Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп’ютеризованих технологій
(повна назва)

Кафедра Комп’ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ________другий (магістерський)________________________
Модернізація програмного модуля комп’ютеризованої
системи безпровідної передачі інформації в промислових мережах
(тема)

Виконав:
здобувач 2 року навчання,
групи КТРСм-23-2

_________Дяченко Едуард Станіславович __________
(прізвище, ініціали)

Спеціальність _174 Автоматизація, комп’ютерно-інтегровані технології та робототехніка_
(код і повна назва спеціальності)

Тип програми ____Освітньо-професійна____________
(освітньо-професійна або освітньо-наукова)

Освітня програма ___Комп’ютеризовані та роботизовані системи_____________
(повна назва освітньої програми)

Керівник _______доц. каф. КІТАР Стародубцев М. Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ___________ _Невлюдов І. Ш._______
(підпис) (прізвище, ініціали)

2025 р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКТ
Кафедра КІТАР
Рівень вищої освіти другий (магістерський)
Спеціальність 174 Автоматизація, комп’ютерно-інтегровані технології та робототехніка
Тип програми Освітньо-професійна
Освітня програма Комп’ютеризовані та роботизовані системи
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри ___________________
(підпис)
«__»______________2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Дяченку Едуарду Станіславовичу
(шифр і назва)

1. Тема роботи: Модернізація програмного модуля комп’ютеризованої системи безпровідної передачі інформації в промислових мережах
Затверджена наказом університету від 25.11.2024 р. №1239 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 21.01.2025р.
3. Вихідні дані до роботи:
3.1 Мова програмування Go;
3.2 Мова програмування Python;

4. Перелік питань, що потрібно опрацювати в роботі:
4.1 Вступ.
4.2 Аналіз сучасних засобів комунікації та промислових мереж.
4.3 Методи та інструменти дослідження.
4.4 Реалізація експерименту та аналіз результатів.
4.5 Охорона праці.
4.6 Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій: Графічний демонстраційний матеріал в форматі PowerPoint(*.ppt) формату А4 – 12 с. А4

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по-батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз сучасних засобів комунікації та промислових мереж	25.05.2024	виконано
2	Загальна концепція експериментів	12.06.2024	виконано
3	Критерії оцінювання	24.06.2024	виконано
4	Реалізація експерименту та аналіз результатів	20.08.2024	виконано
5	Опис стенду для виконання експерименту	16.10.2024	виконано
6	Архітектура програми та обрані точки для тестування	27.11.2024	виконано
7	Налаштування для тесту	25.12.2024	виконано
8	Проведення тесту та результати	05.01.2025	виконано

Дата видачі завдання 25.11.2024 р.

Студент ____________ Дяченко Е. С.
(підпис) (прізвище, ініціали)

Керівник роботи ____________ Стародубцев М. Г.
(підпис) (прізвище, ініціали)

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав та не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

«15» грудня 2024 р. Дяченко Е.С.

РЕФЕРАТ

Пояснювальна записка: 107 с., 1 табл., 38 рис., 2 дод., 26 джерел.

ПРОМИСЛОВІ БЕЗДРОТОВІ МЕРЕЖІ, ПЕРЕДАЧА ДАНИХ, СУЧАСНІ АЛГОРИТМИ СТИСНЕННЯ, СУЧАСНІ ФОРМАТИ ДАНИХ.

Мета роботи – розробити та оцінити підхід до передавання даних у промислових бездротових мережах, який поєднує ефективні алгоритми стиснення, високошвидкісні мережеві протоколи й оптимальні формати серіалізації.

Об’єкт дослідження – технології обміну інформацією в промисловому бездротовому середовищі.

Предмет дослідження – методи стиснення даних, протоколи мережевої взаємодії та формати пакування, що впливають на швидкість і надійність обміну повідомленнями.

У роботі проаналізовано властивості кількох алгоритмів стиснення (gzip, zstd, lz4 та ін.), а також особливості протоколів (HTTP/2, HTTP/3) та форматів серіалізації (JSON, Protocol Buffers, msgpack). Результати досліджень дають змогу визначити найефективніші комбінації з погляду продуктивності мережі і мінімізації витрат системних ресурсів, що зробить передачу повідомлень у промислових додатках швидшою та більш надійною. Також висвітлено аспекти впровадження таких рішень у контексті практичних обмежень та стандартів галузі.

ABSTRACT

Explanatory note: 107 p., 1 table, 38 figures, 2 appendix, 26 sources.

INDUSTRIAL WIRELESS NETWORKS, DATA TRANSFER, MODERN COMPRESSION ALGORITHMS, MODERN DATA FORMATS.

The purpose of the work is to develop and evaluate a data transmission approach for industrial wireless networks, combining efficient compression algorithms, high-speed network protocols, and optimal serialization formats.

The object of research is information exchange technologies in industrial wireless environments.

The subject of research is data compression methods, network interaction protocols, and packaging formats that affect the speed and reliability of message exchange.

This study analyzes several compression algorithms (gzip, zstd, lz4 та ін.), as well as the specifics of protocols (HTTP/2, HTTP/3) and serialization formats (JSON, Protocol Buffers, msgpack). The obtained results allow determining the most effective combinations in terms of network performance and minimal system resource consumption, thus making message transmission in industrial applications faster and more reliable. The work also covers implementation aspects of such solutions, considering practical constraints and industry standards.

ЗМІСТ

Перелік скорочень 9
Вступ 11
1 Аналіз сучасних засобів комунікації та промислових мереж 13
1.1 Актуальність та загальний огляд 13
1.2 Вплив особливостей промислового середовища на процес передачі даних 15
1.3 Основи та підходи до стиснення даних 20
1.4 Мережеві протоколи у промислових системах 28
1.5 Формати представлення й серіалізації даних 31
1.6 Майбутні виклики та потенційні напрями розвитку 39
1.7 Висновки до розділу 41
2 Методи та інструменти дослідження 43
2.1 Загальна концепція експериментів 43
2.2 Обґрунтування вибору інструментарію 44
2.3 Планування сценаріїв та критерії оцінювання 48
2.4 Обмеження та похибки 49
2.5 Висновки до другого розділу 51
3 Реалізація експериментів та аналіз результатів 52
3.1 Опис стенду для виконання експерименту 52
3.2 Архітектура програми та обрані точки для тестування 57
3.2.1 Обрані протоколи передачі даних 58
3.2.2 Обрані алгоритми стиснення даних 60
3.2.3 Обрані формати серіалізації даних 65
3.3 Ключова логіка та метрики 68
3.4 Налаштування для тесту 75
3.5 Проведення тесту та результати 76
3.5.1 Проведення тестування часу стиснення 77
3.5.2 Проведення тестування часу розтиснення 80
3.5.3 Проведення тестування часу ступенів стиснення 82

3.5.4 Проведення тестування використання процесору ... 84
3.5.5 Проведення тестування використання пам’яті ... 87
3.5.6 Проведення тестування запитів в секунду ... 90
3.5.7 Проведення тестування форматів даних ... 91
3.5.8 Підсумок результатів тестування ... 93
3.6 Висновки до третього розділу ... 95
4 Охорона праці ... 99
Висновки ... 102
Перелік джерел посилання ... 104
Додаток А Апробація наукових результатів дослідження ... 108
Додаток Б Демонстраційний матеріал у вигляді презентації ... 121

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення;

API – Application Programming Interface (інтерфейс прикладного програмування);

bzip2 – Burrows-Wheeler transform zip version 2 (архівування bzip2 на основі перетворення Burrows-Wheeler, версія 2);

CBOR – Concise Binary Object Representation (стисле представлення бінарних об'єктів);

CPU – Central Processing Unit (центральний процесор);

DEFLATE – Deflate algorithm (алгоритм Deflate);

gRPC – gRPC Remote Procedure Calls (gRPC віддалені виклики процедур);

gzip – GNU zip (архівування GNU zip);

HTTP – Hypertext Transfer Protocol (протокол передачі гіпертексту);

IIoT – Industrial Internet of Things (промисловий інтернет речей);

IoT – Internet of Things (інтернет речей);

JSON – JavaScript Object Notation (нотація об'єктів JavaScript);

lz4 – LZ4 algorithm (алгоритм LZ4);

LZ77/LZ78 – Lempel-Ziv 77 and Lempel-Ziv 78 algorithms (алгоритми Лемпеля–Зіва 77 та Лемпеля–Зіва 78);

MIMO – Multiple-Input Multiple-Output (багато входів, багато виходів);

NAT – Network Address Translation (трансляція мережевих адрес);

QoS – Quality of Service (якість обслуговування);

RAM – Random Access Memory (оперативна пам'ять);

REST – Representational State Transfer (передача стану представлення);

RLE – Run-Length Encoding (кодування довжини серії);

SSD – Solid State Drive (твердотільний накопичувач);

TCP/IP – Transmission Control Protocol/Internet Protocol (протокол керування передачею/міжмережевий протокол);

TLS – Transport Layer Security (безпека транспортного рівня);

UDP – User Datagram Protocol (протокол дейтаграм користувача);

VPN – Virtual Private Network (віртуальна приватна мережа);

XML – Extensible Markup Language (розширювана мова розмітки);

zstd – Zstandard.

ВСТУП

Сучасні промислові бездротові мережі та методи передавання даних дедалі частіше використовуються в багатьох галузях. Актуальність цього напряму пояснюється зростанням вимог до масштабованості, швидкодії та надійності обміну інформацією, а також прагненням оптимізувати витрати ресурсів і ширше впроваджувати відкриті стандарти. Молоді виробництва та великі підприємства сьогодні все частіше шукають підходи, що поєднують у собі економне використання пропускної здатності, стійкість до перешкод і гнучкість налаштувань, аби відповідати інтенсивним темпам розвитку та задовольняти потреби в оперативній передачі даних.

Однією з найважливіших проблем на цьому шляху залишається пошук збалансованого варіанту між високою швидкістю й надійністю мережевих з’єднань та мінімізацією обчислювальних і матеріальних витрат. Сучасні алгоритми стиснення, такі як zstd, високоефективні протоколи на кшталт HTTP/3 і формати серіалізації на зразок msgpack потенційно здатні істотно покращити роботу мережі, але потребують комплексного дослідження або практичного тестування, аби виявити найоптимальніші комбінації для конкретних сценаріїв.

Мета роботи полягає у розробленні та оцінюванні оновленого підходу до передачі даних у промислових бездротових мережах, що має поєднати сучасні алгоритми стиснення, високоефективні мережеві протоколи та оптимальні формати серіалізації.

Об’єктом дослідження є технології передавання повідомлень у бездротових промислових середовищах.

Предметом дослідження є методи стиснення та формати пакування даних, а також адаптація протоколів для підвищення продуктивності на практиці.

Актуальність дослідження узгоджується з Ціллю сталого розвитку 9 «Промисловість, інновації та інфраструктура», зокрема із завданням 9.4 щодо сприяння прискореному розвитку високо- та середньовисокотехнологічних

секторів переробної промисловості. Розроблення оновленого підходу до передачі даних у промислових бездротових мережах відповідає індикатору 9.4.1. Запропоновані рішення щодо оптимізації мережевих протоколів та алгоритмів стиснення даних спрямовані на підвищення ефективності виробничих процесів у секторі електронної та комп'ютерної продукції, що безпосередньо впливає на показники доданої вартості високотехнологічних підприємств.

Робота виконана згідно [1-3].

1 АНАЛІЗ СУЧАСНИХ ЗАСОБІВ КОМУНІКАЦІЇ ТА ПРОМИСЛОВИХ МЕРЕЖ

1.1 Актуальність та загальний огляд

У сучасних промислових середовищах спостерігається дедалі ширше впровадження бездротових мереж для обміну інформацією між сенсорами, контролерами та серверами збору даних. Це дає змогу оперативно отримувати показники про стан виробничих процесів і вчасно реагувати на зміни, що відбуваються у технологічному циклі. Однак зі збільшенням числа одночасних підключень і зростанням обсягу передавання виникає проблема перевантаження мережі, яка може спричинити затримки або навіть збої у роботі системи.

Стиснення даних є одним із найдієвіших способів скоротити мережеве навантаження. Воно допомагає зменшити розмір пакетів без втрати ключової інформації та забезпечити ефективніше використання пропускної здатності каналу. На практиці застосовують різні алгоритми, серед яких найбільш розповсюджені gzip, lz4, bzip2 і zstd. Вибір конкретного підходу залежить від потреб щодо швидкості стискання, рівня компресії та доступних обчислювальних ресурсів, оскільки надто складні алгоритми можуть збільшити навантаження на процесор і спричинити небажані затримки.

На якість передачі даних також впливає обраний мережевий протокол. Якщо використовувати традиційний HTTP/1.1 або інші застарілі підходи, то постійне встановлення з’єднань чи неможливість мультиплексувати запити можуть сповільнювати обмін. Водночас сучасні рішення на кшталт HTTP/2, HTTP/3 або gRPC ефективно керують потоками, дають змогу відправляти кілька запитів у межах одного з’єднання та знижують затримки на рівні додатків. Це особливо важливо, коли йдеться про великі обсяги даних чи близькі до реального часу вимоги до продуктивності.

Окремим аспектом виступає формат представлення даних, що передаються. Текстові формати, як JSON або XML, є доволі наочними та універсальними, але можуть надмірно збільшувати обсяг переданої інформації. Бінарні ж формати, до прикладу Protocol Buffers або msgpack, досягають суттєвої економії трафіку за рахунок компактнішої структури, але водночас ускладнюють налагодження та прочитання даних «на льоту». Вибір між зручністю та ефективністю зазвичай визначається специфікою задачі, доступними інструментами та вимогами до продуктивності [4].

На рисунку 1.1 представлена блок-схема мережевої передачі даних у IoT зі стисненням даних.

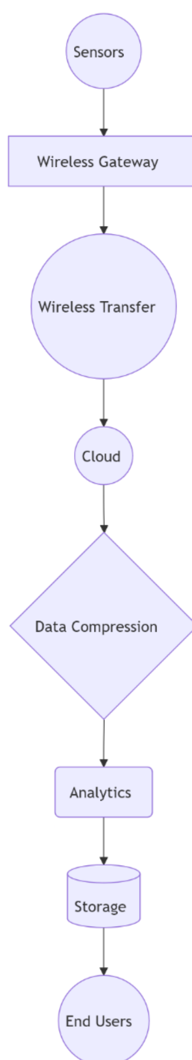

Рисунок 1.1 – Блок–схема мережевої передачі даних у IoT зі стисненням даних

З огляду на зазначене, актуальність дослідження способів модернізації програмного модуля бездротової передачі інформації в промислових мережах викликана потребою у мінімізації затримок, раціональному використанні пропускної здатності та підвищенні надійності передачі. Результатом таких удосконалень є більш гнучка та масштабована інфраструктура, що краще відповідає сучасним викликам індустрії та дозволяє впроваджувати нові технології, не знижуючи ефективності виробництва.

1.2 Вплив особливостей промислового середовища на процес передачі даних

Промислові бездротові мережі, на відміну від звичайних споживчих або корпоративних середовищ, характеризуються суворішими вимогами щодо надійності, часових параметрів та безпеки. Окрім базової передачі даних, критичним є забезпечення стабільного функціонування в умовах потужних електромагнітних впливів, зокрема від великих двигунів чи зварювального обладнання. Ці фактори зумовлюють підвищений рівень вібрацій, електромагнітного шуму та перепадів напруги, що можуть провокувати затримки або навіть втрату пакетів. Тому стиснення даних, яке в теорії дозволяє зменшити кількість переданої інформації та ймовірність колізій, мусить оцінюватися з урахуванням практичних обмежень. Зокрема, потрібно розуміти, як збільшується затримка на стадії компресії й декомпресії, та як це відображається на загальному QoS (Quality of Service).

Важливою відмінністю промислових систем є вимога до жорсткого контрольованого часу відгуку. Чимало технологічних процесів реалізовано за принципом жорсткого або м’якого реального часу. Для «жорсткого» реального часу небажаними є навіть несуттєві відхилення, оскільки вони здатні спричинити відмову системи або небезпечні наслідки для обладнання та персоналу. Зокрема, у випадках з керуванням роботизованими механізмами чи оперативним регулюванням температури в хімічних реакторах, затримка понад припустимі

межі порушує стабільність усього циклу. У таких сценаріях застосування навіть найбільш ефективних алгоритмів стиснення може не виправдати себе, якщо вони збільшать латентність понад критичну межу. І навпаки, якщо дані надходять у переважно асинхронному режимі (наприклад, хмарна аналітика, звітування чи збирання історичних логів), тривалість обходження циклу затримки є другорядною, і тоді виграш від потужних методів компресії перевищує недоліки.

Розглядаючи архітектуру промислових бездротових систем, слід брати до уваги наявність проміжних шлюзів чи контролерів. Часто саме в цих вузлах відбувається консолідація даних зі значної кількості сенсорів і їхнє попереднє опрацювання, тобто фільтрація, агрегація чи навіть елементи штучного інтелекту для виявлення аномалій. Тут постає запитання розподілу функцій: чи доцільно виконувати стиснення вже на рівні сенсора (що потребує здебільшого низькопотужного мікроконтролера та обмежених ресурсів), чи краще делегувати його контролеру з вищою обчислювальною потужністю? З одного боку, стиснувши дані на самій «периферії», можна серйозно заощадити пропускну здатність бездротового каналу. З іншого боку, інтелектуальні шлюзи можуть застосовувати збалансованіші та складніші методи компресії, що дасть кращий коефіцієнт стискання. Утім, потрібно пам’ятати і про часові вимоги: потужне стиснення може бути надто «важким» для мікроконтролера й суттєво розтягнути таймінги на сенсорному боці.

Варто також зважати на особливості промислових протоколів вищого рівня, таких як OPC UA, Profinet, Modbus TCP чи EtherNet/IP. Кожен із них має свої специфікації щодо формування пакетів, механізмів підтвердження (acknowledgement), шифрування тощо. У деяких протоколах передбачено вбудовані засоби фрагментації та повторних трансляцій для втрачених пакетів, тоді як інші покладаються на функції базового транспортного рівня (TCP, UDP) [5]. Коли додаються механізми стиснення, вони можуть поділити великий обсяг даних на кілька «компресованих» блоків, що впливатиме на ефективність відновлення інформації за наявності втрат. Наприклад, якщо пакет, що містить значний фрагмент стисненого потоку, загубиться чи буде пошкоджений, то

повторної передачі може потребувати весь блок - а це додатково зростає латентність. Тому в реальних системах часто застосовують гібридний підхід: поєднують легкі методи стиснення з перевіреними механізмами корекції помилок і сегментації даних.

Безпека є теж важливою складовою, оскільки чимало промислових процесів мають критично важливі показники, від втручання в які залежить не лише економічна вигода, а й життя людей. Багато способів шифрування, зокрема TLS або VPN-рішення, роблять потік стійким до стиснення, адже зашифровані дані втратили очевидні патерни та не піддаються ефективній компресії. З цієї причини часто рекомендують виконувати стиснення перед шифруванням, а якщо інфраструктура передбачає обов’язкове наскрізне шифрування (end-to-end), то доводиться узгоджувати, які саме дані можна стискати на проміжних вузлах без порушення політики безпеки. Такий компроміс між конфіденційністю та ефективністю передачі породжує додаткові вимоги до протоколів: наприклад, протоколи рівня додатків (HTTP/2, gRPC) підтримують попередньо налаштований компресор перед шифруванням TLS [6], що дозволяє знизити обсяг трафіку, не розкриваючи вміст повідомлень у відкритому вигляді.

Таким чином, стиснення даних у промислових бездротових мережах не можна розглядати як одну ізольовану операцію. Це складова глобального завдання підвищення надійності та продуктивності систем, що водночас взаємодіє із численними внутрішніми (архітектура мережі, протоколи, метрики часу) та зовнішніми (умови радіооточення, стандарти безпеки, вимоги до резервування) факторами. Технології бездротового обміну в промисловості швидко розвиваються, зокрема завдяки появі рішень на базі 5G, Wi-Fi 6/6E та спеціалізованих протоколів типу WirelessHART чи ISA100.11a, але питання оптимального стиснення й досі залишається багатокомпонентним компромісом між швидкістю, надійністю, енергозатратами й масштабованістю. Тільки ретельний аналіз усіх цих чинників дозволяє інтегрувати методи компресії та ефективні протоколи в готові промислові рішення без ризику «перевантажити» або «зламати» існуючу критичну інфраструктуру.

У контексті бездротових промислових мереж слід також звертати увагу на масштабування та багатопоточність, адже сучасні підприємства нерідко впроваджують розподілені платформи керування та моніторингу. Якщо в невеликій системі із кількома сенсорами додаткові затримки, спричинені стисненням чи складними протоколами, ще можна компенсувати адаптацією робочого циклу, то в розгалужених мережах із сотнями або навіть тисячами підключених пристроїв така стратегія стає неефективною. У багатьох випадках застосовуються хмарні рішення, де величезні обсяги інформації акумулюються на віддалених серверах для подальшої аналітики, машинного навчання або прогнозування. Тоді питома вартість обчислень на рівні стиснення може зростати при значному збільшенні числа паралельних потоків, що потребує ретельно збалансованої архітектури серверних ресурсів і підходів до організації мережевого трафіку.

Доповнює загальну картину ще й питання управління чергами та перезавантаженням вузлів. У промислових застосунках під час пікових навантажень (наприклад, коли одночасно надходить великий масив даних від усіх сенсорів після технологічного «перезапуску» або критичної події) мережеві вузли та сервери можуть тимчасово не встигати обробляти та відправляти повідомлення з максимальною швидкістю. У такому разі компресія, яка іноді допомагає стримати потоки даних, навпаки, здатна стати додатковою перешкодою, збільшуючи час очікування в чергах через обчислювальне навантаження. Відповідно, ефективне застосування стиснення має бути пов’язане з розумним плануванням обробки, можливо, навіть із виділенням окремих процесорних ядер чи потоків для виконання компресійних процедур у високонавантажених системах.

У деяких промислових сценаріях критерієм оцінки якості передачі стає не стільки загальна пропускна здатність чи середня затримка, скільки гарантований час проходження пакета. Такі системи проектуються за принципами критичного або гарантованого обслуговування, де існує режим пріоритетних повідомлень. Наприклад, пакети з оповіщенням про аварію мають транслюватися в мережі з

максимально можливою швидкістю, тоді як менш критичний трафік (звіти про загальний стан із сенсорів, журнали процесів) може передаватися з використанням глибших рівнів стиснення. Реалізація подібної стратегії вимагає адаптивних алгоритмів, здатних динамічно вмикати або вимикати компресію (а також обирати відповідний протокол) залежно від рівня пріоритету даних та поточних умових функціонування мережі.

На рисунку 1.2 представлена загальна ідея експерименту та передачі даних у бездротовій мережі, яка пов’язує всі впливаючі фактори.

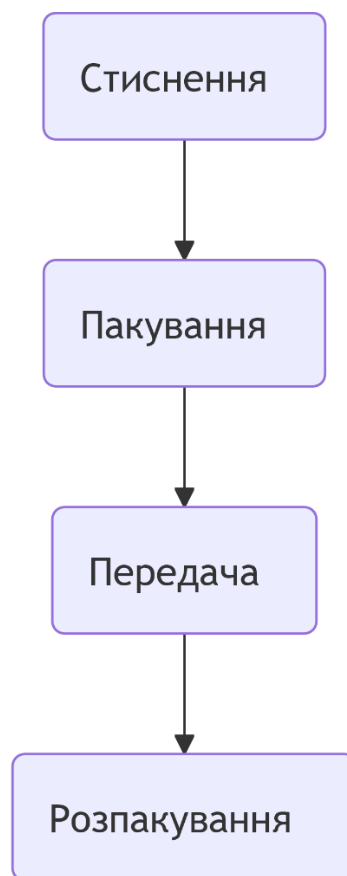

Рисунок 1.2 – Ідея експерименту та передача даних у бездротовій мережі

З огляду на усе вищезазначене, процес вибору оптимальної схеми передачі даних у промислових бездротових мережах нагадує «багаторівневу оптимізацію», що охоплює апаратну частину, транспортний рівень, алгоритми стиснення та структуру даних. Істотним у цьому процесі є гнучка адаптація до

динамічних умов середовища: рівень перешкод, стрибки навантаження, зміни сценаріїв використання. Грамотне поєднання протоколів разом із належним вибором компресії і способу пакування формує підґрунтя для створення сучасних та ефективних промислових мереж. Належний консенсус між затраченими обчислювальними ресурсами й економією мережевого каналу дозволяє досягнути найвищої продуктивності за мінімальних додаткових витрат, що є кінцевою метою процесу модернізації.

1.3 Основи та підходи до стиснення даних

Загалом усі сучасні методи стиснення ґрунтуються на ідеї виявлення вхідного обсягу надлишкової чи повторюваної інформації та трансформації її у більш компактне представлення. Теоретичну базу цього процесу заклав Клод Шеннон у своїх фундаментальних роботах із теорії інформації, де було введено поняття ентропії як міри випадковості (чи невпорядкованості) даних. Якщо дані мають низьку ентропію, тобто містять значні повторення, вони краще піддаються стисненню. У випадку промислових мереж це часто означає, що telemetric payload (потоки телеметрії) можуть мати велике число ідентичних або мало змінюваних полів (наприклад, зчитування температури чи тиску з незначною динамікою), що є сприятливим для алгоритмів компресії.

Кожен алгоритм стиснення має дві основні цілі: зменшити розмір даних і забезпечити достатню швидкість кодування та декодування. При цьому завжди доводиться жертвувати чимось: чим щільніший ступінь стиснення, тим зазвичай повільнішим стає процес. Деякі класичні алгоритми зосереджені на статистичних підходах (наприклад, кодування Хаффмана або арифметичне кодування), які застосовують імовірнісні моделі частоти появи символів чи бітових патернів. І хоча вони можуть бути досить ефективними, у промислових застосунках часто віддають перевагу «словниковим» підходам (наприклад, сімейство LZ77/LZ78, до якого належать gzip, lz4, zstd та інші) через їхню кращу продуктивність на типовому обладнанні [7]. Словникові алгоритми генерують

таблицю (словник) повторюваних послідовностей, що трапляються в потоці, й замінюють їх на коротші посилання, досягаючи при цьому значного зменшення обсягу у випадку, коли вхідні дані містять одні й ті самі фрагменти кілька разів.

На рисунку 1.3 для прикладу наведено життя запиту в протоколі HTTP [8].

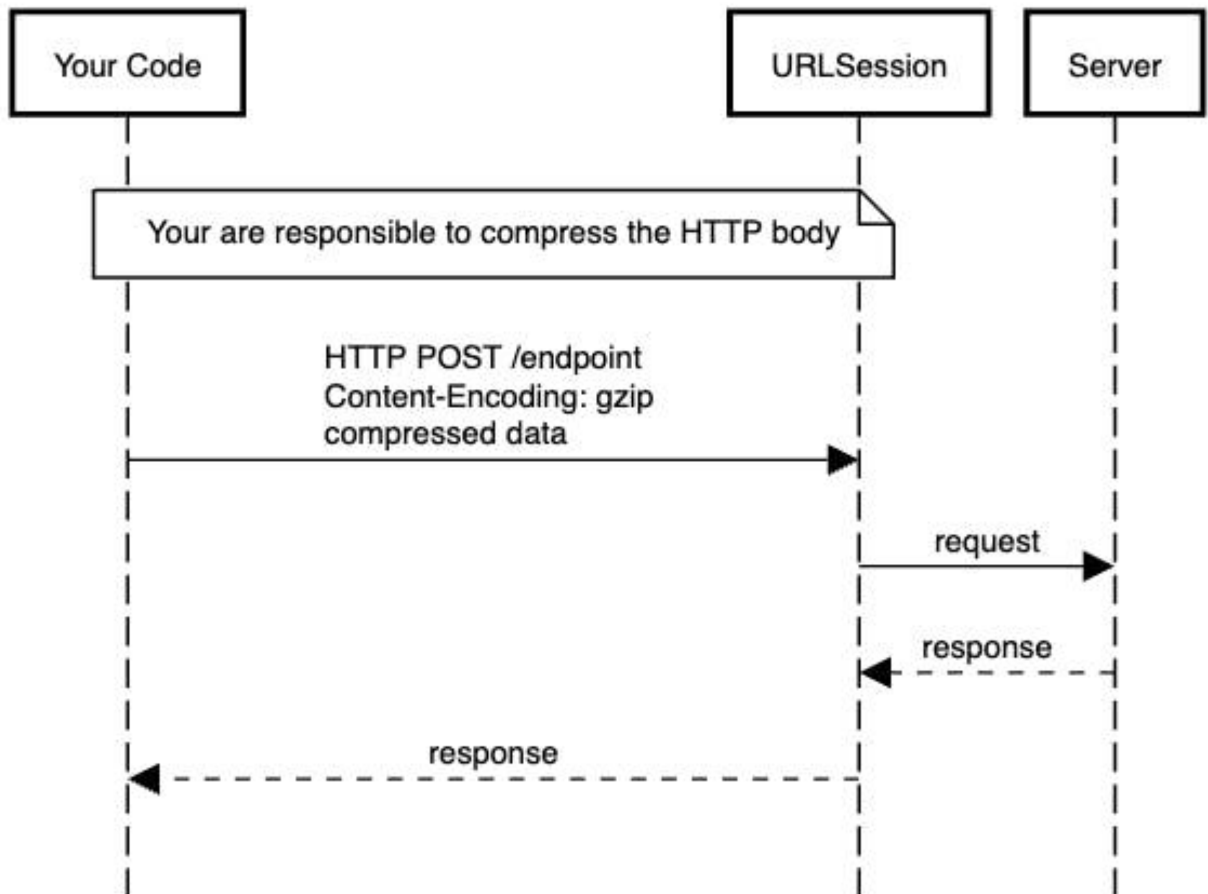

Рисунок 1.3 – Запит в протоколі HTTP

Щоб краще зрозуміти відмінності між алгоритмами стиснення, слід розглянути, приміром, gzip (заснований на DEFLATE, похідному від LZ77) і zstd, розроблений Facebook із фокусом на високій швидкості та гнучкості налаштувань ступеня стиснення [9]. GZIP широко використовується в мережевих протоколах (зокрема HTTP) завдяки своїй простоті та вже усталеній підтримці в багатьох середовищах [10]. Проте його доцільність у промисловому секторі може бути обмеженою, якщо потрібно максимізувати швидкодію або досягати високих коефіцієнтів стиснення. Натомість zstd пропонує широку

шкалу налаштувань від «дуже швидкого, але з невеликим коефіцієнтом» до «повільнішого, але із суттєвим ущільненням даних». У випадку передавання великих обсягів телеметричної інформації zstd здатний краще відповідати потребі економії пропускної здатності й підвищеної швидкодії за належної апаратної підтримки.

В індустріальних умовах важливо, що багато алгоритмів стиснення дають найбільшу вигоду за умови, коли вони мають справу з відносно великими блоками даних, де можна знайти повторення на доволі довгих відрізках. Однак у реальних сенсорних системах дані нерідко надходять крихітними порціями (наприклад, пакет у кілька десятків або сотень байтів) з високою частотою. Тут надто потужні алгоритми можуть «не встигати» ефективно розкрити свої переваги, а підсумкова затримка на стиск/декомпресію робить процес недоцільним. У таких сценаріях можливе використання більш швидких, хоч і менш «агресивних» рішень. Наприклад, lz4, що спеціалізується на високій швидкості стиснення з помірним коефіцієнтом скорочення даних. Ба більше, на проміжних вузлах (шлюзах) можна збирати невеликі пакети від кількох сенсорів у більший буфер, а потім стискати його цілком, досягаючи таким чином більшої ефективності компресії [11].

Окрім чисто словникових або статистичних методів, у промислових середовищах рідко, але все ж зустрічаються дослідницькі спроби застосувати «розумні» методи (наприклад, нейронні стиснення чи спеціалізовані алгоритми на основі контекстного моделювання). Хоча вони більш продуктивні для деяких складних типів даних (зображення високої роздільної здатності, звуки, нетривіальна структурована інформація), їх рідко залучають для потокової телеметрії через високе обчислювальне навантаження та недоцільність у швидкісних промислових процесах. Водночас, імовірно, з розвитком промислового інтернету речей (IIoT) та інтегрованих модулів машинного навчання можна очікувати зростання інтересу до більш адаптивних, самооптимізувальних алгоритмів із використанням машинного інтелекту навіть на рівні стиснення.

На рисунку 1.4 зображений графік зменшення швидкості стиснення при підвищенні рівня стиснення для алгоритмів zstd, lzma, zlib, brotli.

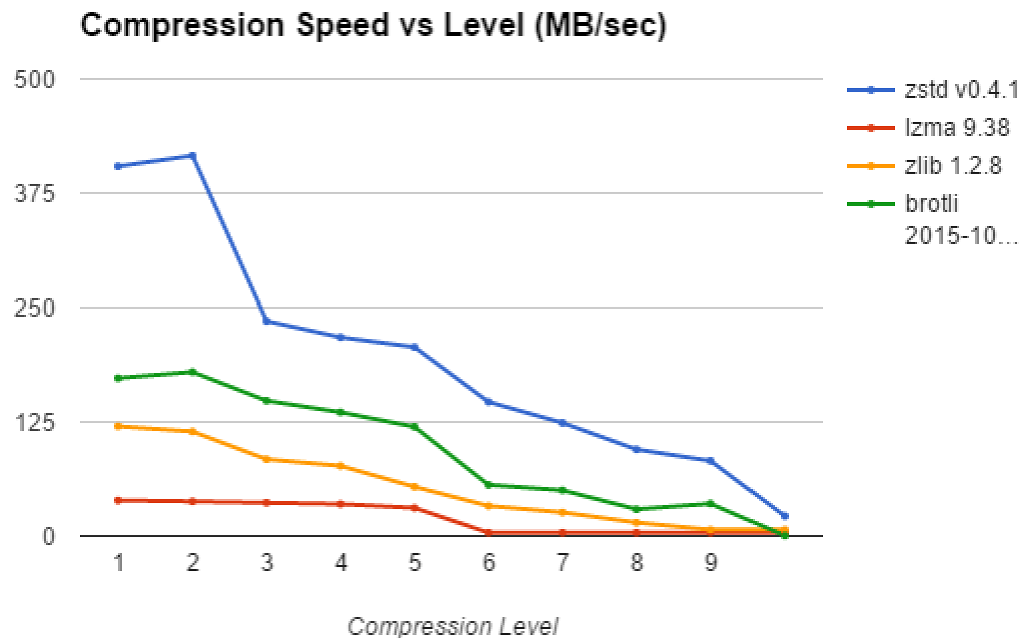

Рисунок 1.4 – Графік зменшення швидкості стиснення при підвищенні рівня стиснення

Важливо усвідомлювати, що принципове розрізнення між безвтратним (lossless) та з втратами (lossy) стисканням набуває особливого значення в промисловому контексті. У більшості випадків для технологічних процесів критично зберегти точність вимірювань, оскільки навіть незначна втрата даних може призвести до хибних висновків системи керування. Тому найчастіше використовуються безвтратні алгоритми, такі як: gzip, zstd, lz4 тощо. Однак існують окремі сфери, де припустиме невелике наближення, наприклад під час передавання деяких відеопотоків для нагляду за виробничими лініями, коли компресія з втратами (наприклад, H.264, H.265) дозволяє суттєво зменшити трафік, а незначне зниження чіткості не позначається на здатності оператора виявляти збої [12].

Окрім промислових застосувань, апаратне прискорення алгоритмів кодування та декодування (H.264, H.265, AV1 тощо) широко використовується і

в консьюмерських рішеннях. Зокрема, сучасні смартфони зазвичай мають спеціалізовані чипи або ядра, призначені для обробки відео та зображень [13]. Це дозволяє ефективно розвантажувати центральний процесор під час відтворення й запису контенту у високій роздільній здатності, а також економити енергію акумулятора без погіршення якості зображення чи швидкодії додатків.

Відеокарти та багатофункціональні графічні процесори (GPU) у персональних комп’ютерах теж мають апаратну підтримку кодування і декодування сучасних відеостандартів. Це особливо актуально для геймінгу, стрімінгу, відеоконференцій чи професійної роботи з мультимедійними файлами, адже вивільняються ресурси центрального процесора, а прискорені графічні модулі можуть впоратися з великими обсягами даних швидше та ефективніше.

На рисунку 1.5 відображене порівняння розміру відеофайла в залежності від обраного алгоритму (H.264, H.265).

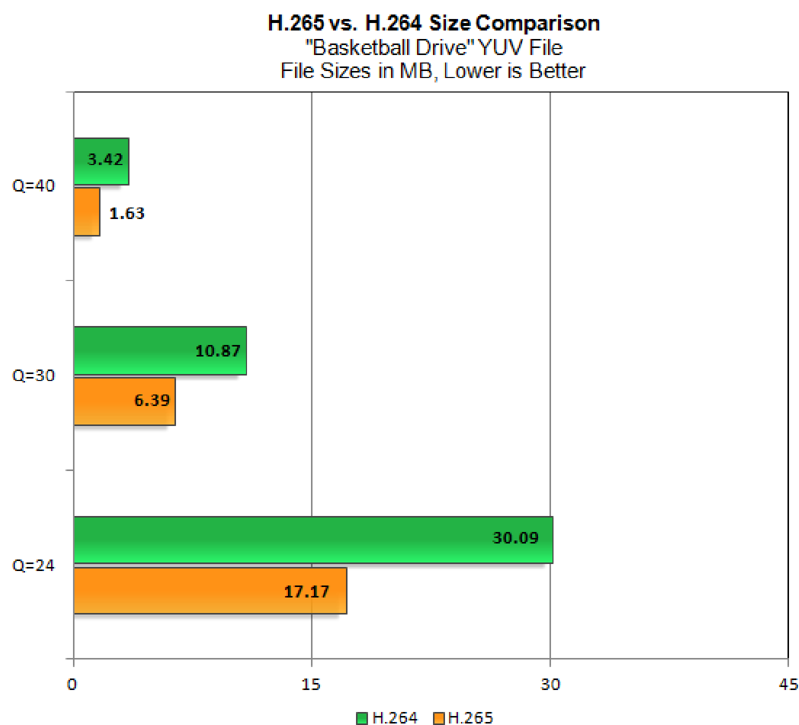

Рисунок 1.5 – Порівняння розміру відеофайла в залежності від обраного алгоритму (H.264, H.265)

Для глибшого розуміння суті процесу безвтратного стиснення слід розглянути центральні концепції кодування. Хаффман-кодування ґрунтується на побудові бінарного дерева, де найбільш імовірні символи отримують найкоротший код, а найменш поширені – довший. У контексті промислових мереж застосовувати «чисте» Хаффман-кодування до сирих потоків даних досить рідко, проте воно використовується як підмножина більш розвинених методик (наприклад, алгоритм DEFLATE, що лежить в основі gzip) [14]. Натомість арифметичне кодування працює з імовірнісним проміжком, послідовно його уточнюючи, що за певних умов дозволяє досягати кращої ефективності, але й навантажує систему складнішими обчисленнями. Для потокових промислових даних із швидкозмінними патернами його доцільність обмежена, і тому у практиці частіше зустрічаються LZ-алгоритми та їх модифікації.

Не менш важливим є поняття адаптивного стиснення, коли потік аналізується «на льоту» й модель (словник, статистика) оновлюється відповідно до нових порцій даних. Це актуально для промислових застосунків із динамічно змінною природою трафіку: скажімо, один сенсор може довго надсилати приблизно незмінні значення, після чого різко переходить до іншого діапазону. Якщо алгоритм здатний швидко адаптуватися, коефіцієнт стискання залишатиметься високим. Утім, адаптивність потребує додаткового управління внутрішніми структурами, що збільшує складність реалізації й ризик «зайвих» обчислень. З цього погляду, вибір між статичним і динамічним варіантами словника стає нетривіальним: у деяких сценаріях ефективніше мати наперед ініціалізований словник для типових форматів телеметрії, а в інших - покладатися на постійну побудову й актуалізацію внутрішньої моделі.

Висока варіативність промислового середовища означає, що заглиблене моделювання чи спеціалізовані алгоритми варто впроваджувати там, де очікувана вигода перевищує витрати на реалізацію та підтримку. У проєктах, які обслуговують сотні чи тисячі сенсорів, кожен із яких дає схожі за структурою дані, досить легко досягти значного виграшу від базового словникового підходу.

Однак якщо маємо справу зі складними об'єктами типу потоків зображень або різнорідної інформації (лог-файли, бінарні журнали налаштувань), краще застосовувати або різні профілі стиснення для кожного типу, або більш гнучкі, але складні архітектури. Практичний прикладом може бути ситуація коли дані телеметрії стискаються за допомогою швидкої lz4, тоді як фрагменти відеоспостереження надсилаються із кращим ступенем економії за рахунок H.265. Тобто, питання компресії слід розглядати в комплексі з особливостями трафіку, а не як універсальне рішення.

Іншим важливим аспектом безвтратних методів стиснення є механізми буферизації та управління потоком даних. Алгоритми типу gzip, zstd або lz4 зазвичай працюють зі «шматками» (chunks) або блоками, що послідовно обробляються компресором. Кількість та розмір цих блоків можуть суттєво впливати на продуктивність. Наприклад, збільшення розміру блоку часто дає змогу алгоритму «помітити» більше повторюваних патернів для зменшення обсягу даних, проте збільшує затримку, адже дані «очікують» у проміжній пам'яті, поки розмір не досягне певного порогу. У промислових застосунках такий підхід не завжди прийнятний, особливо коли система повинна реагувати на зміни «на льоту». Тому іноді доводиться обирати менший розмір блоку на догоду швидкості відгуку, розуміючи, що загальний коефіцієнт стискання знизиться.

Додаткові тонкощі з'являються, коли йдеться про поточне (streaming) стиснення, де дані надходять безперервним потоком, а компресор повинен видавати результат у міру отримання чергових частин. Таке рішення критичне для великомасштабних додатків реального часу, наприклад, коли сенсори щохвилини передають тисячі нових значень, і кожна затримка в обробці може спричинити небажане наростання черг. У цьому разі знову ж таки постає питання компромісу: чи варто «дочікуватися» повного блоку, ризикуючи збільшити затримку, чи ж краще негайно стискати незначні партії даних, отримавши мінімальну, але швидку компресію.

Ще один напрямок розвитку, що набуває популярності в промисловому IoT це апаратне прискорення стиснення. Спеціалізовані прискорювачі (hardware accelerators) або вбудовані криптографічні модулі нерідко можуть виконувати певні операції стиснення значно швидше та з меншою витратою енергії, ніж звичайні центральні процесори. Це особливо актуально для пристроїв із обмеженими ресурсами, зокрема сенсорних вузлів чи шлюзів, які мають зберігати низьке енергоспоживання. Утім, інтеграція таких прискорювачів вимагає узгодження на рівні драйверів, прошивки та підтримки в містках (middleware), що ускладнює розробку і підвищує витрати на створення або оновлення обладнання. Якщо ж проект має достатній бюджет, таке рішення може надати значну перевагу: дозволити використовувати більш «глибокі» алгоритми з високим ступенем стискання, не сповільнюючи при цьому систему реального часу.

Значний інтерес викликає й питання багатостадійного стиснення, коли на початковому етапі застосовується «легка» компресія, а при отриманні чи обробці даних центральним сервером (чи в хмарі) ці дані, за потреби, можуть додатково стискатися іншим алгоритмом. Аналогічно, існують сценарії з попереднім відфільтровуванням даних. Приміром, якщо певні сенсори генерують «шум» або дубльовану інформацію, то варто перед передаванням застосувати базову фільтрацію й лише потім стискати результат. Такий підхід можливий, якщо контролер або шлюз виконує первинну обробку та може виявляти редундантні записи чи усереднювати їх перед передаванням. Хоча це не зовсім називається «стисненням» у прямому сенсі, однак часто є більш ефективним методом з конкретно-прикладного погляду, бо відразу позбавляється вторинних чи знецінених даних.

Нарешті, при виборі алгоритму стиснення слід брати до уваги й потреби у розпакуванні на іншому кінці. У промисловій мережі часто є багато «приймачів» даних, від резервних систем зберігання до автоматизованих додатків, кожен із яких повинен своєчасно й коректно розбирати вміст повідомлень. Якщо формат із занадто складною декомпресією уповільнює аналіз даних у вузлі обробки або

вносить елементи нестабільності (наприклад, через велику кількість проміжних буферів), загальний виграш може виявитися сумнівним. Ба більше, декомпресія повинна бути стійкою до помилок: якщо у випадку втрати чи пошкодження одного пакета з усієї сесії розпаковується вся решта даних, чи процес зупиниться? У деяких алгоритмах пропуск одного фрагмента не дає змоги декодувати подальший потік, тому треба вибудовувати механізми відновлення або повторної передачі, сумісні з протоколом, що використовується.

Отже, нинішній стан технологій стиснення даних у поєднанні з різними протоколами та форматами передачі пропонує широкий спектр можливостей для підвищення ефективності промислових бездротових мереж. Вибір конкретного рішення залежить від цілей проекту, апаратних обмежень, вимог до затримки й безпеки, а також від того, який тип даних переважає в трафіку. У подальших розділах роботи буде розглянуто конкретні приклади реалізації, а також наведено результати дослідних вимірювань, що дозволять встановити найбільш збалансований варіант для типових промислових сценаріїв.

1.4 Мережеві протоколи у промислових системах

Важливою складовою успішної передачі даних у промислових мережах виступає вибір та налаштування мережевого протоколу. Незважаючи на традиційну поширеність TCP/IP як універсальної транспортної основи, у високонавантажених та часовочутливих середовищах часто виникають вимоги, що перевищують можливості стандартних конфігурацій. Такі фактори, як багатопоточність, оптимізація таймерів підтвердження (acknowledgment), механізми повторних передач (retransmissions) та стабільність при втраті пакетів, відіграють ключову роль у досягненні бажаного рівня надійності й продуктивності. Для прикладу, у корпоративному сегменті перенавантаження трафіку може призвести лише до зниження швидкості завантаження, тоді як у промисловому середовищі критичні пакети, що не прийшли вчасно, здатні спричинити відмову в системі керування або навіть техногенну аварію.

Оскільки традиційний TCP має тенденцію до «предбачуваної» поведінки з огляду на контроль потоку й роботи з втраченими пакетами, деякі промислові застосунки звертаються до UDP для мінімізації затримки [5]. UDP, будучи позбавленим механізмів гарантування доставки, при вмілому налаштуванні може забезпечити кращу латентність, проте водночас потребує додаткового логічного рівня для оброблення потенційних втрат і відновлення [6]. У промислових протоколах реального часу (такі як EtherCAT, Profinet RT чи Ethernet/IP) зазвичай поєднують характеристики обох підходів, реалізовуючи власний «тонкий» транспорт, котрий оптимізований під низький рівень втрат або застосовує чітку пріоритизацію та контроль над часом обробки кадрів.

Для додатків вищого рівня останнім часом привабливими стають HTTP/2 та HTTP/3. HTTP/2 пропонує мультиплексування. Мультиплексування – це здатність одночасно надсилати кілька потоків у межах одного з'єднання, що суттєво скорочує затримки на етапі встановлення сеансу. Його бінарний формат специфікації заголовків дозволяє точніше оптимізувати надлишкові дані, а вбудований mechanism пріоритизації потоків дає змогу розподіляти ресурси між критичними запитами та менш важливими [15]. HTTP/3, зі свого боку, базується на протоколі QUIC і змінює парадигму, зокрема позбавиваючись багатьох проблем TCP, пов'язаних із блокуванням згідно з правилом «head-of-line». QUIC реалізує надійність на рівні користувацького простору, обходячи можливі затримки в ядрі операційної системи, а також дозволяє швидше відновлюватися після втрачених пакетів [16].

gRPC доповнює цю картину як популярна фреймворк-рішення для побудови розподілених систем. Замість типової взаємодії запит/відповідь, що характерна для HTTP, gRPC підтримує потокові виклики, де дані можуть надходити у режимі real-time або напівреального часу в обидва боки «клієнт-сервер». Завдяки внутрішньому використанню HTTP/2 (а планується й повноцінна інтеграція HTTP/3) та бінарного формату серіалізації (Protocol Buffers) gRPC демонструє вищу ефективність порівняно з текстовими підходами на базі REST + JSON. У промисловому середовищі такий підхід може бути

привабливим для швидкого й надійного обміну з великим числом закритих вузлів або модулів, де необхідна прозора передача складних структур даних при одночасному збереженні низької латентності [17].

На рисунку 1.6 зображене порівняння затримки між HTTP/1, HTTP/2, HTTP/3 (QUIC).

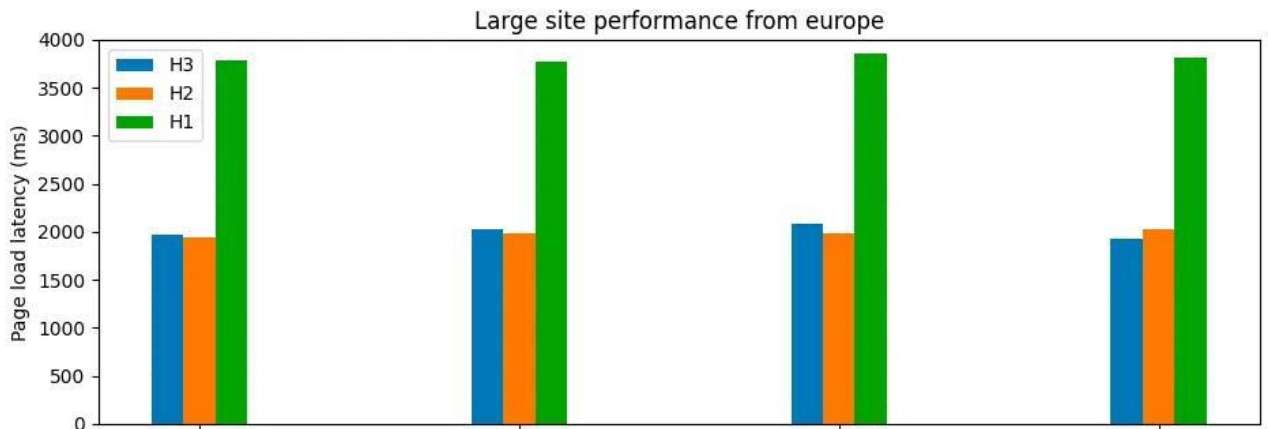

Рисунок 1.6 – Порівняння затримки між HTTP/1, HTTP/2, HTTP/3 (QUIC)

У реальних проектах вибір мережевого протоколу додатково визначається особливостями взаємодії з «бізнес-логікою» та вже існуючою технічною інфраструктурою. Якщо виробничий комплекс тривалий час експлуатувала систему диспетчеризації на основі Modbus TCP, можлива потреба адаптуватися до встановлених пристроїв, які не здатні мігрувати на більш сучасні рішення. У такому разі інтегратору доводиться або розвивати власні механізми стискання всередині протоколу, або застосовувати проміжні шлюзи-проксі з можливістю конвертації поетапно. Натомість у нових проєктах, де специфікації дозволяють впровадити, скажімо, HTTP/3 чи gRPC із початку, архітектори мають ширшу свободу побудови структури передачі даних і можуть планувати більш системну взаємодію протоколу із рівнями компресії та безпеки.

1.5 Формати представлення й серіалізації даних

У промислових бездротових мережах не менш важливу роль відіграє спосіб, у який дані організовано для передачі. Від того, наскільки компактно та структуровано вони зашифровані в пакеті, залежить як розмір корисного навантаження, так і складність наступної обробки. Традиційно в додатках, що історично виросли з вебмсередовища, популярним форматом є JSON (JavaScript Object Notation) – доволі простий і людиночитний спосіб опису структурованої інформації [18]. JSON широко підтримується і зручний для налагодження (debugging), проте має значні накладні витрати за рахунок дужок, ком і лапок, порівняно з компактнішими бінарними форматами.

Бінарні формати, такі як Protocol Buffers (protobuf), Thrift, msgpack чи CBOR – дозволяють помітно зменшити розмір пакета й пришвидшити його обробку, оскільки в них відсутня зайва текстова надмірність [19]. Protocol Buffers, розроблений Google, набув особливої популярності завдяки близькій інтеграції з gRPC, а також чіткій схемі (schema) опису структур даних, яка спрощує генерування коду для різних мов програмування. msgpack і CBOR, своєю чергою, наголошують на простоті відображення даних із ключем-значенням (ключ це рядок або число, значення це різні типи), але без додаткових схем [20]. Цей підхід часто приваблює своєю гнучкістю, оскільки формат повідомлення може змінитися без необхідності переписувати суворі описові файли. Утім, зі зростанням масштабності систем LLC (low-latency communication) у промислових умовах така «вільність» може виявитися джерелом помилок, адже різні вузли мають чітко розуміти, яка конкретна структура надходить.

Однією з ключових переваг бінарних форматів у поєднанні зі стисненням є те, що вони й без компресії мають менший обсяг, що полегшує завдання алгоритму стиснення. Якщо ж доповнити їх gzip, lz4 або іншим підходом, нерідко вдається досягти винятково високих коефіцієнтів стискання на повторюваних фрагментах, особливо в системах, де дані мають схожу

«структуру» (наприклад, однакові поля сенсорів, що змінюються в обмеженому діапазоні). Натомість JSON чи XML із тією самою інформацією потребуватимуть більшого підсумкового розміру, а відповідно й робота компресора може виявитися не настільки швидкою й ефективною.

Нерідко у промислових системах трапляються також власні (custom) бінарні формати серіалізації, створені спеціально під конкретне завдання: вони максимально оптимізовані під усталений набір полів і містять жорстко зафіксовані відступи байтів для числових або рядкових значень. Такий підхід має переваги в продуктивності: мінімальні накладні витрати на розбір (parsing), відсутні «зайві» поля і метадані. Однак він може створити труднощі в еволюції системи, оскільки будь-яка зміна структури даних потребуватиме сумісних оновлень на всіх вузлах. Попри це, у строго регульованих ланцюгах (де набір полів стабільний) власний бінарний формат може забезпечити максимальну ефективність і простоту впровадження компресії.

На рисунку 1.7 представлене порівняння json, protobuf у мові програмування Go.

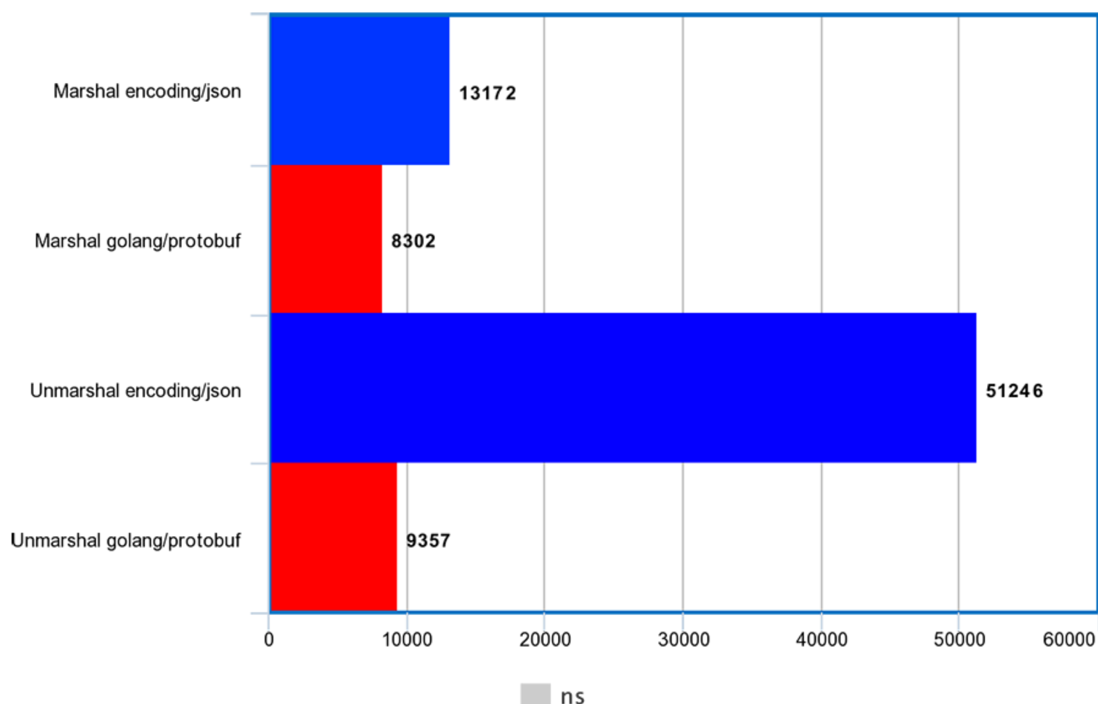

Рисунок 1.7 – Порівняння json, protobuf

Таким чином, вибір формату серіалізації слід сприймати як важливу складову стратегічного планування передачі даних. Свідоме рішення на користь текстового (читаємого) чи бінарного варіанта тісно пов’язане зі складністю системи, частотою змін структури повідомлень, вимогами до відлагодження та швидкості обробки, а також характером трафіку.

1.6 Взаємодія алгоритмів стиснення, мережевих протоколів і форматів

Глибинне розуміння промислових бездротових систем деталізується не лише аналізом окремих шарів та методів стискання чи форматів представлених даних, а й детальним вивченням взаємодії між різними технологічними компонентами на кожному етапі передачі. Уявити реальну роботу такої системи можна, розглянувши послідовність винятків та впливів: якщо на рівні форматування відбувається перехід із текстового JSON на бінарний формат диференційованого кодування, це неодмінно позначається на характеристиках стиснення, які охоплюють структуру вхідних даних, обсяг надмірної інформації та можливості алгоритму шукати повторювані фрагменти. Ця зміна також може впливати на наступну стадію декомпресії, оскільки зростання швидкості стискання не завжди прямо корелює з адекватною пропускною здатністю на приймальному боці, особливо якщо у каналі циркулює кілька різнорідних потоків. У багатошаровій архітектурі ці фактори продовжують множитися, оскільки взаємодія між протоколами передачі, вибраними методами шифрування, а також специфікою апаратних можливостей кінцевих пристроїв накладає додаткові обмеження та вимоги.

Наприклад, якщо сенсори передають інформацію у вигляді структур діагностичних даних з високою часовою роздільною здатністю, збільшення обсягу стиснення може урізноманітнити розмір буферів, що використовуються в проміжних вузлах типу шлюзів. У випадку, коли компресор отримує бінарні потоки з високим ступенем повторюваності у байтовому діапазоні, він може ефективніше знаходити закономірності та губити надлишкову інформацію,

зменшуючи загальний вихідний обсяг. Однак таке рішення спричиняє вплив на затримку, оскільки ефективне стиснення часто передбачає додаткову обробку, що може бути недоцільною в реальному часі, особливо в критичних сценаріях керування технологічним обладнанням. Подібна суперечність між необхідною якістю зв'язку та вимогами до швидкодії є центральною проблемою в багатьох проєктах промислового інтернету речей, де на кону стоїть стабільне функціонування конвеєрів, роботизованих вузлів чи термінально важливих систем моніторингу. Усвідомлення цих обмежень вимагає глибшого аналізу взаємодії між апаратним та програмним забезпеченням, а також адаптивних підходів до оновлення стискальних алгоритмів залежно від динаміки вхідних даних.

У межах ієрархічної схеми передачі даних, притаманної великим промисловим мережам, різні рівні мають різну толерантність до пропускної затримки, різні ресурси обчислювальної потужності та різні вимоги до обсягу передачі. Зазвичай на найнижчому рівні, де сенсор зчитує величину температури, тиску, вібрації чи інші критичні параметри, застосовується мікроскопічна операція обробки, оскільки наявна апаратна платформа часто має обмежену пам'ять та центральний процесор невисокої продуктивності. У такому випадку, якщо намагаються впровадити надмірно складне стискання, це може призвести до недостатньої швидкодії, зривів у реальному часі або підвищеного енергоспоживання. Водночас шлюз, який розміщений вище, володіє суттєво вищим рівнем ресурсів і виконує централізоване об'єднання, верифікацію та агрегування вхідних даних, що надає йому змогу застосовувати більш інтенсивні схеми стискання. Таким чином, якщо на рівні сенсора додаткове стиснення мінімізується чи взагалі відсутнє, шлюз виконує консолідований підхід, збираючи кілька потоків і стискаючи їх одним викликом відповідного алгоритму в той момент, коли це оптимально з точки зору сукупного навантаження. У центральної частини мережі або в хмарі стають можливими ще більш дорогі обчислення, які нерідко застосовуються для глибокого архівування чи післяопераційної аналітики. Це ілюструє, наскільки важливо враховувати

багаторівневу природу передачі інформації, а також наскільки сильно відрізняються технічні умови на кожному з шарів, включно з потенційним переходом від простих обмінів пакетами до повноцінної багатоканальної комунікації.

При додаванні різних транспортних протоколів, таких як UDP та TCP, виникає ще більше ускладнень, оскільки вони реалізовують принципово різні підходи до контролю трафіку, відновлення втрат та гарантій доставки. UDP пропонує мінімальні накладні витрати при передачі, що корисно у реальних середовищах із критичним наданням часу, однак відсутність механізму повторної передачі пакетів може призвести до втрат даних, які в низці випадків є неприпустимими в промислових процесах. TCP, зі свого боку, забезпечуючи надійне доставлення та послідовність сегментів, може породжувати зайві затримки через механізми квитування і буферизації, що теж може бути проблемним у системах, де необхідна гарантовано коротка реакція на зміну параметрів середовища. HTTP/2 або подібні протоколи вищого рівня розв’язують окремі завдання мультиплексування потоків і ефективного кодування заголовків, але їх застосування накладає власні обмеження: складніша реалізація, більші вимоги до процесорної потужності та ймовірність конфлікту з існуючими інструментами моніторингу або збору статистики. Не можна недооцінювати й ситуації, коли для оптимізації потоку чи узгодження типів повідомлень використовують проміжні програмні адаптери або проксі. З одного боку, це надає гнучкість, дозволяючи швидко адаптуватися до нових форматів чи випадків застосування. З іншого боку, кожен такий адаптер створює точку можливої затримки і потребує ретельного налаштування, щоб обмежити негативний вплив на пропускну здатність мережі.

Стискання всередині цих складних структур не можна розглядати відірвано від реалій безпеки, оскільки переважна більшість промислових даних підлягає зберіганню та передачі в зашифрованому вигляді. Якщо в системі використовується наскрізне шифрування, проміжні елементи не розпаковуватимуть дані для вторинної компресії, тим самим потенційно

знижуючи загальний коефіцієнт стиснення, адже повторне стиснення зашифрованих даних зазвичай не дає відчутного результату. Натомість методологія попереднього стиснення, коли дані шифруються вже після того, як було виконано алгоритм зменшення розміру, є більш оптимальною з погляду зменшення мережевих витрат. Проте це може бути несумісно з протоколами безпеки, в яких окремі вузли не мають права бачити нешифровані дані. У разі, коли є сегменти інформації, позбавлені критичної чутливості, інколи застосовують частковий розподіл на зашифровані та відкриті поля, щоб підвищити застосовність стиснення, мінімізувавши ризик витоку конфіденційних значень. Такі гібридні моделі вимагають серйозного доопрацювання в частині керування ключами, трансформаціями та контролю доступу на кожному з проміжних етапів, щоб ані шлюз, ані центральний сервер не порушували загальних політик захисту при використанні власних схем зберігання й аналізу даних.

Важливо зазначити, що складність, яка виникає через багатошаровість архітектури, варіюється залежно від масштабів системи та ступеня інтеграції сторонніх пристроїв і служб. У деяких випадках, якщо сенсорів небагато і вони працюють у відносно вузькому діапазоні частот опитування, більшість залежностей можна унормувати спрощеним, майже статичним налаштуванням. Але в масштабних системах, де кількість сенсорів і шлюзів може обчислюватися сотнями або тисячами, а канали збору даних охоплюють різні виробничі цехи чи навіть різні географічні регіони, відбувається лавиноподібне зростання складності. У подібних умовах механізми «розумної» маршрутизації, балансування трафіку та гнучкого стиснення починають грати вирішальну роль. З’являється потреба розробляти спеціалізовані програми-транзитери, які здатні певною мірою передбачати коливання обсягу вхідних потоків, адаптувати ступінь стискання залежно від пікових навантажень і автоматизовано перемикатися між різними форматами даних. Іноді такі системи вбудовують елементи машинного навчання, що можуть прогнозувати часові ряди та виявляти

аномалії в режимі реального часу, підлаштовуючи алгоритми шифрування й стиснення до профілю трафіку.

Докладний розгляд усіх цих аспектів важливий і в контексті довготривалих перспектив розвитку промислових бездротових мереж. З погляду масштабованості, спроектована архітектура повинна пропускати можливість додавання нових пристроїв чи сервісів, які можуть суттєво відрізнятися за структурою даних, схемою доступу чи інтенсивністю передавання. Це вимагає, зокрема, модульного підходу до впровадження стиснення, де алгоритмічний блок можна оновити, замінити чи розподілити на інші вузли без повного перевпровадження протоколів чи форматів. Водночас надмірна універсальність може ускладнити як початкову розробку, так і подальшу підтримку: чим гнучкіше рішення, тим більше воно потребує складних схем відстеження версій, синхронізації між компонентами та управління залежностями. Надмірне ускладнення також тягне за собою вищі вимоги до кваліфікації команди та часто передбачає збільшення загальних витрат на проєкт.

Не менш суттєвим є питання про вибір відповідних стандартів, що регулюють прийнятну сумісність обладнання. Виробники промислових сенсорів часто пропонують власні протоколи зв’язку, дотримуючись єдиних рекомендацій лише частково. Це додає ще одну змінну до рівнянь, що описують оптимальний розподіл модулів стиснення та формування пакетів: у деяких реалізаціях може виявитися неможливим використовувати сторонній формат через обмеження мікропрограмного інтерфейсу. Потрібна тонка гармонізація між поточним набором пристроїв і потенційними розширеннями в майбутньому, аби система функціонувала стабільно та ефективно протягом тривалого життєвого циклу.

Одним з головних координуючих чинників у цих процесах стає управління виробничими циклами даних. Через те, що в багатьох прикладних задачах системи працюють безперервно, паузи на оновлення, перепрошивку чи переконфігурування окремих блоків стиснення є обмеженими. Будь–яке переривання потоку може призвести до втрати критичних показників стану

обладнання. Тому конфігураційні зміни бажано впроваджувати покроково та із надмірним дублюванням функцій, щоб уникнути простоїв. Це накладає вимогу на систему управління конфігураціями, яка повинна бути здатна гнучко керувати більшою кількістю версій протоколів і алгоритмів, водночас гарантувати зворотну сумісність зі старими пристроями. Ретельна організація подібної багаторівневої інфраструктури дозволяє збалансувати оперативну надійність системи і потребу в регулярних оновленнях чи покращеннях методів стискання.

Вирішальним залишається і фактор контролю пропускної здатності у каналів, що використовуються в промислових мережах. Хоча в розвинених регіонах дедалі частіше застосовують швидкі та стійкі бездротові стандарти, все ще залишається багато випадків, коли передача обмежена вузькосмуговим каналом або ж робота відбувається у середовищі з високим рівнем перешкод. У таких сценаріях недостатня продуктивність стискання або неправильний вибір бінарних форматів можуть спричинити велике накопичення даних в чергах та затримки в отриманні критично важливої телеметричної інформації. Більше того, у виробничих умовах можливі миттєві сплески інформаційного потоку, пов’язані з аварійними сигналами або короткотривалим збільшенням частоти зчитування, які неодмінно призведуть до ще більшого зростання затримок, якщо алгоритми стиснення не здатні оперативно адаптуватися. Це показує, що ефективне налаштування механізмів стискання на додачу до грамотного вибору протоколів транспорту є взаємопов’язаними факторами, які істотно впливають на загальну продуктивність.

У підсумку, всі згадані аспекти – від специфіки взаємодії ланцюгів стиснення та протоколів з урахуванням шифрування, до потреби в ієрархічному підході та орієнтації на довгострокову масштабованість – ілюструють важливість комплексної інтеграції методів обробки даних у промислових бездротових мережах. Погано узгоджена конфігурація навіть одного рівня може викликати каскад проблем, які виявляться на інших ділянках з суттєвою затримкою у часі, підвищивши ймовірність збоїв у ключових ланках виробництва або допустивши витоки конфіденційної інформації. Відповідно,

інтегроване проектування кожного з етапів, починаючи від первинних сенсорів і закінчуючи високорівневою аналітикою в хмарному середовищі, є єдиним надійним способом забезпечити високу доступність, швидкодію та гнучкість систем, які мають витримувати розширення кількості пристроїв та ускладнення алгоритмів аналізу в майбутньому. Це дає можливість побудувати масштабовані, надійні мережеві рішення, що зосереджуються на продуктивних методах стискання і дотриманні належних протоколів безпеки, одночасно враховуючи обмеження реального часу та пріоритети безаварійної експлуатації сучасних промислових об’єктів.

1.6 Майбутні виклики та потенційні напрями розвитку

Розвиток промислових бездротових мереж на перетині сучасних інформаційних технологій та реальних виробничих потреб впливає на низку науково–технічних завдань. З одного боку, спостерігається постійне збільшення обсягів даних, включно з потоками телеметрії, логами роботи машин, відеоаналітикою й іншою інформацією, що надходить від різнорідних пристроїв. З іншого – збільшуються вимоги до швидкості та надійності обміну: високі темпи автоматизації, бажання оперативно реагувати на зміни в технологічному процесі та підвищення рівня безпеки вимагають від мережі нижчої латентності й кращого рівня відмовостійкості.

У контексті стиснення даних з’являються нові алгоритмічні напрацювання, орієнтовані на виконання машинного чи глибинного навчання безпосередньо «на краю» (edge AI). Ідея полягає в тому, що можна зменшити обсяги передаваних пакетів, заздалегідь обробляючи й фільтруючи непотрібну або малозначущу інформацію. Так, якщо сенорні дані свідчать про нормальний режим експлуатації, вузол-обчислювач передає лише стислу «статистичну» суму, а повні дані відправляються у випадках, коли алгоритм помічає потенційну аномалію. Такий підхід прибирає зайве навантаження з бездротових каналів і дає змогу дотримуватися вимог жорстоких часових правил там, де потрібна швидка

реакція. Водночас він ставить складні питання верифікації: чи достатньо надійний локальний аналіз, аби не пропустити критичну подію?

Ще один перспективний напрям – розвиток «резервованих» (redundant) шляхів передачі з інтелектуальним розподілом трафіку залежно від зміни рівня перешкод або гостроти подій. У таких системах бездротова мережа може використовувати відразу декілька радіоканалів чи різні протоколи, й автоматично перемикатися на найменш завантажений або найбільш стабільний у даний момент. Стиснення при цьому може вмикатися вибірково, там, де переповнення каналу стає загрозливим, чи навпаки вимикатись, щоб підвищити швидкість реакції. Ця «адаптивна» модель зв'язку передбачає, що алгоритми прийняття рішень змінюють профілі компресії й мережеві налаштування буквально в режимі реального часу, оптимізуючи потік даних у міру виявлення стану мережі та виробничої ситуації.

Узгодити стиснення, протокол і безпеку стає особливо складно, коли застосовують методи контейнеризації та мікросервісну архітектуру у межах автоматизованих фабрик. Кожен мікросервіс може мати власні обмеження й вимоги, наприклад, обробляти багаторічні статистичні дані чи аналізувати відеопотік у режимі 24/7. Коли ця інфраструктура працює у хмарі з платіжною моделлю «pay-as-you-go», оптимізація обсягу передавання стає безпосередньо пов'язаною з вартістю обчислень і мережевого трафіку. Завчасна компресія на прикладному рівні, налаштування оптимального рівня стиснення в gRPC чи використання бінарних форматів серіалізації можуть суттєво вплинути на загальні витрати, водночас потребуючи додаткових обчислювальних ресурсів на «периферії». Тут знову виникає потреба в ретельному балансуванні параметрів.

Насамкінець, виникає виклик стандартизації: через розмаїття підходів, власних рішень і протоколів у промислових мережах стоїть завдання забезпечити сумісність обладнання та внутрішніх алгоритмів. Поряд із сертифікацією безпеки, закритість деяких форматів і складність впровадження передових методів компресії стримують процес їх поширення. Наприклад, формат Protocol Buffers, хоча й добре задокументований, є під патронатом Google, що може

ускладнювати його офіційне впровадження в певні стандарти. Подібні перепони стосуються й інтеграції інтелектуальних механізмів стискання, для яких необхідно доопрацьовувати апаратну й програмну інфраструктуру. Проте продовжують формуватися галузеві альянси, дослідницькі ініціативи, і можна очікувати, що в найближчій перспективі високопродуктивне, гнучке й безпечне стиснення стане невіддільною частиною екосистеми промислового інтернету речей.

1.7 Висновки до розділу

Проведений у цьому розділі огляд продемонстрував розгалужену й багатогранну структуру проблеми стиснення даних у промислових бездротових мережах. Високу затребуваність таких підходів пояснює поєднання суворих вимог до надійності, часових параметрів та безпеки з дедалі більшим обсягом інформації, який генерують сучасні виробництва. У цих умовах навіть незначне зменшення розміру повідомлень чи скорочення латентності передачі здатне позитивно позначитися на пропускній здатності мережі, збільшити оперативність реагування на зміни технологічного процесу й водночас знизити навантаження на обчислювальні ресурси.

У межах розгляду найчастіше згадувалися три центральні складові:

– алгоритми стиснення (gzip, lz4, zstd та інші), що пропонують широкий вибір між швидкістю компресії, ступенем стиснення та споживаними ресурсами;

– транспортні протоколи (TCP, UDP, HTTP/2, HTTP/3, gRPC), які різняться за рівнем гнучкості, масштабовності й механізмами відновлення при втраті пакетів;

– формати представлення даних (JSON, Protocol Buffers, msgpack, власні бінарні формати), які по-різному впливають на обсяг переданих повідомлень, простоту налагодження та сумісність учасників мережі.

Окрему увагу було приділено особливостям промислового середовища, де наявність електромагнітних перешкод, суворих температурних режимів,

чутливості до затримок та вимог до безпеки істотно ускладнює підхід до організації бездротових каналів. Обговорювалися техніки адаптивного увімкнення чи вимкнення компресії, поєднання різних рівнів протоколу й мультикастових/бродкастових режимів залежно від пріоритетності даних та реальної завантаженості каналу.

Особливе місце займало питання безпеки та сумісності шифрування зі стисканням, оскільки промислове обладнання часто передає інформацію, що містить комерційно чутливі чи критично важливі параметри. Стискання після шифрування зазвичай малоефективне, а попереднє (pre-encryption) вимагає правильної інтеграції з протоколами, щоб не порушувати політик конфіденційності.

Розглянуто низку перспективних тенденцій, включно з інтегрованим «крайовим» штучним інтелектом (edge AI) та багатопотоковими стратегіям. Ці напрями показують, що промислові бездротові мережі рухаються до складних, самоналаштовних систем, які в реальному часі обирають оптимальні умови для передачі даних, від зміни алгоритму компресії до автоматичного перемикання каналів.

Проведений аналіз формує основу для експериментальної частини роботи, в якій буде досліджено ефективність різних поєднань (протокол + формат + алгоритм стиснення) в умовах, наближених до реальних промислових сценаріїв. Це дає змогу привести теоретичні міркування про компроміси між пропускною здатністю, швидкість відгуку, надійністю й ресурсозатратами до кількісних результатів і рекомендацій, застосовних у реальних інженерних проєктах.

2 МЕТОДИ ТА ІНСТРУМЕНТИ ДОСЛІДЖЕННЯ

2.1 Загальна концепція експериментів

Для забезпечення належної валідності та надійності результатів, дослідження потребує ретельно розробленої методики, яка б охоплювала як якісний (описовий) бік оцінювання продуктивності, так і кількісні показники.

Важливо врахувати характер промислового середовища та імітувати в експериментах умови, що наближені до реальних – зокрема, нестабільність радіоканалу, затримки в його обслуговуванні, фонове навантаження на мережу тощо.

Основна ідея полягатиме у побудові стенду, де можна гнучко змінювати:

– алгоритми стиснення;

– рівні «агресивності» стиснення (наприклад, preset = 1, 5 чи 9 у випадку gzip і т.д.);

– серіалізаційні формати (JSON, Protocol Buffers, msgpack);

– транспортні протоколи (HTTP/2, HTTP/3).

Для кожної комбінованої конфігурації необхідно зібрати статистику щодо:

– середньої затримки передачі (latency);

– пропускної здатності (throughput) або загального обсягу даних, що передано за одиницю часу;

– відсотка втрат пакетів чи повторних передач (якщо це критично для певного протоколу);

– обчислювального навантаження на процесор або інші ресурси (CPU usage, енергоспоживання).

Паралельно до цих «чисто мережевих» метрик, експерименти можуть фіксувати інші критерії важливості, такі як можливість масштабування і стійкість до помилок (як система поводиться, коли частина даних пошкоджується).

2.2 Обґрунтування вибору інструментарію

Для реалізації та моніторингу експериментів знадобиться комплекс інструментів та бібліотек:

– засоби генерації даних: це можуть бути як випадкові (randomized) пакетні потоки, так і «реальні» виробничі логи чи історичні вибірки телеметрії. Щоб наблизити експерименти до промислових умов, доцільно створити synthetic датасети, в яких значна частина полів повторюється, але є також певна динаміка змін;

– лабораторний стенд для радіоканалу: для моделювання реальних умов можна використовувати апаратний симулятор перешкод (noise injector) чи застосовувати програмне утиліту netem (Network Emulator) у Linux, щоб регулярно додавати штучні затримки, обмежувати пропускну здатність, насичувати канал втратою пакетів певного відсотка тощо;

– бібліотеки стиснення: офіційні реалізації gzip/zlib, lz4, zstd, а також можливо інші, залежно від інтересу. Ці реалізації доступні для більшості мов програмування (C/C++, Python, Go, Java тощо);

– бібліотеки та фреймворки для протоколів: для HTTP/2, HTTP/3 (наприклад, QUIC-стек) і gRPC існують готові імплементації, які варто оглянути з позицій зручності інтеграції та наявності потрібних інструментів тестування.

Таким чином, добирається набор як готових open-source рішень, так і самописних скриптів для пружної (flexible) конфігурації різноманітних сценаріїв.

З боку мови програмування, вибором для написання програмного додатку для передачі даних між пристроями була обрана мова програмування Go.

Go (Golang) є перспективною мовою програмування для розробки мережевих застосунків завдяки поєднанню високої продуктивності та простоти синтаксису. В основу Go покладена ідея ефективності: він використовує статичну типізацію та машинну компіляцію, що надає йому перевагу над інтерпретованими мовами з погляду швидкості виконання коду. Одночасно

Golang зберігає зручність високорівневих мов, пропонуючи компактний і зрозумілий синтаксис. Це критично важливо для швидкої розробки мережевих рішень, коли необхідно не лише досягти високої продуктивності, а й скоротити час до впровадження продукту [21].

На рисунку 2.1 представлені компанії, що користуються мовою програмування Go [22].

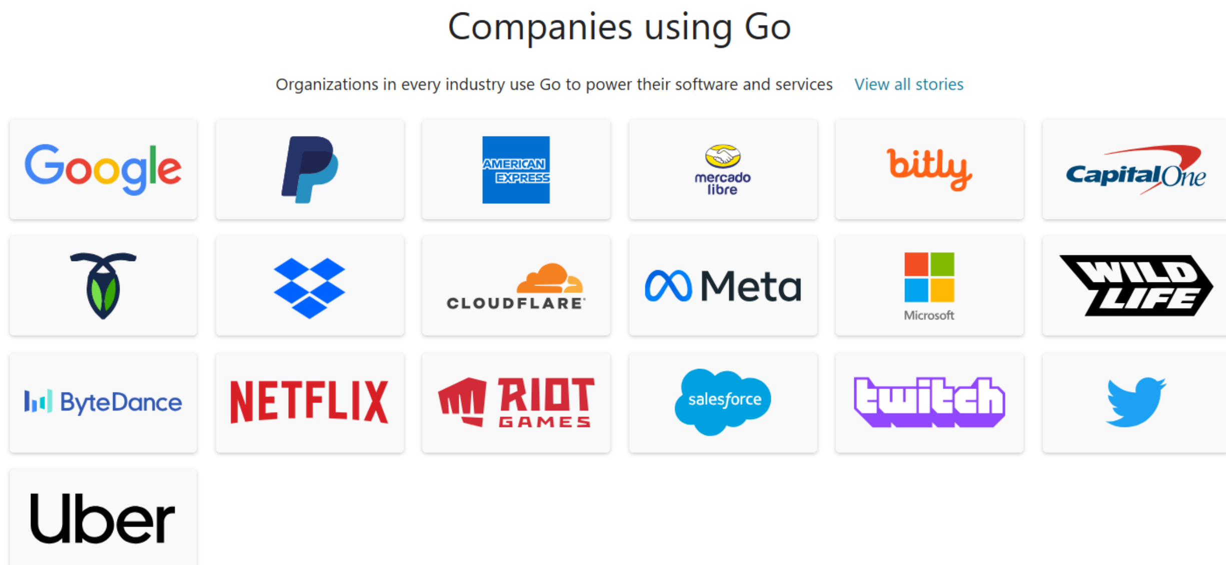

Рисунок 2.1 – Компанії, що користуються Go

Ще однією суттєвою перевагою Go є інструментарій для конкурентних обчислень. Механізм goroutine забезпечує легку та ефективну реалізацію паралельного виконання, не створюючи додаткового навантаження на систему через надмірне споживання ресурсів. На відміну від традиційних потоків в операційних системах, goroutine займають порівняно мало пам'яті та запускаються практично миттєво. Паралельне оброблення великої кількості мережевих з'єднань або асинхронних викликів є ключовим чинником у високонавантажених застосунках, а Go робить це без складних механізмів керування потоками та блокуваннями.

Крім того, Go пропонує зручний інструментальний ланцюг для розробки, тестування та розгортання. Інструмент go test надає швидкий та простий спосіб

організувати тестові сценарії, що особливо корисно для мережевих застосунків, де важливо перевіряти коректність оброблення протоколів та даних у різних умовах. Також наявність вбудованого інструмента go tool cover дозволяє автоматично оцінювати відсоток покриття тестами, що стимулює створення надійних та якісних рішень у галузі мережевого програмування.

Частиною стандартної бібліотеки Go є пакет net/http, який суттєво спрощує розробку веб-серверів та клієнтів. Він пропонує зрозумілі функції для роботи з HTTP-протоколом, взаємодії з заголовками, обробки запитів і відповідей. Завдяки продуманій структурі та зручним інтерфейсам розробники можуть швидко створювати надійні RESTful API, інтеграційні сервіси чи веб-сервіси для хмарних платформ. При цьому, якщо необхідні більш низькорівневі інструменти, програміст без додаткових бібіліотек може користуватися необробленими TCP/UDP з’єднаннями, адже пакет net має відповідний базовий функціонал.

Вельми помітною перевагою Go є його вбудована система керування залежностями та підтримка модулів (Go Modules). Це сильно спрощує інтеграцію зовнішніх бібліотек, а також полегшує структурування великих проектів, поширених у сфері мережевого програмування. Завдяки чіткому оголошенню версій та ефективному кешуванню пакетів скорочується ризик несумісності бібліотек, що особливо важливо для безпечної і стабільної роботи сервісів, які обробляють мережеві дані.

Таким чином, Golang водночас поєднує в собі високий рівень продуктивності, легкість конкуренції та багату екосистему стандартних пакетів. Для створення та підтримки мережевих додатків ключовим аспектом успіху є можливість швидко впроваджувати зміни, зберігаючи високу ефективність обробки даних. Go робить це завдання досяжним завдяки своєму дизайну та орієнтації на сучасні виклики мережевого середовища, випереджаючи альтернативні мови за багатьма критеріями, особливо у випадках, коли пріоритетом є масштабованість і стабільна робота додатка.

Для аналізу даних була обрана мова програмування Python.

На рисунку 2.2 представлений рейтинг StackOverflow Developer Survey 2024, за результатами якого Python посідає 4 місце, а якщо брати мови програмування, то друге місце по популярності серед професійних розробників [23].

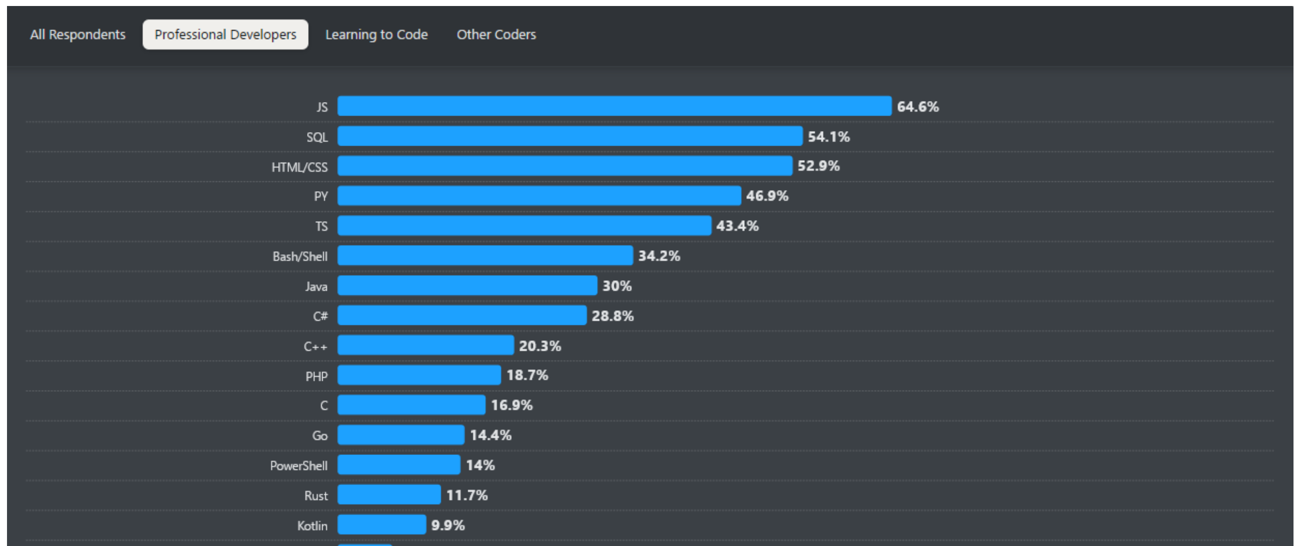

Рисунок 2.2 – StackOverflow Developer Survey 2024

Python широко визнаний як одна з провідних мов програмування для аналізу даних завдяки своїй зручності, читабельності та різноманітності спеціалізованих бібліотек [24]. Структурована та мінімалістична синтаксична конструкція спрощує реалізацію алгоритмів наукового аналізу, роблячи процес дослідження та оброблення інформації швидшим і менш схильним до помилок. Крім того, сильна типізація та чіткий інструментарій тестування підтримують акуратний і контрольований процес розробки.

Водночас значний внесок у популярність Python відіграє масштабна спільнота розробників та ентузіастів, які діляться своїм досвідом і створюють нові інструменти. Безперервна підтримка відкритого коду, свіжі оновлення та приклади застосування гарантують, що навіть новачки швидко знаходять ефективні рішення. Розгалужена екосистема також передбачає доступ до різноманітних форумів, конференцій та вебінарів, збільшуючи можливості для обміну знаннями й безперервного професійного розвитку.

Комплексний набір зовнішніх бібліотек, зокрема NumPy, Pandas та SciPy, суттєво розширюють аналітичні можливості Python. Їхній функціонал охоплює широкий спектр завдань, включно з маніпуляцією табличними наборами даних, ефективною роботою з багатовимірними масивами та виконанням складних статистичних процедур. Завдяки цьому стає можливим обробляти великі обсяги інформації, не вдаючись до громіздких методів або надмірної ручної оптимізації.

Важливою перевагою Python для візуалізації даних є його інтерактивне середовище, реалізоване в Jupyter Notebook. Використання таких ноутбуків дає змогу миттєво змінювати налаштування побудови графіків, експериментувати з різними типами діаграм і швидко виявляти ті способи представлення результатів, які найкраще відповідають вимогам дослідження. За потреби, Plotly чи Bokeh надають інтерактивні веб-орієнтовані засоби для візуального дослідження великих та різноманітних наборів даних.

Таким чином, Python відзначається не лише зручним синтаксисом і легкістю інтеграції різних аналітичних методик, а й широкими можливостями для якісної візуалізації. Саме завдяки цьому аналіз та побудова графіків з вихідних даних стають більш зрозумілими, гнучкими й інтерактивними, що посилює ефективність наукових досліджень і пришвидшує отримання результатів у найрізноманітніших галузях.

2.3 Критерії оцінювання

Для експерименту потрібно розробити критерії оцінювання, за якими буде знайдено найкраще рішення.

Було визначено наступний набір тестових змінних:

- вибір формату даних (JSON / Protocol Buffers / msgpack / XML тощо);
- увімкнута / вимкнута компресія (з різними рівнями);
- протокол (HTTP/2 / HTTP/3);
- штучні обмеження каналу (пропускна здатність, затримки).

Для якісної оцінки варто зафіксувати ключові показники (KPI):

– усереднене значення RTT (round-trip time), медіана й процентилі (99th percentile);

– усереднена пропускна здатність і значення пікового завантаження;

– використання процесору та пам’яті на відправній і приймальній сторонах для кожного режиму;

– середній коефіцієнт стиснення (compressed / uncompressed size) та час, витрачений на стиск / розпакування.

При достатньому обсязі експериментальних вимірювань буде можливо побудувати порівняльні графіки і таблиці, що засвідчать у яких умовах конкретна технологія дає найліпший результат, а де може бути недоречною.

2.4 Обмеження та похибки

Насамперед слід усвідомити, що жоден лабораторний експеримент не зможе на сто відсотків відтворити всі чинники, притаманні справжній промисловій експлуатації. Одна з головних причин полягає в тому, що в дослідженні зазвичай використовується обмежена кількість вузлів відправлення та приймання. У реальних виробничих мережах таких вузлів може бути значно більше, що автоматично формує інший профіль навантаження та вимагає інакших рішень з точки зору керування трафіком. До того ж у багатьох промислових системах з’єднання між вузлами передбачає складніші схеми, наприклад наявність проміжних шлюзів і додаткових механізмів безпеки, таких як VPN чи міжмережеві екрани, які помітно впливають на затримки.

Ще одним викликом є штучні затримки і втрати, які здебільшого генеруються інструментами на кшталт netem. Такі моделювання хоч і можуть ввести випадкові перешкоди, усе ж залишаються лише приблизним відображенням реальних завад, що діють у промисловому середовищі. Скажімо, раптові електромагнітні імпульси, котрі можуть виникати у виробничому цеху, мають непередбачувану періодичність і часто перевищують рівень «штучного

шуму», закладеного в умовах експерименту. Крім того, у лабораторії зазвичай неможливо повноцінно імітувати роботу великої кількості різних пристроїв і способів передачі даних, через що істинне навантаження на мережу може відрізнятись від отриманого під час випробувань.

Важливо враховувати й те, що стендові ресурси, включно з апаратним забезпеченням, можуть відрізнятись від того, що реально застосовується у виробництві. Наприклад, у тестовому середовищі доступні процесори іншого покоління або інші мережеві карти, які по-різному впливають на продуктивність компресії чи рівень енергоспоживання. Середовище розробки теж не завжди ідентичне бойовому, оскільки бібліотеки компресії та операційні системи можуть мати різні версії, власну специфіку планування завдань чи додаткові функції безпеки. Усе це впливає на коректне узгодження сокетів і загальну поведінку програмного забезпечення в мережі нового покоління, що часто не помітно в мінімалістичних умовах експерименту.

Не варто також нехтувати потребою у статистичній оцінці результатів. Для формування дійсно репрезентативних висновків кожен тип випробувань слід запускати багаторазово з різними початковими умовами, такими як відмінні «рандомні зерна». Якщо ж нехтувати цим правилом, то виникає ризик похибок, зумовлених випадковими артефактами, які можуть суттєво спотворити висновки про ефективність застосованих технологій. Усі згадані моменти вказують на те, що, незважаючи на високий рівень наближеності дослідження до реальних виробничих обставин, результати лишаються переважно індикативними. Щоби досягти максимальної точності та остаточно шліфувати параметри, необхідно проводити додаткові тести у безпосередньому виробничому середовищі або, принаймні, у його наближеній копії, де можна відтворити всі аспекти промислової інфраструктури з максимально можливою детальністю.

2.5 Висновки до другого розділу

У цьому розділі визначено структуру, цілі та деталі реалізації експериментальних випробувань, що покликані оцінити ефективність різних поєднань стиснення, протоколів і форматів передачі у промислових бездротових мережах. Зроблено наголос на важливості:

– імітації реальних умов за допомогою інструментів на зразок netem або апаратних генераторів перешкод;

– варіативності сценаріїв (різні обсяги даних, різні вимоги до часу);

– комплексної оцінки результатів (latency, throughput, коефіцієнт стиснення, навантаження на CPU тощо);

– статистичної надійності результатів (необхідність повторних запусків, ретельного моніторингу).

Усе це дає змогу покроково перевірити гіпотези, сформульовані в першому (теоретичному) розділі, і надалі розробити обґрунтовані рекомендації для використання алгоритмів компресії і мережевих технологій у промислових бездротових мережах. У наступних частинах роботи буде описано безпосереднє розгортання лабораторного стенду, викладено методи обробки отриманих даних, а також продемонстровано результати експериментального порівняння.

3 РЕАЛІЗАЦІЯ ЕКСПЕРИМЕНТІВ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Опис стенду для виконання експерименту

Сервер у стенді побудований на базі процесора AMD Ryzen 5 5700X3D, що належить до покоління високопродуктивних чипів із розширеним об’ємом кеш–пам’яті.

На рисунку 3.1 зображені характеристики AMD Ryzen 5 5700X3D.

AMD Ryzen™ 7 5700X3D

Regional Availability: Global, China, NA, EMEA, APJ, LATAM	**Platform:** Boxed Processor	**Product Family:** AMD Ryzen™ Processors
Product Line: AMD Ryzen™ 7 Desktop Processors	**# of CPU Cores:** 8	**# of Threads:** 16
Max. Boost Clock: Up to 4.1GHz	**Base Clock:** 3.0GHz	**L1 Cache:** 512KB
L2 Cache: 4MB	**L3 Cache:** 96MB	**Default TDP:** 105W
Processor Technology for CPU Cores: TSMC 7nm FinFET	**CPU Socket:** AM4	**Socket Count:** 1P
Thermal Solution (PIB): Not Included	**Max. Operating Temperature (Tjmax):** 90°C	**Launch Date:** 1/8/2024
***OS Support:**		

amd.com

Рисунок 3.1 – Технічні характеристики процесору AMD Ryzen 5 5700X3D

Цей процесор передбачає 8 фізичних ядер з підтримкою багатопоточності, що загалом дає 16 потоків виконання інструкцій. Така конфігурація дає змогу ефективно обробляти одночасно значну кількість завдань, зокрема пов’язаних із мережею.

Численні потоки виявляються корисними, коли виникає потреба запускати декілька паралельних процесів, наприклад серверну службу й додатковий аналізатор трафіку чи інструменти для моніторингу пропускної здатності.

Обсяг оперативної пам’яті 32 ГБ з частотою 3200 МГц забезпечує запас для оброблення пакунків даних і прискорених операцій з буферами, що напряму впливає на швидкодію при тестах передачі даних.

Сервер оснащено твердотільними накопичувачем (SSD) Samsung 970 Evo PLUS, який має високу швидкість читання/запису.

На рисунку 3.2 зображений цей твердотільний накопичувач.

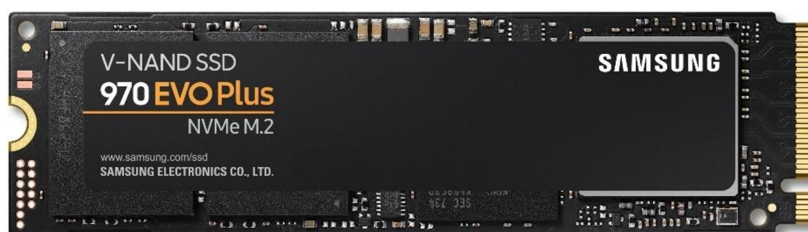

Рисунок 3.2 – Твердотільний накопичувач Samsung 970 Evo PLUS

На рисунку 3.3 зображені його технічні характеристики.

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 – Технічні характеристики

Для досліджень мережевої продуктивності це може бути важливим, коли йдеться про збереження та зчитування великих обсягів даних у стислий проміжок часу.

Наприклад, якщо проводяться тести з передачею відеофайлів чи інших великих об’єктів, швидкісний SSD дасть змогу уникнути додаткового обмеження

з боку системи зберігання. Сама система розрахована на інтенсивні операції вводу-виводу, що дає змогу симулювати сценарії високих навантажень.

Використання такого SSD гарантує мінімальний вплив операцій читання чи запису на диск на кінцевий результат.

Для зв'язку із зовнішньою мережею сервер використовує гігабітний мережевий адаптер, під'єднаний через відповідний інтерфейс материнської плати. Наявність сучасного мережевого чипа дозволяє працювати на швидкостях до 1 Гбіт/с у дротовому середовищі, а також може мати функції апаратного прискорення оброблення пакетів. Таке рішення зменшує затримки при передаванні даних і сприяє отриманню стабільних результатів тестів пропускної здатності. Важливо, що ця конфігурація забезпечує мінімальний рівень накладних витрат на оброблення пакетів, аби результати тестів були якомога точнішими.

Клієнтський бік стенду представлено пристроєм на основі процесора Intel Core i5-8250U з вбудованим мережевим адаптером Realtek Wi-Fi. Цей чип має чотири фізичні ядра і вісім потоків виконання, що хоча й менше за серверну конфігурацію, усе одно забезпечує достатній рівень багатопоточності для типової клієнтської машини. Завдяки своєму енергоефективному дизайну i5-8250U часто зустрічається у ноутбуках чи компактних системах, і його параметр енергоспоживання (TDP) суттєво нижчий за настільні аналоги. Утім, для досліджень мережевої продуктивності це не заважає повноцінно оцінювати швидкість обміну даними, адже базова тактова частота і підтримка динамічного розгону дозволяють пристрою працювати в межах помірних навантажень без «вузького горлечка» на рівні обчислень.

Wi-Fi карта Realtek, інтегрована у клієнта, використовує типові стандарти 802.11n/ac, що дає змогу перевірити роботу з широким діапазоном частот і каналів. Реальний рівень пропускної здатності, який можна отримати в межах практичних тестів, залежить від характеристик цієї карти та її драйверів, а також від сумісності з налаштуваннями роутера. Важливим чинником під час експериментів стають показники стабільності сигналу, особливо коли

випробування проводяться на різній відстані від точки доступу або при наявності завад. Драйвер Realtek для Windows може виявляти особливості маршрутізації пакетів, а також підтримує розширені конфігурації безпеки, що може впливати на продуктивність у зашифрованих з’єднаннях.

Як точка доступу для тестування використовується маршрутизатор TP-Link Archer C20. Цей пристрій підтримує стандарт 802.11ac і пропонує передачу даних у двох діапазонах: 2,4 ГГц та 5 ГГц. Завдяки поєднанню смуг можна досягати сумарної теоретичної пропускної здатності, яка є важливою при випробуваннях високошвидкісних з’єднань. У реальних тестах, особливо коли йдеться про великі файли або потокове відео, різниця між двома діапазонами може бути суттєвою, з огляду на різні рівні завад і прохідності сигналу. TP–Link Archer C20 має підтримку шифрування WPA2, що дозволяє вивчати вплив додаткового навантаження через криптографічні операції на кінцеву швидкість обміну даними.

Маршрутизатор TP-Link Archer C20 представлений на рисунку 3.4.

Рисунок 3.4 – Маршрутизатор TP-Link Archer C20

Сама конструкція TP-Link Archer C20 передбачає кілька зовнішніх антен, які відповідають за стабільний рівень покриття в обох діапазонах. Така апаратна

характеристика допомагає уникати проблем зі зниженням швидкості на більших відстанях. Під час експериментів можна спостерігати деградацію сигналу чи падіння пропускної здатності, щойно витримується певна дистанція від роутера або коли з'являються додаткові джерела електромагнітного зашумлення. Здатність TP-Link Archer C20 витримувати основні навантаження без критичного спаду продуктивності робить його зручним тестовим пристроєм для оцінювання середньостатистичної домашньої чи офісної мережі.

Окремий інтерес становить програмна частина, вбудована в Archer C20. Окрім роботи з двома діапазонами, у ньому передбачені механізми балансування трафіку та інтелектуальна маршрутизація, що впливає на затримку і загальний час відгуку мережі. Крім того, його здатність транслювати внутрішні адреси (NAT) при великій кількості під'єднаних пристроїв має значення для стабільності тестів. Якщо маршрутизатор не оптимізований для високої кількості одночасних з'єднань, під час тестів великим об'ємом даних можлива перевірка механізмів розподілу ресурсів.

Із боку операційної системи для більшості сценаріїв використовується середовище WSL2, запущене під керуванням Windows, а також встановлена дистрибуція Arch Linux із ядром 5.15.167.4-microsoft-standard-WSL2. Така комбінація дає змогу швидко перемикатись між середовищами та перевіряти, як різні мережеві стеки поводяться в тих же самих умовах. WSL2 (Windows Subsystem for Linux 2) забезпечує інтегровану віртуалізацію, що дозволяє Linux-середовищу напряму звертатися до багатьох апаратних ресурсів з незначною додатковою затримкою. Для тестування продуктивності це дає широкий спектр інструментів командного рядка Linux, не покидаючи Windows як основної системи.

У такому середовищі можна гнучко встановлювати й налаштовувати пакунки для аналізу пропускної здатності мережі, затримок, jitter, а також вимірювання продуктивності TCP, UDP чи інших протоколів. Ядро 5.15.167.4-microsoft-standard-WSL2 має оновлені механізми керування потоком, що особливо важливо для масштабних випробувань із багатьма паралельними

потоками. Наприклад, коли потрібно порівняти поведінку кількох сотень одночасних підключень, сучасні виправлення в ядрі дають кращу стабільність і прогнозованість результатів.

Важливо, що для систем на основі Linux доступні інструменти на кшталт iperf, netperf або ж спеціалізовані пакети для вимірювання продуктивності мережі. Усе це можна проводити або безпосередньо в Arch Linux на «рідному» залізі, або ж у WSL2. Порівняння цих умов допомагає з'ясувати, чи вносить рівень віртуалізації чи специфіка ОС додаткову затримку в ході передавання даних. Робота в такому гібридному середовищі дозволяє виконати цілісний аналіз: від базового рівня заліза (через Arch) до користувацьких сценаріїв у Windows, що можуть застосовувати WSL2-інструменти.

Таким чином, комплексна конфігурація серверу на Ryzen 5 5700X3D, клієнта з Intel i5-8250U і Realtek-адаптером, маршрутизатора Archer C20 з підтримкою актуальних Wi-Fi стандартів і програмного середовища WSL2 та Arch Linux з ядром 5.15.167.4-microsoft-standard-WSL2 формує універсальний тестовий стенд для експериментів із високошвидкісною передачею даних по дротових і бездротових каналах. Кожен із цих компонентів робить внесок у загальну картину продуктивності, забезпечуючи широкий спектр сценаріїв: від однієї клієнтської машини з необхідністю швидкісного доступу до серверу, до складніших конфігурацій із паралельними з'єднаннями й керуванням трафіком на різних рівнях мережевої моделі. Усі означені характеристики допомагають створити реалістичні умови для проведення наукових експериментів і надають доволі повну картину про роботу сучасних мережевих технологій у типових випадках.

3.2 Архітектура програми та обрані точки для тестування

У процесі розробки та оптимізації мережевих систем критично важливим є вибір правильного набору технологій та протоколів, які забезпечать ефективну та надійну передачу даних. Цей розділ присвячений детальному аналізу та

обґрунтуванню вибору конкретних протоколів передачі даних, алгоритмів стиснення та форматів серіалізації, які будуть використовуватися в експериментальній частині дослідження. Особлива увага приділяється взаємодії різних компонентів системи та їх впливу на загальну продуктивність. Вибір технологій базується на глибокому аналізі сучасних потреб промислових систем, враховуючи такі критичні фактори як надійність, швидкодія, масштабованість та сумісність з існуючими рішеннями. Кожен компонент системи був ретельно відібраний для забезпечення максимально повного та об'єктивного дослідження.

3.2.1 Обрані протоколи передачі даних

Для тестування було обрано два протоколи передачі даних: HTTP/2 та HTTP/3.

HTTP, або Hypertext Transfer Protocol, становить собою фундаментальний протокол прикладного рівня, який визначає способи обміну даними між клієнтами та веб-серверами. Історично він виник як засіб отримання та передачі гіпертекстових документів у системі World Wide Web, проте згодом став універсальним механізмом комунікації для широкого спектра онлайн сервісів. У початкових версіях кожен ресурс вимагав окремого запиту, а масштабованість ускладнювалася обмеженнями на кількість одночасних з’єднань. Це зумовило необхідність подальшого розвитку протоколу, зокрема переходу на сучасніші редакції.

HTTP/2 був розроблений для покращення ефективності та продуктивності. Мультиплексування у межах одного TCP з’єднання дає змогу передавати кілька потоків даних одночасно, знижуючи затримки завантаження [14]. Формат бінарних фреймів оптимізує процеси розбору та маршрутизації, а стиснення заголовків зменшує накладні витрати на повторювані елементи на кшталт cookies чи User-Agent. Водночас базування HTTP/2 на транспортному протоколі TCP призводить до механізму Head-of-Line Blocking, коли втрата одного пакета тимчасово затримує усі мультиплексовані потоки, що не завжди бажано в

нестабільних мережах. Також важливим елементом HTTP/2 стала можливість пріоритезації різних потоків і технологія Server Push, яка надає серверу змогу надсилати ресурси заздалегідь, проте її реальне використання виявилося доволі обмеженим через специфічне налаштування та потенційні проблеми з кешування.

HTTP/3 постає наступним кроком у розвитку, прагнучи позбутися недоліків, зумовлених застосуванням TCP. Він базується на QUIC, який часто називають «TCP over UDP»: усі механізми надійності, контролю потоку та шифрування залишаються, проте реалізуються поверх UDP, дозволяючи обійти вбудовані обмеження традиційного TCP [15]. Коли стається втрата пакета, вона не зупиняє роботу всіх активних потоків, оскільки QUIC пропонує незалежну реконструкцію для кожного з них. Окрім того, QUIC об’єднує криптографічні функції безпосередньо у транспортний рівень, що скорочує кількість рукостискань і прискорює встановлення з’єднання. Важливим аспектом HTTP/3 є підтримка швидкого (0-RTT) відновлення з’єднання, завдяки чому можлива повторна взаємодія клієнта з сервером без типових для TCP затримок, а також більш гнучке масштабування, що робить HTTP/3 привабливим вибором у складних чи мобільних мережевих середовищах.

Основна різниця між HTTP/2 і HTTP/3 полягає в тому, що перший використовує TCP, який забезпечує гарантовану доставку пакетів, але в разі втрати одного значно сповільнює всю передачу, тоді як другий працює з UDP, допомагаючи уникнути блокування потоків у разі помилок. TCP дає змогу отримати впорядковану передачу в послідовному режимі, натомість UDP дозволяє реалізувати гнучкіші підходи й мінімізувати затримки. У контексті сучасних мереж, включно зі швидкісними оптоволоконними лініями, мобільними мережами та Wi-Fi, гнучкість і адаптація PROQUIC дають помітні переваги, коли важливо зменшити чутливість до втрати пакетів та переспрямовувань.

Для дослідницької чи впроваджувальної роботи порівняння HTTP/2 та HTTP/3 є логічним вибором, адже ці протоколи відображають найпоширеніші

сценарії веб-комунікацій сьогодення. HTTP/2 наразі залишається домінуючим стандартом, який використовується переважною більшістю хмарних сервісів, хостингів і веб-додатків. Водночас впровадження HTTP/3 стрімко зростає, і можна чекати, що у найближчому майбутньому він зацікавить дедалі більшу кількість розробників та користувачів. Завдяки аналізу і тестуванню на HTTP/2 разом із розглядом його еволюції в HTTP/3 можна глибше зрозуміти, як поводяться обидва типи протоколів, і без необхідності створювати власні реалізації поверх TCP чи UDP отримати адекватну картину реальних мережевих умов. Це не лише заощаджує час і ресурси, а й гарантує отримання результатів, релевантних для більшості сучасних систем.

3.2.2 Обрані алгоритми стиснення даних

У межах експерименту було обрано кілька типових алгоритмів стиснення, аби охопити весь спектр рішень – від найбільш традиційних до сучасних високоефективних. Зокрема, розглядалися GZIP, Zstandard (zstd), LZ4, Brotli, Deflate, LZO та проста власна реалізація RLE. Крім того, окремо виконувався сценарій без стиснення (none), що дає змогу порівняти швидкість чистої передачі даних із результатами кожного з алгоритмів. Нижче наведено детальніший опис кожного рішення і пояснення обраних рівнів стиснення, а також згадані бібліотеки, за допомогою яких було реалізовано кожен із методів.

GZIP є класичним алгоритмом на основі LZ77 та Huffman-кодування, який застосовується у величезній кількості систем та інструментів. Для його реалізації використано стандартну бібліотеку Go «compress/gzip». У рамках експерименту виділено рівні 6 та 9 стиснення. Рівень 1 мінімально навантажує процесор і швидко обробляє файли, проте не дає значної економії місця. Рівень 9, навпаки, забезпечує найглибше стиснення, але вимагає суттєвої обчислювальної потужності. Рівень 6 – це проміжний варіант, часто рекомендований як оптимальний компроміс для веб-сервісів та застосунків, що потребують задовільної компресії без істотних затримок [10].

На рисунку 3.5 можна побачити реалізацію GZIP

```go
// Helper functions for compression/decompression
func compressWithWriter(in []byte, factory writerFactory) ([]byte, error) {
	buf := bytebufferpool.Get()
	defer bytebufferpool.Put(buf)

	w, err := factory(buf)
	if err != nil {
		return nil, err
	}

	if _, err := w.Write(in); err != nil {
		w.Close()
		return nil, err
	}

	if err := w.Close(); err != nil {
		return nil, err
	}

	return append([]byte(nil), buf.Bytes()...), nil
}

func decompressWithReader(in []byte, factory readerFactory) ([]byte, error) {
	r, err := factory(bytes.NewReader(in))
	if err != nil {
		return nil, err
	}
	defer r.Close()

	return io.ReadAll(r)
}

func compressGzip(in []byte, level int) ([]byte, error) {
	level = normalizeGzipLevel(level)
	return compressWithWriter(in, func(w io.Writer) (io.WriteCloser, error) {
		return gzip.NewWriterLevel(w, level)
	})
}

func normalizeGzipLevel(level int) int {
	if level < gzip.HuffmanOnly || level > gzip.BestCompression {
		return gzip.DefaultCompression
	}
	return level
}
```

Рисунок 3.5 – Реалізація GZIP

Zstandard (zstd) було створено головно у Facebook для швидкісного й ефективного стиснення великих обсягів даних.

У цьому проєкті застосовується бібліотека «github.com/DataDog/zstd» із дуже гнучкими можливостями налаштування.

Рівні 4 і 9 дібрано для ілюстрації того, як змінюються вимоги до обчислень і ступінь компресії: при рівні 1 час обробки майже мінімальний, а вихідний файл зменшується несуттєво; натомість на рівні 9 алгоритм досягає помітної економії місця, однак це може збільшувати затримку. Рівень 4 зазвичай точка «золотої середини» для багатьох сценаріїв, де й швидкість, і стискання однаково важливі [9].

На рисунку 3.6 наведено реалізацію Zstandard.

```
func compressZstd(in []byte, level int) ([]byte, error) {
	return compressWithWriter(in, func(w io.Writer) (io.WriteCloser, error) {
		return zstd.NewWriterLevelDict(w, level, nil), nil
	})
}

func decompressZstd(in []byte) ([]byte, error) {
	return decompressWithReader(in, func(r io.Reader) (io.ReadCloser, error) {
		return zstd.NewReaderDict(r, nil), nil
	})
}
```

Рисунок 3.6 – Реалізація Zstandard

LZ4 відомий своїм фокусом на максимальну продуктивність при стисненні та, особливо, при розпакуванні.

Для реалізації застосовується «github.com/pierrec/lz4/v4».

Для тестів використано рівні 4 і 9, що концептуально схожі на решту алгоритмів: рівень 1 найшвидший, але дає найнезначніше стискання, рівень 9 – повільніший, але компактніший вихідний результат.

Якщо дані потребують миттєвої доставки, LZ4 часто виявляється дуже вдалим вибором, адже має менше затримок, ніж інші варіанти з поверхневою компресією [11].

На рисунку 3.7 наведено реалізацію LZ4.

```go
func compressLz4(in []byte, level int) ([]byte, error) {
	return compressWithWriter(in, func(w io.Writer) (io.WriteCloser, error) {
		writer := lz4.NewWriter(w)
		if err := writer.Apply(normalizeLz4Level(level)); err != nil {
			return nil, err
		}
		return writer, nil
	})
}

func decompressLz4(in []byte) ([]byte, error) {
	return decompressWithReader(in, func(r io.Reader) (io.ReadCloser, error) {
		return io.NopCloser(lz4.NewReader(r)), nil
	})
}
```

Рисунок 3.7 – Реалізація LZ4

Brotli – сучасний алгоритм від Google, який часто використовують для стиснення веб-контенту (HTML, CSS, JavaScript). У роботі взято бібліотеку «github.com/andybalholm/brotli». Шкала рівнів тягнеться від 0 до 11, тож було включено щаблі 6 і 11. Перший рівень майже не знижує пропускну спроможність системи, проте й не дає надприродного результату стосовно стиснення. Натомість рівень 11 робить дані максимально компактними, проте процес стає вимогливим до процесора. Середні рівні (зокрема 4-6) зазвичай обирають для статичних ресурсів, які передаються часто, але небажано вносити великі затримки [23].

На рисунку 3.8 наведено реалізацію Brotli.

```go
func compressBrotli(in []byte, level int) ([]byte, error) {
	level = normalizeBrotliLevel(level)
	return compressWithWriter(in, func(w io.Writer) (io.WriteCloser, error) {
		return brotli.NewWriterLevel(w, level), nil
	})
}

func decompressBrotli(in []byte) ([]byte, error) {
	return decompressWithReader(in, func(r io.Reader) (io.ReadCloser, error) {
		return io.NopCloser(brotli.NewReader(r)), nil
	})
}
```

Рисунок 3.8 – Реалізація Brotli

Deflate – стандарт, що лежить в основі формату ZIP і є одним із найпоширеніших методів архівації. За допомогою Go він реалізується бібліотекою «compress/flate». Параметри 6 та 9 так само емулюють різні стратегії стиснення: швидке, збалансоване та глибоке. Deflate історично популярний для зменшення обсягу текстових файлів і масивів, а також широко відомий завдяки сумісності з ZIP-архівами та GZIP [23].

На рисунку 3.9 наведено реалізацію Deflate.

```
func normalizeDeflateLevel(level int) int {
	if level < flate.HuffmanOnly || level > flate.BestCompression {
		return flate.DefaultCompression
	}
	return level
}

func compressDeflate(in []byte, level int) ([]byte, error) {
	level = normalizeDeflateLevel(level)
	return compressWithWriter(in, func(w io.Writer) (io.WriteCloser, error) {
		return flate.NewWriter(w, level)
	})
}

func decompressDeflate(in []byte) ([]byte, error) {
	return decompressWithReader(in, func(r io.Reader) (io.ReadCloser, error) {
		return flate.NewReader(r), nil
	})
}
```

Рисунок 3.9 – Реалізація Deflate

LZO – алгоритм, що має репутацію швидкого й водночас достатньо ефективного рішення, хоча й менш поширеного у мейнстрим розробці порівняно з іншими форматами [24]. Використовується бібліотека «github.com/cyberdelia/lzo». Для експерименту виділено рівні 4 та 9, проте варто відзначити, що в LZO немає такої гнучкої шкали, як у GZIP чи Brotli – ці значення більше слугують для зіставлення з іншими алгоритмами, а не відображають реальну «глибину» компресії.

На рисунку 3.10 наведено реалізацію LZO.

```go
func normalizeLzoLevel(level int) int {
	if level < 1 || level > 9 {
		return 1
	}
	return level
}

func compressLzo(in []byte, level int) ([]byte, error) {
	level = normalizeLzoLevel(level)
	return compressWithWriter(in, func(w io.Writer) (io.WriteCloser, error) {
		return lzo.NewWriterLevel(w, level)
	})
}

func decompressLzo(in []byte) ([]byte, error) {
	return decompressWithReader(in, func(r io.Reader) (io.ReadCloser, error) {
		return lzo.NewReader(r)
	})
}
```

Рисунок 3.10 – Реалізація LZO

3.2.3 Обрані формати серіалізації даних

У межах дослідження було зосереджено увагу на трьох ключових форматах для передачі даних: JSON, MessagePack і Protobuf. Вони суттєво різняться за підходом до серіалізації та мають власний набір переваг і недоліків. Крім того, додатково було протестовано сценарій, коли файли (наприклад, зображення чи js-бандли з популярних npm-пакетів) передаються без жодних перетворень. Такий підхід дає змогу зрозуміти «чисту» продуктивність стиснення, не обтяжену вартістю кодування.

JSON, або JavaScript Object Notation, широко використовується у веб-розробці завдяки текстовому формату та подібності до синтаксису JavaScript [16]. Він відзначається простотою впровадження та читання для людини, що робить його універсальним рішенням і фактичним стандартом у багатьох прикладних програмах. Водночас текстова природа JSON може суттєво збільшувати розмір переданих даних і підвищувати навантаження на процесор під час парсингу, особливо при великих обсягах.

MessagePack пропонує двійковий формат із метою досягнення більшої компактності й швидшого процесу серіалізації та десерілізації. За своєю логікою він близький до JSON, однак завдяки мінімалізму у форматуванні дає змогу значно зменшити загальний розмір даних. Важливо, що застосування MessagePack не потребує додаткових описів схем і легко інтегрується у більшість проектів. Така гнучкість особливо цінується, коли структура даних частково або повністю змінюється на етапі розробки [18].

Protobuf, або Protocol Buffers, розроблений компанією Google. Це потужний синтаксис для опису структурованих даних із суворим визначенням типів у файлах .proto. Завдяки бінарній формі повідомлень він забезпечує чудові показники ефективності й швидкості обробки. Але така найвища оптимізація передбачає додаткові вимоги до процесу розробки: необхідно вести й оновлювати файли зі схемами, генерувати відповідний код, підтримувати його сумісність із різними версіями у разі оновлень або розширень [17].

Паралельно з тестами цих трьох форматів аналізувалася й передача даних без пакування взагалі, зокрема передавалися зображення та js-бандли (набори коду). Такий підхід репрезентує «розору» перевірку пропускної здатності, коли жодна серіалізація не впливає на результати. Порівнюючи JSON, Protobuf і MessagePack, можна відзначити, що перший вирізняється простотою впровадження, другий – максимальною оптимізацією і контролем через схеми, а третій демонструє вдале поєднання компактності та невисоких вимог до інтеграції. На мою думку, найбільш компромісним форматом виступає саме MessagePack, адже він дає змогу помітно зменшити обсяг переданих даних, не ускладнюючи процес розробки схем чи генерації коду. Такий вибір доречний у більшості сценаріїв, де водночас важливі висока продуктивність та гнучкість у зміні структури даних.

Реалізацію серіалізації та десерілізації наведено на рисунках 3.11 та 3.12 відповідно.

```go
func Serialize(p *Payload, df DataFormat) ([]byte, error) {
	df = DataFormat(strings.ToLower(string(df)))
	df = DataFormat(strings.TrimSpace(string(df)))
	switch df {
	case JSONFormat:
		return json.Marshal(p)
	case MsgPackFormat:
		return msgpack.Marshal(p)
	case ProtobufFormat:
		// Convert our Payload struct to the generated PayloadProto struct
		pb := &PayloadProto{
			Iteration:           int32(p.Iteration),
			ClientSendTimeUnix:  p.ClientSendTimeUnix,
			ClientSendTimeNanos: p.ClientSendTimeNanos,
			FileName:            p.FileName,
			FileSize:            int32(p.FileSize),
			Note:                p.Note,
			Data:                p.Data,
		}
		return proto.Marshal(pb)
	default:
		panic("unknown format: " + df)
	}
}
```

Рисунок 3.11 – Серіалізація

```go
func Deserialize(data []byte, df DataFormat) (*Payload, error) {
	df = DataFormat(strings.ToLower(string(df)))
	df = DataFormat(strings.TrimSpace(string(df)))

	switch df {
	case JSONFormat:
		var p Payload
		if err := json.Unmarshal(data, &p); err != nil {
			return nil, err
		}
		return &p, nil

	case MsgPackFormat:
		var p Payload
		if err := msgpack.Unmarshal(data, &p); err != nil {
			return nil, err
		}
		return &p, nil

	case ProtobufFormat:
		var pb PayloadProto
		if err := proto.Unmarshal(data, &pb); err != nil {
			return nil, err
		}
		// Convert from the proto struct to our Payload struct
		return &Payload{
			Iteration:           int(pb.GetIteration()),
			ClientSendTimeUnix:  pb.GetClientSendTimeUnix(),
			ClientSendTimeNanos: pb.GetClientSendTimeNanos(),
			FileName:            pb.GetFileName(),
			FileSize:            int(pb.GetFileSize()),
			Note:                pb.GetNote(),
		}, nil

	default:
		panic("unknown format: " + df)
	}
}
```

Рисунок 3.12 – Десеріалізація

3.3 Ключова логіка та метрики

Для реалізації програми був написаний пакет logic, в якому описана більша частина операцій які буде робити тест. На рисунках вказаних раніше вже можна було побачити логіку стиснення та пакування у формат. Тепер роздивимось логіку метрик.

З метрик які збирають сервер та клієнт було виділено наступні:

– час запису, у наносекундах;
– час читання, у наносекундах;
– час перетворення у формат, у наносекундах;
– час перетворення з формату, у наносекундах;
– час стиснення, у наносекундах;
– час розтиснення, у наносекундах;
– використання пам’яті, у байтах;
– використання процесору, у відсотках від всього процесору.

Для цього був написаний metrics.go, структури який він зберігає можна побачити на рисунку 3.13.

```
type Metrics struct {
    // Separate mutexes for different metric types to reduce contention
    timingMu   sync.Mutex
    resourceMu sync.Mutex

    // Basic counters - atomic, no mutex needed
    TotalRequests   atomic.Uint64
    serverBytesRead atomic.Uint64

    // Distribution metrics
    writeTime      []float64
    marshalTime    []float64
    readTime       []float64
    compressTime   []float64
    decompressTime []float64
    unmarshalTime  []float64
    compressedSize []float64
    decodedSize    []float64

    // Resource usage samples
    memoryUsage []float64
    cpuUsage    []float64
}
```

Рисунок 3.13 – Структури метрик

Цей код використовує сервер та клієнт, додаючи нові дані на кожен запит, та оновлюючи дані про процесор та оперативну пам'ять кожні 250 мілісекунд.

Для реалізації HTTP/2 серверу використовується пакет net/http, в той час як для реалізації HTTP/3 був використаний пакет github.com/quic-go/quic-go.

Цю реалізацію можна побачити на рисунку 3.14.

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

	var transport http.RoundTripper
	if c.HTTPVer == HTTPVersion3 {
		transport = &http3.Transport{
			TLSClientConfig: loadTLSConfig(),
			QUICConfig: &quic.Config{
				KeepAlivePeriod: 30 * time.Second,
			},
		}
	} else {
		transport = &http.Transport{
			Proxy:                 http.ProxyFromEnvironment,
			ForceAttemptHTTP2:     c.HTTPVer == HTTPVersion2,
			MaxIdleConns:          100,
			IdleConnTimeout:       90 * time.Second,
			TLSHandshakeTimeout:   10 * time.Second,
			ExpectContinueTimeout: 1 * time.Second,
			TLSClientConfig:       loadTLSConfig(),
		}
	}

	em, err := NewNetworkEmulator(NetworkEmulatorConfig{
		Wrapped:     transport,
		MinLatency:  c.MinLatency,
		MaxLatency:  c.MaxLatency,
		FailureRate: c.FailureRate,
		Bandwidth:   c.Bandwidth,
	})
	if err != nil {
		return nil, fmt.Errorf("create network emulator: %w", err)
	}

	return &http.Client{Transport: em}, nil
}
```

Рисунок 3.14 – Реалізація клієнту для HTTP/2 та HTTP/3

На рисунку 3.14 також можна побачити NetworkEmulator. Це кастомна структура, яка побудована поверх інтерфейсів бібліотеки net/http для створення затримок чи пропускної здібності клієнту. Його реалізацію можна побачити на рисунку 3.15.

```go
func (ne *NetworkEmulator) RoundTrip(req *http.Request) (*http.Response, error) {
	// Simulate random network failure
	ne.randMu.Lock()
	shouldFail := ne.rand.Float64() < ne.failureRate
	latencyDelta := int64(0)
	if ne.maxLatency > ne.minLatency {
		latencyDelta = ne.rand.Int63n(int64(ne.maxLatency - ne.minLatency))
	}
	ne.randMu.Unlock()

	if shouldFail {
		return nil, errors.New("simulated network failure")
	}

	// Add artificial latency
	if ne.minLatency > 0 || ne.maxLatency > 0 {
		latency := ne.minLatency + time.Duration(latencyDelta)
		time.Sleep(latency)
	}

	// Wrap the body with a rate-limited reader if bandwidth is specified
	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
	}

	// Also rate-limit the response body if bandwidth is specified
	if ne.bandwidth > 0 && resp.Body != nil {
		resp.Body = newRateLimitedReader(resp.Body, ne.bandwidth)
	}

	return resp, nil
}
```

Рисунок 3.15 – Реалізація NetworkEmulator

NetworkEmulator для емуляцій затримок використовує time.Sleep(), а для реалізації ліміту пропускної здібності використовує rateLimitedReader. Реалізацію rateLimitedReader можна побачити на рисунку 3.15.

```go
func newRateLimitedReader(reader io.ReadCloser, bandwidth int64) *rateLimitedReader {
	return &rateLimitedReader{
		reader:    reader,
		bandwidth: bandwidth,
		startTime: 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 {
		now := time.Now()
		r.totalBytes += int64(n)

		// Calculate how long all reads should have taken based on bandwidth
		expectedDuration := time.Duration(float64(r.totalBytes) / float64(r.bandwidth) * float64(time.Second))
		actualDuration := now.Sub(r.startTime)

		// Sleep if we're reading too fast
		if actualDuration < expectedDuration {
			time.Sleep(expectedDuration - actualDuration)
		}
	}

	return n, err
}
```

Рисунок 3.16 – Реалізація rateLimitedReader

Цей набір коду дає відправляти повідомлення до серверу за допомогою HTTP/2 або HTTP/3, та емулювати умови реальної мережі.

Реалізацію сервера можна побачити на рисунку 3.17. Цей сервер підтримує як HTTP/2, так і HTTP/3 в залежності від встановленої тестом конфігурації.

```go
func newServer(cfg config) (*server, error) {
	if err := cfg.validate(); err != nil {
		return nil, fmt.Errorf("invalid configuration: %w", err)
	}

	s := &server{
		cfg:          cfg,
		compSettings: logic.ExtractCompressionFromString(cfg.compression),
		startTime:    time.Now(),
		shutdownChan: make(chan struct{}),
	}

	mux := http.NewServeMux()
	mux.HandleFunc("/upload", s.handleUpload)
	mux.HandleFunc("/exit", s.handleExit)

	if cfg.httpVer == 3 {
		s.httpServer = &http3.Server{
			Addr:    fmt.Sprintf(":%d", cfg.port),
			Handler: mux,
			Port:    cfg.port,
			TLSConfig: &tls.Config{
				MinVersion: tls.VersionTLS13,
			},
		}
	} else {
		s.httpServer = &http.Server{
			Addr:    fmt.Sprintf(":%d", cfg.port),
			Handler: mux,
		}
	}

	return s, nil
}
```

Рисунок 3.17 – Реалізація HTTP/2 та HTTP/3 серверів

Для удобного тестування було написано Bash-скрипт та контроль-сервер.

Bash-скрипт запускає всі варіації тесту, а контроль сервер стартує потрібний для тесту сервер. Контроль сервер також завершує виконання серверу після того як завершився тест, за командою клієнту, що надає змогу безперервно робити тестування не використовуючи зайві ресурси серверного обладнання.

Логіку контроль-серверу можна побачити на рисунку 3.18.

```go
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 logic.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 v := server.Load(); v != nil {
		if err := v.Process.Signal(syscall.SIGTERM); err != nil {
			slog.LogAttrs(r.Context(), slog.LevelError, "Failed to kill server", slog.String("err", err.Error()))
		}
	}
}

func launchServer(ctx context.Context, req logic.LaunchCommand) {
	slog.LogAttrs(ctx, slog.LevelInfo, "Launching server",
		slog.String("name", req.ExecutableName),
		slog.Any("args", req.Args),
	)
	cmd := exec.Command(req.ExecutableName, req.Args...)
	cmd.Stdout = os.Stdout
	cmd.Stderr = os.Stderr
	if err := cmd.Start(); err != nil {
		slog.LogAttrs(ctx, slog.LevelError, "Failed to start server", slog.String("err", err.Error()))
		return
	}
	server.Store(cmd)
	slog.LogAttrs(ctx, slog.LevelInfo, "Server launched", slog.String("name", req.ExecutableName))

	if err := cmd.Wait(); err != nil {
		slog.LogAttrs(ctx, slog.LevelError, "Server failed", slog.String("err", err.Error()))
	}

	slog.LogAttrs(ctx, slog.LevelInfo, "Server exited", slog.String("name", req.ExecutableName))
}
```

Рисунок 3.18 – Логіка контроль-серверу

Код, який виконує клієнт для старту та завершення серверу показано на рисунку 3.19.

```go
func askToLaunch(ctx context.Context) error {
	req := logic.LaunchCommand{
		ExecutableName: "./bin/http_s",
		Args: []string{
			"-port", "8000",
			"-compression", cfg.compression,
			"-format", cfg.formatFlag,
			"-output-dir", cfg.outputDir,
			"-http-ver", strconv.Itoa(cfg.httpVer),
		},
	}

	b, err := json.Marshal(req)
	if err != nil {
		return fmt.Errorf("marshal launch command: %w", err)
	}

	slog.Info("Sending launch request")

	if err := logic.PostData(ctx, &logic.PostConfig{
		URL:         cfg.controlURL + "/launch",
		Data:        b,
		ContentType: "application/json",
	}, http.DefaultClient); err != nil {
		return fmt.Errorf("post launch request: %w", err)
	}

	slog.Info("Launch request sent")
	return nil
}
```

Рисунок 3.19 – Код старту та завершення серверу зі сторони клієнта

Основну логіку Bash-скрипту який виконує тест можна побачити на рисунку 3.20. Цей код по черзі запускає всі можливі комбінації тестів зі встановлених параметрів конфігурації у самому скрипті. При цьому, цей скрипт також пише в консоль статус тесту, наявність помилок та який по черзі цей тест на даний момент. Завдяки цьому скрипту було автоматизоване виконання всіх можливих конфігурацій, які були обрані для тесту. Ця кількість становить 160 різних конфігурацій.

```
TOTAL_SERVERS=$(( ${#CLIENT_TYPES[@]} * ${#DATA_FORMATS[@]} * ${#COMPRESSION_OPTIONS[@]} ))
echo "Starting client tests..."
echo "[$(date)] Starting test suite - Total tests: $TOTAL_SERVERS"
for client_spec in "${CLIENT_TYPES[@]}"; do
    IFS='|' read -r client_type client_args <<< "$client_spec"

    client_type="${client_type## }"
    client_type="${client_type%% }"
    client_args="${client_args## }"
    client_args="${client_args%% }"

    for data_format in "${DATA_FORMATS[@]}"; do
        for compression in "${COMPRESSION_OPTIONS[@]}"; do
            PORT=$((BASE_PORT + SERVER_COUNT))

            echo
            echo "================================================================="
            echo " Running test $((SERVER_COUNT + 1)) of $TOTAL_SERVERS"
            echo "   Client    : $client_type"
            echo "   Client Args: $client_args"
            echo "   Format    : $data_format"
            echo "   Compression: $compression"
            echo "================================================================="
            echo

            if ! run_client "$client_type" "$data_format" "$compression" "$client_args"; then
                echo "[FAILED] Test failed"
                ((FAILED_TESTS++))
                continue
            fi

            echo "[SUCCESS] Test completed successfully"
            sleep 5

            SERVER_COUNT=$((SERVER_COUNT + 1)) || true
            echo "DEBUG: Current test count: $SERVER_COUNT"
        done
    done
done
```

Рисунок 3.20 – Логіка Bash-скрипту

Після цього за допомогою pandas, matplotlib, seaborn та numpy буде виконано побудування наступних графіків:

– порівняння затримки;

– порівняння коефіцієнту стиснення;

– порівняння використаних ресурсів системи;

– порівняння результатів відносно комбінації з HTTP/2, JSON, GZIP.

Останній пункт є найцікавішим з всіх. Комбінація з HTTP/2, JSON, GZIP є на даний момент стандартом, який використовують майже всі вебсервери, а саме тому комбінація відносно якої потрібно отримати краще результати.

Програма на Python також записує CSV-файл (Comma separated values), з якого можна буде побудувати таблички для порівняння та інші графіки.

3.4 Налаштування для тесту

Як частину випробувань було виокремлено ряд параметрів, що моделюють поведінку мережі в лабораторних умовах.

MinLatency було встановлено на 15 мс. Таке мінімальне значення дозволяє відобразити затримку, яка часто трапляється навіть у вельми стабільних каналах зв'язку (приміром, у локальних мережах або при підключеннях на короткі відстані). Обрання 15 мс як початкової точки дозволяє виявити, наскільки чутливими є алгоритми стиснення до невеликої, але присутньої у реальних умовах затримки.

MaxLatency було встановлено на 75 мс. У межах експерименту це верхня межа затримки, що наслідує можливі «піки» у більш складних або завантажених мережевих середовищах (наприклад, при підключеннях через кілька проміжних вузлів або за наявності просідань у канал). Межа у 75 мс дає змогу перевірити, як поведуть себе алгоритми в ситуації погіршених мережевих умов.

FailureRate було встановлено 0. На етапі основного тестування вирішено відмовитися від ін'єкції помилок чи пакетних втрат. Мета полягає в тому, щоб спочатку оцінити базову ефективність алгоритмів стиснення в умовно «ідеальній» мережі без додаткових проблем на кшталт втрат пакетів або перезапитів. Якщо згодом постане необхідність дослідити толерантність до помилок, рівень FailureRate можна буде збільшити, і повторити тести задля порівняння результатів.

Bandwidth було встановлено 10 МБ/с. Така пропускна здатність відображає доволі високу швидкість передачі. Встановлення Bandwidth на рівні 10 МБ/с дозволяє з'ясувати, наскільки саме у швидкій мережі оперативність та ефективність кожного алгоритму впливають на загальний час доставки даних.

Разом із тим, щоб уникнути надмірно ідеалізованих сценаріїв (із фактично необмеженою швидкістю), це значення залишається достатньо реалістичним для більшості промислових рішень.

Для симулювання великої кількості запитів було встановлене значення 256 для Concurrency. Мова Go ідеально підходить для одночасного виконання великої кількості запитів. Це дає просимулювати велике навантаження.

Кожен з тестів, який передає файл – в сумі передає приблизно 1000 файлів, розміром від 1 МБ до 5 МБ. Всі файли – різного формату, і включають до себе: картинки, відео, текстові файли, JS-бандли популярних бібліотек з npm та ін.

Кожен з тестів, який передає структуровані дані – передає 100000 запитів, кожен з яких має структуровані дані та додаткові байти, розміром від 1 кБ до 500 кБ, що дозволяє протестувати ефективність стиснення.

3.5 Проведення тесту та результати

Виконання тестування за допомогою обраних клієнту та сервера зайняло в сумі 43 хвилин та було розроблено 160 файлів з метриками які були описані раніше. Така висока кількість результатів потребує подальше фільтрування для того, щоб зробити якісні виводи, тому в наступних етапах планується поступове видалення результатів що не влаштовують нас, поки не буде знайдено результати що можуть модернізувати передачу даних в промислових мережах. Результат виконання можна побачити на рисунку 3.21.

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

Рисунок 3.21 – Результат виконання тесту

3.5.1 Проведення тестування часу стиснення

З точки зору практичної реалізації систем безпровідної передачі даних у промислових мережах, першочерговим етапом аналізу ефективності алгоритмів стиснення доцільно визначити час, необхідний для виконання операції стиснення.

Такий підхід дозволить на початкових етапах дослідження відсіяти ті алгоритми, які, незважаючи на потенційно високий ступінь стиснення, виявляються неприйнятними з точки зору часових затримок, що вносяться в процес обробки даних.

Для пристроїв, що функціонують в режимі реального часу та повинні забезпечувати безперервну передачу інформації, надмірно тривалий час стиснення може стати критичним фактором, що обмежує їхню застосовність у промислових середовищах.

Таким чином, аналіз часових характеристик стиснення є важливим етапом у відборі оптимальних алгоритмів, здатних забезпечити баланс між ступенем стиснення та швидкістю обробки даних у комп'ютеризованих системах безпровідної передачі інформації.

На рисунку 3.22 відображено графік часу стиснення.

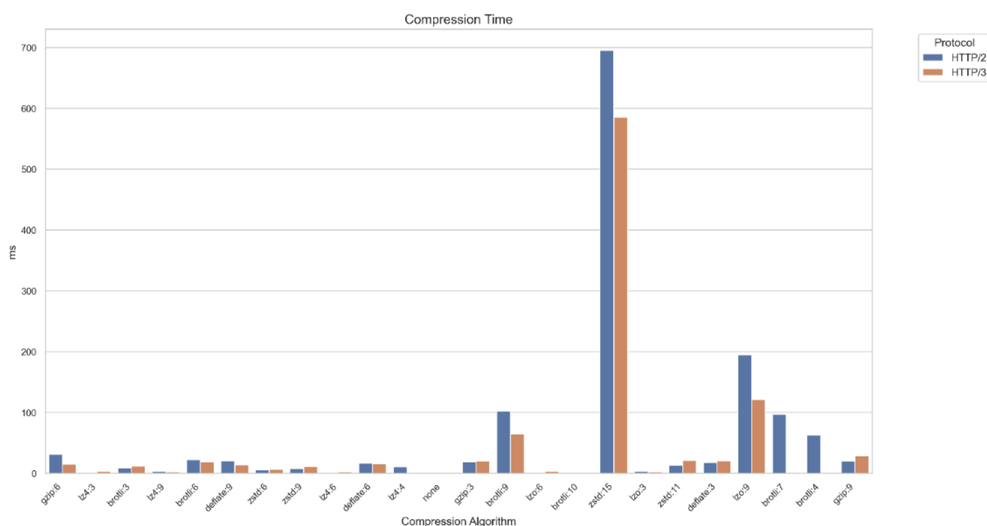

Рисунок 3.22 – Порівняння результатів часу стиснення

Аналізуючи діаграму на рисунку 3.22, можна зробити певні висновки щодо ефективності різних алгоритмів стиснення в контексті протоколів HTTP/2 та HTTP/3. Діаграма наочно демонструє час, витрачений на стиснення даних (у мілісекундах) для різноманітних алгоритмів, що є важливим параметром при оцінці продуктивності систем передачі даних, особливо в умовах обмежених ресурсів промислових мереж.

Загальний огляд діаграми показує значну варіативність часу стиснення залежно від обраного алгоритму. Деякі алгоритми, такі як lz4:3, lz4:9, deflate:6, lz4:6, deflate:9, демонструють надзвичайно низький час стиснення, що свідчить про їхню високу швидкодію. На противагу їм, алгоритми zstd:15 та brotli:10 вимагають значно більшого часу на стиснення, що може вказувати на їхню орієнтацію на максимальний ступінь стиснення за рахунок швидкості обробки.

Особливу увагу привертає алгоритм zstd:15, який показує найвищий час стиснення серед усіх протестованих алгоритмів, особливо при використанні протоколу HTTP/2. Проте, варто зазначити, що високий час стиснення часто корелює з вищим ступенем стиснення, що може бути важливим у сценаріях, де пріоритетом є економія пропускної здатності каналу зв'язку. З іншого боку, алгоритми lz4 та deflate демонструють стабільно низький час стиснення, що робить їх привабливими для застосувань, де швидкість обробки даних є критичною.

Виходячи з цього, аналіз алгоритмів zstd:15, brotli:9, lzo:9, brotli:7 можна припинити ще на цьому кроку, так як вони не підходять за вимогами по часу. Враховуючи дуже велику кількість результатів (160), це дозволить зробити аналіз набагато легше в наступних кроках.

На рисунку 3.23 представлено оновлену діаграму часу стиснення, що включає лише алгоритми, які продемонстрували прийнятні часові показники на попередньому етапі аналізу.

Оновлений графік без цих алгоритмів можна побачити на рисунку 3.23.

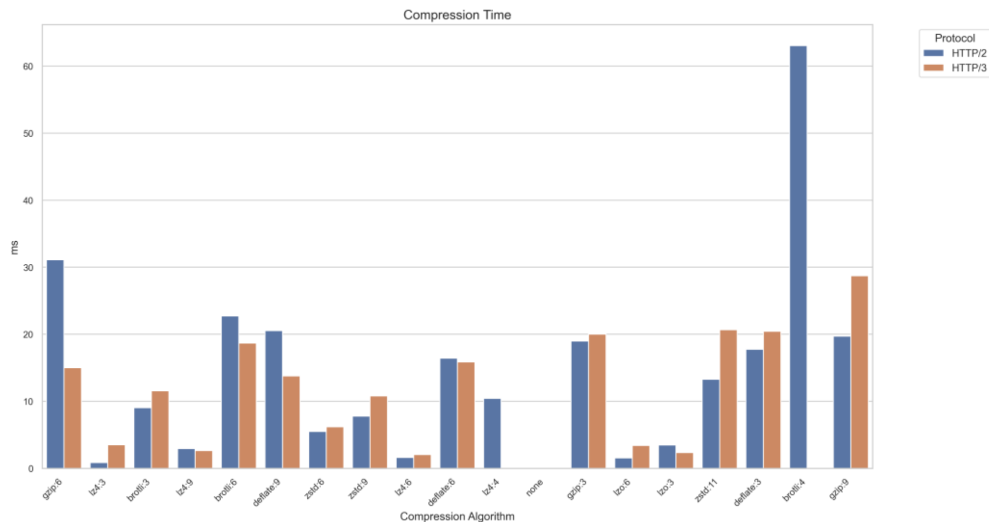

Рисунок 3.23 – Оновлений графік часу стиснення

Розглядаючи цей графік, можна відзначити, що алгоритми сімейства lz4, а саме lz4:3, lz4:6 та lz4:9, показують найменший час стиснення серед усіх представлених, часто не перевищуючи декількох мілісекунд. Алгоритми deflate:3, deflate:6 та deflate:9 також демонструють досить низький час стиснення, хоча дещо вищий, ніж у lz4. Серед алгоритмів gzip, варіант gzip:3 виділяється відносно низьким часом стиснення, тоді як gzip:6 та gzip:9 вимагають дещо більше часу на обробку.

Варто відзначити і zstd, який у всіх своїх варіантах був або швидше або рівноцінний gzip:6.

Важливо відзначити, що для більшості алгоритмів, представлених на рисунку 3.23, протокол HTTP/3 демонструє або порівнянний, або дещо менший час стиснення у порівнянні з HTTP/2, що може свідчити про певні оптимізації в реалізації протоколу HTTP/3 для швидкої обробки даних або недостатньо точні результати тестування, через те, що розмір запиту ранжувався він 1 кілобайту до 500 кілобайтів, хоч і використовувався достатньо надійний алгоритм рандомізації.

Дивлячись на результати brotli:4 можна побачити статистичну аномалію, так як він зайняв довше часу ніж більш важкий його варіант brotli:4 чи дуже схожий на нього brotli:3. Через це було вирішено прибрати його з порівняння.

Результат на якому видно алгоритми, що задовільнюють наші вимоги по часу можна побачити на рисунку 3.24.

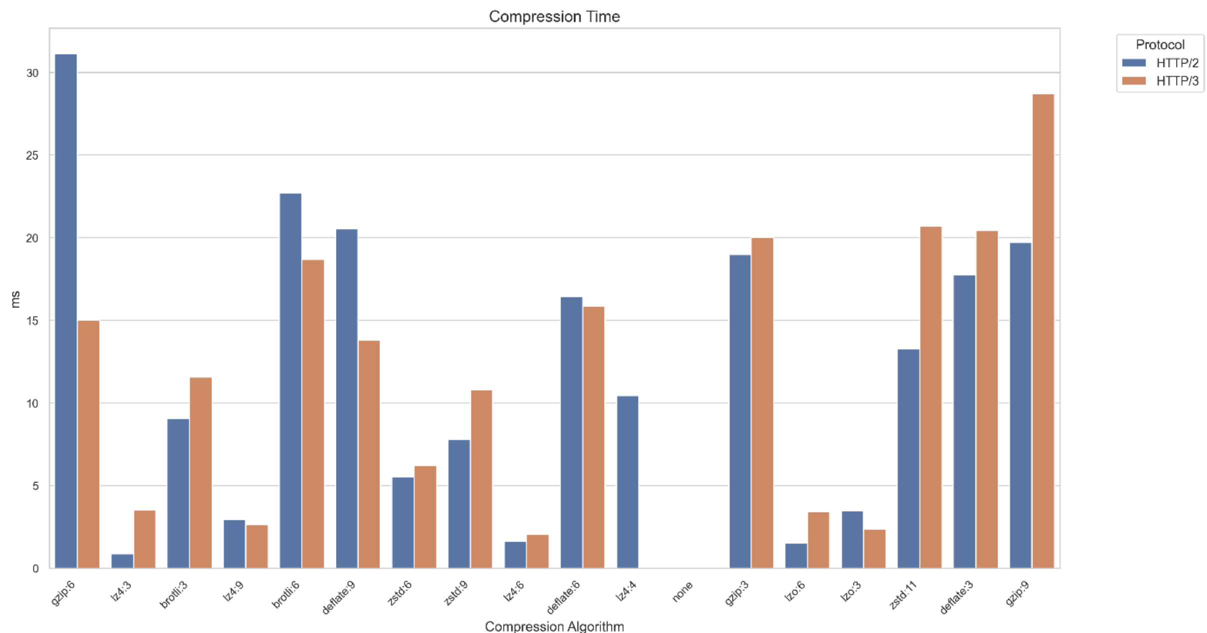

Рисунок 3.24 – Оновлений графік часу стиснення

3.5.2 Проведення тестування часу розтиснення

Після детального розгляду часу стиснення та відбору алгоритмів, що продемонстрували прийнятні показники швидкості, наступним логічним кроком у нашому дослідженні є аналіз часу розтиснення. Операція розтиснення, будучи зворотною до стиснення, відіграє не менш важливу роль у загальній продуктивності системи передачі даних, особливо в контексті промислових мереж, де дані можуть оброблятися на різних етапах та різними пристроями.

Швидкість розтиснення безпосередньо впливає на затримки при отриманні та обробці інформації на приймальній стороні, що може бути критичним для застосувань реального часу та систем управління в промисловості.

Таким чином, для повної оцінки ефективності обраних алгоритмів стиснення необхідно проаналізувати їхні часові характеристики не лише для операції стиснення, але й для операції розтиснення. Надалі ми перейдемо до

розгляду метрик, що характеризують час розтиснення, та порівняємо продуктивність різних алгоритмів у цьому аспекті.

На рисунку 3.25 відображений графік часу розтиснення.

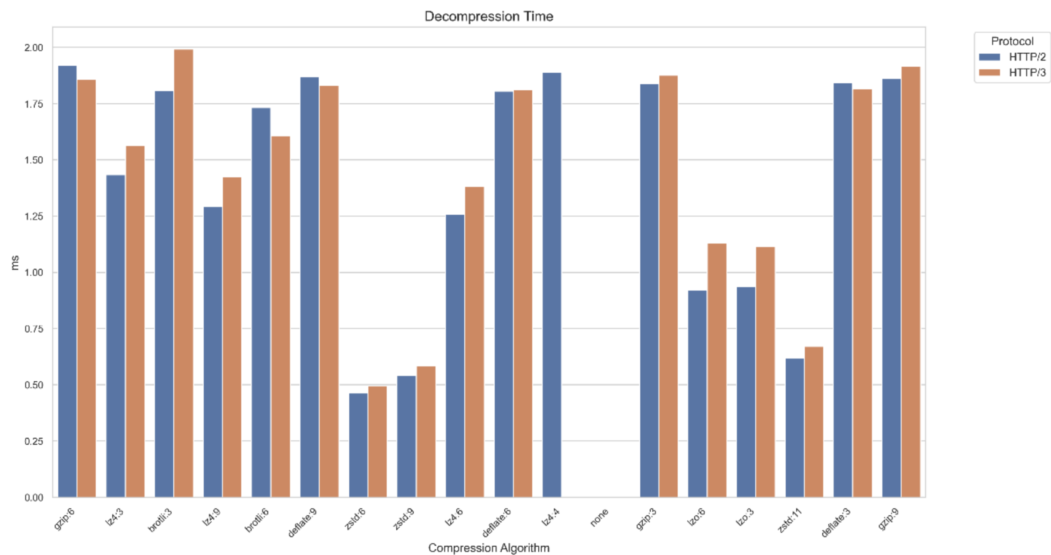

Рисунок 3.25 – Порівняння часу розтиснення

Аналізуючи діаграму на рисунку 3.25, можна зробити ряд важливих спостережень щодо часу розтиснення різних алгоритмів. Загалом, час розтиснення для більшості алгоритмів є відносно невеликим, що свідчить про їхню придатність для застосування в системах, де потрібна швидка обробка даних.

Треба також враховувати, що пристрій, на якому виконується тестування, набагато швидший ніж звичайний мікроконтроллер, тому треба дивитись більше на відносне співвідношення, ніж на абсолютний час.

Серед алгоритмів, що демонструють найменший час розтиснення, виділяються всі варіації алгориту zstd, де навіть показники рівня 11 були швидшими, ніж любий з інших алгоритмів.

Загалом, аналіз часу розтиснення підтверджує високу швидкодію алгоритмів zstd, роблячи їх перспективними кандидатами для застосування в

промислових бездротових мережах, де важлива як швидкість стиснення, так і швидкість розтиснення даних.

3.5.3 Проведення тестування часу ступенів стиснення

Після аналізу часових характеристик операцій стиснення та розтиснення, наступним ключовим параметром, що визначає ефективність алгоритмів стиснення, є ступінь стиснення (compression ratio). Ступінь стиснення відображає відношення розміру даних до стиснення до розміру даних після стиснення, і є прямим показником того, наскільки ефективно алгоритм зменшує обсяг інформації. У контексті промислових бездротових мереж, де пропускна здатність каналу зв'язку часто є обмеженим ресурсом, досягнення високого ступеня стиснення є критично важливим для оптимізації використання мережевих ресурсів, зменшення затримок передачі даних та підвищення загальної продуктивності системи. Таким чином, аналіз ступеня стиснення різних алгоритмів дозволить оцінити їхню здатність економити пропускну здатність каналу та визначити найбільш ефективні алгоритми з точки зору зменшення обсягу переданих даних.

На рисунку 3.26 відображено діаграму ступенів стиснення.

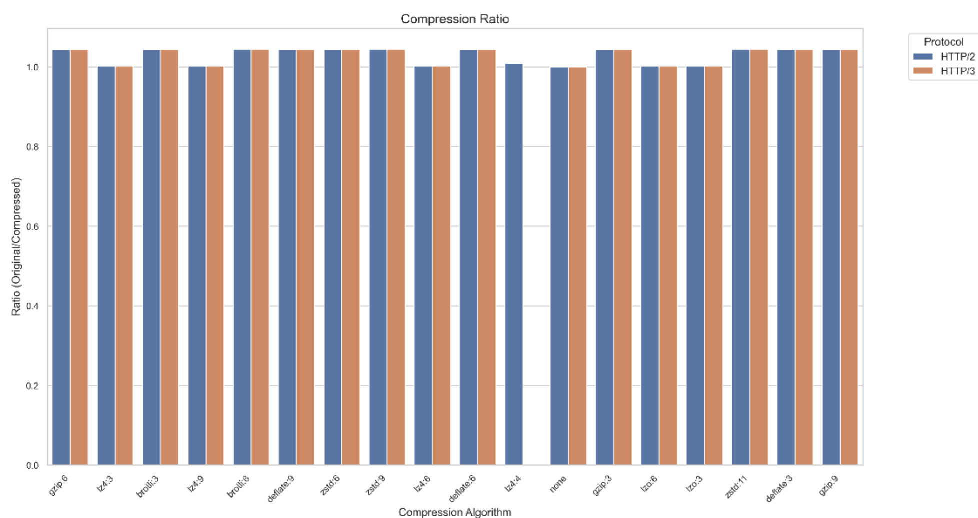

Рисунок 3.26 – Результат порівняння ступенів стиснення

Як можна побачити на графіку, деякі алгоритми не підходять для стиснення даних, що є типовими для даних, що передаються в промислових мережах. Такими є: lz4:3, lz4:6, lz4:9, lzo:6, lzo:3.

Таким чином, алгоритми сімейства LZ4 та LZO не підходять для цієї задачі та будуть виключені з наступних порівнянь.

Хоч ці алгоритми, LZ4 та LZO, і дуже часто використовуються в ядрі Linux для таких функцій як zRAM, де вони дуже гарно себе проявляють, важливо розуміти контекст їхнього застосування. У випадку zRAM, головним пріоритетом є мінімізація затримок та навантаження на процесор при операціях стиснення та розтиснення в оперативній пам'яті. Швидкість є критично важливою, навіть якщо це досягається за рахунок дещо нижчого ступеня стиснення. Однак, коли мова йде про передачу даних в мережах, ситуація дещо змінюється. Типові дані, що передаються в мережах, часто мають структуру та повторюваність, що дозволяє досягти значно вищого ступеня стиснення за допомогою більш складних алгоритмів, ніж ті, що використовуються в LZ4 та LZO. Хоча LZ4 та LZO забезпечують дуже швидке стиснення та розтиснення, їхній відносно низький ступінь стиснення може виявитися недостатнім для ефективної економії пропускної здатності в промислових мережах. У сценаріях, де оптимізація використання каналу зв'язку є ключовою, а час обробки даних не є настільки критичним, як у випадку zRAM, алгоритми, що забезпечують вищий ступінь стиснення, такі як gzip, deflate, brotli або zstd, можуть виявитися більш придатними, навіть якщо вони вимагають дещо більших обчислювальних ресурсів.

Таким чином, хоча LZ4 та LZO є чудовими інструментами для певних задач, їхні характеристики роблять їх менш оптимальними для стиснення типових мережевих даних у промислових середовищах, де баланс між швидкістю та ступенем стиснення є більш важливим, ніж виключно швидкість обробки.

3.5.4 Проведення тестування використання процесору

Після детального аналізу часових характеристик стиснення та розтиснення, а також ступеня стиснення, важливим аспектом оцінки ефективності алгоритмів є аналіз використання процесорних ресурсів (CPU usage).

У контексті промислових бездротових мереж, де обчислювальні ресурси пристроїв часто обмежені, а енергоефективність відіграє важливу роль, рівень використання процесора стає критичним параметром. Надмірне споживання процесорних ресурсів може призвести до збільшення енергоспоживання, зниження продуктивності пристроїв, та обмеження можливостей для виконання інших важливих задач. Тому, для забезпечення оптимальної роботи програмного модуля комп'ютеризованої системи безпровідної передачі інформації, необхідно проаналізувати та порівняти використання процесора різними алгоритмами стиснення при різних протоколах передачі даних. Надалі ми перейдемо до розгляду відповідних метрик та оцінимо вплив вибору алгоритму стиснення на навантаження на процесор.

Порівняння використання процесору можна побачити на рисунку 3.27.

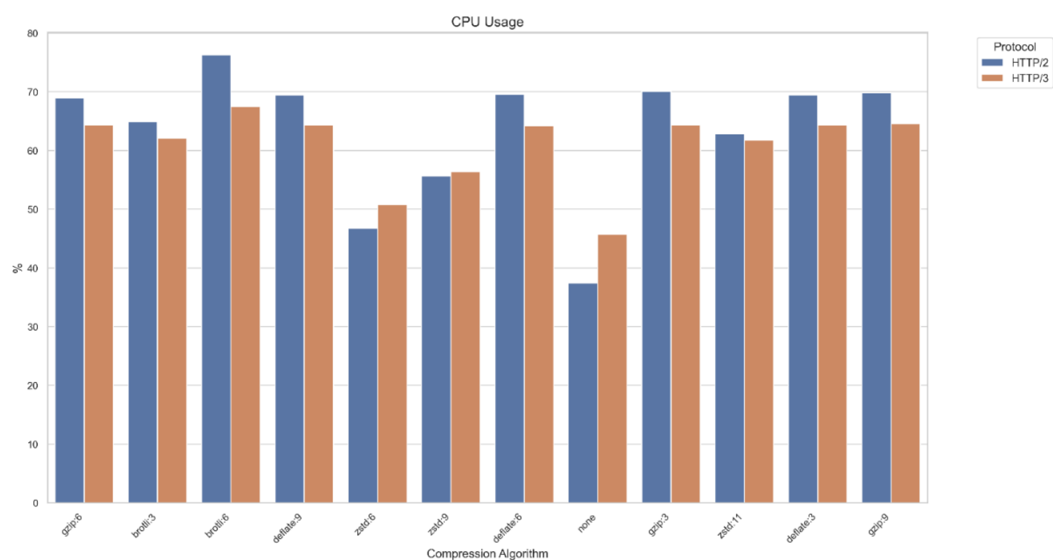

Рисунок 3.27 – Порівняння використання процесору

Розглядаючи діаграму на рисунку 3.27, можна відзначити, що рівень використання процесора варіюється в залежності від обраного алгоритму стиснення. Загалом, для більшості алгоритмів, використання процесора знаходиться в діапазоні від 50% до 75%, що свідчить про значне навантаження на процесор при виконанні операцій стиснення.

Серед алгоритмів, що демонструють найвище використання процесора, виділяються brotli:6 та deflate:9. Для алгоритму brotli:6, використання процесора досягає найвищих значень серед усіх протестованих алгоритмів, перевищуючи 75% для протоколу HTTP/2. Алгоритм deflate:9 також показує високий рівень використання процесора, наближаючись до 70% для обох протоколів. Таке високе навантаження на процесор може бути пов'язане зі складністю цих алгоритмів та їхньою орієнтацією на досягнення максимального ступеня стиснення, що вимагає значних обчислювальних ресурсів.

Алгоритми gzip:6, gzip:9, deflate:6 та deflate:3 демонструють дещо нижче, але все ще значне використання процесора, в діапазоні від 60% до 70%. Ці алгоритми також вимагають істотних обчислювальних ресурсів, хоча і менших, ніж brotli:6 та deflate:9.

Алгоритми zstd:6, zstd:9 та zstd:11 показують відносно нижче використання процесора порівняно з іншими алгоритмами, за винятком варіанту "none" (без стиснення). Для zstd:6 та zstd:9, використання процесора знаходиться в діапазоні від 55% до 60%, а для zstd:11 – дещо нижче, близько 60%. Ці результати свідчать про те, що алгоритм zstd є більш ефективним з точки зору використання процесорних ресурсів порівняно з brotli та deflate, принаймні в протестованих конфігураціях.

Варіант «none», що представляє передачу даних без стиснення, очікувано демонструє найнижче використання процесора, близько 40%. Це значення відображає базове навантаження на процесор, пов'язане з передачею даних без додаткової обробки стиснення.

Порівнюючи протоколи HTTP/2 та HTTP/3, не спостерігається чіткої та однозначної тенденції щодо впливу протоколу на використання процесора. Для

деяких алгоритмів, таких як gzip:6, brotli:6 та deflate:9, протокол HTTP/3 демонструє дещо нижче використання процесора, тоді як для інших, наприклад gzip:9 та deflate:6, різниця між протоколами є незначною або відсутня. Це може свідчити про те, що вплив протоколу на використання процесора залежить від конкретного алгоритму стиснення та особливостей його реалізації в різних протоколах.

Загалом, аналіз використання процесора показує, що операції стиснення даних, особливо з використанням алгоритмів brotli та deflate, можуть створювати значне навантаження на процесор. Алгоритм zstd демонструє відносно кращі показники з точки зору використання процесорних ресурсів.

При виборі оптимального алгоритму стиснення для промислових бездротових мереж, важливо враховувати не лише ступінь стиснення та час обробки, але й рівень використання процесора, особливо в умовах обмежених обчислювальних ресурсів та вимог до енергоефективності.

Аналізуючи отримані результати щодо використання процесора, можна дійти висновку, що деякі алгоритми створюють надмірне навантаження на обчислювальні ресурси, що може бути неприйнятним для промислових застосувань з обмеженими ресурсами або високими вимогами до енергоефективності.

Зокрема, алгоритми brotli:6, deflate:9, deflate:6, gzip:3 та gzip:9 продемонстрували відносно високий рівень використання процесора, часто перевищуючи 60-70% і навіть досягаючи понад 75% у випадку brotli:6. Таке значне навантаження може негативно вплинути на загальну продуктивність системи, збільшити енергоспоживання та обмежити можливості для виконання інших важливих задач. Зважаючи на ці міркування, подальший аналіз ефективності алгоритмів стиснення буде зосереджено на алгоритмах, що продемонстрували більш помірне використання процесорних ресурсів. Таким чином, алгоритми brotli:6, deflate:9, deflate:6, gzip:3 та gzip:9 будуть виключені з подальших порівнянь на основі критерію надмірного використання процесора.

3.5.5 Проведення тестування використання пам’яті

Після того, як ми детально розглянули часові характеристики, ступінь стиснення та використання процесорних ресурсів, наступним важливим етапом у нашому дослідженні є аналіз використання пам'яті (Memory Usage). В контексті промислових бездротових мереж, де пристрої часто характеризуються обмеженими обчислювальними ресурсами та об'ємом пам'яті, ефективне використання пам'яті є критично важливим фактором. Надмірне споживання пам'яті може призвести до зниження продуктивності системи, нестабільної роботи програмного забезпечення, або навіть до необхідності збільшення апаратних ресурсів, що є небажаним з точки зору вартості та енергоефективності. Тому, для забезпечення оптимальної роботи програмного модуля комп'ютеризованої системи безпровідної передачі інформації, необхідно проаналізувати та порівняти обсяг пам'яті, що використовується різними алгоритмами стиснення при різних протоколах передачі даних. Надалі ми перейдемо до розгляду метрик, що характеризують використання пам'яті, та оцінимо вплив вибору алгоритму стиснення на споживання пам'яті системою.

На рисунку 3.28 відображено результати аналізу використання пам’яті

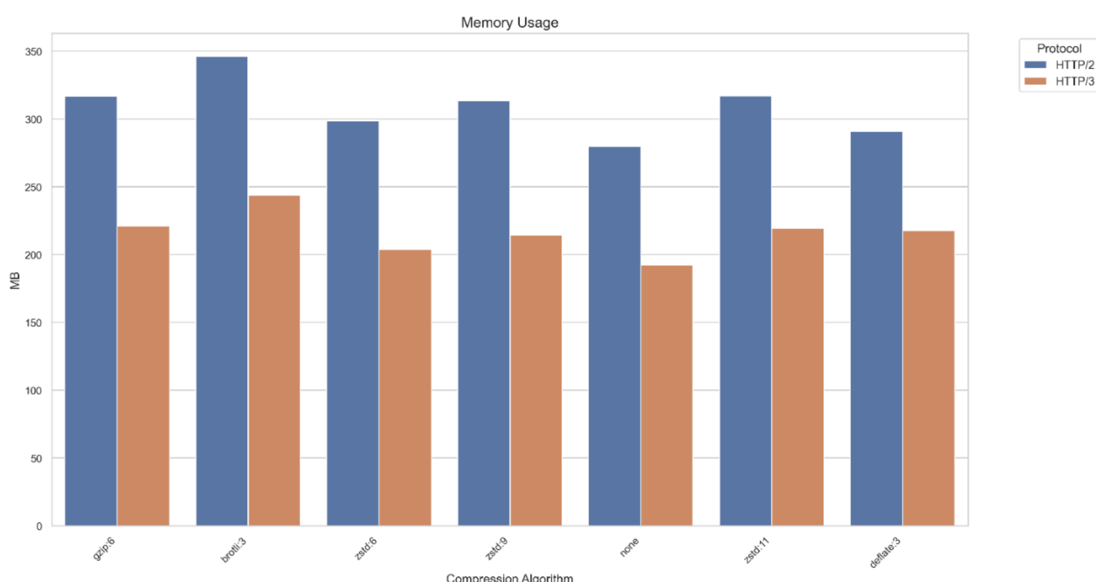

Рисунок 3.28 – Результат аналізу використання пам’яті

Аналізуючи діаграму на рисунку 3.28, можна спостерігати значну варіативність у використанні пам'яті залежно від обраного алгоритму стиснення. Загалом, обсяг використаної пам'яті коливається в діапазоні від приблизно 200 МБ до 350 МБ, що є досить суттєвим для вбудованих систем та пристроїв з обмеженим обсягом оперативної пам'яті, але це нормально, враховуючи що тест повинен показати стресову ситуацію.

Серед алгоритмів, що демонструють найвище використання пам'яті, виділяється brotli:3. Цей алгоритм показує найбільше споживання пам'яті серед усіх протестованих варіантів, досягаючи майже 350 МБ при використанні протоколу HTTP/2. Такий високий рівень споживання пам'яті робить алгоритм brotli:3 менш привабливим для застосувань в умовах обмежених ресурсів. Враховуючи, що brotli:3 демонструє найвище споживання пам'яті в сімействі алгоритмів brotli, і зважаючи на попередні виключення алгоритму brotli:6 через надмірне використання процесора, доцільно виключити все сімейство алгоритмів brotli з подальшого розгляду. Таким чином, алгоритм Brotli вибуває з подальшого аналізу через неприйнятно високе споживання пам'яті та процесорних ресурсів.

Алгоритми gzip:6, zstd:6, zstd:9 та zstd:11 також демонструють досить високе використання пам'яті, в діапазоні від 300 МБ до 320 МБ для протоколу HTTP/2. Проте, для протоколу HTTP/3, спостерігається чітка тенденція до зменшення використання пам'яті для цих алгоритмів. Наприклад, для gzip:6 та zstd:11, використання пам'яті при використанні HTTP/3 знижується до рівня близько 220 МБ, що є значним зменшенням порівняно з HTTP/2.

Алгоритм «none» (без стиснення) та deflate:3 демонструють відносно нижче використання пам'яті, в діапазоні від 280 МБ до 290 МБ для протоколу HTTP/2. Подібно до інших алгоритмів, для «none» та deflate:3 також спостерігається тенденція до зменшення використання пам'яті при використанні протоколу HTTP/3, хоча і менш виражена, ніж для gzip та zstd.

Загалом, аналіз використання пам'яті виявляє чітку тенденцію до зменшення споживання пам'яті при використанні протоколу HTTP/3 порівняно з

HTTP/2 для більшості протестованих алгоритмів. Ця тенденція може бути пов'язана з оптимізаціями в реалізації протоколу HTTP/3, спрямованими на зменшення накладних витрат пам'яті. Серед розглянутих алгоритмів, сімейство zstd демонструє відносно високе використання пам'яті, але при використанні протоколу HTTP/3, споживання пам'яті значно зменшується. Алгоритм deflate:3 та варіант «none» показують найнижче використання пам'яті серед алгоритмів, що залишилися після попередніх етапів відбору. На наступних етапах аналізу, важливо буде врахувати комбінований вплив часу обробки, ступеня стиснення, використання процесора та пам'яті для остаточного вибору оптимального алгоритму стиснення для промислових бездротових мереж.

Зважаючи на чітку тенденцію до зменшення використання пам'яті при використанні протоколу HTTP/3, а також беручи до уваги загальні тенденції розвитку мережевих технологій, де HTTP/3 поступово стає домінуючим протоколом, подальший аналіз ефективності алгоритмів стиснення буде зосереджено виключно на протоколі HTTP/3.

Протокол HTTP/2, хоч і є важливим етапом в еволюції веб-протоколів, демонструє в цілому гірші або порівнянні показники використання пам'яті, а в деяких випадках і часу обробки та використання процесора, порівняно з HTTP/3.

Виключення HTTP/2 з подальшого порівняння дозволить спростити аналіз, зосередившись на більш перспективному та ефективному протоколі HTTP/3, що є більш актуальним для сучасних та майбутніх промислових бездротових мереж. Таким чином, всі наступні етапи порівняння та висновки будуть базуватися виключно на результатах, отриманих для протоколу HTTP/3.

Важливо зазначити, що на отримані результати часу стиснення може впливати різниця між транспортними протоколами, що лежать в основі HTTP/2 та HTTP/3. HTTP/2 традиційно використовує TCP (Transmission Control Protocol), тоді як HTTP/3 базується на UDP (User Datagram Protocol) через QUIC (Quick UDP Internet Connections). TCP є протоколом з встановленням з'єднання, гарантованою доставкою та контролем перевантаження, що може вносити додаткові затримки та впливати на час вимірювання операцій, включаючи

стиснення. UDP, навпаки, є протоколом без встановлення з'єднання, що забезпечує швидшу передачу даних, але не гарантує доставку та порядок пакетів.

Ця різниця в характеристиках TCP та UDP може впливати на виміряний час стиснення, особливо в умовах мережевих затримок або перевантаження.

Таким чином, при інтерпретації результатів порівняння часу стиснення для HTTP/3, важливо враховувати потенційний вплив використання UDP як транспортного протоколу, що може сприяти зменшенню виміряного часу стиснення порівняно з HTTP/2, де використовується TCP.

3.5.6 Проведення тестування запитів в секунду

Переходимо до фінального етапу нашого аналізу, де ми розглянемо найважливішу метрику для оцінки продуктивності системи передачі даних – кількість запитів за секунду (Requests Per Second, RPS).

Саме RPS відображає реальну пропускну здатність системи та її здатність обробляти великі обсяги даних за одиницю часу.

Високий показник RPS є ключовим для забезпечення швидкої та ефективної передачі даних у промислових мережах, де оперативність та надійність є критично важливими.

На цьому етапі ми зосередимося на порівнянні різних алгоритмів стиснення виключно в контексті протоколу HTTP/3, який, як ми вже з'ясували, є більш перспективним та ефективним для нашої задачі.

На рисунку 3.29 відображено порівняння запитів в секунду.

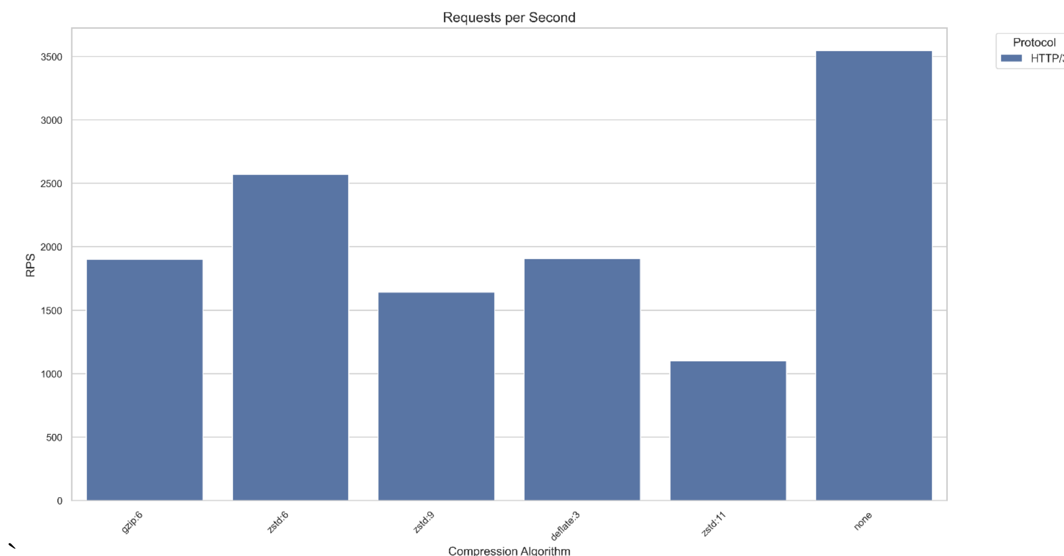

Рисунок 3.29 – Запити в секунду

Аналізуючи діаграму на рисунку 3.29, стає очевидним, що алгоритм zstd:6 виділяється як явний лідер за показником RPS. Він демонструє найвищу кількість запитів за секунду серед усіх протестованих алгоритмів, програючи лише варіант «none» (без стиснення). Це свідчить про те, що zstd:6 не лише забезпечує ефективне стиснення, але й дозволяє досягти максимальної пропускної здатності системи при використанні протоколу HTTP/3, у ситуаціях, де пропускна здатність обмежена швидкістю мережі або важливе зменшення об'єму даних.

3.5.7 Проведення тестування форматів даних

Тепер, коли ми визначили оптимальний алгоритм стиснення, а саме zstd:6, для використання з протоколом HTTP/3, наступним важливим кроком є дослідження впливу формату даних на загальну ефективність системи.

Формат даних, у якому серіалізується інформація перед стисненням та передачею, може суттєво впливати на ступінь стиснення, час обробки та, як наслідок, на загальну продуктивність системи.

Традиційно, для передачі даних у веб–застосунках часто використовується текстовий формат JSON (JavaScript Object Notation). Однак, існують

альтернативні бінарні формати, такі як MessagePack та Protobuf (Protocol Buffers), які обіцяють бути більш ефективними з точки зору розміру даних та швидкості обробки.

Тому, для подальшої оптимізації системи передачі даних, необхідно провести порівняльний аналіз ефективності використання різних форматів даних – JSON, MessagePack та Protobuf – у поєднанні з протоколом HTTP/3 та алгоритмом стиснення zstd:6.

На рисунку 3.30 відображене порівняння форматів даних.

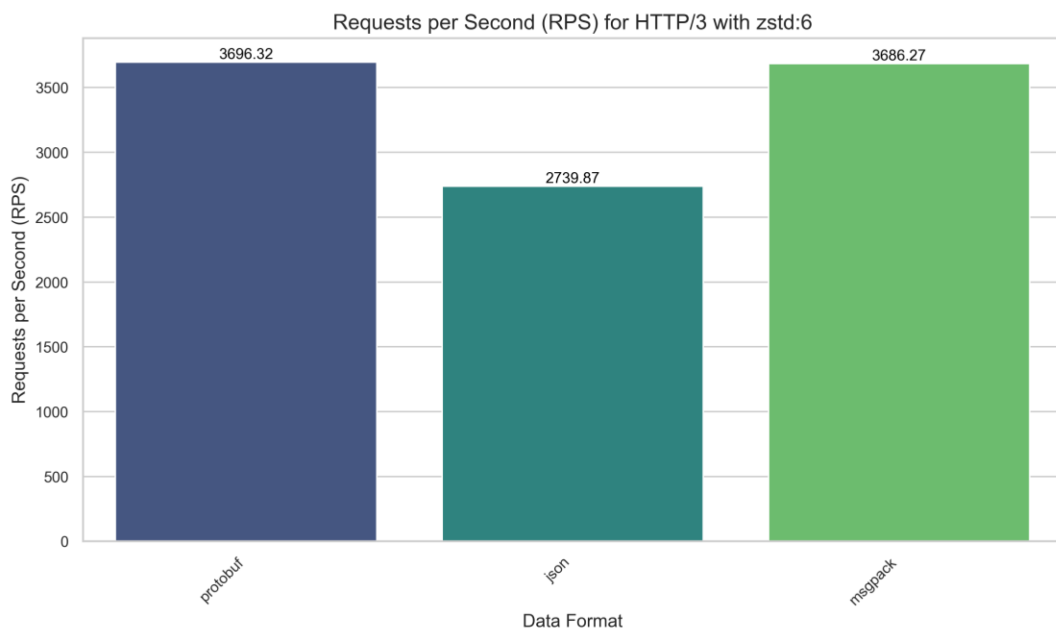

Рисунок 3.30 – Порівняння форматів даних

Аналізуючи діаграму на рисунку 3.30, чітко видно, що формат JSON демонструє значно нижчу кількість запитів за секунду порівняно з Protobuf та MessagePack. Показник RPS для JSON становить приблизно 2740, тоді як Protobuf та MessagePack досягають значно вищих значень – близько 3696 та 3686 RPS відповідно. Це свідчить про те, що бінарні формати Protobuf та MessagePack забезпечують суттєво кращу пропускну здатність системи порівняно з текстовим форматом JSON при використанні HTTP/3 та zstd:6.

Порівнюючи Protobuf та MessagePack між собою, можна відзначити, що їхні показники RPS є дуже близькими. Protobuf демонструє дещо вищий RPS

(3696) порівняно з MessagePack (3686), але ця різниця є незначною і може бути в межах похибки вимірювання. З точки зору продуктивності, обидва бінарні формати є практично еквівалентними та значно перевершують JSON.

Однак, при виборі між Protobuf та MessagePack, важливо враховувати не лише продуктивність, але й зручність використання та ергономічність формату. MessagePack, на відміну від Protobuf, є більш легким та динамічним форматом, який не вимагає попереднього визначення схеми даних та компіляції коду. Це робить MessagePack більш гнучким та зручним у розробці та інтеграції, особливо в умовах швидкої розробки та зміни вимог. Protobuf, хоч і забезпечує високу продуктивність, вимагає більш складної процедури визначення схеми даних та генерації коду, що може ускладнити процес розробки та підтримки.

Враховуючи практично ідентичну продуктивність Protobuf та MessagePack, а також значно більшу зручність та ергономічність MessagePack, можна зробити висновок, що MessagePack є оптимальним вибором формату даних для використання в поєднанні з протоколом HTTP/3 та алгоритмом стиснення zstd:6 в промислових бездротових мережах.

MessagePack забезпечує високу пропускну здатність, порівнянну з Protobuf, але при цьому є значно простішим та зручнішим у використанні, що спрощує розробку та зменшує витрати на підтримку програмного забезпечення.

3.5.8 Підсумок результатів тестування

Таким чином, підсумовуючи результати проведеного дослідження, можна з впевненістю стверджувати, що модернізація системи передачі даних шляхом переходу з протоколу HTTP/2 та алгоритму стиснення gzip:6 у поєднанні з форматом даних JSON на протокол HTTP/3, алгоритм стиснення zstd:6 та формат даних MessagePack, дозволяє досягти покращення по всьому спектру ключових параметрів продуктивності.

Аналіз показав, що комбінація HTTP/3, zstd:6 та MessagePack забезпечує:

– зменшення часу стиснення: Алгоритм zstd:6 демонструє менший час стиснення порівняно з gzip:6, що прискорює процес підготовки даних до передачі;

– зменшення часу розтиснення: zstd:6 також забезпечує швидке розтиснення даних на стороні отримувача, мінімізуючи затримки при обробці отриманої інформації;

– зниження використання процесора: Використання zstd:6 призводить до меншого навантаження на процесор, що дозволяє економити енергію та вивільнити обчислювальні ресурси для інших задач;

– зменшення використання пам'яті: HTTP/3 у поєднанні з zstd:6 та MessagePack сприяє більш ефективному використанню пам'яті, що особливо важливо для пристроїв з обмеженими ресурсами;

– збільшення кількості запитів за секунду (RPS): Перехід на HTTP/3, zstd:6 та MessagePack дозволяє досягти значно вищої пропускної здатності системи, збільшуючи кількість оброблених запитів за одиницю часу.

Ці покращення в сукупності свідчать про те, що модернізована система передачі даних є не тільки швидшою та ефективнішою, але й більш економічною з точки зору використання обчислювальних ресурсів.

Отримані результати підтверджують доцільність переходу на протокол HTTP/3, алгоритм стиснення zstd:6 та формат даних MessagePack для оптимізації передачі даних у промислових бездротових мережах.

Враховуючи всі результати, зробимо підсумкову таблицю, де буде зроблено порівняння базової конфігурації, у вигляді комбінації з HTTP/2, gzip:6 та JSON з найкращою, що ми знайшли, у вигляді HTTP/3, zstd:6, Msgpack.

В таблиці будуть абсолютні значення, але враховуючи конфігурації пристроїв на яких виконувалось тестування, найбільш головний висновок з цього тесту повинен бути зроблений у відносному порівнянні між результатами. Це порівняння наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння конфігурацій

Тип клієнту та серверу	HTTP/2	HTTP/3
Стиснення	gzip:6	zstd:6
Формат даних	Json	msgpack
Запитів в секунду	2976,862	4915,785
Байтів в секунду	144,020	237,901
Середнє використання пам’яті, МБ	107,782	55,679
Середнє використання процесору, %	74,048	48,709
Середній час стиснення, мс	4,246	0,622
Середній час розтиснення, мс	2,69	0,231
Середній розмір після стиснення, байти	101460,264	101492,735
Середній розмір який читав сервер, після розтиснення, байти	134653,4918	101482,665

3.6 Висновки до третього розділу

В процесі модернізації програмного модуля для промислових бездротових мереж, я провів дослідження, яке показало значні позитивні зміни при переході на конфігурацію з HTTP/3, алгоритмом стиснення zstd:6 та форматом даних msgpack, у порівнянні з попередньою конфігурацією на базі HTTP/2, gzip:6 та JSON.

Продуктивність розробленої системи значно зросла, про що свідчить збільшення кількості оброблених запитів за секунду приблизно на 65% та аналогічне зростання обсягу переданих даних за секунду. В контексті промислових мереж, де часто потрібно обробляти велику кількість даних від датчиків, виконавчих механізмів та інших пристроїв в реальному часі, це є дуже важливим результатом. Збільшення кількості запитів за секунду означає, що моя оновлена система здатна обробляти значно більший потік інформації. Це може бути критично важливим для систем моніторингу, керування виробничими процесами або будь-яких інших застосувань, де потрібна висока пропускна

здатність мережі. Вважаю, що перехід на HTTP/3, який використовує протокол QUIC, дозволив зменшити затримки та покращити ефективність передачі даних, особливо в умовах нестабільного бездротового з'єднання, що і сприяло збільшенню кількості оброблених запитів.

Подібне зростання спостерігається і в кількості байтів за секунду, яка збільшилась на 65,19%. Цей показник тісно пов'язаний з попереднім і підтверджує, що система не тільки обробляє більше запитів, але й передає значно більший обсяг даних за одиницю часу. Це важливо для передачі великих обсягів даних, наприклад, відео з камер спостереження, даних вимірювань високої роздільної здатності або оновлень програмного забезпечення для промислових пристроїв. Знову ж таки, ефективність HTTP/3 у поєднанні з оптимізованими форматами даних та алгоритмами стиснення відіграє ключову роль у цьому покращенні, що підтверджує правильність обраного підходу.

Дуже важливим є зменшення середнього використання пам'яті на 48,34%. В промислових системах, особливо на вбудованих пристроях або серверах з обмеженими ресурсами, ефективне використання пам'яті є критичним. Зменшення споживання пам'яті означає, що моя система стала більш ресурсоефективною. Це дозволяє використовувати менш потужне (і, можливо, дешевше) обладнання, або ж дозволяє системі обробляти більші навантаження без ризику вичерпання пам'яті та збоїв. Я вважаю, що використання msgpack замість JSON сприяє зменшенню споживання пам'яті, оскільки msgpack є бінарним форматом, який зазвичай є більш компактним та ефективним у обробці, ніж текстовий JSON.

Аналогічно, середнє використання процесора зменшилось на 34,22%. Зниження навантаження на процесор має багато позитивних наслідків. По-перше, це зменшує енергоспоживання, що може бути важливим для автономних пристроїв або великих промислових установок, де сумарне енергоспоживання є значним. По-друге, це звільняє обчислювальні ресурси процесора для інших задач. Наприклад, сервер може використовувати звільнені ресурси для обробки більшої кількості клієнтів, виконання додаткових аналітичних задач або просто

працювати більш стабільно та надійно. Я припускаю, що використання zstd:6, який є дуже швидким алгоритмом стиснення, та msgpack, який є ефективним у обробці, сприяло зменшенню навантаження на процесор.

Середній час стиснення зменшився на вражаючі 85,35%, а середній час розтиснення – на 91,41%. Ці показники є критично важливими для систем реального часу, де затримки в обробці даних повинні бути мінімальними. Значне скорочення часу стиснення та розтиснення означає, що дані обробляються набагато швидше, що зменшує загальну затримку передачі даних. Це особливо важливо для промислових мереж, де швидка реакція на події може бути критичною для безпеки та ефективності виробничих процесів. На мою думку, використання zstd:6 замість gzip:6 є ключовим фактором у цьому покращенні, оскільки zstd:6 відомий своєю високою швидкістю стиснення та розтиснення, особливо при високих рівнях стиснення.

Середній розмір після стиснення практично не змінився, збільшившись лише на 0,03%. Це означає, що, незважаючи на значне покращення швидкості стиснення, я не втратив в ефективності стиснення даних. zstd:6 забезпечує порівнянний рівень стиснення з gzip:6, але робить це значно швидше, що є важливим компромісом у контексті моєї роботи.

Нарешті, середній розмір, який читав сервер після розтиснення, зменшився на 24,63%. Це цікавий результат, який потребує подальшого вивчення. Можливо, це пов'язано з тим, що msgpack, як бінарний формат, більш ефективно представляє дані, ніж JSON, що призводить до меншого обсягу даних після розпакування. Інша можлива причина може бути пов'язана з особливостями вимірювання цих даних, наприклад, різницею в тому, як вимірюється розмір даних на різних етапах обробки. В будь-якому випадку, зменшення обсягу даних, які серверу потрібно обробляти, може сприяти подальшому підвищенню ефективності системи.

Підсумовуючи, результати мого дослідження демонструють, що модернізація програмного модуля з використанням HTTP/3, zstd:6 та msgpack

принесла комплексні покращення, охоплюючи продуктивність, ефективність використання ресурсів та швидкість обробки даних.

Ці покращення є значними і, на мою думку, можуть мати велику цінність для промислових бездротових мереж, де вимоги до надійності, швидкодії та ефективності постійно зростають.

4 ОХОРОНА ПРАЦІ

Охорона праці під час проведення мережевих випробувань має велике значення для безпеки працівників і стабільності обладнання. У процесі тестування використовують сервери, маршрутизатори, точки доступу, різноманітні кабелі та електронні компоненти, які можуть становити певний ризик, оскільки потребують підключення до блоків живлення, інтенсивно нагріваються, випромінюють радіочастоти та вимагають дотримання правил ергономіки.

Важливо зважати на електричні загрози та уважно перевіряти справність живильних кабелів і розеток. Серверні блоки живлення часто мають велику потужність і можуть бути джерелом небезпечних напруг, тому необхідна наявність заземлення, захисних автоматів і пристроїв захисного відключення. Рекомендується також використовувати системи безперебійного живлення, що захищають від стрибків напруги.

Сервери, маршрутизатори та інше мережеве обладнання можуть перегріватися, якщо немає належної вентиляції або коли забруднені кулькові вентилятори та радіатори. Перегрів збільшує ризик пожежі, тож слід регулярно перевіряти роботу охолоджувальних систем, розміщувати обладнання з урахуванням руху повітря і зберігати приміщення в чистоті. Наявність вогнегасників, що годяться для електричних установок, підвищує безпеку під час можливих несправностей.

Кабелі, які лежать на підлозі, можуть спричинити падіння, а неправильна експлуатація роз’ємів підвищує ризик короткого замикання чи поломки. Краще прокладати дроти в організованих кабель-каналах, уникати зламів оптоволоконних ліній, стежити за чистотою конекторів і при потребі користуватися антистатичними браслетами, щоб не пошкодити чутливі компоненти електростатичним розрядом.

У разі використання бездротового обладнання працівники мають звертати увагу на радіочастотне випромінювання, зокрема якщо застосовують потужні передавачі або спеціальні антени. Найчастіше таке випромінювання перебуває в безпечних межах, але важливо встановлювати точки доступу так, щоб люди не перебували впритул до антен протягом тривалого часу.

Тестова робота з великою кількістю обладнання передбачає можливі незручності: важкі сервери можуть спричинити травми спини, якщо їх підіймати без допомоги або належних засобів. Працівники можуть довго сидіти за комп'ютерами, тож рекомендується належна організація робочого місця та перерви для зняття втоми. Затісні приміщення чи недолік освітлення підвищують ризик травм, тому важливо облаштовувати простір із достатньою площею, вентиляцією та кондиціонуванням.

У приміщеннях із великим тепловиділенням від серверів варто стежити за температурою та вологістю повітря, не допускати поширення пилу, а для кращої циркуляції повітря регулярно чистити фільтри кондиціонерів. Високотемпературне середовище шкодить і техніці, і людям, спричиняючи швидку втому персоналу і більшу ймовірність збоїв роботи апаратури.

У багатьох країнах є норми, які вимагають регулярних інструктажів, технічних перевірок, відповідності електробезпеці та стандартам пожежогасіння. Працівники мають уміти правильно поводитися з серверним обладнанням, розумітися на планах евакуації, знати, де розташовано головний автомат відключення живлення і системи пожежної сигналізації.

Перед початком робіт кожному слід пройти навчання, дізнатися про особливості обладнання та правила безпечного підключення в мережу. Потрібно знати основні процедури реагування на задимлення чи відключення електрики та напрацювати навички користування вогнегасниками. Варто також пам'ятати про засоби індивідуального захисту. Антистатичні браслети чи спеціальні рукавиці можуть захистити електронні схеми від пошкоджень, а самих працівників від опіків під час контакту з розпеченими радіаторами.

Антистатичне взуття і належне вбрання допомагають запобігти накопиченню зарядів електростатики.

У підсумку, під час проведення мережевих тестів книга правил з безпеки повинна охоплювати електробезпеку, пожежну безпеку, ергономіку, забезпечення належного мікроклімату та дотримання законодавчих норм. Постійний моніторинг технічного стану стенду та підвищення кваліфікації працівників дозволяють уникати аварійних ситуацій, зберігати здоров’я людей і забезпечувати надійну роботу досліджуваних систем.

ВИСНОВКИ

Проведене дослідження з модернізації програмного модуля для промислових бездротових мереж виявило значний потенціал для покращення ключових характеристик систем передачі даних. Аналіз результатів експериментальних випробувань чітко демонструє, що впровадження комплексу технологічних рішень, а саме перехід на протокол HTTP/3 у поєднанні з алгоритмом стиснення zstd:6 та форматом даних MessagePack, забезпечує відчутні переваги у порівнянні з традиційними підходами, представленими конфігурацією HTTP/2, алгоритмом стиснення gzip:6 та форматом даних JSON. Модернізована система виявилася значно продуктивнішою, що підтверджується суттєвим зростанням кількості оброблених запитів за секунду та аналогічним збільшенням обсягу переданих даних за одиницю часу. Це покращення продуктивності є особливо важливим у контексті промислових застосувань, де обробка великих потоків даних у реальному часі є критичною вимогою для забезпечення ефективної роботи виробничих процесів та систем моніторингу.

Окрім підвищення продуктивності, модернізація програмного модуля також призвела до значної оптимізації використання обчислювальних ресурсів. Зменшення середнього споживання пам'яті та процесора свідчить про підвищення ресурсоефективності системи, що є важливим фактором для промислових застосувань, де пристрої часто працюють в умовах обмежених ресурсів та високих вимог до енергоспоживання. Зниження навантаження на обчислювальні ресурси не лише зменшує енергетичні витрати, але й звільняє ресурси для виконання інших важливих задач, таких як додаткова обробка даних або розширення функціональності системи.

Важливим аспектом модернізації є також значне прискорення обробки даних. Скорочення часу стиснення та розтиснення, що досягає вражаючих показників, безпосередньо впливає на зменшення затримок при передачі даних, що є критичним параметром для систем реального часу. Швидша обробка даних

дозволяє забезпечити оперативнішу реакцію системи на події та зміни у виробничих процесах, що є особливо важливим для забезпечення безпеки та ефективності технологічних операцій.

Таким чином, комплексний аналіз результатів дослідження дозволяє стверджувати, що модернізація програмного модуля шляхом впровадження протоколу HTTP/3, алгоритму стиснення zstd:6 та формату даних MessagePack є ефективним підходом для покращення характеристик систем передачі даних у промислових бездротових мережах. Запропоновані технологічні рішення забезпечують значне підвищення продуктивності, оптимізацію використання ресурсів та прискорення обробки даних, що робить модернізовану систему більш привабливою для широкого спектру промислових застосувань, де надійність, швидкодія та ефективність є ключовими вимогами. Отримані результати дослідження безпосередньо відповідають Цілі сталого розвитку 9 «Промисловість, інновації та інфраструктура», зокрема завданню 9.4 щодо розвитку високотехнологічних секторів промисловості. Розроблені рішення з модернізації програмного модуля для промислових бездротових мереж (впровадження HTTP/3, zstd:6 та MessagePack) сприяють підвищенню показників індикатора 9.4.1. Досягнуті результати з оптимізації використання обчислювальних ресурсів та енергоспоживання, значного підвищення продуктивності систем передачі даних та прискорення обробки даних у реальному часі безпосередньо впливають на ефективність виробництва електронної та комп'ютерної продукції, що входить до високотехнологічного сектору згідно з КВЕД, та сприяють підвищенню конкурентоспроможності вітчизняних підприємств на глобальному ринку. Отримані результати підкреслюють важливість комплексного підходу до оптимізації систем передачі даних, враховуючи взаємодію різних технологічних компонентів та їх вплив на загальну ефективність системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 174 Автоматизація, комп’ютерно-інтегровані технології та робототехніка, освітньо-професійних програм: «Комп’ютерно-інтегровані технологічні процеси і виробництва», «Комп’ютеризовані та робототехнічні системи» / Упоряд. І. Ш. Невлюдов, Р. В. Артюх, В. В. Безкоровайний, Н. П. Демська, В. В. Євсєєв, О. І. Филипенко, О. М. Цимбал. Харків: ХНУРЕ, 2024. 57 с.

2. ДСТУ 3008: 2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. К.: ДП «УкрНДНЦ». 2016. 30 с. 3. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація та комп’ютерно-інтегровані технології»: довід. /І. Ш. Невлюдов, А. О. Андрусевич, О. В. Токарєва, Г. В. Пономарьова. Київ, 2018. 320 с.

3. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація та комп’ютерно-інтегровані технології»: довід. / І. Ш. Невлюдов, А. О. Андрусевич, О. В. Токарєва, Г. В. Пономарьова. Київ, 2018. 320 с.

4. Дяченко Е.С. Сучасні формати даних та їх вплив на швидкодії ВЕБ-додатків / Automation and Development of Electronic Devices, Харків. 2023. №1. с. 56–60.

5. User datagram protocol [Електронний ресурс] // IETF | Internet Engineering Task Force. – Режим доступу: https://www.ietf.org/rfc/rfc768.txt (дата звернення: 10.01.2025). – Назва з екрана.

6. Transmission control protocol [Електронний ресурс] // IETF | Internet Engineering Task Force. – Режим доступу: https://www.ietf.org/rfc/rfc1951.txt (дата звернення: 10.01.2025). – Назва з екрана.

7. Ziv L. A universal algorithm for sequential data compression [Електронний ресурс] / Lempel Ziv // Duke Computer Science. – Режим доступу:

https://courses.cs.duke.edu/spring03/cps296.5/papers/ziv_lempel_1977_universal_algorithm.pdf (дата звернення: 10.01.2025). – Назва з екрана.

8. RFC 7694: Hypertext transfer protocol (HTTP) client-initiated content-encoding [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc7694 (дата звернення: 10.01.2025). – Назва з екрана.

9. RFC 8478: zstandard compression and the application/zstd media type [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc8478 (дата звернення: 10.01.2025). – Назва з екрана.

10. RFC 1952: GZIP file format specification version 4.3 [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc1952 (дата звернення: 10.01.2025). – Назва з екрана.

11. LZ4 documentation [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/lz4/lz4/blob/dev/doc/lz4_Block_format.md (дата звернення: 10.01.2025). – Назва з екрана.

12. The H.264 video coding standard [Електронний ресурс] / Humberto De Jesus Ochoa Dominguez [та ін.] // IEEE potentials. – 2014. – Т. 33, № 2. – С. 32–38. – Режим доступу: https://doi.org/10.1109/mpot.2013.2284525 (дата звернення: 10.01.2025). – Назва з екрана.

13. AV1 bitstream & decoding process specification [Електронний ресурс] // aomediacodec.github.io. – Режим доступу: https://aomediacodec.github.io/av1-spec/av1–spec.pdf (дата звернення: 10.01.2025). – Назва з екрана.

14. RFC 1951: DEFLATE compressed data format specification version 1.3 [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc1951 (дата звернення: 10.01.2025). – Назва з екрана.

15. RFC 9113: HTTP/2 [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc9113 (дата звернення: 10.01.2025). – Назва з екрана.

16. RFC 9000: QUIC: a UDP-based multiplexed and secure transport [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/rfc9000/ (дата звернення: 10.01.2025). – Назва з екрана.

17. GRPC documentation [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-HTTP2.md (дата звернення: 10.01.2025). – Назва з екрана.

18. RFC 8259: the JavaScript object notation (JSON) data interchange format [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc8259 (дата звернення: 10.01.2025). – Назва з екрана.

19. Protocol buffers version 3 language specification [Електронний ресурс] // Protocol Buffers Documentation. – Режим доступу: https://protobuf.dev/reference/protobuf/proto3-spec/ (дата звернення: 10.01.2025). – Назва з екрана.

20. Msgpack specification [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/msgpack/msgpack/blob/master/spec.md (дата звернення: 10.01.2025). – Назва з екрана.

21. Case studies – the Go programming language [Електронний ресурс] // The Go Programming Language. – Режим доступу: https://go.dev/solutions/case-studies (дата звернення: 10.01.2025). – Назва з екрана.

22. Why you should use Google Go [Електронний ресурс] // Dittofi. – Режим доступу: https://www.dittofi.com/wp-content/uploads/2023/02/Companies-That-Use-Google-Go.png (дата звернення: 10.01.2025). – Назва з екрана.

23. Official python website [Електронний ресурс] // Python.org. – Режим доступу: https://www.python.org/ (дата звернення: 10.01.2025). – Назва з екрана.

24. RFC 7932: Brotli compressed data format [Електронний ресурс] // IETF Datatracker. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc7932 (дата звернення: 10.01.2025). – Назва з екрана.

25. LZO documentation [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/nemequ/lzo/blob/master/doc/LZO.TXT (дата звернення: 10.01.2025). – Назва з екрана.

26. Про охорону праці [Електронний ресурс]: Закон України від 14.10.1992 р. № 2694-XII: станом на 31 берез. 2023 р. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 10.01.2025). – Назва з екрана.