

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Підходи до вирішення проблем продуктивності
проксі для протоколів HTTP та SOCKS5 для
випадку багатопортового безпарольного доступу»**

Виконав студент групи ПЗм-22-2
НОНІК Олександр Олександрович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
 ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
 ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
 КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
 програмного забезпечення
 Тетяна ВАКАЛЮК
 «18» вересня 2023 р.

ЗАВДАННЯ на кваліфікаційну роботу

Здобувач вищої освіти: **НОНІК Олександр Олександрович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**, к.т.н., доц., доцент
 кафедри інженерії програмного забезпечення

Тема роботи: **«Підходи до вирішення проблем продуктивності проксі для
 протоколів HTTP та SOCKS5 для випадку багатопортового безпарольного
 доступу»**,

затверджена Наказом закладу вищої освіти від «18» вересня 2023 р. № 479/с

Вихідні дані для роботи: академічні роботи студентів, а також вимоги до
 оформлення академічних робіт, алгоритми та методи первинної перевірки
 дипломних робіт у форматі PDF, елементи оформлення дипломних робіт, які
 підлягають автоматизованій перевірці.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів,
 що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	ЛОБАНЧИКОВА Н.М.		
2	ЛОБАНЧИКОВА Н.М.		
3	ЛОБАНЧИКОВА Н.М.		

РЕФЕРАТ

Випускна кваліфікаційна магістерська робота присвячена вивченню та вирішенню проблеми, з якою стикаються всі досить великі проксі-сервіси, які намагаються будувати роботу на загальнодоступних ресурсних рішеннях.

Основна проблема полягає в тому, що більшість користувачів використовують проксі для браузерів, таких як Chrome, які мають обмежену підтримку аутентифікації за логіном та паролем. Точніше кажучи, Chrome не впровадив підтримку такого типу авторизації протягом десяти років. Більшість сервісів обходять цю проблему, використовуючи авторизацію за IP-адресою та підходять "один порт - один проксі". Цей підхід в принципі допомагає вирішити проблему, але при вертикальному масштабуванні сервісів виникає серйозна проблема. Наприклад, при наявності 20 000 проксі стає дуже складно ефективно керувати 20 000 портами. І якщо додати розподіл за типами та статусами, то кількість портів збільшується до 40 000, що удвічі більше.

Було проведено дослідження, яке показало, що продуктивність знижується пропорційно кількості портів, що прослуховуються, а іноді навіть 2х збільшується при подальшому збільшенні кількості портів. Дослідження також виявило, що навіть 3 проксі можуть негативно вплинути на працездатність сервера, якщо вони слухають навіть тисячі портів.

Був знайдений спосіб вирішення подібної проблеми за допомогою використання ефективного вводу-виводу (eBPF) в ядрі Linux, а також додавання універсального обробника проксі, який може автоматично визначати, чи це SOCKS, HTTP або інший тип проксі.

Мета роботи проксі-реактора полягає у зміні вхідних заголовків пакетів, а вся схема роботи проксі-реактора базується на завантаженні невеликого, дуже легкого фрагмента коду в ядро Linux для динамічних змін. За цією концепцією, порт за замовчуванням може слухати необмежену кількість портів, більш точно, 65,000 портів і деякі інші цифри. Фізично у системі буде всього один сокет, який

обслуговуватиме цей порт, незалежно від того, яким чином відбуваються операції. Підхід спрямований на оптимізацію і розв'язання вищезазначених проблем.

В першому розділі описано та досліджено проблеми продуктивності проксі серверів. В другому розділі описано та досліджено готові рішення. В третьому розділі надано зроблено та описано застосування низькорівневих механізмів. В третьому розділі зроблено високорівневу частину системи обробки мережевого трафіку обробки протоколів.

Пояснювальна записка складається зі вступу, трьох розділів, переліку умовних позначень та списку використаної літератури. Записка дипломної роботи містить 14 ілюстрацій та 32 посилання на джерела бібліографії. Повний обсяг роботи становить 65 сторінкуи, з них 62 сторінок основного тексту.

КЛЮЧОВІ СЛОВА: SOCKS, SOCKS5, HTTP, HTTPS, GO, LINUX, IP, TCP, UDP

ABSTRACT

The final qualifying master's thesis is devoted to the study and solution of the problem faced by all fairly large proxy services that try to build work on publicly available resource solutions.

The main problem is that most users use proxies for browsers like Chrome, which have limited support for login and password authentication. To be more specific, Chrome hasn't implemented support for this type of authorization in a decade. Most services circumvent this problem by using IP-address authorization and a "one port - one proxy" approach. This approach, in principle, helps to solve the problem, but when scaling services vertically, a serious problem arises. For example, with 20,000 proxies, it becomes very difficult to effectively manage 20,000 ports. And if you add the distribution by types and statuses, the number of ports increases to 40,000, which is twice as much.

A study was done that showed that performance decreases proportionally to the number of ports being listened to, and sometimes even increases by 2x as the number of ports increases further. The study also found that even 3 proxies can negatively affect server performance if they are listening on even thousands of ports.

A solution to this problem was found by using efficient I/O (eBPF) in the Linux kernel and adding a universal proxy handler that can automatically determine whether it is a SOCKS, HTTP, or other type of proxy.

The purpose of the proxy reactor is to modify the incoming packet headers, and the entire proxy reactor scheme is based on uploading a small, very lightweight piece of code to the Linux kernel for dynamic changes. In this concept, a default port can listen to an unlimited number of ports, more precisely 65,000 ports and some other numbers. Physically, there will be only one socket in the system that will serve this port, regardless of how the operations take place. The approach is aimed at optimizing and solving the above-mentioned problems.

The first chapter describes and explores the performance issues of proxy servers. In the second section, ready-made solutions are described and studied.

In the third section, the application of low-level mechanisms is provided and described. In the third section, the high-level part of the network traffic processing protocol processing system is made.

The explanatory note consists of an introduction, three sections, a list of notations and a list of used literature. The thesis note contains 14 illustrations and 32 references to bibliographic sources. The full volume of the work is 65 pages, of which 62 pages are the main text.

KEYWORDS: SOCKS, SOCKS5, HTTP, HTTPS, GO, LINUX, IP, TCP, UDP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1. ОГЛЯД ПРОБЛЕМ ПРОДУКТИВНОСТІ ПРОКСІ-СЕРВЕРІВ	13
1.1 Основні аспекти проксі-серверів для протоколів HTTP та SOCKS5	13
1.2 Різновиди та Класифікація Проксі-Серверів	16
1.3 Переваги і недоліки використання проксі-серверів.....	20
1.4 Огляд основних протоколів проксі-серверів.	21
Висновок до 1 розділу.....	23
РОЗДІЛ 2. ПЕРЕВІРКА ТА ДОСЛІДЖЕННЯ ГОТОВИХ РІШЕНЬ	25
2.1 Огляд популярних рішень для проксі-серверів.	25
2.2 Зпроху: Всебічний Огляд та Налаштування.	26
2.3 Squid: Розширений Огляд та Конфігурація Проксі-Сервера.....	30
2.4 Використання Apache, налаштування та застосування на Ubuntu.....	33
Висновки до розділу 2.....	35
РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ І ЗАСТОСУВАННЯ РОЗРОБЛЕНИХ НИЗЬКОРІВНЕВИХ МЕХАНІЗМІВ	36
3.1 Загальний опис логіки та налаштування сервісу.....	36
3.2 Розробка низькорівневого алгоритму аналізу та маркування за допомогою eBPF..	38
3.3 Реалізація сигнатурного аналізу для визначення протоколу вхідного з'єднання	40
Висновки до 3 розділу.....	42
РОЗДІЛ 4. РОЗРОБКА ВИСОКОРІВНЕВОГО КОДУ ОБРОБКИ МЕРЕЖЕВОГО ТРАФІКА ОБРОБКИ ПРОТОКОЛІВ	43
4.1 Впровадження та розробка алгоритму підключення до цільового сервера...43	
4.2 Реалізація оптимізованої черги пакетів.....	44
4.3 Розробка обробника протоколу SOCKS5	46
4.4 Розробка обробника протоколу HTTP(S).....	51
4.5 Порівняння продуктивності з готовим рішенням.	56
Висновки до 4 розділу.....	61
ВИСНОВКИ	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SOCKS – Socket Secure

SOCKS5 – The latest version of the SOCKS protocol

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

GO – A high-level programming language developed by Google

LINUX – Open-source operating system

IP – Internet Protocol

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

ВСТУП

Актуальність теми: Проблема забезпечення високої продуктивності проксі-серверів є однією з найважливіших у сучасній мережевій інфраструктурі. Проксі-сервери широко використовуються для різних цілей, включаючи доступ до заблокованих ресурсів, анонімне переглядання веб-сторінок та захист від шкідливого програмного забезпечення. Зростаючий попит на ці послуги призвів до збільшення навантаження на проксі-сервери.

Актуальність теми дипломної роботи обумовлена кількома факторами. По-перше, це зростаючий попит на послуги проксі. Проксі-сервери широко використовуються для різних цілей, включаючи доступ до заблокованих ресурсів, анонімне переглядання веб-сторінок та захист від шкідливого програмного забезпечення. Зростаючий попит на ці послуги призвів до збільшення навантаження на проксі-сервери.

Недостатня продуктивність існуючих проксі-серверів. Багато існуючих проксі-серверів не здатні забезпечити необхідну продуктивність для задоволення зростаючого попиту. Це може призвести до затримок у доступі до ресурсів, а також до інших проблем.

Необхідність підтримки багатопортового безпарольного доступу. Багато сучасних застосунків використовують багатопортовий доступ до мережі. Це може призвести до додаткового навантаження на проксі-сервери.

Для вирішення цієї проблеми розроблено ряд підходів, таких як використання паралельних та асинхронних алгоритмів, кешування та оптимізації трафіку. Однак ці підходи мають свої обмеження, і в деяких випадках вони не можуть забезпечити необхідну продуктивність.

Особливо актуальною є проблема продуктивності проксі-серверів для випадку багатопортового безпарольного доступу. Багато сучасних застосунків використовують багатопортовий доступ до мережі. Це може призвести до додаткового навантаження на проксі-сервери, оскільки кожний порт вимагає окремої обробки запитів.

Мета даної дипломної роботи є розробка нових підходів до вирішення проблем продуктивності проксі-серверів для протоколів HTTP та SOCKS5 для випадку багатопортового безпарольного доступу.

З метою досягнення поставленої задачі були сформульовані наступні **завдання**:

1. Аналіз існуючих проблем продуктивності проксі-серверів:
 - дослідження поточного стану проксі-серверів для протоколів HTTP та SOCKS5;
 - виявлення ключових проблем продуктивності, пов'язаних з багатопортовим доступом та безпарольними методами аутентифікації.
2. Теоретичне обґрунтування нових підходів:
 - розробка теоретичних основ для нових методів оптимізації продуктивності;
 - аналіз потенційних переваг та недоліків запропонованих підходів.
3. Розробка алгоритмів та методів оптимізації:
 - створення алгоритмів для виділення основного потоку обробки запитів;
 - розробка паралельних та асинхронних алгоритмів для обробки запитів;
 - впровадження динамічного розподілу навантаження між портами.
4. Експериментальна перевірка та тестування:
 - реалізація розроблених алгоритмів у практичному середовищі;
 - проведення тестування для оцінки ефективності нових підходів;
 - аналіз отриманих результатів та порівняння з існуючими рішеннями.

Об'єктом дослідження виступають проксі-сервери для протоколів HTTP та SOCKS5. Ці системи є центральними в дослідженні, оскільки аналізується їх продуктивність, виявляються існуючі проблеми та розробляються нові методи для їх оптимізації.

Предметом дослідження є методи та техніки оптимізації продуктивності проксі-серверів. Це включає в себе детальний аналіз різних алгоритмів, стратегій та технічних рішень, які можуть бути використані для підвищення ефективності проксі-серверів, особливо в контексті багатопортового безпарольного доступу.

Окрім того, взаємодія між проксі-серверами та мережевими протоколами: Аналіз того, як проксі-сервери взаємодіють з протоколами HTTP та SOCKS5, та вивчення впливу цих взаємодій на загальну продуктивність системи.

Наукова новизна дипломної роботи полягає в розроблених нових підходах до вирішення проблем продуктивності проксі-серверів для випадку багатопортового безпарольного доступу. Запропоновані підходи базуються на розроблених рішеннях:

1. Виділення основного потоку обробки запитів. Основний потік обробки запитів відповідає за основні операції з запитами, такі як аналіз запиту, встановлення з'єднання з сервером та передача даних. Інші потоки обробки запитів відповідають за додаткові операції, такі як аутентифікація користувача, кешування та оптимізація трафіку.

2. Використання паралельних та асинхронних алгоритмів для обробки запитів. Паралельні алгоритми дозволяють обробляти кілька запитів одночасно, а асинхронні алгоритми дозволяють виконувати обробку запитів без блокування.

3. Використовування динамічного розподілу навантаження між портами. Динамічний розподіл навантаження дозволяє розподіляти навантаження між портами проксі-сервера таким чином, щоб забезпечити максимальну продуктивність.

4. Використання eBPF для розширення можливостей ядра Linux. eBPF дозволяє виконувати код на рівні ядра, що дає можливість більш ефективно обробляти мережевий трафік.

5. Використання єдиного порту для прослуховування вхідних підключень. Це дозволяє зменшити навантаження на процесор, оскільки не потрібно створювати окремий сокет для кожного порту.

6. Використання алгоритму зворотного відстеження для визначення оригінального порту. Цей алгоритм дозволяє визначити оригінальний порт на основі інформації, що міститься в заголовку пакета.

Практична цінність полягає в наступних перевагах, які дозволяють істотно підвищити продуктивність проксі-серверів для випадку багатопортового безпарольного доступу. Також масштабованими, тобто їх можна використовувати для проксі-серверів з великою кількістю портів. Та саме головне – гнучкими, тобто їх можна адаптувати до різних умов роботи проксі-серверів.

Запропоновані підходи були реалізовані в програмному продукті, який був протестований на реальних навантаженнях. Результати експериментів показали, що запропоновані підходи дозволяють істотно підвищити продуктивність проксі-серверів для випадку багатопортового безпарольного доступу. Таким чином, запропоновані підходи є новими і мають практичну спрямованість, є завершеними та готові до використання.

Методологія роботи. Під час дослідження було використано загальнонаукові та емпіричні наукові методи, що ґрунтуються на системному підході, такі як абстрагування, аналіз, синтез, порівняння, узагальнення, систематизація, методи об'єктно-орієнтованого проектування та моделювання

Для розробки системи було вирішено використати мову програмування Golang, систему Linux, eBPF в ядрі Linux.

Особистий внесок здобувача. Усі результати роботи отримані автором самостійно.

Апробація результатів роботи. Матеріали кваліфікаційної роботи доповідалися на II Всеукраїнській науково-технічній конференції «Комп'ютерні технології: інновації, проблеми, рішення» 23-24 листопада 2023 року.

РОЗДІЛ 1. ОГЛЯД ПРОБЛЕМ ПРОДУКТИВНОСТІ ПРОКСІ-СЕРВЕРІВ

1.1 Основні аспекти проксі-серверів для протоколів HTTP та SOCKS5

Проксі-сервери відіграють важливу роль у сучасному інтернет-просторі, виступаючи як шлюз між користувачами та інтернетом. Ці системи діють як посередники, перенаправляючи запити від користувачів до відповідних серверів, а потім повертаючи отримані відповіді. Це не тільки спрощує доступ до інтернет-ресурсів, але й забезпечує додатковий рівень безпеки, оскільки проксі-сервер може фільтрувати вхідний та вихідний трафік, а також блокувати доступ до шкідливих сайтів [1].

Ці сервери також відіграють ключову роль у забезпеченні анонімності та конфіденційності в інтернеті. Вони дозволяють користувачам приховувати свої реальні IP-адреси, що є важливим для захисту особистої інформації та обходу географічних обмежень на доступ до контенту. Це особливо корисно для обходу гео-обмежень або доступу до контенту, який може бути заблокований у регіоні користувача [2].

Використання різних IP-адрес від імені користувачів допомагає приховувати їхні справжні адреси від веб-серверів. Це не тільки забезпечує анонімність, але й допомагає уникнути потенційних кібератак, оскільки справжня IP-адреса користувача залишається невідомою. Проксі-сервери можуть виконувати різні функції, включаючи перенаправлення запитів від клієнтських програм до інших серверів [3].

Проксі-сервери використовуються для спрощення та оптимізації доступу до інтернету в межах локальних мереж, таких як офісні будівлі або навчальні заклади. Вони дозволяють централізовано управляти інтернет-трафіком, забезпечуючи ефективність та безпеку [4].

Дія цих серверів як посередників між клієнтами, які шукають ресурси, та серверами, які ці ресурси надають, дозволяє ефективно розподіляти навантаження на мережеві ресурси та оптимізувати доступ до веб-контенту [5].

Вони виконують дії від імені інших серверів, що є ключовим для підвищення продуктивності та безпеки в інтернеті. Вони можуть використовуватися для кешування даних, зменшення навантаження на сервери та підвищення загальної ефективності мережевих операцій [6].

Схема послідовності, яка ілюструє передачу даних між користувачем та кінцевим ресурсом, як з використанням проксі-сервера, так і без нього представлена на рисунку 1.1.

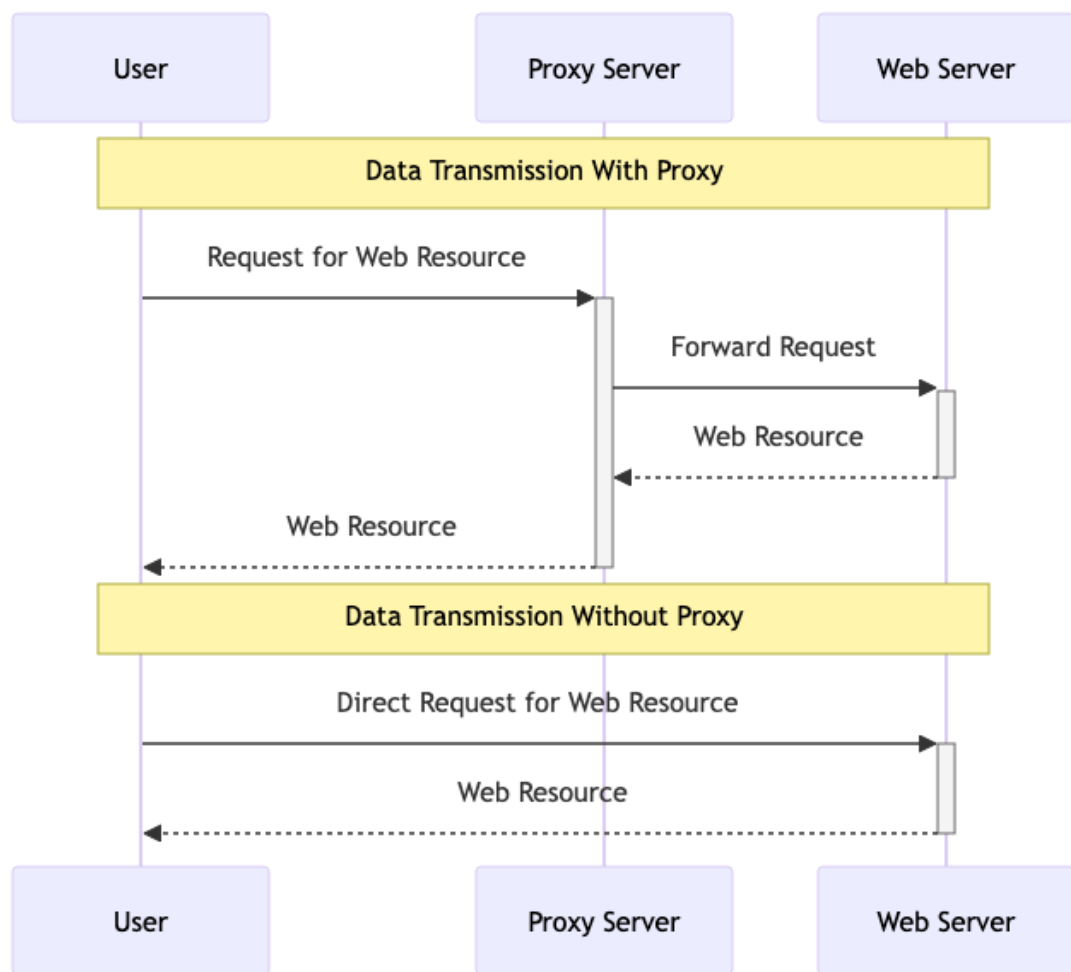


Рис. 1.1. Схема послідовності, яка ілюструє передачу даних між користувачем та кінцевим ресурсом

Завдяки цим функціям, проксі-сервери стають незамінними у сучасному цифровому світі, де безпека та конфіденційність даних є важливими аспектами. Вони допомагають користувачам залишатися захищеними та анонімними під час взаємодії з інтернетом [7].

Проху в перекладі з англійської означає посередник. У контексті Інтернету, Проху-сервер - це програма, яка діє як посередник між клієнтом і сервером. Він отримує запити від клієнта і перенаправляє їх на сервер. Сервер відповідає на запит, і Проху повертає відповідь клієнту [8].

На схемі, що представлена на рис.1.1 можна побачити, як запити користувача перенаправляються через проксі-сервер до веб-сервера, а потім відповідь від веб-сервера повертається через проксі-сервер назад до користувача. У випадку безпосередньої передачі даних, запит користувача надсилається прямо на веб-сервер, а відповідь від веб-сервера надходить безпосередньо користувачеві.

Детально оглядаючи, як проксі-сервери допомагають нам можна з огляду декілька існуючих питань які вони вирішують. До прикладу підвищення ефективності IPTV де проксі-сервери використовуються для зниження затримки при перегляді IPTV, зокрема при зміні каналів та відтворенні відео за запитом (VoD). Використання P2P та проксі-серверів значно зменшити час відгуку та покращує якість послуг [9]. Розробка "живих" прототипів у смарт-містах. Проксі-технології використовуються для створення інноваційних прототипів у смарт-містах, що дозволяє збирати детальні дані та взаємодіяти з користувачами в реальному часі, сприяючи розробці та процесу дизайну [10].

Кешування веб-сервісів: проксі-сервери використовуються для кешування даних веб-сервісів, забезпечуючи високу продуктивність та ефективність. Це особливо важливо для SOAP-базованих веб-сервісів, де правильно налаштований кеш може значно покращити продуктивність [11]. Цільове субсидування медичних послуг: Проксі-сервери використовуються для визначення цільових груп для субсидованих медичних послуг у країнах з обмеженими державними можливостями, таких як М'янма. Це допомагає забезпечити, що субсидії надаються тим, хто їх найбільше потребує [12].

1.2 Різновиди та класифікація проксі-серверів

Проксі-сервери можуть виконувати різні функції, тому їх можна розділити на кілька типів за різними критеріями. Один із найважливіших критеріїв - це протокол, який використовується проксі-сервером. Проведемо аналіз протоколів, рис.1.2:

- FTP проху використовуються для завантаження даних на сервері FTP;
- CGI проху (анонімайзери) допомагають відкрити будь-який веб-сайт у браузері, маскуючи IP-адресу користувача;
- SMTP, POP3 та IMAP проху використовуються для надсилання та отримання електронної пошти;
- HTTP та HTTPS проху призначені для перегляду веб-сторінок. HTTP проху використовуються для звичайного перегляду веб-сторінок, а HTTPS проху - для перегляду веб-сторінок з шифруванням;
- SOCKS проху є найанонімнішим типом проксі-серверів, оскільки вони перенаправляють всі дані на кінцевий сервер як клієнт.

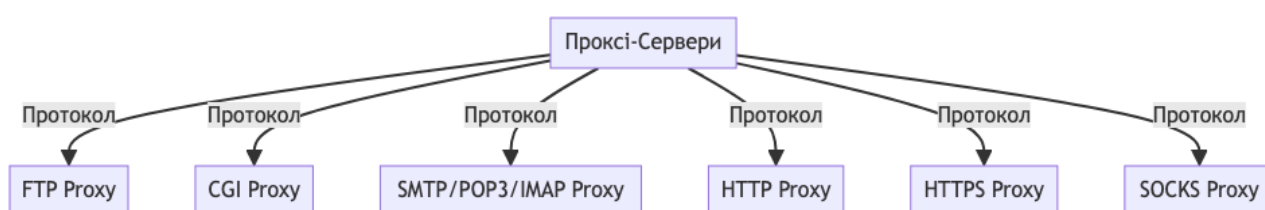


Рис. 1.2. Діаграма, яка ілюструє різні типи проксі-серверів за протоколом

HTTP, HTTPS та SOCKS проху використовуються найчастіше. Кожен тип проксі-сервера відіграє унікальну роль у мережевій комунікації та має свої специфічні функції.

За рівнем анонімності проксі-серверів поділяються на, рис.1.3:

- Прозорі проху-сервери не змінюють запитувані або отримані дані, тому анонімність не передбачається, і веб-сайти можуть прочитати вашу IP-

адресу. Вони використовуються для кешування вмісту або для забезпечення безпеки локальних мереж.

- Анонімні проху-сервери змінюють вашу IP-адресу на адресу проху-сервера, щоб приховати вашу справжню IP-адресу. Це може бути корисно для захисту конфіденційності або для доступу до веб-сайтів, які заблоковані в певних регіонах.

- Проху-сервери з високою анонімністю використовують додаткові заходи для захисту вашої конфіденційності, такі як шифрування даних або періодична зміна IP-адреси. Вони є найбільш безпечним варіантом для анонімного перегляду веб-сайтів.

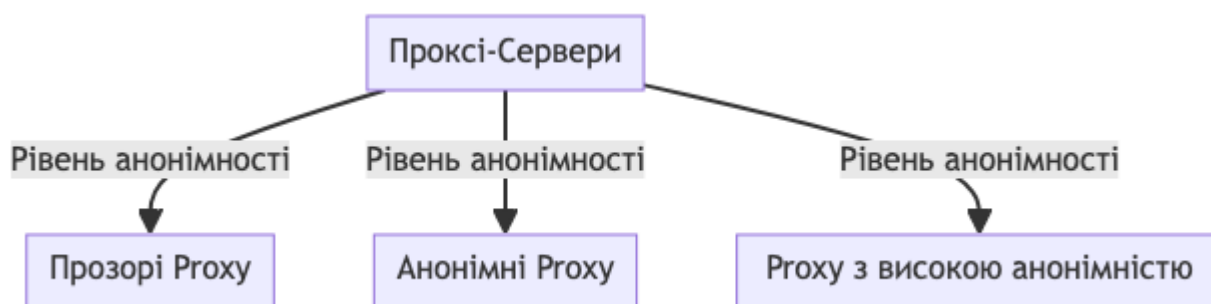


Рис. 1.3. Діаграма, яка ілюструє різні рівні анонімності проксі-серверів

За походження IP, діаграма представлена на рис.1.4:

- Проху центру обробки даних надаються компаніями, які спеціалізуються на наданні послуг проксі-серверів. Вони мають високу пропускну здатність і можуть обробляти велику кількість запитів одночасно.

- Житлові проху-сервери надаються фізичними особами, які використовують свій особистий комп'ютер як проксі-сервер. Вони зазвичай мають нижчу пропускну здатність, ніж проху центру обробки даних, але вони можуть бути більш безпечними, оскільки ви знаєте, кому ви довіряєте свій трафік.

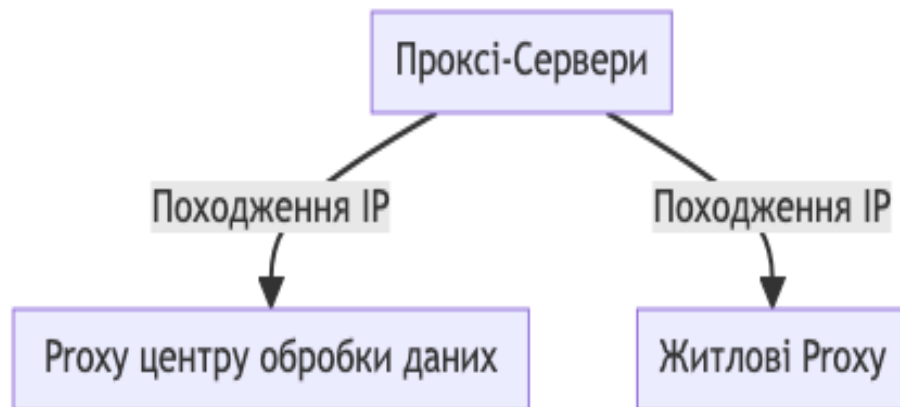


Рис. 1.4. Діаграма, яка ілюструє різні типи проксі-серверів за походженням IP-адрес

За способом використання, рис.1.5:

- Публічні проху-сервери доступні безкоштовно і не вимагають реєстрації. Вони є хорошим варіантом для одноразового використання, але вони можуть бути повільними і небезпечними.
- Приватні проху-сервери надаються компаніями або фізичними особами. Вони зазвичай коштують гроші, але вони пропонують кращу продуктивність і безпеку, ніж публічні проху-сервери.



Рис. 1.5. Графічна діаграма, яка ілюструє різні типи проксі-серверів за способом використання

На основі тривалості, рис. 1.6:

- Необоротні або статичні IP-проху-сервери завжди мають одну й ту ж IP-адресу. Це може бути зручно для використання з програмами, які вимагають постійного доступу до певного сервера.
- Оборотний або змінний IP-проху-сервери періодично змінюють свою IP-адресу. Це може бути корисно для захисту конфіденційності або для обходу блокувань.

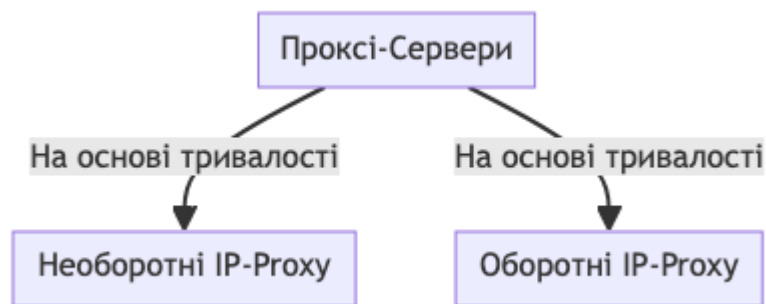


Рис. 1.6. Графічна діаграма, яка ілюструє різні типи проксі-серверів на основі тривалості використання IP-адрес

На основі модифікації даних, рис.1.7:



Рис. 1.7. Графічна діаграма, яка ілюструє різні типи проксі-серверів на основі модифікації даних

- Прямі проху-сервери просто перенаправляють запити від клієнтів на сервери призначення. Вони не вносять жодних змін у дані.

– Зворотні проху-сервери можуть виконувати різні функції, такі як кешування вмісту, фільтрація запитів або перенаправлення трафіку на різні сервери.

1.3 Аналіз переваг і недоліки використання проху-серверів

Переваги Проксі-Серверів. Захист мережі та користувачів: Проксі-сервери захищають як мережу, так і її користувачів, пропонуючи безпечне та швидке підключення до Інтернету. Вони особливо корисні у бізнесі та академічному середовищі [13].

Підвищення ефективності VoD систем: Проксі-сервери використовуються для підвищення ефективності систем відео на вимогу (VoD), знижуючи затримку та покращуючи якість послуг [14].

Ефективний вибір проксі-сервера у кооперативному кешуванні: Проксі-сервери можуть використовуватися для ефективного вибору сервера, що пропонує найкращий час відгуку клієнту, знижуючи загальне споживання мережевого трафіку та покращуючи середній час відгуку [15].

Покращення архітектури кешування: Інтеграція браузерних та проксі-кешів може знизити інтернет-трафік, інтегруючи браузерні кеші з проксі-серверами для покращення управління кешем [16].

Забезпечення доступу до цифрових ресурсів: Проксі-сервери використовуються бібліотеками для надання доступу до пропрієтарних баз даних для користувачів поза кампусом, а також для обмеження доступу до певних баз даних або класів користувачів всередині бібліотеки [17].

До недоліків проксі-серверів слід віднести:

1. Обмеження в пропускній здатності та швидкості: проксі-сервери можуть мати обмеження у пропускній здатності та швидкості, особливо у випадку великої кількості одночасних запитів [18].

2. Потенційні проблеми з безпекою: проксі-сервери можуть створювати потенційні ризики, особливо якщо вони не належним чином налаштовані або застарілі [19].

3. Складність у масштабуванні: проксі-сервери можуть мати складнощі з масштабуванням, особливо у випадку великої кількості користувачів або високого обсягу даних [20]. Проксі-сервери можуть призвести до зниження продуктивності, оскільки вони додають додатковий крок у процесі обробки запитів.

1.4 Огляд основних протоколів проксі-серверів

HTTP-проксі - це найпоширеніший тип проксі. Його основне призначення - організація роботи браузерів, а також інших програм, які використовують HTTP. Принцип роботи: програма або браузер посилає запит проксі-серверу на відкриття певного URL-ресурсу, проксі-сервер отримує дані з запитуваного ресурсу та передає ці дані вашому браузеру.

За допомогою HTTP-проксі з'являється можливість контролювати такі запити:

- Кешування даних (картинок, сторінок тощо). Забезпечує приріст швидкості при статичному контенті та низькій пропускну здатності зовнішнього каналу зв'язку. Обмеження доступу до деяких ресурсів: створення "чорного списку" заборонених сайтів або "білого списку" дозволених.
- Підміна запитуваного ресурсу, відмінного від того, який запитується користувачем: наприклад, замість банерів із рекламою видавати прозорі картинки, які будуть суттєво економити час завантаження та трафік.
- Рівномірний розподіл пропускну смуги між користувачами локальної мережі: наприклад, можна встановити ліміт швидкості завантаження файлів для уникнення завантаженості сервера.
- Ведення логів: можна підраховувати трафік за кожним користувачем, дивитися список популярних сайтів.

– Маршрутизація запитів: частину запитів направляти безпосередньо, а іншу частину перенаправляти через інші проксі.

Однак відсутність підтримки шифрування робить неможливою роботу з сайтами, які використовують HTTPS. Також, HTTP-проксі може передавати ваш реальний IP-адресу в заголовках запиту, що повністю виключає цей тип проксі з розряду анонімних [21].

HTTPS-проксі фактично є HTTP-проксі, який використовує шифрування (S – Secure). Весь трафік, що проходить через проксі-сервер, шифрується стійким до зламу алгоритмом. При такому підході відсутня можливість дізнатися, яка саме інформація (картинка, відео, документ) передається через проксі-сервер, оскільки проксі-сервер не бере участь у процесі шифрування та дешифрування інформації. Це дозволяє використовувати HTTPS-проксі для передачі практично будь-якого TCP-протоколу: POP3, SMTP, IMAP, NNTP тощо. HTTPS все так само може передавати ваш реальний IP-адресу в заголовках запиту, що повністю виключає цей тип проксі з розряду анонімних [22-23].

Socks-проксі – найпрогресивніший протокол передачі інформації. Протокол SOCKS розроблявся для того, щоб програми, які не підтримують використання проксі, могли користуватися ресурсами мережі. Протокол являє собою транслятор, але порівняно з іншими проксі Socks-клієнт знаходиться між прикладним та транспортним рівнем у мережі. Socks-сервер розташовується на прикладному рівні, що виключає прив'язку до протоколів високого рівня.

Socks-сервер не передає ваш IP у заголовку запиту, будучи анонімним. Socks-проксі підтримують практично всі протоколи, включаючи HTTP, HTTPS, FTP, SSH, Telnet тощо. Це робить їх універсальним інструментом для роботи з мережею.

Він маскує вашу реальну IP-адресу, приховує фізичне розташування та не дозволяє стороннім відстежувати ваші дії в Інтернеті. Хоча проксі SOCKS5 не такий надійний та універсальний в плані захисту, як VPN, він пропонує декілька серйозних переваг.

Ось деякі важливі аспекти SOCKS5 проксі:

- Анонімність: Ваш IP-адреса приховується, і інтернет-трафік проходить через проксі-сервер, що робить вас менш вразливим до відстеження.
 - Обхід блокувань: SOCKS5 дозволяє обходити географічні обмеження та блокування контенту.
 - Швидкість: Він пропонує високу швидкість передачі даних [24-27].
- Вибір проксі-сервера залежить від потреб і вимог. Якщо потрібен проксі для роботи з браузером, то HTTP-проксі або HTTPS-проксі будуть хорошим вибором. Якщо потрібен проксі для роботи з програмами, які не підтримують використання проксі, то Socks-проксі буде найкращим варіантом. Якщо вам потрібна анонімність, то Socks-проксі є єдиним варіант [28-30].

Висновки до першого розділу

Проведено огляд ролі та значення проксі-серверів у сучасному інтернет-просторі. Визначено, що проксі-сервери виступають як ключові елементи у забезпеченні безпеки, анонімності та ефективності доступу до інтернет-ресурсів. Виділено, що вони діють як посередники між користувачами та цільовими серверами, виконуючи важливі функції, такі як фільтрація трафіку, кешування даних, обхід географічних обмежень та захист від кібератак.

Встановлено, що різноманітність типів проксі-серверів, включаючи HTTP, HTTPS та SOCKS, підкреслює їх універсальність та специфічність застосування в залежності від потреб користувачів. Встановлено, що кожен тип проксі-сервера має свої унікальні характеристики та призначення, від анонімності та безпеки до специфічних функцій, як-от кешування та маршрутизація трафіку.

Виявлено недоліки, такі, як потенційні обмеження у пропускній здатності, проблеми з безпекою та складнощі у масштабуванні. Ці виклики вимагають уважного підходу до вибору та налаштування проксі-серверів, щоб забезпечити оптимальний баланс між продуктивністю, безпекою та анонімністю.

В результаті проведеного аналізу встановлено, що у цілому, проксі-сервери є невід'ємною частиною інтернет-інфраструктури, що сприяє

підвищенню ефективності, безпеки та доступності інтернет-ресурсів. Їх роль у сучасному цифровому світі продовжує залишатися актуальною, а розуміння їх функціональності та потенційних обмежень є ключовим для ефективного використання цих технологій.

РОЗДІЛ 2. ПЕРЕВІРКА ТА ДОСЛІДЖЕННЯ ГОТОВИХ РІШЕНЬ

2.1. Огляд популярних рішень для проксі-серверів

Існує безліч софтверних рішень, за допомогою яких можна підняти проксі-сервер. Ось деякі з найпопулярніших:

- Squid – це безкоштовний і відкритий проксі-сервер, який підтримує широкий спектр протоколів і функцій.
- Apache – це веб-сервер, який також може використовуватися як проксі-сервер.
- Nginx – це ще один веб-сервер, який може використовуватися як проксі-сервер.
- HAProxy – це високодоступний проксі-сервер, який можна використовувати для балансування навантаження між різними серверами.
- Varnish – це швидкий і ефективний проксі-сервер, який можна використовувати для кешування вмісту.
- Shadowsocks – це популярний протокол проксі-сервера, який підтримує різні методи шифрування та анонімності.
- V2Ray – це новий протокол проксі-сервера, який є більш швидким і ефективним, ніж Shadowsocks.
- Trojan – це ще один протокол проксі-сервера, який є популярним у Китаї.

Також можна перелічити: GoAgent, Clash, X-Tunnel, Psiphon, Windscribe.

Окрім того можна виділити деякі інші софтверні рішення, які можна використовувати для підняття проксі-сервера:

- ProxyChains – це інструмент командного рядка, який дозволяє використовувати проксі-сервери для різних програм.
- Proxifier – це інструмент для Windows, який дозволяє використовувати проксі-сервери для різних програм.

- FoxyProxy – це розширення для браузера Firefox, яке дозволяє використовувати проксі-сервери для різних веб-сайтів.

- SwitchyOmega – це розширення для браузера Chrome, яке дозволяє використовувати проксі-сервери для різних веб-сайтів.

Вибір конкретного софтверного рішення залежить від потреб і вимог. Якщо потрібен простий і надійний проксі-сервер, то Squid або Apache – це хороший вибір. Якщо потрібен високодоступний проксі-сервер, то HAProxy – це хороший вибір. Якщо потрібен швидкий і ефективний проксі-сервер для кешування вмісту, то Varnish – це хороший вибір.

2.2. Огляд та аналіз Zproxy

Zproxy це популярне рішення для створення та налаштування проксі-серверів. Він відомий своєю легкістю налаштування, підтримкою різних протоколів та функцій, а також високою продуктивністю та стабільністю.

Функції:

- Підтримує численні протоколи, включаючи HTTP, HTTPS, FTP, SOCKS, SMTP, POP3, IMAP, TELNET та інші.

- Забезпечує різні типи анонімності, від прозорих до високоефективних.

- Дозволяє здійснювати фільтрацію трафіку на основі IP-адрес, користувачів, доменних імен, протоколів та заголовків.

- Кешує вміст для покращення продуктивності. Підтримує балансування навантаження для розподілу трафіку між кількома серверами.

- Має вбудовану систему захисту від атак.

Переваги:

- Легко налаштовується та використовується.

- Відкритий код, доступний для безкоштовного використання та модифікації.

- Підтримує широкий спектр протоколів і функцій.

- Висока продуктивність та стабільність.
- Багато онлайн-ресурсів для навчання та підтримки.

Недоліки:

- Може бути складним для розширеного налаштування для досвідчених користувачів.
- Деякі функції потребують додаткових модулів або конфігурації.
- Не гарантує повної анонімності, особливо за безкоштовних налаштувань.

Оцінка: 3проху є потужним та гнучким рішенням для підняття проксі-серверів. Він підходить як для початківців, так і для досвідчених користувачів, які потребують широкого спектру можливостей та функціоналу. Однак, для забезпечення високої анонімності або складних конфігурацій, може знадобитися додатковий досвід та налаштування.

Встановлення 3проху на Ubuntu. Щоб встановити 3проху на Ubuntu, слід виконати такі дії:

1. Оновити список пакетів:

```
sudo apt update
```

2. Встановіть необхідні залежності:

```
sudo apt install build-essential libssl-dev libpcre3-dev
```

3. Завантажте вихідний код 3проху:

```
wget  
https://github.com/z3APA3A/3proxy/releases/download/0.9.3/3proxy-  
0.9.3.tar.gz
```

4. Розпакуйте вихідний код:

```
tar -zxvf 3proxy-0.9.3.tar.gz
```

5. Перейдіть до каталогу з вихідним кодом:

```
cd 3proxy-0.9.3
```

6. Скомпілювати та встановити 3проху:

```
make  
sudo make install
```

Налаштування 3проху. Після встановлення 3проху необхідно налаштувати його. Для цього відкрийте файл конфігурації:

```
sudo nano /etc/3proxy/3proxy.cfg
```

У цьому файлі можна змінити такі параметри:

- `listen`: Порт, на якому буде прослуховуватися проксі-сервер. За замовчуванням це порт 8080.
- `logdir`: Каталог для зберігання журналів. За замовчуванням це каталог `/var/log/3proxy`.
- `cachedir`: Каталог для зберігання кешу. За замовчуванням це каталог `/var/cache/3proxy`.
- `allow`: Список дозволених IP-адрес.
- `deny`: Список заборонених IP-адрес.

Ось приклад файлу конфігурації:

```
listen = 8080
logdir = /var/log/3proxy
cachedir = /var/cache/3proxy
allow = 127.0.0.1
deny = 0.0.0.0/0
```

Після зміни параметрів збережіть файл і перезапустіть проксі-сервер:

```
sudo service 3proxy restart
```

Перевірка роботи 3proxy

Щоб перевірити роботу 3проху, відкриваємо веб-браузер і вводимо у адресному рядку IP-адресу вашого сервера та порт 3проху, наприклад:

```
http://192.168.1.1:8080
```

Отримуємо повідомлення "**3proxy proxy server**", то проксі працює правильно.

Слід відзначити про такі додаткові можливості 3проху

3проху надає безліч додаткових можливостей, які можна налаштувати у файлі конфігурації. Ось деякі з них:

- Фільтрація трафіка: 3проху може використовуватися для фільтрації трафіка, наприклад, для блокування певних веб-сайтів або додатків.
- Кешування даних: 3проху може кешувати дані, щоб прискорити доступ до них.
- Балансування навантаження: 3проху може використовуватися для балансування навантаження між декількома серверами.

– Контроль доступу: Зпроху може використовуватися для контролю доступу до певних ресурсів [31].

2.3 Розширений огляд та конфігурація проксі-сервера Squid

Squid - це безкоштовний і відкритий проксі-сервер, який використовується для підвищення продуктивності, безпеки та контролю мережі. Він був розроблений в Національному центрі суперкомп'ютингу в США і вперше випущений у 1996 році.

Squid підтримує широкий спектр протоколів, включаючи HTTP, HTTPS, FTP, SMTP, IMAP та POP3. Він також може використовуватися для балансування навантаження між декількома серверами. Однією з основних функцій Squid є кешування даних. Кешування - це процес зберігання копій часто запитуваних даних у пам'яті проксі-сервера. Це дозволяє зменшити затримки та підвищити швидкість завантаження веб-сайтів та інших ресурсів. Squid також може використовуватися для фільтрації трафіку. Фільтрація трафіку – це процес блокування або дозволу певних типів трафіку. Це може бути корисно для захисту мережі від шкідливого програмного забезпечення та інших загроз.

Squid також може використовуватися для контролю доступу до мережі. Контроль доступу – це процес дозволяння або заборони доступу до певних веб-сайтів або ресурсів. Це може бути корисно для компаній, які хочуть захистити своїх співробітників від шкідливого контенту. Squid є популярним вибором для веб-серверів, корпоративних мереж та інших організацій, які потребують підвищення продуктивності, безпеки та контролю мережі

Ось деякі з основних переваг використання Squid:

- Покращення безпеки
- Контроль доступу
- Балансування навантаження
- Кешування даних
- Фільтрація трафіку

Установка Squid Proxy Server на Ubuntu включає декілька кроків. Ось загальна інструкція. Перш ніж встановлювати нові пакети, завжди корисно оновити список пакетів і самі пакети:

```
sudo apt update  
sudo apt upgrade
```

Крок 2: Встановлення Squid. Встановлюємо Squid за допомогою команди apt:

```
sudo apt install squid
```

Крок 3: Конфігурація Squid. Редагування основного конфігураційного файлу. Файл розташований за адресою `/etc/squid/squid.conf`:

```
sudo nano /etc/squid/squid.conf
```

Налаштування доступу. Додаємо правила доступу (ACL) та налаштування проксі. Наприклад, для дозволу доступу всім внутрішнім IP-адресам у мережі:

```
acl localnet src 192.168.1.0/24 # замініть на вашу мережу  
http_access allow localnet  
http_access deny all
```

Крок 4: Перезавантаження Squid. Після зміни конфігурації перезавантажуємо Squid, щоб застосувати зміни:

```
sudo systemctl restart squid
```

Крок 5: Перевірка статусу Squid

```
sudo systemctl status squid
```

Крок 6: Налаштування Firewall

Якщо включений UFW (Uncomplicated Firewall), додаємо правило для Squid:

```
sudo ufw allow 3128/tcp
```

(3128 – це стандартний порт Squid, змініть його, якщо ви використовували інший порт у конфігурації).

Тепер Squid повинен бути налаштований і готовий до використання на сервері Ubuntu. Ви можете налаштувати клієнтські системи для використання цього проксі-сервера з вказаним IP-адресою сервера та портом (за замовчуванням 3128).

Squid є потужним інструментом, який може використовуватися для підвищення продуктивності, безпеки та контролю мережі. Він є безкоштовним і відкритим, що робить його доступним для широкого кола користувачів [32].

2.4 Використання Apache, налаштування та застосування на Ubuntu

Apache reverse proxy) - це тип проксі-сервера, який розміщується за веб-сервером. Він приймає запити від клієнтів і перенаправляє їх на веб-сервер. Це дозволяє клієнтам отримувати доступ до ресурсів веб-сервера, не знаючи про його IP-адресу або порт.

Apache прокс ітоді можна використовувати для наступних цілей:

Балансування навантаження: Apache проксі можна використовувати для балансування навантаження між декількома веб-серверами. Це може допомогти підвищити продуктивність веб-сайту.

Кешування: Apache проксі можна використовувати для кешування часто запитуваних ресурсів. Це може допомогти зменшити затримки і підвищити швидкість завантаження веб-сайту.

Фільтрація трафіка: Apache проксі можна використовувати для фільтрації трафіка, наприклад, для блокування певних веб-сайтів або контенту.

Контроль доступу: Apache прокс ітоді можна використовувати для контролю доступу до певних ресурсів, наприклад, для обмеження доступу до певних веб-сайтів або контенту для певних користувачів.

Щоб встановити Apache проксі на Ubuntu, виконайте наступні кроки:

1. Оновити список пакетів:

```
sudo apt update
```

2. Встановити модуль проксі:

```
sudo apt install apache2-mod-proxy-http
```

3. Активуйте модуль прокс ітоді:

```
sudo a2enmod proxy_http
```

4. Перезапустіть веб-сервер Apache:

```
sudo service apache2 restart
```

Щоб налаштувати Apache проксі, відкрити файл конфігурації веб-сервера Apache:

```
sudo nano /etc/apache2/sites-available/default
```

Додаємо наступний код до файлу конфігурації:

```
ProxyPass / http://[IP-адреса веб-сервера]:[порт веб-сервера]
```

Замість [IP-адреса веб-сервера] і [порт веб-сервера] на фактичні IP-адресу і порт веб-сервера.

Зберігаємо файл конфігурації і перезапустіть веб-сервер Apache:

```
sudo service apache2 restart
```

Після цього зможемо отримати доступ до ресурсів веб-сервера, використовуючи URL-адресу Apache прокс ітоді. Наприклад, якщо IP-адреса Apache проксі - 192.168.1.1, то клієнти можуть отримати доступ до веб-сайту, розташованого на веб-сервері з IP-адресою 192.168.1.2, використовуючи URL-адресу `http://192.168.1.1/`.

Висновки до другого розділу

Проведено аналіз в результаті якого встановлено, що існує безліч софтверних рішень, за допомогою яких можна підняти проксі-сервер. Встановлено вибір конкретного рішення залежить від потреб і вимог.

Встановлено, що для простих і надійних проксі-серверів, які підтримують широкий спектр протоколів і функцій, можна використовувати Squid або Apache. Ці рішення є безкоштовними і відкритими, а також мають велику спільноту користувачів і розробників. Окрім цих рішень, існує безліч інших варіантів, які можна використовувати для підняття проксі-сервера. Для конкретних потреб можна використовувати спеціалізовані рішення, такі як Shadowsocks, V2Ray, Trojan, GoAgent, Clash, X-Tunnel, Psiphon, Windscribe. Також можна використовувати інструменти командного рядка, такі як ProxyChains, або інструменти для Windows, такі як Proxifier. А для використання проксі-сервера в браузері можна використовувати розширення, такі як FoxyProxy або SwitchyOmega.

Враховуючи ці фактори, можна вибрати оптимальне рішення, яке буде відповідати конкретним потребам і вимогам.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ І ЗАСТОСУВАННЯ РОЗРОБЛЕНИХ НИЗЬКОРІВНЕВИХ МЕХАНІЗМІВ

3.1 Загальний опис логіки та налаштування сервісу.

Проксі-реактор працює на основі спільної роботи низькорівневих механізмів (iptables, ebpf) та високорівневого коду обробки мережевого трафіку (golang, gnet). Коротко процес обробки вхідного з'єднання виглядає так:

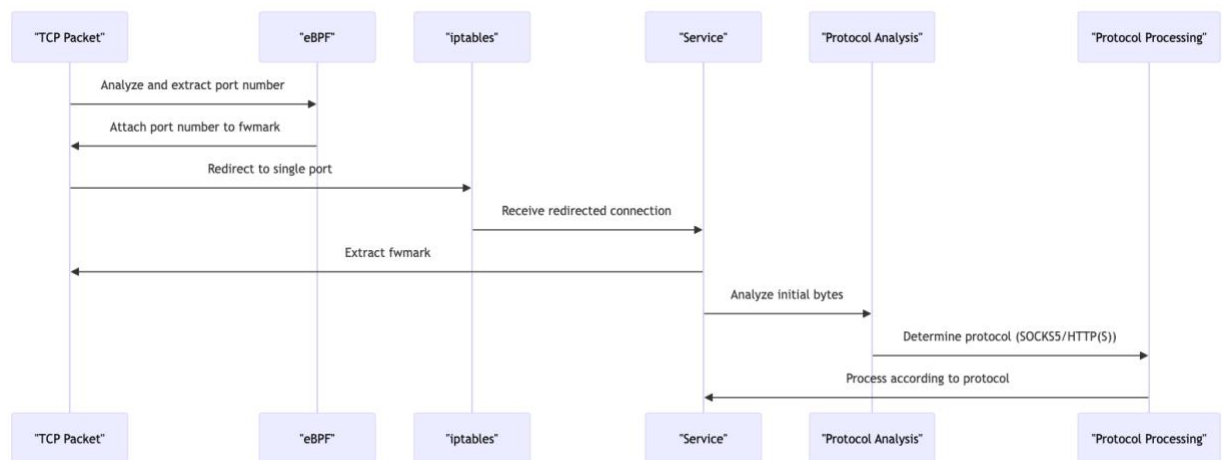


Рис.3.1. Діаграма послідовності, що ілюструє процес обробки вхідного з'єднання.

Вхідний tcp пакет аналізується і номер вхідного порту записується в мітку пакета (fwmark) за допомогою eBPF. iptables згідно з правилом перенаправляє будь-який порт у зазначеному діапазоні на єдиний порт. Сервіс приймає з'єднання і витягує маркер, прикріплений на першому кроці. Таким чином дізнається оригінальний порт, на який користувач відправляв запит на з'єднання.

У вхідному пакеті аналізуються початкові байти і на основі них визначається протокол, який використовує користувач (SOCKS5 або HTTP(S)). Запускається обробка протоколу, який був визначений на попередньому кроці.

Тобто, проксі-реактор працює, спочатку аналізуючи вхідний tcp пакет і записуючи номер вхідного порту в мітку пакета (fwmark) за допомогою eBPF. Потім iptables перенаправляє будь-який порт у зазначеному діапазоні на єдиний порт. Після цього сервіс приймає з'єднання і витягує маркер, прикріплений на першому кроці. Це дозволяє сервісу визначити оригінальний порт, на який користувач відправляв запит на з'єднання.

Потім аналізуються початкові байти вхідного пакета, щоб визначити протокол, який використовує користувач. Якщо протоколом є SOCKS5 або HTTP(S), то запускається відповідна обробка. Ось більш детальне пояснення кожного кроку:

Крок перший: Вхідний tcp пакет аналізується за допомогою eBPF. eBPF - це технологія, яка дозволяє розробникам писати код, який виконується безпосередньо в ядрі Linux. У цьому випадку код eBPF використовується для вилучення номера вхідного порту з пакета. Номер порту потім записується в мітку пакета (fwmark).

Крок 2: iptables - це брандмауер Linux, який можна використовувати для фільтрації мережевого трафіку. У цьому випадку iptables налаштовується для перенаправлення будь-якого порту у зазначеному діапазоні на єдиний порт. Це дозволяє сервісу приймати з'єднання на будь-якому порту, а потім перенаправляти їх до відповідного призначення.

Крок 3: Сервіс приймає з'єднання і витягує маркер, прикріплений на першому кроці. Маркер містить оригінальний порт, на який користувач відправляв запит на з'єднання.

Крок 4: Аналізуються початкові байти вхідного пакета. Перші кілька байтів пакета SOCKS5 або HTTP(S) містять унікальний ідентифікатор, який можна використовувати для визначення протоколу, який використовує користувач.

Крок 5: Якщо протоколом є SOCKS5 або HTTP(S), то запускається відповідна обробка. Наприклад, якщо протоколом є SOCKS5, то сервіс створить SOCKS5 проксі-з'єднання до сервера призначення.

Обробка SOCKS5. У разі SOCKS5 проксі-сервер виконує наступні дії:

- Передає дані від користувача до сервера призначення.
- Отримує дані від сервера призначення і передає їх користувачеві.
- SOCKS5 проксі-сервер може виконувати додаткові функції, такі як:
- Фільтрація трафіку
- Налаштування безпеки
- Журналування

У разі HTTP(S) проксі-сервер виконує наступні дії:

- Отримує HTTP-запиту від користувача.
- Відправляє HTTP-запит серверу призначення.
- Отримує HTTP-відповідь від сервера призначення.
- Передає HTTP-відповідь користувачеві.

HTTP(S) проксі-сервер може виконувати додаткові функції, такі як:

- Кешування даних.
- Прискорення завантаження веб-сторінок.
- Налаштування безпеки.
- Журналування.

Подальші дослідження будуть направлені на аналіз eBPF.

3.2 Розробка низькорівневого алгоритму аналізу та маркування за допомогою eBPF.

eBPF – це технологія, яка дозволяє розробникам писати код, який виконується безпосередньо в ядрі Linux. Це дозволяє використовувати eBPF для різних завдань, таких як аналіз мережевого трафіку, фільтрація трафіку та моніторинг продуктивності.

Для аналізу мережевого трафіку за допомогою eBPF можна використовувати такі кроки:

- Створити програму eBPF, яка буде аналізувати мережевий трафік.
- Завантажити програму eBPF в ядро Linux.

– Налаштувати ядро Linux для запуску програми eBPF для певного мережевого інтерфейсу.

```
// clang -O2 -emit-llvm -c port_to_mark.c -Wall -Werror -o - | llc
-march=bpf -filetype=obj -o bpf.o
#include <asm/types.h>
#include <asm/byteorder.h>

#include <linux/bpf.h>
#include <linux/filter.h>
#include <linux/in.h>
#include <linux/if_ether.h>
#include <linux/ip.h>
#include <linux/tcp.h>

#include "helpers.h"

__section("classifier") int lb_main(struct __sk_buff *skb)
{
    if (unlikely(skb->protocol != __constant_htons(ETH_P_IP)))
        return -1;
    if (unlikely(skb->mark != 0))
        return -1;

    int nh_off = sizeof(struct ethhdr);

    __u8 ip_proto = load_byte(skb, nh_off +
                                offsetof(struct iphdr,
protocol));
    if (unlikely(ip_proto != IPPROTO_TCP))
        return -1;

    __u8 ip_vl = load_byte(skb, nh_off);
    if (likely(ip_vl == 0x45))
        nh_off += sizeof(struct iphdr);
    else
        nh_off += (ip_vl & 0xF) << 2;

    __u16 dport = load_half(skb, nh_off + offsetof(struct tcphdr,
dest));
    skb->mark = dport;
    re
```

Ця програма витягує інформацію про вхідний пакет і перевіряє, що він є частиною протоколу TCP і записує в маркер fwmark (skb->mark) вхідний номер порту. Підключається до мережевого інтерфейсу наступними командами:

```
tc qdisc add dev <interface> clsact
tc filter add dev <interface> ingress bpf obj <full path>
```

Перенаправлення трафіку за допомогою iptables. Трафік перенаправляється наступною командою:

```
iptables -t nat -A PREROUTING -i <interface> -p tcp --dport
10000:15000 -j REDIRECT --to-port 9999
```

Усі вхідні з'єднання, що знаходяться на портах починаючи з 10000 до 15000, перенаправляються на порт 9999.

Залучення оригінального порту на стороні сервісу (golang). Функція вилучення оригінального порту виглядає наступним чином:

```
func GetMark(c gnet.Conn) (int, error) {
    return syscall.GetsockoptInt(c.Fd(), syscall.SOL_SOCKET,
    syscall.SO_MARK)
}
```

Потрібно окремо відзначити, що за замовчуванням syscall.SO_MARK не працює. Для її роботи потрібно попередньо активувати флаг ядра net.ipv4.tcp_fwmark_accept. Наприклад, наступна команда: sysctl -w net.ipv4.tcp_fwmark_accept=1.

3.3 Реалізація сигнатурного аналізу для визначення протоколу вхідного з'єднання.

У зв'язку з тим, що можливих протоколів всього 2 достатньо просто сформулювати евристичні правила визначення протоколу. HTTP(S): завжди

починається з символів ascii і для визначення зрівнюються перші 3 байти на потрапляння в діапазон дозволених значень:

```
func (h *httpProxy) DetectProtocol(c gnet.Conn) (bool, error) {
    buf, err := c.Peek(httpHeaderLen)
    if err != nil {
        return false, err
    }

    for _, b := range buf {
        isUpper := b >= upperLetterRangeStart && b <=
upperLetterRangeEnd
        isLower := b >= lowerLetterRangeStart && b <=
lowerLetterRangeEnd
        if !isUpper && !isLower {
            return false, nil
        }
    }
    return true, nil
}
```

SOCKS5: це бінарний протокол першої байтової версії, а другий байт не має бути рівним нулю (згідно з RFC 1928):

The client connects to the server, and sends a version identifier/method selection message:

```
+----+-----+-----+
|VER | NMETHODS | METHODS  |
+----+-----+-----+
| 1  |    1    | 1 to 255 |
+----+-----+-----+
```

The VER field is set to X'05' for this version of the protocol. The

NMETHODS field contains the number of method identifier octets that appear in the METHODS field.

Код визначення:

```
func (s *socksProxy) DetectProtocol(c gnet.Conn) (bool, error) {
    buf, err := c.Peek(socksHeaderLen)
    if err != nil {
        return false, err
    }
    return buf[0] == socks5.Ver && buf[1] != 0, nil
}
```

Таким чином були представлені методів та підходів до визначення протоколу з'єднання.

Висновки до третього розділу

В третьому розділі можна зробити висновок про високе значення та ефективність використання низькорівневих механізмів у сучасних мережесистемах, демонструючи їх важливість у створенні надійних та гнучких рішень для управління мережесистемами.

Оскільки рішення прийняті у проксі реакторі надає широкі можливості для подальшого розвитку та оптимізації, включаючи розширення підтримки протоколів, покращення алгоритмів аналізу трафіку та інтеграцію з іншими інструментами мережесистемного моніторингу. Використання eBPF для аналізу та маркування мережесистемного трафіку підкреслює потужність цієї технології у реалізації складних мережесистемних операцій безпосередньо в ядрі Linux.

Система показує високий рівень гнучкості та масштабованості, здатну обробляти різноманітні мережесистемні сценарії та адаптуватися до змінних умов мережі. Використання iptables та eBPF забезпечує високий рівень безпеки та надійності у управлінні мережесистемним трафіком, знижуючи ризики несанкціонованого доступу та забезпечуючи стабільність системи

.

РОЗДІЛ 4. РОЗРОБКА ВИСОКОРІВНЕВОГО КОДУ ОБРОБКИ МЕРЕЖЕВОГО ТРАФІКА ОБРОБКИ ПРОТОКОЛІВ

4.1 Впровадження та розробка алгоритму підключення до цільового сервера.

Для установки з'єднання з цільовим сервером використовується спеціальний клієнт, який встановлює асинхронне з'єднання з цільовим сайтом. Для цього відповідає файл `internal/client/downstream.go`.

Функція підключення:

```
func (d *Downstream) Dial(downstreamAddr string, localAddr
*net.TCPAddr, from gnet.Conn, timeout config.Timeout, queue
*proxy.TrafficQueue, closeCb CloseCallback) (*Context, error) {
    dialer := net.Dialer{Timeout: timeout.Connect, LocalAddr:
localAddr, KeepAlive: timeout.KeepAlive}
    conn, err := dialer.Dial("tcp", downstreamAddr)
    if err != nil {
        metrics.DialErrCounter.Inc()
        return nil, err
    }

    current, err := d.Client.Enroll(conn)
    if err != nil {
        metrics.DialErrCounter.Inc()
        _ = conn.Close()
        return nil, err
    }
    _ = current.SetLinger(0)

    select {
    case <-time.After(timeout.Connect):
        metrics.DialErrCounter.Inc()
        return nil, TimeoutErr
    case <-d.openedCon(current):
    }

    context := &Context{
        Current: current,
        To:      from,
        Closed:  false,
```

```

    closeCb: closeCb,
    queue:    queue,
  }
  current.SetContext(context)
  metrics.DialCounter.Inc()
  return context, nil
}

```

Проведемо аналіз черги, як частини обробки трафіку.

4.2 Реалізація оптимізованої черги пакетів.

Одна з найважливіших частин обробки трафіку є черга пакетів. У зв'язку з асинхронністю архітектури, що важливіше через частого дисбалансу між швидкістю каналів між користувачем і проксі сервером і проксі сервером і сервером до якого проксується трафік.

Один із найпоширеніших випадків коли між клієнтом та проксі сервером має більший пінг та малу швидкість, а проксі сервер має широкий канал та мінімальний пінг:

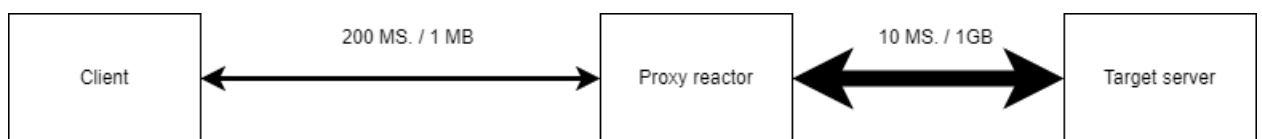


Рис. 4.1. Демонстрація дисбалансу між пропускною здатністю між клієнтом та проксі, проксі та сервером

Ця особливість змушує реалізовувати чергу пакетів, які складаються в пам'яті і відправляються по можливості. За це відповідає файл `internal\queue\queue.go`.

Головна частина логіки описана в наступних функціях:

```

func (t *TrafficQueue) write(to gnet.Conn, buf []byte) error {
    q := t.q
    if q == nil {
        return nil
    }
}

```

```

}

q.Enqueue(buf)

cbuf := q.Dequeue()
if cbuf == nil {
    return nil
}
return to.AsyncWrite(cbuf, func(c gnet.Conn, err error) error {
    byteslice.Put(cbuf)
    return nil
}))
}

func (t *TrafficQueue) CopyAsync(from gnet.Conn, to gnet.Conn)
(int64, error) {
    buf := byteslice.Get(from.InboundBuffered())
    _, err := from.Read(buf)
    if err != nil {
        byteslice.Put(buf)
        return 0, err
    }

    return int64(len(buf)), t.write(to, buf)
}

```

Цей код замість прямолінійного підходу у циклі:

1. Прочитати від відправника
2. Записати одержувачу

Який призводить до блокування цілого потоку на час очікування (що йде врозріз з архітектурою проксі реактора і в цілому не так ефективно).

Використовується складніший підхід, який не вимагає блокування потоку:

- Асинхронно отримати від відправника порцію трафіку
- Записати в кінець черги отриману порцію трафіку
- Отримати з черги перший пакет у черзі
- Надіслати на асинхронний запис одержувачу

Цей підхід складніший, але дозволяє гарантувати послідовність відправлення отриманого трафіку за умов різного пінгу, різних пропускових здібностей та інших мережових чинників.

```
byteslice pool
```

Замість використання алокації масивів байт для обробки та тимчасового зберігання пакетів у всьому проекті я намагаюся використовувати `byteslice`, який є спеціалізованою версією `sync.Pool` для масивів байт. Це дозволяє знизити навантаження на збирач сміття та в цілому оптимізувати використання пам'яті шляхом мінімізації фрагментації пам'яті.

4.3 Обробка протоколу SOCKS5

В основі обробки цієї протоки лежить машина станів, яка обробляє послідовність привітання - авторизація - підключення - тунелювання. У коді це описує такі стани:

```
const (
    negotiationStatus socksStatus = iota
    authStatus
    connectStatus
    tunnelStatus
)
```

Код обробки виглядає так:

```
func (s *socksProxy) handleTraffic(state *State, connection
gnet.Conn) error {
    var resp io.WriterTo
    var err error

    switch state.state {
    case tunnelStatus:
        if state.downstream.Closed {
            return proxy.ConnectionAborted
        }
    }
```

```

_, err := state.queue.CopyAsync(connection,
state.downstream.Current)
return err
case negotiationStatus:
    resp, err = s.negotiateHandle(state, connection)
case authStatus:
    resp, err = s.authHandle(state, connection)
case connectStatus:
    resp, err = s.handleConnect(state, connection)
}
if resp != nil {
    _, _ = state.queue.CopyWriterAsync(resp, connection)
}
return err
}

```

negotiateHandle

Відповідно до специфікації протоколу SOCKS5 в першому пакеті надсилається способи авторизації, що підтримуються клієнтом. Цей метод може перейти в наступні стани:

authStatus - клієнт підтримує авторизацію за логіном та паролем. Потрібно обробити наступний пакет у якому буде надіслано логін та пароль

connectStatus - клієнт надіслав, що не підтримує жодних способів авторизації (безпарольна авторизація). Якщо ір клієнта є у списку, то можливо пропустити етап авторизації і перейти відразу до стану connectStatus

У разі помилки протоколу або неможливості авторизації надсилається помилка згідно з протоколом SOCKS5 і з'єднання закривається.

```

func (s *socksProxy) negotiateHandle(state *State, c gnet.Conn)
(io.WriterTo, error) {
    req, err := socks5.NewNegotiationRequestFrom(c)
    if err != nil {
        return nil, err
    }

    var hasMethodUsernamePassword bool
    var hasMethodNone bool
    for _, method := range req.Methods {
        switch method {
            case socks5.MethodUsernamePassword:

```

```

        hasMethodUsernamePassword = true
    case socks5.MethodNone:
        hasMethodNone = true
    }
}

if hasMethodUsernamePassword {
    state.state = authStatus
    return userPassAuthResp, nil
}

if !hasMethodNone {
    return unsupportedAllResp, proxy.AuthRequiredErr
}
port, err := proxy.GetMark(c)
if err != nil {
    return unsupportedAllResp, err
}

addr := c.RemoteAddr().(*net.TCPAddr)
authInfo, authenticated := s.auth.AuthIp(port, addr.IP)
if !authenticated {
    return unsupportedAllResp, proxy.AuthRequiredErr
}
authState(state, authInfo)
return noneAuthResp, nil
}

func authState(state *State, authInfo *auth.Info) {
    state.authInfo = authInfo
    state.logger = state.logger.With(zap.String("user",
authInfo.Username), zap.String("local",
authInfo.LocalAddr.String()))
    state.state = connectStatus
}

```

authStatus

Пакет з логіном і паролем клієнта, що надсилається, розпарюється і перевіряється на коректність. Може переключатися в стан connectStatus, або відправити помилку авторизації клієнта та закрити з'єднання.

```

func (s *socksProxy) authHandle(state *State, c gnet.Conn)
(io.WriterTo, error) {

```

```

negotiationRequest, err :=
socks5.NewUserPassNegotiationRequestFrom(c)
if err != nil {
    return userPassStatusFailureResp, proxy.AuthRequiredErr
}

port, err := proxy.GetMark(c)
if err != nil {
    return userPassStatusFailureResp, err
}
authInfo, ok := s.auth.AuthUserPassword(port,
negotiationRequest.Uname, negotiationRequest.Passwd)
if !ok {
    return userPassStatusFailureResp, proxy.AuthRequiredErr
}
authState(state, authInfo)
return userPassStatusSuccessResp, nil
}

```

connectStatus

Відповідно до специфікації протоколу SOCKS5 клієнт надсилає команду на підключення до таргет серверів. У цей момент перевіряються всі бібліотеки (порт, айпі) і запускається підключення до сервера, який надіслав клієнт. Може переключатися в стан tunnelStatus, або надіслати помилку протоколу клієнту та закрити з'єднання.

```

func (s *socksProxy) handleConnect(state *State, connection
gnet.Conn) (io.WriterTo, error) {
    req, err := socks5.NewRequestFrom(connection)
    if err != nil {
        return nil, err
    }

    if req.Cmd != socks5.CmdConnect {
        return reply(req.Atyp, commandNotSupported4Resp,
commandNotSupported6Resp), proxy.InvalidRequestErr
    }

    downstreamAddr := req.Address()

    addr, err := dns.ResolveTCPAddr(downstreamAddr,
state.authInfo.Blacklist.IsBlacklistedHost)
    if err != nil {

```

```

        return reply(req.Atyp, hostUnreachable4Resp,
hostUnreachable6Resp), err
    }

    if state.authInfo.Blacklist.IsBlacklistedPort(addr.Port) {
        return reply(req.Atyp, hostUnreachable4Resp,
hostUnreachable6Resp), err
    }

    if state.authInfo.Blacklist.IsBlacklistedIp(addr.IP) {
        return reply(req.Atyp, hostUnreachable4Resp,
hostUnreachable6Resp), err
    }

    state.logger.Info("dial downstream", zap.String("target",
downstreamAddr))

    downstream, err := s.downstream.Dial(addr.String(),
state.authInfo.LocalAddr, connection, s.timeout, state.queue,
func(err error) {
        state.closed = true
    })
    if err != nil {
        return reply(req.Atyp, hostUnreachable4Resp,
hostUnreachable6Resp), err
    }

    state.downstream = downstream
    state.state = tunnelStatus
    return socks5.NewReply(socks5.RepSuccess, req.Atyp, req.DstAddr,
req.DstPort), nil
}

```

tunnelStatus

Кінцевий статус у якому обробник перекачує трафік між клієнтом та сервером використовуючи чергу пакетів:

```

if state.downstream.Closed {
    return proxy.ConnectionAborted
}
_, err := state.queue.CopyAsync(connection,
state.downstream.Current)
return err

```

4.4 Обробка протоколу HTTP(S)

У цьому обробнику за фактом приховано 2 протоколи, які досить сильно відрізняються у своїй логіці. Але для оптимізації роботи обробка була об'єднана в 1 обробник у зв'язку з тим, що обидва протоколи використовують формат HTTP для початкових пакетів (і наступних у разі HTTP)

Приклад пакету для проксіювання HTTP запиту:

```
GET http://example.com/ HTTP/1.1
Host: example.com
Proxy-Authorization: Basic <base64 encoded user:password>
User-Agent: curl/8.4.0
Accept: */*
Proxy-Connection: Keep-Alive
```

Приклад першого пакета для проксіювання запиту HTTPS:

```
CONNECT example.com:443 HTTP/1.1
Host: example.com:443
Proxy-Authorization: Basic <base64 encoded user:password>
User-Agent: curl/8.4.0
Proxy-Connection: Keep-Alive
```

Обробка обох протоколів починається з очікування того, як від клієнта надійдуть усі заголовки запиту:

```
func headersReady(c gnet.Conn) (bool, error) {
    buf, err := c.Peek(-1)
    if err != nil {
        return false, err
    }

    contains := bytes.Contains(buf, []byte("\r\n\r\n"))
    return contains, nil
}
```

```
func (h *httpProxy) negotiateHandle(state *State, c gnet.Conn)
([]byte, error) {
    if ready, err := headersReady(c); !ready || err != nil {
        return nil, err
    }
}
```

Далі вручну обробляються заголовки і формується "очищений" пакет, який у разі HTTP протоколу буде відправлений на таргет сервер. Використовуємо перший заголовок для отримання інформації про сервер, до якого потрібно підключитися. Цей підхід універсальний для HTTP та HTTPS

Якщо це заголовок Proxy-Authorization, то відзначаємо факт того, що користувач надіслав логін і пароль для подальшої авторизації (т.к. можлива безпарольна авторизація на підставі айпі користувача).

Видаляються “небезпечні” заголовки у разі їх наявності (Proxy-Authenticate, Proxy-Authorization, Proxy-Connection) задля забезпечення конфіденційності користувача у разі протоколу HTTP, т.к. надсилання цих заголовків розкриє перед таргет сервером факт використання користувачем проксі.

```
hasAuth := false
authenticated := false
first := true

port, err := proxy.GetMark(c)
if err != nil {
    return badRequestResp, err
}

var clientUsername string
headerCount := 0
for {
    line, err := untilRune(c, '\n')
    if err != nil {
        return nil, err
    }

    if bytes.Equal(line, headersEnd) {
        _, _ = buf.Write(headersEnd)
        break
    }
}
```

```

}

if first {
    first = false
    downstreamAddrTemp, err := getDownstream(line)
    if err != nil {
        return badGatewayResp, err
    }
    downstreamAddr = downstreamAddrTemp
    if len(downstreamAddrTemp) != 0 {
        line, err = transformRequestLine(line)
        if err != nil {
            return badRequestResp, proxy.InvalidRequestErr
        }
    }
    _, _ = buf.Write(line)
    continue
}

name := getHeaderName(line)
if bytes.EqualFold(name, authHeader) {
    hasAuth = true
    authInfo, authenticated, clientUsername = verifyAuth(line,
port, h.auth)
} else if !isBlacklistedHeader(name) {
    _, _ = buf.Write(line)
    headerCount += 1
}
}

if !hasAuth {
    addr := c.RemoteAddr().(*net.TCPAddr)
    authInfo, authenticated = h.auth.AuthIp(port, addr.IP)
}

if !authenticated {
    return authRequiredResp, fmt.Errorf("auth failed (hashAuth: %t,
username: %s)", hasAuth, clientUsername)
}

```

Подальша обробка поділяється на протоколи HTTPS і HTTP.

Цей протокол певною мірою дуже схожий з підходом SOCKS5. Проксі сервер після авторизації встановлюється підключення з сервером таргет і далі просто перекачує трафік в обидві сторони.

```
var (
    connectMethod = []byte("CONNECT")
)

parser := wilddcat.NewSizedHTTPParser(headerCount)
state.tunnel = bytes.EqualFold(parser.Method, connectMethod)

func (h *httpProxy) handleTraffic(state *State, c gnet.Conn) error
{
    ...
    if state.tunnel {
        _, err := state.queue.CopyAsync(c, state.downstream.Current)
        return err
    }
    ...
}
```

Логіка обробки цього протоколу складніша за HTTPS через те, що неможливо просто “передавати” трафік між сторонами, а потрібна повноцінна обробка пакета, трансформація заголовка та їх аналіз для визначення реального розміру запиту.

На попередньому кроці під час обробки трансформації заголовків був сформований вже безпечний для відправки на таргет сервер пакет, але для коректної роботи потрібно додатково дізнатися розмір тіла запиту шляхом вилучення зазначеного в заголовку *Content-Length* кількості байт.

На підставі цієї інформації запускається така логіка:

1. Підключаємо до таргету серверу
2. Відправляємо на таргет сервер очищені заголовки
3. Записуємо в state з'єднання у байт із заголовка *Content-Length* який потрібно передати “як є”.

Передача зазначеної кількості байт знаходиться в методі `httpProxy.handleTraffic`:

```
func (h *httpProxy) handleTraffic(state *State, c gnet.Conn) error {
    ...
    if state.tunnelSized > 0 {
        t, err := state.queue.CopyAsync(c, state.downstream.Current)
        if err != nil {
            state.tunnelSized = 0
            return err
        }

        atomic.AddInt64(&state.tunnelSized, -t)
        return nil
    }
    ...
}
```

Після завершення передачі вказаної кількості байт, система перевіряє, чи є ще дані для обробки. Якщо вхідний потік даних не вичерпаний, обробник продовжує приймати та аналізувати наступні пакети, забезпечуючи неперервну передачу даних між клієнтом та сервером. Це дозволяє обробляти множинні запити або тривалі сесії без переривань.

4.5 Порівняння продуктивності з готовим рішенням

Заміри будуть проводитися на vps у хостері DigitalOcean:

Вимоги: система повинна мати можливість слухати 5000 портів HTTP(S) + SOCKS5 і підтримувати без пароля авторизацію через IP.

Процес тестування: з сервера в цьому ж датацентрі звертається 100 потоків до випадкових портів в діапазоні 5000 тисяч через проксі викачується файл розміром 16 КБ.

Заміри будуть проводитися на vps у хостері DigitalOcean:

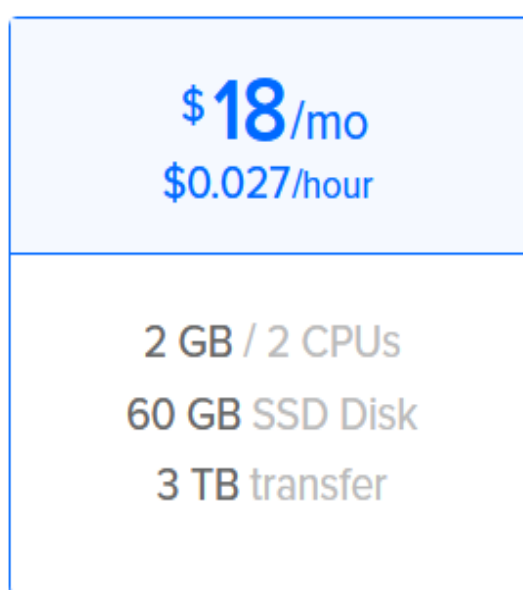


Рис.4.2. Демонстрація vps у хостері DigitalOcean

```
IP-проксі сервер: 206.81.31.130
```

Зрешту - програмне забезпечення не підтримує 1 порт = 2 протоколи, для виконання цього обмеження потрібно слухати 5000 * 2 портів. Конфігурації файлу:

```
nscache 65536
nserver 8.8.8.8
nserver 8.8.4.4
```

```
config /conf/3proxy.cfg
monitor /conf/3proxy.cfg

log /logs/3proxy-%y%m%d.log D
rotate 60
counter /count/3proxy.3cf

users $/conf/passwd

include /conf/counters
include /conf/bandlimiters

auth iponly
allow * 164.90.217.227

socks -n -p10000
proxy -n -p20000
socks -n -p10001
proxy -n -p20001
socks -n -p10002
proxy -n -p20002

... ..

socks -n -p14998
proxy -n -p24998
socks -n -p14999
proxy -n -p24999
socks -n -p15000
proxy -n -p25000

flush
allow admin

admin -p8080
```

Стан простою системи представлено на рис. 4.3. Демонстрація навантаження 3proxy на vps під час простою дозволяє проводити аналіз та приймати управлінські рішення.

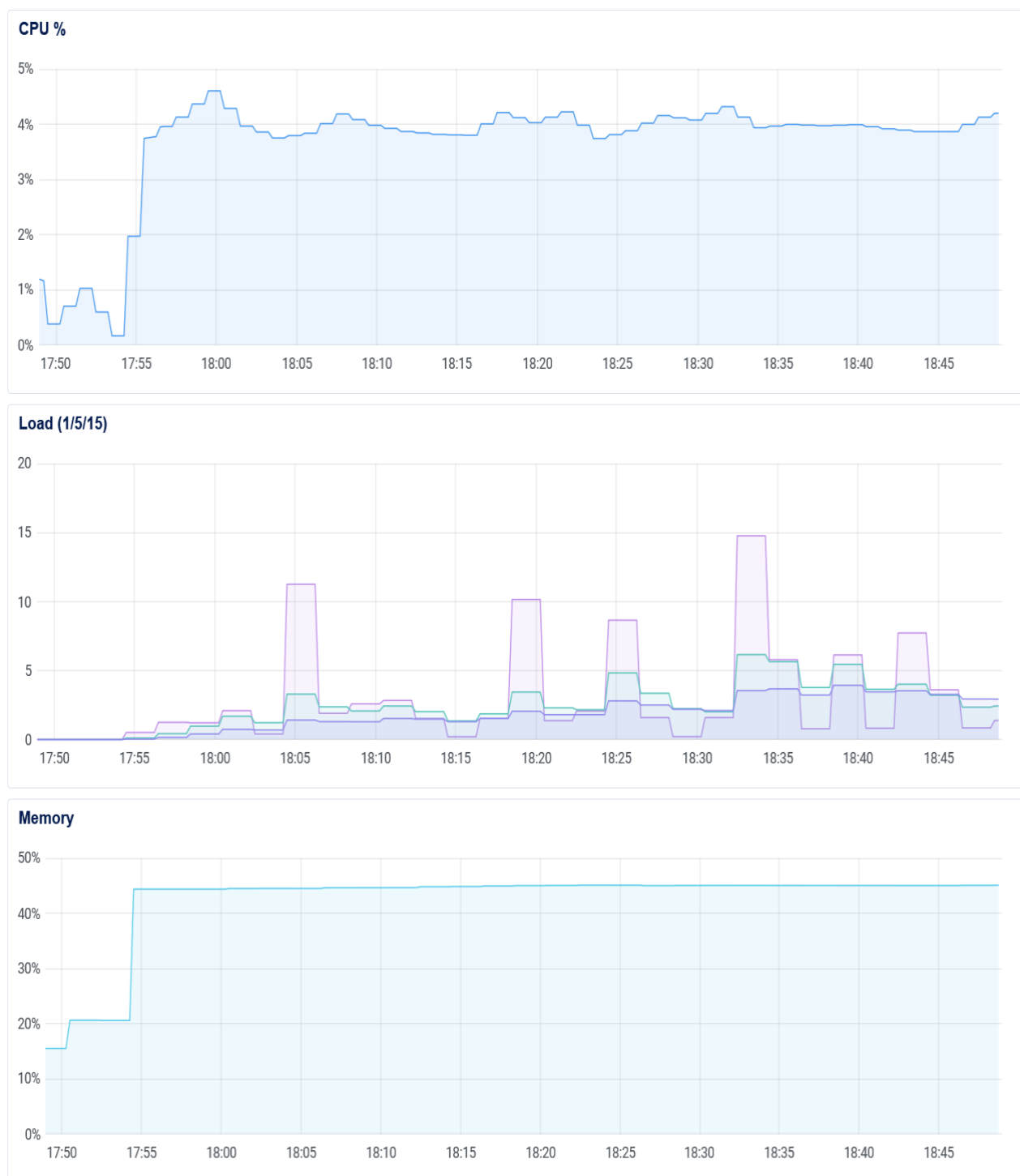


Рис.4.3. Демонстрація навантаження Зроху на vps під час простою.

Під час навантаження:

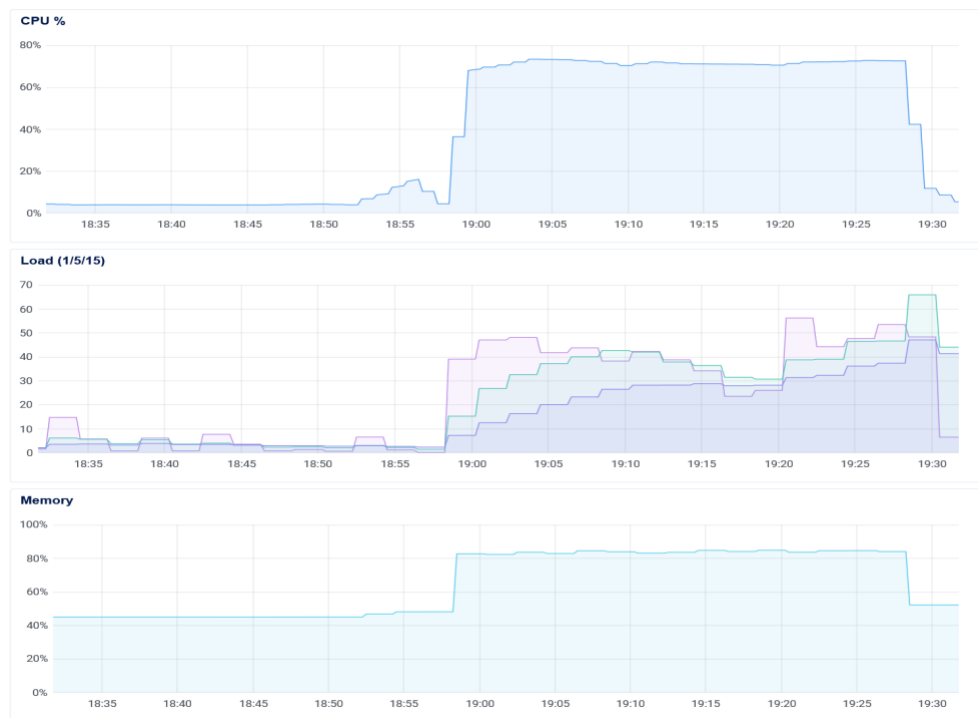


Рис.4.4. Демонстрація навантаження Зроху на vps під час навантаження

Демонстрація навантаження Проксі-реактора на vps під час простою представлено на рис.4.5.

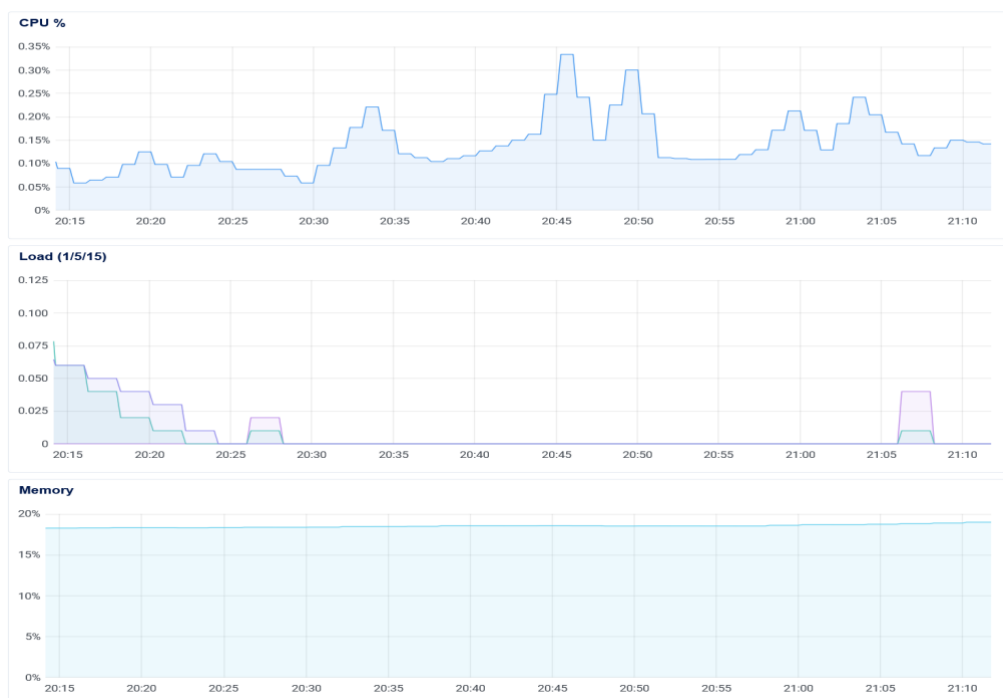


Рис.4.5. Демонстрація навантаження Проксі-реактора на vps під час простою.

Під час навантаження:

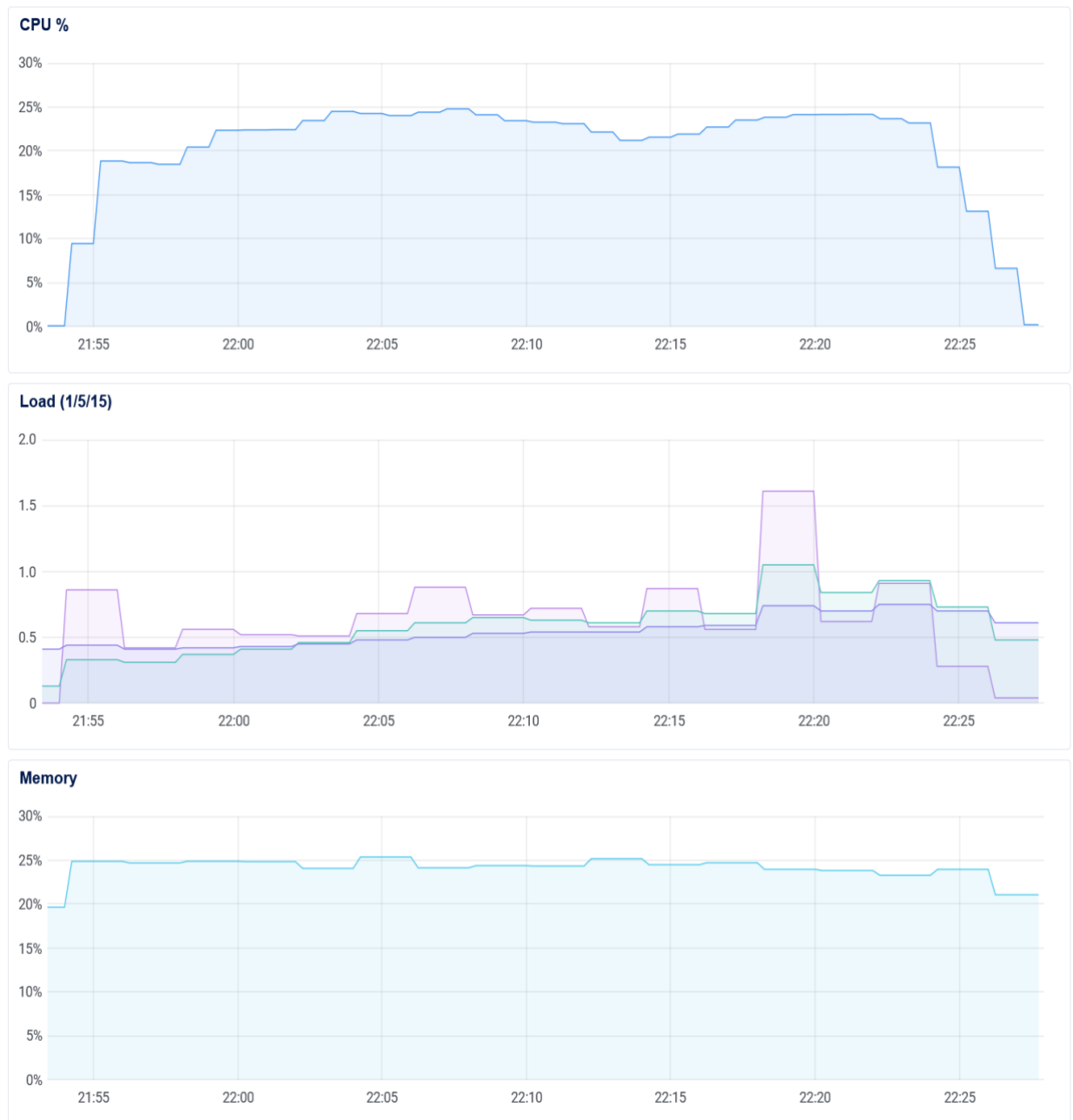


Рис.4.6. Демонстрація навантаження Проксі-реактора на vps під час простою.

Під час навантаження проксі-реактор продемонстрував значно кращу продуктивність та ефективність управління ресурсами порівняно з Зргоху, забезпечуючи стабільність та високу швидкість обробки запитів навіть при великій кількості одночасних з'єднань.

Висновки до 4 розділу

Описано використання високорівневих методів обробки мережевого трафіку, зокрема в контексті проксі-серверів. Розроблена система використовує асинхронні методи для установки з'єднання з цільовими серверами, що дозволяє оптимізувати обробку запитів та підвищити загальну продуктивність.

Визначено, що ключовим елементом системи є реалізація черги пакетів, яка враховує асинхронність архітектури та дисбаланс швидкостей каналів між користувачем, проксі-сервером та цільовим сервером. Ця черга забезпечує ефективне управління трафіком та знижує ризики втрати даних при високих навантаженнях.

Обробка протоколів SOCKS5 та HTTP(S) виконується за допомогою машини станів, що дозволяє точно керувати процесом обробки з'єднань та тунелювання. Для протоколу SOCKS5 використовується послідовність станів для обробки привітання, авторизації, підключення та тунелювання, тоді як для HTTP(S) використовується більш складна логіка обробки, що включає аналіз заголовків та трансформацію запитів.

Особливу увагу заслуговує використання byteslice pool для оптимізації використання пам'яті та зниження навантаження на збирач сміття. Це підкреслює система зосереджена на деталях та прагненні до оптимізації ресурсів.

Загалом, розроблена система ефективно поєднує асинхронність, гнучкість та продуктивність, що робить її важливим внеском у сферу мережових технологій.

ВИСНОВКИ

Кваліфікаційна магістерська робота присвячена питанню оптимізації продуктивності проксі-серверів, відображає глибоке розуміння актуальних викликів у сфері мережевих технологій. Актуальність теми підкріплена постійним зростанням навантаження на проксі-сервери, яке спричинене збільшенням потреб у високопродуктивних, надійних та безпечних мережевих послугах.

Основна увага у роботі приділена аналізу існуючих проблем продуктивності, які особливо виражені в контексті багатопортового доступу та безпарольних методів аутентифікації. Розроблені теоретичні основи нових методів оптимізації, включаючи використання паралельних та асинхронних алгоритмів, динамічного розподілу навантаження між портами, а також інноваційні рішення з використанням eBPF та алгоритмів зворотного відстеження для визначення оригінального порту.

Експериментальна перевірка та тестування запропонованих підходів показали їх ефективність у практичних умовах, що дозволило істотно підвищити продуктивність проксі-серверів. Отримані результати не тільки підтверджують наукову новизну роботи, але й відкривають шлях для подальшого розвитку та удосконалення мережевих технологій.

Таким чином, дана дипломна робота вносить вклад у сферу мережевих технологій, пропонуючи нові рішення для вирішення проблем продуктивності проксі-серверів, що мають важливе практичне значення для сучасної мережевої інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fortinet [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.fortinet.com/resources/cyberglossary/proxy>.
2. Varonis [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.varonis.com/blog/what-is-a-proxy-server>.
3. UpGuard: Free Writing AI Assistance [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.upguard.com/blog/proxy-server>.
4. Kaspersky [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.kaspersky.com/resource-center/definitions/what-is-a-proxy-server>
5. Cisco [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.cisco.com/c/en/us/support/docs/security/web-security-appliance/117995-qna-wsa-00>.
6. TechTarget: [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.techtarget.com/searchnetworking/definition/proxy>.
7. Microsoft [Електронний ресурс]. – Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/windows-server/networking/technologies/bgp/bgp-routing>.
8. Merriam-Webster. (n.d.). Proxy. In Merriam-Webster.com dictionary. [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.merriam-webster.com/dictionary/proxy>.
9. S. R. Bhandari, G. M. Lee and N. Crespi, "Peer to Peer Proxy Based IPTV Services," 2009 IEEE Globecom Workshops, Honolulu, HI, USA, 2009.
10. Michelle Boonen, B. Lievens, 2018 [Електронний ресурс]. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-319-91125-0_17.
11. Peep Küngas, M. Dumas, 2011 [Електронний ресурс]. – Режим доступу до ресурсу: <https://dl.acm.org/doi/10.1145/1982185.1982523>.

12. Soe Htet, T. Ludwick, A. Mahal, 2019 [Электронный ресурс]. – Режим доступа до ресурсу: <https://onlinelibrary.wiley.com/doi/10.1111/tmi.13286>.
13. A. Acosta, Carlos Marin, P. G. García, Yuri Vanessa Nieto Acevedo, "Method of auto-configuration for corporate proxies". [Электронный ресурс]. – Режим доступа до ресурсу: <https://dx.doi.org/10.16925/IN.V13I21.1723>.
14. Xiong Chunsha, "Design of Internet Proxy Server in Information Service Network". [Электронный ресурс]. – Режим доступа до ресурсу: http://www.c-s-a.org.cn/csaen/article/abstract/20020515?st=article_issue.
15. I. Khalil, Ge PeiQi, "Efficient Proxy Selection in Cooperative Web Caching". [Электронный ресурс]. – Режим доступа до ресурсу: <https://ieeexplore.ieee.org/document/4444116>.
16. Al Imtiaz, Md. Jayed Hossain, "Distributed cache management architecture: To reduce the internet traffic by integrating browser and proxy caches". [Электронный ресурс]. – Режим доступа до ресурсу: <https://ieeexplore.ieee.org/document/6919088>
17. J. K. Duke, Xiaochang Yu, "Authenticating users inside and outside the library: implementing a proxy server", [Электронный ресурс]. – Режим доступа до ресурсу: https://www.tandfonline.com/doi/abs/10.1300/J136v04n03_05
18. Ari Luotonen, "Web Proxy Servers. [Электронный ресурс]. – Режим доступа до ресурсу: <https://dl.acm.org/doi/abs/10.5555/275677>
19. Cheng Ming, "The Project of User Management System of Multifunctional IP Proxy Server". [Электронный ресурс]. – Режим доступа до ресурсу: <https://dl.acm.org/doi/abs/10.334555/12327>.
20. Hukeun Kwak, Kyusik Chung, "A Performance Improvement Scheme for a Wireless Internet Proxy Server Cluster". [Электронный ресурс]. – Режим доступа до ресурсу: https://www.researchgate.net/publication/239440573_A_Performance_Improvement_of_Linux_TCPIP_Stack_based_on_Flow-Level_Parallelism_in_a.
21. An Educational HTTP Proxy Server by M. Sysel, O. Dolezal. [Электронный ресурс]. – Режим доступа до ресурсу: https://www.researchgate.net/publication/261104026_An_Educational_HTTP_Proxy_Server.

22. Pretty-Bad-Proxy: An Overlooked Adversary in Browsers' HTTPS Deployments" by Shuo Chen, Ziqing Mao, Yi-Min Wang, Ming Zhang. [Электронный ресурс]. – Режим доступа до ресурсу: <https://ieeexplore.ieee.org/document/5207655>
23. R. Fielding, U.C. Irvine, J. Gettys, J. Mogul – "Standards Track Hypertext Transfer Protocol -- HTTP/1.1" June 1999. [Электронный ресурс]. Режим доступа до ресурсу: <https://datatracker.ietf.org/doc/html/rfc2616>.
24. Hypertext Transfer Protocol - HTTP/1.1 draft-ietf-http-v11-spec-rev-06. [Электронный ресурс]. Режим доступа до ресурсу: <https://datatracker.ietf.org/doc/draft-ietf-http-v11-spec-rev/06/>
25. HTTP/2: The Long-Awaited Sequel [Электронный ресурс]. - Режим доступа до ресурсу: <https://docs.microsoft.com/en-us/archive/blogs/ie/http2-the-long-awaited-sequel>.
26. HTTP: A Probe-Resistant Proxy" by Sergey Frolov, Eric Wustrow. [Электронный ресурс]. – Режим доступа до ресурсу: <https://dblp.org/rec/conf/uss/FrolovW20.html>.
27. Vpnunlimited [Электронный ресурс]. – Режим доступа до ресурсу: <https://www.vpnunlimited.com/ua/help/specials/what-is-socks5-proxy>.
28. Surfshark [Электронный ресурс]. – Режим доступа до ресурсу: <https://surfshark.com/uk/blog/proxy-server>.
29. Zoogvpn [Электронный ресурс]. – Режим доступа до ресурсу: <https://zoogvpn.com/products/socks5-proxy/>.
30. Nordvpn [Электронный ресурс]. – Режим доступа до ресурсу: <https://nordvpn.com/uk/blog/proksi-proty-vpn/>.
31. 3proxy [Электронный ресурс]. – Режим доступа до ресурсу: <https://3proxy.org/documents/>.
32. Squid-cache [Электронный ресурс]. – Режим доступа до ресурсу: <http://www.squid-cache.org/Doc/>