МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н. Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

освітньо-кваліфікаційний рівень: ***магістр***

на тему ***«Розробка багатоагентої системи динамічного управління SDN мережами»***

Виконав: студент 2 курсу, групи МФ-61 (другий магістерський рівень), спеціальності 122 «Комп'ютерні науки» освітньо-наукової програми «Інформатика»
Борейко А.О.

Керівник: доктор технічних наук, професор кафедри теоретичної та прикладної інформатики
Руккас К.М.

Рецензент: доктор технічних наук, професор кафедри математичного моделювання та штучного інтелекту НАУ «ХАІ»
Скоб Ю.О.

Харків — 2026 рік

Анотації

Борейко А.О. Розробка багатоагентної системи динамічного управління SDN мережами.

У роботі розглядається задача динамічного управління трафіком у програмно-визначених мережах (SDN) із використанням мультиагентної системи на основі навчання з підкріпленням. Запропоновано архітектуру, в якій для кожної пари комутаторів створюється окремий Q-learning агент, що обирає оптимальний маршрут на основі поточного стану мережі. Ключовою інновацією є використання відносного стану шляху, координаційного менеджера для узгодження одночасних рішень агентів та ініціалізації Q-таблиці пріором Дейкстри. Проведено порівняльний аналіз із статичним алгоритмом Дейкстри та адаптивним алгоритмом Галлагера.

Boreiko A.O. Development of a Multi-Agent System for Dynamic Management of SDN Networks.

This thesis addresses the problem of dynamic traffic management in Software-Defined Networks (SDN) using a multi-agent system based on reinforcement learning. The proposed architecture assigns an independent Q-learning agent to each switch pair, selecting optimal routes based on real-time network state. Key innovations include path-relative state representation, a coordination manager for synchronizing simultaneous agent decisions, and Dijkstra-prior Q-table initialization. A comparative analysis with static Dijkstra routing and Gallager's adaptive algorithm is presented.

Зміст

ВСТУП

Актуальність теми. Сучасні комп'ютерні мережі характеризуються зростаючою складністю та динамічністю навантаження. Програмно-визначені мережі (Software-Defined Networking, SDN) [1] забезпечують централізоване управління мережевою інфраструктурою шляхом відокремлення площини управління від площини передачі даних, що створює можливості для реалізації інтелектуальних алгоритмів маршрутизації. Проте традиційні алгоритми, зокрема алгоритм Дейкстри з урахуванням пропускної здатності, обирають статичний найкоротший шлях незалежно від поточного навантаження мережі. При одночасній передачі декількох потоків це призводить до перевантаження спільних каналів, тоді як альтернативні маршрути з достатньою ємністю залишаються невикористаними.

Адаптивні алгоритми, такі як алгоритм розподіленої маршрутизації Галлагера [2], реагують на поточне навантаження, проте не мають механізму координації одночасних рішень. Застосування методів навчання з підкріпленням [3] та мультиагентних систем [4] є перспективним напрямком для вирішення цієї задачі, оскільки дозволяє агентам навчатися оптимальній поведінці на основі взаємодії із середовищем та координувати свої дії.

Мета і завдання дослідження. Мета роботи — розробити мультиагентну систему на основі навчання з підкріпленням для динамічного управління трафіком у SDN-мережах та експериментально дослідити її ефективність порівняно з класичними алгоритмами маршрутизації.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. провести аналіз існуючих підходів до маршрутизації в SDN-мережах, включаючи статичний алгоритм Дейкстри та адаптивний алгоритм Галлагера;

2. розробити архітектуру мультиагентної системи з Q-learning агентами для кожної пари комутаторів;
3. запропонувати механізми координації агентів та представлення стану мережі, що забезпечують ефективне навчання;
4. реалізувати прототип системи у середовищі Mininet з контролером Ryu;
5. провести порівняльне тестування запропонованої системи з базовим контролером Дейкстри та контролером Галлагера.

Об'єкт дослідження — процес маршрутизації трафіку у програмно-визначених мережах з надлишковою топологією.

Предмет дослідження — мультиагентна система на основі Q-learning для динамічного розподілу трафіку між альтернативними маршрутами.

Стислий огляд відомих результатів. Задача інтелектуальної маршрутизації у SDN-мережах активно досліджується протягом останнього десятиліття. У роботі [9] запропоновано підхід на основі глибокого навчання з підкріпленням (DRL) для оптимізації маршрутизації, проте він використовує єдиного агента з глобальним станом, що обмежує масштабованість. Лін та ін. [10] розглядають RL-підхід для QoS-маршрутизації у багаторівневих SDN, але без координації між агентами. Хуанг та ін. [11] застосовують DRL для управління мультимедійним трафіком у SDN, проте фокусуються на одному потоці. Алгоритм Галлагера [2], хоча й є класичним розподіленим адаптивним рішенням, не досліджувався у контексті SDN з OpenFlow.

Відомості про одержані результати та їх новизна. Запропоновано оригінальну архітектуру мультиагентної системи для SDN-маршрутизації, що включає три ключові інновації:

1. *відносний стан шляху (path-relative state)* — агент спостерігає дискретизовану утилізацію каналів, що належать саме його кандидатним маршрутам, а

не глобальний стан мережі, що забезпечує інформативність стану;

2. *координаційний менеджер*, натхненний моделлю Галлагера: протягом часового вікна рішення агентів відстежуються, і наступний агент коригує Q-значення з урахуванням уже зайнятих каналів;

3. *ініціалізація Q-таблиці пріором Дейкстри*, що гарантує поведінку не гіршу за статичний алгоритм до накопичення достатнього досвіду.

Методи дослідження: табличний Q-learning з TD(0) оновленнями для навчання агентів маршрутизації; ε-жадібна стратегія з лінійним зменшенням для балансу дослідження та використання; алгоритм Дейкстри з урахуванням пропускної здатності для статичного baseline та ініціалізації Q-таблиці; адаптивний алгоритм Галлагера з динамічним оновленням вартості каналів для проміжного baseline; емуляція мережевого середовища у Mininet з контролером Ryu (OpenFlow 1.3); вимірювання пропускної здатності за допомогою iperf; статистичний аналіз результатів з трьома ітераціями на кожний сценарій.

Практичне значення одержаних результатів. Розроблена система може бути застосована для оптимізації розподілу трафіку у корпоративних та дата-центрових SDN-мережах з надлишковою топологією. Експериментальні результати демонструють середнє покращення пропускної здатності на $+34.9\%$ порівняно зі статичним алгоритмом Дейкстри та на $+24.2\%$ порівняно з алгоритмом Галлагера, з максимальним покращенням до $+83.6\%$ у сценаріях з конкуруючими потоками.

Структура та обсяг роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел (14 найменувань) та додатків. Розділ 1 містить огляд сучасного стану та постановку задачі. У розділі 2 описано методи дослідження та архітектуру розробленої системи. Розділ 3 присвячено результатам експериментального дослідження та їх аналізу.

Розділ 1
Огляд сучасного стану та постановка задачі

1.1. Програмно-визначені мережі

Програмно-визначені мережі (SDN) — це парадигма побудови мережевої інфраструктури, в якій площина управління (control plane) відокремлена від площини передачі даних (data plane) [1]. На відміну від традиційних мереж, де кожний комутатор самостійно приймає рішення щодо маршрутизації, в SDN ці рішення централізовано приймає контролер, який має глобальне уявлення про стан мережі.

Протокол OpenFlow [5] є стандартним інтерфейсом взаємодії між SDN-контролером та комутаторами. Контролер встановлює правила обробки пакетів (flow rules) у таблицях потоків комутаторів. Кожне правило складається з поля збігу (match field), набору дій (actions) та пріоритету. Коли пакет надходить на комутатор і не відповідає жодному правилу, він пересилається контролеру через повідомлення `PacketIn`, і контролер приймає рішення щодо маршруту.

Ключові переваги SDN-архітектури для задачі динамічної маршрутизації полягають у глобальному баченні мережі, що дозволяє контролеру отримувати повну інформацію про топологію, стан каналів і навантаження; програмованості, завдяки якій алгоритми маршрутизації реалізуються на рівні програмного забезпечення і можуть змінюватися без впливу на апаратну частину; а також у можливості моніторингу в реальному часі через механізм `PortStatsRequest/Reply`, що забезпечує регулярне отримання статистики трафіку з портів комутаторів.

1.1.1. Архітектура SDN та протокол OpenFlow

Архітектура SDN складається з трьох рівнів. *Рівень інфраструктури* (data plane) містить мережеві пристрої — комутатори, які виконують пересилання пакетів відповідно до правил, встановлених контролером. *Рівень управління* (control plane) реалізується SDN-контролером, який підтримує глобальну модель мережі та приймає рішення щодо маршрутизації. *Рівень застосувань* (application plane) містить програми, що використовують API контролера для реалізації мережевих політик, моніторингу та оптимізації.

Протокол OpenFlow [5] визначає взаємодію між контролером та комутаторами через захищений канал (TLS). Основні типи повідомлень:

- `PacketIn` — комутатор пересилає пакет контролеру, якщо жодне правило потоку не відповідає заголовку пакета;
- `FlowMod` — контролер встановлює, змінює або видаляє правила потоків у таблиці комутатора;
- `PortStatsRequest/Reply` — контролер запитує статистику портів (лічильники байтів та пакетів), що дозволяє обчислювати поточну утилізацію каналів;
- `FeaturesRequest/Reply` — контролер отримує інформацію про можливості комутатора (кількість портів, підтримувані дії).

Кожне правило потоку у таблиці комутатора складається з: полів збігу (match fields), що визначають, які пакети відповідають правилу (за MAC-адресами, IP-адресами, портами тощо); набору дій (actions), що визначають обробку пакета (пересилання на вказаний порт, відкидання, модифікація заголовків); пріоритету, що визначає порядок перевірки правил; та лічильників, що фіксують кількість оброблених пакетів та байтів.

Версія OpenFlow 1.3, що використовується у даній роботі, підтримує множинні таблиці потоків (multi-table pipeline), групові таблиці (group tables) для реалізації

multipath-маршрутизації, та вимірювальні таблиці (meter tables) для обмеження швидкості трафіку. У контексті даного дослідження ключовою є можливість встановлення правил з обмеженим часом життя (`hard_timeout`), що дозволяє автоматично видаляти застарілі правила під час навчання агентів.

1.2. Алгоритм Дейкстри з урахуванням пропускної здатності

Класичний підхід до маршрутизації в SDN-мережах полягає у побудові найкоротшого шляху за алгоритмом Дейкстри [6] з метрикою, що враховує пропускну здатність каналів. Вартість каналу між комутаторами s_i та s_j визначається як

$$c(s_i, s_j) = \frac{B_{\text{ref}}}{B(s_i, s_j)}, \tag{1.1}$$

де $B(s_i, s_j)$ — пропускна здатність каналу (біт/с), B_{ref} — опорна пропускна здатність (константа). Канали з більшою пропускною здатністю мають меншу вартість.

Алгоритм Дейкстри гарантує знаходження оптимального шляху при статичних вагах, проте не враховує поточне навантаження каналів. Якщо два потоки одночасно потребують передачі, обидва будуть спрямовані через один і той же найкоротший шлях, навіть якщо існують альтернативні маршрути з достатньою вільною ємністю.

1.3. Алгоритм Галлагера

Алгоритм розподіленої адаптивної маршрутизації, запропонований Р. Галлагером [2], є одним із перших підходів до маршрутизації з урахуванням поточного навантаження. Кожний вузол мережі підтримує оцінки затримки для кожного ви-

хідного каналу та кожного призначення. Вузли періодично обмінюються цими оцінками з сусідами та оновлюють таблиці маршрутизації методом градієнтного спуску.

У контексті SDN цей підхід адаптується так: контролер періодично отримує статистику портів та оновлює вартість каналів за формулою

$$c_G(s_i, s_j) = \frac{B_{\text{ref}}}{\max\big(B(s_i, s_j) - R(s_i, s_j),\ 0.05 \cdot B(s_i, s_j)\big)}, \tag{1.2}$$

де $R(s_i, s_j)$ — поточна швидкість передачі даних на каналі. Коли канал не навантажений, формула зводиться до статичної вартості (1.1). Коли канал завантажений, вартість зростає, і алгоритм обирає альтернативний шлях.

Однак алгоритм Галлагера не має механізму координації: якщо два потоки стартують одночасно, обидва бачать «мережу вільною» і обирають один шлях.

1.4. Навчання з підкріпленням

Задачу адаптивної маршрутизації можна формалізувати як марковський процес прийняття рішень (MDP) [3], що визначається кортежем $\langle S, A, P, R, \gamma \rangle$, де S — множина станів, A — множина дій, $P(s'|s, a)$ — функція переходу, $R(s, a)$ — функція нагороди, $\gamma \in [0, 1]$ — коефіцієнт дисконтування. Мета агента — знайти політику $\pi : S \to A$, що максимізує очікувану суму дисконтованих нагород:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \;\middle|\; s_0 = s,\ a_t = \pi(s_t)\right]. \tag{1.3}$$

Q-learning [7] оцінює функцію якості дії $Q(s, a)$ через TD(0) оновлення:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right], \tag{1.4}$$

де α — швидкість навчання, r — нагорода, s' — наступний стан.

Для балансу між дослідженням (exploration) та використанням (exploitation) застосовується ε-жадібна стратегія: з ймовірністю ε агент обирає випадкову дію, а з ймовірністю $1 - \varepsilon$ — дію з найвищим Q-значенням. Параметр ε зменшується протягом навчання, що дозволяє агенту спочатку активно досліджувати простір дій, а потім зосередитися на використанні накопиченого досвіду.

Збіжність Q-learning до оптимальної політики гарантується за умов: кожна пара стан-дія відвідується нескінченну кількість разів, швидкість навчання α_t задовольняє умовам Роббінса–Монро ($\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$), та середовище є стаціонарним [7]. У практичних застосуваннях використовується фіксоване α, що забезпечує адаптацію до нестаціонарного середовища ціною втрати гарантії точної збіжності.

Альтернативою табличному Q-learning є методи глибокого навчання з підкріпленням (Deep RL), зокрема DQN [9], де Q-функція апроксимується нейронною мережею. Перевага DQN — здатність працювати з неперервним простором станів. Проте для задач з компактним дискретним простором станів (як у даній роботі) табличний підхід є більш стабільним та інтерпретованим, оскільки Q-значення зберігаються явно та можуть бути проаналізовані.

1.4.1. Мультиагентне навчання з підкріпленням

У мультиагентному середовищі [4] головна складність — нестаціонарність: дії одного агента змінюють середовище для інших. Підхід CTDE (Centralized Training, Decentralized Execution) [8] дозволяє використовувати глобальну інформацію під час навчання, зберігаючи децентралізоване виконання.

1.5. Огляд існуючих підходів до RL-маршрутизації у SDN

Стампа та ін. [9] запропонували DRL-підхід для оптимізації маршрутизації у SDN, але з єдиним агентом та глобальним станом, що обмежує масштабованість. Лін та ін. [10] розглядають RL для QoS-маршрутизації у багаторівневих SDN без координації між агентами. Хуанг та ін. [11] застосовують DRL для мультимедійного трафіку, фокусуючись на одному потоці. Є та ін. [12] досліджують DRL для розподілу ресурсів у V2V-комунікаціях.

Аналіз літератури показує, що існуючі підходи переважно використовують єдиного агента або не мають механізму координації одночасних рішень. Це є відкритою проблемою, яка й мотивує дане дослідження.

1.6. Постановка задачі

Нехай задано SDN-мережу з множиною комутаторів $\mathcal{S} = \{s_1, \ldots, s_n\}$ та множиною каналів $\mathcal{L} \subseteq \mathcal{S} \times \mathcal{S}$, де кожний канал $(s_i, s_j) \in \mathcal{L}$ має пропускну здатність $B(s_i, s_j)$. Множина хостів $\mathcal{H}$ підключена до крайніх комутаторів. У момент часу t надходить набір потоків $\mathcal{F}^t = \{f_1, \ldots, f_k\}$, де кожний потік $f_i = (h_{\text{src}}, h_{\text{dst}})$ потребує маршруту від джерела до призначення. Для кожного потоку існує множина кандидатних маршрутів $\mathcal{P}_i = \{P_1, \ldots, P_m\}$. **Задача:** побудувати систему прийняття рішень $\pi : S \times \mathcal{P} \to P$, що для кожного потоку обирає маршрут, максимізуючи сумарну пропускну здатність мережі:

$$\max_{\pi} \sum_{f_i \in \mathcal{F}} \text{Throughput}(f_i, \pi(f_i)), \tag{1.5}$$

з урахуванням обмежень на пропускну здатність каналів та можливої одночасності потоків.

Розділ 2
Методи дослідження та розробка мультиагентної системи

2.1. Експериментальне середовище

Для дослідження розроблено топологію з 6 комутаторів та 4 хостів у емуляторі Mininet [13] з контролером Ryu [14] (OpenFlow 1.3). Mininet дозволяє створювати віртуальні мережі з реальним стеком протоколів на одному фізичному комп'ютері, що забезпечує відтворюваність експериментів. Контролер Ryu — відкрита платформа для розробки SDN-контролерів на мові Python, що підтримує механізми виявлення топології, збору статистики портів та встановлення правил потоків.

Топологія (рис. 2.1) утворює сітку з асиметричними пропускними здатностями каналів:

- верхній коридор: s_1–s_2–s_3 з пропускною здатністю 15 Мбіт/с на кожному каналі;
- нижній коридор: s_4–s_5–s_6 з пропускною здатністю 10 Мбіт/с;
- вертикальні з'єднання: s_1–s_4 та s_3–s_6 з пропускною здатністю 12 Мбіт/с;
- перехресний канал: s_2–s_5 з пропускною здатністю 8 Мбіт/с.

Хости підключені до крайніх комутаторів каналами 100 Мбіт/с, які не є вузьким місцем: $h_1 \to s_1$, $h_2 \to s_4$, $h_3 \to s_3$, $h_4 \to s_6$. Така топологія обрана з наступних міркувань: між кожною парою хостів існує від 2 до 6 альтернативних маршрутів різної пропускної здатності та довжини, що створює нетривіальну задачу балансування навантаження.

Наприклад, для пари $h_1 \to h_4$ (тобто $s_1 \to s_6$) існують маршрути:

1. $s_1 \to s_2 \to s_3 \to s_6$ — 3 хопи, bottleneck: s_3–s_6 = 12 Мбіт/с;
2. $s_1 \to s_4 \to s_5 \to s_6$ — 3 хопи, bottleneck 10 Мбіт/с;
3. $s_1 \to s_2 \to s_5 \to s_6$ — 3 хопи, bottleneck 8 Мбіт/с;
4. $s_1 \to s_4 \to s_5 \to s_2 \to s_3 \to s_6$ — 5 хопів, bottleneck 8 Мбіт/с.

Статичний алгоритм Дейкстри завжди обирає маршрут 1 як найвигідніший. Проте якщо одночасно передаються потоки $h_1 \to h_4$ та $h_3 \to h_4$, обидва будуть спрямовані через канал s_3–s_6 і ділитимуть його ємність 12 Мбіт/с. Оптимальним рішенням є розведення потоків: один через маршрут 1, інший через маршрут 2, що дає сумарну пропускну здатність до 22 Мбіт/с замість 12 Мбіт/с.

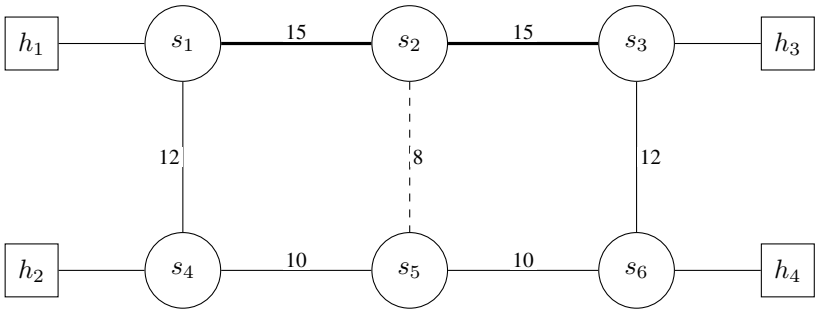

Рис. 2.1: Топологія експериментальної мережі (пропускна здатність каналів у Мбіт/с).

Для вимірювання пропускної здатності використовується утиліта iperf у режимі TCP з тривалістю кожного тесту 15 секунд. Статистика портів збирається через механізм `OFPPortStatsRequest/Reply` з періодом 3 секунди.

2.2. Загальна архітектура мультиагентної системи

Розроблена мультиагентна система складається з п'яти основних компонентів (рис. 2.2):

1. **SDN-контролер (Ryu)** — приймає повідомлення `PacketIn` від комутаторів, делегує рішення щодо маршрутизації адаптивному маршрутизатору та встановлює правила потоків через `FlowMod`.

2. **Монітор каналів (LinkMonitor)** — фоновий процес, який кожні 3 секунди надсилає запити `PortStatsRequest` до всіх комутаторів та обчислює поточну утилізацію кожного каналу за формулою:

$$U(s_i, s_j) = \frac{R(s_i, s_j)}{B(s_i, s_j)}, \tag{2.1}$$

де $R(s_i, s_j)$ — поточна швидкість передачі (біт/с), обчислена як різниця лічильників порту між двома послідовними запитами.

3. **Мультиагентна система (MultiAgentSystem)** — містить набір Q-learning агентів та керує процесом навчання. Кожному унікальному напрямку маршрутизації $(s_{\text{src}}, s_{\text{dst}})$ відповідає окремий агент.

4. **Координаційний менеджер (CoordinationManager)** — відстежує рішення агентів у межах часового вікна Δt та надає інформацію про зайняті канали наступним агентам.

5. **Адаптивний маршрутизатор (AdaptiveRouter)** — центральний компонент, що об'єднує всі підсистеми.

Взаємодія компонентів відбувається наступним чином. Коли пакет надходить на комутатор і не відповідає жодному правилу потоку, комутатор пересилає його контролеру через `PacketIn`. Контролер визначає MAC-адреси джерела та призначення, знаходить відповідні комутатори та делегує вибір маршруту адаптивному маршрутизатору. Маршрутизатор запитує у монітора каналів поточну утилізацію, будує стан для відповідного агента, отримує рішення та реєструє його у координаційному менеджері. Після цього контролер встановлює правила потоків на всіх комутаторах обраного маршруту.

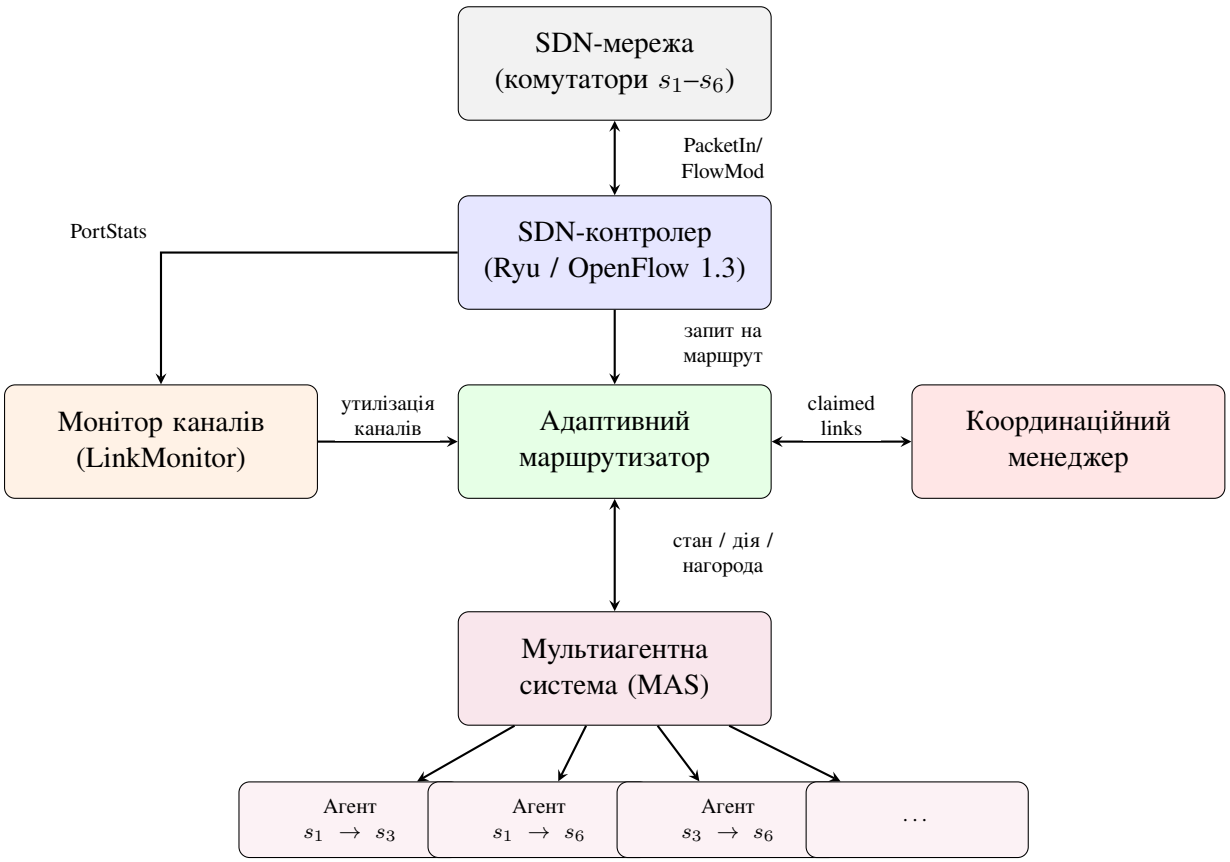

Рис. 2.2: Загальна архітектура мультиагентної системи динамічного управління SDN-мережею.

2.3. Структура окремого агента

Кожний агент маршрутизації є незалежною сутністю, що відповідає за вибір маршруту для конкретної пари комутаторів $(s_{\text{src}}, s_{\text{dst}})$. Внутрішня структура агента представлена на рис. 2.3 та включає наступні компоненти.

Кодувальник стану. Приймає на вхід утилізацію каналів від монітора та список кандидатних маршрутів від маршрутизатора. Обчислює відносний стан шляху — кортеж дискретизованих утилізацій для кожного маршруту та кількість активних потоків.

Селектор дії. Отримує стан від кодувальника, Q-значення з таблиці та координаційну інформацію. Під час навчання використовує ε-жадібну стратегію: з ймовірністю ε обирає випадкову дію, з ймовірністю $1 - \varepsilon$ — дію з найвищим Q-значенням. Під час тестування використовує координаційно-зважену експлуа-

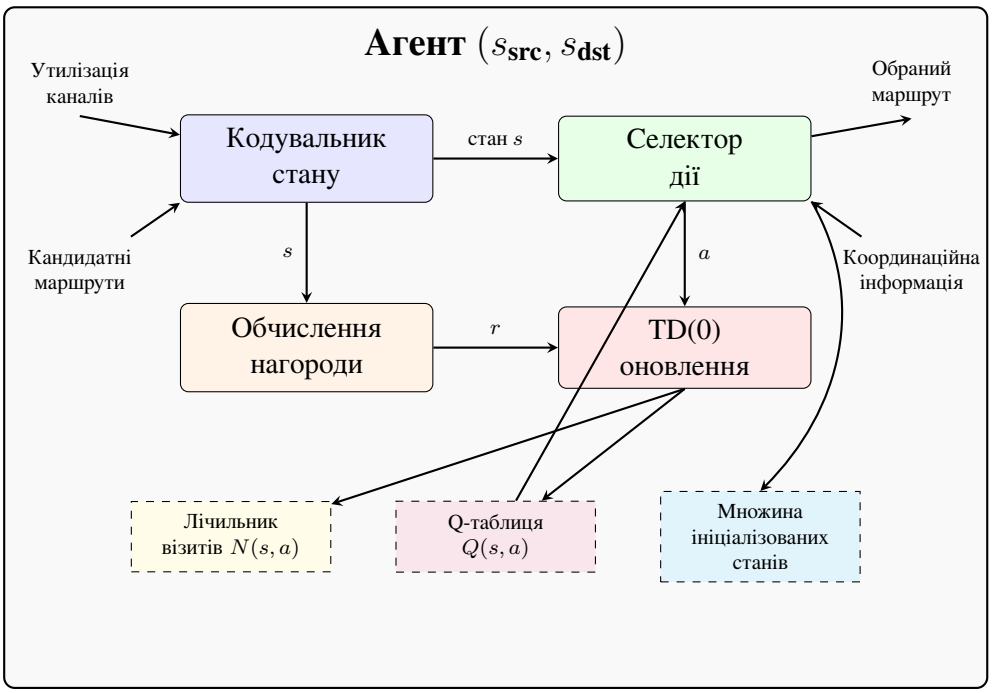

Рис. 2.3: Внутрішня структура Q-learning агента маршрутизації.

тацію за формулою (2.4).

Обчислення нагороди. Оцінює якість обраного маршруту за формулою (2.5) на основі утилізації каналів у момент прийняття рішення, довжини маршруту та ступеня перетину з іншими потоками.

TD(0) оновлення. Оновлює Q-значення за правилом (1.4). Зберігає лічильник візитів для кожної пари (s, a), який використовується для визначення достатності досвіду.

Q-таблиця. Словник виду $\{(\text{стан}, \text{дія}) \to \text{Q-значення}\}$, реалізований як вкладений `defaultdict`. Нові стани автоматично ініціалізуються пріором Дейкстри.

Множина ініціалізованих станів. Запобігає повторній ініціалізації Q-значень для вже відвіданих станів.

Агенти не обмінюються інформацією безпосередньо між собою. Координація відбувається *опосередковано* через координаційний менеджер, що відповідає парадигмі CTDE [8]: під час навчання агенти мають доступ до глобальної інформації про зайняті канали, а під час тестування кожний агент приймає рішення на основі свого локального стану та координаційної інформації від менеджера.

2.4. Відносний стан шляху

Вибір представлення стану є ключовим фактором ефективності навчання з підкріпленням. Принциповим питанням є те, яку саме інформацію про мережу агент повинен спостерігати для прийняття рішення. Глобальний стан мережі (утилізація всіх каналів) створює надто великий простір станів для табличного Q-learning, а надто агрегований стан (наприклад, лише загальний рівень завантаженості) не несе достатньо інформації для прийняття конкретного рішення.

Запропонований підхід — *відносний стан шляху* — полягає у тому, що стан прив'язується до конкретних маршрутів агента:

$$s = \big(d(u_1),\ d(u_2),\ \ldots,\ d(u_N),\ f \big), \tag{2.2}$$

де $u_i = \max\limits_{(s_j, s_k) \in P_i} U(s_j, s_k)$, $d(\cdot)$ — дискретизація (0 якщо $u < 0.3$, 1 якщо $0.3 \le u < 0.65$, 2 якщо $u \ge 0.65$), $f = \min(\text{активні потоки}, 2)$. Для агента з 4 маршрутами загальна кількість станів становить $3^4 \times 3 = 243$, що є достатньо компактною для табличного Q-learning та водночас інформативною.

Ключова перевага: стан $((0, 2, 0, 1), 1)$ читається як «маршрут 0 вільний, маршрут 1 завантажений, маршрут 2 вільний, маршрут 3 помірно завантажений, у мережі 1 активний потік». Агент обирає маршрут 0 або 2, уникаючи завантажені шляхи.

2.5. Ініціалізація Q-таблиці пріором Дейкстри

Для нових станів Q-значення ініціалізуються:

$$Q_0(s, i) = \max\big(0.2,\ 0.7 - 0.12 \cdot (|P_i| - |P_{\min}|)\big). \tag{2.3}$$

Для кратчайшого шляху $Q_0 = 0.7$, для шляху на один хоп довше — 0.58, для двох хопів довше — 0.46. Це забезпечує *безпечний старт* (поведінка не гірша за Дейкстру) та *можливість навчання* (Q-значення не заморожені).

2.6. Координаційний менеджер

Координаційний менеджер вирішує проблему одночасних конфліктних рішень. Алгоритм Дейкстри повністю ігнорує цю проблему. Алгоритм Галлагера реагує із затримкою 3 с. Координаційний менеджер вирішує її безпосередньо:

1. При надходженні запиту перевіряється, чи минуло $\Delta t = 2$ с з початку поточного раунду.
2. Якщо так — новий раунд, список зайнятих каналів очищується.
3. Якщо ні — агент отримує зайняті канали попередніх рішень.
4. Після рішення маршрут реєструється.

При тестуванні Q-значення коригуються:

$$\tilde{Q}(s,i) = Q(s,i) - 0.2 \cdot \text{overlap}(P_i, \mathcal{C}). \tag{2.4}$$

2.7. Функція нагороди

Нагорода обчислюється у момент прийняття рішення:

$$R(P) = 0.40 \cdot R_{\text{cong}} + 0.25 \cdot R_{\text{len}} + 0.35 \cdot R_{\text{share}} - \text{penalty}, \tag{2.5}$$

де:

$$R_{\text{cong}} = \max\bigl(0,\ 1 - 0.6 \cdot u_{\max} - 0.3 \cdot u_{\text{avg}}\bigr), \tag{2.6}$$

$$R_{\text{len}} = \frac{1}{1 + 0.15 \cdot (|P| - 1)}, \tag{2.7}$$

$$R_{\text{share}} = \frac{1}{1 + 0.3 \cdot n_{\text{overlap}}}, \tag{2.8}$$

$$\text{penalty} = 0.25 \cdot n_{\text{round}}. \tag{2.9}$$

Домінування $u_{\max}$ над u_{avg} (0.6 проти 0.3) обґрунтоване тим, що пропускна здатність маршруту визначається bottleneck. Для маршруту з 2 хопами $R_{\text{len}} = 0.87$, з 3 — 0.77, з 5 — 0.63.

Зауваження. Важливо обчислювати нагороду саме у момент прийняття рішення, а не після початку передачі потоку. Відкладене оцінювання створює хибний сигнал: утилізація зростає через власний потік агента, і він фактично карається за успішну передачу.

2.8. Алгоритм роботи мультиагентної системи

Повний алгоритм роботи системи при надходженні нового потоку наведено нижче.

1. Знайти кандидатні маршрути $\mathcal{P}$ (до 4 найкоротших простих шляхів).
2. Отримати від координаційного менеджера множину зайнятих каналів $\mathcal{C}$.
3. Для кожного маршруту P_i обчислити максимальну утилізацію u_i.
4. Побудувати стан $s = (d(u_1), \ldots, d(u_N), \min(f, 2))$.
5. Ініціалізувати Q-значення для стану s пріором Дейкстри (якщо новий).
6. Обрати дію:
   - *Навчання:* ε-жадібний вибір.
   - *Тестування:* $a = \arg\max_i [Q(s, i) - 0.2 \cdot \text{overlap}(P_i, \mathcal{C})]$.
7. Обчислити нагороду r за формулою (2.5).
8. Зареєструвати маршрут у координаційному менеджері.

9. Оновити Q-таблицю: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s,a') - Q(s,a)]$.

10. Встановити правила потоків на всіх комутаторах маршруту.

Часова складність одного виклику: пошук маршрутів $O(V+E)$, пошук максимуму в Q-таблиці $O(|\mathcal{P}|)$ — достатньо для роботи в реальному часі.

2.9. Процес навчання

Параметри навчання: $\alpha = 0.15$, $\gamma = 0.85$, ε: лінійне зменшення від 0.5 до 0.05 за 3000 рішень, 280 епізодів.

Програма навчання:

1. *Фаза 1* (10%): одиночні потоки — базова структура мережі.

2. *Фаза 2* (70%): конкуруючі потоки (crossing, same-destination) — навчання розведення потоків.

3. *Фаза 3* (20%): випадкові комбінації 1–3 потоків — узагальнення.

Під час навчання правила потоків мають `hard_timeout=20` с. Модель зберігається кожні 40 рішень.

Розділ 3
Результати експериментального дослідження

3.1. Методологія тестування

Порівняння проводилось для трьох контролерів:

1. **Dijkstra** — статичний контролер із вартістю каналу за формулою (1.1);
2. **Gallager** — адаптивний контролер із вартістю каналу за формулою (1.2), що оновлюється кожні 3 с;
3. **Multi-Agent RL (MAS)** — запропонована мультиагентна система.

Тестування проводилось у чотирьох сценаріях, кожний з трьома ітераціями:

1. *Single flow*: одиночний потік $h_1 \to h_3$;
2. *Competing same dst*: два потоки до одного призначення $h_1 \to h_4$ та $h_3 \to h_4$;
3. *Crossing flows*: перехресні потоки $h_1 \to h_4$ та $h_2 \to h_3$;
4. *Bidirectional*: двонаправлений потік $h_1 \leftrightarrow h_3$.

Пропускна здатність вимірювалась за допомогою iperf (TCP, 15 с на кожний тест).

3.2. Результати порівняння

Результати порівняльного тестування наведені у таблиці 3.1.

Табл. 3.1: Порівняння пропускної здатності (Мбіт/с).

Сценарій	**Dijkstra**	**Gallager**	**MAS**	**Gal/Dij**	**MAS/Dij**
Single flow	13.50	13.73	13.83	$+1.7\%$	$+2.5\%$
Competing dst	11.49	15.31	21.10	$+33.2\%$	$+83.6\%$
Crossing flows	13.81	15.14	21.11	$+9.6\%$	$+52.8\%$
Bidirectional	27.70	27.23	27.83	-1.7%	$+0.5\%$
Середнє				$\mathbf{+10.7\%}$	$\mathbf{+34.9\%}$

3.3. Аналіз результатів

Single flow ($+2.5\%$). При одиночному потоці всі три контролери обирають кратчайший шлях $s_1 \to s_2 \to s_3$, обмежений пропускною здатністю 15 Мбіт/с. Результати на рівні паритету підтверджують коректність ініціалізації Q-таблиці пріором Дейкстри.

Competing same dst ($+83.6\%$). Сценарій з найбільшим виграшем. Dijkstra направляє обидва потоки через найкоротші шляхи, що перетинаються на каналі s_3–s_6 (12 Мбіт/с), і потоки ділять цю ємність. MAS навчилась розподіляти потоки різними коридорами, використовуючи повну пропускну здатність обох шляхів. Gallager ($+33.2\%$) також реагує на завантаженість, але з затримкою 3 с: при одночасному старті обидва потоки бачать «мережу вільною». MAS вирішує це через координаційний менеджер.

Crossing flows ($+52.8\%$). Два потоки ($h_1 \to h_4$ та $h_2 \to h_3$) потенційно перетинаються на спільних каналах. MAS розводить їх різними шляхами. Gallager показує скромніший результат ($+9.6\%$) з високою дисперсією (від 11.2 до 17.2 Мбіт/с), що підтверджує нестабільність аналітичного підходу при одночасних рішеннях.

Bidirectional ($+0.5\%$). При двонаправленій передачі обидва потоки йдуть через один коридор s_1–s_2–s_3 (15 Мбіт/с у кожному напрямку), тому додаткова маршрутизація не дає переваги.

3.4. Порівняльні діаграми

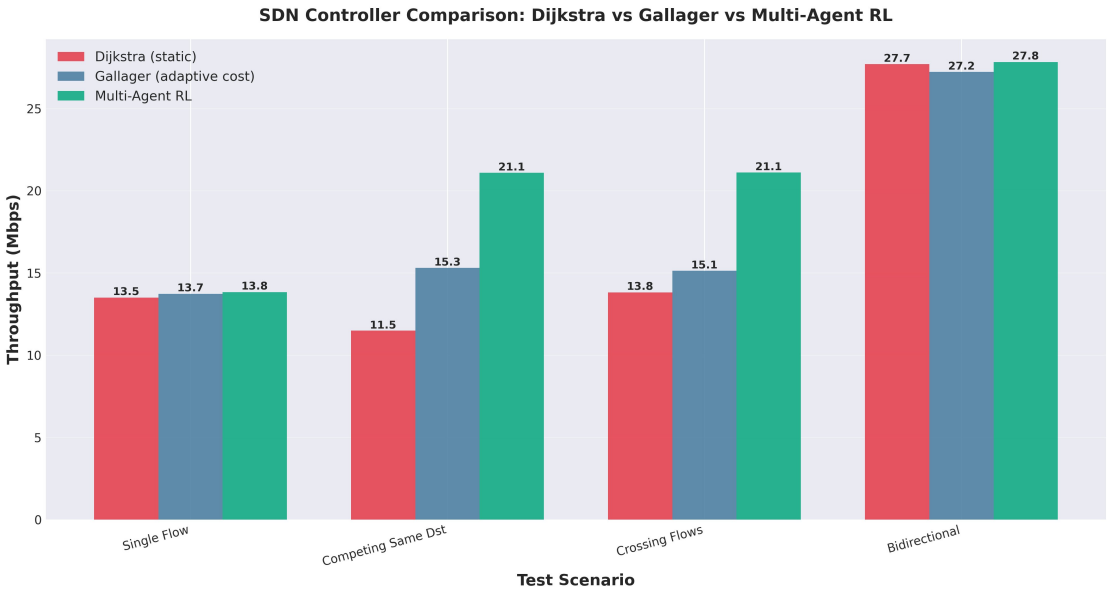

Рис. 3.1: Порівняння пропускної здатності трьох контролерів.

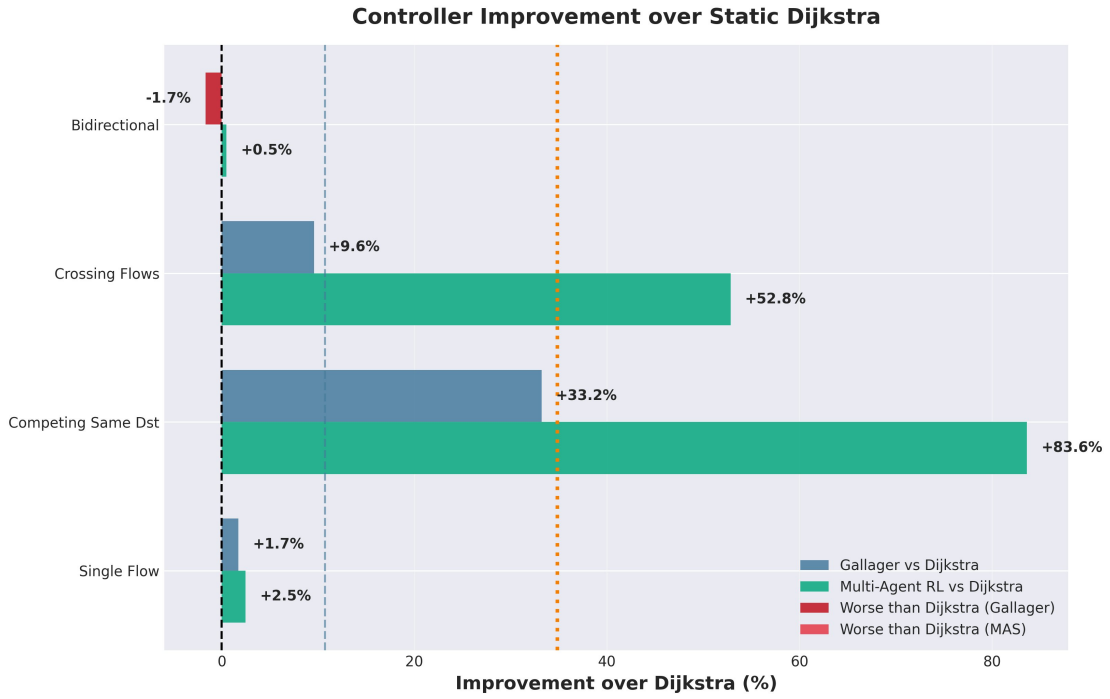

Рис. 3.2: Покращення відносно статичного алгоритму Дейкстри.

Діаграма покращення (рис. 3.2) наочно демонструє градацію підходів: Gallager дає помірне покращення завдяки адаптивності, а MAS суттєво перевершує обидва класичних алгоритми завдяки навчанню та координації.

3.5. Аналіз кривої навчання та статистика агентів

Крива навчання (рис. 3.3) демонструє осциляцію ковзного середнього між 0.2 та 0.6. Ця нестабільність пояснюється нестаціонарністю середовища навчання: нагорода залежить не тільки від дії агента, але й від типу сценарію. При одиночних потоках (10% епізодів) агент отримує нагороду ~ 0.9, при конкуруючих (70% епізодів) — нижчу через sharing_score. Чередування сценаріїв створює видиму осциляцію, але Q-таблиця при цьому сходиться, що підтверджується стабільністю результатів тестування (розкид між ітераціями < 0.1 Мбіт/с для MAS проти ~ 6 Мбіт/с для Gallager у crossing flows).

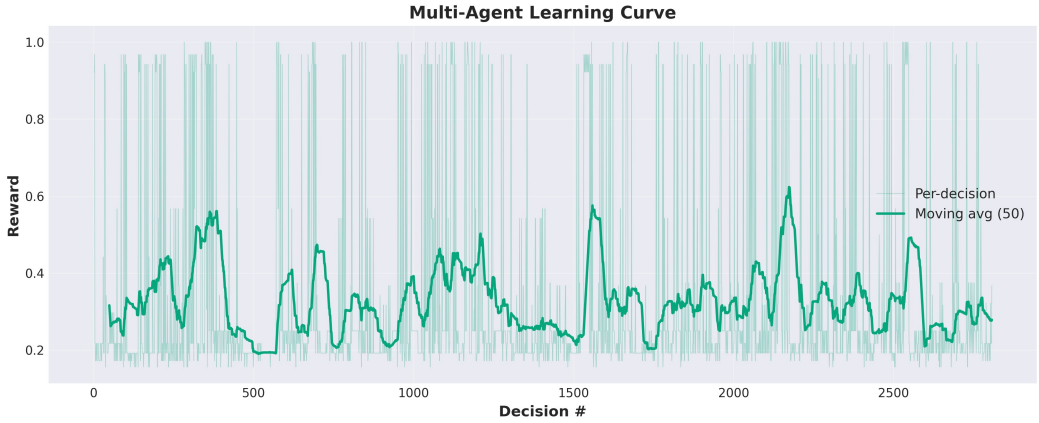

Рис. 3.3: Крива навчання мультиагентної системи.

Рис. 3.4: Q-значення та розподіл навчальних візитів за агентами.

Після навчання система містить 6 агентів з 529 записами у Q-таблицях. Мінімальне Q-значення — 0.22, максимальне — 2.85, середнє — 0.76. Ненульові мі-

німальні Q-значення підтверджують коректність ініціалізації пріором Дейкстри. Розподіл навчальних візитів (рис. 3.4) показує, що найбільш активними є агенти $s_3 \to s_6$ та $s_3 \to s_4$, що відповідає акценту на конкуруючих сценаріях у програмі навчання.

3.6. Порівняння стабільності контролерів

Окрім середніх значень пропускної здатності, важливим показником якості контролера є стабільність результатів між ітераціями тестування. У таблиці 3.2 наведено стандартне відхилення пропускної здатності для кожного сценарію.

Табл. 3.2: Стандартне відхилення пропускної здатності (Мбіт/с).

Сценарій	**Dijkstra**	**Gallager**	**MAS**
Single flow	0.35	0.12	0.09
Competing dst	0.28	0.47	0.01
Crossing flows	0.12	2.79	0.02
Bidirectional	0.08	0.12	0.16

MAS демонструє найвищу стабільність у сценаріях з конкуруючими потоками: стандартне відхилення становить лише 0.01–0.02 Мбіт/с, тоді як Gallager має відхилення до 2.79 Мбіт/с у crossing flows. Це пояснюється тим, що рішення Gallager залежать від моменту оновлення статистики: якщо обидва потоки стартують між циклами оновлення (3 с), обидва бачать однаковий стан мережі і обирають один маршрут. MAS, завдяки координаційному менеджеру, завжди розводить потоки незалежно від часу старту.

Dijkstra показує стабільні результати (відхилення 0.08–0.35 Мбіт/с), оскільки його рішення детерміновані — один і той же вхід завжди дає один маршрут. Невелике відхилення обумовлене варіацією мережевого стеку Mininet.

3.7. Аналіз ефективності використання мережевих ресурсів

Для оцінки ефективності використання мережевих ресурсів введемо коефіцієнт утилізації мережі як відношення сумарної досягнутої пропускної здатності до теоретичного максимуму:

$$\eta = \frac{\sum_{f_i \in \mathcal{F}} \text{Throughput}(f_i)}{\sum_{f_i \in \mathcal{F}} \min_{(s_j, s_k) \in P_i^*} B(s_j, s_k)}, \tag{3.1}$$

де P_i^* — оптимальний маршрут для потоку f_i за умови, що маршрути не перетинаються.

У сценарії Competing same dst теоретичний максимум при оптимальному розведенні потоків становить $12 + 10 = 22$ Мбіт/с (один потік через s_3–s_6 = 12 Мбіт/с, інший через s_4–s_5–s_6 = 10 Мбіт/с). MAS досягає 21.10 Мбіт/с, що відповідає $\eta = 95.9\%$. Dijkstra досягає лише 11.49 Мбіт/с ($\eta = 52.2\%$), a Gallager — 15.31 Мбіт/с ($\eta = 69.6\%$).

У сценарії Crossing flows оптимальне розведення дає теоретичний максимум $12+10 = 22$ Мбіт/с. MAS досягає 21.11 Мбіт/с ($\eta = 95.9\%$), Gallager — 15.14 Мбіт/с ($\eta = 68.8\%$), Dijkstra — 13.81 Мбіт/с ($\eta = 62.8\%$).

Таким чином, мультиагентна система використовує мережеві ресурси на рівні, близькому до теоретичного оптимуму, тоді як класичні алгоритми залишають значну частину ємності мережі невикористаною.

ВИСНОВКИ

У роботі розроблено мультиагентну систему для динамічного управління трафіком у програмно-визначених мережах на основі навчання з підкріпленням. Проведено комплексне дослідження, що включає теоретичний аналіз, програмну реалізацію та експериментальне порівняння з класичними алгоритмами маршрутизації.

Проведено аналіз існуючих підходів до маршрутизації в SDN-мережах. Показано, що статичний алгоритм Дейкстри не враховує поточне навантаження каналів, а адаптивний алгоритм Галлагера, хоча й реагує на завантаженість, не має механізму координації одночасних рішень, що обмежує його ефективність при одночасній передачі декількох потоків.

Запропоновано архітектуру мультиагентної системи, в якій для кожної пари комутаторів створюється незалежний Q-learning агент. Розроблено три ключові механізми: *відносний стан шляху*, що дозволяє агенту спостерігати утилізацію саме своїх кандидатних маршрутів та забезпечує інформативність стану; *координаційний менеджер*, що усуває проблему одночасних конфліктних рішень шляхом відстеження зайнятих каналів у межах часового вікна; та *ініціалізація Q-таблиці пріором Дейкстри*, що гарантує поведінку не гіршу за статичний алгоритм до накопичення достатнього досвіду.

Реалізовано прототип системи у середовищі Mininet з контролером Ryu (OpenFlow 1.3). Додатково реалізовано контролер на основі алгоритму Галлагера як проміжний baseline для тристороннього порівняння.

Проведено експериментальне порівняння у чотирьох сценаріях. Мультиагентна система продемонструвала середнє покращення пропускної здатності на $+34.9\%$

порівняно з алгоритмом Дейкстри та на $+24.2\%$ порівняно з алгоритмом Галлагера. Найбільший виграш спостерігається у сценаріях з конкуруючими потоками: $+83.6\%$ для потоків до одного призначення та $+52.8\%$ для перехресних потоків. У сценаріях без конкуренції (одиночний потік, двонаправлена передача) мультиагентна система зберігає паритет із статичним алгоритмом, що підтверджує коректність механізму ініціалізації.

Подальші дослідження можуть бути спрямовані на масштабування системи на топології більшого розміру із застосуванням глибокого навчання з підкріпленням для роботи з неперервним простором станів, додавання механізму переналаштування маршрутів для активних потоків при зміні навантаження, дослідження стійкості системи до відмов каналів та комутаторів, а також застосування глобальної функції нагороди для покращення координації агентів.

Список використаних джерел

[1] Kreutz D., Ramos F. M. V., Verissimo P., Rothenberg C. E., Azodolmolky S., Uhlig S. Software-Defined Networking: A Comprehensive Survey // Proceedings of the IEEE. — 2015. — Vol. 103, No. 1. — P. 14–76.

[2] Gallager R. G. A Minimum Delay Routing Algorithm Using Distributed Computation // IEEE Transactions on Communications. — 1977. — Vol. 25, No. 1. — P. 73–85.

[3] Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. — 2nd ed. — Cambridge: MIT Press, 2018. — 552 p.

[4] Busoniu L., Babuska R., De Schutter B. Multi-Agent Reinforcement Learning: An Overview // Innovations in Multi-Agent Systems and Applications. — Berlin: Springer, 2010. — P. 183–221.

[5] McKeown N., Anderson T., Balakrishnan H., Parulkar G., Peterson L., Rexford J., Shenker S., Turner J. OpenFlow: Enabling Innovation in Campus Networks // ACM SIGCOMM Computer Communication Review. — 2008. — Vol. 38, No. 2. — P. 69–74.

[6] Dijkstra E. W. A Note on Two Problems in Connexion with Graphs // Numerische Mathematik. — 1959. — Vol. 1. — P. 269–271.

[7] Watkins C. J. C. H., Dayan P. Q-learning // Machine Learning. — 1992. — Vol. 8, No. 3–4. — P. 279–292.

[8] Lowe R., Wu Y., Tamar A., Harb J., Abbeel P., Mordatch I. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments // Advances in Neural Information Processing Systems. — 2017. — Vol. 30. — P. 6379–6390.

[9] Stampa G., Arias M., Sanchez-Charles D., Muntes-Mulero V., Cabellos A. A Deep-Reinforcement Learning Approach for Software-Defined Networking Routing Optimization // arXiv preprint arXiv:1709.07080. — 2017.

[10] Lin S. C., Akyildiz I. F., Wang P., Luo M. QoS-Aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach // Proc. IEEE Int. Conf. Services Computing (SCC). — 2016. — P. 25–33.

[11] Huang X., Yuan T., Qiao G., Ren Y. Deep Reinforcement Learning for Multimedia Traffic Control in Software Defined Networking // IEEE Network. — 2021. — Vol. 35, No. 4. — P. 320–327.

[12] Ye H., Li G. Y. Deep Reinforcement Learning for Resource Allocation in V2V Communications // IEEE Trans. Vehicular Technology. — 2019. — Vol. 68, No. 4. — P. 3163–3173.

[13] Lantz B., Heller B., McKeown N. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks // Proc. 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets). — 2010.

[14] Ryu SDN Framework Community. Ryu SDN Framework [Електронний ресурс]. — Режим доступу: https://ryu-sdn.org/ (дата звернення: 15.04.2026).

Додатки

Для реалізації системи використано Python 3, фреймворк Ryu, емулятор Mininet, бібліотеки matplotlib та NetworkX. Нижче наведені ключові фрагменти коду.

Лістинг 3.1: Кодування відносного стану шляху

```
class PathRelativeStateEncoder:
    @staticmethod
    def discretize(u):
        if u < 0.3:
            return 0   # low
        elif u < 0.65:
            return 1   # medium
        return 2       # high

    @staticmethod
    def encode(paths, monitor, adjacency, active_flow_count):
        path_utils = []
        for path in paths:
            max_util = monitor.get_path_max_utilization(
                path, adjacency)
            path_utils.append(
                PathRelativeStateEncoder.discretize(max_util))
        fc = min(active_flow_count, 2)
        return (tuple(path_utils), fc)
```

Лістинг 3.2: Q-learning агент з ініціалізацією пріором Дейкстри

```
class RouteAgent:
    def __init__(self, src, dst, alpha=0.15, gamma=0.85,
                 epsilon=0.5):
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.q_table = defaultdict(lambda: defaultdict(float))
        self.visit_count = defaultdict(lambda: defaultdict(int))

```

```
    def seed_state(self, state, paths):
        """Dijkstra prior: shorter paths get higher Q."""
        if state in self._seeded_states:
            return
        self._seeded_states.add(state)
        min_hops = min(len(p) - 1 for p in paths)
        for i, path in enumerate(paths):
            hops = len(path) - 1
            prior = max(0.2, 0.7 - 0.12 * (hops - min_hops))
            if self.q_table[state][i] == 0.0:
                self.q_table[state][i] = prior

    def update_td(self, state, action, reward,
                  next_state, num_next_actions):
        """TD(0) update."""
        old_q = self.q_table[state][action]
        max_next_q = max(
            self.q_table[next_state][a]
            for a in range(num_next_actions)
        ) if num_next_actions > 0 else 0.0
        td_target = reward + self.gamma * max_next_q
        self.q_table[state][action] = (
            old_q + self.alpha * (td_target - old_q))
        self.visit_count[state][action] += 1
```

Лістинг 3.3: Координаційний менеджер

```
class CoordinationManager:
    def __init__(self, window_sec=2.0):
        self.window = window_sec
        self.current_round_start = 0.0
        self.round_paths = []

    def begin_decision(self):
        now = time.time()
        if now - self.current_round_start > self.window:
            self.current_round_start = now
            self.round_paths = []
        return self.round_paths[:]

    def record_decision(self, path):
```

```
        self.round_paths.append(path)

    def path_overlap_count(self, candidate, existing):
        if not existing:
            return 0
        cand_links = set()
        for s1, s2 in zip(candidate[:-1], candidate[1:]):
            cand_links.add((s1, s2))
            cand_links.add((s2, s1))
        overlap = 0
        for epath in existing:
            for s1, s2 in zip(epath[:-1], epath[1:]):
                if (s1, s2) in cand_links:
                    overlap += 1
        return overlap
```

Лістинг 3.4: Функція нагороди

```
def compute_reward(self, path, existing_paths=None):
    utilizations = []
    for s1, s2 in zip(path[:-1], path[1:]):
        if s1 in self.adjacency and s2 in self.adjacency[s1]:
            port = self.adjacency[s1][s2]
            util = self.monitor.get_link_utilization(
                s1, s2, port)
            utilizations.append(util)

    link_sharing = 0
    for flow_path, _ in self.active_flows.values():
        flow_links = set(
            (a, b) for a, b in zip(
                flow_path[:-1], flow_path[1:]))
        for a, b in zip(path[:-1], path[1:]):
            if (a, b) in flow_links:
                link_sharing += 1

    max_util = max(utilizations) if utilizations else 0
    avg_util = (sum(utilizations) / len(utilizations)
                if utilizations else 0)
    hops = len(path) - 1

```

```
    congestion = max(0, 1.0 - 0.6*max_util - 0.3*avg_util)
    length = 1.0 / (1.0 + (hops - 1) * 0.15)
    sharing = 1.0 / (1.0 + link_sharing * 0.3)

    reward = 0.40*congestion + 0.25*length + 0.35*sharing
    return max(0.01, min(1.0, reward))
```

Лістинг 3.5: Адаптивна вартість каналу (контролер Галлагера)

```
def get_link_cost(self, s1, s2):
    """Load-sensitive cost: higher when congested."""
    bw = KNOWN_BANDWIDTHS.get((s1, s2), REFERENCE_BW)
    port = self.adjacency[s1].get(s2)
    if port is None:
        return REFERENCE_BW / bw
    current_rate = self.monitor.get_link_rate(s1, port)
    remaining = max(bw - current_rate, bw * 0.05)
    return REFERENCE_BW / remaining
```

Лістинг 3.6: Топологія мережі (Mininet)

```
class RedundantTopo(Topo):
    def build(self):
        h1, h2 = self.addHost('h1'), self.addHost('h2')
        h3, h4 = self.addHost('h3'), self.addHost('h4')
        s1, s2, s3 = [self.addSwitch('s%d' % i)
                      for i in range(1, 4)]
        s4, s5, s6 = [self.addSwitch('s%d' % i)
                      for i in range(4, 7)]
        # Host links (100 Mbps, not bottleneck)
        for h, s in [(h1,s1),(h2,s4),(h3,s3),(h4,s6)]:
            self.addLink(h, s, bw=100)
        self.addLink(s1, s2, bw=15)
        self.addLink(s2, s3, bw=15)
        self.addLink(s4, s5, bw=10)
        self.addLink(s5, s6, bw=10)
        self.addLink(s1, s4, bw=12)
        self.addLink(s3, s6, bw=12)
        self.addLink(s2, s5, bw=8)
```