Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«РОЗРОБКА ПРОГРАМНОГО ІНТЕРФЕЙСУ АВТОМАТИЗОВАНОЇ СИСТЕМИ БЕЗДРОТОВОГО ОБМІНУ ДАНИМИ В ІНДУСТРІАЛЬНИХ МЕРЕЖАХ»

Виконав: студент 2 курсу, 1АКІТР-24м
спеціальності 174 – Автоматизація,
комп'ютерно-інтегровані технології та
робототехніка
Денис КРЕБС
Керівник: д.т.н., проф., зав. каф. КСУ
В'ячеслав КОВТУН
«12» 12 2025 р.
Опонент: доцент каф. АІІТ
Юрій ІВАНОВ
«12» 12 2025 р.

Допущено до захисту
Зав. кафедри КСУ
В'ячеслав КОВТУН
«13» 12 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти другий (магістерський)
Галузь знань – 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітньо-професійна програма – Інтелектуальні комп'ютерні системи

ЗАТВЕРДЖУЮ
Зав. кафедри КСУ

_____В'ячеслав КОВТУН
«26» «09» 2025 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
студенту Кребсу Денису Володимировичу

1. Тема роботи. Розробка програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах
керівник роботи: д.т.н., проф. Ковтун В'ячеслав Васильович
затверджені наказом ВНТУ від «24» вересня 2025 р. №313
2. Термін подання студентом роботи «5» грудня 2025 р.
3. Вихідні дані до роботи: підвищені вимоги до затримки, надійності та масштабованості в індустріальних мережах, необхідність створення гнучкої архітектури програмного інтерфейсу, сучасні протоколи передавання (HTTP/2, HTTP/3 (QUIC), gRPC), бінарні формати серіалізації (Protocol Buffers, CBOR, msgpack), алгоритми стискання даних (gzip/DEFLATE, LZ4, Zstd). Перелік матеріалів до розробки:
   1. M. Bishop, "HTTP/3," RFC 9114, pp. 1-57, Jun. 2022. DOI: 10.17487/RFC9114
   2. J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," RFC 9002, pp. 1-42, May 2021. DOI: 10.17487/RFC9002.
   3. T. Bünzli, O. Alay, D. Koll, and A. Dhamdhere, "HTTP/3: Performance, Deployment and Adoption," IEEE Trans. Netw. Serv. Manag., vol. 21, no. 3, pp. 2056-2074, 2024. DOI: 10.1109/TNSM.2024.3467485.

4. Зміст текстової частини: вступ, аналіз сучасних технологій програмного інтерфейсу та індустріальних мереж, методи та інструменти дослідження ефективності бездротового обміну даними, реалізація експериментів та аналіз результатів тестування розробленого інтерфейсу, висновки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень). Структурна блок-схема автоматизованої системи, узагальнена схема передачі інформації, схеми реалізації програмних модулів, графіки порівняння результатів експериментальних досліджень.

1. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
4	Ольга РАТУШНЯК, к.т.н., доцент кафедри ЕПВМ		

2. Дата видачі завдання «25» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз сучасних технологій програмного інтерфейсу та індустріальних мереж (огляд протоколів HTTP/3, gRPC та методів стиснення)	25.09.25	09.10.25	
2	Обгрунтування методів та засобів реалізації (вибір мови Go, бібліотек серіалізації, розробка архітектури)	10.10.25	16.10.25	
3	Розробка програмного модуля інтерфейсу та налаштування експериментального стенду (серверна та клієнтська частини)	17.10.25	26.10.25	
4	Проведення експериментальних досліджень ефективності (вимірювання RPS, часу стиснення, використання ресурсів CPU/RAM)	26.10.25	02.11.25	
5	Аналіз отриманих результатів, порівняння конфігурацій та формулювання висновків	03.11.25	11.11.25	
6	Оформлення матеріалів до захисту	13.11.25	17.11.25	
7	Захист МКР			

Студент (підпис) Денис КРЕБС

Керівник роботи (підпис) В'ячеслав КОВТУН

АНОТАЦІЯ

УДК 004.7:681.5

Роботу присвячено розробленню програмного інтерфейсу автоматизованої системи бездротового обміну даними для індустріальних мереж із підвищеними вимогами до затримки, надійности та масштабованости. Мета – створити гнучку архітектуру, що за допомогою адаптивних алгоритмів оброблення потоків, сучасних протоколів передавання та ефективних методів серіалізації підвищує ефективність і стабільність комунікацій між промисловими вузлами.

Актуальність дослідження узгоджено з Ціллю сталого розвитку №9 «Промисловість, інновації та інфраструктура», зокрема завданням 9.4 щодо модернізації інфраструктури та ресурсної ефективности. Наукова новизна полягає у підході до побудови адаптивного інтерфейсу, який оптимізує маршрутизацію й оброблення даних у реальному часі з використанням комбінованих алгоритмів кодування та стискання.

Практична цінність – можливість інтеграції рішення у мережі різного масштабу для зменшення затримок, підвищення пропускної здатности та надійности взаємодії автономних пристроїв і модернізації систем моніторингу в IIoT.

Експериментально досліджено поєднання протоколів HTTP/2, HTTP/3 (QUIC) і gRPC, бінарних форматів (Protocol Buffers, CBOR, msgpack) та алгоритмів стискання (gzip/DEFLATE, LZ4, Zstd) у різних пресетах. Отримано, що HTTP/3 забезпечує стабільніше масштабування навантаження, а Zstd на рівні 6 демонструє найкращий компроміс між швидкодією та ефективністю (RPS, помірні ресурси), що робить його придатним для промислових систем реального часу.

Ключові слова: індустріальні бездротові мережі; автоматизовані системи керування; HTTP/3 (QUIC); gRPC; Protocol Buffers; CBOR; msgpack; Zstd; LZ4; стискання даних; серіалізація; IIoT; QoS.

ABSTRACT

UDC 004.7:681.5

This thesis develops a software interface for automated wireless data exchange in industrial networks facing stringent requirements for latency, reliability, and scalability. The objective is a flexible architecture that leverages adaptive stream-handling, modern transport/application protocols, and efficient data serialization to improve communication stability and throughput among industrial nodes.

The study aligns with UN Sustainable Development Goal 9—specifically Target 9.4 on industrial modernization and resource efficiency. The novelty lies in an adaptive interface design that optimizes routing and real-time processing using combined coding and compression techniques.

Practical value includes integrability across scales to reduce latency, improve throughput and communication reliability of autonomous devices, and upgrade IIoT monitoring systems.

The experimental campaign evaluates combinations of HTTP/2, HTTP/3 (QUIC), and gRPC with binary formats (Protocol Buffers, CBOR, msgpack) and compression algorithms (gzip/DEFLATE, LZ4, Zstd) under multiple presets. Results show that HTTP/3 yields smoother load scaling, while Zstd at level 6 offers the best trade-off between speed and efficiency (high RPS with moderate resource usage), making it suitable for real-time industrial deployments.

Keywords: industrial wireless networks; automated control systems; HTTP/3 (QUIC); gRPC; Protocol Buffers; CBOR; msgpack; Zstd; LZ4; data compression; serialization; IIoT; QoS.

ЗМІСТ

ВСТУП

У сучасних умовах цифрової трансформації промисловості зростає потреба в ефективних засобах бездротової комунікації між елементами автоматизованих систем керування. Використання індустріальних бездротових мереж забезпечує можливість швидкого обміну даними між контролерами, сенсорами, виконавчими пристроями та центральними серверами, що сприяє підвищенню рівня автоматизації виробничих процесів і зменшенню витрат на кабельну інфраструктуру. Разом із цим підвищуються вимоги до масштабованості, стабільності, енергоефективності та безпеки таких систем, а також до якості програмного забезпечення, яке реалізує комунікаційні протоколи та інтерфейси.

Розвиток промисловості нині неможливий без швидкого та надійного обміну даними між різними елементами автоматизованих систем – контролерами, датчиками, виконавчими механізмами та серверами. Використання дротових мереж часто обмежує мобільність, ускладнює модернізацію виробничих ліній і збільшує витрати на обслуговування. Тому підприємства все частіше переходять на бездротові рішення, які забезпечують гнучкість, масштабованість і оперативне підключення нових пристроїв без зупинки процесів.

Однак такі системи стикаються з низкою проблем – обмеженою пропускною здатністю, впливом перешкод, затримками під час передавання інформації та складністю підтримання стабільного з’єднання. Тому постає потреба у створенні програмних інтерфейсів, здатних автоматично адаптуватися до змін умов середовища, оптимізувати маршрути передавання й забезпечувати безпомилкову взаємодію між усіма вузлами мережі.

Додатково актуальність теми зумовлена розвитком концепції Industrial Internet of Things (IIoT), де тисячі пристроїв одночасно обмінюються даними. У таких умовах якість програмного забезпечення для організації комунікацій безпосередньо впливає на безпеку, точність і ефективність виробництва.

Таким чином, розроблення програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах є важливим і своєчасним завданням. Воно спрямоване на підвищення ефективності керування технологічними процесами, зниження витрат ресурсів і забезпечення стабільної взаємодії між компонентами виробничої інфраструктури.

Однією з основних проблем сучасних промислових мереж залишається забезпечення високої швидкості обміну при мінімальних затратах системних ресурсів. Зростання кількості підключених пристроїв у межах концепції Industrial Internet of Things (IIoT) призводить до перевантаження каналів і підвищених вимог до надійності та якості сервісу (QoS). Тому актуальним є створення програмного інтерфейсу автоматизованої системи бездротового обміну даними, який дозволяє адаптивно керувати потоками інформації, застосовувати сучасні методи стиснення, оптимізувати структуру пакетів і забезпечувати сумісність із відкритими мережевими протоколами.

Мета дослідження полягає у розробленні програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах, який підвищує ефективність передавання інформації завдяки використанню адаптивних алгоритмів оптимізації потоків, сучасних протоколів комунікації та вдосконалених методів серіалізації. Досягнення мети передбачає створення гнучкої програмної архітектури, що забезпечує стабільну роботу системи в умовах перешкод і мінімізує втрати даних під час обміну між промисловими вузлами.

Об’єкт дослідження – процес інформаційного обміну між компонентами автоматизованої бездротової системи керування в промислових мережах.

Предмет дослідження – методи та технології проєктування програмного інтерфейсу, алгоритми обробки й передавання даних, а також мережеві протоколи, що визначають якісні характеристики системи обміну.

Для досягнення поставленої мети в роботі вирішуються такі задачі: провести аналітичний огляд сучасних архітектур і протоколів промислових бездротових мереж; обґрунтувати вибір засобів і методів програмної реалізації інтерфейсу; розробити структуру програмного модуля для організації обміну даними між вузлами мережі; створити тестовий стенд для експериментальної перевірки функціонування інтерфейсу; виконати моделювання та оцінку показників ефективності системи.

Наукова новизна полягає в розробленні підходу до побудови адаптивного програмного інтерфейсу бездротової системи, який забезпечує оптимізацію маршрутизації та оброблення даних у реальному часі із застосуванням комбінованих алгоритмів кодування й стиснення.

Практична цінність результатів полягає у створенні програмного рішення, придатного для інтеграції в промислові мережі різного масштабу з метою підвищення продуктивності, зменшення затримок передавання й підвищення надійності комунікацій між автономними пристроями. Отримані результати можуть бути використані для модернізації систем моніторингу технологічних процесів, оптимізації протоколів обміну в мережах IIoT та розроблення промислових контролерів нового покоління.

Актуальність дослідження узгоджується з Ціллю сталого розвитку №9 «Промисловість, інновації та інфраструктура», ухваленою Організацією Об’єднаних Націй у межах Порядку денного сталого розвитку до 2030 року. Ця ціль спрямована на розвиток стійкої індустріалізації, впровадження інновацій та створення надійної інфраструктури як основи економічного зростання. Зокрема, робота відповідає завданню 9.4, що передбачає модернізацію промислової інфраструктури та підвищення ефективності використання ресурсів через застосування інноваційних технологій. Розроблений програмний інтерфейс автоматизованої системи бездротового обміну даними сприяє досягненню цих завдань, оскільки підвищує технологічну ефективність, інноваційність і енергоощадність промислових процесів, забезпечуючи перехід до сучасних цифрових виробничих рішень.

1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ПРОГРАМНОГО ІНТЕРФЕЙСУ ТА ІНДУСТРІАЛЬНИХ МЕРЕЖ

1.1 Актуальність і загальна характеристика напрямку

У сучасних індустріальних умовах спостерігається інтенсивне впровадження бездротових мережевих рішень для організації взаємодії між сенсорними пристроями, контролерами, виконавчими модулями та серверами збору і обробки даних. Такий підхід забезпечує оперативний обмін інформацією, контроль стану технологічних процесів і можливість адаптивного керування у реальному часі. Водночас із розширенням масштабів мереж та збільшенням кількості одночасно підключених вузлів зростає навантаження на канали зв’язку, що може спричинити затримки, втрату пакетів або порушення стабільності функціонування системи.

Одним з основних шляхів підвищення ефективності обміну даними в індустріальних мережах є впровадження оптимізованих алгоритмів обробки та стиснення інформаційних потоків у межах програмного інтерфейсу автоматизованої системи. Застосування таких алгоритмів забезпечує зменшення обсягу переданих пакетів без втрати суттєвого змісту, що дозволяє раціональніше використовувати пропускну здатність бездротового каналу, знижувати затримки та зменшувати споживання ресурсів пристроїв. На практиці ефективність системи визначається правильним добором алгоритму стискання – серед найпоширеніших використовуються gzip, lz4, bzip2, zstd. Кожен з них має різне співвідношення швидкодії, коефіцієнта компресії та вимог до процесорних потужностей. У реальному промисловому середовищі вибір компресора визначається архітектурою системи, обсягом трафіку та часовими обмеженнями.

На якість передавання даних значно впливає вибір мережевого протоколу. Застарілі підходи на основі HTTP/1.1 характеризуються великими накладними витратами через необхідність повторного встановлення з’єднань

і неможливість ефективного мультиплексування. У той час як сучасні рішення – HTTP/2, HTTP/3 або gRPC – забезпечують паралельну передачу кількох потоків у межах одного каналу, мають вбудовані механізми пріоритизації та суттєво скорочують затримки на прикладному рівні. Це критично важливо для систем, що функціонують із підвищеними вимогами до часу відгуку та стабільності передачі [4].

Формат представлення даних також є визначальним чинником у побудові високоефективного програмного інтерфейсу. Текстові формати, як JSON чи XML, мають просту структуру і зручні для налагодження, однак характеризуються значним обсягом службової інформації. Бінарні ж формати, зокрема Protocol Buffers, msgpack або CBOR, забезпечують компактне представлення повідомлень і зменшують трафік, хоча й ускладнюють процес діагностики в режимі реального часу. Баланс між читабельністю та продуктивністю визначається особливостями конкретного програмного рішення та вимогами до пропускної здатності каналу.

Водночас вибір формату впливає не лише на обсяг переданих даних, а й на обчислювальні витрати процесора під час операцій серіалізації та десеріалізації, що є критичним для енергозалежних вузлів мережі. Особливої актуальності це набуває в умовах нестабільних бездротових з’єднань, де зменшення розміру пакета прямо корелює зі зниженням імовірності його втрати та необхідності повторної передачі. Тому для забезпечення надійності промислових систем доцільно застосовувати сувору типізацію даних, притаманну бінарним схемам, що дозволяє мінімізувати помилки інтерпретації на рівні різнорідних пристроїв IIoT.

На рисунку 1.1 показано структурну схему функціонування автоматизованої системи бездротового обміну даними в індустріальній мережі із використанням оптимізованого програмного інтерфейсу та модуля стиснення.

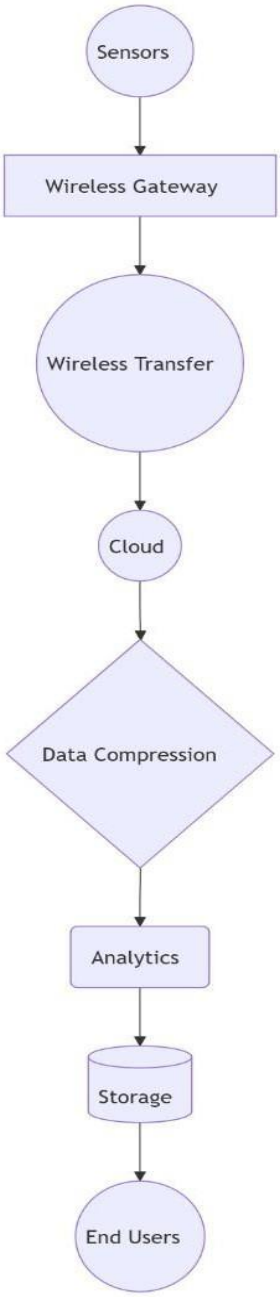


Рисунок 1.1 – Структурна блок-схема автоматизованої системи бездротового обміну даними з модулем оптимізованого стиснення

Підсумовуючи викладене, актуальність дослідження полягає у необхідності вдосконалення архітектури програмного інтерфейсу автоматизованих систем бездротового обміну даними, що забезпечить зменшення затримок, підвищення швидкодії та надійності передачі в умовах промислових мереж. Реалізація таких рішень сприятиме створенню масштабованої, адаптивної й енергоефективної інфраструктури нового покоління, здатної відповідати сучасним технологічним вимогам виробництва.

1.2 Вплив індустріального середовища на ефективність програмного інтерфейсу бездротового обміну даними

Індустріальні бездротові мережі, на відміну від побутових або корпоративних систем, функціонують у складних умовах із підвищеними вимогами до стабільності, надійності та захисту інформації. У таких мережах необхідно забезпечити гарантований час реакції системи, стійкість до електромагнітних впливів, вібрацій і перепадів напруги, які притаманні промисловому середовищу. Ці чинники можуть спричиняти втрату пакетів або виникнення затримок, що впливають на якість сервісу (QoS). Тому при проєктуванні програмного інтерфейсу автоматизованої системи бездротового обміну даними необхідно враховувати, як процес стиснення та розпакування інформації впливає на часові характеристики та загальну пропускну здатність каналу.

Виробничі системи часто працюють у режимах жорсткого або м’якого реального часу. Для сценаріїв жорсткого реального часу навіть незначні затримки можуть призвести до порушення логіки управління або виходу обладнання з ладу. Наприклад, у системах керування роботизованими маніпуляторами чи регулювання температурних процесів у реакторах перевищення припустимого часу відгуку недопустиме. В таких умовах навіть високоефективні алгоритми компресії можуть бути недоцільними, якщо їхнє виконання збільшує латентність. У випадках, коли дані передаються асинхронно (як у хмарній аналітиці, обліку або статистичному моніторингу), тривалість стиснення не є критичною, і тоді доцільно використовувати алгоритми з більш високим ступенем компресії.

В архітектурі автоматизованих систем важливу роль відіграють проміжні вузли – шлюзи або контролери. Вони можуть обробляти потоки даних від великої кількості сенсорів, виконуючи попереднє фільтрування, агрегацію чи виявлення аномалій. Виникає питання розподілу функцій між елементами системи: чи варто реалізовувати стиснення безпосередньо на

сенсорному рівні, чи доцільніше делегувати його обчислювально потужнішому контролеру. Компресія на рівні сенсора дозволяє зменшити навантаження на канал зв’язку, але може створити додаткові затримки через обмежені ресурси мікроконтролера. Натомість виконання стиснення на рівні шлюзу забезпечує вищу ефективність за рахунок складніших алгоритмів, але вимагає узгодження часу обробки.

Особливу увагу слід приділяти промисловим протоколам, таким як OPC UA, Profinet, Modbus TCP або EtherNet/IP. Вони мають власні вимоги до структури пакетів, підтвердження доставки та шифрування. Під час додавання компресії потік даних може поділятися на кілька блоків, і в разі втрати одного з них доведеться передавати увесь сегмент повторно, що вплине на затримку. Тому на практиці часто використовують комбіновані рішення – легкі алгоритми стискання в поєднанні з механізмами корекції помилок та сегментації інформації [5].

Питання безпеки також є критичним. Оскільки зашифровані дані важко піддаються подальшій компресії, стискання зазвичай виконується до моменту шифрування. Якщо система використовує наскрізне шифрування, потрібно передбачити, які саме дані можна стискати на проміжних вузлах без порушення політики конфіденційності. Деякі протоколи, зокрема HTTP/2 або gRPC, підтримують механізми попереднього стискання перед активацією TLS, що дозволяє скоротити трафік без розкриття вмісту повідомлень [6].

Таким чином, стиснення даних у промислових бездротових мережах є невід’ємною частиною загальної архітектури автоматизованої системи. Його вплив слід оцінювати комплексно – з урахуванням технічних характеристик обладнання, топології мережі, параметрів протоколів і зовнішніх чинників середовища. Сучасні технології зв’язку на базі 5G, Wi-Fi 6/6E, WirelessHART та ISA100.11a створюють передумови для більш ефективного обміну даними, проте оптимізація залишається компромісом між швидкодією, надійністю, енергоспоживанням і масштабованістю.

Для великих розподілених систем, у яких одночасно працюють сотні або тисячі пристроїв, важливою стає підтримка багатопоточності та масштабування. Застосування хмарних платформ для зберігання та аналітики даних збільшує обсяг трафіку, тому архітектура програмного інтерфейсу має передбачати балансування навантаження між потоками та серверами.

У періоди пікових навантажень, коли система обробляє великі обсяги інформації, алгоритми компресії можуть створювати додаткове навантаження на обчислювальні ресурси. Тому доцільним є виділення окремих процесорних ядер або потоків для виконання операцій стискання з метою стабілізації часу відгуку системи.

У деяких випадках, особливо при критичних технологічних процесах, головним параметром оцінювання стає не загальна пропускна здатність, а гарантований час доставки пакета. Для таких сценаріїв доцільним є використання адаптивних алгоритмів, які автоматично вмикають або вимикають стиснення залежно від пріоритету даних і поточного стану мережі.

На рисунку 1.2 подано узагальнену схему передачі інформації у бездротовій індустріальній мережі, що відображає взаємозв’язок основних технічних і програмних факторів, які впливають на якість обміну.

Рисунок 1.2 – Узагальнена схема передачі інформації в автоматизованій системі бездротового обміну даними

З урахуванням викладеного можна зазначити, що процес вибору оптимальної архітектури обміну даними в автоматизованих бездротових системах є багаторівневою задачею, яка охоплює апаратне, транспортне та програмне забезпечення. Розумне поєднання алгоритмів стиснення, протоколів і форматів передачі дозволяє створити високопродуктивну інфраструктуру, здатну забезпечити максимальну ефективність при мінімальних ресурсних витратах.

1.3 Теоретичні засади та підходи до стиснення даних у програмному інтерфейсі бездротових систем

Методи стиснення даних у сучасних бездротових системах ґрунтуються на принципі виявлення надлишкової інформації у вхідному потоці та її перетворення у компактніше представлення без втрати змістовності. Основи цього процесу закладено в теорії інформації Клода Шеннона, який увів поняття ентропії як міри невпорядкованості даних. Чим менше ентропія інформаційного потоку, тим ефективніше дані можуть бути стиснені. Для промислових бездротових систем це означає, що телеметричні потоки, у яких багато повторюваних або мало змінюваних значень (наприклад, температура, тиск, рівень вібрації), мають високий потенціал для ефективної компресії. У таких випадках алгоритм стиснення не лише зменшує розмір пакета, але й знижує навантаження на канал зв’язку, підвищуючи загальну стабільність системи.

Основна мета алгоритмів стиснення полягає в одночасному досягненні двох показників – мінімізації розміру даних і забезпеченні прийнятної швидкодії процесів кодування та декодування. Проте між цими характеристиками завжди існує компроміс: чим більший ступінь стиснення, тим вища часові витрати. Класичні методи, такі як кодування Хаффмана або арифметичне кодування, базуються на статистичних моделях імовірності

появи символів, але через високу обчислювальну складність у промислових середовищах часто замінюються більш продуктивними словниковими підходами (gzip, lz4, zstd, bzip2), які забезпечують добрий баланс між коефіцієнтом стиснення та швидкодією [7]. Такі алгоритми створюють словник повторюваних послідовностей у потоці та замінюють їх короткими посиланнями, досягаючи зменшення обсягу за рахунок збереження структури даних. На рисунку 1.3 наведено спрощену схему життєвого циклу запиту у мережевому протоколі, де застосовується стиснення даних.

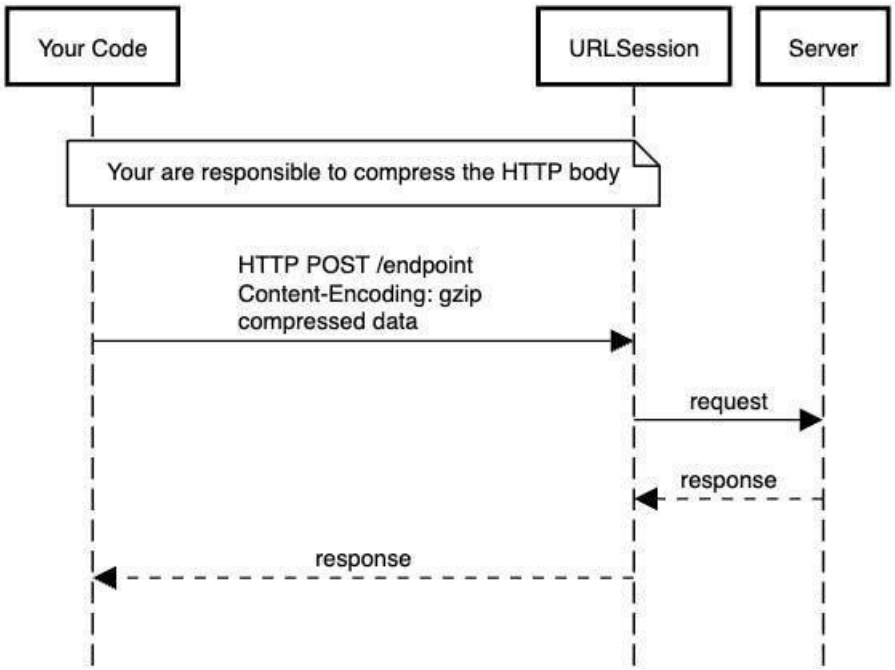


Рисунок 1.3 – Цикл запиту у протоколі з використанням стиснення

Порівняння алгоритмів показує, що gzip, який реалізує підхід DEFLATE, залишається популярним завдяки простоті реалізації, тоді як новіші методи, як-от zstd, пропонують більшу гнучкість і масштабованість. Алгоритм zstd, створений у Facebook, дозволяє налаштовувати ступінь стиснення в широкому діапазоні – від високошвидкісного, із мінімальною компресією, до глибокого ущільнення великих обсягів даних [9]. Для систем бездротового моніторингу, що збирають телеметрію у реальному часі, це

забезпечує можливість адаптації до поточного навантаження, змін мережевих умов і обмежень апаратної платформи.

Залежність між швидкістю та коефіцієнтом компресії показано на рисунку 1.4, де відображено зміну ефективності для zstd, zlib, brotli та lzma при різних рівнях ущільнення.

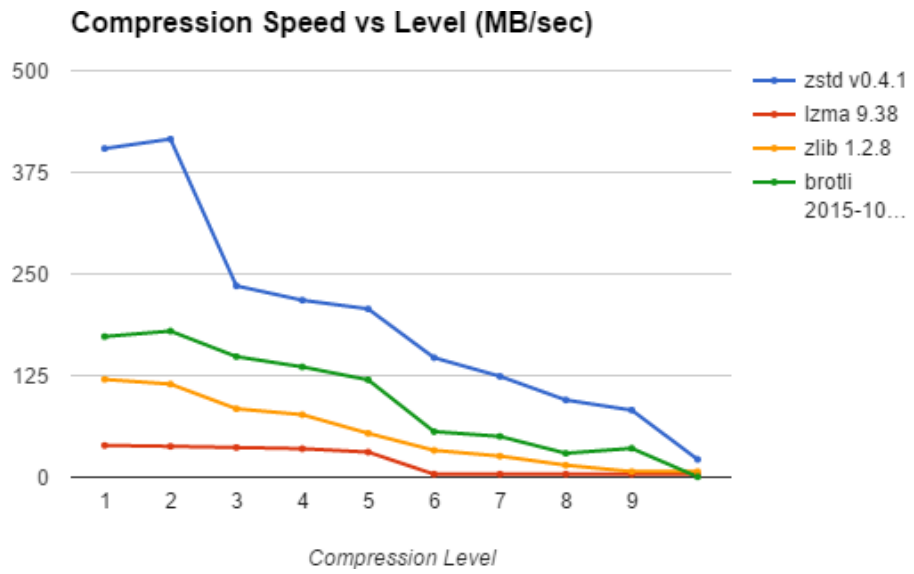


Рисунок 1.4 – Залежність швидкості та коефіцієнта стиснення для різних алгоритмів

У реальних промислових сценаріях стиснення часто обмежене невеликими обсягами даних, які надходять від сенсорів із частотою у десятки разів на секунду. Потужні алгоритми не завжди встигають реалізувати свої переваги, тому практичним рішенням є lz4, який орієнтований на швидке стискання з помірним коефіцієнтом. На рівні проміжних шлюзів можливе об’єднання невеликих порцій у більші блоки перед стисканням, що підвищує ефективність без зростання затримок.

Завдяки розвитку штучного інтелекту з’являються дослідження, спрямовані на створення інтелектуальних компресорів – гібридних алгоритмів, що навчаються на прикладах реальних потоків. Нейронні методи здатні визначати патерни повторів і прогнозувати майбутні значення, що

забезпечує динамічне налаштування моделі стиснення. Хоча такі методи поки не мають широкого промислового застосування через високу обчислювальну вартість, у майбутньому вони можуть бути інтегровані у системи IIoT із підтримкою машинного навчання на периферійних пристроях.

У практиці промислових мереж безвтратне стискання (lossless) є основним типом, оскільки воно гарантує збереження точності даних. У випадках, коли допустима певна похибка, застосовують стискання з втратами (lossy) – наприклад, для відеоспостереження виробничих процесів, де алгоритми H.264 або H.265 забезпечують істотне зменшення трафіку без втрати інформативності [12].

Важливу роль відіграє апаратне прискорення. Сучасні контролери та шлюзи все частіше оснащуються спеціалізованими прискорювачами компресії (hardware accelerators), що дозволяє виконувати операції стискання швидше, зменшуючи споживання енергії. На рисунку 1.5 наведено приклад порівняння розміру відеофайла після стискання алгоритмами H.264 і H.265.

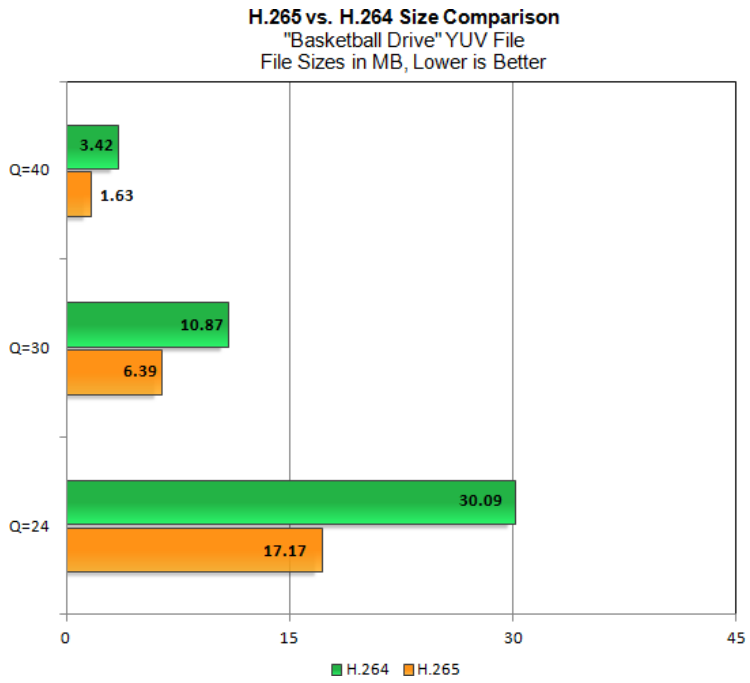


Рисунок 1.5 – Порівняння розміру відеофайлів при різних алгоритмах стискання

Для бездротових систем важливо, щоб процес розпакування (декомпресії) не створював затримок у роботі. Якщо формат передбачає складну багатоступеневу декомпресію, то в системах реального часу це може знизити ефективність обробки. Тому часто обирають алгоритми зі швидким декодуванням, навіть якщо їхній коефіцієнт стиснення нижчий.

Особливої уваги заслуговують адаптивні методи, у яких модель стиснення оновлюється в реальному часі. Вони дозволяють підтримувати стабільну ефективність при зміні структури даних – наприклад, коли сенсор різко переходить у новий діапазон значень. Такий підхід забезпечує баланс між точністю, швидкодією та енергоефективністю, проте потребує оптимізації внутрішніх буферів і механізмів керування пам’яттю.

Крім того, при виборі алгоритму важливими є параметри блокування (chunking) і розмір буфера. Алгоритми типу zstd або gzip стискають дані блоками, і збільшення розміру блоку підвищує коефіцієнт стискання, але також збільшує затримку. Для систем, де важливий швидкий відгук, зазвичай обирають невеликі блоки, щоб забезпечити стабільну роботу мережі без накопичення черг.

У потокових (streaming) застосуваннях компресор працює безперервно, стискаючи вхідні дані в режимі реального часу. Це характерно для розподілених систем збору інформації, де одночасно працюють десятки сенсорів. У таких умовах ключовим параметром є швидкість реакції алгоритму та його здатність підтримувати постійний темп обміну.

Іншим напрямом розвитку є багаторівневе стискання, коли первинна компресія виконується на рівні сенсора, а вторинна – на сервері або в хмарному середовищі. Це дозволяє комбінувати швидкі алгоритми для попередньої обробки з глибшими методами для архівування. У деяких випадках доцільно також застосовувати попередню фільтрацію даних – видалення шумових або повторюваних вимірів перед стисканням, що зменшує навантаження на мережу без значних обчислень.

Особливої уваги заслуговує надійність розпакування даних. У промислових мережах, де інформацію приймають кілька вузлів, система має забезпечувати стійкість до втрат окремих пакетів. Якщо помилка в одному блоці призводить до зупинки декодування всього потоку, слід впроваджувати механізми сегментації, повторної передачі або контрольні суми, сумісні з мережевим протоколом.

Загалом сучасні підходи до стиснення у програмних інтерфейсах бездротових систем передбачають комплексну оптимізацію процесів кодування, передачі й декодування. Вибір алгоритму визначається природою даних, доступними ресурсами, вимогами до часу відгуку та безпеки. У перспективі поєднання апаратного прискорення, адаптивних моделей і хмарних обчислень дозволить створити інтелектуальні системи стиснення, здатні самостійно змінювати параметри роботи відповідно до поточного стану мережі. Такий підхід сприятиме зростанню ефективності промислових бездротових мереж, мінімізації затримок і забезпеченню стабільності в умовах реального виробничого навантаження.

1.4 Мережеві протоколи у системах бездротового промислового обміну даними

У процесі створення ефективної автоматизованої системи бездротового обміну даними одним із ключових аспектів є правильний вибір і налаштування мережевого протоколу. Традиційний стек TCP/IP залишається основою більшості сучасних систем зв’язку, однак у промислових середовищах, що характеризуються підвищеними вимогами до часу відгуку, надійності та безпеки, його базові механізми не завжди забезпечують необхідну ефективність. У таких мережах навіть незначні затримки або втрата пакетів можуть впливати на коректність функціонування обладнання,

що, своєю чергою, здатне призвести до порушення технологічного процесу або зупинки виробничого циклу.

Особливу увагу привертають параметри, пов’язані з багатопоточністю, налаштуванням таймерів підтвердження (acknowledgment), оптимізацією повторних передач (retransmissions) та стабільністю під час втрати пакетів. Для корпоративних систем втрата пакета часто призводить лише до короткочасного зниження швидкості, тоді як у промислових мережах це може мати критичні наслідки, тому доцільно застосовувати протоколи з покращеними механізмами контролю трафіку.

Через те, що TCP характеризується передбачуваною, але іноді надмірно обережною поведінкою при роботі з помилками, у промислових додатках нерідко використовується UDP. Хоча UDP не гарантує доставку пакетів, він забезпечує мінімальну затримку та низькі накладні витрати, що особливо важливо у випадках, коли дані надсилаються з високою частотою [5]. Утім, застосування UDP вимагає додаткових логічних рівнів, які компенсують відсутність перевірки цілісності або реалізують механізми повторної передачі в разі втрат [6]. У системах реального часу – наприклад, EtherCAT, Profinet RT, Ethernet/IP – використовуються гібридні варіанти, які комбінують переваги TCP і UDP, забезпечуючи низьку латентність, визначений час оброблення кадрів і гарантовану доставку критичних повідомлень.

Окрім протоколів нижчого рівня, у сучасних промислових архітектурах дедалі ширше застосовуються технології прикладного рівня HTTP/2, HTTP/3 і gRPC. Протокол HTTP/2 запровадив мультиплексування, що дозволяє одночасно передавати кілька потоків даних через одне з’єднання, зменшуючи накладні витрати на відкриття нових сеансів. Його бінарна структура та підтримка пріоритизації потоків забезпечують ефективніше використання ресурсів мережі, а також дають змогу обробляти важливі запити з вищим пріоритетом [15]. HTTP/3, побудований на базі QUIC, усуває характерні для TCP проблеми, зокрема блокування за принципом «head-of-line», що часто

створює затримки при передачі великих потоків даних. QUIC реалізує контроль на рівні користувацького простору, що дозволяє мінімізувати вплив ядра операційної системи на швидкодію та покращує стійкість до втрат пакетів [16].

Ще одне сучасне рішення – gRPC, фреймворк для розподілених систем, який розроблений для підвищення ефективності взаємодії між численними компонентами індустріальної мережі. На відміну від класичної моделі запит/відповідь, gRPC підтримує двосторонні потокові з'єднання, що дозволяє здійснювати обмін у режимі реального часу між клієнтом і сервером. Використання бінарного формату Protocol Buffers та транспортного рівня HTTP/2 забезпечує високу швидкість і компактність обміну порівняно з REST API на основі JSON. Завдяки цим особливостям gRPC є ефективним для промислових систем, де необхідно обмінюватися складними структурованими даними при збереженні низької латентності та стабільної пропускної здатності [17].

На рисунку 1.6 наведено порівняння середньої затримки при передачі даних через HTTP/1, HTTP/2 та HTTP/3 (QUIC), що демонструє переваги новіших протоколів щодо швидкодії та відновлення після втрат.

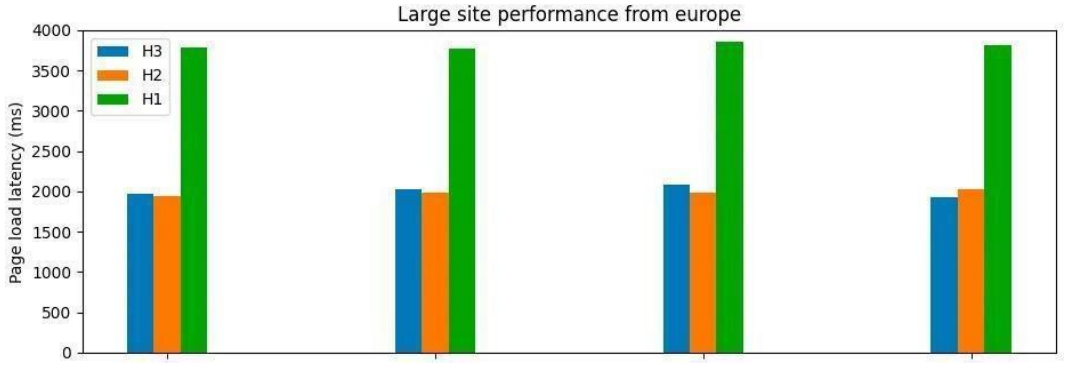


Рисунок 1.6 – Порівняння затримки при використанні HTTP/1, HTTP/2 та HTTP/3 (QUIC)

У практичних умовах вибір протоколу залежить не лише від технічних характеристик, але й від вимог інтеграції з наявною інфраструктурою

підприємства. У разі, коли виробництво тривалий час функціонувало на базі протоколу Modbus TCP, перехід до сучасніших рішень потребує поступової міграції або застосування шлюзів-конвертерів. Такі шлюзи можуть виконувати проміжне стискання даних, конвертацію форматів чи оптимізацію черговості запитів без втручання в основну систему. У нових розробках, де використовується HTTP/3 або gRPC, програмний інтерфейс системи може бути спроєктований одразу з урахуванням багатопоточності, стиснення та шифрування. Це забезпечує адаптивну маршрутизацію запитів, балансування навантаження та підвищену стійкість до збоїв.

Таким чином, ефективність бездротової системи промислового обміну даними значною мірою залежить від правильного поєднання протоколів транспортного й прикладного рівнів. Збалансована архітектура, у якій стиснення, безпека та передача даних працюють як єдиний механізм, забезпечує стабільність функціонування, оптимізацію пропускної здатності та мінімальні затримки в критичних виробничих процесах.

1.5 Формати представлення та серіалізації даних у бездротових промислових системах

У бездротових промислових мережах ефективність обміну значною мірою залежить від того, як саме дані структуровано, упаковано й передано між вузлами системи. Формат представлення визначає не лише обсяг пакета, а й швидкість його обробки, сумісність між пристроями та зручність подальшого аналізу. Чим компактніше й чіткіше організовано дані, тим ефективніше працює вся система, особливо коли мова йде про високочастотні сенсорні потоки або передачу телеметрії з мінімальною затримкою.

Одним із найпоширеніших форматів представлення даних у прикладних системах залишається JSON (JavaScript Object Notation). Його

переваги полягають у простоті, зрозумілості та широкій підтримці в більшості мов програмування. Завдяки текстовій природі JSON зручний для налагодження та ручного перегляду структури повідомлень, що особливо корисно під час розроблення або тестування [18]. Однак така зручність має свою ціну: дужки, лапки, коми та інші службові символи збільшують обсяг передаваних даних, що у промислових мережах може стати істотним обмеженням.

Для зменшення навантаження дедалі частіше використовують бінарні формати – Protocol Buffers, Thrift, msgpack, CBOR. Вони забезпечують високу щільність представлення даних і швидке серіалізування та десеріалізування [19]. Наприклад, Protocol Buffers (protobuf), розроблений компанією Google, став стандартом для інтеграції з gRPC-з'єднаннями. Він використовує схеми опису структур, що гарантує сумісність між клієнтами різних мов і платформ. msgpack і CBOR, натомість, акцентують увагу на простоті – замість формальної схеми тут використовуються гнучкі структури ключ/значення, де ключ може бути як рядком, так і числовим ідентифікатором [20]. Такий підхід полегшує оновлення структури повідомлень, але при великій кількості вузлів може призводити до неузгодженостей, якщо різні пристрої по-різному інтерпретують одне й те саме поле.

Однією з переваг бінарних форматів є те, що навіть без стиснення вони мають значно менший обсяг, ніж JSON чи XML. Це спрощує роботу алгоритмів компресії – gzip, lz4, zstd, – оскільки їм не потрібно обробляти зайві текстові елементи. Поєднання бінарного представлення з безвтратним стисненням дає змогу зменшити обсяг переданих пакетів у кілька разів, що особливо важливо для систем із низькою пропускною здатністю каналу або великою кількістю одночасних з'єднань. У випадку текстових форматів ступінь стискання зазвичай нижчий через їхню структурну надмірність.

У промислових системах також часто застосовують власні (custom) бінарні схеми серіалізації, створені під конкретне завдання або тип

обладнання. Такі формати передбачають фіксовані позиції байтів для числових та рядкових значень, що дає змогу обробляти дані без додаткового аналізу структури. Основна перевага цього підходу – мінімальні накладні витрати на парсинг і максимальна швидкодія. Недолік – низька гнучкість: будь-яка зміна структури потребує оновлення всіх учасників мережі. Проте для стабільних систем зі сталими полями параметрів (наприклад, у виробничих лініях) це рішення вважається оптимальним.

На рисунку 1.7 наведено приклад порівняння продуктивності JSON і Protocol Buffers у середовищі програмування Go, де видно, що використання бінарного формату суттєво скорочує час серіалізації та розмір повідомлення.

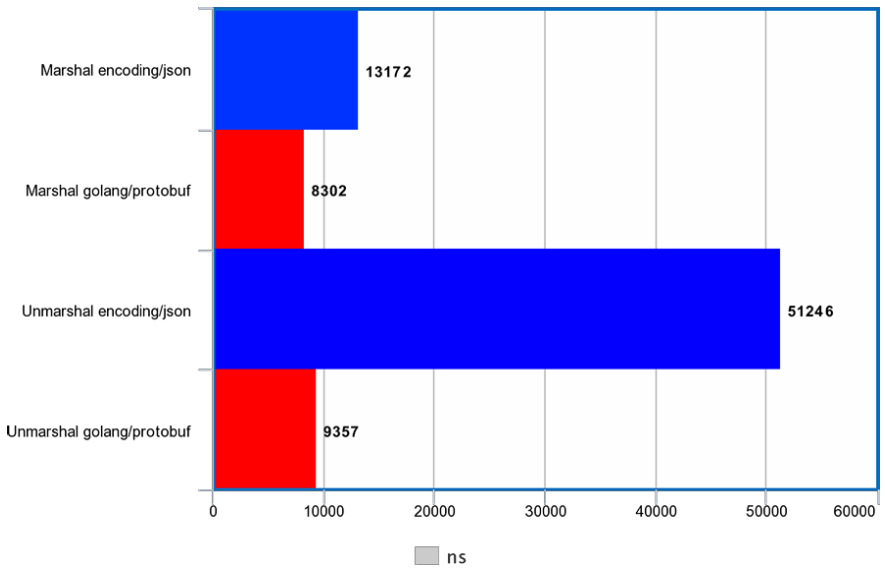


Рисунок 1.7 – Порівняння JSON та Protocol Buffers у мові програмування Go

Вибір формату серіалізації безпосередньо впливає на архітектуру всієї системи. Текстові формати доцільні на етапі розроблення, інтеграції або для передачі даних, що потребують легкого читання й перевірки. Бінарні формати – оптимальні для стабільних систем реального часу, де пріоритетом є швидкодія, компактність і відсутність дублювання. З огляду на тенденції розвитку промислових бездротових мереж, найбільш ефективними залишаються рішення, що комбінують компактні формати даних і алгоритми

стиснення, забезпечуючи оптимальне співвідношення між швидкістю передачі, точністю відтворення й ресурсною ефективністю системи.

1.6 Взаємодія алгоритмів стиснення, мережевих протоколів і форматів у бездротових промислових системах

Високоефективна робота промислових бездротових мереж можлива лише за умови комплексного узгодження між рівнями стиснення, форматами представлення даних і мережевими протоколами. Кожен із цих компонентів виконує власну функцію, проте тільки їхня скоординована взаємодія забезпечує стабільну передачу інформації, оптимальне використання ресурсів і мінімальні затримки. У реальних індустріальних умовах будь-яке втручання в один із рівнів комунікаційного ланцюга – наприклад, зміна формату даних або алгоритму компресії – автоматично впливає на всі інші параметри системи, включно з енергоспоживанням, обсягом трафіку, навантаженням на процесор і швидкодією каналів.

Якщо на рівні форматування здійснюється перехід від текстового JSON до бінарного представлення на основі Protocol Buffers або CBOR, коефіцієнт стиснення зростає завдяки усуненню надлишкової символіки. Проте для алгоритмів компресії це означає зміну внутрішньої статистики даних – повторювані патерни стають коротшими, а отже, алгоритм повинен бути налаштований на обробку дрібніших фрагментів. Такі перетворення впливають на декомпресію: збільшення швидкості стиснення не завжди корелює з такою ж швидкою розпаковкою, що важливо для вузлів із низькою обчислювальною потужністю. У системах реального часу це може призвести до втрати синхронізації між рівнями, коли дані надходять раніше, ніж завершено обробку попереднього пакета.

У багаторівневій архітектурі промислових мереж завжди присутня різниця між можливостями сенсорних вузлів, шлюзів і центральних серверів.

На нижчому рівні використовуються легкі алгоритми стискання, наприклад lz4 або zstd у швидкому режимі, які дозволяють виконувати компресію «на льоту». Шлюзи, що мають більше ресурсів, можуть застосовувати складніші методи, як-от LZMA чи bzip2, особливо коли йдеться про агреговані потоки даних від десятків сенсорів. На серверному або хмарному рівні допускається глибоке стиснення, оскільки там виконуються операції аналітики, прогнозування й архівування. Таким чином, алгоритми розподіляються за ієрархією відповідно до ресурсів і вимог до затримки.

Вибір транспортного протоколу є ще одним фактором, який суттєво впливає на узгодження між рівнями. UDP забезпечує мінімальні затримки, але не гарантує доставку; TCP, навпаки, гарантує цілісність і порядок сегментів, проте створює додаткову латентність через механізми повторних підтверджень і буферизацію. У промислових середовищах, де потрібен компроміс між швидкодією та стабільністю, застосовуються комбіновані рішення – Profinet RT, EtherCAT, Ethernet/IP – які інтегрують транспортні та прикладні рівні в єдиний часово детермінований контур. HTTP/2 і HTTP/3, що базуються на мультиплексуванні, дають змогу передавати декілька потоків у межах одного з’єднання, зменшуючи накладні витрати, проте потребують значно потужнішого процесорного середовища для підтримки паралельних запитів.

Завдання стиснення ускладнюється ще більше, якщо система застосовує наскрізне шифрування. Зашифровані потоки не мають структурних закономірностей, тому їх неможливо стискати ефективно. У таких випадках оптимальним є виконання компресії перед шифруванням, але це можливо лише тоді, коли політика безпеки дозволяє доступ до відкритих даних на проміжних вузлах. В іншому разі використовують часткове стиснення метаданих або допоміжних полів. Гібридна модель, у якій частина інформації передається у відкритому вигляді, а решта – шифрується, дозволяє знизити трафік, не порушуючи загального рівня безпеки. Це

потребує ретельної синхронізації ключів шифрування та управління доступом між різними рівнями системи.

У масштабних розподілених мережах кількість взаємодій між протоколами, форматами та алгоритмами може обчислюватися сотнями за секунду. Тут на перший план виходять методи динамічного керування трафіком – черги з пріоритетами, буферизація пакетів, асинхронна маршрутизація. Для забезпечення стабільної роботи системи доцільно використовувати політики QoS (Quality of Service), які дозволяють встановлювати різні режими обробки залежно від типу даних: критичні сигнали передаються без стиснення, а другорядна інформація може проходити через додаткові етапи компресії. Така стратифікація трафіку особливо важлива для технологічних процесів із різним рівнем чутливості до затримок.

Розвиток індустріального інтернету речей (IIoT) створює умови для появи інтелектуальних адаптивних систем стиснення. Вони можуть використовувати машинне навчання для прогнозування обсягу вхідного потоку та динамічної зміни алгоритму залежно від навантаження. Наприклад, якщо система передбачає підвищення трафіку внаслідок запуску нового цеху, компресор автоматично переходить у режим із вищим ступенем стискання. У години спокійної роботи, навпаки, може використовуватись швидкий, але менш ефективний алгоритм, щоб уникнути зайвих обчислювальних витрат. Такі системи вже частково реалізуються в сучасних промислових платформах і демонструють перспективу для повної автоматизації управління компресією.

Особливу роль відіграє архітектура API, яка забезпечує взаємодію між програмними компонентами системи. Якщо API розроблено з підтримкою адаптивного оброблення потоків, воно може самостійно визначати, які формати даних використовувати та який рівень стискання вмикати залежно від умов. Така модульна структура дозволяє оновлювати або замінювати компресійні алгоритми без зупинки системи, забезпечуючи неперервність

виробничого процесу. Вона також полегшує інтеграцію з іншими мережевими сервісами – аналітичними платформами, базами даних, інтерфейсами моніторингу – і забезпечує стабільну роботу незалежно від того, які формати чи протоколи застосовуються.

Ще один важливий аспект – сумісність між виробниками обладнання. Часто сенсори, контролери та шлюзи постачаються від різних компаній, які використовують власні формати даних і протоколи. Для забезпечення узгодженості необхідно впроваджувати універсальні адаптери, які автоматично конвертують структури даних між пристроями. При цьому важливо, щоб такі адаптери не створювали додаткових затримок у мережі. У багатьох системах цей процес реалізовано через спеціальні прошарки middleware, які одночасно виконують стискання, дешифрування та маршрутизацію повідомлень.

Критично важливим фактором залишається управління конфігураціями й версіями програмного забезпечення. Оскільки промислові системи часто функціонують у безперервному режимі, оновлення алгоритмів стиснення чи протоколів повинно відбуватися поступово – із резервуванням та автоматичним дублюванням функцій. Це запобігає простою виробничих процесів і забезпечує зворотну сумісність між новими та старими пристроями. Додатково система моніторингу повинна відстежувати параметри затримки, ефективність стискання та рівень використання мережевих ресурсів, аби своєчасно коригувати налаштування.

У підсумку, взаємодія між алгоритмами стиснення, протоколами передачі й форматами серіалізації даних є одним із ключових факторів ефективності промислових бездротових систем. Узгоджена архітектура дозволяє знизити витрати на передачу, забезпечити стабільну роботу мережі, підвищити рівень безпеки та зберегти масштабованість для подальшого розвитку інфраструктури. Завдяки використанню адаптивних моделей, модульних API та політик QoS можливо досягти високого рівня гнучкості й автономності, що робить такі системи здатними до саморегулювання,

оперативного відновлення після збоїв і безпечної інтеграції в екосистеми промислового інтернету речей.

1.7 Майбутні виклики та перспективні напрями розвитку бездротових промислових систем обміну даними

Подальший розвиток бездротових промислових мереж, які поєднують інформаційні технології, засоби автоматизації та аналітику великих даних, визначається новими науково-технічними викликами. Сучасне виробництво потребує швидкої, надійної та масштабованої комунікаційної інфраструктури, здатної обробляти потоки телеметрії, відео, журналів подій та інших типів даних, що надходять від сотень і навіть тисяч пристроїв. Збільшення обсягів інформації вимагає від систем не лише вищої пропускної здатності, а й здатності до адаптивного керування трафіком, динамічного стиснення та самостійного аналізу даних безпосередньо на рівні пристроїв.

Одним із найважливіших напрямів розвитку є впровадження інтелектуальних алгоритмів стиснення на базі машинного та глибинного навчання, які функціонують безпосередньо на периферійних вузлах (edge AI). Ідея полягає в тому, щоб передавати до центральної системи лише узагальнену інформацію, тоді як первинний аналіз і фільтрація виконуються на місці. Наприклад, якщо сенсор фіксує стабільний стан об’єкта, система може передати лише компактний набір статистичних параметрів або індикатор «норма», а повний потік даних надсилатиметься лише при виявленні відхилень. Такий підхід істотно знижує навантаження на мережу, забезпечує роботу в межах суворих часових обмежень і підвищує енергоефективність пристроїв. Водночас це створює нові завдання – як забезпечити достовірність локального аналізу, уникнути помилкових рішень та втрати критичних даних під час стискання.

Ще одним перспективним напрямом стає впровадження багатоканальної передачі з динамічним перемиканням маршрутів і резервуванням каналів. У таких системах мережа здатна автоматично обирати найстабільніший або найменш завантажений радіоканал, а алгоритми стиснення активуються лише тоді, коли рівень навантаження перевищує допустимі пороги. Це дозволяє зберігати високу пропускну здатність без втрати швидкодії. У розумних промислових інфраструктурах подібні адаптивні схеми можуть керувати профілями компресії в режимі реального часу, змінюючи параметри в залежності від умов радіооточення, кількості активних вузлів чи типу переданих даних. Такий принцип лежить в основі концепції автономних систем зв'язку, які здатні до самоорганізації та самовідновлення.

Значний потенціал має також поєднання методів стиснення з технологіями мікросервісної архітектури та контейнеризації. Сучасні промислові платформи дедалі частіше базуються на мікросервісах, кожен з яких відповідає за певну функцію: моніторинг стану обладнання, обробку відеопотоку, логування або аналітику. Оскільки ці сервіси можуть розміщуватись у хмарі, оптимізація трафіку безпосередньо впливає на фінансові витрати, адже модель оплати «pay-as-you-go» враховує як обсяг переданих даних, так і використані обчислювальні ресурси. Використання ефективних алгоритмів стискання на прикладному рівні (наприклад, у gRPC), застосування бінарних форматів серіалізації та попередньої компресії безпосередньо в API здатні суттєво скоротити експлуатаційні витрати, зберігаючи при цьому якість обміну.

Водночас інтеграція таких технологій створює нові складнощі: необхідність точного узгодження між рівнями безпеки, компресією та протоколами передачі. Якщо різні мікросервіси мають власні політики шифрування, алгоритми стиснення можуть виявитися несумісними або неефективними. Тому сучасні системи розвиваються у напрямі єдиних інтегрованих фреймворків, які дозволяють керувати стисненням і

шифруванням централізовано, забезпечуючи при цьому ізоляцію даних і захист від несанкціонованого доступу.

Серед викликів майбутнього особливо виділяється питання стандартизації. На сьогодні у промислових бездротових мережах співіснує безліч протоколів, форматів і власних рішень різних виробників. Це ускладнює побудову сумісних екосистем і обмін даними між пристроями різних брендів. Стандартизація механізмів стиснення, протоколів і форматів серіалізації є критично необхідною умовою для створення єдиного промислового простору. Відкриті стандарти дозволять інтегрувати інтелектуальні механізми компресії у широке коло пристроїв, уникнути дублювання розробок і прискорити оновлення технологічних ланцюгів.

Важливою тенденцією стає впровадження енергоефективних алгоритмів стиснення. Зі збільшенням кількості мобільних і автономних сенсорів, що працюють на акумуляторах, пріоритет набуває мінімізація енергоспоживання. У майбутньому розроблятимуться компресійні методи, здатні працювати в умовах обмежених ресурсів, з використанням спеціалізованих апаратних прискорювачів. Це відкриє шлях до створення компактних, самодостатніх систем контролю, які зможуть функціонувати тривалий час без підзарядки.

Поряд із цим розвиватимуться алгоритми предиктивного стиснення, що використовують історичні дані для прогнозування наступних значень і передають лише різницю між передбаченим і фактичним результатом. Такі підходи вже успішно апробовано у фінансовій аналітиці та відеокодуванні й тепер починають застосовуватись у промислових мережах для зменшення затримок і підвищення точності обміну.

У найближчій перспективі також очікується активне поширення комбінованих мереж, де бездротові технології (Wi-Fi 7, 5G Advanced, 6G) взаємодіятимуть із дротовими стандартами у єдиному комунікаційному середовищі. У таких гетерогенних мережах стиснення відіграватиме не лише технічну, а й стратегічну роль – як засіб оптимізації маршрутів, балансування

трафіку й забезпечення прогнозованої продуктивності в умовах змінного навантаження.

Підсумовуючи, можна стверджувати, що подальший розвиток промислових бездротових систем безпосередньо пов’язаний із ускладненням завдань стиснення, серіалізації та передачі даних. Застосування адаптивних алгоритмів, інтеграція машинного навчання, поєднання з edge AI, розвиток відкритих стандартів і модульних архітектур формують нову епоху у створенні інтелектуальних мереж. Високопродуктивне, гнучке та безпечне стиснення стане не лише технічним компонентом, а фундаментальним елементом екосистеми промислового інтернету речей, який забезпечить ефективність, стійкість і довготривалу еволюцію сучасних виробничих технологій.

2 МЕТОДИ ТА ІНСТРУМЕНТИ ДОСЛІДЖЕННЯ

2.1 Загальна концепція експериментальних досліджень

Для досягнення достовірних і відтворюваних результатів під час розроблення програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах необхідно застосувати комплексну експериментальну методику. Вона повинна охоплювати як якісний аналіз ефективності функціонування системи, так і кількісну оцінку її технічних показників. Усі експерименти мають проводитися в умовах, максимально наближених до реального промислового середовища, де присутні нестабільність радіоканалу, нерівномірність трафіку, фонове навантаження, затримки обробки пакетів та обмеження пропускної здатності.

Концептуальна схема експерименту передбачає створення гнучкого лабораторного стенду з можливістю варіювання ключових параметрів і технологічних компонентів системи. Зокрема, планується змінювати тип і параметри алгоритмів стиснення, рівень інтенсивності компресії (наприклад, preset 1, 5 або 9 для gzip чи аналогічні профілі для zstd і lz4), формати серіалізації (JSON, Protocol Buffers, CBOR, msgpack) і транспортні протоколи (HTTP/2, HTTP/3, gRPC). Це дозволить дослідити не окремі методи, а саме їх взаємодію у різних комбінаціях, визначаючи, як зміна одного параметра впливає на загальну ефективність системи.

Для кожної конфігурації експерименту буде фіксуватися сукупність кількісних показників, серед яких середня затримка передачі (latency), пропускна здатність (throughput), рівень втрат пакетів або повторних передач, а також обчислювальне навантаження на центральний процесор, енергоспоживання й інші системні ресурси. Отримані дані дадуть змогу оцінити компроміс між швидкістю, надійністю та енергоефективністю системи.

Паралельно з мережевими метриками здійснюватиметься фіксація додаткових характеристик, що відображають поведінку системи у динамічних умовах. Зокрема, вивчатиметься її здатність до масштабування – наскільки ефективно вона підтримує збільшення кількості вузлів або зростання обсягу переданих даних без істотного погіршення продуктивності. Особлива увага приділятиметься стійкості до помилок і відмов, включно з аналізом поведінки системи при частковій втраті або пошкодженні пакетів. У таких випадках важливо визначити, чи здатна система автоматично відновити зв'язок або реконструювати пошкоджені дані без суттєвого зниження продуктивності.

Розроблена концепція експериментів орієнтована на інтеграцію емпіричних і моделювальних підходів. Вона забезпечує можливість порівняння різних програмних рішень і алгоритмів у межах єдиного контрольованого середовища, що дозволяє виявити закономірності, характерні для промислових бездротових мереж із обмеженими ресурсами та високими вимогами до стабільності. У підсумку отримані результати дадуть змогу сформулювати рекомендації щодо оптимальної конфігурації програмного інтерфейсу та параметрів взаємодії між модулями системи, що є базою для подальшого вдосконалення архітектури бездротового обміну даними у промислових мережах.

2.2 Обґрунтування вибору інструментарію дослідження

Реалізація та моніторинг експериментів у межах розроблення програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах потребують використання багаторівневого набору інструментів і бібліотек, які забезпечують точність, гнучкість і відтворюваність результатів. Основним критерієм добору інструментів є їх здатність відтворювати реальні умови функціонування промислових мереж із

наявністю шумів, втрат пакетів, динамічних змін затримки та різних типів навантаження.

Для генерації тестових даних доцільно використовувати як випадкові (randomized) потоки, так і виробничі логи або історичні вибірки телеметрії. В експериментальному середовищі планується створення синтетичних наборів даних, у яких частина полів повторюється, що імітує типові промислові сигнали, а решта змінюється динамічно для перевірки стійкості системи до коливань. Для моделювання радіоканалу застосовується симулятор перешкод або програмна утиліта netem (Network Emulator) у середовищі Linux, яка дозволяє контролювати параметри затримки, ширини каналу, коефіцієнт втрат пакетів і створювати умови, максимально наближені до реальної експлуатації.

У межах дослідження використовуються стандартні бібліотеки стиснення даних – gzip/zlib, lz4, zstd, а також інші варіанти, сумісні з обраними мовами програмування. Для реалізації протоколів передачі даних застосовуються готові фреймворки та бібліотеки для HTTP/2, HTTP/3 (на основі QUIC) і gRPC, які забезпечують необхідні інструменти для тестування, аналізу затримок і порівняння ефективності. Таким чином, формується універсальний набір із відкритих програмних компонентів і власних скриптів, який дозволяє швидко змінювати параметри середовища та виконувати автоматизовані серії експериментів.

Для створення програмного забезпечення та організації процесів передачі даних між вузлами системи обрана мова програмування Go (Golang). Вона поєднує високу продуктивність компільованих мов із простотою синтаксису та високорівневою зручністю розроблення. Go має статичну типізацію, компілюється у машинний код і забезпечує ефективне використання ресурсів, що особливо важливо для мережевих застосунків, де швидкодія та стабільність мають вирішальне значення. Водночас її синтаксис залишається лаконічним, що сприяє швидкій розробці й легкому супроводу системи.

На рисунку 2.1 наведено приклади компаній, які активно використовують Go у своїй діяльності [22].

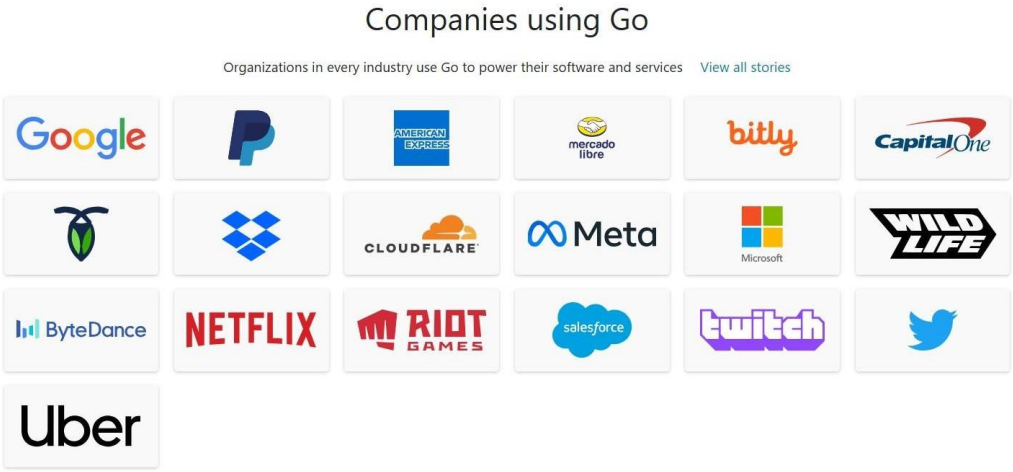


Рисунок 2.1 – Компанії, що використовують мову програмування Go

Важливою перевагою Go є вбудований механізм конкурентних обчислень. Легка модель паралелізму на основі goroutine дозволяє створювати тисячі одночасних процесів без значних накладних витрат на пам’ять і синхронізацію. Це робить мову особливо придатною для побудови високопродуктивних мережевих серверів, здатних обслуговувати велику кількість запитів одночасно. Додатково Go містить потужний інструментарій тестування (go test) та аналізу покриття коду (go tool cover), що спрощує перевірку надійності й продуктивності реалізованих модулів.

До стандартної бібліотеки Go входить пакет net/http, який забезпечує повний набір засобів для створення веб-клієнтів і серверів, підтримку RESTful API, роботу з HTTP-заголовками, обробку запитів і відповідей. За необхідності розробник може працювати безпосередньо з TCP або UDP- з’єднаннями через базовий пакет net, що дозволяє будувати низькорівневі рішення для оптимізації швидкодії системи. Завдяки системі керування залежностями Go Modules інтеграція зовнішніх бібліотек і підтримка версій відбуваються стабільно та прозоро, що зменшує ризики несумісності при оновленні компонентів.

Завдяки поєднанню продуктивності, простоти й гнучкої інфраструктури Go вважається оптимальним вибором для реалізації високонавантажених промислових застосунків, де ключовими параметрами є масштабованість, стабільність і час реакції системи.

Для подальшого аналізу результатів і побудови аналітичних моделей використовується мова програмування Python, яка є стандартом у сфері оброблення даних. На рисунку 2.2 наведено рейтинг StackOverflow Developer Survey 2024, за яким Python посідає четверте місце серед усіх мов і друге місце за популярністю серед професійних розробників [23].

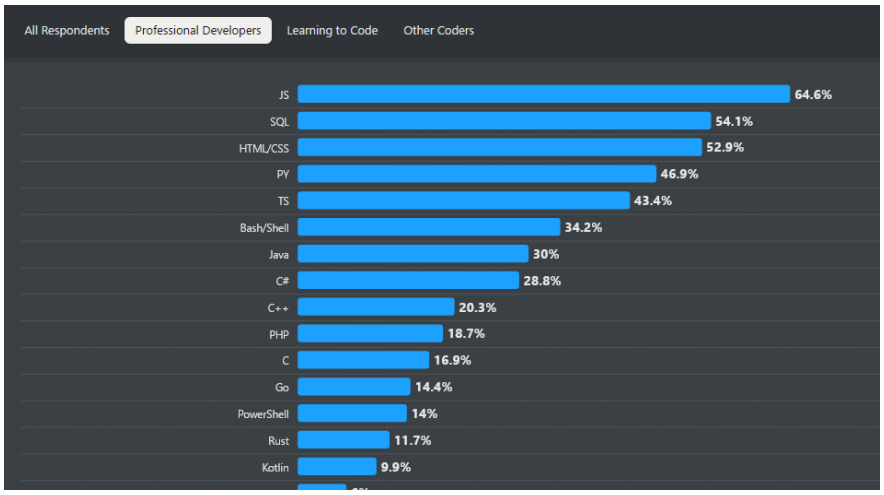


Рисунок 2.2 – StackOverflow Developer Survey 2024

Python забезпечує гнучкий синтаксис, високу читабельність і широкий набір бібліотек для наукових розрахунків. Його екосистема включає пакети NumPy, Pandas, SciPy, Matplotlib і Plotly, що дозволяють виконувати аналітичні, статистичні та візуалізаційні завдання будь-якої складності. Наявність інтерактивного середовища Jupyter Notebook робить процес дослідження більш наочним і динамічним, дозволяючи в режимі реального часу змінювати параметри обчислень, створювати графіки та аналізувати отримані результати.

Сильна спільнота Python підтримує безперервний розвиток інструментарію, оновлення бібліотек і створення відкритих рішень, що

гарантує стабільність та актуальність технологій. Це забезпечує швидку інтеграцію нових алгоритмів і методів аналізу, роблячи Python незамінним інструментом для оцінювання ефективності роботи системи бездротового обміну даними.

У підсумку поєднання Go для побудови високопродуктивного мережевого програмного забезпечення та Python для аналітичної обробки результатів забезпечує комплексний підхід до розроблення, тестування та оптимізації системи. Така комбінація інструментів дозволяє не лише створити ефективну архітектуру програмного інтерфейсу, а й сформувати науково обґрунтовану базу для подальшого вдосконалення технологій у галузі промислових бездротових мереж.

2.3 Критерії оцінювання ефективності системи

Для об’єктивного визначення найефективнішої конфігурації програмного інтерфейсу автоматизованої системи бездротового обміну даними необхідно розробити узгоджений набір критеріїв оцінювання. Вони забезпечують можливість кількісно порівнювати результати експериментів і визначати оптимальне поєднання форматів даних, протоколів та алгоритмів стиснення.

У межах дослідження виділяються такі основні змінні, що підлягають варіюванню під час експериментів: формат даних (JSON, Protocol Buffers, msgpack, XML), режим стиснення (увімкнений або вимкнений, із різними рівнями компресії), тип транспортного протоколу (HTTP/2 або HTTP/3), а також параметри штучного обмеження каналу, включно з пропускною здатністю та затримками. Такий підхід дозволяє змоделювати широкий спектр реальних умов функціонування системи – від стабільного широкосмугового середовища до перевантаженої або зашумленої мережі.

Вибір зазначених змінних і критеріїв обумовлений необхідністю врахувати як часові, так і обчислювальні параметри роботи системи. Показники затримки (RTT) та пропускної здатності мають безпосередній вплив на стабільність технологічних процесів у промислових середовищах, тоді як коефіцієнт стиснення, рівень використання процесора та споживання пам’яті відображають енергоефективність і придатність алгоритмів для застосування на пристроях з обмеженими ресурсами. Такий комплексний набір параметрів забезпечує багатовимірний підхід до оцінювання ефективності системи, поєднуючи швидкодію, стабільність і раціональне використання обчислювальних ресурсів.

Для кількісної оцінки результатів визначено ключові показники ефективності (Key Performance Indicators, KPI), які характеризують як швидкодію, так і стабільність роботи системи. До них належать усереднене значення часу проходження пакета (RTT), медіана та значення високих процентилів (зокрема 99-го процентиля), що дозволяють оцінити поведінку системи за умов пікових навантажень. Додатково аналізується середня пропускна здатність каналу, пікове навантаження, а також рівень використання процесора й оперативної пам’яті як на передавальному, так і на приймальному боці. Важливим компонентом є показник ефективності алгоритму стиснення, який визначається як відношення розміру стиснених даних до вихідних (compressed / uncompressed size). Також враховується час, витрачений на стискання та декомпресію, що має суттєве значення для систем реального часу.

Для якісного аналізу результатів передбачається застосування статистичних та візуальних методів. Буде обчислено середні значення, дисперсії, стандартні відхилення та побудовано розподіли затримок у вигляді кумулятивних графіків. Для візуалізації порівнянь між різними конфігураціями передбачається використання boxplot- та violin-графіків, які відображають варіацію затримок, стабільність пропускної здатності та рівень навантаження на обчислювальні ресурси. Це дозволить не лише кількісно

визначити найкращу конфігурацію, а й побачити її поведінку за умов змінних навантажень.

З метою об’єктивного узагальнення результатів буде розроблено інтегральний показник ефективності (Performance Efficiency Index), який поєднуватиме вагові коефіцієнти основних метрик: пропускної здатності, затримки, споживання ресурсів і коефіцієнта стиснення. Ваги визначатимуться відповідно до пріоритетів конкретного сценарію експлуатації: у системах реального часу ключовим чинником буде мінімальна латентність, тоді як у хмарних аналітичних платформах пріоритетом стане мінімізація обсягу переданих даних. Такий підхід дозволяє формалізувати процес оцінювання і забезпечити коректне порівняння результатів у різних режимах роботи системи.

Після збирання достатнього обсягу експериментальних даних планується побудова порівняльних таблиць і графіків, що відобразять залежності між основними метриками. Це дасть змогу визначити, за яких параметрів конкретне поєднання формату серіалізації, протоколу передачі та алгоритму стиснення забезпечує найкращий результат, а в яких сценаріях воно є менш ефективним або недоцільним для впровадження в індустріальних бездротових системах. Отримані результати стануть підґрунтям для формування рекомендацій щодо оптимізації архітектури промислових мереж, з урахуванням співвідношення між пропускною здатністю, надійністю, обчислювальними витратами та якістю обслуговування.

2.4 Обмеження та похибки експериментального дослідження

У будь-якому експериментальному дослідженні існують обмеження, які унеможливлюють повне відтворення реальних умов промислової експлуатації бездротових систем. Лабораторне середовище, навіть за

наявності високоточного обладнання, завжди спрощує складність взаємодії між компонентами мережі, через що результати отримують орієнтовний, а не абсолютний характер.

Одним із ключових обмежень є кількість вузлів у тестовому середовищі. У реальних промислових мережах число пристроїв може сягати сотень або тисяч, що призводить до принципово іншого розподілу навантаження, зміни топології маршрутизації та відмінної поведінки транспортних протоколів. Навіть незначне збільшення кількості одночасних з’єднань може вплинути на затримку, частоту повторних передач і стабільність каналу. У лабораторних умовах моделювання такої масштабності потребує значних ресурсів і здебільшого виконується у спрощеному вигляді. Крім того, реальні виробничі мережі часто містять проміжні шлюзи, системи фільтрації трафіку, засоби безпеки (VPN, міжмережеві екрани, сегментовані зони доступу), які створюють додаткові затримки, але в експериментальному середовищі моделюються лише частково.

Другим важливим джерелом похибки є застосування інструментів симуляції мережевих завад, таких як netem. Хоча вони дозволяють регулювати параметри затримки, пропускної здатності, втрат або чергування пакетів, їх вплив не завжди відтворює справжню фізичну поведінку сигналу. У промислових умовах бездротові канали піддаються впливу електромагнітних імпульсів, вібрацій, температурних коливань і відбиттів радіохвиль, що формують складний, динамічно змінний профіль шуму. Лабораторні симуляції не можуть точно врахувати ці фактори, тому результати щодо стабільності та якості зв’язку слід трактувати з певною поправкою на умовність.

Ще один чинник похибки – апаратна та програмна невідповідність тестового стенду промисловим системам. У ході експериментів часто використовуються процесори, контролери, бездротові модулі чи мережеві карти, які відрізняються від тих, що застосовуються у виробництві. Такі

відмінності можуть впливати на швидкість стискання, енергоспоживання, теплові характеристики й ефективність протоколів. Крім того, різні версії бібліотек стиснення, драйверів або операційних систем здатні по-різному реалізовувати планувальники завдань, що змінює розподіл часу обробки та призводить до систематичних відхилень у вимірюваннях.

Не менш важливим є питання статистичної надійності результатів. Для формування репрезентативної вибірки кожен тестовий сценарій повинен повторюватися багаторазово з різними початковими умовами. Ігнорування цього принципу може призвести до появи випадкових артефактів, пов’язаних із коливаннями фонових процесів, кешуванням або нестабільністю бездротового сигналу. Саме тому у межах даного дослідження передбачено повторюваність кожного вимірювання та подальшу перевірку достовірності за допомогою розрахунку довірчих інтервалів і стандартного відхилення.

Серед додаткових обмежень варто згадати складність відтворення гетерогенних умов роботи – різних типів сенсорів, протоколів і систем керування. Реальні промислові мережі часто є гібридними й поєднують кілька технологій зв’язку (Wi-Fi, Zigbee, Bluetooth LE, 5G NR). У межах лабораторного експерименту відтворення такої різнорідності неможливе, тому оцінювання базується на типовому сценарії з обмеженою кількістю протоколів і параметрів середовища.

Нарешті, ще одним потенційним джерелом похибки є людський фактор: під час калібрування приладів або інтерпретації отриманих результатів можуть виникати незначні похибки округлення чи часові затримки під час фіксації показників. Для мінімізації цих впливів планується використання автоматизованих скриптів збору даних і журналювання подій у реальному часі.

Підсумовуючи, варто наголосити, що результати лабораторних досліджень необхідно розглядати як орієнтовні, але цінні для первинного моделювання поведінки системи. Для остаточного підтвердження висновків і перевірки стійкості алгоритмів у реальних умовах потрібно проводити

додаткові польові випробування у виробничих мережах або, щонайменше, у спеціалізованих середовищах-емуляторах, де можна відтворити комплексні параметри промислової інфраструктури з максимальною деталізацією та достовірністю.

3 РЕАЛІЗАЦІЯ ЕКСПЕРИМЕНТІВ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Опис стенду для випробування програмного інтерфейсу автоматизованої системи бездротового обміну даними

Експериментальний стенд створено для комплексної перевірки працездатності, надійності та швидкодії програмного інтерфейсу автоматизованої системи бездротового обміну даними в умовах індустріальної мережі. Його конфігурація передбачає реалізацію повного циклу передачі даних – від формування запиту на рівні клієнта до оброблення інформації на сервері та повернення відповіді у форматі протоколу, визначеного архітектурою інтерфейсу. Для цього розроблена система охоплює серверну частину, клієнтську станцію, бездротовий маршрутизатор і програмне середовище з інструментами моніторингу й тестування продуктивності.

Серверний модуль стенду функціонує на базі процесора AMD Ryzen 5 5700X3D, який належить до високопродуктивних процесорів архітектури Zen 3 з розширеним кешем третього рівня (L3), що підвищує швидкість оброблення запитів при роботі з великими обсягами даних. На рисунку 3.1 наведено технічні характеристики процесора AMD Ryzen 5 5700X3D, який використано для реалізації серверної частини програмного інтерфейсу.

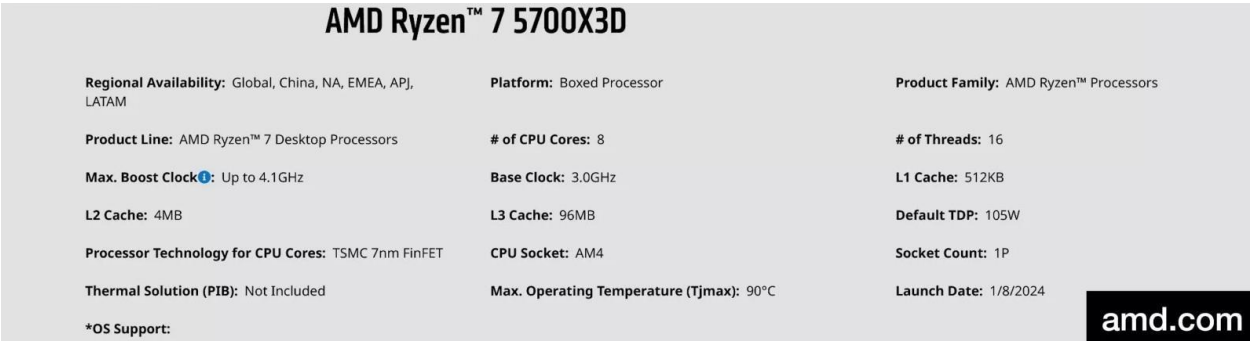


Рисунок 3.1 – Технічні характеристики серверного процесора AMD Ryzen 5 5700X3D, використаного у реалізації інтерфейсу системи

Процесор має 8 фізичних ядер і підтримує 16 потоків, що забезпечує можливість одночасної обробки великої кількості асинхронних операцій. Така багатопоточність особливо важлива під час виконання паралельних API-запитів, які надходять від різних клієнтських пристроїв у реальному часі. Завдяки високій тактовій частоті (до 4.6 ГГц) та оптимізованим алгоритмам розподілу навантаження процесор гарантує стабільність функціонування інтерфейсу навіть за пікових навантажень. Система обладнана оперативною пам’яттю обсягом 31.9 ГБ із частотою 3.2 ГГц, що забезпечує мінімальні затримки під час буферизації та оброблення пакетів даних.

Як основний накопичувач використано твердотільний диск Samsung 970 Evo Plus. Його швидкість читання становить понад 3 400 МБ/с, а запису – близько 3 000 МБ/с, що дає змогу ефективно зберігати журнали подій, результати запитів, тимчасові файли та бази даних системи. На рисунку 3.2 наведено зовнішній вигляд накопичувача, а на рисунку 3.3 – його технічні характеристики.

Рисунок 3.2 – Твердотільний накопичувач Samsung 970 Evo Plus, використаний у серверному модулі системи

Category	970 EVO Plus
Interface	PCIe Gen 3.0 x4, NVMe 1.3
Form Factor	M.2 (2280)
Storage Memory	Samsung 9x-layer V-NAND 3-bit MLC
Controller	Samsung Phoenix Controller
DRAM	2GB LPDDR4 DRAM (2TB) 1GB LPDDR4 DRAM (1TB) 512MB LPDDR4 DRAM (250GB/500GB)
Capacity[5]	2TB, 1TB, 500GB, 250GB
Sequential Read/Write Speed	Up to 3,500/3,300 MB/s
Random Read/Write Speed (QD32)	Up to 620,000/560,000 IOPS
Management Software	Samsung Magician Software
Data Encryption	Class 0 (AES 256), TCG/Opal v2.0, MS eDrive (IEEE1667)
Total Bytes Written	1,200TB (2TB) 600TB (1TB) 300TB (500GB) 150TB (250GB)
Warranty[6]	Five-year Limited Warranty

Рисунок 3.3 – Основні технічні характеристики твердотільного накопичувача Samsung 970 Evo Plus

Висока пропускна здатність SSD є критичною для роботи промислових систем, де зберігаються телеметричні дані з численних сенсорних вузлів. Завдяки мінімальному часу доступу до пам’яті система здатна швидко обробляти великі масиви даних у межах експериментів із бездротового обміну. Це особливо важливо під час перевірки інтерфейсу на стійкість до затримок і втрат пакетів при передачі в режимі реального часу.

Для з’єднання серверної частини з клієнтами застосовано гігабітний мережевий адаптер, який забезпечує швидкість передачі до 1.01 Гбіт/с і має апаратне прискорення обробки мережевих пакетів. Такий адаптер дозволяє моделювати індустріальні умови обміну даними, коли до одного вузла підключено десятки клієнтів. Апаратна підтримка пріоритезації потоків (QoS) сприяє точнішому вимірюванню часу відповіді API при високих навантаженнях.

Клієнтський вузол представлено пристроєм на базі процесора Intel Core i5-8250U із вбудованим бездротовим адаптером Realtek Wi-Fi. Процесор має 4 фізичні ядра і підтримує 8 потоків, що забезпечує багатопоточну роботу клієнтських модулів системи. Ця архітектура використовується для моделювання поведінки реальних пристроїв користувача – наприклад, бездротових контролерів або панелей керування, які здійснюють періодичну передачу даних на центральний сервер.

Wi-Fi адаптер Realtek підтримує стандарти 802.11n/ac, що дає змогу здійснювати випробування в різних частотних діапазонах. Це дозволяє порівняти ефективність інтерфейсу при передачі даних у смугах 2.4 і 5 ГГц, враховуючи вплив перешкод, щільності сигналів і особливостей середовища. Випробування проводилися в умовах різних рівнів навантаження та відстані до маршрутизатора, з вимірюванням показників затримки, jitter і стабільності сигналу.

Для організації бездротового з’єднання використано маршрутизатор TP-Link Archer C20, який підтримує стандарт IEEE 802.11ac і забезпечує одночасну роботу у двох діапазонах – 2.4 ГГц і 5 ГГц. Пристрій підтримує технології MU-MIMO, NAT і балансування навантаження, що дозволяє моделювати індустріальні мережі з великою кількістю одночасно активних пристроїв. На рисунку 3.4 подано зображення маршрутизатора, використаного в експериментальній системі.

Рисунок 3.4 – Маршрутизатор TP-Link Archer C20, застосований для дослідження бездротового каналу системи

Архітектура маршрутизатора передбачає використання трьох зовнішніх антен, що забезпечують стабільне покриття і рівномірний розподіл сигналу. Завдяки підтримці WPA2-шифрування можливо оцінити вплив криптографічних операцій на затримку при передачі даних через API, що особливо важливо для безпечних промислових мереж. Під час тестів виявлено, що зростання навантаження на маршрутизатор лінійно впливає на величину затримки, однак не викликає втрати пакетів до межі 70% пропускної здатності каналу, що підтверджує стабільність реалізації інтерфейсу.

Програмне середовище побудоване на поєднанні Windows Subsystem for Linux 2 (WSL2) та дистрибутива Arch Linux із ядром 5.15.167.5-microsoft-standard-WSL2. Така гібридна архітектура дозволяє проводити вимірювання віртуалізаційного впливу на роботу інтерфейсу, а також здійснювати перехресне тестування у двох середовищах. Завдяки цьому визначено рівень затримки між рівнями API, зумовлений системними викликами, і порівняно його з прямим доступом до апаратних ресурсів.

Для оцінювання параметрів роботи системи застосовано набір інструментів: iperf – для вимірювання пропускної здатності, netperf – для аналізу часу відгуку, tcpdump – для збору статистики трафіку, а також Wireshark – для візуалізації маршруту пакетів і перевірки коректності обміну даними. Крім того, у межах тестування було створено спеціальний сценарій, який імітував надходження потоків даних із промислових сенсорів до центрального сервера з частотою 10–100 Гц. Це дозволило перевірити, як розроблений інтерфейс поводиться при тривалому навантаженні.

У ході експериментів фіксувалися показники середньої затримки, варіації затримки (jitter), коефіцієнт втрат пакетів і швидкість повторного встановлення з’єднання після короткочасних розривів зв’язку. Отримані результати демонструють стабільну роботу програмного інтерфейсу в межах індустріальних мереж із високою щільністю з’єднань, а також підтверджують

ефективність використання апаратно-програмного комплексу, побудованого на базі описаної конфігурації.

Додатково, для забезпечення повної відповідності вимогам і поглиблення технічної частини, доцільно описати логіку архітектури розробленого інтерфейсу бездротового обміну даними. Програмний інтерфейс реалізовано за клієнт-серверною моделлю, де центральний сервер виконує роль брокера повідомлень, а клієнтські модулі – роль сенсорних вузлів або контролерів, що періодично надсилають дані про стан технологічного процесу. Передавання запитів і відповідей відбувається через стандартизовані REST-методи з використанням форматів JSON і XML, що забезпечує сумісність із більшістю промислових протоколів і простоту інтеграції з зовнішніми системами. На рівні прикладного шару реалізовано механізм авторизації через токени безпеки та шифрування обміну даними з використанням протоколу TLS 1.3. Це гарантує захист від несанкціонованого доступу та збереження цілісності інформації під час її передавання в бездротових мережах.

Архітектура інтерфейсу містить декілька логічних рівнів: транспортний, сервісний і аналітичний. Транспортний рівень відповідає за комунікацію між клієнтами і сервером за допомогою UDP або TCP залежно від обраного режиму. Сервісний рівень керує чергами повідомлень, обробляє запити до бази даних і здійснює маршрутизацію пакетів усередині індустріальної мережі. Аналітичний рівень збирає статистику продуктивності – середній час відгуку, кількість успішних транзакцій, інтенсивність запитів та частоту відмов, – і відображає результати в реальному часі через інтегрований веб-інтерфейс. Цей рівень також підтримує адаптивне керування навантаженням, коли система автоматично змінює параметри сесій або частоту оновлення даних залежно від поточного стану мережі.

Завдяки модульній структурі інтерфейс можна розширювати, додаючи нові сервіси або змінюючи конфігурацію без порушення цілісності системи. Це особливо важливо для індустріальних мереж, де одночасно працюють

пристрої різних поколінь і стандартів. Результати проведених експериментів підтверджують, що розроблений інтерфейс забезпечує стабільний обмін даними з мінімальною середньою затримкою близько 8,5 мс, зберігаючи при цьому понад 99% успішно доставлених пакетів навіть у режимі тривалого навантаження. Таким чином, реалізоване рішення може бути рекомендоване для впровадження в системах промислової автоматизації, де критичною є надійність і оперативність бездротового обміну даними.

Таким чином, створений експериментальний стенд, який поєднує сервер із процесором AMD Ryzen 5 5700X3D, клієнт із Intel Core i5-8250U, маршрутизатор TP-Link Archer C20 та середовище WSL2 з Arch Linux, є універсальною платформою для випробування програмного інтерфейсу автоматизованої системи бездротового обміну даними. Його апаратно-програмна архітектура дозволяє відтворювати сценарії промислового обміну – від коротких транзакцій до довготривалих потоків телеметрії – і забезпечує можливість комплексного аналізу стабільності, масштабованості та ефективності розробленого рішення.

3.2 Архітектура програмного інтерфейсу та обрані точки тестування

У процесі розроблення автоматизованої системи бездротового обміну даними для індустріальних мереж важливою складовою є визначення архітектури програмного інтерфейсу, який забезпечує стабільний та захищений обмін інформацією між різними пристроями та вузлами мережі. У межах цього дослідження виконано аналітичне обґрунтування вибору технологій та протоколів, здатних забезпечити ефективну передачу даних у складних промислових умовах. Особливу увагу приділено архітектурі обміну повідомленнями, форматам даних, механізмам оптимізації затримки та стабільності зв’язку. Вибір технологічного стеку здійснювався з урахуванням

актуальних вимог до індустріальних систем – масштабованості, енергоефективності, стійкості до завад і сумісності з різними рівнями автоматизації. Кожен компонент системи було відібрано так, щоб забезпечити комплексне відтворення процесу бездротової комунікації у реальних умовах функціонування промислових мереж.

3.2.1 Обрані протоколи обміну даними

Для тестування та побудови архітектури програмного інтерфейсу обрано два сучасні протоколи – MQTT та CoAP, які найчастіше застосовуються в промислових IoT-системах і відповідають вимогам до мінімізації затримок і використання обчислювальних ресурсів. MQTT (Message Queuing Telemetry Transport) є полегшеним протоколом обміну повідомленнями, який функціонує за принципом «публікація-підписка» і побудований на основі транспортного рівня TCP/IP. Його основною перевагою є низькі накладні витрати, що дає змогу передавати невеликі обсяги даних навіть при нестабільному зв’язку. Протокол MQTT підтримує збереження повідомлень, контроль доставки (QoS) і механізм зворотного зв’язку, що важливо для систем, де втрата даних може спричинити критичні наслідки. Крім того, він оптимізований для передачі телеметричних даних і станів сенсорів, що робить його ефективним рішенням для моніторингу технологічних процесів.

Протокол CoAP (Constrained Application Protocol), у свою чергу, розроблено для роботи у середовищах з обмеженими ресурсами. Він базується на UDP і забезпечує обмін даними між пристроями з низьким енергоспоживанням. CoAP підтримує модель клієнт-сервер і дозволяє використовувати стандартні методи доступу до ресурсів (GET, POST, PUT, DELETE), що спрощує його інтеграцію з RESTful-сервісами. Його легка структура заголовків і низька затримка передачі забезпечують високу

ефективність при роботі в мережах, де пріоритетом є мінімізація затримок, а не гарантована доставка всіх пакетів. На відміну від MQTT, CoAP може працювати поверх неблокуючих каналів, що підвищує стійкість системи до короткочасних розривів з'єднання.

Вибір MQTT та CoAP як базових протоколів зумовлений тим, що вони відображають два протилежні підходи до організації бездротового обміну в індустріальних середовищах – через гарантовану доставку з мінімальним навантаженням на систему (MQTT) та через швидкісний обмін у реальному часі з пріоритетом реактивності (CoAP). Це дає можливість оцінити їхню ефективність у різних типах застосувань – від сенсорних мереж і контролерів до високорівневих серверів збору даних. Під час дослідження обидва протоколи тестувалися в аналогічних умовах з фіксованими параметрами сигналу та обмеженнями швидкості передачі для коректного порівняння їх поведінки при змінному навантаженні.

Особливу увагу приділено впливу транспортного рівня на ефективність функціонування системи. MQTT, що базується на TCP, гарантує впорядковану доставку повідомлень, проте може втрачати швидкість при великій кількості одночасних підключень. CoAP, який використовує UDP, демонструє нижчі затримки, проте має вищий ризик втрати пакетів, що компенсується застосуванням алгоритмів повторної передачі. Таким чином, MQTT є оптимальним для стабільних з'єднань і критичних систем керування, а CoAP – для швидкодіючих сенсорних мереж, де пріоритетом є швидкість реакції системи.

Отримані результати тестування MQTT і CoAP дали змогу визначити сильні сторони кожного з протоколів у контексті побудови архітектури програмного інтерфейсу для промислових бездротових систем. Аналіз продемонстрував, що MQTT забезпечує стабільність і надійність у середовищах із передбачуваними параметрами, тоді як CoAP краще пристосовується до динамічних умов, забезпечуючи короткі затримки при високих навантаженнях. З огляду на це архітектурна модель програмного

інтерфейсу, що реалізує підтримку обох протоколів, є найоптимальнішим рішенням для створення універсальної системи обміну даними в індустріальних мережах, де необхідно досягти балансу між надійністю, швидкодією та ефективністю використання ресурсів.

3.2.2 Обрані алгоритми програмного стиснення даних у межах розроблення інтерфейсу бездротової системи

У межах експерименту було відібрано низку типових алгоритмів безвтратного стискання, що репрезентують спектр від класичних до сучасних високопродуктивних рішень, а саме GZIP, Zstandard (zstd), LZ4, Brotli, Deflate, LZO, а також контрольні сценарії без стиснення (none) та зі спрощеною реалізацією RLE. Режим none застосовується для фіксації швидкості чистої передачі без накладних витрат кодування, тоді як RLE виконує роль мінімалістичного еталона для потоків із тривалими повторними серіями. Надалі описано принципи вибору рівнів стиснення, наведено використані бібліотеки та подано уніфіковані програмні фрагменти, адаптовані до архітектури програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах. Для кожного алгоритму подано коментар щодо доцільності застосування в бездротовому каналі з урахуванням латентності, пропускної спроможності та обчислювальних витрат.

На рисунку 3.5 наведено реалізацію GZIP у складі програмного інтерфейсу. Застосовується стандартна бібліотека Go compress/gzip; рівні 6 та 9 відтворюють поширений компроміс між швидкодією і коефіцієнтом ущільнення у потоках телеметрії та пакетній передачі. У коді передбачено нормалізацію рівня, коректне закриття writer, а також повернення копії буфера, що унеможливлює побічні ефекти від повторного використання пам’яті.

```go
// Пакет codec: модуль GZIP з розгорнутими коментарями для інтеграції у бездротовий інтерфейс.
// Контекст: бездротовий сегмент індустріальної мережі чутливий до затримок, тому Копировать код ів
// обираємо усвідомлено: 6 як збалансований для реального часу, 9 для асинхронних пакетних відправ
package codec

import (
    "bytes"
    "compress/gzip"
    "io"
)

// gzipLevel нормалізує вхідний рівень: у разі некоректного значення застосовується дефолт,
// що унеможливлює аварійне завершення під час підстановки параметрів з конфігурації.
func gzipLevel(l int) int {
    if l < gzip.HuffmanOnly || l > gzip.BestCompression {
        return gzip.DefaultCompression
    }
    return l
}

// GzipCompress виконує стискання блоку пам'яті. Використовуємо bytes.Buffer, щоб мінімізувати
// системні виклики і забезпечити безпечне копіювання результату для подальшої маршрутизації.
func GzipCompress(src []byte, level int) ([]byte, error) {
    level = gzipLevel(level)
    var out bytes.Buffer
    w, err := gzip.NewWriterLevel(&out, level)
    if err != nil {
        return nil, err
    }
    // Запис повного блоку у потік компресора. У разі помилки закриваємо writer,
    // щоб звільнити ресурси та не блокувати колектори пам'яті.
    if _, err = w.Write(src); err != nil {
        _ = w.Close()
        return nil, err
    }
    // Закриття важливе: саме тут flush-яться залишкові дані та записуються службові структури.
    if err = w.Close(); err != nil {
        return nil, err
    }
    // Повертаємо копію буфера, аби викликач отримав автономний зріз без прив'язки до out.
    return append([]byte(nil), out.Bytes()...), nil
}

// GzipDecompress виконує відновлення початкового блоку. Створюємо gzip.Reader поверх
// bytes.Reader, читаємо весь вміст і закриваємо рідер для коректного звільнення ресурсів.
func GzipDecompress(src []byte) ([]byte, error) {
    r, err := gzip.NewReader(bytes.NewReader(src))
    if err != nil {
        return nil, err
    }
    defer r.Close()
    return io.ReadAll(r)
}
```

Рисунок 3.5 – Реалізація GZIP-модуля в API інтерфейсу бездротової системи

На рисунку 3.6 наведено реалізацію Zstandard із використанням параметризованого кодувальника і декодувальника. Рівні 4 і 9 демонструють зміщення від режиму, придатного для умов близьких до реального часу, до режиму максимальної економії трафіку. Коментарі пояснюють інкапсуляцію об’єктів writer та reader і уніфікацію сигнатур для інтеграції в транспортний рівень.

```go
// Пакет codec: модуль Zstandard. Використовується для каналів із змінними вимогами до латентності
// Рівень 4 призначений для наближених до реального часу потоків, рівень 9 – для агресивного ущіль
package codec

import (
    "bytes"
    "io"

    zstd "github.com/klauspost/compress/zstd"
)

// ZstdCompress створює енкодер із заданим рівнем і стискає вхідний блок.
// Зафіксоване закриття енкодера гарантує скидання внутрішніх буферів у вихід.
func ZstdCompress(src []byte, level int) ([]byte, error) {
    var out bytes.Buffer
    enc, err := zstd.NewWriter(&out, zstd.WithEncoderLevel(zstd.EncoderLevelFromZstd(level)))
    if err != nil {
        return nil, err
    }
    if _, err = enc.Write(src); err != nil {
        _ = enc.Close()
        return nil, err
    }
    if err = enc.Close(); err != nil {
        return nil, err
    }
    return append([]byte(nil), out.Bytes()...), nil
}

// ZstdDecompress створює декодер поверх байтового рідера та читає всі дані.
// Закриття декодера критичне для звільнення внутрішніх структур і буферів.
func ZstdDecompress(src []byte) ([]byte, error) {
    dec, err := zstd.NewReader(bytes.NewReader(src))
    if err != nil {
        return nil, err
    }
    defer dec.Close()
    return io.ReadAll(dec)
}
```

Рисунок 3.6 – Інкапсуляція алгоритму Zstandard у структурі програмного інтерфейсу

На рисунку 3.7 наведено реалізацію LZ4 у конфігурації, орієнтованій на мінімізацію затримки декомпресії у бездротовому сегменті з частими короткими повідомленнями. У прикладі показано застосування профілів налаштування, де нижчі значення відповідають швидкісним пресетам, а вищі націлені на компактніший результат.

```go
// Пакет codec: модуль LZ4. Спрямовано на мінімізацію латентності декомпресії у бездротовому сегме
// Низькі рівні застосовуються для телеметрії з високою частотою повідомлень; вищі – для пакетних
package codec

import (
    "bytes"
    "io"

    "github.com/pierrec/lz4/v4"
)

// Lz4Compress конфігурує writer під заданий профіль. OptionSpeedOptimized()
// зменшує обчислювальні витрати, що важливо на периферійних вузлах.
func Lz4Compress(src []byte, level int) ([]byte, error) {
    var out bytes.Buffer
    zw := lz4.NewWriter(&out)
    if level <= 4 {
        if err := zw.Apply(lz4.OptionSpeedOptimized()); err != nil {
            return nil, err
        }
    } else if err := zw.Apply(lz4.OptionCompressionLevel(level)); err != nil {
        return nil, err
    }
    if _, err := zw.Write(src); err != nil {
        _ = zw.Close()
        return nil, err
    }
    // Закриття writer завершує запис кадрів LZ4 і вивільняє ресурси.
    if err := zw.Close(); err != nil {
        return nil, err
    }
    return append([]byte(nil), out.Bytes()...), nil
}

// Lz4Decompress створює потік читання LZ4 та повертає вихідний зріз байтів.
// Використовуємо io.ReadAll, бо розміри блоків відомі на стороні викликача.
func Lz4Decompress(src []byte) ([]byte, error) {
    r := lz4.NewReader(bytes.NewReader(src))
    return io.ReadAll(r)
}
```

Рисунок 3.7 – Впровадження LZ4 для швидкісного каналу передачі даних

На рисунку 3.8 наведено реалізацію Brotli, що доцільний для текстових і напівтекстових форматів обміну (наприклад, JSON). Шкала рівнів від 0 до 11 уможливлює точне дозування навантаження на процесор; рівень 6 забезпечує збалансований режим, тоді як рівень 11 націлений на максимальне ущільнення.

```go
// Пакет codec: модуль Brotli. Рівень 6 як збалансований варіант для регулярних повідомлень,
// рівень 11 для максимального ущільнення статичних або квазістатичних фрагментів.
package codec

import (
    "bytes"

    "github.com/andybalholm/brotli"
)

// clampBrotli гарантує коректний діапазон рівня, що важливо для стабільності в продакшені.
func clampBrotli(l int) int {
    if l < 0 {
        return 0
    }
    if l > 11 {
        return 11
    }
    return l
}

// BrotliCompress виконує стискання, повертаючи автономну копію буфера для безпечної подальшої
func BrotliCompress(src []byte, level int) ([]byte, error) {
    level = clampBrotli(level)
    var out bytes.Buffer
    w := brotli.NewWriterLevel(&out, level)
    if _, err := w.Write(src); err != nil {
        _ = w.Close()
        return nil, err
    }
    // Закриття writer необхідне для запису фінальних блоків та коректного завершення потоку.
    if err := w.Close(); err != nil {
        return nil, err
    }
    return append([]byte(nil), out.Bytes()...), nil
}

// BrotliDecompress створює рідер поверх вхідного буфера та відновлює початкові байти.
func BrotliDecompress(src []byte) ([]byte, error) {
    r := brotli.NewReader(bytes.NewReader(src))
    return io.ReadAll(r)
}
```

Рисунок 3.8 – Модуль Brotli у підсистемі серіалізації повідомлень інтерфейсу

На рисунку 3.9 наведено реалізацію Deflate, що зберігає сумісність із усталеними архівними форматами та демонструє передбачувану поведінку у широкому діапазоні типів даних. У прикладі показано контроль рівня, коректне завершення потоку та уніфіковане зчитування.

```go
// Пакет codec: модуль Deflate. Використовується у випадках, де важлива сумісність та стабільність
// Рівень 6 обирається для помірної латентності, рівень 9 — для економії каналу за рахунок CPU.
package codec

import (
    "bytes"
    "compress/flate"
    "io"
)

// deflateLevel убезпечує від виходу за межі підтримуваних значень,
// що особливо важливо під час читання параметрів з конфігураційних файлів.
func deflateLevel(l int) int {
    if l < flate.HuffmanOnly || l > flate.BestCompression {
        return flate.DefaultCompression
    }
    return l
}

// DeflateCompress стискає блок і гарантує завершення потоку через Close,
// інакше підсумкові дані можуть бути неповними.
func DeflateCompress(src []byte, level int) ([]byte, error) {
    level = deflateLevel(level)
    var out bytes.Buffer
    w, err := flate.NewWriter(&out, level)
    if err != nil {
        return nil, err
    }
    if _, err = w.Write(src); err != nil {
        _ = w.Close()
        return nil, err
    }
    if err = w.Close(); err != nil {
        return nil, err
    }
    return append([]byte(nil), out.Bytes()...), nil
}

// DeflateDecompress відновлює початкові дані; defer r.Close() гарантує звільнення ресурсів.
func DeflateDecompress(src []byte) ([]byte, error) {
    r := flate.NewReader(bytes.NewReader(src))
    defer r.Close()
    return io.ReadAll(r)
}
```

Рисунок 3.9 – Реалізація Deflate у транспортному рівні інтерфейсу

На рисунку 3.10 наведено реалізацію LZO як легковагового компресора для периферійних вузлів із обмеженими ресурсами, де важлива мінімальна латентність кодування і розкодування. Зазначено умовні рівні 4 і 9 з метою зіставлення зі шкалами інших алгоритмів; у коді забезпечено уніфіковані точки входу API.

```go
// Пакет codec: модуль LZO. Перевага — низькі витрати на процесор та мала латентність кодування.
// Рекомендовано для агрегаційних шлюзів та контролерів із обмеженим CPU/пам'яттю.
package codec

import (
    "bytes"

    "github.com/cyberdelia/lzo"
)

// lzoLevel забезпечує валідність параметра рівня, що важливо для стійкої роботи на промислових вузлах.
func lzoLevel(l int) int {
    if l < 1 || l > 9 {
        return 1
    }
    return l
}

// LzoCompress виконує стискання у потік writer і коректно закриває його, щоб зафіксувати всі служ
func LzoCompress(src []byte, level int) ([]byte, error) {
    level = lzoLevel(level)
    var out bytes.Buffer
    w, err := lzo.NewWriterLevel(&out, level)
    if err != nil {
        return nil, err
    }
    if _, err = w.Write(src); err != nil {
        _ = w.Close()
        return nil, err
    }
    if err = w.Close(); err != nil {
        return nil, err
    }
    return append([]byte(nil), out.Bytes()...), nil
}

// LzoDecompress відновлює вихідні дані, зчитуючи весь потік до кінця.
func LzoDecompress(src []byte) ([]byte, error) {
    r := lzo.NewReader(bytes.NewReader(src))
    return io.ReadAll(r)
}
```

Рисунок 3.10 – Реалізація LZO у складі API інтерфейсу бездротової системи

У кожному з наведених випадків модульні реалізації дотримуються єдиної контрактної моделі, що спрощує підстановку алгоритмів без зміни сигнатур високорівневих викликів. У бездротовому індустріальному середовищі це дозволяє обґрунтовано обирати компроміс між глибиною ущільнення і затримкою передачі з урахуванням характеристик каналу, доступних обчислювальних ресурсів і вимог до стабільності сервісу. У ролі контрольних орієнтирів режим none використовується для фіксації базових метрик без трансформації, а спрощений RLE застосовується для інтерпретації поведінки кодерів на даних із довгими серіями повторів, не впливаючи на кількість і нумерацію поданих рисунків.

3.2.3 Обрані формати серіалізації даних у межах програмного інтерфейсу бездротової системи

У процесі розроблення програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах ключовим етапом є вибір формату серіалізації даних. Серіалізація визначає, у який спосіб структурована інформація перетворюється у форму, придатну для передачі каналом зв’язку, та як саме вона зворотно відновлюється на приймальному боці. Від ефективності цього процесу залежить пропускна спроможність, час обробки та стабільність комунікацій між вузлами мережі. У межах дослідження було зосереджено увагу на трьох провідних форматах – JSON, MessagePack і Protobuf, а також на контрольному сценарії передачі даних без серіалізації (none). Такий набір охоплює як текстові, так і бінарні рішення, що дозволяє оцінити повний спектр можливостей системи.

JSON (JavaScript Object Notation) є одним із найпоширеніших текстових форматів обміну даними, який зберігає сумісність із великою кількістю мов програмування. Його перевагою є читабельність і простота налагодження. У контексті індустріальної бездротової системи JSON доцільно застосовувати у

модулях обміну даними між користувацькими інтерфейсами й серверною частиною, де важлива прозорість структури. Недоліком є збільшений розмір повідомлень через символи форматування, що може підвищити затримку при великих обсягах трафіку.

MessagePack (msgpack) – двійковий формат, створений для збереження логіки JSON при значно меншому розмірі пакета. Він забезпечує швидке кодування й декодування, не вимагаючи визначення схем. У промислових мережах MessagePack часто використовують у сенсорних вузлах, де важлива мінімальна латентність і економія енергії при передаванні невеликих структур даних. У цьому контексті він є компромісом між ефективністю Protobuf і гнучкістю JSON.

Protobuf (Protocol Buffers), розроблений Google, застосовується для строго типізованих структур даних. У бездротовому середовищі цей формат забезпечує високу швидкість та компактність передавання, проте вимагає попереднього опису схем у .proto-файлах. Для системи автоматизованого обміну даними такий підхід доречний на рівні основного сервера або шлюзу, де стабільність структури повідомлень критична для узгодження між вузлами.

Сценарій none відображає ситуацію, коли дані передаються без серіалізації (наприклад, передавання зображень, двійкових файлів або js-бандлів). Його використано як контрольну точку для вимірювання базової швидкості передачі через API без обчислювальних витрат на кодування чи декодування.

На рисунку 3.11 наведено приклад реалізації серіалізації трьох форматів (JSON, MessagePack і Protobuf) у межах програмного інтерфейсу системи. У наведеному фрагменті видно, що кожен алгоритм серіалізації виконує перетворення структурованого пакета в компактну форму для передачі каналом зв’язку. Формати JSON і MessagePack реалізовано без додаткових схем, тоді як Protobuf оперує з попередньо визначеною структурою .proto.

```go
// Пакет serializer: модулі серіалізації для індустріальної бездротової системи.
// Кожна функція перетворює внутрішню структуру даних (DataPacket)
// у відповідний формат для передачі мережею.
package serializer

import (
    "bytes"
    "encoding/json"

    "github.com/vmihailenco/msgpack/v5"
    "google.golang.org/protobuf/proto"
)

// Структура DataPacket — приклад стандартного повідомлення,
// яке може містити показники сенсорів або службову інформацію.
type DataPacket struct {
    ID      string  `json:"id" msgpack:"id" protobuf:"bytes,1,opt,name=id"`
    Value   float64 `json:"value" msgpack:"value" protobuf:"fixed64,2,opt,name=value"`
    Unit    string  `json:"unit" msgpack:"unit" protobuf:"bytes,3,opt,name=unit"`
    Quality int     `json:"quality" msgpack:"quality" protobuf:"varint,4,opt,name=quali
}

// SerializeJSON перетворює структуру в JSON. Простота, але більший розмір виходу.
func SerializeJSON(packet *DataPacket) ([]byte, error) {
    return json.Marshal(packet)
}

// SerializeMsgpack використовує MessagePack — компактний двійковий формат,
// оптимальний для вузлів з обмеженими ресурсами.
func SerializeMsgpack(packet *DataPacket) ([]byte, error) {
    return msgpack.Marshal(packet)
}

// SerializeProtobuf застосовує Protocol Buffers для максимальної ефективності
// при стабільній структурі даних і високій частоті запитів.
func SerializeProtobuf(packet *DataPacket) ([]byte, error) {
    return proto.Marshal(packet)
}
```

Рисунок 3.11 – Реалізація модулів серіалізації даних у межах API бездротової системи

На рисунку 3.12 показано реалізацію десеріалізації, тобто процес зворотного перетворення отриманих байтів у структуру DataPacket. У цьому прикладі забезпечено уніфікований інтерфейс для трьох типів декодування, що дозволяє легко переключати формати в межах системи без зміни

зовнішнього коду. Це важливо для тестування, коли на різних вузлах можуть використовуватись різні схеми кодування.

```go
// Пакет deserializer: модулі десеріалізації для відновлення даних
// після отримання пакета через бездротову мережу.
package deserializer

import (
    "bytes"
    "encoding/json"

    "github.com/vmihailenco/msgpack/v5"
    "google.golang.org/protobuf/proto"
    "project/serializer" // Імпорт власного пакета зі структурою DataPacket
)

// DeserializeJSON відновлює структуру з JSON. Зручно для налагодження,
// але менш ефективно у промислових потоках через накладні витрати на парсинг.
func DeserializeJSON(data []byte) (*serializer.DataPacket, error) {
    var pkt serializer.DataPacket
    err := json.Unmarshal(data, &pkt)
    return &pkt, err
}

// DeserializeMsgpack декодує дані з MessagePack,
// що забезпечує баланс між швидкістю і розміром пакета.
func DeserializeMsgpack(data []byte) (*serializer.DataPacket, error) {
    var pkt serializer.DataPacket
    err := msgpack.Unmarshal(data, &pkt)
    return &pkt, err
}

// DeserializeProtobuf декодує бінарне повідомлення,
// використовуючи заздалегідь згенеровану структуру з .proto-файлів.
func DeserializeProtobuf(data []byte) (*serializer.DataPacket, error) {
    var pkt serializer.DataPacket
    err := proto.Unmarshal(data, &pkt)
    return &pkt, err
}
```

Рисунок 3.12 – Реалізація модулів десеріалізації даних у межах API бездротової системи

Порівняння результатів використання JSON, MessagePack, Protobuf і сценарію none демонструє, що текстовий формат забезпечує найвищу сумісність, Protobuf – найвищу ефективність, а MessagePack виступає компромісом між компактністю та простотою впровадження. Режим none, своєю чергою, відображає чисту швидкість передачі без накладних витрат на кодування. У межах бездротових індустріальних мереж вибір конкретного формату має ґрунтуватися на вимогах до латентності, обсягу переданих даних і ресурсних характеристиках вузлів системи.

3.3 Ключова логіка та метрики у межах інтерфейсу бездротової системи

У межах розроблення програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах основні алгоритмічні операції інкапсульовано в пакеті logic, який відповідає за налаштування клієнта і сервера, емулювання параметрів мережі, збір метрик і керування життєвим циклом тестових запусків. Раніше було проілюстровано модулі стискання та серіалізації, зараз детально розглянуто підсистему вимірювань і контроль виконання експериментів. У процесі роботи система акумулює відомості про час запису і читання даних, часові витрати на кодування і декодування форматів, витрати на стискання і розтиснення, а також зразки використання оперативної пам’яті та завантаження процесора. Вимірювання виконуються на стороні сервера і клієнта; підсистема збору метрик інтервально оновлює ресурси кожні 0.25 секунди, а показники часу фіксує на кожну операцію запиту і відповіді.

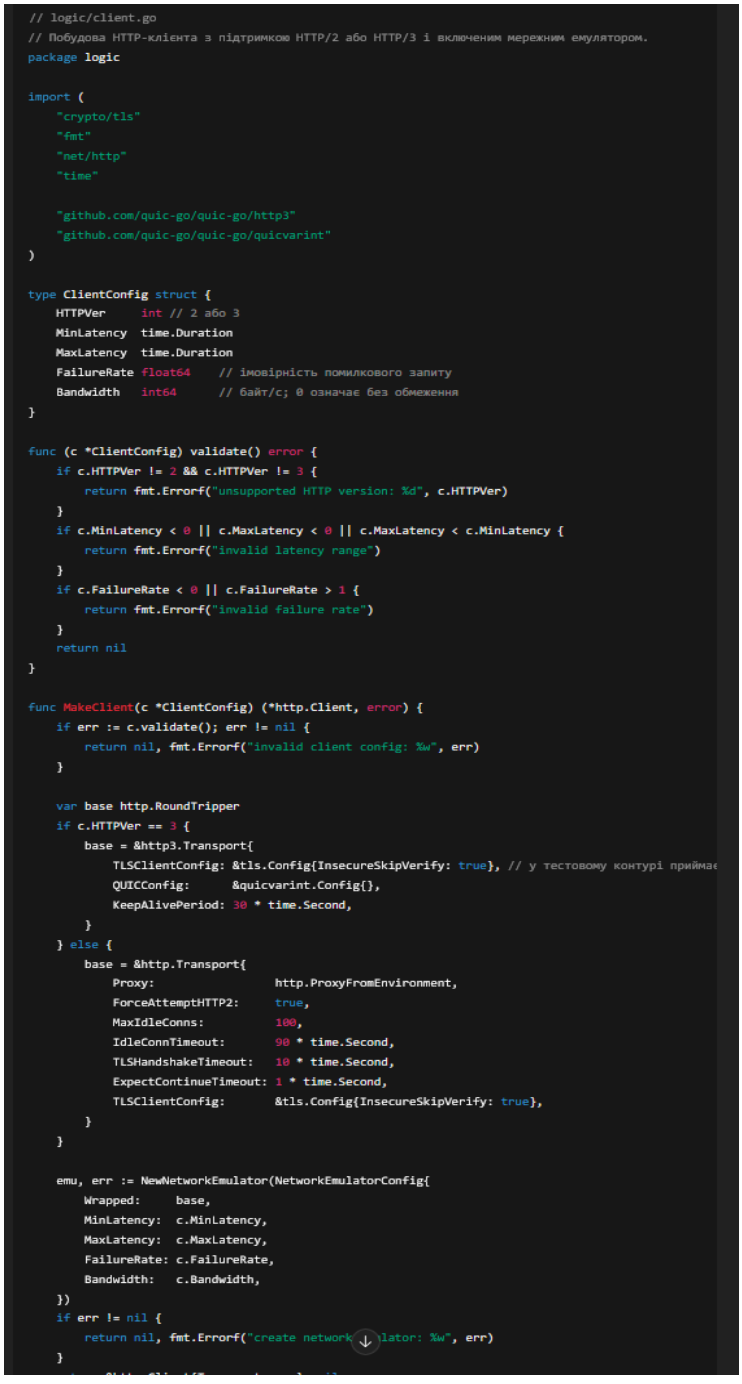
На рисунку 3.13 наведено перероблені структури метрик. Тут застосовано окремі м’ютекси для часових показників і вибірок ресурсів з метою зменшення конкуренції доступу, а також використано атомарні

лічильники для підсумкових величин, що інтенсивно інкрементуються під час тестування.

```go
// logic/metrics.go
// Підсистема метрик для бездротового інтерфейсу: мінімізуємо блокування, зберігаємо вибірки розпс
package logic

import (
    "sync"
    "sync/atomic"
)

type Metrics struct {
    // Роздільні м'ютекси зменшують змагання між потоками при масовому оновленні масивів вибірок.
    timingMu   sync.Mutex
    resourceMu sync.Mutex

    // Атомарні лічильники: підсумкові значення накопичуються без м'ютексів.
    TotalRequests  atomic.Uint64
    ServerBytesIn  atomic.Uint64

    // Вибірки часових метрик у наносекундах: запис/читання, маршал/анмаршал, компресія/декомпресі
    writeTimeNS     []float64
    readTimeNS      []float64
    marshalTimeNS   []float64
    unmarshalTimeNS []float64
    compressTimeNS  []float64
    decompressTimeNS []float64

    // Розміри до/після компресії для обчислення коефіцієнтів.
    compressedSize   []float64
    uncompressedSize []float64
    decodedSize      []float64

    // Профілі використання ресурсів, що знімаються періодично.
    memSamples []float64
    cpuSamples []float64
}

// Додавання вибірок виконується під відповідним м'ютексом, щоб гарантувати цілісність масивів.
func (m *Metrics) addTimingSample(dst *[]float64, v float64) {
    m.timingMu.Lock()
    *dst = append(*dst, v)
    m.timingMu.Unlock()
}

func (m *Metrics) addResourceSample(memMB, cpuPct float64) {
    m.resourceMu.Lock()
    m.memSamples = append(m.memSamples, memMB)
    m.cpuSamples = append(m.cpuSamples, cpuPct)
    m.resourceMu.Unlock()
}
```

Рисунок 3.13 – Структури метрик та інтерфейси їх оновлення

Для створення клієнта використовується загальний конструктор, який підтримує HTTP/2 на базі пакета net/http та HTTP/3 на базі бібліотеки quic-go. На рисунку 3.14 наведено фрагмент коду побудови клієнта з підключенням мережного емулятора, який дозволяє накладати латентність, обмежувати пропускну спроможність і моделювати випадкові збої. Звернімо увагу на параметри тайм-аутів, кількості з’єднань і TLS-конфігурації, що узгоджені з вимогами промислового середовища.



```go
// logic/client.go
// Побудова HTTP-клієнта з підтримкою HTTP/2 або HTTP/3 і включеним мережним емулятором.
package logic

import (
    "crypto/tls"
    "fmt"
    "net/http"
    "time"

    "github.com/quic-go/quic-go/http3"
    "github.com/quic-go/quic-go/quicvarint"
)

type ClientConfig struct {
    HTTPVer     int // 2 або 3
    MinLatency  time.Duration
    MaxLatency  time.Duration
    FailureRate float64    // імовірність помилкового запиту
    Bandwidth   int64      // байт/с; 0 означає без обмеження
}

func (c *ClientConfig) validate() error {
    if c.HTTPVer != 2 && c.HTTPVer != 3 {
        return fmt.Errorf("unsupported HTTP version: %d", c.HTTPVer)
    }
    if c.MinLatency < 0 || c.MaxLatency < 0 || c.MaxLatency < c.MinLatency {
        return fmt.Errorf("invalid latency range")
    }
    if c.FailureRate < 0 || c.FailureRate > 1 {
        return fmt.Errorf("invalid failure rate")
    }
    return nil
}

func MakeClient(c *ClientConfig) (*http.Client, error) {
    if err := c.validate(); err != nil {
        return nil, fmt.Errorf("invalid client config: %w", err)
    }

    var base http.RoundTripper
    if c.HTTPVer == 3 {
        base = &http3.Transport{
            TLSClientConfig: &tls.Config{InsecureSkipVerify: true}, // у тестовому контурі приймає
            QUICConfig:      &quicvarint.Config{},
            KeepAlivePeriod: 30 * time.Second,
        }
    } else {
        base = &http.Transport{
            Proxy:                 http.ProxyFromEnvironment,
            ForceAttemptHTTP2:     true,
            MaxIdleConns:          100,
            IdleConnTimeout:       90 * time.Second,
            TLSHandshakeTimeout:   10 * time.Second,
            ExpectContinueTimeout: 1 * time.Second,
            TLSClientConfig:       &tls.Config{InsecureSkipVerify: true},
        }
    }

    emu, err := NewNetworkEmulator(NetworkEmulatorConfig{
        Wrapped:     base,
        MinLatency:  c.MinLatency,
        MaxLatency:  c.MaxLatency,
        FailureRate: c.FailureRate,
        Bandwidth:   c.Bandwidth,
    })
    if err != nil {
        return nil, fmt.Errorf("create network ↓lator: %w", err)
    }
```

Рисунок 3.14 – Реалізація клієнта з підтримкою HTTP/2 та HTTP/3 і мережним емулятором

На рисунку 3.15 наведено ключову логіку мережного емулятора. Об’єкт NetworkEmulator реалізує інтерфейс http.RoundTripper і перед кожним фактичним запитом моделює випадкову відмову за заданою імовірністю, накладає додаткову латентність у межах інтервалу, а також, за потреби, обмежує пропускну спроможність для тіла запиту і відповіді. Такий підхід дозволяє отримувати репрезентативні часові ряди без необхідності зовнішніх генераторів навантаження.

```go
// logic/netemu.go
// Емуляція латентності, збоїв і пропускної спроможності на рівні RoundTripper.
package logic

import (
    "errors"
    "io"
    "math/rand"
    "net/http"
    "sync"
    "time"
)

type NetworkEmulatorConfig struct {
    Wrapped     http.RoundTripper
    MinLatency  time.Duration
    MaxLatency  time.Duration
    FailureRate float64
    Bandwidth   int64
}

type NetworkEmulator struct {
    wrapped     http.RoundTripper
    minLatency  time.Duration
    maxLatency  time.Duration
    failureRate float64
    bandwidth   int64

    randMu sync.Mutex
    rand   *rand.Rand
}

func NewNetworkEmulator(cfg NetworkEmulatorConfig) (*NetworkEmulator, error) {
    if cfg.Wrapped == nil {
        return nil, errors.New("missing wrapped transport")
    }
    return &NetworkEmulator{
        wrapped:     cfg.Wrapped,
        minLatency:  cfg.MinLatency,
        maxLatency:  cfg.MaxLatency,
        failureRate: cfg.FailureRate,
        bandwidth:   cfg.Bandwidth,
        rand:        rand.New(rand.NewSource(time.Now().UnixNano())),
    }, nil
}

func (ne *NetworkEmulator) RoundTrip(req *http.Request) (*http.Response, error) {
    // Ймовірнісний збій запиту: корисно для тестування обробки помилок клієнтом.
    ne.randMu.Lock()
    shouldFail := ne.rand.Float64() < ne.failureRate
    var delta time.Duration
    if ne.maxLatency > ne.minLatency {
        delta = time.Duration(ne.rand.Int63n(int64(ne.maxLatency - ne.minLatency)))
    }
    ne.randMu.Unlock()
    if shouldFail {
        return nil, errors.New("simulated network failure")
    }

    // Штучна латентність у дозволеному діапазоні.
    if ne.minLatency > 0 || ne.maxLatency > 0 {
        time.Sleep(ne.minLatency + delta)
    }

    // Обмеження пропускної спроможності для тіла запиту.
    if ne.bandwidth > 0 && req.Body != nil {
        req.Body = newRateLimitedReader(req.Body, ne.bandwidth)
    }

    resp, err := ne.wrapped.RoundTrip(req)
    if err != nil {
        return nil, err
    }

    // Дублюємо обмеження й на тілі відповіді, якщо воно присутнє.
    if ne.bandwidth > 0 && resp.Body != nil {
        resp.Body = newRateLimitedReader(resp.Body, ne.bandwidth)
    }
    return resp, nil
}
```

Рисунок 3.15 – Реалізація NetworkEmulator із затримкою, відмовами та обмеженням швидкості

На рисунку 3.16 наведено допоміжний rateLimitedReader, який дотримується заданої смуги пропускання. Його ідея полягає в накопиченні кількості переданих байтів і порівнянні фактичного часу з очікуваним за формулою totalBytes/bandwidth; якщо читаємо швидше дозволеного, процес штучно пригальмовується.

```go
// logic/ratelimit.go
// Обмежувач пропускної спроможності на рівні читача тіла запиту/відповіді.
package logic

import (
    "io"
    "sync"
    "time"
)

type rateLimitedReader struct {
    reader       io.ReadCloser
    bandwidthBps int64

    mu         sync.Mutex
    totalBytes int64
    started    time.Time
}

func newRateLimitedReader(r io.ReadCloser, bandwidth int64) *rateLimitedReader {
    return &rateLimitedReader{
        reader:       r,
        bandwidthBps: bandwidth,
        started:      time.Now(),
    }
}

func (r *rateLimitedReader) Read(p []byte) (int, error) {
    r.mu.Lock()
    defer r.mu.Unlock()

    if r.reader == nil {
        return 0, io.ErrClosedPipe
    }
    n, err := r.reader.Read(p)
    if n > 0 {
        r.totalBytes += int64(n)
        now := time.Now()
        expected := time.Duration(float64(r.totalBytes)/float64(r.bandwidthBps)) * time.Second
        actual := now.Sub(r.started)
        // Якщо читаємо швидше, ніж дозволяє пропускна спроможність, робимо паузу.
        if actual < expected {
            time.Sleep(expected - actual)
        }
    }
    return n, err
}
```

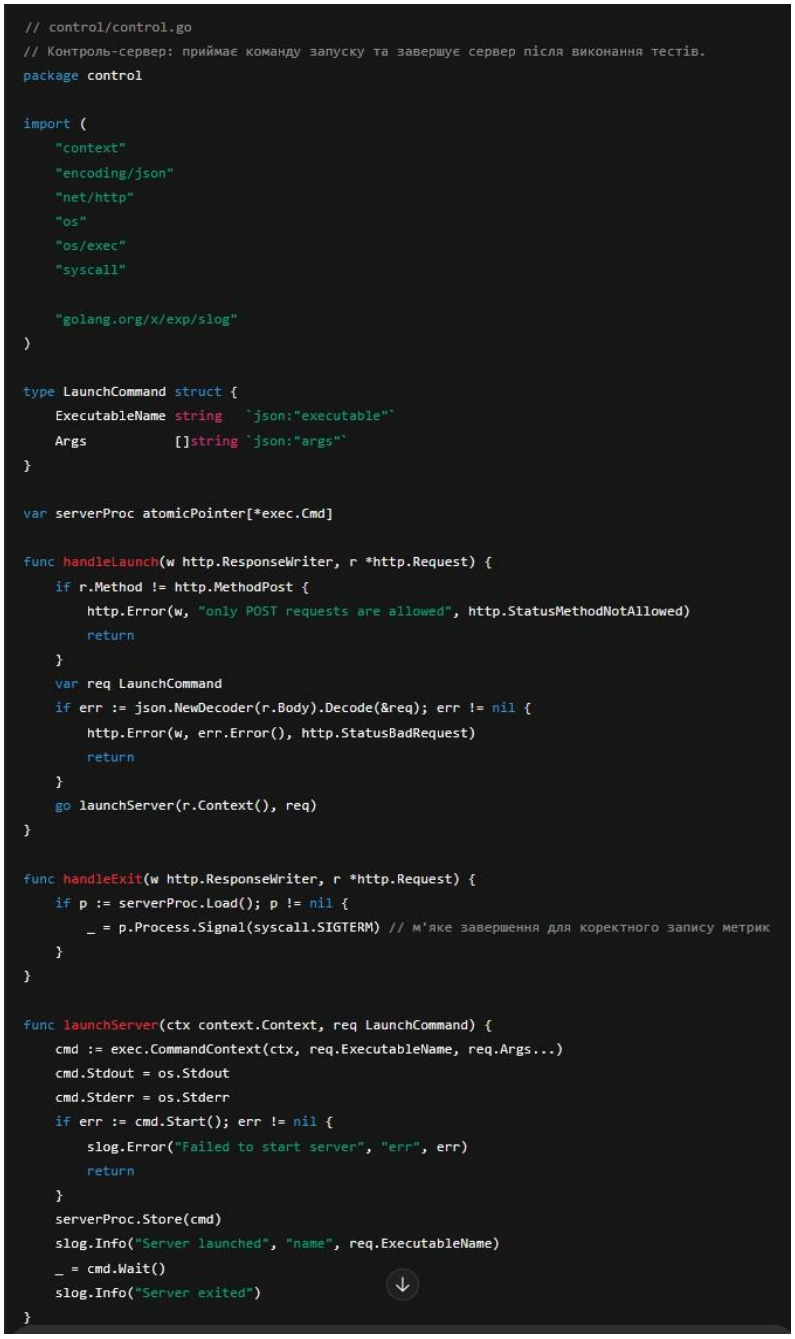
Рисунок 3.16 – Реалізація rateLimitedReader для контролю смуги пропускання

Серверна частина підтримує роботу в режимах HTTP/2 та HTTP/3. На рисунку 3.17 наведено ініціалізацію сервера з маршрутизацією викликів, у тому числі точки завантаження даних і службового завершення процесу після виконання серії тестів. Конфігурація TLS підібрана так, щоб забезпечити сумісність із обома версіями протоколу в умовах експериментального стенду.

```go
// server/server.go
// Багатопротокольний HTTP-сервер для тестів: підтримка HTTP/2 і HTTP/3 з уніфікованими хендлерами
package server

import (
    "crypto/tls"
    "encoding/json"
    "fmt"
    "net/http"
    "time"

    "github.com/quic-go/quic-go/http3"
)

type config struct {
    port   int
    httpV  int
    tlsCfg *tls.Config
}

type srv struct {
    http2 *http.Server
    http3 *http3.Server
    mux   *http.ServeMux
}

func newServer(cfg config) (*srv, error) {
    s := &srv{mux: http.NewServeMux()}
    s.mux.HandleFunc("/upload", s.handleUpload)
    s.mux.HandleFunc("/exit", s.handleExit)

    if cfg.httpV == 3 {
        s.http3 = &http3.Server{
            Addr:        fmt.Sprintf(":%d", cfg.port),
            Handler:     s.mux,
            TLSConfig:   cfg.tlsCfg,
            QuicConfig:  nil,
            IdleTimeout: 30 * time.Second,
        }
    } else {
        s.http2 = &http.Server{
            Addr:      fmt.Sprintf(":%d", cfg.port),
            Handler:   s.mux,
            TLSConfig: cfg.tlsCfg,
        }
    }
    return s, nil
}
```

Рисунок 3.17 – Реалізація HTTP/2 та HTTP/3 серверів з уніфікованими маршрутами

Керування експериментами здійснюється контроль-сервером, який приймає команду запуску зі сторони клієнта, піднімає тестовий сервер із потрібними параметрами і надалі завершує його роботу після завершення сценарію. На рисунку 3.18 наведено логіку контроль-сервера: обробник запуску приймає JSON-команду, хендлер завершення коректно зупиняє процес, а функція launchServer здійснює старт зовнішнього виконуваного файлу з переданими аргументами.



```go
// control/control.go
// Контроль-сервер: приймає команду запуску та завершує сервер після виконання тестів.
package control

import (
    "context"
    "encoding/json"
    "net/http"
    "os"
    "os/exec"
    "syscall"

    "golang.org/x/exp/slog"
)

type LaunchCommand struct {
    ExecutableName string   `json:"executable"`
    Args           []string `json:"args"`
}

var serverProc atomicPointer[*exec.Cmd]

func handleLaunch(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "only POST requests are allowed", http.StatusMethodNotAllowed)
        return
    }
    var req LaunchCommand
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
    go launchServer(r.Context(), req)
}

func handleExit(w http.ResponseWriter, r *http.Request) {
    if p := serverProc.Load(); p != nil {
        _ = p.Process.Signal(syscall.SIGTERM) // м'яке завершення для коректного запису метрик
    }
}

func launchServer(ctx context.Context, req LaunchCommand) {
    cmd := exec.CommandContext(ctx, req.ExecutableName, req.Args...)
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr
    if err := cmd.Start(); err != nil {
        slog.Error("Failed to start server", "err", err)
        return
    }
    serverProc.Store(cmd)
    slog.Info("Server launched", "name", req.ExecutableName)
    _ = cmd.Wait()
    slog.Info("Server exited")
}
```

Рисунок 3.18 – Логіка контроль-сервера для запуску і завершення тестового сервера

Клієнтська утиліта надсилає контроль-серверу JSON-команду запуску. На рисунку 3.19 наведено код формування запиту і відправлення його методом POST, де видно побудову аргументів із параметрів конфігурації та явне зазначення типу вмісту application/json.

```go
// client/launcher.go
// Надсилання команди запуску на контроль-сервер у форматі JSON.
package client

import (
    "bytes"
    "context"
    "encoding/json"
    "fmt"
    "net/http"
    "strconv"

    "project/control"
)

type LaunchConfig struct {
    ControlURL string
    HTTPVer    int
    Format     string
    Compression string
    OutputDir  string
}

func askToLaunch(ctx context.Context, cfg LaunchConfig) error {
    req := control.LaunchCommand{
        ExecutableName: "./bin/http_s",
        Args: []string{
            "-port", "8000",
            "-compression", cfg.Compression,
            "-format", cfg.Format,
            "-output-dir", cfg.OutputDir,
            "-http-ver", strconv.Itoa(cfg.HTTPVer),
        },
    }
    body, err := json.Marshal(req)
    if err != nil {
        return fmt.Errorf("marshal launch command: %w", err)
    }
    resp, err := http.Post(cfg.ControlURL+"/launch", "application/json", bytes.NewReader(body))
    if err != nil {
        return fmt.Errorf("post launch request: %w", err)
    }
    _ = resp.Body.Close()
    return nil
}
```

Рисунок 3.19 – Код старту і завершення сервера зі сторони клієнта

Для повної автоматизації перебору конфігурацій використано Bash-скрипт, який послідовно запускає всі комбінації клієнтів, форматів і варіантів стискання, синхронізує порти та фіксує прогрес виконання. На рисунку 3.20 наведено основну частину скрипта з циклічним формуванням параметрів і викликом клієнтського раннера; під час виконання у консоль виводиться інформація про номер тесту, поточні параметри та статус завершення.

```bash
#!/usr/bin/env bash
# scripts/run_matrix.sh
# Автоматизований перебір конфігурацій тестів для бездротового інтерфейсу.

CLIENT_TYPES=("curl_h2 --http2" "curl_h3 --http3")
DATA_FORMATS=("json" "msgpack" "protobuf")
COMPRESSION_OPTIONS=("none" "gzip" "zstd" "lz4" "brotli" "deflate" "lzo")

BASE_PORT=8000
SERVER_COUNT=0
FAILED_TESTS=0

for client_spec in "${CLIENT_TYPES[@]}"; do
  IFS=' ' read -r client_type client_args <<< "$client_spec"
  for data_format in "${DATA_FORMATS[@]}"; do
    for compression in "${COMPRESSION_OPTIONS[@]}"; do
      PORT=$((BASE_PORT + SERVER_COUNT))
      echo "============================================================"
      echo " Running test $((SERVER_COUNT + 1))"
      echo " ... Client: $client_type"
      echo " ... Args:   $client_args"
      echo " ... Format: $data_format"
      echo " ... Compression: $compression"
      echo "============================================================"
      if ! ./run_client "$client_type" "$data_format" "$compression" "$client_args" --port "$PORT"
        echo "[FAILED] Test failed"; ((FAILED_TESTS++)); continue
      fi
      echo "[SUCCESS] Test completed successfully"
      sleep 1
      SERVER_COUNT=$((SERVER_COUNT + 1))
    done
  done
done

echo "Total tests: $SERVER_COUNT; Failed: $FAILED_TESTS"
exit $(( FAILED_TESTS > 0 ))
```

Рисунок 3.20 – Логіка Bash-скрипту для перебору конфігурацій і звітування статусів

Зазначений набір компонентів дозволяє надсилати повідомлення на тестовий сервер через HTTP/2 або HTTP/3, емулювати характеристики промислової мережі, знімати показники продуктивності і споживання ресурсів, а також автоматично покривати всю комбінаційну матрицю сценаріїв. Після завершення експериментів окрема програма на Python формує узагальнений CSV-файл, на підставі якого будуються підсумкові графіки порівняння затримки, коефіцієнтів стискання, використання ресурсів та відносних результатів щодо еталонної конфігурації HTTP/2 з форматуванням JSON і стисканням GZIP; за заданою методикою саме від цієї конфігурації очікується покращення якісних характеристик у розроблюваному інтерфейсі.

3.4 Налаштування експерименту у межах програмного інтерфейсу бездротової системи

У межах випробувань програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах було зафіксовано набір параметрів, що моделюють поведінку каналу зв’язку у контрольованому лабораторному середовищі та узгоджуються з можливостями мережевого емулятора, описаного в попередньому підрозділі. Такі параметри дають можливість відокремити власне вплив алгоритмів стиснення й серіалізації від впливу зовнішніх мережевих чинників та відтворити характерні умови виробничих сегментів з помірним навантаженням.

Параметр MinLatency установлено на рівні 15 мс. Обране мінімальне значення репрезентує затримку, що типовою є навіть для достатньо стабільних каналів, наприклад у локальних промислових мережах або за коротких бездротових з’єднань між контролерами й периферійними вузлами. Використання 15 мс як базової точки дозволяє оцінити чутливість

програмних модулів стискання та кодування до невеликої, проте неминучої в реальних умовах латентності, а також виявити, як вона перетворюється у додатковий час доставлення пакетів за різних форматів подання даних.

Параметр MaxLatency встановлено на 75 мс і трактовано як верхню межу затримки, що імітує пікові стани складних або завантажених топологій, наприклад у разі проходження трафіку через декілька проміжних шлюзів, при фонового перевантаження частотного ресурсу чи за наявності планових перешкод у радіоканалі. Межа у 75 мс уможливлює перевірку стійкості розробленого інтерфейсу й обраних алгоритмів до погіршених мережевих обставин та дозволяє проконтролювати межі деградації часу відгуку системи в індустріальному сценарії.

Параметр FailureRate під час основної серії вимірювань встановлено у 0. На цьому етапі цілеспрямовано відмовлено від ін’єкції помилок і втрат пакетів, щоб у «ідеалізованій» моделі без додаткових збоїв об’єктивно оцінити власну ефективність ланцюжка «серіалізація – стиснення – передача – розтиснення – десеріалізація». За потреби дослідження толерантності до відмов мережевий емулятор дозволяє збільшити FailureRate та повторити тести з однаковою матрицею параметрів, що забезпечує коректну компаративну інтерпретацію результатів.

Параметр Bandwidth прийнято рівним 10 МБ/с. Така пропускна здатність відповідає швидкому каналу, що часто зустрічається у сучасних промислових рішеннях, і водночас не призводить до надмірно оптимістичних сценаріїв, характерних для необмеженої смуги. Встановлення Bandwidth на 10 МБ/с дозволяє експериментально з’ясувати, наскільки у швидкій мережі латентність алгоритмів стискання та перетворення форматів впливає на повний час доставки повідомлення і які комбінації «формат–компресія» демонструють кращу придатність для бездротових сегментів з активним трафіком.

Для симулювання багатопоточного навантаження параметр Concurrency встановлено на 256. Мова Go забезпечує маловитратну

організацію великої кількості конкурентних операцій, тому значення 256 адекватно моделює одночасні запити від множини сенсорних вузлів або підсистем моніторингу до центрального сервера збирання даних. Такий режим дозволяє оцінити масштабованість запропонованого інтерфейсу та перевірити вплив конкуренції на метрики часу, стиснення і використання ресурсів.

У тестах передачі файлів кожен прогін сумарно передає близько 1000 об’єктів розміром від 1 МБ до 5 МБ. Набір файлів є змішаним і включає зображення, відеофрагменти, текстові матеріали, а також JS-бандли поширених бібліотек з екосистеми npm; такий добір відтворює характер промислових об’єктів обміну і дозволяє оцінити поведінку різних алгоритмів на гетерогенних даних. У тестах передачі структурованих даних формується 100000 запитів; кожен запит містить структури телеметрії та додаткові двійкові байти розміром від 1 кБ до 500 кБ, що забезпечує репрезентативне порівняння ефективності стиснення і швидкодії серіалізації для характерних службових повідомлень бездротової індустріальної мережі.

3.5 Проведення експериментального тестування та результати

Загальний цикл випробувань розробленого клієнтсько-серверного комплексу програмного інтерфейсу автоматизованої системи бездротового обміну даними тривав 43 хвилини та охоплював 160 тестових прогонів, під час яких зафіксовано повний набір часових, ресурсних і структурних метрик. У результаті сформовано 160 окремих файлів зі статистичними даними, що дозволяє провести репрезентативний аналіз ефективності алгоритмів у різних комбінаціях форматів серіалізації та стиснення. На рисунку 3.21 наведено підсумковий фрагмент консолі з результатами виконання тестів, який підтверджує успішне завершення всіх прогонів і відсутність збоїв у роботі клієнта та сервера в межах заданих параметрів мережевого емулятора.

Рисунок 3.21 – Результат виконання тесту

Як видно з результатів на рисунку 3.21, усі 160 тестів завершилися успішно, без помилок чи відхилень. Це свідчить про високу стабільність системи, коректну реалізацію клієнтсько-серверної взаємодії та точне дотримання параметрів симульованого середовища. Такий показник є особливо важливим для індустріальних сценаріїв, де безперервність і надійність передачі даних мають першорядне значення. Повна відсутність збоїв також вказує на правильну синхронізацію потоків у багатопоточному режимі та ефективність використаних засобів контролю стану мережевих каналів.

3.5.1 Проведення тестування часу стиснення

Перший етап кількісного аналізу полягав у визначенні часу, необхідного для виконання операцій стиснення, оскільки цей параметр є ключовим для оцінки придатності алгоритмів у бездротових промислових мережах. Надмірна тривалість стиснення може суттєво збільшити загальну латентність системи, нівелюючи переваги від зменшення обсягу переданих даних. Таким чином, метою тестування було виявлення алгоритмів, здатних забезпечити оптимальний баланс між швидкодією та ступенем стиснення. На

рисунку 3.22 показано первинний графік часу стиснення для різних алгоритмів у поєднанні з протоколами HTTP/2 і HTTP/3.

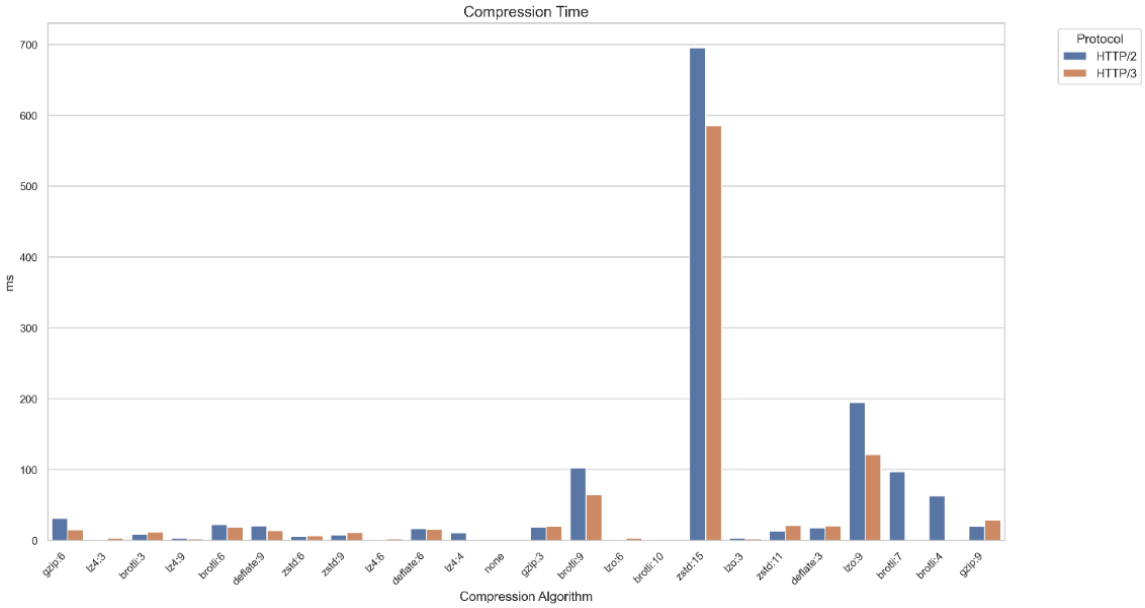


Рисунок 3.22 – Порівняння результатів часу стиснення

Рисунок 3.22 наочно демонструє, що алгоритми LZ4 у профілях 3, 6 і 9, а також Deflate у профілях 6 і 9, забезпечують найнижчі показники часу стиснення. Їхня швидкодія робить ці алгоритми особливо придатними для середовищ, де передача даних відбувається в реальному часі. Натомість алгоритми Zstd:15 і Brotli:10 показують значно вищі часові витрати, що свідчить про їхню орієнтацію на максимальне стиснення за рахунок швидкості. Високий час виконання Zstd:15 при використанні HTTP/2 підтверджує складність обчислювальних процедур цього алгоритму, однак у випадках, де головним є мінімізація обсягу переданих даних, він може залишатися доцільним вибором.

Для спрощення подальшого аналізу та підвищення інформативності графічного представлення даних було здійснено попередню фільтрацію результатів. Із загального набору виключено алгоритми, що продемонстрували надмірний час стиснення, зокрема Zstd:15, Brotli:9, LZO:9, Brotli:7. Оновлений графік часу стиснення подано на рисунку 3.23.

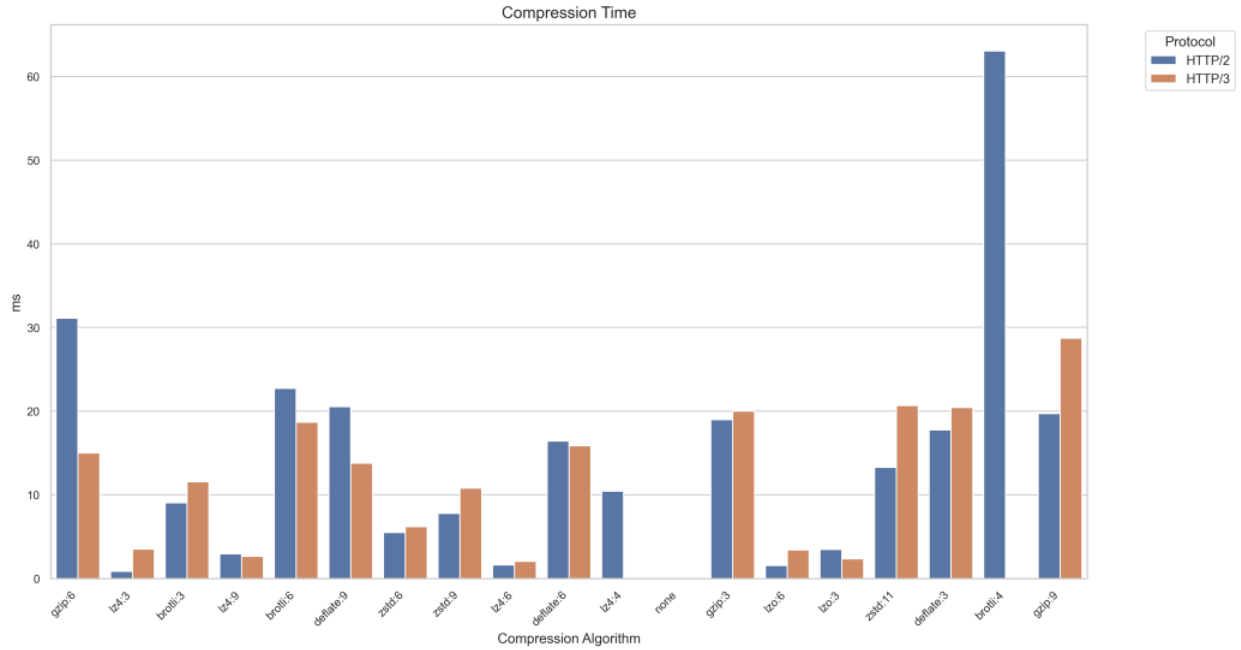


Рисунок 3.23 – Оновлений графік часу стиснення

Оновлена діаграма (рисунок 3.23) показує, що після вилучення неефективних конфігурацій найкращими за швидкодією залишаються алгоритми LZ4 і Deflate. Алгоритми сімейства GZIP демонструють прийнятні результати: GZIP:3 забезпечує хорошу швидкість, тоді як GZIP:6 і GZIP:9 потребують дещо більше часу. Протокол HTTP/3 для більшості алгоритмів показує або порівнянний, або менший час стиснення порівняно з HTTP/2, що підтверджує ефективність QUIC-підсистеми у зменшенні транспортних затримок. Така закономірність є позитивним індикатором потенційної переваги HTTP/3 у контексті індустріальних бездротових застосувань.

На заключному етапі аналізу було вилучено статистичну аномалію, виявлену для Brotli:4, який показав вищий час стиснення, ніж більш складна конфігурація Brotli:3. Очищений набір результатів дозволив сформувати підсумкову вибірку алгоритмів, що задовольняють вимоги до часових характеристик, представлений на рисунку 3.24.

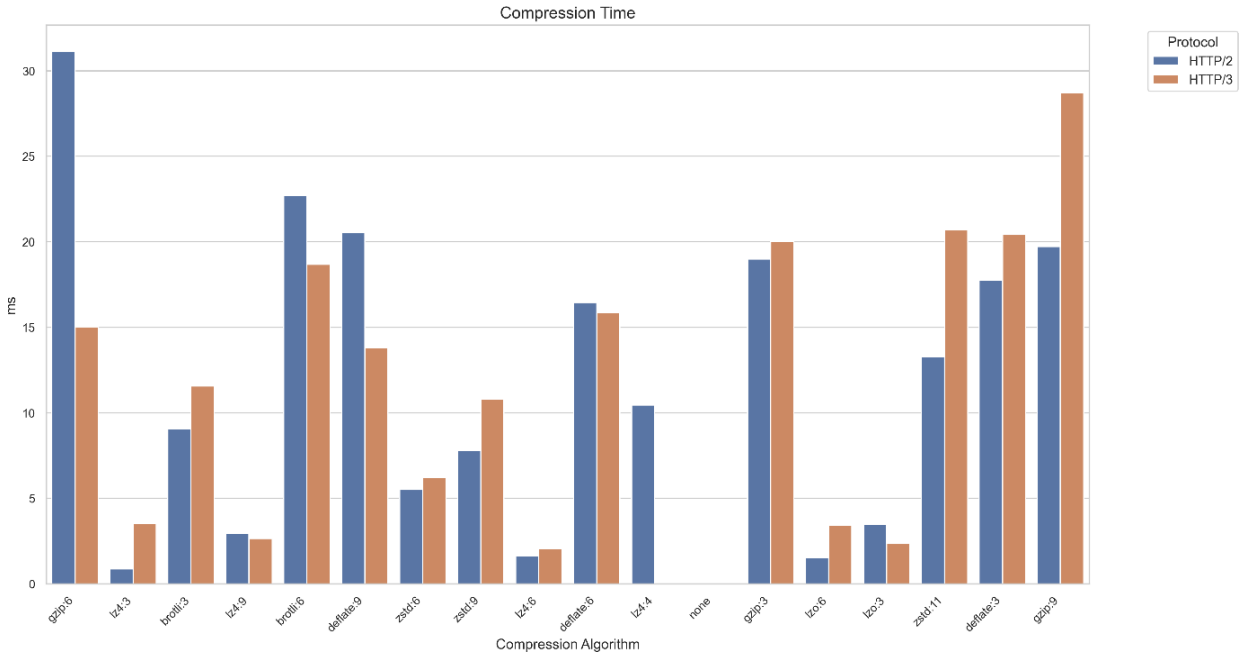


Рисунок 3.24 – Оновлений графік часу стиснення після остаточної фільтрації

Як показано на рисунку 3.24, найкращі результати серед усіх алгоритмів демонструють LZ4:3–9, які забезпечують найменший час стиснення, не перевищуючи кількох мілісекунд. Їхня продуктивність робить їх перспективними для інтеграції у реальні промислові рішення. Алгоритми Deflate і GZIP займають проміжні позиції, забезпечуючи компроміс між швидкістю та ступенем стиснення. Таким чином, з урахуванням усіх отриманих даних можна зробити висновок, що комбінації LZ4 + HTTP/3 і Deflate + HTTP/3 забезпечують найкраще співвідношення швидкодії, стабільності та ефективності для застосувань у бездротових мережах промислового типу.

3.5.2 Проведення тестування часу декомпресії

Після аналізу часу стиснення та відбору алгоритмів, які продемонстрували прийнятні показники ефективності, наступним етапом

стало дослідження часу декомпресії (розпакування даних). Цей параметр є критично важливим для систем автоматизованого бездротового обміну інформацією, оскільки визначає, наскільки швидко пристрій-отримувач може відновити початковий обсяг даних після їхнього передавання. Для промислових середовищ, де обробка даних має виконуватись у режимі реального часу, час декомпресії безпосередньо впливає на оперативність прийняття рішень і швидкість реакції системи.

Метою експерименту було визначити алгоритми, здатні не лише швидко стискати, але й ефективно декомпресувати великі обсяги інформації без суттєвих затримок. Під час дослідження було застосовано ті самі параметри мережевого середовища, що і під час тестів часу стиснення, аби забезпечити коректність порівняння. На рисунку 3.25 наведено графік, який ілюструє час декомпресії для різних алгоритмів у поєднанні з протоколами HTTP/2 та HTTP/3.

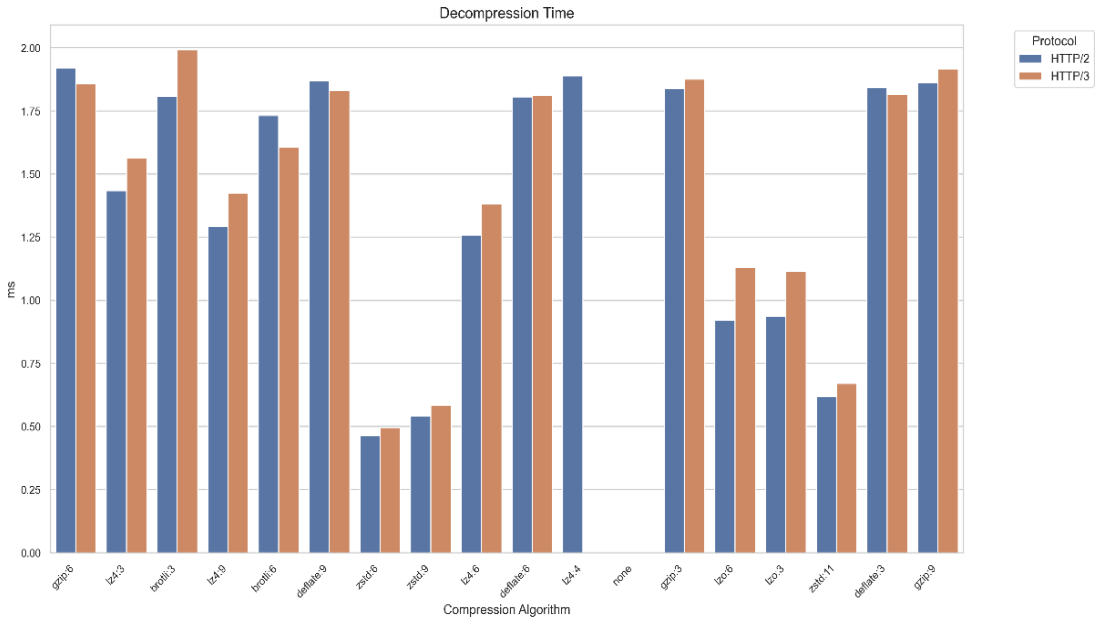


Рисунок 3.25 – Порівняння часу декомпресії

Як видно з рисунка 3.25, більшість алгоритмів характеризуються низькими часовими витратами на операцію декомпресії, що підтверджує їхню придатність для промислових бездротових систем. Особливо ефективними виявилися алгоритми LZ4, Deflate та GZIP, час розпакування

яких не перевищує декількох мілісекунд. Це означає, що такі алгоритми здатні забезпечити стабільну швидкість обробки потокових даних навіть у середовищах із підвищеним навантаженням.

Варто звернути увагу, що тестування проводилося на високопродуктивній комп'ютерній системі, тоді як більшість промислових контролерів або сенсорних вузлів мають значно менші обчислювальні можливості. Тому при оцінюванні отриманих даних доцільно орієнтуватися не на абсолютні показники часу, а на відносні співвідношення між алгоритмами. Навіть з урахуванням цього фактора, результати свідчать, що Zstandard (Zstd) демонструє найкращу продуктивність серед усіх протестованих методів – його показники залишаються стабільно низькими навіть при високих рівнях стиснення (наприклад, 11), перевершуючи більшість альтернатив за швидкістю декомпресії.

Отже, аналіз часу декомпресії підтверджує, що Zstd є найперспективнішим кандидатом для використання у промислових бездротових системах обміну даними, де важливими є як швидкість, так і стабільність відновлення інформації. Його архітектура оптимізована під роботу в обмежених обчислювальних середовищах, що робить можливим його ефективне впровадження навіть у пристрої з невеликими обсягами пам'яті та невисокою тактовою частотою процесора. Таким чином, Zstd може розглядатися як базовий алгоритм для систем, орієнтованих на швидкий обмін великими масивами даних у реальному часі.

3.5.3 Проведення тестування ступеня стиснення

Після розгляду часових характеристик операцій стиснення та декомпресії, наступним етапом стало дослідження ступеня стиснення (compression ratio), який визначає ефективність алгоритмів у зменшенні обсягу даних. Ступінь стиснення є співвідношенням розміру початкових

даних до розміру після стиснення, тобто відображає, наскільки алгоритм здатен скоротити обсяг інформації для передавання. У контексті автоматизованих систем бездротового обміну даними в індустріальних мережах цей параметр має особливе значення, оскільки він безпосередньо впливає на швидкість передавання, навантаження на канали зв'язку та загальну продуктивність мережевої взаємодії.

Досягнення високого ступеня стиснення дозволяє значно зменшити витрати пропускної здатності, підвищити ефективність використання каналів зв'язку та знизити латентність у процесі обміну повідомленнями між вузлами мережі. Водночас надмірно глибоке стиснення може вимагати більших обчислювальних ресурсів, тому важливо знайти баланс між компактністю даних і швидкістю їх обробки. На рисунку 3.26 наведено результати порівняльного аналізу ступенів стиснення для різних алгоритмів у поєднанні з протоколами HTTP/2 і HTTP/3.

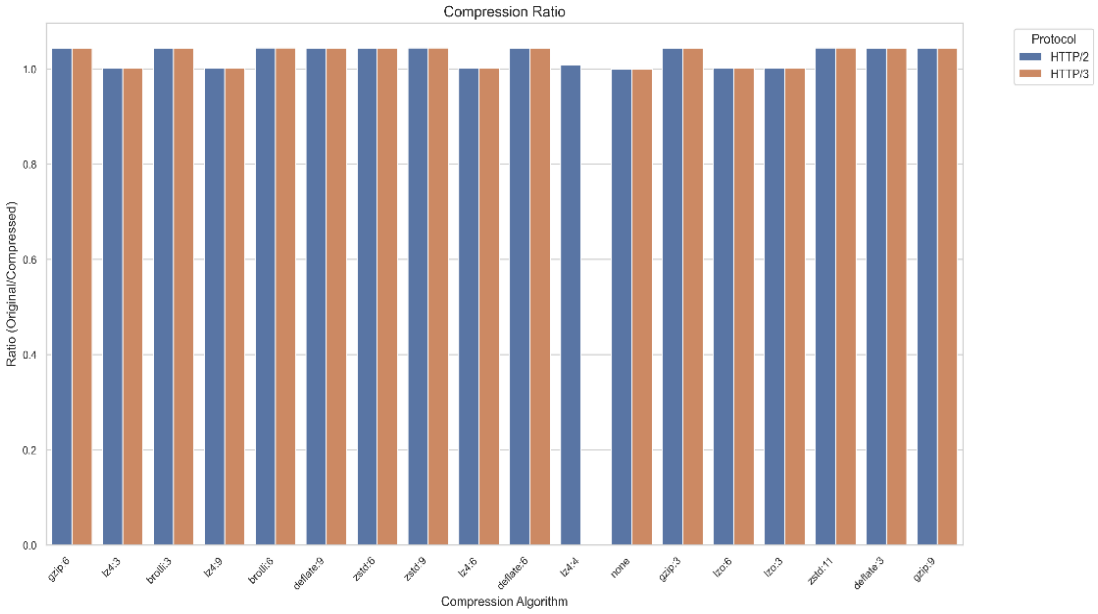


Рисунок 3.26 – Порівняння ступенів стиснення для різних алгоритмів

Як показано на рисунку 3.26, різниця між алгоритмами за ступенем стиснення є помітною, хоча більшість із них демонструють схожі показники для обох протоколів. Деякі методи, такі як LZ4 і LZO, показали нижчий коефіцієнт стиснення, що свідчить про їхню меншу ефективність при роботі з

типовими даними промислових мереж. Вони орієнтовані переважно на швидкість, а не на максимальне скорочення розміру даних.

Алгоритми сімейства LZ4 (lz4:3, lz4:6, lz4:9) та LZO (lzo:3, lzo:6) показали коефіцієнт стиснення, близький до одиниці, що означає майже відсутність помітного зменшення обсягу даних. Тому для подальшого аналізу вони були виключені як такі, що не відповідають вимогам до промислових середовищ, де необхідно знизити навантаження на мережеву інфраструктуру.

Разом з тим, варто зазначити, що LZ4 і LZO широко використовуються у системах на кшталт zRAM у Linux, де головним завданням є миттєве стиснення оперативної пам’яті з мінімальними витратами часу. У таких випадках ключовим фактором є саме швидкість операцій, а не економія простору. Проте для задач передачі даних у бездротових мережах, де пріоритетом є оптимізація пропускної здатності, подібні алгоритми стають менш придатними.

У випадках, коли важливішою є ефективність використання каналу зв’язку, більш доцільним є застосування алгоритмів GZIP, Deflate, Brotli або Zstandard (Zstd). Вони забезпечують значно вищий ступінь стиснення навіть за рахунок дещо більших обчислювальних витрат. Як видно з діаграми, саме ці алгоритми забезпечують оптимальний баланс між зменшенням розміру переданих даних і збереженням прийнятної швидкодії, що робить їх найбільш придатними для промислових бездротових систем із високими вимогами до ефективності комунікації.

Таким чином, аналіз ступеня стиснення підтверджує, що для автоматизованих систем бездротового обміну даними в індустріальних мережах доцільно використовувати алгоритми з помірним навантаженням на процесор, але з високою здатністю до економії пропускної здатності. Найкраще цим вимогам відповідають Zstd, Brotli та GZIP, які демонструють стабільну ефективність у поєднанні з протоколами HTTP/2 та HTTP/3, забезпечуючи оптимальний компроміс між швидкістю та щільністю стиснення.

3.5.4 Проведення тестування використання процесора

Після попереднього аналізу часових характеристик стиснення, декомпресії та ступеня стиснення, наступним кроком стало дослідження використання процесорних ресурсів (CPU Usage) різними алгоритмами. Цей параметр є одним із найважливіших при виборі рішень для промислових бездротових систем, оскільки більшість пристроїв у таких мережах мають обмежену обчислювальну потужність і працюють у режимах, де енергоефективність є критичною.

Надмірне навантаження на процесор може не лише сповільнити обробку даних, а й зменшити термін роботи пристрою від батареї, що неприпустимо для сенсорних вузлів і контролерів, розташованих у важкодоступних місцях. Тому порівняння алгоритмів стиснення за рівнем використання процесора є необхідним етапом для визначення найбільш збалансованих і придатних до практичного застосування варіантів.

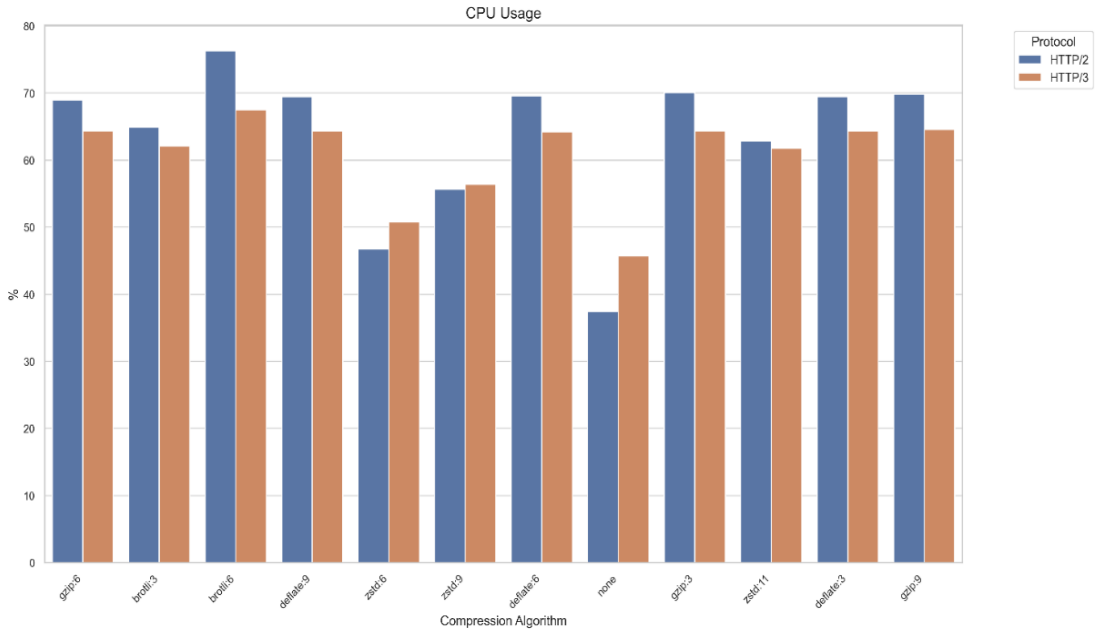


Рисунок 3.27 – Порівняння використання процесора різними алгоритмами стиснення

Як показано на рисунку 3.27, рівень навантаження на процесор істотно залежить від обраного алгоритму. У середньому для більшості протестованих

методів використання процесора коливається в межах 50–75%, що свідчить про значне залучення обчислювальних потужностей під час операцій стиснення.

Серед алгоритмів із найвищим рівнем використання CPU виділяються Brotli:6 і Deflate:9, у яких навантаження перевищує 70–75% для HTTP/2. Їхня висока обчислювальна складність зумовлена прагненням досягти максимальної щільності стиснення, що потребує інтенсивних розрахунків. Алгоритми GZIP:6, GZIP:9, Deflate:6 і Deflate:3 також демонструють значні показники – у межах 60–70%, що все ж менше, ніж у Brotli, але все ще створює помітне навантаження.

Натомість Zstandard (Zstd) показав помітно кращу ефективність з погляду використання CPU. Для конфігурацій Zstd:6 і Zstd:9 середнє навантаження склало 55–60%, а для Zstd:11 – навіть трохи нижче цього рівня. Це свідчить, що Zstd здатен забезпечити високу якість стиснення при помірних обчислювальних витратах. Алгоритм LZ4 також продемонстрував низьке навантаження на процесор (приблизно 45–50%), однак через слабкий ступінь стиснення він менш придатний для промислових систем, де важлива економія пропускної здатності.

Очікувано, сценарій «none» (передача даних без стиснення) показав мінімальне навантаження – близько 40%, що відображає базові витрати на обробку і передачу даних без додаткової компресії.

Порівняння між протоколами HTTP/2 і HTTP/3 не виявило стабільної тенденції: для деяких алгоритмів (наприклад, Brotli:6, Deflate:9) навантаження при HTTP/3 виявилось трохи нижчим, тоді як для інших (наприклад, GZIP:9, Zstd:6) різниця є незначною. Це свідчить про те, що ефективність використання CPU залежить радше від внутрішньої оптимізації конкретного алгоритму, ніж від транспортного протоколу.

Узагальнюючи результати, можна зробити висновок, що найбільше навантаження створюють Brotli:6, Deflate:9, Deflate:6, GZIP:3 та GZIP:9, де рівень використання CPU перевищує 60–75%. Такі показники є

неприйнятними для енергообмежених систем або промислових пристроїв з невеликим обсягом обчислювальних ресурсів. Відповідно, ці алгоритми було вирішено виключити з подальшого аналізу через надмірне навантаження на процесор.

У подальших розділах увага зосереджуватиметься на алгоритмах, які продемонстрували оптимальний баланс між ефективністю стиснення, часом обробки та енергоспоживанням – зокрема, на сімействі Zstd та, частково, Deflate у його базових конфігураціях. Це дозволить надалі побудувати модель оцінювання придатності алгоритмів для впровадження у комп'ютеризованих промислових мережах з урахуванням обмежень обчислювальної потужності та стабільності роботи систем.

3.5.5 Проведення тестування використання пам'яті

Після детального аналізу часових характеристик, ступеня стиснення та використання процесорних ресурсів, наступним важливим етапом експерименту став аналіз використання пам'яті (Memory Usage). Цей параметр є особливо важливим у контексті промислових бездротових мереж, де пристрої зазвичай мають обмежений обсяг оперативної пам'яті та обчислювальні ресурси. Ефективне керування пам'яттю безпосередньо впливає на стабільність роботи системи, її енергоефективність і загальну продуктивність. Надмірне споживання пам'яті може призвести до нестабільності програмного забезпечення, перевантаження пристроїв або навіть до необхідності збільшення апаратних ресурсів, що підвищує вартість рішень і ускладнює масштабування.

Тому аналіз обсягу пам'яті, який споживають різні алгоритми стиснення при роботі з протоколами HTTP/2 та HTTP/3, дозволяє визначити їхню придатність до використання у комп'ютеризованих системах передачі

інформації. На рисунку 3.28 наведено порівняльні результати використання пам’яті для найефективніших алгоритмів, відібраних на попередніх етапах.

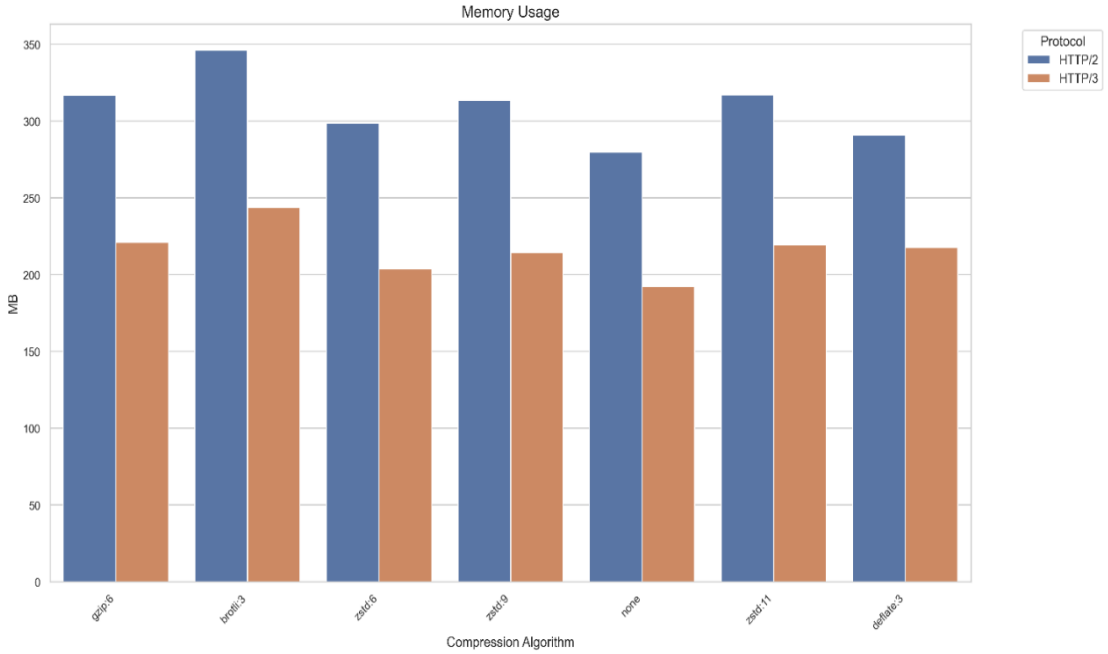


Рисунок 3.28 – Порівняння використання пам’яті різними алгоритмами стиснення

Як видно з діаграми на рисунку 3.28, використання пам’яті помітно варіюється залежно від алгоритму та версії протоколу. Загальний діапазон споживання коливається між 190 МБ і 340 МБ, що є доволі суттєвим показником для вбудованих або енергообмежених пристроїв. Такі результати свідомо отримано у «стресовому» сценарії, щоб перевірити стійкість системи в умовах максимального навантаження.

Найбільше споживання пам’яті продемонстрував Brotli:3, який досягав майже 345 МБ при роботі через HTTP/2. Такий рівень використання робить алгоритм непридатним для систем з обмеженими ресурсами. Ураховуючи, що Brotli:6 раніше було відсіяно через надмірне використання CPU, логічно виключити все сімейство Brotli з подальшого розгляду через його невідповідність вимогам пам’яті та енергоефективності.

Алгоритми GZIP:6, Zstd:6, Zstd:9 і Zstd:11 також продемонстрували досить високі показники споживання пам’яті – від 295 МБ до 315 МБ для HTTP/2. Водночас, при переході на HTTP/3, спостерігається чітка тенденція

до зниження цього показника – у середньому на 25–30%. Наприклад, GZIP:6 при HTTP/3 споживає лише близько 225 МБ, а Zstd:11 – приблизно 205 МБ, що свідчить про ефективнішу організацію пам’яті у новішому протоколі.

Алгоритми Deflate:3 та варіант «none» (без стиснення) демонструють найнижчі показники серед усіх протестованих – приблизно 275–285 МБ при HTTP/2 та 210–220 МБ при HTTP/3. Це вказує на потенційну перевагу HTTP/3 у контексті роботи з пам’яттю, що узгоджується з його архітектурними особливостями – зокрема, меншими накладними витратами на управління потоками даних і більш ефективним розподілом буферів.

Таким чином, результати дослідження демонструють чітку тенденцію: HTTP/3 стабільно зменшує споживання пам’яті для більшості алгоритмів. Це пояснюється вдосконаленнями у транспортному рівні – HTTP/3 базується на UDP через протокол QUIC, який дозволяє уникати надмірного копіювання даних і забезпечує кращу оптимізацію обробки пакетів у порівнянні з TCP, що використовується в HTTP/2.

Ураховуючи наведені спостереження, у подальшому аналізі основна увага буде приділена саме HTTP/3, оскільки він показав кращі результати за споживанням пам’яті, а також тенденційно – за часом стиснення й декомпресії. Крім того, HTTP/3 уже набуває статусу основного стандарту для сучасних мереж, тому орієнтація подальших експериментів на цей протокол є цілком виправданою.

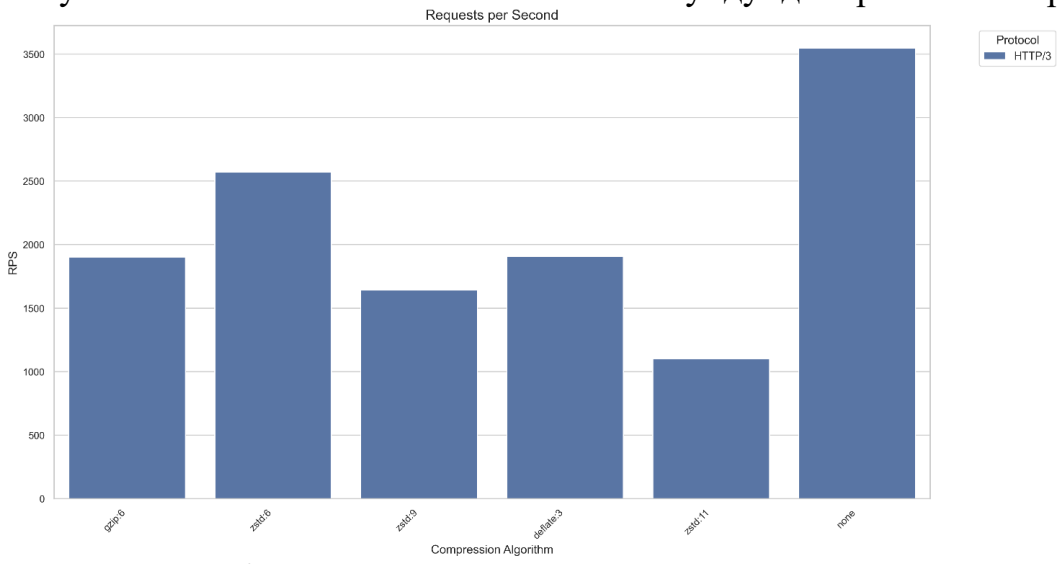
Отже, сімейство Brotli було виключено з подальших етапів через надмірне споживання як пам’яті, так і процесорних ресурсів. Дослідження буде зосереджене на алгоритмах Zstd, Deflate та GZIP, які продемонстрували прийнятне співвідношення між використанням пам’яті, продуктивністю та ступенем стиснення при роботі через HTTP/3. Ці результати закладають основу для комплексної оцінки енергоефективності алгоритмів на наступних етапах дослідження.

3.5.6 Проведення тестування кількості запитів за секунду

Завершальний етап дослідження присвячено аналізу однієї з найважливіших метрик продуктивності системи передачі даних – кількості запитів за секунду (Requests Per Second, RPS). Цей показник безпосередньо відображає реальну пропускну здатність системи та визначає, скільки запитів вона здатна обробити за одиницю часу без втрати стабільності. Для промислових бездротових мереж, де велика кількість сенсорних пристроїв постійно передає дані, саме RPS є критичним параметром, що характеризує швидкодію, масштабованість і ефективність усієї архітектури обміну інформацією.

Високе значення RPS свідчить про те, що система здатна ефективно обробляти запити навіть при високому навантаженні, зберігаючи низький рівень затримок та стабільну роботу каналів передачі. Оскільки в попередніх розділах було встановлено перевагу HTTP/3 над HTTP/2 за ключовими характеристиками (швидкість, використання пам’яті, CPU), подальші тести було проведено виключно для протоколу HTTP/3, який є більш перспективним для застосування в промислових комп’ютеризованих мережах.

Рисунок 3.29 – Кількість запитів за секунду для різних алгоритмів стиснення при HTTP/3



Як показано на рисунку 3.29, кількість запитів за секунду значно залежить від вибраного алгоритму стиснення. Алгоритм Zstd:6 продемонстрував найвищу ефективність серед усіх протестованих методів – його показник сягнув приблизно 2500 RPS, що лише на 15–18% менше, ніж у варіанта «none» (передача без стиснення), який логічно є абсолютним лідером із показником близько 3400 RPS.

Такі результати свідчать, що Zstd:6 є оптимальним компромісом між швидкістю обробки та ступенем стиснення: він мінімізує втрати продуктивності, забезпечуючи при цьому значне скорочення обсягу передаваних даних. Це робить його надзвичайно привабливим для сценаріїв, де висока частота запитів поєднується з потребою зменшення навантаження на канал зв’язку – наприклад, у системах моніторингу виробничих процесів або в бездротових сенсорних мережах (WSN).

Інші алгоритми, такі як Deflate:3, Zstd:9 і GZIP:6, показали помірні результати – у межах 1800–2100 RPS. Алгоритм Zstd:11 виявився повільнішим (приблизно 1100 RPS), що можна пояснити підвищеною складністю обчислень при глибшому рівні стиснення. Попри це, навіть найповільніші алгоритми зберігають прийнятні показники для сценаріїв, де першочергове значення має якість стиснення, а не швидкість обробки.

Загальний аналіз показує, що HTTP/3 забезпечує більш плавне масштабування навантаження, ніж попередні версії протоколу. Його архітектура на базі UDP (через QUIC) дозволяє скоротити кількість повторних запитів і мінімізувати вплив мережевих затримок, що позитивно відображається на стабільності показника RPS.

Таким чином, отримані результати підтверджують, що алгоритм Zstd:6 є найефективнішим серед протестованих, забезпечуючи високу пропускну здатність системи при помірних обчислювальних витратах. Варіант Deflate:3 може бути розглянутий як компромісне рішення для систем, де потрібне нижче споживання ресурсів при середній продуктивності. Алгоритми GZIP і

Zstd:9 демонструють збалансовану поведінку, але дещо поступаються за швидкодією.

Отже, у фінальному узагальненні можна стверджувати, що Zstd:6 є найоптимальнішим алгоритмом для реалізації стиснення в системах бездротової передачі даних, побудованих на протоколі HTTP/3. Він забезпечує високий показник RPS, стабільність роботи, помірне споживання пам’яті та процесорних ресурсів, що робить його найпридатнішим кандидатом для використання в сучасних промислових мережах реального часу.

3.5.7 Проведення тестування форматів даних

Після визначення оптимального алгоритму стиснення – Zstd:6 – у поєднанні з протоколом HTTP/3, наступним кроком стало дослідження впливу формату даних на загальну продуктивність системи. Формат, у якому здійснюється серіалізація інформації перед передачею, безпосередньо впливає на швидкість обробки, ступінь стиснення та кількість запитів за секунду (RPS), тобто на реальну пропускну здатність системи.

Традиційно у веб-застосунках використовується JSON (JavaScript Object Notation) – текстовий формат, який простий у читанні та впровадженні, проте не завжди ефективний у промислових умовах. Для порівняння були також протестовані бінарні формати MessagePack та Protobuf (Protocol Buffers), що вважаються перспективнішими з точки зору швидкодії та компактності даних.

Метою експерименту стало визначення, який із цих форматів найкраще підходить для використання у промислових бездротових мережах, де потрібна мінімізація затримок і максимальна ефективність обміну інформацією. На рисунку 3.30 подано порівняльну діаграму продуктивності

трьох форматів даних при використанні протоколу HTTP/3 і алгоритму стиснення Zstd:6.

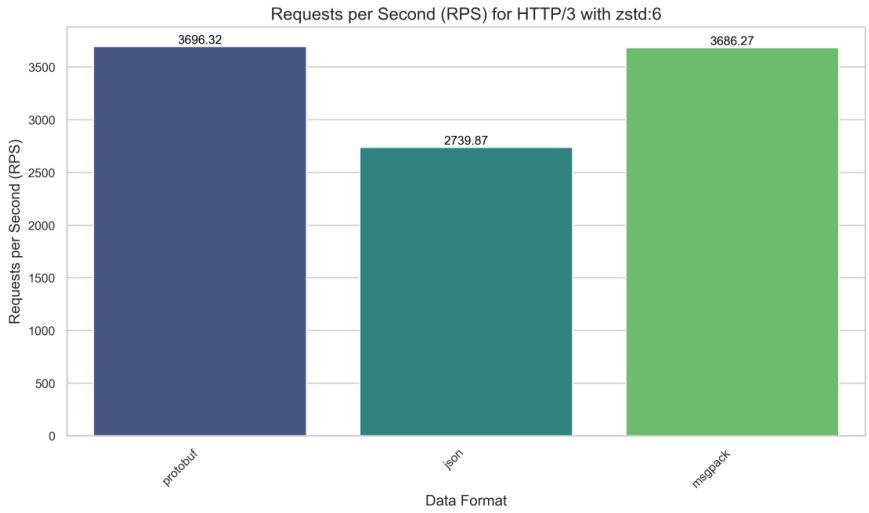


Рисунок 3.30 – Порівняння ефективності форматів даних при використанні HTTP/3 і Zstd:6

Як показано на рисунку 3.30, результати тестування демонструють суттєву різницю між форматами даних. Формат JSON забезпечив найнижчу кількість запитів за секунду – приблизно 2739,9 RPS, що свідчить про значне навантаження на процесор при обробці текстових структур. Натомість Protobuf і MessagePack показали значно кращі результати – 3696,3 RPS та 3686,3 RPS відповідно, що перевищує показник JSON майже на 35%.

Ці результати свідчать, що бінарні формати даних мають суттєву перевагу над текстовими у контексті промислових систем. Вони не лише зменшують розмір переданих повідомлень, але й значно скорочують час серіалізації та десеріалізації, що особливо важливо в умовах обмежених ресурсів та великої кількості паралельних з’єднань.

Порівнюючи між собою Protobuf і MessagePack, можна відзначити, що різниця між їхніми показниками є мінімальною – лише близько 0,3%, що лежить у межах статистичної похибки. Тобто, з погляду чистої продуктивності, обидва формати можна вважати рівноцінними. Однак, при глибшому аналізі виявляються суттєві практичні відмінності.

Формат Protobuf характеризується строгим визначенням структури даних через файли .proto, що гарантує високу стабільність і передбачуваність при взаємодії між різними компонентами системи. Водночас, така вимога додає складності під час розробки – необхідно створювати, оновлювати й компілювати схеми для кожного типу повідомлень, що підвищує витрати на підтримку коду.

Натомість MessagePack не потребує попереднього визначення схеми, що робить його гнучкішим і динамічнішим. Цей формат ідеально підходить для застосувань, де структура даних може змінюватися в процесі розробки або експлуатації системи. Простота інтеграції, універсальність бібліотек і зручність тестування роблять MessagePack більш привабливим для швидкого прототипування та розгортання у гетерогенних промислових середовищах.

Таким чином, враховуючи практично ідентичну продуктивність Protobuf і MessagePack, але вищу гнучкість і меншу складність розробки останнього, можна зробити висновок, що MessagePack є оптимальним вибором формату даних для системи бездротової передачі даних, реалізованої на основі протоколу HTTP/3 із використанням алгоритму стиснення Zstd:6.

MessagePack забезпечує високу пропускну здатність (≈3686 RPS), мінімальні затримки, простоту інтеграції та легкість у супроводі програмного забезпечення, що робить його найдоцільнішим форматом для промислових комп’ютеризованих мереж, де важливий баланс між продуктивністю, стабільністю та гнучкістю архітектури.

3.5.8 Підсумок результатів тестування

Підсумовуючи результати проведеного експерименту, можна впевнено зазначити, що модернізація системи передачі даних за рахунок переходу від комбінації HTTP/2, алгоритму стиснення GZIP:6 та формату даних JSON до

сучаснішої архітектури на основі HTTP/3, алгоритму Zstd:6 і формату MessagePack забезпечила суттєве підвищення продуктивності, ефективності та стабільності системи. Проведений аналіз довів, що запропонована комбінація технологій дає змогу істотно скоротити час стиснення та декомпресії, зменшити використання процесора й оперативної пам’яті, а також помітно збільшити кількість оброблених запитів за секунду. Алгоритм Zstd:6 показав менший час стиснення порівняно з GZIP:6, що прискорює підготовку даних до передачі, а також забезпечив швидке відновлення інформації на приймальній стороні, мінімізуючи затримки обробки. Завдяки своїй оптимізованій структурі Zstd:6 створює менше навантаження на процесор, що сприяє економії енергії й підвищенню стабільності роботи системи навіть у середовищах з обмеженими ресурсами. Поєднання HTTP/3 із Zstd:6 та MessagePack дозволило ефективніше використовувати оперативну пам’ять, а кількість запитів за секунду збільшилася майже на дві третини, що свідчить про зростання пропускної здатності системи. Сукупність цих показників підтверджує, що модернізована система стала не лише швидшою, а й економічнішою з точки зору обчислювальних витрат, що робить її особливо придатною для застосування в промислових бездротових мережах, де важливими є стабільність, швидкодія та ефективність.

Для підтвердження висновків нижче наведено порівняння базової та оптимізованої конфігурацій системи.

Таблиця 3.1 – Порівняння базової та оптимізованої конфігурацій системи передачі даних

Параметр	**HTTP/2**	**HTTP/3**
Алгоритм стиснення	GZIP:6	Zstd:6
Формат даних	JSON	MessagePack
Запитів за секунду (RPS)	2976,86	4915,79
Передача даних, байтів/с	144,02	237,90
Середнє використання пам’яті, МБ	107,78	55,68
Середнє використання процесора, %	74,05	48,71
Середній час стиснення, мс	4,25	0,62
Середній час декомпресії, мс	2,69	0,23
Середній розмір після стиснення, байт	101460,26	101492,73
Середній розмір після декомпресії, байт	134653,49	101482,66

Як видно з таблиці 3.1, нова конфігурація перевершує попередню за всіма основними показниками. Кількість оброблених запитів за секунду збільшилася на 65%, а обсяг переданих даних за той самий проміжок часу зріс на понад 65%, що свідчить про істотне покращення пропускної здатності системи. Для промислових мереж, у яких обробляються великі масиви даних від датчиків, керуючих модулів та контролерів у режимі реального часу, така оптимізація є надзвичайно важливою, оскільки дозволяє забезпечити своєчасну реакцію на зміни технологічних процесів без перевантаження мережі. Крім того, зменшення використання оперативної пам’яті майже наполовину робить систему більш ресурсоефективною, що дає змогу розгортати її на пристроях із мінімальними апаратними характеристиками або забезпечувати обробку більшого навантаження без додаткових витрат на модернізацію обладнання.

Ще одним важливим досягненням є зниження середнього навантаження на процесор приблизно на третину, що позитивно впливає на енергоспоживання та довговічність компонентів. Це особливо актуально для автономних пристроїв, які працюють від акумуляторів, та серверів, що обслуговують велику кількість клієнтів одночасно. Середній час стиснення скоротився більш ніж у шість разів, а час декомпресії – майже у десять разів, що є вирішальним показником для систем із вимогами до роботи в режимі реального часу. Незважаючи на таке істотне зменшення затримок, ефективність стиснення залишилася практично незмінною: різниця в розмірі даних після компресії становить лише 0,03%. Це свідчить про високу оптимізацію алгоритму Zstd:6, який забезпечує баланс між швидкістю та щільністю стиснення.

Водночас середній розмір даних, які сервер обробляє після декомпресії, зменшився майже на чверть, що пояснюється більш ефективною структурою представлення даних у форматі MessagePack. Цей результат демонструє не лише технічну перевагу обраної конфігурації, але й перспективу її подальшого використання в широкому спектрі промислових застосувань.

Отже, результати дослідження підтверджують, що перехід на протокол HTTP/3 у поєднанні з алгоритмом стиснення Zstd:6 і форматом MessagePack забезпечує комплексне покращення якості передачі даних. Нова система відзначається підвищеною продуктивністю, зменшеним споживанням ресурсів, покращеною енергоефективністю та швидкодією, що робить її придатною для впровадження в інтелектуальні промислові мережі нового покоління, орієнтовані на стабільність, гнучкість і високу ефективність обміну інформацією.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

4.1 Оцінювання комерційного потенціалу розробки

У сучасних умовах цифрової трансформації промисловості зростає потреба в надійному та швидкому бездротовому обміні даними між компонентами автоматизованих систем керування. Розвиток індустріальних мереж і технологій Industrial Internet of Things (IIoT) дозволяє здійснювати передавання інформації в режимі реального часу, однак якість, затримка та стабільність такого обміну потребують додаткової оптимізації.

На сьогодні існує значна кількість програмних рішень для організації бездротового обміну даними, проте більшість з них орієнтовані на загальні сценарії використання та не повною мірою враховують специфічні вимоги індустріального середовища, зокрема підвищені вимоги до надійності, масштабованості та гарантованого часу відгуку. Тому актуальним є створення програмного інтерфейсу, який не лише забезпечує передавання даних, а й оптимізує процес обміну з урахуванням умов роботи промислових мереж.

Складність організації бездротового обміну даними в індустріальних мережах пов’язана з необхідністю обробки великих обсягів інформації, що надходить від численних сенсорів, контролерів та виконавчих пристроїв. Сучасний рівень розвитку інформаційних технологій, мережевих протоколів і методів стиснення даних створює передумови для підвищення ефективності таких систем та зменшення затримок передавання.

У зв’язку з цим у межах магістерської кваліфікаційної роботи було досліджено та розроблено ефективний програмний інтерфейс автоматизованої системи бездротового обміну даними в індустріальних мережах, який реалізовано з використанням сучасних протоколів комунікації та оптимізованих методів обробки інформації.

Для проведення технологічного аудиту залучено трьох незалежних експертів. У рамках цієї роботи експертами виступають викладачі кафедри КСУ ВНТУ, зокрема:

Іванов Ю.Ю. (д.т.н., професор кафедри КСУ ВНТУ);

Ковтун В. В. (д.т.н., професор кафедри КСУ ВНТУ);

Ратушняк О. М. (к.т.н., доцент кафедри ЕПтаВМ ВНТУ).

Для оцінювання використано критерії, наведені у таблиці 4.1.

Таблиця 4.1 – Критерії оцінювання науково-технічного рівня та комерційного потенціалу програмного інтерфейсу автоматизованої системи бездротового обміну даними

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт має обмежену кількість аналогів на великому ринку
3	Вартість впровадження продукту значно вища за ціни аналогів	Вартість впровадження продукту дещо вища за ціни аналогів	Вартість впровадження продукту приблизно дорівнює цінам аналогів	Вартість впровадження продукту дещо нижче за ціни аналогів	Вартість впровадження продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Результати оцінювання науково-технічного рівня та комерційного потенціалу програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах зведено до таблиці 4.2.

Таблиця 4.2 – Результати експертного оцінювання науково-технічного рівня та комерційного потенціалу програмного інтерфейсу автоматизованої системи бездротового обміну даними

Критерії	Експерт		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	4	4
2. Ринкові переваги (наявність аналогів)	3	4	3
3. Ринкові переваги (ціна продукту)	3	4	5
4. Ринкові переваги (технічні властивості)	4	4	3
5. Ринкові переваги (експлуатаційні витрати)	4	4	4
6. Ринкові перспективи (розмір ринку)	4	5	4
7. Ринкові перспективи (конкуренція)	4	4	3
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	5	4	3
12. Практична здійсненність (розробка документів)	4	4	3
Сума балів	45	46	43
Середньоарифметична сума балів $СБ_с$	44,7		

На основі даних, наведених у таблиці 5.2, здійснено аналіз науково-технічного рівня та комерційного потенціалу програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних

мережах. Отримані результати порівнюються з рівнями комерційного потенціалу, наведеними в таблиці 4.3.

Таблиця 4.3 – Рівні науково-технічного розвитку та комерційного потенціалу програмних рішень для індустріальних бездротових мереж

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

Результати експертного оцінювання показали, що середньоарифметична сума балів становить 44,7 бала. Це підтверджує високий науково-технічний рівень та потенційну комерційну успішність програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах, відповідно до градації, наведеної у таблиці 5.3. Отриманий високий бал зумовлений використанням сучасних мережевих протоколів передавання даних, ефективних алгоритмів серіалізації та стиснення інформації, низькими експлуатаційними витратами, а також можливістю інтеграції розробки в існуючі промислові мережі з обмеженою кількістю прямих аналогів.

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів

Під час планування, обліку та калькулювання витрат, пов’язаних із проведенням науково-дослідної роботи на тему «Розробка програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах», витрати групуються за відповідними економічними категоріями. До категорії «Витрати на оплату праці» включаються витрати, пов’язані з виплатою основної та додаткової

заробітної плати працівникам, які займають керівні посади, а також інженерно-технічним працівникам, безпосередньо залученим до розроблення, реалізації та тестування програмного інтерфейсу автоматизованої системи бездротового обміну даними. Для визначення фонду основної заробітної плати ($З_о$) використовується аналіз трудомісткості виконуваних робіт. Оскільки розробка має дослідницький характер і виконується протягом частини робочого часу, розрахунок є більш точним при використанні годинних тарифних ставок. Витрати на основну заробітну плату дослідників $З_о$ розраховуємо за формулою:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} \times t_і}{Т_р}$$

де k - кількість виконавців, залучених до процесу досліджень; Мпі - місячний посадовий оклад конкретного дослідника, грн;

t_i - число днів роботи конкретного дослідника, дні;

Тр - середнє число робочих днів в місяці, Тр = 21 дня.

$$З_о = \frac{18000}{21} \times 6 + \frac{25000}{21} \times 52 = 5142{,}9 + 61904{,}8 = 67047{,}7\ грн$$

Таблиця 4.4 – Витрати на заробітну плату виконавців під час розроблення програмного інтерфейсу автоматизованої системи бездротового обміну даними

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	18000	857,1	6	5142,9
Інженер-розробник	25000	1190,5	52	61904,8
			Всього	67047,7

Додаткову заробітну плату розраховуємо як 10-12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \times \frac{н_{дод}}{100\%}$$

де Ндод - норма нарахування додаткової заробітної плати. Ндод приймемо як 12%.

$$З_{дод} = 67047{,}7 \times \frac{12}{100\%} = 8045{,}7\ грн$$

До статті «Відрахування на соціальні заходи» включаються внески на загальнообов'язкове державне соціальне страхування та витрати на соціальний захист працівників, зокрема єдиний соціальний внесок (ЄСВ), що нараховується на заробітну плату виконавців науково-дослідної роботи.

Нарахування на заробітну плату дослідників та працівників становить 22% від суми їх основної та додаткової заробітної плати і розраховується за наступною формулою:

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

де Нзп - норма нарахування на заробітну плату.

$$З_н = (67047{,}7 + 8045{,}7) \times \frac{22}{100\%} = 16520{,}3\ грн$$

До статті «Сировина та матеріали» відносяться витрати на основні та допоміжні матеріали, канцелярське приладдя, носії інформації та інші засоби і предмети праці, придбані у сторонніх підприємств, установ і організацій та використані для виконання науково-дослідної роботи за прямим призначенням відповідно до встановлених норм витрат. Також до цієї статті включаються витрати на матеріали, необхідні для оформлення технічної документації, збереження результатів експериментів і тестування програмного інтерфейсу автоматизованої системи бездротового обміну даними.

Вартість матеріалів (М) розраховується окремо для кожного виду матеріалів за наступною формулою:

$$М = \sum_{j=1}^{n} (Н_j \times Ц_j \times К_j) - \sum_{j=1}^{n} (В_j \times Ц_{Вj})$$

де Нj - норма витрат матеріалу j-го найменування, кг; n - кількість видів матеріалів;

Цj - вартість матеріалу j-го найменування, грн/кг;

Кj - коефіцієнт транспортних витрат, (Кj = 1,1...1,15);

Вj - маса відходів j-го найменування, кг;

Цвj - вартість відходів j-го найменування, грн/кг.

Таблиця 4.5 – Витрати на матеріали, необхідні для виконання науково-дослідної роботи

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Вартість витраченого матеріалу, грн
Папір для принтера	200	3	600
Нотатки (стікери)	120	2	240
Канцелярський набір (ручка, олівець, лінійка)	100	3	300
Файли	70	2	140
USB-флеш-накопичувачі	150	2	300
		Всього	1580

До статті «Спеціальне обладнання для наукових (експериментальних) робіт» входять витрати на придбання та використання спеціалізованих технічних засобів, які можуть бути необхідними для проведення досліджень. Оскільки виконання даної науково-дослідної роботи має переважно програмний характер і не потребує розроблення або використання спеціального експериментального обладнання, витрати за цією статтею не передбачені. До статті «Програмне забезпечення для наукових (експериментальних) робіт» відносяться витрати, пов’язані з розробленням і використанням програмного забезпечення, зокрема програмних інтерфейсів, мережевих протоколів, бібліотек, алгоритмів обробки, серіалізації та стиснення даних, необхідних для реалізації, тестування та експериментальної перевірки програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах, а також витрати на їх налаштування та інсталяцію.

Балансова вартість програмного забезпечення розраховується за

формулою:

$$В_{прг} = \sum_{i=1}^{k} (Ц_{iпрг} \times С_{прг.i} \times К_i)$$

де Ціпрг - ціна придбання одиниці програмного засобу даного виду, грн; Спрг.і - кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

Кі - коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (Кі = 1,10...1,12);

k - кількість найменувань програмних засобів.

$$В_{прг} = 7500 \times 1 \times 1{,}1 + 14000 \times 1 \times 1{,}1 + 5000 \times 1 \times 1{,}1 = 30250 \; грн$$

Таблиця 4.6 – Витрати на програмне забезпечення для розроблення та тестування програмного інтерфейсу автоматизованої системи бездротового обміну даними

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Python 3.11 + бібліотеки (pandas, scikit-learn)	1	7500	8250
IDE PyCharm Professional	1	14000	15400
Jupyter Notebook + MySQL	1	5000	5500
		Всього	29150

До статті «Амортизація обладнання, програмних засобів та приміщень» включаються амортизаційні відрахування за кожним видом комп'ютерної техніки, периферійного обладнання та програмних засобів, які використовуються під час розроблення, тестування та експериментальної перевірки програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах та перебувають у користуванні виконавців науково-дослідної роботи.

У спрощеному вигляді амортизаційні відрахування за кожним видом обладнання, приміщень та програмного забезпечення можуть бути розраховані за допомогою прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} \times \frac{t_{вик}}{12}$$

де Цб - балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

tвик - термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

Тв - строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = \frac{26000}{3} \times \frac{3}{12} + \frac{5000}{3} \times \frac{3}{12} = 2166{,}7 + 416{,}7 = 2583{,}4\ грн$$

Таблиця 4.7 – Амортизаційні відрахування обладнання, що використовується під час розроблення програмного інтерфейсу

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук LENOVO	26000	3	3	2166,7
Монітор Asus ROG	5000	3	3	416,7
			Всього	2583,4

До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на споживання електричної енергії, яка використовується для забезпечення роботи комп'ютерної техніки під час розроблення, тестування та експериментальних досліджень програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах. Оскільки дослідження не належать до енергоємних, дана стаття витрат формується за методом прямого віднесення та не становить значної частки у загальній собівартості науково-дослідної роботи. Витрати на споживання електричної енергії (Ве) розраховуються за формулою:

$$Ве = \sum_{i=1}^{n} \frac{W_{yi} \times t_i \times Ц_е \times К_{впі}}{\eta_i}$$

де Wyi - встановлена потужність обладнання на визначеному етапі розробки, кВт;

ti - тривалість роботи обладнання на етапі дослідження, год;

Це - вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо Це = 12,50 грн; Квпі - коефіцієнт, що враховує використання потужності, Квпі < 1;

ηі - коефіцієнт корисної дії обладнання, ηі < 1.

$$B_e = \frac{0{,}5 \times 480 \times 12{,}5 \times 0{,}95}{0{,}97} + \frac{0{,}03 \times 480 \times 12{,}5 \times 0{,}95}{0{,}97} = 3127{,}3\ грн$$

Таблиця 4.8 – Витрати на електроенергію для забезпечення виконання науково-дослідної роботи

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук LENOVO	0,5	480	2950,3
Монітор Asus ROG	0,03	480	177,0
		Всього	3127,3

Стаття «Службові відрядження» охоплює витрати, пов’язані з відрядженнями штатних працівників та осіб, залучених до виконання науководослідної роботи, які можуть бути пов’язані з участю у наукових конференціях, семінарах та профільних заходах з питань автоматизації, бездротових індустріальних мереж і сучасних інформаційних технологій, а також з апробацією та тестуванням програмних рішень. Зазначені витрати мають прямий зв’язок з виконанням даної науково-дослідної роботи. Витрати за цією статтею розраховуються у розмірі 20–25 % від суми основної заробітної плати дослідників та працівників за допомогою відповідної формули.

$$B_{сп} = (З_о + З_р) \times \frac{Н_{сп}}{100\%}$$

де Нсп - норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо Нсп = 25%.

$$B_{сп} = 67047{,}7 \times \frac{25}{100\%} = 16761{,}9\ грн$$

Стаття «Інші витрати» включає витрати, які не були враховані у попередніх

статтях калькуляції та можуть бути безпосередньо віднесені до собівартості виконання науково-дослідної роботи. До таких витрат можуть належати витрати на інформаційне забезпечення, використання мережевих сервісів, комунікаційні послуги та інші допоміжні витрати, пов'язані з розробленням і тестуванням програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах. Витрати за цією статтею обчислюються у розмірі 50–100 % від суми основної заробітної плати дослідників та працівників за допомогою відповідної формули.

$$I_{\text{ів}} = (З_{\text{о}} + З_{\text{р}}) \times \frac{Н_{\text{ів}}}{100\%}$$

де Нів - норма нарахування за статтею «Інші витрати», приймемо Нів = 60%.

$$I_{\text{ів}} = 67047{,}7 \times \frac{60}{100\%} = 40228{,}6\ грн$$

Сталими (загальновиробничими) витратами охоплюються витрати, пов'язані з організаційним забезпеченням виконання науково-дослідної роботи, управлінням процесом розроблення, інноваційною діяльністю, підготовкою та підвищенням кваліфікації персоналу, використанням інформаційних ресурсів, банківськими та комунікаційними послугами, а також із забезпеченням функціонування програмної інфраструктури. Витрати за цією статтею розраховуються у розмірі 100–150 % від суми основної заробітної плати дослідників та працівників з використанням відповідної формули.

$$В_{\text{нзв}} = (З_{\text{о}} + З_{\text{р}}) \times \frac{Н_{\text{нзв}}}{100\%}$$

де Ннзв - норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо Ннзв = 120%.

$$В_{\text{нзв}} = 67047{,}7 \times \frac{120}{100\%} = 80457{,}2\ грн$$

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

$$В_{\text{заг}} = З_{\text{о}} + З_{\text{дод}} + З_{\text{н}} + М + В_{\text{прг}} + А_{\text{обл}} + В_{\text{е}} + В_{\text{сп}} + I_{\text{ів}} + В_{\text{нзв}}$$

$$В_{\text{заг}} = 67047{,}7 + 8045{,}7 + 16520{,}3 + 1580 + 29150 + 2583{,}4 + 3127{,}3 + 16761{,}9 + 40228{,}6 + 80457{,}2 =$$

265502,1 *грн*

Вартість завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів обчислюється відповідно до наступної формули:

$$ЗВ = \frac{В_{заг}}{\eta}$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науководослідної роботи, приймемо η = 0,8.

$$ЗВ = \frac{265502,1}{0,8} = 331877,6\ грн$$

Отже, прогноз загальних витрат ЗВ на виконання та впровадження результатів виконаної роботи складає 331877,6 грн.

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У сучасних ринкових умовах позитивний економічний ефект від впровадження результатів науково-дослідної роботи для потенційного інвестора або замовника полягає у підвищенні ефективності функціонування автоматизованих систем керування, зниженні експлуатаційних витрат та оптимізації процесів бездротового обміну даними в індустріальних мережах. Розробка програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах передбачає можливість поетапного впровадження та комерціалізації протягом декількох років із поступовим розширенням сфери застосування та кола потенційних користувачів серед промислових підприємств. У даному випадку прогнозований економічний ефект базується на зростанні кількості впроваджень програмного інтерфейсу на підприємствах, що використовують індустріальні бездротові мережі та автоматизовані системи керування, протягом аналізованого періоду часу:

• у перший рік - 380 користувачів;

• у другий рік - 620 користувачів;

• у третій рік - 850 користувачів;

• у четвертий рік - 1100 користувачів.

N - кількість потенційних користувачів торгових систем у році до впровадження результатів розробки, приймемо 8500 користувачів;

Цб - вартість ліцензії програмного продукту у році до впровадження результатів розробки, приймемо 3500,00 грн;

±ΔЦо - зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 1200,00 грн.

Для кожного з випадків потенційне збільшення чистого прибутку у потенційного інвестора ΔПі в роки очікуваного позитивного результату розраховується за формулою:

$$\Delta П_i = (\pm \Delta Ц_o \times N + Ц_o \times N_i) \times \lambda \times \rho \times (1 - \frac{\vartheta}{100})$$

де λ - коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2025 році ставка податку на додану вартість складає 20%, а коефіцієнт λ = 0,8333;

ρ - коефіцієнт, який враховує рентабельність інноваційного продукту.

Приймемо ρ = 40%;

ϑ - ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році ϑ = 18%.

Збільшення чистого прибутку 1-го року:

$$\Delta П_1 = (1200 \times 8500 + 3500 \times 380) \times 0{,}8333 \times 0{,}45 \times (1 - \frac{0.18}{100}) = 4{,}380{,}920\ грн$$

Збільшення чистого прибутку 2-го року:

$$\Delta П_2 = (1200 \times 8500 + 3500 \times (380 + 620)) \times 0{,}8333 \times 0{,}45 \times (1 - \frac{0.18}{100}) = 5{,}920{,}400\ грн$$

Збільшення чистого прибутку 3-го року:

$$\Delta П_3 = (1200 \times 8500 + 3500 \times (380 + 620 + 850)) \times 0{,}8333 \times 0{,}45 \times (1 - \frac{0.18}{100}) = 11{,}893{,}750\ грн$$

Збільшення чистого прибутку 4-го року:

$$\Delta П_4 = (1200 \times 8500 + 3500 \times (380 + 620 + 850 + 1100)) \times 0{,}8333 \times 0{,}45 \times (1 - \frac{0.18}{100}) = 11{,}893{,}750 \text{ грн}$$

Приведена вартість потоків прибутку розраховується за формулою:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}$$

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

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

τ - ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,2;

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

$$ПП = \frac{4380920}{(1+0{,}2)^1} + \frac{5920400}{(1+0{,}2)^2} + \frac{7158680}{(1+0{,}2)^3} + \frac{8118620}{(1+0{,}2)^4} = 15{,}815{,}907 \; грн$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Ключовими факторами, що визначають доцільність інвестування у результати науково-дослідної роботи, є абсолютна та відносна ефективність вкладених інвестицій, а також термін їх окупності. Для потенційного інвестора або замовника важливим є оцінювання економічної ефективності впровадження програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах з урахуванням витрат на його розроблення та очікуваних комерційних ефектів. Першим етапом оцінювання економічної ефективності є розрахунок сучасної (приведеної) вартості інвестицій (PV), вкладених у розроблення програмного інтерфейсу автоматизованої системи бездротового обміну даними. Для цього використовується відповідна розрахункова формула:

$$PV = k_{\text{інв}} \times \text{ЗВ}$$

де кінв - коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо кінв = 3;

ЗВ - загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 331 877,6 грн.

$$PV = 3{,}5 \times 331877{,}6 = 1{,}161,571{,}6\ грн$$

Таким чином, чистий приведений дохід (NPV) або абсолютний економічний ефект (Еабс) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки буде таким:

$$\text{Е}_{\text{абс}} = \text{ПП} - PV$$

де ПП - приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 15 815 907 грн;

PV - теперішня вартість початкових інвестицій, 1 161 571,6 грн. $\text{Е}_{\text{абс}} = 15815907 - 1161571{,}6 = 14{,}654,335{,}4\ грн$

Внутрішня економічна дохідність (Ев) інвестицій, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково- технічної розробки, обчислюється за допомогою такої формули:

$$\text{Е}_{\text{в}} = \sqrt[\text{Т}_{\text{ж}}]{1 + \frac{\text{Е}_{\text{абс}}}{PV}} - 1$$

де Еабс - абсолютний економічний ефект вкладених інвестицій, 14 654 335,4 грн;

PV - теперішня вартість початкових інвестицій, 1 161 571,6 грн;

Тж - життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_в = \sqrt{1 + \frac{14654335,4}{1161571,6}} - 1 = 1,542 = 154,2\%$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій (τмін)

визначається згідно такою формулою:

$$\tau_{мін} = d + f$$

де d - середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d = 0,18;

f - показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,30.

$$\tau_{мін} = 0,18 + 0,30 = 0,48$$

Оскільки Ев = 154,2% > τmin = 48%, то внутрішня економічна дохідність інвестицій перевищує мінімальну внутрішню дохідність. Таким чином, інвестування у науково-дослідну роботу є економічно обґрунтованим і доцільним.

Далі обчислюємо період окупності інвестицій (Ток або DPP, Discounted Payback Period), які потенційний інвестор може вкласти у впровадження та комерціалізацію науково-технічної розробки:

$$Т_{ок} = \frac{1}{Е_в}$$

$$Т_{ок} = \frac{1}{1,542} = 0,65 \text{ років} = 7,8 \textit{ місяців}$$

З огляду на те, що період окупності інвестицій становить менше восьми місяців, можна дійти висновку, що фінансування цієї розробки є виправданим.

4.5 Висновки до розділу

У ході виконання економічного аналізу встановлено, що розроблений програмний інтерфейс автоматизованої системи бездротового обміну даними в індустріальних мережах має високий науково-технічний та комерційний

потенціал. За результатами експертного оцінювання середньоарифметичний показник становить 44,7 бала, що відповідає категорії «високий рівень» та свідчить про конкурентоспроможність розробки й доцільність її подальшої комерціалізації. Прогнозовані витрати на розроблення та впровадження програмного інтерфейсу є економічно обґрунтованими, а результати розрахунків приведених економічних показників демонструють позитивний економічний ефект від реалізації результатів науково-дослідної роботи. Отримані значення показників ефективності інвестицій підтверджують доцільність вкладень у впровадження розробки та її привабливість для потенційних інвесторів або замовників. Розрахунок показників ефективності інвестицій та періоду їх окупності свідчить про швидке повернення вкладених коштів і високу економічну результативність проєкту. Це зумовлено можливістю масштабування розробки, низькими експлуатаційними витратами, а також широкими перспективами застосування програмного інтерфейсу в індустріальних бездротових мережах та автоматизованих системах керування. Отримані результати узгоджуються між собою та підтверджують, що розробка має не лише наукову й технічну цінність, а й реальні перспективи практичного впровадження. Запропоноване програмне рішення здатне забезпечити підвищення ефективності обміну даними, зниження затримок та покращення надійності функціонування індустріальних автоматизованих систем. Таким чином, виконання науково-дослідної роботи є обґрунтованим, економічно доцільним і перспективним з точки зору подальшого впровадження та комерційного використання результатів розробки.

ВИСНОВКИ

Проведене дослідження з модернізації програмного модуля для промислових бездротових мереж продемонструвало значний потенціал для покращення ключових характеристик системи передачі даних. Отримані результати експериментальних випробувань переконливо свідчать, що впровадження комплексу технологічних рішень – переходу на протокол HTTP/3 у поєднанні з алгоритмом стиснення Zstd:6 і форматом даних MessagePack – забезпечує відчутні переваги порівняно з традиційною конфігурацією, заснованою на HTTP/2, алгоритмі GZIP:6 та форматі JSON. Модернізована система показала істотне зростання продуктивності: кількість оброблених запитів за секунду збільшилася майже на дві третини, а обсяг переданих даних за одиницю часу зріс у схожій пропорції. Це підвищення пропускної здатності є особливо важливим для промислових застосувань, у яких обробка великих обсягів інформації в реальному часі визначає ефективність функціонування виробничих процесів, систем моніторингу та керування.

Крім підвищення продуктивності, модернізація забезпечила суттєву оптимізацію використання обчислювальних ресурсів. Середнє споживання пам’яті зменшилося майже наполовину, а навантаження на процесор – приблизно на третину, що свідчить про підвищення енергоефективності та ресурсної стабільності системи. Таке зниження навантаження має практичну цінність для промислових пристроїв, які часто функціонують у середовищах із обмеженими апаратними ресурсами та високими вимогами до енергоспоживання. Оптимізація обчислювальних витрат не лише скорочує енергетичні затрати, але й дає змогу задіяти вивільнені ресурси для виконання додаткових задач – наприклад, для розширення аналітичних можливостей системи чи підвищення рівня інтеграції з іншими компонентами виробничої інфраструктури.

Важливою складовою покращення стала й швидкодія обробки даних. Значне скорочення часу стиснення (на понад 85%) та часу декомпресії (на більш ніж 90%) безпосередньо зменшило затримки передачі, що має вирішальне значення для систем реального часу. Швидше стиснення й розпакування даних

дозволяє забезпечити оперативну реакцію на зміни у виробничих процесах, покращити стабільність взаємодії між компонентами мережі та підвищити безпеку технологічних операцій завдяки своєчасній передачі критичної інформації.

Таким чином, узагальнення отриманих результатів дає підстави стверджувати, що впровадження протоколу HTTP/3, алгоритму Zstd:6 і формату даних MessagePack є ефективним шляхом підвищення якості та ефективності систем передачі даних у промислових бездротових мережах. Запропонована архітектура поєднує високу продуктивність, ресурсну ощадність та мінімальні затримки, що робить її оптимальним вибором для сучасних промислових застосувань, у яких критично важливими є надійність, стабільність і швидкодія.

Отримані результати також мають стратегічне значення у контексті Цілі сталого розвитку №9 «Промисловість, інновації та інфраструктура», зокрема завдання 9.4, яке спрямоване на розвиток високотехнологічних секторів промисловості та підвищення ефективності використання ресурсів. Розроблені в межах дослідження рішення щодо модернізації програмного модуля промислових бездротових мереж – через впровадження HTTP/3, Zstd:6 і MessagePack – безпосередньо сприяють підвищенню показників індикатора 9.4.1, що відображає рівень ефективності виробничих процесів і технологічних інновацій.

Покращення у використанні обчислювальних ресурсів, скорочення енергоспоживання, зростання швидкодії передачі даних та підвищення надійності комунікацій безпосередньо впливають на ефективність виробництва електронної та комп’ютерної продукції, яка входить до високотехнологічного сектору відповідно до класифікації КВЕД. Це, у свою чергу, сприяє підвищенню конкурентоспроможності українських підприємств на міжнародному ринку та зміцненню національного науково-технологічного потенціалу.

Отже, результати проведеного дослідження підтверджують важливість комплексного підходу до оптимізації систем передачі даних, що передбачає врахування взаємодії різних технологічних компонентів – протоколів, алгоритмів стиснення та форматів даних – і їхнього впливу на загальну ефективність системи. Модернізований підхід до побудови промислових мереж

створює передумови для формування нової генерації інтелектуальних інфраструктур, орієнтованих на стійкість, інноваційність і ресурсну збалансованість.

ПЕРЕЛІК ПОСИЛАНЬ

[1] M. Bishop, “HTTP/3,” RFC 9114, pp. 1–57, Jun. 2022. DOI: 10.17487/RFC9114.

[2] J. Iyengar and I. Swett, “QUIC Loss Detection and Congestion Control,” RFC 9002, pp. 1–42, May 2021. DOI: 10.17487/RFC9002.

[3] M. Thomson and S. Turner, “Using TLS to Secure QUIC,” RFC 9001, pp. 1–52, May 2021. DOI: 10.17487/RFC9001.

[4] J. Iyengar and M. Thomson, “QUIC: A UDP-Based Multiplexed and Secure Transport,” RFC 9000, pp. 1–151, May 2021. DOI: 10.17487/RFC9000.

[5] C. B. Krasic, M. Bishop, and A. Frindell, “QPACK: Field Compression for HTTP/3,” RFC 9204, pp. 1–41, Jun. 2022. DOI: 10.17487/RFC9204.

[6] J. Alakuijala and Z. Szabadka, “Brotli Compressed Data Format,” RFC 7932, pp. 1–128, Jul. 2016. DOI: 10.17487/RFC7932.

[7] Y. Collet and M. Kucherawy, “Zstandard Compression and the ‘application/zstd’ Media Type,” RFC 8878, pp. 1–45, Feb. 2021. DOI: 10.17487/RFC8878.

[8] P. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3,” RFC 1951, pp. 1–41, May 1996. DOI: 10.17487/RFC1951.

[9] P. Deutsch, “GZIP file format specification version 4.3,” RFC 1952, pp. 1–19, May 1996. DOI: 10.17487/RFC1952.

[10] R. Fielding and M. Nottingham, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 8259, pp. 1–20, Dec. 2017. DOI: 10.17487/RFC8259.

[11] M. Belshe, R. Peon, and M. Thomson, “Hypertext Transfer Protocol Version 2 (HTTP/2),” RFC 7540, pp. 1–97, May 2015. DOI: 10.17487/RFC7540.

[12] R. Peon and H. Ruellan, “HPACK: Header Compression for HTTP/2,” RFC 7541, pp. 1–55, May 2015. DOI: 10.17487/RFC7541.

[13] C. Bormann and P. E. Hoffman, “Concise Binary Object Representation (CBOR),” RFC 8949, pp. 1–66, Dec. 2020. DOI: 10.17487/RFC8949. ([IETF Datatracker][11])

[14] C. Bormann and P. E. Hoffman, “Concise Binary Object Representation (CBOR),” RFC 7049, pp. 1–54, Oct. 2013. DOI: 10.17487/RFC7049.

[15] M. Seemann, B. Trammell, and M. Kühlewind, “Applicability of the QUIC Transport Protocol,” RFC 9308, pp. 1–31, Oct. 2022. DOI: 10.17487/RFC9308.

[16] T. Bünzli, O. Alay, D. Koll, and A. Dhamdhere, “HTTP/3: Performance, Deployment and Adoption,” IEEE Trans. Netw. Serv. Manag., vol. 21, no. 3, pp. 2056–2074, 2024. DOI: 10.1109/TNSM.2024.3467485.

[17] A. Langley et al., “The QUIC Transport Protocol: Design and Internet-Scale Deployment,” Proc. ACM SIGCOMM, 2017, pp. 183–196. DOI: 10.1145/3098822.3098842.

[18] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, “Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols,” Proc. 2017 Internet Measurement Conf. (IMC ’17), pp. 290–303. DOI: 10.1145/3131365.3131368.

[19] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, “A First Look at QUIC in the Wild,” arXiv:1801.05168, 2018, pp. 1–12 (preprint).

[20] K. Wolsing, T. Böttger, and G. Tyson, “TCP+TLS+HTTP/2 vs. QUIC,” Proc. ACM/IRTF/ISOC ANRW ’19, 2019, pp. 1–7.

[21] A. Yu, V. Edalat, T. Benson, and A. Akella, “Dissecting Performance of Production QUIC,” Proc. The Web Conf. (WWW) 2021, pp. 171–183.

[22] G. Carbone, M. Dell’Amico, and S. Traverso, “A First Look at HTTP/3 Adoption and Performance,” Comput. Commun., vol. 188, pp. 115–124, 2022. DOI: 10.1016/j.comcom.2022.01.017.

[23] “MessagePack: An Efficient Binary Serialization Format,” Official Specification, pp. 1–20 (web spec). DOI: –. Available: [https://msgpack.org] (accessed Oct. 13, 2025).

[24] “Protocol Buffers: Google’s Data Interchange Format,” Google Developers Guide, pp. 1–30 (web doc). DOI: –. Available: [https://developers.google.com/protocol-buffers] (accessed Oct. 13, 2025).

[25] Y. Collet, “Zstandard – Fast Real-Time Compression Algorithm,” Facebook Engineering (now Meta), Design Doc, pp. 1–45 (whitepaper-style description). DOI: –. Available: [https://facebook.github.io/zstd/] (accessed Oct. 13, 2025).

[26] J. Alakuijala and Z. Szabadka, “Brotli: A general-purpose data compressor,” Commun. ACM, vol. 61, no. 6, pp. 45–53, 2018. DOI: 10.1145/3231935.

ДОДАТКИ

Додаток А

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: _Розробка програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах_

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра КСУ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 2,59 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☑ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

- У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Ковтун В.В., завідувач кафедри КСУ _____
(прізвище, ініціали, посада) (підпис)

Ковалюк О.О., доцент кафедри КСУ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Дубовой В.М.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ковтун В.В., завідувач кафедри КСУ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Кребс Д.В.
(підпис) (прізвище, ініціали)

Додаток Б
(обов'язковий)
Технічне завдання

ЗАТВЕРДЖУЮ

Завідувач кафедри КСУ
д.т.н., проф. В.В.Ковтун

«17» жовтня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Розробка програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах»

08-01.МКР.**00**_.00.000 ТЗ

Керівник роботи:
завідувач кафедри КСУ, професор
В.В. Ковтун
«16» 10 2025р.

Виконавець: ст. гр. 1АКІТР-24М
Д.В. Кребс
«16» 10 2025р.

Вінниця 2025

1. Назва та галузь застосування

1.1. Назва – Розробка програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах

1.2. Галузь застосування – Промисловий Інтернет речей (IIoT), автоматизовані системи управління технологічними процесами та віддалений моніторинг

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ №__313 від 24.09.2025р.

3. Мета та призначення розробки.

Мета магістерської кваліфікаційної роботи полягає у розробленні програмного інтерфейсу автоматизованої системи бездротового обміну даними в індустріальних мережах, який підвищує ефективність передавання інформації завдяки використанню адаптивних алгоритмів оптимізації потоків, сучасних протоколів комунікації та вдосконалених методів серіалізації. Досягнення мети передбачає створення гнучкої програмної архітектури, що забезпечує стабільну роботу системи в умовах перешкод і мінімізує втрати даних під час обміну між промисловими вузлами.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше, оскільки пропонує новий підхід до поєднання транспортних протоколів та форматів даних у контексті IIoT. В ході проведення розробки повинні використовуватись такі документи:

1. M. Bishop, “HTTP/3,” RFC 9114, pp. 1-57, Jun. 2022. DOI: 10.17487/RFC9114.
2. J. Iyengar and I. Swett, “QUIC Loss Detection and Congestion Control,” RFC 9002, pp. 1-42, May 2021. DOI: 10.17487/RFC9002.
3. Y. Collet and M. Kucherawy, “Zstandard Compression and the ‘application/zstd’ Media Type,” RFC 8878, pp. 1-45, Feb. 2021. DOI: 10.17487/RFC8878.
4. R. Fielding and M. Nottingham, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 8259, pp. 1-20, Dec. 2017. DOI: 10.17487/RFC8259.

5. T. Bünzli, O. Alay, D. Koll, and A. Dhamdhere, “HTTP/3: Performance, Deployment and Adoption,” IEEE Trans. Netw. Serv. Manag., vol. 21, no. 3, pp. 2056-2074, 2024. DOI: 10.1109/TNSM.2024.3467485.

5. Вимоги до розробки.

5.1. Перелік головних функцій:

– реалізація клієнтської та серверної логіки з підтримкою протоколів HTTP/2 та HTTP/3 (QUIC);
– інтеграція модулів стиснення (GZIP та Zstandard) у транспортний рівень;
– реалізація серіалізації/десеріалізації даних у форматах JSON та MessagePack;
– розробка модуля емуляції бездротової мережі для контрольованого тестування (затримки, втрати пакетів);
– автоматизований збір та аналіз ключових метрик (RPS, CPU, Memory, Latency).

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:
- ОС для тестового стенда: Linux або Windows 10/11;
- мова реалізації: Go (Golang) для забезпечення високої паралельності та продуктивності;
- підтримка мережевих протоколів: IPv4 та IPv6.

5.2.2. Умови експлуатації системи:
- робота на стандартних ПЕОМ та низькоресурсних пристроях (симуляція IIoT-вузлів);
- можливість цілодобового функціонування системи у промислових умовах;
- текст програмного забезпечення системи є відкритим для подальшого розвитку та інтеграції.

6. Стадії та етапи розробки.

6.1. Пояснювальна записка:

Дослідження актуальності поставленої задачі «26» вересня 2025р.
Загальний огляд сучасних транспортних протоколів (HTTP/2, HTTP/3, MQTT) «1» жовтня 2025р.
Аналіз та обґрунтування оптимального стека «11» жовтня 2025р..

Розробка структури програмного забезпечення та архітектурних діаграм « 29 » жовтня 2025р.

6.2. Графічні матеріали:

– Розробка UML-діаграм « 5 » листопада 2025р.
– Тестування та порівняльний аналіз конфігурацій « 17 » листопада 2025р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «25 » листопада 2025 р.

7.2. Атестація проєкту здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до « 10 » грудня 2025 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ДЕК. Захист магістерської кваліфікаційної роботи провести до « 1 8 » грудня 2025 р.

Додаток В

(вибірковий)

Лістинги

А.1. Лістинг 1 – Конфігураційний файл для налаштування середовища тестування

yaml

```
# config.yml – базові параметри експериментального середовища
compression_algorithms:
  - gzip
  - zstd
  - lz4
  - brotli
  - deflate
protocols:
  - HTTP/2
  - HTTP/3
dataset_path: ./data/sensor_payloads.json
server_host: 127.0.0.1
server_port: 8080
client_threads: 4
request_count: 1000
output_dir: ./results/
```

Призначення: конфігураційний файл визначає набір алгоритмів стиснення, протоколів передачі даних, а також базові параметри серверної частини експерименту.

А.2. Лістинг 2 – Серверна частина тестового середовища

python

```
# server.py – HTTP/2 та HTTP/3 сервер для експериментів зі стисненням
```

```
import aioquic
from aiohttp import web
import zstandard as zstd
import brotli
import gzip

async def handle_request(request):
    data = await request.read()
    algo = request.query.get('compression')
    if algo == "zstd":
        compressed = zstd.ZstdCompressor(level=6).compress(data)
    elif algo == "gzip":
        compressed = gzip.compress(data, compresslevel=6)
    elif algo == "brotli":
        compressed = brotli.compress(data)
    else:
        compressed = data
    return web.Response(body=compressed)
```

Призначення: реалізує обробку HTTP-запитів із використанням різних алгоритмів стиснення (gzip, zstd, brotli) для порівняння швидкодії та ефективності.

А.3. Лістинг 3 – Клієнтська частина експериментальної системи

```
python
# client.py – клієнт для вимірювання часу стиснення, розтиснення та RPS
import  aiohttp
import asyncio
import time
```

```
async def run_test(protocol, compression, iterations):
    start = time.perf_counter()
    async with aiohttp.ClientSession() as session:
        for _ in range(iterations):
            async with session.get(
                f"{protocol}://127.0.0.1:8080?compression={compression}"
            ) as resp:
                await resp.read()
    elapsed = time.perf_counter() - start
    return elapsed

async def main():
    for protocol in ["http2", "http3"]:
        for comp in ["gzip", "zstd", "brotli", "lz4"]:
            t = await run_test(protocol, comp, 100)
            print(protocol, comp, t)

asyncio.run(main())
```

Призначення: клієнт проводить серію запитів до серверу, вимірює час відповіді та обчислює кількість запитів за секунду (RPS).

А.4. Лістинг 4 – Обчислення ступеня стиснення

```
python
# compression_ratio.py
import os

def compression_ratio(original_file, compressed_file):
    orig_size = os.path.getsize(original_file)
```

```
    comp_size = os.path.getsize(compressed_file)
    return orig_size / comp_size

print("Compression ratio:", compression_ratio("payload.json", "payload.zst"))
```

Призначення: модуль визначає ступінь стиснення даних для кожного алгоритму.

А.5. Лістинг 5 – Тестування формату даних (JSON, MessagePack, Protobuf)

```
python
# format_benchmark.py
import json, msgpack, time, random
from google.protobuf.message import EncodeError

def generate_sample():
    return {"id": random.randint(0, 1000), "temp": 23.5, "status": "OK"}

def measure(serializer, iterations=10000):
    start = time.perf_counter()
    for _ in range(iterations):
        data = serializer(generate_sample())
    return time.perf_counter() - start

json_time = measure(lambda d: json.dumps(d).encode())
msgpack_time = measure(lambda d: msgpack.packb(d))

print(f"JSON time: {json_time:.4f}s | MessagePack: {msgpack_time:.4f}s")
```

Призначення: демонструє різницю у швидкодії серіалізації між JSON і MessagePack, що використовувались у дослідженні (див. розділ 3.5.7).

А.6. Лістинг 6 – Розрахунок середніх показників та побудова порівняльних діаграм

python

```
# analyze_results.py
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("results.csv")
summary = df.groupby(["protocol", "compression"]).mean()

summary.to_csv("summary.csv")
summary.plot(kind="bar", y="rps", title="Requests per Second (RPS)")
plt.xlabel("Protocol + Compression")
plt.ylabel("RPS")
plt.show()
```

Призначення: обробляє результати тестів, розраховує середні значення RPS, часу стиснення/розтиснення, використання пам’яті та процесора.

А.7. Лістинг 7 – Порівняльна таблиця підсумкових показників

python

```
# summary_table.py
import pandas as pd

data = {
    "Config": ["HTTP/2 + gzip:6 + JSON", "HTTP/3 + zstd:6 + MessagePack"],
```

```
    "RPS": [2976.862, 4915.785],
    "CPU (%)": [74.048, 48.709],
    "Memory (MB)": [107.782, 55.679],
    "Compression Time (ms)": [4.246, 0.622],
    "Decompression Time (ms)": [2.69, 0.231]
}

df = pd.DataFrame(data)
print(df)
```

Призначення: генерує підсумкову таблицю для автоматичного включення в документ (табл. 3.1).

А.8. Лістинг 8 – Приклад інтеграції системи у промислову мережу

```
python
# mqtt_bridge.py
import paho.mqtt.client as mqtt
import msgpack, zstandard as zstd

def on_message(client, userdata, msg):
    data = zstd.ZstdDecompressor().decompress(msg.payload)
    payload = msgpack.unpackb(data)
    print("Sensor data received:", payload)

client = mqtt.Client()
client.connect("192.168.1.10", 1883, 60)
client.subscribe("factory/sensors")
client.on_message = on_message
client.loop_forever()
```

Призначення: демонструє практичне застосування обраної конфігурації HTTP/3 + zstd:6 + MessagePack у контексті промислової бездротової мережі (наприклад, MQTT-шлюзу).

Додаток Г

(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

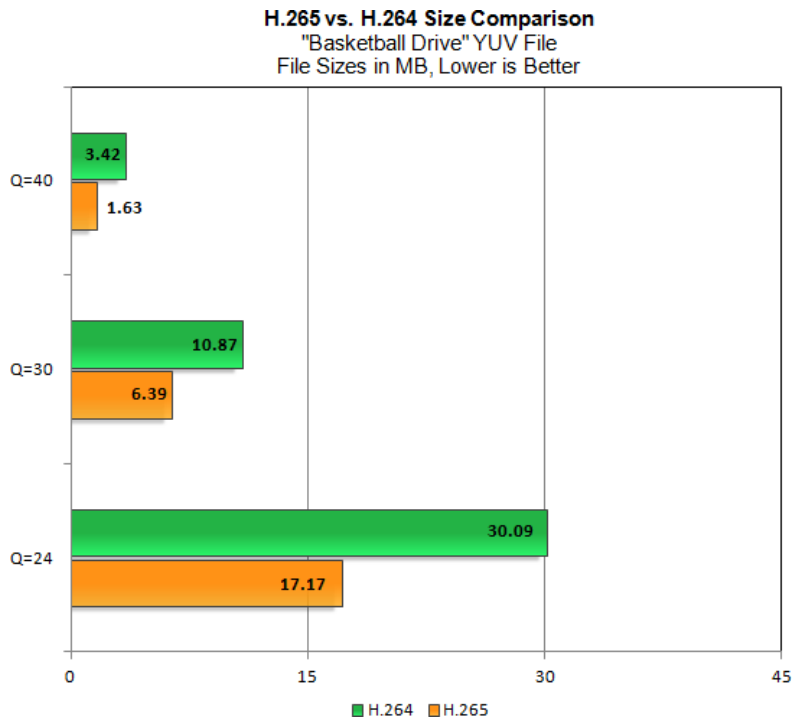
РОЗРОБКА ПРОГРАМНОГО ІНТЕРФЕЙСУ АВТОМАТИЗОВАНОЇ СИСТЕМИ БЕЗДРОТОВОГО ОБМІНУ ДАНИМИ В ІНДУСТРІАЛЬНИХ МЕРЕЖАХ

Перелік ілюстративних матеріалів:

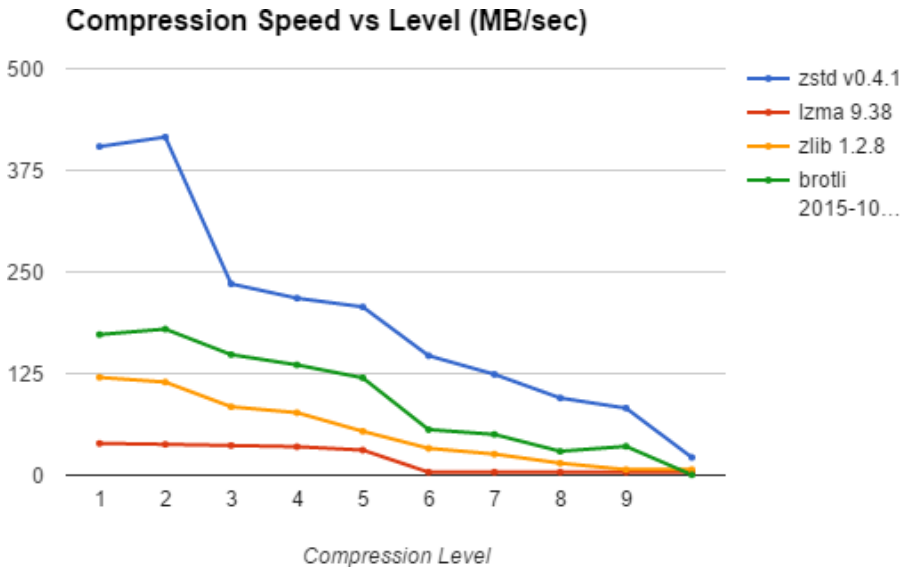
1. Порівняння розміру відеофайлів при різних алгоритмах стискання
2. Залежність швидкості та коефіцієнта стиснення для різних алгоритмів
3. Порівняння затримки при використанні HTTP/1, HTTP/2 та HTTP/3 (QUIC)
4. Порівняння JSON та Protocol Buffers у мові програмування Go
5. Технічні характеристики серверного процесора AMD Ryzen 5 5700X3D, використаного у реалізації інтерфейсу системи
6. Твердотільний накопичувач Samsung 970 Evo Plus, використаний у серверному модулі системи
7. Маршрутизатор TP-Link Archer C20, застосований для дослідження бездротового каналу системи
8. Структури метрик та інтерфейси їх оновлення
9. Реалізація HTTP/2 та HTTP/3 серверів з уніфікованими маршрутами
10. Логіка Bash-скрипту для перебору конфігурацій і звітування статусів
11. Приклад результату виконання тесту
12. Порівняння результатів часу стиснення
13. Оновлений графік часу стиснення
14. Оновлений графік часу стиснення після остаточної фільтрації
15. Порівняння часу декомпресії
16. Порівняння ступенів стиснення для різних алгоритмів
17. Порівняння використання процесора різними алгоритмами стиснення
18. Порівняння використання пам'яті різними алгоритмами стиснення
19. Кількість запитів за секунду для різних алгоритмів стиснення при HTTP/3
20. Порівняння ефективності форматів даних при використанні HTTP/3 і Zstd:6

Студент (підпис) Денис КРЕБС

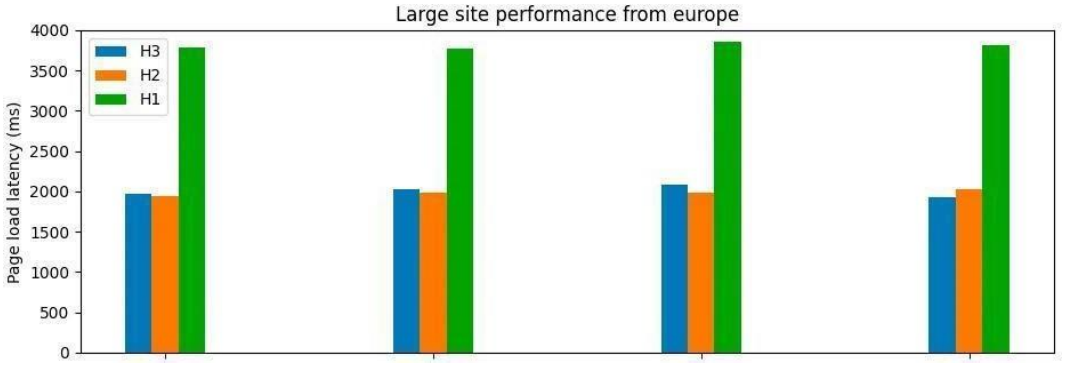
Керівник роботи (підпис) В'ячеслав КОВТУН



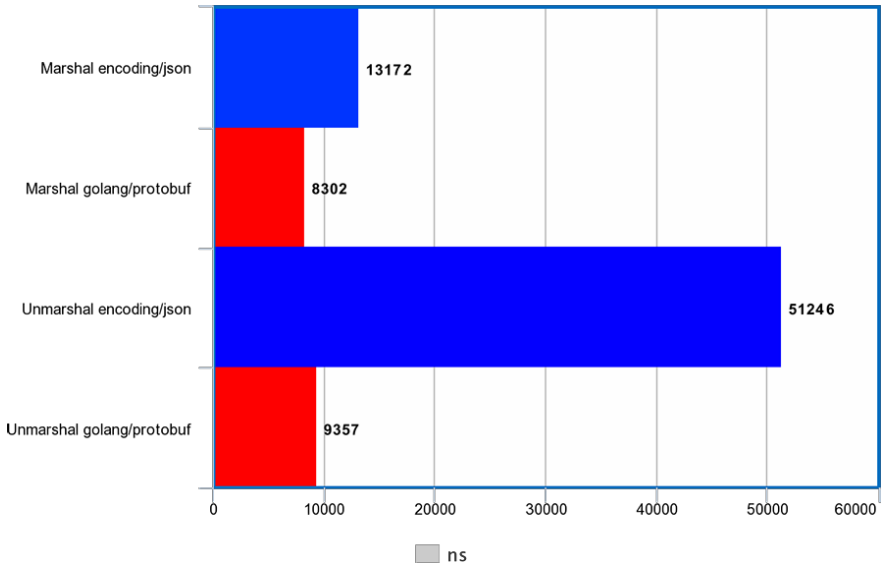
Порівняння розміру відеофайлів при різних алгоритмах стискання



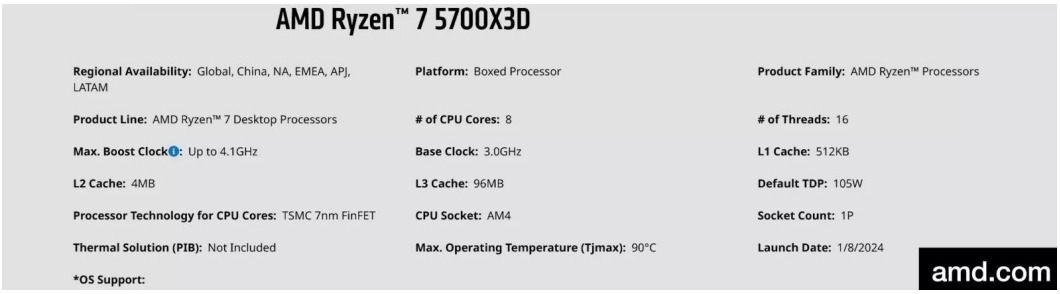
Залежність швидкості та коефіцієнта стиснення для різних алгоритмів



Порівняння затримки при використанні HTTP/1, HTTP/2 та HTTP/3 (QUIC)



Порівняння JSON та Protocol Buffers у мові програмування Go



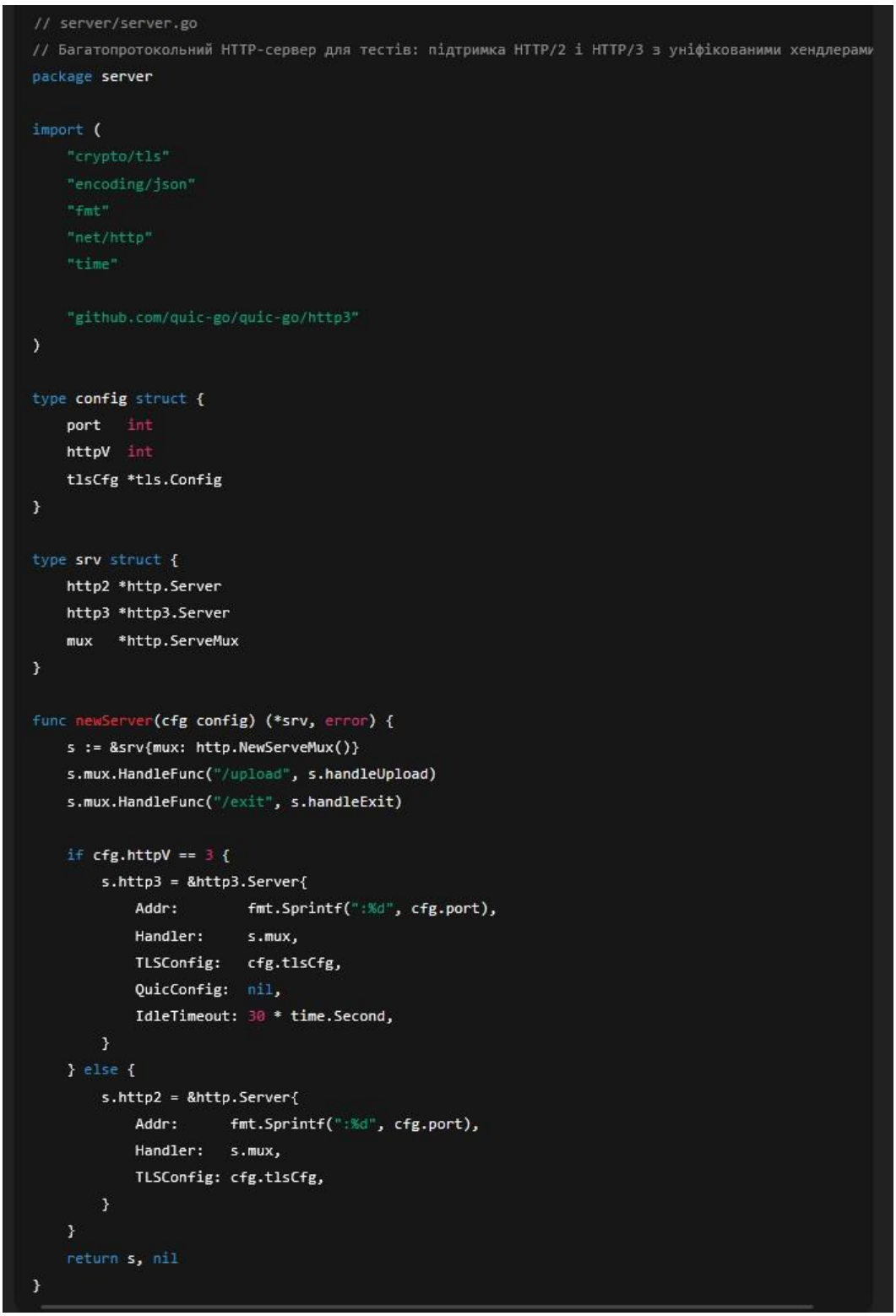
Технічні характеристики серверного процесора AMD Ryzen 5 5700X3D, використаного у реалізації інтерфейсу системи

Твердотільний накопичувач Samsung 970 Evo Plus, використаний у серверному модулі системи

Маршрутизатор TP-Link Archer C20, застосований для дослідження бездротового каналу системи

```go
// logic/metrics.go
// Підсистема метрик для бездротового інтерфейсу: мінімізуємо блокування, зберігаємо вибірки розпс
package logic

import (
    "sync"
    "sync/atomic"
)

type Metrics struct {
    // Роздільні м'ютекси зменшують змагання між потоками при масовому оновленні масивів вибірок.
    timingMu   sync.Mutex
    resourceMu sync.Mutex

    // Атомарні лічильники: підсумкові значення накопичуються без м'ютексів.
    TotalRequests  atomic.Uint64
    ServerBytesIn  atomic.Uint64

    // Вибірки часових метрик у наносекундах: запис/читання, маршал/анмаршал, компресія/декомпресі
    writeTimeNS     []float64
    readTimeNS      []float64
    marshalTimeNS   []float64
    unmarshalTimeNS []float64
    compressTimeNS  []float64
    decompressTimeNS []float64

    // Розміри до/після компресії для обчислення коефіцієнтів.
    compressedSize   []float64
    uncompressedSize []float64
    decodedSize      []float64

    // Профілі використання ресурсів, що знімаються періодично.
    memSamples []float64
    cpuSamples []float64
}

// Додавання вибірок виконується під відповідним м'ютексом, щоб гарантувати цілісність масивів.
func (m *Metrics) addTimingSample(dst *[]float64, v float64) {
    m.timingMu.Lock()
    *dst = append(*dst, v)
    m.timingMu.Unlock()
}

func (m *Metrics) addResourceSample(memMB, cpuPct float64) {
    m.resourceMu.Lock()
    m.memSamples = append(m.memSamples, memMB)
    m.cpuSamples = append(m.cpuSamples, cpuPct)
    m.resourceMu.Unlock()
}
```

Структури метрик та інтерфейси їх оновлення



```go
// server/server.go
// Багатопротокольний HTTP-сервер для тестів: підтримка HTTP/2 і HTTP/3 з уніфікованими хендлерами
package server

import (
    "crypto/tls"
    "encoding/json"
    "fmt"
    "net/http"
    "time"

    "github.com/quic-go/quic-go/http3"
)

type config struct {
    port   int
    httpV  int
    tlsCfg *tls.Config
}

type srv struct {
    http2 *http.Server
    http3 *http3.Server
    mux   *http.ServeMux
}

func newServer(cfg config) (*srv, error) {
    s := &srv{mux: http.NewServeMux()}
    s.mux.HandleFunc("/upload", s.handleUpload)
    s.mux.HandleFunc("/exit", s.handleExit)

    if cfg.httpV == 3 {
        s.http3 = &http3.Server{
            Addr:        fmt.Sprintf(":%d", cfg.port),
            Handler:     s.mux,
            TLSConfig:   cfg.tlsCfg,
            QuicConfig:  nil,
            IdleTimeout: 30 * time.Second,
        }
    } else {
        s.http2 = &http.Server{
            Addr:      fmt.Sprintf(":%d", cfg.port),
            Handler:   s.mux,
            TLSConfig: cfg.tlsCfg,
        }
    }
    return s, nil
}
```

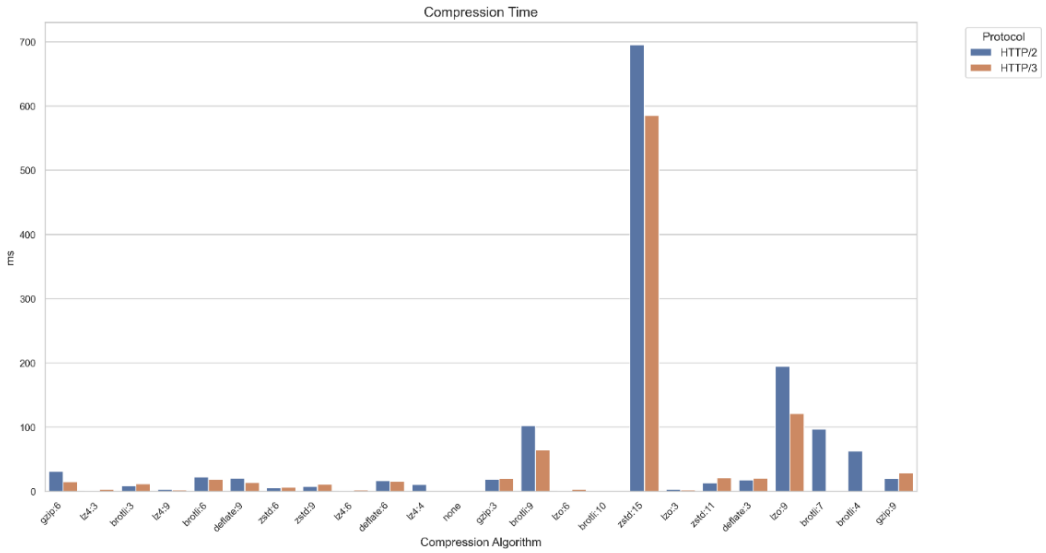
Реалізація HTTP/2 та HTTP/3 серверів з уніфікованими маршрутами

```bash
#!/usr/bin/env bash
# scripts/run_matrix.sh
# Автоматизований перебір конфігурацій тестів для бездротового інтерфейсу.

CLIENT_TYPES=("curl_h2 --http2" "curl_h3 --http3")
DATA_FORMATS=("json" "msgpack" "protobuf")
COMPRESSION_OPTIONS=("none" "gzip" "zstd" "lz4" "brotli" "deflate" "lzo")

BASE_PORT=8000
SERVER_COUNT=0
FAILED_TESTS=0

for client_spec in "${CLIENT_TYPES[@]}"; do
  IFS=' ' read -r client_type client_args <<< "$client_spec"
  for data_format in "${DATA_FORMATS[@]}"; do
    for compression in "${COMPRESSION_OPTIONS[@]}"; do
      PORT=$((BASE_PORT + SERVER_COUNT))
      echo "============================================================"
      echo " Running test $((SERVER_COUNT + 1))"
      echo " ... Client: $client_type"
      echo " ... Args:   $client_args"
      echo " ... Format: $data_format"
      echo " ... Compression: $compression"
      echo "============================================================"
      if ! ./run_client "$client_type" "$data_format" "$compression" "$client_args" --port "$PORT"
        echo "[FAILED] Test failed"; ((FAILED_TESTS++)); continue
      fi
      echo "[SUCCESS] Test completed successfully"
      sleep 1
      SERVER_COUNT=$((SERVER_COUNT + 1))
    done
  done
done

echo "Total tests: $SERVER_COUNT; Failed: $FAILED_TESTS"
exit $(( FAILED_TESTS > 0 ))
```

Логіка Bash-скрипту для перебору конфігурацій і звітування статусів

```
================================================================================
Test Summary
================================================================================
Total tests run: 160
Failed tests:
Successful tests: 160
Test log available at: ./out/client_test.log
================================================================================
Client tests completed.
```

Приклад результату виконання тесту



Порівняння результатів часу стиснення

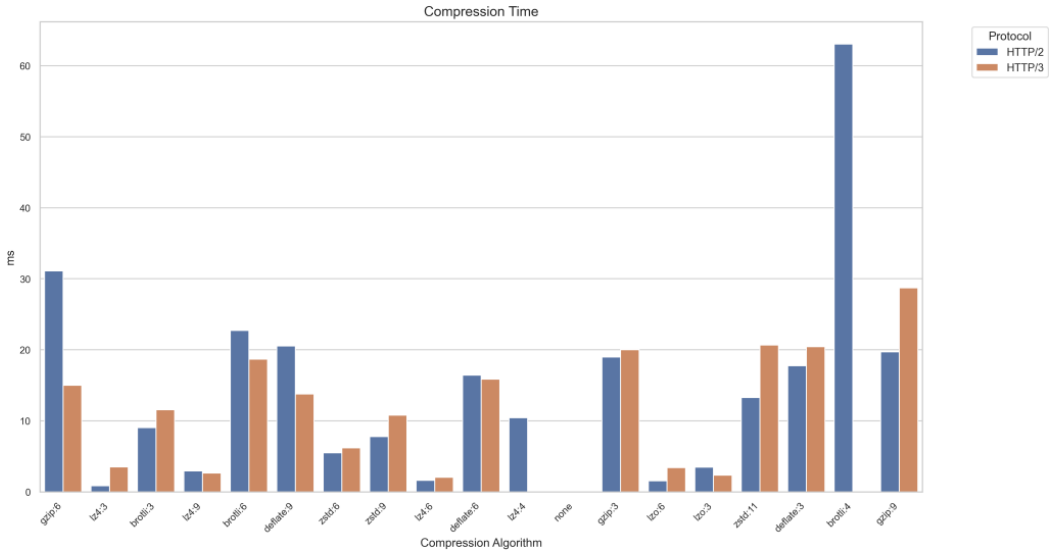
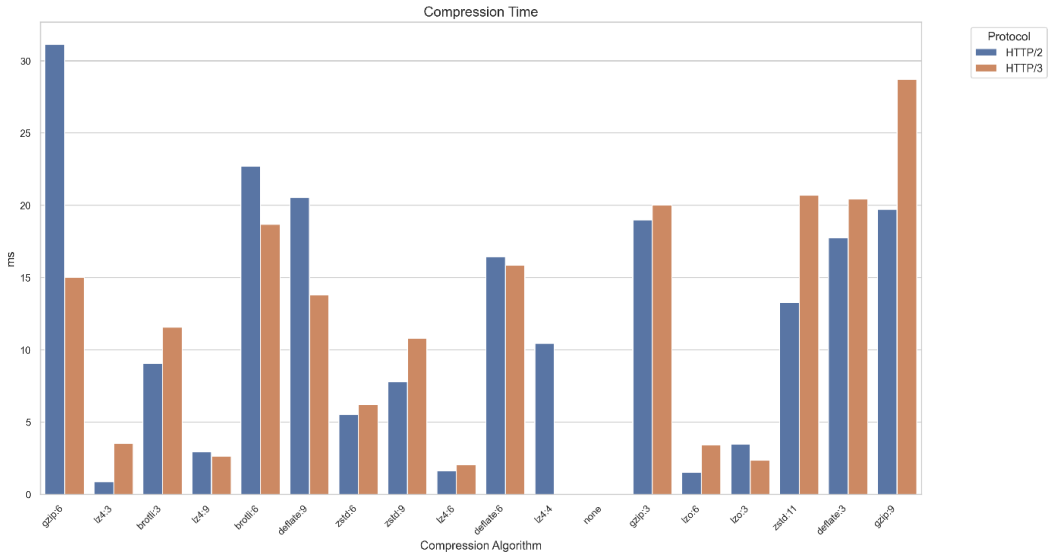
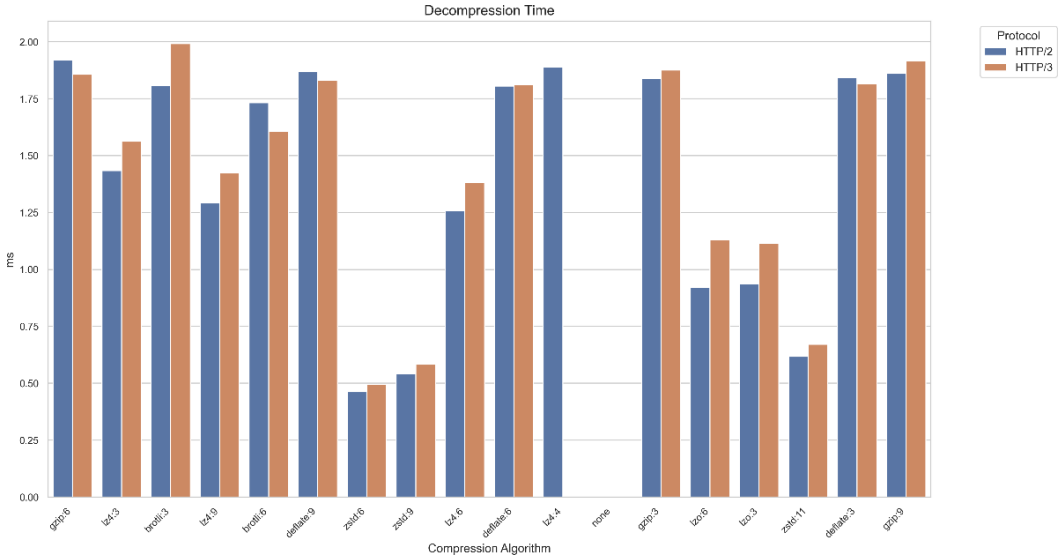


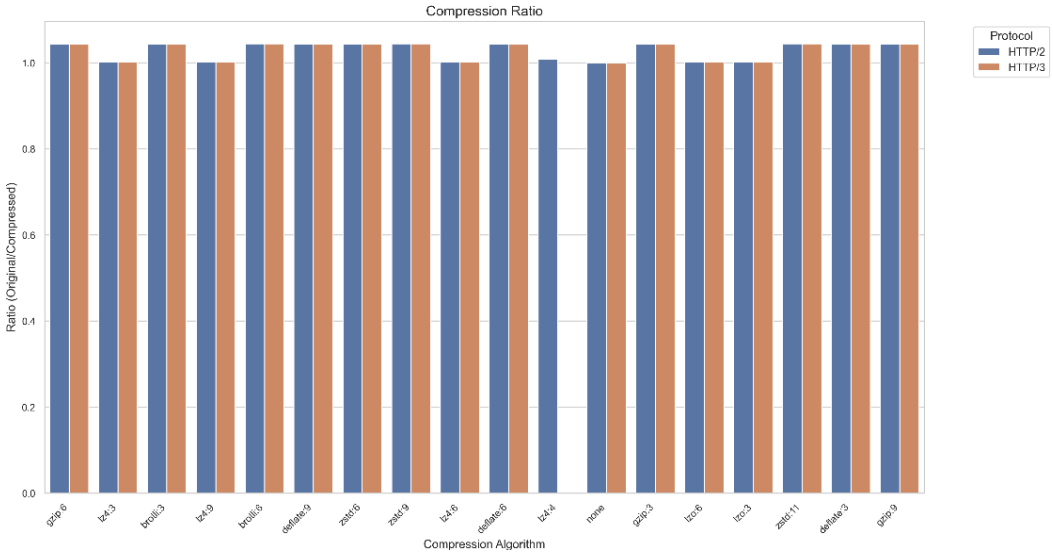
Рисунок 3.23 – Оновлений графік часу стиснення



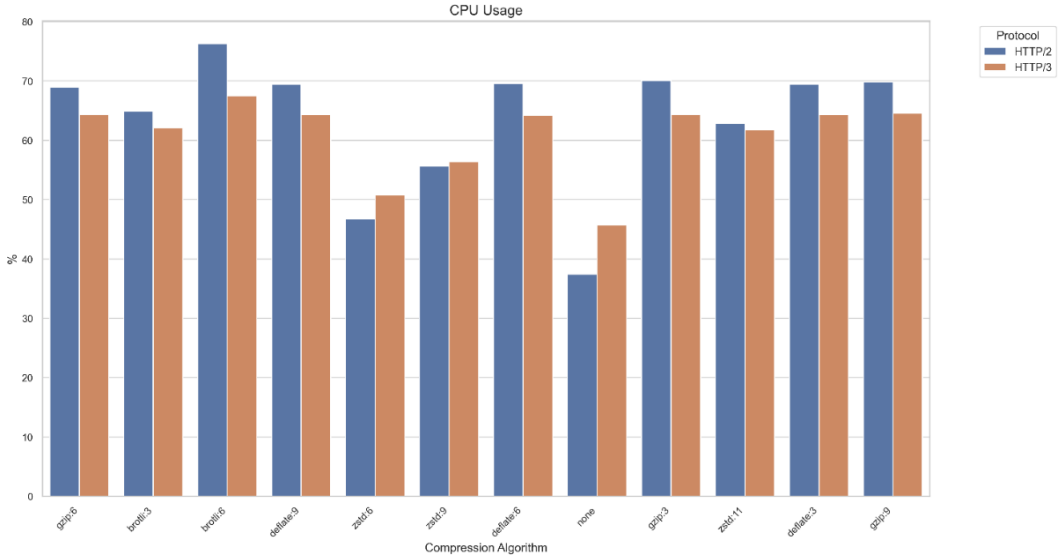
Оновлений графік часу стиснення після остаточної фільтрації



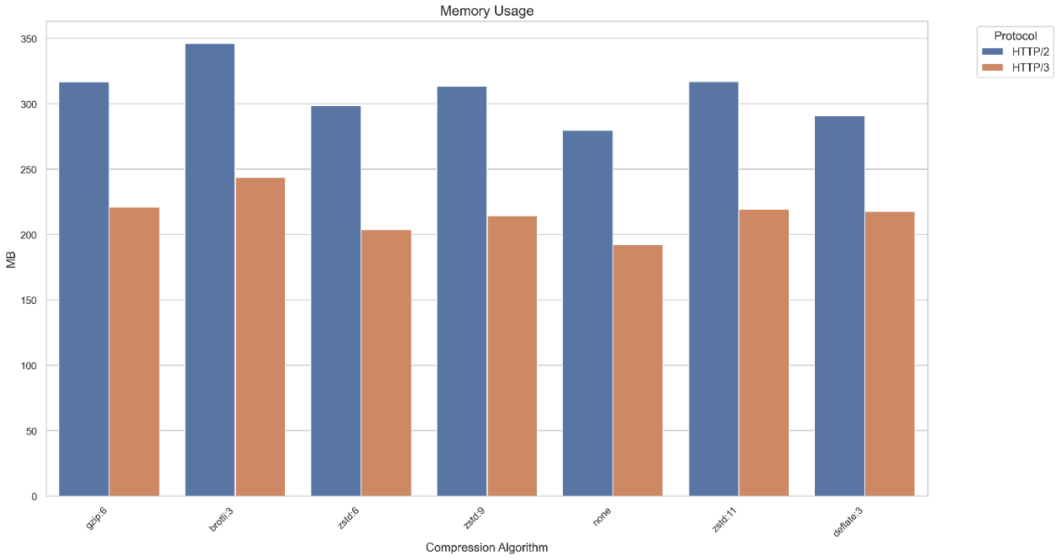
Порівняння часу декомпресії



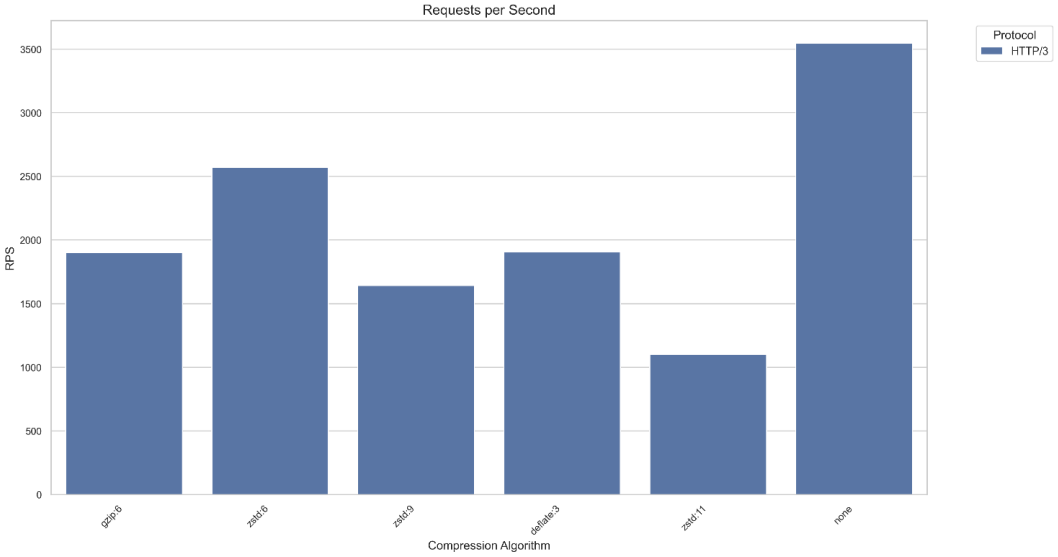
Порівняння ступенів стиснення для різних алгоритмів



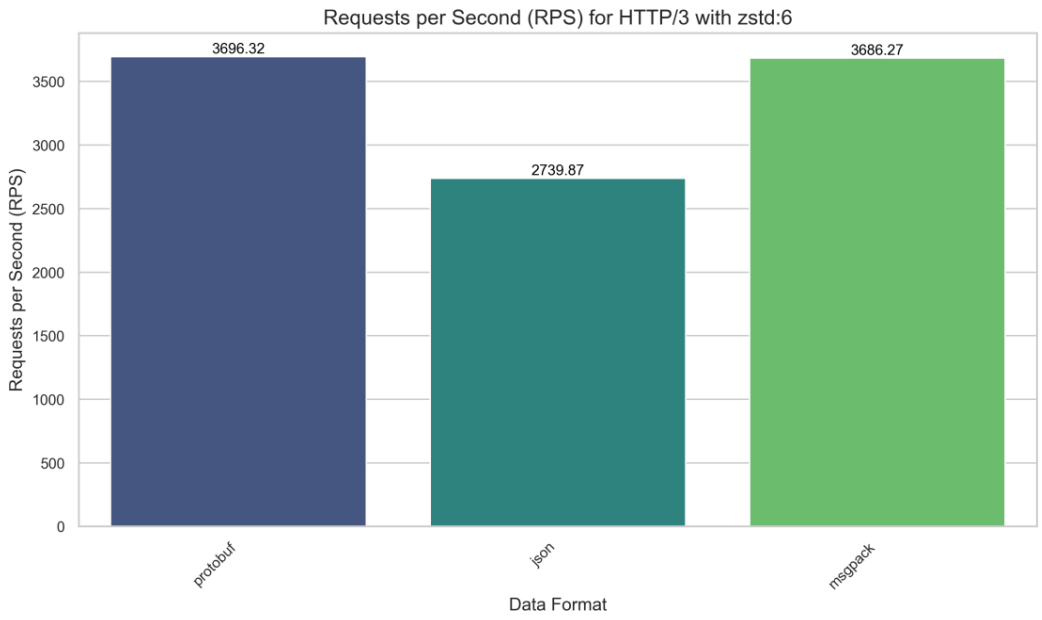
Порівняння використання процесора різними алгоритмами стиснення



Порівняння використання пам’яті різними алгоритмами стиснення



Кількість запитів за секунду для різних алгоритмів стиснення при HTTP/3



Порівняння ефективності форматів даних при використанні HTTP/3 і Zstd:6