ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Підвищення швидкості та надійності мережевих послуг за допомогою динамічного керування програмно-визначених мереж»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

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

_____ *Дмитро СЛЕСАРЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КІД-42
Дмитро СЛЕСАРЕНКО
Ім'я, ПРІЗВИЩЕ
Керівник: Артем АНТОНЕНКО
к.т.н.,, доцент Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
_____Ім'я, ПРІЗВИЩЕ
«_____» _____ 2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Слесаренко Дмитру Андрійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Підвищення швидкості та надійності мережевих послуг за допомогою динамічного керування програмно-визначених мереж
керівник кваліфікаційної роботи к.т.н., доцент Артем АНТОНЕНКО
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «__ _____» _____2024р. № _____
2. Строк подання кваліфікаційної роботи «_____» _____ 2024р.
3. Вихідні дані до кваліфікаційної роботи:
   1. Методи динамічного керування мережевим трафіком
   2. Інтернет ресурси стосовно алгоритмів опитимізації шляхів передачі даних.
   3. Науково-технічна література

Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Програмно-визначені мережі
2. Covisor: динамічна композиція додатків
3. Реалізація прототипу Covisor

4. Перелік ілюстративного матеріалу: *презентація*
5. Дата видачі завдання «_____» _____ 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	26.02-22.03.2024	Виконано
2	Обробка матеріалу	23.03-05.04.2024	Виконано
3	Програмно-визначені мережі	06.04-10.04.2024	Виконано
4	Covisor: динамічна композиція додатків	11.04-30.04.2024	Виконано
5	Реалізація прототипу Covisor	01.05-03.05.2024	Виконано
6	Висновки за результатами аналізу	04.05-08.05.2024	Виконано
7	Розробка презентації	09.05-11.05.2024	Виконано
8	Передзахист	14.05-17.05.2024	Виконано
9	Здача в деканат	27.05-3.06.2024	Виконано

Здобувач(ка) вищої освіти _____ Дмитро СЛЕСАРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Артем АНТОНЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 55 стор., 14 рис., 2 табл., 17 джерел.

Мета роботи – Дослідження можливостей підвищення швидкості та надійності мережевих послуг за допомогою використання динамічного керування програмно-визначених мереж.

Об'єкт дослідження – процес підвищення швидкості та надійності мережевих послуг.

Предмет дослідження – програмно-визначені мережі.

Короткий зміст роботи: В роботі представлено CoVisor - композиційний гіпервізор, який дозволяє адміністраторам об'єднувати декілька контролерів для спільної обробки мережевого трафіку. CoVisor використовує комбінацію нових алгоритмів і структур даних для ефективної компіляції політик в інкрементному режимі. Визначено архітектуру нового типу композиційного гіпервізора, який дозволяє додаткам, написаним різними мовами і на різних контролерах, спільно обробляти пакети; представлено новий алгоритм, який дозволяє компілювати паралельний, послідовний та перевизначальний оператори; використано спеціальні структури даних, які використовують обмеження контролю доступу, що часто є джерелом накладних витрат, для подальшого скорочення часу компіляції.

КЛЮЧОВІ СЛОВА: ПРОГРАМНО-ВИЗНАЧЕНІ МЕРЕЖІ, ДИНАМІЧНЕ КЕРУВАННЯ ТРАФІКОМ, ШВИДКІСТЬ ТА НАДІЙСНІСТЬ МЕРЕЖЕВИХ ПОСЛУГ, ОПТИМІЗАЦІЯ МЕРЕЖЕВОГО ТРАФІКУ

ABSTRACT

The text part of the qualification work for the bachelor's degree: 55 pages, 14 figures, 2 tabl., 17 sources.

The purpose of the work is Exploring the possibilities of increasing the speed and reliability of network services by using dynamic control of software-defined networks.

The object of research is the process of increasing the speed and reliability of network services.

The subject of research is software-defined networks.

Summary of the work: This paper presents CoVisor, a composable hypervisor that allows administrators to combine multiple controllers to jointly process network traffic. CoVisor uses a combination of new algorithms and data structures to efficiently compile policies in an incremental manner. It defines the architecture of a new type of composite hypervisor that allows applications written in different languages and on different controllers to process packets together; presents a new algorithm that allows compiling parallel, sequential, and override statements; and uses special data structures that exploit access control constraints, which are often a source of overhead, to further reduce compilation time.

KEYWORDS: SOFTWARE-DEFINED NETWORKS, DYNAMIC TRAFFIC MANAGEMENT, SPEED AND RELIABILITY OF NETWORK SERVICES, NETWORK TRAFFIC OPTIMIZATION

ЗМІСТ

ВСТУП

Комп'ютерні мережі відіграють важливу роль у сучасному суспільстві. Сьогодні багато інтернет-сервісів, таких як пошукові системи, соціальні мережі та електронна комерція, розміщені в центрах обробки даних, де сотні тисяч комп'ютерів з'єднані між собою масштабними мережами дата-центрів. Ці дата-центри з'єднані між собою глобальними мережами, які охоплюють всю планету. Кінцеві користувачі використовують свої персональні комп'ютери, мобільні телефони та планшети для доступу до цих інтернет-послуг через мережі Ethernet, WiFi та стільникові мережі. Управління цими мережами для надання швидких, надійних і безпечних мережевих послуг є центральною проблемою дослідження комп'ютерних мереж.

Управління мережею має вирішальне значення для надання швидких, надійних і безпечних мережевих послуг. Програмно-визначені мережі (SDN) - це нова мережева архітектура, яка спрощує управління мережею шляхом інтеграції управління мережею в централізовану платформу управління.

Мережеві оператори запускають різні додатки на платформі управління для виконання різних завдань управління, таких як маршрутизація, моніторинг, балансування навантаження і брандмауер. Ці додатки мають складну взаємодію один з одним, що ускладнює розгортання та аналіз їхньої поведінки. Часті мережеві події, такі як зміщення трафіку, кібератаки та збої в роботі пристроїв, ще більше загострюють проблему. Кожна програма повинна переналаштовувати мережу, щоб реагувати на ці події. Складно правильно і ефективно об'єднати зміни конфігурації від декількох додатків, поширити ці зміни на розподілену колекцію мережевих пристроїв і скоординувати зміни між мережевими пристроями на різних рівнях.

Актуальність теми обумовлена постійним зростанням обсягу трафіку в мережах, яке вимагає вдосконалення методів керування та оптимізації ресурсів для забезпечення високої швидкості передачі даних і стабільності роботи мережі. Використання програмно-визначених мереж та динамічного керування може

значно поліпшити якість обслуговування користувачів та забезпечити ефективніше використання інфраструктури мережі.

У цій кваліфікаційній роботі було представлено нову архітектуру управління, яка може ефективно обробляти мережеві події для декількох додатків і на мережевому та оптичному рівнях. Було визначено та досліджено CoVisor: Мережевий гіпервізор, який може об'єднувати декілька додатків та ефективно об'єднувати зміни конфігурації цих додатків у разі виникнення мережевих подій. Щоб захистити мережу від шкідливих і помилкових додатків, CoVisor також забезпечує віртуалізацію топології і тонкий контроль доступу, щоб обмежити, що може бачити і робити кожна програма.

1 ПРОГРАМНО-ВИЗНАЧЕНІ МЕРЕЖІ

1.1 Управління мережею

Комп'ютерні мережі відіграють важливу роль у сучасному суспільстві. Сьогодні багато інтернет-сервісів, таких як пошукові системи, соціальні мережі та електронна комерція, розміщені в центрах обробки даних, де сотні тисяч комп'ютерів з'єднані між собою масштабними мережами дата-центрів. Ці дата-центри з'єднані між собою глобальними мережами, які охоплюють всю планету. Кінцеві користувачі використовують свої персональні комп'ютери, мобільні телефони та планшети для доступу до цих інтернет-послуг через мережі Ethernet, WiFi та стільникові мережі. Управління цими мережами для надання швидких, надійних і безпечних мережевих послуг є центральною проблемою дослідження комп'ютерних мереж.

Управління мережею включає в себе багато різних завдань. Оператори мережі налаштовують мережеві робочі пристрої, наприклад, комутатори та маршрутизатори, для реалізації цих завдань. Обробку пакетів у мережевих пристроях можна змоделювати як обробку за принципом відповідності, коли мережеві пристрої порівнюють заголовки пакетів за певними ознаками (наприклад, відповідність IP-адреси призначення IP-префіксу) і виконують певні дії (наприклад, відкидають пакети або пересилають пакети на вихідний порт) над пакетами, що відповідають цим ознакам. Поведінку переадресації комутатора називається політикою комутатора. Аналогічно, поведінка переадресації у всій мережі політикою, яка будується на основі політик усіх комутаторів мережі. Політики змінюються з часом, тому що операторам потрібно переналаштовувати мережеві пристрої в залежності від різних мережевих подій, таких як зміщення трафіку, кібератаки, збої в роботі пристроїв, мобільність хостів тощо.

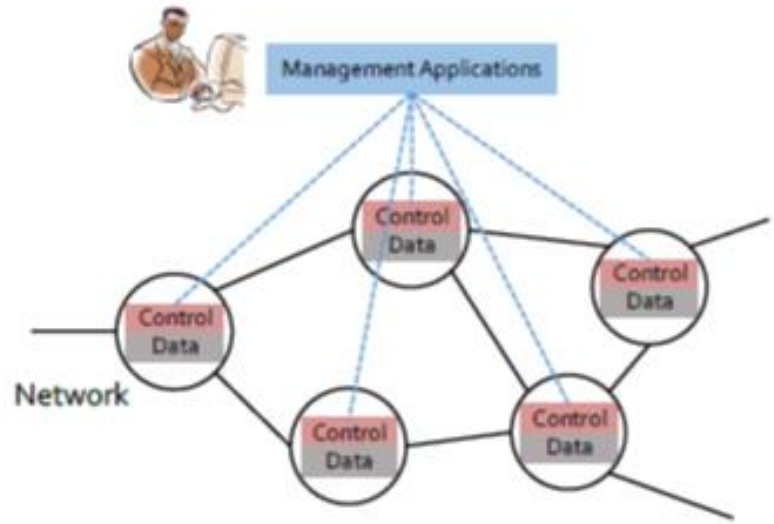


Рисунок 1.1 – Сучасне управління мережею

1.2 Проблеми з сучасним управлінням мережею

Управління мережею можна розділити на дві площини: площину управління та площину даних. Площина керування приймає рішення про пересилання пакетів; площина даних пересилає пакети на високій швидкості на основі конфігурації з площини керування. Правильна і ефективна реалізація завдань управління в сучасній мережі є складним завданням. Оператори витрачають величезні зусилля і час на конфігурацію мережевих пристроїв. Зокрема, сьогоднішнє управління мережею має наступні проблеми.

Поєднання площини керування та площини даних: У сучасній мережі площина керування поєднана з площиною даних, як показано на рисунку 1.1. Площина керування на кожному пристрої обмінюється інформацією між собою, вирішує, як пакети повинні оброблятися на пристрої, і конфігурує площину даних. Оскільки рівень керування розподілений по пристроях, він не має глобального бачення мережі і не може приймати правильні рішення для всієї мережі.

Закритий, власний інтерфейс: Інтерфейс між площиною керування та площиною даних є закритим і пропрієтарним. Постачальники пристроїв продають монолітні коробки, які містять як площину управління, так і площину даних в одній коробці. Оператори не можуть змінити лише площину керування або площину даних, не змінюючи іншу. Закритий інтерфейс також перешкоджає інноваціям, оскільки оператори та треті сторони не можуть легко розробляти нові функціональні можливості без обмежень з боку постачальників пристроїв.

Різнорідні пристрої та конфігурації для кожного пристрою: У мережі є різні пристрої, які виконують різні завдання. Наприклад, комутатори пересилають пакети на основі MAC-адреси, маршрутизатори пересилають пакети на основі IP-адреси, брандмауери фільтрують пакети на основі п'яти кортежів (тобто IP-адреси джерела і призначення, номери портів джерела і призначення, номер протоколу), а балансувальники навантаження розподіляють пакети на основі IP-адрес джерела і призначення. Хоча обробку пакетів на цих пристроях можна змоделювати як загальну обробку в стилі match-action, виробники реалізують для них різні площини даних і площини керування. Кожен тип пристрою має свій власний інтерфейс конфігурації, і цей інтерфейс варіюється в залежності від постачальника. Через це операторам доводиться виконувати конфігурації для кожного пристрою і ретельно планувати конфігурації для різних пристроїв, щоб правильно впроваджувати політики для всієї мережі.

Для виконання завдання маршрутизації оператори повинні налаштувати комутатори для маршрутизації на рівні 2 і налаштувати маршрутизатори для маршрутизації на рівні 3. Комутатори та маршрутизатори дозволяють операторам використовувати лише певні протоколи, наприклад, протокол Spanning Tree Protocol (STP) для маршрутизації на рівні 2 та протокол Open Shortest Path First (OSPF) для маршрутизації на рівні 3. Оператори не можуть гнучко налаштувати площину керування для нових протоколів маршрутизації. Аналогічно, оператори повинні використовувати спеціалізовані пристрої для завдання балансування навантаження і завдання брандмауера. Для збору статистики трафіку для завдання моніторингу оператори повинні мати справу з різними інтерфейсами, які надаються

пристроями. Всі конфігурації виконуються для кожного пристрою окремо. Для операторів є головним болем міркувати про взаємодію між різними пристроями в мережі.

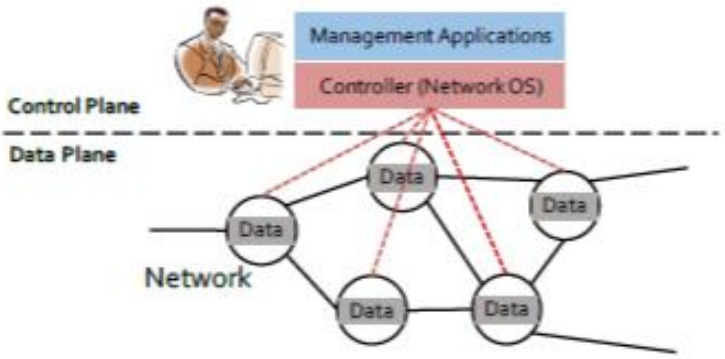


Рисунок 1.2 - Управління мережею за допомогою програмно-визначених мереж (SDN)

1.3 Розвиток програмно-визначених мереж

Програмно-визначені мережі (SDN) з'явилися в останні роки, щоб докорінно змінити те, як ми проектуємо, будуємо та керуємо мережами. Вона має наступні відмінні риси від сучасної мережевої архітектури.

Відокремлення площини керування від площини даних: SDN відокремлює площину керування від площини даних, як показано на рисунку 1.2. Площина управління - це логічно централізований контролер. Він збирає інформацію з площини даних і надає оператору глобальне уявлення. Завдання управління реалізовані у вигляді додатків, що виконуються поверх контролера. Ці додатки приймають рішення про обробку пакетів на основі глобальної картини і передають рішення в площину даних через контролер.

Відкритий, стандартний інтерфейс: Інтерфейс між площиною керування та площиною даних є відкритим та стандартним (наприклад, OpenFlow). Постачальники пристроїв продають лише пристрої площини даних, не поєднуючи

їх з площиною керування. Пристрої можуть бути апаратними комутаторами, виготовленими за допомогою ASIC або FPGA, або програмними комутаторами, що працюють на серверах. Інженери-програмісти можуть легко розробляти контролери та додатки з різними функціональними можливостями. Оператори можуть комбінувати пристрої, контролери та додатки, які найкраще відповідають їхнім потребам.

Загальна модель обробки пакетів та уніфікована конфігурація: SDN моделює мережеві пристрої як пристрої загальної обробки пакетів, використовуючи таблиці відповідності, незалежно від того, чи працює пристрій як комутатор другого рівня, маршрутизатор третього рівня, балансувальник навантаження або брандмауер.

Таблиця 1.1 - Приклад блок-схеми для політики маршрутизації

Priority	Match	Action
2	$dstip = 1.0.0.0/24$	$fwd(3)$
1	$dstip = 1.0.0.0/16$	$fwd(2)$
0	$*$	$drop$

Таблиця відповідності містить список правил. Кожне правило має кілька компонентів. Найважливішими компонентами є пріоритет, відповідність і дія. Компонент збігу визначає структуру заголовків пакетів, компонент дії визначає обробку пакетів, а пріоритет визначає порядок, в якому пакет відповідає декільком правилам. Наприклад, в Таблиці 1.1 показана таблиця відповідності дій для політики маршрутизації, яка пересилає пакети на основі IP-адреси призначення. SDN надає уніфікований інтерфейс для площини управління для налаштування площини даних для реалізації різних завдань управління.

SDN спрощує проектування і розгортання завдань управління мережею.

Для виконання завдань, описаних у 1.1, операторам потрібно лише встановити програми керування на контролері. Додаток маршрутизації може використовувати власні алгоритми маршрутизації, засновані на глобальному огляді, що надається площиною керування, і може легко приймати рішення про переадресацію пакетів на основі різних полів заголовків. Операторам не потрібно турбуватися про те, чи підтримує площина даних тільки маршрутизацію 2-го рівня

або маршрутизацію 3-го рівня. Брандмауер і балансування навантаження не потребують використання спеціалізованих компонентів. Вони реалізовані так само, як і маршрутизація, і можуть бути розгорнуті на будь-яких комутаторах в мережі. Для моніторингу оператори можуть збирати статистику трафіку з різних точок мережі і отримувати інформацію про трафік по всій мережі.

1.4 Проблеми проектування платформи управління

SDN - це нова архітектура для управління мережею. Дослідники продемонстрували переваги SDN, розробивши різні додатки для управління маршрутизацією, моніторингом, балансуванням навантаження, асинхронізація, брандмауер та енергозбереження. Платформа управління має вирішальне значення для підтримки цих додатків і повної реалізації переваг SDN. У цьому розділі детально розглядаються деякі проблеми, пов'язані з розробкою платформи управління.

1.4.1 Багато додатків для контролерів

Компонування декількох додатків для контролерів: Оператори розгортають багато додатків на платформі керування для управління мережею. Маршрутизація, моніторинг, балансування навантаження і брандмауер - ось кілька прикладів додатків, описаних в 1.1. Кожна програма повинна конфігурувати площину даних за допомогою політики, щоб реалізувати свою мету управління. Платформа керування повинна надавати операторам підтримку для визначення взаємозв'язку між декількома додатками та коректної компіляції політик з декількох додатків в одну політику на основі цієї специфікації. Без підтримки з боку платформи

управління додаткам доводиться самостійно здійснювати координацію з іншими, що не тільки ускладнює логіку роботи додатків, але й лягає важким тягарем на розробників.

Захист від зловмисних і помилкових додатків: Оскільки додатки для управління можуть бути від третьої сторони, оператори не повністю довіряють цим додаткам. Ці програми можуть мати зловмисну поведінку та обробляти пакети не так, як вони повинні. Також часто додатки містять помилки, які можуть спричинити неочікувані результати під час роботи. Тому платформа управління повинна дозволяти операторам встановлювати тонкий контроль доступу до того, що може бачити і робити кожна програма, щоб захистити мережу від зловмисних і помилкових додатків.

Забезпечення гнучкості у виборі мови програмування для розробки: Існуючі контролери, такі як ONOS, OpenDaylight, Ryu, Floodlight та POX, змушують розробників використовувати ту ж мову, що і контролер, для розробки додатків. В ідеалі, платформа управління повинна забезпечувати гнучкість для розробників у виборі їх улюблених мов програмування, не обмежуючи їх конкретною мовою програмування.

1.4.2 Багато мережевих подій

Мережі сповнені динаміки. Приклади мережевих подій включають зміну трафіку, кібератаки, збої пристроїв, оновлення пристроїв, мобільність хостів і перевантаження серверів. Важливо, щоб рівень управління швидко і ефективно адаптувався до цих подій. В іншому випадку ці події можуть призвести до значних втрат для операторів. Наприклад, зміщення трафіку може створити затори в мережі і погіршити якість обслуговування користувачів, а кібератаки можуть призвести до відключення інтернет-сервісів і витоку даних. Щоб відреагувати на ці події, програми обчислюють нові політики і оновлюють площину даних відповідно до

нових політик. Замість того, щоб кожна програма оновлювала свою політику в індивідуальному порядку, платформа управління повинна забезпечити загальне і ефективне рішення. Існує три основні проблеми, пов'язані з оновленням політик.

Складання оновлень з декількох додатків: Кожна програма генерує оновлення для власної політики. Оскільки в мережі розгорнуто кілька додатків, рівень керування повинен об'єднати оновлення з декількох додатків в єдине оновлення для мережі. Простим рішенням є переобчислення мережевої політики з оновлених політик додатків, а потім встановлення нової політики в мережі. Однак таке рішення призводить до великих обчислювальних витрат, оскільки потрібно перерахувати всю політику, в той час як більшість частин політики можуть залишитися незмінними. Крім того, простий перерахунок нової політики без урахування існуючої політики може призвести до непотрібних змін у правилах, які вже є в комутаторі. Нам доведеться оновити багато існуючих правил, а також додати нові правила і видалити застарілі. Щоб вирішити цю проблему, нам потрібні нові алгоритми, які можуть обчислювати нову мережеву політику інкрементно на основі існуючої політики і вносити якомога менше змін в існуючу політику.

Розповсюдження оновлень між кількома комутаторами: Оновлення політики часто впливає на декілька комутаторів у мережі. Оновлення на різних комутаторах залежать одне від одного, тому що зміна політики одного комутатора може вплинути на трафік на інших комутаторах. Якщо оновлення виконується неакуратно, під час оновлення можуть виникнути серйозні проблеми, такі як петлі маршрутизації, чорні діри, порушення політики та перевантаження. Крім того, через нерівномірність завантаження комутаторів та процесорів, операції оновлення на різних комутаторах можуть сильно відрізнятися. Це може призвести до тривалих затримок у виконанні оновлень для всієї мережі. Рівень керування повинен ретельно планувати оновлення, щоб усунути небажані перехідні проблеми і мінімізувати час оновлення.

Координація оновлень між кількома рівнями: Деякі мережеві оператори, наприклад, інтернет-провайдери, контролюють не лише комутатори на мережевому рівні, але й оптичні пристрої на оптичному рівні. Подібно до оновлень

на декількох комутаторах, оновлення на різних рівнях також повинні бути ретельно скоординовані, щоб усунути перехідні проблеми. Наприклад, якщо оператор відключає оптичний канал і встановлює новий на оптичному рівні без попереднього переміщення трафіку на мережевому рівні, то під час оновлення оптичного каналу весь трафік, який використовує цей канал, буде відключений. Щоб уникнути цих проблем, рівень управління повинен ретельно координувати оновлення між різними рівнями.

1.4.3 Багато шарів

Управління мережею включає в себе як управління комутаторами на мережевому рівні, так і управління оптичними пристроями на оптичному рівні. Існуючі платформи управління зосереджені на обробці пакетів на мережевому рівні, де пристрої виконують обробку трафіку в стилі match-action. Однак оптичні пристрої сьогодні широко використовуються в глобальних мережах, і зростає інтерес до інтеграції оптичних технологій в центри обробки даних для зменшення витрат, продуктивності та енергоспоживання. Інтернет-провайдери мають окремі команди для управління оптичними пристроями на оптичному рівні та комутаторами на мережевому рівні. Оптичні пристрої не часто реконфігуруються разом з комутаторами на мережевому рівні для оптимізації передачі даних. Платформа керування не підтримує спільне керування оптичним та мережевим рівнями.

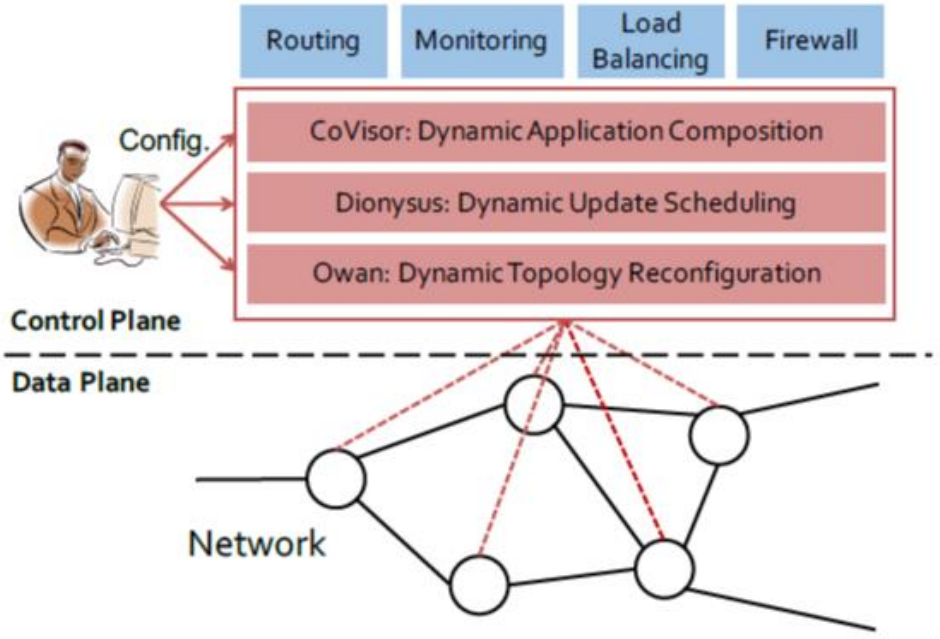


Рисунок 1.3 - Огляд архітектури динамічного управління мережею.

2 COVISOR: ДИНАМІЧНА КОМПОЗИЦІЯ ДОДАТКІВ

Цей розділ присвячений підтримці композиції декількох додатків для управління і компіляції політик з декількох додатків в єдину політику. Щоб повністю реалізувати бачення SDN, оператори повинні мати можливість зібрати колекцію незалежно розроблених "найкращих у своєму класі" додатків, написаних різними мовами програмування і працюючих на різних контролерах. Хоча мережеві гіпервізори здатні розміщувати декілька контролерів в одній мережі, існуючі гіпервізори підтримують лише розщеплення, тобто поділ мережі на окремі частини для окремого управління окремими контролерами. У цьому розділі представлено CoVisor, новий тип мережевого гіпервізора, який дозволяє декільком контролерам спільно керувати одним і тим же спільним трафіком. Таким чином, мережеві адміністратори можуть використовувати CoVisor для створення набору програм - брандмауера, балансувальника навантаження, шлюзу, маршрутизатора, монітора трафіку - і можуть застосовувати ці програми разом або окремо до потрібного трафіку. CoVisor також абстрагується від конкретних топологій, надаючи замість них користувацькі віртуальні топології, і дозволяє операторам визначати контроль доступу, який регулює пакети, що може бачити, модифікувати, контролювати або перенаправляти певна програма.

Основним технічним внеском CoVisor є новий набір ефективних алгоритмів для створення політик контролерів, для компіляції віртуальних мереж у конкретні правила OpenFlow та для ефективної обробки оновлень правил контролерів.

Було побудовано прототип CoVisor і показано, що він працює на кілька порядків швидше, ніж проста реалізація.

Фундаментальний принцип програмно-визначених мереж (SDN) полягає в тому, щоб відокремити логіку управління від апаратного забезпечення конкретного виробника. Таке відокремлення дозволяє операторам розгортати як програмне, так і апаратне забезпечення, що найбільше відповідає їхнім потребам, замість того, щоб йти на компроміси на одному або обох фронтах через відсутність ідеального

рішення. Щоб повністю реалізувати це бачення вільної збірки "найкращих у своєму класі" рішень, оператори повинні мати можливість запускати будь-яку комбінацію додатків для контролерів у своїх мережах. Якщо оптимальний додаток для моніторингу написаний на Python на Ryu, а найкращий додаток для маршрутизації написаний на Java на Floodlight, оператор повинен мати можливість розгорнути обидва додатки в мережі.

Мережевий гіпервізор є природним рішенням цієї проблеми об'єднання розрізнених контролерів. Однак, існуючі гіпервізори обмежують кожен контролер окремим фрагментом мережевого трафіку. Хоча вони корисні в таких сценаріях, як багатокористувацька оренда, де кожен користувач контролює свій власний трафік, вони не дозволяють декільком додаткам спільно обробляти один і той самий трафік. Таким чином, гіпервізор SDN повинен бути здатним на більше, ніж просто нарізка.

Оператор мережі повинен мати можливість збирати декілька контролерів у гнучкий та конфігурований спосіб. Було складено політики площини даних трьома способами: паралельно (дозволяє декільком контролерам діяти незалежно над одними і тими ж пакетами одночасно), послідовно (дозволяє одному контролеру обробляти певний трафік раніше за іншого), і за допомогою надмірного руху (дозволяє одному контролеру вибирати, що робити, або передавати управління іншому контролеру). Однак, на відміну від Frenetic та споріднених систем, даний гіпервізор не залежить від конкретних мов, бібліотек або платформ контролерів, що використовуються для побудови клієнтських додатків. Замість цього гіпервізор перехоплює та обробляє стандартні повідомлення OpenFlow, збираючи та трансформуючи їх відповідно до політики композиції, визначеної оператором. Для ефективної роботи потрібні нові інкрементні алгоритми для обробки оновлень правил.

Визначення абстрактних топологій. Щоб захистити фізичну інфраструктуру, оператор повинен мати можливість обмежити те, що кожен контролер може бачити з фізичної топології. Представлений гіпервізор підтримує це, дозволяючи оператору надавати кожному контролеру власну віртуальну топологію, тим самим

полегшуючи повторне використання (фізичного) коду, незалежного від топології. Наприклад, для контролера брандмауера оператор може абстрагувати мережу як "великий віртуальний комутатор"; брандмауеру не потрібно знати базову топологію, щоб визначити, чи слід переслати пакет або відкинути його. На противагу цьому, контролер маршрутизації потребує точної топології для ефективного виконання свого завдання. Крім того, абстракція топології допомагає оператору реалізувати складну функціональність у модульний спосіб. Деякі комутатори, наприклад, шлюз між островом Ethernet і ядром IP, можуть грати кілька ролей в мережі. Гіпервізор може створити один віртуальний комутатор для кожної ролі, призначити кожному з них додаток-контролер, точно орієнтований на його єдине завдання, і скомпілювати політики, написані для віртуальної мережі, у фізичну мережу.

Захист від неправильної поведінки контролерів. На додаток до обмеження того, що контролер може бачити про фізичну топологію, оператор може також захотіти встановити тонкий контроль над тим, як контролер може обробляти пакети. Такий контроль доступу важливий для захисту від помилок або зловмисних дій сторонніх контролерів. Наприклад, контролеру брандмауера не можна дозволити модифікувати пакети, а MAC-засіб не повинен мати змогу перевіряти заголовки IP або TCP. Гіпервізор забезпечує дотримання цих обмежень, обмежуючи функціональність віртуальних комутаторів, доступних кожному контролеру.

Основним технічним викликом, пов'язаним зі створенням такого повнофункціонального гіпервізора, є ефективність. Гіпервізор повинен містити десятки контролерів, кожен з яких встановлює десятки тисяч правил. Ще більше ускладнює ситуацію те, що ці контролери постійно оновлюють правила, як це диктує логіка їх застосування (наприклад, інженерія трафіку, відновлення після збоїв та виявлення атак. Простий дизайн гіпервізора полягає в тому, щоб перекомпілювати складену політику з нуля для кожного оновлення правил, а потім встановити в таблиці потоків кожного комутатора різницю між існуючою та оновленою політикою. Це надумане рішення є надто дорогим як з точки зору часу

на компіляцію нової політики, так і з точки зору часу на встановлення нових правил на комутаторах.

У цьому розділі представлено CoVisor, гіпервізор, який використовує нові ефективні алгоритми для компіляції та оновлення політик, що надходять від вищих контролерів, кожен з яких має власний погляд на топологію мережі.

Рисунок 2.1 ілюструє архітектуру CoVisor. CoVisor слугує прозорим шаром між контролерами та фізичною мережею. Кожен з п'яти додатків, показаних у верхній частині рисунку 2.1, є незміненою SDN-програмою, що працює на власному контролері; кожен контролер виводить правила OpenFlow для віртуальної топології, показаної під ним, не знаючи про те, що ця віртуальна топологія не існує фізично. CoVisor приймає правила OpenFlow, виведені всіма п'ятьма контролерами, і компілює їх в єдину політику для фізичної мережі за допомогою двофазного процесу.

По-перше, CoVisor використовує новий алгоритм для інкрементальної компіляції додатків у спосіб, визначений оператором. Основна ідея полягає в тому, що пріоритети правил утворюють зручну алгебру для обчислення пріоритетів для нових правил, що позбавляє від необхідності перекомпіляції з нуля для кожного оновлення правил. По-друге, CoVisor перетворює створену політику в правила для фізичної топології. Зокрема, було розробляємо новий алгоритм компіляції для випадку, коли один фізичний комутатор зіставляється з декількома віртуальними комутаторами. На обох етапах CoVisor використовує ефективні структури даних, щоб ще більше зменшити накладні витрати на компіляцію, використовуючи знання про структуру політик, надані обмеженнями контролю доступу. Після компіляції політики CoVisor надсилає необхідні оновлення правил на комутатори.

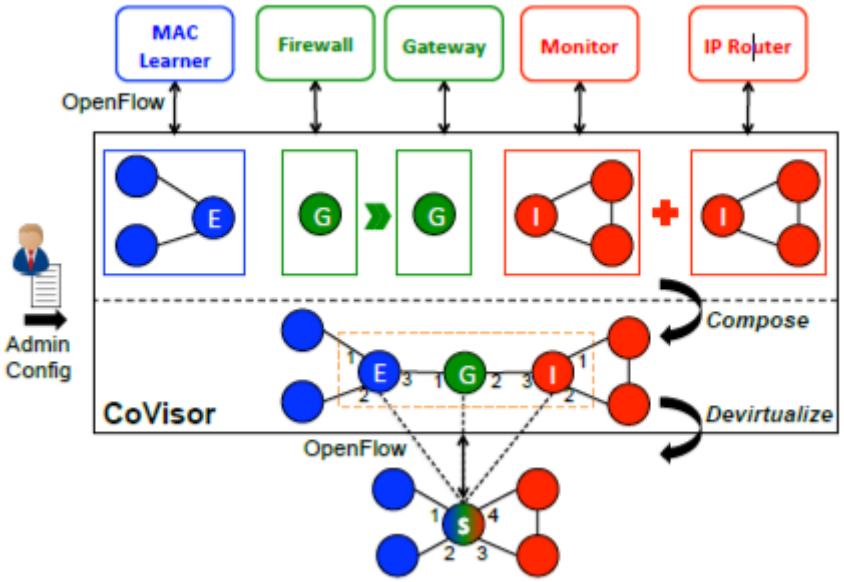


Рисунок 2.1 – Огляд CoVisor

У крайньому лівому кутку на рисунку 2.1 CoVisor приймає вхідні дані конфігурації від оператора. Ці конфігураційні обов'язки мають три складові: (1) визначення того, як мають бути зібрані політики контролерів; (2) створення віртуальної мережі для кожного контролера шляхом визначення компонентів, які мають бути включені, і відображення фізичного і віртуального; і (3) встановлення обмежень контролю доступу для кожного контролера.

Підсумовуючи, можна зробити наступні висновки:

- визначаємо архітектуру нового типу композиційного гіпервізора, який дозволяє додаткам, написаним різними мовами і на різних контролерах, спільно обробляти пакети.

- розробляємо новий алгоритм, який дозволяє компілювати паралельний, послідовний та перевизначальний оператори.

- розробляємо новий, інкрементний алгоритм для компіляції політик, написаних для віртуальних топологій, у правила для фізичних комутаторів.

- використовуємо спеціальні структури даних, які використовують обмеження контролю доступу, що часто є джерелом накладних витрат, для подальшого скорочення часу компіляції.

2.1 Огляд CoVisor

Функції CoVisor поділяються на дві категорії:

- ті, що об'єднують додатки, які працюють на декількох контролерах, для створення єдиної таблиці потоків для кожного фізичного комутатора;

- ті, що обмежують уявлення окремого контролера про топологію і можливості обробки пакетів.

Для реалізації цих функцій CoVisor використовує двофазний процес компіляції. На першому етапі політики окремих контролерів, написані для їх власних віртуальних мереж, збираються в єдину політику для всієї віртуальної мережі. На другому етапі ця складена політика для віртуальної мережі компілюється в політику для фізичної мережі, яка реалізує наміри, виражені віртуальною політикою.

2.1.1 Складання декількох контролерів

CoVisor дозволяє мережевим операторам об'єднати специфікації обробки пакетів декількох контролерів в єдину специфікацію для фізичної мережі, "специфікації обробки пакетів", що виводяться кожним контролером, політиками, а єдину специфікацію - складеною політикою. На практиці, політики учасників визначаються командами OpenFlow, що надсилаються від контролера до CoVisor.

Оператор мережі налаштовує CoVisor для створення контролерів за допомогою простої мови команд. Нехай T - це діапазон політик, визначених у мові команд. Ця мова дозволяє операторам вказати, що деяка дія за замовчуванням (a) повинна бути застосована до набору пакетів, що має застосовуватися певна політика учасника (x), що дві окремі політики мають застосовуватися паралельно (T1 + T2), що дві окремі політики мають застосовуватися послідовно (T1 ≫T2),

або що має застосовуватися одна політика учасника, а якщо вона не відповідає пакету, то за замовчуванням має діяти інша політика (x⊳ T).

Найпростішою складеною політикою є атомарна дія обробки пакетів a. Такі дії включають будь-яку функцію від пакета до набору пакетів, реалізовану у Open-Flow, наприклад, дії відкинути пакет (drop), переслати пакет через певний порт (fwd(3)), або надіслати пакет контролеру (to controller(x)).

Паралельний оператор (+): Паралельна композиція двох політик T_1 + T_2 працює шляхом логічного (хоча не обов'язково фізичного) копіювання пакета, застосування T_1 до однієї копії і T_2 до іншої, і об'єднання результатів. Наприклад, нехай M - політика моніторингу, а Q - політика маршрутизації. Якщо M рахує пакети на основі IP префікса джерела, а Q пересилає пакети на основі IP префікса призначення, то M + Q виконує обидві операції над усіма пакетами.

Послідовний оператор (≫): Послідовний оператор дозволяє двом контролерам обробляти трафік один за одним. Наприклад, нехай L - це політика балансування навантаження, а Q - політика маршрутизації. Зокрема, для пакетів, призначених для широкомовної IP-адреси 3.0.0.0, L переписує IP-адресу призначення в IP-адресу репліки сервера на основі префікса IP-адреси джерела, а Q пересилає пакети на основі префікса IP-адреси призначення. Щоб отримати комбіновану поведінку L і Q - спочатку переписати IP-адресу призначення, а потім переслати переписаний пакет у потрібне місце - мережевий оператор використовує політику L ≫Q.

Оператор перевизначення (⊳): Кожен контролер x надає CoVisor політику учасника, яка визначає, як x хоче, щоб мережа обробляла пакети. Політика x ⊳ T намагається застосувати політику учасника x до будь-якого вхідного пакета t. Якщо в політиці x не вказано, як обробляти t, то за замовчуванням використовується T.

2.1.2 Обмеження для окремих контролерів

На додаток до створення політик учасників, CoVisor дозволяє оператору віртуалізувати базову топологію і обмежити можливості обробки пакетів, доступні кожному контролеру. Це допомагає операторам приховати інформацію про інфраструктуру від сторонніх контролерів, повторно використовувати незалежні від топології алгоритми та забезпечити захист від зловмисного або помилкового програмного забезпечення.

Обмеження на видимість топології

Замість того, щоб розкривати повну інформацію про фізичну топологію для кожного контролера, CoVisor надає кожному власну віртуальну топологію. У таблиці 2.1 показано API для створення власної віртуальної мережі. createVSw створює віртуальний комутатор. Він може бути використаний для створення двох типів фізично-віртуальних відображень наступним чином. (1) багато до одного (багато фізичних комутаторів відображаються до одного віртуального комутатора): викликати функцію один раз зі списком ідентифікаторів фізичних комутаторів; (2) один-до-багатьох (один фізичний комутатор зіставляється з багатьма віртуальними комутаторами): викликати функцію кілька разів з тим самим ідентифікатором фізичного комутатора. createVPort створює віртуальний порт. Щоб зіставити його з фізичним портом, оператор включає відповідний фізичний комутатор і номер порту. createVLink створює віртуальне з'єднання, з'єднуючи два віртуальні порти. connectHost з'єднує хост з віртуальним портом.

Таблиця 2.1 – API для побудови віртуальної мережі. Дужки <> позначають необов'язкові аргументи.

Command	Parameters
createVSw	pSw_1 <pSw_2, ..., pSw_n>
createVPort	vSw <pSw pPort>
createVLink	vSw_1 $vPort_1$ vSw_2 $vPort_2$
connectHost	vSw vPort host

Приклад. Розглянемо приклад відображення фізичної та віртуальної топології, показаний на рисунку 2.1. Фізична топологія представляє мережу підприємства, що складається з острівця Ethernet (показаний синім кольором на рисунку 2.1), з'єднаного маршрутизатором-шлюзом (різнокольоровим і позначеним літерою S) з IP ядром (червоним кольором). Ми абстрагуємося від комутатора-шлюзу S до трьох віртуальних комутаторів: E, G та I. На рисунку 2.2 показано, як оператор використовує API CoVisor для створення віртуального відображення.

```
E = createVSw S              // vswitch E
G = createVSw S              // vswitch G
I = createVSw S              // vswitch I
E1 = createVPort E S 1       // port 1 on E
E2 = createVPort E S 2       // port 2 on E
E3 = createVPort E           // port 3 on E
G1 = createVPort G           // port 1 on G
L1 = createVLink E 3 G 1     // link E – G
...      remaining commands omitted for brevity.
```

Рисунок 2.2 – Конфігурація адміністратора для створення (підмножини)

фізично-віртуального відображення показано на рисунку 2.1.

Ці чотири команди дозволяють оператору створити один рівень віртуальної топології поверх фізичної мережі. Для створення декількох рівнів абстракції топології оператор може запустити один екземпляр CoVisor поверх іншого.

Обмеження на обробку пакетів

CoVisor накладає тонкий контроль доступу на те, як контролер може обробляти пакети, віртуалізуючи функціональність комутатора. Оператор встановлює власні можливості на віртуальних комутаторах кожного контролера, таким чином вибираючи, які функції фізичної мережі будуть доступні для кожного контролера окремо.

Шаблон: Оператор вказує, з якими полями заголовка може співпадати контролер і як може співпадати кожне поле (тобто, точна відповідність, префіксна

відповідність або довільна шаблонна відповідність). Наразі CoV-isor підтримує 12 полів у специфікації OpenFlow 1.0, причому префіксна відповідність доступна лише для IP-адрес джерела та призначення.

Дія: Оператор визначає дії, які контролер може виконати над відповідними пакетами. Наразі CoVisor підтримує дії зі специфікації OpenFlow 1.0, включаючи ward, drop (позначений порожнім списком дій) і modify (оператор визначає, які поля можна змінювати). Оператор також контролює, чи може контролер запитувати пакети та лічильники від комутаторів та надсилати пакети до комутаторів.

Приклад. У прикладі на Рисунку 2.1 оператор може обмежити пошук MAC-адрес тільки за MAC-адресами джерела і призначення та портом, а брандмауер - тільки за п'ятьма кортежами. Також оператор може заборонити обом програмам модифікувати пакети.

2.1.3 Робота з помилками

Контролери, перемикачі та сам CoVisor можуть виходити з ладу під час роботи.

Відмова контролера: Оператор налаштовує CoVisor з політикою за замовчуванням для кожного контролера, яка буде виконуватися в разі відмови контролера. Політика за замовчуванням залежить від програми. Наприклад, логічним значенням за замовчуванням для контролера брандмауера є відкидання (видалення всіх правил, що зупинилися, і встановлення правила, яке відкидає всі пакети), оскільки брандмауер повинен бути відмовостійким. На відміну від цього, політика за замовчуванням для контролера моніторингу може бути id (ідентична, тобто залишити всі правила в комутаторі), оскільки правила моніторингу не є критичними для роботи мережі, а лічильники можуть бути використані повторно, якщо контролер моніторингу відновиться.

Відмова перемикача: Якщо комутатор виходить з ладу, всі його правила видаляються, і CoVisor повідомляє про це відповідні контролери. Більш того, у випадку віртуалізації "багато до одного" CoVisor дозволяє віртуальному комутатору залишатися функціональним, перенаправляючи трафік в обхід фізичного комутатора, що вийшов з ладу (якщо це можливо у фізичній мережі).

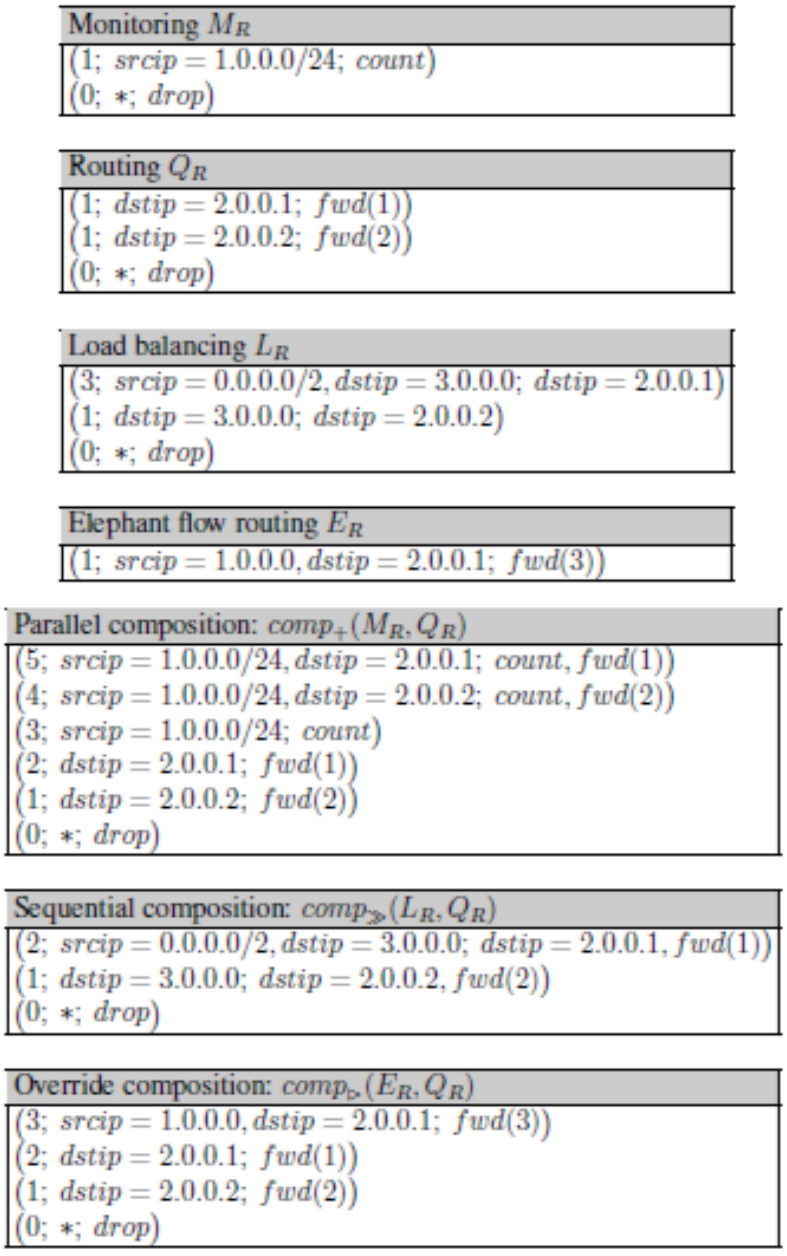


Рисунок 2.3 – Приклад компіляції політики

2.2 Інкрементна компіляція політики

Управління мережею є динамічним процесом. Додатки оновлюють свої політики у відповідь на різні мережеві події, такі як зміна матриці трафіку, збої в роботі комутаторів і каналів зв'язку, а також виявлення атак. Тому CoVisor отримує потоки оновлень політик учасників від контролерів і повинен часто перекомпілювати та оновлювати складену політику. У цьому підрозділі ми спочатку розглянемо компіляцію політик і представимо приблизне рішення, а потім опишемо ефективне рішення, засноване на зручній алгебрі пріоритетів правил.

2.2.1 Загальні відомості про розробку політики

Перший етап компіляції політики полягає в об'єднанні політик учасників в єдину загальну політику. Контролери реалізують політики учасників, надсилаючи правила OpenFlow до CoVisor. Правило *r* - це трійка r = (p; m; a), де *p* - пріоритет, *m* - шаблон відповідності, а *a* - список дій. Маючи правило *r = (p; m; a),* ми використовуємо позначення *r.priority* для позначення *p, r.match* для позначення *m* і *r.action* для позначення *a*. Ми позначаємо множину пакетів, що відповідають *r.match*, як *r.mSet*. Ми припускаємо, що всі реалізації політик включають лише правила OpenFlow 1.0 і що кожен комутатор має єдину таблицю потоків.

Паралельний оператор (+): Для компіляції $T_1 + T_2$ ми спочатку компілюємо T_1 і T_2 у реалізації R_1 і R_2. (На практиці, кожен контролер надсилає свою політику учасника до CoVisor у вже скомпільованому вигляді. Ми явно включаємо цей крок, оскільки він представляє базовий випадок рекурсивного процесу). Потім ми обчислюємо $comp_+ (R_1 , R_2)$ шляхом перебору $(r_{1i} , r_{2j}) \in R_1 \times R_2$, де r_{1i} і r_{2j} беруться з R_1 і R_2 , відповідно, за пріоритетом у порядку спадання. Ми створюємо правило r

у складеної реалізації, якщо перетин r_{1i} *.mSet* і r_{2j} *.mSet* не порожній. *r.match* - це перетин r_{1i} .match і r_{2j} .match, а *r.actions* - це об'єднання r_{1i} *.actions* і r_{2j} *.actions*.

Розглянемо приклад $comp_{+}\ (M_R\ ,\ Q_R)$ на рисунку 2.3. Нехай $M_R = m_1\ ,\ .\ .\ .\ ,\ m_n$ та $Q_R = q_1\ ,\ .\ .\ .\ ,\ q_k$. Почнемо з розгляду m_1 та q_1 . Окільки $m_1\ .mSet \cap q_1\ .mSet \neq \emptyset$, ми створюємо перше правило r_1 в $comp+\ (M_R\ ,\ Q_R)$ з шаблоном відповідності *{srcip = 1.0.0.0/24, dstip = 2.0.0.1}* і список дій *{count, fwd(1)}*. Складання всіх $(m_i\ ,\ q_j)$ дають складену реалізацію політики $comp_{+}\ (M_R\ ,\ Q_R)$ композиції політик $M + Q$.

Послідовний оператор (≫): Для компіляції $T_1 \gg T_2$ ми знову починаємо з генерації реалізацій. Потім ми обчислюємо $comp_{\gg}\ (R_1\ ,\ R_2)$. Як і у випадку з $comp_{+}\ (R_1\ ,\ R_2)$, ми перебираємо $(r_{1i}\ ,\ r_{2j}) \in R_1 \times R_2$, де r_{1i} та r_{2j} беруться з R_1 та R_2 відповідно, за пріоритетом у порядку зменшення. Однак тепер ми створюємо правило r у скомпонованій політиці, якщо перетин r_{2j}*.mSet* і множини пакетів, отриманих шляхом застосування r_{1i}*.action* до всіх пакетів у r_{1i}*.mSet*, не є порожнім. Розглянемо приклад $comp_{\gg}(L_R,\ Q_R)$ на рис. 2.3. Знову ж таки, ми починаємо ітерацію над парами $(l_i\ ,\ q_j\ ) \in L_R \times Q_R$ з розгляду l_1 та q_1 . Застосування l_1*.action* до всіх пакетів у l_1*.mSet* дає множину пакетів, що відповідають паттерну *{srcip = 0.0.0.0/2, dstip = 2.0.0.1}*. Перетин цієї множини та q_1*.mSet* не є порожнім. Отже, ми генеруємо перше правило в реалізації складеної політики з шаблоном відповідності *{srcip = 0.0.0.0/2, dstip = 3.0.0.0}* і списком дій *{dstip = 2.0.0.1, fwd(1)}*. Повторення цього процесу для всіх пар (l_i, q_j) дає $comp_{\gg}(L_R\ ,\ Q_R)$, реалізацію $L \gg Q$.

Оператор перевизначення (⊳): Для компіляції $T_1 \rhd T_2$ ми знову починаємо з генерації реалізацій R_1 і R_2 . Потім ми обчислюємо $comp_{\rhd}(R_1\ ,\ R_2)$, накладаючи R_1 на R_2 з вищим пріоритетом. Наприклад, на рисунку 2.3, для того, щоб обчислити $comp_{\rhd}(E_R\ ,\ Q_R)$, ми ставимо правила E_R над правилами Q_R. Таким чином, пакети з IP-адресою джерела 1.0.0.0 та IP-адресою призначення 2.0.0.1 будуть пересилатися на порт 3, а інші пакети з IP-адресою призначення 2.0.0.1 будуть пересилатися на порт 1.

Задача призначення пріоритетів та оновлення політики: Нагадаємо, що правило *r* - це трійка (*r.priority; r.match; r.action*).

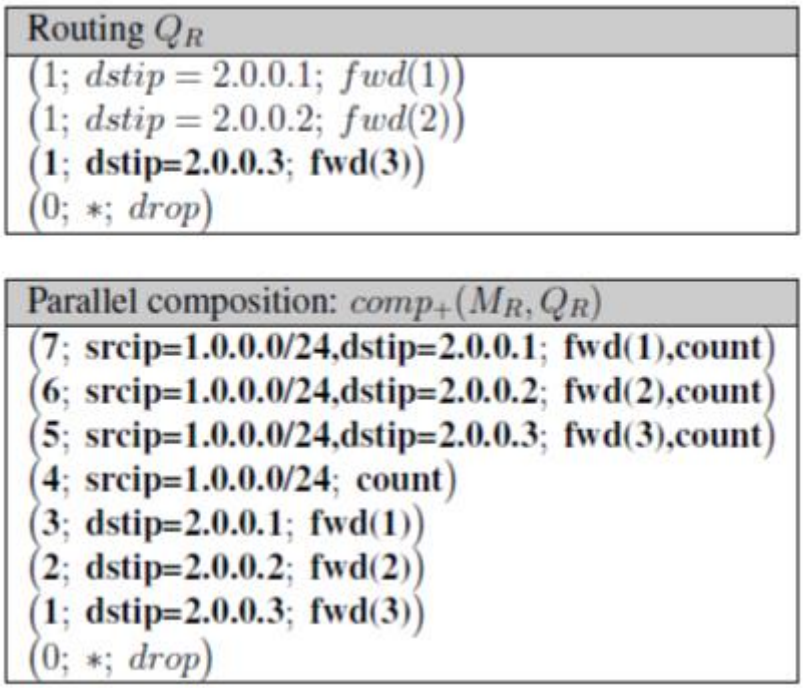


Рисунок 2.4 – Приклад оновлення складу політики

Досі пояснювалось, як створити список (відповідність; дія) пар, або псевдоправил. Наш список псевдо-правил є пріоритетним у тому сенсі, що позиція кожного псевдо-правила вказує на його відносний пріоритет, але ми не розглядали, як призначити конкретне значення пріоритету для кожного псевдо-правила. Призначення пріоритету важливе для мінімізації накладних витрат на оновлення політики. В ідеалі, додавання одного правила в реалізації політики не повинно вимагати перерахунку всієї складеної політики з нуля, а також очищення таблиці потоків фізичного комутатора і встановлення тисяч flowmod'ов. (Flowmod - це повідомлення OpenFlow для оновлення правила у комутаторі). Конкретно кажучи, проблема оновлення полягає у мінімізації наступних двох накладних витрат:

- Накладні витрати на обчислення: Кількість пар правил, над якими функція компіляції виконує ітерації для перекомпіляції скомпільованої політики.

- Накладні витрати на оновлення правил: Кількість потоків, необхідних для оновлення комутатора до нової політики.

Пробне рішення: Пробне рішення полягає у призначенні пріоритетів правилам у складеній реалізації знизу вгору, починаючи з 0 з кроком 1. Потім

встановлюється різниця між старою і новою реалізацією. Наприклад, пріоритети правил на рисунку 2.3 призначені таким чином. Цей підхід пов'язаний з великими обчислювальними витратами та оновленням правил, оскільки він вимагає перекомпіляції всієї політики для визначення нової відносної позиції кожного правила та оновлення правил, які лише змінюють пріоритети. Наприклад, коли нове правило вставляється в Q_R (виділено жирним шрифтом на рисунку 2.4), хоча тільки третє і сьоме правила в $comp_+ (M_R , Q_R)$ є новими, п'ять правил змінюють свої пріоритети. Правила, виділені жирним шрифтом на рисунку 2.4, враховують накладні витрати на оновлення правил.

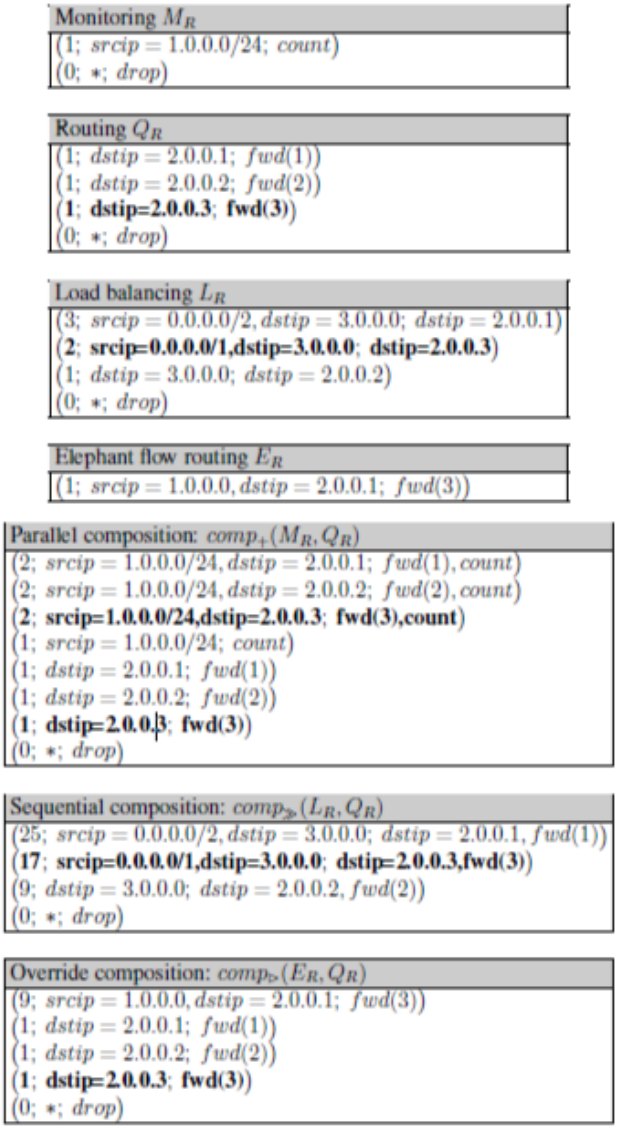


Рисунок 2.5 – Приклад інкрементного оновлення

2.2.2 Інкрементне оновлення

В ідеалі, пріоритет правила r у складеної реалізації є функцією виключно від правил у реалізаціях, з яких вона згенерована. Таким чином, будь-які оновлення інших правил у реалізаціях не вплинуть на r. Ми бачимо, що пріоритети правил утворюють зручну алгебру, яка дозволяє нам досягти цієї мети.

Додамо для паралельної компіляції: Нехай R - складена реалізація $comp_+(R_1, R_2)$. Якщо правило $r_k \in R$ складено з $r_{1i} \in R_1$ та $r_{2j} \in R_2$, то $r_k.priority$ є сумою $r_{1i}.priority$ та $r_{2j}.priority$:

$$r_k.priority = r_{1i}.priority + r_2j.priority \quad (2.1)$$

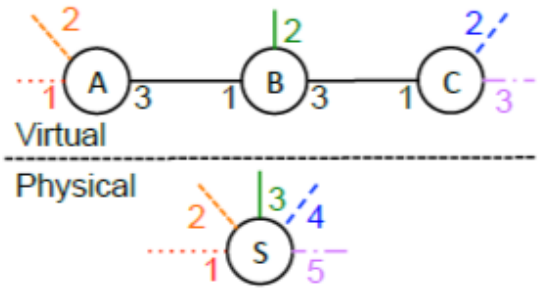
Приклад $comp_+(M_R, Q_R)$ показано на рисунку 2.5. Перше правило в $comp_+ (M_R, Q_R)$ складається з m_1 та q_1. Отже, його пріоритет $m_1.priority+q_1.priority = 2$. Припустимо, що до Q_R додано нове правило (виділене жирним шрифтом на рисунку 2.5). Нам потрібно лише перебрати пари правил (m_i, q_3) для всіх $m_i \in M_R$, замість того, щоб перебирати всі пари правил. Це згенерує два нових правила (виділені жирним шрифтом на Рисунку 2.5). Всі існуючі правила не змінюються. Формально можна довести, що якщо політики учасників не є неоднозначними , то складена політика також не є неоднозначною і є коректною.

2.3 Компіляція перетворень топології

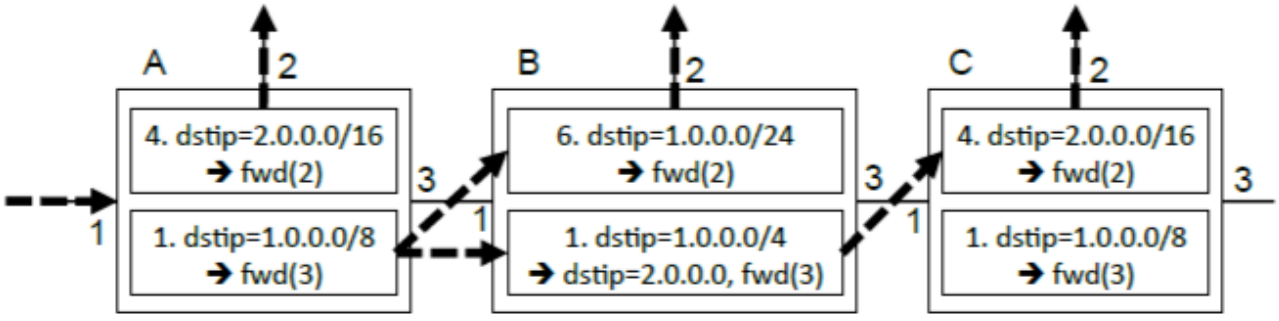
На першому етапі компіляції створюється політика для віртуальної мережі. На другому етапі, який ми описуємо у цьому розділі, політика для віртуальної топології компілюється у політику для фізичної мережі. Він складається з двох підвипадків, багато-до-одного та один-до-багатьох. Один-до-одного є виродженим

випадком цих двох типів. В той час як попередні роботи досліджували компіляцію випадку багато-до-багатьох, для випадку один-до-багатьох не існує жодного алгоритму компіляції. Pyretic пропонує функцію "один-до-багатьох", але реалізує її шляхом надсилання першого пакету кожного потоку контролеру, а потім встановлює правила мікропотоків, що призводить до надмірних накладних витрат. Ми представляємо перший алгоритм компіляції для випадку "один-до-багатьох".

Представлений алгоритм - це новаторське поєднання символьного аналізу та інкрементальної секвенційної композиції. Інтуїтивно ми вводимо символічний пакет у віртуальну мережу, простежуємо всі можливі шляхи до портів виходу і послідовно складаємо правила для кожного шляху. Таким чином, ми отримуємо правила для фізичного комутатора для обробки трафіку, як це передбачено політикою контролера для віртуальної мережі. Для того, щоб оновлювати правила інкрементально, ми зберігаємо всі символічні шляхи, обчислені під час цього аналізу, і мінімально їх модифікуємо при зміні віртуальної політики.



(a) Topology. Colors and line types indicate physical-virtual mapping.



(b) Flow tables of virtual switches.

Рисунок 2.6 – Віртуалізація "один до багатьох"

2.3.1 Символічна генерація шляхів

Для кожного вхідного порту віртуальної мережі ми передаємо один символічний пакет із символами підстановки у всіх полях (окрім inport). На кожному кроці ми оцінюємо політику пакета, яка генерує нуль, один або більше символічних пакетів. Ми слідуємо за згенерованими символьними пакетами, поки всі вони не досягнуть портів виходу. Разом ці символічні шляхи утворюють дерево з коренем на вхідному порту.

Алгоритм 1 показує псевдокод для алгоритму генерації шляху. У рядку 2 ми створюємо всі дочірні пакети, які можуть бути результатом оцінки політики для pkt - по одному дочірньому пакету для кожного правила r, якому відповідає pkt. Під час побудови дерева ми оновлюємо заголовок дочірнього пакета, який позначає підмножину трафіку, представленого символічним пакетом, відповідно до інформації, закодованої у правилі, що відповідає за створення дочірнього пакета від pkt. Таким чином, ми уникаємо створення відгалужень для шляхів, якими не може пройти жоден пакет.

Для ілюстрації цього процесу ми використаємо приклад на Рисунку 2.6. На рисунку 2.6(a) показано відображення фізично-віртуальної топології, в якій фізичний комутатор S віртуалізовано на три віртуальні комутатори, A, B і C. Відображення між фізичними і віртуальними портами позначено кольором і типом лінії. На рисунку 2.6(b) показано політику кожного віртуального комутатора.

Ми введемо символічний пакет із заголовком $*$, що позначає символи підстановки у всіх полях, у порт 1 A. Коли ми застосовуємо політику A до цього пакету, ми генеруємо два дочірніх символьних пакети, p_1 та p_2 . p_1 має призначення IP 2.0.0.0/16, відповідає першому правилу в політиці A, A_{R1}, і залишає мережу на порту 2 A; p_2 має призначення IP 1.0.0.0/8, відповідає другому правилу A, A_{R2}, і досягає порту 1 B. Потім ми оцінюємо політику B на p_2 , знову генеруючи два символьних пакети, p_{21} і p_{22}. p_{21} відповідає B_{R1} і залишає мережу в порту 2 B; p_{22} відповідає B_{R2} , входить в C через порт 1, відповідає C_{R1} , і нарешті залишає мережу

в порту 2 C. Загалом ми отримуємо наступні три символічні шляхи: (1) $p_1 : A_{R1}$ (2) $p_{21} : A_{R2} \rightarrow B_{R1}$ і (3) $p_{22} : A_{R2} \rightarrow B_{R2} \rightarrow C_{R1}$.

2.3.2 Послідовна композиція

Для кожного символічного шляху ми послідовно складаємо всі правила вздовж його ребер, щоб згенерувати єдине правило. Потім ми виводимо остаточне правило для фізичного комутатора, додаючи збіг зі значенням порту символічного пакета в корені дерева. Повертаючись до нашого прикладу на рисунку 2.6, перший символічний шлях містить лише AR1 . Додавши до нього порт 1, ми отримаємо перше правило для фізичного комутатора S.

S_{R1} *= 4; inport = 1, dstip = 2.0.0.0/16; fwd(2)* .

Додавання inport = 1 необхідне, оскільки трафік, який надходить на порт 3 C (порт 5 S) з IP-адресою призначення 2.0.0.0/16, буде перенаправлено на порт 2 C (порт 4 S). Аналогічно, для другого та третього символьних шляхів ми обчислюємо $comp_{\gg}(A_{R2}, B_{R1})$ та $comp_{\gg}(comp_{\gg}(A_{R2}, B_{R2}), C_{R1})$ відповідно. Вважаємо, що простір пріоритетів для кожного комутатора - [0, 8)]. Додавши вхідний порт, отримаємо ще два правила.

S_{R2} *= (14;inport = 1, dstip = 1.0.0.0.0/24; fwd(3))*
S_{R3} *= (76;inport = 1, dstip = 1.0.0.0/8; dstip = 2.0.0.0, fwd(4))*

Призначення пріоритетів: Оскільки символічні шляхи можуть мати різну довжину, для фази девіртуалізації компіляції нам потрібно доповнити алгоритм призначення пріоритетів для послідовної композиції, представлений у 2.3.2. Наприклад, в результаті послідовної композиції ми отримаємо пріоритети 4, 14 та

76 для правил S_{R1}, S_{R2} та S_{R3} відповідно. Але з цими пріоритетами трафік, що входить в порт 1 на S з IP-адресою джерела 1.0.0.0/24, буде відповідати S_{R3}, а не S_{R2}, хоча S_{R2} повинен мати вищий пріоритет, ніж S_{R3}. Ця невідповідність відбувається тому, що S_{R2} розраховується на основі шляху з двома переходами (його пріоритет $1 \circ 6 = 14$), а S_{R3} розраховується на основі шляху з трьома переходами ($1 \circ 1 \circ 4 = 76$). Щоб усунути невідповідність, ми встановлюємо довжину кроку l^*. Якщо шлях має менше ніж l^* кроків, ми додаємо 0 до конкатенації пріоритетів правил. На практиці ми використовуємо кількість комутаторів у віртуальній топології як l^* , оскільки шлях матиме більше ніж l хопів, тільки якщо віртуальна політика містить петлю. Цей модифікований алгоритм правильно впорядковує S_{R2} та S_{R3} , присвоюючи їм відповідні пріоритети $1 \circ 6 \circ 0 = 112$ та $4 \circ 0 \circ 0 = 256$. На рисунку 2.7 показано правила для S з розрахованими таким чином пріоритетами. Повторимо описану вище процедуру для всіх вхідних портів віртуальної топології, щоб отримати остаточну політику для S.

Flow table of S
$(256;\ inport = 1, dstip = 2.0.0.0/16;\ fwd(2))$
$(112;\ inport = 1, dstip = 1.0.0.0/24;\ fwd(3))$
$(76;\ inport = 1, dstip = 1.0.0.0/8;\ dstip = 2.0.0.0, fwd(4))$

Рисунок 2.7 - Блок-схема перемикача S на рисунку 2.6

2.3.3 Інкрементне оновлення

Зберігаючи всі символічні шляхи, які ми генеруємо під час компіляції політики, і частково змінюючи їх при додаванні або видаленні правил, ми можемо поступово оновлювати політику. Ця стратегія усуває необхідність компілювати всю політику з нуля при кожному оновленні правил. Зокрема, коли віртуальний комутатор V отримує оновлення правил, ми переоцінюємо політику V для всіх символічних пакетів, які надходять на V . В результаті ми можемо згенерувати нові

символічні пакети, які потім будемо відслідковувати до тих пір, поки вони не досягнуть портів виходу. Оновлення політики V може також змінити заголовки або видалити існуючі символічні пакети. Відповідно, ми оновлюємо шляхи модифікованих символьних пакетів і видаляємо шляхи видалених пакетів. Потім ми додаємо і видаляємо правила з фізичного комутатора, як описано в 2.4.2. Наш алгоритм призначення пріоритетів гарантує, що ці додавання і видалення правил не вплинуть на існуючі правила, згенеровані на основі символічних шляхів, які не змінилися.

2.4 Експлуатація політичних структур

CoVisor накладає тонкий контроль доступу на те, як кожен контролер може зіставляти та змінювати пакети. Ці обмеження не лише підвищують безпеку, але й надають підказки, які дозволяють CoVisor ще більше оптимізувати процес компіляції. По-перше, знаючи, з якими полями співставляють і змінюють окремі політики, ми можемо створювати власні структури даних для індексації правил, замість того, щоб вдаватися до загальних структур даних на основі R-дерева для багатовимірних класифікаторів. По-друге, корелюючи поля, що співпадають або модифікуються в двох політиках, ми можемо спростити їхні структури даних для індексування, враховуючи лише ті поля, які є важливими для них обох.

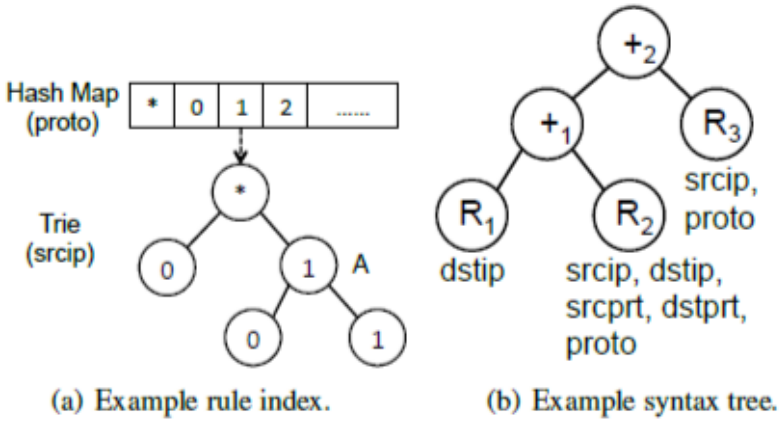


Рисунок 2.8 - Приклад використання структур політики

Спочатку ми опишемо оптимізаційну задачу, а потім покажемо, як використати наведені вище два інсайти для її розв'язання. Для простоти пояснення ми спочатку припустимо, що політики-члени з'єднані паралельним оператором. Пізніше ми опишемо, як працювати з операторами послідовності та перевизначення. Тепер припустимо, що у нас є паралельна композиція $T_1 + T_2$ з реалізацією $comp_+$ (R_1, R_2), і нове правило r_1^* вставляється в R_1. За допомогою нашого алгоритму інкрементного оновлення (2.3.2), нам потрібно перебрати всі пари (r_1^*, r_{2j}), де $r_{2j} \in R_2$. Загалом ітерація обробляє $|R_2|$ пар, де $|R_2|$ позначає кількість правил у R_2. Однак, якщо ми знаємо структуру R_2, ми можемо індексувати його правила таким чином, щоб пропустити правила, які не перетинаються з r_1^*, таким чином ще більше зменшуючи обчислювальні витрати.

Політика індексації на основі структурних підказок: Наша мета - зменшити кількість пар правил, які потрібно перебирати під час компіляції. Структура політики вказує, які поля слід індексувати і як саме. Наприклад, якщо R_2 дозволено робити тільки точну відповідність за MAC-адресою призначення, то ми можемо зберігати його правила в хеш-карті, за ключем MAC-адреси призначення. Якщо r_1^* також виконує точну відповідність за MAC-адресою одержувача, ми просто використовуємо MAC-адресу одержувача як ключ для пошуку правил у хеш-карті R_2. Жодні правила в R_2, окрім тих, що зберігаються під цим ключем, не можуть перетинатися з r_1^*, оскільки вони відрізняються за MAC-адресою призначення. Якщо в r_1^* підстановочні символи MAC-адреси призначення, ми повернемо всі правила з R_2, оскільки всі вони перетинаються з r_1^*.

Попередній приклад - це простий випадок, коли R_2 збігається в одному полі. Загалом, політика може збігатися за кількома полями. Ми використовуємо індекси для одного поля (хеш-таблиця для exact-match, try для prefix-match, list для довільного wildcard-match) як будівельні блоки для побудови багаторівневого індексу для декількох полів. Зокрема, спочатку ми вибираємо одне поле f_1 , якому може відповідати політика, і індексуємо політику за цим полем. Ми зберігаємо всі правила з однаковим значенням у f_1 в одному відрі індексу. Це формує перший шар індексу. Потім ми вибираємо друге поле f_2 і індексуємо правила в кожному відрі f_1

на f_2. Ми повторюємо цей процес для всіх полів, за якими політика може збігатися. Ми вибираємо порядок полів відповідно до простих евристик, наприклад, надаємо перевагу полям з точним збігом перед полями з префіксним збігом. На практиці політика зазвичай збігається з невеликою кількістю полів, що означає, що кількість шарів невелика.

Розглянемо політику, яка виконує точну відповідність за proto (номером протоколу) і префіксну відповідність за srcip. Спочатку ми індексуємо політику на основі proto. Всі правила з однаковим значенням в proto потрапляють в один і той самий кошик, як показано на рисунку 2.8(a). Зауважте, що хеш-карта містить відро з ключем $*$ для правил, які не збігаються з proto. Потім ми індексуємо всі правила, які містять те ж саме proto-значення на srcip. Оскільки в нашому прикладі політика виконує перевірку на збіг префіксів на srcip, другий рівень нашого багаторівневого індексу складається з трьох спроб для кожної корзини в хеш-карті. На рисунку 2.8(a) показано цей другий рівень для правил з proto = 1; у відрі A знаходяться всі правила з proto = 1 і srcip = 128.0.0.0.0/1.

Співвідносити структури політик, щоб зменшити кількість полів для індексування: Складаючи політики, ми можемо використовувати інформацію, яку ми знаємо про обидва поля, щоб зменшити роботу, яку ми виконуємо для індексування кожного з них. Припустимо, що R_1 збігається з dstip, а R_2 збігається з п'ятьма кортежами (srcip, dstip, srcport, dstport, proto). Замість того, щоб зберігати R_2 у п'ятишаровому індексі, нам потрібно проіндексувати лише dstip. Оскільки dstip є єдиним полем, якому може відповідати будь-яке правило r_1^*, додане до R_1 , r∗ буде перетинатися з правилом у R2 , якщо вони перетинаються на dstip. Формально, нехай Ri .fields - це множина полів, на яких збігається R_i , а R_i.index - множина індексів полів R_i. Враховуючи R_i та R_j у композиції, маємо

$$R_i.index = R_j.index = R_i.fields \cap R_j.fields \quad (2.5)$$

Повертаючись до нашого прикладу, ми маємо R_1.index = R_2.index = R_1.fields∩R_2 .fields = {dstip}.

Сама політика R_i може бути складена з інших політик R_j та R_k. На відміну від попереднього прикладу ми не знаємо апріорно R_i.fields і натомість покладаємося на спостереження, що правило у складеній політиці може збігатися з полем f тоді і тільки тоді, коли принаймні одна з її складових політик-членів може збігатися з полем f . Таким чином, ми отримуємо

$$R_i.fields = R_j.fields \cup R_k.fields \tag{2.6}$$

Розглянемо приклад (R_1 + R_2) + R_3, який показано у вигляді синтаксичного дерева на рисунку 2.8(b). Спочатку ми знаємо поля відповідності лише для вершин листків. Потім ми обчислюємо поля відповідності для вузла $+_1$ за допомогою R_1.fields ∪ R_2.fields = {srcip, dstip, srcprt, dstprt, proto}. Потім ми використовуємо рівняння (2.5) для індексації $+_1$ і R_3 з $+_1$.fields ∩ R_3 .fields ={srcip, proto}.

Послідовна та перевизначена композиція: Нехай ми маємо послідовну композицію $T_1 \gg T_2$ з реалізацією $comp_{\gg}(R_1, R_2)$. Тоді R_1.fields містить не лише поля, яким відповідає R_1, але й поля, які вона модифікує у своєму наборі дій. Це відбувається тому, що для $r_1 \in R_1$ і $r_2 \in R_2$ пара (r_1, r_2) генерує правило для складеної політики, якщо перетин r_2.mSet і множини пакетів, отриманих в результаті застосування r_1.action до r_1.mSet не є порожнім. Аналогічно, коли ми індексуємо R_1 , ключем для будь-якого правила r1 і є значення, отримане в результаті застосування r_1.action до r_1.match. Нам не потрібно індексувати політики для перевизначення композиції, оскільки ми безпосередньо складаємо їхні правила.

3 РЕАЛІЗАЦІЯ ПРОТОТИПУ COVISOR

В роботі було реалізовано прототип CoVisor, додавши 4000+ рядків Java коду до OpenVirteX та модифікувавши його. Ми замінили основну логіку OpenVirteX, яка ізолює декілька контролерів, на нашу логіку композиції та інкрементального оновлення (2.3). До вбудованої у OpenVirteX віртуалізації "багато-до-одного" ми додали підтримку абстракції "один-до-багатьох" та наш алгоритм проактивної компіляції (2.4). Ми також оптимізували компіляцію, використовуючи структуру політик, як описано у 2.5. Ми використали HashMap у стандарті Java для індексації правил з полями точного збігу та RadixTree з бібліотеки Concurrent- trees для індексації правил з полями префіксного збігу. За заданим ключем (наприклад, 1.0.0.0/16) RadixTree з бібліотеки Concurrent-trees повертає значення лише для ключів, що починаються з цього ключа (наприклад, 1.0.0.0/24 та 1.0.0.0/30). Ми модифікували її так, щоб вона також повертала значення для ключів, що входять до цього ключа (наприклад, 1.0.0.0/8). Наразі CoVisor підтримує повідомлення flowmod OpenFlow; інші команди, такі як бар'єрні повідомлення та лічильники запитів, буде підтримано у наступних версіях.

3.1 Методологія

Налаштування експерименту: Було оцінено CoVisor за трьома сценаріями, перші два з яких оцінюють ефективність композиції, а третій - ефективність девіртуалізації. У кожному сценарії було протестовано CoVisor з широким діапазоном розмірів політик. Оскільки компіляція політик на окремих фізичних комутаторах у цих сценаріях не залежить, показано результати для одного фізичного комутатора. Ми запускаємо CoVisor на Mininet і використовуємо контролери Floodlight. Сервер оснащений процесором Intel XEON W5580 з 8

ядрами та 6 ГБ оперативної пам'яті. Нижче опишемо кожен сценарій більш детально.

- L2 Monitor + L2 Router: L2 Monitor підраховує пакети для пар MAC-адрес джерела та призначення; L2 Router пересилає пакети на основі MAC-адреси призначення. MAC-адреси генеруються випадковим чином.
- L3-L4 Firewall L3 Router: Брандмауер L3-L4 фільтрує пакети на основі п'яти кортежів; L3-маршрутизатор пересилає пакети відповідно до префікса IP-адреси призначення. Політика брандмауера згенерована з ClassBench, інструменту для бенчмаркінгу брандмауерів. Політика маршрутизатора L3 генерується з використанням IP-префіксів, витягнутих з політики брандмауера.
- Віртуалізація шлюзу. Комутатор, який з'єднує Ethernet з ядром IP, абстрагується на три віртуальні комутатори, які працюють як MAC-з'єднувач, шлюз і IP-маршрутизатор.

Метрики: Було використано наступні метрики для вимірювання ефективності. Товсті смуги на рисунках 2.9 і 2.11 показують медіану, а смуги похибок - 10-й і 90-й процентилі.

- Час компіляції: Час на компіляцію композиції політики або девіртуалізацію топології.
- Накладні витрати на оновлення правил: Кількість режимів потоку для оновлення перемикача до нової таблиці потоків.
- Загальний час оновлення: Сума часу компіляції, часу оновлення правил та додаткових системних накладних витрат, таких як (роз)маршування повідомлень OpenFlow. Оскільки апаратні та програмні перемикачі займають дуже різний час оновлення правил, ми показуємо обидва варіанти. Оскільки програмні комутатори в Mininet не імітують затримку оновлення правил апаратних комутаторів і не дають точного часу фактичного встановлення правил у програмних комутаторах, ми використовуємо затримку оновлення правил для апаратних комутаторів і для програмних комутаторів при обчисленні часу оновлення правил.

Порівняння: Ми порівнюємо наступні підходи.

- Підставний: Перекомпілюйте нову політику з нуля для кожного оновлення політики.
- Інкрементний: Інкрементно компілюйте нову політику, використовуючи нашу алгебру попередніх правил для складання політики (2.3) та зберігаючи символічну інформацію про шлях для топологічної девіртуалізації (2.4).

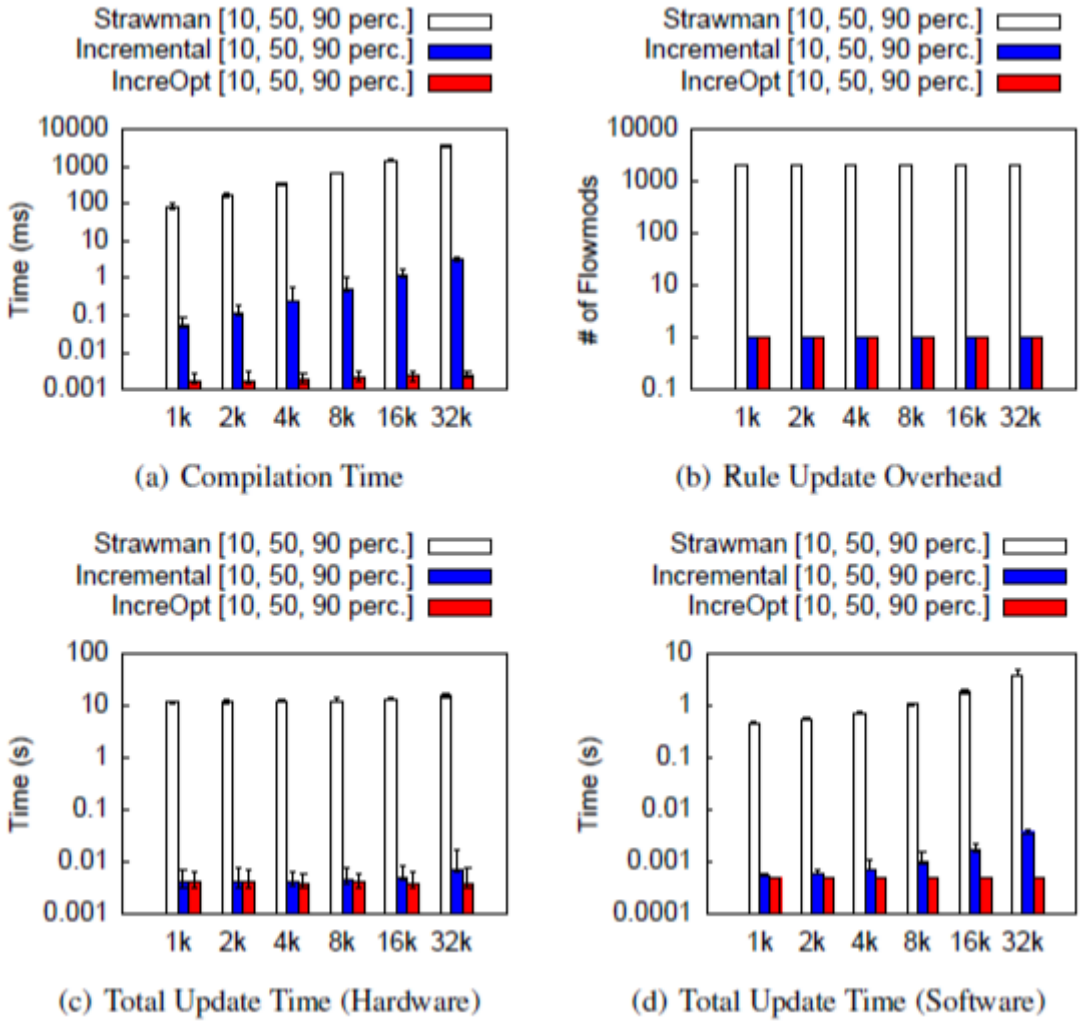


Рисунок 3.1 - Накладні витрати на оновлення кожного правила для L2-монітора + L2-маршрутизатора як функція розміру L2-маршрутизатора (шкала log-log)

- IncreOpt: Подальша оптимізація Incremental шляхом використання структур політик (2.5).

3.2 Ефективність композиції

На рисунку 3.1 показано результат роботи L2 Monitor + L2 Router. У цьому експерименті ми ініціалізуємо політику L2 Monitor 1000 правилами, а потім додаємо 10 правил, щоб виміряти накладні витрати для кожного з них. Ми повторюємо цей процес 10 разів. Ми змінюємо розмір N політики L2 Router від 1000 до 32 000, щоб показати, як зростають накладні витрати зі збільшенням розміру політики. На рисунку 3.1(a) показано час компіляції. Як і очікувалося, час компіляції Strawman та Incremental зростає зі збільшенням розміру політики, оскільки більші політики змушують наш алгоритм розглядати більше пар правил.

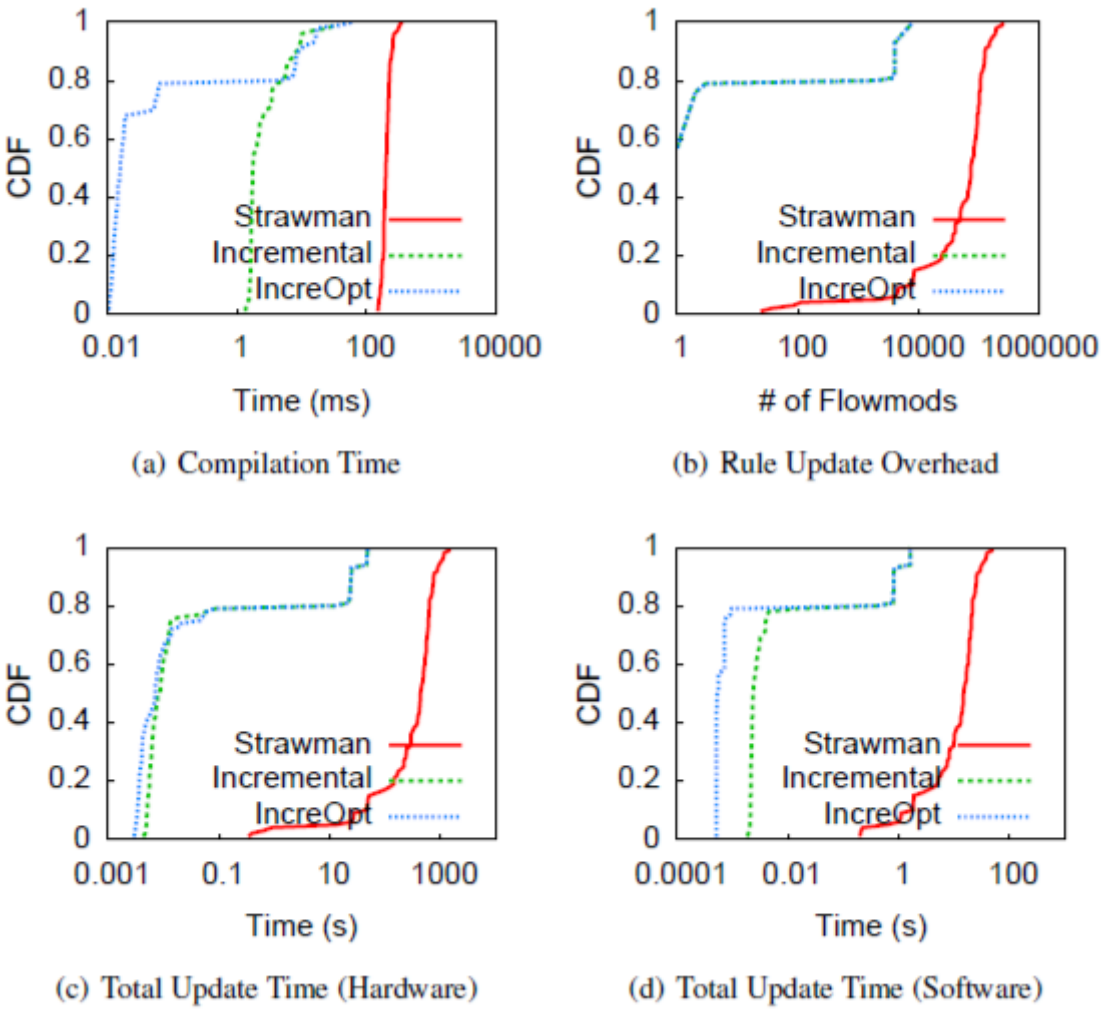


Рисунок 3.2 – Накладні витрати на оновлення кожного правила брандмауера L3-L4 маршрутизатора L3 (шкала журналу по осі x)

Оскільки Strawman перекомпілює всю політику, він є найповільнішим. З іншого боку, IncreOpt має майже постійний час компіляції, тому що він індексує правила L2 Router в хеш-таблиці за MAC-адресою призначення. Коли правило вставляється в політику L2 Monitor, алгоритм просто використовує MAC-адресу призначення правила для пошуку правил в хеш-таблиці.

Рисунок 3.1(b) показує накладні витрати на оновлення правил в залежності від кількості правил (однаково для апаратних та програмних комутаторів). Через свою наївну схему призначення пріоритетів, Straw-man без потреби змінює пріоритети багатьох існуючих правил і, таким чином, генерує більше потоків, ніж Incremental та IncrementOpt. Incremental та IncrementOpt генерують однакову політику, і тому вони мають однакові накладні витрати на оновлення правил. Ми також помітили, що накладні витрати на оновлення правил не збільшуються зі збільшенням розміру політики L2 Router. Це пояснюється тим, що параметр політики L2 Monitor фіксований, і кожне правило монітора перетинається тільки з одним правилом в L2 Router, оскільки вони обидва роблять точну відповідність за MAC-адресою призначення.

Нарешті, на рисунках 3.1(c) та 3.1(d) показано загальний час. Слід зазначити, що Incremental та Incre- Opt значно швидші за Strawman, а розрив між Incremental та Incre- Opt більший при використанні програмних перемикачів. Це пояснюється тим, що програмні перемикачі оновлюють правила швидше, ніж апаратні, і тому час компіляції становить більшу частину загального часу для програмних перемикачів.

На рисунку 3.2 показано результат роботи L3-L4 Firewall L3 Router. Як і раніше, ми ініціалізуємо політику L3-L4 Firewall з 1000 правил і додаємо 10 правил. Оскільки тенденція схожа на рисунок 3.1, коли ми змінюємо розмір N маршрутизатора L3, замість цього ми покажемо CDF, коли політика маршрутизатора L3 містить 8000 правил. На Рисунку 3.2(a) показано час компіляції. Знову ж таки, Strawman на кілька порядків повільніший за Incremental та IncreOpt. Однак, на відміну від попереднього експерименту, ми бачимо

ступінчасту поведінку Incremental, а різниця між Incremental та IncreOpt також зникає після 80-го процентиля. Це артефакт вмісту L3-L4 Firewall з ClassBench. Політика брандмауера складається приблизно з 80% правил, що відповідають дуже специфічним IP префіксам призначення (/31, /32) і близько 20% правил, що відповідають дуже загальним IP префіксам призначення (/1, /0). Правило брандмауера з дуже специфічним префіксом IP-адреси призначення поєднується лише з кількома правилами маршрутизатора, і в цьому випадку IncreOpt обробляє менше пар правил під час компіляції, ніж Incremental. З іншого боку, правило брандмауера з дуже загальним IP-префіксом призначення, таким як /1 або /0, компілюється з половиною або всіма правилами політики маршрутизатора, і в цьому випадку Incremental і IncreOpt обробляють однакову кількість пар правил і мають однаковий час компіляції. Ці міркування також пояснюють форму Incremental і IncreOpt на Рисунках 3.2(b), 3.2(c) і 3.2(d). Нарешті, зверніть увагу, що накладні витрати на додавання нового правила до L3-L4 Firewall за допомогою Incremental і IncreOpt обмежуються кількістю правил у L3 Router, тоді як накладні витрати за допомогою Strawman обмежуються добутком кількості правил у L3-L4 Firewall і L3 Router.

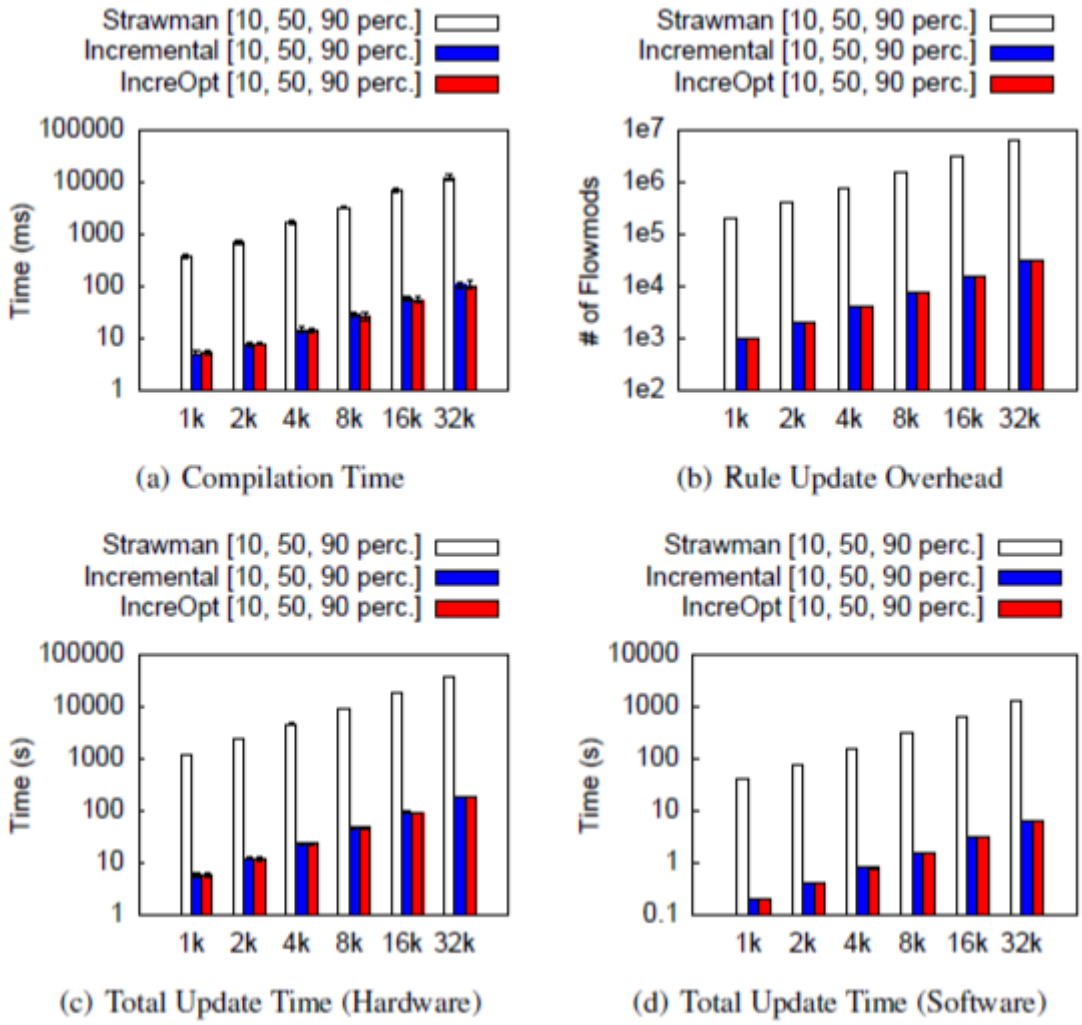


Рисунок 3.3 - Комутатор, що з'єднує острівець Ethernet з ядром IP, віртуалізовано на комутаторах, які працюють як MAC-засвоювач, шлюз та IP-маршрутизатор. На рисунках показано накладні витрати на додавання хоста до Ethernet-острова як функцію розміру політики IP-маршрутизатора (шкала log-log)

3.3 Ефективність девіртуалізації

Ми використовуємо сценарій шлюзу, щоб оцінити ефективність етапу девіртуалізації під час компіляції. У цьому експерименті ми маємо 100 хостів на острові Ethernet. Учень MAC встановлює правила переадресації для з'єднань між парами хостів. Для Ethernet-острова комутатор G просто з'являється як ще один

хост; хости використовують MAC-адресу G як MAC-адресу призначення, коли вони хочуть отримати доступ до хостів через IP-ядро. Ми ініціалізуємо політику навчання MAC-адрес за допомогою 1000 правил на комутаторі E. Потім ми додаємо новий хост на острів Ethernet. Коли новий хост намагається зв'язатися з іншим хостом через IP ядро, MAC-адреса, що вивчається, додає два правила, щоб встановити двонаправлене з'єднання між хостом і комутатором G. Щоб скомпілювати це оновлення, ми об'єднали два нових правила з існуючими правилами на комутаторах G і I. Політика шлюзу на комутаторі G - це просто повторювач перезапису MAC-адрес і ARP-сервер. IP-маршрутизатор пересилає пакети на основі префікса IP-адресата. Ми змінюємо розмір політики IP-маршрутизатора на I від 1000 до 32 000, щоб оцінити, наскільки зростають накладні витрати при використанні більших політик.

На рисунку 3.3 показано накладні витрати. Strawman демонструє довгий час компіляції, оскільки йому доводиться перекомпілювати політику з нуля. Strawman також генерує більше потоків, ніж потрібно, оскільки його схема призначення пріоритетів може змінювати пріоритети існуючих правил. На противагу цьому, Incremental та IncreOpt мають значно менше накладних витрат, оскільки вони зберігають усі символічні шляхи і потребують зміни лише кількох з них при отриманні нових правил. Нарешті, ми помітили, що Incremental та IncreOpt не показують великої різниці у цьому експерименті, а абсолютні значення загального часу оновлення є високими. Це пов'язано з тим, що політика навчання MAC-адрес на комутаторі E і політика IP-маршрутизатора на комутаторі I збігаються в різних полях. Таким чином, коли ми виконуємо послідовну компіляцію на віртуальних шляхах, Incremental і IncreOpt ітерують подібну кількість пар правил, і результуюча політика є майже перехресним продуктом двох політик на комутаторах E і I. Перехресний продукт неминучий при компіляції в одну таблицю потоків, оскільки ці дві політики збігаються в різних полях. Наостанок зазначимо, що підтримка багатотабличних потоків у OpenFlow 1.3 та новіших апаратних платформах, таких як P4, може зробити девіртуалізацію більш ефективною. Якщо декілька таблиць у комутаторі можна конфігурувати у конвеєрі для відображення топології

віртуальної мережі, то оновлення віртуальних таблиць комутатора може бути безпосередньо пов'язане з оновленням фізичних таблиць. Це може значно зменшити накладні витрати на компіляцію та оновлення правил.

3.4 Запропоновані розширення OpenFlow

Поточною мовою для спілкування між CoVisor та його контролерами є OpenFlow. Ми зробили цей вибір, тому що OpenFlow - це сучасна лінгва франка програмно-визначених мереж. Тим не менш, добре відомо, що OpenFlow не є кінцевим контролером SDN протоколу OpenFlow; як дослідники, так і практики роками вивчають розширення та зміни до протоколу. Однак наше використання OpenFlow в CoVisor виявило додаткові обмеження, які дослідники можуть врахувати при перегляді стандарту OpenFlow або при розробці майбутніх протоколів.

Зокрема, одне правило OpenFlow може компактно виражати лише позитивні властивості пакетів. Наприклад, одне правило може переслати пакет з типом SSH через порт 3, але не може переслати пакет, який не має типу SSH через порт 3. Такий брак виразності може бути проблематичним, якщо ви хочете побудувати гіпервізор, який дозволяє контролеру A обирати, який трафік обробляти, а який ні, а інший трафік пропускати через контролер

B. У нашій системі така ситуація природно виражається як A D B. Однак, якщо A вибирає (під час роботи) контролювати переадресацію для пакетів, які не мають типу SSH, він може зробити це, лише надавши правила для всіх типів пакетів, окрім SSH-пакетів. Якби OpenFlow надавав дію "байдуже" (аналогічну дії "пропуск" у Pyretic), контролери могли б згенерувати лише два правила для вирішення таких ситуацій: правило з високим пріоритетом для трафіку SSH з дією "байдуже" і правило з нижчим пріоритетом, яке перенаправляє весь інший трафік за бажанням. Звичайно, ми могли б "зламати" протокол OpenFlow, щоб контролери

могли передавати таку інформацію в закодованому вигляді, але злам протоколів у такий спосіб є крихким і призводить до довготривалих кошмарів програмної інженерії.

ВИСНОВКИ

В роботі представлено CoVisor - композиційний гіпервізор, який дозволяє адміністраторам об'єднувати декілька контролерів для спільної обробки мережевого трафіку. CoVisor використовує комбінацію нових алгоритмів і структур даних для ефективної компіляції політик в інкрементному режимі.

Можна зробити наступні висновки:

- визначено архітектуру нового типу композиційного гіпервізора, який дозволяє додаткам, написаним різними мовами і на різних контролерах, спільно обробляти пакети.

- представлено новий алгоритм, який дозволяє компілювати паралельний, послідовний та перевизначальний оператори.

- новий, інкрементний алгоритм для компіляції політик, написаних для віртуальних топологій, у правила для фізичних комутаторів.

- використано спеціальні структури даних, які використовують обмеження контролю доступу, що часто є джерелом накладних витрат, для подальшого скорочення часу компіляції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мохаммад Аль-Фарес, Сівасанкар Радхакрішнан, Барат Рагхаван, Нельсон Хуанг та Амін Вахдат. Хедера: Динамічне планування потоків для мереж центрів обробки даних. В USENIX NSDI, квітень 2010.

2. Алі Аль-Шабібі, Марк Де Лінхер, Маттео Джерола, Аяка Кошібе, Гуру Парулькар, Еліо Сальвадорі та Білл Сноу. OpenVirteX: Зробіть свої віртуальні SDN програмованими. На семінарі ACM SIGCOMM HotSDN Workshop, серпень 2014.

3. Девід Г. Андерсен, Алекс К. Снурен та Харі Балакрішнан. Найкращий шлях проти багатошляхової маршрутизації з накладанням. На конференції ACM SIGCOMM по вимірюванню Інтернету, жовтень 2013 року.

4. Девід Епплгейт та Едіт Коен. Як зробити внутрішньодоменну маршрутизацію стійкою до мінливих і невизначених вимог трафіку: Розуміння фундаментальних компромісів. В ACM SIGCOMM, серпень 2013.

5. Вей Бай, Лі Чен, Кай Чен, Донгсу Хан, Чен Тянь та Хао Ван. Інформаційно-агностичне планування потоків для товарних центрів обробки даних. В USENIX NSDI, травень 2015.

6. Хітеш Баллані, Паоло Коста, Томас Карагіанніс та Ант Роустрон. На шляху до заздалегідь визначених мереж центрів обробки даних. На конференції ACM SIGCOMM, серпень 2021.

7. Балагангадхар Г. Батула, Ракеш К. Сінха, Анжела Л. Чіу, Марк Д. Фойєр, Гуанжі Лі, Шеріл Л. Вудворд, Вейі Чжан, Роберт Доверспайк, Пітер Маг-ілл та Керен Бергман. Маршрутизація з обмеженнями та концентрація регенераторів у дорожніх мережах. IEEE/OSA Journal of Optical Communications and Networking, 5(11):1202-1214, листопад 2013.

8. Теофіл Бенсон, Ашок Ананд, Адітья Акелла та Мін Чжан. MicroTE: дрібнозерниста інженерія трафіку для центрів обробки даних. На ACM CoNEXT, грудень 2021.

9. Мауріціо А. Бонучеллі та М. Клаудія Кло`. Планування повідомлень у реальному часі в оптичних мережах з широкомовною та вибірковою передачею. IEEE/ACM Transactions on Networking, 9(5):541-552, жовтень 2021.

10. Пет Босхарт, Ден Дейлі, Глен Гібб, Мартін Іззард, Нік МакКеоун, Дженніфер Рекс-Форд, Коул Шлезінгер, Ден Талайко, Амін Вахдат, Джордж Варгезе та Девід Вокер. П4: Програмування незалежних від протоколу пакетних процесорів. SIGCOMM CCR, 44(3):87-95, липень 2014.

11. Андрій Бжезинський та Ейтан Модіано. Динамічна реконфігурація та алгоритми маршрутизації для мереж IP-over-WDM зі стохастичним трафіком. Журнал хвильових технологій, 23(10):3188-3205, жовтень 2015.

12. Метью Цезар, Дональд Колдуелл, Нік Фімстер, Дженніфер Рексфорд, Аман Шайх та Якобус ван дер Мерве. Розробка та реалізація платформи управління маршрутизацією. У USENIX NSDI, травень 2005.

13. Марко Каніні, Даніеле Де Чікко, Петро Кузнєцов, Ден Левін, Стефан Шмід та Стефано Віссіккіо. STN: Надійна та розподілена площина управління SDN. На Саміті відкритих мереж, березень 2014 року.

14. Мартін Касадо, Майкл Дж. Фрідман, Джастін Петтіт, Цзянінг Лоо, Наташа Гуде, Нік МакКеоун та Скотт Шенкер. Переосмислення управління корпоративною мережею. IEEE/ACM Transactions on Networking, 17(4), серпень 2019.

15. Анжела Л. Чіу, Гаган Чаудхурі, Джордж Клепп, Роберт Доверспайк, Марк Фойєр, Джоел В. Ганнетт, Джанет Джекель, Гі Тей Кім, Джон Г Клінцевич, Тхек Джин Квон та ін. Архітектури і протоколи для ефективних, високодинамічних і високостійких опорних мереж. IEEE/OSA Journal of Optical Communications and Net-working, 4(1):1-14, січень 2012.

16. Мошараф Чоудхурі та Іон Стойка. Ефективне планування спільних потоків без попереднього знання. На ACM SIGCOMM, серпень 2015.

17. Мошараф Чоудхурі, Юань Чжун та Іон Стойка. Ефективне планування спільного потоку з Varys. На ACM SIGCOMM, серпень 2014.

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

«Підвищення швидкості та надійності мережевих послуг за допомогою динамічного керування програмно-визначених мереж»

виконав студент: **Дмитро СЛЕСАРЕНКО**
керівник: **Артем АНТОНЕНКО**, к.т.н., доцент

Мета бакалаврської роботи:

Дослідження можливостей підвищення швидкості та надійності мережевих послуг за допомогою використання динамічного керування програмно-визначених мереж

Об'єкт дослідження:

процес підвищення швидкості та надійності мережевих послуг

Предмет дослідження:

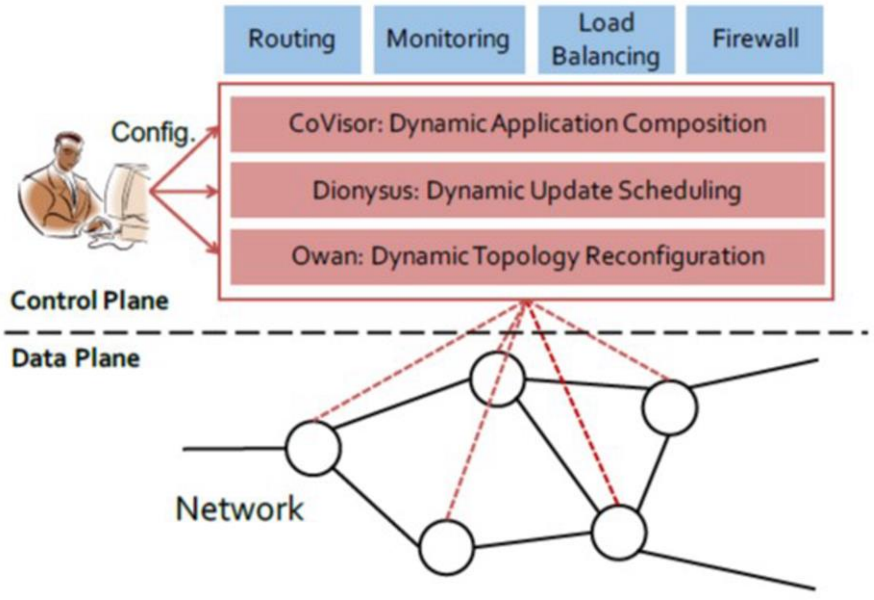
програмно-визначені мережі

Актуальність:

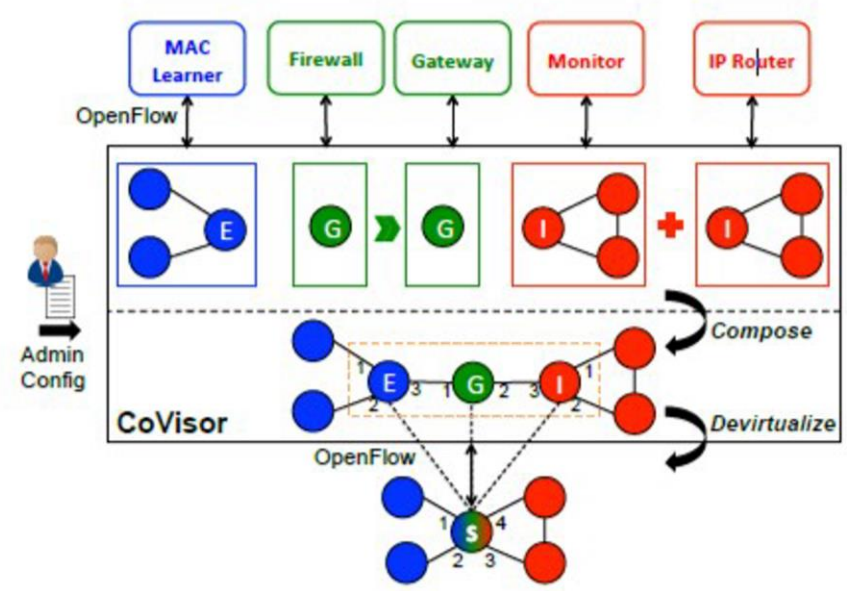
Управління мережею має вирішальне значення для надання швидких, надійних і безпечних мережевих послуг. Програмно-визначені мережі (SDN) - це нова мережева архітектура, яка спрощує управління мережею шляхом інтеграції управління мережею в централізовану платформу управління.

Актуальність теми обумовлена постійним зростанням обсягу трафіку в мережах, яке вимагає вдосконалення методів керування та оптимізації ресурсів для забезпечення високої швидкості передачі даних і стабільності роботи мережі. Використання програмно-визначених мереж та динамічного керування може значно поліпшити якість обслуговування користувачів та забезпечити ефективніше використання інфраструктури мережі.

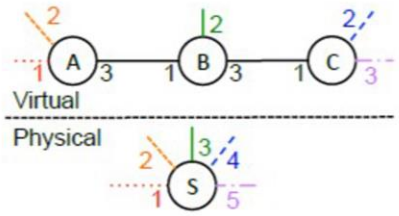
Огляд архітектури динамічного управління мережею



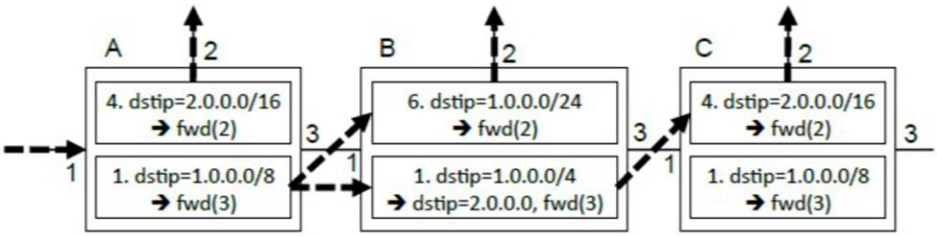
Огляд CoVisor



Віртуалізація "один до багатьох"

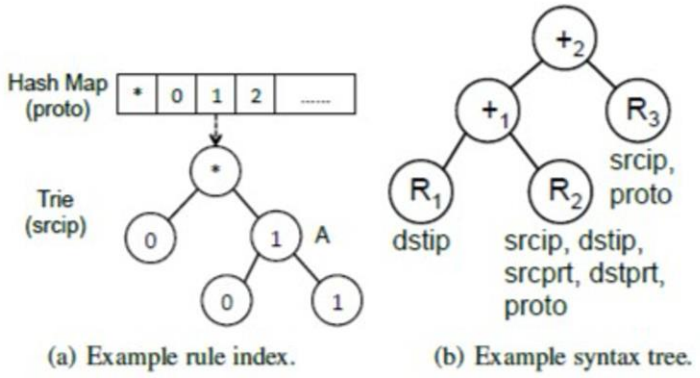


(a) Topology. Colors and line types indicate physical-virtual mapping.



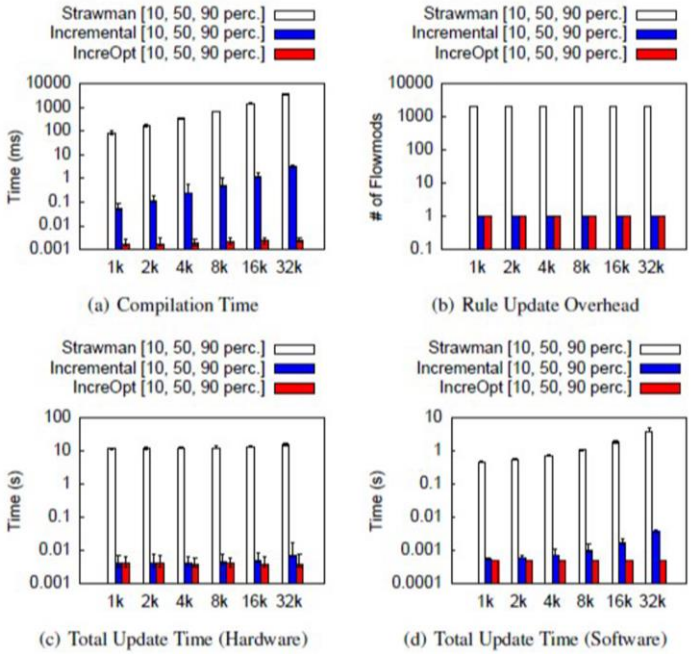
(b) Flow tables of virtual switches.

Приклад використання структур політики



(a) Example rule index. (b) Example syntax tree.

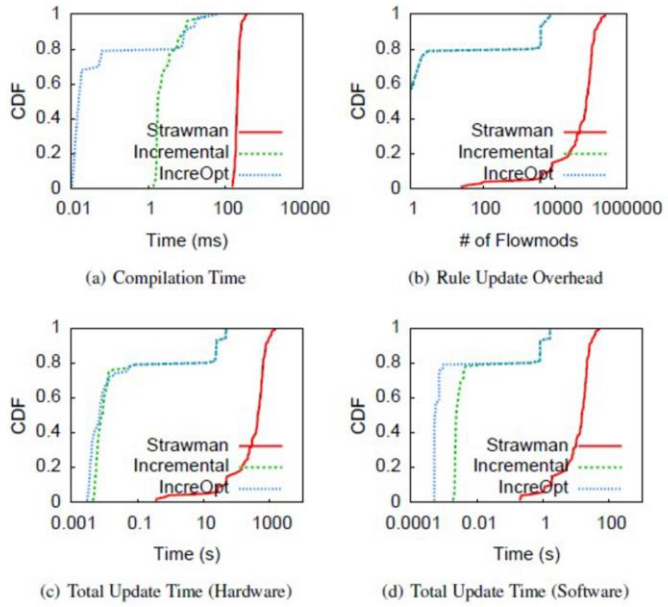
Накладні витрати на оновлення кожного правила для L2-монітора + L2-маршрутизатора як функція розміру L2-маршрутизатора



(a) Compilation Time (b) Rule Update Overhead

(c) Total Update Time (Hardware) (d) Total Update Time (Software)

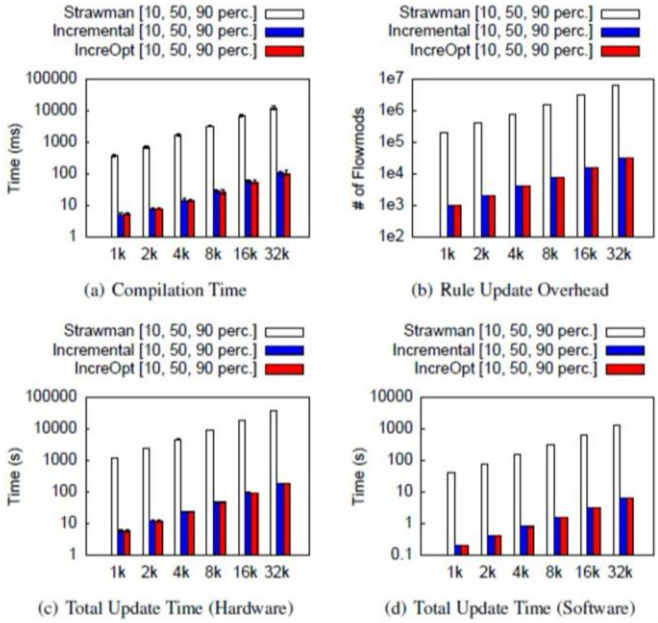
Накладні витрати на оновлення кожного правила брандмауера L3-L4 маршрутизатора L3



(a) Compilation Time (b) Rule Update Overhead

(c) Total Update Time (Hardware) (d) Total Update Time (Software)

Комутатор, що з'єднує Ethernet з ядром IP, віртуалізовано на комутаторах, які працюють як MAC-засвоювач, шлюз та IP-маршрутизатор



(a) Compilation Time (b) Rule Update Overhead

(c) Total Update Time (Hardware) (d) Total Update Time (Software)

Висновки

В роботі представлено CoVisor - композиційний гіпервізор, який дозволяє адміністраторам об'єднувати декілька контролерів для спільної обробки мережевого трафіку. CoVisor використовує комбінацію нових алгоритмів і структур даних для ефективної компіляції політик в інкрементному режимі.

Можна зробити наступні висновки:

- визначено архітектуру нового типу композиційного гіпервізора, який дозволяє додаткам, написаним різними мовами і на різних контролерах, спільно обробляти пакети.
- представлено новий алгоритм, який дозволяє компілювати паралельний, послідовний та перевизначальний оператори.
- новий, інкрементний алгоритм для компіляції політик, написаних для віртуальних топологій, у правила для фізичних комутаторів.
- використано спеціальні структури даних, які використовують обмеження контролю доступу, що часто є джерелом накладних витрат, для подальшого скорочення часу компіляції.