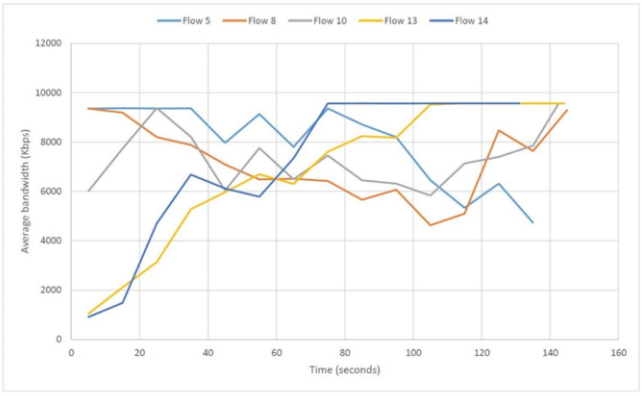
Топологія першого експерименту в Mininet



Пропускна здатність з п'ятьма критичними потоками

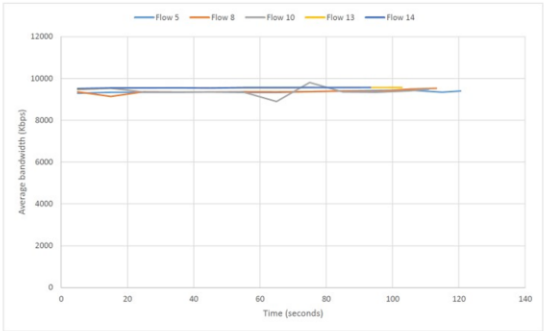


Пропускна здатність для кожного критичного потоку

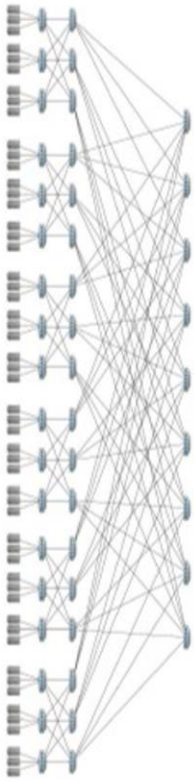


Пропускна здатність для критичних потоків без модуля маршрутизації QoS

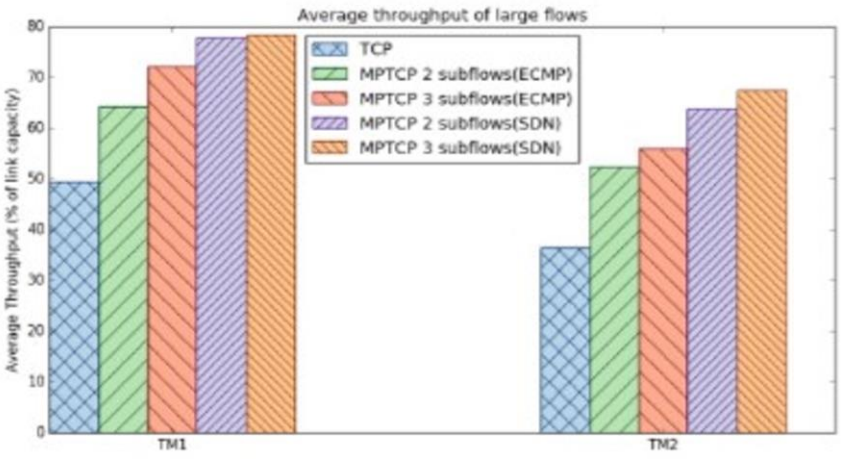
Пропускна здатність для критичних потоків з модулем маршрутизації QoS



Топологія мережі



Середня пропускна здатність великих потоків



Висновки

У даній магістерській роботі було досліджено різні механізми, які спрямовані на підвищення якості послуг, що надаються обраним транспортним потокам. Основним напрямком даного дослідження є розробка зручних методів і систем, які мають можливість надавати різним мережевим додаткам гарантовану якість обслуговування. Дане дослідження зосереджене на певних типах потоків мережевого трафіку, які мають різні аспекти вимог QoS. Використовуючи програмно-визначену мережеву архітектуру, було розроблено та оцінено три механізми дослідження:

- Підхід до маршрутизації QoS на основі SDN для покращення надання якості послуг для потоків трафіку, що вимагають пропускної здатності. Маючи централізований контролер, який здійснює безперервний моніторинг мережевих вимірювань, програма маршрутизації QoS розміщує потоки трафіку на шляхах, які мають достатню доступну пропускну здатність. Результати оцінки показують, що такий підхід може значно покращити пропускну здатність потоків, що вимагають пропускної здатності, порівняно з алгоритмом маршрутизації найкоротшим шляхом, який використовується в існуючих мережах.

- Гнучка структура для забезпечення якості обслуговування трафіку, чутливого до затримок. Затримка є важливим вимірюванням QoS для різних мережевих додатків.

На це впливає затримка черги мережевих пристроїв, а також поточний стан пройденого шляху. Представлений в роботі фреймворк на основі SDN дозволяє своєчасно доставляти пакети даних, що належать до трафіку, чутливого до затримок. Цей фреймворк забезпечує QoS чутливий до затримок трафік з акцентом на визначенні пріоритетів для різних класів трафіку та механізмі призначення невикористаної ємності мережі.

- Архітектура на основі SDN для максимізації пропускної здатності великих потоків за допомогою Multipath TCP. Інженерія трафіку в сучасних мережах центрів обробки даних повинна використовувати наявні багатошляхові топології. Використовуючи багатопроменеві TCP і SDN, було розроблено рішення, яке максимізує коефіцієнт приросту пропускної здатності великих потоків. Представлений в роботі підхід дозволяє додаткам досягати більш високої пропускної здатності для великих потоків шляхом динамічного створення декількох підпотоків MPTCP. Ці підпотоки MPTCP маршрутизуються централізованим контролером SDN через найменш перевантажені шляхи.