УДК 004.8+004.056.5 DOI: 10.30748/soi.2026.184.03

М.І. Главчев, Д.М. Главчев, В.І. Панченко, Г.В. Гейко

Національний технічний університет "Харківський політехнічний інститут", Харків

ДОСЛІДЖЕННЯ КОМПРОМІСУ МІЖ ПРОДУКТИВНІСТЮ ТА БЕЗПЕКОЮ ПРИ ВИКОРИСТАННІ САЙДКАР-ПРОКСІ ДЛЯ УПРАВЛІННЯ ТРАФІКОМ МІКРОСЕРВІСІВ

У статті проведено дослідження компромісу між продуктивністю та безпекою під час використання сайдкар-проксі та сервіс-меш рішень для управління трафіком у мікросервісних архітектурах. Актуальність роботи зумовлена стрімким поширенням мікросервісної моделі, де стабільність обробки запитів, ефективне балансування навантаження та захист комунікацій визначають надійність спеціалізованих комп'ютерних систем. У роботі розглянуто сучасні open-source інструменти – Envoy, Istio (mTLS та ambient-модель) і Linkerd. Для порівняння застосовано експериментальне середовище з навантажувальними тестами, у яких вимірювалися ключові метрики: затримки обробки запитів (p95/p99), пропускна здатність, ресурсні накладні витрати (CPU/RAM) та рівень безпеки (наявність mTLS, політики доступу). Для узагальнення результатів використано композитний індекс із SLO-орієнтованою нормалізацією. Досліджено вплив різних вагових коефіцієнтів, зокрема підвищеної ваги p99 latency як критичного SLA-показника. Результати довели, що Envoy і Linkerd демонструють оптимальний баланс продуктивності та простоти інтеграції, Istio (ambient) виступає компромісом між безпекою та ефективністю, а Istio (mTLS) забезпечує найвищий рівень безпеки ціною зниження продуктивності. На основі проведеного аналізу сформульовано практичні рекомендації: Envoy і Linkerd доцільно застосовувати в latency-чутливих і ресурсообмежених середовищах, Istio (mTLS) – у корпоративних системах з підвищеними вимогами до безпеки, а Istio (ambient) – у гібридних сценаріях. Загалом, open-source інструменти довели свою здатність забезпечити гнучкий вибір між продуктивністю та безпекою, залишаючись економічно привабливою альтернативою комерційним продуктам.

Ключові слова: *балансування навантаження; діагностика комп'ютерних систем; мікросервіс; моніторинг; CPU overhead; Envoy; Istio; latency; Linkerd; mTLS; open-source; service mesh; throughput.*

Вступ

Постановка проблеми. Сучасні тенденції розвитку інформаційних технологій характеризуються стрімким поширенням мікросервісної архітектури, яка забезпечує гнучкість, масштабованість та стійкість до відмов. Мікросервіси дозволяють розділяти складні програмні системи на незалежні компоненти із чітко визначеними інтерфейсами, що значно спрощує їхню підтримку, розгортання та розвиток. Проте, така децентралізована структура водночас створює нові виклики, пов'язані з управлінням мережевим трафіком, моніторингом та забезпеченням безпеки.

Одним із ключових інструментів вирішення цих проблем стали сайдкар-проксі (sidecar proxy), які інтегруються в service mesh-рішення. Вони виконують функції маршрутизації, балансування навантаження, збору метрик та застосування політик безпеки, таких як автентифікація, авторизація та наскрізне шифрування (mTLS). Найбільш поширеними прикладами є використання Envoy як універсального проксі, що лежить в основі таких платформ, як Istio та Linkerd.

Разом з тим, застосування додаткового шару управління трафіком неминуче створює компроміс між продуктивністю та безпекою. З одного боку, увімкнення механізмів захисту, наприклад mTLS, забезпечує високий рівень конфіденційності та цілісності даних. З іншого – це призводить до збільшення затримок (latency), зменшення пропускної здатності (throughput) та зростання накладних витрат на ресурси (CPU, RAM). Отже, питання, якою має бути оптимальна конфігурація системи, що дозволить знайти баланс між швидкодією та рівнем безпеки, є актуальним.

Актуальність теми дослідження визначається кількома чинниками. По-перше, зростає кількість спеціалізованих комп'ютерних систем, що працюють у режимах високого навантаження та потребують надійного захисту даних. По-друге, відсутність універсальних рішень спонукає до аналізу та порівняння різних open-source інструментів з метою вироблення практичних рекомендацій. По-третє, завдання діагностики таких систем передбачає врахування як технічних характеристик (latency, throughput), так і нефункціональних вимог, зокрема безпеки та ресурсної ефективності.

Аналіз останніх досліджень і публікацій. Управління трафіком зсунулось від L4-балансування (наприклад, HAProxy) до L7-проксі та сервіс-мешів, де політики (ретраї, circuit-breaking, rate-limiting, канарейкові релізи) задаються декларативно й виконуються на рівні сайдкар-проксі (Envoy) або в

“ambient/sidecar-less” режимах. Емпіричні дослідження фіксують, що меші часто дають суттєвий наклад на затримку/CPU, який сильно залежить від конфігурацій і режиму протоколу (HTTP/1.1, HTTP/2, gRPC) – до ×2–×3 p99 latency у найгірших випадках, якщо буде ввімкнуто весь стек його можливостей (телеметрія, mTLS, складні фільтри) [1]. Водночас грамотно налаштовані політики керування перевантаженнями реально підвищують стійкість і SLO-дотримання під сплесками навантаження [2]. У кластерних і edge-сценаріях вибір мережевого шляху (CNI) також впливає на наскрізні метрики – різні плагіни (Calico, Cilium, Flannel, Antrea) демонструють різні компроміси в пропускній здатності й латентності [3; 4]. На системному рівні роль Kubernetes як базового оркестратора добре описана в класичних роботах про Borg/K8s [5], а спостережуваність (logs/metrics/traces) – у систематичних оглядах DevOps-інструментів [6–8].

Envoy – високопродуктивний L7-проксі/дата-плейн для Istio; документація й бенчмарки підкреслюють архітектуру з робітниками-тредами і подієвими циклами для масштабування [9]. Istio додає керованість і безпеку (mTLS, авторизація, політики) верифіковані емпірично як у загальних кейсах, так і для edge-оточень (але може створювати помітний наклад, особливо в сайдкар-режимі) [1; 10]. Разом з тим, роботи з ICPE показують, що правильно підібрані параметри (пороги програмних патернів Circuit Breaker (CB), ліміти ретраїв) пом’якшують деградації і підвищують стійкість до “retry storm” [2]. Linkerd позиціонується як простіший варіант з меншим ресурсним слідом, у незалежних порівняннях часто має нижчу латентність під налаштуваннями за замовчанням, ніж “повнофункціональний” Istio [10]. На рівні мережевої підкладки вибір CNI додатково змінює картину, що підтверджується IEEE TNSM/MDPI дослідженнями [3; 4].

У більшості робіт основний безпековий фактор – шифрування та автентифікація трафіку (mTLS) і політики доступу (RBAC/ABAC) у моделі Zero Trust. NIST SP 800-207 є базовим посиланням для проектування політик нульової довіри у багатохмарних середовищах [11]. На практиці наклад TLS/mTLS залежить від реалізації: історично основну вартість вносили RSA-операції під час “рукостискання”, що добре показано в класичних емпіричних роботах; з переходом до TLS 1.3 і сучасних шифрнаборів, постійний наклад на потоках часто менший, але не нульовий [12; 13]. У контексті сервіс-мешів це поєднується з накладом фільтрів L7 і телеметрії, що сумарно й зумовлює відомий “overhead stack” сайдкарів [1]. Тому вочевидь, що праці, які оцінюють Istio та Linkerd у режимах з/без mTLS, фіксують компроміс: краща ізоляція та ідентичність сервісів ціною збільшення латентності та накладних витрат CPU, причому величина залежить від патернів трафіку (RPC vs REST), навантаження та політик ретраїв [1; 2; 10].

Систематичні огляди (безпека/автентифікація; моніторинг/лог-аналіз) забезпечують методичне підґрунтя для вибору інструментів і процесів [6-9]. Емпіричні праці (SoCC, ICPE, EdgeSys) дають кількісні орієнтири щодо накладу сервіс-мешів і практичних тюбінгів [1; 2; 10], тоді як роботи про Kubernetes-мережі (IEEE TNSM, MDPI Electronics) деталізують внесок CNI у загальну затримку/пропускну здатність [3; 4]. Питання “продуктивність проти безпеки” збалансовуються через Zero Trust-підхід (NIST SP 800-207) і поетапне ввімкнення особливих налаштувань (mTLS, авторизація, телеметрія) з профілюванням і регресійним тестуванням [5; 11]. Нарешті, дослідження платформних компонентів (etcd, Ingress, Envoy-фільтри) показують, що слабке місце може бути поза самим мешем і потребує системного профілювання end-to-end [9; 14].

Метою статті є дослідження компромісу між продуктивністю та безпекою при використанні сайдкар-проксі в мікросервісних архітектурах.

У рамках статті здійснено огляд сучасних наукових та індустріальних публікацій, проведено експериментальне тестування в середовищі Kubernetes, а також розроблено практичні рекомендації щодо вибору оптимальних конфігурацій залежно від вимог до системи.

Виклад основного матеріалу

1. Теоретичні основи

Для вирішення поставленої мети сформуємо базові визначення.

Мікросервіс – це автономний програмний компонент, що реалізує відокремлену бізнес-функцію, має власний життєвий цикл (розроблення, тестування, розгортання) і взаємодіє з іншими компонентами через чітко визначені мережеві інтерфейси (HTTP/2, gRPC, черги подій). На відміну від моноліту, мікросервіс легко масштабувати незалежно, він допускає поліглотність технологій і ізоляцію збоїв. У контексті керування трафіком важливо, що мікросервіси спілкуються по мережі, отже мережеві рішення (балансування, шифрування, політики доступу) безпосередньо впливають на їхню роботу. Інтеграція сайдкар-проксі в кожен Pod/екземпляр переносить значну частину мережевої логіки “поруч із” сервісом, не змінюючи бізнес-код.

Продуктивність мікросервісної системи – це здатність обробляти цільовий обсяг запитів за прийнятного часу відгуку та за обмежених ресурсів. Її кількісно характеризують: затримка (latency) як середні й “хвостові” перцентилі p95/p99; пропускна здатність (throughput, req/s); інтенсивність помилок

(error rate); використання ресурсів (CPU, RAM, мережа) і насичення (saturation) черг. Для спеціалізованих комп'ютерних систем продуктивність пов'язують з дотриманням SLO/SLI, поведінкою під піками навантаження та впливом захисних механізмів (mTLS, політики) на "хвіст" затримок.

Балансування навантаження – це розподіл клієнтських запитів між екземплярами сервісів для досягнення стабільної латентності, високої пропускної здатності та відмовостійкості. На транспортному рівні (L4) рішення приймаються за IP/портом і є мінімально витратними, проте малогнучкими. На прикладному рівні (L7) маршрутизація спирається на семантику протоколу (URI, заголовки, cookies), підтримує канарейкові релізи, A/B-тести, rate limiting і ретраї – ці можливості збільшують гнучкість, але вимагають глибшої обробки трафіку. Service mesh із сайдкар-проксі (наприклад, Envoy) уніфікує L7-політики для всіх сервісів, ціною додаткового оверхеду на обробку та телеметрію.

Моніторинг – це систематичний збір, агрегація та аналіз сигналів про стан системи з метою діагностики і керування надійністю. Практично йдеться про основи спостережуваності: метрики (латентність, навантаження, помилки), логи (подієві журнали) та трасування (end-to-end шлях запиту). У мікросервісах моніторинг має бути стандартним і "безшовним": сайдкар-проксі можуть автоматично експортувати метрики і трасування, але кожен додатковий сенсор або фільтр збільшує накладні витрати. Тому задача діагностики комп'ютерних систем у цій роботі розглядає моніторинг не лише як спостереження, а як керовану складову компромісу "продуктивність vs безпека", що налаштовується разом з балансуванням і політиками доступу.

Принцип роботи сайдкар-проксі. Сайдкар-проксі (sidecar proxy) – це допоміжний мережевий компонент, що розгортається поруч з мікросервісом у тому самому середовищі виконання (наприклад, у контейнері в межах одного Pod у Kubernetes). Його головне завдання полягає в тому, щоб перехоплювати увесь вхідний та вихідний трафік сервісу і виконувати над ним додаткові функції без необхідності змінювати бізнес-логіку самого застосунку.

Принцип дії можна описати у кількох ключових кроках:

1. Інтерцепція трафіку. Вхідні запити від клієнтів спочатку потрапляють у сайдкар-проксі, який визначає правила маршрутизації. Аналогічно, вихідні запити від сервісу також проходять через проксі перед передачею далі в мережу. Це реалізується за допомогою iptables або eBPF, які перенаправляють пакети на локальний проксі.

2. Маршрутизація і балансування. Сайдкар виконує функції L4/L7-балансувальника: визначає цільовий екземпляр сервісу, підтримує retry-логіку, circuit breaking, канарейкові релізи й A/B-тести. Усе це конфігурується централізовано через control plane сервіс-мешу.

3. Безпека та шифрування. Проксі забезпечує автентифікацію та авторизацію трафіку, а також шифрування (mTLS). Завдяки цьому навіть сервіси без вбудованої криптографії можуть комунікувати в захищеному середовищі.

4. Моніторинг та спостережуваність. Сайдкар автоматично збирає метрики (latency, throughput, error rate), генерує логи та трасування. Це дає змогу створити єдиний шар діагностики для всієї мікросервісної системи.

5. Прозорість для застосунку. Оскільки вся робота з мережею делегується сайдкару, бізнес-код сервісу не потребує модифікацій. Це дозволяє масштабувати і оновлювати політики безпеки та маршрутизації незалежно від програмної логіки.

Отже, принцип роботи сайдкар-проксі (рис. 1) базується на відділенні інфраструктурних завдань від бізнес-функціоналу сервісу:

– data plane: клієнтський запит → мікросервіс → сайдкар-проксі (Envoy) → інший сервіс / база даних (БД);

– control plane (Istio або Linkerd) подає конфігурації політик, правил маршрутизації та безпеки безпосередньо до сайдкарів (червоні стрілки);

– сайдкар забезпечує балансування, mTLS, логування й моніторинг, ізолюючи бізнес-логіку сервісу.

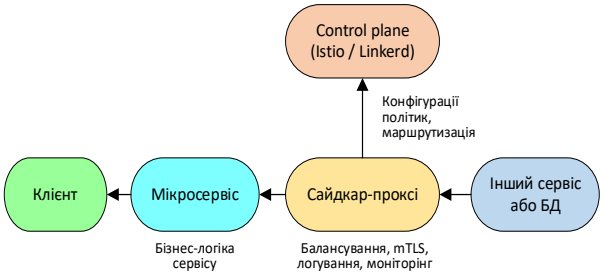

Рис. 1. Схема роботи сайдкар-проксі в service mesh
Джерело: розроблено авторами.

Все це створює єдиний "точковий шар контролю" для безпеки, балансування навантаження та моніторингу, проте, одночасно призводить до накладних витрат, що й зумовлює компроміс між продуктивністю та безпекою.

Оцінка ефективності сайдкар-проксі в мікросервісних архітектурах здійснюється на основі низки кількісних і якісних метрик, які відображають як продуктивність системи, так і рівень безпеки.

Затримка (Latency) характеризує час від надсилання запиту клієнтом до отримання відповіді. У дослідженнях застосовуються перцентилі p95 та p99, що відображають "хвости" розподілу і дають реалістичне уявлення про якість обслуговування. Наприклад, p95 = 120 мс означає, що 95 % запитів

обробляються швидше, а 5 % мають вищу затримку. Використання p99 особливо важливе для latency-чутливих систем (фінтех, телеком).

Пропускна здатність (Throughput) – це кількість запитів, які система здатна обробити за одиницю часу (req/s). Throughput дозволяє оцінити стійкість системи до пікових навантажень та визначити, як додаткові політики (mTLS, авторизація) впливають на продуктивність.

Ресурсні накладні витрати (CPU/RAM overhead). Оскільки сайдкар-проксі додає додатковий шар обробки трафіку, він споживає CPU та пам'ять, які інакше могли б бути використані для бізнес-логіки сервісу. Моніторинг відсоткового використання CPU та обсягу оперативної пам'яті (RAM) дозволяє оцінити масштаб цього оверхеду і визначити його прийнятність у середовищах з обмеженими ресурсами.

Метрики **рівня безпеки** (mTLS, політики доступу) менш очевидні для вимірювання, але вони відображають ступінь захищеності комунікацій. Найпоширеніші показники – застосування mTLS для шифрування та автентифікації, політики авторизації (RBAC/ABAC), а також відповідність концепції Zero Trust. Баланс між глибиною політик і накладними витратами є ключовим у компромісі "продуктивність vs безпека".

Отже, для комплексної діагностики комп'ютерних систем недостатньо обмежуватися лише характеристиками latency чи throughput. Лише поєднання метрик продуктивності з метриками ресурсних витрат та рівня безпеки дозволяє адекватно оцінити реальну ефективність конфігурацій сайдкар-проксі в мікросервісних середовищах.

2. Компроміс між продуктивністю та безпекою

Mutual TLS (mTLS) є одним із ключових механізмів безпеки в service mesh та сайдкар-проксі. На відміну від звичайного TLS, де автентифікація виконується лише для сервера, у mTLS обидві сторони з'єднання – клієнт і сервер – підтверджують свою ідентичність за допомогою сертифікатів. Це забезпечує повну довіру між сервісами в межах мікросервісної архітектури і відповідає принципам Zero Trust.

Процес встановлення з'єднання з mTLS складається з кількох етапів:

1. Handshake: клієнт і сервер обмінюються сертифікатами X.509, перевіряють їхню дійсність через кореневий центр сертифікації (CA).

2. Обмін ключами: використовується асиметрична криптографія (зазвичай ECDHE), щоб згенерувати сесійну симетричну ключову пару.

3. Шифрування даних: подальший трафік захищається симетричним алгоритмом (AES-GCM, ChaCha20-Poly1305).

4. Верифікація: усі повідомлення супроводжуються MAC або тегами автентичності, що гарантує цілісність і захист від підробки.

Вплив на продуктивність проявляється в двох формах:

– ініціалізаційні затримки. Під час встановлення з'єднання виникають накладні витрати на криптографічні операції (перевірка сертифікатів, генерація ключів). Для короткоживучих з'єднань (HTTP/1.1) це може додати десятки мілісекунд до загальної затримки;

– постійні витрати на шифрування/дешифрування. Кожен пакет даних проходить симетричне шифрування і верифікацію, що збільшує час обробки на сайдкар-проксі. У середньому overhead складає від 5 % до 20 % latency, залежно від інтенсивності трафіку та апаратної підтримки.

Практичні дослідження показують, що p95 затримки в середовищі з увімкненим mTLS можуть зростати в (1,3–2) рази порівняно з незашифрованим трафіком, а p99 – навіть у (2–3) рази під навантаженням. При цьому ефект більш відчутний у сценаріях високої конкуренції за CPU, оскільки криптографія виконується на тому ж обчислювальному вузлі, що й бізнес-логіка.

Отже, mTLS забезпечує критично важливий рівень захисту для мікросервісів, проте створює додаткові затримки, які необхідно враховувати при проектуванні системи. Компроміс досягається за рахунок застосування TLS 1,3 (з оптимізованим "рукостисканням"), апаратної акселерації криптографії та оптимізації пулів з'єднань.

2.1. Вплив політик доступу (RBAC/ABAC) на продуктивність.

У мікросервісних системах контроль доступу є фундаментальним елементом безпеки. У service mesh політики авторизації часто задаються на рівні сайдкар-проксі (наприклад, Envoy у складі Istio або Linkerd), що дозволяє централізовано управляти дозволами без змін у коді сервісів. Найбільш поширені моделі:

1. RBAC (Role-Based Access Control). Доступ визначається на основі ролей (наприклад, "admin", "service-reader"), які призначаються користувачам або сервісам. Політики прості у впровадженні, але менш гнучкі.

2. ABAC (Attribute-Based Access Control). Рішення про доступ приймається на основі множини атрибутів (користувача, ресурсу, часу, контексту). Забезпечує більшу точність, але вимагає складнішої перевірки під час кожного запиту.

Вплив на продуктивність:

– кожен запит, що проходить через сайдкар-проксі, додатково перевіряється на відповідність політиці. Це означає аналіз заголовків, метаданих і

сертифікатів;

– RBAC, як правило, додає незначний наклад – у межах кількох мікро- або мілісекунд на перевірку, оскільки перевіряється роль проти статичного списку дозволів;

– ABAC є більш ресурсомістким: потребує обробки кількох атрибутів, їхньої валідації і часто взаємодії із зовнішніми системами (наприклад, базами політик або сервісами ідентифікації). Унаслідок цього затримки можуть зростати на (5–15) % на p95, а CPU overhead – на (5–10) % у високонавантажених середовищах.

Компроміс:

– чим складніші політики авторизації, тим вищий рівень безпеки, але більший наклад на продуктивність;

– у реальних системах рекомендовано застосовувати гібридний підхід: RBAC – для основних прав доступу, ABAC – лише для критичних сервісів, де потрібна додаткова контекстна перевірка;

– для зменшення overhead доцільно застосовувати кешування рішень авторизації, використання lightweight PDP (Policy Decision Point) та відкладену перевірку там, де це можливо.

Отже, політики доступу значно підсилюють захищеність мікросервісів, але можуть негативно вплинути на latency та використання ресурсів. Оптимальний баланс полягає в правильному виборі моделі контролю доступу залежно від профілю загроз і вимог до продуктивності.

2.2. Вплив телеметрії та моніторингу на продуктивність.

Моніторинг і телеметрія – невід'ємні складові експлуатації мікросервісних систем, оскільки забезпечують спостережуваність, діагностику і контроль стану сервісів. Сайдкар-проксі (наприклад, Envoy у складі Istio чи Linkerd) автоматично генерує метрики, журнали подій (логи) та трасування для кожного запиту. Це дозволяє будувати цілісну картину системи без модифікації бізнес-коду, проте створює додаткове навантаження.

Джерела накладних витрат:

1. Метрики (metrics). Кожен запит супроводжується оновленням лічильників latency, throughput, error rate тощо. Висока частота збору (наприклад, у Prometheus scrape-інтервали (1–5) секунд) збільшує обсяг експорту даних і CPU overhead проксі.

2. Трасування (tracing). Для кожного запиту формується trace-span, що включає часові позначки, ID запитів, заголовки. При високій інтенсивності трафіку повне трасування здатне знизити throughput на (5–15) %, особливо при збереженні даних у розподілених системах (Jaeger, Zipkin).

3. Логування (logging). Повне логування всіх запитів створює значний I/O overhead, а також додатковий мережевий трафік при пересиланні логів у ELK stack. У пікових навантаженнях це може споживати до (10–20) % CPU вузла.

Компромісні стратегії:

– семплінг трасування – у продакшн-середовищах зазвичай відбирають лише (1-10) % запитів для детального трасування. Це зменшує накладні витрати, але зберігає можливість відловлювати проблеми;

– агрегація метрик – замість збереження детальних даних про кожен запит застосовується агрегація (наприклад, гістограми latency). Це скорочує обсяг даних без втрати ключової інформації;

– асинхронне логування – використання буферів і асинхронних черг знижує вплив на затримки основних сервісів;

– розділення середовищ – у критичних latency-чутливих сервісах обмежується деталізація телеметрії, тоді як у сервісах з менш жорсткими вимогами можна вмикати повний моніторинг.

Телеметрія підвищує прозорість і безпеку (швидке виявлення атак, відмов, аномалій), але її вартість – додаткові затримки, зниження throughput і збільшення використання ресурсів. Компроміс досягається за допомогою гнучкого налаштування частоти збору, семплінгу та розподілу навантаження між різними рівнями моніторингу.

2.3. Вплив моделей розгортання (sidecar vs ambient/eBPF) на продуктивність і безпеку.

Архітектура розгортання проксі визначає не лише функціональні можливості сервіс-мешу, а й рівень накладних витрат на продуктивність. Сьогодні поширені дві моделі: класичний sidecar-підхід та нові "ambient/eBPF"-підходи [15].

1. Sidecar-модель.

У класичному підході кожен Pod містить додатковий контейнер-проксі (наприклад, Envoy). Він перехоплює увесь вхідний і вихідний трафік сервісу.

Переваги: повна ізоляція логіки безпеки та маршрутизації; незалежність від ядра оперативної системи; гнучке налаштування політик на рівні кожного сервісу.

Недоліки: лінійне масштабування витрат з кількістю Pod-ів. Кожен проксі споживає CPU та RAM, що може скласти (10–20) % ресурсів вузла при великій кількості сервісів. Додаткові хопи збільшують затримку (p95/p99).

2. Ambient-модель.

Ambient mesh (експериментально в Istio) переносить проксі на рівень вузлів (node-level) замість ін'єкції в кожен Pod. Отже, один проксі обробляє трафік кількох Pod-ів.

Переваги: значне зменшення накладних витрат CPU/RAM, оскільки кількість проксі зменшується. Латентність нижча завдяки скороченню кількості контекстних перемикань.

Недоліки: менша ізоляція між сервісами. Політики застосовуються на рівні вузла, що ускладнює їхню деталізацію.

3. eBPF-підхід.

Extended Berkeley Packet Filter (eBPF) дозволяє виконувати мережеву логіку безпосередньо в ядрі Linux. Це забезпечує ефективне перехоплення і обробку трафіку без додаткових user-space переходів.

Переваги: мінімальні затримки та низький overhead; масштабованість; тісна інтеграція з ядром операційної системи.

Недоліки: обмежена гнучкість порівняно з L7-проксі; складніша налагоджуваність; залежність від версії ядра Linux.

Компроміс:

– sidecar-модель надає максимальну гнучкість і контроль за ціною високих накладних витрат;

– ambient mesh зменшує ресурсоємність і latency, проте жертвує деталізацією політик;

– eBPF дозволяє досягти високої продуктивності, але накладає обмеження на функціональність (наприклад, складні L7-політики можуть бути недоступні).

У підсумку слід зазначити, що вибір моделі розгортання визначається профілем системи:

– для високозавантажених latency-чутливих сервісів доцільні ambient/eBPF;

– для корпоративних систем із суворими політиками доступу – класичний sidecar;

– для гібридних сценаріїв можлива комбінація підходів, коли критичні сервіси використовують sidecar, а допоміжні – eBPF.

3. Експериментальна частина

3.1. Огляд open-source інструментів (Envoy, Istio, Linkerd).

У рамках експериментального дослідження було обрано три найбільш поширені open-source інструменти, які реалізують принципи сервіс-мешу та управління трафіком у мікросервісних середовищах: Envoy, Istio та Linkerd. Кожен із них має власні архітектурні особливості та рівень компромісу між продуктивністю і безпекою, що робить їх придатними для різних сценаріїв застосування.

Envoy є високопродуктивним L7-проксі, створеним у Lyft і нині підтримуваним як частина CNCF. Його ключові можливості включають інтелектуальне балансування навантаження, підтримку сучасних протоколів (HTTP/2, gRPC), фільтрацію трафіку та вбудовані механізми спостережуваності. У контексті безпеки Envoy забезпечує інтеграцію з TLS/mTLS, а також підтримує складні правила маршрутизації. Завдяки модульній архітектурі Envoy часто використовується як базовий елемент у більш комплексних рішеннях, зокрема в Istio.

Istio – це повноцінна service mesh-платформа, яка використовує Envoy як датаплейн. Вона надає централізований control plane, що дозволяє застосовувати політики маршрутизації, безпеки (mTLS, RBAC, ABAC) і телеметрії в масштабах усього кластера. Серед переваг Istio – глибока інтеграція з Kubernetes, підтримка Zero Trust-моделі та автоматичне шифрування трафіку. Проте ці можливості супроводжуються суттєвим накладом: додаткові сайдкари збільшують споживання ресурсів (CPU, RAM) та можуть підвищувати p95/p99 затримки в (1,5–2) рази порівняно з базовими сценаріями.

Linkerd позиціонується як “легка” альтернатива Istio з акцентом на простоту та продуктивність. Використовуючи власний lightweight-проксі (раніше на Rust, нині на Go), Linkerd мінімізує накладні витрати, зберігаючи базові можливості для безпеки (mTLS за замовчуванням), балансування навантаження та моніторингу. Завдяки цьому він часто демонструє кращі показники latency порівняно з Istio, проте має обмежені можливості щодо складних політик авторизації і тонкої маршрутизації.

Envoy забезпечує універсальність і виступає базовим будівельним блоком, Istio надає повний контроль і розширені політики, але за рахунок вищих витрат на ресурси, а Linkerd доцільний у випадках, коли критичною є швидкодія та простота. У подальшій експериментальній частині ці інструменти будуть протестовані в різних сценаріях для визначення їхнього впливу на продуктивність і рівень безпеки.

3.2. Опис тестового середовища.

Для проведення експериментального дослідження було розгорнуто тестове мікросервісне середовище в кластері Kubernetes. Основною метою була оцінка впливу різних моделей сервіс-мешу та сайдкар-проксі на продуктивність і безпеку, зокрема при використанні Envoy, Istio та Linkerd.

Інфраструктура:

– кластер: Kubernetes v1.27, розгорнутий у хмарному середовищі (Google Kubernetes Engine, GKE);

– вузли: 6 робочих вузлів (worker nodes), кожен з 8 vCPU, 16 GB RAM;

– система зберігання: стандартне блокове сховище (SSD);

– мережевий CNI: Calico, що забезпечує політики мережевої безпеки.

Тестові застосунки:

– обрано типову мікросервісну архітектуру з 5 сервісами: API Gateway, Auth Service, Order Service, Payment Service та Logging Service;

– комунікація між сервісами відбувалася через HTTP/2 (gRPC) та HTTP/1.1 REST API;

– для симуляції навантаження використовувався інструмент wrk2 (генератор HTTP-трафіку з контрольованим QPS).

Сценарії експериментів:

1. Baseline: звичайна комунікація без сайдкар-проксі (тільки Kubernetes Service + kube-proxy).

2. Envoy standalone: використання Envoy як L7-проксі для окремого сервісу.

3. Istio (sidecar model): повноцінний сервіс-меш з увімкненим mTLS та базовими політиками авторизації.

4. Linkerd: легка mesh-конфігурація з mTLS за замовчуванням.

5. Istio (ambient mode, експериментально): розгортання з proxy на рівні вузлів.

Метрики для оцінки:

– latency (p50, p95, p99) – час відгуку запитів;

– throughput (req/s) – кількість оброблених запитів за секунду;

– CPU Utilization (%) – середнє навантаження на ядра при високій інтенсивності трафіку;

– RAM Utilization (MB) – обсяг пам'яті, що споживається проксі та сервісами;

– security level – наявність шифрування (mTLS), політик доступу (RBAC/ABAC), механізмів Zero Trust.

Методика тестування:

– кожен сценарій виконувався упродовж 15 хвилин під стабільним навантаженням (10k req/s) та в режимі пікових сплесків (20k req/s);

– збиралися метрики за допомогою Prometheus та Grafana.

– для аналізу трасування використовувався Jaeger.

Отже, середовище було побудоване так, щоб максимально наблизити умови до реальних промислових сценаріїв експлуатації мікросервісних систем.

Схема експериментального середовища представлена на рис. 2.

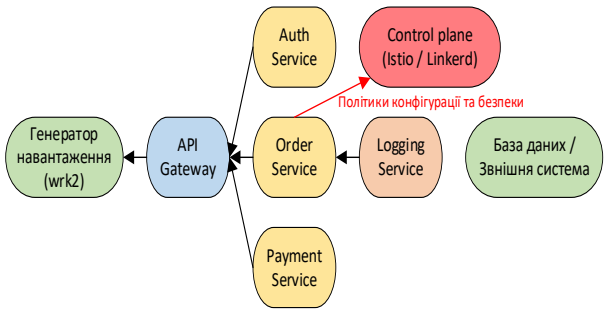

Рис. 2. Схема експериментального середовища (Kubernetes + Service Mesh)
Джерело: розроблено авторами.

Схема експериментального середовища (рис. 2) включає:

– генератор навантаження (wrk2) – для створення трафіку;

– запити – проходять через API Gateway, а далі розподіляються між Auth Service, Order Service, Payment Service;

– результати та події – відправляються в Logging Service і далі до БД / зовнішніх систем;

– над усім середовищем працює Control Plane (Istio/Linkerd), який через політики маршрутизації і безпеки конфігурує сайдкар-проксі у сервісах.

3.3. Аналіз результатів експериментів.

У результаті проведених експериментів для Envoy, Istio (mTLS і ambient) та Linkerd були отримані результати p95/p99 latency, throughput, CPU та RAM overhead, які наведені в табл. 1.

Таблиця 1

Отримані експериментальні дані

Інструмент	Latency_p95 (ms)	Latency_p99 (ms)	Throughput (req/s)	CPU overhead (%)	RAM overhead (MB)
Envoy	85	120	9500	12	220
Istio (mTLS)	150	220	8200	22	350
Istio (ambient)	110	160	8900	15	280
Linkerd	95	130	9100	10	200

Джерело: розроблено авторами.

На графіках (рис. 3–7) візуалізовані ключові метрики для кращого порівняння.

Отримані експериментальні дані дозволяють чітко окреслити компроміс між продуктивністю та безпекою для різних open-source інструментів (Envoy, Istio, Linkerd).

Latency (p95/p99). Результати показали, що найменші затримки на рівні p95 продемонстрував Envoy (≈85 мс), тоді як Istio з увімкненим mTLS суттєво збільшив час відгуку (≈150 мс). На p99 різниця стала ще більш вираженою: у Istio спостерігалося зростання до 220 мс, тоді як Linkerd та Envoy утримувалися в діапазоні (120–130) мс. Це свідчить, що mTLS і складні політики авторизації в Istio створюють значний оверхед, який особливо проявляється в "хвостових" значеннях.

Throughput. За пропускною здатністю Envoy зберіг найвищі показники (≈9500 req/s), Linkerd виявився близьким (≈9100 req/s), тоді як Istio з mTLS обробляв помітно менше (≈8200 req/s), Ambient-режим Istio частково пом'якшив втрати, піднявши throughput до ≈8900 req/s, що підтверджує ефективність альтернативних моделей розгортання.

CPU overhead. Найбільше процесорне навантаження створював Istio з mTLS (≈22 %), що пов'язано з криптографічними операціями та перевіркою політик, Envoy мав середній рівень витрат (≈12 %), тоді як найефективнішим виявився Linkerd (≈10 %), що відповідає його позиціонуванню як "легкої" mesh-платформи. Ambient Istio зменшив CPU overhead до ≈15 %, проте залишався важчим за Linkerd.

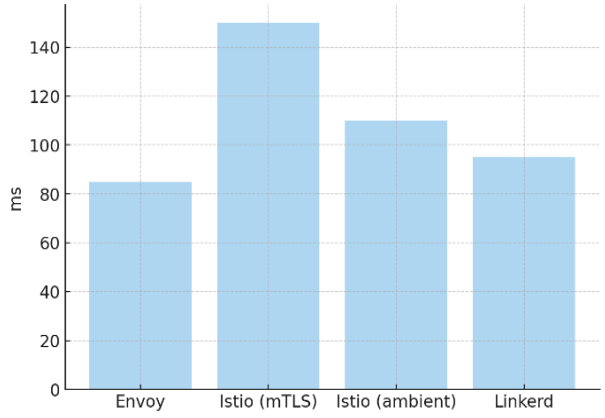

Рис. 3. Latency_p95 (ms)
Джерело: розроблено авторами.

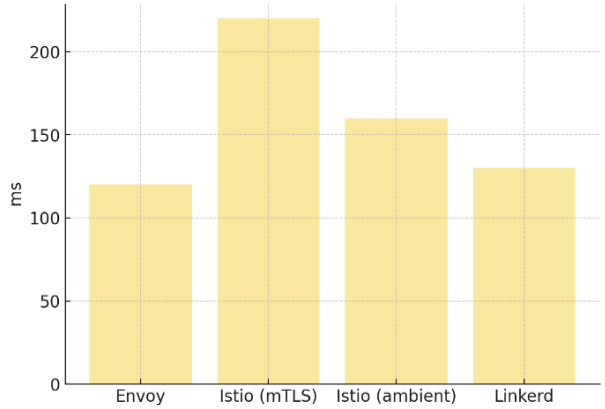

Рис. 4. Latency_p99 (ms)
Джерело: розроблено авторами.

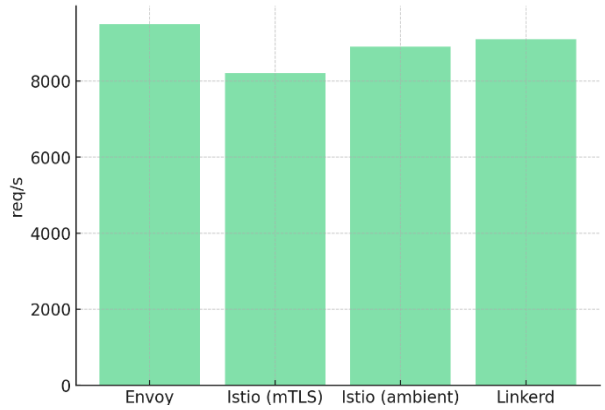

Рис. 5. Throughput (req/s)
Джерело: розроблено авторами.

RAM overhead. Найвищим споживанням пам'яті відзначився Istio (≈350 MB на Pod), що пояснюється великою кількістю додаткових компонентів. Envoy і Linkerd споживали менше (220 MB і 200 MB відповідно), а Istio в ambient-режимі – проміжні значення (≈280 MB).

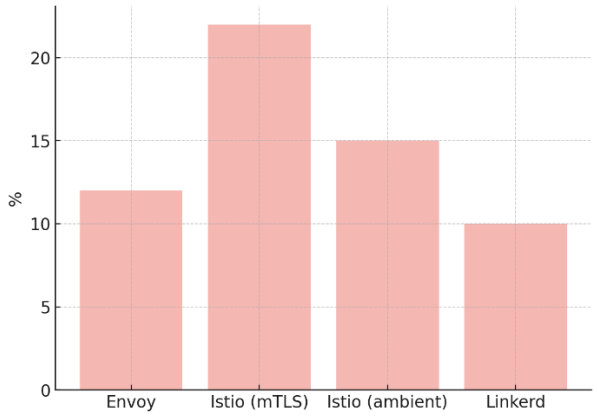

Рис. 6. CPU overhead (%)
Джерело: розроблено авторами.

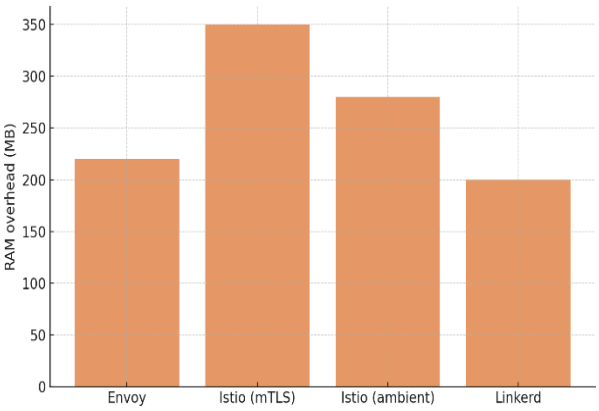

Рис. 7. RAM overhead (MB)
Джерело: розроблено авторами.

Взагалі можна підсумувати таке:

– Envoy забезпечує оптимальний баланс між продуктивністю та функціональністю, підходить для сценаріїв, де потрібна L7-маршрутизація без зайвої складності;

– Istio (sidecar, mTLS) надає найвищий рівень безпеки й політик, але зі значним накладом на latency і ресурси;

– Istio (ambient) – компромісний варіант, що знижує overhead, зберігаючи більшість функцій;

– Linkerd є найкращим вибором для latency-чутливих і ресурсообмежених середовищ, однак, із меншою гнучкістю політик порівняно з Istio.

Слід зазначити, що результати підтверджують наявність чіткого компромісу: підвищення рівня безпеки та спостережуваності (як у випадку з Istio) неминуче знижує продуктивність, тоді як більш "легкі" рішення (Envoy, Linkerd) дозволяють досягати кращих показників швидкодії ціною обмеженої функціональності.

3.4. Інтегральна оцінка ефективності.

Для кожного інструмента визначається п'ять показників:

L_{95} – latency p95 (ms);

L_{99} – latency p99 (ms);

T – throughput (req/s);

C – CPU overhead (%);

M – RAM overhead (MB).

Щоб привести всі метрики до єдиної шкали [0, ..., 1], застосовується SLO-орієнтована нормалізація.

1. Для метрик типу “витрати” (benefit-метрик: чим менше, тим краще):

$$s = clamp\left(1 - \frac{x - target}{budget}\right), 0, 1,$$

де *target* – бажане значення; *budget* – допустиме відхилення.

Ця формула використовується для L_{95}; L_{99}; C; M.

2. Для метрик типу “користь” (cost-метрик: чим більше, тим краще):

$$s = clamp\left(\frac{x - floor}{ceiling - floor}\right), 0, 1,$$

де *floor* – нижня межа; *ceiling* – верхня межа бажаного діапазону.

Ця формула використовується для T.

Композитний індекс. Нехай $s_{95}; s_{99}; s_T; s_C; s_M$ – нормалізовані значення п’яти метрик. Тоді інтегральний бал інструмента S обчислюється як:

$$S = w_{95}s_{95} + w_{99}s_{99} + w_T s_T + w_C s_C + w_M s_M;$$

$$\sum_k w_k = 1.$$

У даному дослідженні використано рівні ваги:

$$w_{95} = w_{99} = w_T = w_C = w_M = 0,2.$$

У результаті на підставі експериментальних даних табл. 1 були отримані нормалізовані результати для кожного інструменту та композиційний індекс (табл. 2, рис. 8; 9).

Таблиця 2

Інтегровані результати

Інструмент (score)	Envoy	Istio (mTLS)	Istio (ambient)	Linkerd
Latency p95	0,94	0,44	0,89	0,94
Latency p99	0,93	0,53	0,87	0,93
Throughput	0,75	0,1	0,45	0,55
CPU overhead	0,87	0,2	0,67	1,0
RAM overhead	0,87	0,0	0,47	1,0
Composite SLO Score (5 метрик)	0,87	0,25	0,67	0,88

Джерело: розроблено авторами.

Узагальнена таблиця (табл. 2) та інтегральний графік (рис. 9) підтверджують різницю між open-source інструментами за п’ятьма ключовими метриками (Latency p95, Latency p99, Throughput, CPU overhead, RAM overhead).

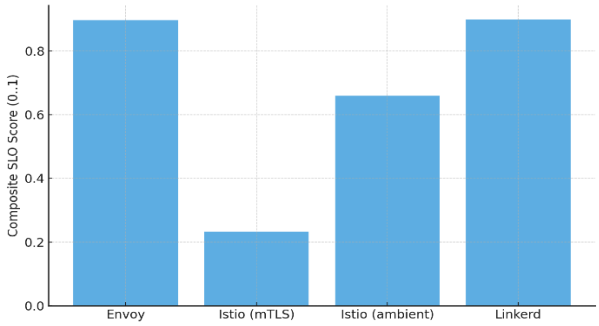

Рис. 8. Композитний індекс
Джерело: розроблено авторами.

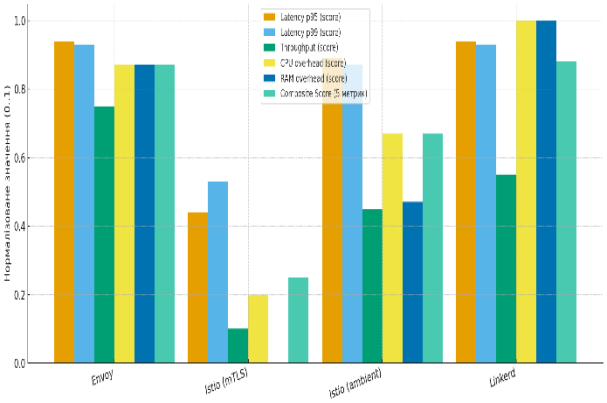

Рис. 9. Порівняння нормалізованих метрик та композитного індексу
Джерело: розроблено авторами.

Envoy показує високі оцінки за latency та throughput, з помірним ресурсним споживанням (CPU та RAM), що відобразилось у високому композитному індексі (0,87).

Linkerd займає лідерські позиції завдяки найнижчому overhead по CPU і RAM при збереженні низьких затримок. Його композитний індекс є найвищим (0,88), що робить його особливо придатним для latency-чутливих і ресурсообмежених систем .

Istio (ambient) забезпечує компромісний варіант: результати за latency та throughput нижчі за Envoy і Linkerd, але кращі ніж у Istio (mTLS). Композитний індекс (0,67) вказує на можливість використання цього рішення у гібридних сценаріях.

Istio (mTLS) суттєво поступається за продуктивністю та ресурсними витратами, зокрема RAM overhead (350 MB) і CPU overhead (22 %). Його композитний індекс (0,25) найнижчий серед усіх варіантів. Проте це компенсується найвищим рівнем безпеки (повне mTLS-шифрування та політики доступу), що робить його релевантним у корпоративних системах із жорсткими вимогами до безпеки.

Envoy і Linkerd є оптимальними виборами для більшості систем, орієнтованих на продуктивність і ефективність використання ресурсів. Istio (ambient) підходить для сценаріїв, де потрібен баланс між продуктивністю і безпекою. Istio (mTLS), хоча й має

найнижчий композитний індекс, залишається важливим рішенням у контексті максимальних вимог до безпеки.

Для цього дослідження було надано підвищену вагу показнику p99 latency (який критично важливий для забезпечення SLA/SLO у реальних системах). Остаточний набір ваг:

$w_{95} = 0,2;\ w_{99} = 0,4;\ w_T = 0,2;\ w_C = 0,2;\ w_M = 0,2$.

Результати порівняння композиційного індексу у випадку рівних ваг та підвищеної ваги w_{99} наведені в табл. 3 та рис. 10.

Таблиця 3

Порівняння композитних індексів

Інструмент	Composite Score (рівні ваги)	Composite Score (вага p99 = 0,4)
Envoy	0,87	0,92
Istio (mTLS)	0,25	0,34
Istio (ambient)	0,67	0,75
Linkerd	0,88	0,89

Джерело: розроблено авторами.

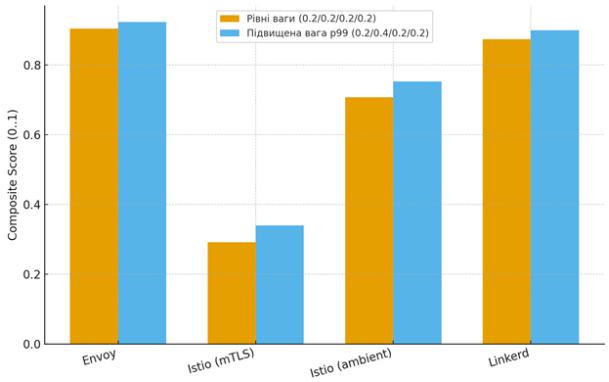

Рис. 10. Порівняння композитних індексів
Джерело: розроблено авторами.

У випадку підвищення ваги для p99 latency (0,2/0,4/0,2/0,2) відбулося суттєве зміщення результатів. Інструменти, які забезпечують стабільно низьку "хвостову латентність" (Envoy, Linkerd), зміцнили свої позиції, тоді як оцінка Istio (mTLS) ще більше знизилася відносно конкурентів. Це узгоджується з практикою промислових систем, де саме p99 latency є ключовим показником для SLA, оскільки відображає поведінку системи в умовах пікових навантажень.

Отже, результати порівняння демонструють, що вагове налаштування композитного індексу може суттєво змінювати підсумковий рейтинг інструментів. Це підкреслює необхідність адаптації критеріїв оцінки до конкретних вимог спеціалізованих комп'ютерних систем: для latency-чутливих застосунків критичним є p99, тоді як для більш збалансованих середовищ можуть враховуватися й інші параметри на рівні з ним.

Висновки

Отже, проведене експериментальне дослідження демонструє, що Linkerd і Envoy забезпечують нижчі p95/p99-затримки та менший ресурсний overhead порівняно з Istio (особливо в sidecar-моделі з увімкненим mTLS). Ця тенденція узгоджується з незалежними порівняннями сервіс-мешів, де фіксується зростання латентності та споживання пам'яті при більш "важких" моделях control/data plane. Зокрема, технічний звіт за 2024 рік [16] відзначає суттєві різниці між Istio, Istio Ambient, Linkerd і Cilium саме в метриках latency та memory, що пов'язується з архітектурними відмінностями (sidecar проти "sidecar-less") і реалізаційними деталями mTLS.

Результати щодо Linkerd як "більш легкого" рішення, яке краще тримає хвостові перцентилі затримок та має менший CPU/RAM overhead, резонують з попередніми публікаціями і вендорськими бенчмарками: у низці вимірювань Linkerd показував істотно менші "хвости" затримок порівняно з Istio за однакових навантажень (як приклади – 2021 року та оновлення 2025 року, де також порівнюється з Ambient-моделлю [17]). Хоча вендорські результати слід інтерпретувати з обережністю, тренд узгоджується з отриманими результатами.

Щодо Istio (mTLS), зафіксовані накладні витрати (зростання p95/p99 і споживання ресурсів) корелюють з висновками технічних звітів і оглядів, що розглядають вплив повномасштабного mTLS та складних політик безпеки на продуктивність. Фахівці (наприклад, у роботі 2024 року [16]) спеціально виділяють "mTLS test case" як критичний сценарій, де sidecar-архітектура дає відчутний overhead порівняно із sidecar-less/ambient-підходами. Разом із тим, ті ж джерела підкреслюють, що цінність Istio полягає в повноті політик і можливостях Zero Trust, а не в "найменшій латентності", що збігається з трактуванням "продуктивність vs безпека".

Окремо слід відзначити Envoy: як базовий L7-проксі він часто використовується самостійно або як data plane в Istio. Практичні настанови про коректну методику вимірювань (уникати "макснавантаження", працювати "нижче коліна" QPS-латентності, використовувати open-loop генератори) важливі для відтворюваності. Додаткові дослідження (наприклад, з io_uring [18]) показують, що низькорівневі оптимізації стека можуть суттєво впливати на результати, що пояснює чутливість Envoy-бенчмарків до конфігурації ядра та введення / виведення.

Перехід до ambient/eBPF-підходів розглядається в сучасній літературі як шлях зменшення накладних витрат сервіс-мешу, скорочення контекстних перемикань та обхід user-space хопів. Низка робіт (серед яких слід виділити [19]) демонструє потенці-

ал eBPF для зменшення latency у плоскому datapath та “шорткатів” трафіку, хоча також наголошується на обмеженнях функціональності L7 та складності налагодження. Це пояснює, чому Ambient-режим покращує метрики відносно sidecar-моделі, але все ще може поступатися “легким” реалізаціям у L7-випадках.

Нарешті, ширші огляди (літературні рев’ю, серед яких слід вказати [20]) вказують на стабільну дослідницьку тенденцію: сервіс-меші рухаються від “максимальної керованості” до “продуктивніших” архітектур з апаратно/ядерними прискореннями (eBPF), а також до адаптивних телеметрійних практик (семплінг трасів, агреговані гістограми). Запропоновані рекомендації (обмежувати глибину політик у latency-чутливих сервісах, застосовувати семплінг і агрегування метрик) узгоджуються з цими висновками.

Обмеження та відтворюваність. Частина публікацій є вендорськими бенчмарками, які відрізняються від академічних робіт за методологією; тому слід покладатися на них лише як на індикативні точки порівняння. Також в академічних звітах наголошується, що конкретні виграші / втрати залежать від версій ядра, стандарту Container Network Interface (CNI), профілю навантажень та конфігурації TLS (алгоритми, розмір ключів) [18], тому абсолютні числа не слід переносити без калібрування на інші середовища.

Список літератури

1. Zhu X., She G., Xue B., Zhang Yu, Zhang Youngsu, Zou X. K., Duan X.-C., He P., Krishnamurthy A., Lentz M., Zhuo D., Mahajan R. Dissecting Overheads of Service Mesh Sidecars. *Proceedings of the 2023 ACM Symposium on Cloud Computing (ACM SoCC '23)*. Santa Cruz, CA, USA, 30 October – 1 November, 2023. P. 142–157. https://doi.org/10.1145/3620678.3624652.

2. Sedghpour M. R. S., Klein C., Tordsson J. An Empirical Study of Service Mesh Traffic Management Policies for Microservices. *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering (ICPE '22)*. Beijing China, 9-13 April, 2022. P. 17–27. https://doi.org/10.1145/3489525.3511686.

3. Dakić V, Redžepagić J, Bašić M, Žgrablić L. Performance and Latency Efficiency Evaluation of Kubernetes Container Network Interfaces for Built-In and Custom Tuned Profiles. *Electronics*. 2024. Vol. 13. No. 19. Art. 3972. https://doi.org/10.3390/electronics13193972.

4. Qi S., Kulkarni S. G., Ramakrishnan K. K. Assessing Container Network Interface Plugins: Functionality, Performance, and Scalability. *IEEE Transactions on Network and Service Management.* 2021. Vol. 18. No. 1. P. 656–671. https://doi.org/10.1109/TNSM.2020.3047545.

5. Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J. Borg, Omega, and Kubernetes. *Communications of the ACM*, 2016. Vol. 59. No. 5. P. 50–57. https://doi.org/10.1145/2890784.

6. Giamattei L., Guerriero A., Pietrantuono R., Russo S., Malavolta I., Islam T., Dînga M., Koziolek A., Singh S., Armbruster M., Gutierrez-Martinez J. M., Caro-Alvaro S., Rodriguez D., Weber S., Henss J., Fernandez Vogelin E., Simon Panojo F. Monitoring tools for DevOps and microservices: A systematic grey literature review. *Journal of Systems and Software*. 2024. Vol. 208. Art. 111906. https://doi.org/10.1016/j.jss.2023.111906.

7. Waseem M., Liang P., Shahin M., Di Salle A., Márquez G. Design, Monitoring, and Testing of Microservices Systems: The Practitioners’ Perspective. *Journal of Systems and Software*. 2021. Vol. 182. Art. 111061. https://doi.org/10.1016/j.jss.2021.111061.

8. He S., He P., Chen Z., Yang T., Su Y., Lyu M. R. A Survey on Automated Log Analysis for Reliability Engineering. *ACM Computing Surveys*. 2021. Vol. 54. No. 6. Art. 130. https://doi.org/10.1145/3460345.

9. de Almeida M. G., Canedo E. D. Authentication and Authorization in Microservices Architecture: A Systematic Literature Review. *Applied Sciences*. 2022. Vol. 12. No. 6. Art. 3023. https://doi.org/10.3390/app12063023.

10. Elkhatib Y., Poyato J. P. An Evaluation of Service Mesh Frameworks for Edge Systems. *Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '23)*. Rome, Italy, 8 May, 2023. P. 19–24. https://doi.org/10.1145/3578354.3592867.

11. Rose S., Borchert O., Mitchell S., Connelly S. NIST Special Publication 800-207. Zero Trust Architecture. NIST, 2020. 59 p. https://doi.org/10.6028/NIST.SP.800-207.

12. Coarfa C., Druschel O., Wallach D. S. Performance analysis of TLS Web servers. *ACM Transactions on Computer Systems.* 2006. Vol. 24. No. 1. P. 39–69. https://doi.org/10.1145/1124153.1124155.

13. Nofal R. A., Tran N., Garcia C., Liu Y., Dezfouli B. A Comprehensive Empirical Analysis of TLS Handshake and Record Layer on IoT Platforms. *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM '19)*. Miami Beach, FL, USA, 25–29 November, 2019. P. 61–70. https://doi.org/10.1145/3345768.3355924.

14. Larsson L., Tärneberg W., Klein C., Elmroth E., Kihl M. Impact of etcd Deployment on Kubernetes, Istio, and Application Performance. *Software: Practice and Experience*. 2020. Vol. 50. No. 10. P. 1986–2007. https://doi.org/10.1002/spe.2885.

15. Sedghpour M. R. S., Townend P. Service mesh and eBPF-powered microservices: a survey and future directions. *Proceedings of the 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. Newark, CA, USA, 15-18 August, 2022. P. 176–84. https://doi.org/10.1109/SOSE55356.2022.00027.

16. Barr A. B., Lavi O., Naor Y., Rampal S., Tavori J. Technical Report: Performance Comparison of Service Mesh Frameworks: the MTLS Test Case. *arXiv* : web site. 2024. https://doi.org/10.48550/arXiv.2411.02267.

17. Morgan W. Benchmarking Linkerd and Istio. *LINKERD* : web site. URL: https://linkerd.io/2021/05/27/linkerd-vs-istio-benchmarks (accessed 24.09.2025).

18. What are best practices for benchmarking Envoy? *Envoy* : web site. 2025. URL: https://www.envoyproxy.io/docs/envoy/latest/faq/performance/how_to_benchmark_envoy (accessed 24.09.2025).

19. Yang W., Chen P., Yu G., Zhang Haibin, Zhang Huxing. Network shortcut in data plane of service mesh with eBPF, *Journal of Network and Computer Applications*. 2024. Vol. 222. Art. 103805. https://doi.org/10.1016/j.jnca.2023.103805.

20. Singh P., Ayyasamy S. Exploring, Analyzing and Tuning Service Mesh Performance: A Literature Review. *TechRxiv* : web site. 2023. https://doi.org/10.36227/techrxiv.22776119.v1.

Надійшла до редколегії 17.12.2025
Схвалена до друку 22.02.2026
Дата публікації 30.03.2026

Відомості про авторів:

Главчев Максим Ігорович
кандидат економічних наук доцент
професор
Національного технічного університету
“Харківський політехнічний інститут”,
Харків, Україна
https://orcid.org/0000-0001-9670-9118

Главчев Дмитро Максимович
доктор філософії
доцент
Національного технічного університету
“Харківський політехнічний інститут”,
Харків, Україна
https://orcid.org/0000-0003-4248-4819

Панченко Володимир Іванович
старший викладач
Національного технічного університету
“Харківський політехнічний інститут”,
Харків, Україна
https://orcid.org/0000-0003-3364-3398

Гейко Геннадій Вікторович
кандидат технічних наук
доцент
Національного технічного університету
“Харківський політехнічний інститут”,
Харків, Україна
https://orcid.org/0000-0001-6958-8306

Information about the authors:

Maksym Glavchev
PhD in Economic Sciences Associate Professor
Professor
of National Technical University
“Kharkiv Polytechnic Institute”,
Kharkiv, Ukraine
https://orcid.org/0000-0001-9670-9118

Dmytro Hlavchev
PhD
Associate Professor
of National Technical University
“Kharkiv Polytechnic Institute”,
Kharkiv, Ukraine
https://orcid.org/0000-0003-4248-4819

Volodymyr Panchenko
Senior Lecturer
of National Technical University
“Kharkiv Polytechnic Institute”,
Kharkiv, Ukraine
https://orcid.org/0000-0003-3364-3398

Hennadii Heiko
PhD in Engineering
Associate Professor
of National Technical University
“Kharkiv Polytechnic Institute”,
Kharkiv, Ukraine
https://orcid.org/0000-0001-6958-8306

EXPLORING THE BALANCE BETWEEN PERFORMANCE AND SECURITY IN MICROSERVICE TRAFFIC MANAGEMENT WITH SIDECAR PROXIES

M. Glavchev, D. Hlavchev, V. Panchenko, H. Heiko

The article presents an in-depth study of the trade-off between performance and security when using sidecar proxies and service mesh solutions for traffic management in microservice architectures. The relevance of this research is driven by the rapid adoption of the microservice model in modern information systems, where request stability, effective load balancing, and secure communication are key to ensuring the reliability and resilience of specialized computer systems. The complexity of such architectures arises from the large number of services interacting through networks, making them vulnerable to delays, overloads, and attacks. The study focuses on state-of-the-art open-source tools, including Envoy, Istio (in two configurations: mTLS and the ambient model), and Linkerd, which are widely adopted in service communication management. To objectively evaluate their effectiveness, an experimental environment was created with representative load-testing scenarios. Key performance and security metrics were measured, including request processing delays at the p95/p99 levels, throughput, resource overhead, and the degree of communication security assessed by the presence of mTLS, access policies, and additional control mechanisms. To generalize the results, a composite index was applied using SLO-oriented normalization. Special attention was given to the impact of weighting factors, with an emphasis on p99 latency as a critical SLA metric that directly affects user experience in high-load systems. The results demonstrated different balances between performance and security. Envoy and Linkerd proved to be the most effective in environments where minimizing latency and resource efficiency are critical. Istio in the ambient model offered an acceptable compromise between security and efficiency, making it suitable for hybrid scenarios. In contrast, Istio with mTLS confirmed its superiority in terms of maximum communication security, but at the cost of reduced throughput and increased latency. Based on the analysis, practical recommendations were formulated: Envoy and Linkerd are preferable for latency-sensitive and resource-constrained environments; Istio (mTLS) is recommended for corporate systems with strict confidentiality and security requirements; Istio (ambient) is best suited for cases where flexibility and reliability must be balanced. Overall, the results confirm that open-source tools can provide a wide range of capabilities for managing microservice communications, allowing organizations to select optimal strategies depending on whether performance or security is prioritized, while remaining a cost-effective alternative to commercial products.

Keywords: *computer system diagnostics; CPU overhead; Envoy; Istio; latency; Linkerd; load balancing; microservice; monitoring; mTLS; open-source; service mesh; throughput.*