ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ ТА МЕРЕЖ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ОПТИМІЗАЦІЯ АРХІТЕКТУРИ ТЕЛЕКОМУНІКАЦІЙНИХ МЕРЕЖ З УРАХУВАННЯМ ВИМОГ КОРИСТУВАЧІВ»

на здобуття освітнього ступеня магістра
зі спеціальності 172 Електронні комунікації та радіотехніка
(код, найменування спеціальності)
освітньо-професійної програми Телекомунікаційні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Євгеній ЛІСОВЕНКО
(підпис) *(Ім'я ПРІЗВИЩЕ)*

Виконав: здобувач вищої освіти гр. ТСДМ-62
Євгеній ЛІСОВЕНКО
Ім'я, ПРІЗВИЩЕ

Керівник: Олександр ДРОБИК
к.т.н., професор *Ім'я, ПРІЗВИЩЕ*

Рецензент: _____
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Телекомунікацій

Кафедра ___Телекомунікаційних систем та мереж_____
Ступінь вищої освіти __магістр_____
Спеціальність ___172 Електронні комунікації та радіотехніка _____
Освітньо-професійна програма _____Телекомунікаційні системи та мережі_____

ЗАТВЕРДЖУЮ
Завідувач кафедри __ТСМ_____
_____ Віктор ЗАЇКА
«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лісовенку Євгенію Володимировичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Оптимізація архітектури телекомунікаційних мереж з урахуванням вимог користувачів
керівник кваліфікаційної роботи Олександр ДРОБИК, к.т.н., професор
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» 10. 2025 р. № 467

2. Строк подання кваліфікаційної роботи «___» _____ 2025 р.

3. Вихідні дані до кваліфікаційної роботи:
1. Оптимізація архітектури.
2. Вимоги користувачів.
3. Використання гібридної архітектури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Архітектура для користувачів.
2. Оптимізація навантаження на мережу за допомогою диференціальних оновлень iot.
3. Передчасна оптимізація архітектури.
4. Використання гібридної архітектури.

5. Перелік ілюстративного матеріалу: ___презентація на____слайдах_____

6. Дата видачі завдання «___» _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	15.10.2025	Викон.
2	Розробка першого розділу	23.10.2025	Викон.
3	Розробка другого розділу	03.11.2025	Викон.
4	Розробка третього розділу	13.11.2025	Викон.
5	Розробка четвертого розділу	23.11.2025	Викон.
6	Висновки, вступ, реферат	03.12.2025	Викон.
7	Розробка презентації	13.12.2025	Викон.

Здобувач вищої освіти _____ Євгеній ЛІСОВЕНКО
(підпис) *(Ім'я ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Олександр ДРОБИК
(підпис) *(Ім'я ПРІЗВИЩЕ)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут телекомунікацій
ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувач Лісовенко Є.В. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 172 Електронні комунікації та радіотехніка
(код, найменування спеціальності)
освітньо-професійної програми Телекомунікаційні системи та мережі
(назва)
на тему: «Оптимізація архітектури телекомунікаційних мереж з урахуванням вимог користувачів»

Кваліфікаційна робота і рецензія додаються.

Директор ННІТ _____ Владислав КРАВЧЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Лісовенко Євгеній Володимирович показав гарну теоретичну та інженерну підготовку, уміння володіти новими комп'ютерними технологіями, користуватися навчальною, довідковою і науково-технічною літературою. Працюючи над завданнями, які були поставлені керівником, проявив ініціативність, сумлінність та хист до наукової роботи.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Лісовенка Євгенія Володимировича на оцінку «відмінно» та присвоїти йому кваліфікацію магістр з електронних комунікацій та радіотехніки.

Керівник кваліфікаційної роботи _____ Олександр ДРОБИК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

«____» _____ 20___ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Лісовенко Є.В. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедри телекомунікаційних систем та мереж _____ Віктор ЗАЇКА
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

«____» _____ 20___ року

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу
на здобуття освітнього ступеня магістра

здобувача вищої освіти Лісовенка Євгенія Володимировича
(прізвище, ім'я, по батькові)
на тему «Оптимізація архітектури телекомунікаційних мереж з урахуванням вимог користувачів»

Актуальність.
З появою 5G, IoT, відеострімінгу та сервісів у реальному часі проблема оптимізації архітектури телекомунікаційних мереж набуває критичного значення. Крім того, користувачі вже не просто "споживають мережу" — вони очікують миттєвого завантаження, адаптивної якості контенту, безперервності сервісів та високого рівня обслуговування (QoE). Саме тому виникає потреба у конвергентних (гібридних) архітектурах, здатних поєднувати централізоване та децентралізоване управління, використовуючи інтелектуальне балансування навантаження, on-demand transcoding, диференціальні оновлення IoT та нові мережеві протоколи (HTTP/2, HTTP/3).
Позитивні сторони.
1. У роботі окрему увагу приділено метрикам якості (QoE, MOS, VMAF, SSIM), що демонструє глибоке розуміння взаємозв'язку між технічними параметрами мережі та задоволеністю користувачів.
2. У роботі наведено реальні сценарії використання протоколів HTTP/1.1, HTTP/2, HTTP/3, мультиплексування та паралельного завантаження.
Недоліки.
1. У роботі не проведено порівняльного аналізу архітектур за конкретними технічними показниками, такими як масштабованість, затримка, вартість реалізації.
2. У роботі не проведено експериментального моделювання або симуляції трафіку (наприклад, моделювання залежності QoE від RTT або втрати пакетів).

Відзначені зауваження не впливають на загальну позитивну оцінку магістерської кваліфікаційної роботи

Висновок: *кваліфікаційна робота на здобуття ступеня магістра заслуговує оцінку "відмінно", а здобувач Лісовенко Євгеній Володимирович заслуговує присвоєння кваліфікації: магістр з електронних комунікацій та радіотехніки.*

Рецензент _____ _____
науковий ступінь, вчене звання (підпис) *(Ім'я ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 78 стор., 23 рис., 2 табл., 24 джерела.

Мета роботи – аналіз, удосконалення та оптимізація архітектури телекомунікаційних мереж з урахуванням вимог до якості користувацького досвіду (QoE), пропускної здатності, затримок, масштабованості та стійкості мереж при зростанні трафіку, використанні IoT-рішень та потокових сервісів.

Об’єкт дослідження – архітектура сучасних телекомунікаційних мереж, які забезпечують передавання мультимедійних та сервісних даних у цифровому середовищі.

Предмет дослідження – методи та технології оптимізації мережевої архітектури.

Короткий зміст роботи: У роботі проведено аналіз сучасних архітектур телекомунікаційних мереж та розглянуто методи їх оптимізації з урахуванням потреб користувачів. Досліджено чотири основні моделі: клієнт-серверну, хабову, пірингову та конвергентну. Встановлено, що конвергентна (гібридна) архітектура є найбільш збалансованим рішенням, оскільки поєднує централізований контроль із децентралізованою стійкістю до збоїв. Результати показали, що ключовим принципом оптимізації мереж є адаптивність — здатність системи реагувати на зміну навантажень, обсягів трафіку та вимог до QoE/QoS. Також підкреслено важливість ефективного управління трафіком, автоматизації, балансування навантаження та прогнозування стану мережі.

КЛЮЧОВІ СЛОВА: ТЕЛЕКОМУНІКАЦІЙНІ МЕРЕЖІ, АРХІТЕКТУРА МЕРЕЖ, ОПТИМІЗАЦІЯ, QOE, QOS, АДАПТИВНІСТЬ, 5G, 6G, IOT, КОНВЕРГЕНТНА АРХІТЕКТУРА, БАЛАНСУВАННЯ НАВАНТАЖЕННЯ, СТІЙКІСТЬ МЕРЕЖІ, ПРОГНОЗУВАННЯ ТРАФІКУ, ПОТОКОВІ СЕРВІСИ

ABSTRACT

Text part of the master's qualification work: 78 pages, 23 figures, 2 table, 24 sources.

The purpose of the work is to analyze, improve and optimize the architecture of telecommunication networks, taking into account the requirements for the quality of user experience (QoE), bandwidth, delays, scalability and stability of networks with increasing traffic, the use of IoT solutions and streaming services.

Object of the study is the architecture of modern telecommunication networks that provide the transmission of multimedia and service data in a digital environment.

Subject of the study is methods and technologies for optimizing network architecture.

Summary of the work: The paper analyzes modern telecommunication network architectures and considers methods for optimizing them taking into account user needs. Four main models are studied: client-server, hub, peering and convergent. It is established that the convergent (hybrid) architecture is the most balanced solution, since it combines centralized control with decentralized fault tolerance. The results showed that the key principle of network optimization is adaptability - the ability of the system to respond to changes in loads, traffic volumes and QoE/QoS requirements. The importance of effective traffic management, automation, load balancing and network state prediction is also emphasized.

KEYWORDS: TELECOMMUNICATIONS NETWORKS, NETWORK ARCHITECTURE, OPTIMIZATION, QOE, QOS, ADAPTABILITY, 5G, 6G, IOT, CONVERGENT ARCHITECTURE, LOAD BALANCING, NETWORK RESILIENCE, TRAFFIC FORECASTING, STREAMING SERVICES

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1 АРХІТЕКТУРА ДЛЯ КОРИСТУВАЧІВ 11
1.1 Ключові вимоги до мережевої архітектури 11
1.2 Вимоги до завантаження 14
1.3 Вимоги на прикладі відео 21
РОЗДІЛ 2 ОПТИМІЗАЦІЯ НАВАНТАЖЕННЯ НА МЕРЕЖУ ЗА ДОПОМОГОЮ ДИФЕРЕНЦІАЛЬНИХ ОНОВЛЕНЬ ІОТ 27
2.1 Оновлення snap пакетів 27
2.2 Диференціальні snap пакети 31
РОЗДІЛ 3 ПЕРЕДЧАСНА ОПТИМІЗАЦІЯ АРХІТЕКТУРИ 41
3.1 Проблеми при плануванні ТК архітектур 41
3.2 Основні серверні проблеми 44
РОЗДІЛ 4 ВИКОРИСТАННЯ ГІБРИДНОЇ АРХІТЕКТУРИ 53
4.1 Клієнт-серверна архітектура 54
4.2 Хабова архітектура телекомунікаційних систем 60
4.3 Пірингова архітектура телекомунікаційних систем 65
4.4 Конвергентна архітектура телекомунікаційних систем 71
ВИСНОВКИ 77
ПЕРЕЛІК ПОСИЛАНЬ 79
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 81

ВСТУП

Сучасний етап розвитку телекомунікацій характеризується стрімким зростанням обсягів передавання даних, кількості користувацьких пристроїв та складності мережевих систем. У цих умовах питання оптимізації архітектури телекомунікаційних мереж набуває особливого значення, адже саме від архітектурного підходу залежить ефективність, стабільність і безпека функціонування усієї інфраструктури.

З розвитком цифрових технологій користувачі очікують не лише високої швидкості зв’язку, але й мінімальних затримок, гарантованої якості обслуговування (QoS), стабільного доступу до сервісів та надійного захисту персональних даних. Тому телекомунікаційні компанії змушені постійно модернізувати мережі, удосконалювати методи маршрутизації, балансування навантаження та керування трафіком. Усе це потребує глибокого розуміння архітектурних принципів, на яких ґрунтуються сучасні системи зв’язку.

Традиційні архітектури, такі як клієнт-серверна, хабова чи пірингова, мають свої переваги та недоліки, однак у реальних умовах вони не завжди здатні забезпечити гнучкість, масштабованість і безперервність обміну даними. З огляду на це з’явився новий підхід — конвергентна (гібридна) архітектура, що поєднує елементи централізації та децентралізації, створюючи баланс між продуктивністю, безпекою й ефективністю управління. Такий підхід дозволяє будувати телекомунікаційні мережі, які відповідають реальним потребам користувачів і здатні адаптуватися до швидких змін технологічного середовища.

Метою дипломної роботи є аналіз і оптимізація архітектури телекомунікаційних мереж з урахуванням вимог користувачів, визначення основних принципів побудови ефективних систем зв’язку, а також пошук шляхів підвищення їх надійності, швидкодії та безпеки.

У роботі було розглянуто ключові архітектурні моделі, виявлено їхні сильні й слабкі сторони, проведено порівняльний аналіз за основними критеріями ефективності та розроблено узагальнену концепцію оптимізації мережевої

структури з урахуванням сучасних технологічних трендів — впровадження 5G, розширення IoT-інфраструктури, використання штучного інтелекту та автоматизованих систем управління.

Отже, актуальність теми визначається необхідністю пошуку універсальної архітектурної моделі, здатної забезпечити стабільну роботу телекомунікаційних систем у динамічних умовах цифрового середовища, задовольняючи при цьому зростаючі вимоги користувачів до якості, безпеки та доступності послуг зв’язку.

1 АРХІТЕКТУРА ДЛЯ КОРИСТУВАЧІВ

1.1 Ключові вимоги до мережевої архітектури

Метрики користувацького досвіду

Існує низка показників, що дозволяють оцінити рівень задоволеності користувачів сервісом. Наприклад, CSat (Customer Satisfaction Score) — це оцінка, яку користувач залишає додатку у вигляді зірочок у магазині застосунків; NPS (Net Promoter Score) — відображає, чи готовий користувач рекомендувати сервіс своїм знайомим і колегам; CSI (Customer Satisfaction Index) — показує співвідношення між очікуваннями, отриманою користю та якістю; індекс TRI*M враховує досвід взаємодії, готовність радити продукт і продовжувати його використання, а також загальне сприйняття переваг сервісу порівняно з конкурентами. Усі ці параметри можна вимірювати за допомогою спеціальних опитувань.

Окрім цього, існує безліч технічних метрик, що застосовуються для систем моніторингу. У більшості команд визначені SLI/SLO, за допомогою яких відстежується доступність (availability) сервісів; вимірюється та оптимізується затримка (latency), втрата пакетів, а також показники роботи застосунку — такі як PLT (Page Load Time), TTI (Time to Interactive) та LCP (Largest Contentful Paint).

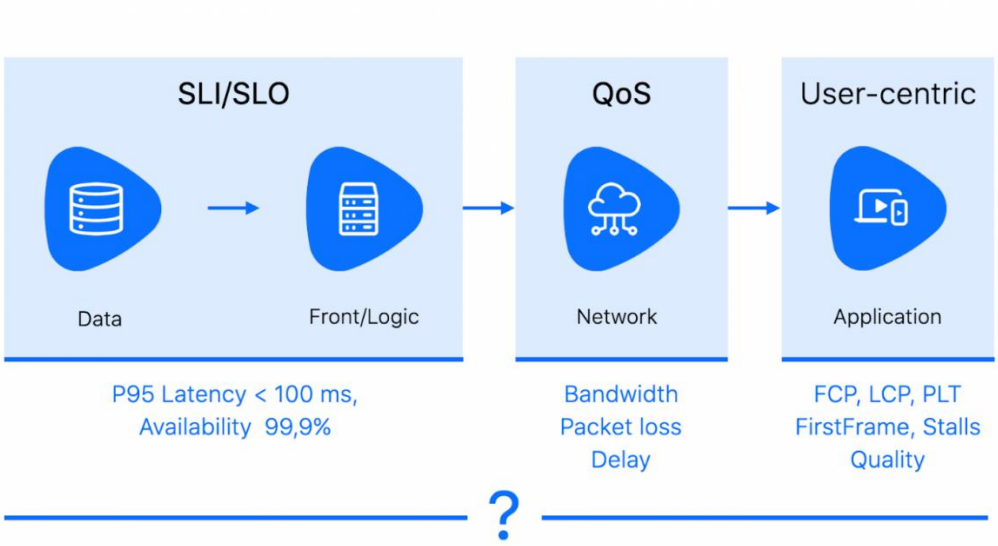


Рис. 1.1 Метрики для оцінки

Усі ці показники можна узагальнити за допомогою поняття QoE — Quality of Experience (якість користувацького досвіду). Це інтегральна метрика, що походить зі сфери телекомунікацій. Вона враховує різноманітні чинники, які впливають на сприйняття системи загалом: від технічних параметрів, таких як затримка та якість мережі, до характеристик самого користувача — країни проживання, типу пристрою, рівня освіти тощо. Зазвичай QoE проявляється у рівні залученості користувача (engagement) і пов'язаному з нею рівні задоволеності або «щастя» від користування. Очевидно, що користувач не буде активно використовувати сервіс, якщо він не відповідає його очікуванням. Водночас ступінь задоволення залежить також від ціни, репутації бренду та інших факторів.

QoE тісно пов'язана з продуктовими метриками. Її моніторинг дозволяє зрозуміти, як зміни в архітектурі системи або застосовані технологічні рішення впливають на сприйняття продукту користувачами. Найпростіший спосіб виміряти QoE — використовувати MOS (Mean Opinion Score), тобто періодично пропонувати користувачам оцінити якість сервісу за п'ятибальною шкалою — від «відмінно» до «погано».

Якщо зібрати достатню кількість таких оцінок у різних умовах, можна визначити залежність між рівнем задоволеності користувачів і певними технічними параметрами. Наприклад, у дослідженні, присвяченому QoE-орієнтованому відеострімінгу в LTE-мережах, показано кореляцію між значенням MOS та бітрейтом відео.

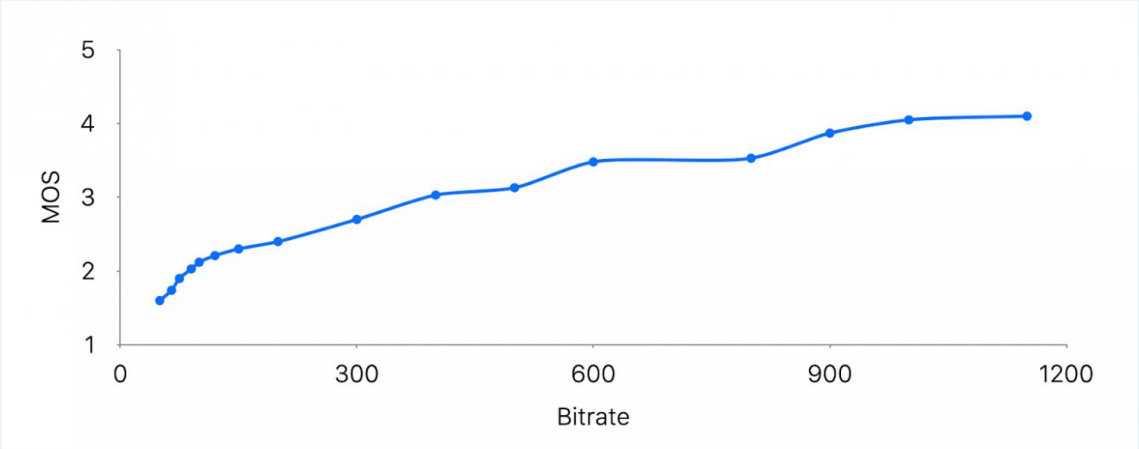


Рис. 1.2 Залежність MOS від бітрейту відео

В іншому дослідженні показник MOS оцінювали відносно частоти кадрів (FPS). Отримані результати виявилися суперечливими: різниця між 15 та 30 FPS суттєво впливає на сприйняття, тоді як підвищення з 30 до 60 FPS дає лише незначне покращення оцінки MOS. Частоти понад 60 FPS майже не впливають на якість користувацького досвіду. Проте ці дані датовані 2011 роком, і цілком можливо, що сьогодні користувачі стали уважнішими й більш чутливими до відмінностей у плавності відтворення.

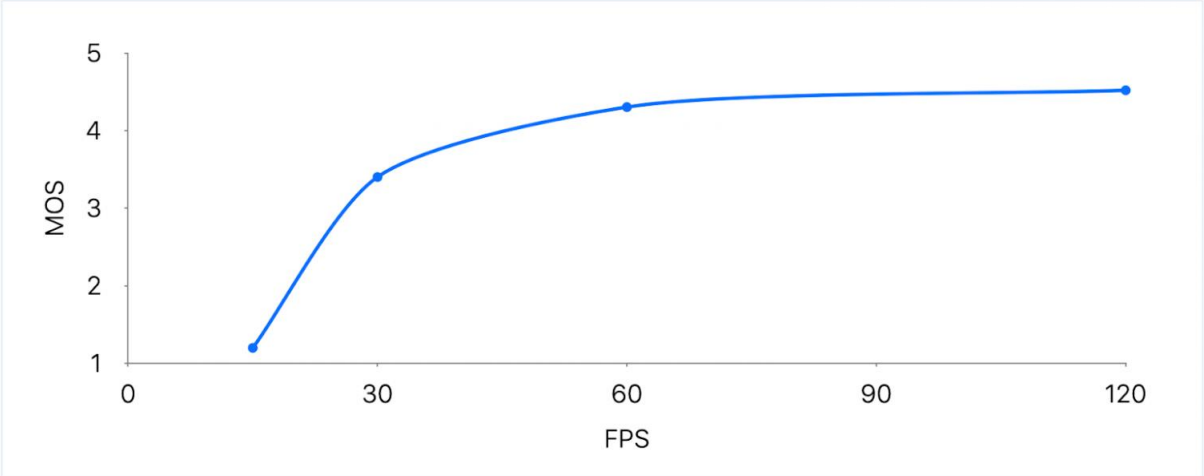


Рис. 1.3 Залежність MOS відносно частоти кадрів (FPS)

Очікування користувачів постійно зростають: те, що ще недавно вважалося нормою якості, з часом може сприйматися як недостатньо задовільне. Крім того, у певних умовах окремі метрики можуть суперечити одна одній. Наприклад, при обмеженій пропускній здатності мережі доводиться шукати компроміс між бітрейтом і частотою кадрів (FPS), і доцільно робити це на основі метрик та експериментальних вимірювань.

Підхід QoE застосовується також до вебсервісів. Зокрема, спостерігається, що для різних типів дій існують різні вимоги до часу завантаження сторінки (Page Load Time, PLT): зображення користувач очікує побачити миттєво, тоді як результати пошуку він готовий почекати трохи довше. Однак при затримці, що перевищує приблизно 200 мілісекунд, оцінка якості досвіду починає помітно знижуватися.

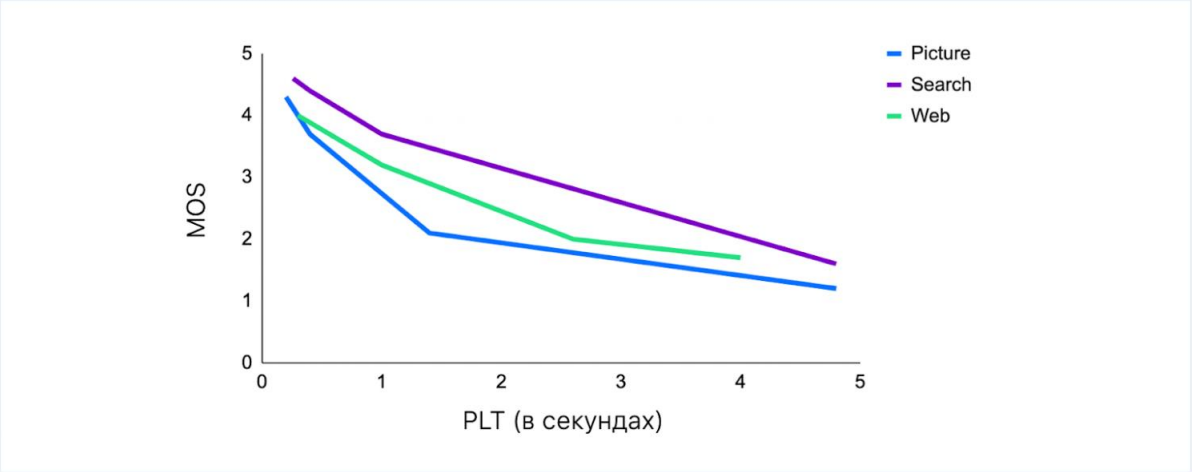


Рис. 1.4 Вимоги до Page Load Time (PLT)

Високі показники QoE самі по собі не гарантують успіху серед користувачів, проте без них завоювати лояльну аудиторію майже неможливо. Далі на прикладах буде розглянуто основні компоненти відеоплатформи та ключові характеристики, які безпосередньо впливають на якість користувацького досвіду.

1.2 Вимоги до завантаження

Завантаження: швидкість

Вимоги користувачів до процесу завантаження та скачування файлів чудово ілюструє мультфільм, створений понад двадцять років тому.

Відтоді істотно змінилися як розміри файлів, так і очікування щодо якості та швидкості завантаження. Сучасні сервіси мають забезпечувати стійкість до збоїв і можливість відновлення перерваного завантаження — як цього досягти, докладно розглядається в іншій статті. Тут же постає питання швидкості: що вважати «швидко»?

Згідно з аналітичними даними, якщо невеликий файл не завантажується протягом 5–7 секунд, користувач починає дратуватися. На наведеному нижче графіку показано приклад для файлу розміром 25 Мбайт — це приблизно коротке відео, кілька знімків або невелика презентація. Також видно, що навіть для більших обсягів ніхто не готовий чекати дві хвилини на завершення завантаження.

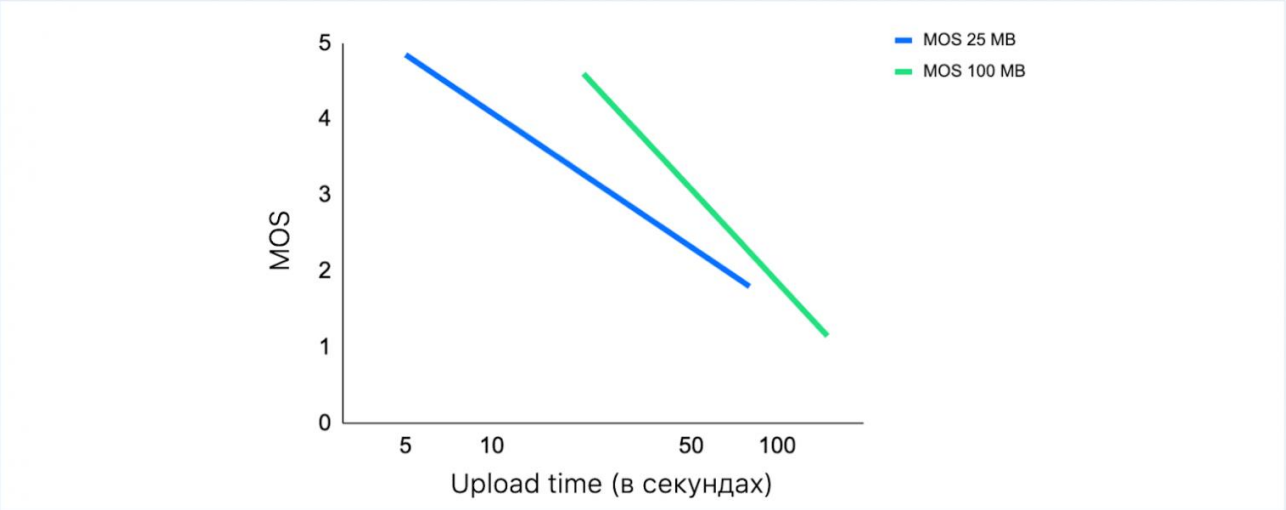


Рис. 1.5 Приклад для файлу розміром 25 Мбайт

Звичайний користувач не прагне й не зобов'язаний розуміти, у чому саме причина проблеми: у нестабільному з'єднанні, слабких характеристиках пристрою чи в недоліках самого сервісу. Найімовірніше, він просто залишиться незадоволеним роботою застосунку.

Щоб підвищити швидкість передавання даних (upload), необхідно ефективніше використовувати пропускну здатність мережі. Можливі підходи до цього обмежуються кількома основними варіантами: HTTP/1.1, HTTP/2 та HTTP/3.

HTTP/1.1 і завантаження за допомогою чанків

Принцип роботи цього методу полягає в тому, що великий файл ділиться на невеликі частини — так звані чанки. Спочатку клієнт надсилає перший блок даних через POST-запит і чекає відповіді від сервера з кодом 200 OK. Лише після цього надсилається наступний блок у новому запиті. Саме на етапі очікування відповіді виникає затримка, пов'язана з RTT (Round Trip Time) — часом, який потрібен для повного циклу запиту й відповіді між клієнтом і сервером.

Наприклад, якщо розмір кожного чанка становить 4 Мбайти, то для завантаження файлу обсягом 1 Гбайт потрібно передати 256 таких частин. Кожен із них викликає додаткову затримку RTT, унаслідок чого загальний час очікування збільшується. При середньому RTT близько 150 мілісекунд це дає приблизно 38 секунд додаткової затримки лише на очікування відповідей від сервера. І це ще не

найгірший сценарій, адже в деяких випадках RTT може сягати й 400 мілісекунд — наприклад, при слабкому мобільному сигналі або роботі через нестабільні мережі.

Таким чином, хоча HTTP/1.1 є перевіреним і сумісним майже з усіма системами протоколом, його механізм послідовного завантаження чанків створює значні втрати часу. Для великих файлів це може призводити до помітного погіршення користувацького досвіду, тому в сучасних рішеннях усе частіше застосовують новіші версії протоколу, які дозволяють передавати дані ефективніше.

HTTP/2 і мультиплексування

У порівнянні з HTTP/1.1, де передавання кожного запиту відбувається послідовно, протокол HTTP/2 дозволяє істотно підвищити швидкодію завдяки механізму мультиплексування. Його суть полягає в тому, що в межах одного TCP-з'єднання можна одночасно передавати кілька потоків даних. Це означає, що різні частини файлу або навіть декілька файлів можуть завантажуватися паралельно без потреби чекати завершення попереднього запиту. Завдяки цьому загальний час завантаження суттєво скорочується, а пропускна здатність каналу використовується ефективніше.

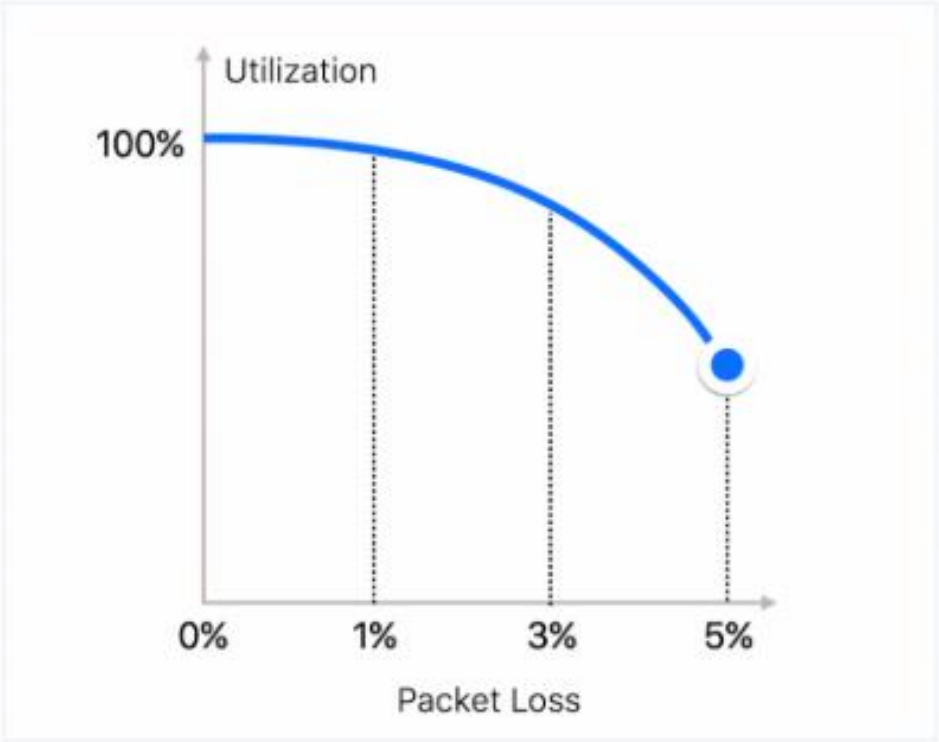


Рис. 1.6 Кореляція зниження ефективності утилізації мережевого каналу до збільшення втрати пакетів

Однак у HTTP/2 є й слабке місце — чутливість до втрати пакетів. Якщо в мережі виникають навіть незначні втрати, ефективність використання каналу різко знижується. Це пов’язано з тим, що всі потоки в межах одного TCP-з’єднання залежать від цілісності даних: коли один із пакетів губиться, передавання решти потоків частково блокується, доки втрачені дані не будуть повторно доставлені. Така поведінка суттєво впливає на стабільність швидкості, особливо в мобільних або нестабільних мережах, де втрата пакетів є звичайним явищем.

Таким чином, хоча HTTP/2 значно перевершує HTTP/1.1 у плані швидкості та оптимізації, його ефективність безпосередньо залежить від якості мережевого середовища. При стабільному з’єднанні цей протокол забезпечує помітне прискорення завантаження, але в умовах високої латентності чи частих втрат пакетів переваги мультиплексування можуть практично зникати. Саме тому в сучасних системах дедалі частіше переходять на HTTP/3, який покликаний вирішити ці недоліки.

HTTP/1.1 і паралельне завантаження

Після серії експериментів було зроблено висновок, що одним із найефективніших способів прискорення передавання даних є використання паралельного завантаження. Суть методу полягає в тому, щоб відкривати одразу кілька з’єднань із сервером — стільки, скільки дозволяє браузер або застосунок, — і завантажувати різні частини файлу одночасно в декілька потоків, зазвичай у 4–6. Такий підхід дозволяє максимально задіяти пропускну здатність мережі та зменшити загальний час очікування завершення передачі.

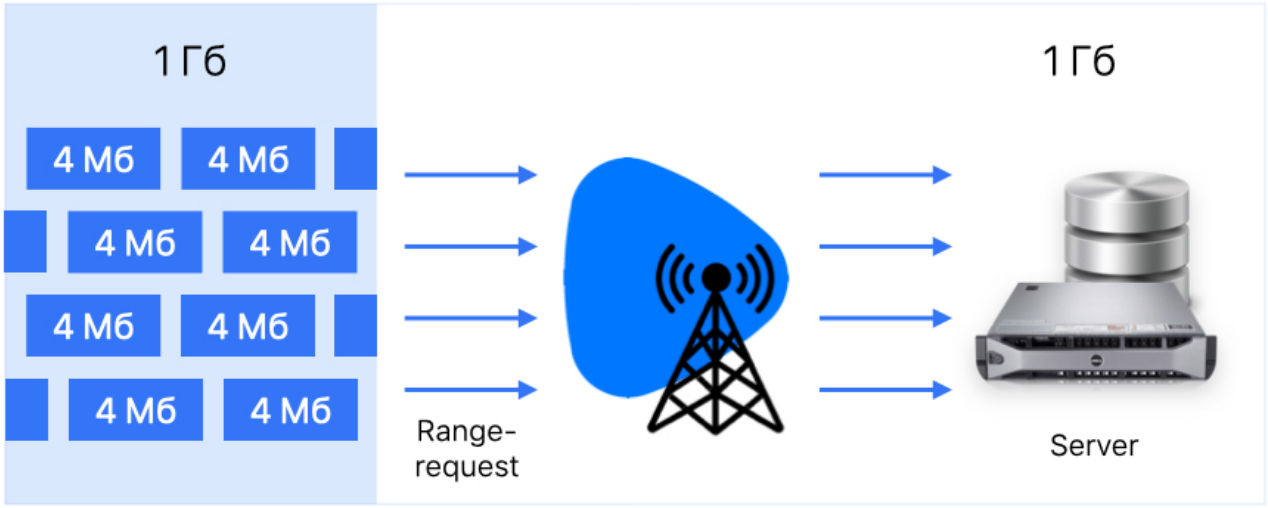


Рис. 1.7 Принцип паралельного завантаження

Попри те, що HTTP/1.1 вважається застарілим протоколом, цей спосіб показує дуже хороші результати: за швидкістю він не поступається HTTP/3, а іноді навіть перевершує його. Це пояснюється тим, що паралельне відкриття декількох TCP-з'єднань дозволяє ефективніше розподіляти трафік, використовуючи так званий «нечесний» (не fair) підхід до спільного каналу зв'язку. Інакше кажучи, клієнт отримує більше пропускної здатності, ніж при справедливому розподілі між потоками, що забезпечує кращу продуктивність.

Крім того, TCP усе одно залишається необхідним для резервного сценарію (fallback), тому навіть при використанні новіших протоколів, таких як HTTP/3, наявність можливості швидкого паралельного завантаження у межах HTTP/1.1 є корисною.

Якщо ж немає часу або ресурсів на розробку власного завантажувача, можна скористатися готовими відкритими рішеннями. Один із найпопулярніших варіантів — бібліотека tus, яку зручно інтегрувати з S3-сховищами. Хоча така комбінація може потребувати додаткового налаштування та оптимізації для досягнення стабільної якості сервісу, вона вже підтримує функціонал паралельного завантаження, що дозволяє швидко впровадити цей підхід у робочий процес.

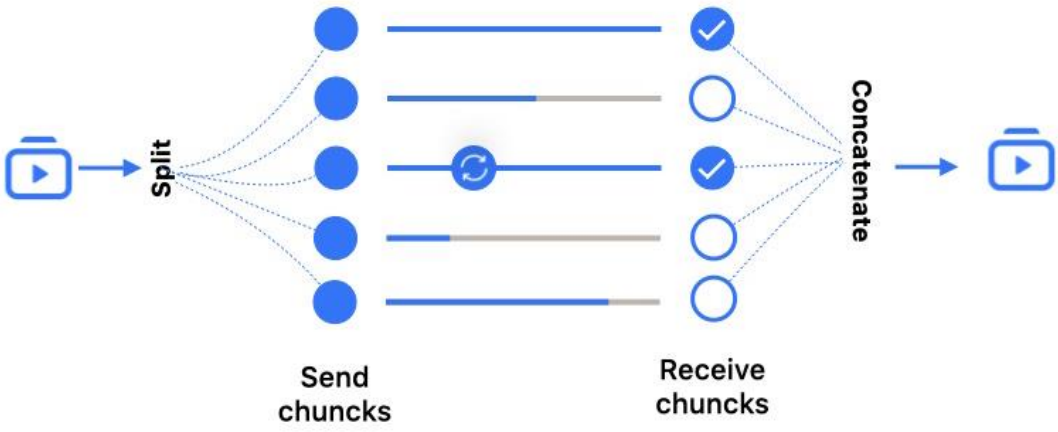


Рис. 1.8 Відкрита бібліотека tus

Що може піти не так під час завантаження?

Коли система обробляє мільйони нових відеофайлів щодня — скажімо, понад 14 мільйонів роликів на добу, — будь-яка потенційна проблема рано чи пізно проявиться. У таких масштабах навіть незначна технічна дрібниця може перерости у серйозну помилку, що вплине на стабільність роботи всієї платформи.

Розглянемо приклад із паралельним завантаженням, коли файли передаються частинами — чанками. Припустімо, ми розподіляємо їх між потоками за принципом «роздачі карт»: перший потік отримує перший, п’ятий, дев’ятий чанк і далі за чергою, другий — другий, шостий, десятий тощо. Така схема здається зручною, але вона має серйозний недолік — якщо один із потоків раптово зупиниться або почне працювати повільніше, у структурі файлу виникає багато пропусків. Список таких «дір» для повторного завантаження може стати настільки великим, що перевищить максимальний розмір заголовка HTTP. У результаті сервер просто не зможе обробити запит повторної передачі.

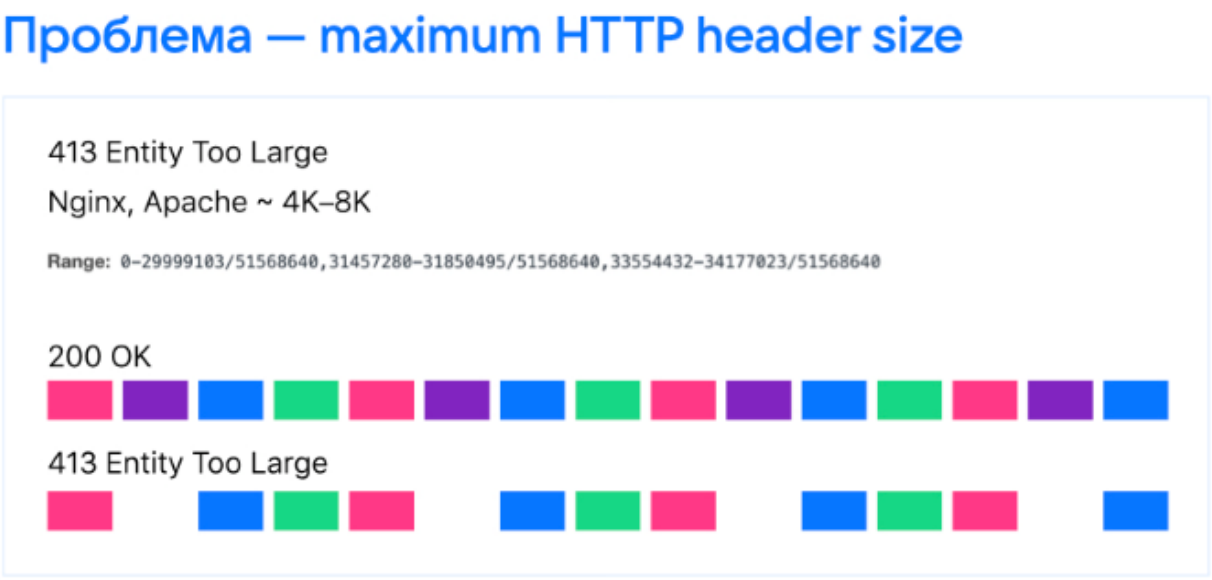


Рис. 1.9 Проблема – maximum HTTP header size

Розмір HTTP-заголовків має жорсткі обмеження, які залежать від конкретного вебсервера — nginx, Apache чи іншої системи. Навіть якщо ліміт досить великий, він усе одно обмежений, тому при перевищенні допустимого обсягу завантаження зупиняється з помилкою. Щоб уникнути цього, у продуманих системах чанки розподіляють між потоками інакше — не через пропуски, а послідовно, тобто кожен потік отримує найближчі ще не передані блоки даних. Це дозволяє мінімізувати кількість повторних запитів і запобігти переповненню HTTP-заголовків.

Ще одна потенційна проблема — DNS-запити. Вона трапляється рідко, проте може виявитися дуже підступною. Через особливості роботи DNS і параметр TTL (time to live) два послідовні запити на розв’язання одного й того самого доменного імені можуть повернути різні IP-адреси — причому навіть у різних дата-центрах. У такій ситуації різні частини одного файлу завантажуються в різні фізичні місця, що не лише сповільнює процес, а й може створити колізії, коли одні й ті самі чанки передаються паралельно в різні вузли. Це призводить до додаткових затримок і дублювання даних.

Побудова високонавантаженої та відмовостійкої системи вимагає врахування таких сценаріїв. Необхідно передбачити механізми, які зможуть виявляти подібні

розбіжності в DNS-відповідях, перенаправляти запити у правильний дата-центр і синхронізувати стан передавання даних між серверами.

Завдяки комплексному підходу до оптимізації процесу завантаження — від корекції логіки розподілу чанків до вдосконалення мережевої інфраструктури та балансування запитів — нам вдалося суттєво підвищити надійність системи. У результаті рівень успішних завантажень зріс на 7%, що прямо вплинуло на зростання залученості користувачів і загальний показник QoE. Це доводить, що навіть невеликі поліпшення в архітектурі процесу передавання даних здатні забезпечити помітний приріст якості користувацького досвіду.

1.3 Вимоги на прикладі відео

Специфіка відеофайлів

Відео складається з окремих кадрів. Якщо передавати його у вигляді послідовності повних кадрів без стиснення, то для відтворення відео у форматі 4K із частотою 60 кадрів на секунду знадобиться канал із пропускною здатністю близько 18 Гбіт/с. Таку швидкість може забезпечити, наприклад, HDMI-кабель на кілька метрів, але не звичайна мережа доступу в Інтернет. Саме тому відео передається у закодованому вигляді: лише частина кадрів зберігається повністю, тоді як решта кодується на основі відмінностей від попередніх кадрів. Завдяки цьому обсяг переданих даних зменшується в сотні разів — і для того самого 4K-відео достатньо каналу лише на 50 Мбіт/с.

Кожні кілька років з’являються нові кодеки, які приблизно вдвічі підвищують ефективність стиснення порівняно з попередніми, але водночас потребують більше обчислювальних ресурсів для кодування й декодування. Серед відкритих стандартів найбільш відомими є VP8 та VP9, які розробляє переважно компанія Google. До ліцензованих форматів належать H.264, H.265 і H.266, що використовуються в професійному обладнанні та комерційних продуктах. Окрему групу становить AV1 — сучасний відкритий кодек, який створюється «Альянсом за

відкриті медіа», до складу якого входять такі технологічні гіганти, як AMD, Nvidia, Apple, Google, Microsoft та інші.

Завдяки постійному вдосконаленню алгоритмів стиснення відео стає доступним у дедалі вищій якості без істотного збільшення навантаження на мережу, що безпосередньо впливає на зручність користувача та показники QoE.

Зазвичай, чим старіший кодек, тим ширше його підтримка серед пристроїв і програм. Тому у великих сервісах із різноманітною аудиторією необхідно забезпечувати сумісність одразу з кількома форматами та вміти автоматично перемикатися між ними залежно від умов відтворення.

Звичні розширення файлів — MP4, (MKV), 3GP — насправді не є самими відеоформатами, а виступають контейнерами. Специфікація контейнера визначає спосіб зберігання мультимедійних даних у межах одного файлу. По суті, контейнер містить відео- й аудіопотоки, упаковані в певній структурі, що дозволяє перемотувати запис, починати відтворення з будь-якого моменту або зручно обробляти окремі фрагменти під час трансляції.



Рис. 1.10 Відео в контейнері

Транскодування: якість

Щодня користувачі завантажують на платформу мільйони нових відео у різних кодеках, контейнерах і роздільностях. Завдання сервісу полягає в тому, щоб ці відео коректно відтворювалися на будь-яких пристроях і за будь-якої якості мережі. Для цього файли необхідно перетворювати у всі потрібні формати — саме цей процес і називається транскодуванням.

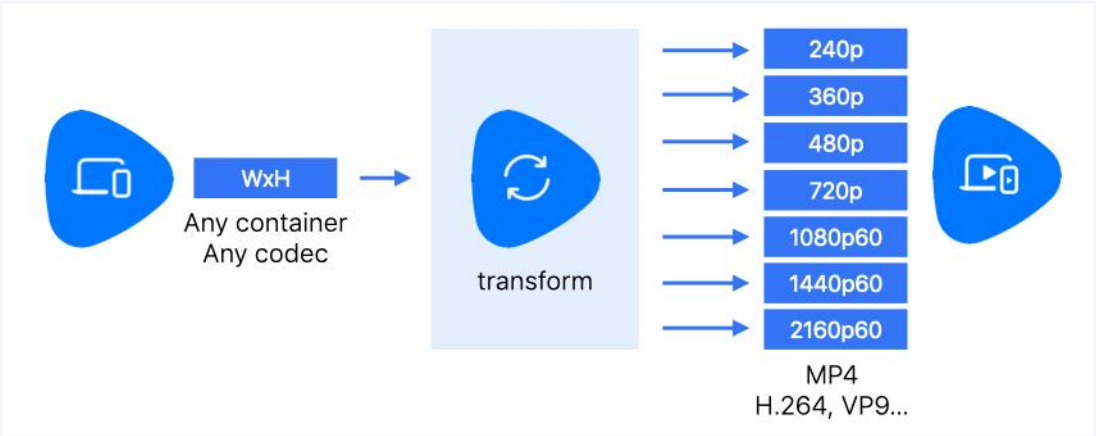


Рис. 1.11 Процес транскодування

Для оцінювання якості відео використовують спеціальні метрики, серед яких найпоширенішими є PSNR, SSIM та VMAF. Кожна з них по-своєму відображає рівень спотворень і ступінь відповідності зображення або відео оригіналу після стиснення.

Метрика PSNR (Peak Signal-to-Noise Ratio) — це пікове відношення сигналу до шуму, яке найчастіше розраховується через середньоквадратичну похибку між оригінальним і відтвореним зображенням. Вона показує, наскільки сильно стиснення вплинуло на точність відтворення пікселів. Проте PSNR оцінює лише фізичні відхилення, не враховуючи того, як ці спотворення сприймає людське око.

Індекс SSIM (Structural Similarity Index) має інший підхід: він не просто порівнює яскравість чи контраст, а враховує структурні зміни в зображенні, що дозволяє краще відтворити людське сприйняття якості. На відміну від PSNR, SSIM оцінює не «різницю у сигналах», а ступінь подібності структур, тому вважається більш «перцептивною» метрикою — тобто ближчою до того, як людина реально бачить відмінності між оригіналом і стисненим відео.

Ще точніше якість суб’єктивного сприйняття моделює VMAF (Video Multimethod Assessment Fusion) — метрика, розроблена компанією Netflix у 2016 році. Її особливість у тому, що вона поєднує результати кількох різних методів аналізу, створюючи «зведену» оцінку, яка максимально наближена до людського судження. VMAF використовується для прогнозування того, як глядач сприйматиме відео після стиснення, і сьогодні є одним із галузевих стандартів для тестування якості потокового відео.

Загалом усі ці метрики мають спільну мету — відтворити те, як людина оцінює якість зображення чи відео після кодування. На графіках зазвичай порівнюють змодельовані оцінки з реальними, отриманими від користувачів. По горизонтальній осі відкладають MOS (Mean Opinion Score) — середню суб’єктивну оцінку якості для різних зразків відео, а по вертикальній — результат, передбачений метрикою. Чим ближче точки розташовані до діагоналі, тим точніше модель передбачає реальне сприйняття користувачів. Відповідно, менша відстань між ними свідчить про вищу точність метрики та її кращу придатність для практичного використання.

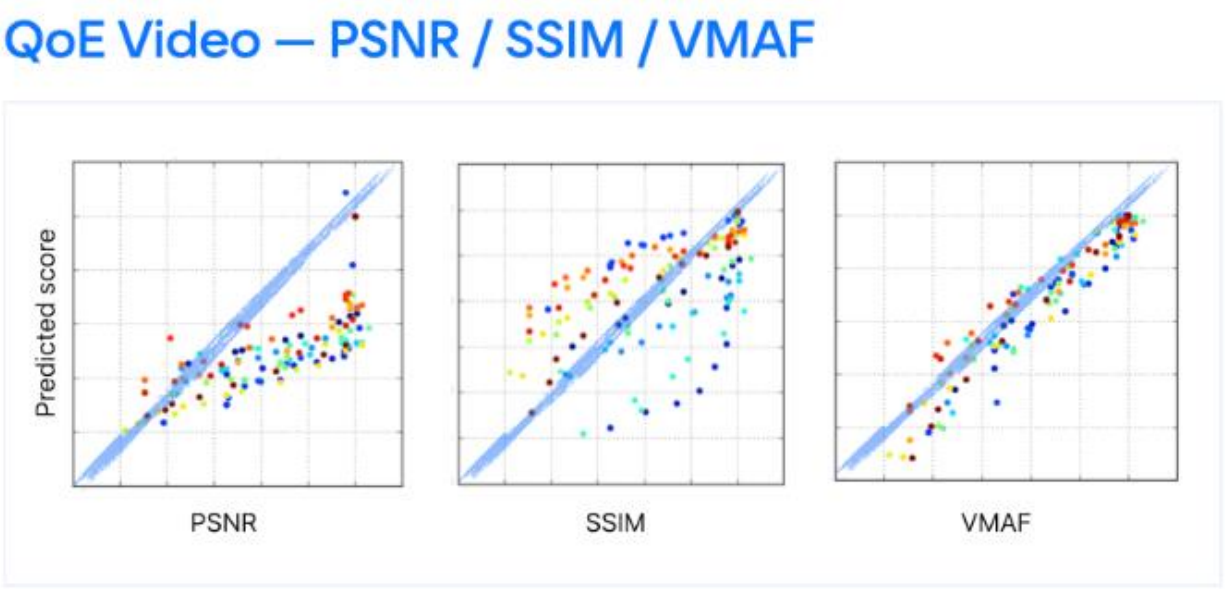


Рис. 1.12 Метрики для оцінки відео

Миттєва доступність

Після того як було оптимізовано процеси завантаження та транскодування, ми помітили, що користувачі можуть починати перегляд відео значно швидше, ніж

на більшості інших платформ. Спочатку цей результат здавався очевидним успіхом — адже швидший старт відтворення напряму покращує показники QoE. Проте відгуки авторів контенту виявили іншу сторону ситуації. Вони справедливо запитували: «Чому моє відео не доступне одразу після завантаження? Чому інші можуть його побачити лише через кілька хвилин?» Для деяких категорій контенту, наприклад новинного, така затримка дійсно є критичною — кожна секунда має значення.

Тому ми переглянули підхід до всього процесу і впровадили транскодування за запитом (on-demand transcoding). Це означає, що відео можна відтворювати одразу у вихідному форматі, а якщо користувачеві або пристрою потрібне інше розділення чи кодек, система динамічно обробляє його «на льоту». Фактично цей підхід нагадує механізм прямої трансляції, тільки застосований до великих відеофайлів — обсягом у кілька гігабайтів — і з можливістю почати перегляд із будь-якого місця у файлі.

Схема роботи виглядає доволі елегантно. Система спершу читає заголовок файлу, визначає структуру контейнера та знаходить потрібні семпли. Потім ці дані передаються у транскодер, побудований на основі ffmpeg, де здійснюється швидке перекодування лише запитаних фрагментів. Результат одразу віддається серверу, який формує відеосегменти для відтворення клієнтом. На даному етапі система підтримує лише формат MP4, оскільки понад 80 % усіх завантажень відбувається саме в ньому. Якщо ж користувач завантажує відео в іншому контейнері, наприклад MKV, файл спочатку проходить процес ремуксингу в MP4. Це трохи збільшує час обробки, проте все одно залишається швидшим за повне транскодування.

Спочатку ідея транскодування «на вимогу» здавалася нереалістичною — здавалося, що обробка терабайтів даних у реальному часі створить непосильне навантаження. Але завдяки ефективному кешуванню вдалося мінімізувати ресурсні витрати. Навіть при великій кількості запитів використання обчислювальних потужностей не перевищує приблизно 4 000 vCore. Крім того, підхід забезпечив ще одну вагому перевагу: зниження обсягу збережених даних на 40–50 %. Тепер у

різних роздільностях зберігаються лише найпопулярніші відео, тоді як для решти достатньо оригінального файлу, який перекодовуються лише за необхідності.

У підсумку, рішення про впровадження миттєвої доступності відео дало подвійний ефект: значно покращило користувацький досвід і водночас оптимізувало витрати ресурсів. Такий баланс між швидкодією, якістю обслуговування та ефективністю використання інфраструктури став одним із ключових кроків у розвитку сервісу.

2 ОПТИМІЗАЦІЯ НАВАНТАЖЕННЯ НА МЕРЕЖУ ЗА ДОПОМОГОЮ ДИФЕРЕНЦІАЛЬНИХ ОНОВЛЕНЬ IOT

2.1 Оновлення snap пакетів

До Інтернету речей (IoT) підключається дедалі більше пристроїв, і разом із цим зростають обсяги мережевого трафіку. Під час розгортання периферійних систем необхідно заздалегідь оцінити, як часто та в яких обсягах дані передаватимуться між пристроями та хмарною інфраструктурою, де вони аналізуються. Якщо таких пристроїв тисячі, витрати на передавання даних стають відчутними. Крім того, збільшення обсягів трафіку може знизити ефективність роботи IoT-рішень, спричиняючи затримки та перевантаження каналів зв’язку.

Наслідки проблем із пропускною здатністю або затримками залежать від конкретного сценарію використання, частоти обміну й обсягу даних. Оскільки IoT-рішення дуже різноманітні, у деяких випадках час передавання може бути несуттєвим, а обсяг даних настільки малим, що затримкою можна знехтувати. Втім, навіть за таких умов організаціям варто обирати механізм передавання, який буде водночас ефективним, надійним і економічно доцільним.

Дані в IoT мають не лише бізнесову цінність — вони також забезпечують актуальність, безпеку й стабільність роботи віддалених систем. З огляду на це, доцільним рішенням є використання програмного забезпечення, яке передає лише змінені частини (так звані диференціальні оновлення), а не весь пакет даних чи застосунок повністю. Для цього чудово підходить формат snap — універсальна система пакування застосунків для Linux. Snap-пакети підтримуються більш ніж у 40 дистрибутивах Linux, а сама операційна система є найпоширенішою платформою для вбудованих пристроїв, тому цей формат ідеально відповідає вимогам до розгортання програмного забезпечення в середовищах IoT.

У данному розділі розглядається, як саме можна застосовувати диференціальні snap-пакети для ефективного та автоматичного передавання

оновлень по безпроводовому зв’язку (OTA — over-the-air) на IoT-пристрої, суттєво зменшуючи споживання мережевого трафіку.

Оновлення snap-пакетів на IoT-пристроях

Використання snap-пакетів на системах Ubuntu Core, Ubuntu Classic та інших дистрибутивах Linux надає низку важливих переваг для підключених пристроїв та рішень у сфері Інтернету речей (IoT). Однією з головних переваг є можливість безпечного транзакційного оновлення. Це означає, що система виконує оновлення як цілісну операцію: якщо під час процесу виникає помилка, попередня стабільна версія автоматично відновлюється без втручання користувача. Завдяки цьому ризики збою або пошкодження системи під час оновлення практично зводяться до нуля.

Така модель значно спрощує впровадження автоматичних оновлень, оскільки інфраструктура snap самостійно обробляє більшість типових сценаріїв помилок. Розробникам і постачальникам програмного забезпечення не потрібно створювати власні механізми резервного відновлення чи ручного керування процесом оновлення — система бере це на себе. У результаті компанії можуть зосередитися на основному завданні — розробці, тестуванні й випуску актуальних версій своїх продуктів.

Snap-пакети також відкривають широкі можливості для масового розгортання оновлень. Постачальники програм можуть випускати нові версії своїх застосунків і розгортати їх на тисячах або навіть мільйонах пристроїв упродовж кількох годин без будь-яких дій із боку користувачів. Це особливо важливо для IoT-інфраструктур, де пристрої розташовані на віддалених майданчиках або розкидані по всьому світу.

Для аналізу ефективності таких оновлень зазвичай використовують спеціальні моніторингові графіки. На вертикальній осі відображається частка пристроїв, які працюють із певною версією snap-пакета, а на горизонтальній — дати за останні три місяці. Кожна версія програмного забезпечення позначається власним кольором, що дозволяє наочно відстежувати швидкість поширення оновлень.

Як показує статистика, після виходу нової версії snap-пакета оновлення на більшості пристроїв відбувається майже миттєво. Уже впродовж перших кількох днів активність оновлення стрімко зростає, а через тиждень не менше 90 % усіх IoT-пристроїв переходять на актуальну версію програмного забезпечення. Така оперативність забезпечує високу узгодженість систем, підвищує рівень безпеки, спрощує технічну підтримку та гарантує стабільну роботу мережі з мінімальними витратами ресурсів.

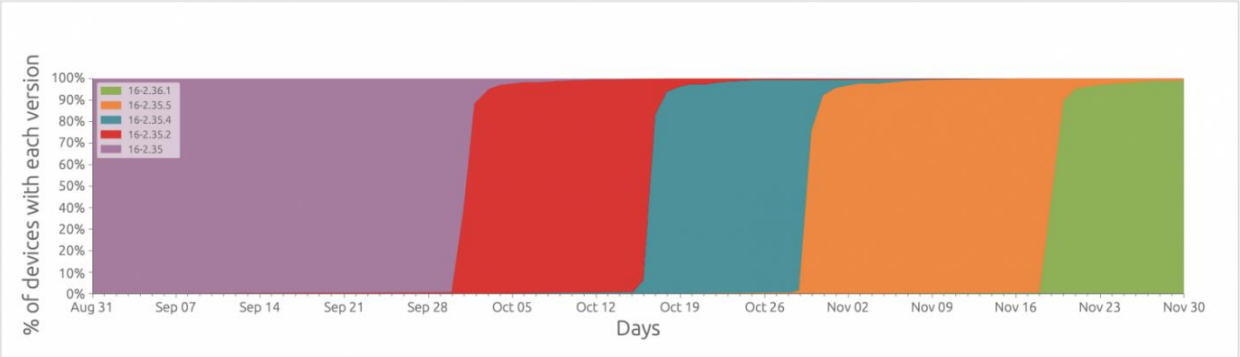


Рис. 2.1 Частота випусків та поширення оновлень для snap-пакету Core

На рисунку показано частоту виходу та динаміку розповсюдження оновлень для snap-пакета Core. Це спеціальний системний пакет, який містить базовий набір бібліотек, необхідних для роботи більшості застосунків, і встановлюється на всі пристрої, що використовують технологію snap. Зазвичай нові версії цього пакета випускаються раз на місяць, однак у разі критичних оновлень або виявлення вразливостей оновлення може виходити позапланово. Як видно з графіка, після публікації нової версії приблизно 90% пристроїв отримують оновлення вже протягом першого тижня, що свідчить про високу ефективність механізму автоматичного розповсюдження snap-оновлень.

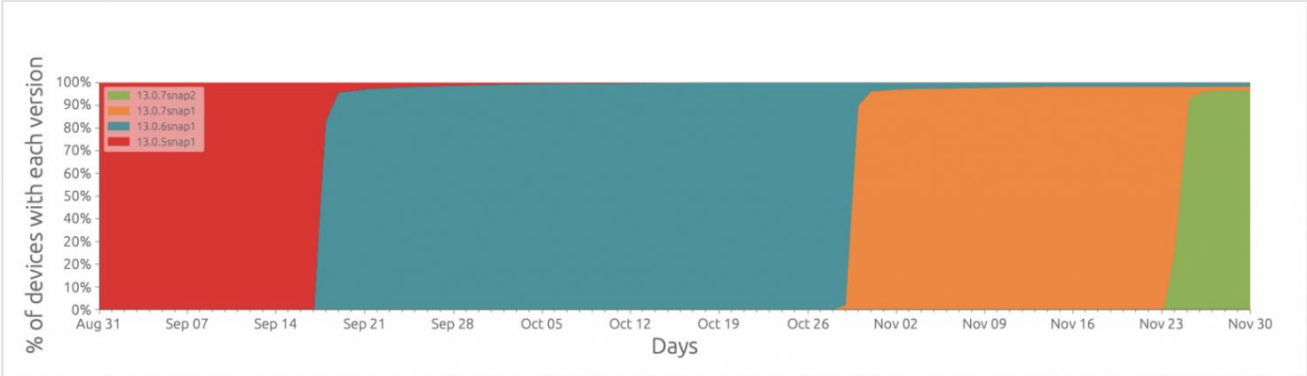


Рис. 2.2 Дані для програмного забезпечення Nextcloud для власних хмарних сервісів

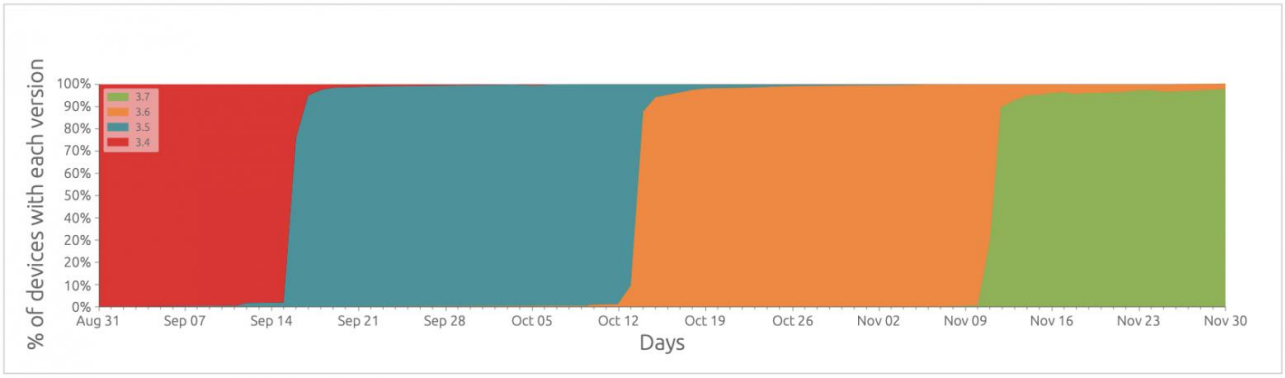


Рис. 2.3 Дані для програми Canonical LXD

Для операторів і виробників обладнання це означає, що оновлення систем безпеки або нові функції можуть встановлюватися одночасно на весь парк пристроїв, незалежно від їхнього місцезнаходження чи кількості користувачів. Такий підхід гарантує узгодженість програмного забезпечення, оперативне усунення вразливостей і стабільну роботу всієї інфраструктури IoT.

Однак, якщо обсяг оновлення вимірюється сотнями мегабайт або більше, процес передавання може створити значне навантаження на мережу й зайняти тривалий час. Цей аспект стає особливо важливим у масштабних розгортаннях, коли йдеться про тисячі пристроїв або про системи, що працюють у середовищах із обмеженою пропускною здатністю — наприклад, у віддалених локаціях чи при передаванні даних через стільникову мережу. У таких випадках навіть незначна оптимізація трафіку може істотно вплинути на стабільність і швидкість оновлення.

Щоб зменшити обсяг передаваних даних під час оновлення та водночас не втрачати надійність, у системі snap реалізовано механізм диференціальних snap-

пакетів. Він працює за принципом передавання лише змінених частин пакета, а не всієї програми цілком. Таким чином, якщо оновлення торкнулося лише невеликої частини коду чи конфігурацій, передається лише цей фрагмент, а не повний пакет. Це дозволяє істотно знизити споживання трафіку — іноді в кілька разів — і прискорити процес розгортання нових версій.

Такий підхід особливо ефективний для IoT-пристроїв, які працюють автономно або в умовах нестабільного з’єднання. Завдяки диференціальному оновленню система мінімізує кількість переданих даних, забезпечує безперервність роботи й скорочує витрати на обслуговування мережі. У підсумку оператори отримують швидші оновлення, стабільнішу роботу інфраструктури та економію ресурсів як на боці сервера, так і на боці клієнтських пристроїв.

2.2 Диференціальні snap пакети

Що таке диференціальні snap-пакети?

Диференціальні snap-пакети (або «дельта-оновлення») — це спеціальний механізм, який дозволяє передавати лише різницю між двома версіями пакета: тією, що вже встановлена на пристрої (вихідною), і тією, яку потрібно встановити (цільовою). Іншими словами, під час оновлення пристрій не завантажує повний обсяг пакета повторно — передаються тільки змінені файли або сегменти, що дає змогу суттєво скоротити обсяг трафіку та прискорити процес оновлення.

Диференціальні snap-пакети застосовуються лише тоді, коли оновлюється вже інстальований пакет. Зазвичай початкові версії snap-пакетів попередньо встановлюються на пристрої ще на етапі виробництва — наприклад, під час заводського налаштування. Надалі пристрої отримують лише оновлення, що робить процес максимально швидким і надійним.

Усі snap-пакети завантажуються з Snap Store — централізованого репозиторію, який використовується для публікації, пошуку та поширення програмного забезпечення у форматі snap. Коли видавець пакета (у випадку IoT-пристроїв це зазвичай виробник обладнання) випускає нову версію свого snap-

пакета, сервери Snap Store автоматично аналізують, яка версія найімовірніше встановлена на більшості пристроїв, і створюють диференціальні оновлення між цими двома версіями. У результаті пристрої, на яких уже є вихідна версія, отримують лише оновлену частину файлів, після чого встановлений пакет повністю відповідає новій цільовій версії — так, ніби він був завантажений цілком, але з мінімальними витратами трафіку.

Такий підхід можна використовувати як для оновлення (upgrade), так і для відкату (rollback) до попередніх версій. Це особливо корисно в середовищах, де важлива стабільність системи — наприклад, у промислових або медичних пристроях, де кожна помилка може бути критичною.

Важливо, що робочий процес розробників при цьому не змінюється. Створення диференціальних пакетів не потребує жодних додаткових дій із боку видавців: усі необхідні оновлення формуються автоматично на серверах Snap Store. Розробники можуть публікувати свої пакети будь-яким зручним способом — локально через Snapcraft (інструмент командного рядка для упаковки та публікації snap-пакетів), за допомогою build.snapcraft.io, через electron-builder або через інтеграцію з системами безперервної інтеграції, такими як Travis CI чи інші CI/CD-сервіси. Незалежно від методу публікації, кінцеві користувачі отримують оптимізовані диференціальні оновлення, які відповідають їхній поточній версії системи.

Застосування диференціальних оновлень на пристрої супроводжується невеликою затримкою, пов’язаною з процесом обробки змін. Проте ця затримка зазвичай становить лише кілька секунд і не впливає на роботу системи навіть на малопотужних IoT-пристроях. Натомість вигода у вигляді зменшеного обсягу трафіку та скороченого часу оновлення значно переважає такі незначні затримки.

У підсумку диференціальні snap-пакети стали ключовою технологією для ефективного оновлення пристроїв Інтернету речей. Вони забезпечують швидке розгортання, економію пропускної здатності, мінімізацію часу простоїв і стабільність усієї системи навіть у великих розподілених інфраструктурах.

У спеціальній таблиці наведено приклади стандартних snap-пакетів різного розміру, відношення обсягу диференціального файлу до повного пакета, а також середній час, необхідний для виконання оновлення на IoT-пристроях і звичайних ПК-системах. Ці дані демонструють, наскільки суттєвим може бути виграш у швидкості та зниженні навантаження на мережу при використанні диференціальних оновлень.

Таблиця 2.1

Стандартні snap пакети

Ім'я snap-пакета	**Розмір повного файлу**	**Розмір диференціального файлу**	**Співвідношення розмірів (повний = 100 %)**	**Час застосування оновлення (Broadcom BCM2837 ARM Cortex-A53 USB)**	**Час застосування оновлення (Intel Core i7-4500U 1.80 ГГц SSD)**
alsa-utils	6,5 MiB	62,8 KiB	0,94 %	0,45 с	0,045 с
nextcloud	197 MiB	112 MiB	56,85 %	24,0 с	1,16 с
core	87,9 MiB	27,55 MiB	31,34 %	10,75 с	0,42 с
pc-kernel	154,77 MiB	54,20 MiB	35,01 %	18,9 с	0,788 с

Ефективність диференціальних оновлень залежить від багатьох чинників. У середньому розмір диференціального файлу становить близько 30 % від повного пакета. На наведеному графіку показано середні значення розмірів повних і диференціальних snap-пакетів, розраховані за добовими вибірками протягом 4,5 місяців. Це наочно демонструє співвідношення між повним пакетом та відповідним йому диференціальним файлом.

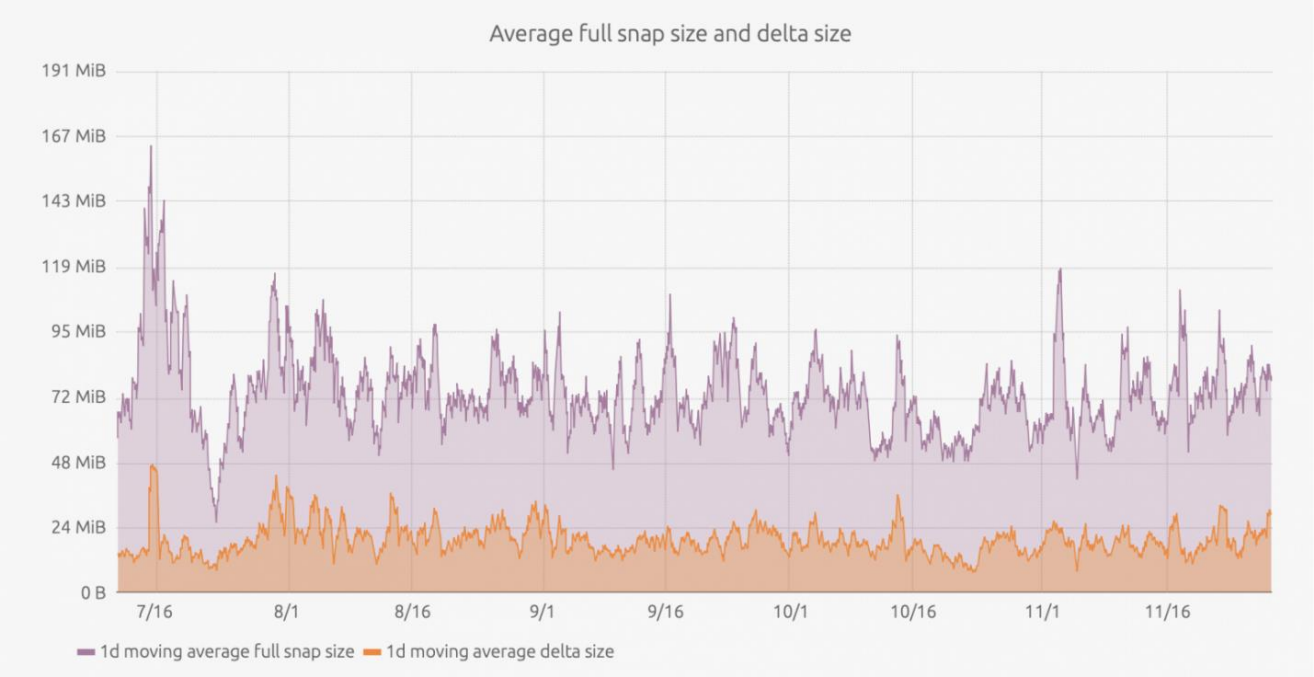


Рис. 2.4 Середні розміри повних і диференціальних snap-пакетів (ковзне добове середнє) протягом 4,5 місяців

Надсилання та створення диференціального файлу

Технологія диференціальних snap-пакетів забезпечує найбільший ефект саме під час завантаження оновлень на клієнтські пристрої, однак її можливості не обмежуються лише цим. Вона також дозволяє значно зменшити витрати часу та мережевого трафіку на етапі публікації нових версій програмного забезпечення розробниками.

У типовому сценарії розробник, підготувавши оновлення застосунку, створює нову версію snap-пакета та надсилає її до Snap Store — офіційного сховища, де публікуються та зберігаються всі пакети у форматі snap. Зазвичай при такій публікації потрібно було б передати весь пакет цілком, навіть якщо зміни в коді мінімальні. Проте завдяки використанню механізму диференціальних

оновлень інструмент Snapcraft автоматично формує спеціальний файл, що містить лише відмінності між поточною версією в Snap Store і новою версією, яку створює розробник.

Цей «дельта»-файл передається до Snap Store, після чого сервер виконує зворотну операцію — застосовує зміни до вже наявної версії пакета та перевіряє цілісність отриманого результату за допомогою цифрового підпису. У підсумку формується повноцінний snap-пакет, який повністю відповідає новій версії, але створений на основі часткових оновлень, без потреби передавати весь обсяг даних.

Такий підхід дозволяє розробникам економити не лише трафік, а й час, адже обсяг переданих даних може бути у кілька разів меншим, ніж при звичайному завантаженні. Крім того, сам процес оновлення відбувається прозоро, без змін у звичному робочому циклі: розробникам не потрібно виконувати додаткові дії чи змінювати інструменти. Snapcraft автоматично обробляє створення диференціального пакета, а Snap Store — його перевірку, збирання та розгортання.

Як і у випадку з клієнтськими пристроями, цей процес гарантує надійність та точність результату: оновлений пакет, зібраний із диференціальних даних, абсолютно ідентичний повній версії. Таким чином, використання диференціальних оновлень стає ефективним рішенням не лише для економії ресурсів користувачів, а й для оптимізації всієї інфраструктури розробки та публікації програмного забезпечення у форматі snap.

Диференціальні snap-пакети на персональних комп’ютерах

Механізм диференціальних snap-пакетів дозволяє економити мережевий трафік не лише в середовищах Інтернету речей (IoT), а й на звичайних настільних комп’ютерах. Цю технологію можна ефективно використовувати у будь-якій операційній системі на базі Linux. Наприклад, у Ubuntu 18.10 за замовчуванням уже встановлено кілька системних snap-пакетів, а користувач має можливість додатково завантажувати десятки класичних застосунків, які легко інсталюються та автоматично оновлюються через Snap Store.

Одразу після встановлення Ubuntu система самостійно перевіряє наявність оновлень для попередньо встановлених пакетів. У звичайному сценарії, без

використання диференціальних механізмів, розмір повного оновлення становив би близько 293 МБ. Проте завдяки технології диференціальних оновлень завантажується лише 69,5 МБ даних — тобто майже у чотири рази менше. Незважаючи на невеликий розмір, після оновлення користувач отримує повністю актуальні версії пакетів, ідентичні тим, які встановлювалися б у повному обсязі.

Якщо уявити масштаб такої економії на рівні всієї організації, де сотні або тисячі комп'ютерів використовують Ubuntu, обсяг зекономленого трафіку буде вражаючим. Це особливо важливо у корпоративних мережах, де канал зв'язку обмежений або використовується для критичних службових процесів. У таких випадках диференціальні snap-пакети не лише зменшують навантаження на мережу, а й підвищують стабільність оновлень, запобігаючи зривам через нестачу пропускної здатності.

Крім того, на робочих станціях із Ubuntu Desktop великі програми часто постачаються у вигляді snap-пакетів — наприклад, офісні пакети, редактори, браузери та засоби розробки. Це означає, що кожне подальше оновлення може супроводжуватись суттєвими обсягами передавання даних. Використання диференціальних оновлень у цьому контексті забезпечує стабільну роботу системи, швидке встановлення нових версій і мінімізацію витрат трафіку навіть на персональних ПК.

У наступній таблиці наведено приклади основних snap-пакетів, які постачаються разом із системою Ubuntu Desktop, що ще раз підкреслює практичність і універсальність технології диференціальних оновлень.

Таблиця 2.2

Приклади основних snap-пакетів

Ім'я snap-пакета	**Середнє число оновлень на місяць**	**Середній розмір повного пакета (МБ)**	**Середній розмір диференціального пакета (МБ)**	**Трафік за місяць без диференціальних (ГБ)**	**Трафік за місяць із диференціальними (ГБ)**
0ad	0.58	897.46	185.32	0.510	0.105
brave	2.11	179.81	93.01	0.370	0.191
bw (bitwarden)	1.01	14.75	4.73	0.015	0.005
core *	1.81	82.08	31.28	0.145	0.055
darktable	0.37	92.96	32.38	0.034	0.012
firefox	2.89	182.48	85.09	0.516	0.240
gimp	1.21	155.18	32.32	0.183	0.038
gitkraken	2.91	161.75	35.08	0.460	0.100
gnome-3-26-1604 *	1.32	133.33	25.19	0.172	0.033
gnome-calculator *	1.57	2.61	1.06	0.004	0.002
gnome-characters *	1.72	15.92	0.94	0.027	0.002
gnome-logs *	1.50	18.18	4.95	0.027	0.007
gnome-system-monitor *	1.66	5.52	0.92	0.009	0.001

Продовження таблиці 2.2

Ім'я snap-пакета	**Середнє число оновлень на місяць**	**Середній розмір повного пакета (МБ)**	**Середній розмір диференціального пакета (МБ)**	**Трафік за місяць без диференціальних (ГБ)**	**Трафік за місяць із диференціальними (ГБ)**
gtk-common-themes *	1.60	36.24	7.18	0.057	0.011
jgalaxian	0.57	87.93	21.19	0.049	0.012
notepad-plus-plus	2.95	34.70	1.98	0.100	0.006
ruby	4.29	32.24	14.56	0.135	0.061
sftpclient	8.85	207.05	64.90	1.789	0.561
skype	5.81	139.70	42.26	0.793	0.240
slack	0.54	129.42	56.22	0.068	0.030
taskbook	0.68	23.81	12.36	0.016	0.008
tmnationsforever	0.97	125.68	8.31	0.119	0.008
vscode	0.57	200.06	6.18	0.111	0.003
vlc	1.92	110.50	27.95	0.207	0.052
wifi-ap	0.26	25.86	9.48	66.12	24.25
fictional-snap	1.12	175.48	51.56	1921.05	564.46
			Сума	5.394	1.682
			Економія в %	68.82%	

Система, на якій встановлено всі наведені snap-пакети з відповідною частотою оновлення, споживатиме приблизно на 68,8 % менше мережевого трафіку завдяки використанню диференціальних оновлень. Це демонструє, наскільки ефективною може бути дана технологія навіть у масштабі звичайного робочого середовища. Вона зменшує навантаження на мережеву інфраструктуру, скорочує час оновлення системи та робить розгортання програмного забезпечення більш стабільним і прогнозованим.

Диференціальні snap-пакети у хмарних середовищах

Як було показано раніше, диференціальні snap-пакети забезпечують значну економію трафіку в системах IoT та на персональних комп'ютерах. Проте їхнє застосування не обмежується лише цими напрямами — ця технологія також має своє місце в хмарних розгортаннях, де операційні системи з підтримкою snap (наприклад, Ubuntu) часто використовуються як базова інфраструктура.

У хмарних середовищах зазвичай доступна значно більша пропускна здатність каналів зв'язку, тому економія трафіку не є головним завданням. Проте навіть у таких умовах диференціальні оновлення можуть бути корисними — особливо у великих корпоративних або гібридних інфраструктурах, де відбуваються регулярні масові оновлення контейнерів, віртуальних машин чи базових образів систем.

Для публічних хмарних платформ, таких як Amazon Web Services (AWS), Canonical забезпечує спеціальний механізм локального кешування snap-пакетів. Завдяки цьому оновлення не завантажуються з глобального Snap Store щоразу, а передаються безпосередньо в межах самої хмарної інфраструктури. Це скорочує затримки, знижує навантаження на зовнішні канали та підвищує швидкість оновлення тисяч віртуальних машин одночасно.

У приватних (on-premise) хмарах або корпоративних середовищах можна встановити Canonical Enterprise Proxy — спеціальний сервер, який виконує роль локального кешу snap-пакетів. Він дає ті самі переваги, що й у публічних хмарах: усі системи в мережі отримують оновлення безпосередньо з локального сховища, без потреби кожного разу звертатися до центрального Snap Store.

Варто зазначити, що технологія диференціальних snap-пакетів працює повністю прозоро — користувачу або адміністратору не потрібно налаштовувати її вручну. Якщо система підтримує цю функцію, оновлення автоматично оптимізуються, незалежно від типу середовища — чи то IoT-пристрої, настільні системи, чи хмарні сервери.

Таким чином, навіть у випадках, коли пропускна здатність мережі не є вузьким місцем, застосування диференціальних snap-пакетів залишається доцільним. Вони підвищують ефективність оновлень, забезпечують узгодженість версій у великих розгортаннях і зменшують навантаження на інфраструктуру. Це робить їх важливою складовою сучасних систем керування програмним забезпеченням у хмарних екосистемах.

3 ПЕРЕДЧАСНА ОПТИМІЗАЦІЯ АРХІТЕКТУРИ

3.1 Проблеми при плануванні ТК архітектур

Звідки виникають збої у телекомунікаційних системах і архітектурах?

Основні причини більшості проблем у телекомунікаційних архітектурах мають системний характер. Перша й найпоширеніша — це оновлення або впровадження нових версій сервісів та додатків. Коли змінюється програмний код або логіка обробки запитів у вузлах мережі, часто виникають непередбачувані конфлікти з існуючими компонентами. Результатом стають помилки в маршрутизації, відмова елементів управління трафіком чи навіть повне зупинення роботи окремих сервісів.

Друга типова причина — неконтрольоване зростання навантаження та проблеми масштабованості. У міру розвитку телекомунікаційної мережі або під час пікових подій (наприклад, великих акцій, масових підключень, аварій у сусідніх сегментах) різко збільшується кількість користувацьких запитів і обсяг трафіку. Якщо архітектура не має адаптивного механізму розподілу навантаження або резервних ресурсів, навіть короткочасний стрибок трафіку може призвести до деградації продуктивності, затримок або збоїв у доступі до сервісів.

Третя причина, яку часто недооцінюють, — помилки проєктування самої архітектури мережі. Вони не проявляються одразу, але з часом стають джерелом серйозних аварій. Це можуть бути неузгодженості між рівнями моделі OSI, неефективне розташування вузлів обробки даних, неправильний вибір топології або відсутність гнучкості при інтеграції нових технологій. У таких випадках система виходить з ладу не через перевантаження чи збій коду, а через структурні недоліки, закладені ще на етапі проєктування.

Таким чином, стійкість телекомунікаційної архітектури визначається не лише надійністю обладнання чи якістю програмного забезпечення, а передусім якістю архітектурних рішень. Грамотне планування мережевих рівнів, прогнозування навантаження, використання модульних рішень і гнучких протоколів

масштабування дозволяють уникнути більшості типових проблем і забезпечити безперебійну роботу навіть у складних умовах зростання мережі.

Помилки в проєктуванні телекомунікаційних архітектур

Якщо порівняти галузь телекомунікацій із традиційними інженерними сферами, можна помітити разючу різницю. Після завершення будівництва мосту або хмарочоса ми майже ніколи не бачимо, щоб споруда раптово зруйнувалася. Коли прокладають водогін, він, як правило, служить роками, не виходячи з ладу одразу після запуску. Натомість у телекомунікаційних і IT-системах ситуація часто зовсім інша: створюється архітектура, розгортається нова інфраструктура — і вже під час введення в експлуатацію виявляється, що вона не відповідає реальним умовам роботи, не масштабується або має критичні недоліки.

Причина цього полягає в тому, що телекомунікаційна галузь є відносно молодою порівняно з класичними інженерними дисциплінами. Вона ще не має такого обсягу усталених практик, методик і стандартів проєктування, які десятиліттями напрацьовувалися в будівництві чи енергетиці. Крім того, сучасні телекомунікаційні рішення постійно ускладнюються: з’являються нові рівні в архітектурі, додаткові програмні компоненти, механізми віртуалізації, автоматизації, балансування тощо. Кожен новий рівень складності збільшує ризик збоїв і знижує надійність експлуатації.

Чим більше компонентів має система — тим вищою стає ймовірність відмови. Якщо, наприклад, імовірність збою окремого вузла становить лише 5 %, а іншого — 20 %, то сумарний ризик буде не 20 %, а близько 24 %. При зростанні кількості таких вузлів цей показник може збільшуватись експоненційно, створюючи критичні ситуації, коли навіть незначний збій одного елемента здатен паралізувати роботу всієї мережі.

Виникає парадокс: чим складнішою стає телекомунікаційна архітектура, тим важче її підтримувати і забезпечувати її стійкість. Без ретельного планування, резервування, сегментації трафіку й моделювання навантаження навіть найсучасніші рішення можуть призвести до непередбачуваних наслідків.

Причини виникнення надмірної складності в телекомунікаційних архітектурах

У класичних інженерних галузях майже ніхто навмисно не створює складні системи. Інженер, який проєктує будівлю або електромережу, не ставить собі за мету зробити конструкцію заплутанішою — навпаки, прагне до стабільності, простоти та передбачуваності. У сфері інформаційних технологій, зокрема в телекомунікаціях, ситуація часто інша: під час супроводу або аудиту мережевої архітектури можна побачити рішення, складність яких не має чіткої технічної логіки. Часто система виявляється «перепроєктованою» просто тому, що колись було цікаво спробувати щось нове або «експериментальне».

Однією з головних причин цього є унікальність задач, які вирішуються в галузі телекомунікацій. Не завжди існує готове рішення, яке можна просто взяти й застосувати. Коли інженери стикаються з новою проблемою — наприклад, нестандартним способом маршрутизації трафіку, розподілом навантаження чи оптимізацією каналів у гетерогенному середовищі — вони змушені самостійно шукати вихід. Часто ці спроби супроводжуються помилками, переробками й додатковими рівнями коду або конфігурацій, які з часом накопичуються та ускладнюють архітектуру.

Інша причина — швидкий розвиток технологій і нестача усталених практик. У той час як авіаційні або енергетичні інженери навчаються десятиліттями перевіреним методам, у сфері IT ситуація зовсім інша: технології змінюються настільки швидко, що університети просто не встигають навчати актуальним інструментам. Тому інженери часто змушені вчитися на власних помилках і вигадувати рішення з нуля, навіть якщо подібні задачі вже колись вирішувалися іншими. Як наслідок, архітектура системи стає більш громіздкою, виникають надлишкові залежності, а ризики відмов і помилок — вищими.

Є й третій, не менш поширений чинник — людський інтерес і прагнення до експериментів. Телематика й програмна інженерія приваблюють людей, яким справді цікаво працювати з новими технологіями. Іноді бажання «спробувати щось нове» бере гору над прагматизмом. Наприклад, команда вирішує інтегрувати

контейнеризацію Docker, хоча для конкретного завдання це не є необхідним. Такі ініціативи виникають із щирого ентузіазму, але часто створюють додаткову складність у системі, яку потім важко підтримувати.

У підсумку, зайва складність архітектури часто є наслідком не помилок у проєктуванні, а поєднання творчого підходу, відсутності перевірених стандартів і прагнення інженерів до самореалізації через нові технології. Щоб уникнути цього, телекомунікаційним компаніям варто формувати культуру архітектурної дисципліни: документувати рішення, дотримуватись єдиних стандартів, оцінювати технічну доцільність змін і впроваджувати лише ті технології, які реально покращують ефективність системи, а не просто додають їй «цікавості».

3.2 Основні серверні проблеми

У процесі розвитку телекомунікаційних систем настає момент, коли проєкт починає стрімко зростати, а разом із цим — збільшується навантаження на сервери. У цей період постає низка типових викликів, серед яких найважливішими є горизонтальне масштабування та резервування.

Коли один сервер уже не справляється з обсягом трафіку, необхідно передбачити можливість рівномірного розподілу навантаження між кількома вузлами системи. Це дозволяє підтримувати стабільну роботу сервісів навіть під час пікових навантажень. Водночас важливо забезпечити коректну синхронізацію даних між усіма компонентами, щоб уникнути розривів у логіці роботи або втрати інформації.

Другий критичний аспект — відмовостійкість. Адміністратори мають бути впевнені, що у випадку виходу з ладу основного сервера система продовжить функціонувати без простоїв. Для цього впроваджують механізми автоматичного перемикання на резервні вузли, реплікацію даних і кластеризацію сервісів.

Отже, навіть якщо інфраструктура побудована на потужному виділеному обладнанні, без продуманого плану масштабування та резервування вона залишатиметься вразливою. Грамотно спроєктована телекомунікаційна архітектура

має передбачати зростання навантаження, забезпечувати гнучкість розширення та гарантувати безперервність роботи у будь-яких умовах.

Балансування навантаження та резервування у телекомунікаційних архітектурах

У сучасних телекомунікаційних системах навантаження на вебсервіси зазвичай розподіляється між кількома екземплярами (інстансами) вебсерверів, що забезпечує стабільність роботи та кращу масштабованість. Аналогічно, для підвищення відмовостійкості створюється кілька серверів баз даних, які синхронізуються між собою через механізм реплікації. Це дозволяє розподіляти запити на читання й запис між різними вузлами, зменшуючи затримки та підвищуючи ефективність роботи системи.

Проте типовою помилкою залишається те, що всі вебсервери розміщуються в одному дата-центрі, часто навіть у межах однієї серверної стійки. З технічного погляду це створює єдину точку відмови: навіть якщо вебвузли продубльовані, збій у роботі самого дата-центру або проблеми з енергоживленням зроблять усю систему недоступною. Тому одне з базових правил побудови надійної архітектури полягає у тому, що резервні інстанси повинні розміщуватись у географічно віддаленому дата-центрі, бажано з незалежним каналом живлення й з’єднанням.

Ще один важливий аспект — наявність резервування не означає наявності резервних копій (бекапів). Часто системи мають дублюючі сервери, на які можна перенаправити трафік у разі збою, однак це не гарантує збереження даних у разі пошкодження основного вузла. Саме тому резервне копіювання залишається окремою обов’язковою частиною архітектури.

Щодо віртуалізації — попри її очевидні переваги, вона часто створює додаткові труднощі. Віртуальні середовища схильні до затримок, конфліктів при масштабуванні, проблем із дисковими підсистемами або мережею. Тому надмірна залежність від віртуалізованих рішень може знизити стабільність телекомунікаційної інфраструктури.

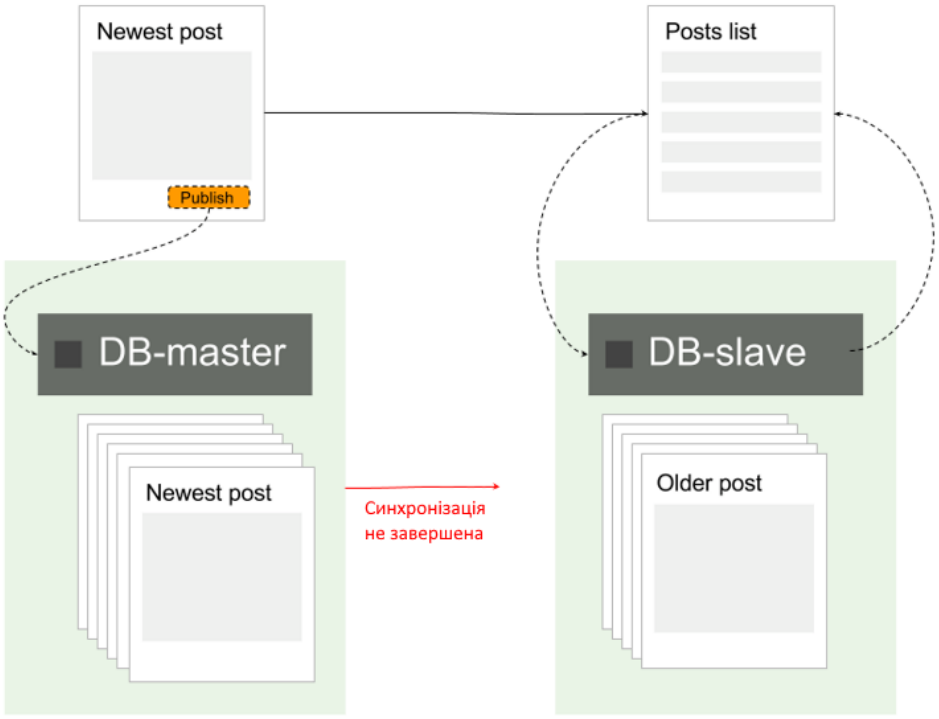


Рис. 3.1 Розподілення навантаження між базами даних

Типові проблеми при використанні кількох серверів баз даних

Коли використовується модель із master/slave-реплікацією, часто виникає ситуація, коли дані, записані на основному сервері (master), ще не встигли оновитись на допоміжному (slave). У цей момент система може видати неактуальні дані при запиті на читання. Це типова проблема затримки реплікації, яка може призвести до порушення цілісності даних.

Щоб уникнути цього, необхідно постійно контролювати стан реплікації через систему моніторингу, зокрема:

- перевіряти активність реплікації (чи працює вона взагалі);
- відстежувати затримку реплікації (іноді дані можуть передаватись із запізненням на години або навіть доби);
- перевіряти консистентність даних між master і slave, щоб переконатися, що копії синхронізовані.

Навіть синхронна реплікація не дає повної гарантії узгодженості, оскільки на практиці можуть виникати тимчасові розриви або відставання через мережеві затримки. Саме тому схема «записати на master — прочитати зі slave» хоч і поширена, але вважається ризикованою для критично важливих систем.

Балансування навантаження та точки відмови

При розподілі навантаження між кількома вебвузлами найчастіше використовується єдиний балансувальник навантаження, на який вказує головна DNS-запис (A-record). Він приймає весь вхідний трафік і далі передає його між вебвузлами. Проблема в тому, що такий балансувальник сам стає точкою відмови: у разі його збою весь проєкт припиняє роботу, оскільки немає компонента, який міг би перенаправляти запити.

Щоб уникнути цього, балансування має бути реалізовано з урахуванням механізму failover. Це означає, що балансувальник або керуючий застосунок повинні постійно опитувати вузли системи, щоб визначати, які з них працюють, а які вийшли з ладу. Якщо один із серверів перестає відповідати, його потрібно автоматично виключати з пулу активних вузлів. В іншому випадку частина користувачів отримуватиме помилки або неповний контент, тоді як інші — доступ до робочої частини системи.

Отже, ефективна телекомунікаційна архітектура повинна бути побудована з урахуванням географічного резервування, активного моніторингу, відмовостійкого балансування та постійного контролю реплікації даних. Лише такий підхід гарантує стабільність, безперервність обслуговування та захист від непередбачених збоїв.

Файлові системи та керування даними в телекомунікаційній інфраструктурі

Сучасні телекомунікаційні проєкти оперують величезною кількістю медіафайлів — відео, аудіо, зображень, логів та конфігураційних даних. Зрозуміло, що для ефективної роботи всі вебвузли (web-nodes) повинні мати доступ до спільного сховища, у якому зберігаються ці файли. Ідеально, коли кожен вузол може читати та записувати інформацію незалежно, але при цьому всі зміни одразу синхронізуються між серверами.

Найпростішим рішенням часто вважають NFS (Network File System) — класичну технологію мережевого доступу до файлів, яка існує вже десятиліттями. Вона дійсно зручна у налаштуванні та добре відома системним адміністраторам, тому її часто використовують для створення спільних сховищ. Проте в масштабованих архітектурах NFS має глобальні недоліки.

Якщо, наприклад, зникає зв’язок між master-сервером NFS та підключеними вебвузлами, або ж потрібно перезавантажити сервер із мережею NFS, усі вузли втрачають доступ до даних. Монтування точки зависає, і система фактично «заморожується», поки не відбудеться фізичне перезавантаження. Це типова та відома проблема NFS, яка не має повноцінного вирішення в межах самої технології.

Організація деплою проєктів

На початкових етапах розробки, коли проєкт невеликий, можна обійтися простими сценаріями оновлення коду, наприклад, через git pull і скрипт, який зберігає попередні версії. Такий підхід простий, надійний і не потребує складної інфраструктури.

Однак із розвитком системи з’являється спокуса впровадити CI/CD (Continuous Integration / Continuous Delivery). Це дає змогу автоматизувати процес оновлень, але також додає значної складності. Часто малі або середні проєкти намагаються реалізувати CI/CD раніше, ніж це справді потрібно, витрачаючи ресурси на надмірну автоматизацію, яка створює нові ризики.

Типова помилка — відсутність можливості швидкого відкату (rollback). Якщо після оновлення коду з’являється критична помилка, команда намагається повернути стару версію, але база даних уже пройшла нові міграції, які не сумісні зі старим кодом. У результаті виникає ситуація, коли дані частково оновлені, частково — ні, і система опиняється в стані логічної нестабільності.

Щоб уникнути таких проблем, необхідно тестувати систему деплою на безпечних середовищах і перевіряти можливість відкату перед кожним релізом. У більшості випадків для середніх телекомунікаційних систем доцільніше залишатися на простому, але контрольованому процесі оновлень через скрипти.

Пошук альтернатив NFS

Коли проєкт зростає, виникає потреба замінити NFS на більш стійке рішення. Серед найпопулярніших варіантів — Ceph і MooseFS.

- Ceph — потужна система розподіленого зберігання з високим рівнем надійності, але складна у конфігурації. Для її ефективної роботи потрібен глибокий досвід і багато часу на налаштування.
- MooseFS — легша альтернатива, простіша у розгортанні, добре підходить для середніх систем. Вона розподіляє дані між кількома вузлами та забезпечує прозорий доступ до файлів.

Проте навіть у стабільних системах можливі непередбачувані збої — наприклад, відключення живлення у дата-центрі або пізній запуск резервного генератора, що призводить до втрати доступу до частини файлів. Саме тому важливо не лише обирати файлову систему, а й продумувати архітектуру з відмовостійким сховищем, кешуванням і регулярними бекапами.

Приклад коду для простого деплою через Bash

Нижче наведено приклад простого скрипта, який можна використати для безпечного оновлення коду без складних CI/CD систем:

```
#!/bin/bash
# Простий деплой коду з можливістю відкату

PROJECT_DIR="/var/www/project"
BACKUP_DIR="/var/backups/project_$(date +%F_%H-%M-%S)"

echo "=== Створення резервної копії ==="
cp -r $PROJECT_DIR $BACKUP_DIR

echo "=== Отримання останньої версії з Git ==="
cd $PROJECT_DIR
git pull origin main

echo "=== Оновлення завершено ==="
echo "Якщо потрібно відкотитися, використайте:"
echo "rm -rf $PROJECT_DIR && cp -r $BACKUP_DIR $PROJECT_DIR"
```

Рис. 3.2 Скрипт для безпечного оновлення коду без складних CI/CD систем

Такий підхід забезпечує мінімальні ризики при оновленні телекомунікаційних сервісів: він дозволяє швидко зробити резервну копію, оновити код і повернутися до попередньої версії у разі помилки.

Таким чином, ефективна робота з файлами у телекомунікаційних архітектурах передбачає збалансоване поєднання простоти, стабільності та відмовостійкості. Не завжди варто гнатись за модними технологіями — іноді правильно налаштоване базове рішення працює надійніше за складні системи, що вимагають постійного контролю.

Проблеми з файловими системами, середовищами розгортання та навантаженням у телекомунікаційних архітектурах

Після певного часу експлуатації великої телекомунікаційної системи часто з’являються неочевидні, але критичні проблеми. Наприклад, система накопичує десятки терабайт статичних файлів — зображень, відео, журналів, які розподілені між кількома серверами. Після збою електроживлення або короткочасного відключення, усі ці сервери мають синхронізуватись між собою. І тут починається справжнє випробування.

Синхронізація може тривати кілька днів, а на великих об’ємах даних (понад 20–30 ТБ) часто зривається. Наприклад, у системі MOOSEFS після чотирьох днів роботи процес синхронізації може обірватися на 90% і почати спочатку. У результаті проєкт п’ять днів залишається без доступу до статичного контенту — тобто без зображень, відео, CSS і JS-файлів. Така ситуація неприпустима для будь-якої продуктивної архітектури.

Навіть якщо в інтернеті можна знайти рішення проблеми — скажімо, на китайських форумах чи технічних блогах, — покладатися на такі джерела в корпоративних або критичних телекомунікаційних середовищах небезпечно. Відсутність офіційної підтримки та нестабільність файлової системи перетворюють її експлуатацію на лотерею.

Типові проблеми під час деплою

Коли система деплою вже налагоджена, з’являються інші ризики — випадки помилок у production. Часто після оновлення коду користувачі бачать помилки 500

через баги, які не проявлялись на тестових (dev) середовищах. Причини цього зазвичай три:

1. Різниця у розмірах баз даних. На dev середовищі база має 50 МБ і тисячу записів, а на prod — 10 ГБ і мільйон записів. Запит, який на тестовій базі виконується за 0,1 секунди, на продакшні може працювати десятки секунд, створюючи затримки й таймаути.
2. Різниця у навантаженні. Dev середовище зазвичай не має паралельних запитів чи фонових процесів, тому результати тестів не відображають реальної поведінки системи під навантаженням.
3. Різниця у конфігураціях. У production може бути відсутній якийсь модуль, бібліотека чи розширення PHP, що призводить до помилок, яких не було у тестовому середовищі.

Додатково можуть впливати й відмінності у “залізі”: dev-сервер має одне ядро на 4 ГГц, а prod — багато ядер по 2 ГГц. У результаті код, який миттєво виконується локально, на продакшні поводиться зовсім інакше.

Критична проблема — високе навантаження на базу даних

Найчастіше “вузьким місцем” у телекомунікаційних архітектурах стає база даних. При зростанні навантаження перше, що спадає на думку — поставити потужніше “залізо”. Але це рішення лише тимчасове: рано чи пізно апаратні ресурси знову вичерпаються.

Інший підхід — тюнінг бази даних. Програмісти часто звертаються до адміністраторів із проханням оптимізувати параметри кешу, пул з’єднань, буферів тощо. Це дійсно може покращити продуктивність, але не вирішує проблему концептуально. Часто причиною перевантаження є не конфігурація СУБД, а неефективні запити або відсутність індексів.

Щоб виявити джерело проблеми, необхідно:

- зібрати статистику довгих запитів (slow queries),
- проаналізувати, коли саме навантаження зростає (наприклад, з 3:00 до 4:00 через імпорт даних),
- розглянути можливість кластеризації або шардінгу таблиць.

Наприклад, якщо таблиця зберігає статистику за рік, але 95% запитів стосується лише останнього тижня — доцільно розділити її на 12 окремих таблиць за місяцями. Це скоротить час вибірки майже в 10–12 разів.

Проблеми масштабних телекомунікаційних систем

На великих проєктах головні труднощі зазвичай не технічні, а організаційні. Інженери прагнуть використовувати найновіші технології — Docker, Consul, Chef, RabbitMQ, OpenStack — навіть тоді, коли це не обґрунтовано потребами проєкту. Такі експерименти додають складності й ризикують стати джерелом нестабільності.

Приклад: створення кластера RabbitMQ “для відмовостійкості” часто закінчується тим, що падіння одного вузла призводить до розсинхронізації черг і втрати повідомлень. Або впровадження Chef робить централізоване оновлення зручним, але блокує можливість швидкого локального виправлення.

Такі підходи корисні лише тоді, коли компанія має достатньо ресурсів і досвідчену команду, здатну їх підтримувати. Якщо ж технологія впроваджується лише тому, що “всі так роблять” або “так радять на StackOverflow”, вона швидше за все створить більше проблем, ніж вирішить.

```sql
-- Виявлення 10 найповільніших запитів у MySQL
SELECT
    db AS 'База даних',
    COUNT(*) AS 'Кількість виконань',
    ROUND(SUM(TIME_MS)/1000, 2) AS 'Сумарний час (сек)',
    DIGEST_TEXT AS 'SQL-запит'
FROM performance_schema.events_statements_summary_by_digest
ORDER BY SUM(TIME_MS) DESC
LIMIT 10;
```

Рис. 3.3 Приклад коду для збору статистики повільних запитів MySQL

Такий запит допомагає швидко визначити, які SQL-операції створюють найбільше навантаження на базу даних і в який час це відбувається.

4 ВИКОРИСТАННЯ ГІБРИДНОЇ АРХІТЕКТУРИ

Підрозділам, що відповідають за телекомунікаційні та інформаційні системи підприємств, доводиться приймати стратегічно важливе рішення — яку саме архітектуру мережі обрати під час створення або модернізації системи управління телекомунікаційною інфраструктурою. Від цього вибору безпосередньо залежить стабільність, масштабованість і швидкість реакції всієї мережі.

Серед найпоширеніших типів архітектур виділяють три основні моделі:

1. Клієнт-серверна архітектура, де всі користувачі (клієнти) підключаються до центрального сервера, який виконує обробку запитів, зберігання та розподіл інформації.
2. Архітектура типу “хаб” (централізований вузол), у якій один головний центр комунікації координує роботу периферійних елементів мережі.
3. Пірингова (однорангова) архітектура, де всі вузли рівноправні, мають власні ресурси та можуть безпосередньо взаємодіяти між собою без єдиного контролюючого центру.

У сфері телекомунікацій ідеальної архітектури не існує — кожен підхід має свої переваги й обмеження, залежно від масштабу мережі, вимог до надійності, рівня безпеки та швидкості обміну даними. Для великих операторів із розгалуженою інфраструктурою найчастіше використовують гібридні рішення, які поєднують централізоване управління та елементи децентралізації, що підвищує гнучкість і стійкість до збоїв.

Для інженерів телекомунікацій найважливішим завданням є підтримка актуальності даних у режимі реального часу — від стану каналів зв’язку до завантаженості комутаційних вузлів. Вони повинні оперативно реагувати на запити систем моніторингу та звернення користувачів, забезпечуючи виконання SLA (Service Level Agreement) — угод про рівень якості послуг.

З іншого боку, керівники ІТ-та ТК-департаментів мають бачити повну й достовірну аналітичну картину мережевої інфраструктури: які сегменти перевантажені, де потрібна модернізація, які ресурси варто оптимізувати. Від якості

даних, які збирає система, залежить ефективність управлінських рішень — від розподілу бюджету на мережеве обладнання, ліцензії та серверні ресурси до планування розвитку корпоративної телекомунікаційної екосистеми.

Таким чином, вибір архітектури телекомунікаційної системи — це не лише питання технічної реалізації, а й стратегічна складова управління інфраструктурою. Оптимальне рішення полягає у поєднанні централізованого контролю, високого рівня автоматизації, гнучкого масштабування та швидкої реакції на події в мережі — саме така архітектура забезпечує стійкість, прозорість і ефективність сучасних ТК-систем.

4.1 Клієнт-серверна архітектура

Загальні принципи клієнт-серверної архітектури

Клієнт-серверна архітектура є однією з найпоширеніших структур у сучасних телекомунікаційних системах і корпоративних мережах. Вона використовується у всіх сферах, де існує потреба в централізованому зберіганні, обробці й наданні даних численним користувачам або вузлам мережі.

Сутність архітектури полягає у розподілі функціональності між двома основними компонентами — клієнтом і сервером. Клієнт виступає ініціатором запиту, а сервер — постачальником послуг або ресурсів. Клієнти можуть бути представлені у вигляді персональних комп’ютерів, смартфонів, терміналів, IoT-пристроїв, прикладних програм чи вебінтерфейсів. Сервер же зазвичай є більш потужним вузлом, який забезпечує обчислювальні ресурси, виконує логіку обробки запитів, керує базами даних і гарантує безперервний доступ до інформації.

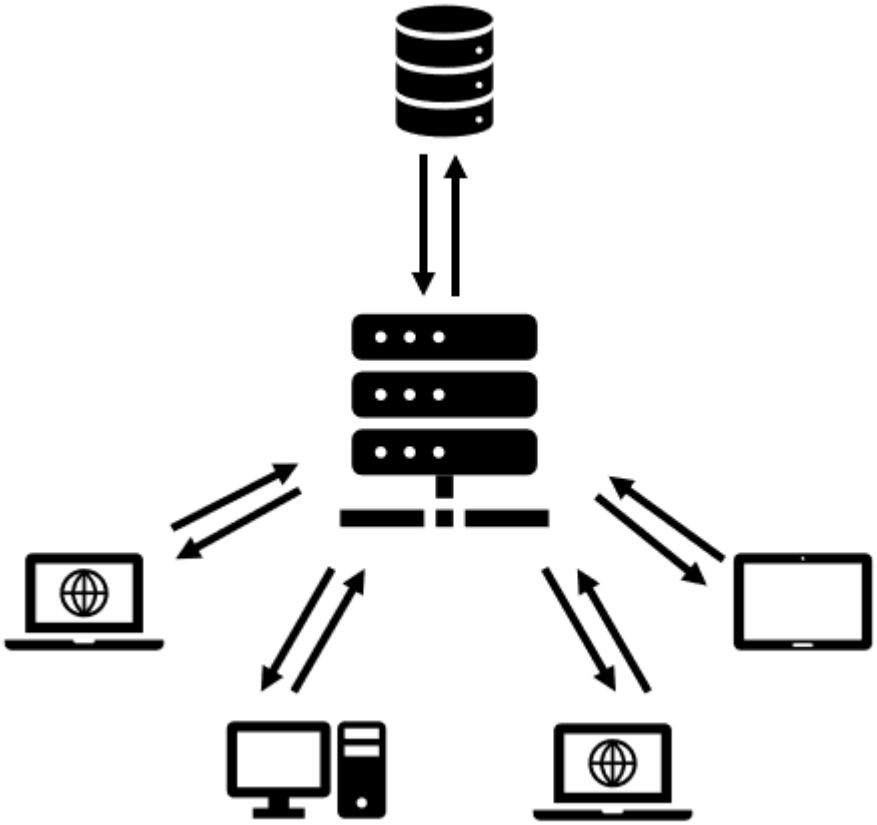


Рис. 4.1 Клієнт-серверна архітектура

В контексті телекомунікацій, така модель дозволяє централізовано керувати потоком даних, обробляти запити користувачів і забезпечувати високу узгодженість інформації у масштабі великої мережі. Наприклад, у системах моніторингу мережевого трафіку сервер збирає телеметрію з усіх маршрутизаторів, комутаторів і вузлів, а клієнтські термінали відображають ці дані у вигляді графіків, карт і звітів.

Логічна структура та принцип взаємодії компонентів

Архітектура клієнт-серверної системи зазвичай має **три рівні**:

1. **Рівень клієнта** — користувацький інтерфейс, через який здійснюється взаємодія з системою.
2. **Рівень застосунків (middle-tier)** — проміжна логіка, що відповідає за обробку запитів, перевірку даних, автентифікацію, кешування та координацію між клієнтами й серверами баз даних.
3. **Рівень даних** — фізичне зберігання, обробка та резервування інформації (бази даних, файлові сховища, резервні репозиторії).

Типовий процес взаємодії виглядає так:

1. Клієнтська частина формує запит (наприклад, отримати список абонентів мережі).
2. Сервер приймає запит, виконує обробку, отримує необхідні дані з бази.
3. Оброблений результат повертається клієнтові у зручній для відображення формі.

Такий підхід забезпечує чітке розмежування обов'язків, що спрощує масштабування, адміністрування й оновлення системи. Наприклад, заміна або оновлення клієнтського інтерфейсу не впливає на роботу серверної частини — і навпаки.

Переваги клієнт-серверної архітектури в телекомунікаційних системах

Клієнт-серверна модель є основою для більшості мережних інформаційних платформ — від простих систем моніторингу до складних корпоративних мережових центрів керування (NOC). Її головні переваги полягають у наступному:

1. **Централізоване управління даними.**

   Усі дані зберігаються в єдиній базі, що полегшує контроль, аудит і захист інформації. Це критично важливо для телекомунікаційних компаній, які обслуговують тисячі клієнтів і потребують точності при роботі з технічними журналами, лічильниками трафіку або записами викликів.

2. **Обробка інформації в реальному часі.**

   Сервери телекомунікаційної мережі можуть обробляти мільйони пакетів даних на секунду, забезпечуючи аналітику, маршрутизацію та динамічне балансування навантаження в режимі онлайн.

3. **Простота централізованої адміністрації.**

   Завдяки централізованій структурі адміністратори мають змогу керувати користувачами, оновленнями ПЗ, політиками доступу, а також виконувати моніторинг і діагностику системи з єдиного пункту.

4. **Високий рівень контролю безпеки.**

   Центральний сервер дозволяє впроваджувати єдину політику авторизації, шифрування та управління сертифікатами, що особливо важливо у

корпоративних телекомунікаційних мережах і дата-центрах операторів зв’язку.

5. **Гнучкість і масштабованість.**

   Додавання нових клієнтів або розширення функціоналу відбувається без повного перезапуску системи. Це дозволяє масштабувати інфраструктуру відповідно до потреб зростання навантаження.

Основні обмеження та проблеми

Попри численні переваги, клієнт-серверна архітектура має й низку суттєвих недоліків, особливо помітних у великих телекомунікаційних системах:

1. **Обмеження по кількості одночасних підключень.**

   Кожен сервер має фізичну межу пропускної здатності — і при досягненні певного навантаження він перестає обслуговувати нові запити. У практиці великих провайдерів це може бути близько 14 тисяч активних з’єднань для звичайного каналу або до 65 тисяч при використанні оптимізованих протоколів.

2. **Можливість виникнення “мережевого шторму”.**

   Якщо тисячі клієнтів одночасно надсилають запити до однієї служби — сервер може бути перевантажений, а мережеві комутатори почнуть втрачати пакети. Це призводить до **циклічних збоїв**, коли система намагається відновитись, але лише посилює навантаження.

3. **Висока вартість утримання інфраструктури.**

   Для забезпечення стабільної роботи потрібно купувати нові сервери, розширювати сховища, інвестувати у систему охолодження та енергозабезпечення. Телекомунікаційні оператори витрачають значні кошти на модернізацію дата-центрів, балансувальників і систем резервування.

4. **Залежність від центрального вузла.**

   У разі виходу з ладу головного сервера — уся мережа може зупинитися. Навіть короткочасний збій може призвести до втрати даних або відключення користувачів, що в сфері зв’язку є неприпустимим.

5. **Складність підтримки.**

   Обслуговування серверних систем вимагає висококваліфікованих фахівців — адміністраторів, аналітиків безпеки, розробників програмного забезпечення, що збільшує витрати компанії.

Сценарії застосування в телекомунікаційних мережах

Клієнт-серверна архітектура найчастіше використовується в таких випадках:

- **Системи керування мережею (Network Management Systems, NMS).**

  Центральний сервер збирає інформацію про стан каналів, вузлів і пристроїв, а клієнти (оператори) здійснюють моніторинг і керування з робочих місць.

- **Системи білінгу та обліку трафіку.**

  Сервер обробляє запити з тисяч абонентських станцій, веде логування, розрахунок тарифів, виставлення рахунків.

- **Хмарні сервіси для корпоративних користувачів.**

  Користувачі з будь-якої точки світу можуть підключатися до центральних серверів для доступу до пошти, документів, VPN-з’єднань або віртуальних телефонних номерів.

- **Роздрібні мережі (ритейл).**

  Працівники компанії використовують ноутбуки чи планшети для віддаленої роботи — під час зустрічей із клієнтами, у торгових точках або під час виїздів. Усі дані при цьому передаються через сервер компанії, що гарантує актуальність інформації.

Оптимізація клієнт-серверних систем

Щоб уникнути перевантажень і підвищити стійкість системи, телекомунікаційні компанії впроваджують ряд рішень:

- **Розподіл навантаження (Load Balancing).**

  Запити клієнтів розподіляються між кількома серверами, що знижує ризик відмови одного вузла.

- **Кешування даних.**

  Використання проміжних серверів кешу дозволяє зменшити кількість запитів до основної бази, прискорюючи обробку даних.

- **Резервування (Redundancy).**

  Наявність кількох серверів із синхронізованими даними забезпечує безперервність роботи навіть у випадку аварії.

- **Віртуалізація та контейнеризація.**

  Завдяки таким технологіям, як Docker або VMware, можна швидко відновити сервери після збоїв, масштабувати потужності без простою системи.

Перспективи розвитку клієнт-серверної архітектури в телекомунікаціях

Сучасні тенденції свідчать, що класична клієнт-серверна модель еволюціонує у бік хмарних і гібридних архітектур. Розвиток 5G-мереж, SDN (Software Defined Networking) і MEC (Multi-access Edge Computing) призводить до того, що обчислювальні ресурси частково переміщуються до периферії мережі (edge), зменшуючи затримки і покращуючи якість обслуговування.

Проте принцип клієнт-серверної взаємодії залишається базовим — навіть у таких нових концепціях, як віртуалізовані мережеві функції (VNF) чи архітектури з розподіленим обчисленням. Сервери стають "розподіленими", а клієнти — більш автономними, але взаємодія між ними все одно відбувається за тією ж логікою: запит → обробка → відповідь.

Клієнт-серверна архітектура залишається основою сучасних телекомунікаційних систем, забезпечуючи баланс між централізованим управлінням, контролем безпеки та зручністю масштабування. Вона ефективно працює як у корпоративних мережах, так і в глобальних інфраструктурах операторів зв’язку.

Разом із тим, ефективність її функціонування залежить від правильного проєктування, оптимізації каналів зв’язку, впровадження систем резервування та моніторингу. При грамотному підході клієнт-серверна архітектура дозволяє побудувати стійку, гнучку і керовану телекомунікаційну мережу, що відповідає сучасним вимогам швидкості, безпеки та надійності.

4.2 Хабова архітектура телекомунікаційних систем

Загальна концепція та сутність хабової архітектури

Хабова архітектура є одним із базових підходів до побудови телекомунікаційних та інформаційних систем, орієнтованих на централізоване керування потоками даних і контроль взаємодії між усіма мережевими компонентами.

На відміну від клієнт-серверної моделі, де сервер безпосередньо обслуговує клієнтів, хабова архітектура використовує центральний вузол — хаб (hub), який виступає посередником між усіма підключеними пристроями. Усі запити, обміни даними чи сигнали проходять саме через цей хаб, що дозволяє повністю контролювати трафік, структуру доступу та безпеку інформаційних потоків.

У телекомунікаційному контексті хаб виконує роль “серця мережі”, де сходяться усі канали зв’язку. Через нього здійснюється маршрутизація, фільтрація, пріоритизація даних, а також обробка службової інформації. У масштабних корпоративних або державних структурах така архітектура забезпечує максимальну керованість і надійність, адже всі дії користувачів відстежуються, а передача даних відбувається лише через контрольований центр.

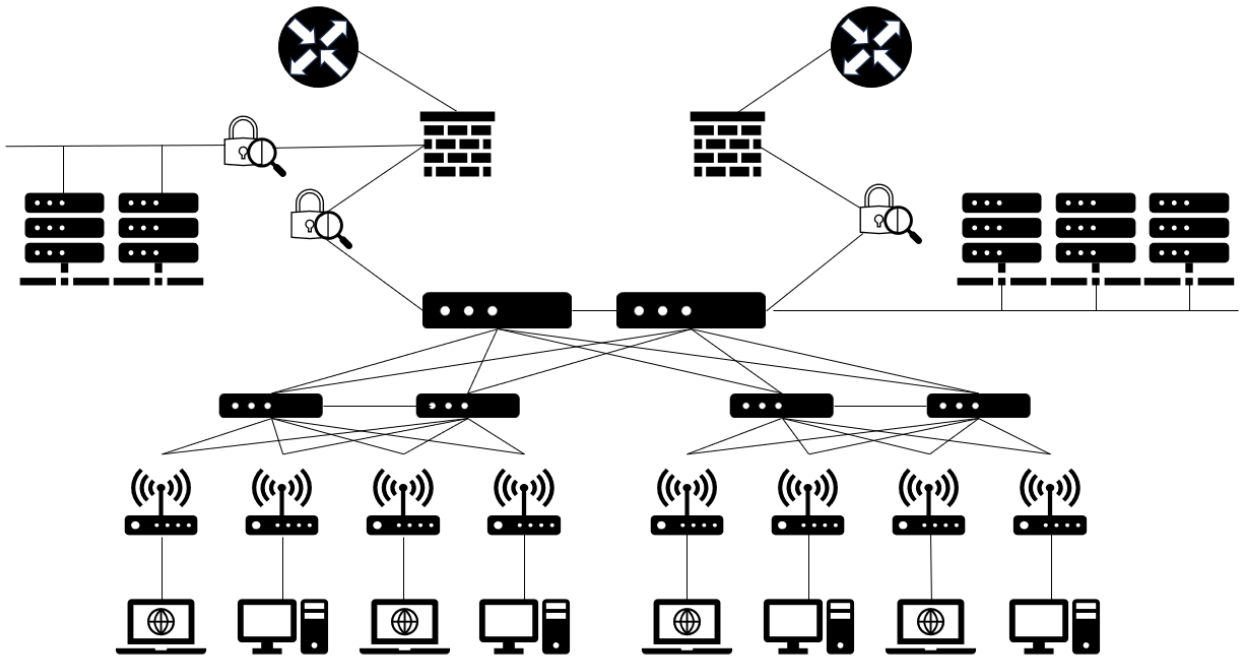


Рис. 4.2 Хабова архітектура

Принципи функціонування хабової архітектури

Основна ідея полягає у тому, що всі кінцеві точки — робочі станції, термінали, комутатори, маршрутизатори чи сенсори — підключені до єдиного хаба, який координує обмін інформацією.

Такий підхід дозволяє побудувати ієрархічну структуру, де кожен локальний хаб обслуговує певну кількість пристроїв, а вищий рівень — координує обмін між локальними хабами.

Типова архітектура має вигляд «зірки»:

- у центрі — головний хаб або головний сервер керування;
- навколо нього — підлеглі вузли (локальні хаби), що об’єднують групи пристроїв;
- ще нижче — окремі кінцеві точки мережі (комп’ютери, модеми, датчики, контролери тощо).

Таким чином, дані передаються від нижчого рівня до вищого, послідовно, у певному часовому порядку. Це дозволяє уникнути хаотичного обміну інформацією і запобігти одночасному перевантаженню каналів.

Однак, якщо декілька підмереж спробують одночасно передати великий обсяг даних на головний вузол — це може спричинити ефект DDoS-навантаження або навіть тимчасову недоступність системи. Тому планування каналів зв’язку і черговість обміну є критично важливими факторами.

Централізоване управління як ключова риса хабової архітектури

Головною перевагою хабової моделі є повна централізація управління. Центральний хаб не лише контролює обмін даними, а й:

- виконує аутентифікацію користувачів і пристроїв;
- аналізує журнали доступу та виявляє потенційні загрози;
- управляє політиками безпеки (наприклад, обмежує доступ до певних ресурсів);
- здійснює резервне копіювання та відновлення даних;
- регламентує час та черговість передачі пакетів.

Завдяки цьому телекомунікаційна система стає прогнозованою, безпечною і керованою.

Центр управління (Network Operation Center — NOC) може в реальному часі бачити стан усіх елементів інфраструктури, контролювати навантаження на канали, а також локалізувати проблеми без потреби втручання у кожен окремий вузол.

Переваги хабової архітектури

1. **Підтримка великої кількості кінцевих точок.**

   Завдяки ієрархічній структурі мережу можна розширювати поступово, додаючи нові хаби без кардинальної зміни основної конфігурації. Кожен локальний хаб обслуговує свою підмережу, а головний контролює весь обмін.

2. **Високий рівень інформаційної безпеки.**

   Всі канали проходять через контрольовану точку, тому можна повністю відстежувати потоки даних. Це дозволяє виявляти витоки, несанкціоновані підключення, шкідливі активності або втручання.

3. **Можливість сегментування мережі.**

   Завдяки чіткому поділу на рівні, адміністратор може легко ізолювати частину системи (наприклад, при зараженні шкідливим ПЗ), не порушуючи роботу всієї мережі.

4. **Простота моніторингу та аналітики.**

   Центральний вузол збирає статистику в єдиному форматі, що спрощує аналіз трафіку, журналів подій, технічних інцидентів і дає можливість автоматизувати реагування.

5. **Високий ступінь контролю за доступом.**

   Усі користувачі проходять через централізовану систему автентифікації, що дозволяє впроваджувати політику “єдиного входу” (Single Sign-On) або багатофакторну авторизацію.

Недоліки та ризики використання

Хабова архітектура, попри свої переваги, має низку обмежень, які необхідно враховувати при проєктуванні телекомунікаційних систем.

1. **Відсутність обміну в режимі реального часу.**

Через те, що хаб передає дані послідовно, а не паралельно, виникає затримка при великому потоці запитів. Для телекомунікаційних систем, де критично важлива мінімальна затримка (наприклад, у VoIP або системах управління рухом даних), це може стати проблемою.

2. **Вразливість центрального вузла.**
   Втрата зв'язку з головним хабом або його вихід із ладу може паралізувати роботу всієї мережі. Тому необхідно передбачати дублюючі елементи або резервні хаби.
3. **Високі вимоги до пропускної здатності центральних каналів.**
   Усі дані проходять через центральну точку, тому канали зв'язку до неї повинні мати високу швидкість, надійність і мінімальні затримки.
4. **Складність модернізації в централізованих структурах.**
   Будь-які зміни або оновлення в хабі вимагають ретельного планування, адже помилка може вплинути на всю систему.

Приклади практичного застосування

Хабова архітектура особливо ефективна у тих організаціях, де інформаційна безпека є пріоритетом.

Наприклад:

- **Банківські установи**, де всі транзакції проходять через центральний хаб контролю безпеки, що аналізує операції в реальному часі.
- **Державні структури** з обмеженим доступом до внутрішніх мереж, де всі співробітники працюють у закритому середовищі й мають доступ лише до певних ресурсів.
- **Промислові підприємства**, які застосовують SCADA-системи для управління обладнанням, де центральний хаб контролює датчики, контролери та автоматизовані лінії.

Також хабова архітектура активно використовується у корпоративних дата-центрах і віртуалізованих середовищах, де необхідно мати централізований контроль над усіма віртуальними машинами та контейнерами.

Хабова архітектура в сучасних телекомунікаційних мережах

У сучасних мережах зв’язку хабова архітектура набуває нового сенсу в умовах розвитку концепцій SDN (Software-Defined Networking) та NFV (Network Function Virtualization).

У таких системах “хаб” уже не є фізичним пристроєм, а виступає віртуальним центром керування, який координує взаємодію між мережевими елементами через програмно визначені правила.

Таким чином, хаб перетворюється на “інтелектуальний контролер”, який не лише передає пакети, а й приймає рішення про маршрутизацію, балансування навантаження, застосування політик QoS і шифрування. Такі рішення активно впроваджуються у великих телекомунікаційних операторів і хмарних платформах, наприклад у системах управління віртуальними базовими станціями 5G (vRAN) або у міжмережевих шлюзах хмарної інфраструктури.

Перспективи розвитку

Майбутнє хабової архітектури тісно пов’язане з еволюцією гібридних мереж, де централізований контроль поєднується з елементами децентралізації. Сучасні тенденції вказують на формування “розумних хабів” (Smart Hubs) — пристроїв або програмних модулів, які можуть самостійно аналізувати трафік, виявляти аномалії, оптимізувати маршрутизацію та автоматично відновлювати з’єднання після збоїв.

Особливу роль відіграють телекомунікаційні хаби нового покоління, які використовують машинне навчання для прогнозування навантаження та оптимізації роботи каналів зв’язку. Такі системи стають ключовим елементом побудови інтелектуальних мереж (Smart Networks) і інфраструктури 6G, де кожен вузол мережі — від базової станції до термінала користувача — взаємодіє через динамічно керовані хаби.

Хабова архітектура залишається одним із найбільш стабільних і контрольованих рішень у сфері телекомунікаційних систем. Вона дозволяє забезпечити централізоване керування, глибокий контроль безпеки та високу надійність при великій кількості вузлів і користувачів. Разом із тим, її ефективність залежить від правильного проєктування, розрахунку

навантажень, резервування каналів і впровадження сучасних технологій віртуалізації та аналітики.

У майбутньому хабові архітектури стануть ще гнучкішими, отримають функції самонавчання, автоматичного балансування та адаптації до динамічних змін у мережі. Таким чином, вони залишатимуться невід'ємною частиною інфраструктури сучасних телекомунікацій, де стабільність, безпека й централізований контроль є ключовими факторами ефективності.

4.3 Пірингова архітектура телекомунікаційних систем

Суть та значення пірингової архітектури

Пірингова або однорангова архітектура (від англ. *Peer-to-Peer, P2P*) — це сучасний тип побудови мережевих систем, який ґрунтується на децентралізованому принципі взаємодії між вузлами. На відміну від клієнт-серверної чи хабової моделі, де є чіткий центр керування або вузол, через який проходить весь трафік, пірингова архітектура передбачає повну рівноправність усіх учасників мережі.

У телекомунікаційній сфері така архітектура означає, що кожен вузол (peer) одночасно виконує функції і клієнта, і сервера — він може як запитувати, так і надавати дані іншим вузлам без посередників. Завдяки цьому мережа стає розподіленою, стійкою до збоїв і надзвичайно гнучкою.

У сучасних умовах розвитку глобальних мереж і цифрових сервісів пірингова архітектура дедалі частіше використовується в телекомунікаційних технологіях, оскільки забезпечує реальний обмін даними у режимі реального часу та дозволяє підтримувати величезну кількість користувачів без централізованої інфраструктури.

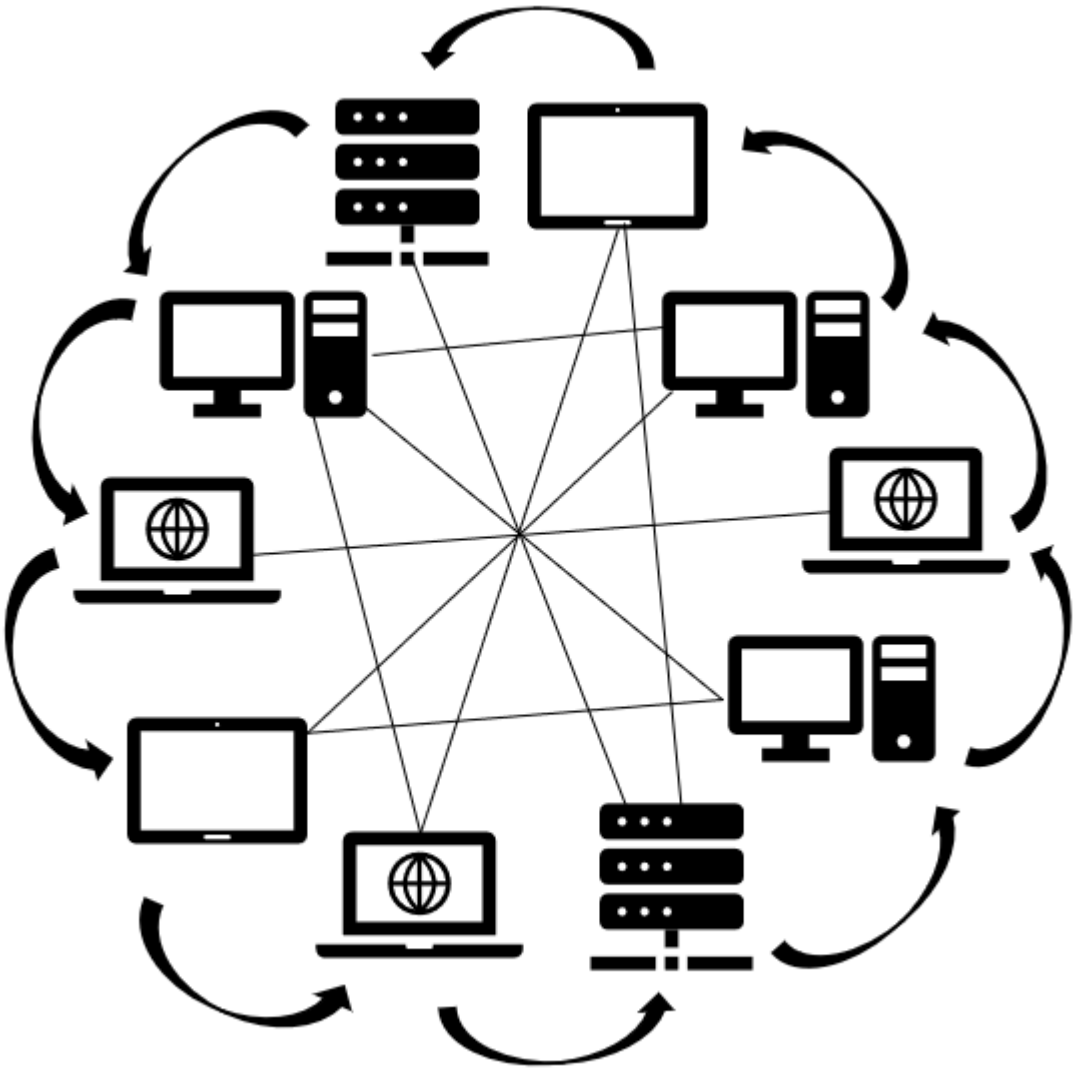


Рис. 4.3 Іллюстрація Пірингової архітектури

Принцип децентралізації та автономності вузлів

Ключова ідея піринговой архітектури — повна автономність кожного елемента системи.

Кожен вузол має власні ресурси — процесорну потужність, пам’ять, мережевий канал — і може використовувати їх для зберігання, обробки або передачі інформації. Всі вузли рівноправні, тобто не існує “головного” сервера, який контролює або керує іншими.

Це означає, що:

- відмова одного вузла не впливає на стабільність мережі;
- дані зберігаються розподілено на багатьох пристроях;

- алгоритми маршрутизації автоматично перебудовують шляхи передавання трафіку.

Одним із найефективніших підходів, який використовується у пірингових системах, є MESH-топологія. У такій структурі кожен вузол має з'єднання з кількома сусідами, і якщо один шлях недоступний, мережа автоматично вибирає інший маршрут.

Це гарантує високу надійність і самовідновлення навіть у разі масштабних відмов або перебоїв у роботі окремих елементів.

Переваги пірингових мереж у телекомунікаціях

Пірингова архітектура має ряд переваг, завдяки яким вона поступово стає основою для багатьох сучасних рішень у сфері зв'язку, обміну даними та розподілених обчислень:

1. **Передача даних у реальному часі.**

   У пірингових мережах інформація передається без посередників, що мінімізує затримки (latency). Це особливо важливо для мультимедійних сервісів, відеоконференцій, IP-телефонії чи інтерактивних ігор.

2. **Стабільність і відмовостійкість.**

   Завдяки децентралізованій природі мережі навіть вихід з ладу кількох вузлів не впливає на загальну працездатність системи. Інші вузли автоматично беруть на себе функції відсутніх.

3. **Простота масштабування.**

   Додавання нових пристроїв не потребує зміни структури або налаштування центральних серверів. Кожен новий вузол "самостійно" приєднується до мережі, отримуючи необхідну інформацію від сусідів.

4. **Висока ефективність використання ресурсів.**

   Кожен учасник мережі не лише споживає, а й надає ресурси. Це дозволяє зменшити навантаження на інфраструктуру та підвищити загальну продуктивність системи.

5. **Низька вартість розгортання.**

Відсутність потреби у центральних серверах значно знижує витрати на апаратне забезпечення, оренду дата-центрів і утримання адміністраторів. Кожен вузол сам виконує роль частини інфраструктури.

6. **Гнучкість та адаптивність.**

   Мережа легко адаптується до змін — як технічних (додавання чи видалення вузлів), так і логічних (зміна маршрутів, розподіл навантаження, пріоритети передачі даних).

Недоліки та обмеження пірингових систем

Попри численні переваги, пірингові мережі мають і певні обмеження, які варто враховувати при їх використанні у телекомунікаційних проектах:

1. **Обмеження у закритих або контрольованих мережах.**

   У корпоративних або військових структурах, де застосовуються суворі політики безпеки, пірингова взаємодія може бути заблокована. Такі мережі не дозволяють прямі з’єднання між вузлами через наявність фаєрволів, NAT або систем DPI.

2. **Відсутність централізованого контролю.**

   Хоча децентралізація є перевагою, вона ускладнює моніторинг і управління. У випадку кіберінцидентів або помилкової конфігурації знайти джерело проблеми може бути складно.

3. **Безпекові ризики.**

   Через відкриту природу пірингової архітектури, вузли можуть піддаватися атакам або спробам поширення шкідливого ПЗ. Необхідно впроваджувати криптографічний захист, автентифікацію та шифрування каналів.

4. **Нестабільність швидкості передачі даних.**

   Пропускна здатність мережі залежить від характеристик окремих вузлів, їхньої завантаженості та якості з’єднання.

Типові сценарії застосування пірингової архітектури

Пірингові мережі застосовуються у найрізноманітніших сферах, зокрема там, де потрібен розподілений обмін даними без участі центрального вузла:

- **Оновлення програмного забезпечення.**

Компанії можуть розповсюджувати оновлення одночасно серед тисяч пристроїв без навантаження на сервери. Кожен пристрій, що вже отримав оновлення, передає його далі іншим — ефект “ланцюгової передачі”.

- **Системи розподілених обчислень.**
  Наприклад, мережі BOINC чи Folding@home, де обчислювальні задачі діляться між тисячами комп’ютерів у всьому світі. Це дозволяє використовувати ресурси користувачів як єдиний суперкомп’ютер.
- **Сервіси контент-дистрибуції (CDN, P2P streaming).**
  Відео або аудіо передається не лише з головного сервера, а й через інші вузли користувачів, що підвищує швидкість і зменшує навантаження на основну мережу.
- **Телекомунікаційні системи для Інтернету речей (IoT).**
  У розподілених IoT-мережах пристрої можуть обмінюватися даними безпосередньо один з одним, що мінімізує затримки та знижує вимоги до центрального контролера.
- **Блокчейн і криптографічні мережі.**
  Це найвідоміший приклад пірингової архітектури, де всі вузли (ноди) рівноправні, а дані зберігаються одночасно на тисячах комп’ютерів по всьому світу. Такий підхід гарантує повну прозорість і стійкість до втручань.

Економічна ефективність і масштабованість

Однією з ключових переваг пірингової архітектури є її економічна доцільність.

Для запуску мережі не потрібно купувати дороге серверне обладнання або підтримувати інфраструктуру дата-центрів. Кожен користувач фактично надає частину своїх ресурсів у спільне користування, створюючи розподілену інфраструктуру.

Піринг також забезпечує високу масштабованість — мережа може зрости з кількох десятків вузлів до мільйонів без зміни архітектури чи потреби у додаткових серверах. Це робить модель ідеальною для інноваційних телекомунікаційних

рішень, таких як розумні міста (Smart City), розподілені сенсорні мережі або додатки 6G-комунікацій.

Пірингові мережі та сучасні алгоритми маршрутизації

Важливим аспектом пірингової архітектури є динамічна маршрутизація даних. У сучасних системах використовуються алгоритми типу *MESH Routing* або *DHT (Distributed Hash Table)*, які дозволяють автоматично знаходити найкоротший шлях для передачі даних між двома вузлами без централізованого диспетчера.

Наприклад, у DHT-мережі кожен вузол знає лише невелику частину інших вузлів, але може знайти будь-який ресурс у системі за лічені переходи. Це робить мережу швидкою, ефективною та масштабованою без лімітів.

Перспективи розвитку пірингових систем у телекомунікаціях

Пірингова архітектура поступово стає основою для нових поколінь мереж — від 5G до 6G, де пріоритетом є розподіленість, автономність і мінімальні затримки. В епоху цифровізації та вибухового зростання IoT пристроїв така модель дозволяє створювати самоорганізовані екосистеми, де пристрої самостійно визначають маршрути, синхронізуються і забезпечують стійкість системи.

Також піринг стає базою для розподілених центрів обробки даних (Edge Computing), де обчислення виконуються ближче до джерела інформації — на рівні користувача або локального пристрою. Це мінімізує затримки й підвищує ефективність системи.

Пірингова архітектура — це найбільш гнучка, масштабована та стійка до відмов форма побудови телекомунікаційних систем. Вона відкриває нові можливості для розвитку розподілених обчислень, автоматизації, IoT, контент-дистрибуції та блокчейн-технологій.

На відміну від клієнт-серверних або хабових структур, пірингові мережі не залежать від центрального вузла, що робить їх більш надійними, економічними та придатними до роботи в глобальному середовищі.

У перспективі саме пірингові системи стануть фундаментом нової ери телекомунікацій, де ключовими принципами будуть децентралізація, автономність,

адаптивність і повна взаємодія між мільйонами пристроїв без потреби у централізованому контролі.

4.4 Конвергентна архітектура телекомунікаційних систем

Новий етап еволюції телекомунікаційних архітектур

У сучасних умовах цифрової трансформації, коли обсяги інформації стрімко зростають, а корпоративні мережі охоплюють десятки тисяч пристроїв, класичні архітектурні підходи — клієнт-серверний, хабовий чи піринговий — уже не здатні повністю забезпечити баланс між безпекою, гнучкістю та продуктивністю. На цьому тлі формується новий тип організації інформаційних систем — конвергентна (гібридна) архітектура, яка поєднує найкращі властивості своїх попередників і пропонує універсальний підхід до управління складними телекомунікаційними середовищами.

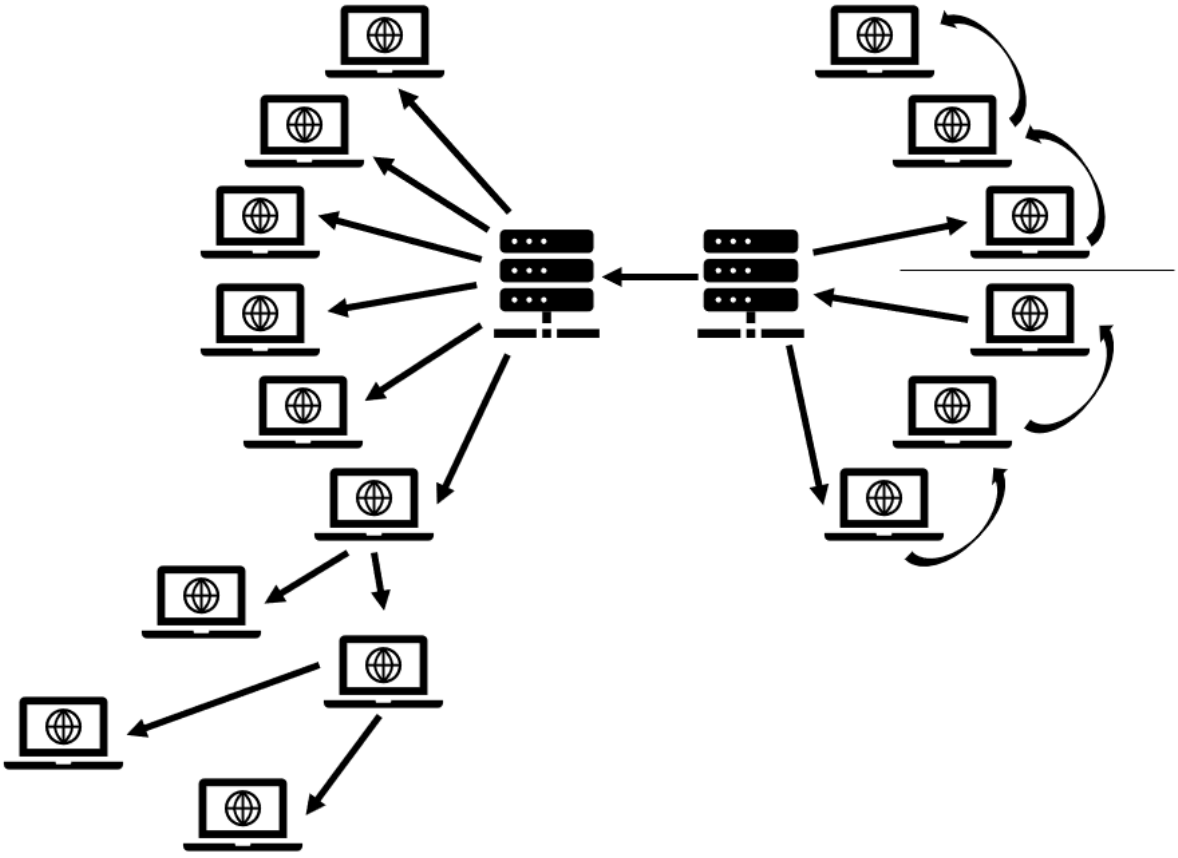


Рис. 4.4 Іллюстрація конвергентної архітектури

Термін «конвергентна» (від лат. *convergere* — “збігатися”) відображає головну ідею цієї моделі — інтеграцію різнорідних технологій і протоколів у єдину систему, здатну ефективно функціонувати навіть у багаторівневих, географічно розподілених і гібридних інфраструктурах.

Загальна характеристика та принцип дії конвергентної архітектури

Конвергентна архітектура — це модель, у якій поєднано елементи централізованого та децентралізованого управління, що дозволяє одночасно збирати, аналізувати та синхронізувати дані з великої кількості пристроїв, розташованих у різних мережах, сегментах або навіть країнах.

Така система здатна обробляти десятки тисяч кінцевих точок — як стаціонарних (робочі станції, сервери, мережеве обладнання), так і мобільних (смартфони, планшети, ноутбуки, IoT-пристрої). Передача даних відбувається періодично, з високою частотою, наприклад — щосекунди або за визначеним інтервалом. Це забезпечує актуальність інформації в режимі, максимально наближеному до реального часу.

Завдяки своїй структурі конвергентна архітектура не потребує єдиного центрального сервера у класичному розумінні. Вона використовує мережу проміжних вузлів і контролерів, які локально збирають дані, попередньо обробляють їх і лише потім передають на головні сервери управління. Такий підхід знижує навантаження на мережу, зменшує затримки, підвищує надійність і захищеність системи.

Особливості архітектурної моделі

1. **Багаторівнева структура.**

   Конвергентна система складається з локальних і регіональних вузлів, які виконують роль посередників між кінцевими пристроями та головними центрами обробки даних. Це дозволяє розділити трафік і мінімізувати затримки навіть у глобальних мережах.

2. **Автономність сегментів.**

   Кожен сегмент може функціонувати незалежно від інших. Навіть якщо один із вузлів або каналів зв’язку буде недоступний, система не втратить

працездатність — дані накопичуються локально й передаються після відновлення зв’язку.

3. **Інтелектуальна маршрутизація даних.**

   Алгоритми системи визначають найоптимальніші маршрути для передавання даних, враховуючи навантаження каналів, доступність вузлів і пріоритетність інформації.

4. **Інтеграція з хмарними технологіями.**

   Гібридні рішення дозволяють частину даних зберігати у приватних або публічних хмарах, тоді як критично важлива інформація залишається у захищених корпоративних сегментах.

Переваги конвергентної архітектури для телекомунікацій

1. **Високий рівень інформаційної безпеки.**

   Конвергентна архітектура забезпечує сегментування мережі, тому злам одного вузла не призводить до компрометації всієї системи. Вразливість не поширюється на інші частини, а локальні модулі мають власні механізми автентифікації та шифрування.

2. **Передача даних у реальному часі.**

   Періодична або потокова синхронізація даних між вузлами забезпечує постійну актуальність інформації для адміністраторів, систем моніторингу та безпеки.

3. **Стабільність і безперервність роботи.**

   Завдяки наявності кількох рівнів обробки система не зупиняється навіть у разі відмови частини інфраструктури. Кожен модуль продовжує функціонувати автономно, що підвищує загальну стійкість телекомунікаційної мережі.

4. **Необмежена кількість підтримуваних пристроїв.**

   На відміну від класичних архітектур, які мають фізичні ліміти підключень, конвергентна система дозволяє масштабувати мережу до сотень тисяч пристроїв без втрати продуктивності.

5. **Гнучке масштабування.**

   Розширення мережі виконується поступово — додавання нових вузлів або регіональних центрів не потребує зміни всієї структури чи зупинки роботи системи.

6. **Економічна ефективність.**

   Вартість розгортання нижча, ніж у традиційних централізованих систем, адже частину обробки виконують периферійні вузли, що зменшує потребу в потужних серверах.

Приклади застосування в телекомунікаціях

Конвергентна архітектура знайшла своє місце у широкому спектрі галузей, де потрібне одночасне управління великою кількістю пристроїв і високий рівень безпеки, зокрема:

- **Оператори зв’язку та провайдери інтернет-послуг.**

  Система дозволяє контролювати стан обладнання, збирати телеметрію з базових станцій, маршрутизаторів, модемів, ідентифікувати відмови у реальному часі.

- **Корпоративні мережі великих компаній.**

  Центральні офіси отримують узагальнені звіти з усіх філій, тоді як локальні підрозділи мають автономний доступ до власних даних без ризику для всієї системи.

- **Системи кібербезпеки.**

  Конвергентна модель дозволяє інтегрувати різні платформи моніторингу загроз, засоби IDS/IPS, антивірусні механізми та системи аналізу трафіку в єдиний центр управління без утворення “вузьких місць”.

- **Інтернет речей (IoT) та промислові телекомунікації.**

  Гібридна архітектура дозволяє об’єднати тисячі сенсорів і контролерів, що працюють у різних сегментах, у єдину екосистему з централізованим моніторингом і розподіленою обробкою даних.

Інформаційна безпека та надійність

Однією з головних причин, чому організації переходять до конвергентних рішень, є посилення вимог до безпеки даних.

На відміну від клієнт-серверних систем, де атака на центральний сервер може паралізувати всю мережу, конвергентна архітектура ізолює загрозу в межах конкретного сегмента. Навіть у випадку зламу один вузол не може поширити шкідливий код або вірус далі, ніж дозволяє його локальний рівень доступу.

Крім того, у таких системах застосовуються:

- **розподілені системи автентифікації**;
- **локальне шифрування каналів зв'язку**;
- **автоматичне резервування критичних даних**;
- **моніторинг поведінкових аномалій**, що дозволяє виявляти відхилення від стандартних шаблонів обміну даними.

Завдяки цьому конвергентна архітектура поєднує безпеку централізованих систем і стійкість децентралізованих.

Складність налаштування та виклики впровадження

Попри очевидні переваги, впровадження гібридної архітектури пов'язане з низкою технічних складнощів:

- **Необхідність ретельного планування взаємодії сегментів.**
  Потрібно визначити, які вузли будуть автономними, які виконуватимуть роль посередників, як синхронізуватимуться дані.
- **Недостатній досвід експлуатації.**
  Оскільки такі рішення є відносно новими, не всі компанії мають спеціалістів, здатних грамотно спроєктувати, розгорнути й підтримувати подібні системи.
- **Високі вимоги до безперервності зв'язку.**
  При розподіленому зборі даних навіть короткочасні збої каналів можуть впливати на точність інформації, тому потрібні механізми повторної доставки (retry mechanisms) і буферизації.

Конвергенція як стратегічний напрямок розвитку

Конвергентна архітектура сьогодні — це не просто технологічна тенденція, а логічний етап еволюції телекомунікаційних систем.

Вона об’єднує можливості різних підходів:

- **централізованого контролю** — для зручного адміністрування;
- **децентралізованої стійкості** — для забезпечення безперервності;
- **пірингової взаємодії** — для підвищення ефективності обміну даними.

Таким чином, конвергентна модель є універсальною платформою, придатною для роботи в умовах сучасних гібридних ландшафтів — коли частина систем розташована у локальній інфраструктурі, а інша — у хмарі або на периферії (Edge).

Конвергентна архітектура — це новий стандарт побудови телекомунікаційних систем, який дозволяє об’єднувати тисячі пристроїв, забезпечуючи при цьому стабільність, безпеку та гнучкість. Вона долає обмеження класичних моделей, оптимізує обмін даними й дає змогу організаціям ефективно функціонувати в умовах зростаючих обсягів інформації та складності IT-ландшафтів.

Завдяки своїй адаптивності, масштабованості та високому рівню кіберзахисту, конвергентна архітектура стане основою для побудови майбутніх інтелектуальних мереж, що поєднують локальні, хмарні та периферійні ресурси у єдину, гнучку телекомунікаційну екосистему.

ВИСНОВКИ

У межах дослідження на тему «Оптимізація архітектури телекомунікаційних мереж з урахуванням вимог користувачів» було проведено всебічний аналіз сучасних архітектурних рішень, підходів до їх оптимізації та впливу користувацьких вимог на ефективність функціонування телекомунікаційних систем. Мета роботи полягала у виявленні закономірностей і принципів, які дозволяють забезпечити баланс між продуктивністю, безпекою, масштабованістю та якістю обслуговування користувачів.

У ході дослідження розглянуто чотири базові архітектурні моделі: клієнт-серверну, хабову, пірингову та конвергентну (гібридну). Клієнт-серверна модель забезпечує швидку обробку даних і централізований контроль, проте має обмеження щодо кількості підключених пристроїв і чутливість до відмови центрального вузла. Хабова архітектура характеризується підвищеним рівнем безпеки та зручністю централізованого управління, але поступається у швидкості обміну даними. Пірингова модель є найбільш гнучкою — вона базується на децентралізації та забезпечує стабільність і відмовостійкість системи, хоча не може повноцінно функціонувати у середовищах із жорсткими мережевими обмеженнями. Найбільш збалансованим рішенням виявилася конвергентна архітектура, яка поєднує переваги різних моделей, дозволяючи об’єднувати тисячі пристроїв, працювати з географічно розподіленими сегментами та забезпечувати високий рівень інформаційної безпеки.

Результати дослідження довели, що оптимізація архітектури телекомунікаційних мереж має базуватися на принципі адаптивності, тобто здатності системи гнучко реагувати на зміни умов експлуатації, обсягів трафіку та вимог користувачів. Визначальними критеріями ефективності стають показники QoS і QoE, швидкість реакції системи, стійкість до збоїв та економічна доцільність обраного рішення. Оптимізація архітектури не обмежується апаратними або програмними змінами — вона передбачає перегляд логіки взаємодії між елементами мережі, використання балансування навантаження, автоматизованого

керування трафіком, а також впровадження інтелектуальних алгоритмів прогнозування стану мережі.

Проведений аналіз показав, що найвищої ефективності досягають гібридні архітектури, які поєднують централізоване управління з децентралізованими елементами, забезпечуючи при цьому гнучкість, стабільність і безпеку. Такі системи дають змогу підтримувати значну кількість кінцевих точок, зменшувати затримки передавання інформації, підвищувати рівень доступності сервісів і швидко адаптувати мережу до зростання навантажень.

Отже, оптимізація архітектури телекомунікаційних мереж — це складний, але необхідний процес, спрямований на створення надійних і масштабованих систем, здатних задовольнити потреби сучасних користувачів. Вона є основою для розвитку інфраструктур нового покоління — 5G, 6G, IoT-мереж і систем штучного інтелекту, що формуватимуть майбутнє глобальних комунікацій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Davidson J., Peters J., Gracely B. End-to-End QoS Network Design: Quality of Service for Rich-Media & Cloud Networks (2-ге вид.). – Cisco Press, 2013. – 608 с.
2. Wang Z. Internet QoS: Architectures and Mechanisms for Quality of Service. – Morgan Kaufmann, 2001. – 239 с.
3. Minei I., Lucek J. MPLS-Enabled Applications: Emerging Developments and New Technologies (3-тє вид.). – Wiley, 2011. – 627 с.
4. Osborne E., Simha A. Traffic Engineering with MPLS. – Cisco Press, 2002. – 608 с.
5. Möller S., Raake A. (ред.) Quality of Experience: Advanced Concepts, Applications and Methods. – Springer, 2014/2016. – 452 с.
6. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach (8-ме вид.). – Pearson, 2021. – 864 с.
7. Subramanian M. Network Management: Principles and Practice (2-ге вид.). – Pearson, 2010. – 848 с.
8. Stallings W. Foundations of Modern Networking: SDN, NFV, QoE, IoT and Cloud. – Addison-Wesley, 2016. – 600 с.
9. Fiedler M., Hossfeld T., Tran-Gia P. Quality of Experience: From Assessment to Application. – Wiley, 2010. – 320 с.
10. Keshav S. Mathematical Foundations of Computer Networking. – Addison-Wesley, 2012. – 496 с.
11. Голь В. Д. Телекомунікаційні та інформаційні мережі: навчальний посібник. – Київ: ІСЗЗІ КПІ ім. І. Сікорського, 2021. – 250 с.
12. Микитишин А. Г., Міщенко М. М. Телекомунікаційні системи та мережі: навчальний посібник. – Львів: Львівська політехніка, 2020. – 300 с.
13. Ordóñez-Lucena J., Ameigeiras P., Lopez D., Ramos-Muñoz J. J., Lorca J., Folgueira J. «Network Slicing for 5G with SDN/NFV: Concepts, Architectures and Challenges». – IEEE Communications Magazine, 2017, Т. 55, № 5. – С. 80–87.

14. Connelly K., et al. «Toward Total Quality of Experience: A QoE Model in a Communication Ecosystem». – IEEE Communications Magazine, 2012, Квітень. – С. 60–68.
15. Kilkki K. «Quality of Experience in Communications Ecosystem». – Journal of Universal Computer Science, 2008, Т. 14, № 5. – С. 615–624.
16. Panwar N., Sharma S., Singh A. K. «A Survey on 5G: The Next Generation of Mobile Communication». – Digital Communications and Networks, 2016, Т. 3, № 3. – С. 404–424.
17. Barakabitze A. A., et al. «QoE Management in Emerging Multimedia Services». – IEEE Communications Magazine, 2012, Т. 50, № 4. – С. 18–27.
18. Hossfeld T., Fiedler M., Tran-Gia P. «QoE-Based Network and Application Management». – Computer Networks, 2011, Т. 55, № 7. – С. 1141–1153.
19. Schatz R., Egger S. «On the Correlation of QoE and QoS in Networks». – Proc. IEEE ICC, 2012. – С. 190–195.
20. Sousa I., et al. «A Survey on QoE-Oriented Wireless Resource Scheduling». – Computer Networks, 2020, Т. 171. – С. 106–121.
21. Zhang H., Liu N., Chu X., Long K., Aghvami A., Leung V. C. M. «Network Slicing Based 5G and Future Mobile Networks: Mobility, Resource Management and Challenges». – IEEE Communications Magazine, 2017, Т. 55, № 8. – С. 138–145.
22. Pessemier T. D., De Marez L., Martens L. «An End-User-Centric Method for QoE Optimization in Multimedia Services». – Computers in Human Behavior, 2013, Т. 29, № 6.
23. Wang J., Huang Y., Schober R., You X. «Distributed Optimization of Hierarchical Small-Cell Networks: A GNEP Framework». – IEEE Transactions on Communications, 2016, Т. 64, № 10.
24. Lam P., et al. «User-Centric Network Design for 5G: A Survey of Key Technologies». – IEEE Access, 2017, Т. 5.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ

КВАЛІФІКАЦІЙНА РОБОТА

ОПТИМІЗАЦІЯ АРХІТЕКТУРИ ТЕЛЕКОМУНІКАЦІЙНИХ МЕРЕЖ З УРАХУВАННЯМ ВИМОГ КОРИСТУВАЧІВ

ВИКОНАВ:
СТУДЕНТ ГРУПИ ТСДМ-62
ЄВГЕНІЙ ЛІСОВЕНКО

МЕТА РОБОТИ ТА АКТУАЛЬНІСТЬ

Мета роботи – аналіз, удосконалення та оптимізація архітектури телекомунікаційних мереж з урахуванням вимог до якості користувацького досвіду (QoE), пропускної здатності, затримок, масштабованості та стійкості мереж при зростанні трафіку, використанні IoT-рішень та потокових сервісів.

Об’єкт дослідження – архітектура сучасних телекомунікаційних мереж, які забезпечують передавання мультимедійних та сервісних даних у цифровому середовищі.

Предмет дослідження – методи та технології оптимізації мережевої архітектури.

Актуальність.

З появою 5G, IoT, відеострімінгу та сервісів у реальному часі проблема оптимізації архітектури телекомунікаційних мереж набуває критичного значення. Крім того, користувачі вже не просто “споживають мережу” — вони очікують миттєвого завантаження, адаптивної якості контенту, безперервності сервісів та високого рівня обслуговування (QoE). Саме тому виникає потреба у конвергентних (гібридних) архітектурах, здатних поєднувати централізоване та децентралізоване управління, використовуючи інтелектуальне балансування навантаження, on-demand transcoding, диференціальні оновлення IoT та нові мережеві протоколи (HTTP/2, HTTP/3).

Ключові вимоги до мережевої архітектури

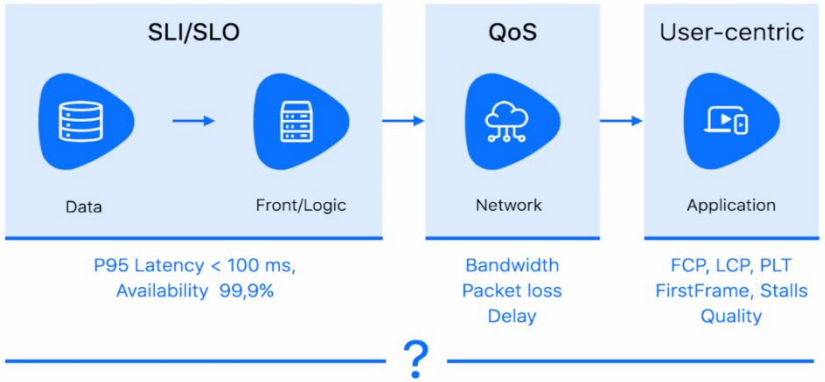
Метрики користувацького досвіду (QoE)

Якість користування сервісом оцінюють через суб'єктивні метрики — CSat, NPS, CSI, TRI*M, які вимірюються опитуваннями та відображають рівень задоволення й готовність рекомендувати продукт. Паралельно застосовуються технічні метрики: SLI/SLO, latency, packet loss, PLT, TTI, LCP, які моніторяться системами спостереження.

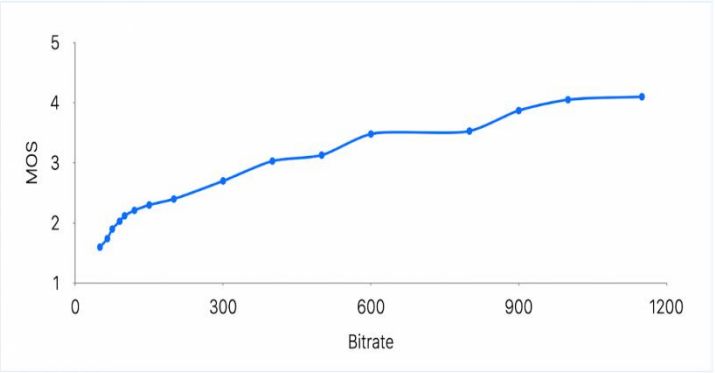
На рисунку 1 наведено приклади ключових метрик оцінки сервісу. Усі ці показники об'єднуються в інтегральну метрику QoE (Quality of Experience), що враховує як технічні параметри, так і поведінку користувачів. Її часто вимірюють через MOS (Mean Opinion Score) — оцінку якості за п'ятибальною шкалою. На рисунках 2 та 3 показано, як MOS залежить від бітрейту та FPS:низький FPS (15) помітно знижує якість, проте збільшення понад 60 FPS майже не впливає на сприйняття відео.

На рисунку 4 наведено залежність якості від Page Load Time (PLT) — затримка понад ~200 мс різко погіршує оцінку QoE.

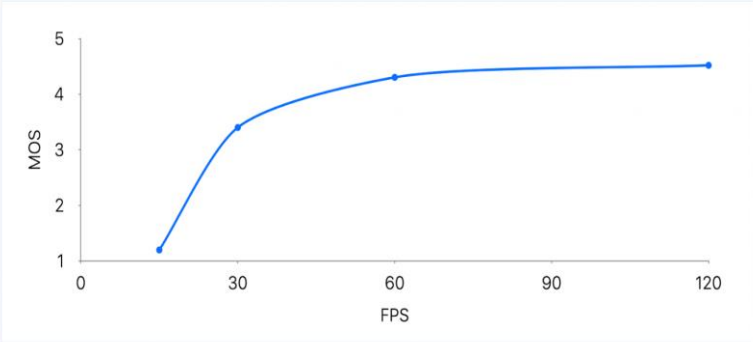
Отже, ефективне управління QoE потребує балансу між технічними характеристиками та очікуваннями користувачів.



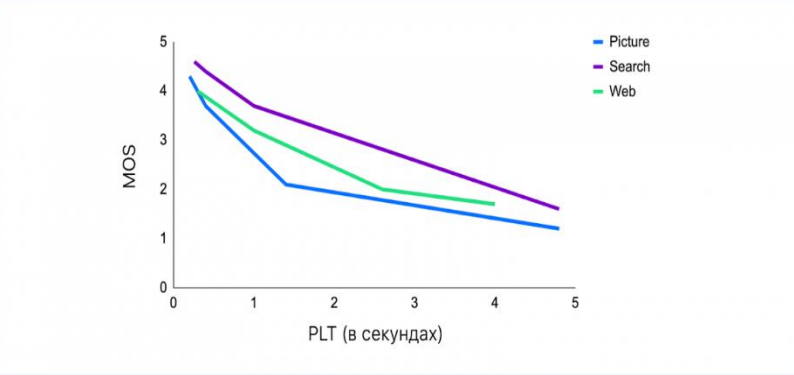
Метрики для оцінки



Залежність MOS від бітрейту відео



Залежність MOS відносно частоти кадрів (FPS)



Вимоги до Page Load Time (PLT)

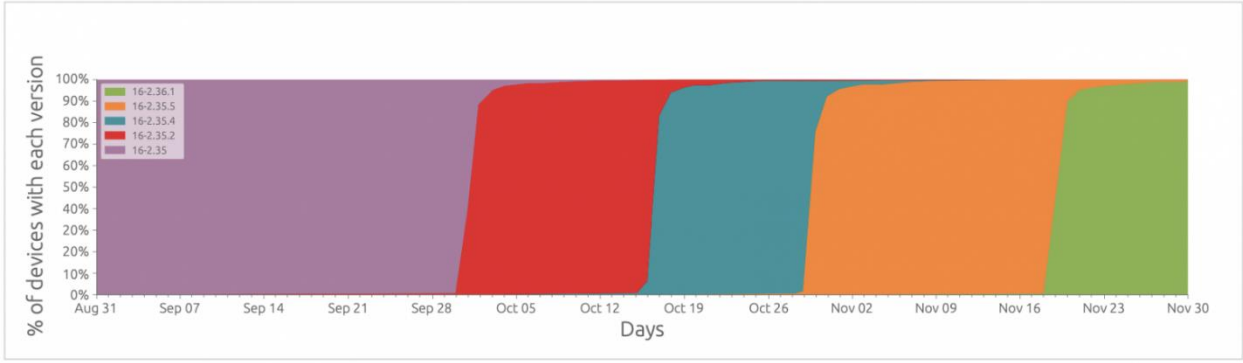
Оновлення snap пакетів

Зі зростанням кількості IoT-пристроїв різко збільшується навантаження на мережу. Щоб уникнути затримок та перевантаження каналів, доцільно застосовувати **диференціальні snap-пакети**, які передають лише змінені частини програмного забезпечення. Це значно скорочує трафік і прискорює OTA-оновлення.

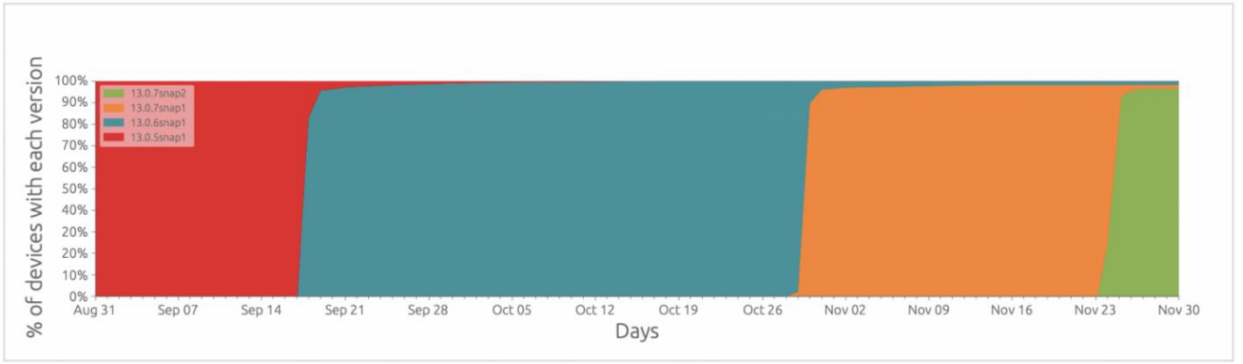
На рисунку 1 показано, що вже протягом **першого тижня понад 90 % пристроїв** отримують оновлення пакета Core — тобто механізм snap забезпечує дуже швидке поширення версій у мережі. **На рисунках 2 та 3** демонструється аналогічна поведінка для пакетів *Nextcloud* та *Canonical LXD*: оновлення розповсюджуються практично миттєво, що підтверджує ефективність механізму.

Таким чином, snap-підхід:

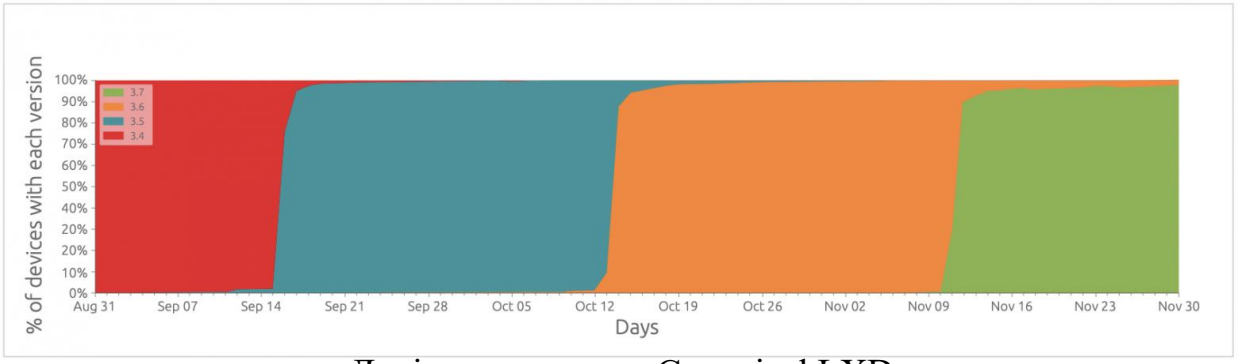
- зменшує обсяг переданих даних;
- забезпечує транзакційне оновлення та автоматичний rollback;
- підтримує централізоване масове розгортання;
- гарантує стабільність навіть у нестабільних мережах.



Частота випусків та поширення оновлень для snap-пакету Core



Дані для програмного забезпечення Nextcloud для власних хмарних сервісів



Дані для програми Canonical LXD

Диференціальні snap пакети

Диференціальні snap-пакети — це механізм **оновлення тільки змінених сегментів** пакета, а не повного файлу. Завдяки цьому оновлення стають **у 3–4 рази швидшими** й потребують значно **менше трафіку**.

Всі пакети надходять із **Snap Store**, де автоматично генеруються «дельта-файли» для оновлення або відкату (rollback). Це **особливо важливо для IoT**, де потрібна стабільність та мінімальні простої. Створення таких оновлень не змінює робочий процес розробників — Snapcraft автоматично формує диференціальний файл.

Переваги:

економія трафіку (до **70%**),

швидше оновлення,

стабільність навіть на слабких пристроях,

прозорий цикл публікації — CI/CD не змінюється.

Ім’я snap-пакета	Середнє число оновлень на місяць	Середній розмір повного пакета (МБ)	Середній розмір диференціального пакета (МБ)	Трафік за місяць без диференціальних (ГБ)	Трафік за місяць із диференціальними (ГБ)
0ad	0.58	897.46	185.32	0.510	0.105
brave	2.11	179.81	93.01	0.370	0.191
bw (bitwarden)	1.01	14.75	4.73	0.015	0.005
core *	1.81	82.08	31.28	0.145	0.055
darktable	0.37	92.96	32.38	0.034	0.012
firefox	2.89	182.48	85.09	0.516	0.240
gimp	1.21	155.18	32.32	0.183	0.038
gitkraken	2.91	161.75	35.08	0.460	0.100
gnome-3-26-1604 *	1.32	133.33	25.19	0.172	0.033
gnome-calculator *	1.57	2.61	1.06	0.004	0.002
gnome-characters *	1.72	15.92	0.94	0.027	0.002
gnome-logs *	1.50	18.18	4.95	0.027	0.007
gnome-system-monitor *	1.66	5.52	0.92	0.009	0.001
gtk-common-themes *	**1.60**	**36.24**	**7.18**	**0.057**	**0.011**
jgalaxian	**0.57**	**87.93**	**21.19**	**0.049**	**0.012**
notepad-plus-plus	**2.95**	**34.70**	**1.98**	**0.100**	**0.006**
ruby	**4.29**	**32.24**	**14.56**	**0.135**	**0.061**
sftpclient	**8.85**	**207.05**	**64.90**	**1.789**	**0.561**
skype	**5.81**	**139.70**	**42.26**	**0.793**	**0.240**
slack	**0.54**	**129.42**	**56.22**	**0.068**	**0.030**
taskbook	**0.68**	**23.81**	**12.36**	**0.016**	**0.008**
tmnationsforever	**0.97**	**125.68**	**8.31**	**0.119**	**0.008**
vscode	**0.57**	**200.06**	**6.18**	**0.111**	**0.003**
vlc	**1.92**	**110.50**	**27.95**	**0.207**	**0.052**
wifi-ap	**0.26**	**25.86**	**9.48**	**66.12**	**24.25**
fictional-snap	**1.12**	**175.48**	**51.56**	**1921.05**	**564.46**
			Сума	**5.394**	**1.682**
			Економія в %	**68.82%**	

Масштабування та резервування у телекомунікаційних системах

У міру зростання навантаження один сервер перестає справлятися, тому необхідні **горизонтальне масштабування та відмовостійкість**. Розподіл трафіку між кількома вузлами забезпечує стабільність, а реплікація серверів дозволяє уникнути простоїв.

Критично важливо використовувати **резервні дата-центри**, адже навіть дубльовані вузли в одному ЦОДі створюють єдину точку відмови. Балансувальники навантаження також повинні мати **failover-механізм**, щоб автоматично виключати непрацюючі вузли.

NFS зручна, але нестабільна при збоях — альтернативи: **Ceph, MooseFS**, обов'язково з бекапами.

CI/CD не завжди доцільний — важливо мати **надійний rollback**.

Основні "вузькі місця" — **реплікація БД, файлові системи та неефективні SQL-запити**. Потрібні моніторинг, шардінг, оптимізація таблиць.

```bash
#!/bin/bash
# Простий деплой коду з можливістю відкату

PROJECT_DIR="/var/www/project"
BACKUP_DIR="/var/backups/project_$(date +%F_%H-%M-%S)"

echo "=== Створення резервної копії ==="
cp -r $PROJECT_DIR $BACKUP_DIR

echo "=== Отримання останньої версії з Git ==="
cd $PROJECT_DIR
git pull origin main

echo "=== Оновлення завершено ==="
echo "Якщо потрібно відкотитися, використайте:"
echo "rm -rf $PROJECT_DIR && cp -r $BACKUP_DIR $PROJECT_DIR"
```

Скрипт для безпечного оновлення коду без складних CI/CD систем

```sql
-- Виявлення 10 найповільніших запитів у MySQL
SELECT
    db AS 'База даних',
    COUNT(*) AS 'Кількість виконань',
    ROUND(SUM(TIME_MS)/1000, 2) AS 'Сумарний час (сек)',
    DIGEST_TEXT AS 'SQL-запит'
FROM performance_schema.events_statements_summary_by_digest
ORDER BY SUM(TIME_MS) DESC
LIMIT 10;
```

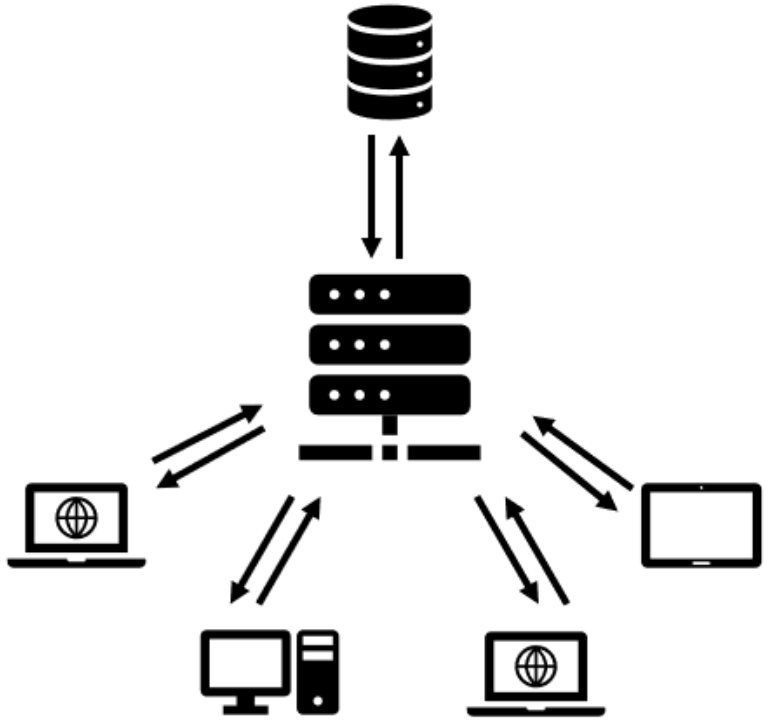
Приклад коду для збору статистики повільних запитів MySQL

Клієнт-серверна архітектура

Клієнт-серверна архітектура — це модель, у якій клієнт формує запит, а сервер обробляє його та надає ресурси. Такий підхід забезпечує централізоване зберігання даних, контроль безпеки й ефективне управління потоком інформації. Структура зазвичай має три рівні: клієнтський інтерфейс, логіку обробки та базу даних.

Переваги: централізація, керованість, обробка даних у реальному часі, масштабованість. **Недоліки:** обмежена кількість підключень, ризик перевантаження, залежність від центрального сервера.

Сьогодні модель еволюціонує в бік хмарних та гібридних рішень (SDN, Edge Computing), але принцип «запит → обробка → відповідь» залишається базовим для телекомунікаційних систем.



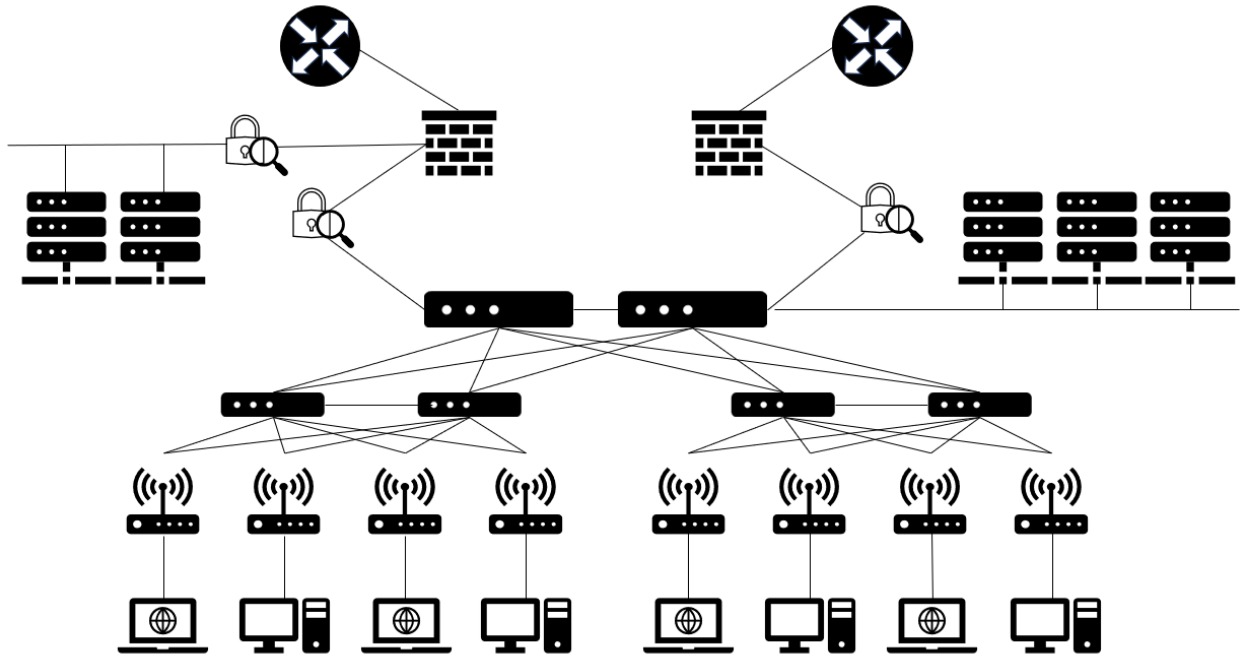
Клієнт-серверна архітектура

Хабова архітектура телекомунікаційних систем

Хабова архітектура — це централізована модель, де всі пристрої підключені до одного головного вузла (хаба), через який проходить весь трафік. Такий підхід забезпечує повний контроль над обміном даними, безпекою та доступом, але створює залежність від центрального елемента.

Модель легко масштабувати та сегментувати, тому вона широко застосовується у корпоративних мережах, дата-центрах, SCADA-системах та банківських структурах.

Сучасні рішення реалізують хаби у віртуальній формі (SDN/NFV), що дозволяє автоматизувати маршрутизацію, моніторинг і балансування навантаження.

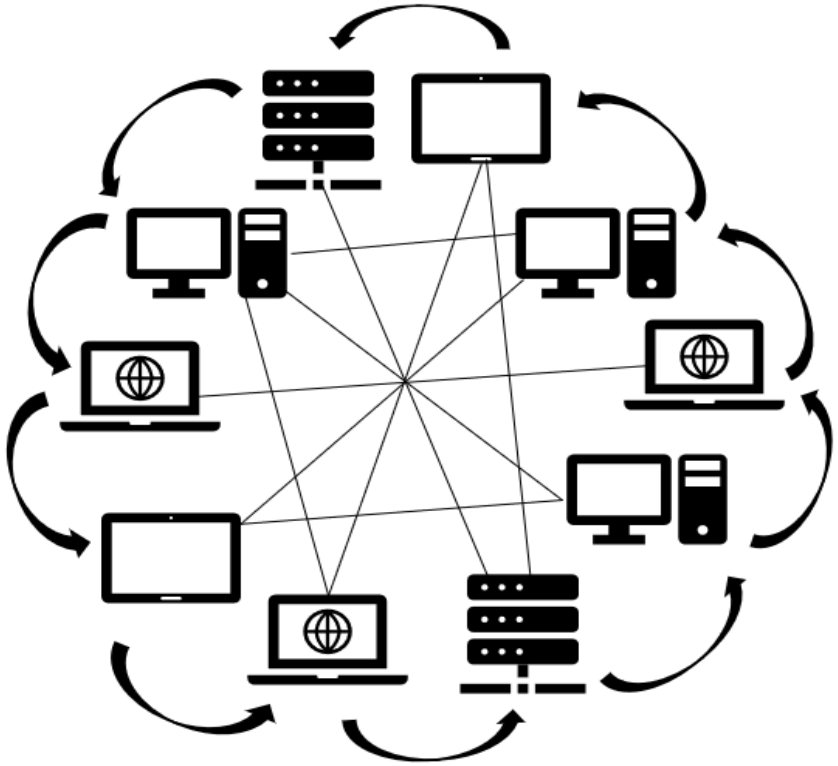


Хабова архітектура

Пірингова архітектура телекомунікаційних систем

Пірингова (P2P) архітектура — це децентралізована модель, де кожен вузол мережі одночасно може бути клієнтом і сервером. Вона не потребує центрального керування, забезпечує автономність, стійкість до збоїв і можливість обміну даними без посередників.

Завдяки MESH-топології та алгоритмам маршрутизації (наприклад, DHT) система автоматично перебудовує шляхи передачі даних. Така модель має низькі затримки, добре масштабується та економно використовує ресурси.



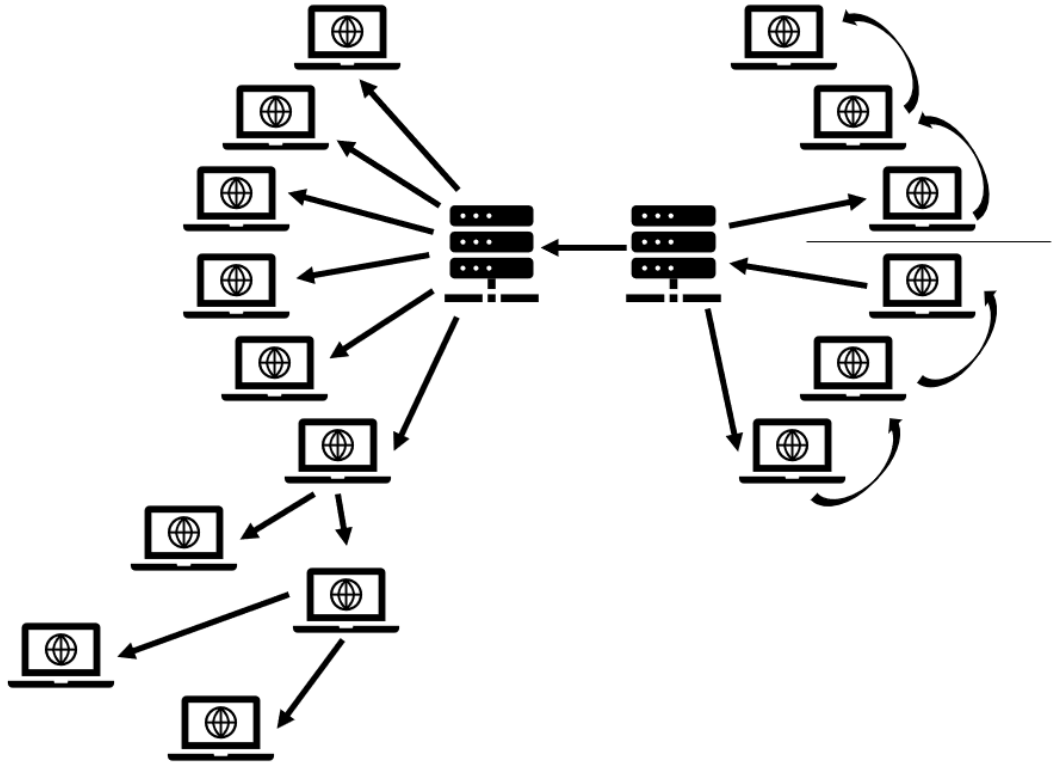
Іллюстрація пірингової архітектури

Конвергентна архітектура телекомунікаційних систем

Класичні моделі телекомунікаційних систем — клієнт-серверна, хабова та пірингова — вже не задовольняють вимоги до масштабованості, безпеки та гнучкості. Тому формується новий підхід — **конвергентна архітектура**, що поєднує централізоване управління з децентралізованою стійкістю.

Конвергентна модель використовує багаторівневу структуру з локальними та регіональними вузлами, підтримує автономність сегментів, інтелектуальну маршрутизацію, а також інтеграцію з хмарними технологіями. Це забезпечує передачу даних у реальному часі та високу стійкість системи.

Переваги:– висока безпека та сегментування;– робота без центрального вузла;– підтримка тисяч пристроїв;– гнучке масштабування.



Іллюстрація конвергентної архітектури

ВИСНОВКИ

- У роботі проведено аналіз сучасних архітектур телекомунікаційних мереж та розглянуто методи їх оптимізації з урахуванням потреб користувачів.
- Досліджено чотири основні моделі: клієнт-серверну, хабову, пірингову та конвергентну.
- Встановлено, що конвергентна (гібридна) архітектура є найбільш збалансованим рішенням, оскільки поєднує централізований контроль із децентралізованою стійкістю до збоїв.
- Результати показали, що ключовим принципом оптимізації мереж є адаптивність — здатність системи реагувати на зміну навантажень, обсягів трафіку та вимог до QoE/QoS.
- Також підкреслено важливість ефективного управління трафіком, автоматизації, балансування навантаження та прогнозування стану мережі.
- Зроблено висновок, що такі підходи формують основу для розвитку мереж 5G, IoT, 6G та інтелектуальних комунікаційних систем.

Дякую за увагу!

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Євгеній ЛІСОВЕНКО

Назва: ОПТИМІЗАЦІЯ АРХІТЕКТУРИ ТЕЛЕКОМУНІКАЦІЙНИХ МЕРЕЖ З УРАХУВАННЯМ ВИМОГ КОРИСТУВАЧІВ

Координатор: Олександр ДРОБИК

Підрозділ: ННІТ

Коефіцієнт подібності 1:1.5

Коефіцієнт подібності 2:0.9

Тривога: 0

Після аналізу Звіту подібності констатую наступне:

☐ виявлені в роботі запозичення є сумлінними і не мають ознак плагіату. Тому робота визнається самостійною і допускається до захисту;

☐ виявлені в роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і самостійності її автора. Роботу направити на доопрацювання;

☐ виявлені в роботі запозичення є недобросовісними і мають ознаки плагіату або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. У зв'язку з чим, робота не допускається до захисту.

…… ……

Дата *Підпис Наукового керівника*