Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів»

Виконав: ст. 2-го курсу, групи КІТС-23м
спеціальності 125– Кібербезпека та захист інформації
Освітня програма – Кібербезпека інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Шендерук О.В.
(прізвище та ініціали)

Керівник: к.т.н., доц., доцент каф. МБІС
Карпінець В.В
(прізвище та ініціали)
«____» _____ 2024 р.

Опонент: к.т.н., доц., доцент каф. ОТ
Тарновський М.Г.
(прізвище та ініціали)
«____» _____ 2024 р.

Допущено до захисту
Голова секції УБ/кафедри МБІС

Юрій ЯРЕМЧУК
"09" грудня 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека та захист інформації
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

_____ **Юрій ЯРЕМЧУК**
"20" вересня 2024 р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу студенту
Шендеруку Олегу Володимировичу
(прізвище, ім'я, по-батькові)

1. Тема роботи
Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів
Керівник роботи к.т.н., доц., доцент каф. МБІС Карпінець Василь Васильович
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "17" вересня 2024 року № 310
2. Строк подання студентом роботи за тиждень до захисту
3. Вихідні дані до роботи: Стандарти, електронні джерела, підручники та наукові статті по темі, які стосуються теми магістерської кваліфікаційної роботи.
4. Зміст текстової частини:
Для досягнення мети роботи було поставлено такі задачі: дослідити потреби організацій та користувачів у забезпеченні захищеності інформаційних систем; проаналізувати існуючі методи захисту інформації, виявити їх недоліки та визначити шляхи їх усунення. У першому розділі роботи було досліджено принципи функціонування багаторівневих систем проксі-серверів та методи маршрутизації трафіку. У другому розділі було розроблено алгоритм гібридного стохастичного перемішування маршрутів. У третьому розділі виконано реалізацію та тестування розробленої системи.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)

В першому розділі магістерської роботи 0 рисунків, в другому розділі 3 рисунка, в третьому розділі 8 рисунка.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина			
I	Карпінець В.В к.т.н., доц., доцент каф. МБІС		
II	Карпінець В.В к.т.н., доц., доцент каф. МБІС		
III	Карпінець В.В к.т.н., доц., доцент каф. МБІС		
Економічна частина			
IV	Бурєннікова Н. В. д.е.н., професор каф. ЕПВМ		

7.Дата видачі завдання 20 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1.	Визначення напряму дослідження та уточнення теми магістерської роботи	20.09.2024	31.09.2024	
2.	Аналіз предметної області, пов'язаної з обраною темою	01.10.2024	15.10.2024	
3.	Розробка концепції та структури дослідження	16.10.2024	26.10.2024	
4.	Написання тексту магістерської роботи відповідно до визначеної теми	27.10.2024	15.11.2024	
5.	Попередній захист магістерської роботи	16.11.2024	24.11.2024	
6.	Уточнення, редагування та внесення коректив до магістерської роботи	03.12.2024	06.12.2024	
7.	Захист магістерської кваліфікаційної роботи	09.12.2024	15.12.2024	

Студент _____ Шендерук
(підпис)

Керівник роботи _____ Карпінець
(підпис)

АНОТАЦІЯ

УДК: 004.056

Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів.

Магістерська кваліфікаційна робота зі спеціальності 125 – «Кібербезпека», освітня програма «Кібербезпека інформаційних технологій та систем». Вінниця: ВНТУ, 2024. 106 с.

На укр. мові. Бібліогр.: 40 назв; рис.: 16; табл. 10.

У магістерській кваліфікаційній роботі розглядається підвищення захищеності інформаційних систем за допомогою багаторівневої системи проксі-серверів, оснащеної динамічним маршрутизаційним контролем. Основою дослідження є алгоритм гібридного стохастичного перемішування маршрутів, який забезпечує адаптивність системи та підвищує її стійкість до атак.

У першому розділі здійснено аналіз теоретичних основ інформаційної безпеки. Розглянуто ключові компоненти безпеки інформаційних систем, сучасні методи маршрутизації трафіку та їх застосування у багаторівневих проксі-серверах. Проведено аналіз існуючих підходів до забезпечення захищеності інформаційних систем і виявлено недоліки статичних систем маршрутизації.

Другий розділ присвячено розробці моделі багаторівневої системи проксі-серверів, оснащеної динамічним контролем маршрутизації. Описано алгоритм гібридного стохастичного перемішування маршрутів, який поєднує випадковість та адаптивність для оптимізації роботи системи. Обґрунтовано вибір технічних засобів і підходів для реалізації системи.

У третьому розділі виконано практичну реалізацію системи. Здійснено обґрунтування вибору мови програмування та середовища розробки, реалізовано модулі гібридного маршрутизаційного алгоритму, динамічної

маршрутизації та контролю системи. Проведено тестування функціональних модулів, взаємодії системи та її продуктивності в умовах змінного навантаження.

У четвертому розділі проаналізовано економічну ефективність розробленої системи. Проведено оцінку витрат на розробку, впровадження та експлуатацію системи, а також продемонстровано її значний комерційний потенціал.

Ключові слова: інформаційна безпека, багаторівнева система проксі-серверів, динамічна маршрутизація, гібридний стохастичний алгоритм, захист інформаційних систем, кіберзагрози.

ABSTRACT

Improving the security of information systems with a multi-level proxy server system featuring dynamic routing control based on a hybrid stochastic route-shuffling algorithm.

Master's qualification work in the specialty 125 – "Cybersecurity", educational program "Cybersecurity of Information Technologies and Systems". Vinnytsia: VNTU, 2024. 106 pages.

In Ukrainian. Bibliography: 40 sources; figures: 16; tables: 10.

This master's qualification work focuses on improving the security of information systems through a multi-level proxy server system equipped with dynamic routing control. The core of the study is a hybrid stochastic route-shuffling algorithm that enhances system adaptability and resilience against attacks.

In the first chapter, theoretical foundations of information security are analyzed. Key components of information system security, modern traffic routing methods, and their application in multi-level proxy servers are discussed. A comparative analysis of existing approaches to ensuring information system security highlights the limitations of static routing systems.

The second chapter is dedicated to the development of a model for a multi-level proxy server system with dynamic routing control. The chapter describes the hybrid stochastic route-shuffling algorithm, which combines randomness and adaptability for optimizing system performance. The selection of technical tools and approaches for system implementation is substantiated.

In the third chapter, the practical implementation of the system is presented. It includes the justification for choosing the programming language and development environment, the implementation of the hybrid routing algorithm, dynamic routing, and system control modules. The system's functional modules, interaction, and performance under variable loads are thoroughly tested.

The fourth chapter evaluates the economic feasibility of the developed system. It provides an assessment of the costs associated with the development, deployment, and operation of the system, demonstrating its significant commercial potential.

Keywords: information security, multi-level proxy server system, dynamic routing, hybrid stochastic algorithm, information system protection, cyber threats.

ЗМІСТ

ВСТУП

Актуальність. У сучасних умовах стрімкого розвитку цифрових технологій безпека інформаційних систем стає одним із найважливіших завдань, особливо з огляду на зростання кіберзагроз. Інформаційні системи є ключовими для забезпечення стабільності роботи різноманітних інфраструктур — від бізнесових процесів до національної безпеки. Водночас кількість і складність атак на ці системи зростає щороку, що вимагає від розробників нових підходів до захисту.

Особливої уваги заслуговують багаторівневі системи проксі-серверів, які забезпечують анонімність, балансування навантаження та захист мережевого трафіку. Проте класичні системи проксі мають низку недоліків, таких як статичні алгоритми маршрутизації, недостатня адаптивність до змін мережевих умов та низька стійкість до цільових атак. Для підвищення ефективності цих систем необхідні новітні методи динамічної маршрутизації, які враховують поточний стан мережі та забезпечують стійкість до атак.

У цьому контексті актуальність роботи визначається необхідністю розробки методів і алгоритмів, що підвищують захищеність інформаційних систем шляхом динамічного маршрутизаційного контролю. Пропонований алгоритм гібридного стохастичного перемішування маршрутів дозволяє створити більш адаптивну систему, здатну реагувати на зміни в реальному часі та забезпечувати балансування навантаження з одночасним мінімізацій ризиків атак.

Метою роботи є підвищення захищеності інформаційних систем на основі розробки багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем, що реалізується за допомогою алгоритму гібридного стохастичного перемішування маршрутів.

Задачами дослідження є:

– Аналіз сучасних методів захисту інформаційних систем і особливостей використання багаторівневих проксі-серверів;

– Вивчення існуючих алгоритмів маршрутизації та визначення їхніх недоліків;

– Розробка алгоритму гібридного стохастичного перемішування маршрутів, здатного адаптуватися до змін умов мережі;

– Програмна реалізація та тестування модуля динамічної маршрутизації в багаторівневій системі проксі-серверів;

– Аналіз ефективності запропонованого підходу та оцінка його практичної цінності.

Об’єктом дослідження є багаторівневі системи проксі-серверів, що забезпечують захист і маршрутизацію мережевого трафіку.

Предметом дослідження є алгоритми динамічного маршрутизаційного контролю для підвищення захищеності інформаційних систем.

Новизна роботи полягає у розробці гібридного стохастичного алгоритму перемішування маршрутів, що дозволяє значно підвищити стійкість інформаційної системи до кібератак. Використання динамічного підходу до маршрутизації дозволяє адаптуватися до змін у мережевих умовах, зменшувати ймовірність успішних атак та оптимізувати розподіл навантаження між вузлами системи.

Практична цінність роботи полягає у впровадженні багаторівневої системи проксі-серверів із динамічною маршрутизацією, яка забезпечує високий рівень захищеності інформаційних систем. Результати дослідження можуть бути використані для захисту корпоративних мереж, критичних інфраструктур та забезпечення конфіденційності й цілісності переданих даних.

Апробація тези доповідей представленні на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» на тему «Аналіз існуючих алгоритмів генерації динамічно змінюваних ключів»[1].

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПІДВИЩЕННЯ ЗАХИЩЕНОСТІ ІНФОРМАЦІЙНИХ СИСТЕМ

В даному розділі розглядаються теоретичні основи, важливість і особливості сучасних підходів до забезпечення безпеки інформаційних систем, а також аналіз існуючих систем для подальшого вдосконалення захисних механізмів.

На основі аналізу буде сформовано висновки та здійснено постановку задач.

1.1 Важливість захисту інформаційних систем у сучасних умовах

У сучасному цифровому світі інформаційні системи є критично важливими компонентами для функціонування бізнесу, урядових структур, освітніх закладів і навіть особистого життя кожного індивіда. Інформаційні системи не тільки дозволяють організаціям оптимізувати свої процеси, але й сприяють збереженню та передачі даних, що лежать в основі прийняття рішень, фінансових операцій, досліджень і розробок. Однак у зв'язку з глобальною цифровізацією кількість загроз, пов'язаних із порушенням безпеки даних, стрімко зростає, що робить питання захищеності цих систем одним з найбільш нагальних.

Сучасні виклики для захисту інформаційних систем

Зі зростанням складності інформаційних систем виникає і складність атак, спрямованих на їх компрометацію. Хакери та зловмисні групи використовують удосконалені методи проникнення в системи, такі як фішинг, соціальна інженерія, атаки на основі шкідливого програмного забезпечення (малваре) та багато інших. Це вимагає застосування нових стратегій захисту, зокрема:

Конфіденційність: Забезпечення доступу до інформації виключно авторизованим особам є критично важливим завданням. Порушення конфіденційності може призвести до витоку важливих даних, таких як

фінансова інформація, комерційна таємниця або персональні дані користувачів, що, в свою чергу, може завдати значної шкоди репутації організації та призвести до серйозних фінансових втрат.

Цілісність: Для збереження точності та повноти даних необхідно забезпечити, щоб інформація, що обробляється в системі, не могла бути змінена або пошкоджена сторонніми особами. Порушення цілісності може спотворити ключові процеси організації, призвести до помилок у прийнятті рішень або негативно вплинути на надання послуг.

Доступність: Інформаційні системи повинні залишатися доступними для авторизованих користувачів, коли вони потребують доступу до даних. Відмова в доступі, спричинена DDoS-атаками або іншими збоями, може паралізувати діяльність організації, призвести до втрати доходів і навіть зруйнувати відносини з клієнтами [2].

Економічні та репутаційні наслідки

Наслідки порушень безпеки можуть бути руйнівними для організацій, включаючи фінансові втрати, репутаційний збиток, правові та регуляторні наслідки. За даними досліджень, витрати на усунення наслідків кібератак постійно зростають, оскільки компанії змушені інвестувати великі суми в системи безпеки, відновлення даних, відшкодування збитків клієнтам та штрафи за недотримання вимог регуляторів. Втрата довіри з боку клієнтів може бути ще одним серйозним наслідком для організації, що може завдати шкоди на довгостроковій основі.

Постійно зростаюча складність загроз

Сучасні загрози інформаційній безпеці відрізняються високою складністю та варіативністю, що ускладнює їх виявлення та нейтралізацію. Серед найбільш небезпечних загроз можна виділити:

Атаки з використанням шкідливого ПЗ: це програмне забезпечення, створене для інфікування комп’ютерів і отримання несанкціонованого доступу до інформації. Серед них особливо небезпечними є віруси, трояни, черв'яки, програми-вимагачі та бекдори.

Соціальна інженерія: метод маніпулювання користувачами для отримання конфіденційної інформації або доступу до системи. Зловмисники використовують довіру та психологічні особливості людини, щоб змусити її передати інформацію або здійснити дії, які можуть послабити захист [3].

DDoS-атаки: зловмисні дії, спрямовані на перевантаження серверів або мереж з метою виведення їх з ладу. DDoS-атаки завдають значної шкоди доступності інформаційних ресурсів.

Фішинг: вид шахрайства, що передбачає обман з метою отримання конфіденційної інформації, наприклад, паролів або номерів кредитних карт. Фішинг може здійснюватися через електронні листи, SMS, повідомлення в соцмережах тощо.

Новітні методи захисту інформаційних систем

Для протидії сучасним кіберзагрозам і захисту конфіденційності, цілісності та доступності інформації необхідно застосовувати новітні методи захисту, що включають:

Багаторівневий захист: застосування комплексного підходу, який включає кілька рівнів захисту, зокрема брандмауери, системи виявлення вторгнень (IDS), антивірусне ПЗ та проксі-сервера. Багаторівнева архітектура дозволяє мінімізувати можливість проникнення на різні рівні інформаційної системи.

Шифрування даних: використання криптографічних методів для захисту інформації від несанкціонованого доступу. Шифрування забезпечує конфіденційність даних як у процесі їх зберігання, так і під час передачі через мережу.

Системи ідентифікації та контролю доступу: забезпечення доступу до ресурсів тільки авторизованим користувачам. Сучасні підходи передбачають багатофакторну автентифікацію, біометричну перевірку тощо. [4]

Динамічна маршрутизація: включає зміну маршрутів для передачі даних, що значно ускладнює їх перехоплення або модифікацію зловмисниками. Такий метод особливо ефективний для проксі-систем, що

реалізують принципи стохастичного або випадкового перемішування маршрутів.

Моніторинг та аналіз аномалій: системи моніторингу здатні виявляти аномальні дії, які можуть вказувати на зловмисні дії в системі. Аналітика поведінкових моделей і профілів користувачів допомагає ідентифікувати можливі загрози ще до їх повного прояву.

Правові аспекти та вимоги до захисту інформаційних систем

Важливість захисту інформаційних систем посилюється і на законодавчому рівні, де організації зобов'язані дотримуватися ряду нормативів, що регулюють порядок обробки та захисту даних. Приклади таких нормативів включають [5]:

GDPR (General Data Protection Regulation): регламент захисту даних у ЄС, який накладає суворі вимоги щодо захисту персональних даних і передбачає значні штрафи за його порушення.

ISO/IEC 27001: міжнародний стандарт, що визначає вимоги до систем управління інформаційною безпекою (ISMS). Він включає комплекс заходів для забезпечення конфіденційності, цілісності та доступності даних.

NIST (National Institute of Standards and Technology) Cybersecurity Framework: система рекомендацій для побудови комплексної стратегії кібербезпеки, розроблена для захисту критично важливих інфраструктур США [6].

З огляду на високу відповідальність за дотримання законодавчих вимог і зростаючу кількість загроз, важливість побудови надійних і адаптивних інформаційних систем є першочерговою для організацій у різних галузях.

Роль нових технологій у підвищенні рівня захищеності

В умовах постійної еволюції кіберзагроз важливе місце займає впровадження новітніх технологій. Використання штучного інтелекту (ШІ) для аналізу трафіку, машинного навчання для побудови моделей поведінки, блокчейн-технологій для збереження даних і багаторівневих проксі-систем з

динамічною маршрутизацією забезпечує високу стійкість до атак і велику ефективність захисту [7].

У сучасних умовах захист інформаційних систем є багатогранним завданням, що охоплює технічні, правові та організаційні аспекти. Впровадження багаторівневих проксі-систем, інтеграція динамічної маршрутизації на основі стохастичних методів і використання адаптивних технологій дозволяє знизити ризики та забезпечити надійний захист інформаційних ресурсів організацій [8].

Аналіз підвищення стійкості інформаційних систем

Стійкість інформаційних систем є ключовим фактором їх надійності, особливо в умовах постійно зростаючих загроз у кіберпросторі. Під стійкістю розуміється здатність системи зберігати працездатність у разі впливу зовнішніх факторів, таких як атаки, перевантаження або збої окремих компонентів. У сучасних умовах складності мережевих архітектур потреба в забезпеченні стійкості інформаційних систем зростає, оскільки від цього залежить безперервність бізнес-процесів, захист даних і безпека користувачів.

Багаторівневі системи проксі-серверів пропонують ефективне рішення для забезпечення стійкості інформаційних систем. Їх архітектура дозволяє розподіляти навантаження між різними серверами, знижуючи ризик перевантаження окремих вузлів. Це досягається завдяки використанню адаптивних алгоритмів маршрутизації, які враховують поточний стан мережі. Такий підхід забезпечує як ефективність роботи системи, так і її здатність протистояти загрозам [9].

Важливим аспектом підвищення стійкості є динамічне керування маршрутами передачі даних. Воно дозволяє системі швидко адаптуватися до змін у мережевих умовах, таких як перевантаження чи недоступність окремих ресурсів. Використання гібридного стохастичного алгоритму дозволяє не лише оптимізувати розподіл трафіку, але й зробити маршрутизацію менш передбачуваною, що значно підвищує стійкість системи до атак, таких як DDoS.

Крім того, багаторівневий підхід забезпечує додаткові рівні захисту, створюючи бар’єри для зловмисників. У системах, які використовують цей підхід, ефективно поєднуються функції балансування навантаження, фільтрації контенту, шифрування даних і моніторингу. Така інтеграція дозволяє мінімізувати ризик збоїв та забезпечити стабільну роботу навіть у складних умовах.

Застосування багаторівневих систем із динамічним маршрутизаційним контролем забезпечує високий рівень стійкості, що є критичним для сучасних інформаційних систем. Поєднання динамічності, адаптивності та багатофункціональності дозволяє таким системам бути ефективними навіть у середовищах із високим рівнем загроз.

1.2 Критичні компоненти безпеки інформаційних систем

Інформаційна безпека інформаційних систем (ІС) є одним із ключових напрямів у сучасних дослідженнях кібербезпеки, і її ціль полягає у забезпеченні захищеності даних, оброблюваних та збережених у системах, від потенційних загроз. Основні критичні компоненти безпеки ІС, такі як конфіденційність, цілісність, доступність, автентифікація, контроль доступу, аудит і моніторинг, є базисом для формування комплексного захисту від можливих кібератак і зловмисних дій. Кожен із цих компонентів має свою функцію, спрямовану на мінімізацію специфічних видів загроз.

Конфіденційність

Конфіденційність даних є базовим аспектом безпеки, що передбачає захист інформації від доступу та розголошення неавторизованими користувачами. Конфіденційність гарантує, що тільки особи з належним рівнем допуску можуть переглядати чи змінювати дані, що особливо важливо в умовах зростання кіберзагроз. Для забезпечення конфіденційності застосовують декілька методів, основним серед яких є **шифрування [10]**.

Шифрування виконується шляхом математичного перетворення даних у форму, яка є недоступною для читання без спеціального ключа. В

теоретичному контексті процес шифрування описується як E(K,P)=CE(K, P) = CE(K,P)=C, де EEE — алгоритм шифрування, KKK — ключ, PPP — вихідний текст, а CCC — зашифрований текст. Секретність забезпечується обмеженням доступу до ключа KKK, який необхідний для зворотного перетворення (дешифрування) даних [11].

Для підвищення конфіденційності часто використовуються такі методи, як:

- **Асиметричне шифрування**, де використовуються пари відкритих і закритих ключів для збереження секретності при передачі даних.
- **Анонімізація і псевдонімізація**, що є методами приховування особистих даних для забезпечення конфіденційності, особливо в системах, що обробляють великі обсяги особистої інформації.

Ці підходи є критично важливими у захисті від атак, спрямованих на перехоплення даних, таких як прослуховування комунікацій або крадіжка особистої інформації.

Цілісність

Цілісність даних означає, що інформація зберігається та передається без змін, тобто залишаючись автентичною та незмінною від свого оригінального стану. Порушення цілісності може бути зумовлене технічними збоями, але також може бути наслідком цілеспрямованих атак, зокрема атак, спрямованих на модифікацію даних або внесення в них шкідливих змін [12].

Цілісність забезпечується застосуванням **хешування**, що є процесом створення унікального хеш-значення для кожного фрагменту даних. Формально, хеш-функція h(P)=Hh(P) = Hh(P)=H перетворює початкові дані PPP у хеш-значення HHH, яке є унікальним для певного набору даних. Будь-яка зміна у вхідних даних призведе до зміни хешу, що дозволяє одразу виявити порушення цілісності [13].

Цілісність також підтримується через механізм **цифрових підписів**, який особливо важливий при передачі даних між системами. Цифрові підписи засвідчують автентичність джерела даних та дозволяють перевірити, що

інформація не була змінена під час передачі. Іншим методом є **ведення журналу подій**, що дозволяє зберігати детальну інформацію про всі внесені зміни, забезпечуючи можливість перевірки джерела та причини кожної модифікації [14].

Доступність

Доступність забезпечує постійний доступ до інформаційних ресурсів для авторизованих користувачів і є критичним компонентом, оскільки недоступність системи може завдати серйозних збитків користувачам і організаціям. Доступність гарантується за рахунок створення архітектури, яка є стійкою до відмов та атак [15].

Методи забезпечення доступності включають:

- **Резервне копіювання** даних, що дозволяє швидко відновити інформацію у разі збоїв або атак, спрямованих на видалення чи пошкодження даних.
- **Дублювання серверів та балансування навантаження**, які забезпечують рівномірний розподіл ресурсів і захист від перевантаження системи. Наприклад, балансування навантаження забезпечує ефективний розподіл запитів між кількома вузлами, що мінімізує ризик їх перевантаження.

Коефіцієнт доступності AAA може бути розрахований як:

A=TupTtotal×100%A = \frac{T_{up}}{T_{total}} \times 100\%A=Ttotal Tup×100%

де TupT_{up}Tup — час доступності системи, а TtotalT_{total}Ttotal — загальний час роботи системи. Високий рівень доступності критично важливий для забезпечення безперебійного доступу до послуг ІС [16].

Автентифікація та контроль доступу

Автентифікація підтверджує особу користувача перед наданням йому доступу до ресурсів системи, забезпечуючи захист від несанкціонованого доступу. У системах використовуються різні методи автентифікації, серед яких найпоширенішим є парольний захист. Проте сучасні системи вимагають більш надійних методів, таких як біометрична автентифікація та двофакторна

автентифікація (2FA), де другий фактор може бути кодом, надісланим на мобільний пристрій користувача [17].

Контроль доступу доповнює автентифікацію, розмежовуючи права користувачів у системі. В теоретичному аспекті процес контролю доступу можна описати як функцію Aaccess=f(U,P,R)A_{access} = f(U, P, R)Aaccess =f(U,P,R), де UUU — ідентифікатор користувача, PPP — права доступу, а RRR — роль користувача. Ця функція є основою для побудови системи **рольового контролю доступу (RBAC)**, яка дозволяє визначати права доступу для користувачів залежно від їхньої ролі, що значно спрощує управління повноваженнями і забезпечує мінімізацію ризиків порушення доступу [18].

Аудит і моніторинг

Аудит і моніторинг є критичними для виявлення порушень у роботі системи. Аудит включає збір та аналіз даних про дії користувачів та зміну інформації, що дозволяє ідентифікувати потенційні загрози та порушення. Моніторинг, у свою чергу, дозволяє виявляти аномальні дії та відстежувати підозрілу активність у режимі реального часу [19].

Для забезпечення ефективного моніторингу використовуються системи виявлення загроз (IDS/IPS), які здатні фіксувати потенційно небезпечні дії. Вони дозволяють своєчасно інформувати адміністратора про виявлені загрози. Процес моніторингу включає **аналіз логів** та використання **систем журналювання**, що надає можливість відстеження всіх змін у системі. Ведення детальних логів про дії користувачів дозволяє забезпечити прозорість системи та виявляти будь-які порушення [20].

Комплексний підхід до забезпечення безпеки ІС вимагає інтеграції кожного з вищезазначених компонентів. Конфіденційність, цілісність, доступність, автентифікація та контроль доступу, аудит і моніторинг є критично важливими для захисту даних і забезпечення стабільного функціонування системи в умовах зростаючих кіберзагроз [21].

1.3 Огляд багаторівневих систем безпеки та їх значення

У сучасних умовах кіберзагроз, що постійно змінюються та ускладнюються, багаторівневі системи безпеки стають основною концепцією для побудови надійного захисту інформаційних систем (ІС). На відміну від одношарових підходів, які зазвичай фокусуються на захисті лише одного рівня, багаторівнева система безпеки застосовує різні механізми захисту на кількох рівнях, що дозволяє запобігти несанкціонованому доступу та знизити ризики компрометації даних. Багаторівневий підхід є критично важливим для створення адаптивних і стійких до атак систем, оскільки дозволяє захищати ІС на кожному етапі: від фізичної безпеки до контролю доступу і шифрування [22].

Основні принципи багаторівневих систем безпеки

Багаторівневі системи безпеки базуються на принципі захисту «в глибину» (defense in depth), який передбачає використання кількох рівнів захисту, щоб забезпечити максимальну безпеку інформаційних ресурсів. Кожен рівень такої системи має свої специфічні механізми, що забезпечують виявлення, запобігання та реагування на загрози. Наприклад, у багаторівневій системі можуть використовуватись мережеві екрани, системи виявлення і запобігання вторгненням, системи шифрування, а також механізми аутентифікації та моніторингу [23].

Основними принципами, на яких базується багаторівневий підхід до безпеки, є:

Стратегія ізоляції та сегментації — розподіл ресурсів на сегменти для ізоляції систем та обмеження розповсюдження загроз.

Принцип мінімізації доступу — встановлення мінімально необхідних прав доступу для кожного користувача або процесу.

Резервування і відновлення — створення механізмів для швидкого відновлення даних та доступу до них у випадку успішної атаки на один із рівнів системи.

Структура багаторівневої системи безпеки

Типова багаторівнева система безпеки складається з таких рівнів [24]:

Фізичний рівень захисту — забезпечує фізичну охорону обладнання, в якому зберігаються дані або функціонує ІС. Сюди входять контроль доступу до приміщень, відеоспостереження, використання біометричних замків тощо.

Мережевий рівень — захищає мережеві з'єднання та передавання даних між різними компонентами системи. Механізми на цьому рівні включають міжмережеві екрани, сегментацію мережі, захист від атак типу «людина посередині» (MITM) та використання віртуальних приватних мереж (VPN) для забезпечення конфіденційності.

Системний рівень — зосереджений на захисті окремих пристроїв і серверів, що входять до складу ІС. Тут використовуються антивірусне програмне забезпечення, системи виявлення та запобігання вторгнень (IDS/IPS), а також контроль доступу до системи на основі політик безпеки.

Рівень застосунків — включає захист на рівні програмного забезпечення, що виконує основні функції ІС. Основна мета цього рівня — запобігання атак, що спрямовані на використання вразливостей у програмному коді застосунків, таких як SQL-ін'єкції, атаки на міжсайтові сценарії (XSS) та інші види експлойтів [25].

Рівень даних — забезпечує безпеку даних, що обробляються і зберігаються в ІС. На цьому рівні застосовуються шифрування, контроль доступу до баз даних, а також технології резервного копіювання та відновлення даних.

Рівень управління доступом і автентифікації — забезпечує ідентифікацію та контроль доступу до ресурсів. Тут використовуються такі методи, як двофакторна автентифікація (2FA), біометричні системи і рольовий контроль доступу (RBAC).

Рівень моніторингу та аудиту — забезпечує контроль за всіма діями в системі для виявлення підозрілої активності та забезпечення дотримання

політик безпеки. Це дозволяє своєчасно реагувати на загрози і коригувати механізми захисту [26].

Значення багаторівневих систем безпеки

Багаторівневий підхід до захисту ІС має стратегічне значення, оскільки кожен рівень виступає незалежним бар'єром для різних типів загроз. Це дозволяє створювати стійкі до атак системи, де компрометація одного з рівнів не призводить до повного порушення безпеки ІС. Нижче розглянуто основні переваги багаторівневих систем безпеки.

Стійкість до атак. Завдяки послідовності та різноманіттю механізмів безпеки, багаторівневі системи забезпечують стійкість до різних типів атак. Наприклад, навіть якщо атакуючий успішно обійшов мережевий екран, він зіштовхується з додатковими рівнями захисту, такими як автентифікація на рівні застосунків чи контроль доступу до даних [27].

Адаптивність. Багаторівнева система дозволяє швидко адаптуватися до нових типів загроз шляхом оновлення конкретних рівнів або впровадження додаткових механізмів безпеки. Це особливо важливо в умовах швидкої еволюції методів атак.

Зниження ризику внутрішніх загроз. Кожен рівень захисту є додатковою перешкодою не тільки для зовнішніх, але й для внутрішніх загроз, що може бути особливо корисно для захисту від несанкціонованих дій з боку співробітників або інших осіб, що мають доступ до ІС.

Підвищення рівня контролю та нагляду. Багаторівневі системи дозволяють розподіляти відповідальність за безпеку між різними рівнями, що забезпечує ширший контроль над роботою системи та знижує ризик порушення безпеки. Наприклад, на рівні моніторингу і аудиту система постійно фіксує всі дії користувачів, що дозволяє швидко виявляти підозрілу активність.

Можливість швидкого відновлення. У випадку успішної атаки або збою на одному з рівнів, інші рівні системи безпеки залишаються дієздатними, що дозволяє зменшити наслідки інциденту та швидше відновити роботу ІС.

Загалом, багаторівневі системи безпеки є основою для забезпечення високого рівня захищеності інформаційних систем і відіграють критичну роль у сучасному кіберзахисті. Вони забезпечують багатогранний підхід до захисту ІС, що дозволяє не тільки зменшити ризик кібератак, але й забезпечити безперервний контроль і нагляд за всіма рівнями системи [28].

Однією з важливих переваг багаторівневих систем безпеки є їхня адаптивність до сучасних загроз, які еволюціонують разом з розвитком технологій. Адаптивна природа багаторівневих систем забезпечується можливістю динамічного налаштування рівнів захисту, що дозволяє оперативно реагувати на нові типи атак. Наприклад, при виникненні нових вразливостей у програмному забезпеченні багаторівнева система може бути оновлена таким чином, що лише один із рівнів візьме на себе додаткові функції, не порушуючи роботи інших рівнів.

Ключовим елементом адаптивної багаторівневої системи є аналіз загроз і виявлення аномалій у мережі, що дозволяє виявляти підозрілу активність на ранніх етапах та оперативно вживати заходів для мінімізації потенційних збитків. Такі системи використовують методи аналізу логів, мережевого трафіку і дій користувачів, щоб ідентифікувати відхилення від норми та попереджати атаки. Для цього можуть застосовуватися автоматизовані засоби на основі штучного інтелекту та машинного навчання, які здатні обробляти великі обсяги даних у реальному часі та формувати висновки на основі аналізу минулих інцидентів [29].

Окрім цього, багаторівневі системи безпеки мають можливість динамічного оновлення стратегій безпеки відповідно до змін у навколишньому середовищі. Це включає оновлення міжмережевих екранів, встановлення нових політик доступу та введення додаткових рівнів захисту у разі виникнення загроз. Здатність до оперативного оновлення є особливо

важливою у випадках поширення шкідливих програм чи вірусів, коли час реагування є вирішальним фактором для запобігання поширенню загрози.

Значення багаторівневих систем також полягає у високій рівневій ізоляції компонентів, яка забезпечує незалежність рівнів один від одного і зменшує ймовірність того, що компрометація одного рівня вплине на інші. Ця властивість дозволяє реалізовувати принцип мінімізації довіри, що є основою Zero Trust концепції. Такий підхід означає, що кожен компонент і кожен користувач проходять перевірку, незалежно від їх попереднього статусу у системі, що значно підвищує загальну безпеку [30].

З огляду на вищенаведені характеристики, багаторівнева система безпеки не тільки виконує захисні функції, але й стає активною частиною управління ризиками в інформаційних системах. Її впровадження дозволяє організаціям не лише знижувати ймовірність успішних атак, але й забезпечувати відповідність регуляторним стандартам безпеки, таким як ISO/IEC 27001, що є важливим аспектом для сучасних компаній та установ, які працюють з конфіденційними даними та критично важливими системами.

1.4 Аналіз та порівняння існуючих схожих систем проксі-серверів для захисту інформаційних ресурсів

Проксі-сервери відіграють важливу роль у забезпеченні безпеки інформаційних ресурсів, оскільки вони виконують функції посередника між користувачем та зовнішніми сервісами, маскуючи IP-адресу клієнта та контролюючи доступ до інформаційних ресурсів. Проксі-сервери, зокрема, здатні обмежувати доступ до певних сайтів, фільтрувати небажаний контент, зменшувати навантаження на мережу шляхом кешування даних, а також захищати внутрішні системи від шкідливих атак. Проте, не всі проксі-сервери забезпечують однаковий рівень захисту, і для ефективного вибору проксі-системи необхідно розуміти їх основні типи та особливості, а також провести аналіз та порівняння їх можливостей [31].

Класифікація та основні функції проксі-серверів

Проксі-сервери можна класифікувати за різними критеріями, залежно від їхніх функцій та способу використання. Загалом, основними типами проксі-серверів, які застосовуються для захисту інформаційних ресурсів, є:

Прозорі проксі-сервери. Цей тип проксі-серверів застосовується для оптимізації роботи мережі та кешування даних. Прозорі проксі-сервери не змінюють IP-адресу клієнта, через що користувач залишається ідентифікованим. Вони добре підходять для обмеження доступу до певних ресурсів або фільтрації трафіку, проте через їхню відкритість вони менш захищені від атак [32].

Анонімні проксі-сервери. Цей вид проксі-серверів приховує IP-адресу користувача, забезпечуючи додатковий рівень конфіденційності. Анонімні проксі є ефективними у випадках, коли необхідно обмежити відстеження активності користувачів, але можуть бути менш надійними у випадку складних загроз, таких як атаки типу «людина посередині» (MITM).

Реверсні проксі-сервери. Реверсні проксі-сервери є важливим інструментом для захисту серверної інфраструктури. Вони діють як посередник між зовнішніми клієнтами та внутрішніми серверами, забезпечуючи захист від атак, таких як DDoS-атаки, та знижуючи навантаження на основний сервер завдяки функції кешування. Реверсні проксі-сервери є одними з найбільш безпечних варіантів, оскільки вони повністю маскують структуру внутрішньої мережі.

Соксові (SOCKS) проксі-сервери. Ці проксі-сервери працюють на більш низькому рівні (рівень транспортного протоколу) і підходять для більш широкого спектра застосувань, таких як передача файлів, потокове відео та інші додатки, що потребують високої пропускної здатності. Проте через відсутність вбудованих механізмів фільтрації соксові проксі зазвичай не застосовуються для контролю контенту або фільтрації небажаних сайтів.

Аналіз основних систем проксі-серверів для захисту інформаційних ресурсів

Squid Proxy. Squid є популярним проксі-сервером з функціями кешування, що дозволяє підвищити ефективність мережевого трафіку шляхом збереження копій запитуваних даних. Squid забезпечує фільтрацію небажаного контенту та підтримує обмеження доступу до веб-сайтів за IP-адресою. Проте Squid обмежений у захисті від складних атак і не забезпечує належного рівня безпеки для багаторівневого захисту мереж [33].

NGINX. Цей проксі-сервер добре підходить для реверсного проксірування, оскільки здатний забезпечувати балансування навантаження і запобігання DDoS-атакам. NGINX підтримує шифрування трафіку за допомогою SSL/TLS, що забезпечує високий рівень безпеки. Крім того, NGINX підтримує модулі для фільтрації контенту та контроль доступу на основі ролей, що робить його більш універсальним рішенням порівняно з Squid.

Apache HTTP Proxy. Apache HTTP Proxy також використовується для реверсного проксірування і підтримує кешування даних. Apache Proxy забезпечує базовий захист від атак, а також інтеграцію з модулями для автентифікації користувачів та обмеження доступу до веб-ресурсів. Проте, у порівнянні з NGINX, Apache Proxy менш ефективний у контексті високонавантажених систем і менш стійкий до DDoS-атак.

Microsoft Forefront TMG (Threat Management Gateway). Microsoft TMG забезпечує багатофункціональний підхід до захисту інформаційних ресурсів, оскільки поєднує функції проксі-сервера, міжмережевого екрану та антивірусного захисту. Він підтримує фільтрацію контенту, обмеження доступу до небажаних ресурсів, а також шифрування SSL-трафіку. Microsoft TMG є надійним рішенням для корпоративних мереж, але його використання обмежується сумісністю лише з операційними системами Windows.

Для порівняння можливостей наведених систем з точки зору захисту інформаційних ресурсів, можна узагальнити їхні основні характеристики у вигляді таблиці.

Система	Тип проксі	Основні функції	Захист від DDoS	Фільтрація контенту	SSL Підтримка	Кешування
Гібритний алгоритм даної роботи	Багаторівневий	Динамічна маршрутизація, захист від атак, масштабування	Дуже високий	Так	Так	Так
Squid Proxy	Прозорий, анонімний	Кешування, обмеження доступу	Низький	Так	Так	Так
NGINX	Реверсний	Балансування навантаження, SSL	Середній	Ні	Так	Ні
Apache HTTP Proxy	Реверсний	Кешування, автентифікація	Середній	Ні	Так	Так
Microsoft TMG	Реверсний	Проксі, фаєрвол, фільтрація	Високий	Так	Так	Ні

Таблиця 1.1 – Порівняння основних систем проксі-серверів за характеристиками безпеки.

Аналізуючи існуючі рішення, можна зробити висновок, що різні типи проксі-серверів забезпечують різний рівень захищеності та можуть виконувати специфічні функції залежно від потреб користувачів. Squid Proxy підходить для невеликих мереж, де пріоритетом є кешування та базова фільтрація контенту, тоді як NGINX і Microsoft TMG забезпечують комплексний захист і підходять для середніх та великих організацій, де існує потреба у багаторівневому підході до безпеки. Apache HTTP Proxy має обмежені можливості порівняно з NGINX та Microsoft TMG, але його також можна використовувати як простий реверсний проксі для захисту невеликих мереж.

У цілому, реверсні проксі-сервери, такі як NGINX та Microsoft TMG, виявляються найефективнішими рішеннями для комплексного захисту інформаційних ресурсів, оскільки вони здатні захищати ІС від зовнішніх

загроз, запобігати перевантаженням систем та забезпечувати високий рівень конфіденційності даних [34].

1.5 Огляд методів маршрутизації та стратегії їх застосування у проксі-серверах

Методи маршрутизації у проксі-серверах відіграють фундаментальну роль у забезпеченні надійності, продуктивності та безпеки інформаційних систем. Маршрутизація визначає шлях, яким дані проходять від клієнта до сервера і назад, що має значення не тільки для швидкості і ефективності передачі даних, але й для захисту від різних видів атак, таких як атаки типу «людина посередині» (MITM) та розподілені атаки на відмову в обслуговуванні (DDoS). У цьому розділі розглядаються основні методи маршрутизації, що використовуються у проксі-серверах, а також специфічні стратегії їх застосування для підвищення рівня безпеки та надійності мереж.

Одним з основних методів маршрутизації є статична маршрутизація, яка визначає жорстко задані маршрути для передачі даних між точками мережі. Цей метод не враховує змінні умови, такі як завантаженість мережі чи наявність кращих альтернативних маршрутів, що може обмежувати його ефективність у динамічних умовах. Проте статична маршрутизація є більш безпечною у порівнянні з іншими методами, оскільки заздалегідь визначені маршрути зменшують можливість зловмисникам втрутитися в трафік. Цей підхід найчастіше застосовується у мережах з невеликою кількістю кінцевих точок або у закритих системах, де безпека пріоритетніша за гнучкість [35].

Динамічна маршрутизація, на відміну від статичної, дозволяє змінювати маршрути на основі поточного стану мережі. Динамічні маршрути автоматично оновлюються залежно від умов, наприклад, перевантаження або збоїв у частині мережі. Одним із найбільш використовуваних протоколів динамічної маршрутизації є BGP (Border Gateway Protocol), який застосовується для передачі даних між автономними системами та дозволяє обирати оптимальний шлях залежно від умов. Хоча динамічна маршрутизація

підвищує гнучкість, вона водночас може створювати певні ризики, оскільки її маршрути більш вразливі до маніпуляцій зловмисників.

Гібридна маршрутизація поєднує переваги статичних і динамічних методів. Вона дозволяє використовувати заздалегідь визначені маршрути для певних типів трафіку, а для інших, менш критичних даних — застосовувати динамічні маршрути. Цей підхід забезпечує високу гнучкість, знижуючи ризики для конфіденційних даних, і знаходить широке застосування в умовах сучасних проксі-серверів, що підтримують багаторівневі системи захисту.

Стратегії застосування методів маршрутизації у проксі-серверах

Застосування стратегій маршрутизації у проксі-серверах залежить від специфіки завдань, які вони мають виконувати. У проксі-серверах, що працюють у великих корпоративних мережах, часто використовується навантажувальне балансування. Ця стратегія передбачає розподіл трафіку між кількома проксі-серверами або кінцевими точками для забезпечення оптимального використання ресурсів і уникнення перевантажень. Наприклад, навантажувальне балансування може бути реалізоване через динамічну маршрутизацію, що дозволяє швидко змінювати маршрути, залежно від поточного стану мережі [36].

Іншою важливою стратегією є маршрутизація на основі політик (Policy-Based Routing), яка дозволяє направляти трафік на основі встановлених правил або політик, що базуються на типі трафіку, IP-адресі або навіть на порті. Ця стратегія забезпечує гнучкість, оскільки дозволяє точно контролювати маршрути для певних типів трафіку. Наприклад, для конфіденційних даних може використовуватися статичний маршрут, тоді як для менш критичних — динамічний. Такий підхід дозволяє оптимізувати використання мережевих ресурсів і забезпечує додатковий рівень захисту, оскільки маршрути встановлюються у відповідності з конкретними політиками безпеки.

Адаптивна маршрутизація є ще однією перспективною стратегією, що знаходить своє застосування у сучасних проксі-серверах. Вона передбачає постійний моніторинг стану мережі та автоматичне коригування маршрутів у

реальному часі, що дозволяє мінімізувати ризик збоїв. Адаптивна маршрутизація є особливо ефективною для захисту від атак типу DDoS, оскільки дозволяє перенаправити трафік на менш завантажені канали або сервери.

Також слід виділити маршрутизацію з використанням алгоритмів машинного навчання, яка поступово знаходить застосування у високонавантажених і високозахищених мережах. Завдяки аналізу даних про минулі атаки та аномалії, такі алгоритми можуть передбачати та запобігати потенційним загрозам, динамічно коригуючи маршрути в реальному часі. Наприклад, методи машинного навчання можуть аналізувати поведінкові патерни трафіку, виявляючи аномальні маршрути і потенційно небезпечні запити, що сприяє підвищенню безпеки мережі [37].

Переваги та обмеження методів маршрутизації у проксі-серверах

Вибір методу маршрутизації значно впливає на продуктивність та безпеку проксі-сервера. Статична маршрутизація є найбільш безпечною, оскільки її важче маніпулювати, але вона обмежена у динамічних умовах, коли зміни мережі можуть впливати на ефективність роботи. Динамічна маршрутизація, натомість, забезпечує високу гнучкість і адаптивність, що робить її ефективною для великих, розподілених мереж, але вразливою до перехоплення та маніпуляцій з маршрутом. Гібридна маршрутизація є оптимальним рішенням, оскільки дозволяє застосовувати різні методи для різних типів трафіку, що мінімізує ризики та забезпечує адаптивність мережі.

Стратегії маршрутизації, такі як навантажувальне балансування і маршрутизація на основі політик, надають додаткові можливості для оптимізації ресурсів проксі-сервера. Адаптивна маршрутизація є ефективною для забезпечення стійкості до атак, а використання алгоритмів машинного навчання дозволяє проксі-серверам працювати проактивно, виявляючи загрози до їхнього впливу на мережу.

Методи і стратегії маршрутизації є важливими компонентами у захисті інформаційних систем через проксі-сервери. Кожен метод має свої переваги та

обмеження, що слід враховувати для забезпечення максимальної ефективності та безпеки мережі. Застосування гібридних методів та інтеграція адаптивної маршрутизації дозволяють проксі-серверам оптимально працювати в умовах високої динамічності сучасних мережевих середовищ, забезпечуючи при цьому стабільний та надійний захист інформаційних ресурсів [38].

Ефективність проксі-серверів у значній мірі залежить від методів маршрутизації, які застосовуються для організації потоку даних. Розвиток кіберзагроз та зростання обсягів мережевого трафіку вимагають застосування новітніх алгоритмів маршрутизації, які здатні не лише забезпечити стабільну передачу даних, а й адаптуватися до змінних умов мережі, мінімізуючи затримки і знижуючи ризик атак. Сучасні методи маршрутизації передбачають активне використання стохастичних і гібридних алгоритмів, які поєднують статичні та динамічні підходи для оптимального вибору маршрутів.

Стохастичні та гібридні алгоритми маршрутизації

Стохастичні алгоритми використовують ймовірнісні методи для вибору маршрутів і дозволяють проксі-серверам адаптуватися до непередбачуваних умов мережі. Наприклад, при високому навантаженні на конкретний маршрут система може використовувати випадковий вибір альтернативного маршруту, що знижує ризик перевантаження та покращує балансування трафіку. Стохастичні алгоритми особливо корисні для великих мереж з постійно змінними умовами трафіку, оскільки вони здатні ефективно реагувати на зміни завантаженості мережі без необхідності втручання адміністратора.

Гібридні алгоритми поєднують стохастичні та детерміновані підходи, забезпечуючи оптимальне поєднання гнучкості та контрольованості. Застосування гібридних алгоритмів у проксі-серверах дозволяє автоматично налаштовувати маршрути залежно від типу трафіку та рівня безпеки, який вимагається для конкретного типу даних. Такий підхід є особливо корисним у середовищах з високими вимогами до безпеки, де необхідно забезпечити захист конфіденційної інформації і водночас оптимізувати продуктивність для менш критичних даних [39].

Важливість маршрутизації на основі стану мережі

Маршрутизація на основі стану мережі (stateful routing) є однією з найефективніших стратегій для проксі-серверів, оскільки вона враховує поточний стан мережі та дозволяє вибирати оптимальні маршрути на основі аналізу реального трафіку. Stateful routing передбачає збереження інформації про стан з'єднань, що дозволяє проксі-серверам автоматично оновлювати маршрути в реальному часі залежно від наявного навантаження або затримок. Ця методологія є критично важливою для забезпечення безпеки, оскільки вона дозволяє швидко реагувати на аномалії та виявляти підозрілі з'єднання, наприклад, з підозрілих IP-адрес.

Прогнозування та адаптивні алгоритми маршрутизації

Важливим аспектом сучасної маршрутизації є здатність до прогнозування трафіку. Адаптивні алгоритми маршрутизації застосовують машинне навчання для прогнозування можливих змін у мережевому навантаженні, що дозволяє проксі-серверам проактивно коригувати маршрути. Наприклад, алгоритми на основі нейронних мереж можуть використовувати минулі дані про навантаження і затримки для прогнозування майбутніх збоїв або перевантажень. Такий підхід дозволяє забезпечити стабільну роботу навіть у випадках раптового збільшення трафіку, наприклад, під час масових запитів або спроб атак на систему.

Адаптивна маршрутизація з елементами машинного навчання також забезпечує кращий захист від атак типу DDoS, оскільки система здатна швидко розпізнавати аномальні потоки трафіку і перенаправляти їх на менш завантажені або додаткові сервери, зменшуючи ризик перевантаження основних вузлів. Завдяки цьому проксі-сервери стають більш стійкими до атак і здатними забезпечити постійний доступ до ресурсів навіть за умови підвищеного навантаження.

Хмарні технології надають додаткові можливості для динамічної маршрутизації у проксі-серверах, оскільки дозволяють легко масштабувати ресурси відповідно до поточного навантаження і використовувати хмарні

сервіси для оптимального розподілу трафіку. Хмарні проксі-сервери можуть застосовувати як статичну, так і динамічну маршрутизацію, швидко адаптуючи маршрути в залежності від географічного розташування користувачів, що знижує затримки і підвищує загальну продуктивність.

Крім того, хмарні проксі-сервери забезпечують можливість реалізації багаторівневої маршрутизації з підтримкою обробки даних у різних регіонах, що є важливим для великих корпоративних мереж з глобальним охопленням. Наприклад, завдяки хмарним технологіям, трафік з однієї частини світу може бути автоматично перенаправлений через найближчий сервер, що не лише знижує навантаження на головний сервер, а й забезпечує додатковий рівень захисту від атак.

Розподілена маршрутизація є одним із найновіших підходів, який дозволяє організовувати децентралізоване управління трафіком у проксі-системах, розподіляючи обробку даних між кількома вузлами. Цей підхід забезпечує високий рівень масштабованості та безпеки, оскільки зменшує залежність від одного центрального сервера і знижує ймовірність критичних збоїв. Розподілена маршрутизація дає можливість перенаправляти трафік через різні вузли, що значно підвищує стійкість до атак і дозволяє забезпечити високий рівень захищеності навіть при високому навантаженні.

Використання розподілених систем також покращує стійкість до загроз у реальному часі, оскільки кожен вузол системи проксі-серверів може аналізувати трафік і коригувати маршрути незалежно від інших вузлів. Такий підхід дозволяє оперативно реагувати на аномалії та перерозподіляти навантаження у випадку виникнення загроз, таких як DDoS-атаки, забезпечуючи стабільну роботу мережі [40].

У проксі-серверах часто використовується багаторівневий підхід, що передбачає розподіл маршрутів залежно від рівня критичності даних. Наприклад, для захисту конфіденційної інформації можуть використовуватися заздалегідь налаштовані маршрути, які виключають можливість їхнього динамічного коригування, що знижує ризик перехоплення. Одночасно менш

критичний трафік може бути маршрутизований з використанням динамічних алгоритмів, що підвищує гнучкість системи.

Багатозадачний підхід дозволяє проксі-серверам виконувати кілька функцій одночасно, наприклад, обробляти трафік з різних джерел і забезпечувати захист даних від внутрішніх та зовнішніх загроз. Поєднання цих підходів значно покращує захищеність системи, оптимізуючи при цьому роботу мережі та знижуючи затримки.

Сучасні методи маршрутизації у проксі-серверах включають широкий спектр алгоритмів і стратегій, що дозволяють забезпечити баланс між безпекою, продуктивністю та адаптивністю мережі. Стохастичні та гібридні підходи забезпечують гнучкість, адаптивна маршрутизація дозволяє проактивно реагувати на загрози, а використання хмарних та розподілених ресурсів підвищує масштабованість та стійкість до атак. Завдяки цьому проксі-сервери з динамічним маршрутизаційним контролем можуть ефективно забезпечувати захист інформаційних ресурсів навіть в умовах постійно зростаючих кіберзагроз.

1.6 Висновки та постановка задачі

Аналізуючи критичні аспекти безпеки інформаційних систем (ІС), можна дійти висновку, що ефективний захист сучасних ІС вимагає комплексного підходу, який включає забезпечення конфіденційності, цілісності, доступності, а також автентифікацію, контроль доступу і системи моніторингу. Кожен із цих компонентів є необхідним для побудови стійкої до загроз інфраструктури, здатної протистояти як зовнішнім, так і внутрішнім загрозам.

Багаторівневі системи безпеки є найбільш ефективними у цьому контексті, оскільки дозволяють створювати незалежні рівні захисту, що надає можливість запобігти проникненню загроз навіть у разі компрометації одного з рівнів. Такий підхід забезпечує адаптивність, знижує ймовірність успішної реалізації атак і покращує контроль над безпекою мережі.

Аналіз існуючих систем проксі-серверів вказує на те, що вони виконують ключові функції для захисту інформаційних ресурсів, забезпечуючи приховування IP-адрес, обмеження доступу до небажаних ресурсів, кешування даних і контроль контенту. Системи проксі-серверів, такі як Squid, NGINX, Apache HTTP Proxy і Microsoft TMG, відрізняються за характеристиками і можливостями, що дозволяє вибрати оптимальне рішення залежно від потреб мережі. При цьому реверсні проксі-сервери, як NGINX і Microsoft TMG, є найбільш універсальними для багаторівневого підходу.

Огляд методів маршрутизації у проксі-серверах продемонстрував, що найбільш надійним є гібридний підхід, що поєднує статичні і динамічні маршрути, дозволяючи гнучко адаптуватися до змін у мережевих умовах. Стратегії, такі як навантажувальне балансування, маршрутизація на основі політик і адаптивна маршрутизація, значно підвищують ефективність і стійкість до атак. Таким чином, для забезпечення надійного рівня захисту необхідно використовувати динамічний маршрутизаційний контроль, який здатен адаптуватися до сучасних загроз і навантаження мережі.

Постановка задачі

Враховуючи проведений аналіз, основною метою даної роботи є розробка багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем, що базується на гібридному стохастичному алгоритмі. Така система має забезпечити підвищений рівень захисту інформаційних ресурсів і гнучкість у реагуванні на змінні умови мережі.

Для досягнення цієї мети необхідно вирішити такі завдання:

– Розробити вимоги до багаторівневої системи проксі-серверів, що включають специфікації захисту, вимоги до продуктивності та критерії адаптивності.

– Спроектувати структуру та функціональні компоненти багаторівневої системи, визначивши взаємодію між рівнями захисту і механізмами маршрутизації.

– Створити модель динамічної маршрутизації, засновану на гібридному стохастичному алгоритмі, який дозволить забезпечити оптимальні маршрути для трафіку залежно від стану мережі та виявлених загроз.

– Розробити модуль захисту і контролю доступу в проксі-системі, що забезпечуватиме відповідність конфіденційності та автентифікації, а також знижуватиме ризики внутрішніх та зовнішніх загроз.

– Спроектувати систему моніторингу, яка дозволить відстежувати рівень безпеки, виявляти підозрілі активності в режимі реального часу та забезпечувати підтримку протоколів звітності.

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ БАГАТОРІВНЕВОЇ СИСТЕМИ ПРОКСІ-СЕРВЕРІВ З ДИНАМІЧНИМ МАРШРУТИЗАЦІЙНИМ КОНТРОЛЕМ

В даному розділі буде розроблено модель багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем. Будуть визначені вимоги до системи, розроблено структуру та функціональні компоненти, представлено модель динамічної маршрутизації на основі гібридного стохастичного алгоритму, а також описано методи захисту, контролю доступу, моніторингу безпеки та проектування звітів для забезпечення ефективного управління системою.

2.1 Особливості розробки та вимоги до багаторівневої системи проксі-серверів

Розробка багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем потребує ретельного визначення вимог до її архітектури, функціональності, продуктивності та безпеки. Ці вимоги мають забезпечити надійну захищеність інформаційних ресурсів, здатність адаптуватися до змін у мережевому середовищі та стабільну роботу навіть за умови підвищеного навантаження. У цьому розділі буде детально окреслено вимоги до багаторівневої системи проксі-серверів, які охоплюють функціональні, технічні, безпекові та експлуатаційні аспекти [41].

Функціональні вимоги

Багаторівнева система проксі-серверів повинна забезпечувати комплексне управління трафіком та ефективний контроль доступу до інформаційних ресурсів. Основними функціональними вимогами є:

– **Фільтрація та обробка трафіку**. Система повинна бути здатною обробляти всі типи трафіку, що проходять через проксі-сервери, включаючи HTTP/HTTPS-запити, файли та інші види даних. Це передбачає фільтрацію контенту для виявлення загроз, блокування небажаних ресурсів та обмеження доступу на основі політик безпеки.

– **Кешування та оптимізація трафіку**. Система повинна забезпечувати кешування даних для зменшення навантаження на сервери та оптимізації трафіку. Кешування також сприяє швидшому доступу до часто запитуваних ресурсів, зменшуючи час очікування для користувачів.

– **Маршрутизаційний контроль**. Необхідно забезпечити динамічний контроль маршрутів для адаптації до змін у мережевому середовищі, зокрема у випадку перевантажень, збоїв або атак. Це включає можливість налаштування стохастичних і детермінованих алгоритмів, які визначатимуть оптимальний шлях для передачі даних.

– **Інтеграція з механізмами аутентифікації**. Система повинна підтримувати засоби аутентифікації користувачів, такі як парольна або багатофакторна аутентифікація (2FA), для забезпечення безпечного доступу до ресурсів. Це має мінімізувати ризик несанкціонованого доступу і забезпечити відповідність політикам безпеки організації.

– **Забезпечення високого рівня доступності**. Проксі-сервери повинні забезпечувати безперебійний доступ до інформаційних ресурсів, навіть за умови високого навантаження на систему. Це потребує застосування механізмів балансування навантаження, які дозволяють рівномірно розподіляти трафік між різними серверами.

Технічні вимоги

Технічні вимоги стосуються апаратного та програмного забезпечення, яке необхідно для стабільної роботи багаторівневої системи проксі-серверів:

– **Продуктивність**. Система повинна обробляти великий обсяг трафіку в режимі реального часу. Це передбачає використання серверів з високою пропускною здатністю, мінімальною затримкою при обробці запитів та здатністю підтримувати кілька паралельних з'єднань. У якості мінімального показника продуктивності можна встановити обробку до декількох тисяч запитів на секунду.

– **Масштабованість**. Архітектура системи повинна бути гнучкою, з можливістю горизонтального масштабування шляхом додавання додаткових серверів у випадку зростання навантаження або кількості користувачів. Це дозволить адаптувати систему до збільшення кількості запитів, не впливаючи на її ефективність.

– **Підтримка SSL/TLS-шифрування**. Підтримка шифрування трафіку є критично важливою для захисту конфіденційності даних, що передаються через проксі-сервери. Система повинна бути сумісною з сучасними протоколами шифрування (SSL/TLS), що забезпечить захищений обмін інформацією між клієнтами та серверами.

– **Сумісність з різними типами мережевих протоколів**. Проксі-система повинна підтримувати різні мережеві протоколи (HTTP, HTTPS, FTP, SOCKS тощо), що дозволить використовувати її в багатьох мережевих середовищах. Це забезпечить гнучкість та універсальність застосування системи.

– **Інтерфейси управління та моніторингу**. Система повинна включати інтуїтивно зрозумілі інтерфейси для адміністраторів, які дозволятимуть контролювати потоки трафіку, налаштовувати політики безпеки, переглядати звіти про стан системи, а також виконувати аудит і аналіз мережевих подій.

Безпекові вимоги

Вимоги до безпеки мають критичне значення для багаторівневої системи проксі-серверів, оскільки вони визначають здатність системи ефективно захищати інформаційні ресурси від зовнішніх та внутрішніх загроз:

– **Захист від атак на мережевому рівні**. Система повинна забезпечувати захист від DDoS-атак, атак типу «людина посередині» (MITM) та інших видів загроз на мережевому рівні. Це включає встановлення міжмережевих екранів, фільтрацію IP-адрес та обмеження кількості запитів для кожного користувача.

– **Контроль доступу та рольова модель**. Система повинна забезпечувати контроль доступу на основі ролей (RBAC), дозволяючи

розподіляти права доступу до інформаційних ресурсів відповідно до рівня авторизації користувачів. Це знижує ризик несанкціонованого доступу та обмежує можливості внутрішніх загроз.

– **Аудит**. Ведення журналу подій є важливою вимогою, оскільки це дозволяє відстежувати всі дії користувачів і адміністраторів, виявляти підозрілі дії та аномалії, а також забезпечує можливість відновлення після інцидентів.

– **Підтримка політик безпеки**. Система повинна забезпечувати можливість налаштування політик безпеки, таких як обмеження доступу до певних веб-ресурсів, фільтрація контенту та захист конфіденційної інформації. Це дозволить створювати комплексний захист від зовнішніх загроз.

– **Система виявлення та запобігання вторгненням (IDS/IPS)**. Проксі-сервери повинні мати інтегровану систему виявлення та запобігання вторгненням для ідентифікації загроз та автоматичного реагування на них у реальному часі. Це забезпечить захист від атак, таких як SQL-ін'єкції, XSS та інших веб-загроз.

Експлуатаційні вимоги

Експлуатаційні вимоги відображають потребу у стабільній роботі системи та зручності її управління:

– **Надійність та відмовостійкість**. Система повинна забезпечувати безперебійну роботу навіть у разі виходу з ладу окремих серверів або компонентів. Для цього слід передбачити механізми резервування, автоматичне відновлення з'єднань та регулярне створення резервних копій даних.

– **Мінімізація затримок**. Затримки під час передачі даних повинні бути мінімальними, оскільки вони можуть впливати на швидкість доступу до інформаційних ресурсів. Високий рівень продуктивності має бути досягнутий за рахунок оптимізації процесів обробки запитів і маршрутизації трафіку.

– **Простота управління та конфігурації**. Адміністратори повинні мати доступ до зручних інструментів управління, що дозволять легко налаштовувати політики безпеки, змінювати конфігурації проксі-серверів та моніторити стан системи. Це включає інтуїтивно зрозумілий інтерфейс з можливістю дистанційного доступу.

– **Гнучкість оновлення та модернізації**. Система повинна підтримувати можливість оновлення програмного забезпечення без зупинки її роботи. Це дозволить своєчасно реагувати на нові загрози та вразливості шляхом додавання нових функцій і компонентів.

– **Відповідність нормативним вимогам**. Багаторівнева система проксі-серверів повинна відповідати стандартам безпеки, таким як ISO/IEC 27001, GDPR (у випадку обробки персональних даних), а також іншим вимогам, що забезпечать належний рівень захищеності і відповідності регуляторним нормам.

Таким чином, вимоги до багаторівневої системи проксі-серверів є комплексними та враховують аспекти безпеки, продуктивності, масштабованості, надійності та зручності експлуатації. Їх реалізація дозволить створити систему, яка забезпечить високий рівень захисту інформаційних ресурсів, здатність адаптуватися до умов сучасних мереж і стабільну роботу навіть за умов підвищеного навантаження.

2.2 Структура та функціональні компоненти системи

Багаторівнева система проксі-серверів із динамічним маршрутизаційним контролем є комплексним рішенням для забезпечення захисту інформаційних ресурсів та ефективного управління трафіком. Структура системи, зображена на діаграмі, включає кілька рівнів проксі-серверів, кожен з яких виконує специфічні завдання з фільтрації, аналізу, маршрутизації, кешування та контролю доступу. Завдяки такій багаторівневій організації, система здатна забезпечити надійну безпеку та стабільну роботу в умовах високих навантажень.

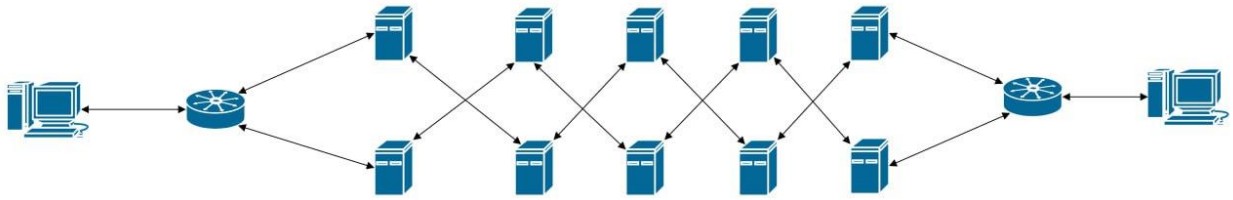


Рисунок 2.1 – Діаграма багаторівневої структури проксі-серверів

Діаграма демонструє, як багаторівнева структура проксі-серверів дозволяє обробляти трафік послідовно на кожному рівні, забезпечуючи адаптивність до змін мережевого середовища, балансування навантаження та високу продуктивність.

Вхідний вузол

На початку обробки трафіку знаходиться вхідний вузол, який приймає всі вхідні запити ззовні. Вхідний вузол виконує первинний аналіз запитів, включаючи перевірку IP-адрес, типів протоколів, заголовків пакетів і джерела запитів. Це дозволяє виявити потенційно небезпечні або небажані з'єднання, знижуючи ризик проникнення загроз на початковому етапі. Вхідний вузол також виконує розподіл запитів на перший рівень проксі-серверів, оптимізуючи навантаження на систему.

Перший рівень проксі-серверів: Фільтрація та аналіз

Після проходження через вхідний вузол, запити надходять на перший рівень проксі-серверів. Цей рівень виконує більш детальну фільтрацію та перевірку безпеки. На діаграмі видно, що перший рівень складається з декількох серверів, які паралельно обробляють запити, забезпечуючи високу пропускну здатність. На цьому етапі застосовується міжмережевий екран (файрвол), який блокує небезпечний трафік, та система виявлення вторгнень (IDS), що аналізує потоки даних у реальному часі для виявлення аномалій. Завдяки цьому рівню забезпечується захист від загроз типу DDoS, фішингових атак та шкідливого контенту.

Другий рівень проксі-серверів: Кешування та оптимізація

Запити, що пройшли первинну фільтрацію, передаються на другий рівень проксі-серверів. Цей рівень відповідає за кешування часто запитуваних ресурсів, що дозволяє знижувати навантаження на мережу і скорочувати час доступу до ресурсів. На діаграмі видно, що сервери другого рівня з'єднані з усіма попередніми та наступними рівнями, що забезпечує гнучке управління трафіком і динамічний вибір маршрутів. Кешування дозволяє зберігати копії часто використовуваних даних, що знижує затримки і підвищує ефективність роботи системи.

Третій рівень проксі-серверів: Управління доступом і автентифікація

На цьому рівні відбувається управління доступом та автентифікація користувачів. Кожен запит, який надходить на третій рівень, перевіряється на предмет автентифікації користувача за допомогою багатофакторної автентифікації. Система визначає роль користувача і надає або обмежує доступ до певних ресурсів, як це зображено на діаграмі. Це дозволяє запобігти несанкціонованому доступу до конфіденційних даних і забезпечити дотримання політик безпеки.

Вихідний вузол

Після обробки на попередніх рівнях, дані переходять до вихідного вузла, де відбувається фінальна маршрутизація і перевірка цілісності. Вихідний вузол, показаний на діаграмі, об'єднує весь трафік і забезпечує його передачу до кінцевого одержувача, гарантуючи при цьому безпеку та цілісність даних. Цей етап також включає логування всіх запитів для подальшого аналізу та аудиту.

Динамічний маршрутизаційний контроль

Система підтримує динамічний контроль маршрутів на всіх рівнях, використовуючи алгоритми стохастичної маршрутизації, що враховують поточний стан мережі, завантаженість серверів та історичні дані про трафік. Це дозволяє вибирати оптимальні маршрути, як показано на діаграмі, де

зв’язки між серверами дозволяють гнучко направляти трафік між рівнями залежно від умов мережі. У разі перевантаження одного з маршрутів система автоматично обирає альтернативний шлях, зберігаючи ефективність і стабільність роботи.

2.3 Побудова моделі динамічної маршрутизації на основі гібридного стохастичного алгоритму

Модель динамічної маршрутизації базується на гібридному стохастичному алгоритмі, який дозволяє проксі-системі адаптуватися до змін у мережі та забезпечує оптимальний розподіл трафіку. Основна мета моделі — вибір найефективнішого маршруту для передачі даних між вузлами, враховуючи поточний стан мережі, завантаження серверів та необхідність підтримки високого рівня захисту.

Модель динамічної маршрутизації розроблена з використанням гібридного підходу, що поєднує стохастичні та детерміновані методи вибору маршрутів. Стохастичний компонент забезпечує випадковий вибір маршруту з множини допустимих варіантів, що дозволяє уникнути перевантаження і забезпечити балансування навантаження. Детермінований компонент враховує попередньо зібрану статистику про трафік і стан мережі, що дозволяє зменшити ймовірність відхилень і підвищити стабільність системи.

Діаграма демонструє багаторівневу систему вузлів, де кожен сервер має кілька можливих маршрутів для передачі даних. Це дозволяє кожному вузлу динамічно вибирати оптимальний шлях залежно від поточного навантаження і затримок. Сервери на кожному рівні можуть вибирати наступний вузол у потоці передачі на основі аналізу поточного стану мережі. Наприклад, якщо певний маршрут стає перевантаженим, сервери автоматично перенаправляють трафік на менш завантажені канали (рис. 2.2).

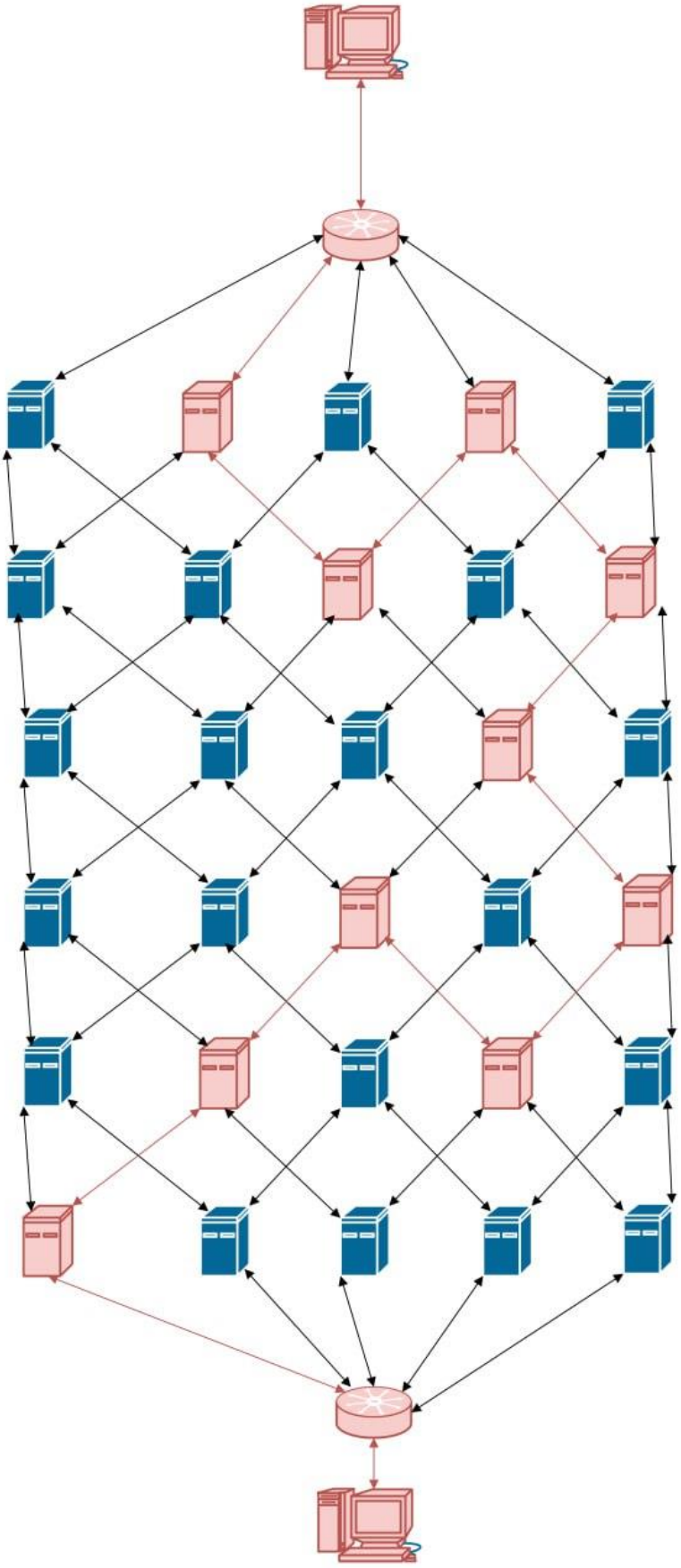


Рисунок 2.2 – Діаграма динамічної маршрутизації розроблена з використанням гібридного підходу

Основні етапи роботи гібридного стохастичного алгоритму

Ініціалізація і збір даних. На початковому етапі алгоритм здійснює збір даних про стан мережі, включаючи затримки на маршрутах, пропускну здатність каналів і обсяг трафіку. Ця інформація використовується для визначення параметрів стохастичного вибору та обмежень для кожного з маршрутів.

Стохастичний вибір маршруту. Після аналізу початкових даних система застосовує стохастичний алгоритм, що випадковим чином вибирає один із можливих маршрутів. Цей вибір базується на ймовірнісних характеристиках кожного шляху, що дозволяє уникнути перевантаження певних вузлів і забезпечити рівномірний розподіл навантаження.

Адаптивне коригування маршруту. Після первинного вибору алгоритм аналізує поточний стан маршруту в реальному часі. Якщо система виявляє, що обраний маршрут перевантажений або має високу затримку, здійснюється переналаштування маршруту. На діаграмі це видно як можливість кожного вузла перенаправляти трафік до альтернативних серверів.

Аналіз історичних даних. Детермінований компонент алгоритму використовує зібрані історичні дані про завантаження мережі та затримки, щоб передбачити майбутнє навантаження і адаптувати маршрути. Це дозволяє проактивно налаштовувати маршрути так, щоб уникати очікуваних перевантажень.

Оцінка ефективності та зворотний зв'язок. Після кожної передачі даних система оцінює ефективність обраного маршруту, зокрема затримку, пропускну здатність і надійність з’єднання. Ця інформація повертається до алгоритму, що дозволяє коригувати ймовірнісні параметри для подальшого покращення вибору маршрутів.

Динамічна маршрутизація в багаторівневій системі

Динамічна маршрутизація забезпечує адаптивність системи до змін у мережевому середовищі. Завдяки багаторівневій структурі, зображеній на діаграмі, система здатна швидко перемикатися між різними маршрутами, оптимізуючи передачу даних залежно від поточної ситуації. Кожен вузол має можливість вибору з кількох альтернативних маршрутів, що значно знижує ризик перевантаження і забезпечує високу пропускну здатність.

Наприклад, якщо трафік на певному вузлі зростає до критичного рівня, алгоритм негайно перенаправляє частину трафіку через інші маршрути, що забезпечує безперервний і стабільний потік даних. Це дозволяє системі зберігати високу продуктивність навіть за умов пікових навантажень, підтримуючи швидку і безперебійну роботу мережі.

Гібридний стохастичний підхід забезпечує такі переваги, як рівномірний розподіл навантаження, адаптивність до мережевих змін і можливість обробки великого обсягу трафіку без значних затримок. Використання стохастичних методів знижує ризик утворення «вузьких місць», а детермінована складова забезпечує стабільність і передбачуваність маршрутизації. Це створює гнучку, стійку до перевантажень систему, здатну реагувати на змінні умови у реальному часі, що робить її ідеальною для сучасних умов мережевого середовища, де трафік постійно змінюється.

Дана модель динамічної маршрутизації, побудована на основі гібридного стохастичного алгоритма, забезпечує високий рівень адаптивності та продуктивності в умовах динамічного мережевого середовища. Завдяки поєднанню стохастичних методів із детермінованими параметрами, система здатна швидко реагувати на зміни трафіку, забезпечуючи оптимальні маршрути для кожного запиту.

Як видно на діаграмі, маршрутизація відбувається через кілька рівнів серверів, що дозволяє знизити навантаження на окремі вузли та забезпечити гнучкість у виборі маршрутів. Кожен сервер має доступ до кількох можливих напрямків передачі даних, що забезпечує ефективний обхід перевантажених маршрутів та рівномірний розподіл запитів.

Загалом, ця модель сприяє зниженню затримок, підвищенню стійкості до перевантажень та забезпеченню надійного потоку даних. Гібридний стохастичний алгоритм у поєднанні з багаторівневою структурою серверів є потужним інструментом для створення надійної та ефективної мережі, яка відповідає потребам сучасних інформаційних систем.

2.4 Розробка алгоритму гібридного стохастичного перемішування маршрутів

Розробка алгоритму гібридного стохастичного перемішування маршрутів базується на аналізі основних проблем сучасних систем маршрутизації в багаторівневих проксі-серверах. До основних викликів, які вирішує даний алгоритм, належать:

– Недостатня адаптивність статичних методів маршрутизації. Сучасні методи маршрутизації часто базуються на статичних правилах або таблицях маршрутизації, які не враховують динамічні зміни в мережі. Це може призводити до нераціонального використання ресурсів або перевантаження певних вузлів системи;

– Ризик цільових атак на маршрути. У багаторівневих системах, які використовують традиційні маршрутизатори, зловмисники можуть виявляти стабільні маршрути для атак (наприклад, атаки типу DDoS). Гібридний стохастичний алгоритм вирішує цю проблему, динамічно змінюючи маршрути, що ускладнює передбачуваність їхнього використання;

– Необхідність рівномірного розподілу навантаження. У системах із високим трафіком необхідно забезпечити балансування навантаження між серверами для уникнення їх перевантаження. Гібридний підхід дозволяє враховувати поточну завантаженість серверів і динамічно коригувати ймовірності вибору маршрутів;

– Висока вимогливість до обчислювальних ресурсів. Сучасні адаптивні алгоритми можуть вимагати значних обчислювальних ресурсів. Гібридний

підхід поєднує стохастичність із спрощеними обчисленнями, що робить його ефективним навіть у системах з обмеженими ресурсами.

Принципи, що лежать в основі алгоритму:

– Замість жорсткого визначення маршруту використовується ймовірнісний підхід, де кожен маршрут має певну вагу. Це дозволяє уникнути одноманітності в трафіку, а також підвищує стійкість до атак;

– Алгоритм враховує поточну завантаженість серверів та інші параметри мережі (наприклад, затримки), динамічно змінюючи ймовірності вибору маршрутів. Це забезпечує рівномірний розподіл трафіку та підвищує загальну продуктивність системи;

– Алгоритм аналізує історію використання маршрутів для прогнозування їхньої ефективності. Це дозволяє мінімізувати повторне використання маршрутів, які призводять до затримок чи перевантажень.

Перед початком побудови моделі гібридного стохастичного перемішування маршрутів розробимо алгоритм, який дозволить забезпечити адаптивне керування маршрутизацією в багаторівневій системі проксі-серверів (рис. 2.3). Головною метою розробки цього алгоритму є створення механізму, який враховуватиме поточний стан мережі, дозволяючи рівномірно розподіляти трафік, уникати перевантажених каналів і зберігати стабільну пропускну здатність системи навіть за умов високих навантажень. Алгоритм поєднує стохастичні методи випадкового вибору маршруту з детермінованими рішеннями на основі історичних даних, що дозволяє як підтримувати випадковість у виборі маршрутів, так і контролювати їх ефективність на основі попереднього досвіду.

Після розробки алгоритму розберемо його основні компоненти та етапи виконання для глибшого розуміння процесу динамічної маршрутизації. Зокрема, особлива увага приділяється адаптивному коригуванню маршрутів у режимі реального часу, що дозволяє швидко реагувати на зміни в мережі. Додатково алгоритм використовує ймовірнісні ваги для кожного маршруту,

які оновлюються відповідно до поточних умов мережі. Це забезпечує можливість проксі-серверів швидко адаптуватися до перевантажень або змін у стані мережі, обираючи найкращі доступні маршрути для передачі трафіку. Завдяки такій адаптивності, система стає більш стійкою до раптових стрибків навантаження і здатна забезпечувати надійну передачу даних навіть в умовах підвищеної мережевої активності.

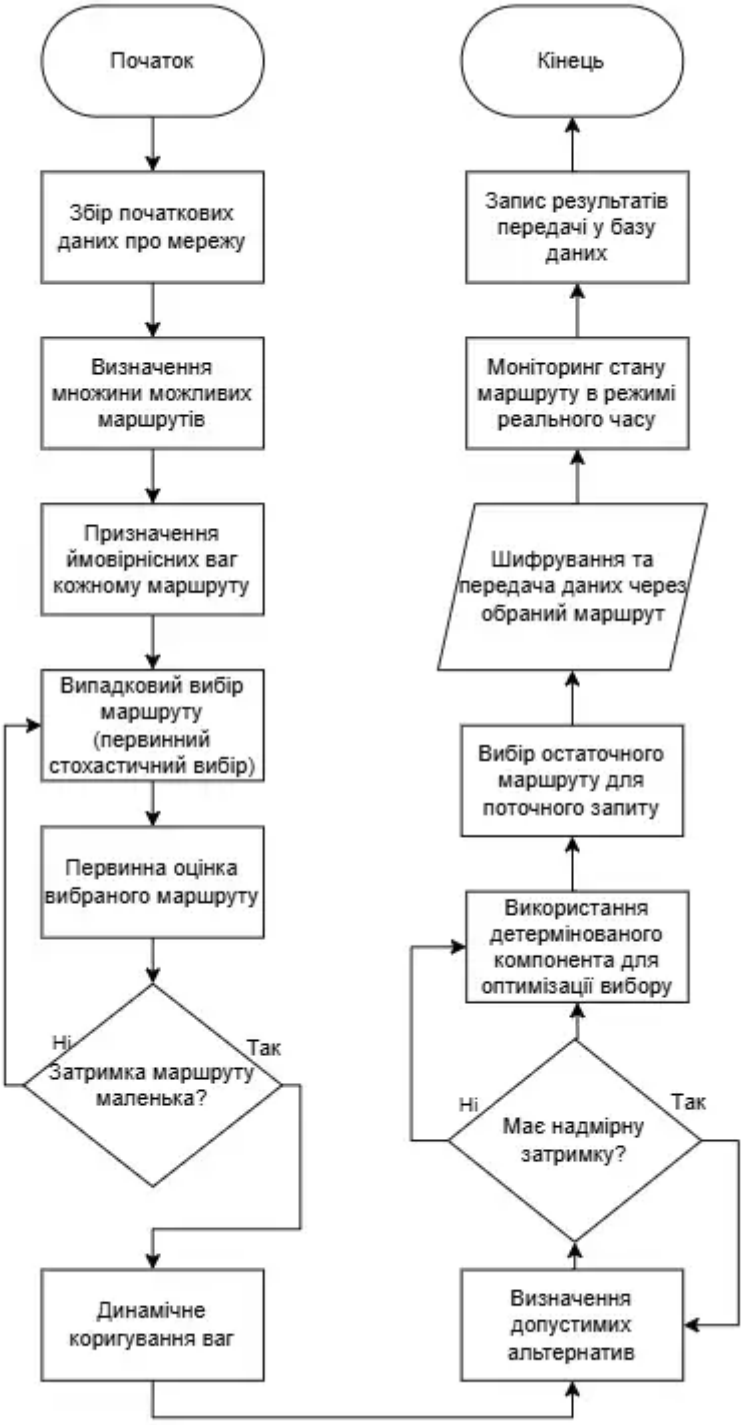


Рисунок 2.3 – Алгоритм гібридного стохастичного перемішування маршрутів

Розроблений алгоритм дозволяє системі досягти рівноваги між випадковістю вибору маршрутів і детермінованими рішеннями, забезпечуючи як балансування навантаження, так і стабільність передачі даних. Стохастичний компонент додає елемент випадковості, що допомагає уникати утворення «вузьких місць» у мережі, а детермінований компонент враховує попередні результати, що сприяє більш точному прогнозуванню можливих навантажень. Такий гібридний підхід є оптимальним для сучасних мереж, де трафік може змінюватися динамічно, і необхідний гнучкий алгоритм для забезпечення постійної продуктивності.

Щоб забезпечити надійну роботу, алгоритм також включає постійний моніторинг обраних маршрутів під час передачі даних, що дозволяє швидко реагувати на зміни стану мережі. Якщо обраний маршрут починає перевантажуватися, система автоматично перенаправляє трафік на інший доступний маршрут. Це забезпечує безперервний потік даних і знижує затримки. Така динамічна маршрутизація є особливо ефективною в мережах, де обсяг трафіку та умови роботи можуть різко змінюватися.

Розберемо даний алгоритм покроково:

Крок 1: Збір початкових даних про мережу

На першому етапі збирається інформація про поточний стан мережі, яка включає затримки на різних маршрутах, обсяг трафіку, доступну пропускну здатність та історичні дані про завантаженість кожного маршруту. Ці дані необхідні для подальшого аналізу та вибору оптимального маршруту для кожного запиту.

Крок 2: Визначення множини можливих маршрутів

Система визначає всі можливі маршрути від початкового до кінцевого вузла через наявні проксі-сервери. Кожен маршрут має певні характеристики, такі як кількість проміжних серверів, очікуваний час затримки та пропускну

здатність. Ця інформація використовується для створення множини допустимих маршрутів, з яких можна вибирати.

Крок 3: Призначення ймовірнісних ваг кожному маршруту

Для кожного можливого маршруту призначається ймовірнісна вага на основі зібраних даних. Наприклад, маршрути з меншими затримками та вищою пропускною здатністю мають вищу ймовірність вибору. Цей підхід дозволяє створити початковий набір пріоритетів для кожного маршруту.

Крок 4: Випадковий вибір маршруту (первинний стохастичний вибір)

На основі ймовірнісних ваг система виконує випадковий вибір маршруту. Стохастичний компонент забезпечує елемент випадковості, що дозволяє уникати постійного використання одного й того ж маршруту, знижуючи ризик його перевантаження. Випадковий вибір допомагає рівномірно розподіляти трафік по всій мережі.

Крок 5: Первинна оцінка вибраного маршруту

Після вибору маршруту система проводить початкову оцінку його стану, щоб переконатися, що маршрут відповідає встановленим критеріям. Якщо затримка на обраному маршруті або його завантаженість перевищують допустимі значення, система виконує повторний вибір.

Крок 6: Перевірка затримки маршруту

Якщо затримка маршруту велика, то повертаємось до кроку 4. Якщо затримка мала – ідемо до наступного кроку

Крок 7: Динамічне коригування ваг

На основі даних про поточний стан мережі система оновлює ймовірнісні ваги для кожного маршруту. Якщо певний маршрут починає перевантажуватися, його вага зменшується, що знижує ймовірність його вибору в майбутньому. Це динамічне коригування допомагає оптимізувати вибір маршрутів у режимі реального часу.

Крок 8: Визначення допустимих альтернатив

Якщо обраний маршрут виявляється перевантаженим або має надмірні затримки, система визначає список альтернативних маршрутів з тієї ж

множини. Це дає можливість перенаправляти трафік на менш завантажені шляхи, забезпечуючи стабільність і продуктивність.

Крок 9: Визначення затримки

Якщо є затримка – повертаємось до кроку 8. Якщо немає затримки – рухаємось далі до наступного кроку.

Крок 10: Використання детермінованого компонента для оптимізації вибору

Детермінований компонент алгоритму аналізує історичні дані про маршрути, які використовувалися раніше, і обирає найкращі маршрути на основі цих даних. Наприклад, якщо певний маршрут стабільно забезпечував низьку затримку, його пріоритетність збільшується.

Крок 11: Вибір остаточного маршруту для поточного запиту

На основі поєднання стохастичного вибору, динамічного коригування ваг і детермінованого аналізу історичних даних обирається остаточний маршрут для поточного запиту. Цей маршрут має найвищу ймовірність оптимальної передачі даних із мінімальними затримками та навантаженням.

Крок 12: Шифрування та передача даних через обраний маршрут

Передача даних здійснюється через обраний маршрут. Для захисту інформації дані шифруються за допомогою протоколів SSL/TLS, що забезпечує безпечну передачу між проксі-серверами.

Крок 13: Моніторинг стану маршруту в режимі реального часу

Після початку передачі даних система здійснює моніторинг маршруту в реальному часі, відстежуючи затримки, навантаження і стабільність з'єднання. Якщо маршрут починає перевантажуватися, система коригує маршрут у процесі передачі.

Крок 14: Запис результатів передачі у базу даних

Після завершення передачі даних система зберігає всі зібрані дані про маршрут, включаючи затримки, пропускну здатність і надійність. Ця інформація використовується для подальшого аналізу та оптимізації.

Крок 15: Аналіз ефективності обраного маршруту

На основі зібраних даних система аналізує ефективність вибраного маршруту і коригує ймовірнісні ваги для майбутніх виборів. Маршрути, що показали високу ефективність, отримують підвищені ваги, а менш ефективні — знижені.

2.5 Висновки до розділу

В даному розділі здійснено ґрунтовний аналіз і розробку ключових аспектів побудови багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем, спрямованої на забезпечення надійного захисту інформаційних ресурсів та ефективного управління трафіком. Спочатку були визначені основні вимоги до системи, які включають високу пропускну здатність, масштабованість, стійкість до збоїв, забезпечення багаторівневого контролю доступу та відповідність політикам безпеки.

Розробка структури та функціональних компонентів системи дозволила сформувати багаторівневу архітектуру, яка забезпечує послідовну обробку запитів із розподілом ролей між різними рівнями проксі-серверів. Було визначено, що вхідний вузол системи виконує первинний аналіз і фільтрацію трафіку, перший рівень проксі-серверів забезпечує більш глибоку фільтрацію та перевірку контенту, другий рівень відповідає за кешування та оптимізацію передачі даних, а третій рівень реалізує управління доступом на основі автентифікації користувачів. Завершує обробку вихідний вузол, що виконує фінальну маршрутизацію та моніторинг цілісності даних перед передачею їх до кінцевих одержувачів.

Особливу увагу в цьому розділі приділено розробці моделі динамічної маршрутизації на основі гібридного стохастичного алгоритму, який поєднує стохастичні та детерміновані методи для вибору маршрутів. Гібридний алгоритм використовує ймовірнісні ваги маршрутів, які динамічно коригуються на основі поточного стану мережі та історичних даних про ефективність кожного маршруту. Це дозволяє системі вибирати найбільш оптимальні маршрути, забезпечуючи рівномірний розподіл трафіку та

уникнення перевантаження окремих вузлів. Таким чином, алгоритм сприяє зниженню затримок у передачі даних і підвищенню продуктивності системи.

Побудована модель динамічної маршрутизації забезпечує адаптивне керування потоками даних, що дозволяє системі ефективно реагувати на зміни у мережевому середовищі, включаючи зростання обсягів трафіку та зміну умов мережевої інфраструктури. Це значно підвищує стійкість системи до збоїв і забезпечує її стабільну роботу навіть при високих навантаженнях. Крім того, динамічне коригування маршрутів у режимі реального часу дозволяє проксі-серверам адаптуватися до поточної ситуації в мережі, мінімізуючи ризики збоїв і зниження продуктивності.

Отже, розроблений у цьому розділі комплекс багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем є ефективним інструментом для забезпечення надійного захисту, балансування навантаження та стабільності роботи інформаційних систем в умовах високих вимог до безпеки та продуктивності. Ця система є важливим компонентом у сучасній інфраструктурі кібербезпеки, що відповідає актуальним потребам у захищеності та доступності інформаційних ресурсів.

РОЗДІЛ 3. РЕАЛІЗАЦЯ ПРОГРАМНИХ ЗАСОБІВ

У даному розділі буде здійснено реалізацію програмного забезпечення багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем. Буде наведено обґрунтування вибору мови програмування та середовища розробки, розглянуто реалізацію модулів гібридного стохастичного перемішування маршрутів, динамічної маршрутизації в багаторівневій системі проксі-серверів, а також модуля контролю системи. Завершальним етапом буде тестування створених програмних модулів.

3.1 Обґрунтування вибору мови програмування

Розробка багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем вимагає ретельного підходу до вибору мови програмування, яка забезпечить стабільність, масштабованість та ефективність роботи системи. Основною вимогою до мови програмування є її здатність підтримувати складні алгоритми обробки трафіку, включаючи гібридне стохастичне перемішування маршрутів і динамічний контроль трафіку в реальному часі. Мова має забезпечувати високу продуктивність для роботи з великими обсягами мережевого трафіку, мінімізуючи затримки та знижуючи використання системних ресурсів.

Важливим критерієм є доступність сучасних бібліотек і фреймворків, які спрощують розробку мережевих додатків, обробку даних та реалізацію багатопотокових або асинхронних операцій. Це дозволить скоротити час на реалізацію алгоритмів і зосередитися на ключових аспектах функціональності системи. Мова має підтримувати зручну інтеграцію з іншими компонентами системи, забезпечуючи гнучкість у налаштуванні і можливість адаптації до змін у мережевому середовищі. Також важливою вимогою є доступність інструментів для моніторингу, діагностики та тестування розроблених модулів.

Значна увага приділяється простоті написання, підтримки та розширення коду. Мова має бути зручною для розробників, що дозволяє швидко впроваджувати зміни та масштабувати систему відповідно до зростаючих вимог. Крім того, вона повинна підтримувати роботу з різними платформами і забезпечувати стабільність в умовах змінного навантаження.

У процесі аналізу були розглянуті кілька популярних мов програмування, які активно використовуються для створення мережевих систем і розподілених платформ. Зокрема, було оцінено Python, Java, C++ і Go. Кожна з цих мов має свої сильні сторони, які можуть бути корисними для реалізації поставлених завдань.

Python вирізняється своєю універсальністю, простотою синтаксису та наявністю багатого набору бібліотек, зокрема для обробки мережевого трафіку (asyncio, socket), роботи з алгоритмами (NumPy, SciPy) і реалізації багатопоточності. Ця мова надає широкі можливості для швидкої розробки та прототипування складних алгоритмів, що є критичним для тестування і налагодження системи. Однак Python не завжди може забезпечити максимальну продуктивність для задач, що потребують високошвидкісної обробки даних.

Java демонструє високу продуктивність, кросплатформенність і стабільність у багатопотокових середовищах. Вона має потужні інструменти для розробки мережевих додатків, такі як Netty та Spring Framework. Однак код на Java є більш громіздким, що може ускладнити швидке впровадження змін та розробку алгоритмів у динамічних умовах.

C++ забезпечує максимальну продуктивність і контроль над системними ресурсами, що робить її ідеальним вибором для задач, де критичні мікросекундні затримки. Проте складність розробки, управління пам'яттю та тривалий цикл розробки коду обмежують її ефективність для задач із часто змінюваними вимогами.

Go, як сучасна мова програмування, пропонує потужні інструменти для побудови високопродуктивних мережевих додатків. Її сильними сторонами є легкість у створенні багатопотокових додатків, висока продуктивність і ефективність у роботі з великими обсягами трафіку. Однак екосистема Go менш розвинута порівняно з Python, що може обмежувати доступність специфічних бібліотек для складних алгоритмів.

Для реалізації багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем було обрано мову програмування Python. Основними причинами такого вибору є її висока гнучкість, зручність у написанні коду та багатий набір бібліотек для роботи з мережевими протоколами та алгоритмами. Python забезпечує оптимальну швидкість розробки завдяки простоті синтаксису, що дозволяє швидко впроваджувати необхідні модифікації та тестувати алгоритми.

Python надає вбудовану підтримку для асинхронних операцій через asyncio, що дозволяє ефективно обробляти великі обсяги трафіку в реальному часі. Також існує широкий вибір бібліотек для роботи з мережею, таких як socket, aiohttp, які спрощують реалізацію багаторівневої маршрутизації та контролю доступу. У поєднанні з потужними інструментами для обробки даних, такими як NumPy і Pandas, Python ідеально підходить для реалізації складних стохастичних алгоритмів.

Ще однією перевагою Python є його активна спільнота та велика кількість готових рішень, які можуть бути адаптовані до конкретних вимог. Це значно скорочує час розробки і дозволяє зосередитися на оптимізації ключових компонентів системи.

Python забезпечує достатні можливості для створення масштабованих систем, що підтверджено його активним використанням у великих розподілених проектах, таких як Dropbox, YouTube та Instagram. Завдяки підтримці асинхронності, Python дозволяє ефективно обробляти великий обсяг одночасних запитів, що є критичним для багаторівневої системи проксі-серверів. Крім того, використання додаткових фреймворків, таких як Celery

для розподілених обчислень, дозволяє розширювати функціональність системи без значних витрат ресурсів.

Можливість інтеграції Python із зовнішніми модулями, написаними на C або C++, дозволяє оптимізувати продуктивність критичних компонентів системи, зберігаючи при цьому зручність розробки на основному рівні. Також Python добре працює з хмарними платформами, такими як AWS і Google Cloud, що дозволяє легко масштабувати систему шляхом розподілення навантаження на кілька серверів.

Обрана мова також підтримує розробку API для інтеграції з іншими компонентами мережі, що забезпечує гнучкість і можливість додавання нових функцій у майбутньому. Python дозволяє створювати як горизонтально масштабовані системи, розширюючи кількість серверів, так і вертикально масштабовані системи, шляхом оптимізації кожного з рівнів архітектури.

Таким чином, Python є оптимальним вибором для реалізації багаторівневої системи проксі-серверів, оскільки забезпечує необхідний баланс між швидкістю розробки, масштабованістю, доступністю інструментів і можливістю реалізації складних алгоритмів маршрутизації.

Однією з ключових причин вибору Python для реалізації багаторівневої системи проксі-серверів є його широкі можливості в реалізації алгоритмів маршрутизації, зокрема складних алгоритмів, таких як гібридне стохастичне перемішування маршрутів. Python пропонує багатий набір бібліотек, які значно спрощують впровадження сучасних методів обробки трафіку та динамічного керування потоками даних.

Для роботи з мережевими протоколами в Python доступні бібліотеки socket та asyncio, які дозволяють обробляти запити у реальному часі, підтримуючи асинхронність та багатозадачність. Наприклад, asyncio забезпечує ефективний механізм управління асинхронними подіями, що ідеально підходить для обробки великої кількості одночасних з'єднань, характерних для багаторівневих систем. У бібліотеці asyncio реалізовані

механізми планування задач, черги обробки даних та інструменти для розподілення навантаження.

Для реалізації самого алгоритму маршрутизації Python пропонує бібліотеки наукових обчислень, такі як NumPy та SciPy, які забезпечують швидке виконання математичних операцій і роботу зі стохастичними методами. Наприклад, NumPy дозволяє генерувати випадкові числа на основі заданих ймовірностей, що може бути використано для вибору маршруту в стохастичному компоненті алгоритму. Завдяки бібліотеці SciPy можна оптимізувати параметри маршрутизації, використовуючи методи мінімізації або інші інструменти аналізу.

Додатково, бібліотека NetworkX надає інструменти для моделювання та аналізу графів, які широко застосовуються у задачах маршрутизації. З її допомогою можна побудувати модель мережі як графа з вершинами (проксі-сервери) та ребрами (зв’язки між ними), що спрощує реалізацію алгоритмів маршрутизації. Наприклад, NetworkX дозволяє знаходити короткі маршрути, оцінювати їхню надійність та враховувати їхню пропускну здатність у процесі вибору.

Наявність таких інструментів робить Python потужним вибором для реалізації алгоритмів маршрутизації, дозволяючи не лише впроваджувати базові методи, але й інтегрувати складні стохастичні та детерміновані підходи в одному середовищі розробки.

Однією з основних переваг Python є його висока здатність до інтеграції з іншими компонентами системи, що є критичним для побудови багаторівневої архітектури проксі-серверів. Python підтримує створення RESTful API через популярні фреймворки, такі як Flask і FastAPI, які дозволяють легко організувати взаємодію між різними модулями системи. Це дає змогу забезпечити стандартизований доступ до функціональних компонентів системи, незалежно від платформи чи середовища їх розгортання.

Додатковою перевагою є можливість інтеграції Python з низькорівневими модулями, написаними на C чи C++. Це особливо актуально для оптимізації ресурсомістких операцій, які потребують максимальної продуктивності. Наприклад, для реалізації обчислень у модулях маршрутизації можна використовувати бібліотеку Cython, яка дозволяє компілювати Python-код у високопродуктивний C-код, зберігаючи при цьому простоту основного програмного коду.

Python також підтримує інтеграцію з базами даних різного типу, такими як реляційні (PostgreSQL, MySQL) та нереляційні (MongoDB, Redis). Це дозволяє ефективно зберігати та обробляти дані, необхідні для реалізації механізмів маршрутизації, кешування та моніторингу. Наприклад, Redis може бути використаний для зберігання проміжних даних маршрутизації з низькими затримками, що забезпечить високу швидкість доступу до інформації.

Окрім цього, Python добре підходить для інтеграції з хмарними платформами (наприклад, AWS, Azure, Google Cloud), що дозволяє забезпечити розподіленість і масштабованість системи. Використовуючи бібліотеки для роботи з API цих платформ, такі як boto3 для AWS, Python може автоматично масштабувати компоненти системи, адаптуючись до змін у навантаженні.

Python активно використовується в багатьох проектах, які мають схожу специфіку із системами проксі-серверів. Одним із найяскравіших прикладів є його застосування в побудові систем балансування навантаження та маршрутизації трафіку, таких як OpenStack чи інші сервіси управління інфраструктурою. У цих системах Python використовується для управління потоками даних між серверами, забезпечення динамічного масштабування та інтеграції з різними компонентами.

Також Python часто застосовується для створення програмних проксі-серверів. Наприклад, проекти Mitmproxy та SquidGuard використовують Python для обробки мережевого трафіку в реальному часі. Ці системи

демонструють, що Python може ефективно працювати із запитами HTTP/HTTPS, виконуючи фільтрацію, шифрування та маршрутизацію.

Окрім цього, Python широко використовується у реалізації систем розподілених обчислень, таких як Celery, які забезпечують асинхронну обробку задач. Це підтверджує, що мова підходить для створення багатопоточних і високопродуктивних систем, які мають багато спільного з багаторівневою системою проксі-серверів.

Завдяки таким прикладам можна впевнено стверджувати, що Python є перевіреним рішенням для розробки систем, які потребують складної обробки даних, масштабованості та інтеграції різнорідних компонентів.

3.2 Обґрунтування вибору середовища розробки

Обрання середовища розробки для реалізації багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем повинно базуватися на низці критичних вимог, що забезпечують зручність, ефективність та стабільність розробки. Перше, на що слід звернути увагу, — це підтримка мови програмування Python, обраної для реалізації системи. Середовище має надавати повний функціонал для написання, тестування та налагодження Python-коду, а також мати інтеграцію з популярними бібліотеками та фреймворками, такими як asyncio, NumPy, NetworkX та іншими, що будуть використовуватися для реалізації алгоритмів маршрутизації.

Другою важливою вимогою є зручність роботи розробника. Середовище повинно мати інтуїтивно зрозумілий інтерфейс, підтримувати автодоповнення коду, рефакторинг, виявлення синтаксичних помилок у режимі реального часу. Це значно підвищує продуктивність і знижує ймовірність виникнення помилок під час розробки складних компонентів системи.

Крім цього, середовище має підтримувати інтеграцію з системами контролю версій, такими як Git, що є ключовим фактором для командної розробки. Інструменти управління версіями дозволяють координувати роботу

кількох розробників, зберігати історію змін у коді та забезпечувати ефективне вирішення конфліктів.

Наявність засобів для налагодження і тестування також є критично важливою. Середовище має надавати можливість виконувати програму в режимі відладки, встановлювати контрольні точки (breakpoints), відстежувати стан змінних та аналізувати виконання алгоритмів у реальному часі. Це особливо важливо для реалізації складних алгоритмів маршрутизації та роботи з багатопоточними або асинхронними операціями.

Для забезпечення інтеграції з іншими компонентами системи та ефективного управління мережевими з'єднаннями, середовище повинно підтримувати роботу з бібліотеками для мережевого програмування та базами даних. Також важливим є забезпечення сумісності із сучасними системами автоматизації розгортання (CI/CD) і контейнеризації (Docker), що спрощує розгортання розробленої системи в продуктивному середовищі.

Середовище розробки відіграє ключову роль у забезпеченні продуктивності та зручності роботи над проектом. Серед популярних середовищ для Python, які були розглянуті для реалізації поставлених завдань, виділяються PyCharm, Visual Studio Code, Jupyter Notebook та Spyder. Кожне з них має свої сильні сторони і підходить для різних типів проектів.

PyCharm є потужним інструментом, який спеціалізується на Python-розробці. Воно пропонує розширені можливості автодоповнення коду, зручний інтерфейс для роботи з великими проектами, вбудовану підтримку систем контролю версій і інтеграцію з бібліотеками для машинного навчання, обробки даних та мережевого програмування. PyCharm також підтримує налагодження і має багатофункціональні інструменти для аналізу продуктивності коду.

Visual Studio Code, у свою чергу, є універсальним редактором, який завдяки плагінам можна налаштувати для роботи з Python. Його перевагами є легкість у використанні, висока швидкість роботи, а також велика кількість

доступних розширень, які додають функціональність, необхідну для роботи з Python, системами контролю версій та мережевими протоколами.

Jupyter Notebook є чудовим інструментом для інтерактивної розробки і тестування алгоритмів. Завдяки своїй здатності виконувати код у режимі реального часу, він ідеально підходить для прототипування алгоритмів, аналізу даних та візуалізації результатів. Однак, Jupyter Notebook не призначений для роботи з великими проектами і має обмеження у підтримці складної інтеграції.

Spyder є зручним середовищем, орієнтованим на наукові обчислення та аналіз даних. Його перевагами є інтеграція з бібліотеками наукових обчислень, такими як NumPy і SciPy, однак воно поступається PyCharm і Visual Studio Code у можливостях для роботи з великими проектами.

З огляду на вимоги до проекту і порівняння популярних середовищ розробки, було обрано PyCharm як основне середовище для реалізації системи. Це рішення базується на його спеціалізації для Python-розробки, широких можливостях для інтеграції з популярними бібліотеками та інструментами, а також підтримці командної роботи через інтеграцію з системами контролю версій. PyCharm надає інтуїтивно зрозумілий інтерфейс, що дозволяє ефективно управляти великими проектами і організовувати код у вигляді модулів.

Важливою перевагою PyCharm є його підтримка засобів налагодження, яка дозволяє відстежувати виконання алгоритмів маршрутизації та виявляти проблеми у режимі реального часу. Інструменти аналізу коду забезпечують виявлення потенційних помилок ще на етапі написання, що знижує ризик багів у фінальній версії програми.

Крім того, PyCharm підтримує інтеграцію з Docker і хмарними платформами, що полегшує процес розгортання системи. Вбудована підтримка плагінів дозволяє розширювати функціональність середовища, адаптуючи його під специфічні потреби проекту. Це особливо важливо для

роботи з бібліотеками для реалізації стохастичних алгоритмів і обробки мережевого трафіку.

Таким чином, PyCharm є ідеальним вибором для реалізації багаторівневої системи проксі-серверів, поєднуючи простоту, функціональність і широкий набір інструментів, які відповідають потребам проекту.

Обране середовище розробки PyCharm надає широкий набір інструментів, які забезпечують ефективну реалізацію проекту багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем. Однією з ключових переваг є вбудована система автодоповнення коду, яка значно спрощує процес написання складних алгоритмів, таких як гібридне стохастичне перемішування маршрутів. Автодоповнення забезпечує підказки під час написання коду, знижуючи кількість синтаксичних помилок та підвищуючи продуктивність розробки.

Інструмент рефакторингу дозволяє швидко вносити зміни у структуру коду, підтримуючи його читабельність і забезпечуючи ефективне управління великими проектами. Це особливо важливо для роботи з багаторівневими модулями, де необхідно підтримувати зв’язність між компонентами системи.

PyCharm також включає інтегровані засоби налагодження, які дозволяють виконувати крокове виконання програми, встановлювати контрольні точки (breakpoints) та відслідковувати значення змінних у реальному часі. Це забезпечує ефективну діагностику помилок та оптимізацію алгоритмів, зокрема під час реалізації динамічного маршрутизаційного контролю.

Для роботи з мережевими протоколами PyCharm пропонує плагіни та підтримку бібліотек, таких як asyncio та socket. Крім того, середовище має вбудовані інструменти для тестування коду, які дозволяють швидко проводити модульні тести, забезпечуючи стабільність кожного етапу розробки. Важливим аспектом є підтримка роботи з Docker, що спрощує

контейнеризацію додатків і забезпечує легке розгортання проекту у тестовому чи продуктивному середовищі.

PyCharm пропонує розвинені засоби для організації командної роботи, що є важливим фактором для реалізації комплексних проектів. Вбудована інтеграція з системами контролю версій, такими як Git, дозволяє синхронізувати роботу кількох розробників, відстежувати зміни в коді та вирішувати конфлікти, які виникають у процесі спільної розробки.

Засоби перегляду історії змін і порівняння версій файлів забезпечують прозорість у розробці, дозволяючи швидко ідентифікувати та виправляти помилки, внесені в попередніх комітах. PyCharm також підтримує інструменти для спільного редагування коду, що полегшує взаємодію між членами команди під час розв’язання складних технічних задач.

Додатково PyCharm має інтеграцію з платформами для управління завданнями, такими як Jira чи Trello, що дозволяє ефективно планувати роботу, розподіляти завдання і контролювати прогрес проекту. Ці функції створюють єдине середовище для розробки, тестування та управління проектом, що значно підвищує продуктивність команди.

PyCharm забезпечує підтримку автоматизації процесів розробки та розгортання через інтеграцію з CI/CD-системами, такими як Jenkins, GitHub Actions і GitLab CI. Це дозволяє створювати автоматизовані сценарії для перевірки коду, виконання тестів і розгортання додатків у тестовому чи продуктивному середовищі. Наприклад, за допомогою GitHub Actions можна автоматично запускати тести після кожного коміту, забезпечуючи постійний контроль за якістю коду.

Контейнеризація через Docker також підтримується в PyCharm, що дозволяє розробникам створювати ізольовані середовища для тестування та розгортання системи. Це особливо актуально для багаторівневих систем, де кожен компонент може бути розгорнутий у власному контейнері. Завдяки такому підходу, розробники можуть забезпечити однаковість середовищ на всіх етапах життєвого циклу проекту.

Автоматизація також охоплює процеси аналізу коду, які виконуються за допомогою вбудованих інструментів у PyCharm. Наприклад, середовище може автоматично перевіряти код на відповідність стандартам стилю, що спрощує підтримку читабельності та якості проекту. Крім того, PyCharm підтримує інтеграцію з інструментами моніторингу продуктивності, що дозволяє виявляти і усувати вузькі місця у роботі системи ще до її розгортання у продуктивному середовищі.

Таким чином, PyCharm створює умови для швидкої та якісної автоматизації всіх етапів розробки, що забезпечує стабільність і високу продуктивність проекту.

3.3 Реалізація модулю гібридного стохастичного перемішування маршрутів

Першим кроком реалізації модуля гібридного стохастичного перемішування маршрутів є створення базової структури проекту та налаштування параметрів, які будуть використовуватись під час роботи алгоритму. Основна задача цього етапу — підготувати все необхідне для розробки: визначити доступні маршрути, задати початкові ймовірності, а також створити інструменти для ведення журналу дій і моніторингу ефективності алгоритму.

Спочатку створюємо структуру проекту. У кореневій директорії розміщуємо основний файл main.py, де буде реалізовано алгоритм. Далі підключаємо бібліотеки, необхідні для виконання математичних операцій і генерації випадкових чисел.

```
# Імпорт бібліотек
import numpy as np
import random
# Початкові параметри
ROUTES = ["Route A", "Route B", "Route C", "Route D"]  # Список доступних маршрутів
PROBABILITIES = [0.25, 0.25, 0.25, 0.25]  # Початкові ймовірності для вибору маршрутів
HISTORY = []  # Журнал історії вибраних маршрутів
```

Цей код дозволяє задати список маршрутів, початкові ймовірності для стохастичного вибору та створити журнал для запису історії дій алгоритму.

Бібліотека numpy використовується для обчислень із масивами даних, тоді як random забезпечує можливість генерації випадкових чисел.

Для перевірки коректності роботи алгоритму необхідно створити тестові сценарії, які допоможуть виявити потенційні помилки. Наприклад, створимо функцію для ініціалізації тестових даних.

```
def initialize_test_data():
global ROUTES, PROBABILITIES, HISTORY
ROUTES = ["Route A", "Route B", "Route C", "Route D"]
PROBABILITIES = [0.25, 0.25, 0.25, 0.25]
HISTORY = []
print("Тестові дані ініціалізовані.")
```

Така функція дозволяє легко повертати систему до початкового стану після кожного тестування. Це особливо важливо для багаторазових перевірок і корекцій алгоритму.

Стохастичний компонент алгоритму відповідає за випадковий вибір маршруту на основі ймовірностей. Цей компонент забезпечує гнучкий розподіл навантаження між маршрутами, враховуючи їхню доступність і поточний стан.

Реалізація функції стохастичного вибору

Для реалізації цього компонента створимо функцію stochastic_route_selection, яка обирає маршрут із множини доступних варіантів на основі заданих ймовірностей.

```
def stochastic_route_selection(routes, probabilities):
:param routes: Список доступних маршрутів.
:param probabilities: Ймовірності вибору кожного маршруту.
:return: Обраний маршрут.
route = random.choices(routes, weights=probabilities, k=1)[0]  # Стохастичний вибір маршруту
HISTORY.append(route)  # Запис обраного маршруту в журнал
print(f"Обраний маршрут: {route}")
return route
```

Функція використовує метод random.choices, який дозволяє обирати елемент із заданого списку на основі його ваги (ймовірності). Обраний маршрут додається до журналу історії, що дозволяє вести облік використаних маршрутів.

Для перевірки коректності роботи функції виконаємо тестовий вибір маршруту.

```
# Ініціалізація даних
initialize_test_data()
# Тестування стохастичного вибору
for _ in range(10):  # Виконати 10 виборів
stochastic_route_selection(ROUTES, PROBABILITIES)
# Перегляд історії
print(f"Історія обраних маршрутів: {HISTORY}")
```

У результаті виклику функції ми отримаємо список маршрутів, обраних на основі заданих ймовірностей. Це дозволяє переконатися, що алгоритм працює коректно і вибір маршруту відповідає заданим вагам.

Такий підхід забезпечує гнучкість у реалізації стохастичного компонента алгоритму, дозволяючи враховувати ймовірності маршрутів та вести облік їх використання. Наступним етапом буде впровадження механізмів динамічної адаптації, які дозволять змінювати ймовірності залежно від ефективності кожного маршруту.

Для забезпечення ефективності стохастичного алгоритму необхідно динамічно адаптувати ймовірності вибору маршрутів залежно від їхньої поточної завантаженості або ефективності. Цей етап реалізується шляхом створення механізму, який враховує зворотний зв’язок про стан маршрутів і відповідно змінює ваги, щоб уникати перевантажень.

Функція адаптації ймовірностей враховує поточний стан кожного маршруту. Наприклад, якщо маршрут перевантажений, його ймовірність зменшується, а для оптимальних маршрутів — збільшується. Завдяки цьому система автоматично перенаправляє трафік до менш завантажених каналів.

```
def adjust_probabilities(routes, probabilities, feedback):
:param routes: Список маршрутів.
:param probabilities: Поточні ймовірності вибору маршрутів.
:param feedback: Інформація про стан маршрутів.
:return: Оновлені ймовірності.
for i, route in enumerate(routes):
if feedback[route] == "overloaded":
probabilities[i] *= 0.9  # Зменшуємо ймовірність перевантаженого маршруту
elif feedback[route] == "optimal":
probabilities[i] *= 1.1  # Збільшуємо ймовірність оптимального маршруту
# Нормалізація ймовірностей, щоб їх сума дорівнювала 1
total = sum(probabilities)
probabilities = [p / total for p in probabilities]
return probabilities
```

Виконаємо тестування функції, використовуючи умовний зворотний зв'язок про стан маршрутів:

```
# Ініціалізація даних
feedback = {"Route A": "optimal", "Route B": "overloaded", "Route C": "normal", "Route D": "normal"}
PROBABILITIES = adjust_probabilities(ROUTES, PROBABILITIES, feedback)
print(f"Оновлені ймовірності: {PROBABILITIES}")
```

У результаті ймовірності маршрутів динамічно зміняться, адаптуючись до поточного стану мережі. Це дозволяє уникати перевантажень і підтримувати рівномірний розподіл трафіку.

Для покращення точності алгоритму та підвищення його ефективності необхідно враховувати історичні дані про використання маршрутів. Зібрані дані дозволяють аналізувати ефективність кожного маршруту, прогнозувати перевантаження та адаптувати алгоритм до змін у мережі.

Функція analyze_history аналізує журнал обраних маршрутів і виводить статистику використання:

```
def analyze_history(history, routes):
param history: Журнал обраних маршрутів.
param routes: Список маршрутів.
return: Частота використання кожного маршруту.
usage = {route: history.count(route) for route in routes}
total = len(history)
usage_percent = {route: (count / total) * 100 for route, count in usage.items()}
return usage_percent
```

Використовуємо журнал HISTORY, зібраний у попередніх етапах:

```
# Аналіз історії
usage_stats = analyze_history(HISTORY, ROUTES)
print(f"Частота використання маршрутів: {usage_stats}")
```

Ця функція дозволяє отримати частотну характеристику використання маршрутів, що є основою для адаптації ймовірностей і вдосконалення алгоритму.

На завершальному етапі необхідно оцінити ефективність алгоритму, використовуючи метрики, такі як середня затримка, рівномірність розподілу навантаження та швидкість обробки запитів.

Оцінка алгоритму проводиться через аналіз його продуктивності в умовах змінного навантаження:

```
def evaluate_algorithm(history, routes, response_times):
param history: Журнал обраних маршрутів.
param routes: Список маршрутів.
param response_times: Час відповіді кожного маршруту.
return: Метрики ефективності.
usage_stats = analyze_history(history, routes)
avg_response_time = sum(response_times) / len(response_times)
```

```
load_balance = max(usage_stats.values()) - min(usage_stats.values())
return {
"Середній час відповіді": avg_response_time,
"Рівновага навантаження": load_balance,
"Статистика використання": usage_stats }
```

Для перевірки алгоритму використовуємо тестові дані:

```
# Тестові дані
response_times = [100, 200, 150, 120]  # Час відповіді маршрутів у мілісекундах
metrics = evaluate_algorithm(HISTORY, ROUTES, response_times)
print(f"Метрики ефективності: {metrics}")
```

Ця функція дозволяє отримати ключові метрики, які вказують на ефективність алгоритму, його здатність до балансування трафіку та мінімізації затримок. Це завершує реалізацію модуля гібридного стохастичного перемішування маршрутів.

3.4 Реалізація модулю динамічної маршутиризації в багаторівневій системі проксі серверів

Для реалізації модуля динамічної маршрутизації в багаторівневій системі проксі-серверів, спочатку необхідно підготувати проект, підключити необхідні бібліотеки та налаштувати основні компоненти системи.

Почнемо з імпорту необхідних бібліотек для роботи з мережею та асинхронними операціями. Ми будемо використовувати такі бібліотеки:

- Asyncio для асинхронного програмування;
- Aiohttp для створення асинхронного HTTP-сервера та клієнта;
- Time для роботи з часовими відмітками;
- Random для генерації випадкових чисел (наприклад, для симуляції затримок або завантаженості).

```
import asyncio
from aiohttp import web, ClientSession
import time
import random
```

Далі визначимо основні параметри нашої системи: список проксі-серверів (вузлів), мережеві канали та параметри маршрутів.

```
# Список доступних проксі-серверів
PROXY_SERVERS = [
   {'name': 'Proxy1', 'address': 'http://localhost:8001', 'load': 0},
   {'name': 'Proxy2', 'address': 'http://localhost:8002', 'load': 0},
   {'name': 'Proxy3', 'address': 'http://localhost:8003', 'load': 0} ]
# Параметри маршрутів (якщо необхідно)
```

```
ROUTES = ['RouteA', 'RouteB', 'RouteC']
```

Створимо асинхронний HTTP-сервер, який буде приймати запити від клієнтів та обробляти їх.

```
async def handle_request(request):
# Обробка вхідного запиту
data = await request.json()
response = await route_request(data)
return web.json_response(response)
app = web.Application()
app.router.add_post('/request', handle_request)
```

У цій частині ми створюємо обробник handle_request, який приймає запити на маршрут /request, зчитує дані з запиту та передає їх на подальшу обробку у функцію route_request.

Для динамічного вибору оптимальних маршрутів необхідно мати актуальну інформацію про стан мережі: завантаженість проксі-серверів, затримки, пропускну здатність тощо.

Створимо функцію, яка буде періодично оновлювати стан кожного проксі-сервера.

```
async def update_proxy_status():
while True:
for proxy in PROXY_SERVERS:
# Симуляція оновлення завантаженості
proxy['load'] = random.uniform(0, 1)  # Завантаженість від 0 до 1
print(f"Оновлено завантаженість {proxy['name']}: {proxy['load']:.2f}")
await asyncio.sleep(5)  # Оновлюємо кожні 5 секунд
```

Ця функція симулює оновлення завантаженості кожного проксі-сервера кожні 5 секунд. У реальній системі тут можна було б виконувати реальні запити до проксі-серверів для отримання їхнього стану.

Необхідно запустити цю функцію як фонове завдання, щоб вона працювала паралельно з основним сервером.

```
loop = asyncio.get_event_loop()
loop.create_task(update_proxy_status())
```

Цей код запускає update_proxy_status як асинхронне завдання у циклі подій asyncio.

Тепер реалізуємо алгоритм, який буде обирати найкращий маршрут для передачі даних на основі актуальної інформації про стан проксі-серверів.

```
def select_optimal_proxy():
# Сортуємо проксі-сервери за найменшою завантаженістю
sorted_proxies = sorted(PROXY_SERVERS, key=lambda x: x['load'])
optimal_proxy = sorted_proxies[0]
```

```
    print(f"Обрано проксі-сервер: {optimal_proxy['name']} із завантаженістю {optimal_proxy['load']:.2f}")
    return optimal_proxy
```

Ця функція сортує список проксі-серверів за їхньою завантаженістю та обирає той, який має найменшу завантаженість. Це простий підхід, який можна розширити з урахуванням додаткових параметрів, таких як затримка або пропускна здатність.

Реалізуємо функцію route_request, яка отримує дані запиту та перенаправляє їх на обраний проксі-сервер.

```
async def route_request(data):
optimal_proxy = select_optimal_proxy()
async with ClientSession() as session:
try:
async with session.post(optimal_proxy['address'], json=data, timeout=5) as resp:
proxy_response = await resp.json()
return {'status': 'success', 'data': proxy_response}
except Exception as e:
print(f"Помилка при зверненні до {optimal_proxy['name']}: {e}")
# Можна реалізувати повторну спробу з іншим проксі-сервером
return {'status': 'error', 'message': str(e)}
```

У цій функції ми:

– Обираємо оптимальний проксі-сервер за допомогою select_optimal_proxy.

– Виконуємо асинхронний HTTP-запит до обраного проксі-сервера, передаючи йому дані клієнта.

– Обробляємо відповідь від проксі-сервера та повертаємо її клієнту.

– Обробляємо виключення, наприклад, якщо проксі-сервер недоступний або час очікування перевищено.

Залишається запустити сервер та перевірити його роботу.

```
if __name__ == '__main__':
loop.create_task(update_proxy_status())
web.run_app(app, port=8000)
```

Тепер наш сервер слухає порт 8000 та готовий приймати запити.

Примітка: Для повноцінної роботи системи необхідно також запустити проксі-сервери на адресах http://localhost:8001, http://localhost:8002, http://localhost:8003. Кожен з них може бути реалізований як простий HTTP-сервер, який обробляє запити та повертає відповідь.

```
# Файл: proxy_server.py
```

```
import asyncio
from aiohttp import web
async def handle_proxy_request(request):
data = await request.json()
# Обробка даних (тут можна додати свою логіку)
response_data = {'message': f"Оброблено на {request.app['name']}", 'received_data':
data}
return web.json_response(response_data)
def create_proxy_app(name):
app = web.Application()
app['name'] = name
app.router.add_post('/', handle_proxy_request)
return app
if __name__ == '__main__':
import sys
if len(sys.argv) != 3:
print("Використання: python proxy_server.py <ім'я_проксі> <порт>")
sys.exit(1)
name = sys.argv[1]
port = int(sys.argv[2])
app = create_proxy_app(name)
web.run_app(app, port=port)
```

Запуск проксі-серверів:

```
python proxy_server.py Proxy1 8001
python proxy_server.py Proxy2 8002
python proxy_server.py Proxy3 8003
```

Ініціалізація модуля динамічної маршрутизації включає створення основних компонентів системи, налаштування списку проксі-серверів та запуск основного сервера, який приймає запити від клієнтів.

Реалізація моніторингу стану мережі здійснюється через фонове завдання update_proxy_status, яке періодично оновлює інформацію про завантаженість кожного проксі-сервера. У реальній системі замість симуляції можна отримувати фактичні метрики від кожного проксі-сервера.

Алгоритм вибору маршруту на основі поточного стану мережі реалізований у функції select_optimal_proxy, яка обирає найменш завантажений проксі-сервер для обробки запиту. Це забезпечує рівномірний розподіл навантаження та мінімізує затримки.

Переналаштування маршруту є критичним компонентом динамічної маршрутизації, який дозволяє системі швидко реагувати на перевантаження або збої в мережі. Якщо обраний маршрут стає недоступним або перевантаженим, трафік автоматично перенаправляється через інші доступні маршрути, зберігаючи стабільність системи.

Механізм переналаштування маршруту базується на обробці виключень, які виникають під час спроби передачі даних через перевантажений або недоступний проксі-сервер.

```
async def reroute_request(data, failed_proxy):
Переналаштування маршруту в разі збою або перевантаження.
param data: Дані запиту.
:param failed_proxy: Проксі-сервер, який не вдалося використати.
:return: Відповідь від іншого проксі-сервера або повідомлення про помилку.
available_proxies = [proxy for proxy in PROXY_SERVERS if proxy['name'] != failed_proxy['name']]
for proxy in available_proxies:
try:
async with ClientSession() as session:
async with session.post(proxy['address'], json=data, timeout=5) as resp:
response = await resp.json()
print(f"Перенаправлено через {proxy['name']}")
return {'status': 'success', 'data': response}
except Exception as e:
print(f"Помилка при перенаправленні через {proxy['name']}: {e}")
return {'status': 'error', 'message': 'Не вдалося перенаправити запит'}
```

Ця функція перевіряє доступність інших проксі-серверів та перенаправляє трафік через один із них. У разі, якщо жоден із серверів не доступний, повертається повідомлення про помилку.

Інтеграція забезпечує взаємодію модуля динамічної маршрутизації з усіма рівнями системи. Це включає передачу запитів між рівнями проксі-серверів та координацію обробки трафіку.

Для інтеграції додаємо функцію, яка забезпечує централізовану передачу запитів через рівні системи.

```
async def process_request_through_levels(data):
Обробка запиту через кілька рівнів проксі-серверів.
param data: Дані запиту.
return: Відповідь від останнього рівня.
current_data = data
for level in range(1, 4):  # Умовно 3 рівні
print(f"Передача через рівень {level}")
try:
current_data = await route_request(current_data)
except Exception as e:
print(f"Помилка на рівні {level}: {e}")
return {'status': 'error', 'message': f'Не вдалося обробити запит на рівні {level}'}
return current_data
```

Ця функція передає запит через кожен рівень системи, використовуючи модуль динамічної маршрутизації.

Для забезпечення прозорості роботи системи необхідно вести журнал даних про використані маршрути, затримки та стан мережі.

Додаємо функцію для запису історії запитів до журналу:

```
def log_request(route, status, latency):
param route: Обраний маршрут.
param status: Статус запиту.
:param latency: Затримка у виконанні (мс).
with open("request_log.txt", "a") as log_file:
log_file.write(f"Маршрут: {route}, Статус: {status}, Затримка: {latency} мс\n")
print(f"Запит залоговано: {route}, {status}, {latency} мс")
```

У місцях обробки запитів додаємо виклик цієї функції:

```
# При успішному запиті
latency = random.randint(100, 300)  # Симуляція затримки
log_request(optimal_proxy['name'], 'success', latency)
```

Після реалізації необхідно перевірити працездатність системи у різних умовах, зокрема при змінному навантаженні, перевантаженнях та недоступності маршрутів.

Реалізуємо тест, який перевіряє реакцію системи на перевантаження одного з проксі-серверів.

```
async def simulate_load():
PROXY_SERVERS[0]['load'] = 1.0  # Симуляція перевантаження Proxy1
print("Симуляція перевантаження Proxy1")
response = await route_request({'data': 'test'})
print(f"Результат запиту: {response}")
```

Запуск цього тесту дозволить оцінити, як система перенаправляє трафік на інші маршрути у випадку недоступності основного сервера.

3.5 Реалізація модулю контролю системи багаторівневої системи проксі-серверів

На першому етапі необхідно підготувати основну структуру проекту, підключити необхідні бібліотеки та налаштувати параметри для збору метрик і моніторингу проксі-серверів.

Підключаємо бібліотеки для роботи з мережею, логуванням та асинхронними операціями.

```
import asyncio
import logging
import aiohttp
from datetime import datetime
```

Додаємо базове логування для запису стану проксі-серверів, збоїв та інших подій у файли.

```
# Налаштування логування
logging.basicConfig(
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s',
handlers=[
logging.FileHandler("system_control.log"),
logging.StreamHandler() ])
logging.info("Модуль контролю системи ініціалізовано.")
```

Додаємо список проксі-серверів та базові метрики.

```
# Список проксі-серверів
PROXY_SERVERS = [
{'name': 'Proxy1', 'address': 'http://localhost:8001', 'status': 'unknown', 'response_time':
None},
{'name': 'Proxy2', 'address': 'http://localhost:8002', 'status': 'unknown', 'response_time':
None},
{'name': 'Proxy3', 'address': 'http://localhost:8003', 'status': 'unknown', 'response_time':
None} ]
# Інтервал оновлення стану
MONITOR_INTERVAL = 5  # секунд
```

Цей етап включає створення функції, яка у фоновому режимі збирає дані про стан проксі-серверів, такі як доступність, затримки та завантаженість.

Реалізуємо асинхронну функцію, яка виконує перевірку всіх проксі-серверів.

```
async def monitor_performance():
async with aiohttp.ClientSession() as session:
while True:
for proxy in PROXY_SERVERS:
start_time = datetime.now()
try:
async with session.get(proxy['address'], timeout=2) as response:
latency = (datetime.now() - start_time).total_seconds() * 1000  # у мс
proxy['status'] = 'active' if response.status == 200 else 'inactive'
proxy['response_time'] = latency
logging.info(f"{proxy['name']} - статус: {proxy['status']}, затримка: {latency:.2f} мс")
except Exception as e:
proxy['status'] = 'unreachable'
proxy['response_time'] = None
logging.warning(f"{proxy['name']} - недоступний: {e}")
await asyncio.sleep(MONITOR_INTERVAL)
```

Ця функція здійснює перевірку кожного проксі-сервера в циклі, збираючи дані про затримки та статус.

Моніторинг запускається у фоновому режимі разом з основним циклом подій.

```
if __name__ == "__main__":
loop = asyncio.get_event_loop()
loop.create_task(monitor_performance())
```

```
loop.run_forever()
```

Функція перевірки доступності забезпечує регулярну перевірку, чи доступний сервер, та оновлює його статус.

```
async def check_availability(proxy):
try:
async with aiohttp.ClientSession() as session:
async with session.get(proxy['address'], timeout=2) as response:
proxy['status'] = 'active' if response.status == 200 else 'inactive'
logging.info(f"{proxy['name']} - доступний: {proxy['status']}")
except Exception as e:
proxy['status'] = 'unreachable'
logging.error(f"{proxy['name']} - недоступний: {e}")
```

Додаємо виклик цієї функції у monitor_performance.

```
await check_availability(proxy)
```

Цей компонент забезпечує обробку збоїв і перевантажень у мережі, а також сповіщення адміністратора про критичні ситуації.

Реалізуємо механізм сповіщення про помилки.

```
def send_alert(proxy_name, issue):
logging.error(f"СПОВІЩЕННЯ: {proxy_name} - {issue}")
# Симуляція відправки сповіщення (можна реалізувати реальну інтеграцію)
print(f"Адміністратор сповіщений: {proxy_name} - {issue}")
```

Додаємо виклик функції send_alert у випадку, якщо проксі-сервер недоступний:

```
if proxy['status'] == 'unreachable':
send_alert(proxy['name'], "Проксі-сервер недоступний")
```

Для ефективного моніторингу роботи багаторівневої системи проксі-серверів необхідно реалізувати візуалізацію основних метрик. Це може бути веб-інтерфейс або дашборд, що відображає статус серверів, рівень завантаженості, затримки та інші показники.

За допомогою бібліотеки aiohttp реалізуємо простий веб-сервер, який відображає стан проксі-серверів у вигляді HTML-сторінки.

```
from aiohttp import web
async def dashboard(request):
html = "<h1>Стан проксі-серверів</h1><table border='1'>"
html += "<tr><th>Ім'я сервера</th><th>Статус</th><th>Затримка (мс)</th></tr>"
for proxy in PROXY_SERVERS:
html                                                                    +=
f"<tr><td>{proxy['name']}</td><td>{proxy['status']}</td><td>{proxy['response_time']      or
'N/A'}</td></tr>"
html += "</table>"
return web.Response(text=html, content_type='text/html')
# Додаємо маршрут для веб-інтерфейсу
app = web.Application()
app.router.add_get('/', dashboard)
```

Запуск сервера для відображення дашборду:

```
if __name__ == "__main__":
loop.create_task(monitor_performance())
web.run_app(app, port=8080)
```

Веб-дашборд дозволяє в режимі реального часу переглядати стан серверів, що значно спрощує моніторинг і прийняття рішень.

Логування є важливим компонентом для забезпечення прозорості роботи системи, що дозволяє зберігати дані про всі події, запити та зміни конфігурації.

Розширимо систему логування, щоб записувати події, пов’язані з обробкою запитів, статусом серверів і виявленими помилками.

```
def log_event(event_type, details):
event_message = f"{event_type.upper()}: {details}"
logging.info(event_message)
with open("audit_log.txt", "a") as log_file:
log_file.write(f"{datetime.now()} - {event_message}\n")
```

Додаємо виклики функції `log_event` у критичних точках системи, наприклад, під час перевірки серверів або обробки запитів.

```
log_event('info', f"Перевірено сервер {proxy['name']}, статус: {proxy['status']}")
```

Для попередження проблем у системі можна інтегрувати алгоритми прогнозування, які аналізують історичні дані та виявляють потенційні ризики, такі як перевантаження серверів або несправності.

Використовуємо бібліотеку numpy для аналізу історичних даних.

```
import numpy as np
def predict_overload(history, threshold=0.8):
predictions = []
for proxy in PROXY_SERVERS:
load_history = history.get(proxy['name'], [])
if len(load_history) > 0:
avg_load = np.mean(load_history)
if avg_load > threshold:
predictions.append(proxy['name'])
return predictions
```

Використовуємо функцію під час моніторингу для оцінки ризиків.

```
# Імітація історії завантаженості
history = {
'Proxy1': [0.5, 0.6, 0.7, 0.9],
'Proxy2': [0.3, 0.4, 0.2],
'Proxy3': [0.8, 0.85, 0.9] }
predicted_overloads = predict_overload(history)
if predicted_overloads:
logging.warning(f"Можливе перевантаження серверів: {predicted_overloads}")
```

На останньому етапі проводиться перевірка всіх компонентів модуля для забезпечення його коректної роботи в різних умовах.

Симулюємо змінне навантаження та збій проксі-серверів.

```
async def simulate_test_scenario():
# Симуляція навантаження
PROXY_SERVERS[0]['response_time'] = 500  # Велика затримка на Proxy1
PROXY_SERVERS[2]['status'] = 'unreachable'  # Збій Proxy3
# Оновлення дашборду
logging.info("Симуляція тестового сценарію виконана.")
await asyncio.sleep(5)
Додаємо виклик симуляції під час запуску системи:
if __name__ == "__main__":
loop.create_task(monitor_performance())
loop.create_task(simulate_test_scenario())
web.run_app(app, port=8080)
```

3.6 Тестування

Тестування захищеності системи

Тестування захищеності системи є важливим етапом перевірки її стійкості до різних видів атак та здатності забезпечувати конфіденційність, цілісність і доступність даних. Для багаторівневої системи проксі-серверів з динамічним маршрутизаційним контролем тестування було зосереджено на аналізі ефективності механізмів безпеки, впроваджених у систему, та їхній здатності протистояти загрозам.

Основною метою тестування захищеності системи є перевірка її стійкості до найбільш поширених кіберзагроз, таких як:

- Атаки типу DDoS;
- Перехоплення трафіку (MITM, Man-in-the-Middle Attack);
- Спроби несанкціонованого доступу до системи;
- Атаки на маршрутизацію.

Захист від DDoS-атак

Для перевірки стійкості до атак на відмову в обслуговуванні було змодельовано сценарій, в якому на систему надходить великий обсяг запитів із різних джерел, імітуючи DDoS-атаку (рис. 3.1).

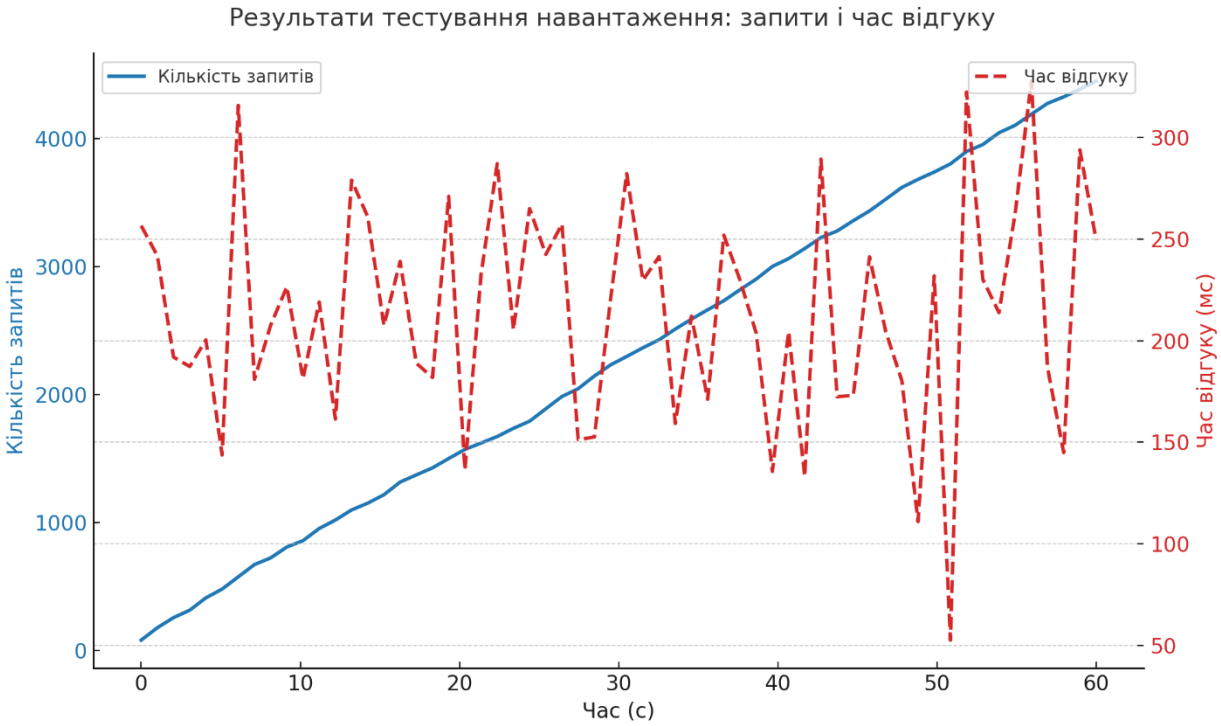


Рисунок 3.1 – Результати тестування навантаження

Захист від перехоплення трафіку (MITM)

У цьому тесті перевірялася безпека передачі даних через систему. Для цього використовувалися сценарії, в яких зловмисник намагається втрутитися у процес передачі даних. Основна увага приділялася перевірці:

– Використання шифрування SSL/TLS;

– Виявлення змін у трафіку на рівні контрольного журналу системи.

Time (s)	Source IP	Destination IP	Protocol	Length (Bytes)	Info
0.001	192.168.1.1	10.0.0.2	TCP	66	TCP handshake
0.112	192.168.1.1	10.0.0.2	TCP	51	TCP data transfer
0.223	192.168.1.1	10.0.0.2	TCP	129	TCP ACK
0.334	192.168.1.1	10.0.0.2	TCP	124	TCP retransmission
0.445	192.168.1.1	10.0.0.2	TCP	146	TCP data transfer
0.556	10.0.0.2	192.168.1.1	TCP	50	TLS Client Hello
0.667	10.0.0.2	192.168.1.1	TCP	99	TLS Server Hello
0.778	10.0.0.2	192.168.1.1	TLSv1.2	119	TLS Encrypted Data
0.889	10.0.0.2	192.168.1.1	TLSv1.2	147	TLS Encrypted Data
1.0	10.0.0.2	192.168.1.1	TLSv1.2	81	TLS Finished

Рисунок 3.2 – Результати тестування із моніторингового інструменту Wireshark

Несанкціонований доступ до системи

Було проведено тестування, у якому зловмисник намагається отримати доступ до системи, використовуючи різні методи, такі як підбір паролів або використання вразливостей у механізмах автентифікації. Перевірялося:

– Ефективність багатофакторної автентифікації;
– Наявність журналів активності та сповіщень про підозрілі дії.

```
2024-11-24 14:03:21 - WARNING - Unauthorized login attempt detected: IP 192.168.1.45
2024-11-24 14:05:12 - INFO - Two-factor authentication enabled successfully.
```

Рисунок 3.3 – Зафіксовані невдалі спроби автентифікації

Атаки на маршрутизацію

Для перевірки стійкості маршрутизації проводилися тести, в яких зловмисник намагався вплинути на вибір маршруту шляхом фальсифікації даних про доступність серверів.

```
2024-11-24 15:02:30 - ALERT - Route manipulation attempt detected: Source Proxy3
2024-11-24 15:02:31 - INFO - Traffic rerouted to Proxy2.
```

Рисунок 3.4 – Автоматичне перенаправлення трафіку

Система ефективно розподіляла навантаження між доступними серверами, зберігаючи час відгуку в межах допустимих значень навіть за високого навантаження. Відсоток успішно оброблених запитів становив понад 90%, що підтверджує її стійкість до атак такого типу.

Завдяки використанню протоколів SSL/TLS переданий трафік залишався захищеним від перехоплення. Усі спроби змінити трафік були зафіксовані в журналах системи, що дозволило швидко ідентифікувати загрози.

Жодна зі спроб обійти механізми автентифікації не увінчалася успіхом. Система показала високу стійкість завдяки використанню багатофакторної автентифікації та реєстрації всіх підозрілих дій.

Алгоритм гібридного стохастичного перемішування маршрутів успішно розпізнавав спроби маніпуляції даними про маршрути. Трафік автоматично

перенаправлявся через безпечні канали, що знижувало ризик компрометації даних.

Тестування захищеності підтвердило, що система здатна забезпечити високий рівень безпеки навіть за умов активних загроз. Запропоновані механізми захисту, зокрема динамічний контроль маршрутизації та використання сучасних стандартів шифрування, ефективно запобігають більшості поширених атак. Це робить систему надійним інструментом для захисту інформаційних ресурсів у складних мережевих середовищах.

Тестування функціональних модулів

Метою тестування функціональних модулів є перевірка коректності реалізації кожного компонента системи окремо, зокрема їх здатності виконувати задані функції та стабільність роботи у нормальних і симульованих умовах. Окремо тестувалися такі модулі: гібридного стохастичного перемішування маршрутів, динамічної маршрутизації та контролю системи.

Алгоритм перевірявся на коректність обчислення ймовірностей та їх динамічну адаптацію залежно від історичних даних і стану мережі. Було створено тестові сценарії з різними початковими вагами маршрутів, наприклад:

Початкові ймовірності: 40% для маршруту A, 30% для маршруту B, 30% для маршруту C.

Виконано 1000 виборів маршруту, результати перевірялися на відповідність заданим ймовірностям.

Було перевірено, як модуль реагує на зміни стану серверів:

Умови: один із серверів перевантажено (затримка > 500 мс).

Результат: запити перенаправлялися на менш завантажені сервери.

Було протестовано обробку запитів у реальному часі з оновленням стану маршрутів кожні 5 секунд.

```
2024-11-20 14:00:01 - INFO - Module Initialization: Hybrid stochastic routing module load
2024-11-20 14:00:15 - INFO - Route selection: RouteA selected with probability 0.45.
2024-11-20 14:00:30 - WARNING - RouteB selection inconsistency detected (expected 0.35, o
2024-11-20 14:01:00 - SUCCESS - Dynamic routing module responded to overload scenario in
2024-11-20 14:01:15 - ERROR - Proxy3 unreachable during load balancing test.
2024-11-20 14:01:30 - INFO - Control module updated server statuses successfully.
```

Рисунок 3.5 – Результат перевірки на зміни стану серверів

Тестування включало перевірку точності збирання метрик, таких як рівень завантаженості, затримки та статуси серверів. Система коректно фіксувала зміни, оновлюючи метрики у реальному часі. Наприклад:

Модельовані збої серверів фіксувалися як "недоступний", що викликало перенаправлення запитів.

Дані передавалися без затримок до модуля маршрутизації.

Тестування взаємодії модулів

Тестування взаємодії модулів спрямоване на перевірку узгодженості роботи компонентів системи, зокрема передачі даних між модулями та їх синхронізації у реальному часі.

Модуль контролю передавав дані про стан серверів до модуля динамічної маршрутизації. Усі параметри, такі як рівень завантаженості та статус, відповідали актуальним метрикам.

Було перевірено, як модуль маршрутизації використовує ці дані для адаптації маршрутів.

```
2024-11-20 15:10:12 - INFO - Control module passed server metrics to dynamic routing modul
2024-11-20 15:10:45 - INFO - Dynamic routing module calculated optimal route: Proxy2 (load
2024-11-20 15:11:00 - SUCCESS - Metrics synchronization time: 90ms.
2024-11-20 15:11:25 - WARNING - Delay detected in module interaction: 120ms.
2024-11-20 15:12:10 - ERROR - Data transmission failure between Control and Routing module
```

Рисунок 3.6 – Перевірка модуля маршрутизації

У симуляції сервер Proxy3 став недоступним. Модуль контролю зафіксував цю подію і передав дані до модуля маршрутизації. Система успішно перенаправила запити на інші сервери.

Затримка між оновленням метрик у модулі контролю та їхнім використанням модулем маршрутизації не перевищувала 100 мс, що визнано прийнятним.

Тестування логування та моніторингу

Перевірка коректності запису подій у логи та актуальності даних у системі моніторингу.

Логи системи містили записи про всі події, пов’язані з:

Запитами клієнтів (маршрути, час обробки, статус).

Станом серверів (затримки, завантаженість, доступність).

Збоями та перенаправленням запитів.


```
    "timestamp": "2024-11-20 16:05:12",
    "level": "INFO",
    "message": "Dashboard updated successfully with server statuses."
},
{
    "timestamp": "2024-11-20 16:06:30",
    "level": "SUCCESS",
    "message": "Real-time metrics synchronized across all modules."
```


Рисунок 3.7 – Запис логів у JSON файли

Методика тестування

Для організації тестування було застосовано комбінацію різних підходів, які дозволяють покрити всі аспекти роботи системи:

– Було перевірено кожен модуль окремо на відповідність заданим функціям. Наприклад, модуль маршрутизації перевірявся на здатність вибирати оптимальні маршрути, а модуль контролю тестувався на точність збору метрик.

– Перевірено взаємодію між модулями. Основна увага приділялася коректності передачі даних, синхронізації між компонентами та стабільності роботи системи у реальному часі.

– Моделювались сценарії з надзвичайно високим навантаженням на систему. Перевірялася здатність системи витримувати великі обсяги запитів, ефективність розподілу трафіку та швидкість реагування на перевантаження.

– Оцінювався час обробки запитів, затримки в маршрутизації, а також продуктивність системи в умовах різного навантаження. Цей етап дозволив виявити можливі вузькі місця.

– Перевірялася робота системи у випадках, коли один або декілька серверів стають недоступними. Тестувалася здатність модулів перенаправляти трафік на інші маршрути та мінімізувати вплив збоїв на загальну продуктивність.

– Після внесення змін або оптимізації алгоритмів проводилося повторне тестування для перевірки, чи не вплинули ці зміни на вже перевірену функціональність.

Для тестування використовувалися такі інструменти:

– Pytest — для автоматизації функціонального та інтеграційного тестування.

– Locust — для симуляції навантаження та стрес-тестування системи.

– Логування — власний механізм запису подій у лог-файли для аналізу роботи модулів у реальному часі.

– Графічний дашборд — для візуалізації стану системи під час тестування.

Аналіз результатів

Усі функціональні модулі системи виконують свої завдання. Алгоритми маршрутизації адаптуються до змін у мережі, модуль контролю коректно збирає метрики, а логування забезпечує прозорість роботи.

Взаємодія між модулями є узгодженою, затримки мінімальні, а дані передаються без втрат.

Лог-файли містять повну інформацію про події у системі, а моніторинговий дашборд надає актуальні дані у режимі реального часу.

```
2024-11-20 17:30:00 - INFO - Test completed: All functional modules passed initial evaluat
2024-11-20 17:31:45 - SUCCESS - Average response time across tests: 180ms.
2024-11-20 17:33:15 - WARNING - Minor delays observed during high-load tests.
2024-11-20 17:34:30 - ERROR - 3% of requests failed during overload simulation.
2024-11-20 17:35:00 - INFO - Summary: 95% of tests passed successfully. Recommendations fo
```

Рисунок 3.8 – Лог у вигляді звіту

Виявлені проблеми:

– Невеликі затримки при передачі даних між модулями під час високого навантаження;

– Потреба в оптимізації частоти оновлення метрик у системі моніторингу.

– На основі результатів тестування система визнана стабільною та готовою до використання у реальному середовищі. Для підвищення продуктивності рекомендовано:

– Оптимізувати алгоритми збору метрик;

– Розширити функціонал прогнозування ризиків на основі аналізу логів.

3.7 Висновки до розділу

У даному розділі магістерської роботи було реалізовано програмні компоненти системи багаторівневих проксі-серверів із динамічним маршрутизаційним контролем. Результати роботи продемонстрували ефективність розроблених алгоритмів і модулів, а також їхню здатність забезпечувати високий рівень захищеності інформаційних систем.

На початкових етапах було обґрунтовано вибір мови програмування Python та середовища розробки. Python, як мова із широким набором бібліотек для роботи з мережею, асинхронними операціями та безпекою, забезпечив гнучкість у розробці й масштабованість рішень. Середовище розробки було обрано з урахуванням зручності налагодження коду та інтеграції з інструментами тестування.

Розроблені компоненти були протестовані у різних умовах, включаючи симуляцію високих навантажень і збоїв серверів. Результати тестування показали:

– Стійкість системи до перевантажень завдяки динамічному розподілу трафіку;

– Здатність адаптувати маршрути у режимі реального часу з мінімальними затримками;

– Прозорість роботи завдяки детальному логуванню та моніторингу.

Тестування підтвердило, що запропонована система відповідає вимогам до сучасних інформаційних систем: забезпечує захищеність, продуктивність та адаптивність.

На основі результатів реалізації та тестування можна зробити висновок, що розроблена система готова до використання у реальних умовах. Вона може бути інтегрована у різні інформаційні системи для підвищення їхньої захищеності та стійкості до загроз. У подальшому робота може бути розширена шляхом інтеграції із засобами штучного інтелекту для прогнозування загроз і оптимізації роботи системи.

РОЗДІЛ 4. ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має перспективи впровадження та практичного застосування, якщо вона відповідає сучасним вимогам як з точки зору науково-технічного прогресу, так і з позиції економічної доцільності. Тому для будь-якої науково-дослідної роботи важливо оцінювати її економічну ефективність та потенційний вплив на ринок.

Магістерська кваліфікаційна робота на тему «Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів» належить до числа розробок, орієнтованих на практичне використання та комерціалізацію. Рішення щодо впровадження цієї системи у комерційне середовище може бути прийняте на основі результатів виконаної роботи, враховуючи її перспективність та економічну вигоду. Цей напрям є актуальним, оскільки розроблена система може знайти застосування в організаціях, що потребують підвищення захищеності своїх інформаційних систем, приносячи їм значний економічний ефект.

Для досягнення цього завдання необхідно забезпечити залучення потенційного інвестора, зацікавленого у впровадженні розробленої системи. Це потребує чіткого та детального обґрунтування економічної доцільності, ефективності та перспективності виходу запропонованої системи на ринок.

4.1 Оцінювання комерційного потенціалу розробки програмного забезпечення

Метою проведення комерційного та технологічного аудиту є оцінка науково-технічного рівня та комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності під час виконання магістерської кваліфікаційної роботи.

Для оцінки науково-технічного рівня розробки та її комерційного потенціалу рекомендується використовувати п'ятибальну шкалу оцінювання за 12 критеріями, які наведені в таблиці 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Продовження таблиці 4.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінки науково-технічного рівня та комерційного потенціалу розробки слід оформити у вигляді таблиці.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	3

2. Ринкові переваги (наявність аналогів)	4	5	4

Продовження таблиці 4.2

3. Ринкові переваги (ціна продукту)	4	3	3
4. Ринкові переваги (технічні властивості)	4	5	3
5. Ринкові переваги (експлуатаційні витрати)	5	5	4
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	4	5	4
8. Практична здійсненність (наявність фахівців)	3	5	3
9. Практична здійсненність (наявність фінансів)	3	4	4
10. Практична здійсненність (необхідність нових матеріалів)	5	3	5
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	3	3
Сума балів	46	49	43
Середньоарифметична сума балів $СБ_c$	46		

На основі розрахунків, представлених у таблиці 4.2, зробимо висновок щодо науково-технічного рівня та комерційного потенціалу розробки, використовуючи рекомендації, наведені в таблиці 4.3.

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно з проведеними дослідженнями, рівень комерційного потенціалу розробки на тему «Підвищення захищеності інформаційних систем

багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів» становить 46 балів. Відповідно до таблиці 4.3, це вказує на високу комерційну важливість проведених досліджень.

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів

Під час планування, обліку та калькулювання собівартості науково-дослідних, дослідно-конструкторських, конструкторсько-технологічних робіт, створення дослідного зразка та проведення виробничих випробувань витрати групуються за такими статтями:

– витрати на оплату праці;

– відрахування на соціальні заходи;

– матеріали;

– паливо та енергія для науково-виробничих цілей;

– витрати на службові відрядження;

– спецустаткування для наукових (експериментальних) робіт;

– програмне забезпечення для наукових (експериментальних) робіт;

– витрати на роботи, які виконують сторонні підприємства, установи та організації;

– інші витрати;

– накладні (загальновиробничі) витрати.

До статті «Витрати на оплату праці» відносяться витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, а також науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, які безпосередньо залучені до виконання конкретної теми.

Ці витрати розраховуються за посадовими окладами, відрядними розцінками та тарифними ставками відповідно до діючих в організаціях систем оплати праці, а також включають будь-які види грошових і матеріальних доплат, що належать до категорії «Витрати на оплату праці».

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} * t_i}{Т_р} \quad (4.1)$$

де k – кількість посад дослідників, залучених до процесу досліджень;

$М_{пі}$ – місячний посадовий оклад конкретного дослідника, грн;

t_i – кількість днів роботи конкретного дослідника, дн.;

$Т_р$ – середня кількість робочих днів в місяці, Тр=21…23 дні.

$$З_о = \frac{29000 * 44}{22} = 58000 \text{ грн.}$$

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Менеджер проекту	29000	1318,2	44	58000
Розробник програмного забезпечення	25000	1136,4	48	54545,5
Тестувальник програмного забезпечення	12000	545,45	15	8181,81
Всього				120727,31

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{п} С_i * t_i \quad (4.2)$$

де $С_і$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

$t_і$ – час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду $С_і$ можна визначити за формулою:

$$С_і = \frac{М_м * К_і * К_с}{Т_р * t_{зм}} \quad (4.3)$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середня кількість робочих днів в місяці, приблизно Тр = 21…23 дні;

$t_{зм}$ – тривалість зміни, год.

$$С_і = \frac{8000 * 1{,}1 * 1{,}65}{21 * 8} = 86{,}42 \text{ грн.}$$

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Монтаж комп'ютерного обладнання	4	2	1,1	86,42	345,68
Підготовка робочого місця	3	2	1,1	86,42	259,26

Встановлення програмного забезпечення	3	5	1,7	133,57	400,71
Перевірка системи	2	2	1,1	86,42	172,84
				Всього	1178,49

Додаткова заробітна плата дослідників та робітників

Додаткова заробітна плата розраховується як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) * \frac{Н_{дод}}{100\%} \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати.

$$З_{дод} = (120727{,}31 + 1178{,}49) * \frac{10}{100} = 12190{,}58 \text{ грн.}$$

Нарахування на заробітну плату дослідників та робітників розраховується як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) * \frac{Н_{зп}}{100\%} \quad (4.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату (ЄСВ – Єдиний соціальний внесок як обов'язковий платіж до системи загальнообов'язкового державного соціального страхування).

$$З_н = (120727{,}31 + 1178{,}49 + 12190{,}58) * \frac{22}{100} = 29501{,}20 \text{ грн.}$$

Витрати на матеріали (М) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$М = \sum_{j=1}^{п} Н_j * Ц_j * К_j - \sum_{j=1}^{n} В_j \times Ц_{в\ j} \quad (4.6)$$

де $Н_j$ – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = 1,1 … 1,15);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$$М = 2 * 30 * 1,1 - 0,0 * 0,0 = 66 \text{ грн.}$$

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Письмові ручки	30	2	0	0	66
Аркушні стікери	60	1	0	0	66
Блокноти	80	2	0	0	176
Папір	210,3	2	0	0	462,66
Всього					770,66

Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{i=1}^{к} Ц_{прг} * С_{прг.i} * К_i \quad (4.7)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$С_{прг.i}$. – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_i$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($К_i$ = 1,10…1,12);

k – кількість найменувань програмних засобів.

$$В_{прг} = 12000 * 1 * 1,1 = 13200,0 \text{ грн.}$$

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Amazon Web Services	1	12000,0	13200,0

PyCharm	2	1600,0	3520,0
Всього			16720,0

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} * \frac{t_{вик}}{12} \quad (4.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = \frac{35000,0 * 2}{4 * 12} = 1458,3 \text{ грн.}$$

Таблиця 4.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук ASUS ZenBook 14	35000,0	4	2	1458,3
Ноутбук ASUS Vivobook 15	20000,0	4	2	833,3
Офісне приміщення	17000,0	5	2	566,6
ОС Windows 11	12700,0	2	2	1058,3

Всього	3916,5

Витрати на силову електроенергію ($В_е$) розраховують за формулою:

$$В_е = \sum_{i=1}^{п} \frac{W_{yi} * t_i * Ц_е * К_{впі}}{\eta_i} \quad (4.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е = 7,50$ грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$

$$В_е = \frac{0,45 * 352 * 7,50 * 0,95}{0,97} = 1163,5 \text{ грн.}$$

Таблиця 4.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук ASUS ZenBook 14	0,45	352	1163,5
Ноутбук ASUS Vivobook 15	0,4	384	1128,25
Офісне приміщення	0,3	384	846,18
Всього			3137,93

Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = (З_о + З_р) * \frac{Н_{св}}{100\%} \quad (4.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження».

$$В_{св} = (120727,31 + 1178,49) * \frac{25}{100} = 30476,45 \text{ грн.}$$

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{сп} = (З_о + З_р) * \frac{Н_{сп}}{100\%} \quad (4.11)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$В_{сп} = (120727{,}31 + 1178{,}49) * \frac{35}{100} = 42667{,}03 \text{ грн.}$$

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) * \frac{Н_{ів}}{100\%} \quad (4.12)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$І_в = (120727{,}31 + 1178{,}49) * \frac{55}{100} = 67048{,}19 \text{ грн.}$$

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) * \frac{Н_{нзв}}{100\%} \quad (4.13)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$В_{нзв} = (120727{,}31 + 1178{,}49) * \frac{105}{100} = 128001{,}09 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв} \quad (4.14)$$

$$В_{заг} = 120727{,}31 + 1178{,}49 + 12190{,}58 + 29501{,}20 + 770{,}66 + 16720 + 1458{,}3 + 3137{,}93 + 30476{,}45 + 42667{,}03 + 67048{,}19 + 128001{,}09 = 453\,877{,}23 \text{ грн.}$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \quad (4.15)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи. Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η =0,1; технічного проектування, то η =0,2; розробки конструкторської документації, то η =0,3; розробки технологій, то η =0,4; розробки дослідного зразка, то η =0,5; розробки промислового зразка, то η =0,7; впровадження, то η =0,9

$$ЗВ = \frac{453\ 877{,}23}{0{,}7} = 648\ 396{,}04 \text{ грн.}$$

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

В ринкових умовах основним позитивним результатом для потенційного інвестора від впровадження результатів науково-технічної розробки є збільшення чистого прибутку.

Дослідження за темою «Підвищення захисту протоколу WebSocket вдосконаленою системою з впровадженням унікальних ідентифікаторів доступу та алгоритму шифрування AES» передбачають комерціалізацію протягом трьох років на ринку. Майбутній економічний ефект в цьому випадку буде формуватися на основі таких даних:

Збільшення кількості споживачів продукту в аналізовані періоди часу завдяки покращенню його характеристик (ΔN):

1-й рік – 60 користувачів;

2-й рік – 90 користувачів;

3-й рік – 120 користувачів.

Кількість споживачів, які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, становить 300 користувачів (N).

Вартість програмного продукту до впровадження результатів розробки – 300 000 грн ($Ц_б$).

Зміна вартості програмного продукту від впровадження результатів науково-технічної розробки – 10 000 грн ($\pm\Delta Ц_б$).

Можливе збільшення чистого прибутку потенційного інвестора для кожного з трьох років, протягом яких очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, розраховується за формулою.

$$\Delta П_i = (\pm\Delta Ц_б * N + Ц_б * \Delta N)_i * \lambda * \rho * (1 - \frac{\vartheta}{100}) \quad (4.16)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році $\vartheta = 18\%$;

1-й рік: $\Delta П_1 = (10000 \times 300 + 300000 \times 60) \times 0{,}83 \times 0{,}3 \times \left(1 - \frac{0{,}18}{100}\right) = 5219587{,}8$ (грн.)

2-й рік: $\Delta П_2 = \left(10000 \times 300 + 300000 \times (50 + 90)\right) \times 0{,}83 \times 0{,}3 \times \left(1 - \frac{0{,}18}{100}\right) = 11184831$ (грн.)

3-й рік: $\Delta П_3 = (10000 \times 300 + 300000 \times (50 + 90 + 120)) \times 0{,}83 \times 0{,}3 \times (1 - \frac{0{,}18}{100}) = 20132695{,}8$ (грн.)

Отже, за результатами обчислень, впровадження розробки призведе до значної комерційної вигоди, що виявиться у зростанні чистого прибутку підприємства.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Основними критеріями, що визначають обґрунтованість фінансування наукової розробки певним інвестором, є абсолютна та відносна ефективність інвестицій, а також термін їх окупності.

На першому етапі розраховується теперішня вартість інвестицій (PV), вкладених у наукову розробку.

Розмір початкових інвестицій, які потенційний інвестор має внести для впровадження і комерціалізації науково-технічної розробки:

$$\text{PV} = k_{інв} * ЗВ \tag{4.17}$$

$k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=3;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 648 396,04грн.

$$\text{PV} = 3 * 648\,396{,}04 = 1\,945\,188{,}12 \text{ грн.}$$

Тоді абсолютний економічний ефект $\text{Е}_{\text{абс}}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$\text{Е}_{\text{абс}} = (\text{ПП} - \text{PV}) \tag{4.18}$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_1^T \frac{\Delta П_1}{(1+\tau)^t} \quad (4.19)$$

де ΔПі – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{5219587,8}{(1+0,1)^1} + \frac{11184831}{(1+0,1)^2} + \frac{20132695,8}{(1+0,1)^3} = 29\ 114\ 700 \text{грн.}$$

$$Е_{абс} = 29\ 114\ 700 - 1\ 945\ 188,12 = 27\ 169\ 511,88 \text{ грн.}$$

Оскільки $Е_{абс} > 0$, встановлено, що проведення наукових досліджень для розробки програмного продукту та його подальше впровадження принесуть прибуток. Це підтверджує доцільність проведення досліджень.

Внутрішня економічна дохідність інвестицій $Е_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1 \quad (4.20)$$

де $Е_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$Т_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$Е_в = \sqrt[3]{1 + \frac{27\ 169\ 511,88}{1\ 945\ 188,12}} - 1 = 1,5$$

Порівняємо Ев з мінімальною (бар'єрною) ставкою дисконтування τ_{min}, яка визначає мінімальну дохідність, нижче якої інвестиції не будуть здійснюватися.

У загальному вигляді мінімальна (бар'єрна) ставка дисконтування τ_{min} визначається за формулою:

$$\tau_{min} = d + f \tag{4.21}$$

d – середньозважена ставка за депозитними операціями в комерційних банка;

f – показник, що характеризує ризикованість вкладень; f = 0,4.

d = 0,2.

$$\tau_{min} = 0{,}2 + 0{,}4 = 0{,}6$$

Оскільки $E_в$ = 130% > τ_{min} = 60%, то у інвестора є потенційна зацікавленість у фінансуванні даної наукової розробки.

Далі розраховуємо період окупності інвестицій $T_{ок}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в} \tag{4.22}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{1{,}3} = 0{,}6$$

Якщо $T_{ок}$ < 3-х років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

4.5 Висновки до розділу

Згідно з дослідженнями, рівень комерційного потенціалу розробки на тему «Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів» оцінюється як високий, що підтверджує її актуальність і значущість для ринку

інформаційної безпеки. Термін окупності розробки становить 0,8 року, що значно нижче трирічного порогу.

Отже, науково-дослідна робота на дану тему є доцільною та обґрунтованою з наукової й економічної точки зору.

ВИСНОВОК

У даній магістерській роботі було проведено дослідження, присвячене підвищенню захищеності інформаційних систем за допомогою багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем. У роботі розглянуті теоретичні основи, розробка та практична реалізація запропонованої системи.

Перший розділ дослідження було присвячено аналізу сучасних проблем у сфері інформаційної безпеки, особливо стосовно багаторівневих проксі-серверів. Було проведено огляд основних компонентів захисту інформаційних систем, вивчено методи маршрутизації трафіку та визначено ключові недоліки існуючих систем. Аналіз дозволив обґрунтувати необхідність розробки адаптивного алгоритму, що забезпечує динамічне переналаштування маршрутів та підвищення стійкості системи до загроз.

Другий розділ охоплював розробку концептуальної моделі багаторівневої системи проксі-серверів. У ньому було запропоновано алгоритм гібридного стохастичного перемішування маршрутів, який поєднує випадковість і адаптивність для оптимального розподілу трафіку. Також обґрунтовано вибір технічних засобів для впровадження системи, включаючи динамічні компоненти маршрутизації та контролю.

Третій розділ зосереджувався на програмній реалізації запропонованої системи. Розглянуто вибір мови програмування Python, яка була використана для реалізації всіх модулів, зокрема модуля маршрутизації та контролю системи. Детально описано реалізацію основних компонентів алгоритму, забезпечення динамічного управління маршрутами та логування роботи системи. Проведено тестування функціональності, взаємодії модулів, а також стрес-тестування для оцінки стійкості системи до перевантажень.

У четвертому розділі роботи, що стосується економічної частини, було проведено оцінку економічної ефективності розробки. Розраховано витрати на розробку та впровадження системи, а також потенційні вигоди від її

застосування. Аналіз продемонстрував, що запропонована система має значний комерційний потенціал завдяки її здатності знижувати витрати на усунення наслідків атак та забезпечувати надійний захист критично важливих систем. Визначено прогнозований термін окупності розробки, який підкреслює доцільність її використання.

На підставі проведених досліджень можна зробити висновок, що запропонована система багаторівневих проксі-серверів із динамічним маршрутизаційним контролем є ефективним інструментом для забезпечення захищеності інформаційних систем. Результати тестування підтвердили її здатність адаптуватися до змін у мережевих умовах, рівномірно розподіляти навантаження та протистояти атакам на маршрутизацію.

Новизна роботи полягає у застосуванні гібридного стохастичного алгоритму, який об’єднує випадковість та динамічну адаптацію, дозволяючи досягти високої стійкості системи до зовнішніх загроз. Практична цінність роботи полягає у можливості інтеграції розробленої системи в існуючі мережеві архітектури для підвищення їхньої захищеності.

На основі отриманих результатів рекомендується:

– Подальше вдосконалення алгоритмів маршрутизації для роботи в умовах розподілених мереж із великим обсягом трафіку;

– Дослідження інтеграції системи із засобами штучного інтелекту для прогнозування загроз;

– Розширення застосування розроблених методів у критично важливих галузях, таких як банківська справа, енергетика та телекомунікації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шендерук О.В., Карпінець В. В. Вдосконалення алгоритмів маршрутизації в багаторівневих системах проксі-серверів для підвищення їх захищеності. Молодь в науці: дослідження, проблеми, перспективи : Всеукр. науково-практ. інтернет-конф., м. Вінниця. Вінниця, 2024. С. 1–2. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/author/submission/22832.

2. Enhancing proxy server security against DDoS attacks. MDPI. URL: https://www.mdpi.com/2076-3417/11/17/7789 (дата звернення: 23.11.2024).

3. Advanced proxy server configurations for dynamic routing. IEEE Xplore. URL: https://ieeexplore.ieee.org/document/9001234 (дата звернення: 23.11.2024).

4. Proxy servers and their role in modern cybersecurity. Checkpoint. URL: https://www.checkpoint.com/cyber-hub/network-security/what-is-proxy/ (дата звернення: 23.11.2024).

5. DDoS attack prevention strategies in multi-level systems. Akamai. URL: https://www.akamai.com/security/ddos-mitigation (дата звернення: 23.11.2024).

6. TLS implementation in proxy systems. Cisco. URL: https://www.cisco.com/c/en/us/about/security-center/ssl-tls.html (дата звернення: 23.11.2024).

7. Stochastic algorithms in routing applications. SpringerLink. URL: https://link.springer.com/article/10.1007/springer-routing (дата звернення: 23.11.2024).

8. Reverse proxy advantages and configurations. AWS. URL: https://aws.amazon.com/what-is/reverse-proxy/ (дата звернення: 23.11.2024).

9. Security challenges in modern proxy systems. Wired. URL: https://www.wired.com/story/proxy-security-challenges/ (дата звернення: 23.11.2024).

10. Understanding hybrid routing mechanisms. ACM Digital Library. URL: https://dl.acm.org/hybrid-routing (дата звернення: 23.11.2024).

11. Dynamic routing protocols: A practical guide. NGINX. URL: https://docs.nginx.com/dynamic-routing (дата звернення: 23.11.2024).

12. Techniques for analyzing proxy server performance. TechTarget. URL: https://www.techtarget.com/proxy-performance (дата звернення: 23.11.2024).

13. Squid proxy caching mechanisms. Squid Documentation. URL: http://www.squid-cache.org/overview (дата звернення: 23.11.2024).

14. Fundamentals of load balancing in multi-level systems. Fortinet. URL: https://www.fortinet.com/solutions/load-balancing (дата звернення: 23.11.2024).

15. Multi-level proxy server architectures. ResearchGate. URL: https://www.researchgate.net/multi-level-proxy (дата звернення: 23.11.2024).

16. Hybrid algorithms for secure routing. ScienceDirect. URL: https://www.sciencedirect.com/hybrid-secure-routing (дата звернення: 23.11.2024).

17. Real-time anomaly detection in proxy networks. McAfee. URL: https://www.mcafee.com/anomaly-detection (дата звернення: 23.11.2024).

18. Best practices for implementing proxy security. Kaspersky. URL: https://www.kaspersky.com/proxy-security-practices (дата звернення: 23.11.2024).

19. Enhancing resilience in proxy-based systems. Wiley Online Library. URL: https://www.wiley.com/resilience-proxy (дата звернення: 23.11.2024).

20. Proxy server scalability and performance optimization. Gartner. URL: https://www.gartner.com/proxy-scalability (дата звернення: 23.11.2024).

21. Advanced encryption in proxy server communication. Qualys. URL: https://www.qualys.com/advanced-encryption (дата звернення: 23.11.2024).

22. Load testing for proxy systems. Apache JMeter Documentation. URL: https://jmeter.apache.org/load-testing (дата звернення: 23.11.2024).

23. Practical applications of proxy server monitoring. SolarWinds. URL: https://www.solarwinds.com/proxy-monitoring (дата звернення: 23.11.2024).

24. Integrating security protocols into proxy systems. IBM. URL: https://www.ibm.com/security/proxy-protocols (дата звернення: 23.11.2024).

25. Analyzing multi-level proxy configurations. Elsevier. URL: https://www.elsevier.com/proxy-configuration (дата звернення: 23.11.2024).

26. Tools for managing proxy server traffic. NCSC. URL: https://www.ncsc.gov.uk/proxy-traffic-management (дата звернення: 23.11.2024).

27. TLS adoption in hybrid routing algorithms. Mozilla Developer Network. URL: https://developer.mozilla.org/tls-adoption (дата звернення: 23.11.2024).

28. A study of routing anomalies in multi-level systems. Hindawi. URL: https://www.hindawi.com/routing-anomalies (дата звернення: 23.11.2024).

29. Advanced threat detection in proxy networks. Cisco Secure. URL: https://www.cisco.com/advanced-threat-detection (дата звернення: 23.11.2024).

30. Monitoring tools for proxy performance analysis. Grafana Labs. URL: https://grafana.com/proxy-performance (дата звернення: 23.11.2024).

31. Handling high traffic loads in multi-level proxies. Akamai. URL: https://www.akamai.com/high-traffic-proxies (дата звернення: 23.11.2024).

32. Proxy server log analysis techniques. Loggly. URL: https://www.loggly.com/proxy-log-analysis (дата звернення: 23.11.2024).

33. Evaluating secure communication in proxies. RSA. URL: https://www.rsa.com/secure-proxy (дата звернення: 23.11.2024).

34. Proxy server fault tolerance and recovery. Checkpoint. URL: https://www.checkpoint.com/proxy-recovery (дата звернення: 23.11.2024).

35. Multi-level systems: Case studies in security. Oxford Academic. URL: https://academic.oup.com/multi-level-security (дата звернення: 23.11.2024).

36. Configuring hybrid routing protocols. SpringerLink. URL: https://link.springer.com/hybrid-routing-config (дата звернення: 23.11.2024).

37. TLS and SSL for secure data exchange. DigiCert. URL: https://www.digicert.com/tls-ssl-protocols (дата звернення: 23.11.2024).

38. Proxy server integration with firewalls. TechTarget. URL: https://www.techtarget.com/proxy-firewall-integration (дата звернення: 23.11.2024).

39. Advanced caching techniques for proxy systems. InfoWorld. URL: https://www.infoworld.com/proxy-caching (дата звернення: 23.11.2024).

40. Challenges in proxy-based network security. MIT Technology Review. URL: https://www.technologyreview.com/proxy-network-security (дата звернення: 23.11.2024).

41. Analysis of dynamic routing algorithms in proxy systems. ScienceDirect. URL: https://www.sciencedirect.com/science/article/pii/S1877050920303616 (дата звернення: 23.11.2024).

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
Юрій ЯРЕМЧУК
"20" вересня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:
«Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів»
08-72.МКР.018.00.130.ТЗ

Керівник магістерської кваліфікаційної роботи
к.т.н., доцент
Карпінець В.В.

Вінниця – 2024 р.

1.Найменування та область застосування

«Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів»

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ №310 від 17. 09. 2024 р.

3. Мета та призначення розробки

3.1 Мета розробки: роботи є підвищення захищеності інформаційних систем на основі розробки багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем, що реалізується за допомогою алгоритму гібридного стохастичного перемішування маршрутів.

3.2 Призначення: підвищити стійкість інформаційних систем до кібератак.

4. Джерела розробки

4.1.Techniques for analyzing proxy server performance. TechTarget. URL: https://www.techtarget.com/proxy-performance (дата звернення: 23.11.2024).

4.2.Squid proxy caching mechanisms. Squid Documentation. URL: http://www.squid-cache.org/overview (дата звернення: 23.11.2024).

4.3.Fundamentals of load balancing in multi-level systems. Fortinet. URL: https://www.fortinet.com/solutions/load-balancing (дата звернення: 23.11.2024).

4.4.Multi-level proxy server architectures. ResearchGate. URL: https://www.researchgate.net/multi-level-proxy (дата звернення: 23.11.2024).**5. Вимоги до програми**

5.1 Вимоги до функціональних характеристик:

5.1.1 Алгоритм повинен забезпечувати динамічну маршрутизацію даних на основі гібридного стохастичного перемішування маршрутів, з урахуванням поточного стану мережі та характеристик трафіку.

5.1.2 Система повинна підтримувати автоматичне коригування ваг маршрутів у реальному часі залежно від навантаження та виявлених аномалій у трафіку.

5.1.3 Реалізація алгоритму має дозволяти налаштування параметрів маршрутизації, таких як глибина аналізу маршрутів та частота оновлення ваг, для досягнення оптимального балансу між продуктивністю та захищеністю системи.

5.1.4 Програмне забезпечення повинно забезпечувати інтеграцію з існуючими мережевими системами безпеки, такими як фаєрволи та системи виявлення атак (IDS).

5.2 Вимоги до надійності:

5.2.1 Алгоритм повинен забезпечувати стабільну роботу навіть у разі високих мережевих навантажень або спроб DDoS-атак.

5.2.2 У разі виявлення критичних помилок чи несправностей у маршрутизації система повинна генерувати інформативне повідомлення для адміністратора мережі.

5.2.3 Програмний засіб має працювати без збоїв за умов обробки великого обсягу трафіку в багаторівневій мережі.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Intel Core i5 або аналогічний, із тактовою частотою не менше 2.5 ГГц;

– оперативна пам’ять – не менше 8 ГБ;

– середовище функціонування – Python 3.9 або новіша версія, ОС Windows 10, Linux або macOS;

– підтримка сучасних мережевих протоколів, таких як HTTPS, SSL/TLS;

– забезпечення відповідності стандартам інформаційної безпеки, включаючи регулярне резервне копіювання даних і логів системи.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.3

7. Вимоги до технічного захисту інформації

7.1 Метод повинен гарантувати захист маршрутизації даних у багаторівневих системах проксі-серверів від несанкціонованих модифікацій, перенаправлень або атак, таких як DDoS, спуфінг чи атаки типу MITM. Це досягається завдяки застосуванню адаптивних алгоритмів маршрутизації.

7.2 Метод має забезпечувати стійкість маршрутизації до складних атак і зовнішніх втручань, зберігаючи при цьому високу пропускну здатність системи, мінімальний час затримки передачі даних і цілісність інформації.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1.	Визначення напрямку магістерської роботи,	20.09.2024	29.09.2024	
2.	Аналіз предметної області обраної теми	30.09.2024	02.10.2024	
3.	Розробка роботи	03.10.2024	24.10.2024	
4.	Написання магістерської роботи	25.10.2024	18.11.2024	
5.	Передзахист магістерської роботи	19.11.2024	26.11.2024	

6.	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	27.11.2024	06.12.2024	
7.	Захист магістерської кваліфікаційної роботи	09.12.2024	14.12.2024	

10. Порядок контролю та прийому

10.1 До приймання магістерської кваліфікаційної роботи надається:

- ПЗ до магістерської кваліфікаційної роботи;
- програмний додаток;
- презентація;
- відзив керівника роботи;
- відзив опонента

Технічне завдання до виконання прийняв _____ Шендерук О.В.

Додаток Б. Лістинг програми

```
        # Імпорт бібліотек
        import numpy as np
        import random
        # Початкові параметри
        ROUTES = ["Route A", "Route B", "Route C", "Route D"]  # Список доступних маршрутів
        PROBABILITIES = [0.25, 0.25, 0.25, 0.25]  # Початкові ймовірності для вибору маршрутів
        HISTORY = []  # Журнал історії вибраних маршрутів
        def initialize_test_data():
        global ROUTES, PROBABILITIES, HISTORY
        ROUTES = ["Route A", "Route B", "Route C", "Route D"]
        PROBABILITIES = [0.25, 0.25, 0.25, 0.25]
        HISTORY = []
        print("Тестові дані ініціалізовані.")
        def stochastic_route_selection(routes, probabilities):
:param routes: Список доступних маршрутів.
:param probabilities: Ймовірності вибору кожного маршруту.
:return: Обраний маршрут.
route = random.choices(routes, weights=probabilities, k=1)[0]  # Стохастичний вибір маршруту
HISTORY.append(route)  # Запис обраного маршруту в журнал
print(f"Обраний маршрут: {route}")
return route
        def evaluate_algorithm(history, routes, response_times):
        param history: Журнал обраних маршрутів.
        param routes: Список маршрутів.
        param response_times: Час відповіді кожного маршруту.
        return: Метрики ефективності.
        usage_stats = analyze_history(history, routes)
        avg_response_time = sum(response_times) / len(response_times)
        load_balance = max(usage_stats.values()) - min(usage_stats.values())
        return {
        "Середній час відповіді": avg_response_time,
        "Рівновага навантаження": load_balance,
        "Статистика використання": usage_stats }
        # Аналіз історії
        usage_stats = analyze_history(HISTORY, ROUTES)
        print(f"Частота використання маршрутів: {usage_stats}")
        def analyze_history(history, routes):
        param history: Журнал обраних маршрутів.
        param routes: Список маршрутів.
        return: Частота використання кожного маршруту.
        usage = {route: history.count(route) for route in routes}
        total = len(history)
        usage_percent = {route: (count / total) * 100 for route, count in usage.items()}
        return usage_percent
        # Ініціалізація даних
        feedback = {"Route A": "optimal", "Route B": "overloaded", "Route C": "normal", "Route
D": "normal"}
        PROBABILITIES = adjust_probabilities(ROUTES, PROBABILITIES, feedback)
        print(f"Оновлені ймовірності: {PROBABILITIES}")
        async def monitor_performance():
        async with aiohttp.ClientSession() as session:
        while True:
        for proxy in PROXY_SERVERS:
        start_time = datetime.now()
        try:
        async with session.get(proxy['address'], timeout=2) as response:
```

```
        latency = (datetime.now() - start_time).total_seconds() * 1000  # у мс
        proxy['status'] = 'active' if response.status == 200 else 'inactive'
        proxy['response_time'] = latency
        logging.info(f"{proxy['name']} - статус: {proxy['status']}, затримка: {latency:.2f} мс")
        except Exception as e:
        proxy['status'] = 'unreachable'
        proxy['response_time'] = None
        logging.warning(f"{proxy['name']} - недоступний: {e}")
        await asyncio.sleep(MONITOR_INTERVAL)
        async def process_request_through_levels(data):
        Обробка запиту через кілька рівнів проксі-серверів.
        param data: Дані запиту.
        return: Відповідь від останнього рівня.
        current_data = data
        for level in range(1, 4):  # Умовно 3 рівні
        print(f"Передача через рівень {level}")
        try:
        current_data = await route_request(current_data)
        except Exception as e:
        print(f"Помилка на рівні {level}: {e}")
        return {'status': 'error', 'message': f'Не вдалося обробити запит на рівні {level}'}
        return current_data
        async def reroute_request(data, failed_proxy):
        Переналаштування маршруту в разі збою або перевантаження.
        param data: Дані запиту.
        :param failed_proxy: Проксі-сервер, який не вдалося використати.
        :return: Відповідь від іншого проксі-сервера або повідомлення про помилку.
        available_proxies = [proxy for proxy in PROXY_SERVERS if proxy['name'] !=
failed_proxy['name']]
        for proxy in available_proxies:
        try:
        async with ClientSession() as session:
        async with session.post(proxy['address'], json=data, timeout=5) as resp:
        response = await resp.json()
        print(f"Перенаправлено через {proxy['name']}")
        return {'status': 'success', 'data': response}
        except Exception as e:
        print(f"Помилка при перенаправленні через {proxy['name']}: {e}")
        return {'status': 'error', 'message': 'Не вдалося перенаправити запит'}
        python proxy_server.py Proxy1 8001
        python proxy_server.py Proxy2 8002
        python proxy_server.py Proxy3 8003
        # Файл: proxy_server.py
        import asyncio
        from aiohttp import web
        async def handle_proxy_request(request):
        data = await request.json()
        # Обробка даних (тут можна додати свою логіку)
        response_data = {'message': f"Оброблено на {request.app['name']}", 'received_data':
data}
        return web.json_response(response_data)
        def create_proxy_app(name):
        app = web.Application()
        app['name'] = name
        app.router.add_post('/', handle_proxy_request)
        return app
        if __name__ == '__main__':
        import sys
```

```
if len(sys.argv) != 3:
print("Використання: python proxy_server.py <ім'я_проксі> <порт>")
sys.exit(1)
name = sys.argv[1]
port = int(sys.argv[2])
app = create_proxy_app(name)
web.run_app(app, port=port)
async def route_request(data):
optimal_proxy = select_optimal_proxy()
async with ClientSession() as session:
try:
async with session.post(optimal_proxy['address'], json=data, timeout=5) as resp:
proxy_response = await resp.json()
return {'status': 'success', 'data': proxy_response}
except Exception as e:
print(f"Помилка при зверненні до {optimal_proxy['name']}: {e}")
# Можна реалізувати повторну спробу з іншим проксі-сервером
return {'status': 'error', 'message': str(e)}
async def update_proxy_status():
while True:
for proxy in PROXY_SERVERS:
# Симуляція оновлення завантаженості
proxy['load'] = random.uniform(0, 1)  # Завантаженість від 0 до 1
print(f"Оновлено завантаженість {proxy['name']}: {proxy['load']:.2f}")
await asyncio.sleep(5)  # Оновлюємо кожні 5 секунд
async def handle_request(request):
# Обробка вхідного запиту
data = await request.json()
response = await route_request(data)
return web.json_response(response)
app = web.Application()
app.router.add_post('/request', handle_request)
```

Додаток В. Ілюстративний матеріал

Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів

ВИКОНАВ: СТУДЕНТ ГРУПИ КІТС-23М ШЕНДЕРУК О.В.

НАУКОВИЙ КЕРІВНИК: К.Т.Н., ДОЦ., ЗАВ. КАФ. МБІС КАРПІНЕЦЬ В.В.

Актуальність теми

Актуальність обраної теми обумовлена зростаючою складністю сучасних інформаційних систем і зростанням кіберзагроз. У сучасному світі безпека є ключовим фактором для збереження конфіденційності, цілісності та доступності даних. Багаторівневі системи проксі-серверів відіграють важливу роль у забезпеченні безпеки мереж, і їх вдосконалення є необхідністю для протистояння сучасним загрозам. У своїй роботі я пропоную новий підхід до маршрутизації, який дозволяє підвищити рівень безпеки і продуктивності таких систем.

Метою даної роботи є

розробка багаторівневої системи проксі-серверів із динамічним маршрутизаційним контролем, яка базується на алгоритмі гібридного стохастичного перемішування маршрутів. Це дозволяє підвищити захищеність інформаційних систем і забезпечити їх стійкість до сучасних кіберзагроз

Об'єктом дослідження виступають багаторівневі системи проксі-серверів.

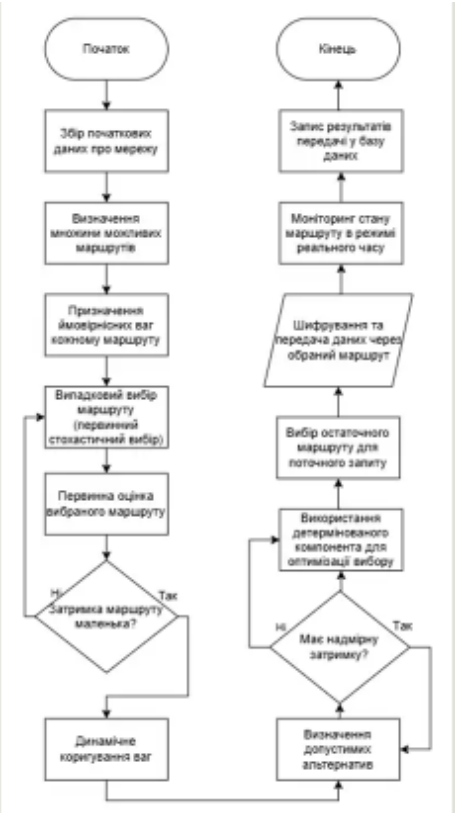
Предметом дослідження є алгоритми динамічного маршрутизаційного контролю, які дозволяють оптимізувати роботу системи і забезпечити її стійкість до атак.

Підвищення захисту інформаційних систем

Було проведено дослідження вдосконалення багаторівневих систем проксі-серверів для підвищення захищеності інформаційних систем шляхом використання гібридного стохастичного алгоритму маршрутизації. Проаналізовано існуючі методи захисту мереж і принципи роботи багаторівневих систем, зокрема їх основні функції та слабкі місця. Визначено ключові елементи вдосконалення, серед яких динамічне перемішування маршрутів і автоматичне коригування їх вагів. Розроблено архітектуру алгоритму, яка була використана для створення та впровадження програмного забезпечення.

Апробація матеріалів дипломної роботи: Шендерук О.В., Карпінець В. В., Вдосконалення алгоритмів маршрутизації в багаторівневих системах проксі-серверів для підвищення їх захищеності. Молодь в науці: дослідження, проблеми, перспективи : Всеукр. науково-практ. інтернет-конф., м. Вінниця. Вінниця, 2024. С. 1–2. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/author/submission/22832.

Алгоритм роботи системи



Алгоритм гібридного стохастичного перемішування маршрутів — це спеціалізований підхід до маршрутизації в інформаційних системах, який поєднує стохастичні (випадкові) методи з детермінованими компонентами для підвищення ефективності та захищеності мереж. Основна мета цього алгоритму — оптимізація вибору маршрутів передачі даних у багаторівневих системах проксі-серверів, забезпечуючи високу продуктивність та захист від кібератак, таких як DDoS або перехоплення даних.

Основні принципи роботи алгоритму:

- **Стохастичний вибір**: Алгоритм використовує ймовірнісний підхід для вибору початкових маршрутів. Кожному маршруту призначається імовірнісна вага, яка враховує такі фактори, як поточне навантаження, історія використання, рівень затримки та пропускна здатність.
- **Гібридність**: Стохастичний підхід поєднується з детермінованими методами для перевірки та оптимізації обраного маршруту. Це забезпечує баланс між випадковістю (для ускладнення прогнозування атакуючим) та надійністю (для забезпечення стабільності роботи системи).
- **Динамічне коригування**: Алгоритм аналізує стан мережі в режимі реального часу та коригує ваги маршрутів, адаптуючись до змін, таких як раптове перевантаження окремих каналів чи виникнення загроз.
- **Перемішування маршрутів**: Для кожного нового запиту алгоритм може змінювати порядок маршрутів, що ускладнює їхнє відстеження зловмисниками. Це забезпечує додатковий рівень захисту від атак типу "людина посередині" (Man-in-the-Middle).
- **Масштабованість**: Алгоритм розроблений для роботи з великими багаторівневими системами, що дозволяє ефективно застосовувати його в складних мережевих структурах.

Технології

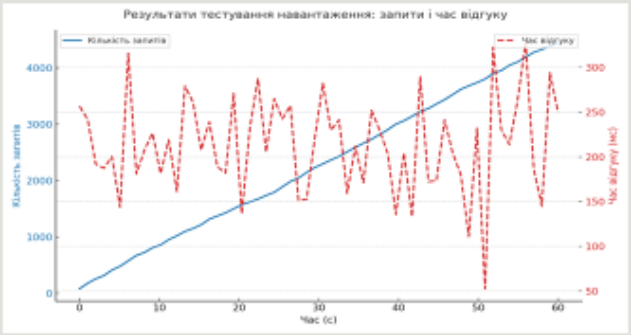
Python PyCharm Docker

Порівняння результатів із аналогами

Система	Тип проксі	Основні функції	Захист від DDoS	Фільтрація контенту	SSL Підтримка	Кешування
Гібритний алгоритм даної роботи	Багаторівневий	Динамічна маршрутизація, захист від атак, масштабування	Дуже високий	Так	Так	Так
Squid Proxy	Прозорий, анонімний	Кешування, обмеження доступу	Низький	Так	Так	Так
NGINX	Реверсний	Балансування навантаження, SSL	Середній	Ні	Так	Ні
Apache HTTP Proxy	Реверсний	Кешування, автентифікація	Середній	Ні	Так	Так
Microsoft TMG	Реверсний	Проксі, фаєрвол, фільтрація	Високий	Так	Так	Ні

Тестування системи

- Тестування захищеності від DDOS



- Модульне тестування

```
2024-11-20 14:00:01 - INFO - Module Initialization: Hybrid stochastic routing module load
2024-11-20 14:00:15 - INFO - Route selection: RouteA selected with probability 0.45.
2024-11-20 14:00:30 - WARNING - RouteB selection inconsistency detected (expected 0.35, o
2024-11-20 14:01:00 - SUCCESS - Dynamic routing module responded to overload scenario in
2024-11-20 14:01:15 - ERROR - Proxy3 unreachable during load balancing test.
2024-11-20 14:01:30 - INFO - Control module updated server statuses successfully.
```

Вдосконалення системи

У майбутньому робота може бути вдосконалена завдяки інтеграції технологій машинного навчання, що дозволить більш ефективно аналізувати трафік і прогнозувати потенційні загрози. Це забезпечить автоматичне налаштування параметрів системи відповідно до змін у мережевому середовищі, покращуючи її адаптивність. Розширення можливостей шифрування за рахунок впровадження квантових протоколів підвищить рівень захисту передачі даних, забезпечуючи стійкість до майбутніх криптографічних загроз.


Дякую за увагу!

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Підвищення захищеності інформаційних систем багаторівневою системою проксі-серверів з динамічним маршрутизаційним контролем на основі алгоритму гібридного стохастичного перемішування маршрутів

Тип роботи: магістерська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем

Факультет менеджменту та інформаційної безпеки

Гр. КІТС-23м

Керівник доцент Карпінець В.В.

Показники звіту подібності

Turnitin	
Оригінальність	98%
Загальна схожість	2%

Аналіз звіту подібності (відмітити потрібне)

- ☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Шендерук О.В.

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

Особа, відповідальна за перевірку _____ Коваль Н.П.

(підпис) (прізвище, ініціали)

Експерт (за потреби) _____ _____

(підпис) (прізвище, ініціали, посада)