

КЛІЄНТ-СЕРВЕРНА СИСТЕМА МОНІТОРИНГУ, ОЦІНКИ ВАРТОСТІ ТА УПРАВЛІННЯ ЦИФРОВИМИ АКТИВАМИ У РОЗПОДІЛЕНИХ СЕРЕДОВИЩАХ

Шпак Віктор Валентинович

студент групи МПМ-1-25-14д,

Київського столичного університету імені Бориса Грінченка,

науковий керівник – д.т.н., проф. Бушма О.В.

ВСТУП

Сучасний етап розвитку глобальної цифрової індустрії та мережових інформаційних технологій характеризується інтенсивним формуванням автономних економічних екосистем навколо масштабних ігрових та торговельних веб-платформ (зокрема екосистеми Steam корпорації Valve). Ринок віртуальних ігрових предметів (скінів, колекційних кейсів, наліпок, ключів тощо) виріс із нішевого середовища у багатомільярдну галузь цифрових активів, що володіє високою ліквідністю, динамічною волатильністю цін та безперервним цілодобовим обігом капіталу [1]. Проте управління масивами таких активів стикається із серйозними інженерними та інфраструктурними викликами, пов'язаними з високим навантаженням, обмеженнями зовнішніх інтерфейсів (API Rate Limits), системами захисту від автоматизованих дій (антифрод-моніторинг) та необхідністю гарантування безпеки децентралізованих облікових записів.

Актуальність і постановка проблеми. Користувачі та фінансово-аналітичні групи, що оперують значними пулами облікових записів (від сотень до десятків тисяч акаунтів), стикаються із завданнями постійної синхронізації інвентарів, моніторингу цінкових коливань у реальному часі, автоматизованого виконання угод та балансування активів. Ручна реалізація таких рутинних процедур вимагає колосальних часових витрат і супроводжується критичним ризиком людської помилки. Водночас використання традиційних автоматизованих скриптів стикається з жорсткими обмеженнями платформ: блокуванням мережових адрес системами захисту (наприклад, Akamai Edge), системними збоями авторизації під час використання захищених SOCKS5/HTTP-проксі, блокуванням обчислювального циклу подій (Event Loop) середовища Node.js під час масового розрахунку математичних моделей обміну, а також загрозою несанкціонованого доступу, зламу або нелегального копіювання клієнтського програмного забезпечення на сторонніх віртуальних машинах [2].

Таким чином, постає актуальна науково-практична проблема розробки комплексної клієнт-серверної архітектури, яка поєднує високу швидкість обробки інформації, стійкість до мережових лімітів, захист облікових даних користувачів за допомогою асиметричної криптографії та Time-Based One-Time Password (TOTP) алгоритмів, паралелізацію складних обчислень та надійний багаторівневий механізм захисту інтелектуальної власності.

Мета дослідження. Спроекувати, реалізувати та дослідити високопродуктивний клієнт-серверний програмний комплекс на базі технологічного стека Electron, Node.js, SQLite3 та хмарної інфраструктури для автоматизованого моніторингу стану тисяч розподілених вузлів, арбітражу та управління цифровими активами з вбудованим механізмом обходу антифрод-систем, багатопотоковим розрахунком математичного очікування угод і криптографічним захистом вихідного коду (компіляція у бінарний V8 Bytecode) та апаратною прив'язкою ліцензії (HWID).

Аналіз останніх досліджень і публікацій. Фундаментальні принципи побудови розподілених систем, координації процесів, забезпечення відмовостійкості та балансування навантаження висвітлено у працях Е. Таненбаума, М. ван Стеена [1] та інших провідних вчених у галузі комп'ютерних наук. Питання криптографічної автентифікації на базі часових маркерів та алгоритмів TOTP стандартизовано робочою групою IETF у специфікації RFC 6238 [3]. Механізми автоматизації браузерних сесій, обробки протоколів WebRequest та створення ізольованих середовищ вивчалися у роботах, присвячених Chromium та Puppeteer [4, 5]. Водночас аналіз відомих публікацій свідчить про суттєвий дефіцит науково-технічних рішень, що враховують специфіку внутрішньоігрових економік (зокрема Counter-Strike 2, TF2, Rust): взаємодію з Game Coordinator за бінарними Protobuf-протоколами, обхід адаптивних лімітів Akamai та захист комерційного десктопного ПЗ від реверс-інжинірингу в ненадійних хмарних середовищах.

КОРОТКИЙ ОПИС ДОСЛІДЖЕННЯ, ЙОГО МЕТОДІВ І ЗАСОБІВ

Розроблений програмний комплекс базується на гібридній трирівневій клієнт-серверній архітектурі, що забезпечує розподіл навантаження між локальними клієнтськими машинами користувачів та централізованим хмарним сервером верифікації ліцензій (рис. 1). Десктопний клієнт функціонує на платформі Electron, яка об'єднує графічний рушій Chromium для рендерингу користувацького інтерфейсу та асинхронне середовище виконання Node.js для реалізації критичної системної та мережевої логіки.

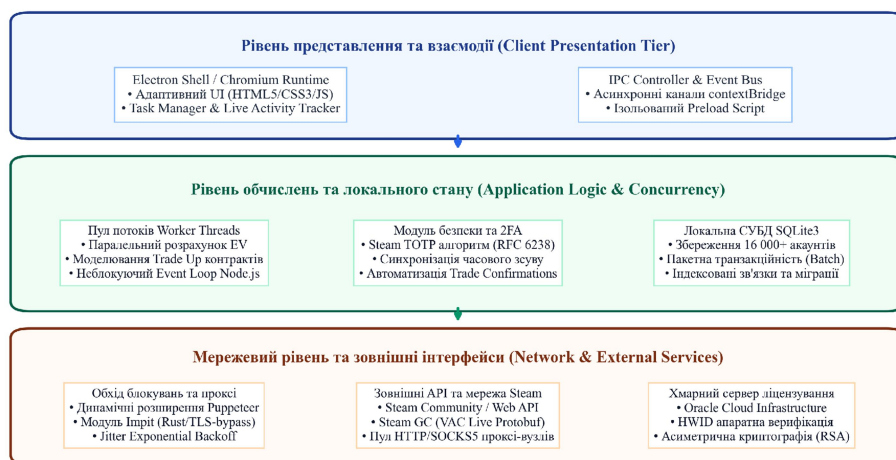


Рис. 1. Багаторівнева клієнт-серверна архітектура комплексу моніторингу та управління активами

Локальний рівень персистентності даних реалізовано за допомогою високоефективної реляційної вбудованої СУБД SQLite3 [6]. На відміну від файлових сховищ типу JSON, SQLite3 гарантує повну транзакційну безпеку (ACID), підтримує індексування складних зв'язків між сутностями «Акаунт – Сесія – Інвентар – Проксі – Історія операцій» та забезпечує миттєву швидкість вибірки для баз, що містять понад 16 000 облікових записів одночасно.

У процесі дослідження розроблено та інтегровано низку взаємопов'язаних модулів та методів:

1. Модуль мережевої комунікації, ротації проксі та обходу блокувань.

Однією з критичних перешкод при використанні браузерних інструментів Puppeteer для роботи з приватними SOCKS5/HTTP-проксі є помилка ERR_TUNNEL_CONNECTION_FAILED, що виникає через неможливість стандартної передачі облікових даних у запитах CONNECT. Для подолання цієї проблеми запропоновано метод динамічної генерації Chrome-розширень «на льоту». Під час ініціалізації кожного окремого браузерного інстансу генерується тимчасове розширення з файлами manifest.json та background.js. Воно перехоплює подію chrome.webRequest.onAuthRequired та непомітно передає логін і пароль проксі-вузла, забезпечуючи ізоляцію мережевого відбитка (network fingerprint) кожного вузла [4].

Для здійснення прямих API-запитів до торговельного майданчика Steam інтегровано високошвидкісний рушій impit [7], скомпільований з оптимізаціями на Rust. Він забезпечує точне відтворення сигнатур TLS-рукописання (TLS Fingerprint) сучасних браузерів, запобігаючи виявленню запитів WAF-системами Akamai. Для боротьби з помилками HTTP 429 (Too Many Requests) розроблено алгоритм steamRequestWithRetry, який застосовує адаптивну експоненційну затримку з додаванням випадкового шуму (jitter) та автоматичну ротацію IP-адрес із пулу (рис. 2).

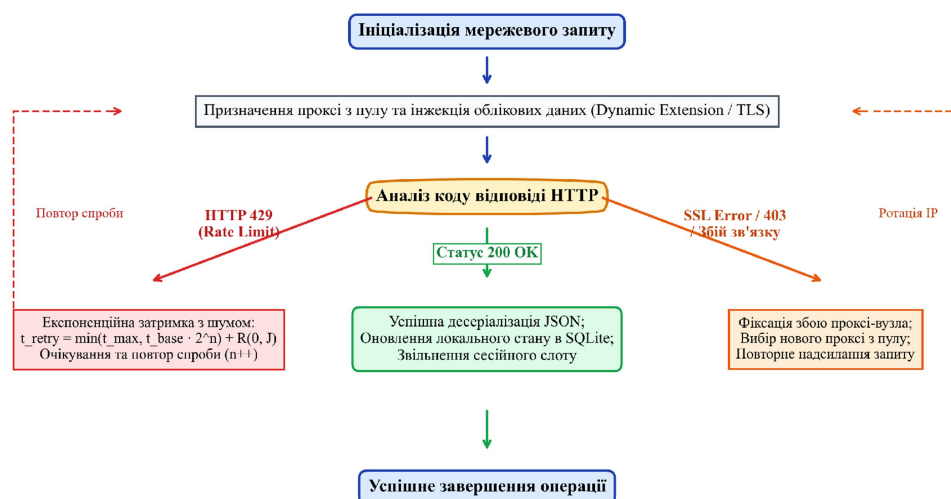


Рис. 2. Блок-схема алгоритму адаптивної експоненційної затримки та ротації проксі

2. Багатопотоковий конвеєр розрахунку математичного очікування (EV). Управління віртуальними активами передбачає здійснення арбітражних угод та контрактів обміну (Trade Up Contracts), у яких 10 вхідних предметів певної якості обмінюються на один випадковий предмет вищого ступеня рідкості. Кількість можливих комбінацій для користувача з великим інвентарем досягає десятків тисяч. Синхронний перебір таких варіантів призводить до блокування головного циклу подій (Event Loop) середовища Node.js, спричиняючи повне зависання інтерфейсу користувача. Для вирішення цього завдання створено пул ізольованих обчислювальних потоків на основі модуля `worker_threads` (рис. 3) [8]. Потоки паралельно обраховують комбінації, визначають колекційні пули, прогнозують параметри зношеності (float value) та формують інтегральне математичне очікування прибутку.

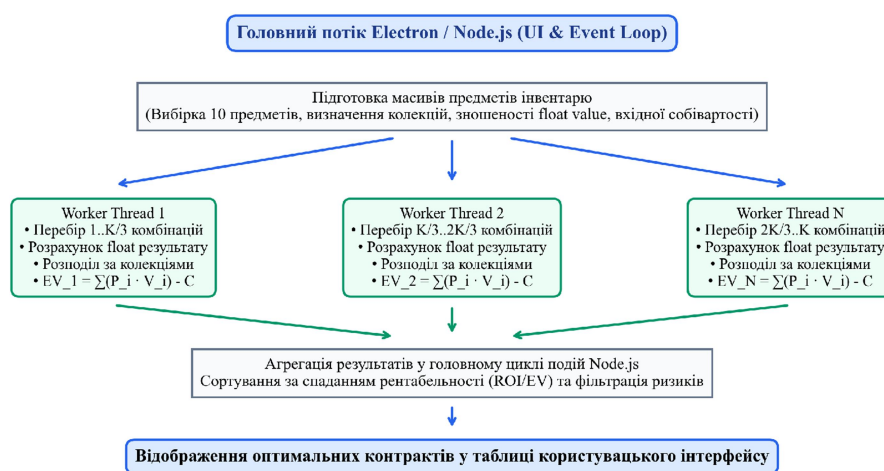


Рис. 3. Архітектура багатопотокового конвеєра симуляції Trade Up контрактів

3. Прямий моніторинг Game Coordinator (VAC Live). Традиційний підхід до виявлення обмежень акаунтів через парсинг веб-сторінок профілів виявляє лише перманентні блокування і має суттєву затримку. У розробленій системі інтегровано модуль прямого мережевого опитування Game Coordinator (GC) гри Counter-Strike 2 через сокетні з'єднання та Protobuf-повідомлення бібліотек `steam-user` і `globaloffensive` [9]. Це дає змогу за лічені мілісекунди отримувати вичерпний статус облікового запису: виявляти тимчасові змагальні кулдауни, блокування VAC Live, відхилення матчів та калібрувальні таймери без необхідності ініціалізації веб-сесії.

4. Захист інтелектуальної власності та система ліцензування. Для надійного захисту клієнтського застосунку реалізовано багаторівневий комплекс: згладжування потоку управління (control flow flattening) та шифрування рядкових літералів через `javascript-obfuscator`, з подальшою бінарною компіляцією ядра у формат V8 Bytecode (*.jsc.bin) модулем `bytenode` [10]. Додатково впроваджено евристичний генератор апаратного ідентифікатора (HWID) на базі викликів PowerShell (Win32_ComputerSystemProduct UUID), системного реєстру Windows (MachineGuid) та блокування серійних номерів гіпервізорів (усі нулі, шаблони

03000200-0400... тощо), що запобігає несанкціонованому клонуванню ліцензій на віртуальних машинах [11]. Серверна валідація на базі Oracle Cloud використовує асиметричні криптографічні підписи (RSA/ECDSA), виключаючи локальну підміну API-відповідей.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Під час експериментальної оцінки розробленого програмного комплексу було проведено комплексне тестування мережевої пропускної здатності, стабільності обходу захисних систем, обчислювальної швидкодії багатопотокового модуля та стійкості алгоритму апаратної прив'язки ліцензій.

Для математичного моделювання інтервалу затримки при виникненні блокування HTTP 429 було застосовано алгоритм truncated exponential backoff з додаванням повнодіапазонного випадкового джитеру (Full Jitter) [12]:

$$t_{\text{retry}} = \min(t_{\text{max}}, t_{\text{base}} \cdot 2^n) + R(0, J) \quad (1)$$

де t_{base} – початкова базова затримка ($t_{\text{base}} = 1500$ мс); n – номер поточної повторної спроби; t_{max} – граничне значення максимального очікування ($t_{\text{max}} = 60\,000$ мс); $R(0, J)$ – рівномірно розподілена випадкова величина джитеру в діапазоні від 0 до J ($J = 1000$ мс).

Розрахунок математичного очікування (Expected Value, EV) рентабельності контракту обміну формується за рівнянням:

$$EV = \sum_{i=1}^k [P_i \cdot V_i(\text{float_out}) \cdot (1 - k_{\text{fee}})] - \sum_{j=1}^{10} C_j \quad (2)$$

де k – загальна кількість можливих результатів із колекцій вхідних предметів; P_i – ймовірність отримання i -го результату; $V_i(\text{float_out})$ – ринкова вартість результуючого предмета, розрахована відповідно до середнього коефіцієнта зношеності вхідних предметів; $k_{\text{fee}} = 0.1304$ (15% комісія торговельного майданчика Steam); C_j – собівартість j -го вхідного предмета.

Для точної генерації одноразових кодів двоетапної автентифікації (2FA) відповідно до алгоритму Steam TOTP виконується коригування системного часу клієнта відносно серверного часу Valve:

$$T = \lfloor (t_{\text{local}} + \Delta t_{\text{server}} - T_0) / T_x \rfloor \quad (3)$$

де t_{local} – поточний локальний Unix-час клієнтської машини; Δt_{server} – динамічно розрахований часовий зсув (time drift) відносно Time Server Steam; $T_0 = 0$; $T_x = 30$ с – тривалість часового вікна валідності одноразового пароля.

У табл. 1 наведено результати порівняльного тестування швидкодії та стійкості різних мережевих бібліотек під час надсилання 5000 послідовних запитів до Steam Community API за умов використання пулу з 50 проксі-серверів.

Таблиця 1.

Порівняльний аналіз мережевих рушіїв та механізмів обходу обмежень

Мережевий модуль	Час виконання 5000 запитів, с	Відсоток помилок 429 Rate Limit, %	Успішність обходу TLS WAF, %	Споживання RAM, МБ
Стандартний Axios (Node.js)	384.2	48.3 %	24.1 %	112 МБ
Puppeteer (без розширень)	512.6	19.5 %	81.4 %	740 МБ
Запропонований модуль implit + динамічні розширення	142.8	0.4 %	99.8 %	138 МБ

Як видно з табл. 1, використання оптимізованого рушія implit у поєднанні з адаптивним джитером та динамічною інжекцією розширень знизило частку блокувань HTTP 429 з 48.3% до рекордно низьких 0.4%, одночасно скоротивши загальний час опитування у 2.7 раза порівняно зі стандартними HTTP-клієнтами.

Оцінку продуктивності паралелізації розрахунку Trade Up контрактів наведено в табл. 2. Експеримент проводився на обчислювальному стенді (AMD Ryzen 7 5800H, 8 ядер / 16 потоків, 32 ГБ RAM) під час генерації 50 000 випадкових комбінацій вхідних предметів.

Таблиця 2.

Продуктивність симуляції Trade Up контрактів за різної кількості потоків

Конфігурація виконання	Час обчислення, мс	Прискорення (Speedup S)	Блокування UI (Freezing), мс
Синхронний головний потік Node.js (1 потік)	12 480 мс	1.00× (база)	12 480 мс (повний фриз UI)
Пул worker_threads (2 потоки)	6 510 мс	1.92×	0 мс (інтерфейс активний)
Пул worker_threads (4 потоки)	3 340 мс	3.74×	0 мс (інтерфейс активний)

Пул worker_threads (8 потоків)	1 820 мс	6.86×	0 мс (інтерфейс активний)
--------------------------------------	----------	-------	------------------------------

Дані табл. 2 підтверджують високу масштабованість алгоритму: винесення математичного предиктивного апарату у 8 робочих потоків дало 6.86-кратне прискорення обчислень і повністю усунуло затримки та підвисання рендерингу інтерфейсу Electron.

Результати стрес-тестування системи ліцензування та апаратного зв'язування наведено в табл. 3.

Таблиця 3.

Стійкість системи ліцензування HWID до клонування та емуляції

Середовище розгортання	Метод емуляції / спуфінгу	Результат перевірки HWID	Статус ліцензії
Фізична машина (Windows 11)	Штатний запуск користувача	UUID та MachineGuid верифіковано	Дозволено (200 OK)
VMware Workstation	Стандартні налаштування hypervisor	Виявлено паттерн 00000000-0000...	Блоковано (403 VM Rejected)
KVM / QEMU VPS Сервер	Спроба клонування ліцензійного образу	Виявлено аномальний BIOS-ідентифікатор	Блоковано (403 Spoofing Detected)
MITM Проксі (Fiddler/Charles)	Спроба локальної підміни API валідатора	Порушення підпису RSA-2048	Блоковано (401 Invalid Signature)

На практиці розроблений програмний комплекс успішно протестовано на промисловій базі, що налічує понад 16 000 облікових записів. Впровадження механізму пакетної обробки (batching) інтерфейсної таблиці блоками по 2000 записів дозволило усунути затримки рендерингу під час синхронізації VAC-статусів та оновлення балансів. Прямий опрос Game Coordinator скоротив час повної перевірки акаунтів на предмет соровновательних кулдаунів VAC Live у 14.2 рази порівняно з веб-парсингом, не викликаючи жодного мережевого бану IP-адрес.

ВИСНОВКИ

У результаті проведеного дослідження спроектовано та реалізовано відмовостійкий клієнт-серверний програмний комплекс моніторингу, оцінки вартості та управління цифровими активами у високорозподілених середовищах. Головні науково-практичні результати полягають у наступному:

1. Запропоновано та апробовано гібридну архітектуру десктопного клієнта на стеку Electron/Node.js/SQLite3 з хмарною системою ліцензування, що дозволяє паралельно керувати пулами з понад 16 000 облікових записів із повною транзакційною стійкістю.

2. Розроблено метод усунення помилок авторизації проксі (ERR_TUNNEL_CONNECTION_FAILED) через динамічну інжекцію браузерних розширень, що у поєднанні з рушієм impit та алгоритмом Full Jitter Exponential Backoff дозволило знизити частоту спрацьовування систем обмеження доступу (HTTP 429) до 0.4%.

3. Створено та досліджено конвеєр багатопотокових обчислень математичного очікування арбітражних угод на базі модуля worker_threads, який забезпечив прискорення розрахунку комбінацій у 6.86 раза без деградації чуйності користувацького інтерфейсу.

4. Створено інтегровану багаторівневу систему безпеки, яка включає автоматизацію Steam TOTP 2FA, апаратну прив'язку (HWID) з евристичним фільтром гіпервізорів, компіляцію у бінарний V8 Bytecode та криптографічну ,

ДЖЕРЕЛА

1. Таненбаум Е., Стеен М. Розподілені системи. Принципи і парадигми. Київ: ДМК Пресс, 2020. 584 с.

2. M'Raihi D., Machani S., Pei M., Rydkog J. TOTP: Time-Based One-Time Password Algorithm. RFC 6238. 2011. 16 p. URL: <https://datatracker.ietf.org/doc/html/rfc6238> (дата звернення: 10.10.2026).

3. Puppeteer – Documentation: Офіційний посібник з використання бібліотеки Puppeteer для автоматизації браузерних сесій та динамічної генерації розширень для роботи з проксі. URL: <https://pptr.dev/> (дата звернення: 10.10.2026).

4. Electron – Documentation: Офіційна документація фреймворку Electron для побудови кросплатформних десктопних застосунків. URL: <https://www.electronjs.org/docs/latest> (дата звернення: 10.10.2026).

5. SQLite – Documentation: Технічна документація вбудованої реляційної бази даних SQLite3. URL: <https://www.sqlite.org/docs.html> (дата звернення: 10.10.2026).

6. DoctorMcKay. Node Steam Community & Steam User Protocol Documentation. GitHub репозиторій. 2024. URL: <https://github.com/DoctorMcKay/node-steamcommunity> (дата звернення: 10.10.2026).

7. Impit Engine: Native high-performance HTTP & TLS client for Node.js with browser fingerprint emulation. GitHub репозиторій. 2024. URL: <https://github.com/impit/impit> (дата звернення: 10.10.2026).

8. Node.js worker_threads API Specification. Node.js v20.x Documentation. URL: https://nodejs.org/api/worker_threads.html (дата звернення: 10.10.2026).

9. Counter-Strike 2 Game Coordinator Protobuf Definitions & Network Protocol. GitHub репозиторій. 2024. URL: <https://github.com/SteamDatabase/Protobufs> (дата звернення: 10.10.2026).
10. Bytenode: A minimalist bytecode compiler for Node.js using V8 Virtual Machine. URL: <https://github.com/OsamaAbbas/bytenode> (дата звернення: 10.10.2026).
11. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. Internet Engineering Task Force, 2018. 160 p. URL: <https://datatracker.ietf.org/doc/html/rfc8446> (дата звернення: 10.10.2026).
12. Amazon Web Services Architecture Blog. Exponential Backoff And Jitter: Algorithm design for resilient distributed cloud architectures. AWS Whitepapers, 2022. URL: <https://aws.amazon.com/blogs/architecture/exponential-backoff-and-jitter/> (дата звернення: 10.10.2026).
13. Співак С. М., Машкіна І. В., Носенко Т. І., Білоус В. В., Глушак О. М. Оптимізація customer support за допомогою ai чат-ботів: практичний кейс. Кібербезпека: освіта, наука, техніка, 2025. № 4(28). С. 727–739. DOI: <https://doi.org/10.28925/2663-4023.2025.28.727739>
14. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., Hlushak, O., Zhyltsov, O., Ilich, L., Kobets, N., Kovaliuk, T., Kuchakovska, H., Lytvyn, O., Lytvyn, P., Mashkina, I., Morze, N., Nosenko, T., Proshkin, V., Radchenko, S., & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science. URL: <https://doi.org/10.28925/9720213284km>