МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗИМИЧ АРТЕМ ПАВЛОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА ПРОКСІ-СЕРВІСУ ДЛЯ ЗАБЕЗПЕЧЕННЯ ДОСТУПНОСТІ ВЕБ-РЕСУРСІВ В УМОВАХ МЕРЕЖЕВИХ ОБМЕЖЕНЬ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Олександр ПАВЛЮК, старший викладач
кафедри інформаційних технологій,
к.т.н.
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Зимич Артем Павлович. Розробка проксі-сервісу для забезпечення доступності веб-ресурсів в умовах мережевих обмежень. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У роботі розроблено багатопротокольний проксі-сервіс для обходу систем обмежень. Реалізовано підтримку AmneziaWG, Xray-core, Hysteria2 та Slipstream. Створено сайт проекту та скрипти автоматизації розгортання серверного середовища. Результатом є програмний комплекс, що забезпечує стабільний та прихований доступ до веб-ресурсів в умовах жорстких мережевих обмежень.

Ключові слова: проксі-сервіс, DPI, обхід блокувань, AmneziaWG, Xray-core, Hysteria2, DNS-tunnel.

55 ст. 8 рис., 3 дод., 36 джерел.

ABSTRACT

Zymych A. Development of a proxy service to ensure the availabillity of web resources under network restrictions. Specialization 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

This paper describes the development of a multi-protocol proxy service designed to bypass network restrictions. Support for AmneziaWG, Xray-core, Hysteria2, and Slipstream has been implemented. A project website and scripts for automating the deployment of the server environment have been created. The result is a software suite that provides stable and stealthy access to web resources under strict network restrictions.

Keywords: proxy service, DPI, restrictions circumvention, AmneziaWG, Xray-core, Hysteria2, DNS-tunnel.

ЗМІСТ

ВСТУП.....5
РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ЗАБЕЗПЕЧЕННЯ ДОСТУПНОСТІ ВЕБ-РЕСУРСІВ.....8
1.1 Передумови та цільові засади мережевих обмежень.....8
1.2 Технічні методи блокування доступу до ресурсів.....13
1.3 Порівняльний огляд протоколів обходу блокувань та засобів приховування трафіку.....21
1.4 Формування вимог до функціональності та безпеки проксі-сервісу.....26
РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ ДОСТУПУ В УМОВАХ ОБМЕЖЕНЬ.....29
2.1. Модель архітектури взаємодії проксі-вузлів.....29
2.2 Модель маршрутизації трафіку та логіка тунелювання.....33
2.3. Проєктування архітектури внутрішнього API та бази даних для автоматизації керування доступом.....35
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕХНІЧНА ОЦІНКА ПРОКСІ-СЕРВІСУ.....41
3.1 Конфігурування протоколів обходу блокувань та створення скриптів автоматизації.....41
3.2 Реалізація веб-ресурсу та клієнтської логіки.....45
3.3 Реалізація серверної частини проксі-сервісу.....48
ВИСНОВКИ.....50
СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ.....52
ДОДАТОК А.....57
ДОДАТОК Б.....61
ДОДАТОК В.....67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Мережа доставки контенту (Content Delivery Network, CDN) - мережа взаємопов'язаних серверів, яка пришвидшує завантаження вебсторінок для вебсайтів із великим обсягом даних.

Глибока перевірка пакетів (Deep Packet Inspection, DPI) - технологія, яка використовується для сканування вмісту кожного пакета даних, що передається комп'ютерною мережею. DPI може застосовуватися для управління мережею, а також для фільтрації, блокування та перенаправлення інтернет-трафіку.

Система доменних імен (Domain Name System, DNS) - ієрархічна система, яка перетворює доменні імена (назву, введену в адресний рядок веббраузера, наприклад, github.com) у IP-адреси, що визначають потрібне місцезнаходження у мережі.

Точки обміну інтернет-трафіком (Internet Exchange Point, IX) - вузли, які дозволяють інтернет-провайдерам обмінюватися трафіком безпосередньо між своїми мережами без використання транзитних послуг третіх сторін.

Сертифікати безпеки транспортного рівня (TLS) - цифрові сертифікати, які забезпечують захищене з'єднання між веб-браузером і сервером.

Технічні засоби протидії загрозам (Технические Средства Противодействия Угрозам, ТСПУ) - обладнання, встановлене в мережах російських інтернет-провайдерів, яке дає змогу здійснювати глибинну перевірку пакетів для фільтрації, блокування, перенаправлення або модифікації інтернет-трафіку відповідно до державних вимог. ТСПУ використовуються, зокрема, для реалізації реєстру заборонених сайтів, блокування протоколів та інших засобів обходу обмежень, а також для нагляду за трафіком.

Мережевий рівень (L3) - третій рівень моделі OSI, відповідальний за логічну адресацію, маршрутизацію та передачу пакетів між різними мережами. Для ідентифікації пристроїв використовуються IP-адреси, а основними протоколами виступають IPv4, IPv6, а також протоколи маршрутизації та діагностичний ICMP.

Транспортний рівень (L4) - четвертий рівень моделі OSI, що забезпечує передачу даних між процесами на кінцевих пристроях із можливістю контролю з’єднання, цілісності та потоку. Ідентифікація кінцевих точок відбувається через порти. Найпоширеніші протоколи: TCP із гарантованою доставкою та UDP, який працює без встановлення з’єднання.

Прикладний рівень (L7) - найвищий сьомий рівень моделі OSI, який забезпечує безпосередню взаємодію застосунків із мережевими сервісами, оперуючи даними, що генеруються програмним забезпеченням. Приклади протоколів: HTTP/HTTPS, FTP, SMTP, SSH, DNS та багато інших.

ВСТУП

Актуальність теми дослідження. Забезпечення стабільного доступу до інформаційних ресурсів у сучасному цифровому просторі стикається з дедалі складнішими викликами: від географічних обмежень до впровадження інтелектуальних систем аналізу та глибокої фільтрації трафіку. Зростаюча складність методів блокування з боку інтернет-провайдерів стимулює розвиток нових технологій приховування мережевої активності. Існуючі базові проксі-рішення часто виявляються вразливими до сучасних методів детектування, що створює потребу в інтеграції більш досконалих засобів обфускації та тунелювання даних. Використання комплексного підходу, який об’єднує протоколи AmneziaWG, Xray-core, Hysteria2 та технологію транспортування Slipstream у межах єдиного сервісу, дозволяє суттєво підвищити стійкість до активного аналізу трафіку. Таким чином, розробка спеціалізованого проксі-сервісу, здатного працювати в умовах суворих мережевих обмежень, є надзвичайно актуальною задачею.

Метою роботи є розробка проксі-сервісу для забезпечення стабільної доступності веб-ресурсів в умовах мережевих обмежень. Для досягнення поставленої мети було визначено такі завдання:

- Проаналізувати сучасні методи мережевих обмежень трафіку та порівняти існуючі протоколи їх обходу.
- Розробити схему маршрутизації та логіку спрямування трафіку через захищені тунелі.
- Спроєктувати архітектуру внутрішнього API та структуру бази даних для автоматизації керування доступом.
- Реалізувати програмні засоби автоматизації на основі скриптів та веб-ресурс для взаємодії з користувачами.

Об’єктом дослідження є процес забезпечення доступності веб-ресурсів у комп’ютерних мережах з обмеженим доступом.

Предметом дослідження є методи, протоколи та програмні засоби побудови проксі-сервісу для обходу мережевих обмежень.

Практичне значення одержаних результатів. Знання та розробки, отримані в ході виконання кваліфікаційної роботи, становлять готову технологічну базу для розгортання стійких до цензури проксі-вузлів. Реалізовані алгоритми автоматизації через внутрішнє API та скрипти дозволяють швидко конфігурувати багатопротокольні шлюзи доступу. Результати дослідження можуть бути використані системними адміністраторами та фахівцями з кібербезпеки для побудови захищених корпоративних або приватних мереж, стійких до методів DPI-фільтрації.

Апробація результатів дослідження. Результати роботи апробації не проходили.

Структура кваліфікаційної роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних посилань та додатків. Основна частина викладена на 54 сторінках. Робота містить 8 рисунків та список використаних посилань із 29 найменувань.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ЗАБЕЗПЕЧЕННЯ ДОСТУПНОСТІ ВЕБ-РЕСУРСІВ

1.1. Передумови та цільові засади мережевих обмежень

Забезпечення стабільного функціонування сучасних інформаційно-телекомунікаційних систем неможливе без впровадження чітко визначених механізмів управління та обмеження мережевого трафіку. Розвиток технологій передачі даних та повсюдна цифровізація суспільних процесів зумовили появу нових викликів, пов’язаних з інформаційною безпекою і з ефективністю використання наявних обчислювальних ресурсів. У цьому контексті мережеві обмеження виступають не лише технічним інструментом фільтрації пакетів, а є складним комплексом заходів, спрямованих на реалізацію стратегічних цілей організацій та виконання вимог державного законодавства. Глибокий аналіз причин запровадження таких обмежень дозволяє виокремити три основні вектори впливу: внутрішньокорпоративний, що фокусується на безпеці та продуктивності; державне регулювання, спрямоване на боротьбу з незаконним контентом; та цензуру як інструмент політичного контролю.

Розглядаючи корпоративний сектор, слід зазначити, що фундаментальною причиною впровадження мережевих обмежень є необхідність створення захищеного периметра організації. В умовах постійного зростання кількості кіберзагроз, корпоративні політики безпеки стають першим ешелоном оборони проти витоків конфіденційної інформації. Запобігання несанкціонованому виведенню інтелектуальної власності, персональних даних клієнтів або фінансової звітності за межі внутрішньої мережі потребує суворого контролю над усіма вихідними з’єднаннями. Використання сучасних міжмережевих екранів дозволяє не лише блокувати підозрілу активність, а й здійснювати глибокий аналіз трафіку на прикладному рівні, забезпечуючи цілісність інформаційних активів підприємства.

Окрім аспектів безпеки, важливим чинником у корпоративному середовищі є економічна доцільність та оптимізація робочого часу персоналу.

Обмеження доступу до нецільових інтернет-ресурсів, таких як соціальні мережі, відеохостинги розважального спрямування та ігрові платформи, розглядається як інструмент підвищення продуктивності праці. Неконтрольований доступ до мультимедійного контенту високої чіткості створює значне паразитне навантаження на канали зв'язку, що призводить до деградації сервісів, критично важливих для бізнес-процесів: систем відеоконференцій, корпоративних рішень для планування ресурсів та хмарних сервісів спільної роботи. Таким чином, мережеві обмеження сприяють раціональному розподілу пропускної здатності каналів, пріоритезуючи трафік, що має безпосереднє відношення до виконання посадових обов'язків співробітників. Технічна реалізація корпоративних обмежень часто базується на принципі мінімальних привілегій, згідно з яким доступ надається лише до тих ресурсів, які є необхідними для виконання конкретних робочих задач. Це дозволяє мінімізувати поверхню атаки всередині мережі та запобігти горизонтальному переміщенню зловмисників у разі компрометації одного з робочих вузлів.

Поряд із корпоративними інтересами, вагому роль у формуванні архітектури мережевих обмежень відіграє державне регулювання інформаційного простору. Основними цілями державного втручання є захист національної безпеки, підтримання правопорядку та боротьба з розповсюдженням протиправного контенту. У сучасних правових системах мережеві обмеження розглядаються як легітимний інструмент протидії тероризму та екстремізму. Блокування ресурсів, що використовуються для координації злочинної діяльності, вербування або розповсюдження деструктивної пропаганди, є обов'язковою умовою функціонування інтернет-провайдерів у межах правового поля більшості країн. Державні регулятори формують переліки заборонених ресурсів, доступ до яких має бути обмежений на рівні магістральних та локальних вузлів зв'язку.

Важливим аспектом державного регулювання є також захист прав інтелектуальної власності. Боротьба з цифровим піратством змушує держави впроваджувати механізми блокування доступу до сайтів, що нелегально

поширюють контент, захищений авторським правом. Це включає не лише одноранговий обмін файлами, а й стрімінгові сервіси, що функціонують поза межами ліцензійних угод. Виконання судових рішень щодо припинення доступу до таких ресурсів вимагає від технічних спеціалістів застосування методів фільтрації за IP-адресами або DNS-блокування. Окремої уваги заслуговує використання мережевих обмежень як інструменту боротьби з незаконними фінансовими операціями та неліцензованою гральною діяльністю. Держава, піклуючись про економічну безпеку та соціальну стабільність, обмежує доступ до незареєстрованих онлайн-казино, платформ для проведення несанкціонованих лотерей та сумнівних інвестиційних проектів. Це дозволяє зменшити ризики шахрайства щодо громадян та забезпечити прозорість фінансових потоків у мережі.

Найбільш дискусійним, проте технічно найбільш розвиненим різновидом мережевих обмежень є цензура як інструмент політичного контролю. На відміну від державного регулювання, що має законодавче обґрунтування та спрямоване на боротьбу з очевидно протиправним контентом, політична цензура має на меті обмеження доступу до інформації, яка може підірвати авторитет влади, спровокувати протестні настрої або поширити альтернативні точки зору. Сучасні дослідження інтернет-цензури пропонують концептуально новий погляд на її природу: цензура не обов’язково має бути абсолютною, щоб досягати цілей уряду. Так звана «пориста цензура», коли доступ до інформації ускладнюється, але не блокується повністю, слугує інтересам держави ефективніше, ніж ідеально функціонуючий фаєрвол. Відповідно до економічної аналогії, цензуру слід розуміти не як заборону, а як податок на інформацію, який робить певні ресурси відносно дорожчими з точки зору витраченого часу, грошей та зусиль [20].

Виокремлюють три основні складові цензури: fear, friction та flooding. Режим страху передбачає чіткі правила щодо дозволеного контенту та покарання за їх порушення. Проте такий підхід має недоліки: він привертає увагу до забороненої інформації, викликає цікавість та обурення. Тому страхова цензура

застосовується цілеспрямовано - переважно щодо публічних осіб із широким охопленням аудиторії [20].

Складова тертя є основним механізмом функціонування багатьох національних систем фільтрації. Він передбачає створення бар'єрів, які ускладнюють доступ до інформації, навіть якщо технічно можливо їх обійти без жодних правових наслідків. Користувачі є раціональними споживачами інформації - вони не споживають увесь доступний контент, а обирають те, що варте зусиль. Коли державний фаєрвол блокує сайт, це підвищує вартість доступу до нього, що вимірюється незручностями, пов'язаними з обходом: пошуком спеціалізованого програмного забезпечення, його оплатою, вивченням інструкцій. Якщо вартість доступу достатньо висока, користувач знайде інше заняття. Це явище описують через поняття еластичності попиту на інформацію: якщо обхід блокування є складним, доступ здійснюватимуть лише ті, для кого інформація є надзвичайно важливою. Дослідження показують, що лише невеликий відсоток користувачів в країнах із жорсткою цензурою регулярно вдається до обходу, причому ця група переважно складається з молоді, політично обізнаних осіб та тих, хто володіє іноземними мовами. Таким чином, фаєрвол створює ефективний, хоч і пористий, бар'єр між тими, хто здатен і бажає обходити блокування, та широкою публікою, яку активний клас міг би мобілізувати [20].

Складова «затоплення» полягає в координації державних зусиль для дезорганізації дискусій та відволікання уваги від небажаних тем. Якщо тертя зменшує доступ до певної інформації, підвищуючи її вартість, то затоплення досягає тієї ж мети шляхом зниження вартості альтернатив. Цей механізм особливо корисний у кризових ситуаціях, коли попит на інформацію є нееластичним, а самого тертя недостатньо. Інфраструктура затоплення включає скоординовані інструкції для засобів масової інформації та армії платних коментаторів, які публікують величезну кількість повідомлень, щоб контролювати та обмежувати обговорення. Метою є не виграш дебатів, а запобігання дебатам як таким [20].

Окремої уваги заслуговує Росія, де за останні роки сформувалася одна з найбільш технологічно розвинених систем мережевого контролю у світі. Починаючи з ухвалення закону про «суверенний інтернет» у 2019 році, держава створила інфраструктуру ТСПУ, які встановлюються на рівні інтернет-провайдерів і дозволяють здійснювати централізовану фільтрацію трафіку. Після повномасштабного вторгнення в Україну в 2022 році цензура набула безпрецедентних масштабів. Влада заблокувала доступ до багатьох незалежних ЗМІ, соціальних мереж, зокрема Facebook, Instagram та X, а також до месенджерів Signal, Viber, WhatAapp та з недавніх пір Telegram. Для обґрунтування таких блокувань використовуються формулювання про нібито використання цих платформ для «терористичних» цілей, хоча незалежні спостерігачі розцінюють це як політично вмотивовану цензуру.

Окрім блокування окремих ресурсів, Росія активно впроваджує методи масової фільтрації на рівні протоколів. У 2023 році було зафіксовано перші випадки блокування протоколів OpenVPN та WireGuard на рівні ТСПУ [30]. Системи глибокого аналізу пакетів навчилися виявляти та блокувати протоколи, що раніше активно використовувалися для обходу обмежень, зокрема деякі транспорти VLESS, тоді як інші конфігурації цього протоколу залишаються працездатними [29]. Це свідчить про еволюцію російської системи цензури: від блокування окремих IP-адрес і доменів до глибокої інспекції пакетів і фільтрації на рівні протоколів, що робить обхід блокувань дедалі складнішим для звичайних користувачів. Також держава активно тисне на закордонних хостинг-провайдерів, блокуючи сайти Amazon Web Services, Cloudflare, DigitalOcean та Hetzner. Це створює додаткові перешкоди для розміщення незалежного контенту та змушує російських користувачів і компанії переходити на локальні, підконтрольні державі платформи.

Особливості функціонування російської системи мережевих обмежень мають безпосередній вплив на доступність інформаційних ресурсів для мешканців тимчасово окупованих територій України. На цих територіях застосовуються ті ж самі методи фільтрації трафіку, що й на решті території

Росії. Доступ до українських інтернет-ресурсів, багатьох міжнародних новинних сайтів та популярних соціальних мереж технічно обмежується на рівні провайдерів, які працюють під контролем місцевої адміністрації. Фіксується також практика блокування певних протоколів та сервісів, які потенційно можуть бути використані для отримання інформації з альтернативних джерел. Таким чином, аналіз причин та цілей запровадження мережевих обмежень демонструє широкий спектр мотивацій - від суто технічних завдань забезпечення безпеки до складних політичних механізмів контролю суспільства. Розуміння цих різнорівневих мотивацій є необхідною умовою для подальшої технічної класифікації методів блокування та створення ефективних засобів забезпечення доступності веб-ресурсів.

1.2 Технічні методи блокування доступу до ресурсів

Технічні методи обмеження доступу до інформаційних ресурсів можна класифікувати відповідно до рівнів моделі OSI, на яких вони реалізуються. Кожен наступний рівень дозволяє здійснювати більш точну та гнучку фільтрацію, але потребує значно більше обчислювальних ресурсів та спеціалізованого обладнання. Від найпростіших методів на мережевому рівні до складних систем глибокої інспекції пакетів на прикладному - еволюція засобів блокування відображає постійне змагання між тими, хто обмежує доступ, і тими, хто намагається його обійти. Розуміння особливостей кожного з цих методів є необхідною передумовою для створення ефективних засобів забезпечення доступності веб-ресурсів.

Найпростішим і найдешевшим з точки зору обчислювальних витрат є блокування на мережевому рівні L3, яке полягає у фільтрації трафіку за IP-адресами відправника або отримувача. Цей метод реалізується на маршрутизаторах та міжмережевих екранах шляхом ведення чорних списків IP-адрес або цілих підмереж. Якщо пакет має адресу призначення, яка потрапляє до такого списку, він відкидається без подальшої обробки. Існує також практика геоблокування, коли обмежується доступ з IP-адрес, зареєстрованих у певних

країнах. Перевагами цього методу є простота реалізації та висока швидкодія порівняно з іншими. Однак недоліки також очевидні: блокування за IP є неефективним проти ресурсів, які часто змінюють свої адреси, або розміщені на спільних хостингах разом з легітимними сайтами. У таких випадках блокування однієї IP-адреси може зробити недоступними десятки або навіть сотні різних ресурсів, що призводить до надмірного обмеження.

Дещо вищим є блокування на транспортному рівні L4, яке оперує не лише IP-адресами, а й портами TCP та UDP. Порт не є окремим рівнем, а є характеристикою протоколів TCP та UDP, які функціонують на транспортному рівні. Один і той самий порт може бути відкритим одночасно для TCP і UDP, і блокування може застосовуватись окремо для кожного з цих протоколів. Найпоширенішими прикладами є блокування порту 25 SMTP для боротьби з розсиланням спаму через корпоративні поштові сервери, блокування порту 53 DNS для примусового використання локальних DNS-резолверів провайдера, а також блокування порту 443 HTTPS у деяких корпоративних мережах з метою примусового проксіювання всього зашифрованого трафіку. Окрім простого блокування, на транспортному рівні можуть застосовуватись більш витончені методи, такі як скидання з’єднань за допомогою підроблених RST-пакетів або штучне обмеження пропускної здатності. Скидання з’єднання є особливо ефективним, оскільки змушує клієнта та сервер негайно закрити сесію, причому на боці клієнта це виглядає як помилка з’єднання без явної вказівки на блокування.

Блокування на прикладному рівні L7 є значно складнішим, оскільки передбачає аналіз вмісту даних, якими обмінюються програми. До цієї категорії належать, зокрема, DNS-фільтрація, блокування за SNI та аналіз HTTP-заголовків. DNS-фільтрація полягає в тому, що DNS-сервер провайдера замість справжньої відповіді повертає неправильну IP-адресу. Це може бути адреса застережної сторінки з повідомленням про блокування, відповідь NXDOMAIN, яка повідомляє, що домен не існує, або просто помилкова адреса, що веде на недоступний ресурс. Важливо зазначити, що DNS-підміна діє лише тоді, коли

користувач використовує DNS-сервер, наданий провайдером. Якщо ж користувач налаштовує свій пристрій на використання публічних DNS-серверів, таких як Google або Cloudflare, DNS-підміна стає неможливою, оскільки запити йдуть безпосередньо до цих зовнішніх серверів, які не підпорядковуються місцевій системі блокувань [10]. У такому разі вступають у дію методи інших рівнів - зокрема, блокування за IP-адресою.

Іншим важливим методом блокування на прикладному рівні є аналіз поля SNI, Server Name Indication, у TLS-з'єднаннях. Під час встановлення зашифрованого HTTPS-з'єднання клієнт на початковому етапі рукостискання передає серверу незашифроване поле SNI, яке містить доменне ім'я, до якого він намагається підключитись. Це зроблено для того, щоб сервер міг підібрати правильний TLS-сертифікат у випадку, коли на одній IP-адресі розміщено кілька різних доменів. Оскільки поле SNI передається у відкритому вигляді, будь-який проміжний вузол, включаючи DPI-системи провайдера, може його прочитати та прийняти рішення про блокування. Таким чином, навіть якщо IP-адреса ресурсу не заблокована, а DNS-відповіді не підмінюються, провайдер може обірвати з'єднання ще на стадії TLS-рукопотискання, виявивши заборонене доменне ім'я у полі SNI. [16] Обмеженням цього методу є те, що сучасні технології, такі як зашифрований SNI або його наступник Encrypted Client Hello, дозволяють приховувати це поле, роблячи SNI-фільтрацію неможливою. Однак впровадження цих технологій на рівні всієї екосистеми відбувається повільно, тому цензори часто просто блокують запит якщо бачать ECH поле.

Ще одним методом, що належить до прикладного рівня, є фільтрація HTTP-заголовків у незашифрованому трафіку. На відміну від SNI, який доступний навіть для HTTPS, аналіз HTTP-заголовків потребує доступу до незашифрованих даних. У випадку з HTTPS тіло запиту та всі заголовки, крім поля Host у деяких реалізаціях, є зашифрованими, тому провайдер не може їх проаналізувати без активного втручання у з'єднання. Для незашифрованого HTTP, який нині використовується рідко, система може блокувати запити за URL-патернами, полем Host, User-Agent або Referer. Цей метод дозволяє

обмежувати доступ до конкретних сторінок або API-ендпоінтів без блокування всього домену. Однак, враховуючи повсюдне впровадження HTTPS, практичне значення цього методу поступово знижується, і він все частіше поступається місцем SNI-фільтрації та глибинній інспекції пакетів.

Глибинна інспекція пакетів, DPI, є найбільш потужним, але водночас і найбільш ресурсомістким інструментом фільтрації трафіку. На відміну від звичайного аналізу заголовків, DPI досліджує не лише службову інформацію, але й тіло кожного пакета, що дозволяє виявляти протоколи навіть тоді, коли вони використовують нестандартні порти або застосовують елементарні методи маскування. Існують два основних підходи до DPI: сигнатурний аналіз та поведінковий. Сигнатурний аналіз полягає у пошуку відомих бітових патернів, сигнатур, у вмісті пакетів. Наприклад, рукостискання WireGuard має специфічну структуру, яка дозволяє DPI-системі однозначно ідентифікувати цей протокол навіть при роботі на нестандартному порту. Поведінковий аналіз є більш складним: він враховує статистичні характеристики потоку - розміри пакетів, інтервали між ними, напрямок передачі, патерни рукостискань. Це дозволяє виявляти зашифровані протоколи навіть тоді, коли вони не мають однозначних сигнатур. Наприклад, OpenVPN може бути виявлений за характерною послідовністю обміну пакетами під час встановлення з’єднання, навіть якщо сам трафік зашифрований.

Активне втручання DPI йде далі за пасивний аналіз і дозволяє не лише виявляти заборонений трафік, але й активно втручатися у з’єднання. Найпоширенішим методом є скидання з’єднання шляхом відправки підробленого RST-пакета клієнту або серверу. Оскільки RST-пакет не потребує встановленого з’єднання, DPI-система може згенерувати його самостійно, і обидві сторони закриють сесію, отримавши помилку. Іншим методом є підміна вмісту відповіді, коли DPI перехоплює пакети, що йдуть від сервера до клієнта, і вставляє в них додаткове HTML-повідомлення про блокування.

Окремо слід розглянути російську практику застосування DPI, яка за останні роки набула значного розвитку. Одним з найбільш характерних методів

є так зване «16 КБ блокування». Суть цього методу полягає в тому, що з'єднання технічно встановлюється, перші 16-20 кілобайт даних, приблизно 10-14 пакетів, передаються, але потім трафік обривається або з'єднання примусово скидається. Цього обсягу даних достатньо, щоб сторінка почала завантажуватися в браузері, створюючи ілюзію доступності, але ключові елементи - основний HTML, CSS-стилі, JavaScript-бандли та API-запити - не проходять. У результаті користувач бачить частково завантажену сторінку, яка не функціонує належним чином. Такий підхід часто застосовується до ресурсів, розміщених на великих зарубіжних хостингах та CDN-мережах, таких як Cloudflare, OVH, AWS, Hetzner, DigitalOcean та інших. Водночас для окремих доменів можуть існувати так звані білі списки - переліки адрес, які пропускаються без обмежень [21]. Визначення домену в білому списку відбувається за полем SNI для HTTPS або Host для HTTP, причому маска поширюється на всі піддомени. Це означає, що якщо, наприклад, example.com потрапляє до білого списку, то будь-який піддомен, як-от d1.example.com, також буде пропущений. Важливо зазначити, що механізми 16 КБ-блокування та білі списки можуть суттєво відрізнятися залежно від провайдера та регіону. У деяких випадках будь-який трафік до заблокованого IP-діапазону обривається миттєво, без передачі навіть перших 16 КБ. [22, 5]. Слід також зазначити, що IP-блокування в Росії формально застосовується лише до ресурсів, внесених до офіційних реєстрів заборонених сайтів, однак на практиці фіксуються численні випадки блокування ресурсів, яких немає в жодному реєстрі, що свідчить про незаконне розширення практики фільтрації.

Ще одним прикладом складної системи фільтрації є білі списки, що застосовуються російськими операторами мобільного зв'язку. У цих системах реалізовано комбінацію методів з різних рівнів моделі OSI. На рівні L3 дозволено лише обмежений набір IP-адрес для доступу в інтернет; всі інші адреси заблоковані. На рівні L4 працюють лише порти TCP 80 та 443, а також частково ICMP; UDP-трафік переважно перебуває в блоці. На рівні L7 застосовується SNI-фільтр: система пропускає лише ті TLS-з'єднання, доменне ім'я в полі SNI яких входить до заздалегідь визначеного білого списку сайтів. Таким чином,

користувач може отримати доступ лише до кількох десятків або сотень заздалегідь схвалених ресурсів, а спроби підключитися до будь-якого іншого сайту блокуються на рівні провайдера навіть до того, як буде встановлено з'єднання. Така багаторівнева система призначена для тотального контролю над інтернет-активністю в мобільних мережах і залишає дуже мало простору для обходу обмежень традиційними методами.

Практичний приклад багаторівневої системи блокування можна розглянути на матеріалі України, де застосовується каскад DNS-фільтрації та L3-блокування. Коли користувач використовує DNS-сервер, наданий його інтернет-провайдером, запит до заблокованого ресурсу може отримати відповідь з CNAME-записом, що вказує на спеціальний заблокований домен, наприклад blocked.domain. Браузер отримує IP-адресу цього домену і замість очікуваного сайту бачить сторінку з повідомленням про блокування. Однак, якщо користувач змінює налаштування DNS на своєму пристрої, перемикаючись на публічний резолвер, DNS-підміна перестає працювати. У цьому випадку запит йде безпосередньо до серверів Google або Cloudflare, які повертають справжню IP-адресу ресурсу. Наступним рівнем захисту є блокування на L3: провайдер вносить IP-адресу відповідного ресурсу до чорного списку, і будь-яка спроба доступу до цієї адреси, незалежно від того, звідки прийшов запит, блокується на маршрутизаторі. Такий підхід демонструє комбінацію методів L7 у вигляді DNS-підміни та L3 у вигляді IP-блокування. Перший є дешевшим і менш ресурсомістким, але легко обходиться зміною DNS. Другий потребує внесення IP-адрес до чорного списку, що може призвести до надмірного блокування, але зате є ефективним незалежно від налаштувань DNS на боці клієнта.

Розглянемо практичне дослідження блокувань на прикладі українського провайдера Volia, автономна система AS25229. Для аналізу було обрано декілька вебсайтів: uz.gov.ua як незаблокований контрольний ресурс, warhammer-guide.ru, ya.ru та opennet.ru як заблоковані сайти з російською IP-адресацією, а також habr.com як заблокований ресурс, що використовує anycast-адресацію. При використанні стандартного DNS-сервера провайдера спостерігаються різні типи

поведінки, як показано на (Рис. 1.1). Для warhammer-guide.ru DNS-запис взагалі не видається, хоча сайт функціонує та інші DNS-резолвери повертають правильну відповідь. Для ya.ru, як і для інших заборонених сайтів, зазначених в указі президента України №184/2020 [1], відбувається перенаправлення на заблокований домен blocked.domain через запис CNAME, який веде на IP-адресу 77.120.62.10, де розміщено застережну сторінку про блокування ресурсу. Для доменів opennet.ru та habr.com DNS-сервер провайдера все ж таки повертає коректні записи, так само як і для uz.gov.ua. Якщо ж змінити DNS-сервер на публічний, всі відповіді стають коректними, що підтверджує вибірковий характер DNS-фільтрації на рівні провайдера.

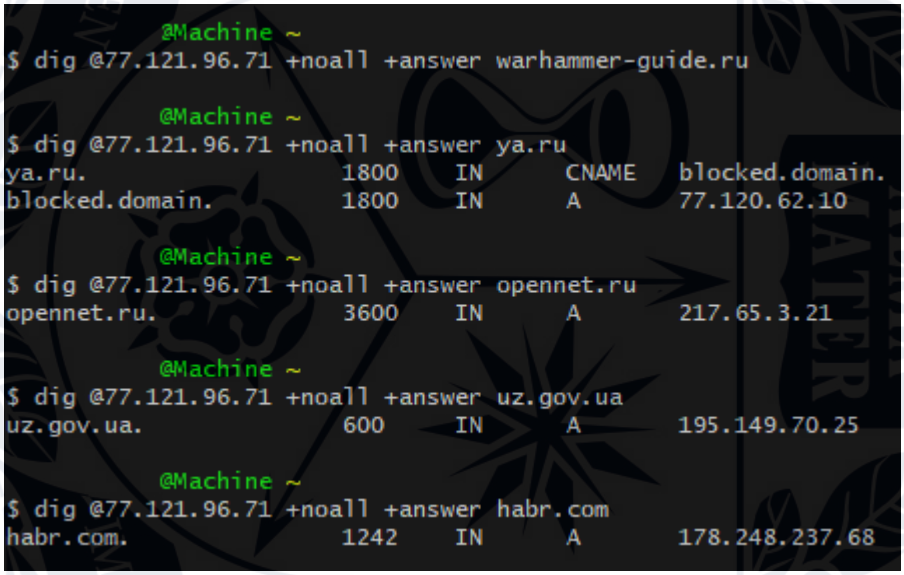

Рисунок 1.1 - Виведення утиліти dig для DNS-запитів на провайдерський сервер

Для подальшого аналізу було виконано трасування маршруту за допомогою інструменту Zenmap з метою визначити, на якому саме етапі відбувається скидання пакетів. Результати трасування, наведені на (Рис. 1.2), показали, що більшість заблокованих ресурсів навіть не залишають локальної мережі провайдера - блокування відбувається на останньому хопі перед виходом у магістральну мережу. Це свідчить про те, що фільтрація реалізована безпосередньо на обладнанні провайдера, а не на вищестоящих маршрутизаторах. На цьому фоні виділяється habr.com, який має anycast-адресацію: трафік до цього ресурсу було спрямовано через магістрального

провайдера RETN до Нідерландів, де, ймовірно, розташований один з вузлів anycast-мережі. Це ускладнює блокування такого ресурсу традиційними методами.

Рисунок 1.2 - Трасування маршруту на задані домени

Варто зазначити, що метою даної роботи не є створення засобів для обходу українських систем блокування, які спрямовані на протидію пропаганді та захист інформаційної безпеки держави. Дослідження зосереджується на розробці інструментарію для забезпечення доступу до відкритої інформації в умовах

політичної цензури, зокрема в країнах, де доступ до світового інформаційного простору штучно обмежується авторитарними режимами. Наведений вище приклад слугує лише ілюстрацією звичайних методів блокування у більшості країн, які необхідно розуміти для проєктування ефективних засобів протидії цензурі, а не для порушення чинного законодавства.

Таким чином, сучасні системи мережевих обмежень рідко покладаються лише на один метод фільтрації. Типовий каскад виглядає наступним чином: на першому етапі застосовується DNS-підміна, яка є швидкою, дешевою і легко масштабується на будь-яку кількість доменів. Якщо DNS-підміна не спрацювала, бо користувач змінив налаштування DNS або використовує DNS over HTTPS чи DNS over TLS, вступає в дію блокування за IP-адресами або підмережами на рівні L3. Цей метод є більш грубим, але ефективним незалежно від дій користувача. Нарешті, для особливо важливих ресурсів або у випадках, коли L3-блокування неможливе, наприклад при використанні спільних хостингів або CDN, застосовується глибинна інспекція пакетів з аналізом SNI, сигнатур протоколів та активним втручанням у з'єднання. Розуміння цієї багаторівневості є необхідною умовою для проєктування ефективних засобів обходу обмежень, оскільки обхід одного методу, наприклад зміна DNS, не гарантує доступу до ресурсу, якщо інші методи продовжують діяти.

1.3 Порівняльний огляд протоколів обходу блокувань та засобів приховування трафіку

Розвиток засобів обходу мережевих обмежень є відображенням постійного змагання між цензорами та тими, хто намагається забезпечити вільний доступ до інформації. Кожна нова технологія блокування стимулює появу більш досконалих методів маскування, і навпаки - вдосконалення обхідних технологій змушує цензорів ускладнювати системи аналізу трафіку. Від найпростіших проксі-серверів до багатошарових систем обфускації - кожен наступний рівень технологічного розвитку підвищує стійкість до детектування, але водночас збільшує вимоги до обчислювальних ресурсів та складності налаштування.

Розуміння принципів роботи та вразливостей кожного з цих інструментів є необхідною умовою для побудови ефективної системи забезпечення доступності веб-ресурсів в умовах сучасної цензури.

Найпростішими засобами обходу блокувань є проксі-протоколи HTTP та SOCKS5. HTTP-проксі використовує метод CONNECT для встановлення тунелю через TCP-з'єднання, що дозволяє клієнту звертатися до зовнішніх ресурсів через проміжний сервер. Однак цей протокол не передбачає жодного шифрування даних, що робить його повністю прозорим для DPI. SOCKS5 є більш універсальним, підтримує TCP та UDP, але також не забезпечує шифрування трафіку. Обидва протоколи легко виявляються за характерними сигнатурами в заголовках з'єднання, тому в умовах сучасної цензури вони можуть використовуватись лише як проміжний шар у складніших схемах.

Shadowsocks є одним з перших зашифрованих проксі, орієнтованих на обхід цензури. Він базується на архітектурі SOCKS5, але додає шар шифрування з використанням таких алгоритмів, як AES-256-GCM, ChaCha20-Poly1305 або XChaCha20-Poly1305, що робить його непридатним для простого аналізу вмісту пакетів [24]. Рукостискання та прикладні дані шифруються, що значно ускладнює ідентифікацію. Тим не менш, певні сигнатури, зокрема нестандартний розмір пакетів, дозволяють деяким DPI-системам виявляти Shadowsocks в умовах високого рівня спостереження. Найсучасніша версія, Shadowsocks-2022, через свою легкість та низьке споживання ресурсів залишається популярною.

Окремо варто згадати протокол MTProto, вбудований у Telegram. Його проксі-сервер MTProxy маскує трафік під звичайний HTTPS, що ускладнює роботу DPI. Однак через власні криптографічні особливості MTProto міг бути виявлений за нестандартним 32-байтовим публічним ключем X25519 та застарілими ідентифікаторами у рукостисканні. Цей протокол показав себе як ненадійний спосіб захисту від DPI, особливо після масштабних блокувань 1 квітня 2026 року, коли вбудований механізм FakeTLS у багатьох користувачів перестав працювати.

Традиційні VPN-протоколи OpenVPN та WireGuard без додаткової обфускації є дуже вразливими до сигнатурного аналізу. OpenVPN використовує стандартний набір криптографічних алгоритмів та підтримує роботу через TCP і UDP, але його пакети мають добре впізнавану структуру: вони починаються з явного байта операції, за яким слідує ключовий індекс. WireGuard є сучаснішим і швидшим протоколом, який використовує мінімалістичний набір криптографічних примітивів. Його рукостискання завжди починається з фіксованої послідовності з чотирьох байтів 0x01000000, що відповідає повідомленню типу INITIATION [31]. Ця фіксована сигнатура робить WireGuard тривіальною мішенню для сигнатурного аналізу. Без додаткових шарів маскування ці протоколи є повністю прозорими перед сучасними DPI.

Утиліти локальної модифікації трафіку, такі як GoodbyeDPI та Zapret, працюють без віддалених серверів, модифікуючи пакети прямо на комп'ютері користувача. GoodbyeDPI, розроблений переважно для Windows, використовує бібліотеку WinDivert для перехоплення та модифікації пакетів на рівні драйвера. Його основні методи включають фрагментацію TCP-пакетів для того, щоб DPI-система не могла скласти цілісну картину та побачити заборонений хост, а також модифікацію заголовків HTTP та HTTPS. Zapret, написаний на C, є більш гнучким рішенням, підтримує Windows та Linux, а також роботу на роутерах. Він використовує автоматичне визначення параметрів DPI через блок-чекінг та має потужну систему скриптів для гнучкого налаштування фільтрації. Існує також Zapret2, де оригінальні стратегії на C замінені більш гнучкими Lua-скриптами, що дозволяє швидко адаптуватися до нових методів детектування [35].

AmneziaWG є форком WireGuard, який зберігає його високу продуктивність, але додає можливість обфускації на транспортному рівні. Основна ідея полягає в тому, що стандартний WireGuard має фіксовану структуру пакетів, яка легко виявляється. AmneziaWG вирішує цю проблему, дозволяючи налаштовувати параметри обфускації, зокрема замінювати заголовки пакетів, додавати паддинг та сміттєві пакети. У версії 2.0 протокол надає механізм мімікрії: у поля I1-I5, які є частиною пакетів WireGuard, можна

записувати будь-які довільні дані, що дозволяє маскувати трафік під QUIC або інші UDP-протоколи [4].

Hysteria2 є протоколом, побудованим на основі QUIC. QUIC - це транспортний протокол, розроблений Google та стандартизований у 2021 році, який працює поверх UDP. Усередині він реалізує мультиплексування потоків, контроль перевантаження, надійну доставку та шифрування. HTTP/3, у свою чергу, є HTTP поверх QUIC. Завдяки цьому Hysteria2 маскує свої з'єднання під звичайний HTTP/3 трафік. Протокол застосовує алгоритм контролю перевантажень Brutal. На відміну від стандартних алгоритмів, які зменшують швидкість при втраті пакетів, Brutal компенсує втрати агресивним збільшенням швидкості, що дозволяє ефективно боротися з штучним обмеженням пропускної здатності. Однак саме це є його слабкою стороною, оскільки така поведінка є нехарактерною для стандартного QUIC, що може створювати підозрілу сигнатуру. Проте варто зазначити, що Hysteria2 також підтримує стандартні алгоритми контролю перевантажень, зокрема BBR, що може бути корисним у сценаріях, де агресивна поведінка Brutal є небажаною [15].

Slipstream є технологією туннелювання, яка передає дані через DNS-запити та відповіді з використанням QUIC як транспортного протоколу. DNS-тунелювання дозволяє встановити з'єднання в будь-якій мережі, де є доступ до DNS-резолвера, навіть якщо клієнт не має прямого доступу до інтернету. Традиційні DNS-тунелі, такі як Iodine або OzymanDNS, мають обмеження, зумовлені природою DNS-протоколу: обмежений максимальний розмір повідомлення, модель «запит-відповідь» та обмеження частоти запитів з боку резолверів. Slipstream вирішує ці проблеми шляхом реалізації сумісності з QUIC, що дозволяє використовувати гарантії надійності QUIC та створювати високопродуктивний DNS-тунель. Він використовує адаптивний контроль перевантаження для роботи з обмеженими за частотою резолверами та підтримує QUIC multipath, що дозволяє об'єднувати пропускну здатність кількох резолверів для значного підвищення продуктивності [26, 9].

Xray-core є потужною платформою з відкритим кодом, яка виступає наступником V2Ray. Вона підтримує безліч протоколів, включаючи власні стандарти VMess та VLESS, а також Shadowsocks, SOCKS, HTTP та інші. VLESS є легким протоколом без шифрування, що покладається на зовнішнє транспортне маскування. Xray-core дозволяє застосовувати різноманітні транспорти, зокрема TCP, WebSocket, gRPC, QUIC. Особливу увагу заслуговує технологія REALITY. На відміну від традиційних TLS-проксі, REALITY не вимагає наявності власного сертифіката. Замість цього сервер REALITY приймає всі вхідні TLS-з'єднання. Якщо клієнт надає правильний пароль, встановлюється проксі-з'єднання. Якщо ні - сервер перенаправляє з'єднання на реальний зовнішній вебсайт, ім'я якого вказане в конфігурації у вигляді SNI. Коли DPI активно зондує порт сервера, Xray повертає йому справжню TLS-відповідь цільового сайту. Цензор, отримуючи легітимну TLS-відповідь, не може підтвердити факт роботи проксі [32].

Однак дослідження показують уразливі місця цієї технології. По-перше, невідповідність між IP-адресою сервера та доменом, що маскується, може викликати підозру. По-друге, якщо реальний цільовий сайт знаходиться у білому списку цензора, DPI-система може спробувати виконати звірку часових міток і побачити, що запити на сервер REALITY та на справжній сайт надходять одночасно з однієї IP-адреси, що демаскує проксі, хоча на практиці таких випадків ще не зафіксовано.

Сучасними альтернативами Xray-core є sing-box та Mihomo, усі можуть здатися схожими на перший погляд, але мають різні філософії. Sing-box робить ставку на продуктивність та сучасну архітектуру. Він підтримує більшість сучасних протоколів, включаючи VLESS з REALITY, Hysteria2, Shadowsocks, WireGuard та інші. Однією з ключових особливостей sing-box є його орієнтація на NaiveProxy. Недоліком вважається відсутність деяких функцій, доступних у Xray-core. Документація sing-box є достатньою, однак деякі аспекти залишаються недостатньо висвітленими на мій погляд. Mihomo, раніше відомий як Clash.Meta, є універсальним ядром, який підтримує всі сучасні протоколи, включаючи REALITY, Hysteria2 та деякі унікальні, такі як AmneziaWG. Його

основною перевагою є зрозуміла YAML-конфігурація, успадкована від Clash. Mihomo та sing-box споживають менше ресурсів порівняно з Xray-core. Що стосується маршрутизації, то гнучкий розподіл трафіку за доменами, гео-IP, портами або типом протоколу підтримується всіма сучасними ядрами [25, 6].

На основі проведеного порівняльного аналізу для подальшої реалізації проксі-сервісу було обрано комбінацію технологій, яка забезпечує оптимальний баланс між стійкістю до DPI, продуктивністю та гнучкістю. Ядром системи виступає Xray-core - платформа, яка завдяки механізму REALITY ефективно протидіє активному зондуванню цензорів та підтримує широкий спектр протоколів для маршрутизації трафіку. Для забезпечення високої пропускної здатності та маскування під HTTP/3 інтегровано Hysteria2, який також дозволяє боротися з штучним обмеженням швидкості. Як транспортний протокол з обфускацією обрано AmneziaWG - форк WireGuard, що надає можливість приховувати VPN-трафік під довільні UDP-патерни, зокрема імітувати QUIC-з’єднання. Додатково, для роботи в мережах з обмеженим прямим доступом до інтернету, передбачено використання Slipstream - технології тунелювання трафіку через DNS-запити поверх QUIC.

1.4 Формування вимог до функціональності та безпеки проксі-сервісу

На основі проведеного аналізу сучасних методів мережевих обмежень та порівняльного огляду протоколів обходу блокувань було визначено чотири ключові технології для побудови проксі-сервісу: Xray-core, Hysteria2, AmneziaWG та Slipstream. Кожен з цих протоколів має власні особливості конфігурації та механізми генерації клієнтських ключів. Для створення автоматизованого сервісу, який поєднує ці протоколи в єдине середовище, необхідно сформулювати чіткі вимоги до функціональності, безпеки та архітектури системи.

Архітектура системи складається з трьох основних компонентів. Головний сервер виконує функції веб-сервера, бази даних та бекенду для обробки платежів. Окремі проксі-сервери забезпечують роботу кожного з обраних протоколів.

Канал зв’язку між головним сервером та проксі-серверами використовується для додавання нових клієнтів, видалення ключів з терміном, що минув, а також моніторингу стану сервісів.

Веб-ресурс має бути реалізований як статично згенерований сайт з мінімальним використанням генератора Hugo. Це забезпечує мінімальне споживання ресурсів, високу швидкість завантаження та відсутність залежностей від серверних мов програмування. Використання JavaScript зведено до мінімуму - лише для генерації UUID, відправки запитів до API та відображення конфігурацій. Сайт не зберігає сесії та не передбачає традиційної авторизації. Ідентифікація клієнта відбувається виключно за допомогою UUID, який генерується в браузері та зберігається у локальному сховищі.

Функціональні вимоги до веб-ресурсу включають можливість перегляду списку доступних проксі-послуг, їх вартості в Monero та терміну дії. Клієнт має можливість поповнити баланс для свого UUID. При цьому система генерує одноразову адресу Monero, прив’язану до цього UUID. Після підтвердження транзакції клієнт може придбати ключ доступу до обраного протоколу. Після успішної покупки система надає клієнту згенеровану конфігурацію у вигляді текстового блоку.

Бекенд системи реалізує фоновий процес для моніторингу надходжень на згенеровані XMR-адреси. Після підтвердження транзакції баланс відповідного UUID у локальній базі даних оновлюється. Логіка покупки включає перевірку достатності коштів, списання суми, генерацію конфігурації для відповідного проксі-сервера та відправку команди на додавання клієнта. Кошти списуються з балансу лише після успішного додавання ключа на проксі-сервері.

Система повинна мати механізм щоденної перевірки термінів дії підписок. Ключі з терміном, що минув, видаляються з локальної бази даних та з конфігурацій проксі-серверів. Такий підхід забезпечує автоматичне очищення системи без ручного втручання.

Вимоги безпеки є критичними для системи. Система не зберігає персональних даних клієнтів - єдиним ідентифікатором є UUID. Усі платежі

виконуються в Monero - криптовалюті з вбудованими механізмами приховування транзакцій. Комунікація між головним сервером та проксі-серверами обов’язково шифрується. Доступ до API бекенду обмежений та автентифікований. Усі вхідні параметри проходять сувору валідацію.

Кожен з обраних протоколів має специфічні вимоги до конфігурації. Для Xray-core використовується схема steal-oneself з fallback на сайт-заглушку. При генерації ключа створюються унікальний ідентифікатор, налаштування транспорту, параметри fallback та SNI цільового сайту. Для Hysteria2 генеруються аутентифікаційні дані у вигляді пароля, адреса сервера та параметри контролю перевантажень. Для AmneziaWG генеруються унікальні ключі з можливістю налаштування параметрів. Для Slipstream необхідно налаштувати адресу DNS-сервера та параметри QUIC. Усі згенеровані конфігурації мають бути представлені у форматі, готовому до імпорту у відповідні клієнтські програми.

Сформульовані вимоги визначають архітектуру системи, яка складається зі статичного веб-сайту, автоматизованого бекенду для обробки платежів та генерації ключів, а також набору проксі-серверів з обраними протоколами маскування. У наступних розділах на основі цих вимог буде розроблено конфігураційні файли, скрипти автоматизації та веб-ресурс, що реалізує описану логіку взаємодії з користувачем.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ ДОСТУПУ В УМОВАХ ОБМЕЖЕНЬ

2.1. Модель архітектури взаємодії проксі-вузлів

Розроблена система є розподіленим рішенням для автоматизованого надання доступу до проксі-серверів, які реалізують протоколи обходу мережевих обмежень. Архітектура побудована за принципом master-slave з активним керуванням, що передбачає безпосереднє ініціювання змін з боку центрального сервера на периферійних вузлах. Такий підхід мінімізує затримку активації підписок та дозволяє уникнути необхідності зберігання стану на проксі-вузлах. Загальна топологія системи включає три типи компонентів: статичний веб-сайт як точку входу для клієнта, центральний master-сервер, що реалізує логіку білінгу та координації, а також один або декілька проксі-вузлів, кожен з яких забезпечує безпосереднє виконання проксі-протоколів AmneziaWG, Xray-core, Hysteria2 та Slipstream. Схему взаємодії компонентів наведено на (Рис. 2.1).

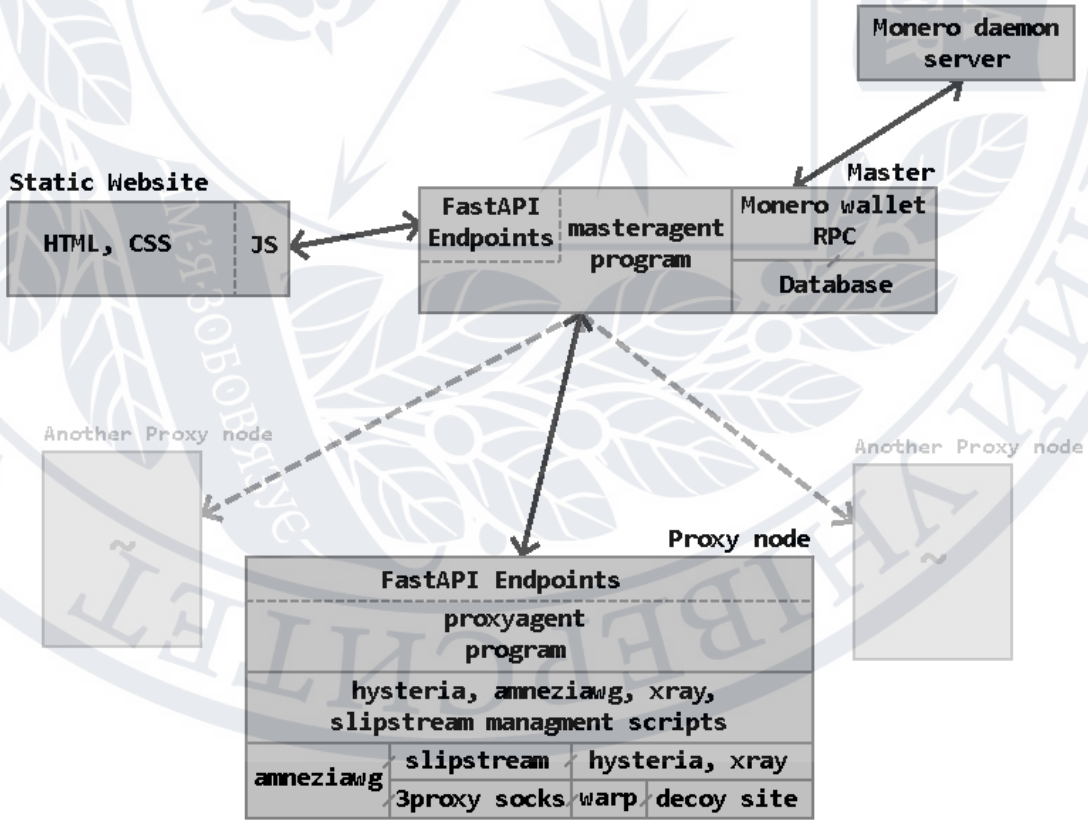

Рисунок 2.1 - Схема взаємодії компонентів проекту

Статичний веб-сайт реалізовано з використанням генератора Hugo [14]. Він є повністю stateless - не зберігає сесії, не використовує cookies та не потребує серверних мов програмування. Уся клієнтська логіка, включаючи генерацію ідентифікатора UUID, надсилання HTTPS-запитів до master-сервера та відображення отриманих конфігурацій, реалізована на JavaScript з мінімальним обсягом коду. Ідентифікатор UUID генерується в браузері та зберігається у локальному сховищі клієнта; він є єдиним засобом ідентифікації клієнта. Усі запити до master надсилаються із заголовком, що містить цей UUID. Такий підхід забезпечує повну анонімність, оскільки сервер не отримує жодної додаткової інформації про користувача.

Master-сервер є центральним координатором системи. Він реалізований на фреймворку FastAPI [8] і виконує наступні функції: приймання запитів від фронтенду для отримання балансу, інформації про підписки та генерації депозитних адрес; взаємодію з monero-wallet-rpc для створення адрес та моніторингу платежів; керування локальною базою даних SQLite; а також надсилання команд на проксі-вузли через HTTPS. Master запускається як systemd-сервіс і містить вбудований планувальник фонових завдань для періодичного опитування мережі Monero, щоденного списання коштів за підписки та очищення застарілих депозитних адрес.

Проксі-вузли є легковагими FastAPI-агентами, які працюють на окремих серверах, де безпосередньо розгорнуті проксі-протоколи. Кожен агент приймає команди від master у вигляді HTTPS-запитів та виконує відповідні дії через виклик попередньо написаних bash-скриптів. До таких команд належать додавання користувача на всі протоколи одночасно, видалення одного або декількох користувачів, отримання списку користувачів для певного протоколу та отримання клієнтської конфігурації у вигляді текстового блоку. Агенти не мають власної бази даних і не зберігають жодного стану - вся інформація про підписки знаходиться виключно на master. Для захисту від несанкціонованого доступу API агентів захищений статичним ключем, а доступ до них обмежений лише з IP-адреси master-сервера.

Модель білінгу базується на депозитному принципі з щомісячним автоматичним списанням. Клієнт ініціює поповнення балансу, вказуючи бажану суму; master генерує одноразову суб-адресу Monero, прив’язану до UUID клієнта. Клієнт надсилає будь-яку суму на цю адресу. Фоновий процес master періодично опитує monero-wallet-rpc [17], виявляє нові транзакції з достатньою кількістю підтверджень та збільшує баланс відповідного клієнта. Якщо після зарахування баланс клієнта досягає або перевищує місячну вартість підписки і при цьому у нього немає активної підписки, master автоматично ініціює активацію. Він випадковим чином обирає один із доступних проксі-вузлів, надсилає команду на додавання користувача з усіма чотирма протоколами, отримує згенеровані конфігурації, списує вартість підписки з балансу та створює запис про активну підписку. Щодня опівночі master перевіряє активні підписки з терміном, що минув, та або списує черговий платіж, якщо баланс достатній, або деактивує підписку та надсилає команді на відповідний вузол видалити користувача.

Уся комунікація між компонентами системи виконується через HTTPS. Між фронтендом та master використовується HTTPS із заголовком X-User-UUID для ідентифікації клієнта. Між master та проксі-вузлами використовується HTTPS зі статичним API-ключем у заголовку. Взаємодія master з monero-wallet-rpc здійснюється через JSON-RPC з прив’язкою до локального інтерфейсу та використанням файла пароля.

Масштабованість системи досягається простотою додавання нових проксі-вузлів - достатньо внести новий запис до конфігурації master, після чого всі нові підписки автоматично розподіляються між вузлами випадковим чином. У разі недоступності одного з вузлів під час активації master виконує повторні спроби з вибором іншого вузла. Така архітектура дозволяє горизонтально масштабувати систему шляхом додавання нових вузлів без змін у логіці master або клієнтській частині. Типовий потік даних від моменту внесення платежу до активації підписки ілюструє рис. 2.2 та рис. 2.3.

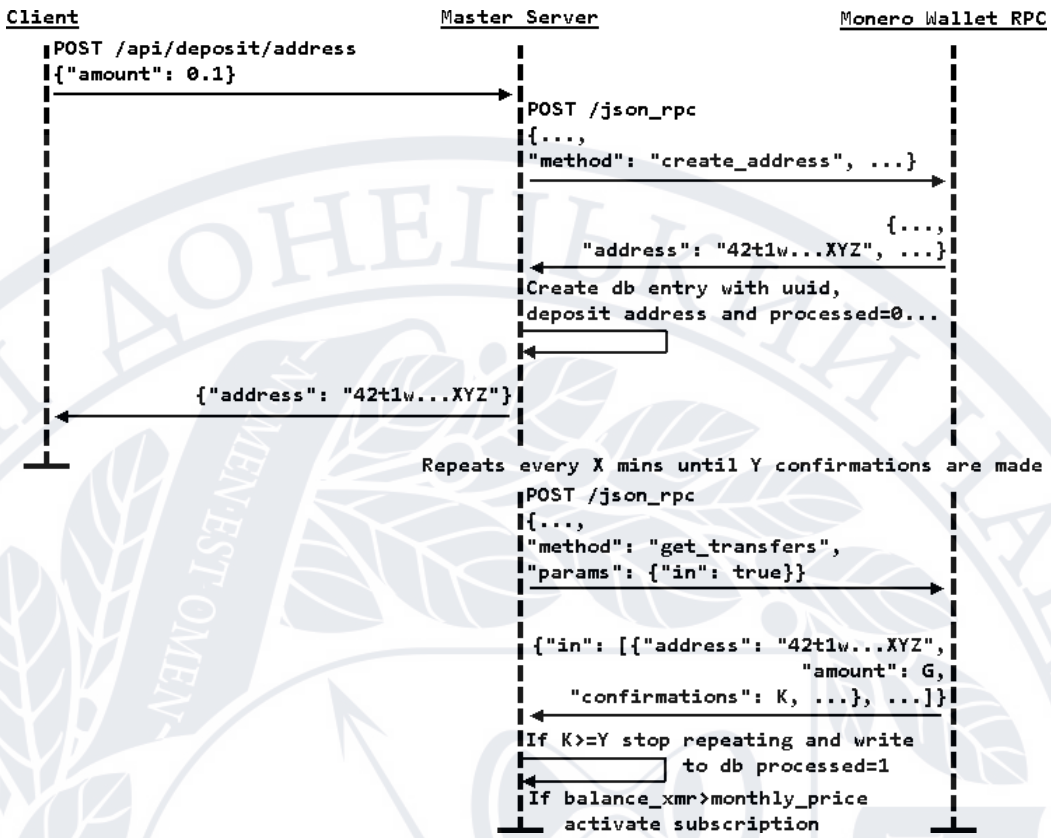

Рисунок 2.2 - Спрощена схема запита адреси поповнення від кліента

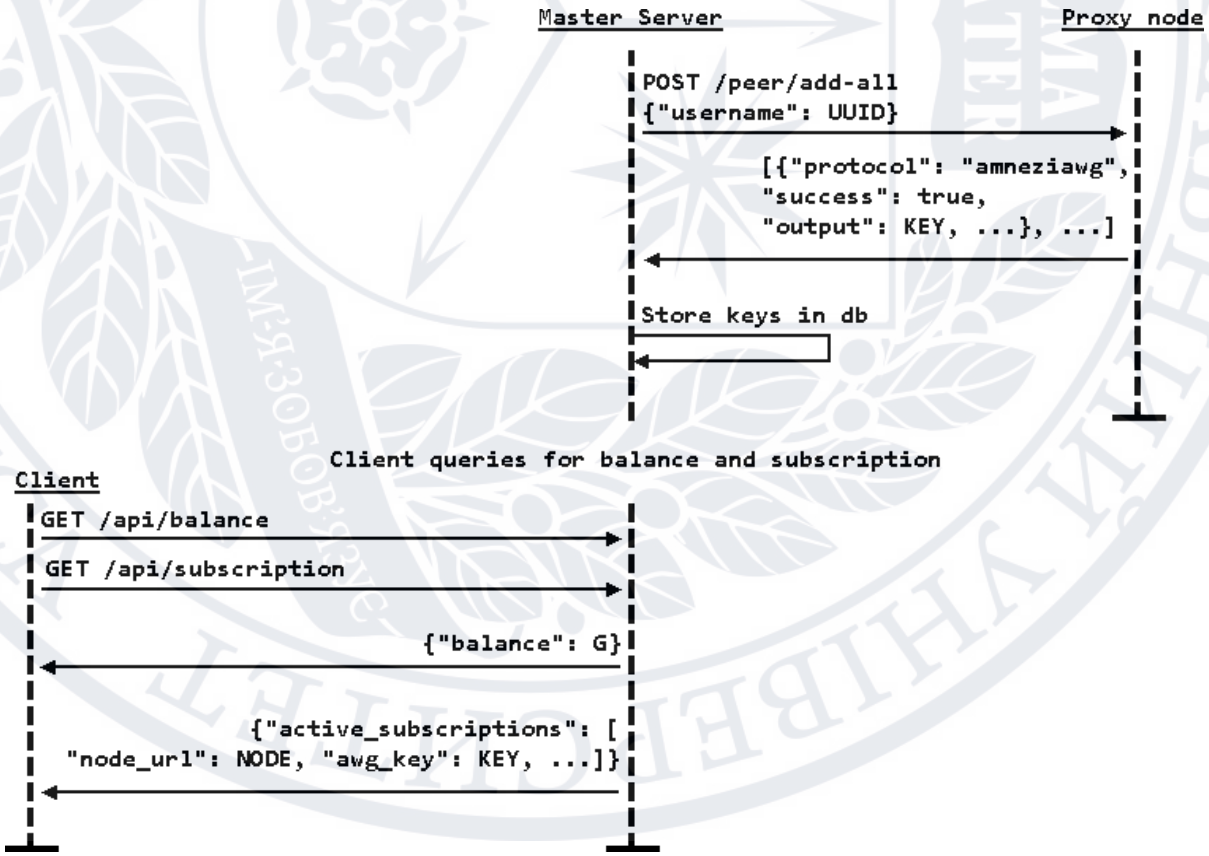

Рисунок 2.3 - В продовження рис. 2.2: додавання ключів, запит балансу та підписки

2.2 Модель маршрутизації трафіку та логіка тунелювання

У розробленій системі маршрутизація мережевого трафіку реалізована на рівні кожного окремого протоколу з використанням специфічних для них механізмів. Архітектура системи побудована таким чином, щоб забезпечити як стійкість до систем глибокого аналізу пакетів, так і ефективне перенаправлення користувацьких даних. Кожен з інтегрованих протоколів - Xray-core, Hysteria2, AmneziaWG та Slipstream - має власну логіку обробки та спрямування трафіку, яка детально розглядається нижче.

Для Xray-core, який є одним з ключових компонентів системи, маршрутизація конфігурується через основний файл налаштувань, що визначає, як саме обробляються вхідні з'єднання. Основним елементом є набір вихідних напрямків, яких у типовій конфігурації три. Перший - це прямий вихід у мережу, який використовує системний мережевий інтерфейс без додаткової обробки. Другий - це вихід через інтерфейс Cloudflare WARP, створений за допомогою AmneziaWG у режимі сумісності з Wireguard [4], що дозволяє динамічно змінювати IP-адресу, з якої здійснюються запити. Третій - це чорна діра для трафіку, реалізована як «blackhole», яка просто скидає будь-який трафік, що потрапляє на неї, що фактично означає блокування.

Правила маршрутизації в Xray-core застосовуються послідовно, зверху вниз, і при збігу з першим релевантним правилом подальша обробка припиняється. Деякі довірені сервіси, наприклад визначені в наборах geosite:telegram, geosite:discord або geosite:youtube, скеровуються безпосередньо через прямий outbound, оминаючи додаткове тунелювання до WARP. Якщо IP-адреса призначення потрапляє до гео-списків geoip:ru, geoip:cn або geoip:ir, таке з'єднання блокується. Всі інші запити за замовчуванням передаються через інтерфейс WARP.

Важливо зазначити, що блокування IP-адрес певних географічних регіонів та перенаправлення через WARP зроблено з міркувань безпеки. Російські ТСПУ та деякі шпигунські програми зіставляють вихідні адреси через кілька джерел і передають результати до відповідних органів. Перенаправляючи весь трафік,

окрім довірених сервісів, через WARP, ми зменшуємо ризик для цільового ресурсу проксі, уникаючи витоку його вихідної адреси. Клієнт, у свою чергу, може налаштувати власний маршрут для отримання доступу до цих ресурсів.

Механізм fallback у Xray-core, реалізований на рівні вхідного з'єднання, обробляє вхідний трафік на порту 443. Якщо з'єднання є легітимним трафіком протоколу VLESS, воно обробляється Xray для подальшого проксування. Якщо ж це звичайний HTTPS-запит, наприклад від браузера, або зондувальний запит DPI, Xray перенаправляє його на внутрішні сервери, налаштовані на додаткових портах [32]. Це робить проксі-сервер невидимим для пасивного аналізу, оскільки він завжди відповідає як звичайний веб-сервер. Схема на Рис. 2.4 описує цей процес. Додатково, функція sniffing дозволяє Xray визначати справжнє призначення зашифрованого трафіку, що необхідно для роботи правил маршрутизації.

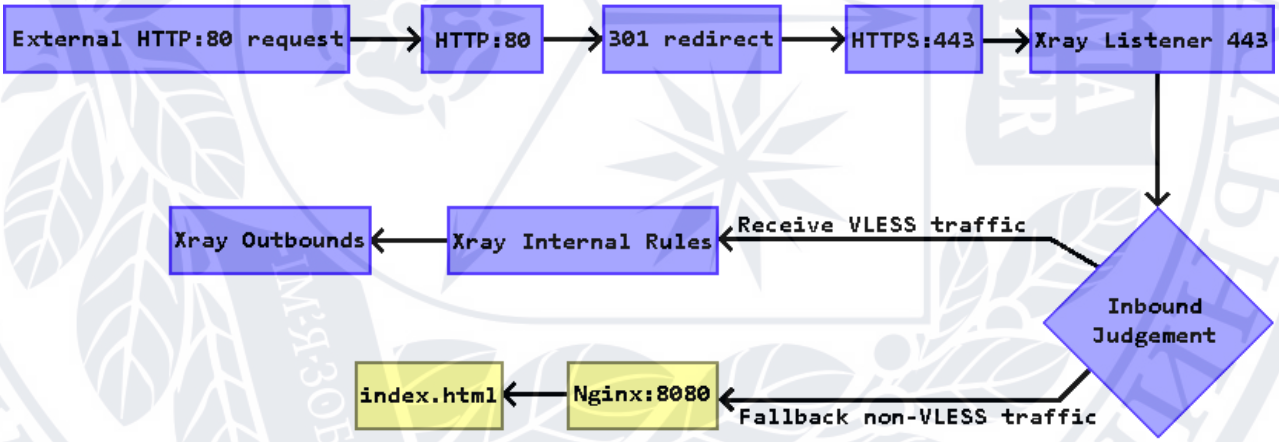

Рисунок 2.4 - Механізм fallback у Xray

Для протоколу Hysteria2 застосовуються правила маршрутизації, подібні до Xray-core: деякі довірені сервіси спрямовуються безпосередньо, IP-адреси з регіонів ru, cn та ir блокуються, а весь інший трафік передається через інтерфейс WARP. Перенаправлення відбувається на локальний сервер. В результаті сторінка виглядає так, ніби вона підтримує версії HTTP/1.1 та HTTP/2 від Xray, а також HTTP/3 від Hysteria2. Для маскування Hysteria2 використовує механізм

masquerade, аналогічний за функцією до fallback. Коли до сервера звертається стандартний браузер або інструмент активного зондування, сервер відповідає заздалегідь заготовленим вмістом або переспрямовує запит, що забезпечує невідрізненність від звичайного HTTP/3 трафіку.

AmneziaWG, на відміну від Xray та Hysteria2, не має вбудованих механізмів гнучкої маршрутизації на рівні програми. Однак він надає набір параметрів для обфускації трафіку. Параметри Jc, Jmin та Jmax визначають кількість та розмір сміттєвих пакетів, що надсилаються перед кожним рукостисканням. Крім того, параметри I1-I5 дозволяють додавати сигнатурні пакети для імітації популярних протоколів. За допомогою спеціальної можна мови опису задати формат пакету таким чином, щоб система аналізу трафіку сприймала його як легальний трафік протоколів QUIC, DNS, WebRTC тощо. Клієнт може самостійно налаштовувати ці параметри для досягнення потрібного рівня маскування [4].

Slipstream, який використовується в системі, є не повноцінним проксі, а тільки DNS-тунелем. Він інкапсулює клієнтський трафік у DNS-запити та передає його через систему доменних імен [9, 26]. Це дозволяє обходити блокування, що застосовуються на рівні TCP/IP, оскільки DNS-трафік рідше піддається глибокому аналізу. Оскільки DNS-тунель сам по собі не є проксі-сервером, у системі його виведення налаштоване на взаємодію з легковагим SOCKS5-сервером 3proxy. Така конфігурація забезпечує кілька переваг. По-перше, Slipstream виступає як транспорт, що долає обмеження, а 3proxy надає універсальний інтерфейс для підключення. По-друге, 3proxy дозволяє налаштувати автентифікацію за логіном та паролем, що додає базовий рівень безпеки. Таким чином, будь-який застосунок, що підтримує SOCKS5, може використовувати цей тунель без жодної додаткової модифікації.

2.3. Проєктування архітектури внутрішнього API та бази даних для автоматизації керування доступом

Для забезпечення автоматизованої взаємодії між компонентами системи було розроблено внутрішній API, що з'єднує статичний веб-сайт, центральний master-сервер та проксі-вузли. API побудовано на основі архітектурного стилю REST з використанням протоколу HTTPS для всіх типів взаємодії. Виділено три категорії таких взаємодій: між фронтендом та master, між master та проксі-вузлами, а також між master та monero-wallet-rpc. Кожна з них має власну специфіку автентифікації та форматів даних, що відображає різні ролі компонентів у системі.

Взаємодія між статичним веб-сайтом та master-сервером є найбільш навантаженою, оскільки обслуговує всі запити клієнтів. Фронтенд надсилає HTTPS-запити з ідентифікатором клієнта у заголовку X-User-UUID, який містить унікальний ідентифікатор, згенерований у браузері та збережений у локальному сховищі. Першим ендпоінтом є отримання балансу за методом GET на адресу /api/balance. Master виконує запит до бази даних для пошуку запису з відповідним UUID та повертає клієнту поточний баланс у Monero у форматі числа з плаваючою комою. Якщо клієнт звертається вперше, запис у таблиці accounts автоматично створюється з нульовим балансом. Наступним ендпоінтом є отримання інформації про активну підписку за адресою /api/subscription. Master перевіряє наявність активного запису в таблиці subscriptions для даного UUID та повертає у відповіді адресу проксі-вузла, а також конфігурації для всіх чотирьох протоколів - AmneziaWG, Xray-core, Hysteria2 та Slipstream. Якщо активної підписки не існує, повертається порожній масив. Ендпоінт генерації депозитної адреси реалізовано методом POST на адресу /api/deposit/address з опціональним полем amount у тілі запиту, яке використовується лише як підказка для клієнта. Master викликає метод create_address monero-wallet-rpc, отримує нову суб-адресу та зберігає її разом з UUID клієнта у таблиці deposit_addresses з прапорцем processed = 0. Після цього адреса повертається клієнту для виконання переказу.

Для оновлення ключів підписки передбачено службовий ендпоінт POST /api/subscription/refresh, який не призначений для безпосереднього використання клієнтами, але доступний на випадок аварійних ситуацій, коли клієнт втратив збережені конфігурації або на проксі-вузлі відбулися зміни. Він звертається до відповідного проксі-вузла через ендпоінт /peer/link для кожного протоколу, отримує свіжі конфігурації та оновлює записи в базі даних. Аналогічно, ендпоінт /api/subscription/activate дозволяє вручну активувати підписку, якщо баланс клієнта достатній, але автоматична активація з якихось причин не спрацювала.

Взаємодія між master-сервером та проксі-вузлами відбувається за ініціативою master у відповідь на дії клієнта. Master надсилає HTTPS-запити на API-ендпоінти кожного slave-агента, використовуючи статичний API-ключ у заголовку X-API-Key для автентифікації. Основним ендпоінтом є додавання користувача на всі протоколи одночасно за адресою /peer/add-all. Master надсилає POST-запит з тілом {"username": "uuid"}, а slave-агент виконує відповідні bash-скрипти для генерації конфігурацій AmneziaWG, Xray-core, Hysteria2 та Slipstream, після чого повертає структуровану відповідь з полями protocol, success, output та error. Успішний результат для кожного протоколу містить у полі output конфігурацію, готову до використання клієнтом. Ендпоінт видалення користувача реалізовано методом POST на адресу /peer/remove з тілом {"usernames": ["uuid"]}. Master використовує цей ендпоінт при деактивації підписки через недостатній баланс. Ендпоінт отримання списку користувачів за адресою /peer/list з опціональним параметром protocol використовується для адміністративних цілей та перевірки стану проксі-вузла. Ендпоінт отримання конфігурації конкретного протоколу за адресою /peer/link з параметрами protocol та username використовується при оновленні підписки через refresh-ендпоінт. Додатково передбачено ендпоінти керування сервісами /service/restart та /service/status, які дозволяють master перезапускати проксі-сервіси після масових змін конфігурації та перевіряти їх працездатність.

Взаємодія master з monero-wallet-rpc реалізована через JSON-RPC протокол на локальному інтерфейсі з використанням файла пароля для

автентифікації. Master викликає метод create_address для генерації нової суб-адреси. Для моніторингу платежів master періодично викликає метод get_transfers з параметром in = true, який повертає всі вхідні транзакції. Master фільтрує отримані транзакції, вибираючи ті, що надійшли на адреси з таблиці deposit_addresses з прапорцем processed = 0. Для кожної такої транзакції перевіряється кількість підтверджень у блокчейні. Якщо кількість підтверджень досягає встановленого порогу, master оновлює баланс відповідного клієнта у таблиці accounts та позначає адресу як оброблену, зберігаючи ідентифікатор транзакції та час підтвердження. Після цього запускається перевірка можливості автоматичної активації підписки.

База даних master-сервера реалізована на SQLite з використанням асинхронного драйвера aiosqlite [2], що дозволяє інтегрувати її з асинхронним циклом FastAPI без блокування. Оскільки проект не є сильнонавантаженим з боку бази даних - очікується не більше кількох сотень активних підписок - використання важких клієнт-серверних СУБД було б надлишковим. SQLite не потребує окремого серверного процесу, що спрощує розгортання та резервне копіювання, а також забезпечує достатню продуктивність при помірному навантаженні. Активовано режим WAL для підвищення продуктивності конкурентного доступу та перевірку зовнішніх ключів для забезпечення цілісності даних. Схему бази даних наведено на рис. 2.5.

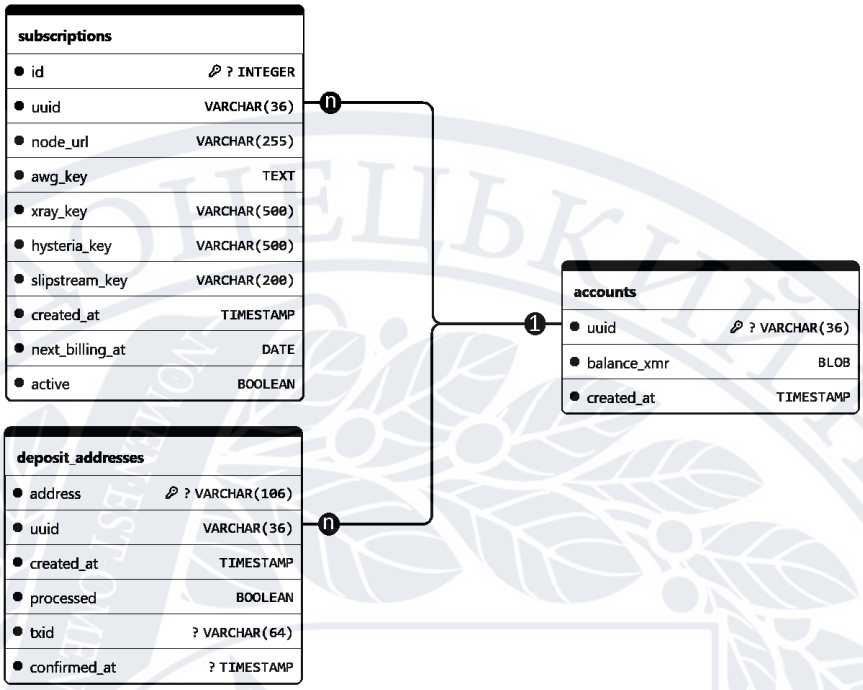

Рисунок 2.5 - Схема бази даних головного сервера

Структура бази даних складається з трьох основних таблиць. Таблиця accounts зберігає інформацію про клієнтів і містить поле uuid як первинний ключ, поле balance_xmr для зберігання поточного балансу в Monero з точністю до восьми знаків після коми та поле created_at з часом створення облікового запису. Таблиця subscriptions містить активні та історичні підписки з полями id як сурогатним ключем, uuid як зовнішнім ключем до таблиці accounts, node_url з адресою проксі-вузла, текстовими полями awg_key, xray_key, hysteria_key, slipstream_key для зберігання конфігурацій протоколів, полями created_at та next_billing_at для відстеження часу створення та дати наступного списання, а також булевим полем active для позначення активності підписки. Таблиця deposit_addresses використовується для відстеження згенерованих Monero-адрес та запобігання повторному зарахуванню платежів. Вона містить поле address як первинний ключ, поле uuid як зовнішній ключ до accounts, поле created_at для часу генерації, поле processed як прапорець обробки, поле txid для зберігання хешу транзакції та поле confirmed_at для часу підтвердження. Індекс на полі processed пришвидшує вибірку ще не оброблених адрес під час моніторингу платежів.

Цілісність даних забезпечується каскадним видаленням: при видаленні запису з accounts автоматично видаляються всі пов’язані підписки та депозитні адреси. Операції зарахування коштів та оновлення прапорця processed виконуються в атомарних транзакціях, що гарантує узгодженість даних навіть у разі збою. Типовими запитами до бази даних є отримання балансу клієнта за його UUID, отримання активної підписки з усіма конфігураціями, додавання коштів після підтвердження платежу з одночасним оновленням балансу та прапорця обробки адреси, а також щоденне списання коштів за активні підписки з оновленням балансу та дати наступного списання.

Для запобігання накопиченню невикористаних адрес щоденно о третій годині ночі виконується фонова задача, яка видаляє з таблиці deposit_addresses записи, створені раніше ніж задану кількість днів тому та з прапорцем processed = 0. Така стратегія дозволяє підтримувати базу даних в актуальному стані та уникати її неконтрольованого зростання.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕХНІЧНА ОЦІНКА ПРОКСІ-СЕРВІСУ

3.1 Конфігурування протоколів обходу блокувань та створення скриптів автоматизації

Практична реалізація проксі-сервісу починається з встановлення та налаштування чотирьох основних протоколів, які забезпечують обхід мережевих обмежень: AmneziaWG, Xray-core, Hysteria2 та Slipstream разом із SOCKS5-сервером 3proxy. Оскільки більшість цих компонентів не входять до стандартних репозиторіїв Debian, для кожного з них було розроблено інсталяційний скрипт, який виконує всі необхідні дії: додавання сторонніх репозиторіїв, встановлення залежностей, компіляцію з вихідних кодів (для Slipstream) та налаштування systemd-сервісів.

Найбільш показовим є скрипт встановлення AmneziaWG. Він додає GPG-ключ репозиторію Amnezia PPA, створює файл джерел amneziawg.sources для Ubuntu Noble, оскільки офіційний пакет відсутній у Debian, та оновлює список пакетів і встановлює amneziawg. Після встановлення скрипт перевіряє наявність модуля ядра за допомогою modinfo amneziawg та рекомендує перезавантажити систему в разі потреби. Для Slipstream інсталяція виявилася складнішою, оскільки slipstream-server написаний на Rust і не поширюється у вигляді готових пакетів. Скрипт slipstream-install.sh встановлює необхідні інструменти (build-essential, cmake, pkg-config, libssl-dev, cargo, rustc), клонує репозиторій з GitHub, збирає виконуваний файл slipstream-server у режимі release, а потім копіює його до /usr/local/bin/. Одночасно скрипт завантажує та встановлює 3proxy з офіційного deb-пакета, створює системних користувачів slipstream та 3proxy, а також генерує systemd-юніти для обох сервісів. Лістинг інсталяційного скрипта для Slipstream наведено в Додатку А.

Після встановлення всіх компонентів необхідно налаштувати конфігураційні файли. Для AmneziaWG використовується шаблон SERVER_TEMPLATE.conf, який містить параметри інтерфейсу, обфускації та

правила маршрутизації. Реальний конфігураційний файл сервера, що працює в системі, має такий вигляд: інтерфейс awg0 отримує внутрішню IPv4-адресу 10.66.66.1/24 та IPv6-адресу fd42:42:42::1/64, слухає порт 54323 через UDP. Параметри обфускації Jc, Jmin, Jmax, S1-S4, H1-H4 задають поведінку патерну пакетів. Блоки PostUp та PostDown містять команди iptables для маршрутизації: дозвіл форвардингу між публічним інтерфейсом eth0 та awg0, заборона петлевого форвардингу, дозвіл вхідних UDP-пакетів на порт 54323 та маскарадинг для вихідного трафіку. Аналогічні правила застосовуються для ip6tables.

Для Xray-core основний конфігураційний файл config.json розгорнуто на проксі-вузлі. Вхідне з’єднання налаштоване на протокол VLESS, порт 443, з використанням TLS та flow xtls-rprx-vision. Поле serverName в TLS-налаштуваннях встановлене на домен проксі вузла, який збігається з ім'ям у сертифікаті. Секція fallbacks перенаправляє з’єднання, які не є легітимним VLESS-трафіком, на внутрішні сервери: для ALPN http/1.1 на порт 6001, для ALPN h2 на порт 6002. Це дозволяє серверу відповідати як звичайний веб-сервер на зондувальні запити DPI. Для обробки fallback використовується nginx, який проксує з’єднання з портів 6001, 6002 та 6003 до unix-сокета copyparty - легковагового файлового сервера, що виконує роль сайту-приманки. Конфігурація nginx передбачає передачу заголовків X-Real-IP та X-Forwarded-For для коректного логування. Секція routing використовує завантажені геодані (geoip.dat, geosite.dat): блокуються IP-адреси та домени з регіонів ir, ru, cn, а довірені сервіси спрямовуються безпосередньо у outbound direct. Весь інший трафік передається через інтерфейс warp.

Інтерфейс warp створюється за допомогою AmneziaWG, налаштованого в режимі сумісності з WireGuard. Він забезпечує вихід трафіку через мережу Cloudflare, що дозволяє динамічно змінювати IP-адресу, з якої здійснюються запити. Такий підхід знижує ймовірність блокування цільового проксі-вузла за IP-адресою, оскільки зовнішній спостерігач бачить адресу, що належить Cloudflare, а не безпосередньо серверу розробленої системи.

Для Hysteria2 конфігураційний файл config.yaml містить аналогічну логіку маршрутизації. Секція auth використовує тип userpass із парою імен та паролів. Секція masquerade типу proxy перенаправляє звичайні HTTP-запити на внутрішній сервер coryparty за адресою http://127.0.0.1:6003/, зберігаючи заголовки. ACL (список контролю доступу) використовує ті самі геодані, що й Xray: блокування регіонів ir, ru, cn, пряме з'єднання для довірених сервісів та перенаправлення всього іншого трафіку через outbound warp. Алгоритм контролю перевантажень встановлено на BBR з агресивним профілем, що забезпечує високу продуктивність на нестабільних каналах. TLS налаштовано з використанням того самого сертифіката, що й для Xray, що спрощує інфраструктуру.

Для Slipstream конфігурація складається з двох частин. Файл 3proxy.cfg налаштовує SOCKS5-сервер: директива auth strong вмикає автентифікацію, рядок users містить пари username:CL:password, а команда socks -i127.0.0.1 -p7001 -e5.83.147.154 -a запускає сервер, який слухає на локальному інтерфейсі 127.0.0.1:7001 та використовує зовнішню публічну IP-адресу для вихідних з'єднань [36]. Systemd-сервіс slipstream.service виконує команду: slipstream-server --dns-listen-host ... --domain ${DOMAIN} --target-address 127.0.0.1:7001. Таким чином, DNS-тунель приймає з'єднання, інкапсулює їх у DNS-запити, передає через систему доменних імен та розпаковує на сервері, після чого спрямовує на SOCKS5-порт 3proxy.

Для автоматизації керування користувачами розроблено уніфіковані bash-скрипти для кожного протоколу. Всі скрипти підтримують набір однакових команд: add USERNAME, remove USERNAME..., list, link USERNAME, restart, status. Вони перевіряють права root, роблять резервні копії конфігурацій перед змінами, модифікують файли конфігурацій та перезапускають відповідні сервіси. Для скрипта amneziawg-manage.sh ключовою особливістю є функція next_client_ip, яка знаходить вільну IP-адресу в підмережі 10.66.66.0/24 шляхом перевірки наявності клієнта у файлі конфігурації сервера. Генерація ключів виконується за допомогою команд awg genkey, awg pubkey та awg genpsk. Після

додавання клієнта до SERVER_CONF скрипт викликає awg syncconf для оновлення конфігурації без перезапуску інтерфейсу. Клієнтська конфігурація генерується з шаблону CLIENT_TEMPLATE.conf шляхом заміни змінних, при цьому всі параметри обфускації підставляються з файлу PARAMS. Для видалення використовується sed для вилучення блоку клієнта між рядками ### Client NAME та наступним порожнім рядком.

Скрипт xray-manage.sh покладається на утиліту jq для редагування JSON-файлу. Він визначає індекс inbound-блока за тегом vless-in, потім додає новий об’єкт у масив settings.clients з полями id (новий UUID, згенерований через xray uuid або /proc/sys/kernel/random/uuid), flow (значення xtls-rprx-vision) та email (ім’я клієнта). Після збереження файлу виконується systemctl restart xray. Команда link генерує VLESS URI відповідно до стандарту. Усі необхідні параметри (listen_addr, port, alpn, та sni) витягуються з конфігураційного файлу за допомогою jq, що робить скрипт більш-менш універсальним для різних серверних налаштувань.

Скрипт hysteria-manage.sh використовує yq для редагування YAML. Він читає мапу auth.userpass, додає нову пару username: password, пароль генерується за допомогою openssl rand -base64 32 з подальшим фільтруванням символів. Для генерації клієнтської конфігурації скрипт витягує з серверного config.yaml параметри listen, шлях до сертифіката, обчислює його SHA256-відбиток через openssl x509 -fingerprint -sha256, підставляє їх у шаблон CLIENT_TEMPLATE.yaml, а потім викликає вбудовану команду hysteria share -c <tempfile>, яка повертає готовий URI.

Скрипт 3proxy-manage.sh працює з текстовим конфігураційним файлом. Він знаходить рядок, що починається з users, та додає до нього новий запис у форматі username:CL:password. Одночасно додається відповідне правило allow username. Після змін виконується systemctl reload 3proxy, що застосовує нову автентифікацію без розриву існуючих з’єднань. Генерація випадкового пароля виконується аналогічно до Hysteria2. Усі скрипти підтримують видалення одразу кількох користувачів.

Важливою складовою інфраструктури є також конфігурація nginx для обслуговування fallback. У наведеній конфігурації три серверні блоки слухають на портах 6001, 6002 та 6003 локального інтерфейсу. Порти 6001 та 6002 приймають з'єднання з proxy_protocol, що дозволяє Xray передавати реальну IP-адресу клієнта через заголовок PROXY. Всі три блоки проксують запити до unix-сокета coryparty. Сам coryparty - це файловий сервер, який не потребує складної конфігурації та може бути запущений як systemd-сервіс. Сайт-приманка повинен виглядати як легітимний веб-ресурс, тому будь-який зондувальний запит DPI отримує відповідь із HTTP-статусом 200 та вмістом, що не викликає підозр.

3.2 Реалізація веб-ресурсу та клієнтської логіки

Веб-ресурс проксі-сервісу реалізовано як статичний сайт, згенерований за допомогою генератора Hugo. Такий підхід забезпечує повну відсутність стану на стороні сервера - сайт не зберігає сесії, не використовує cookies та не потребує серверних мов програмування. Це суттєво зменшує поверхню атаки та спрощує розгортання. Для візуального оформлення використано тему no-js-hugo-theme, з якої було вилучено скрипт перемикання теми через його непотрібність. Згенерований сайт складається з головної сторінки, де розміщено весь необхідний функціонал: поле для введення UUID, відображення балансу, статусу підписки, кнопки для запиту балансу та поповнення, а також блок для виведення активних ключів доступу.

Клієнтська логіка реалізована у вигляді окремого JavaScript-файлу app.js, який підключається до сторінки. Він не використовує жодних зовнішніх бібліотек або фреймворків - весь код написаний на чистому JavaScript, що забезпечує легкість та незалежність від сторонніх залежностей. Повний код app.js наведено у додатку Б. Основним елементом ідентифікації клієнта є UUID, який зберігається у локальному сховищі браузера. Це дозволяє користувачеві закрити сторінку та повернутися до неї пізніше, не втрачаючи свого ідентифікатора. Функції saveUuidToStorage, getStoredUUID та updateUuidInput

забезпечують збереження, зчитування та відображення поточного UUID у текстовому полі.

Взаємодія з API master-сервера реалізована через функцію apiCall, яка приймає endpoint та опціональні параметри запиту, автоматично додає заголовок X-User-UUID з поточним ідентифікатором та встановлює тип вмісту application/json. Усі запити надсилаються через HTTPS з використанням механізму fetch. Відповідь сервера обробляється як JSON; у разі помилки (статус не в межах 2xx) генерується виняток з текстом помилки.

Функція queryBalance виконує два паралельні запити - до /api/balance для отримання поточного балансу в Monero та до /api/subscription для отримання інформації про активну підписку. Обидва запити виконуються одночасно за допомогою Promise.all. Отриманий баланс відображається з точністю до восьми знаків після коми, а статус підписки показує, чи є активний доступ. Якщо активна підписка існує, функція будує HTML-блок, що містить для кожного протоколу відповідний ключ доступу у вигляді текстового рядка. Кожен ключ супроводжується кнопкою копіювання, яка використовує механізм navigator.clipboard з fallback-варіантом через створення тимчасового текстового поля. Для захисту від XSS-атак усі динамічні дані пропускаються через функцію escapeHtml, яка замінює небезпечні символи на їх HTML-сутності.

Функція addToBalance відповідає за поповнення балансу. Вона зчитує введену користувачем суму, перевіряє, чи є воно додатним числом, та надсилає POST-запит до /api/deposit/address з тілом, що містить цю суму. У відповідь сервер повертає одноразову XMR-адресу, яка прив’язується до поточного UUID. Після отримання адреси функція відображає її у вигляді текстового блоку з поясненням, що після надсилання коштів потрібно дочекатися декількох підтверджень в мережі Monero. Жодних додаткових дій від користувача не вимагається - система автоматично зарахує баланс, як тільки транзакція буде виявлена фоновими процесами master-сервера.

Для роботи з UUID передбачено три допоміжні функції. generateNewUuid створює новий ідентифікатор, записує його у текстове поле, зберігає у

локальному сховищі та очищає всі відображені дані. Функція copyUuid копіює поточний UUID у буфер обміну, що зручно для збереження його в безпечному місці.

Після повного завантаження DOM виконується ініціалізація: відображається поточний UUID з локального сховища, реєструється обробник події для поля введення UUID, та очищуються всі динамічні поля. Це гарантує, що при першому відвідуванні сторінки або після оновлення відображаються лише актуальні дані, а не залишки від попередніх сесій.

Зовнішній вигляд сторінки веб-ресурсу наведено на рис. 3.1.

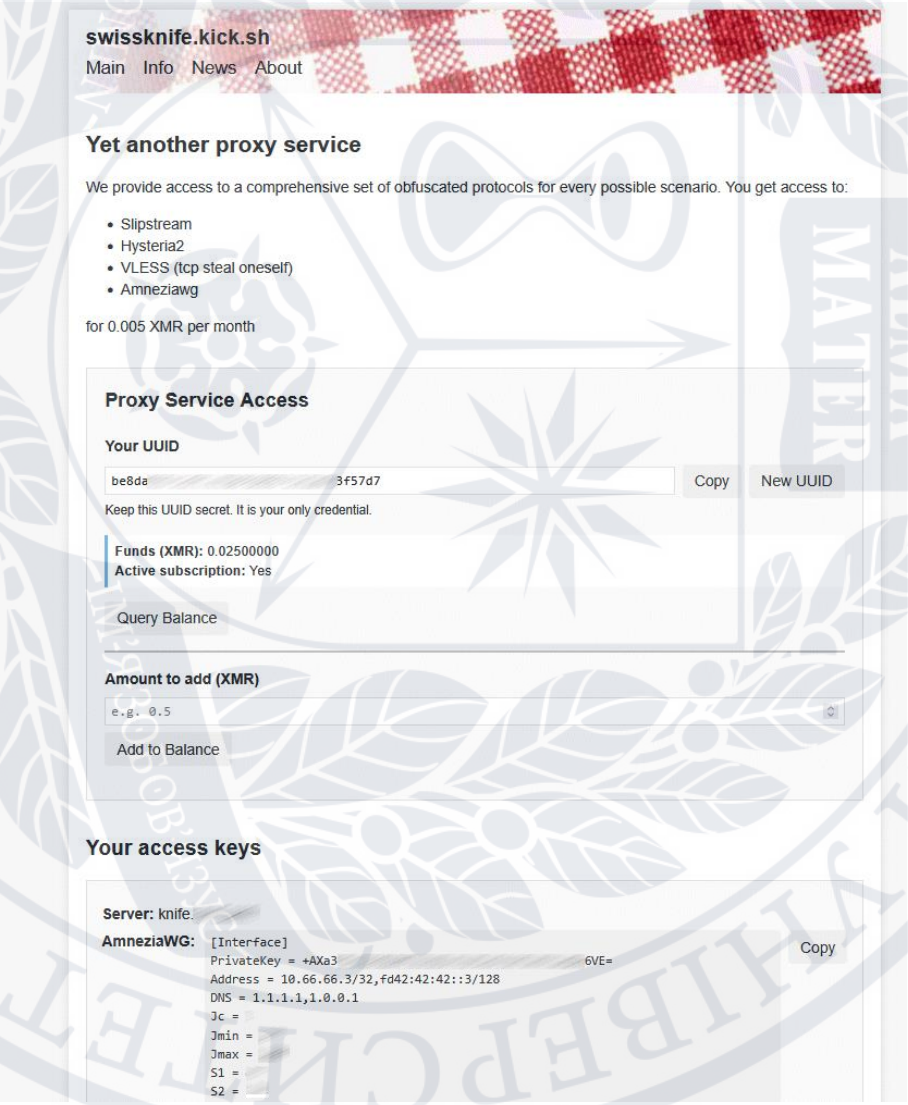

Рисунок 3.1 - Зовнішній вигляд головної сторінки

3.3 Реалізація серверної частини проксі-сервісу

Серверна частина проксі-сервісу реалізована двома окремими застосунками на Python з використанням фреймворку FastAPI. Master-сервер виконує роль центрального координатора: він обробляє запити клієнтів, взаємодіє з гаманцем Monero, керує базою даних та надсилає команди проксі-вузлам. Оскільки код master-сервера є надто об’ємним для повного наведення в роботі, у додатку наведено лише фрагменти, що ілюструють ключові механізми. Натомість slave-агент, чий повний код подано в додатку В, є легковагим і самодостатнім, тому його лістинг наведено повністю.

Slave-агент приймає команди від master та виконує їх через виклик bash-скриптів, описаних у підрозділі 3.1. Конфігурація агента зберігається у YAML-файлі, що містить статичний API-ключ для автентифікації, прапорець use_sudo та словник protocols, де кожному протоколу зіставляється шлях до відповідного скрипта. Ендпоінти агента захищені заголовком X-API-Key, який перевіряється в кожному запиті. Додатково, на рівні мережі застосовано правила iptables, що дозволяють доступ до порту агента лише з IP-адреси master-сервера. Це унеможливлює встановлення навіть TCP-з’єднання з боку будь-якого іншого хоста.

Обробка команд в slave-агенті реалізована через асинхронну функцію run_command. Вона формує командний рядок, додаючи sudo перед шляхом до скрипта, якщо відповідний прапорець встановлено. Можливість використання sudo без введення пароля забезпечується через налаштування файлу sudoers, де для користувача, від імені якого запущено агента, дозволено виконання конкретних скриптів без пароля. Це дозволяє агентові редагувати конфігураційні файли в захищених директоріях та перезапускати сервіси, не надаючи йому загальних прав root. Сам процес запускається за допомогою asyncio.create_subprocess_exec, що дозволяє агентові не блокуватися на час виконання скрипта.

Для ендпоінту /peer/add-all master надсилає POST-запит з тілом {"username": "uuid"}. Slave одночасно запускає скрипти додавання для всіх

налаштованих протоколів за допомогою asyncio.gather, що скорочує час активації підписки. Після завершення процесів агент повертає масив результатів, де кожен елемент містить назву протоколу, прапорець успішності, вихідні дані та повідомлення про помилку. У разі успіху поле output містить конфігурацію клієнта - наприклад, URI для Xray або текстовий блок для AmneziaWG. Якщо ж скрипт повертає ненульовий код, master не створює підписку та не списує кошти.

Master-сервер реалізує ключову логіку автоматизації. Він містить асинхронний моніторинг транзакцій Monero через JSON-RPC. Взаємодія з monero-wallet-rpc відбувається через HTTP без шифрування, оскільки в тестовому середовищі використовується stagenet, а RPC-сервер прив’язаний до локального інтерфейсу. У разі підключення до реальної мережі та віддаленої ноди комунікацію слід захищати через I2P або Onion-сервіси, щоб уникнути витоку конфіденційної інформації про транзакції. Master також реалізує обмеження частоти запитів для захисту від автоматичного підбору UUID або DoS-атак, а всі запити до API вимагають заголовка X-User-UUID, який перевіряється на відповідність формату UUID.

Функція call_slave в master реалізує повторні спроби при недоступності вузла. Якщо один slave не відповідає, master обирає інший випадковим чином. Після успішного додавання клієнта master зберігає отримані конфігурації в базі даних, створюючи запис у таблиці subscriptions. Щоденний білінг виконується за допомогою AsyncIOScheduler: опівночі перевіряються прострочені підписки, списуються кошти або надсилаються команди на видалення користувачів. Додаткове завдання вночі очищає таблицю deposit_addresses від застарілих необроблених адрес.

ВИСНОВКИ

У кваліфікаційній роботі виконано теоретичне та практичне дослідження, спрямоване на розробку проксі-сервісу для забезпечення стабільної доступності веб-ресурсів в умовах сучасних мережевих обмежень. На основі аналізу причин запровадження обмежень та технічних методів блокування визначено ключові виклики, з якими стикаються засоби обходу цензури. У першому розділі проаналізовано російську систему ТСПУ, методи «16 КБ блокування», білі списки та фільтрацію на рівні мобільних операторів. Проведено порівняльний огляд протоколів обходу - від базових HTTP/SOCKS5 до сучасних AmneziaWG, Hysteria2, Xray-core та Slipstream. Обґрунтовано вибір комбінації зазначених протоколів як найбільш стійких до DPI.

Сформульовано функціональні та безпекові вимоги до проксі-сервісу, зокрема: stateless веб-сайт, ідентифікація клієнта за UUID, автоматизований білінг у Monero з щомісячним списанням, генерація одноразових депозитних адрес, а також вимоги до конфігурації кожного протоколу. Розроблено розподілену архітектуру master-slave з активним керуванням, що складається зі статичного веб-сайту, master-сервера та проксі-вузлів з легковагими агентами. Запропоновано схеми маршрутизації трафіку: Для Xray-core та Hysteria2 реалізовано правила маршрутизації, згідно з якими трафік до довірених сервісів спрямовується безпосередньо, IP-адреси та домени з географічних регіонів ru, cn, ir блокуються, а весь інший трафік перенаправляється через інтерфейс Cloudflare WARP; для AmneziaWG надано параметри обфускації; для Slipstream організовано DNS-тунель з виходом на SOCKS5-сервер.

Спроєктовано внутрішній REST API, що забезпечує взаємодію між фронтендом, master-сервером та проксі-вузлами. Основні ендпоінти: отримання балансу, активної підписки, генерація депозитної адреси, оновлення ключів, а також команди додавання, видалення, отримання конфігурацій та керування сервісами на вузлах. Розроблено структуру бази даних SQLite з атомарними

транзакціями, каскадним видаленням та автоматичним очищенням застарілих адрес.

Програмну реалізацію виконано на рівні конфігураційних файлів, bash-скриптів автоматизації, веб-ресурсу та серверної частини. Створено інсталяційні скрипти для AmneziaWG та Slipstream, а також уніфіковані скрипти для кожного протоколу.

Проведене експериментальне дослідження підтвердило працездатність розробленої архітектури в умовах застосування сучасних методів мережевої фільтрації. Обрана комбінація протоколів демонструє високу стійкість до DPI, що робить систему ефективною для використання в країнах із високим рівнем інтернет-цензури.

Отримані результати мають практичне значення: розгорнутий проксі-сервіс дозволяє автоматизувати процес надання доступу до обфускованих тунелів без участі адміністратора, а розроблені скрипти та API можуть бути використані як основа для подібних систем. Перспективи подальшого розвитку включають додавання протоколу sing-box, інтеграцію з анонімними мережами (I2P, Tor) для приховування RPC-запитів до Monero, а також розширення функціоналу веб-ресурсу для перегляду історії платежів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про рішення Ради національної безпеки і оборони України від 28 квітня 2017 року «Про застосування персональних спеціальних економічних та інших обмежувальних заходів (санкцій)»: Указ Президента України № 133/2017. URL: https://www.president.gov.ua/documents/1332017-21850 (дата звернення: 29.05.2026).
2. aiosqlite API Reference. aiosqlite Documentation. URL: https://aiosqlite.omnilib.dev/en/stable/api.html (дата звернення: 29.05.2026).
3. Althouse J. TLS Fingerprinting with JA3 and JA3S. Salesforce Engineering Blog. URL: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967/ (дата звернення: 29.05.2026).
4. AmneziaWG documentation. Amnezia. URL: https://docs.amnezia.org/documentation/amnezia-wg/ (дата звернення: 09.05.2026).
5. Censor has a new method of blocking. Net4People BBS. URL: https://github.com/net4people/bbs/issues/490 (дата звернення: 29.05.2026).
6. Configuration. Mihomo Docs. URL: https://wiki.metacubex.one/en/config/ (дата звернення: 29.05.2026).
7. Donenfeld J. A. WireGuard: Next Generation Kernel Network Tunnel. Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS). San Diego: Internet Society, 2017. P. 1-18.
8. FastAPI Reference Guide. FastAPI Documentation. URL: https://fastapi.tiangolo.com/reference/ (дата звернення: 29.05.2026).
9. Fifield D. DNSTT: DNS Tunnel through DoH or DoT. BAM Software. URL: https://www.bamsoftware.com/software/dnstt/ (дата звернення: 09.05.2026).
10. Fifield D. Threat modeling and circumvention of Internet censorship: diss. … Doctor of Philosophy in Computer Science. University of California, Berkeley. Berkeley, 2017. 91 p.

11. Hall B. Beej’s Guide to Network Concepts. Version 1.0.40. 2025. 178 p. URL: https://beej.us/guide/bgnet0/ (дата звернення: 09.05.2026).
12. How does XTLS REALITY break through the whitelist? REALITY source code analysis. URL: https://objshadow.pages.dev/en/posts/how-reality-works/ (дата звернення: 29.05.2026).
13. How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic. GFW Report. URL: https://gfw.report/publications/usenixsecurity23/en/ (дата звернення: 29.05.2026).
14. Hugo documentation. Hugo. URL: https://gohugo.io/documentation/ (дата звернення: 29.05.2026).
15. Hysteria 2 Documentation. Hysteria. URL: https://v2.hysteria.network/docs/advanced/Full-Server-Config/ (дата звернення: 09.05.2026).
16. Measuring SNI based blocking in Iran. OONI. URL: https://ooni.org/post/2020-iran-sni-blocking/ (дата звернення: 29.05.2026).
17. Monero Wallet RPC API Reference. Monero Documentation. URL: https://docs.getmonero.org/rpc-library/wallet-rpc/ (дата звернення: 29.05.2026).
18. Ramesh R., Raman R. S., Virkud A. et al. Network Responses to Russia’s Invasion of Ukraine in 2022: A Cautionary Tale for Internet Freedom. Proceedings of the 32nd USENIX Security Symposium (August 9-11, 2023, Anaheim, CA, USA). Anaheim: USENIX Association, 2023. P. 2045-2062.
19. Ristić I. Bulletproof TLS and PKI. 2nd ed. London: Feisty Duck, 2022. 550 p.
20. Roberts M. E. Censored: Distraction and Diversion Inside China’s Great Firewall. Princeton: Princeton University Press, 2018. 288 p.
21. Russia continues to restrict mobile internet, introduces website "whitelists". Ukrainska Pravda. URL: https://www.pravda.com.ua/eng/news/2026/03/19/8026171/ (дата звернення: 29.05.2026).

22. Russian internet users are unable to access the open internet. Cloudflare Blog. URL: https://blog.cloudflare.com/russian-internet-users-are-unable-to-access-the-open-internet/ (дата звернення: 29.05.2026).
23. Sandeen O. Iptables Tutorial 1.2.2. Frozen Tux. URL: https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html (дата звернення: 29.05.2026).
24. Shadowsocks AEAD ciphers. Shadowsocks. URL: https://shadowsocks.org/doc/aead.html (дата звернення: 29.05.2026).
25. Sing-box Configuration. Sing-box Documentation. URL: https://sing-box.sagernet.org/configuration/ (дата звернення: 29.05.2026).
26. Slipstream Protocol Design. GitHub Pages. URL: https://endpositive.github.io/slipstream/protocol.html (дата звернення: 09.05.2026).
27. Tanenbaum A. S., Wetherall D. J. Computer networks. 5th ed. Boston: Prentice Hall, 2011. 958 p.
28. Technical Analysis of the IRGFW. MahsaNet Research. URL: https://mahsanet.com/content/blog/6/report-english.pdf (дата звернення: 29.05.2026).
29. TLS connection-based policing applied on some home ISPs. Net4People BBS. URL: https://github.com/net4people/bbs/issues/546 (дата звернення: 29.05.2026).
30. VPN in Russia: from blocking services toblocking protocols. Roskomsvoboda. 2024. 26 p. URL: https://roskomsvoboda.org/uploads/en__vpn_in_russia__from_blocking_services_to_blocking_protocols.pdf (дата звернення: 29.05.2026).
31. WireGuard obfuscation and padding mechanisms. WireGuard Mailing List. URL: https://lists.zx2c4.com/pipermail/wireguard/2018-September/003289.html (дата звернення: 29.05.2026).
32. Xray Core Configuration. XTLS GitHub Pages. URL: https://xtls.github.io/en/config/ (дата звернення: 29.05.2026).

33. Xray REALITY Configuration. XTLS GitHub Pages. URL: https://xtls.github.io/en/config/transports/reality.html (дата звернення: 29.05.2026).
34. Xue D., Mixon-Baca B., Ablove A. et al. TSPU: Russia's Decentralized Censorship System. Internet Measurement Conference. Nice: ACM, 2022. P. 1-16.
35. Zapret: Deep Packet Inspection circumvention utility for Linux. GitHub Repository. URL: https://github.com/bol-van/zapret/blob/master/docs/readme.en.md (дата звернення: 29.05.2026).
36. 3proxy documentation. 3proxy. URL: https://3proxy.org/documents/ (дата звернення: 29.05.2026).

ДОДАТКИ

ДОДАТОК А

Код інсталяційного скрипта slipstream-install.sh

```
#!/bin/bash
set -euo pipefail

YELLOW='\033[0;33m'
NC='\033[0m'
info()   { echo -e "${YELLOW}[info] $*${NC}"; }
ok()     { echo -e "${YELLOW}[ok] $*${NC}"; }
abort()  { echo "[error] $*" >&2; exit 1; }
yprint() { echo -e "${YELLOW}$*${NC}"; }

CONFIG_FILE="CONFIG.conf"
BUILD_DIR="/tmp/slipstream-build-$$"

PROXY_VERSION="0.9.6"
PROXY_DEB_FILE="/tmp/3proxy-${PROXY_VERSION}.x86_64.deb"
PROXY_DEB_URL="https://github.com/3proxy/3proxy/releases/download/
${PROXY_VERSION}/3proxy-${PROXY_VERSION}.x86_64.deb"

SLIPSTREAM_URL="https://github.com/Mygod/slipstream-rust.git"
SLIPSTREAM_USER="slipstream"
SLIPSTREAM_BIN="/usr/local/bin/slipstream-server"

PROXY_USER="3proxy"
PROXY_CONF="/etc/3proxy/3proxy.cfg"
PROXY_CONFIG_DIR="/etc/3proxy"

SERVICE_SLIPSTREAM="/etc/systemd/system/slipstream.service"
SERVICE_3PROXY="/etc/systemd/system/3proxy.service"

if [[ $EUID -ne 0 ]]; then
    abort "Run as root: sudo $0"
fi

if [[ ! -f "${CONFIG_FILE}" ]]; then
    abort "Config not found: ${CONFIG_FILE}"
fi

info "Using config from ${CONFIG_FILE}"
source "${CONFIG_FILE}"
ok "Config loaded"

required_vars=(
    "LISTEN_ADDRESS" "LISTEN_PORT" "CERT_LOCATION" "KEY_LOCATION"
    "DOMAIN" "PROXY_INTERNAL_ADDRESS" "PROXY_INTERNAL_PORT"
    "PROXY_EXTERNAL_ADDRESS"
)
for var in "${required_vars[@]}"; do
```

```
    if [[ -z "${!var:-}" ]]; then
        abort "Variable $var is not set in $CONFIG_FILE"
    fi
done
echo ""
yprint "slipstream-rust + 3proxy installer for Debian Trixie"
echo ""

if id "${SLIPSTREAM_USER}" &>/dev/null; then
    ok "System user '${SLIPSTREAM_USER}' already exists"
else
    info "Creating system user '${SLIPSTREAM_USER}'..."
    useradd --system --shell /usr/sbin/nologin --no-create-home
"${SLIPSTREAM_USER}"
fi

if id "${PROXY_USER}" &>/dev/null; then
    ok "System user '${PROXY_USER}' already exists"
else
    info "Creating system user '${PROXY_USER}'..."
    useradd --system --shell /usr/sbin/nologin --no-create-home
"${PROXY_USER}"
fi

info "Installing dependencies..."
apt-get update -qq
apt-get install -y --no-install-recommends \
    build-essential cmake pkg-config libssl-dev \
    cargo rustc curl git

if ! command -v rustc &>/dev/null; then
    info "Installing rustup..."
    curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
-s -- -y --default-toolchain stable --profile minimal
    source "$HOME/.cargo/env"
    rustup update stable
fi

info "Creating directories..."
mkdir -p "${PROXY_CONFIG_DIR}"
mkdir -p /run/3proxy
chown ${PROXY_USER}:${PROXY_USER} /run/3proxy
chmod 755 /run/3proxy

# build it lol
if [[ ! -f "${SLIPSTREAM_BIN}" ]]; then
    info "Building slipstream-rust..."
    git clone --recursive "${SLIPSTREAM_URL}" "${BUILD_DIR}"
    cd "${BUILD_DIR}"
    cargo build --release
    cp "${BUILD_DIR}/target/release/slipstream-server"
"${SLIPSTREAM_BIN}"
```

```
    chmod 755 "${SLIPSTREAM_BIN}"
    chown "${SLIPSTREAM_USER}:${SLIPSTREAM_USER}"
"${SLIPSTREAM_BIN}"
    rm -rf "${BUILD_DIR}"
    ok "slipstream-server built"
else
    ok "slipstream-server already installed"
fi

if ! command -v 3proxy &>/dev/null; then
    info "Downloading 3proxy .deb..."
    curl -fsSL "${PROXY_DEB_URL}" -o "${PROXY_DEB_FILE}"
    dpkg -i "${PROXY_DEB_FILE}"
    rm -f "${PROXY_DEB_FILE}"
    ok "3proxy installed"
else
    ok "3proxy already installed"
fi

if [[ -f "${PROXY_CONF}" ]]; then
    BACKUP="${PROXY_CONF}.backup.$(date +%Y%m%d_%H%M%S)"
    cp "${PROXY_CONF}" "${BACKUP}"
    info "Backed up existing config to ${BACKUP}"
fi

info "Generating 3proxy config..."
cat > "${PROXY_CONF}" <<EOF
daemon
pidfile /run/3proxy/3proxy.pid
log /dev/log syslog

nserver 1.1.1.1
nserver 1.0.0.1

auth strong
flush

socks -p${PROXY_INTERNAL_PORT} -i${PROXY_INTERNAL_ADDRESS} -
e${PROXY_EXTERNAL_ADDRESS}
EOF
chmod 644 "${PROXY_CONF}"
chown root:${PROXY_USER} "${PROXY_CONF}"

if [[ ! -f "${SERVICE_SLIPSTREAM}" ]]; then
    info "Creating slipstream service..."
    cat > "${SERVICE_SLIPSTREAM}" <<EOF
[Unit]
Description=slipstream daemon
After=network.target 3proxy.service
StartLimitBurst=0
StartLimitIntervalSec=0
```

```
[Service]
Type=simple
User=${SLIPSTREAM_USER}
Group=${SLIPSTREAM_USER}
SupplementaryGroups=www-data
CapabilityBoundingSet=CAP_NET_ADMIN CAP_NET_BIND_SERVICE
AmbientCapabilities=CAP_NET_ADMIN CAP_NET_BIND_SERVICE
ExecStart=${SLIPSTREAM_BIN} --dns-listen-host ${LISTEN_ADDRESS} --
dns-listen-port ${LISTEN_PORT} --cert ${CERT_LOCATION} --key
${KEY_LOCATION} --domain ${DOMAIN} --target-address
${PROXY_INTERNAL_ADDRESS}:${PROXY_INTERNAL_PORT}
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF
fi

if [[ ! -f "${SERVICE_3PROXY}" ]]; then
    info "Creating 3proxy service..."
    cat > "${SERVICE_3PROXY}" <<EOF
[Unit]
Description=3proxy SOCKS5 server
After=network.target

[Service]
Type=forking
User=${PROXY_USER}
Group=${PROXY_USER}
RuntimeDirectory=3proxy
RuntimeDirectoryMode=0755
ExecStart=/usr/bin/3proxy ${PROXY_CONF}
ExecReload=/bin/kill -SIGUSR1 $MAINPID
LimitNOFILE=65536
LimitNPROC=32768
Restart=on-failure
RestartSec=5


[Install]
WantedBy=multi-user.target
EOF
fi

systemctl daemon-reload
systemctl enable --now 3proxy
systemctl enable --now slipstream

echo ""
yprint "Installation completed!"
```

ДОДАТОК Б

JavaScript код app.js

```
"use strict";

const CONFIG = {
    API_BASE_URL: "/api",
    STORAGE_KEY: "proxy_user_uuid",
};

function generateUUID() {
    return "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g,
(c) => {
        const r = (Math.random() * 16) | 0;
        const v = c === "x" ? r : (r & 0x3) | 0x8;
        return v.toString(16);
    });
}

function saveUuidToStorage(uuid) {
    if (uuid && uuid.trim() !== "") {
        localStorage.setItem(CONFIG.STORAGE_KEY, uuid.trim());
    } else {
        localStorage.removeItem(CONFIG.STORAGE_KEY);
    }
}

function getStoredUUID() {
    return localStorage.getItem(CONFIG.STORAGE_KEY) || "";
}

function updateUuidInput() {
    const uuidField = document.getElementById("uuid-display");
    if (uuidField)
        uuidField.value = getStoredUUID();
}

function escapeHtml(str) {
    if (!str)
        return "";
    const div = document.createElement("div");
    div.textContent = str;
    return div.innerHTML;
}

function getUUID() {
    const uuid = document.getElementById("uuid-
display")?.value?.trim();
    if (!uuid) {
        throw new Error("No UUID provided");
    }
```

```
    return uuid;
}

function clearDisplay() {
    const balanceSpan = document.getElementById("balance-value");
    const statusSpan = document.getElementById("active-status");
    const keysDiv = document.getElementById("subscription-keys");
    const depositInfo = document.getElementById("deposit-info");
    const amountInput = document.getElementById("deposit-amount");

    if (balanceSpan)
        balanceSpan.textContent = "—";
    if (statusSpan)
        statusSpan.textContent = "—";
    if (keysDiv)
        keysDiv.innerHTML = "";
    if (depositInfo)
        depositInfo.innerHTML = "";
    if (amountInput)
        amountInput.value = "";
}

async function apiCall(endpoint, options = {}) {
    const uuid = getUUID();
    const config = {
        headers: {
            "Content-Type": "application/json",
            "X-User-UUID": uuid,
        },
        ...options,
    };
    const response = await
fetch(`${CONFIG.API_BASE_URL}${endpoint}`, config);
    if (!response.ok) {
        const errorText = await response.text();
        throw new Error(`API error ${response.status}:
${errorText}`);
    }
    return response.json();
}

function copyToClipboard(text) {
    if (navigator.clipboard && navigator.clipboard.writeText) {
        navigator.clipboard.writeText(text)
            .catch(() => fallbackCopyText(text));
    } else {
        fallbackCopyText(text);
    }
}

function fallbackCopyText(text) {
    const textarea = document.createElement('textarea');
```

```
    textarea.value = text;
    document.body.appendChild(textarea);
    textarea.select();
    document.execCommand('copy');
    document.body.removeChild(textarea);
}

window.queryBalance = async function() {
    const balanceSpan = document.getElementById("balance-value");
    const statusSpan = document.getElementById("active-status");
    const keysDiv = document.getElementById("subscription-keys");

    if (!balanceSpan || !statusSpan)
        return;

    const uuid = document.getElementById("uuid-
display")?.value?.trim();
    if (!uuid) {
        alert("Please enter or generate a UUID first.");
        return;
    }

    try {
        balanceSpan.textContent = "Loading...";
        statusSpan.textContent = "Loading...";

        const [balanceData, subData] = await Promise.all([
            apiCall("/balance"),
            apiCall("/subscription"),
        ]);

        const balance = typeof balanceData.balance === "number" ?
balanceData.balance.toFixed(8) : "0";
        balanceSpan.textContent = balance;

        const hasActive = subData.active_subscriptions?.length >
0;
        statusSpan.textContent = hasActive ? "Yes" : "No";

        if (keysDiv) {
            if (hasActive) {
                let html = "<h2>Your access keys</h2>";
                for (const sub of subData.active_subscriptions) {
                    html += `<div class="key-card">`;
                    html += `<p><strong>Server:</strong>
${escapeHtml(sub.node_url)}</p>`;

                    const addKeyRow = (label, keyValue) => {
                        if (!keyValue)
                            return '';
                        const keyId = `key-${Date.now()}-
${Math.random()}`;
```

```
                        return `
                <div class="key-row">
                  <strong>${label}:</strong>
                  <code id="${keyId}">${escapeHtml(keyValue)}</code>
                  <button class="btn" data-
target="${keyId}">Copy</button>
                </div>
              `;
                        };

                        html += addKeyRow('AmneziaWG', sub.awg_key);
                        html += addKeyRow('Xray', sub.xray_key);
                        html += addKeyRow('Hysteria2',
sub.hysteria_key);
                        html += addKeyRow('Slipstream',
sub.slipstream_key);

                        html += `<hr /></div>`;
                    }
                    keysDiv.innerHTML = html;

                    document.querySelectorAll('.key-row
.btn').forEach(btn => {
                        btn.addEventListener('click', () => {
                            const targetId = btn.getAttribute('data-
target');
                            const codeElem =
document.getElementById(targetId);
                            if (codeElem)
                                copyToClipboard(codeElem.textContent);
                        });
                    });
                }
            }
        } catch (err) {
            console.error(err);
            if (balanceSpan)
                balanceSpan.textContent = "Error";
            if (statusSpan)
                statusSpan.textContent = "Error";
        }
};

window.addToBalance = async function() {
    const amountInput = document.getElementById("deposit-amount");
    const depositInfo = document.getElementById("deposit-info");
    if (!depositInfo)
        return;

    const uuid = document.getElementById("uuid-
display")?.value?.trim();
    if (!uuid) {
```

```
            alert("Please enter or generate a UUID first.");
            return;
        }

        try {
            const amount = parseFloat(amountInput?.value);
            if (isNaN(amount) || amount <= 0) {
                alert("Please enter a valid amount");
                return;
            }

            depositInfo.innerHTML = "<p>Requesting deposit
address...</p>";
            const data = await apiCall("/deposit/address", {
                method: "POST",
                body: JSON.stringify({
                    amount
                }),
            });

            if (data?.address) {
                depositInfo.innerHTML = `
            <div>
              <p>Send exactly <strong>${amount} XMR</strong> to this
address:</p>
              <code>${escapeHtml(data.address)}</code>
              <p><small>The system will credit your balance after 1
confirmation.</small></p>
            </div>
          `;
            } else {
                throw new Error("No address returned");
            }
        } catch (err) {
            depositInfo.innerHTML = `<p>Error: ${err.message}</p>`;
        }
    };

    window.copyUuid = function() {
        const uuidField = document.getElementById("uuid-display");
        if (!uuidField)
            return;
        const uuid = uuidField.value.trim();
        if (!uuid) {
            alert("No UUID to copy.");
            return;
        }

        if (navigator.clipboard?.writeText) {
            navigator.clipboard.writeText(uuid)
                .catch(() => fallbackCopyText(uuid));
        } else {
```

```
            fallbackCopyText(uuid);
        }
    };

    window.generateNewUuid = function() {
        const newUuid = generateUUID();
        const uuidField = document.getElementById("uuid-display");
        if (uuidField)
            uuidField.value = newUuid;
        saveUuidToStorage(newUuid);
        clearDisplay();
        console.log("New UUID generated and applied:", newUuid);
    };

    function onUuidInput() {
        const uuidField = document.getElementById("uuid-display");
        if (uuidField)
            saveUuidToStorage(uuidField.value);
    }

    document.addEventListener("DOMContentLoaded", () => {
        updateUuidInput();
        const uuidField = document.getElementById("uuid-display");
        if (uuidField)
            uuidField.addEventListener("input", onUuidInput);
        clearDisplay();
    });
```

ДОДАТОК В

Код proxyagent.py для proxy ноди

```
#!/usr/bin/env python3

import asyncio
import sys
from pathlib import Path

import yaml
from fastapi import FastAPI, HTTPException, Header
from pydantic import BaseModel
from typing import List, Dict, Any, Optional
import uvicorn

CONFIG_PATH = "/etc/proxyagent/config.yaml"

def load_config():
    with open(CONFIG_PATH, "r") as f:
        return yaml.safe_load(f)

config = load_config()
API_KEY = config["api_key"]
USE_SUDO = config.get("use_sudo", True)
TIMEOUT = config.get("timeout_seconds", 30)
LISTEN_HOST = config.get("listen_host", "127.0.0.1")
LISTEN_PORT = config.get("listen_port", 9000)

PROTOCOLS: Dict[str, Path] = {}
for name, info in config["protocols"].items():
    path = Path(info["path"])
    if path.exists() and path.is_file():
        PROTOCOLS[name] = path
    else:
        print(f"Warning: protocol {name} script not found at
{path}", file=sys.stderr)

if not PROTOCOLS:
    print("ERROR: No valid protocol scripts configured",
file=sys.stderr)
    sys.exit(1)

app = FastAPI()

class AddAllResponse(BaseModel):
    protocol: str
    success: bool
    output: str
    error: str = ""
```

```
class RemoveRequest(BaseModel):
    usernames: List[str]
    protocols: Optional[List[str]] = None

async def run_command(protocol: str, command: str, args:
List[str]) -> Dict[str, Any]:
    script_path = PROTOCOLS.get(protocol)
    if not script_path:
        return {
            "protocol": protocol,
            "success": False,
            "output": "",
            "error": f"Protocol '{protocol}' not found"
        }
    cmd_parts = []
    if USE_SUDO:
        cmd_parts.append("sudo")
    cmd_parts.append(str(script_path))
    cmd_parts.append(command)
    cmd_parts.extend(args)

    try:
        proc = await asyncio.create_subprocess_exec(
            *cmd_parts,
            stdout=asyncio.subprocess.PIPE,
            stderr=asyncio.subprocess.PIPE
        )
        stdout, stderr = await
asyncio.wait_for(proc.communicate(), timeout=TIMEOUT)
        return {
            "protocol": protocol,
            "success": proc.returncode == 0,
            "output": stdout.decode().strip(),
            "error": stderr.decode().strip()
        }
    except asyncio.TimeoutError:
        proc.kill()
        await proc.communicate()
        return {
            "protocol": protocol,
            "success": False,
            "output": "",
            "error": f"Command '{command}' timed out after
{TIMEOUT}s"
        }
    except Exception as e:
        return {
            "protocol": protocol,
            "success": False,
            "output": "",
            "error": str(e)
        }
```

```
class AddRequest(BaseModel):
    username: str

@app.post("/peer/add-all", response_model=List[AddAllResponse])
async def add_all_peers(req: AddRequest, x_api_key: str =
Header(...)):
    if x_api_key != API_KEY:
        raise HTTPException(401, "Invalid API key")

    username = req.username.strip()
    if not username:
        raise HTTPException(400, "Username cannot be empty")

    tasks = [run_command(proto, "add", [username]) for proto in
PROTOCOLS.keys()]
    results = await asyncio.gather(*tasks)

    response = []
    for res in results:
        response.append(AddAllResponse(
            protocol=res["protocol"],
            success=res["success"],
            output=res["output"],
            error=res["error"]
        ))
    return response

@app.post("/peer/remove")
async def remove_peers(req: RemoveRequest, x_api_key: str =
Header(...)):
    if x_api_key != API_KEY:
        raise HTTPException(401, "Invalid API key")
    if not req.usernames:
        raise HTTPException(400, "usernames list cannot be empty")

    protocols_to_use = req.protocols if req.protocols else
list(PROTOCOLS.keys())
    unknown = [p for p in protocols_to_use if p not in PROTOCOLS]
    if unknown:
        raise HTTPException(400, f"Unknown protocols: {unknown}")

    results = []
    for proto in protocols_to_use:
        res = await run_command(proto, "remove", req.usernames)
        results.append(res)
    return {"status": "ok", "results": results}

@app.get("/peer/list")
async def list_peers(protocol: Optional[str] = None, x_api_key:
str = Header(...)):
    if x_api_key != API_KEY:
```

```
        raise HTTPException(401, "Invalid API key")
    if protocol:
        if protocol not in PROTOCOLS:
            raise HTTPException(404, f"Protocol {protocol} not
found")
        res = await run_command(protocol, "list", [])
        if not res["success"]:
            raise HTTPException(500, res["error"])
        return {"protocol": protocol, "users":
res["output"].splitlines()}
    else:
        all_lists = {}
        for proto in PROTOCOLS.keys():
            res = await run_command(proto, "list", [])
            if res["success"]:
                users = [u for u in res["output"].splitlines() if
u]
                all_lists[proto] = users
        return all_lists

@app.get("/peer/link")
async def get_peer_link(protocol: str, username: str, x_api_key:
str = Header(...)):
    if x_api_key != API_KEY:
        raise HTTPException(401, "Invalid API key")
    if protocol not in PROTOCOLS:
        raise HTTPException(404, f"Protocol {protocol} not found")
    res = await run_command(protocol, "link", [username])
    if not res["success"]:
        raise HTTPException(500, res["error"])
    return {"protocol": protocol, "username": username, "config":
res["output"]}

@app.post("/service/restart")
async def restart_service(protocol: str, x_api_key: str =
Header(...)):
    if x_api_key != API_KEY:
        raise HTTPException(401, "Invalid API key")
    if protocol not in PROTOCOLS:
        raise HTTPException(404, f"Protocol {protocol} not found")
    res = await run_command(protocol, "restart", [])
    if not res["success"]:
        raise HTTPException(500, res["error"])
    return {"status": "ok", "message": f"{protocol} restarted"}

@app.get("/service/status")
async def get_service_status(protocol: str, x_api_key: str =
Header(...)):
    if x_api_key != API_KEY:
        raise HTTPException(401, "Invalid API key")
    if protocol not in PROTOCOLS:
        raise HTTPException(404, f"Protocol {protocol} not found")
```

```
    res = await run_command(protocol, "status", [])
    if not res["success"]:
        raise HTTPException(500, res["error"])
    return {"protocol": protocol, "status": res["output"]}

if __name__ == "__main__":
    uvicorn.run("proxyagent:app", host=LISTEN_HOST,
port=LISTEN_PORT, log_level="info
```