МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ТЕХНОЛОГІЇ СУЧАСНИХ КІБЕР-ФІЗИЧНИХ СИСТЕМ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для студентів, які навчаються за спеціальністю 151 "Автоматизація та комп'ютерно-інтегровані технології", освітньопрофесійною програмою "Автоматизація та комп'ютерно-інтегровані технології кібер-енергетичних систем"

Київ
КПІ ім. Ігоря Сікорського 2020

Технології сучасних кібер-фізичних систем: Навчальний посібник [Електронний ресурс]: навч. посіб. для студ. спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», освітньо-професійна програма «Автоматизація та комп'ютерно-інтегровані технології кібер-енергетичних систем»; укладач: Ю.Є. Грудзинський. – Електронні текстові дані (1 файл: 14,8 МБ). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 327 с.

Гриф надано Методичною радою КПІ імені Ігоря Сікорського (протокол № 4 від 10.12.2020 р.) за поданням Вченої ради факультету (протокол № 5 від 30 листопада 2020 р.)

Електронне мережне навчальне видання

ТЕХНОЛОГІЇ СУЧАСНИХ КІБЕР-ФІЗИЧНИХ СИСТЕМ

Укладач: *Грудзинський Юліан Євгенович, ст. викладач*

Відповідальний редактор: *Баган Т.Г.*, к.т.н, доцент

Рецензент: *Аушева Н.М.*, д.т.н., професор, КПІ ім. Ігоря Сікорського

Навчальний посібник призначений для студентів, які навчаються за спеціальністю 151 «Автоматизація та комп'ютерно інтегровані технології», освітньо-професійна програма «Автоматизація та комп'ютерно інтегровані технології кібер-енергетичних систем», що вивчають курс «Технології сучасних кібер-фізичних систем».

Метою посібника «Технології сучасних кібер-фізичних систем: Навчальний посібник» є формування у студентів знань та уявлень про побудову сенсорних мереж та взаємодію пристроїв у таких мережах.

ЗМІСТ

ВСТУП

Ви прокидаєтеся у п'ятницю, 20 травня 2030 року, близько 6:00 ранку київського часу, як завжди. Ваші очі помічають фантастично красивий сонячний ранок, на вулиці 25 °C. Ви проживете день, який буде зовсім іншим, ніж сьогоднішній день 2019 року. Все, що стосується вашого дня, вашого способу життя, вашого здоров'я, ваших фінансів, вашої роботи, вашої поїздки, навіть вашого місця для паркування буде різним. Світ, в якому ви живете, буде відмінним від сьогодення: енергетика, охорона здоров'я, сільське господарство, виробництво, логістика, громадський транспорт, охорона навколишнього середовища, безпека, магазини та навіть одяг. Це вплив підключення звичайних об'єктів до Інтернету або промислового Інтернету речей (IoT).

Перед тим, як ви пробудилися, багато чого відбулося в IoT, що вас оточує. Ваш сон контролюється давачем сну або смарт-подушкою. Дані про сон надсилалися до шлюзу IoT, а потім передавалися до хмарного сервісу, яким ви користуєтеся безкоштовно, і інформація з якого надходить на інформаційну панель вашого смартфону. Вам не потрібен будильник. Ваш кондиціонер з двома зонами обслуговування підключено до іншого постачальника послуг у хмарі через мережу 802.11 Wi-Fi вашого дому, так само, як і вашу пожежну сигналізацію, дзвінок, системи зрошення, двері гаража, камери спостереження та системи безпеки. Вашу собаку чіповано давачем наближення, що дозволяє показувати вам, де він знаходиться.

У вас більше немає комп'ютера. Ні, Ви звичайно, маєте комп'ютер у стилі планшета і смартфон, як свій центральний пристрій творіння, але ваш світ заснований на використанні VR/AR Goggles, оскільки цей екран набагато кращий і більший. У вашій шафі є шлюз для туманних обчислень. Він підключений до постачальника послуг 5G, щоб тримати вас в Інтернеті та глобальній мережі, оскільки проводові з'єднання вже не працюють для вашого способу життя: ви мобільний, підключений онлайн, незалежно від того, де ви знаходитесь - на роботі, вдома, чи в Національному парку Кіллиманджаро. Шлюз також виконує багато дій у вашому домі для вас, наприклад, обробка відеопотоків з веб-камер для виявлення проникнення або аварії в будинку.

Система безпеки постійно перевіряється на наявність аномалій (дивні шуми, можливі витоки води, залишилося світло, ваша собака знову жує меблі). Граничний вузол також виступає як домашній хаб, щодня створює резервну копію вашого телефону, тому що ви маєте тенденцію втрачати їх і слугує вашою приватною хмарою, навіть якщо ви нічого не знаєте про хмарні служби.

Ви їдете на велосипеді до офісу. Ваш одяг використовує давачі, які контролюють ваш пульс і температуру. Ці дані передаються через Bluetooth Low Energy на смартфон одночасно з тим, що ви слухаєте "Океан Ельзи" через аудіосигнал Bluetooth, переданий з телефону на навушники Bluetooth. По дорозі на роботу ви проїджаєте декілька рекламних щитів, на яких відображаються відео та оголошення в реальному часі. Ви зупиняєтеся у вашому місцевому магазині та п'єте каву саме таку, яку замовляли останній раз: 200 мл з вершками. Це було зроблено за допомогою маячка і шлюзу, що розпізнає вашу присутність у межах 1,5 метра від автомата кави. Більшість людей приїжджають на роботу своїм авто і направляються на оптимальне місце для паркування за допомогою розумних давачів у кожному парковочному слоті.

Ваш офіс є частиною програми зеленої енергетики щодо офісного приміщення з нульовими викидами. У кожному приміщенні є давачі близькості. Ваше ім'я-значок, за допомогою якого ви потрапляєте в офіс, одночасно є маячковим пристроєм на 10-річній батареї. Про Вашу присутність стає відомо, як тільки ви заходите у двері: вмикається світло, система опалення, вентиляції і кондиціювання, стельові вентилятори, навіть цифрові вивіски. Центральний вузол туману відстежує всю інформацію по будівлі та синхронізує її з хост-комп'ютером. Запропоновано механізм правил для прийняття рішень у реальному часі на основі зайнятості, часу доби, сезону року, а також внутрішніх і зовнішніх температур. Умови навколишнього середовища в офісі змінюються для мінімального використання енергії.

Це все робиться в реальному часі за допомогою декількох потокових граничних обчислень аналітики і алгоритмів машинного навчання, які були раніше підготовлені у хмарі і завантажені в граничний обчислювач. В офісі розміщено невелику кількість стільників 5G для зовнішнього зв'язку, але вони також містять і велику кількість шлюзів малого розміру (десятки метрів) для

внутрішнього зняття сигналів у межах будівлі. Внутрішній 5G одночасно є і локальною мережею.

Ваш смартфон і планшет переключилися на внутрішній сигнал 5G, і ви вмикаєте програмне накладання власної носимої мережі на корпоративну. Ваш смартфон робить багато роботи для вас: це, по суті, ваш особистий шлюз до вашої носимої мережі, що оточує ваше тіло. Сьогодні ви вирушаєте на свою першу зустріч, але ваш колега не приїхав і запізнився на декілька десятків хвилин. Він приносить вибачення, але пояснює, що його новий автомобіль поінформував виробника про аномалії в компресорі і турбокомпресорі. Зразу було вирішено і строки ремонту.

Того дня вам потрібно зайти на виробничий майданчик в іншій частині міста міста. Це типова фабрична обстановка: кілька машин для лиття під тиском, пристрої для збирання, місця для упаковки, а також вся підтримуюча інфраструктура. Останнім часом якість продукту погіршується. Кінцевий продукт має проблеми з підключенням і поступається в якості продукту минулого місяця. Прибувши на фабрику, ви розмовляєте з менеджером і оглядаєте виробництво. Все виглядає нормальним, але якість, безумовно, була маргіналізована. Ви проходите до пульту керування та відлагоджуєте процес з панелі приладів.

Система використовує ряд давачів (вібрації, температури, швидкості, зір і маячки стеження) для контролю за підлогою. Дані накопичуються і візуалізуються в реальному часі. Існує ряд алгоритмів прогнозування, які спостерігають за різними пристроями для виявлення ознак їх зносу та відмов у роботі. Ця інформація передається також виробнику обладнання та вашій команді. У вас тепер є багато даних. Всі дані з фабрики зберігаються в історичній базі даних довгострокового зберігання. Беручи всі історичні дані через комплексний процесор подій та пакет аналітики, ви швидко розробляєте набір правил, які моделюють якість частин, що відмовляють. Працюючи в зворотному напрямку до подій, які призвели до невдач, ви починаєте розуміти їх чинники:

- для економії електроенергії у літні місяці внутрішня температура робочого простору була зменшена на 2 °C;
- зменшення обсягу виробництва на 1,5% за рахунок поставок;

- одна з формувальних машин наближається до часу проходження капітального ремонту, про що вказує температура і швидкість складання.

Саме так буде виглядати у недалекому майбутньому типовий день спеціаліста з автоматизації. А допоможе у цьому використання технологій Інтернету речей, які викладено у даному посібнику.

Навчальний посібник відповідає робочій програмі навчальної дисципліни «Технології сучасних кібер-фізичних систем», яка читається студентам спеціальності 151 «Автоматизація та комп'ютерно інтегровані технології» освітньопрофесійної програми «Автоматизація та комп'ютерно інтегровані технології кібер-енергетичних систем».

Теорія побудови світу Інтернету речей – одна з основних навчальних дисциплін, яка формує у магістрів знання про загальні принципи і процеси функціонування сучасних кібер-фізичних систем.

Мета дисципліни – формування у студентів знань, умінь та навичок з розв'язання задач аналізу та синтезу екосистеми Інтернету речей у сучасній системі керування технологічними процесами.

Після вивчення курса Магістр повинен знати (мати):

- фундаментальні принципи побудови екосистеми Інтернету речей, їх класифікацію за основними ознаками та особливості;
- різновиди давачів та виконуючих механізмів, особливості енергозабезпечення живлення пристроїв, що застосовуються в IIoT;
- основи теорії зв'язку, особливості поширення радіохвиль різних частотних діапазонів сенсорних мереж, різновиди інтерференції сигналів;
- засади теорії інформації: теорему Шеннона, коефіцієнт помилок, поняття вузькосмугового та широкосмугового зв'язку;
- принципи побудови сенсорних мереж на IP та не IP технологіях;
- принципи побудови чарункових мереж;

- принципи побудови сенсорних мереж далекої дії;
- основи безпеки в сенсорних мережах;
- різновиди групового обладнання для сенсорних мереж і особливості його роботи;
- IIoT протоколи обміну даними MQTT та використання WEB-сокетів;
- уявлення про хмарні, граничні та туманні обчислення, моделі хмарних сервісів, безпеку в хмарах;
- топологію та оркестрування граничних обчислень, вимоги безпеки до них;
- архітектуру туманних обчислень, поняття сервісів та віртуалізації, різновиди туманної топології;
- поняття про Big Data, його характеристики;
- архітектуру Apach Hadoop: розподілену файлову систему HDFS, Map Reduce та YARN;
- використання аналітики даних на хмарних та туманних платформах.

Магістр повинен вміти :

- враховувати інтерференцію в локальних і глобальних мережах, визначати доступну швидкість передачі даних від вузла до вузла чи шлюза;
- оцінити відмовостійкість системи і вартість можливої втрати даних;
- вибрати інтернет-протоколи: MQTT або CoAP чи AMQP;
- продумати, як все це буде працювати в разі переходу на інший хмарний сервіс;
- вирішити, в якій точці буде здійснюватися обробка даних: поруч з джерелом (на границі, в тумані), чи у хмарі;
- вибирати відповідний інструмент аналітики;
- вибирати відповідні джерела енергоживлення для пристроїв сенсорної мережі;
- забезпечувати надійність та безпечність роботи усієї екосистеми Інтернету речей.

1 ІСТОРІЯ СТВОРЕННЯ ТА СФЕРИ ВИКОРИСТАННЯ "ІНТЕРНЕТУ РЕЧЕЙ"

1.1 Історія Інтернету речей

Термін «інтернет речей», зобов'язаний своєю появою Кевіну Ештону, який в 1997 році, працюючи на компанію Proctor&Gamble, застосував технологію радіочастотної ідентифікації (RFID) для керування системою поставок. Завдяки цій роботі в 1999 році його запросили в Масачусетський технологічний інститут, де він з групою однодумців організував дослідний консорціум **Auto-ID Center** (більш детальну інформацію можна знайти на сайті www.smithsonianmag.com/innovation/kevin-ashton-describes-the-internet-of-things-180953749). З тих пір Інтернет речей звершив перехід від простих радіочастотних міток до екосистеми і індустрії. Аж до 2012 року ідея підключення речей до Інтернету переважно відносилася до смартфонів, планшетів, ПК і ноутбуків. По суті, до тих пристроїв, які в усіх відношеннях виступають в якості комп'ютера. До цього, з моменту появи перших зачатків Інтернету (таких як створена в 1969 р. мережа ARPANET), більшості технологій, на яких будується Інтернет речей, просто не існувало. До 2000 року більшість пристроїв, які можна було підключити до Інтернету, представляло собою комп'ютери різних розмірів. Нижче показано поступове підключення речей до Інтернету [3]:

- 1973 - Маріо У. Кардулло отримує патент на першу радіо-частотну мітку
- 1982 - Підключений до Інтернету автомат з газованою водою в університеті Карнегі-Меллон
- 1989 - Підключений до Інтернету тостер на конференції Interop '89
- 1991 - Компанія HP представила HP LaserJet IIISi: перший підключений до мережі Ethernet мережевий принтер
- 1993 - Підключена до Інтернету кавоварка в Кембриджському університеті (перша підключена до Інтернету камера)
- 1996 - Підрозділ General Motors OnStar (дистанційна діагностика 2001)

- 1998 - Поява організації Bluetooth SIG
- 1999 - Холодильник LG Internet Digital DIOS
- 2000 - Перші прояви розробленої компанією HP концепції всепроникної комп'ютеризації (Cooltown): HP Labs, система обчислювальних і комунікаційних технологій, які в поєднанні один з одним створюють підключення до Інтернету для людей, місць і об'єктів
- 2001 - Випуск першого пристрою, що використовує технологію Bluetooth: мобільний телефон KDDI з підтримкою Bluetooth
- 2005- Міжнародний союз електрозв'язку, спеціалізована установа ООН, випустив звіт, в якому вперше були сформульовані прогнози розвитку Інтернету речей
- 2008 - Поява першого IoT-спільноти IPSO Alliance, метою якого було сприяння підключенню речей до Інтернету
- 2010 - Успішна розробка напівпровідникових світлодіодних ламп привела до розвитку концепції розумного освітлення
- 2014 - Компанія Apple створила протокол iBeacon для маячків

Як можна впевнитися з малюнка нижче, в майбутньому IoT захопить практично кожен сегмент в сферах енергетики, промисловості, бізнесу, охорони здоров'я і споживчих товарів

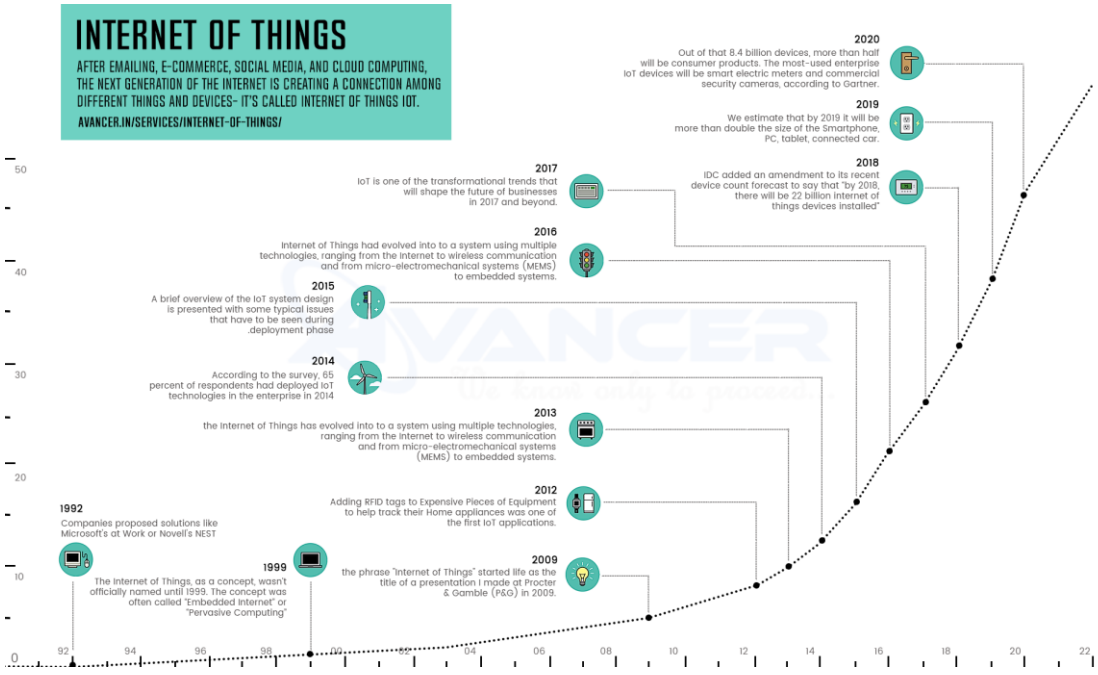


Рисунок 1.1 Розвиток Інтернету речей у часі

1.2 Використання у промисловості та на виробництві

Промислові IoT (IIoT) - один з найшвидших і найбільших сегментів у загальному просторі IIoT за кількістю пов'язаних речей і цінністю, яку ці послуги надають виробництву та автоматизації виробництва. Цей сегмент традиційно був світом операційної технології (OT). Вона включає до себе апаратні та програмні засоби моніторингу фізичних пристроїв. Традиційні ролі інформаційних технологій застосовувалися інакше, ніж ролі OT. OT зачіпає метрики прибутковості, безвідмовної роботи обладнання, збір даних у реальному часі та відповідь на них, а також безпеку систем. Роль IT зосереджується на безпеці, групуванні, доставці даних куди треба і службах. Оскільки IIoT стає переважаючим у промисловості та виробництві, обидва світи IT та OT об'єднуються, поєднуючись з предикативним обслуговуванням тисяч фабрик та виробничих машин, щоб доставляти безпрецедентну кількість даних до приватних та громадських хмарних інфраструктур.

Деякі характеристики цього сегмента включають в себе необхідність надання практичних рішень в режимі реального часу для OT. Це означає, що затримка даних є серйозною проблемою для IIoT у промисловому сегменті. Крім того, болючими точками є можливі простої обладнання (що призводять до втарти частини прибутку) та безпека (що може призвести не тільки до втрати прибутку, а й до масових катастроф). Це призводить до необхідності резервування і організації безпеки мереж приватних хмар та зберігання даних. Промисловий сегмент є одним з найбільш швидкозростаючих ринків. Одним з нюансів цієї галузі є залежність від старих технологій, тобто апаратні та програмні інтерфейси, які вже існують на виробництві не можуть так просто інтегруватися в IIoT. Наприклад, часто 30-річне обладнання використовує послідовний інтерфейс RS-485, а не сучасні бездротові mesh-мережі.

Нижче наведені промислові та виробничі випадки використання IIoT та їх вплив:

- профілактика обслуговування нового та існуючого обладнання;
- зростання виробництва завдяки аналізу попиту в реальному часі;
- економія енергії;

- підвищення безпеки виробництва;
- застосування експертних систем прямо на виробництві.

1.3 Використання в енергетичній сфері

Енергетичний сегмент включає моніторинг виробництва, передачі та споживання електроенергії від виробника до споживача. Значний обсяг досліджень і розробок в області IIoT зосереджується на споживчих і комерційних енергетичних пристроях обліку електроенергії та тепла, які можуть здійснювати передачу поточних даних споживання ресурсів постачальникам без участі людини, за допомогою сучасних безпровідних давачів з низьким енергоспоживанням (що забезпечує час роботи до 10 років без заміни живлення) і відповідними протоколами.

Багато енергетичних виробничих потужностей знаходяться у віддалених районах або небезпечному для здоров'я людини середовищі, таких як пустельні райони для сонячних батарей, круті схили гір чи морська поверхня для вітрових електростанцій, внутріні частини ядерних реакторів. Поточні дані від цих об'єктів можуть знадобитися у реальному часі або в режимі реального часу для адекватного реагування персоналу на критичну ситуацію системи управління виробництвом енергії (подібно до виробничих систем). Це може вплинути на те, як саме слід розгортати системи IIoT на таких об'єктах.

Нижче наведено деякі випадки використання IIoT в енергетиці:

- аналіз тисяч давачів і вузлів збору даних нафтової платформи для підвищення ефективності видобутку нафти/газу;
- віддалений моніторинг та обслуговування сонячних панелей;
- аналіз небезпечних ядерних об'єктів;
- розумні лічильники тепла та електроенергії в загальноміському розгортанні для моніторингу споживання ресурсу та попиту на нього;
- керування в реальному часі в залежності від погоди віддаленими вітровими турбінами.

1.4 Використання у технологіях "розумного міста"

"Розумне місто" - концепція інтеграції декількох інформаційних і комунікаційних технологій (ІКТ) та IoT для управління городським майном, таким як активи міста. Активи міста включають до себе (але не обмежуються!) місцеві відділи інформаційних систем, школи, бібліотеки, транспорт, лікарні, електростанції, системи водопостачання та управління відходами, правоохоронні органи та інші громадські служби. Метою створення "розумного міста" є поліпшення якості життя за допомогою технології міської інформатики для підвищення ефективності обслуговування і задоволення потреб резидентів. ІКТ дозволяють міській владі безпосередньо взаємодіяти з спільнотами і міською інфраструктурою, стежити за тим, що відбувається в місті, як місто розвивається, і які способи дозволяють поліпшити якість життя. За рахунок використання давачів, інтегрованих в режимі реального часу, накопичені дані від міських жителів і пристроїв обробляються і аналізуються. Зібрана інформація є ключем до вирішення проблем ефективності.

ІКТ використовуються для підвищення якості, продуктивності і інтерактивності міських служб, зниження витрат і споживання ресурсів, поліпшення зв'язку між міськими жителями і державою. Застосування технології "розумного міста" розвивається з метою поліпшення управління міськими потоками і швидкої реакції на складні завдання. Тому "розумне місто" більш підготовлене до вирішення проблем, ніж при простих «операційних» стосунках із своїми громадянами.

Однією з характеристик розгортання смарт-міста може бути кількість використаних давачів. Наприклад, інсталяція інтелектуальних камер на кожному куті вулиці в Нью-Йорку вимагатиме понад 3000 камер. В інших випадках місто, наприклад, Барселона, розгортає близько мільйона екологічних давачів для моніторингу електричного споживання, температури, умов навколишнього середовища, якості повітря, рівня шуму та парковки. Всі вони мають низьку пропускну здатність у порівнянні з потоковою відеокамерою, але загальна кількість переданих даних буде майже такою ж, як камери спостереження в Нью-

Йорку. Ці характеристики кількості та пропускної здатності необхідно враховувати при побудові правильної архітектури IoT.

Деякі з випадків використання технологій IoT у "розумному місті" є такими:

- контроль забруднення та регуляторний аналіз за допомогою екологічного зондування;
- мікрокліматичні прогнози погоди з використанням загальноміських сенсорних мереж;
- посилення ефективності та зниження витрат через послугу управління відходами за запитом;
- покращання транспортного потоку та економія палива через розумний контроль світлофорів та застосування шаблонів типових ситуацій;
- енергоефективність міського освітлення (освітлення за потребою);
- розумне прибирання снігу, яке базується на дорожній ситуації в реальному часі, погодних умовах і наявної снігоприбиральної техніки;
- розумне зволоження парків і громадських приміщень залежно від погоди і поточного використання;
- розумні камери для спостереження за злочинами та автоматизовані попередження про них у реальному часі;
- розумні автостоянки для автоматичного пошуку кращого місця паркування на вимогу водія;
- монітори стану мостів, вулиць та інших інфраструктурних об'єктів для підвищення їх довговічності.

1.5 Використання урядом та у військовій справі

Міські, обласні та державні органи керування, а також військові, зацікавлені в розгортанні IoT. Візьмемо, наприклад, розпорядження штату Каліфорнія B-30-15, в якому говориться, що до 2030 року викиди парникових газів, які впливають на глобальне потепління, повинні бути на 40% нижче рівня 1990 року. Для активного досягнення таких цілей, як екологічний моніторинг,

моніторинг електроенергії з використанням машинного інтеллекту, потрібно змінити енергетичні моделі штату згідно вимог Закону, зберігаючи при цьому нормальну роботу економіки Каліфорнії. Військове застосування включає такі проекти, як Інтернет-поле бою речей, автоматизоване виявлення супротивника, та його інфраструктури, керування вогнем та наведення розумних бомб та ракет на об'єкти супротивника.

Роль уряду в IoT також виступає у формі стандартизації, розподілу частотного радіоспектра та його регулюванні.

Нижче наведено деякі урядові та військові випадки використання IoT:

- аналіз загроз терору через аналіз даних від пристроїв IoT та маячків з використанням шаблонів;
- випускання рою давачів дронами над полем бою, чи в інших місцях;
- сенсорні бомби, скинуті на полі бою, формують сенсорні мережі для моніторингу загроз;
- системи відстеження урядових активів;
- служби персонального відстеження та визначення місцезнаходження в режимі реального часу;
- комплексні давачі для моніторингу ворожих середовищ;
- моніторинг рівня води для вимірювання забору греблі та вірогідності затоплення.

Контрольні питання

1. В якому році було підключено перший побутовий пристрій до Інтернету?
2. В якому році з'явився термін **інтернет речей**?
3. В якому році з'явився Альянс розробників інтернету речей?
4. Що обіцяє IoT? Яка кінцева мета IoT?
5. Як саме використовується Інтернет речей на виробництві?
6. Як можна використовувати Інтернет речей в енергетичній сфері?
7. Які основні застосування Інтернету речей у технологіях **розумного міста**?
8. Cisco підрахував, що IoT складатиметься з майже 30 мільярдів об'єктів до 2020 року. Інші мають більш високі оцінки. Якою була їх логіка?

9. Які три основні виклики для Інтернету речей? Чому ці виклики також розглядаються як можливості?

2 АРХІТЕКТУРА ТА ЕКОСИСТЕМА IIoT

Екосистема IIoT починається з найпростіших давачів, розташованих у найвіддаленіших кутках Землі, які переводять аналогові фізичні ефекти довкілля в цифрові сигнали (мову Інтернету). Дані потім здійснюють складну подорож через дротові та бездротові системи передачі, різні протоколи, природні перешкоди, перш ніж досягають Інтернету. Звідти, пакетні дані будуть проходити через різні канали, що надходять до хмари або великого центру обробки даних (ЦОД). Сила IIoT - це не тільки один сигнал від одного давача, але і сукупність сотень, тисяч, потенційно мільйонів давачів і пристроїв [3][8].

2.1 Екосистема Інтернета речей

Екосистема IIoT - це сукупність обладнання, мережевої інфраструктури, алгоритмів обробки та аналізу даних, безпекових алгоритмів, які пристосовані до спільної роботи в певному середовищі автоматизації OT, утворюючі з ним єдине ціле (рис. 2.1).

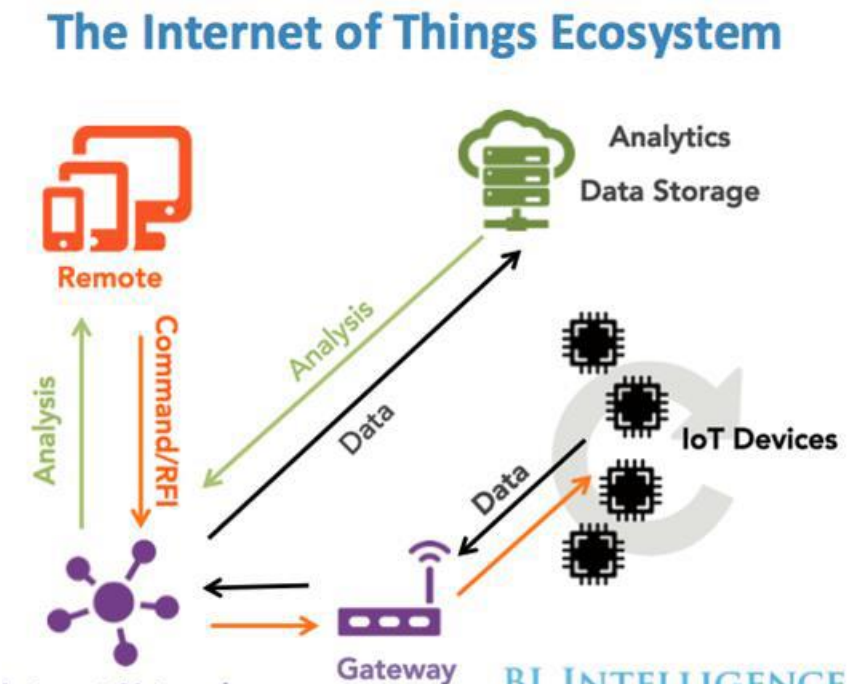


Рисунок 2.1 Вигляд екосистеми Інтернету речей

Нижченаведені сегменти інформаційних технологій утворюють собою екосистему Інтернета речей:

давачі: вбудовані системи, операційні системи реального часу, джерела безперебійного живлення, мікроелектромеханічні системи (МЕМС);

системи зв'язку між давачами: зона охоплення бездротових персональних мереж становить від 0 см до 100 м. Для обміну даними між давачами застосовуються низькошвидкісні малопотужні інформаційні канали, які часто побудовані не на протоколі IP;

локальні обчислювальні мережі: зазвичай це системи обміну данними на основі протоколу IP, наприклад, 802.11 Wi-Fi-мережа для швидкого радіозв'язку, часто це пирингові або зіркоподібні мережі;

агрегатори, маршрутизатори, шлюзи: постачальники вбудованих систем, самі бюджетні складові (процесори, динамічна оперативна пам'ять і система зберігання даних), виробники модулів, виробники пасивних компонентів, виробники тонких кліентов, виробники стільникових і бездротових радіосистем, постачальники кросплатформного програмного забезпечення, розробники інфраструктури туманних обчислень, інструментарій для граничної аналітики, безпека граничних пристроїв, системи управління сертифікатами;

глобальна обчислювальна мережа: оператори стільникового зв'язку, оператори супутникового зв'язку, оператори малопотужних глобальних мереж (Low-Power Wide-Area Network, LPWAN). Зазвичай застосовуються транспортні протоколи інтернету для IIoT і мережевих пристроїв (MQTT, CoAP і навіть HTTP);

хмара: інфраструктура в якості постачальника послуг, платформа в якості постачальника послуг, розробники баз даних, постачальники послуг потокової і пакетної обробки даних, інструменти для аналізу даних, програмне забезпечення в якості постачальника послуг, постачальники озер даних, оператори програмно-визначених мереж / программно-визначених периметрів, сервіси машинного навчання;

аналіз даних: величезні масиви інформації передаються в хмару. Робота з великими обсягами даних і отримання з них користі – це завдання, що вимагає комплексної обробки подій, аналітики і прийомів машинного навчання;

безпека: при зведенні всіх елементів архітектури воєдино встають питання безпеки. Безпека стосується кожного компонента: від давачів фізичних величин до ЦП і апаратного забезпечення, систем радіозв'язку і самих протоколів передачі даних. На кожному рівні необхідно забезпечити безпеку, достовірність і цілісність даних. У цьому ланцюзі не повинно бути слабких ланок, оскільки Інтернет речей стане головною мішенню для атак хакерів в світі.

Така екосистема буде залучати до себе фахівців з різних технічних областей, наприклад, фізиків, що займаються розробкою приладів, проектують

нові види давачів і багаторічні батарейки, інженерів-програмістів вбудованих систем, що працюють над проводовими давачами граничних пристроїв. Мережевих інженерів, які вміють працювати з персональними мережами і глобальними обчислювальними мережами, а також з програмно-конфігурованими мережами. Фахівців з обробки даних, що працюють над новітніми схемами машинного навчання на граничних пристроях і в хмарах. DevOps-інженерів, які успішно реалізують масштабовані хмарні рішення, а також туманні обчислення.

2.2 IIoT проти M2M

Що відрізняє IIoT-світ від технологій міжмашинної взаємодії (M2M)? Технології міжмашинної взаємодії та Інтернету речей дуже схожі за своєю суттю, але є і суттєві відмінності:

M2M: це загальна концепція, яка передбачає, що автономний пристрій безпосередньо обмінюється даними з іншим автономним пристроєм. Під автономністю мається на увазі здатність вузла збирати і передавати інформацію іншому вузлу без участі людини. Форма передачі даних може бути різною. Дуже часто буває, що в пристрої M2M не інтегрований спеціалізований функціонал для передачі інформації. Це відразу відмітає типові пристрої для виходу в інтернет, які регулярно застосовуються для роботи з хмарними сервісами і сховищами. У M2M-системі обмін даними також може здійснюватися без участі протоколу IP - наприклад, через послідовний порт або пропрієтарний протокол;

IIoT: IIoT-системи можуть містити в собі кілька M2M-вузлів (наприклад, мережа Bluetooth Mesh, в якій реалізовано обмін даними без участі протоколу IP), але дані агрегуються на граничному маршрутизаторі або шлюзі. Граничний пристрій, такий як шлюз або маршрутизатор, слугує точкою входу в інтернет. Або ж деякі давачі, що відрізняються більш високою обчислювальною потужністю, можуть виходити на мережевий рівень інтернету самостійно. Незалежно від того, де відбувається вихід в інтернет, відмінною рисою IIoT-системи є те, що тим чи іншим чином вона завжди підключена до інтернету.

Завдяки переміщенню даних з давачів, граничних і розумних пристроїв в інтернет з'являється можливість під'єднати найпростіші пристрої до сталого

світу хмарних сервісів. До того, як хмарні технології і мобільний зв'язок стали частиною повсякденної реальності і перестали бути дорогими, у найпростіших давачів і вбудованих систем був відсутній спосіб за кілька секунд відправляти дані іншим пристроям, зберігати інформацію як завгодно довго і аналізувати дані, щоб виявляти тенденції і шаблони. У міру розвитку хмарних технологій бездротові системи зв'язку стали повсюдними, нові енергетичні пристрої, такі як літій-іонні акумулятори, стали рентабельними, а моделі машинного навчання знайшли нові напрямки практичного застосування при аналізі великих обсягів даних. Це сильно зміцнило позиції інтернету речей. Якби всі ці технології не зійшлися воєдино саме в той момент, коли все це сталося, ми би і досі жили у світі M2M.

2.3 Корисність мережі і закон Меткалфа-Бекстрома

За усталеною традицією корисність мережі розраховується відповідно до закону Меткалфа. У 1980 р. Роберт Меткалф сформулював ідею про те, що корисність будь-якої мережі пропорційна квадрату чисельності користувачів системи. Коли мова йде про інтернет речей, як користувачів можна розглядати давачі або граничні пристрої. В цілому, закон Меткалфа виражається наступною формулою:

$$V \sim N^2, \tag{2.1}$$

де V - корисність мережі;

N - кількість вузлів в цій мережі

Щоб краще розібратися в цій формулі, подивимося графік на рис. 2.2, на якому також показана точка перетину, яка відзначає поріг, перейшовши через який можна очікувати окупності інвестицій (позитивне значення коефіцієнта ROI).

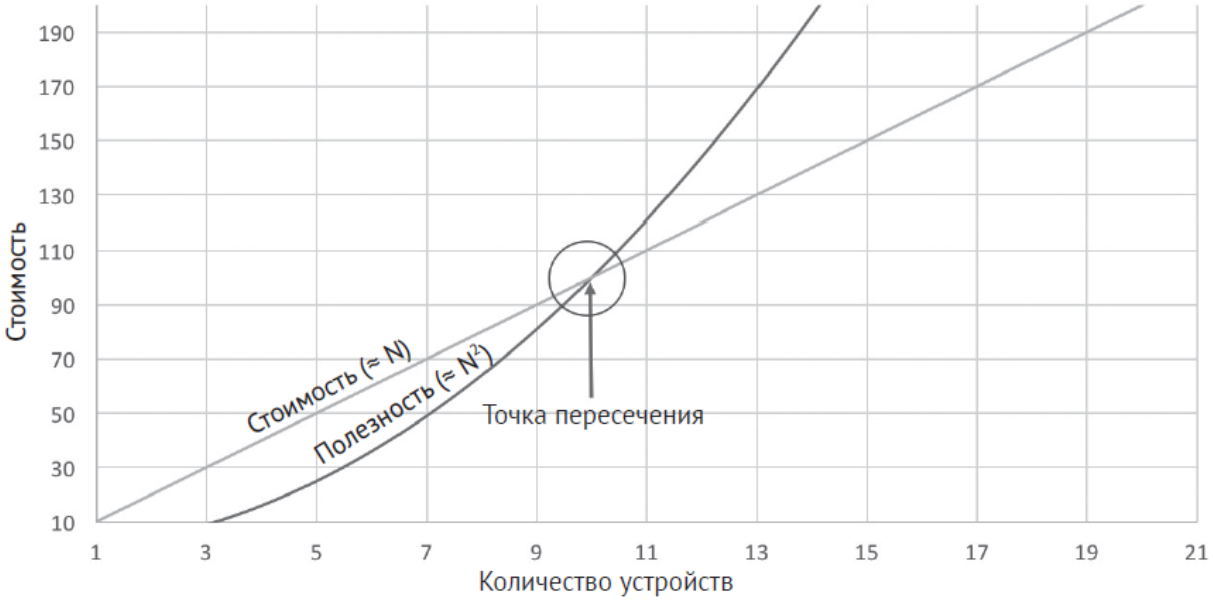


Рисунок 2.2 Графічне представлення закону Меткалфа (корисність і вартість)

Вартість кожного вузла виражена формулою $k*N$, де k - це довільна стала. В даному прикладі значення k дорівнює $10 за кожен граничний давач IIoT-мережі. Важливе значення - це точка перетину, яка з'являється в разі зростання показника корисності і позначає той момент, коли IIoT-система досягає позитивного значення коефіцієнта ROI. **ROI** (Return On Investment) - це коефіцієнт повернення інвестицій, показник рентабельності вкладень. Він в процентному співвідношенні демонструє прибутковість (при значенні більше 100%) або збитковість (при значенні менше 100%) конкретної суми вкладення грошових коштів в певний проект.

Закон Меткалфа не приймає до уваги погіршення якості зв'язку, яке може виникнути в результаті збільшення кількості користувачів або обсягу трафіку на тлі збереження початкової пропускної здатності мережі. Також закон Меткалфа не приймає до уваги відмінність в рівнях мережевого обслуговування, ненадійну інфраструктуру (наприклад, зв'язок стандарту 4G LTE під час керування автомобілем) або несприятливі фактори, що впливають на роботу мережі (наприклад, DoS-атаки).

Щоб зробити поправку на ці обставини, застосовуємо закон Бекстрома:

$$\sum_{i=1}^{n} V_{i,j} = \sum_{i=1}^{n} \sum_{k=1}^{m} \frac{B_{i,j,k} - C_{i,j,k}}{(1+r_k)^{t_k}}, \quad (2.2)$$

де $V_{i,j}$ – означає цінність мережі j для пристрою i в даний час;

i - окремий користувач або пристрій в мережі;

j – сама мережа;

k - окрема транзакція;

$B_{i,j,k}$ - перевага, яку транзакція k дає пристрою i в мережі j;

$C_{i,j,k}$ - вартість транзакції k для пристрою i в мережі j;

r_k - ставка дисконту на момент транзакції k;

t_k - витрачений час (в роках) на транзакцію k;

n – кількість користувачів;

k – кількість транзакцій.

Відповідно до закону Бекстрома, щоб розрахувати корисність мережі (наприклад, IIoT-рішення), ми повинні оцінити всі транзакції з кожного пристрою і підсумувати їх корисність. Якщо мережа j з будь-якої причини дає збій, у скільки це обійдеться користувачеві? Це той вплив, який чинить IIoT-мережа, і тут проглядається сильніша прив'язка до реальних показників корисності. Найскладніше в цьому рівнянні обчислити показник B - перевага транзакції. Якщо розбирати кожен IIoT-давач окремо, значення цього параметра може бути дуже невеликим і несуттєвим (наприклад, індикатор температури будь-якого пристрою не передавав дані протягом години). В інших випадках це значення може бути дуже суттєвим (наприклад, розряд батарейки давача протікання води, і складське приміщення магазину виявилося затопленим, що спричинило за собою великі витрати і підвищення вартості страховки).

Ось ще один приклад використання Закону Бекстрома. Припустимо, ви купуєте протягом року речі на *Amazon*, цінність яких кожен місяць становить $100. Ви могли б купити ці самі речі оффлайн приблизно за ту ж саму ціну, але ви повинні були б при цьому доплатити ще за бензин, щоб дістатися до магазину, плюс вартість витраченого вами часу. Якщо сумарна вартість покупки в магазині склала $150 в місяць,тобто на $50 більше, ніж при покупці онлайн, тоді *value* (купівельна сила) мережі Amazon для вас буде $600 в рік. Відніміть з цього вартість підключення до мережі *Amazon*, можливо, $40 в місяць, вартість підключення до Інтернету, вартість електроенергії, амортизацію комп'ютерної техніки, інші пов’язані витрати і ви отримаєте корисність близько $ 120.

При розробці IIoT-рішень перше, що повинен зробити архітектор - це зрозуміти, яку користь принесе проект. Інакше IIoT-система стає фінансовим тягарем і приносить клієнту тільки збитки.

2.4 Архітектура IIoT

IIoT-архітектура охоплює безліч технологій. Кожен архітектор повинен розуміти, який вплив вибранне проектне рішення буде мати на всю систему в цілому і кожну з її частин окремо. Складнощі і багатогранність інтернету речей пов'язані з тим, що ця технологія набагато більш комплексна, ніж традиційні технології: її відрізняє не тільки великий розмах, але і поєднання різних, часто не пов'язаних між собою типів архітектури. Кількість можливих проектних рішень вражає уяву. Наприклад, на 2017 рік в світі існувало більше 700 IoT-провайдерів, що пропонували хмарні сховища, SaaS-компоненти, системи управління IIoT, системи безпеки IIoT і будь-які види аналізу даних. Додайте сюди величезну кількість різних протоколів персональних, локальних і глобальних мереж, які постійно змінюються і коригуються в залежності від регіону.

Вибір невідповідного протоколу персональної мережі може призвести до проблем з обміном даними і помітно низької якості сигналу, і ситуацію можна виправити, тільки додавши більше вузлів до мережі. Архітектор повинен враховувати інтерференцію в локальних і глобальних мережах: яким чином дані знімаються з граничних пристроїв і передаються в інтернет? Архітектор повинен оцінити відмовостійкість системи і вартість можливої втрати даних. Який рівень повинен відповідати за відмовостійкість системи - нижні рівні або рівень протоколу?

Архітектор також повинен вибрати інтернет-протоколи: MQTT або CoAP чи AMQP, - а також необхідно продумати, як все це буде працювати в разі переходу на інший хмарний сервіс. Також потрібно вирішити, в якій точці буде здійснюватися обробка даних. На цьому етапі можна розглянути туманні обчислення як спосіб обробки даних поруч з джерелом, що вирішує проблему запізнювання і, що важливіше, дозволяє знизити завантаження мережі і витрати при передачі даних з глобальних мереж і хмарних сервісів. Невідповідний

інструмент аналітики може стати причиною захаращення системи надлишковими даними або змусить вас користуватися алгоритмами, які вимагають занадто великих обчислювальних ресурсів для роботи на граничних вузлах. А як запити, що направляються від хмари до давача, вплинуть на роботу батарейки самого давача? Крім усього цього широкого діапазону можливих варіантів, ми не повинні забувати про систему безпеки, оскільки створена нами IIoT-система стає найбільшою мішенню для атак в місті. Як ви бачите, вибір величезний і кожне рішення впливає на інші. На даний момент нам доступні більше 1,5 млн. різних комбінацій архітектурних рішень (див. приклад на рис. 2.3):

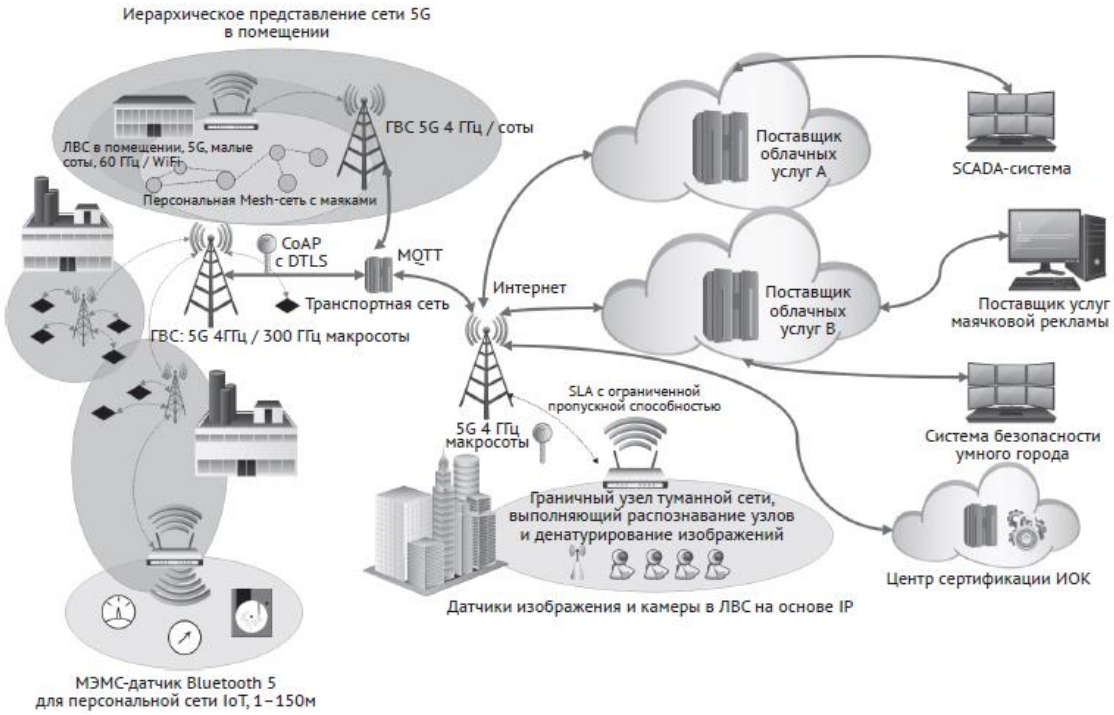


Рисунок 2.3 Варіанти IIoT-рішень. Повний спектр різних варіантів на всіх рівнях IIoT-архітектури: від давача до хмари, і навпаки

2.5 Роль архітектора IIoT системи

Термін «архітектор» часто застосовується в технічних дисциплінах. Існують архітектори ПЗ, системні архітектори і архітектори рішень. Навіть у вузьких областях, таких як інформатика і програмування, зустрічаються спеціальності, назви яких звучать як SaaS-архітектор, архітектор хмарних рішень, архітектор інтелектуальної обробки даних та ін.

Це люди, які відмінно розбираються в зазначених сферах, професіонали своєї справи. Ці вертикальні предметні області перетинають велику кількість

горизонтальних технологій. Тематика даного курсу призначена для IIoT-архітекторів. Ми станемо розглядати горизонтальний зріз, тобто розглянемо декілька з зазначених вище предметних областей і об'єднаємо їх в зручну, безпечну систему з великим потенціалом подальшого розвитку.

В цьому курсі ми поринемо в теоретичні аспекти, щоб зрозуміти, як формується IIoT-система. У якісь моменти нас буде цікавити тільки гола теорія, наприклад, теорія інформації та зв’язку. В інших випадках ми будемо говорити про речі, які тим чи іншим чином стосуються IIoT-систем або пов'язані з іншими технологіями. Вивчивши цей курс, архітектор отримає готове керівництво по різних складових інтернету речей, кожна з яких необхідна для створення ефективної системи. Неважливо, в чому ви найкраще розбираєтесь: будь то електроніка, інформатика, проектування CAP, теплоенергетика, - цей курс допоможе вам зрозуміти всю структуру в цілому, що, за визначенням, і повинен робити архітектор.

Контрольні питання

1. Що таке закон Мура? Коли це вперше спостерігали? Чому це стосується IIoT?
2. Чому технологія функціонування (OT) знаходиться під тиском для інтеграції з інформаційними технологіями (IT)?
3. Uber використовує дані гірометра смартфона для моніторингу швидкості водіїв. Що таке “гірометр”? Як це працює? Де його вперше застосували?
4. Які три найкращі приналежності хмарних обчислень? Що вони означають?
5. У табличному форматі порівняйте IaaS, PaaS та SaaS. Перелічіть приклад для кожного.
6. Які основні відмінності між віртуальними машинами та контейнерами у віртуалізації? Наведіть приклад технології контейнерів. Якому підходу ви віддаєте перевагу і чому?
7. Перелічіть дві основні функції протоколу TCP / IP - хліб і масло сучасного Інтернету.
8. Навіщо нам потрібні протоколи TCP та IP?

9. Експерти з досвіду користування часто говорять, що «найкращий інтерфейс для системи - це не інтерфейс користувача». Що означає таке твердження? Коли це зазвичай застосовується? Наведіть приклад мережевих технологій.
10. Це питання складається з чотирьох частин:
    a) Що таке мережі з комутацією каналів та мережі з комутацією пакетів? Перелічіть приклад кожного використання.
    b) Навіщо нам потрібна технологія з комутацією пакетів?
    c) У таблиці перелічіть три основні відмінності між комутацією пакетів та комутацією каналів?
    d) Який підхід є кращим для Інтернету та чому?
11. Що таке протокол, орієнтований на підключення? Що таке протокол без з’єднання? Наведіть приклад кожного.
12. Деякі компанії використовують термін IoE замість IIoT. Яка їх логіка?

3 ДАВАЧІ ТА СИСТЕМА ЖИВЛЕННЯ В IoT

На відміну від багатьох існуючих IT-пристроїв, інтернет речей здебільшого пов'язаний з фізичною дією або подією. Він видає реакцію на якийсь фактор реального світу. Іноді при цьому один-єдиний давач може згенерувати величезний обсяг даних, наприклад, акустичний давач профілактичного огляду валу турбіни. В інших випадках одного лише біта даних достатньо, щоб передати життєво важливі відомості про стан здоров'я пацієнта. Якою б не була ситуація, системи давачів еволюціонували і, відповідно до закону Мура, зменшилися до субнанометрових розмірів і стали істотно дешевше. Саме до цього апелюють ті, хто прогнозує, що до інтернету речей будуть підключені мільярди пристроїв, і саме тому ці прогнози виправдаються. Ця тема присвячена мікроелектромеханічним системам, давачам і іншим типам недорогих граничних пристроїв, що мають застосування в IIoT і їх електрофізичних властивостей. Також тут розповідається про те, які силові і енергетичні системи необхідні для живлення цих граничних пристроїв. Не можна вважати, що граничні пристрої забезпечуються енергією самі по собі. Мільярдам маленьких давачів все одно потрібен великий обсяг енергії. Ми обговоримо, як безневинні зміни в хмарному сервісі можуть кардинальним чином вплинути на всю енергетичну архітектуру системи в цілому [3][9].

3.1 Сенсорні пристрої

3.1.1 Термопари і давачі температури

Давачі температури - це найбільш поширений тип давачів. Вони застосовуються повсюдно: від інтелектуальних термостатів для холодних складів до топок котлів теплостанцій.

3.1.1.1 Термопара

Термопара - це пристрій для вимірювання температури, якому не потрібне джерело живлення, тому що воно саме генерує сигнал малої амплітуди (зазвичай, мілівольти). Термопара - це два провідника, виготовлені з двох різних матеріалів, з'єднані в точці вимірювання температури. На металевому електроді, в

залежності від його температури, виникає електричний потенціал. У різних металів рівень цього потенціалу різний. Цей ефект відомий як електрорушійний ефект Зеєбека, його суть полягає в тому, що різниця потенціалів між двома різними металами знаходиться в нелінійної залежності від їх температури.

Величина напруги залежить від властивостей вибраного металу. Вкрай важливо, щоб кінці проводів були термічно ізольовані від системи (і проводи повинні мати однакову контрольовану температуру). На малюнку нижче приведена блок-схема вимірювання температури за допомогою термопари. Для підвищення точності вимірювань різниця потенціалів, індукована термопарою, зазвичай вимірюється методом, який називається методом компенсації.

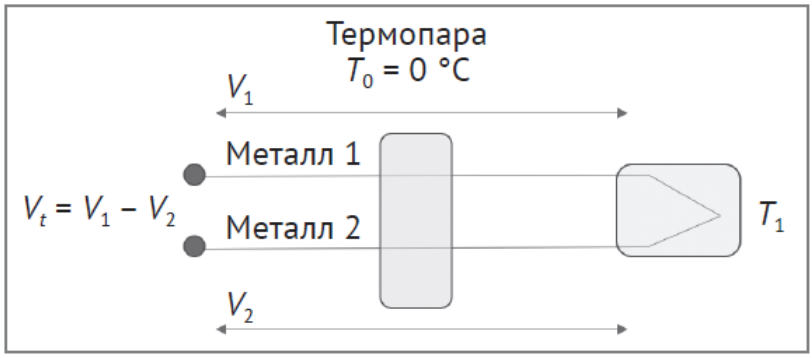


Рисунок 3.1 Блок-схема вимірювання температури за допомогою термопари

Оскільки різним температурам відповідають різні рівні напруги, а залежність між вимірюваною температурою і індукованою напругою нелінійна, для переведення виміряної напруги в температуру, як правило, використовується довідкова таблиця.

Термопари слід використовувати при виконанні не дуже відповідальних вимірювань, оскільки покази окремих термопар, при інших рівних умовах, можуть різнитися. Це викликано тим, що різні тонкі домішки, що входять до складу вимірювальних електродів, можуть призводити до невідповідностей з довідковими таблицями. Можна, звичайно, скористатися високоточними (прецизійними) термопарами, але вони і

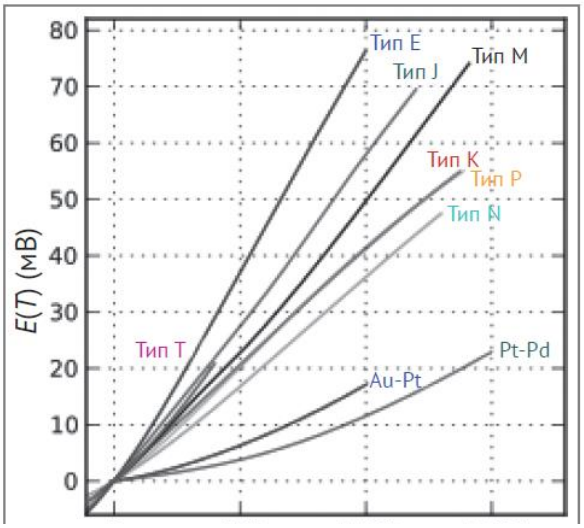


Рисунок 3.2 Залежність ЕРС термопари від її температури E(T): T

коштувати будуть, відповідно, дорожче. Іншим ефектом, що впливає на точність вимірювань, є старіння. Так як термопари часто використовуються в промислових умовах, високотемпературні середовища з плином часу можуть погіршувати точність давачів. Тому IoT-рішення повинні враховувати зміни, що відбуваються з давачами в процесі їх експлуатації.

Термопари добре працюють в широкому діапазоні температур. Для різних комбінацій металів прийнято маркування кольором і буквами, що вказують тип термопари (наприклад, E, M, PT-PD). Термопари часто використовуються для вимірювань в енергетиці, в промислових і високотемпературних середовищах при проведенні вимірів в місцях, віддалених від оператора.

На Рис. 3.1 наведена залежність EPC від температури для декількох типів термопар.

3.1.1.2 Резистивні давачі температури

Резистивні давачі температури (Resistance Temperature Detectors - RTD) працюють у вузькому діапазоні температур (нижче 600 °C), але дозволяють виконувати вимірювання з більшою точністю, ніж термопари. Зазвичай вони виготовляються з дуже тонкого платинового дроту, щільно намотаного на керамічний або скляний сердечник. Електричний опір такої конструкції пропорційний її температурі. Оскільки в основі вимірів лежить вимірювання опору, для роботи з RTD необхідне зовнішнє джерело живлення з вихідною силою струму 1 мА.

RTD виготовляються відповідно до прийнятих стандартів, в яких визначені допустимий діапазон і крок вимірів. Наприклад, для давача 200 PT100 RTD крок вимірювань становить 0,00200 Ом / °C, а діапазон вимірювань лежить в межах від 0 до 100 °C. В межах цього діапазону залежність опору RTD від його температури зберігає лінійний характер. У відповідності зі стандартами RTD випускаються в двох-, трьох- і чотирьох- провідному виконанні, чотирипровідні моделі використовуються виключно в системах високоточного калібрування. Для збільшення роздільної здатності вимірювань RTD часто використовують у мостових схемах, при цьому зазвичай покази давача лінеарізуются програмно.

RTD рідко використовуються в діапазоні вище 600 °C, що обмежує їх застосування в промисловості. При високих температурах платина може забруднюватися, що призводить до помилкових показань, однак при вимірах в межах заданого діапазону, RTD демонструють досить точні і стабільні результати.

3.1.1.3 Термістори

Термістор - це теж давач температури, електричний опір якого залежить від його температури. Цей тип давачів забезпечує більш високу точність вимірювань в порівнянні з RTD. По суті, це терморезистори, але з дуже нелінійною залежністю опору від температури. Їх часто використовують в якості згладжуючих фільтрів, для обмеження стрибків струму, а також у випадках, коли необхідна висока ступінь роздільної здатності вимірювань у вузькому діапазоні температур. Існує два типи термісторів: NTC (їх опір зменшується при підвищенні температури) і PTC (їх опір зростає з підвищенням температури). Основна відмінність від RTD полягає в тому, що термістори виготовляються з кераміки або полімерів, тоді як основою RTD завжди є метал.

Слід мати на увазі, що PTC термістори мають свого роду точку перелому і сильно змінюють опір при деякій температурі. Це робить взаємодію з PTC термісторами трохи складнішим. З цієї причини в більшості вимірювачів температури краще використовувати NTC термістори.

Термістори знаходять застосування в медичному і науковому обладнанні, в харчовій промисловості, інкубаторах і в таких побутових приладах, як термостати.

Таблиця 3.1 Зведена таблиця давачів температури

Категорія	**Термопара**	**Резістивні давачі температури**	**Термістор**
Температурний діапазон, °C	-180…+2320	-200…+500	-90…+130
Час реакції	Швидко (мікросекунди)	повільно (секунди)	повільно (секунди)

Категорія	Термопара	Резістивні давачі температури	Термістор
Розміри	Великі (~10 мм)	Невеликі (~5 мм)	Невеликі (~5 мм)
Точність	Низька	Середня	Дуже висока

3.1.2 Давачі Холла та давачі струму

Давач Холла - це смужка металу, через яку пропущений електричний струм. Потік заряджених частинок, що проходять через магнітне поле, відхиляється від прямолінійного напряму. Якщо напрямок магнітного поля перпендикулярний плоскому провіднику, то на його протилежних сторонах виникатиме різниця потенціалів, обумовлена тим, що різнойменно заряджені частинки будуть збиратися на протилежних його сторонах.

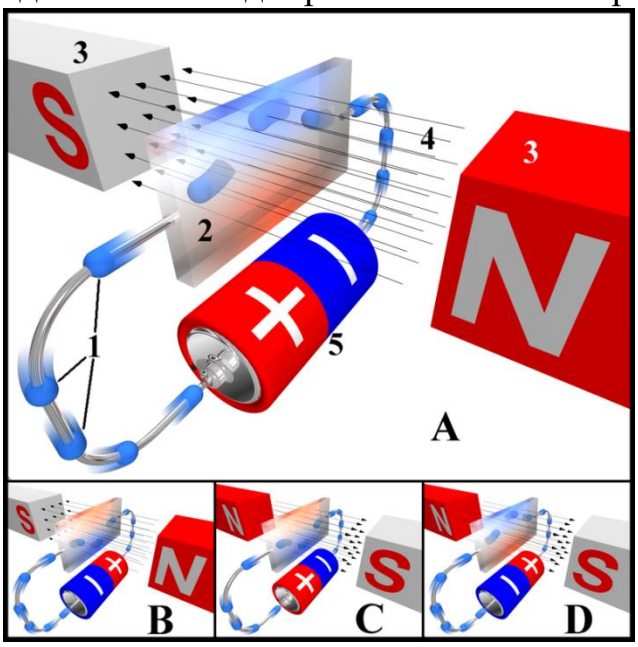


Рисунок 3.3 Ілюстрація дії ефекту Холла

Таким чином, одна сторона плоского провідника виявиться заряджена позитивно, а інша негативно, і виникне різниця потенціалів. Така різниця потенціалів називається напругою Холла, а сам ефект виникнення цієї напруги називається ефектом Холла. Це проілюстровано на **Ошибка! Источник ссылки не найден.**, наведеному поряд. Коли через м еталеву смугу, вміщену в магнітне поле, проходить струм, електрони притягуються до однієї її сторони, а дірки (позитивні заряди) – до іншої. Таке розшарування породжує електричне поле, яке можна виміряти. Якщо поле досить сильне, воно нейтралізує дію магнітного поля, і носії заряду зберігають прямолінійний рух.

Ефект Холла застосовується в давачах струму, для вимірювання змінного і постійного струму. Існує два типи таких давачів: з розімкненим і замкнутим

контуром. Давачі з замкнутим контуром дорожчі, їх часто використовують у схемах з живленням від батарей.

Типова область застосування давачів Холла: давачі положення, магнітометри, високонадійні перемикачі та покажчики рівня води. Вони використовуються також в промислових давачах для вимірювання швидкості обертання різних вузлів і механізмів. Крім того, що ці давачі недорогі у виготовленні, вони не вимогливі до умов експлуатації та можуть працювати в найсуворіших умовах.

3.1.3 Фотоелектричні давачі

Давачі для виявлення світла або визначення його інтенсивності використовуються в багатьох пристроях IoT. Такі пристрої необхідні, наприклад, в системах безпеки, інтелектуальних комутаторах або в системах управління вуличним освітленням. Існує два типи таких давачів, принцип дії яких зрозумілий з їх назви. Фоторезистор змінює опір в залежності від інтенсивності світла, а фотодіод перетворює світло в електричний струм.

Фоторезистори виготовляються з напівпровідників з високим опором. Їх опір зменшується при збільшенні інтенсивності освітлення. У темряві опір фоторезистора може мати досить високе значення (порядку мегаом). Фотони, що поглинаються напівпровідником, переводять електрони в зону провідності, тим самим збільшуючи провідність матеріалу. Фоторезистори чутливі до довжини хвилі падаючого світла, тому їх типів і модифікацій існує величезна безліч.

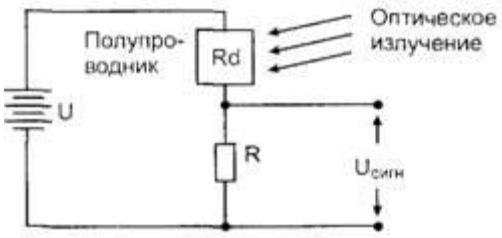


Рисунок 3.4 Типове включення фоторезистора

Фоторезистори є менш світлочутливими за фотодіоди, оскільки ті є справжніми напівпровідниковими приладами, у той час як фоторезистор є пасивним компонентом і не має p-n-переходу. Фотоопір (електричний опір) будь якого фоторезистора може змінюватися у широких межах у залежності від температури навколишнього середовища, що робить їх непридатними для застосувань, які вимагають точного вимірювання або чутливості до світла.

Для фоторезисторів також характерна деяка затримка між дією світла і наступною зміною опору, значення якої, як правило, складає близько 10 мс. Час затримки за переходу від освітлених до темних середовищ, є навіть ще більшим, і часто досягає 1 секунди. Ця властивість робить фоторезистори непридатними до вимірювання об'єктів, які швидко блимають.

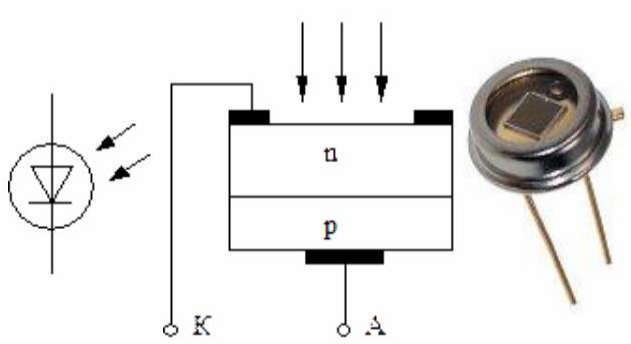


Рисунок 3.5 Умовне позначення, структура та зовнішній вигляд фотодіода

А ось фотодіоди - це повноцінні напівпровідникові прилади з p-n-переходом. Такі пристрої реагують на світло, створюючи електронно-діркову пару. Потік дірок, що рухаються до анода, і електронів, що рухаються до катода, створює електричний струм. Таким чином працюють традиційні сонячні батареї, що виробляють електрику під впливом сонячних променів. Якщо на фотодіод подати зворотню напругу, то можна регулювати його чутливість або час відгуку. Фотодіод може працювати в двох режимах:

фотогальванічний - без зовнішньої напруги. В такому режимі фотодіод використовують у якості джерела живлення пристрою (сонячна батарея);

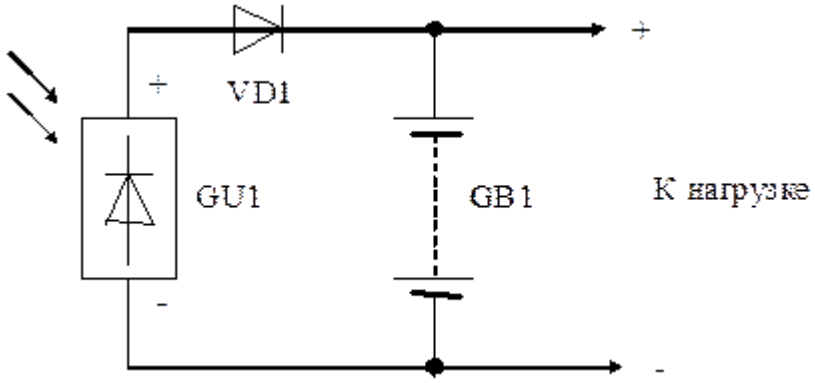


Рисунок 3.6 Фотогальванічний режим

фотодіодний - із зовнішньою зворотною напругою. Основний режим роботи, в якому зворотний струм через фотодіод буде пропорційний освітленності.

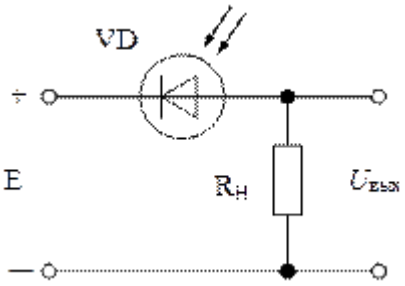


Рисунок 3.7 Фотодіодний режим

Таблиця 3.2 Загальні характеристики фотоелектричних давачів

Категорія	Фоторезистор	Фотодіод
Світлочутливість	низька	висока
Активний / пасивний	пасивний	активний
Чуттєвість до температури	висока	низька
Час реакції на зміну освітленності	тривалий (10…1000 мс)	короткий (< 1 мс)

3.1.4 MEMS давачі тиску

Промислове виробництво **мікроелектромеханічних систем** (*Microelectromechanical systems* - **MEMS**) почалося в 1980-х р.р. Але вперше вони з'явилися ще в 1960-х р.р., Коли компанією *Kulite Semiconductor* був представлений пьезорезисторний давач тиску. По суті, це мініатюрні механічні структури, які взаємодіють з електронним блоком управління. Як правило, розміри таких давачів лежать в діапазоні від 1 до 100 мкм. На відміну від інших давачів, вивчених вище, механічні структури MEMS можуть обертатися, розтягуватися, згинатися, змінювати форму, що і викликає зміни електричного сигналу. Цей сигнал, отриманий від окремого конкретного давача, фіксується і вимірюється.

Процес виготовлення MEMS-пристроїв типовий для виробництва напівпровідникових приладів. На кристал кремнію наноситься багатошарова маска, далі йдуть операції літографії, осадження і травлення. Потім матриця MEMS доповнюється іншими елементами, такими як операційні підсилювачі, аналого-цифрові перетворювачі та інші допоміжні схеми. Як правило, пристрої MEMS мають відносно великі розміри від 1 до 100 мікрон, тоді як типові кремнієві структури мають розміри 28 нм або менше. Тому виготовлення MEMS проводиться пошарово, послідовне протравлення різних шарів створює пристрій з тривимірною структурою.

Крім застосування в сенсорних системах, пристрої MEMS можна зустріти, наприклад, в друкуючих головках струменевих принтерів або в сучасних проекторах, таких як цифрові світлопропускаючі екрани (*digital light processor - DLP*). Можливість створення чутливих MEMS-пристроїв розміром зі шпилькову головку, дозволяє збільшити кількість об'єктів IoT до мільярдів.

Тензодавачі тиску знаходять застосування в широкому діапазоні IoT: від контролю за інфраструктурою розумних міст до промислового виробництва. Зазвичай вони використовуються для вимірювання тиску рідин і газів. Основою давача є п'єзоелектричний елемент, до якого організовано фізичний доступ через діафрагму (отвір) в підкладці над або під елементом. Підкладка робиться гнучкою, що дозволяє п'єзоелементам під впливом тиску згинатися, змінюючи свій електричний опір (**Ошибка! Источник ссылки не найден.**). Чутливим елементом давача є кристал монокристалічного кремнію. Кремнієві перетворювачі мають високу чутливість завдяки зміні питомого об'ємного опору напівпровідника при деформації тиском.

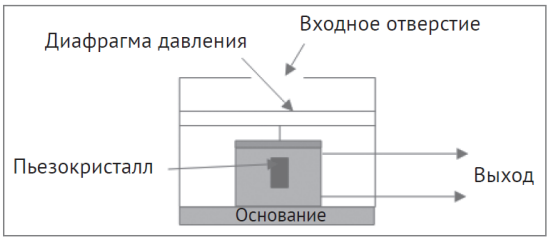


Рисунок 3.8 Будова давача тиску

Для вимірювання тиску чистих неагресивних середовищ застосовуються так звані Low cost - рішення, засновані на використанні чутливих елементів або без захисту, або з захистом силіконовим гелем.

Для вимірювання агресивних середовищ і більшості промислових застосувань використовується перетворювач тиску в герметичному метало-

скляному корпусі, з розділовою діафрагмою з нержавіючої сталі, що передає тиск вимірюваного середовища за допомогою кремнійорганічної рідини.

Струм від зовнішнього джерела живлення, що пройшов через давач, вимірюється за допомогою моста Уітстона. Такі мости можуть можуть бути виконані в двох-, чотирьох- або шестипровідних модифікаціях. Міст вимірює зміни потенціалу в ланцюзі, коли п'єзоелектрична підкладка згинається і змінює свій опір (малюнок нижче).

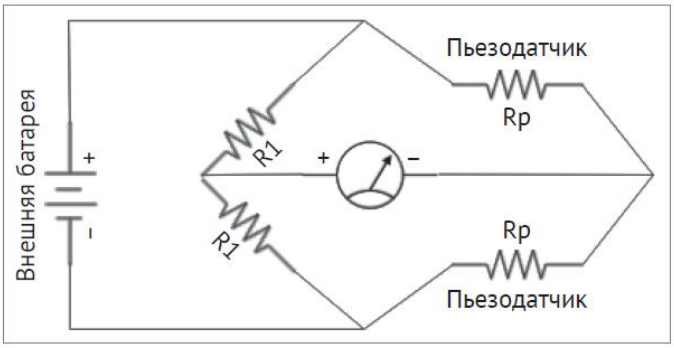


Рисунок 3.9 Міст Уітстона, який використовується для вимірювання опору

3.2 Інтелектуальні давачі

У практиці вимірювань і вимірювальних перетворень багатовимірних масивів інформації, представлених множиною електричних сигналів, поряд з основною метою вимірювання висувається ряд супутніх завдань: *режекція* (придушення) і *селекція* (виділення) по заданій ознаці одного з декількох сигналів, сортування сигналів за інформаційною ознакою, поділ множини сигналів на підкласи, адресна ідентифікація одного з каналів передачі, на який впливає сигнал із заданою інформаційною ознакою та ін. Вимірювання з цими та іншими операціями та алгоритмами обробки, що функціонують в автоматизованому або автоматичному режимах, прийнято називати **інтелектуальними**. Інтелектуальними засобами вимірювань можуть бути різні прилади – інтелектуальні давачі, автомати, автоматизовані установки, які являють собою набір засобів для реєстрації,

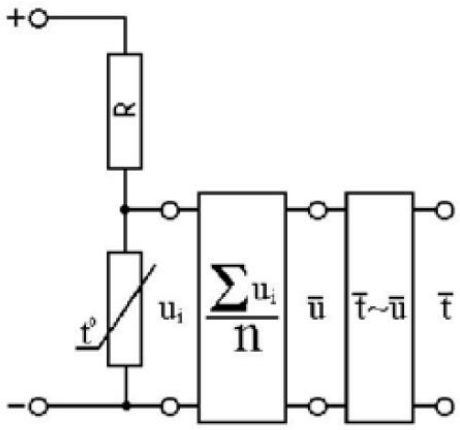


Рисунок 3.10 Приклад схеми інтелектуального давача

передачі та обробки даних, з урахуванням застосування інтелектуальних алгоритмів на основі баз знань. Найчастіше термін "інтелектуальні" вживають у вузькому розумінні по відношенню до пристроїв-давачів, які за рахунок використання вже в них самих переробки інформації (зазвичай на основі мікропроцесора) набувають нові функціональні можливості. В сучасні давачі все частіше вбудовуються мікропроцесори, що дозволяють за рахунок попередньої математичної обробки інформації безпосередньо в процесі вимірювання і активного управління вимірюванням значне підвищити точність. Найпростіша система вимірювань може складатися з давача, підключеного до системи обробки його сигналів – це може бути *цифровий процесор сигналів DSP* для обробки сигналів на апаратному рівні (рис. 3.10).

Особливості і переваги, що отримуються від використання «інтелектуальних» давачів, пов'язані із залученням обчислювальних ресурсів в сам давач. Обробка даних вже проводиться в кожному індивідуальному давачу, на відміну від обробки в центральному контролері системи, як в більшості традиційних систем. При цьому інтелектуальний давач разом з отриманням звичайної корисної інформації може бути динамічно запрограмований залежно від змін у вимогах користувача. Це зменшує необхідність в дорогих, спеціально орієнтованих на дане застосування давачах, оскільки дешеві програмовані спільноцільові давачі достатні для більшості застосувань.

Застосування цифрових методів обробки інформації дозволяє підвищити не тільки якість вимірювань, але і значно розширити функції приладів. Окрім вже відомих можливостей (настройка меж вимірювання, фільтрація сигналу, коректування похибок) з'являються і інші функції (реалізація фунцій регуляторів, завдання допустимих значень, самодіагностика, збільшення об'єму передаваної інформації по польових шинах і ін.).

3.3 Об'єднання давачів

До всіх сенсорних пристроів, які ми розглянули, може бути застосована концепція **об'єднання давачів**. Це процес об'єднання декількох різних давачів з метою отримання більшого обсягу інформації, ніж може забезпечити один давач. У просторі IoT це важливо, оскільки, наприклад, одиничний температурний

давач поняття не має про те, що саме викликає швидку зміну температури. Але в поєднанні з іншими давачами, наприклад, давачами PIR, що фіксують рух і інтенсивність освітленості, система IoT може зрозуміти, що на певному місці зібралася велика кількість людей і яскраво світить сонце, на цій підставі вона може прийняти рішення про посилення циркуляції повітря. А один термодавач просто зафіксує поточне значення температури без будь-якого усвідомлення того, що температура зростає через те, що зібралися люди і світить сонце.

На основі більшої кількості даних від більшої кількості давачів, відповідно корельованих у часі, система може приймати більш зважені рішення. Це одна з причин того, що кількість давачів, що розміщуються в хмару IoT, зростає, викликаючи зростання обсягів даних. Давачі стають дешевшими, легше інтегруються, і на прикладі давача DHT11 легко бачити, як комбінація давачів полегшує спільне бачення.

Існує два режими об'єднання давачів:

- *централізований*: дані передаються в центральний офіс, де і відбувається їх об'єднання (приклад - хмарні технології);
- *децентралізований*: кореляція даних безпосередньо в давачі (або поруч з ним).

В основі кореляції даних давача лежить центральна гранична теорема, на підставі якої два незалежних вимірювання x_1 і x_2 об'єднуються з урахуванням їх дисперсій (відхилень від норми), щоб отримати третє значення x_3. Тобто, це просто розрахунок середньозваженого значення перших двох величин:

$$x_3 = (\sigma_1^{-2} + \sigma_2^{-2})^{-1}(\sigma_1^{-2}x_1 + \sigma_2^{-2}x_2), \quad (3.1)$$

де σ_i – дисперсія i-ої величини;

x_i – i-та величина.

Іншими методами об'єднання інформації давачів є фільтри Калмана і Байєсовські мережі.

3.4 Пристрої виведення

Вихідні пристрої в екосистемі IoT можуть бути практично будь-якими: від простого світлодіода до повноцінної відеосистеми. До інших типів вихідних

пристроїв відносяться виконавчі механізми, крокові двигуни, гучномовці та аудіосистеми, промислові клапани і т.д. Зрозуміло, ці пристрої потребують різних систем управління різної складності. Залежно від типу виходу та варіанту використання, також слід очікувати, що велика частина контролю і управління повинна проводитися безпосередньо поблизу пристрою (на противагу повного контролю в хмарі). Наприклад, відеосистема може передавати дані хмарних провайдерів, але для цього потрібно обладнання виведення і буферизації.

У загальному випадку системам виведення потрібні значні обсяги електроенергії для її перетворення в механічний рух, теплову енергію або в світло. Наприклад, невеликий соленоїд для управління потоком рідини або газу при напрузі живлення 9-24 В споживатиме ток приблизно в 100 мА, при цьому створить направляюче зусилля всього в п'ять ньютонів. А промислові соленоїди працюють з напругами в сотні вольт.

3.5 Джерела живлення

3.5.1 Існуючі проблеми живлення в IoT

Керування живленням - дуже широка тема, яка стосується як програмного, так і апаратного забезпечення. При розгортанні IoT важливо розуміти роль ефективного управління енергоспоживанням віддалених пристроїв і пристроїв довготривалої експлуатації та володіти прийомами керування ім.

Проектувальник повинен будувати бюджет енергоспоживання, з огляду на наступні фактори:

- активна потужність давача;
- частота збору даних;
- споживана потужність бездротового радіозв'язку, необхідна для забезпечення покриття заданої площі;
- частість зв'язку;
- потужність мікропроцесора або мікроконтролера в залежності від тактової частоти ядра;
- потужність пасивної складової;
- втрати енергії від її витікання або неефективності живлення;

- резервування електроенергії для живлення приводів і двигунів.

Бюджет просто відображає сумарне енергоспоживання споживачів, що забезпечуються джерелом живлення (батареєю). З плином часу режим роботи батареї не є лінійним. Оскільки батарея втрачає енергоємність при розрядці, її напруга падає нелінійно. Це створює проблеми для систем бездротового зв'язку. Якщо батарея розрядиться до деякої критично мінімальної напруги, радіо або мікропроцесор, не маючи порогової напруги живлення, просто відключиться. Наприклад, комбінований давач температури та вологості DHT11 має максимальний споживаний струм (при перетворенні кожну секунду) 2,5 мА. Якщо для живлення застосовувати літієву батарею CR2032, ємність якої складає 240 мА*год, то її енергії вистачить усього лише на

240 / 2,5 * 2 * 0,8 = 153 год, або менше 6 днів.

Методи управління живленням різноманітні. Це і використання неелектронних реле часу, зниження тактової частоти процесорів або мікроконтролерів, регулювання частоти, обмеження ширини смуги радіоканалу, стратегії відстрочки сеансів зв'язку і різні режими сну. Ці методи широко використовуються, а в обчислювальній техніці вже стали загальноприйнятою практикою. Вони мінімізують енергоспоживання за допомогою динамічного управління напругою, частотного регулювання та інших схем. Але існують і нові методи, з якими слід познайомитися: це приблизний розрахунок і імовірносне проектування. Обидві ці схеми засновані на тому факті, що абсолютна точність в показаннях давача не завжди необхідна, особливо якщо сигнал підлягає обробці для передачі за допомогою бездротового зв'язку. Округлення обчислень може виконуватися на апаратному або програмному рівні простим округленням до цілого. Це тим більш прийнятно, коли мова йде про адреси або про добуток чисел (наприклад, значення 17,962 досить близько до 17,97). Імовірносне проектування передбачає лише задану ступінь надійності, а не максимально можливу, що знімає багато конструктивних обмежень. Обидва методи в сукупності здатні знизити енергоспоживання майже експоненціально в порівнянні зі звичайними схемами.

3.5.2 Відновлення живлення (підзарядка)

Відновлення живлення, концепція не нова, але в плані IoT дуже важлива. По суті, будь-яка система, що стежить за змінами умов навколишнього середовища (наприклад, від спеки до холоду, отримує радіосигнал, освітленість), може перетворювати форми енергії, що спостерігаються, в електричну. У деяких пристроях це використовується як єдине джерело енергії, а є і гібридні системи, які використовують таке перетворення для підзарядки батарей і продовження їх терміну служби. З іншого боку, вироблена таким чином енергія може зберігатися і використовуватися ощадливо для живлення низькоенергетичних пристроїв, наприклад давачів в IoT. У будь-якому випадку, системи повинні бути ефективні при перетворенні і збереженні енергії. Це вимагає розширених можливостей в управлінні живленням. Наприклад, якщо система виробництва електроенергії використовує технологію п'єзоелектричної підкладки, вбудованої в тротуар, то повинна бути передбачена компенсація на випадок, коли пішохідний рух буде недостатнім. Постійний зв'язок з перетворювачами енергії теж веде до збільшення енергоспоживання, яке також треба компенсувати. При розгортанні IoT, як правило, використовуються передові технології управління живленням, що виключають повну втрату функціональності систем. Часто використовуються такі методи, як очікування в режимі зниженого енергоспоживання, скорочення витрат, періодичні включення і виключення з

використанням реле часу. На рис. 3.11 нижче показана область енергоспоживання, яка ідеально підходить для живлення від перетворювачів, а також технології, які можуть це забезпечити. Проектувальник повинен подбати про те, щоб система не відчувала дефіциту в електроенергії, але і не мала її в надлишку. Системи перетворення і відтворення електроенергії, в цілому, мають низький потенціал і ефективність. Тому питання про підзарядку доцільно розглядати лише в умовах дійсного надлишку невикористовуваної енергії, наприклад, в умовах промислового виробництва.

3.5.2.1 Перетворення сонячної енергії

Енергія світла, природного або штучного, легко може бути використана як джерело електроенергії. Раніше ми вже обговорювали фотодіоди, коли розглядали світлочутливі давачіи. Такі діоди, але в більших кількостях, можна використовувати для створення традиційних сонячних батарей. Їх здатність генерувати енергію

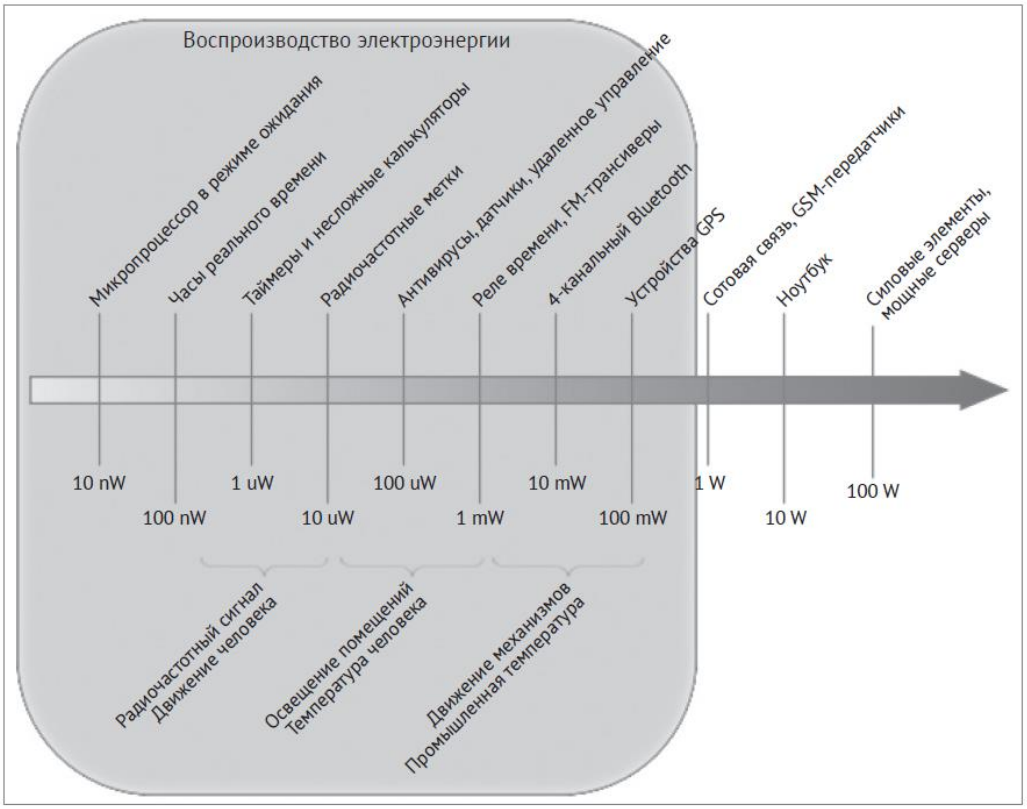


Рисунок 3.11 Область енергоспоживання, яка підходить для підзарядки відновленням

безпосередньо залежить від їх площі. Найбільш ефективні батареї, встановлені під прямими сонячними променями, а не в умовах штучного освітлення.

Класифікуються такі панелі по максимально можливій потужності на вході, що оцінюється у Вт.

Ефективність перетворення сонячної енергії залежить від сонячної активності, яка змінюється сезонно і географічно. На малюнку нижче наведена мапа сонячної інсоляції території України з розподіленням на зони, відповідно до щільності сонячної енергії:

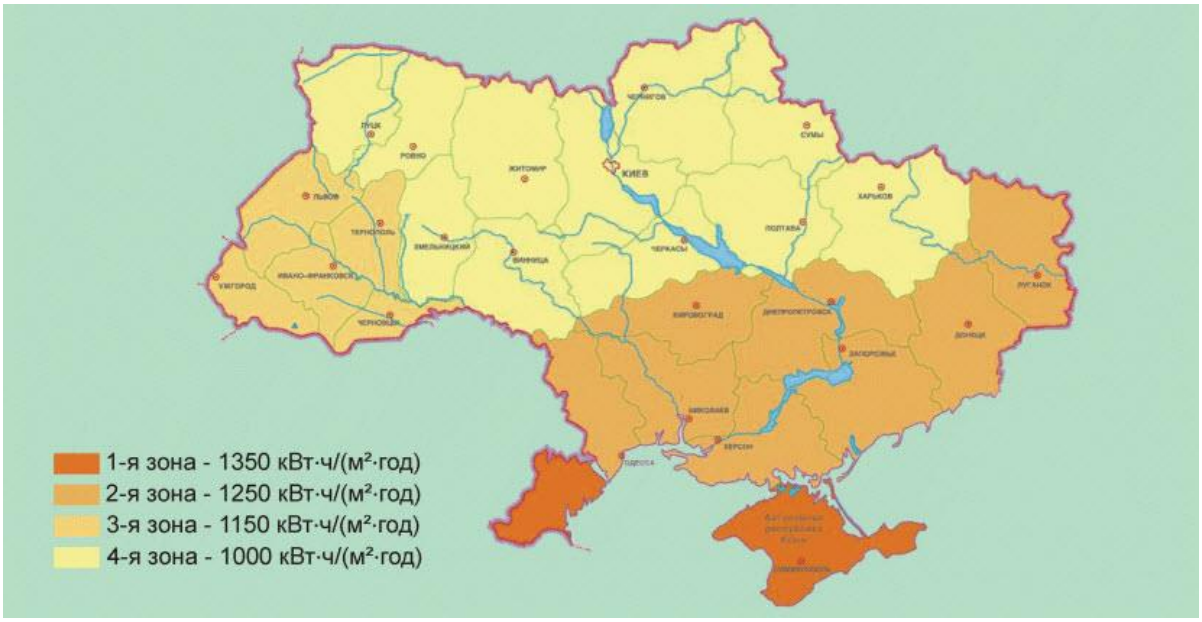


Рисунок 3.12 Мапа сонячної інсоляції території України

У південних регіонах України використання сонячної енергії буде досить ефективним, чого не можна відразу сказати про її північні регіони. Сонячні фотоелектричні елементи зазвичай неефективні. Їх ККД коливається в межах від 8% до 20%, при цьому показник в 12%, є найбільш типовим. Проте, сонячна батарея площею 25 $см^2$ при максимально можливій освітленості може генерувати потужність до 300 мВт. Іншим фактором, що впливає на генерацію потужності, є кут падіння сонячних променів. Максимальної ефективності сонячний колектор досягає при перпендикулярному розташуванні щодо сонячного світла. При зміні кута падіння, у міру руху Сонця, ефективність батареї падає. Наприклад, колектор з максимальним ККД в 12% при нахилі променів в 60° працюватиме з ККД в 9,6%.

Основа сонячного колектора - це сонячний елемент, який представляє собою простий напівпровідник з p-n-переходом і повністю аналогічний раніше описаним фотоелектричним давачам. Як пояснювалося вище, в такому елементі при опроміненні світлом між областями *p* і *n* генерується електричний потенціал.

3.5.2.2 П'єзомеханічне перетворення

Як уже згадувалося раніше, п'єзоелектричні ефекти можуть використовуватися в якості давачів, але вони ж можуть бути використані і для генерації енергії. Механічні деформації, викликані рухом, вібрацією і навіть звуком можуть бути перетворені в електроенергію. Такі генератори можуть використовуватися в розумних дорогах, навіть якщо вони вбудовані в бетон. Подібні пристрої генерують потужність порядку міліват і, отже, підходять для дуже невеликих систем, призначених для збору і зберігання інформації. Технологія може бути побудована із застосуванням п'єзомеханічних пристроїв MEMS, електростатичних і електромагнітних систем.

Принцип дії електромагнітної системи заснований на законі Фарадея, що стверджує, що зміна магнітного потоку через котушку з проводом індукує в ній електричний струм. Щоб забезпечити такі зміни, вібрація передається або на котушку, або на магніт. На жаль, така схема забезпечує занадто малу напругу.

Електростатичні системи використовують ефект, що виникає при зміні відстані між двома ємнісними пластинами, на яких підтримується постійна напруга V або заряд Q. Будь-яка вібрація пластин викликає зміну відстані між ними, що призводить до зміни ємності C еквівалентного конденсатора і відповідній зміні напруги на пластинах. Виникаюча при цьому енергія E може бути проілюстрована за допомогою наступної моделі:

$$E = \frac{1}{2} QV^2 = \frac{Q^2}{2C}, \tag{3.2}$$

Ємність може бути виражена через довжину пластини L_w, відносну діелектричну проникність ε_0 і відстань між пластинами d наступним співвідношенням:

$$C = \varepsilon_0 L_w / d, \tag{3.3}$$

Перевага електростатичного перетворення полягає в тому, що воно може бути масштабованим і економічно доцільним в умовах мікромашинобудування і виробництва напівпровідників.

Останній метод механіко-електричного перетворення - це п'єзомеханічний метод, про який вже згадувалося. Ця ж сама базова концепція, що використовується в давачах, застосовується і при відновленні електроенергії.

П'єзомеханічний пристрій MEMS, протидіючи прикладеному до нього зусиллю, генерує електричну енергію.

Ще одним аспектом перетворення механічної енергії в електричну є необхідність випрямлення і згладжування отриманої напруги. Зазвичай для цих цілей використовується пасивний випрямляч з підключеним конденсатором великої ємності в якості згладжувального фільтра. Інші форми відновлення електроенергії не потребують таких перетворень.

3.5.2.3 Перетворення енергії радіохвиль

Бездротове живлення за допомогою радіохвиль (RF) ведеться вже протягом багатьох років, тому приклад радіомаячки (*RFID tags*). RFID, як правило, знаходиться в безпосередній близькості від трансивера, який не тільки обмінюється з ним інформацією, а й постачає його енергією.

Але віддаленим пристроям потрібно живитися електроенергією з широкомовних радіотрансляцій. Передача в широкомовному діапазоні ведеться майже повсюдно: це і телебачення і сигнали стільникового зв'язку, і радіо.

Відновлення електроенергії з радіохвиль - це найбільш важке завдання в порівнянні з іншими формами генерації, оскільки радіохвилі мають найменшу щільність енергії. Прийом радіохвиль сигналів здійснюється за допомогою антени, яка налаштована на певну смугу частот. Типові діапазони частот, що використовується для радіомовлення в Україні, становлять 531…1611 кГц (діапазон середніх хвиль), 66…108 мГц (УКХ та FM мовлення). Слід зауважити, що з розвитком цифрового мовлення (так само, як це було і з телевізійним), використання для живлення широкомовних трансляцій стане проблематичним.

3.5.2.4 Перетворення тепла

В електроенергію може бути перетворена теплова енергія будь-якого пристрою, якому потрібне відведення теплоти. Зазвичай для цього використовуються два основні процеси:

- термоелектричний: пряме перетворення теплової енергії в електричну внаслідок ефекту Зеєбека;

- термічний: також відомий як термотуннелювання. Електрони викидаються (емітуються) з електрода, що нагрівається в напрямку холодного електрода.

Термоелектричний ефект (ефект Зеєбека) виникає в провідних матеріалах при наявності градієнта температур. Потік носіїв з гарячої в холодну область між двома різними електричними провідниками створює різницю потенціалів. Термопара, або термоелектричний генератор (ТЕГ), може ефективно генерувати напругу навіть внаслідок різниці температури людського тіла і температури навколишнього середовища. Різниця температур в 5 °C може генерувати потужність в 40 мвВт при напрузі 3 В. Коли тепло передається від гарячого електрода до холодного, гаряча сторона індукує потік електронів в напрямку холодного електрода. В сучасних термоелектричних пристроях, в основному, використовується телурид вісмуту, *n*- і *p*-типу. У таких пристроях один електрод елемента (теж званого термопарою) піддається впливу джерела тепла, а інший ізольований. Енергія, вироблена термопарою, пропорційна квадрату напруги і різниці температур між електродами. Математично це можна виразити таким рівнянням:

Рисунок 3.13 Термоелектричний генератор ТКГ-3

$$V = \int_{T_l}^{T_H} [S_1(T) - S_2(T)]\, dT, \qquad (3.4)$$

де $T_H - T_L$ – різниця температур;

S_1 і S_2 - коефіцієнти Зеєбека для кожного з двох матеріалів (n- і p-типу). Оскільки коефіцієнт Зеєбека - це функція температури і існує різниця температур, то результатом є різниця напруги. Ця напруга, як правило, дуже мала, тому для підвищення ефективності термоелемента послідовно з'єднується кілька термопар.

Однією з істотних проблем сучасних термопар є їх низька ефективність перетворення енергії (менше 10%). Однак і їх переваги очевидні: це і невеликі розміри, і простота виготовлення, і невисока вартість. Тривалість терміну їх служби - більше 100 000 годин. Основна ж проблема полягає в знаходженні

постійного джерела теплової дисперсії (різниці температур). Використання такого пристрою протягом декількох сезонів з різними температурами є досить складним завданням. Генерована потужність таких пристроїв IoT зазвичай лежить в діапазоні 50 мВт.

Термічна генерація заснована на ефекті викиду (емісії) електронів гарячим електродом у сторону холодного внаслідок збільшення енергії електронів до рівня, достатнього для подолання потенціального бар'єру. Потенціальний бар'єр - це фізична характеристика матеріалу. Цей метод найкраще використовувати, коли є джерело теплової енергії значної потужності. Хоча його ефективність вище ефективності термоелектричних систем, енергія, необхідна для подолання потенціального бар'єру, робить його в цілому непридатним для давачів IoT.

3.5.3 Використання сховищ енергії

Типовим сховищем енергії для давача IoT є батарея або суперконденсатор (**іоністор**) живлення. При розгляді живлення потужного давача необхідно враховувати кілька аспектів:

- енергія, необхідна для живлення силової підсистеми: чи зуміє батарея її забезпечити;
- енергоємність батареї;
- доступність: якщо пристрої замуровані в бетон, чи буде доцільна дорога заміна, якщо вони вийдуть з ладу, тим більше, що їх можливості по регенерації енергії вельми скромні;
- вага: цей параметр дуже критичний, якщо пристрій призначений, наприклад, для безпілотного літального апарату;
- як часто буде потрібно заряджати акумулятор (якщо використано його енергію);
- чи є відновлення енергії батареї, чи воно постійно доступне або епізодично, як, наприклад, сонячна енергія;
- характеристики потужності батареї: як енергія батареї буде змінюватися з часом, в процесі розряду;
- чи встановлений давач в температурно обмеженому середовищі, що може вплинути на термін служби батареї і її надійність;

- чи гарантує батарея мінімально допустимий струм.

3.5.3.1 Енергія та потужність батареї

Ємність батареї живлення вимірюється в ампер-годинах. Спрощену залежність для оцінки терміну служби джерела живлення можна записати так:

$$t = \frac{C_p}{I^n}, \tag{3.5}$$

де C_p - ємність Пейкерта;

I - струм розряду, А;

n - показник Пейкерта.

Рівняння Пейкерта, як відомо, допомагає передбачити термін роботи батареї, коли ємність батареї зменшується з різною швидкістю зі збільшенням розряду. Рівняння показує, що більш швидке розрядження забирає у акумулятора більше енергії. І, навпаки, розрядження з низькою швидкістю збільшує ефективний час роботи батареї.

Наведемо приклад цього явища. Акумулятор, розрахований виробником на ємність в 100 А*год, повністю розряджається через 20 годин (при силі струму - 5А). Якщо його розрядити швидше (скажімо, за 10 годин), його реальна ємність виявилася б нижче. Але, якщо розряджати той же акумулятор повільніше (наприклад, більше 40 годин), його реальна ємність виявилася б більше. Однак, як показано на графіку нижче, цей зв'язок є нелінійним. Показник Пейкерта зазвичай лежить між 1,1…1,3. Чим більше батарея відрізняється від ідеальної, чим швидше вона розряджається зі збільшенням струму розряду, тим вище її показник n. Приклади кривих Пейкерта наведені на рис. 3.14.

Різниця в швидкості розряду для різних типів батарей очевидна. Одним з переваг лужних батарей є те, що швидкість розряду майже лінійна на великій частині графіка, як показано внизу на рис. 3.15.

Літій-іонна батарея має ступінчасту функцію продуктивності, що ускладнює прогнозування її розряду. Проте, літій-іонні елементи забезпечують стійкий рівень напруги протягом тривалого періоду часу і дуже зручні для живлення електроніки протягом всього терміну служби (рис. 3.15).

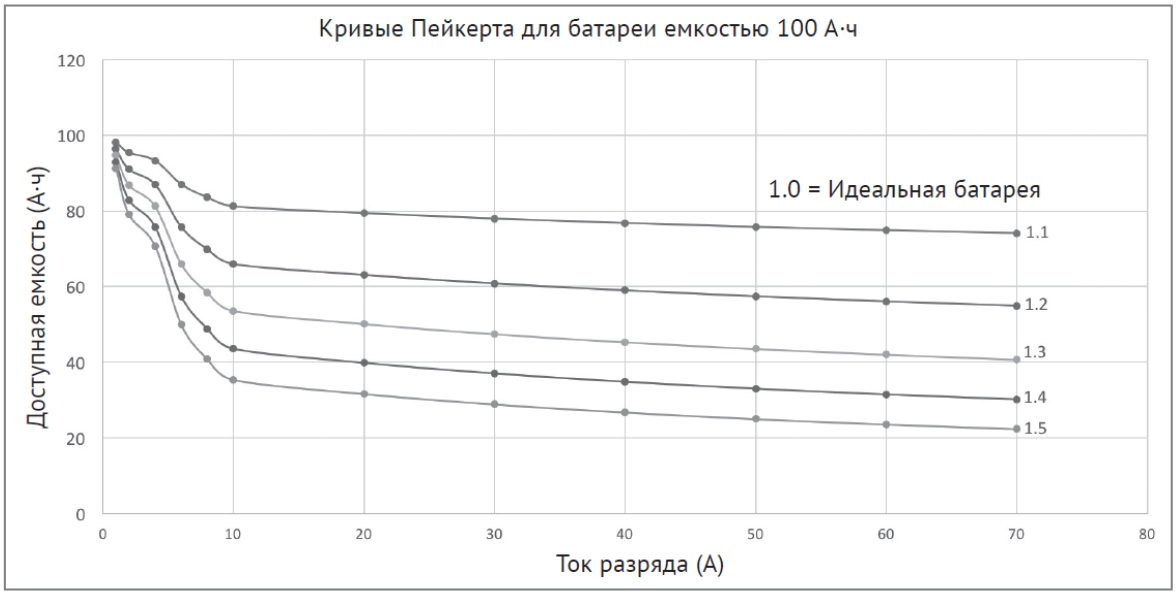


Рисунок 3.14 Криві Пейкерта, застосовані до свинцево-кислотної батареї ємністю 100 А*год

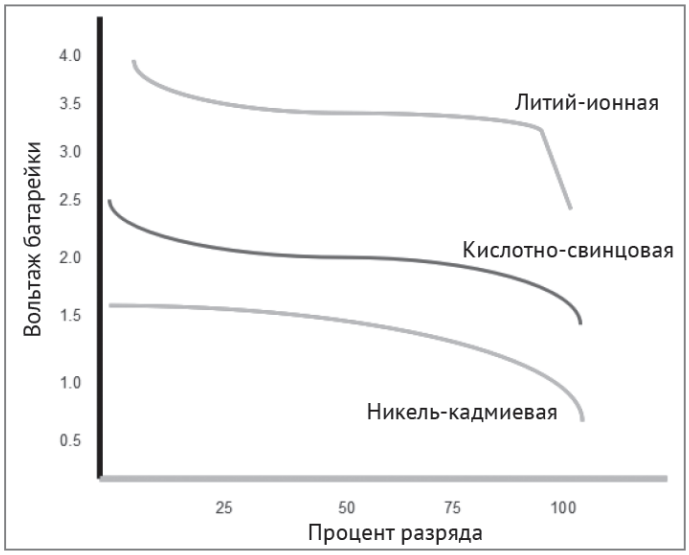


Рисунок 3.15 Приклад відносної швидкості розряду для різних типів батарей

На рис. 3.16 показана залежність між густиною потужності на кілограм ваги та густиною енергії на кілограм.

Літій-іонні (Li-ion) батареї мають більш високу густину енергії і швидкість розряду, ніж нікель-кадмієві (Ni-Cd) та нікель-гидридні (Ni-MH) батареї. Конденсатори мають дуже високу щільність потужності, але відносно малу щільність енергії. Зверніть увагу, що графік побудований на логарифмічною шкалою і відображає час розряду для різних систем зберігання.

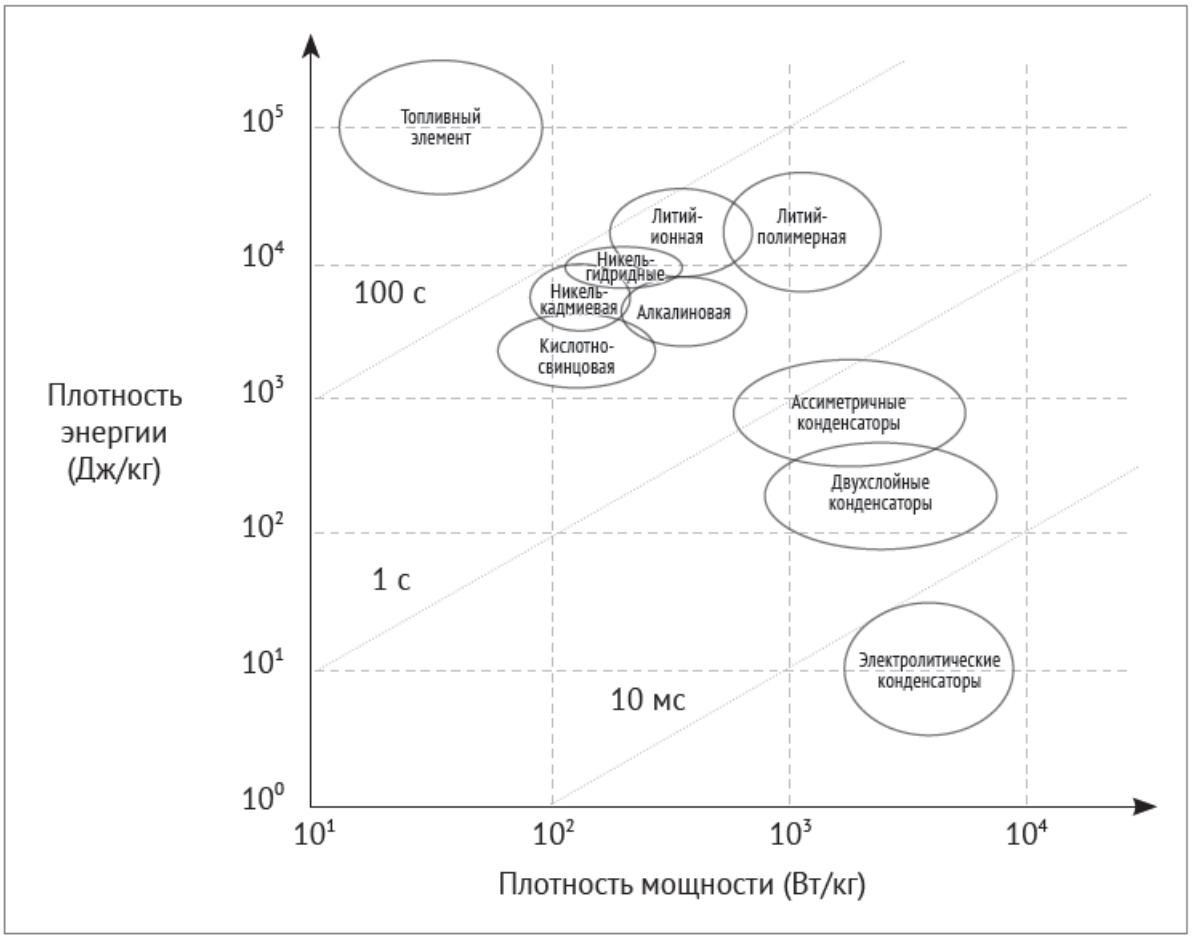


Рисунок 3.16 Графік Ругоні для конденсаторів, іоністорів, акумуляторів і паливних елементів демонструє відмінності в показниках по можливості тривалого енергопостачання і обсягами енергії, що запасається

3.5.3.2 Батареї живлення

На сьогоднішній день, літій-іонна (Li-ion) батарея, завдяки високої густини її енергії, стала стандартним елементом живлення в мобільних пристроях. У такій батареї іони літію фізично переміщаються від негативного електрода до позитивного. Під час перезарядки іони повертаються в негативну область. Це явище відоме як іонний рух.

Батареї схильні до старіння. Це виражається у втраті ємності від початкової після деякої кількості циклів перезарядки (наприклад, втрата 30% ємності після 1000 циклів). Це погіршення безпосередньо корелює з температурою навколишнього середовища, в теплих середовищах втрати будуть ще більші. Тому проектувальнику необхідно подбати про дотримання температурного режиму, якщо він використовує літій-іонну батарею.

Іншим негативним фактором автономного режиму є саморозряд. В процесі експлуатації в батареї виникають небажані хімічні реакції, які ведуть до втрати

енергії. Величина і швидкість таких втрат залежить від хімічного складу і температури. Як правило, термін служби літій-іонного елемента становить 10 років (з втратою ~ 2% в місяць за рахунок саморозряду), а лужної NiMH батареї - тільки 5 років (15-20% втрати енергії в місяць за рахунок саморозряду). В таблиці нижче наведені головні властивості батарей різних систем, які розробник IoT рішень повинен враховувати при виборі живлення давачів.

Таблиця 3.3 Порівняльні характеристики чотирьох найбільш часто використовуваних типів акумуляторних систем, із зазначенням усереднених параметрів

Параметр	Свинцово-кислотные	NiCd	NiMH	Li ion		
				Кобальт лития	Литий-марганце-вые	Литий-ферро-фосфат-ные
Удельная плотность энергии, Втч / кг	30-50	45-80	60-120	150-190	100-135	90-120
Внутреннее сопротивление[1], (mΩ)	<100 аккум. блок 12В	100-200 аккум. блок 6В	200-300 аккум. блок 6В	150-300 7,2В	25-75[2] на элемент	25-50[2] на элемент
Жизненный цикл[4] (80% разряда)	200-300	1000[3]	300-500[3]	500-1000	500-1000	1000-2000
Время быстрой зарядки	8-16ч	обычно 1ч	2-4ч	2-4ч	1ч или менее	1ч или менее
Терпимость к перезарядке	Высокая	Средняя	Низкая	Низкая. Не переносят постоянную подзарядку		
Саморазрядка/месяц (при комнатной температуре)	5%	20%[5]	30%[5]	Менее 10%[6]		
Напряжение в элементе (номинальное)	2В	1,2В[7]	1,2В[7]	3,6В[8]	3,8В[8]	3,3В
Напряжение отсечки при зарядке (В/элемент, 1С)	около 2,4 и 2,25			4,2		3,6
Напряжение отсечки при разряде (В/элемент, 1С)	1,75	1,00		2,5-3,0		2,8
Пиковый ток нагрузки (лучшие результаты)	5С[9] (0,2С)	20С (1С)	5С (0,5С)	>3С (<1С)	>30С (<10С)	>30С (<10С)
Температура зарядки	от -20°С до 50°С	от 0°С до 45°С		от 0°С до 45°С[10]		
Температура разрядки	от -20°С до 50°С	от -20°С до 65°С		от -20°С до 60°С		
Требования к обслуживанию	3-6[11] месяцев (подзарядка)	30-60 дней (разрядка)	60-90 дней (разрядка)	Не требуется		
Требования к безопасности	термически стабильны	Термически стабильны, обычно используются термопредохранители		Обязательный защитный контур[12]		
Используются с	конца 1800х	1950	1990	1991	1996	1999

3.5.3.3 Іоністори

Рисунок 3.17 Зовнішній вигляд різних іоністорів

Суперконденсатори (або іоністори) можуть зберігати енергію в значно більших обсягах, ніж звичайні конденсатори. Звичайний конденсатор має густину енергії близько 0,01 Вт*год / кг. Іоністор має густину енергії від 1 до 10 Вт*год / кг, за цим параметром він наближається до

батарей, щільність енергії яких може досягати 200 Вт*год / кг. Як і конденсатор, іоністор зберігає енергію між електростатично заряджених пластин але, на відміну від акумулятора, не використовує хімічне перенесення енергії. Зазвичай іоністори виготовляються з досить екзотичних матеріалів, таких, наприклад, як графен, що, природно, впливає на їх вартість. Очевидна перевага іоністорів - їх швидкість зарядки. До повного потенціалу вони заряджаються за секунди, в той час як, щоб зарядити літій-іонні батареї на 80%, буде потрібно мінімум кілька десятків хвилин. Крім того, іоністор неможливо перезарядити, а ось надмірне зарядження літій-іонної батареї може привести до серйозних проблем безпеки. Іоністори бувають двох видів:

- **електричні двошарові конденсатори** (*Electric double-layer capacitors - EDLC*): його обкладки виготовляються з активованого вугілля, а енергія зберігається у вигляді електростатичного потенціалу між обкладками;
- **псевдоконденсатори** (*Psuedocapacitors*): в якості діелектрика між його обкладинками використовується оксид металу, з якого виготовлені обкладинки, для збереження енергії використовується електрохімічне перенесення заряду.

Ще одна перевага іоністорів перед батареями - можливість точного прогнозування часу їх служби. Остаточний обсяг енергії іоністора легко розрахувати по напрузі на його клемах, вона змінюється з плином часу. На відміну від літій-іонної батареї, у якій профіль напруги зберігається постійним, що ускладнює оцінку решти обсягів енергії. Оскільки профіль напруги іоністора змінюється з плином часом, для компенсації виникаючих змін необхідний DC-DC перетворювач.

Основними проблемами іоністора, як і будь-якого іншого конденсатора, є витік струму і вартість. Як показано в Таблиця 3.4 нижче, іоністори знаходять досить широке застосування. Їх часто застосовують в гібридних рішеннях, в комплексі зі звичайними батареями для забезпечення миттєвої потужності (наприклад, при прискоренні електромобіля), в той час як акумулятор забезпечує транспортну потужність.

Таблиця 3.4 Порівняння різних системних факторів, які повинні враховуватися при виборі батарейного джерела живлення

Категория	Литий-ионная батарея	Ионистор
Плотность энергии	200 кВт·ч/кг	8–10 кВт·ч/кг
Количество циклов перезарядки	Падение емкости после 100–1000 циклов	Не ограничено
Время разряда	1–10 часов	От миллисекунд до секунд
Температура эксплуатации	От –20 до +60 °С	От –40 до +85 °С
Рабочее напряжение	От 1,2 до 4,2 В	От 1 до 3 В
Изменение напряжения	Постоянное в течение срока службы	Линейное или экспоненциальное снижение
Скорость заряда	(очень медленно) 40 С/х	(Очень быстро) 5 С/х
Срок службы	От 0,5 до 5 лет	От 5 до 20 лет
Размеры	Очень малые	Большие
Стоимость	Невысокая ($250–1000)	Высокая ($10 000)

3.5.3.4 Радіоактивні джерела живлення

Рисунок 3.18 Радіоізотопне джерело живлення космічного апарату «New Horizons»

Радіоактивне джерело живлення характеризується високою густиною енергії (105 кДж/см^3), він генерує теплову енергію внаслідок високої кінетичної енергії випромінюваних частинок. Період напіврозпаду (період часу, після закінчення якого потужність джерела падає вдвічі) такого джерела, як, наприклад, цезій-137, становить 30 років, а густина потужності 0,015 Вт/г. Такі джерела можуть генерувати кіловати потужності, але він непридатний для низькорівневих потужностей при розгортанні IoT. Зате ці технології успішно застосовуються в космічних апаратах вже протягом десятиліть. Дуже перспективно виглядають розробки з використанням п'єзоелектричних MEMS, які при захопленні електронів (джерелом електронів служить радіоактивне джерело) надають рух мікроарматурі, тим самим вироблячи механічну енергію, яку вже далі можна перетворити в електричну. Вторинним ефектом радіоактивного розпаду є відносно слабкий профіль густини потужності в силу тривалості періоду напіврозпаду. З іншого боку, вони, подібно до іоністорів, здатні, при необхідності, забезпечити миттєву потужність. Останньою проблемою радіоактивних джерел є їх значна питома вага, в більшій

мірі обумовлена масою свинцю, необхідного для захисту від радіації. Наприклад, для цезію-137 потрібен свинцевий екран товщиною 80 мм/Вт, це не може не відбиватися на їх вартості.

3.5.4 Розрахунок часу життя батареї живлення (строку роботи пристрою)

Цінною вправою для архітектора IoT є вміння оцінити строк життя батарейного пристрою з точки зору його потужності та інтенсивності роботи. По суті, пристрій IoT має можливість відправити лише обмежену кількість пакетів даних (байтів), перш ніж напруга батареї досягне межі, де пристрій відключиться, якщо не буде відновлено або поповнено живлення.

Зробимо такий розрахунок для маячка iBeacon, що описаний в п. 5.2.8. iBeacon публікує оголошення кожні 500мс, а довжина пакета даних становить - 31 байт (може бути і більше). Крім того, пристрій використовує батарейку CR2032 з номінальною потужністю 220мА × 3,7В. Електроніка радіомаяка споживає струм 49мкА при 3В. Тепер ми можемо спрогнозувати тривалість роботи батарейки маяка і ефективність передачі:

- споживання енергії маяком = 49мкА × 3В = 0,147 мВт;
- байт в секунду = 31 × (1с / 500мс) × 3 канали = 186 байт/с;
- біт в секунду = 186 байт/с × 8 = 1488 біт/с;
- використана енергія на передачу одного біту = 0,147мВт / (1488 біт/с) = 0,099 мкДж/біт;
- використана енергія для передачі одного повного оголошення = 0,099 мкДж/біт × 31 байт × 8 біт/байт = 24,50 мкДж / повідомлення;
- початкова енергія нової батареї становить: 220 мА*г × 3,7 В × 3600 с = 2930 Дж;
- строк життя батареї = (2930 Дж × (1 000 000 мкДж / Дж)) / ((24,30 мкДж / оголошення) × (1 оголошення / 0,5 с)) × 0,7 = 42 201 646 з = 488 днів = 1,3 року.

Константа 0,7 використовується для зниження часу автономної роботи. 1,3 року є теоретичною межею і, швидше за все, не буде досягнена в польових умовах через такі фактори, як струм витоку, а також спрацьовування інших функцій пристрою, які можуть знадобитися для періодичного запуску.

Контрольні питання

1. Чому в мережах IoT потрібні виконавчі механізми?
2. Яке визначення давача в Інтернеті речей? Чому в більшості давачів потрібні аналого-цифрові перетворювачі?
3. Чому давачі необхідні для перетворення фізичних сигналів в електричні?
4. У таблиці перелічіть і порівняйте різні типи виконавчих механізмів. Який тип виконавчого механізму вважається екологічно чистим і чому?
5. Які ключові відмінності давачів від виконавчих механізмів?
6. Зв'язування об'єктів керування не є самоціллю, збір даних з давачів також не є самоціллю. Яка ж тоді бізнес-мета для зв'язку речей та збору даних? Як досягти такої мети?
7. Що таке автономний давач? Коли він повідомляє сусідні системи або шлюз IoT? У чому різниця між **автономним** та **керованим** давачами?
8. У таблиці перелічіть і порівняйте десять характеристик хороших давачів. Яка характеристика, на вашу думку, є найважливішою і чому?
9. Що таке визначення чутливості та динамічного діапазону? Які типові одиниці чутливості та динамічного діапазону?
10. Що таке гістерезис? Що є типовою одиницею гістерезису?
11. Як працюють сенсорні екрани за наявності сенсорних давачів?
12. У таблиці перелічіть п'ять прикладів галузей, де використовуються давачі тиску. У кожному випадку перелічіть принаймні одну основну заявку.
13. Деякі люди висловлювали занепокоєння з приводу можливого вторгнення у власну приватність у рішеннях з підтримкою RFID (наприклад, відстежувати місцезнаходження людини, яка перевірила книгу у бібліотеці з підтримкою RFID). Це головне занепокоєння? Як би ви вирішили це?
14. Опишіть, як RFID працює для управління пральнею. Перелічіть три переваги.
15. Наведіть приклад того, як RFID працює для інтерактивного маркетингу.

16. Як RFID відстежує місцезнаходження активів або працівників у реальному часі? Яку ще технологію можна використовувати для відстеження місцезнаходження працівника в режимі реального часу?
17. Як роздрібні продавці використовують дані точки доступу Wi-Fi разом із відстеженням відео для покращення продажів та взаємодії з клієнтами?
18. В IoT використовуються три різні способи отримання інформації від **розумних речей**: давачі, RFID та відстеження відео. У таблиці порівняйте ці три технології вирішення:
    a) Переваги
    b) Недоліки
    c) Основні вимоги до речей
    d) Дві програми
19. Що таке перетворювачі? Як вони пов’язані з давачами та виконавчими механізмами?
20. Давачі швидкості вітру зазвичай включають обертовий елемент, який приводиться в рух вітром. Ці давачі повідомляють про частоту обертання цього рухомого елемента. Додаток, який отримує показання частоти, повинен застосувати функцію перерахунку, щоб перевести частоту в реальну швидкість вітру. На станції моніторингу погоди в Міжнародному аеропорту Ванкувера встановлені два давачі швидкості вітру: давач вітру RM Young 05103 та давач вітру Vaisala WM30.

    Перший має наступну функцію перерахунку:

    Швидкість вітру (м/с) = 0,0980 * Частота (Гц).

    Другий має таку функцію перерахунку:

    Швидкість вітру (м/с) = 0,699 * Частота (Гц) - 0,24.

    (а) Якщо давач RM Young повідомляє про частоту 20 Гц і припускаючи, що обидва давачі вимірюють однакове значення швидкості вітру, то якою буде частота, яку повідомляє давач Vasiala?

    (б) Якою буде фактично виміряна швидкість вітру?

4 ОСНОВИ ТЕОРІЇ ЗВ'ЯЗКУ ТА ІНФОРМАЦІЇ ДЛЯ IoT

Ми починаємо обговорення глобальних мереж (*Wide Area Network – WAN*) з огляду радіосигналів і факторів, що впливають на якість зв'язку, на обмеження, перешкоди для зв'язку, пропускну здатність каналу зв'язку і використовуваний діапазон. У різних частотних діапазонах є багато протоколів обміну WAN, і архітектор повинен розуміти плюси і мінуси вибору одного діапазону в порівнянні з іншим [3][8].

Рис. 4.1 допомагає визначитися з використанням різних діапазонів і швидкості передачі даних для бездротових протоколів, які ми розглянемо далі. **Безпровідні персональні мережі** (*Wareless Personal Area Network – WPAN*) часто використовується з іншими акронімами для ближнього зв'язку, такими як *Field Area Network (FAN)*, *Wireless Local Area Network (WLAN)*, бездротова *Home Area Network (HAN)*, мережа бездротового сусідства *wireless Neighborhood Area Network (NAN)* і бездротова мережа *Wireless Body Area Network (WBAN)* .

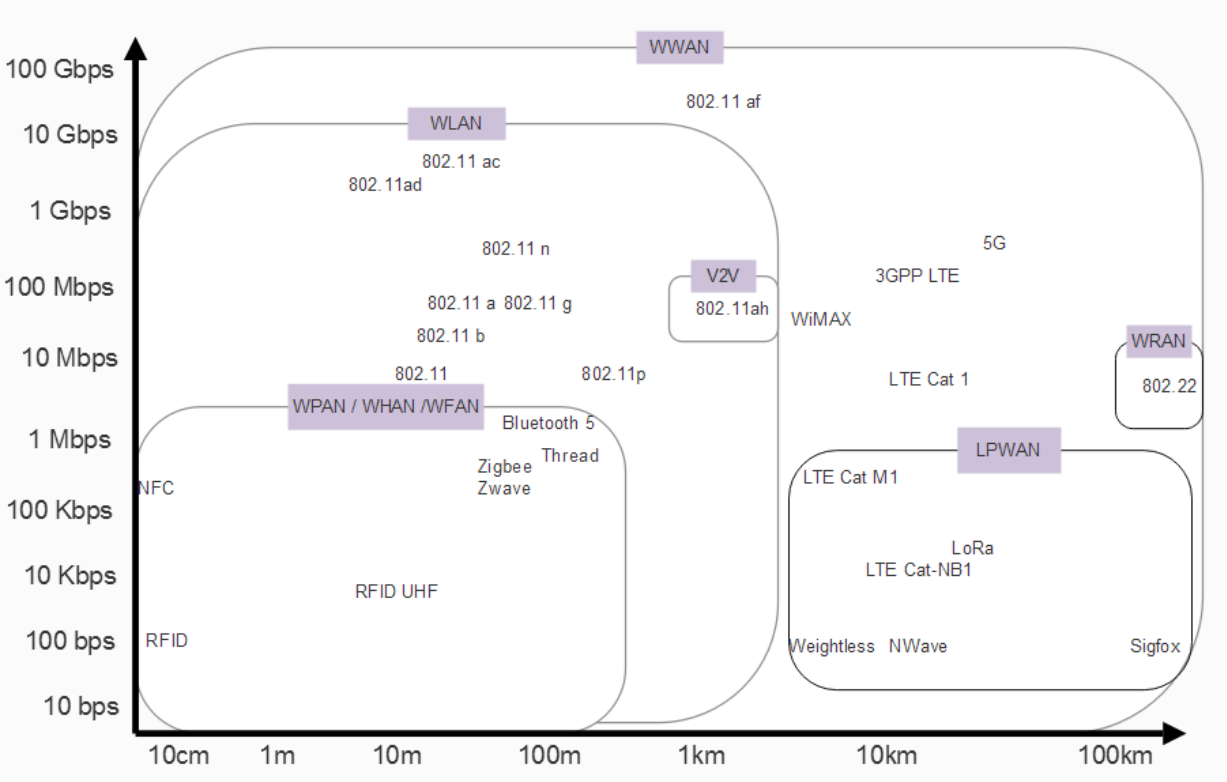


Рисунок 4.1 Різні протоколи і категорії бездротового зв'язку, призначені для різних діапазонів, швидкості передачі даних і варіантів використання (потужність, транспорт і т.і.)

4.1 Деякі питання теорії зв'язку

Інтернет речей є конгломератом багатьох окремих пристроїв, які автономно передають та/або приймають дані вже на верхньому шарі мереж і протоколів. Важливо розуміти обмеження в побудові систем зв'язку для IoT або для будь-якого іншого типу мереж. Інтернет речей об'єднує персональні мережі, локальні мережі та глобальні мережі в єдину мережу каналів зв'язку. Велика частина того, що робить IoT можливим, будується навколо комунікаційної основи; тому тут ми розглянемо основи роботи мережевих і комунікаційних систем. Ми зосередимося на системах зв'язку та сигналізації, вивчимо діапазон, енергію і обмеження систем зв'язку і те, як архітектор буде використовувати ці інструменти для розробки успішного рішення IoT.

4.1.1 Зв'язок між частотним діапазоном та поширенням радіохвиль

Конкуруючі протоколи розрізняються один від одного використовуваними діапазонами, швидкістю передачі і потужністю. Архітектори повинні враховувати різні протоколи і варіанти дизайну при реалізації повного рішення.

Найкращі умови для радіопередачі - це ділянка безперешкодної прямої видимості, на якій відсутні додаткові радіосигнали. У більшості ситуацій ця ідеальна модель не буде існувати. У реальному світі є перешкоди, відбиття сигналів, наявність інших радіочастотних сигналів і шум.

При розгляді конкретної мережі і сигналів різних діапазонів частот, наприклад, 900 МГц у порівнянні з 2,4 ГГц, за допомогою рівняння Фріїса можна отримати величину ослаблення сигналу залежно від частоти:

$$P_r = P_t G_{Tx} G_{Rx} \frac{\lambda^2}{(4\pi R)^2}, \quad (4.1)$$

де P_r - потужність приймача, Вт;

P_t - потужність передавача, Вт;

G_{Tx} - коефіцієнт посилення передавача;

G_{Rx} - коефіцієнт посилення приймача;

λ - довжина хвилі, м;

R - відстань між передавачем і приймачем, м.

В більш звичній **децибельній формі** відношення (4.1) буде виглядати наступним чином:

$$P_r = P_t + G_{Tx} + G_{Rx} + 20\log_{10}\frac{\lambda}{4\pi R}, \quad (4.2)$$

Наприклад, сигнал з частотою 900 МГц на 10 м буде мати втрати 51,5 дБ, а сигнал 2,4 ГГц на 10 м буде мати втрати 60,0 дБ.

Можна показати, як потужність і діапазон впливають на якість сигналу, використовуючи відношення, що називається енергетичним балансом каналу (ЕБК). Це порівняння потужності передачі з рівнем чутливості приймача, і воно вимірюється за логарифмічною шкалою (дБ). Можна просто забажати підвищити рівень потужності передавача для задоволення вимог енергетичного балансу до діапазону, але в багатьох випадках це порушує нормативні вимоги або впливає на термін служби батареї. Кращий варіант полягає в поліпшенні рівня чутливості приймача. ЕБК визначається співвідношенням потужності передавача і рівня сигналу на вході приймача, як показано нижче:

ЕБК = Потужність передачі (T_x) / Рівень сигналу на вході (S_x) *(4.3)*

Використовуючі логарифмічні одиниці (дБ) рівень сигналу в приймачі можна визначити наступним чином:

S_x (дБ) = T_x – ЕБК = T_x (дБ) + Підсилення (дБ) – Втрати (дБ) *(4.4)*

Припускаючи, що немає ніякого фактора, що сприяє посиленню сигналу (наприклад, посилення антени), є тільки два способи поліпшення прийому: збільшення потужності передачі або зменшення втрат. На практиці, замість рівняння (4.2), зручніше використовувати похідну від нього оцінку втрат сигналу за допомогою так званого фактора **Free-Space Path Loss (FSPL).** Це величина втрати сигналу електромагнітної хвилі по прямій видимості у вільному просторі (без перешкод). FSPL залежить від частоти (f) сигналу, відстані (R) між передавачем і приймачем і швидкості світла (c). У термінах обчислення FSPL в децибелах рівняння буде виглядати таким чином:

$$FSPL\ (\text{дБ}) = 10\,log_{10}\left(\frac{4\pi Rf}{c}\right)^2 = 20\,log_{10}\left(\frac{4\pi Rf}{c}\right) =$$

$$= 20\,log_{10}\,R + 20\,log_{10}\,f - 147{,}55 \quad (4.5)$$

Формула (4.5) є першим наближенням розрахунку втрат сигналу на шляху до приймача. Краще наближення повинно ще враховувати відбиття і хвильові перешкоди від земної поверхні, що називаються втратами "на плоскій землі". Типи перешкод, які відносяться сюди, включають:

- відбиття сигналу: коли електромагнітна хвиля, що розповсюджується, натикається на об'єкт, відбивається від нього у різних напрямках і призводить до багатопроменевого розповсюдження;
- дифракція: коли на шляху розповсюдження радіохвилі між передавачем і приймачем зустрічаються об'єкти з негладкими краями (будинки, дерева та інші);
- розсіювання: коли у середовищі, через яке проходить хвиля, присутні об'єкти, чий розмір менший за довжину хвилі і кількість таких об'єктів є значною.

Вираз для розрахунку втрат "на плоскій землі" має наступний вигляд:

$$L_{\text{Втрати на плоскій землі}}\ (\text{дб}) \approx \frac{h_t^2 h_r^2}{R^4}, \qquad (4.6)$$

де h_t - висота передавальної антени (над рівнем землі), м;

h_r - висота приймальної антени (над рівнем землі), м;

R - відстань між передавачем та приймачем, м.

Збільшення частоти, природно, збільшує втрати у вільному просторі (наприклад, сигнал 2,4 ГГц має покриття на 8,5 дБ менше, ніж сигнал 900 МГц). Взагалі кажучи, сигнали 900 МГц будуть надійними на подвоєній дистанції сигналів 2,4 ГГц. Сигнали 900 МГц мають довжину хвилі 333 мм проти 125 мм для сигналу 2,4 ГГц. Це дозволяє сигналу з частотою 900 МГц мати кращу проникаючу здатність і не так сильно залежати від розсіювання.

Розсіювання є важливою проблемою для вузлів мережі, так як багато розгортань не мають прямої видимості між антенами - замість цього сигнал повинен проникати крізь стіни, стелі і підлоги. При цьому деякі матеріали вносять свій додатковий внесок в ослаблення сигналу.

Втрати на 6 дБ рівноцінні 50% зниженню потужності сигналу, а втрати на 12 дБ дорівнюють 75% зниження. В таблиці нижче наведені типові втрати потужності радіосигналу, в залежності від матеріалу на шляху хвиль. Більше інформації по втратах наведено в роботі "БЫСТРАЯ ОЦЕНКА МОЩНОСТИ Wi-Fi-СИГНАЛА ПРИ ПРОХОЖДЕНИИ ПРЕПЯТСТВИЙ В ПРЕДЕЛАХ ЗДАНИЯ"

Таблиця 4.1 Втрати радіосигналу в залежності від матеріалу

Матеріал	**Втрати, в дБ**	
	на 900 МГц	**на 2,4 ГГц**
Скло, 6 мм	-0,8	-3
Кладка з цегли або цегляних блоків (20 см)	-13	-15
Гіпсокартон	-2	-3
Двері з цільного дерева	-2	-3
Вікно без тонування (відсутнє металізоване покриття)	-0,8	-3
Вікно з тонуванням (металізоване покриття)	-1,5…-2,5	-5…-8
Міжкімнатна стіна (15,2 см)	-14	-15…-20
Несуча стіна (30,5 см)	-20	-20…-25
Бетонна підлога / стеля	-15…-20	-15…-25
Монолітне залізобетонне перекриття	-20	-20…-25

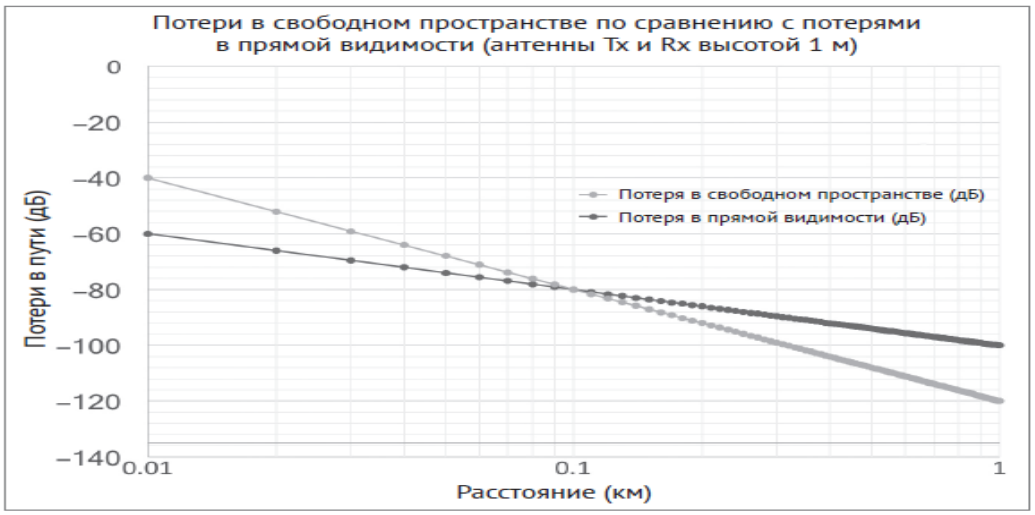


Рисунок 4.2 Втрата у вільному просторі в порівнянні з плоскою землею (в дБ) з використанням сигналу 2,4 ГГц з антенами висотою 1 метр

Багато комерційно доступних протоколів використовують у всьому світі діапазон 2,4 ГГц. Діапазон 2,4 ГГц забезпечує пропускну здатність у п'ять разів більшу в порівнянні з діапазоном 900 МГц і його пристрої можуть мати набагато меншу антену. Крім того, діапазон 2,4 ГГц неліцензовано і він доступний для використання в багатьох країнах світу. Порівняння частот 900 МГц та 2,4 ГГц показано в наступній таблиці:

Таблиця 4.2 Порівняння діапазонів 900 МГц та 2,4 ГГц

	Діапазон частот	
	900 МГц	**2,4 ГГц**
Сила сигналу	*в основному, надійний*	*переповнений діапазон, схильний до інтерференції*
Відстань	*у 2,67х далі, ніж 2,4 ГГц*	*коротша, але може компенсувати це поліпшеним кодуванням (Bluetooth 5)*
Проникнення	*велика довжина хвилі дозволяє проникати через більшість матеріалів і перешкод*	*потенційно піддається впливу більшості будівельних матеріалів*

	Діапазон частот	
	900 МГц	**2,4 ГГц**
Швидкість передачі даних	*обмежена*	*від 2-х до 3-х разів швидше, ніж 900 МГц*
Вплив на сигнал	*сигнал може залежати від високих об'єктів і перешкод, краще проходить через листя*	*менша ймовірність взаємодії каналу з певними об'єктами*
Вплив на канал	*інтерференція з бездротовими телефонами 900 МГц, сканерами RFID, сигналами стільникового зв'язку, дитячими моніторами*	*Інтерференція з 802.11 Wi-Fi*
Вартість	*середня*	*низька*

Рівняння (4.5) та (4.6) дають теоретичну модель; ніякі аналітичні розрахунки не можуть надати точного передбачення для певних реальних сценаріїв, таких, наприклад, як багатопроменеве поширення радіохвиль.

4.1.2 Інтерференція сигналу

Проблема інтерференції характерна для багатьох видів бездротових технологій, оскільки загальні діапазони часто неліцензовані. Через те, що може бути досить багато пристроїв, що випромінюють в одному і тому самому діапазоні, буде виникати нтерференція.

Візьмемо стандарти Bluetooth і 802.11 Wi-Fi; обидва працюють в загальному спектрі 2,4 ГГц, але залишаються працездатними навіть в перевантажених середовищах. Протокол Bluetooth Low Energy (BLE), буде випадковим чином вибирати один із 40 каналів шириною 2 МГц кожен при стрибкоподібній зміні несучої частоти. Ми бачимо на рис. 4.4 одинадцять безкоштовних каналів (три з яких є каналами оповіщення) на BLE, які мають 15%

вірогідність колізій (тим більше, що 802.11 НЕ перескакує між каналами). Нова специфікація Bluetooth 5 надає такі методи, як маски доступу до слотів для блокування областей Wi-Fi зі списку частотних переходів. Існують і інші методи, які ми розглянемо пізніше. Тут ми показуємо діапазон ILM для Zigbee і Bluetooth Low Energy. Також показано можливе суперництво з трьома каналами Wi-Fi в спектрі 2,4 ГГц.

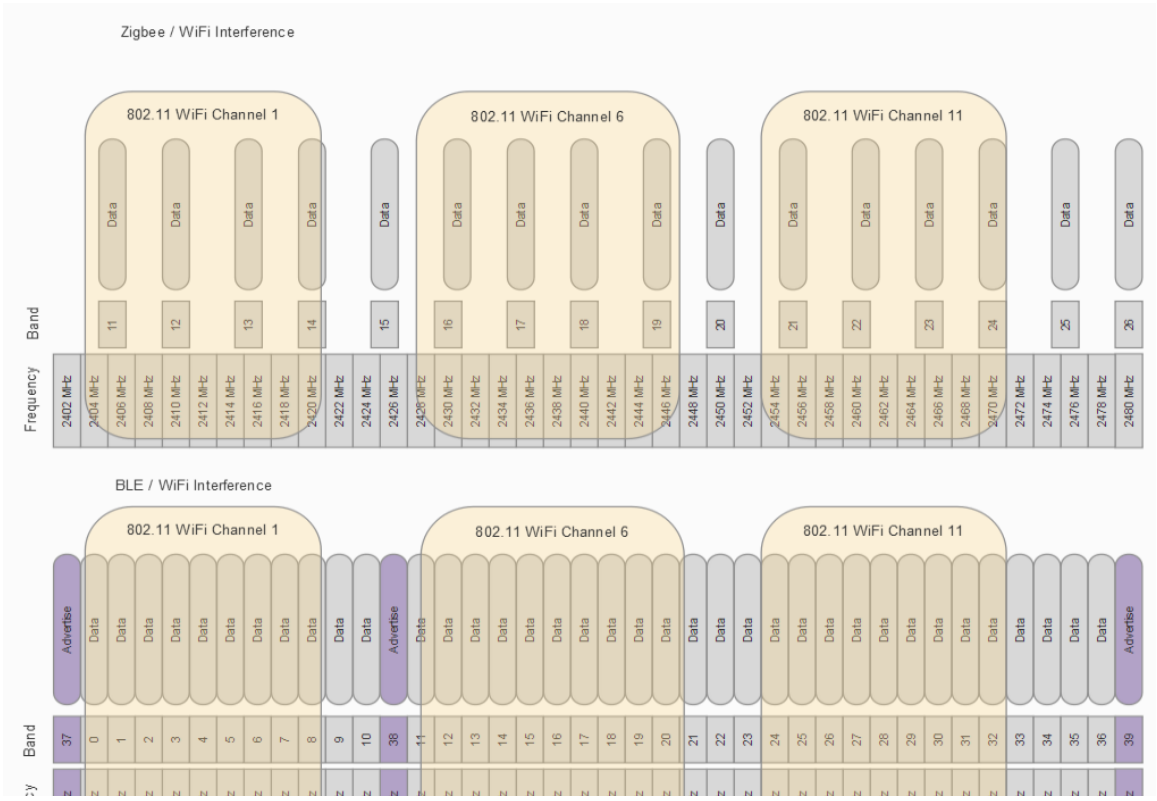


Рисунок 4.3 Порівняння Bluetooth Low Energy (BLE) і Zigbee Interference з 802.11 Wi-Fi-сигналами в діапазоні 2,4 ГГц. BLE забезпечує більшу кількість слотів і стрибкоподібну зміну частоти зв'язку в разі колізій Wi-Fi

4.2 Засади теорії інформації для IoT

Бітова швидкість, **бітрейт** (англ. *Bitrate*) - швидкість проходження бітів информації по каналу зв'язку за одну секунду. Бітову швидкість прийнято використовувати для вимірювання ефективної швидкості передачі информації по каналу зв'язку, тобто швидкості передачі «*корисної информації*» (адже крім такої є ще службова інформація, наприклад, стартові і стопові символи за асинхронної передачі, або ж контрольні символи за наявності надлишкового кодування).

Бітова швидкість визначається у бітах за секунду (*біт/с, bps*), а також похідних величинах із префіксами *кіло-, мега-* тощо. За допомогою теорії

інформації буде показано, як бітрейт впливає на потужність передачі, що, в свою чергу, впливає і на використовуваний діапазон. Як буде показано, існують обмеження цілісності даних і бітрейтів.

4.2.1 Максимальна швидкість передачі і теорема Шеннона-Гартлі

Мета передачі даних полягає в максимізації швидкості і відстані передачі у межах обмежень, які накладає використовуваний частотний діапазон і шум каналу. Теорема Шеннона-Гартлі складається з роботи Клода Шеннона з Массачусетського технологічного інституту в 1940-х рр. і Ральфа Гартлі з Bell Labs в 1920-х рр. Вона базується на фундаментальній роботі Гаррі Найквиста (також Bell Labs), який визначив у 1928 р. максимальну кількість телеграфних посилань (або біт), які могли передаватися по телеграфному каналу (20-40 роки XX століття!) за одиницю часу. По суті, Найквіст встановив мінімально потрібну ширину смуги пропускання каналу зв'язку, яка потрібна для передачі сигналу з заданою частотою дискретизації. Це називається теоремою відліків Віттакера - Найквіста - Котельнікова - Шеннона і представлено в спрощенному вигляді як:

$$f_p \leq 2B, \tag{4.7}$$

де f_p - частота дискретизації, Гц;

B - ширина смуги каналу, Гц.

Це означає, що максимальна швидкість передачі обмежена шириною смуги пропускання каналу, яка повинна бути не меншою за подвійне значення швидкості передачі.

Потім Гартлі розробив спосіб кількісної оцінки передаваємої інформації в залежності від швидкості лінії. Швидкість лінії визначається бітрейтом (наприклад, Мбіт/с). Це відомо як закон Гартлі, і він є попередником теореми Шеннона. Закон Гартлі просто стверджує, що максимальна кількість розпізнаних приймачем амплітуд імпульсів, які можуть передаватися надійно, обмежена динамічним діапазоном сигналу і точністю, з якою приймач може достовірно інтерпретувати кожен окремий сигнал. Шеннон посилив рівняння Гартлі, розглядаючи ефекти гауссовського шуму, і завершив рівняння Гартлі ввівши до нього відношення сигнал / шум. Таким чином, підсумкове співвідношення, яке

встановлює зв'язок між максимально можливою швидкістю каналу *C*, шириною його смуги пропускання і співвідношення сигнал / шум в каналі виглядає наступним чином:

$$C = B \log_2 \left(1 + \frac{S}{N}\right), \tag{4.8}$$

де *B* – ширина каналу, в Гц;

S – потужність сигналу, в Вт;

N – потужність шуму, в Вт.

Ефект цього співвідношення є невеликим, але важливим. Для кожного збільшення рівня шуму до рівня сигналу в децибелах потужність різко падає. Аналогічним чином, збільшення рівня сигнал/шум збільшує пропускну здатність каналу. Без шуму потужність була б нескінченною. Також можливо поліпшити теорему Шеннона-Гартлі, додавши в рівняння множник *n*. Тут *n* являє собою додаткові антени, тобто, канали. Такий прийом називається технологією **множинного введення, множинного виведення (MIMO)**:

$$C = Bn \log_2 \left(1 + \frac{S}{N}\right), \tag{4.9}$$

Щоб зрозуміти, як правило Шеннона відноситься до обмежень бездротових систем, згаданих нами, необхідно виразити рівняння (4.9) з точки зору енергії на біт, а не співвідношення **сигнал/шум (SNR)**. Корисним прикладом на практиці є визначення мінімального SNR, необхідного для досягнення певного бітрейта. Наприклад, якщо ми хочемо передавати зі швидкістю *C* = 200 кбіт/с по каналу з пропускною спроможністю *B* = 5000 кбіт/с, тоді мінімальна SNR задається як:

$$SNR \geq 2^{C/B} - 1 \tag{4.10}$$

Підставляючі відповідні значення, отримаємо, що *SNR = 0,028*, чи в децибельній формі *10lg(0,028) = - 15,528* дБ. Отримане значення показує, що з відповідною швидкістю по каналу можливо передавати дані, навіть коли рівень корисного сигналу нижче рівня шума!

Проте, межа швидкості передачі даних таки існує. Щоб показати це, нехай E_b представляє енергію передачі одного біта даних в джоулях. Нехай N_o

представляє спектральну щільність шуму у Вт/Гц. E_b/N_o являє собою безрозмірну одиницю (як правило, виражену в дБ), яка представляє собою значення SNR на біт, або широко відому як енергоефективність каналу. Використання енергоефективності дозволяє усунути S/N з рівняння (4.9). Припустимо, що система повністю ідеальна, причому $R_B = C$, де R - пропускна здатність. Теорема Шеннона-Гартлі тоді може бути переписана як:

$$\frac{C}{B} = \log_2\left(1 + \frac{E_b C}{N_0 B}\right),$$

$$\frac{E_b}{N_0} = \frac{2^{\frac{C}{B}}+1}{\frac{C}{B}},$$

$$\frac{E_b}{N_0} \geq \lim_{\frac{C}{B}\to 0} \frac{2^{\frac{C}{B}}+1}{\frac{C}{B}} = \ln 2 = 1{,}59 \text{ дБ} \qquad (4.11)$$

Нерівність (4.11) відома як **границя Шеннона** для аддитивного білого гауссовського шуму **(AWGN)**.

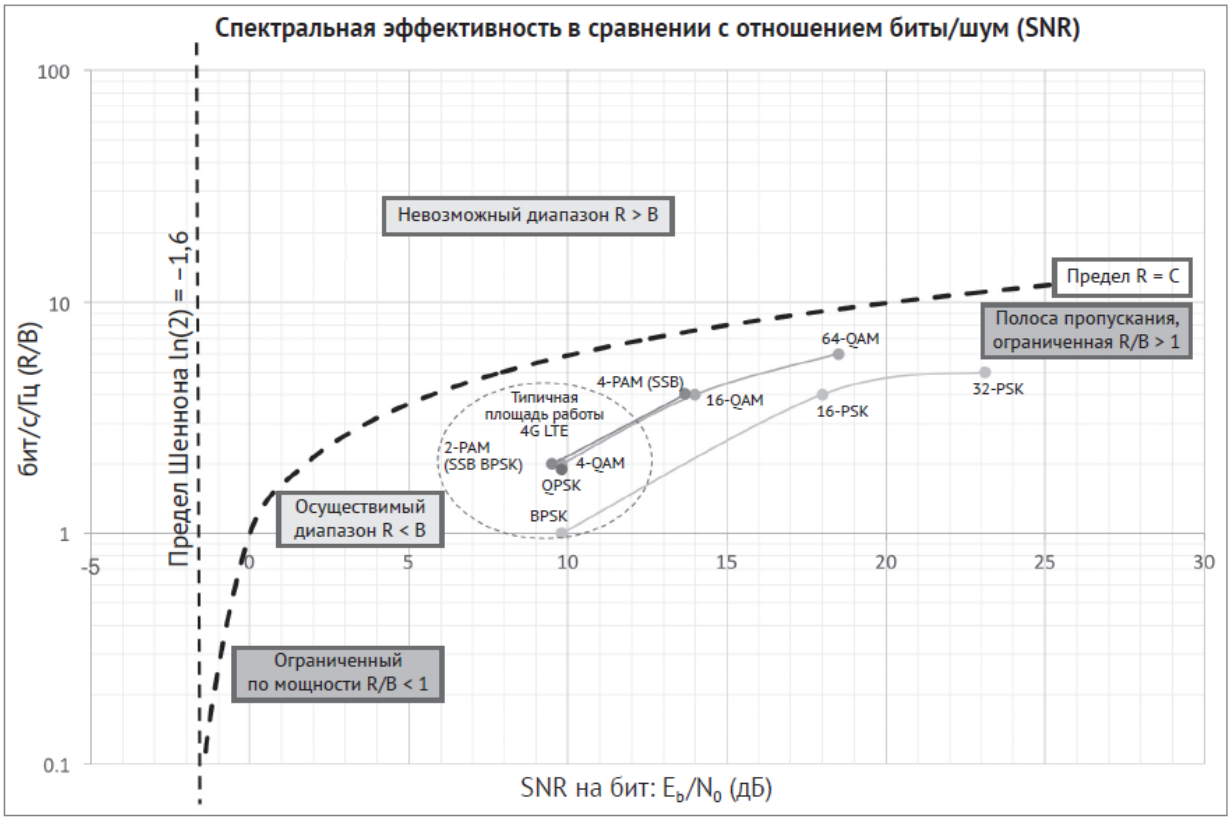


Рисунок 4.4 Крива залежності спектральної ефективності від SNR (енергетична ефективність). Пунктирна лінія являє собою межу Шеннона, яка сходиться до ln (2) = -1,6. Різні схеми модуляції показані в межі Шеннона з типовим діапазоном сигналів 4G LTE

Під границею Шеннона (рис. 4.4) розуміється максимальна швидкість передачі, для якої є можливість виправити помилки в каналі із заданим

відношенням сигнал/шум. Джерела шуму завжди присутні в природі і включають такі речі, як теплові коливання, випромінювання чорного тіла і залишкові ефекти Великого Вибуху. Під «білим» шумом мається на увазі рівна кількість шуму, яка додається до кожної частоти. Границю Шеннона можна намалювати на графіку, що показує спектральну ефективність в порівнянні з SNR на біт і показаному на малюнку нижче:

«Неможлива область» на рис. 4.4 включає всі значення $R > B$. Ця область вище межі Шеннона для кривої. Теорема Шеннона каже, що надійна форма обміну інформацією не може бути вище граничної. Область нижче границі Шеннона називається «можливим діапазоном», у якому всі значення $R < B$. Кожен протокол і технологія модуляції у будь-якій формі комунікації (такі, як акустичний чи електрозв'язок) намагаються наблизитися якомога ближче до границі Шеннона. На малюнку можна побачити, де розташовується типовий 4G LTE з використанням різних форм модуляції.

Є ще дві області, що становлять інтерес. Область «Обмежена смуга пропускання» справа вгорі допускає високу спектральну ефективність і хороші значення SNR E_b/N_0. Єдиним обмеженням в цьому просторі є компроміс з фіксованою або санкціонованою спектральною ефективністю проти необмеженої потужності передачі P, що означає, що ємність каналу значно зросла за тієї ж самої ширини смуги пропускання. Протилежний ефект називається областю обмеженою потужності в напрямку нижнього лівого кута діаграми. Область «Обмеженої потужності» - це та, де SNR E_b/N_0 дуже низький, тому границя Шеннона призводить нас до низьких значень спектральної ефективності. Тут ми жертвуємо спектральною ефективністю для отримання заданої якості передачі P.

Прикладом обмеженого використання енергії є космічний апарат, такий як зонд Кассіні, відправлений до Сатурну. В цьому випадку втрата сигналу в просторі надзвичайно велика, і єдиним способом отримати надійні дані є зниження швидкості передачі даних до дуже низьких значень (20 Кбіт/с). Ми також бачимо це в Bluetooth 5, де швидкість Bluetooth знижується з 1 або 2 Мбіт/с до 125 Кбіт/с для поліпшення спектральної ефективності діапазону і цілісності даних.

На рис. 4.4 також показані деякі типові схеми модуляції, використовувані сьогодні, такі як фазовий зсув, QAM і інші. Границя Шеннона також показує нам, що просте поліпшення методу модуляції, наприклад квадратурної амплітудної модуляції з 4-QAM до 64-QAM, не масштабується лінійно. Перевага більш високих порядків модуляції (наприклад, 64-QAM проти 4-QAM) полягає в тому, що ви можете передавати більше біт на символ (шість проти двох). Основним недоліком більш високих порядків модуляції є:

- використання модуляції вищого порядку вимагає більшого SNR для нормальної роботи;
- вищі порядки модуляції вимагають набагато більш складних схем і алгоритмів обробки DSP, що збільшують складність і вартість сигнального процесора;
- збільшення передачі кількості біт на символ збільшує коефіцієнт помилок.

4.2.2 Коефіцієнт помилок

Іншою важливою характеристикою передачі даних є **частість бітових помилок** *(Bit error rate - BER)*. BER показує відносну кількість бітових помилок, отриманих по каналу зв'язку. BER - це безрозмірна величина, виражена через відношення або відсоток і являє собою вірогідність виникнення помилок. Наприклад:

якщо початкова послідовність така: 1 0 1 0 1 1 0 1 0 0,

а отримана послідовність така: ***0*** *0 1 0 1* ***0 1 0 1*** *0,*

тоді BER буде 5 помилок / 10 переданих біт = 50%.

На BER впливає шум каналу, інтерференція, завмирання внаслідок багатопроменевого поширення і ослаблення сигналу. Методи поліпшення BER включають в себе збільшення потужності передачі, поліпшення чутливості приймача, використання методів модуляції нижчого порядку або додавання додаткових контрольних даних. Останній метод зазвичай називають **прямою корекцією помилок** *(forward error correction - FEC)*. FEC просто додає додаткову інформацію до тої, що передається. У самому простому випадку, можна було б додати потрійну надмірність і алгоритм вибору більшості; проте

це зменшить пропускну здатність в 3 рази. Сучасні методи FEC включають завадостійке кодування і кодування Ріда-Соломона. BER може бути виражений як функція від SNR E_b/N_o. На рис. 4.5 показані різні методи модуляції і їх відповідні BER для різних SNR.

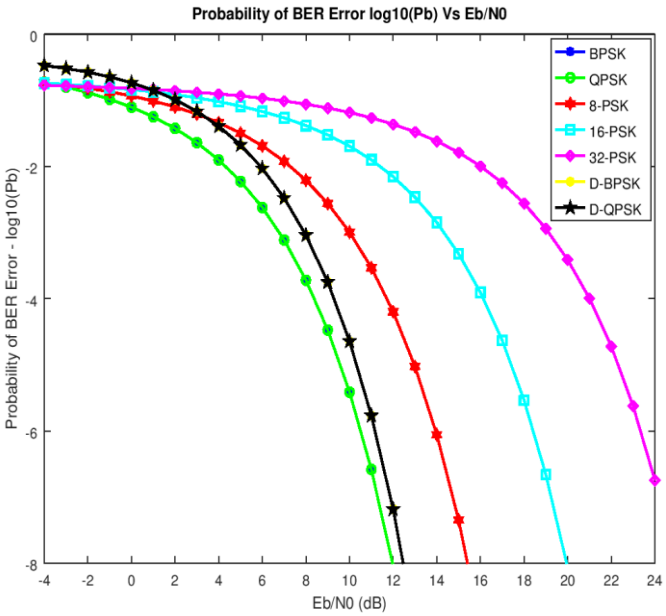


Рисунок 4.5 Частота бітових помилок (Pb) в порівнянні з енергоефективністю (Eb/No) SNR для різних схем модуляції. У міру того як SNR збільшується вправо, BER природно зменшується

На цьому етапі потрібно розуміти наступне:

- тепер ми можемо розрахувати мінімальне SNR, необхідне для досягнення певної швидкості передачі даних для системи;
- єдиний спосіб збільшити ємність або пропускну здатність для бездротової передачі - це:
  - збільшити ширину спектра і ємності каналу, що збільшує смугу пропускання лінійно;
  - додати більше антен (MIMO), що знов лінійно покращує смугу пропускання;
  - поліпшити SNR за допомогою вдосконалених антен і приймачів, що логарифмічно покращує смугу пропускання;
- межа Шеннона є кінцевою границею цифрової передачі; перевищення границі фізично можливо, але цілісність даних буде втрачено;
- чинники, які вносять вклад в шум;
- не можна просто збільшити кількість рівнів модуляції, не збільшуючи витрати на компенсацію помилок і складність обладнання.

Стільникові сигнали 4G LTE, які будуть нами розглянуті далі, працюють в діапазоні від 700 МГц до 5 ГГц з десятками сегрегованих (окремих) смуг. Стільниковий телефон (або пристрій IoT на батареї) має значно меншу потужність, ніж БС стільникового зв'язку, але часто буває, що пристрій IoT буде передавати дані давача у хмару. Висхідний канал від пристрою IoT - це те, що ми розглядаємо тут. Межа потужності висхідної лінії зв'язку становить 200 мВт (23

дБп). Це обмежує загальний діапазон передачі. Ця межа, однак, є динамічною і буде залежати від ширини смуги каналу і швидкості передачі даних. 4G, як і кілька пристроїв WPAN і WLAN, використовують мультиплексування з ортогональним частотним поділом. Кожен канал має безліч піднесучих частот для вирішення проблем багатопроменевого завмирання сигналу. Якщо підсумовувати всі дані, що передаються по піднесучих, можливе досягнення високих швидкостей передачі даних.

Метод оцінки максимальної відстані, на якій ще працює бездротовий пристрій, - це **максимальна втрата зв'язку (***maximum coupling loss - MCL***)**. MCL може бути визначена як максимальна втрата в рівні потужності, яку система може стерпіти і все ще працювати (визначається мінімально прийнятним рівнем вхідного сигналу). MCL - дуже поширений спосіб вимірювання покриття системи. MCL включатиме в себе посилення антени, втрату у просторі, затінення та інші радіоефекти. Різні технології зв'язку, що використовуються в IoT, мають наступні рівні MCL:

Таблиця 4.3 Рівні MCL для різних бездротових технологій

S. No.	Radio Access Technology	Maximum Coupling Loss (MCL) in dB
1	E-GPRS	~ 164 dB
2	LTE	~ 144 dB
3	LTE-M or Cat-M1	~ 160 dB
4	LTE-NB or NB-IoT	~ 164 dB
5	SIGFOX	~ 162 dB
6	LORA	~ 157 dB

Більш детально ми розглянемо MCL при дослідженні стільникових технологій IoT, таких як NB-IoT та LoRa.

4.2.3 Поняття вузькосмугового та широкосмугового зв'язку

Багато бездротових протоколів, що використовуються в IoT, відомі як широкосмугові. Але і альтернативний вузькосмуговий зв'язок також має місце, особливо для LPWAN. Перш, ніж розглянемо відмінності між ними, введемо поняття когерентності каналу. **Смуга когерентності** (*coherence bandwidth*) є статистичною мірою діапазону частот, за яким канал пропускає всі спектральні

компоненти сигналу з приблизно однаковим коефіцієнтом ослаблення і лінійною зміною фази. Таким чином, смуга когерентності являє собою діапазон частот, в межах якого частотні компоненти сигналу мають велику ймовірність амплітудної кореляції. Іншими словами, на всі спектральні компоненти цього діапазону канал впливає однаково, наприклад, проявляючи або не проявляючи завмирання.

Відмінності між вузькосмуговою та широкосмуговою мережею полягають в наступному:

- **вузькосмуговий зв'язок** (*Narrowband*): радіоканал, пропускна здатність якого не перевищує пропускну здатність когерентності каналу. Як правило, коли ми говоримо про вузькосмуговий зв'язок, ми говоримо про сигнали, які мають пропускну здатність 100 кГц або навіть менше. При вузькосмуговому зв'язку багатопроменеве поширення змінює амплітуду і фазу сигнала однаково для усіх частот у діапазоні. Вузькосмуговий сигнал буде зникати рівномірно, тому додавання більшої кількості частот не приносить користі. Вузькополосні канали також називаються плоскими завмираючими каналами, тому що вони, зазвичай, передають все спектральні компоненти з рівним коефіцієнтом ослаблення та однаковою зміною фази синалу для усіх частот;
- **широкосмуговий канал** (*Wideband*): радіоканал, пропускна здатність якого може значно перевищувати когерентну смугу пропускання. Такі канали, як правило, мають ширину, що перевищує 1 МГц в смузі пропускання. Тут багатопроменеве поширення викликає проблему з самоінтерференцією. Широкосмугові канали також називаються частотно-вибірковими, оскільки різні частини загального сигналу будуть передаватися на різних частотах широкосмугового зв'язку. Ось чому широкосмугові сигнали використовують широкий діапазон частот для розподілення потужності сигналу по багатьом смугам когерентності для зменшення ефектів завмирання.

Час когерентності є мірою мінімального часу, необхідного для зміни амплітуди або фази, щоб стати некоррельованим з попереднім значенням.

Ми розглянули деякі форми ефектів завмирання сигнала - проте їх набагато більше. Втрата на шляху розповсюдження сигналу - типовий випадок, коли

втрата пропорційна відстані. Тінь - це те місце, де ландшафт, будівлі і пагорби створюють перешкоди сигналу в порівнянні з вільним простором, а завмирання багатопроменевості відбувається при рекомбінації розсіяння і хвильової інтерференції радіосигналу на об'єктах (через дифракцію та відбиття). Інші втрати включають допплерівський зсув частоти, якщо джерело радіочастотного сигналу знаходиться у русі. Існують дві категорії завмирання сигналу:

- **швидке завмирання**: це характерно для багатопроменевого завмирання, коли час когерентності невеликий. Канал буде змінюватися кожні кілька символів; тому час когерентності буде низьким. Цей тип завмирання також відомий як *завмирання Релея*, що є ймовірністю випадкових дисперсій, які викликає радіочастотний сигнал на атмосферних частинках або у сильно забудованих мегаполісах;
- **повільне завмирання**: це відбувається, як правило, через доплерівське поширення або затінення, коли час когерентності великий і відбувається рух на великі відстані. Тут час когерентності досить великий, щоб успішно передавати значно більше символів, ніж при швидкому завмиранні.

На малюнку нижче іллюструються ефекти від різних видів завмирань:

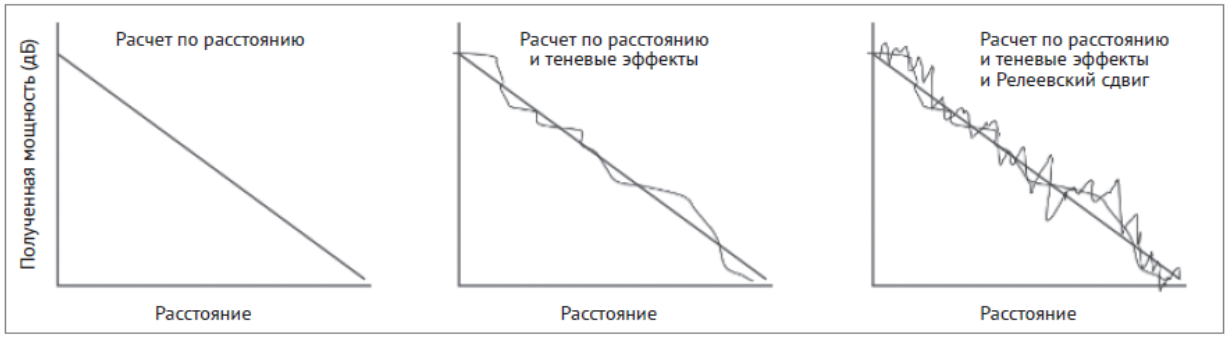


Рисунок 4.6 Різні ефекти завмирання радіочастотного сигналу. Зліва направо: загальні втрати на шляху прямої видимості. Середній: ефекти повільного завмирання через наяність великих структур або ландшафту. Справа: комбіновані ефекти відстані, повільного і швидкого

Когерентна смуга пропускання визначається як статистичний діапазон частот, в яких канал вважається плоским. Це період часу, коли дві частоти, ймовірно, будуть мати однакове завмирання. Ширина когерентної смуги пропускання B_c обернено пропорційна розкиду часу затримки сигнала D:

$$B_c \approx {}^{1}/_{D} \qquad (4.12)$$

де B_c- ширина когерентної смуги пропускання, Гц;

D - час затримки сигнала, с.

Час, протягом якого символ може передаватися без межсимвольной інтерференції, дорівнює *D*. На рис. 4.7 проілюстровано цю ситуацію:

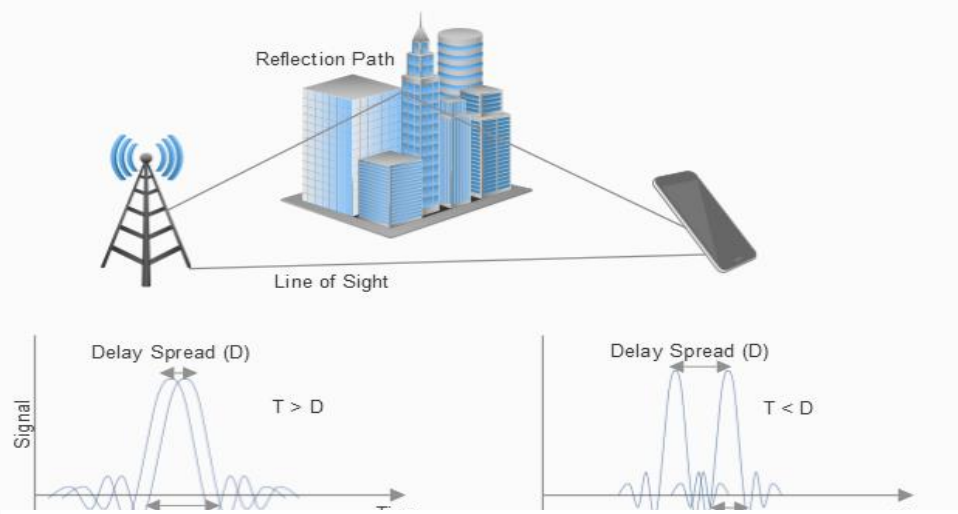


Рисунок 4.7 Приклад затримки розповсюдження. Тут сигнал до приймача розповсюджується двома шляхами. Якщо поширення затримки D менше, ніж тривалість імпульсу T, то багатопроменеві сигнали можуть не заважати одне одному. В іншому випадку, коли T < D існує конфлікт

На загальний бітрейт буде впливати розкид затримки. Наприклад, припустимо, що ми використовуємо модуляцію QPSK, а BER = 10^{-4}, тоді для різних значень затримки *D* маємо:

- D = 256мкс: 8 КБ/с;
- D = 2.5 мкс: 80 КБ/с;
- D = 100 нс: 2 Мб/с.

Контрольні питання

1. Дискретний давач видає послідовність 3-символьних повідомлень A_{i1}, A_{i2}, A_{i3} (перший індекс показує значення елемента, а другий - його номер в послідовності), які обираються з дискретного алфавіту (i = $\overline{0, K-1}$; K=8 - обсяг алфавіту давача). Скільки різних повідомлень N може видати такий давач? Випишіть реалізації повідомлень, в яких два перших символа будуть a_{11}, a_{72}.

2. Дані з сервера видаються в двійковому коді (m=2) кодовими комбінаціями, що містять n=7 символів. Скільки таких повідомлень може видати джерело? Напишіть дві реалізації такого джерела, приймаючи $a_{ik} \in \{0, 1\}$.
3. Як змінюється радіальна складова поля Е в даній точці в залежності від кута φ між напрямом на дану точку і віссю короткої антени?
4. Визначити максимально можливий бітрейт при передачі даних з давача при часі затримки D= 500 нс, 1 мкс, 10 мкс у когерентній смузі пропускання.
5. У чому полягає різниця між вузькосмуговим та широкосмуговим зв'язком? Навести приклади.
6. Як буде впливати коефіцієнт помилок на максимальний бітрейт? Навести приклади.
7. Що таке когерентна смуга пропускання і як вона впливає на передачу сигнала? Навести приклади.
8. Як розрахувати втрати радіосигналу від давача до мережевого шлюза?

5 ПЕРСОНАЛЬНІ БЕЗДРОТОВІ МЕРЕЖІ НЕ НА IP ОСНОВІ

В екосистемі інтернету речей зв'язок з давачем або виконавчим механізмом може здійснюватися по мідних проводах або за допомогою **бездротової персональної мережі** регіону (*WPAN*). У даний час WPAN є найпоширенішим методом промислового, комерційного та споживчого підключення до речей в інтернеті. Зв'язок на основі проводів використовується як і раніше, але, перш за все, на існуючих системах і в областях промисловості, які не є сприятливими для радіосигналів чи де існують безпекові обмеження. Існує безліч різних каналів зв'язку між кінцевою точкою і інтернетом; деякі можуть бути побудовані на традиційному стеку IP (6LoWPAN), а інші використовують не-IP (інтернет-протокол) для максимізації економії енергії (BLE).

Ми поділяємо IP і не-IP мережі, оскільки IP-системи зв'язку потребують додаткової деталізації, яка не є обов'язковою для не-IP мереж. Системи зв'язку, відмінні від IP, оптимізовані для економії витрат і енергії, тоді як рішення на базі IP зазвичай мають менше обмежень (наприклад, 802.11 Wi-Fi).

Слід мати на увазі, що з самого початку термін WPAN використовувся для позначення виключно середовища і особистої мережі для конкретної людини, пов'язаної з мобільними пристроями, але тепер його сенс значно розширився (на промислові мережі).

5.1 Існуючі стандарти бездротової персональної локальної мережі

Протоколи і мережеві моделі, що використовуються у WPAN, у своїй більшості базуються або засновані на стандартах, розроблених робочими групами IEEE 802.15. Спочатку група 802.15 була створена для того, щоб зосередитися на розробці стандартів підключення для мобільних пристроїв (еквівалент персональної мережі). Однак надалі їхня робота була значно розширена і тепер фокусується на більш високих протоколах швидкості передачі даних, дальності зв'язку до кілометрів і спеціальних комунікаціях. Понад мільйон пристроїв з використанням протоколу 802.15.x виготовляється щодня. Нижче наведено список різних основних протоколів, стандартів і специфікацій, які IEEE підтримує і регулює:

- 802.15 – загальні визначення бездротової локальної мережі WPAN;
- 802.15.1 – оригінальний стандарт Bluetooth PAN;
- 802.15.2 - специфікації співіснування для Bluetooth WPAN і інших безпровідних пристроїв, що працюють в неліцензованих діапазонах частот (наприклад, WLAN);
- 802.15.3 - висока швидкість передачі даних (55 Мб/с і більше) в WPAN для використання в мультимедіа;
- 802.15.4 - низька швидкість передачі даних (до 250 Кб/с), простий дизайн, багаторічна робота батарей (протокол Zigbee), низька вартість;
- 802.15.5 - чарункова (mesh) мережа;
- 802.15.6 - мережева інфраструктура для медицини і розваг;
- 802.15.7 - видимий світловий зв'язок з використанням структурованого освітлення.

5.2 Стандарт 802.15.1 Bluetooth

Bluetooth сьогодні – це не тільки підключення гарнітури до смартфонів, чи використання у відеоіграх [17]. Наразі він широко застосовується в розгортанні IoT, будучи основним пристроєм при використанні в режимі низького споживання енергії (LE) для маяків, бездротових давачів, систем відстеження стану технологічного обладнання, пультів дистанційного керування, моніторів працездатності і систем сигналізації. В силу цих причин Bluetooth став широко поширеним, і ми будемо розглядати новий протокол Bluetooth 5, який було ратифіковано консорціумом Bluetooth SIG в 2016 р. Історія розвитку стандарту викладена, наприклад, в статті "Правдива історія Bluetooth. Як він розвивався і що буде далі".

5.2.1 Топології і процес зв'язку у Bluetooth 5

Стандарт Bluetooth побудовано на основі двох бездротових технологій: *Basic Rate (BR)* і *Low Energy* (LE або BLE). Вузли можуть бути публікаторами, сканерами чи ініціаторами за наступним визначенням:

- **публікатор**: пристрій, який *передає* пакети рекламних (широкомовних) повідомлень;
- **сканер**: пристрій, який *приймає* пакети від публікатора без наміру підключитися до нього;
- **ініціатор** - пристрій, який намагається сформувати з'єднання.

У Bluetooth мережі існує декілька різновидів подій:

- **публікація (реклама)** - ініціюється публікатором для трансляції на сканери, щоб попередити їх про наявність пристрою, який бажає або ретранслювати повідомлення від іншого публікатора, або просто передати своє власне повідомлення у пакеті;
- **підключення** - процес парування пристрій-хост;
- **періодична публікація** (для Bluetooth 5) - дозволяє публікатору періодично рекламувати 37 непервинних каналів шляхом переходу з каналу на канал з інтервалом від 7,5 до 81,91875 с;
- **розширена публікація** (для Bluetooth 5) - дозволяє використовувати розширений розмір пакета даних (*protocol data unit – PDU*) для підтримки ланцюжка рекламних повідомлень і великих PDU, а також нові варіанти використання аудіо або інших мультимедіа.

У BLE режимі пристрій може завершити з'єднання, просто використовуючи канал публікації. В якості альтернативи з'єднання може вимагати *дуплексний* (двосторонній) канал і примусово вимагати формального підключення пристроїв. Пристрої, які повинні сформувати цей тип з'єднання, почнуть процес, прослуховуючи пакети рекламних повідомлень. В цьому випадку сканер називається ініціатором. Якщо публікатор передає рекламне повідомлення, з якого випливає необхідність подальшого підключення сканера до публікатора, ініціатор може зробити запит на з'єднання з використанням того ж фізичного каналу, на якому він отримав рекламний пакет від публікатора.

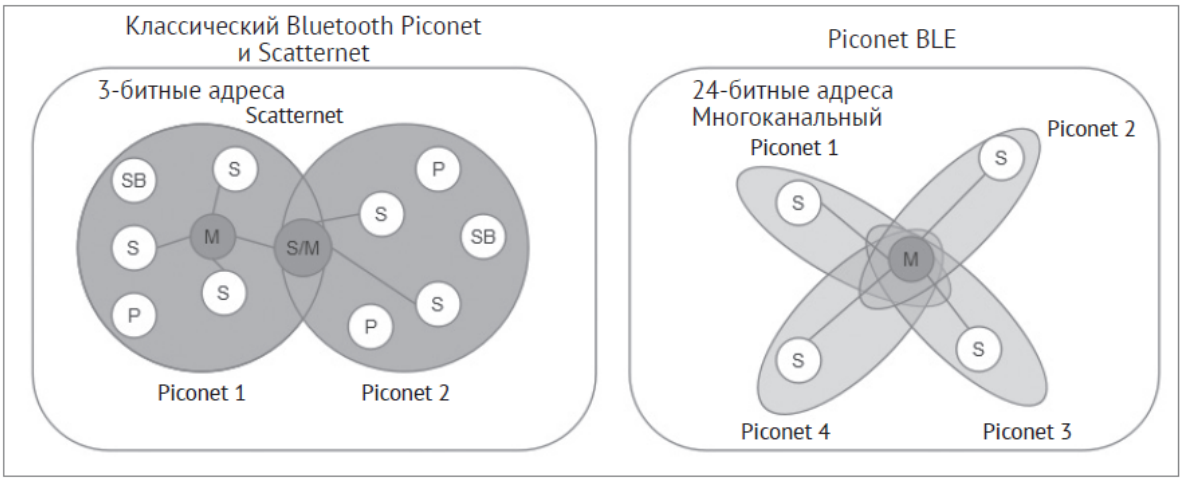


Рисунок 5.1 Різниця між класичними (BR/EDR) і BLE пікомережами. У режимі BR/EDR до семи ведених пристроїв можна зв'язати в одну пікомережу через трибітну адресацію. Всі вони мають загальний канал між сімома веденими. Інші пікомережі можуть приєднуватися до мережі

Публікатор далі може визначити, чи хоче він встановити з'єднання. Якщо з'єднання сформовано, подальша публікація рекламних повідомлень публікатором завершується, і ініціатор тепер називається **майстром (master)**, а публікатор – **веденим (slave)**. Це з'єднання називається пікомережа. Всі події з'єднання відбуваються на одному і тому ж початковому каналі між ведучим і веденим. Після обміну даними і завершенням з'єднання для пари може бути виділений новий канал з використанням стрибкоподібної перебудови частоти.

Пікомережі формуються у двох різних режимах в залежності від режиму роботи пристроїв: BR/EDR або BLE. BR/EDR пікомережа використовує трибітову адресацію і дає можливість підключення до семи ведених (slave) пристроїв в одну пікомережу. Декілька пікомереж можуть утворювати об'єднання, що називається **scatternet**, але тут повинен бути присутнім другий майстер-пристрій для підключення до другої мережі і управління нею. Ведений/головний вузол бере на себе відповідальність за об'єднання двох пікомереж разом. У режимі BR/EDR мережа використовує один і той самий графік стрибкоподібної перебудови частоти, і всі вузли будуть гарантовано перебувати на одному і тому самому каналі в кожний момент часу. У режимі BLE система використовує 24-бітну адресу, тому кількість пристроїв, пов'язаних з майстром, виражається вже у мільйонах. Кожне відношення «майстер-ведений» - є пікомережею і може використовувати унікальний канал. У пікомережі вузли можуть бути **ведучі** (M), **ведені** (S) або **резервними** (SB). Резервний режим

(очікування) - це стан за замовчанням для вузла. У такому стані він має можливість знаходитися у режимі споживання малої потужності. До 255 інших пристроїв можуть працювати у режимі роботи SB в одній і тій самій пікомережі (див. рис. 5.1).

5.2.2 Стек Bluetooth 5.0

Bluetooth має три основних компоненти: апаратний контролер, програмне забезпечення для хоста і профілі застосувань. Пристрої Bluetooth поставляються в одно- і дворежимних версіях, що означає, що вони або підтримують тільки стек BLE, або одночасно підтримують класичний режим (BR/EDR) і BLE. Bluetooth дозволяє підключати один або кілька контролерів до одного хосту.

Стек складається з рівнів або протоколів і профілів:

- **протоколи** - горизонтальні рівні або шари, що представляють функціональні блоки. На рис. 5.2 показаний стек протоколів;
- **профілі** – являють собою вертикальні функції, що використовують протоколи. Профілі будуть частково описані далі.

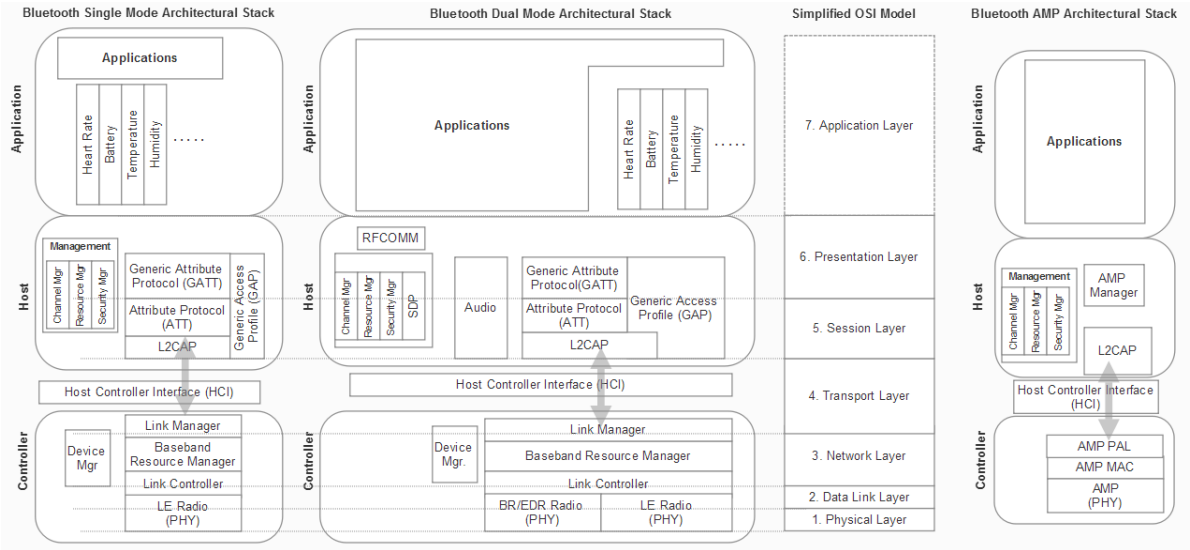


Рисунок 5.2 Bluetooth Single Mode (тільки BLE) і Dual Mode (Classic та BLE)

Існують три режими роботи Bluetooth 5, показані на рис. 5.2 (кожен з яких вимагає різні параметри *фізичного каналу* (PHY)):

- **режим низького споживання енергії** (LE) - використовується смуга ISM 2,4 ГГц і FHSS (стрибкоподібну зміну частоти) для захисту від перешкод. PHY відрізняється від PHY режимів BR/EDR і AMP

модуляцією, кодуванням і швидкістю передачі даних. LE підтримує кілька швидкостей передачі даних: 125 Кбіт/с, 500 Кбіт/с, 1 Мбіт/с і 2 Мбіт/с;

- **режим базової швидкості / поліпшеної швидкості передачі даних** (BR/EDR) - використовується не таке радіо, як BLE і AMP, але воно також працює в діапазоні ISM 2,4 ГГц. Базова радіостанція (BR) підтримує швидкість передачі до 1 Мбіт/с. EDR підтримує швидкість передачі даних до 2 або до 3 Мбіт/с. Це радіо також використовує FHSS для захисту від перешкод;
- **альтернативний MAC/PHY** (AMP) - це додаткова функція, яка використовує стандарт Wi-Fi 802.11 для високошвидкісної передачі до 24 Мбіт/с. Цей режим вимагає, щоб ведучий і ведений пристрій підтримували AMP. Це вторинний фізичний контролер, але він вимагає, щоб система мала контролер BR/EDR для встановлення початкового з'єднання і узгодження.

5.2.3 Робота в режимі BR/EDR

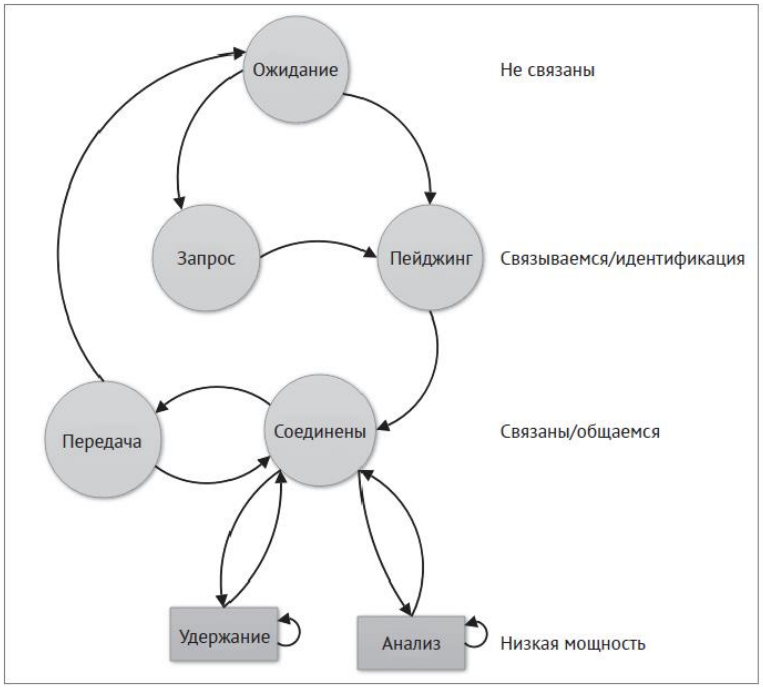


Рисунок 5.3 Процес підключення для Bluetooth BR/EDR від непідключеного пристрою в режимі очікування, запит пристрою та його виявлення, режим підключення / передачі і режими з низьким енергоспоживанням

Режим класичного Bluetooth (BR/EDR) орієнтований на з'єднання. Зараз цей режим використовується там, де вже існують раніше побудовані мережі на

основі Bluetooth 4. Якщо пристрій підключено, зв'язок підтримується, навіть якщо дані не передаються. Перш ніж будь-яке з'єднання Bluetooth з'явиться, пристрій має бути виявлено, щоб він відповідав на сканування фізичного каналу і згодом відповів своєю адресою та іншими параметрами. Пристрій повинен знаходитися в режимі підключення для контролю сканування сторінки.

Процес підключення (рис. 5.3) виконується в три етапи:

1) **запит** - на цьому етапі два пристрої Bluetooth ніяк не асоційовані або не пов'язані; вони нічого не знають один про одного. Пристрої повинні виявити один одного через запит. Якщо інший пристрій слухає, він може відповісти своєю адресою BR_ADDR;
2) **пейджинг** - це утворення з'єднання між двома пристроями. Кожен пристрій знає BD_ADDR один одного в цей момент;
3) **підключено** - існує чотири підрежиму стану підключення. Це нормальний стан, коли два пристрої активно спілкуються:
    a. **активний режим** - це нормальний режим роботи для передачі і прийому даних Bluetooth або очікування наступного слота передачі;
    b. **режим аналізу** - режим енергозбереження. Пристрій, по суті, спить, але буде прослуховувати передачі під час певних слотів, які можуть бути змінені програмно (наприклад, 50 мс);
    c. **режим очікування** - це тимчасовий режим низького споживання енергії, ініційований ведучим або веденим пристроєм. Пристрій не буде слухати передачі, як в режимі аналізу, і ведений тимчасово ігнорує всі ACL-пакети. В цьому режимі перемикання в підключений стан відбувається дуже швидко;
    d. **режим парковки** - цей режим застарів і в Bluetooth 5 не використовується.

Діаграма стану цих фаз має вигляд, приведений на рис. 5.3. Якщо процес підключення завершується успішно, два пристрої можуть бути примусово автоматично спаровані (підключені), коли вони знаходяться в зоні дії одне одного. Прикладом такого одноразового процеса парування може бути

підключення до смартфону Bluetooth-гарнітури, але той самий процес може застосовуватися в будь-якому місці і в IIoT. Спаровані пристрої будуть використовувати ключ аутентифікації, про який буде сказано далі.

5.2.4 Робота в режимі BLE

Саме режим BLE повинен використовуватися у всіх нових розробках сенсорних мереж на основі Bluetooth. У режимі BLE існує п'ять станів зв'язку, які узгоджені хостом і пристроєм:

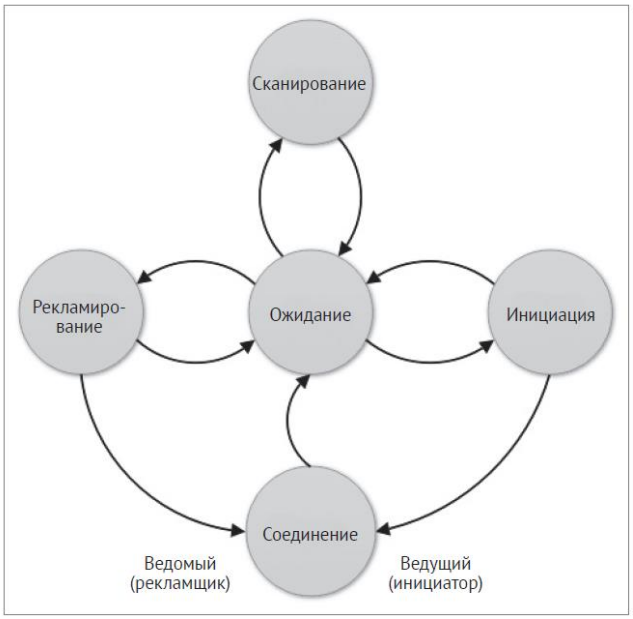


Рисунок 5.4 Стани зв'язку BLE

- **рекламування** - пристрої, які передають рекламні пакети по рекламним каналам;
- **сканування** - пристрої, які отримують рекламу по рекламним каналам без наміру підключитися. Сканування може бути активним чи пасивним:
  - - **активне сканування** - канальний рівень прослуховується на предмет рекламних блоків PDU. Залежно від прийнятого PDU сканер може зажадати від рекламодавця відправити додаткову інформацію;
  - - **пасивне сканування** - канальний рівень буде тільки приймати пакети, передача відключена;
- **ініціювання** - пристрої, яким необхідно сформувати з'єднання з іншим пристроєм, прослуховують рекламні пакети і ініціюють відправку пакета підключення при отриманні "свого" пакета;
- **підключено** - між підключеним ведучим і веденим пристроєм встановлено зв'язок. Ведучий - ініціатор, а ведений - публікатор:
  - - центральний - ініціатор стає центральним пристроєм;
  - - периферійний - публікатор стає периферійним пристроєм;
- **режим очікування** - пристрій в непідключеному стані.

Стан публікації має кілька функцій і властивостей. Оголошення можуть бути *широкомовними*, коли пристрій транслює загальне запрошення на усі інші пристрої в мережі. Оголошення можуть бути і *орієнтованими* на конкретний пристрій мережі. Такі рекламні пакети містять не лише адресу публікатора, але і адресу потрібного пристрою.

Коли приймаючий пристрій розпізнає потрібний йому пакет, він негайно відправить запит на з'єднання. *Спрямована реклама* - це швидка і негайна увага публікатору, тут рекламні оголошення відправляються зі швидкістю 3,75 мс, але тільки на протязі 1,28 с. Інший тип рекламних повідомлень – *без встановлення з'єднання*, по суті, є маяком (і може навіть не мати приймачів!). На рис. 5.4 показані п'ять станів зв'язку при роботі в режимі BLE.

Пристрій BLE, який раніше не зв'язувався з хостом, ініціює комунікацію, публікуючи рекламні оголошення по трьом рекламним каналам. Хост може відповісти SCAN_REQ, щоб запитати додаткову інформацію з публікатора. Периферійний пристрій відповідає SCAN_RSP і включає ім'я пристрою або, можливо, служби.

SCAN_RSP може вплинути на використання енергії на периферійному пристрої. Якщо пристрій підтримує відповіді на сканування, він повинен постійно підтримувати своє радіо активним в режимі прийому, споживаючи енергію. Це відбувається, навіть якщо жоден хост-пристрій не видає SCAN_REQ. Доцільно відключити відповіді сканування на периферійних пристроях IIoT, що мають обмеження по живленню.

Після сканування хост (сканер) ініціює CONNECT_REQ, після чого сканер і публікатор відправлять порожні пакети PDU для вказівки підтвердження. Сканер тепер називається майстром (ведучим), а публікатор називається веденим. Ведучий пристрій може виявляти профілі відомих і служби через GATT. Після того, як виявлення завершено, дані можуть обмінюватися від веденого пристрою до ведучого і навпаки. По завершенні ведучий повернеться в режим сканування, а ведений пристрій повернеться в режим публікатора.

5.2.5 Профілі Bluetooth

Інтерфейс застосувань з різними пристроями Bluetooth будується з використанням профілів. Профілі визначають функціональні можливості і функції для кожного рівня стека Bluetooth. По суті, профілі пов'язують стек разом по вертикалі і визначають, як саме різні рівні стека взаємодіють один з одним. Профіль описує характеристики виявлення, які публікує пристрій. Вони також використовуються для опису форматів даних послуг і їх характеристик, які застосування використовуються для читання і запису на пристрій. Профіль не існує на пристрої; скоріше, вони є зумовленими програмними конструкціями, підтримуваними і керованими організацією Bluetooth SIG.

Профіль Bluetooth BR/EDR повинен містити GAP (*Generic Access Profile* – базовий профіль для всіх інших профілів), як зазначено в специфікації. GAP визначає параметри радіо, смугу пропускання, менеджер зв'язків, L2CAP (*logical Link Control and Adaptation Protocol* - використовується для мультиплексування локальних з'єднань між двома пристроями, що використовують різні протоколи більш високого рівня. Дозволяє фрагментувати і збирати заново пакети), виявлення служби для пристроїв BR/EDR. Аналогічно, для пристрою BLE GAP визначатиме радіо, рівень каналу, L2CAP, диспетчер безпеки, протокол атрибутів і загальний профіль атрибутів.

Рисунок 5.5 Стек протоколів BLE

Протокол атрибутів ATT (*attribute protocol*) є протоколом клієнт-сервер, оптимізованим для малопотужних пристроїв (наприклад, значення довжини пакету ніколи не передається через BLE, вона задається розміром самого PDU). ATT є дуже загальним протоколом, тому багато чого ще залишається для профілю GATT (*Generic Attribute Profile* - профіль загальних атрибутів). Профіль ATT складається з:

- 16-розрядного дескриптора;

- UUID для визначення типу атрибута;
- значення, що містить довжину.

GATT логічно знаходиться поверх ATT (див. рис. 5.5) і використовується в основному, якщо не виключно, лише для пристроїв BLE. GATT визначає ролі сервера і клієнта. Клієнт GATT зазвичай є периферійним пристроєм, а сервером GATT є хост (ПК, смартфон). Профіль GATT містить дві компоненти:

- **служби** - служба розбиває дані на логічні об'єкти. У профілі може бути кілька служб, і кожна служба має унікальний свій UUID, щоб відрізняти її від інших;
- **характеристики** - характеристикою є найнижчий рівень профілю GATT, він містить необроблені дані, пов'язані з пристроєм. Формати даних відрізняються 16-бітовим або 128-бітовим UUID. Розробник може вільно створювати свої власні характеристики, які може інтерпретувати тільки його додаток.

На малюнку нижче представлений приклад профілю Bluetooth GATT з відповідними UUID для різних сервісів і характеристик.

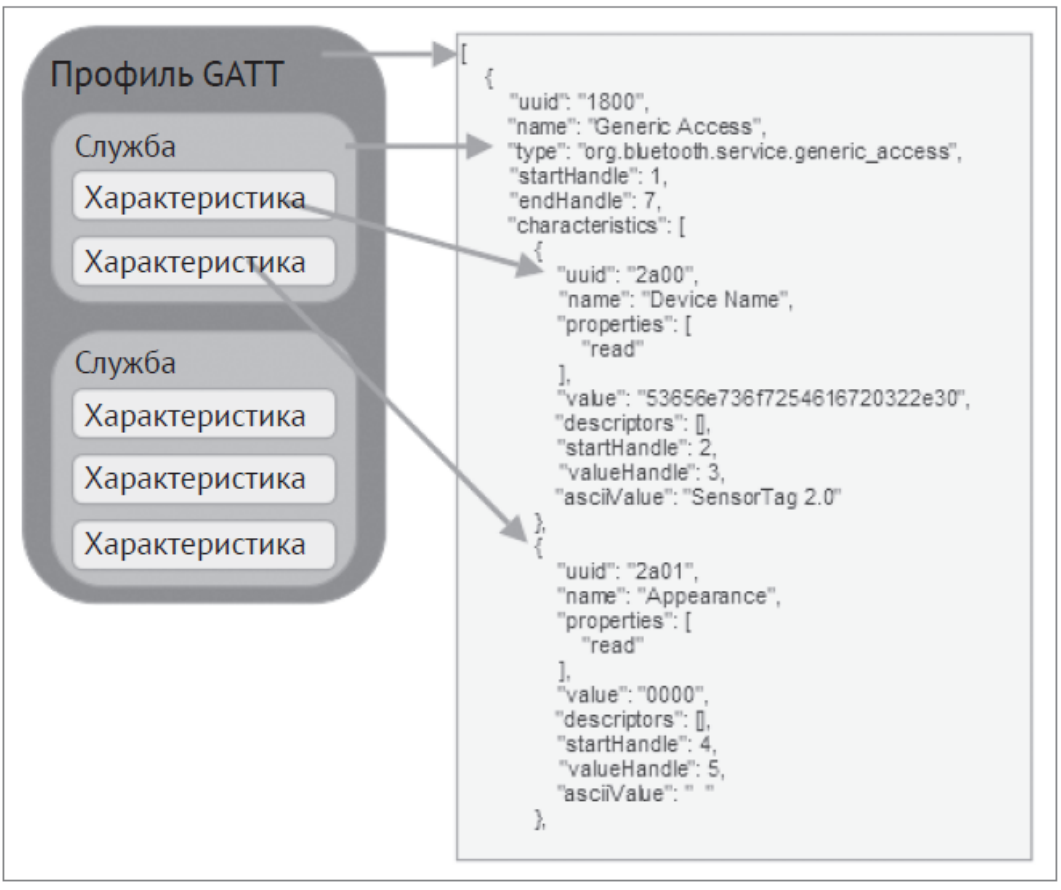


Рисунок 5.6 Ієрархія профілю GATT і приклад GATT, який використовується в давачі Texas Instruments CC2650 SensorTag

Bluetooth SIG підтримує багату колекцію профілів GATT. На початок 2019 року на Bluetooth SIG підтримувалося 95 профілів GATT (Популярні профілі, Інші профілі). Профілі, підтримувані SIG, включають в себе все: від моніторів здоров'я, велосипедних і фітнес-пристроїв до моніторів навколишнього середовища, пристроїв інтерфейсу користувача, внутрішнього позиціонування, передачі об'єктів і місця розташування і навігаційних служб, а також багато інших.

5.2.6 Безпека в режимі BR/EDR

Для забезпечення безпечної роботи мережі в режимах BR/EDR і BLE рекомендується використовувати останнє керівництво з безпеки, надане Національним інститутом стандартів і технологій США: Посібник з безпеки Bluetooth, Спеціальна публікація NIST (SP), 800-121 Rev. 2, NIST, 5/8/2017.

Для парування пристроїв одне з одним потрібна генерація секретного симетричного ключа. У режимі BR/EDR це називається ключем з'єднання, тоді як в режимі BLE він називається довгостроковим ключем. Старі пристрої Bluetooth використовували режим парування з **персональним ідентифікаційним номером (PIN)**, щоб ініціювати ключі з'єднання. Нові пристрої (починаючи з версії 4.1+) використовують безпечне просте сполучення.

Безпечне просте сполучення (SSP) забезпечує процес сполучення за допомогою ряду різних моделей асоціацій для різних випадків використання. SSP також використовує криптографію з відкритим ключем для захисту від підслуховування і атак типу «*людина-посередині*» (MITM). SSP підтримує чотири моделі аутентифікації:

- **числове порівняння** - для випадків використання, коли обидва пристрої Bluetooth можуть відображати шестизначне числове значення, що дозволяє користувачеві вводити відповідь «так»/«ні» на кожному пристрої, якщо числа збігаються;
- **введення загального ключа** - використовується в ситуаціях, коли один пристрій має числовий дисплей, а інший має цифрову клавіатуру. У цьому випадку користувач вводить значення, що

відображається на дисплеї першого пристрою на клавіатурі другого пристрою;

- **JustWorks** - для ситуацій, коли один пристрій не має клавіатури або дисплея. За фактом аналогічний числовому порівнянню, проте число завжди дорівнює «000000». Він забезпечує мінімальну аутентифікацію і не буде перешкоджати атаці MITM;

Рисунок 5.7 OOB - Apple Watch при паруванні показує хмару точок, яку потрібно відсканувати камерою iPhone

- **позасмуговий канал (**Out Of Band - OOB**)** - використовується, коли пристрій має додатковий канал зв'язку, оптичний чи радіо, наприклад, Wi-Fi. Вторинний канал використовується для виявлення і обміну криптографічними значеннями. Він захистить тільки від підслуховування і MITM, якщо канал у свою чергу OOB захищений.

Аутентифікація в режимі BR/EDR - це виклик-відповідь; наприклад, введення PIN-коду на клавіатурі. Якщо аутентифікація не вдалася, пристрій буде очікувати деякий час, перш ніж дозволити нову спробу. Інтервал зростає експоненціально з кожною невдалою спробою. Це просто розчаровує людину, що намагається вручну підібрати код ключа. Шифрування в режимі BR/EDR може бути встановлено таким чином, щоб:

- воно було відключено для всього трафіку;
- шифрується тільки трафік даних, але широкомовний зв'язок (публікація рекламних оголошень) буде відкритим;
- шифрується увесь зв'язок.

Шифрування використовує криптографію AES-CCM.

5.2.7 Безпека в режимі BLE

Парування BLE починається з пристрою, який ініціює запит *Pairing_Request* і далі пристрої починають обмінюватися характеристиками, вимогами, тощо. На початковому етапі процесу парування не виникає нічого, пов'язаного з профілями безпеки. В цьому відношенні безпека парування аналогічна чотирьом методам BR/EDR (також відомим як моделі асоціації), але в Bluetooth BLE 4.2 вона трохи відрізняється:

- **числове порівняння** - це те ж саме, що і JustWorks, але в кінці обидва пристрої генерують значення підтвердження, яке відображається на екранах хоста і пристроя, щоб користувач міг перевірити відповідність;
- **введення загального ключа** - подібно режиму BR/EDR, за винятком того, що неініціюючий пристрій створює випадкове 128-бітове число, яке називається ключем доступу для аутентифікації з'єднання. Кожен біт ключа доступу аутентифікується окремо на кожному пристрої, генеруючи значення підтвердження для цього біта. Пристрої обмінюються значеннями підтвердження, які повинні співпасти. Процес триває до тих пір, поки всі біти ключа доступу НЕ будуть оброблені. Це забезпечує досить надійне рішення для атак MITM;
- **JustWorks** - після того, як пристрої обмінюються відкритими ключами, неініціюючий пристрій генерує випадковий код для створення значення підтвердження C_b. Далі він передає випадковий код і C_b на пристрій, що ініціював з'єднання, а той, в свою чергу, генерує свій власний випадковий код і передає його першому пристрою. Ініціюючий пристрій потім підтвердить автентичність неініціюючого випадкового коду, створивши власне значення C_a, яке повинно відповідати C_b. Якщо воно не відповідає, з'єднання пошкоджено. Це знову-таки відрізняється від аналогічного режиму BR/EDR;

- **позасмуговий канал (OOB)** – працює так само, як і в режимі BR/EDR. Він захистить тільки від підслуховування і MITM, якщо канал OOB захищений.

У BLE (як у випадку Bluetooth 4.2) генерація ключів використовує безпечні з'єднання LE. Безпечне з'єднання LE було розроблено для вирішення проблеми безпеки парування BLE, яке дозволяло *підслуховуючому* побачити обмін парування. У цьому процесі для шифрування з'єднання використовується довгостроковий ключ (*long-term key - LTK*). Ключ заснований на криптографії з відкритим ключем алгоритма Діффі-Хеллмана на еліптичних кривих (*elliptical-curve Diffie-Hellman - ECDH*).

І ведучий, і ведений будуть генерувати пару ключів публічний - приватний ECDH. Ці два пристрої будуть обмінюватися публічними частинами своїх відповідних пар і обробляти у себе ключ Діффі-Хеллмана. На цьому етапі з'єднання може бути зашифровано з використанням криптографії AES-CCM.

BLE також має можливість зробити випадковим свій *BD_ADDR*. Згадаймо, що *BD_ADDR* - це 48-бітна MAC-адреса пристрою. Замість постійної статичної адреси для пристрою, можливо використовувати ще три варіанти з випадковими адресами:

- **випадковий статичний** - ця адреса або прошивається на пристрої під час його виготовлення, або генерується при періодичному ввімкненні живлення пристрою. Якщо пристрій зазвичай працює періодично, генерується унікальна адреса тільки на час його роботи, і він залишається захищеним, поки частота циклів ввімкнення достатньо висока. У середовищі давачів IoT це може бути не так;
- **випадковий приватний розв'язний** - цей метод адресації може використовуватися тільки в тому випадку, якщо протягом процесу зв'язування обидва пристрої обмінюються ідентифікаційним ключем (*Identity Resolving Key - IRK*) між собою. Пристрій буде використовувати IRK для кодування своєї адреси в випадкову адресу в рекламному пакеті. Другий пристрій, який також має той самий IRK, перетворює випадкову адресу назад в автентичну першому

пристрою адресу. У цьому методі пристрій буде періодично генерувати нову випадкову адресу, засновану на IRK;

- **випадковий приватний нерозв'язний** - адреса пристрою являє собою просто випадкове число, і в будь-який момент може бути згенерована нова адреса пристрою. Це забезпечує найвищий рівень безпеки.

5.2.8 Маяки

Маяки Bluetooth - вторинний ефект від BLE; проте це важлива і значуща технологія для IIoT. Маяки не обов'язково мають бути давачами, хоча деякі з них надають інформацію про виявлення в рекламному пакеті. Маяк просто використовує пристрої Bluetooth в режимі LE для реклами на **періодичній** основі. Маяки ніколи не з'єднуються і не сполучаються з хостом. Якби маяк підключався, всі рекламні оголошення від нього зупинялися, і ніякий інший пристрій вже не зміг би чути цей маяк. Наведемо три випадки використання маяків, важливі для роздрібної торгівлі, освіти, інфраструктури міста:

- лондонське агентство Beelieve Creative Developer застосувало технологію iBeacon для заохочення пунктуальності в шкільних класах і відстеження відвідуваності уроків, розробивши додаток *BeHere*. Воно перетворює iPad вчителя в імпровізований, але ефективний Bluetooth-маяк. Як тільки пристрій студента, що також встановив додаток, виявляється в «зоні видимості», інформація про прибуття передається на iPad вчителя. Такий підхід дозволяє припинити щоденні рутинні перевірки відвідуваності, даючи можливість витратити ці кілька хвилин на додаткове навчання або обговорення пройденого матеріалу. Таким чином, технологія виконує одну з головних своїх завдань - усуває непотрібні, безглузді дії і звільняє час для важливих справ;
- у місті Хартфорд штату Коннектикут в США на муніципальному рівні вирішили впроваджувати маяки для розвитку локального бізнесу та туризму в районі Парквілл. Адміністрація планує встановити 100 маяків по всьому району і розвісити уздовж жвавих

вулиць банери на підтримку акції. Некомерційні та бізнес-організації отримали можливість на рік взяти маяк в оренду. Технологія дозволяє повідомляти жителів району і туристів, які встановлять додаток *Hello Parkville*, наприклад, про страву дня в ресторані або знижки в місцевих магазинчиках. Також додаток повідомляє про розважальні події та інших заходах району;

- Петко-парк в Сан-Дієго і Доджер-Стедіум в Лос-Анджелесі - перші стадіони, оснащені технологією Bluetooth-маяків. Якщо користувач купує квиток на гру в мобільному додатку, маяки допомагають йому знайти своє місце в секторі, вказують на розташування туалетів, кафе і магазинів. Ресторани та магазини, в свою чергу, відправляють фанатам рекламні повідомлення та купони зі знижками. Оплатити покупку в фан-магазині завдяки iBeacon можна через PayPal. Також вболівальники, які зайдуть в музей стадіону, отримають замітки про експонати та історії команди.

Рисунок 5.8 Петко-парк в Сан_Дієго

У рекламі Bluetooth використовується повідомлення, яке містить додаткову інформацію в широкомовному UUID. Додаток на мобільному пристрої може відреагувати на це оголошення і виконати відповідну дію, якщо отримано правильне оголошення. У типовому випадку для роздрібу використовуватиметься мобільний додаток, який буде реагувати на присутність реклами маяка поблизу і відображати інформацію про рекламу або продаж на мобільному пристрої користувача. Мобільний пристрій буде зв'язуватися через Wi-Fi або стільниковий зв'язок, щоб отримати додатковий контент, а також надати важливі дані про ринок і покупця для компанії.

Маяки можуть передавати свою калібровану потужність сигналу RSSI у рекламному повідомленні. Сила сигналу маяків калібрується виробниками, як

правило, на відстань в один метр. Внутрішня навігація за допомогою маяків може виконуватися трьома способами:

- **кілька маяків в кімнаті** - це простий метод тріангуляції для визначення місця розташування користувача на основі оголошеної потужності сигналу RSSI, зібраної від численних маяків в кімнаті. З огляду на переданий в рекламному оголошенні калібрований і виміряний рівень сигналу від кожного маяка, алгоритм може визначити приблизне місце розташування приймача в кімнаті методом триангуляції. Це передбачає, що всі маяки знаходяться на фіксованих місцях і не змінюють своє положення;
- **один маяк в кімнаті** - за цією схемою в кожній кімнаті розміщується один маяк, що дозволяє користувачеві переміщатися між кімнатами і залами з точністю до місця, де знаходиться кімната. Це корисно в музеях, аеропортах, концертних залах і т.ін.;
- **кілька маяків на одну будівлю** - в поєднанні з акселерометром і гіроскопами в мобільному пристрої кілька маяків в будівлі можуть допускати здатність до орієнтації у великому відкритому просторі. Це дозволяє одному маяку встановлювати початкове місце розташування, а мобільного пристрою - визначати його місце розташування на основі руху користувача.

5.2.9 Максимальна відстань передачі між пристроями в Bluetooth 5 та збільшення швидкості

Bluetooth має різні рівні потужності, діапазони і потужність передачі, засновані на наступній класифікації кожного пристрою, як показано в таблиці нижче.

Таблиця 5.1 Класи Bluetooth

Клас пристрою	Максимальний вихідний рівень (дБп)	Максимальна вихідна потужність (мВт)	Максимальна дальність (м)	Використання
1	20	100	100	Адаптер USB, точки доступу

Клас пристрою	Максимальний вихідний рівень (дБп)	Максимальна вихідна потужність (мВт)	Максимальна дальність (м)	Використання
1,2	10	10	30 (зазвичай 5)	Маяки та носима на тілі електроніка
2	4	2,5	10	Мобільні пристрої, адаптери Bluetooth, зчитувачі смарт-карт
3	0	1	0,1	Адаптери Bluetooth

Bluetooth 5 збільшив максимальну відстань, а також швидкість передачі даних за межі застарілих обмежень Bluetooth. Нова радіостанція PHY доступна в Bluetooth 5 під назвою LE2M. Вона дозволяє подвоїти швидкість передачі необроблених даних Bluetooth з 1 Мсимвол/с до 2 Мсимвол/с. Для передачі однакової кількості даних по Bluetooth 5, на відміну від Bluetooth 4, потрібно менше часу. Це особливо актуально для пристроїв IIoT, що працюють на батарейках. Новий PHY також збільшує потужність передавача від +10 дБп до +20 дБп, що дозволяє збільшити дальність.

Що стосується дальності передачі, у Bluetooth 5 є ще одна опція PHY для передачі на більшу дальність в BLE. Цей допоміжний PHY має кодування LE. Цей PHY, який раніше використовував швидкість 1 Мсімвол/с, як і в Bluetooth 4.0, знижує швидкість передачі пакетів до 125 Кбіт/с або 500 Кбіт/с і збільшує потужність передачі на +20 дБп. Це призводить до збільшення максимальної відстані передачі у 4 рази в порівнянні з Bluetooth 4.0 і кращому проникненню в будівлі. Слід мати на увазі, що використання цієї опції LE-кодованого PHY при збільшенні відстані одночасно збільшує споживання енергії.

5.3 Чарункові (mesh) мережі на основі Bluetooth

5.3.1 Стек mesh-мережі Bluetooth 5

Bluetooth SIG опублікував профіль mesh-мережі, пристрій і специфікацію моделі 1.0 13 липня 2017 р. Це відбулося через шість місяців після випуску специфікації Bluetooth 5.0. До представлення офіційної специфікації Bluetooth 5 у Bluetooth SIG існували власні і спеціальні схеми, які використовують старіші

версії Bluetooth для створення mesh-середовищ. Три нові специфікації, що були опубліковані в 2017 р. Bluetooth SIG, такі:

- специфікація профілю Mesh 1.0 - визначає основні вимоги, що дозволяють інтегроване рішення mesh-мережі, описує функціонування та функціонал Mesh-мережі та формати повідомлень;
- специфікація моделі Mesh 1.0 - базова функціональність вузлів в mesh-мережі з відповідними форматами повідомлень;
- властивості пристрою Mesh 1.0 - визначає необхідні властивості пристрою (вузла) мережі, необхідні для задоволення умовам специфікації моделі mesh-мережі.

Mesh-мережа Bluetooth заснована на BLE і розташовується на фізичному і канальному рівнях BLE, описаних раніше. Поверх цих рівнів знаходиться стек рівнів, специфічних для mesh-мережі:

- **моделі** - реалізує поведінку, стани і прив'язки по одній або декільком специфікаціям моделі;
- **основні моделі** - настройка і управління mesh-мережею;
- **рівень доступу** - визначає формат даних програми, процес шифрування і перевірку даних;
- **верхній транспортний рівень** - управляє аутентифікацією, шифруванням і розшифровкою даних, переданих на рівень доступу і з нього. Транспортує керуючі повідомлення, такі як *дружба* і *серцебиття*;
- **нижній транспортний рівень** - виконує сегментацію і повторне збирання (*segmentation and reassembly - SAR*) фрагментованих блоків PDU, якщо це необхідно;
- **мережевий рівень** - визначає, який мережевий інтерфейс виводить повідомлення. Управляє різними типами адрес і підтримує різні види середовищ передачі;
- **рівень носія** - визначає, як обробляються PDU. Підтримуються два типи PDU: рекламний носій і носій GATT. Рекламний носій обробляє

передачу і прийом PDU mesh-мережі, в той час як носій GATT надає проксі-сервер для пристроїв, які не підтримують рекламні носії;

- **BLE** - повна специфікація Bluetooth LE.

Чарункова мережа Bluetooth може включати інші чарункові мережі або функції BLE. Пристрій, здатний підтримувати mesh-мережі і BLE, може зв'язуватися з іншими пристроями, такими як смартфони, або мати функції маяка. На рис. 5.9 показаний стек mesh-мережі Bluetooth. Важливо розуміти заміну стека вище канального рівня.

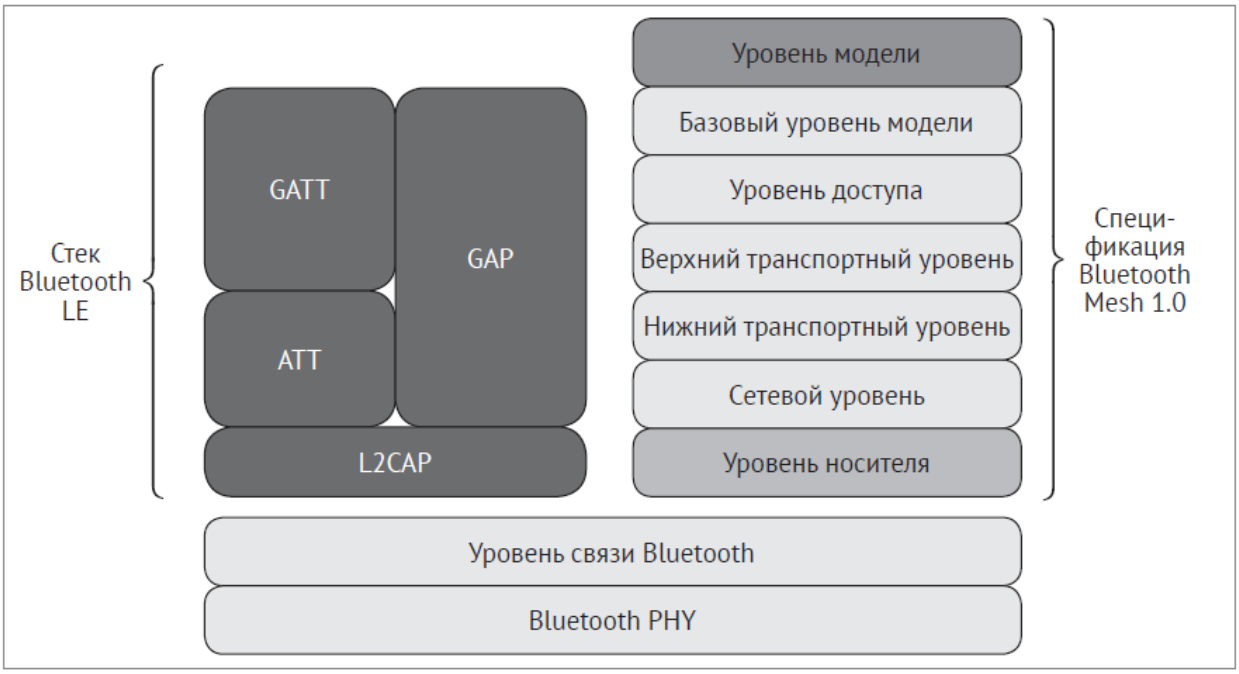


Рисунок 5.9 Специфікація 1.0 стека mesh-мережі Bluetooth

5.3.2 Топологія чарункової мережі Bluetooth

Mesh-мережа Bluetooth використовує концепцію *потопу*. У такій мережі кожен вхідний пакет, отриманий вузлом, ретранслюється далі через кожне вихідне з'єднання вузла, за винятком з'єднання з *джерелом* повідомлення. Перевага потопу полягає в тому, що, якщо пакет може бути доставлений, він буде доставлений (хоча, ймовірно, багато разів багатьма маршрутами). Пакет сам автоматично знайде найкоротший маршрут (який може змінюватися в залежності від якості сигналу і відстані в динамічній mesh-мережі). Такий алгоритм є найпростішим у реалізації протоколів маршрутизації. Крім того, він не вимагає центрального менеджера, такого як в мережі Wi-Fi, що заснована на центральному маршрутизаторі. Для порівняння, альтернативні типи

маршрутизації mesh-мережі включають в себе *алгоритми на основі дерева*. При використанні алгоритмів дерева (або алгоритмів кластерного дерева) необхідна наявність в мережі координатора для створення екземпляра нової підмережі і він стає для неї батьківським вузлом. Однак дерева не обов'язково є істинними mesh-мережами. Інші протоколи маршрутизації mesh-мереж включають *проактивну маршрутизацію*, яка підтримує сучасні таблиці маршрутизації на кожному вузлі і *реактивну маршрутизацію*, яка тільки оновлює таблиці маршрутизації на кожному вузлі на вимогу; наприклад, коли дані повинні відправлятися через вузол. Рисунок 5.10 ілюструє архітектуру потопу. Час прибуття на кожному рівні може динамічно змінюватися від вузла до вузла. Крім того, mesh-мережа повинна бути стійкою проти дублювання повідомлень, що надходять на будь-який вузол, як у випадку вузла 7 і вузла П.

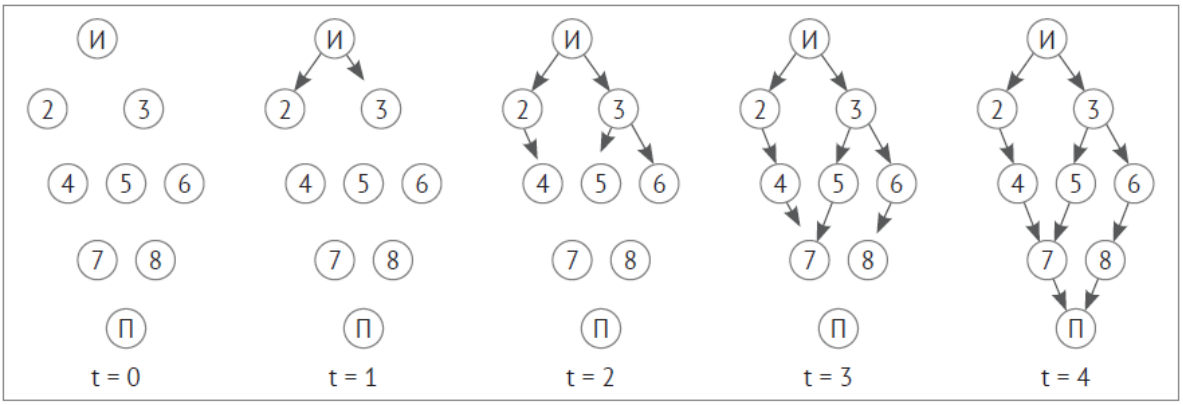


Рисунок 5.10 Архітектура потопу в mesh-мережі. И = Відправник, П = Одержувач. Відправник виробляє дані, які поширюються і проходять через кожен вузол в мережі

Основним недоліком мережі з потопом є пропускна здатність. Залежно від потоку від кожного вузла, перевантаження в mesh-мережі Bluetooth може бути значним. Інша проблема - атака "відмова в обслуговуванні". Якщо mesh-мережа просто відключає окремі зациклені повідомлення, необхідно знати, коли припиняти їх передачу. Bluetooth виконує це через ідентифікатори часу життя повідомлення, які ми розглянемо пізніше.

Вузли в mesh-мережі Bluetooth можуть бути таких типів:

вузли - це пристрої Bluetooth, які були попередньо підготовлені і є членами mesh-мережі;

непідготовлені пристрої - це пристрої, які фізично вже існують, можуть приєднатися до mesh-мережі, але які ще не є її частиною і не були підготовлені;

елементи вузла – пристрої, які входять до складу даного вузла. Кожен пристрій у вузлу може керуватися і адресуватися незалежно. Прикладом може служити вузол Bluetooth з давачами температури, вологості і освітленності. Це буде єдиний вузол (давач) з трьома елементами;

шлюз mesh-мережі - вузол, через який може провадитися з'єднання між mesh-мережею і усіма іншими технологіями (мережами).

Після підготовки вузол може підтримувати необов'язковий набір функцій, які включають до себе:

реле - вузол, що підтримує можливість ретрансляції повідомлень і може ретранслювати отримані повідомлення далі;

проксі - дозволяє пристроям Bluetooth LE, які від самого початку не підтримують mesh-мережі Bluetooth (наприклад 4.1, 4.0), взаємодіяти з вузлами в mesh-мережі. Така можливість надається шляхом використання проксі-вузла. Проксі-сервер вузла надає інтерфейс GATT для застарілого пристрою Bluetooth, і визначає проксі-протокол, заснований на каналі, орієнтованому на з'єднання. Успадкований пристрій зчитує і записує протокол проксі-сервера GATT, а проксі-вузол перетворює повідомлення від нього в справжні PDU mesh-мережі;

мала потужність - деякі вузли в mesh-мережі повинні мати надзвичайно низькі рівні енергоспоживання. Вони можуть обслуговувати інформацію від екологічних давачів (наприклад, температури) один раз на годину, а конфігуруватися за допомогою керуючого вузла або хмари один раз на рік. Цей тип пристрою не може бути поміщений в режим прослуховування, оскільки повідомлення на такий пристрій буде надходити тільки один раз на рік. Вузол входить в роль, звану вузлом низької потужності (*low power node - LPN*), котрий з'єднує його з дружнім вузлом. LPN переходить в стан глибокого сну, а коли просипається - просто опитує пов'язаного з ним друга на наявність будь-яких повідомлень, які могли з'явитися для нього під час сну;

друг - дружній вузол, пов'язаний з LPN, який не повинен обов'язково обмежувати свою потужність, як LPN. Друг може використовувати окреме живлення або енергію живлення від навколишнього середовища. Обов'язок друга полягає в тому, щоб зберігати і буферизувати повідомлення, призначені

для LPN, до тих пір, поки LPN не прокинеться і не опитає його на наявність нових повідомлень для нього. Багато повідомлень можуть бути збережені, і друг передасть їх в порядку, використовуючи прапор додаткових даних (*more data - MD*).

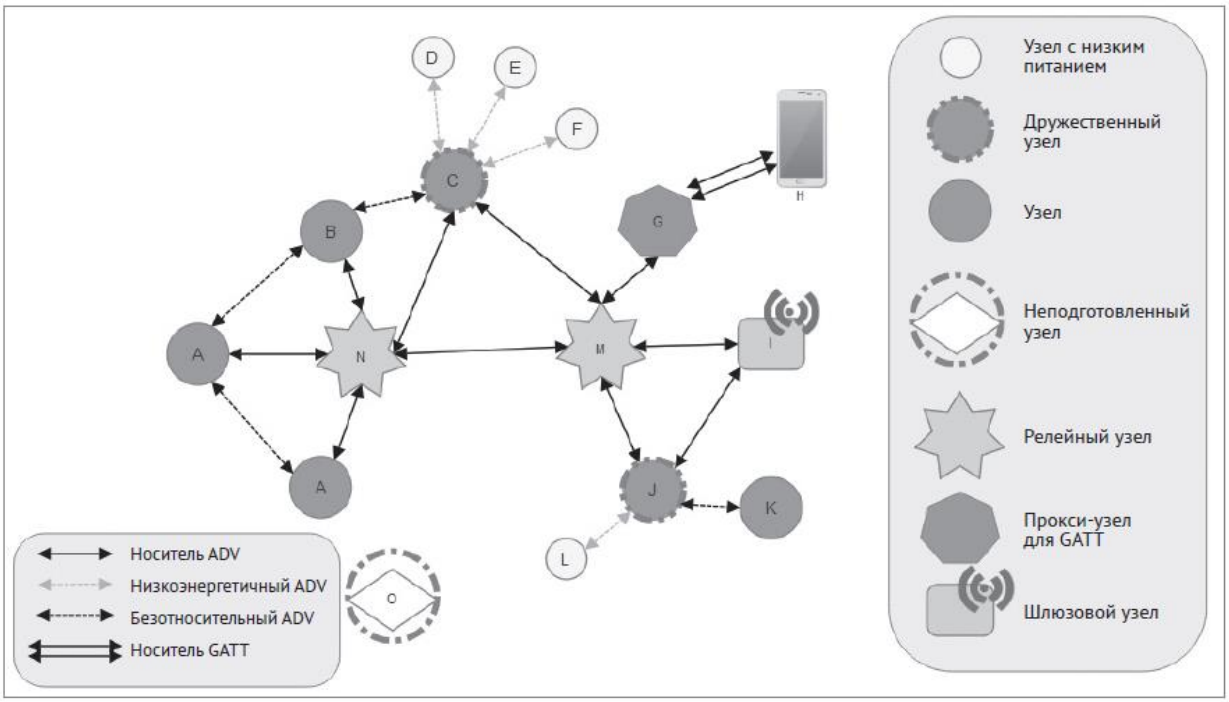


Рисунок 5.11 Топологія чарункової мережі Bluetooth. Зверніть увагу на класи, включаючи вузли, LPN, друзів, шлюзи, релейні та непідготовлені вузли

Рис. 5.11 ілюструє топологію мережі Bluetooth з різними компонентами, оскільки вони будуть пов'язані один з одним в реальну mesh-мережу.

Mesh-мережа Bluetooth буде кешувати повідомлення на кожному вузлі; це має вирішальне значення для мережі потопу. Оскільки одне й те саме повідомлення може надійти кілька разів з різних джерел, кеш забезпечує пошук останніх повідомлень, отриманих повідомлень і оброблених повідомлень. Якщо нове повідомлення ідентично одному з кешованих, воно відкидається. Це забезпечує ідемпотентність (властивість, яка проявляється в тому, що повторна її дія над будь-яким об'єктом уже не змінює результату) системи.

Кожне повідомлення містить *поле часу життя* (*time-to-live field - TTL*). Якщо повідомлення отримано вузлом і потім повторно передається, поле TTL цього повідомлення зменшується на одиницю. Цей механізм безпеки призначений для запобігання нескінченним циклам курсування повідомлень у мережі, які потенційно можуть створювати всередині мережі атаки типу «відмова в обслуговуванні».

Повідомлення про *серцебиття* періодично передається від кожного вузла в mesh-мережі. Серцебиття повідомляє середовищу, що вузол все ще присутній і здоровий. Він також дозволяє mesh-мережі знати, як далеко знаходиться вузол і чи змінилася відстань з моменту останнього серцебиття. По суті, він підраховує кількість переходів для досягнення вузла. Цей процес дозволяє mesh-мережі реорганізовуватися і самовідновлюватися.

5.3.3 Режими адресації mesh-мережі Bluetooth

Mesh-мережа Bluetooth використовує три види адресації:

- **одноадресна адресація** - унікально ідентифікує один елемент в mesh-мережі. Адреса елементу призначається в процесі підготовки пристрою при паруванні;
- **групова адресація** - це форма багатоадресної адресації, яка може представляти один або декілька елементів. Вони або зумовлені ідентифікатором Bluetooth SID як фіксовані групові адреси SIG, або призначені «на льоту»;
- **віртуальна адресація** - адрес може бути призначений більш ніж для одного вузла і декількох елементів. Віртуальна адресація використовує 128-бітний UUID. Передбачається, що UUID може бути налаштований виробником, щоб дозволити йому звертатися до свого продукту в усьому світі.

Протокол mesh-мережі Bluetooth починається з 384-байтових повідомлень, які сегментуються в 11-байтові посилки. Весь зв'язок в mesh-мережі Bluetooth орієнтований на повідомлення. Існують дві форми повідомлень, які можуть бути передані:

- **підтверджувані повідомлення** - для них потрібна відповідь від вузла (вузлів), які отримують таке повідомлення. Підтвердження також включає дані, які відправник запитує у свого адресата в повідомленні. Тому підтверджуване повідомлення служить двом цілям;
- **непідтверджувані повідомлення** - вони не вимагають відповіді від одержувача.

Відправлення повідомлення з вузла називається **публікацією**. Вузли, налаштовані для обробки обраних повідомлень, відправлених на певні адреси, називаються **передплатниками**. Кожне повідомлення зашифровано і аутентифікується з використанням мережевого ключа і ключа додатка.

Ключ додатка відноситься до програмового додатка або випадку використання (наприклад, ввімкнення світла в залежності від освітленності середовища). Вузли будуть публікувати події (світлові перемикачі), інші вузли будуть підписуватися на ці події (лампи і лампочки). На малюнку нижче показан приклад топології мережі Bluetooth. Тут вузли можуть підписуватися на кілька подій (освітлення передпокою і освітлення вітальні). Кола на малюнку являють собою групові адреси. Вимикач освітленності публікує події для відповідної групи.

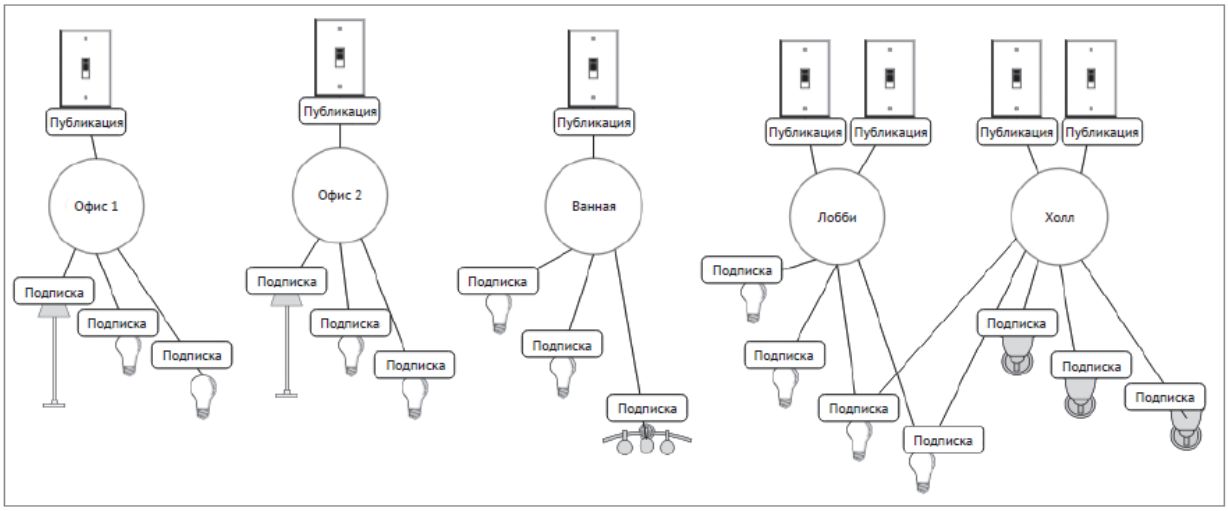


Рисунок 5.12 Модель публікація - передплатник mesh-мережі Bluetooth

Mesh-мережа Bluetooth являє собою концепцію групового обміну повідомленнями. У mesh-мережі у вас може бути зроблено групування схожих об'єктів, таких як освітлення у ванній або освітлення в передпокої. Це допомагає в зручності використання; наприклад, якщо додається нова освітлювальна арматура, тільки ці лампи повинні бути підготовлені, а інша частина mesh-мережі не потребує ніяких змін.

Вимикачі світла в попередньому прикладі мають два стани: включений і виключений. Mesh-мережа Bluetooth визначає стан, і в цьому випадку вони позначаються загальним станом *OnOff*. Загальні стани та шляхі сполучення підтримують багато типів пристроїв: від джерел світла до вентиляторів та приводів. Це швидкий спосіб повторного використання моделі для досягнення

спільної мети. Коли система переходить з одного стану в інший, це називається зміною стану об'єкта в mesh-мережі. Стани також можуть бути пов'язані один з одним. Якщо стан об'єкта змінюється, це може вплинути на його перехід в інший стан. Як приклад, mesh-мережа, що управляє стельовим вентилятором, може мати стан управління швидкістю, який, коли його значення досягає нуля, змінює загальний стан включення-виключення.

Властивості аналогічні станам, але мають числові, а не логічні значення. Наприклад, температурний давач може мати стан температури 8 для позначення і публікації восьмибітового значення температури. Властивість елемента/об'єкта може бути встановлена як виробником (тільки для читання) - постачальником елемента або адміністратором, який має доступ на читання і запис. І стани і властивості передаються через повідомлення в mesh-мережі. Повідомлення бувають трьох типів:

- **запитати** - запитує значення заданого стану з одного або декількох вузлів;
- **встановити** - змінює значення стану;
- **статус** - є відповідь на запит, що містить дані.

Всі ці концепції станів і властивостей в кінцевому підсумку утворюють модель (найвищий рівень стека mesh-мережі Bluetooth). Модель може бути *сервером*, і в цьому випадку вона може визначати стани і перехід між ними. Модель може бути і *клієнтом*, який не визначає будь-які стани; скоріше, він визначає повідомлення взаємодії зі станом для використання із *запитом, встановленням і статусом*. Модель керування усією мережею може підтримувати гібрид моделей серверів і клієнтів.

Mesh-мережа також може використовувати стандартні функції Bluetooth 5, такі як анонімні рекламні оголошення і безліч рекламних наборів. Візьмемо, як приклад, випадок з mesh-мережею, що з'єднує телефон для голосового зв'язку з гарнітурою, але одночасно передає пакети і для інших цілей. Використовуючи кілька наборів оголошень, обидва варіанти використання можуть використовуватися в керуванні мережею одночасно.

5.3.4 Підготовка пристрою до роботи вузлом mesh-мережі Bluetooth

Вузол може приєднуватися до mesh-мережі за допомогою дії ініціалізації. *Підготовка* - це безпечний процес, який приймає непідготовлений і небезпечний пристрій і перетворює його в безпечний вузол mesh-мережі. Спочатку вузол буде захищено **NetKey мережевим ключем**, отриманим від mesh-мережі. Принаймні, один ключ NetKey повинен знаходитися на кожному пристрої в mesh-мережі для з'єднання. Пристрої додаються в mesh-мережу через процедуру підготовки. Пристрій, що підготовлює, розподіляє мережевий ключ і унікальну адресу на непідготовлений пристрій.

В процесі ініціалізації використовується обмін між пристроями ключами Діффі-Хеллмана для створення тимчасового ключа шифрування мережевого ключа. Це забезпечує безпеку від атаки «людина-в-середині» під час підготовки. Ключ пристрою, отриманий за допомогою тимчасового ключа, використовується для шифрування повідомлень, відправлених пристроєм, що проводить підготовку пристрою по включенню до мережі.

Процес підготовки наступний:

1) непідготовлений пристрій транслює рекламний пакет маяка;

2) пристрій, що підготовлює, надсилає запрошення на перший пристрій. Непідготовлений пристрій у відповідь відсилає свої характеристики в пакетах PDU для забезпечення доступу;

3) пристрій, що підготовлює і непідготовлений пристрої обмінюються відкритими ключами;

4) непідготовлений пристрій виводить довільне число користувачеві. Користувач вводить цифри (або ідентифікатор) в пристрій, що підготовлює і криптографічний обмін починає завершувати фазу аутентифікації;

5) ключ сеансу обраховується кожним з двох пристроїв на основі закритого ключа і обмінних відкритих ключів. Ключ сеансу використовується для захисту даних, необхідних для завершення процесу підготовки, включаючи забезпечення безпеки пересилання ключа NetKey на пристрій, що підготовлюється;

6) пристрій змінює свій стан з *непідготовленого* на *вузол* і тепер володіє мережевим ключем NetKey, своєю адресою в мережі і параметром безпеки mesh-мережі, так званим індексом IV.

Контрольні питання

1. Дати визначення наступним типам вузлів: публікатор, сканер, ініціатор. Навести приклади застосування.
2. Чим розширена публікація відмінна від простої і періодичної публікації. В яких випадках застосовуються ці види публікації?
3. Що таке пікомережі і які види пікомереж з Bluetooth вам відомі? Наевсти характеристики.
4. Розповісти про різницю між класичними та BLE пікомережами з Bluetooth.
5. Яка роль ведучого, веденого та резервного вузла у пікомережі?
6. З чого складається стек Bluetooth 5.0? Що таке протоколи і профілі?
7. Розповісти про три режими роботи Bluetooth. Коли слід перемикатися в який режим?
8. Які стани зв'язку існують у режимі BLE? Як вони взаємодіють один з одним?
9. Як виконується підключення нового вузла до пікомережі?
10. Які засоби використовує режим BLE Bluetooth 5.0 для забезпечення безпеки з'єднання?
11. Що таке режим маяка у пристрої IoT?
12. Які характеристики має стандарт Bluetooth 5.0? (сила сигнала, максимальна дальність, тощо)
13. Що таке чарункова мережа, її характеристики?
14. Розповісти про топологію чарункової мережі, як вона працює?
15. Що таке архітектура потопу?
16. Які режими адресації існують у чарунковій мережі Bluetooth?
17. Як підготувати пристрій до роботи у чарунковій мережі?

6 ПЕРСОНАЛЬНІ БЕЗДРОТОВІ МЕРЕЖІ НА IP ОСНОВІ

6.1 Бездротові локальні мережі стандарту 802.11 (WiFi)

Бездротові локальні мережі стандарту 802.11 використовуються у промисловій автоматизації при створенні сенсорних мереж в основному завдяки тому, що вони прості в розгортанні і зручні в зксплуатації [4][20]. З точки зору користувача їх функції та характеристики такі ж, як і у локальних дротових мережах Ethernet. Станції стандарту 802.11 не володіють здатністю виявляти колізії, як це роблять Ethernet-станції, які здійснюють множинний доступ до мережі з контролем несучої і виявленням колізії. Внаслідок цього для доступу до середовища необхідний більш складний і масштабований підрівень MAC при мінімальних додаткових витратах.

6.1.1 Різновиди використовуваних стандартів WiFi

При створенні сенсорних мереж у данний час поширення отримали наступні стандарти IEEE 802.11:

IEEE 802.11g — стандарт бездротових локальних мереж, заснований на бездротовій передачі даних в діапазоні 2,4 ГГц. Діапазон поділений на три непересічні канали, тобто на одній території, не впливаючи одна на одну, можуть працювати три різні бездротові мережі. Для збільшення швидкості обміну даними при ширині каналу, схожій з попереднім стандартом 802.11b, застосований метод модуляції з *ортогональним частотним мультиплексуванням* (OFDM*, Ortogonal Frequency Division Multiplexing*), а також метод *двійкового пакетного згорткового кодування* PBCC (*Packet Binary Convolutional Coding*). Швидкість передачі до 108 Мб/с (до 54 Мб/с у кожну сторону). У даний момент важається застарілим стандартом, але все ще використовується.

IEEE 802.11n — сучасний стандарт бездротових локальних мереж, заснований на бездротовій передачі даних в діапазонах 2,4 та 5,0 ГГц. Теоретично 802.11n здатний забезпечити швидкість передачі даних до 600 Мб/с, застосовуючи передачу даних MIMO одразу чотирма антенами. За однієї антени - до 150 Мбіт/с (до 75 Мб/с в одну сторону).

Крім того, пристрої 802.11n можуть працювати в трьох режимах:

- *успадковане (Legacy)*, в якому забезпечується підтримка пристроїв 802.11b/g і 802.11a;
- *змішаному (Mixed)*, в якому підтримуються одночасно пристрої 802.11b/g, 802.11a і 802.11n;
- *«чистому» режимі - 802.11n* (саме в цьому режимі і можна скористатися перевагами підвищеної швидкості і збільшеною дальністю передачі даних, що забезпечуються стандартом 802.11n).

IEEE 802.11ah - це протокол бездротової мережі, опублікований в 2017 році і названий *Wi-Fi HaLow* (вимовляється як «Хей-лоу»), розроблений як додаток до стандарту бездротової мережі IEEE 802.11. Цей протокол працює на частоті 900 МГц, що не вимагає ліцензування, для забезпечення розширеного діапазону Wi-Fi мереж, в порівнянні зі звичайними мережами Wi-Fi, що працюють в діапазонах 2,4 ГГц і 5 ГГц. Його низьке енергоспоживання є перевагою, яка дозволяє створювати великі групи станцій або давачів, що взаємодіють один з одним, щоб поширювати сигнал (як в чарункових мережах), підтримуючи концепцію Інтернету речей. Низький рівень споживання енергії протоколу конкурує з Bluetooth і має додаткову перевагу - більш високі швидкості передачі даних (до 340 Мб/с) і більш широкий діапазон покриття чарунковою мережею з ретрансляторами.

IEEE 802.11p — затверджена поправка до стандарту IEEE 802.11, метою якої є надання можливості використання бездротового зв'язку в рухомих середовищах (наприклад, автомобілі, потязі, тощо). Це включає передачу даних між рухомими на великій швидкості транспортними засобами, а також транспортних засобів із придорожньою інфраструктурою на ліцензованій частоті 5.9 GHz (5.85-5.925 GHz).

IEEE 802.11e (QoS, *Quality of service*) — додатковий стандарт, що дозволяє забезпечити гарантовану якість обміну даними шляхом перестановки пріоритетів різних пакетів; необхідний для роботи таких потокових сервісів як VoIP або IPTV.

IEEE 802.11i — стандарт, що знімає недоліки у сфері безпеки попередніх стандартів. 802.11i вирішує проблеми захисту даних канального рівня і дозволяє створювати безпечні сенсорні мережі практично будь-якого масштабу.

6.1.2 Огляд використовуваних раніше топологій WiFi

Мережі стандарту 802.11 можна конструювати по-різному. Розробник може вибрати будь-яку з наступних топологій:

- **Незалежні базові зони обслуговування** (*Ad-Hoc, IBSS*).
- **Базові зони обслуговування** (*BSS*).
- **Розширені зони обслуговування** (*ESS*).

Зона обслуговування в даному випадку - це логічно згруповані пристрої. Технологія WLAN забезпечує доступ до мережі шляхом передачі широкомовних сигналів через ефір на несучій в діапазоні радіочастот. Приймаюча станція може отримувати сигнали в діапазоні роботи декількох передавальних станцій. Передавальна станція спочатку передає ідентифікатор зони обслуговування SSID. Станція-приймач використовує SSID для фільтрації одержуваних сигналів і виділення того, який їй потрібен.

6.1.2.1 Незалежні базові зони обслуговування (IBSS)

IBSS є групою станцій, що працюють відповідно до стандарту 802.11, та зв'язуються безпосередньо одна з одною (*одноранговa мережа)*. На **Ошибка! Источник ссылки не найден.** показано, як три станції, обладнані бездротовими мережевими інтерфейсними картами (NIC) стандарту 802.11, можуть формувати IBSS і безпосередньо зв'язуватися одна з одною.

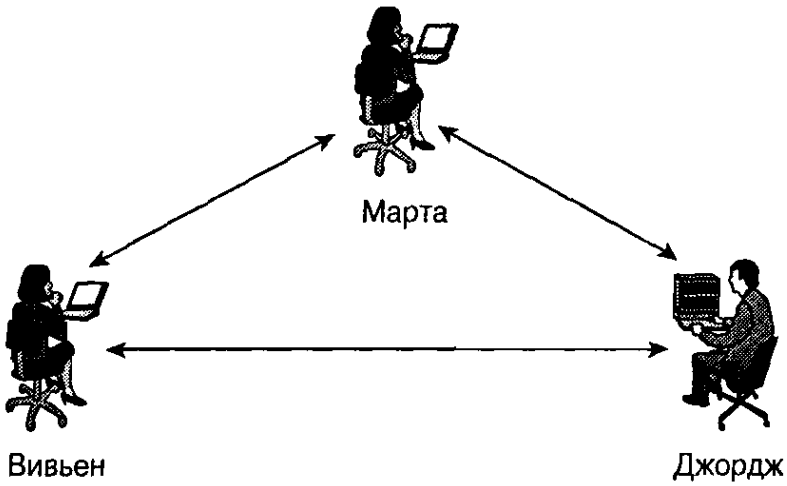


Рисунок 6.1 Непланова (ad-hoc) мережа

Спеціальна мережа, або незалежна базова зона обслуговування (IBSS), виникає, коли окремі пристрої-клієнти формують самопідтримуючу мережу без окремої точки доступу. При створенні таких мереж не розроблюються будь-які мапи місця їх розвернення і попередні плани, тому вони зазвичай невеликі і мають обмежену протяжність, достатню для передачі спільно використовуваних даних при виникненні такої необхідності. На відміну від варіанту використання розширеної зони обслуговування (ESS), клієнти безпосередньо встановлюють з'єднання один з одним, в результаті чого створюється тільки одна базова зона обслуговування (BSS), яка не має інтерфейсу для підключення до дротової локальної мережі (тобто відсутня будь-яка розподільча система, яка необхідна для об'єднання декількох BSS в ESS). Не існує будь-яких обумовлених стандартом обмежень на кількість пристроїв, котрі можуть входити в одну незалежну базову зону обслуговування. Але, оскільки кожен пристрій є клієнтом, часто певна кількість членів IBSS не може зв'язатися одне з одним внаслідок проблеми прихованого вузла.

Проблема прихованого вузла виникає, коли два або кілька вузлів мережі (абонентів) намагаються отримати доступ до іншого вузла або базової станції (точки доступу) мережі, але при цьому не бачать один одного, тобто фізично не можуть приймати сигнали в ефірі один від одного (наприклад, з -за великої дальності, умов поширення сигналів і т. д.). Це призводить до проблем з Управлінням Доступом в ефірі (media access control, MAC), так як більшість існуючих способів доступу в цифрові мережі з боку абонентів цієї мережі використовують визначення зайнятості каналів шляхом прослуховування сигналів від сусідніх абонентів.

Для прикладу візьмемо топологію Зірка, утворену точкою доступу (Access Point) з декількома вузлами, що перебувають на окружності, радіус якої близький до максимально досяжної дальності дії для даного виду зв'язку. Тоді, навіть якщо всі вузли виявляться в поле дії зв'язку з точкою доступу, не всі з них зможуть приймати сигнал один від одного. Наприклад, вузол, що знаходиться на одній стороні кола, швидше за все зможе відстежувати сигнал від вузла, розташованого

на тій же стороні кола, але навряд чи цей же вузол буде приймати сигнал від вузла, що знаходиться на протилежному боці кола, на відстані *2R*, де *R* - радіус кола. Ці вузли друг для друга будуть прихованими. Через те, що приховані вузли не відстежують сигнали один одного, вони можуть почати посилати пакети в точку доступу одночасно. Описана вище ситуація може призводити до неможливості встановлення зв'язку або обриву вже встановленого зв'язку.

В IBSS не існує будь-якого механізму для реалізації функції ретрансляції.

Оскільки в IBSS відсутня точка доступу, розподіл часу здійснюється нецентралізовано. Клієнт, який розпочинає передачу в IBSS, задає сигнальний (його ще називають маячковий) інтервал для створення набору моментів часу передачі маячкових сигналів (TBTT). Коли завершується час TBTT, кожний клієнт IBSS виконує наступне:

• припиняє всі, що ще не спрацювали таймери затримки з попереднього TBTT;

• визначає нову випадкову затримку;

• якщо маячковий сигнал надходить до закінчення випадкової затримки, відновлює роботу призупинених таймерів затримки. Якщо ніякій маячковий сигнал не надходить до закінчення випадкової затримки, відправляє маячковий сигнал і відновлює роботу призупинених раніше таймерів затримки.

IBSS не використовується для сенсорних мереж.

6.1.2.2 Базові зони обслуговування (BSS)

BSS - це група працюючих за стандартом 802.11 станцій, що зв'язуються одна з одною через так звану *точку доступу (Hot-Spot)*. Технологія BSS передбачає наявність особливої станції, яка називається точкою доступу (*acces point*). Точка доступу - це центральний пункт зв'язку для всіх станцій BSS. Клієнтські станції не зв'язуються безпосередньо одна з одною. Замість цього вони зв'язуються з точкою доступу, а вже вона спрямовує фрейми станції-адресату. Точка доступу може мати порт висхідного каналу (*uplink port*), через який BSS підключається до дротової мережі (наприклад, висхідний канал

Ehernet). Тому BSS іноді називають інфраструктурою BSS. На **Ошибка! И сточник ссылки не найден.** представлена типова інфраструктура BSS.

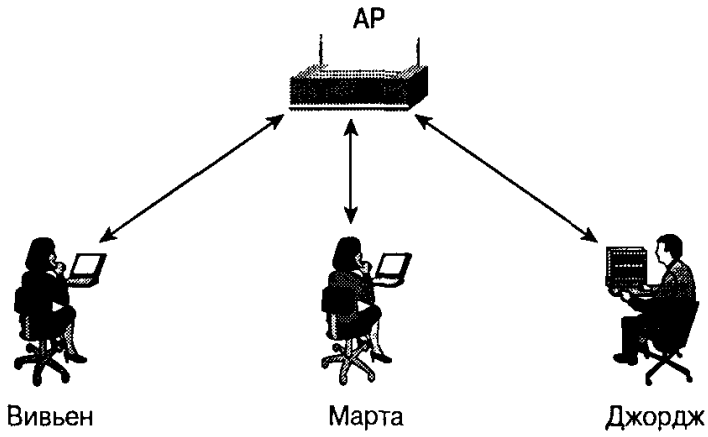


Рисунок 6.2 Інфраструктура безпровідної локальної мережі BSS

BSS має використання у сенсорних мережах.

6.1.2.3 Розширені зони обслуговування (ESS)

Кілька інфраструктур BSS можуть бути з'єднані через їх інтерфейси висхідного каналу. Там, де діє стандарт 802.11, інтерфейс висхідного каналу з'єднує BSS з розподільчою системою (*distribution system*, DS). Кілька BSS, з'єднаних між собою через розподільчу систему, утворюють розширену зону обслуговування (ESS). Висхідний канал до розподільчої системі не обов'язково повинен використовувати дротове з'єднання. На **Ошибка! И сточник ссылки не найден.** представлений приклад практичного втілення ESS. Специфікація стандарту 802.11 залишає можливість реалізації цього каналу у вигляді бездротового. Але частіше висхідні канали до розподільчої системи є канали дротового Ehernet.

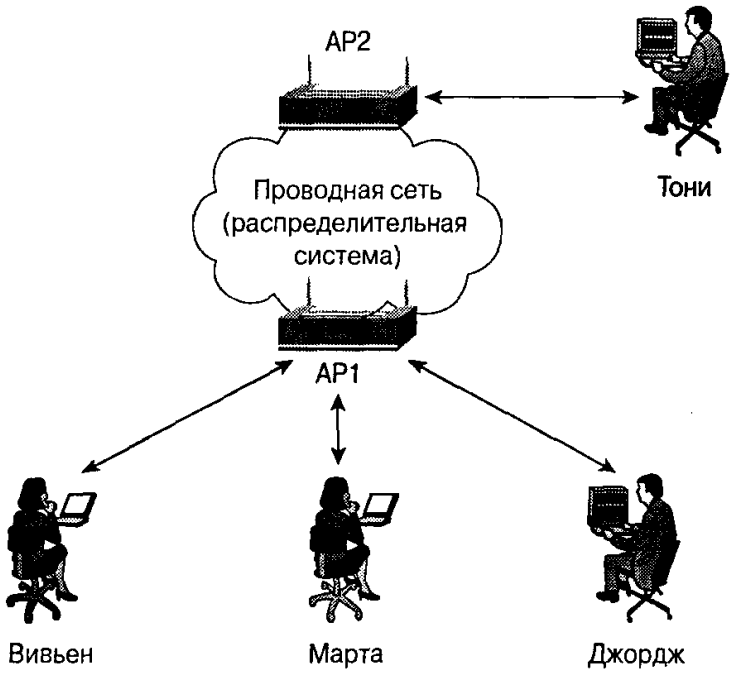


Рисунок 6.3 Розширена зона обслуговування ESS безпровідної локальної мере

6.1.3 Захист в мережі WiFi

Пристрої стандарту 802.11 зв'язуються один з одним, використовуючи в якості переносника даних сигнали, що передаються в діапазоні радіочастот. Дані передаються по радіо відправником, який вважає, що приймач також працює в обраному радіодіапазоні. Недоліком такого механізму є те, що будь-яка інша станція, яка використовує цей діапазон, теж здатна прийняти ці дані.

Якщо не використовувати механізм захисту, будь-яка станція стандарту 802.11 зможе обробити дані, послані через бездротову локальну мережу, якщо тільки її приймач працює в тому ж радіодіапазоні. Для забезпечення хоча б мінімального рівня безпеки необхідна наявність наступних компонентів:

• засобів для прийняття рішення щодо того, хто або що може використовувати бездротову LAN. Ця вимога задовольняється за рахунок механізму аутентифікації, що забезпечує контроль доступу до WLAN;

• засобів захисту інформації, переданої через бездротове середовище. Ця вимога задовольняється за рахунок використання алгоритмів шифрування.

На **Ошибка! Источник ссылки не найден.** показано, що захист в б ездротових мережах забезпечується як за рахунок аутентифікації, так і завдяки шифруванню. Жоден з названих механізмів окремо не здатний забезпечити захист бездротової мережі.

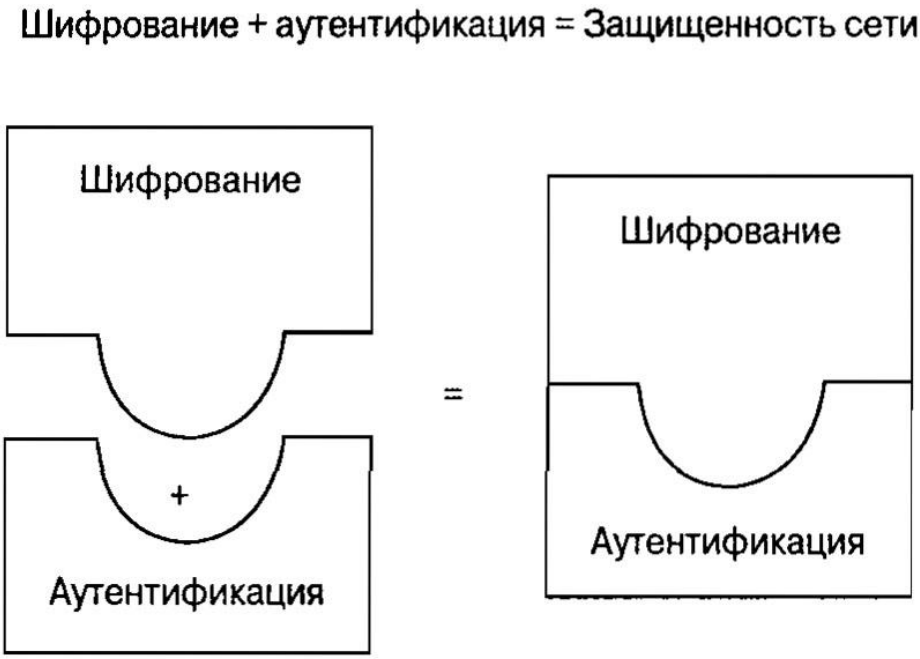


Рисунок 6.4 Захист в WiFi мережах забезпечується за рахунок аутентифікації та шифрування

Як було сказано раніше в п. 6.1.2 існує два основні варіанти пристрою бездротової мережі:

Ad-hoc - передача безпосередньо між пристроями;

Hot-spot - передача здійснюється через точку доступу.

У *Hot-spot* мережах присутня точка доступу, за допомогою якої відбувається не тільки взаємодія всередині мережі, а й доступ до зовнішніх мереж. *Hot-spot* становить найбільший інтерес з точки зору захисту інформації, так як, зламавши точку доступу, зловмисник може отримати інформацію не тільки зі станцій, розміщених в даній бездротовій мережі.

Загрози інформаційної безпеки, що виникають при використанні WiFi мереж, можна умовно розділити на два класи:

прямі - загрози інформаційній безпеці, що виникають при передачі інформації по бездротовому інтерфейсу IEEE 802.11;

опосередковані (непрямі)- загрози, пов'язані з наявністю на об'єкті і поруч з об'єктом великої кількості інших WiFi-мереж, що маскують собою наявність зловмисника.

Різновиди загроз WiFi мереж з їх типовим місцем розташування наведено далі на **Ошибка! Источник ссылки не найден.**

6.1.3.1 Прямі загрози

Радіоканал передачі даних, який використовується в WiFi, потенційно схильний до втручання з метою порушення конфіденційності, цілісності та доступності інформації.

У WiFi передбачені як аутентифікація, так і шифрування, але ці елементи захисту мають свої вади. Шифрування значно знижує швидкість передачі даних, і воно може свідомо відключатися адміністратором для оптимізації трафіку. Початковий стандарт шифрування *WEP (Wired Equivalent Privacy)* був у свій час (років 15 назад) дискредитований за рахунок вразливостей в алгоритмі розподілу ключів RC4. Це у свій час пригальмувало розвиток Wi-Fi ринку і викликало створення інститутом IEEE робочої групи **802.11i** для розробки нового стандарту, що враховує уразливості WEP таким чином, щоб забезпечувати 128-бітове шифрування і аутентифікацію для захисту даних. Wi-Fi Alliance в 2003

році представив свій власний проміжний варіант цього стандарту - WPA (*Wi-Fi Protected Access*). WPA використовує протокол цілісності тимчасових ключів TKIP (*Temporal Key Integrity Protocol*). Також в ньому використовується метод контрольної суми MIC (*Message Integrity Code*), що дозволяє перевіряти цілісність пакетів. У 2004 Wi-Fi Alliance випустив стандарт WPA2, який являє собою поліпшений WPA. Основна відмінність між WPA і WPA2 полягає в технології шифрування: відповідно TKIP і AES. WPA2 забезпечує більш високий рівень захисту мережі, так як TKIP дозволяє створювати ключі довжиною до 128 біт, а AES - до 256 біт.

Загроза блокування інформації в каналі Wi-Fi практично залишена без уваги при розробці технології. Само по собі блокування каналу не є небезпечним для офісних мереж, так як зазвичай WiFi мережі в них є допоміжними, проте блокування може являти собою лише підготовчий етап для атаки «людина посередині», коли між клієнтом і точкою доступу з'являється третій пристрій, який пропускає трафік між ними через себе. Таке втручання дозволяє видаляти, спотворювати або нав'язувати неправдиву інформацію.

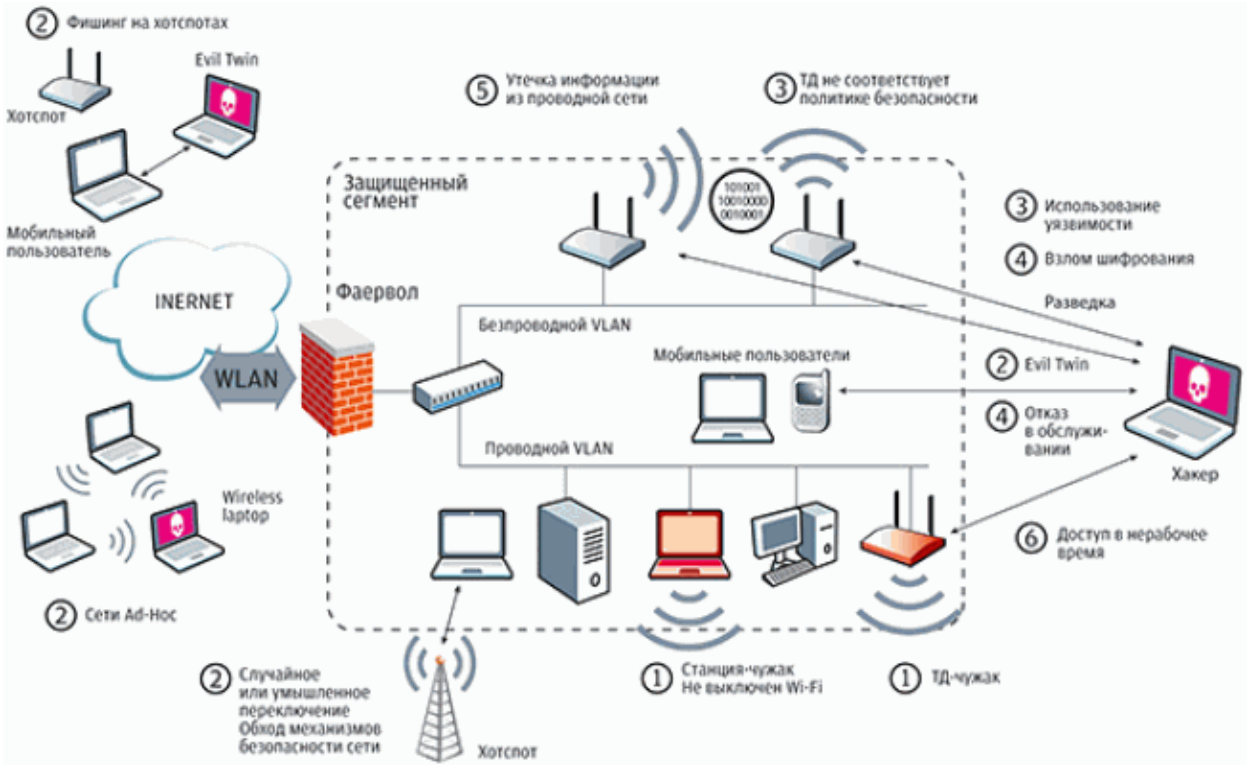


Рисунок 6.5 Різновиди загроз у WiFi мережах

Для сенсорних мереж блокування може виявитися катастрофічним, оскільки ці мережі передають дані з/на давачів для систем керування технологічними процесами.

Чужинцями (*RogueDevices, Rogues*) називаються пристрої, які дають змогу несанкціонованого доступу до мережі, зазвичай в обхід механізмів захисту, визначених політикою безпеки. Заборона на використання пристроїв бездротового зв'язку не захистить від бездротових атак, якщо в мережі, навмисне чи ні, з'явиться чужинець. У ролі чужинця може виступати все, у чого є дротовий і бездротовий інтерфейси: точки доступу (включаючи програмні), сканери, проектори, ноутбуки з обома включеними інтерфейсами і т. д.

Бездротові пристрої можуть змінювати точки підключення до мережі прямо в процесі роботи. Наприклад, можуть відбуватися «випадкові асоціації», коли ноутбук з Windows (що є довірителем до всіх бездротових мереж) або просто некоректно конфігурований бездротовий клієнт автоматично асоціюється і підключає користувача до найближчої бездротової мережі. Таким чином порушник перемикає на себе користувача для подальшого сканування вразливостей, фішингу або атак «людина посередині». А якщо користувач при цьому підключений і до провідної мережі, то він стає точкою входу - чужинцем. До того ж багато користувачів, які підключені до внутрішньої мережі і мають WiFi інтерфейс, незадоволені якістю і політикою роботи мережі, переключаються на найближчу доступну точку доступу (або операційна система робить це автоматично при відмові провідної мережі). При цьому увесь захист мережі терпить крах.

Ще одна проблема - мережі Ad-Hoc, за допомогою яких зручно передавати файли колегам або друкувати на принтері з WiFi. Але така організація мереж не підтримує багато методів забезпечення безпеки, що робить їх легкою здобиччю для порушника. Нові технології *Virtual WiFi* і *WiFi Direct* тільки погіршили ситуацію.

Некоректно сконфігуровані пристрої, пристрої зі слабкими і недостатньо довгими ключами шифрування, які використовують уразливі методи аутентифікації - саме такі пристрої піддаються атакам в першу чергу. Відповідно до звітів аналітиків, більша частина успішних зломів відбувається якраз через неправильні налаштування точок доступу і програмного забезпечення клієнта. Досить підключити неправильно налаштовану точку доступу до загальної мережі для злому останньої. Налаштування «за замовчуванням» не включають

шифрування і аутентифікацію, або використовують ключі, прописані в керівництві користувача і тому всім заздалегідь відомі. Малоймовірно, що користувачі досить серйозно будуть стурбовані безпечною конфігурацією власних пристроїв. Саме такі привнесені точки доступу і створюють основні загрози захищеним мережам. Про захищенність WEP і мови вже немає. Інтернет повний спеціального і зручного у використанні програмного забезпечення для злому цього стандарту, що збирає статистику трафіку до тих пір, поки її не стане достатньо для відновлення ключа шифрування. Стандарти WPA і WPA2 також мають ряд вразливостей різного ступеня небезпеки, що дозволяють їх злом. Тим ні менш вже відомі атаки на WPA2-Enterprise (802.1x). *KrackAttack* був опублікований в жовтні 2017 двома бельгійськими фахівцями в сфері інформатики. Вони відкрили цю уразливість WPA-2 ще в 2016 році.

6.1.3.2 Опосередковані загрози

Сигнали WiFi-пристроїв мають досить складну структуру і широкий спектр, тому ці сигнали, а тим більше, оточуючі пристрої WiFi неможливо ідентифікувати звичайними засобами радіомоніторингу. Впевнене виявлення сигналу WiFi сучасними комплексами радіомоніторингу в широкій смузі частот можливо тільки за енергетичною ознакою при одночасному паралельному аналізі смуг шириною кілька десятків МГц на швидкості не менше 400 МГц/с і лише в ближній зоні. Сигнали точок доступу, що знаходяться в далекій зоні, мають рівень сигналу нижчий рівня шумів приймача. Виявлення WiFi-передавачів при послідовному скануванні вузькосмуговими приймачами взагалі неможливо.

Виходячи з того, що практично кожен об'єкт оточує безліч «чужих» WiFi мереж, відрізнити легальних клієнтів своєї мережі і сусідніх мереж від порушників вкрай складно, що дозволяє успішно маскувати несанкціоновану передачу інформації серед легальних WiFi-каналів.

WiFi-передавач випромінює так званий «OFDM сигнал». Це означає, що в кожен момент часу пристрій передає сигнал, що займає широку смугу частот (близько 20 МГц) за рахунок наявності декількох несучих інформацію частот - *піднесучих інформаційних каналів*, які розташовані так близько один від одного,

що при прийомі їх на звичайному приймальному пристрої, сигнал виглядає як єдиний «купол». Виділити в такому «куполі» піднесучі і ідентифікувати передавальні пристрої можна тільки спеціальним приймачем-сканером.

Пропускна здатність WiFi мереж дозволяє передавати звук і відео в реальному часі. Це спрощує зловмисникові використання акустичних і оптичних каналів витоку інформації - досить легально купити WiFi-відеокамеру і встановити її в якості пристрою негласного отримання інформації.

Приклади:

З WiFi відеокамери з мікрофоном інформація передається на точку доступу, що працює в режимі ретранслятора (репітера). Точка розташована на даху і має спрямовану антену - таким чином можна значно збільшити дальність передачі сигналу - до декількох кілометрів. Сам сигнал приймається на контрольному пункті.

Смартфон співробітника за допомогою вірусу записує навколишній звук і передає його зловмисникові за допомогою WiFi. В якості контрольного пункту використовується точка доступу з прихованим ім'ям SSID, щоб виявити її було важче.

Якщо на об'єкті обмежений винос носіїв інформації і вихід в Інтернет обмежений, то одним з варіантів прихованої передачі великого обсягу інформації є WiFi. Потрібно підключитися до об'єктових WiFi мереж, залишаючись непоміченим серед легальних користувачів.

6.1.3.3 Особливості функціонування WiFi мереж

У бездротових мережах WiFi наявні деякі особливості, що відсутні в дротових мережах. Ці особливості в цілому впливають на продуктивність, безпеку, доступність і вартість експлуатації бездротової мережі. Їх доводиться враховувати, хоча вони і не мають прямого відношення до шифрування або аутентифікації. Для вирішення цих питань потрібен спеціальний інструментарій і механізми адміністрування і моніторингу.

Активність в неробочий час. Характерно для офісних мереж. Виходячи з того, що політикою безпеки логічно обмежити доступ до мережі поза робочим

часом (аж до фізичного відключення), бездротова активність мережі в неробочий час має відстежуватися, вважатися підозрілою і підлягати розслідуванню. Для сенсорних мереж взяти до уваги цей факт неможливо.

Швидкість підключення. Швидкість підключення залежить від співвідношення сигнал / шум (SNR). Якщо, скажімо, 54 Мбіт/с вимагає SNR в 25 dB, а 2 Мбіт/с вимагає всього 6 dB, то кадри, відправлені на швидкості 2 Мбіт/с «пролетять» далі, тобто їх можна декодувати з більшої відстані, ніж більш швидкісні кадри. Маємо на увазі, що всі службові кадри, а також бродкасти, відправляються на найнижчій швидкості. Це означає, що мережу буде видно на значній відстані. Якщо в мережі, де всі працюють на певній швидкості (офіс територіально обмежений і швидкості підключення у користувачів приблизно однакові) з'являється підключення на 1-2 Мбіт/с - швидше за все це порушник. Також можна відключити низькі швидкості, тим самим підвищивши швидкість передачі інформації в мережі.

Інтерференція. Якість роботи WiFi мережі (що використовує радіоефір) залежить від багатьох факторів. Один з них - інтерференція радіосигналів, яка може значно знизити пропускну здатність мережі і кількість користувачів, аж до повної неможливості використання мережі. Як джерело завад може виступати будь-який пристрій, що випромінює на тій же частоті сигнал достатньої потужності. Це можуть бути як сусідні точки доступу, так і мікрохвильовки. Цю особливість можуть також використовувати зловмисники як атаки відмови в обслуговуванні, або для підготовки атаки «людина посередині», заглушаючи легітимні точки доступу і залишаючи свою з таким же SSID.

Зв'язок. Існують і інші особливості бездротових мереж крім інтерференції. Неправильно налаштований клієнт або збійна антена можуть погіршити якість обслуговування всіх інших користувачів. Слід враховувати ще один момент: не тільки сигнал точки доступу повинен досягти клієнта, а й сигнал клієнта повинен досягти точки доступу. Зазвичай точки потужніші, і щоб домогтися симетрії, можливо доведеться знизити потужність сигналу. Для діапазону 5 ГГц слід пам'ятати, що надійно в ньому працюють тільки 4 канали: 36/40/44/48 (для Європи, для США є ще 5). На інших ввімкнено режим співіснування з радарами (DFS), що може призвести до періодичного зникнення зв'язку.

Робота в режимі енергозбереження. Щоб продовжити термін служби батарей давачів (станцій) в сенсорних мережах, стандарт 802.11 передбачає їх роботу в режимі зниженого енергоспоживання, або режимі енергозбереження. Робота в режимі зниженого енергоспоживання здійснюється в двох варіантах:

• робота з одноадресними кадрами;

• робота з широкомовними / багатоадресними кадрами.

Припущення, на яких базується робота в режимі енергозбереження, прості. Клієнтська станція переходить в режим енергозбереження, коли відключає свою радіостанцію. Точка доступу буферизує кадри, призначені для певної станції, що знаходиться в режимі енергозбереження. Через певний інтервал часу клієнт "прокидається" (активується) і приймає сигнал від точки доступу, що показує, чи є в буфері кадри для даної клієнтської станції.

При роботі з одноадресними кадрами інтервал прослуховування або активізації визначається клієнтом. І навпаки, при роботі в режимі знергосбереженія з широкомовними / багатоадресними кадрами інтервал прослуховування визначається точкою доступу і оголошується в її сигнальних кадрах.

Клієнт періодично активізується і приймає сигнальні кадри точки доступу, щоб визначити, чи є для нього буферізовані кадри. Якщо такі кадри відсутні, клієнт повертається до роботи в режимі енергозбереження і перебуває в ньому до закінчення відповідного наступного періоду "сплячки".

6.1.3.4 Методи обмеження доступу до мережі

Фільтрація MAC-адрес:

Слід зразу сказати, що даний метод не входить до стандарту IEEE 802.11. Однак кожна сучасна точка доступу або роутер вже містять в собі цю можливість. Фільтрацію на цьому рівні можна здійснювати трьома способами:

точка доступу дозволяє отримати доступ станціям з будь-якою MAC-адресою;

точка доступу дозволяє отримати доступ тільки станціям, чиї MAC-адреси знаходяться в "білому" списку;

точка доступу забороняє доступ станціям, чиї MAC-адреси знаходяться в «чорному списку».

Найбільш надійним з точки зору безпеки є другий варіант, хоча він не захищає від підміни MAC-адреси, що легко зараз здійснюють зловмисники, якщо їм стає відома MAC-адреса діючої станції. *Метод можна використовувати виключно як додатковий метод захисту.*

Режим прихованого ідентифікатора SSID (англ. *Service Set IDentifier*):

Для свого виявлення точка доступу періодично розсилає кадри-маячки (англ. *Beacon frames*). Кожен такий кадр містить службову інформацію для підключення і, зокрема, в ньому присутній SSID (ідентифікатор бездротової мережі). У разі прихованого SSID це поле залишається пустим, тобто, стає неможливим просте виявлення вашої бездротової мережі і не можна до неї підключитися, не знаючи значення SSID. Але всі станції в мережі, які раніше вже були підключені до точки доступу, знають цей SSID і при підключенні, коли розсилають *Probe Request* запити, вказують ідентифікатори мереж, наявні в їх профілях підключень. Постійно прослуховуючи робочий трафік мережі, зловмисник з легкістю може отримати значення SSID, необхідне для підключення до бажаної точки доступу. Тому і цей спосіб обмеження доступу не може важатися надійним.

Метод можна використовувати виключно як додатковий метод захисту.

6.1.3.5 Методи аутентифікації у мережах WiFi

В мережах WiFi застосовуються наступні методи аутентифікації.

1. Відкрита аутентифікація (англ. *Open Authentication*).

Робоча станція робить запит аутентифікації, в якому присутня тільки MAC-адреса клієнта. Точка доступу відповідає або відмовою, або підтвердженням аутентифікації. Рішення приймається на основі MAC-фільтрації, тобто по суті це захист бездротової Wi-Fi мережі на основі обмеження доступу, що є небезпечним. Використовується виключно для доступу у відкритих публічних мережах.

Використовувані шифри: без шифрування, статичний WEP, CKIP.

2. Аутентифікація із загальним ключем (англ. *Shared Key Authentication*):

Необхідно налаштувати статичний ключ шифрування алгоритму WEP. Клієнт робить запит до точки доступу на аутентифікацію, на що отримує підтвердження, яке містить 128 байт випадкової інформації. Станція шифрує отримані дані алгоритмом WEP (проводиться побітове підсумовування по модулю 2 даних повідомлення з послідовністю ключа) і відправляє зашифрований текст разом із запитом на підключення. Точка доступу розшифровує текст і порівнює з вихідними даними. У разі збігу відсилається підтвердження підключення, і клієнт вважається підключеним до мережі.

Схема аутентифікації із загальним ключем вразлива до атак «Людина посередині». Алгоритм шифрування WEP - це простий XOR ключової послідовності з корисною інформацією, отже, прослухавши трафік між станцією і точкою доступу, можна відновити частину ключа. У даний час не використовується.

Використовувані шифри: без шифрування, динамічний WEP, CKIP.

3. Wi-Fi Protected Access (WPA)

Після перших успішних атак на WEP було прийнято рішення розробити новий стандарт 801.11i. Але до нього був випущений «проміжний» стандарт WPA, який включав в себе нову систему аутентифікації на базі 801.1x і новий метод шифрування TKIP. TKIP (англ. *Temporal Key Integrity Protocol*, протокол перевірки цілісності ключа) використовує удосконалений спосіб управління ключами і покадрову зміну ключа. Існують два варіанти аутентифікації: за допомогою RADIUS сервера (WPA-Enterprise) і за допомогою встановленого ключа (WPA-PSK). Більш надійним вважається спосіб аутентифікації за допомогою RADIUS-сервера, вбудованого в точку доступа. У даний час не рекомендовано для використання.

Використовувані шифри: TKIP (стандарт), AES-CCMP (розширення), WEP (для зворотної сумісності).

4. WI-FI Protected Access 2 (WPA2, 801.11i)

WPA2 або стандарт 801.11i - це фінальний варіант стандарту безпеки бездротових мереж. В якості основного шифру був обраний стійкий блоковий шифр AES. Система аутентифікації в порівнянні з WPA зазнала мінімальних змін. Також як і в WPA, в WPA2 є два варіанти аутентифікації WPA2-Enterprise з аутентифікацією на RADIUS сервері і WPA2-PSK з передвстановленим ключем. Саме цей метод аутентифікації рекомендовано для використання в сучасних безпровідних мережах WiFi.

Використовувані шифри: AES-CCMP (стандарт), TKIP (для зворотної сумісності з WPA).

5. Cisco Centralized Key Managment (CCKM)

Варіант аутентифікації від фірми CISCO. Підтримує роумінг між точками доступу. Клієнт один раз проходить аутентифікацію на RADIUS-сервері, після чого може перемикатися між точками доступу.

6.1.3.6 Методи шифрування, використовувані в мережах WiFi

1. WEP-шифрування (Wired Equivalent Privacy)

Аналог шифрування трафіку в провідних мережах. Використовується симетричний потоковий шифр RC4 (англ. *Rivest Cipher 4*), який досить швидко функціонує. На сьогоднішній день WEP і RC4 не важають криптостійкими. Є два основні протоколи WEP:

40-бітний WEP (довжина ключа 64 біта, 24 з яких - це вектор ініціалізації, який передається відкритим текстом);

104-бітний WEP (довжина ключа 128 біт, 24 з яких - це теж вектор ініціалізації); Вектор ініціалізації використовується алгоритмом RC4. Збільшення довжини ключа не призводить до збільшення надійності алгоритму.

Основні недоліки:

використання для шифрування безпосередньо пароля, введеного користувачем;

недостатня довжина ключа шифрування;

використання функції CRC32 для контролю цілісності пакетів;

повторне використання векторів ініціалізації і ін.

2. TKIP-шифрування (англ. Temporal Key Integrity Protocol)

Використовується той же симетричний потоковий шифр RC4, але з більшою криптостійкістю. Вектор ініціалізації становить 48 біт. Враховано основні атаки на WEP. Використовується протокол *Message Integrity Check* для перевірки цілісності повідомлень, який блокує станцію на 60 секунд, якщо послані протягом 60 секунд два послідовні повідомлення не пройшли перевірку цілісності. З урахуванням всіх доопрацювань і удосконалень TKIP на сьогоднішній день все одно не вважається криптостійким.

3. CKIP-шифрування (англ. Cisco Key Integrity Protocol)

Має схожість з протоколом TKIP. Створено компанією Cisco. Використовується протокол CMIC (англ. Cisco Message Integrity Check) для перевірки цілісності повідомлень.

4. WPA-шифрування

Замість вразливого RC4, використовується криптостійкий алгоритм шифрування AES (англ. *Advanced Encryption Standard*). Можливе використання протоколу EAP (англ. *Extensible Authentication Protocol*, розширюваний протокол аутентифікації). Протокол має два режими роботи:

Pre-Shared Key (WPA-PSK) - кожен вузол вводить пароль для доступу до мережі;

Enterprise - перевірка здійснюється серверами RADIUS.

Має більшу швидкість роботи при порівнянні з WPA2. Не рекомендовано до застосування у критичних мережах.

5. WPA2-шифрування (IEEE 802.11i)

Прийнято в 2004 році, з 2006 року все обладнання WiFi, що випускається, підтримує WPA2. В даному протоколі застосовується RSN (англ. *Robust Security Network*, мережа з підвищеною безпекою). Спочатку в WPA2 використовується протокол CCMP (англ. *Counter Mode with Cipher Block Chaining Message Authentication Code Protocol*, протокол блочного шифрування з кодом

автентичності повідомлення і режимом зчеплення блоків і лічильника). Основою є алгоритм AES. Для сумісності зі старим обладнанням є підтримка TKIP і EAP з деякими його доповненнями. Як і в WPA є два режими роботи: Pre-Shared Key і Enterprise. Саме WPA2-шифрування рекомендовано для застосування на сучасних WiFi мережах усіх типів.

WPA і WPA2 мають наступні переваги:

ключі шифрування генеруються під час з'єднання, а не розподіляються статично;

для контролю цілісності переданих повідомлень використовується *алгоритм Michael*;

використовується вектор ініціалізації істотно більшої довжини.

6.1.4 Бібліотеки Arduino для роботи з WiFi

Для полегшення програмування обміну в безпровідних WiFi мережах, де в якості станцій/давачів використовується технології Arduino, існує стандартна бібліотека Arduino WiFi. За допомогою додаткових шилдів ця бібліотека дозволяє Arduino підключатися до безпровідної мережі чи Інтернету. Сам шилд може працювати як сервер, що приймає вхідні з'єднання, або як клієнт, що здійснює вихідні з'єднання. Бібліотека підтримує персональне шифрування WEP і WPA2, але не WPA2 Enterprise. Якщо SSID не транслюється в мережі, шилд не зможе з'єднатися.

Плата Arduino спілкується з шилдом Wi-Fi за допомогою шини SPI, використовуючи цифрові контакти 50, 51 та 52 на Arduino Due. Контакт 10 використовується як SS. Апаратний SS-контакт 53 Arduino Due не використовується, але його режим роботи слід встановити як вихідний, або інтерфейс SPI не буде працювати. Цифровий контакт 7 використовується як контакт для хендшейкінга між шилдом Wi-Fi та Arduino, і його не слід використовувати та зачіпати.

Бібліотека WiFi дуже схожа на бібліотеку Ethernet, і багато функціональних викликів в них однакові.

У якості WiFi шилдів можливо використовувати шилди типів ESP8266 та ESP32.

6.1.5 Стандарт 802.11p (для використання на рухомому транспорті)

Транспортні мережі (іноді називаються автомобільними мережами або VANET) є спонтанними і неструктурованими, оскільки автомобіль переміщається по місту при взаємодії з іншими транспортними засобами та інфраструктурою. У цій мережі використовуються моделі передачі **автомобіль-автомобіль** (V2V) і **автомобіль-інфраструктура** (V2I).

IEEE 802.11p вважається виділеною комунікацією малого радіусу дії (*Dedicated Short Range Communication, DSRC*). Мета цієї мережі - забезпечити стандартну та безпечну систему V2V та V2I, яка використовується для забезпечення безпеки руху транспорту, збору дорожніх платежів, статусу дорожнього руху / попереджень, допомоги на дорозі та електронної торгівлі в транспортному засобі.

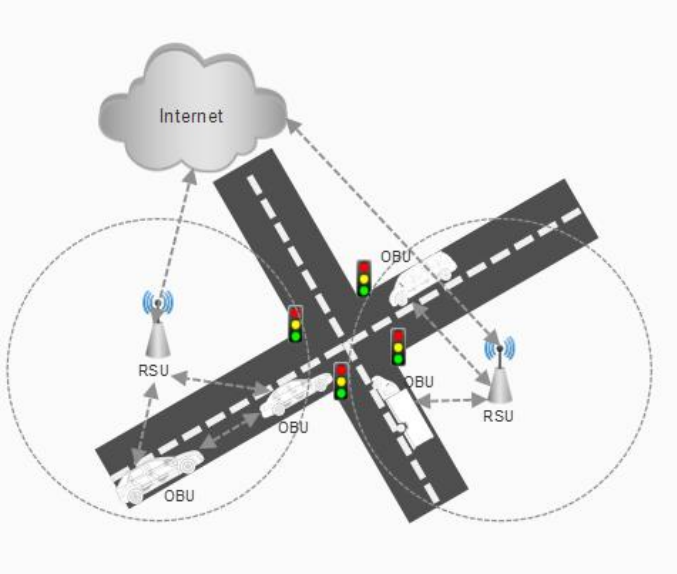


Рисунок 6.6 Варіант використання IEEE 802.11p. Показані OBU в транспортних засобах і RSU з фіксованою інфраструктурою

Топологія і загальний приклад для мережі IEEE 802.11p показаний на рис. 6.6. У мережі є два типи вузлів. По-перше, це дорожній пристрій (*road-side unit, RSU*), який розташовано фіксовано, подібно точці доступу. Він надає доступ транспортному засобу до спеціальних мереж та Інтернет для використання службових програм і доступу до відповідних довірених організацій. Іншим типом вузла є бортовий блок (*on-board unit, OBU*), який знаходиться в

транспортному засобі. Він може взаємодіяти не тільки з фіксованими RSU, але й з іншими OBU, коли це необхідно.

Для транспортних систем в бездротовому зв'язку існує декілька проблем. Є підвищений рівень безпеки, необхідний для використання на транспорті та управлінні. Окремими питаннями слід враховувати такі фізичні ефекти, як доплерівський зсув, ефекти затримки і надійність спеціальних мереж.

Багато що з того, що відрізняє 802.11p від стандартів 802.11, полягає в забезпеченні якості та дальності передачі по швидкості передачі. Іншими факторами є зміни для зменшення часу підключення OBU до мережі.

Нижче наведено короткий огляд особливостей IEEE 802.11p і відмінності від стандартів IEEE 802.11b:

- ширина каналу складає лише 10 МГц замість 20 МГц, використовуваних в 802.11b;
- IEEE 802.11p працює в смузі 75 МГц на частоті 5,9 ГГц. Це означає, що в цілому наявно лише сім доступних каналів (один канал управління, два критичних (резервних) і чотири канали послуг). В 802.11b доступно до 14 каналів;
- 802.11p підтримує лише половину бітових швидкостей у порівнянні з 802.11b, тобто 3/4,5/ 6/9/12/18/24/27 Мбіт/с;
- у нього такі ж схеми модуляції, як і в 802.11b і 52 піднесучих;
- тривалість символу стала подвійною в порівнянні з 802.11b. IEEE 802.11p підтримує 8 мкс, а 11b підтримує 4 мкс;
- охоронний інтервал становить 1,6 мкс в 802.11p, а в 11b - 0,8 мкс;
- такі методи, як використання MIMO і формування діаграми спрямованості, не є необхідними для рухомого об'єкту і відсутні в специфікації стандарту.

Основна модель використання 802.11p - це швидке створення з'єднання та його прив'язка до спеціальних мереж. Особливістю є те, що саме з'єднання швидко з'являється і швидко зникає, оскільки взаємодіючі транспортні засоби часто рухаються на швидкостях по шосе в протилежних напрямках. У стандартній моделі 802.11 BSS буде використовувати топологію мережі, яка

вимагає синхронізації, асоціації та аутентифікації для створення бездротової мережі. 802.11p обов'язково надає символи BSSID в заголовку всіх фреймів, якими обмінюються OBU/RSU (OBU/OBU), і вони можуть почати обмін кадрами даних відразу після доступу до каналу зв'язку.

Протокол IEEE 802.11p виріс з 802.11b, але вносить істотні зміни у ставленні до загальної безпеки і безпеки транспортних засобів. У Таблиця 6.1 показаний пакет протоколів стека 802.11p. Значним відступом від інших стеків IEEE 802.11 є використання стандартів IEEE 1609.x (*Бездротовий доступ в транспортних середовищах (WAVE)*) для вирішення прикладних завдань і для моделей безпеки. Повний стек називається бездротовим доступом в автомобільних середовищах (WAVE) і об'єднує рівні PHY і MAC 802.11p з рівнями IEEE 1609.x.

Таблиця 6.1 Стек протоколу 802.11 в порівнянні з моделлю OSI

<table>
<tr><th colspan="2">Стек протокола 802.11</th><th>Спрощена модель OSI</th></tr>
<tr><td colspan="2">IEEE 1609.1 (Safety and Traffic Efficiency-Applications)</td><td>7. Прикладний рівень</td></tr>
<tr><td colspan="2">IEEE 1609.2 WAVE Security Services</td><td rowspan="3">4. Транспортний рівень</td></tr>
<tr><td>TCP/UPD</td><td rowspan="2">IEEE 1609.3
WSMP</td></tr>
<tr><td>IPv6</td><td>3. Мережевий рівень</td></tr>
<tr><td colspan="2">Logical Link Control</td><td rowspan="2">2. Канальний рівень</td></tr>
<tr><td colspan="2">IEEE 1609.4 MAC SubLayer</td></tr>
<tr><td colspan="2">802.11p
5,9 ГГц
OFDM
3; 4,5; 6; 9; 12; 18; 24; 27 Мб/с</td><td>1. Фізичний рівень</td></tr>
</table>

Конкретні відмінності, які виділяються в стеку, включають:

- 1609.1 - менеджер хвильових ресурсів. Виділяє і забезпечує ресурси в міру необхідності;
- 1609.2 - визначає служби безпеки для повідомлень додатків і керування. Цей рівень також забезпечує два режими шифрування. Можна використовувати алгоритм з відкритим ключем, який використовує підписи ECDSA. В якості альтернативи доступний симетричний алгоритм, заснований на AES-128 в режимі CCM;

- 1609.3 - допомагає встановити з'єднання і управляти пристроями, сумісними з WAVE;
- 1609.4 - забезпечує багатоканальну роботу над рівнем MAC 802.11p.

Безпека дуже важлива в VANET. Тут має місце дуже висока ймовірність потенційних загроз, які можуть безпосередньо впливати на громадську безпеку. Атаки можуть виникати при трансляції фіктивної інформації, яка буде впливати на те, як інші транспортні засоби почнуть реагувати або виконувати дії. Атака такого характеру може бути, наприклад, проведена пристроєм-чужинцем, що передає фіктивну інформацію про небезпеку на дорозі, що у свою чергу призведе до зупинки транспортних засобів і виникненню транспортного коллапсу. Безпека тут також повинна враховувати приватні дані про транспортні засоби таким чином, щоб пристрій-чужинець не мав змоги замаскуватися під інший транспортний засіб і викликати відмову в подальшому наданні послуг. Все це може привести до катастрофічних подій, тому і потребує таких посилених стандартів, як IEEE 1609.2.

6.1.6 Протокол 802.11ah (спеціально для використання в IoT)

802.11ah - це варіант бездротових протоколів, спеціально призначених для роботи з IoT. Протокол оптимізує роботу пристроїв, які вимагають тривалої роботи від одної батареї. В протоколі оптимізовано діапазон і пропускну здатність використовуваного каналу зв'язку. 802.11ah також називається *HaLow*, що, по суті, є грою слів з «ha», що відноситься до «ah», маючи на увазі низьку потужність і більш низьку частоту («low»). Складені разом, вони утворюють похідну від «hello».

Метою групи IEEE 802.11ah було створення протоколу з розширеним діапазоном для зв'язку в сільських районах і розвантаження мобільного трафіку. Вторинної метою було використання протоколу для бездротового зв'язку з низькою пропускною здатністю в діапазоні субгігагерц. Специфікація (чорнетка) була опублікована 31 грудня 2016 р. Від інших стандартів 802.11 архітектура *11ah* найбільше відрізняється наступним:

- працює в діапазоні 900 МГц. Це дозволяє забезпечити добре поширення і проникнення сигналу через матеріали і підвищує незалежність від атмосферних умов;
- ширина каналу змінювана і може бути встановлена на 2, 4, 8 або 16 МГц;
- доступні різні методи модуляції, включаючи BPSK, QPSK, 16-QAM, 64-WAM і 256-QAM;
- використовується модуляція на основі стандарту 802.11ac зі специфічними змінами. В цілому використовує 56 піднесучих частот OFDM; з них 52, призначені для даних, і чотири, призначені для пілотних тонів;
- загальна тривалість символу становить 36 або 40 мкс;
- підтримує формування променя SU-MIMO (*single-user, multi-in multi-out*);
- використовує швидкий зв'язок для мереж, що складаються з тисяч вузлів, в яких присутні два різних метода аутентифікації для обмеження конкуренції;
- забезпечує підключення тисяч вузлів до єдиної точки доступу;
- включає можливість ретрансляції для зменшення потужності на пристрої і дозволяє використовувати грубу форму чарункової mesh-мережі з використанням методу одноразового переходу;
- дозволяє здійснювати розширене управління живленням на кожному вузлі 802.11ah;
- дозволяє спілкуватися у незірковій топології мережі за допомогою Windows з обмеженим доступом;
- дозволяє секторізацію, тобто дає можливість групувати антени для покриття різних областей BSS (звані секторами). Це досягається за допомогою формування діаграми спрямованості, що привнесено з інших протоколів 802.11.

Мінімальна пропускна спроможність такої мережі складе 150 Кбіт/с, при використанні модуляції BPSK на одному потоці MIMO з полосою пропускання

1 МГц. Максимальна теоретична пропускна спроможність складе 347 Мбіт/с на основі 256-WAM-модуляції з використанням 4 потоків MIMO і каналів шириною 16 МГц.

Специфікація IEEE 802.11ah (рис. 6.7) вимагає, щоб вузли підтримували смугу пропускання 1 МГц і 2 МГц. Точки доступу повинні підтримувати канали 1, 2 і 4 МГц. Канали 8 МГц і 16 МГц є додатковими. Чим вужча смуга пропускання каналу, тим більша відстань між вузлами можлива, але тим менша швидкість передачі. Чим більша пропускна здатність каналу, тим менша відстань між вузлами мережі можлива, але вища швидкість передачі.

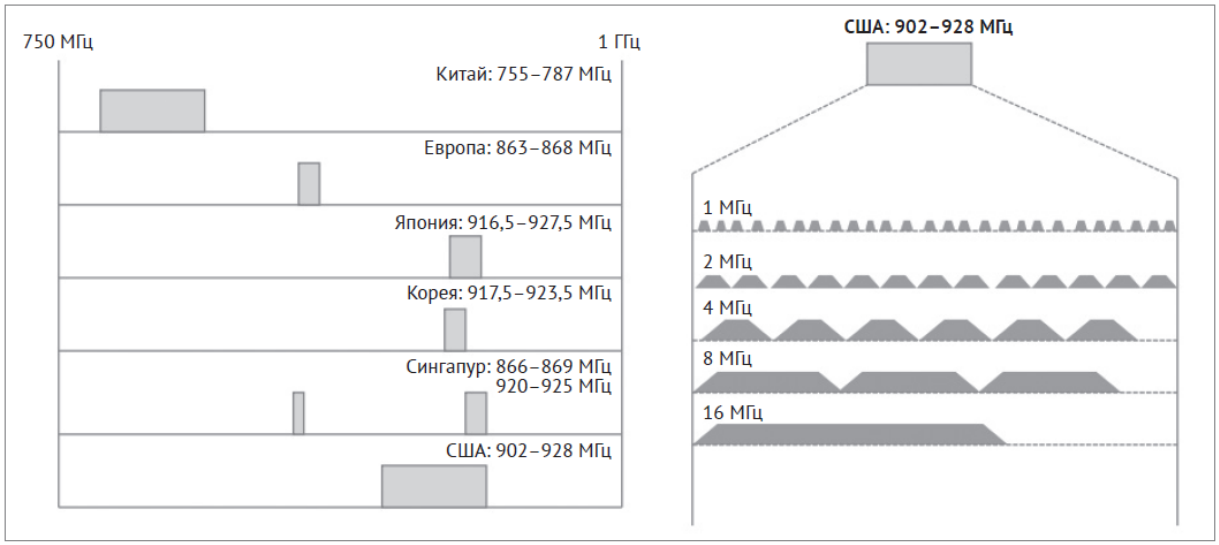


Рисунок 6.7 Зліва: різні варіанти каналоутворення в залежності від регіональних правил. Справа: різні варіанти пропускної здатності і використання каналів в регіоні США при ширині каналів від 1 МГц до 16 МГц

Все в архітектурі стандарту IEEE 802.11ah направлене на оптимізацію загального діапазону і ефективності. Це стосується навіть довжини заголовків MAC.

Підключення декількох тисяч пристроїв до однієї точки доступу також виконується з використанням унікального ідентифікатора асоціації (AID) в 13 біт. Це дозволяє групувати вузли (пристрої) на основі групових критеріїв (освітлення в передпокої, вимикач освітлення і т. ін.). Це дозволяє точці доступу підключатися до більш 8191 вузлів (пристроїв). Стандарт 802.11 може підтримувати тільки 2007 пристроїв. Проте, багато вузлів здатні викликати величезну кількість колізій в каналі. Незважаючи на те, що кількість підключених пристроїв збільшилася, мета при створенні стандарту полягала в скороченні обсягу даних на шляху їх передачі з цих станцій. Цільова група IEEE

виконала це, видаливши кілька полів, що не були особливо важливі для випадків використання IoT, таких як поля QoS (якість обслуговування) і DS. На рис. 6.8 показані кадри низхідної лінії зв'язку 802.11ah і висхідної лінії зв'язку в порівнянні зі стандартом 802.11.

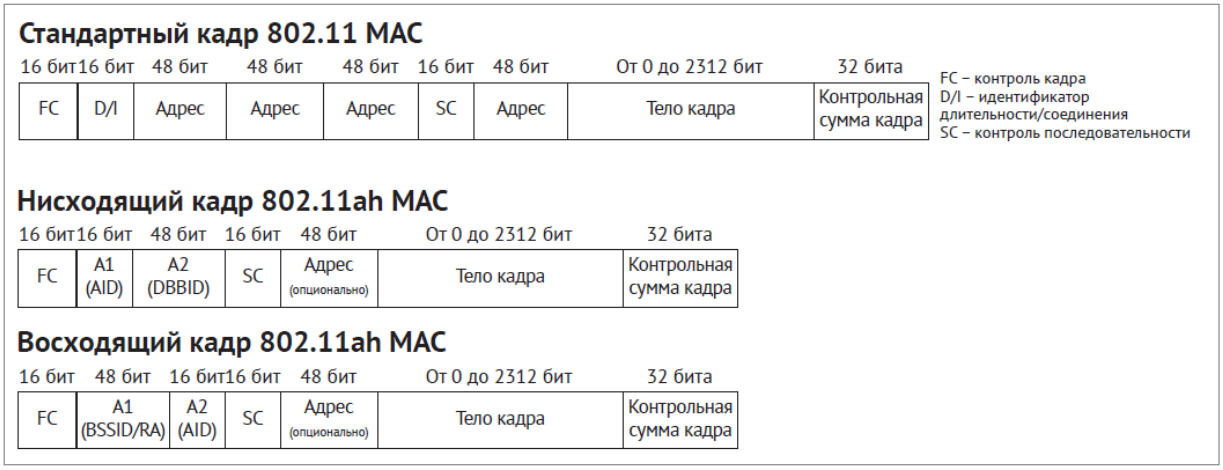


Рисунок 6.8 Порівняння стандартного кадру 802.11 MAC і стисненого кадру 802.11ah

Для покращення керування живленням і ефективністю каналу видалені кадри підтвердження ACK. В стандарті *11ah* ACK є неявним для двонапрямлених даних. Тобто обидва пристрої відправляють і отримують дані один від одного без окремого підтвердження кожного кадру. Зазвичай ACK використовується після успішного прийому всього пакета даних. В режимі *bidi* (BDT) сам факт прийому наступного кадру передбачає, що попередні дані були

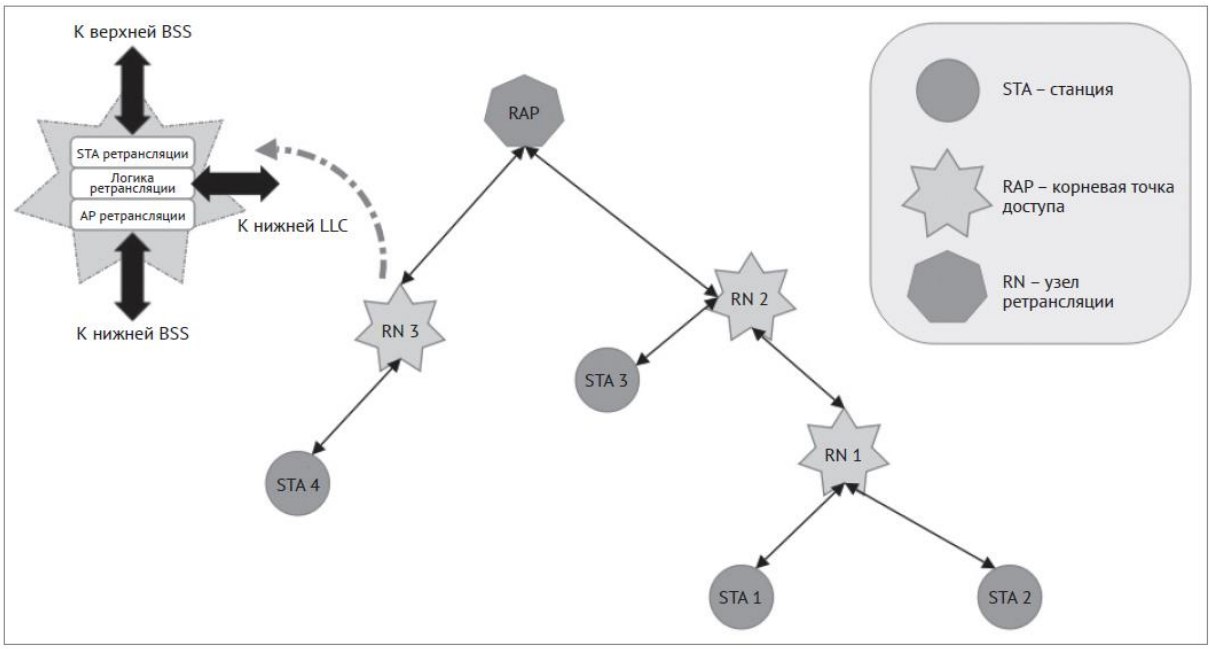


Рисунок 6.9 Мережева топологія IEEE802.11ah

успішно отримані і пакет ACK не потрібен при обміні.

З точки зору топології (рис. 6.9), в мережі *802.11ah* є три типи станцій:

- **Root access point** (RAP) – коренева точка доступу. Як правило, є шлюзом до інших мереж (WAN);
- **STA** - типова станція 802.11 або кінцевий клієнт;
- **Relaynode** (RN) - спеціальний вузол, який об'єднує AP-інтерфейс з STA; що знаходиться в нижній BSS, і інтерфейс STA для інших вузлів ретрансляції або кореневої AP на верхній BSS.

На рис. 6.9 показана топологія *IEEE 802.11ah*. Ця архітектура істотно відрізняється від інших протоколів 802.11 при використанні єдиних ретрансляційних вузлів, які працюють для створення ідентифікованої BSS. Ієрархія співвідношень RN/STA формує велику мережу. Кожне співвідношення діє як AP (точка доступу) і STA.

Окрім основних типів вузлів існують і три стани енергозбереження, в яких може перебувати STA:

карта індикації трафіку, (Traffic Indication Map, TIM) - прослуховує AP для передачі даних. Вузли будуть періодично отримувати інформацію про дані, буферизовані для них своєю точкою доступу. Відправлене повідомлення називається інформаційним елементом TIM;

не TIM станції, - вирішує з AP під час об'єднання з нею, виділення відповідного часового проміжку для зв'язку між AP та STA (*Windows Periodical Restricted Access*, PRAW);

позапланові станції - не прослуховує ніяких маяків і використовує опитування при доступі до каналів.

Енергоспоживання має вирішальне значення для давачів IoT і периферійних пристроїв, які працюють від батарей. Протоколи 802.11 відомі тим, що пред'являють високі вимоги до потужності передавача пристрою. Для відновлення потужності після "сплячки", 802.11ah використовує значення *Max Idle Period* (максимальний період очікування), яке є частиною звичайних специфікацій 802.11. У загальній мережі 802.11 максимальний період очікування становить приблизно 16 годин в залежності від часу 16-бітного дозволу. У

802.11ah перші два біта 16-розрядного таймера є коефіцієнтом масштабування, який дозволяє тривалість сну підвищити до п'яти років.

Додаткову споживану потужність для давачів пом'якшують через зміни в алгоритмі роботи маяка. Як раніше розглядалося, маяки передають інформацію про наявність буферизованих кадрів. Маяки будуть мати бітову карту TIM, яка збільшує їх розмір, оскільки 8191 STA призведе до істотного зростання розміру бітової карти. 802.11ah використовує концепцію, яка називається сегментацією TIM, що скорочує розмір бітової карти TIM. Кожна STA обчислює, коли надходить відповідний маяк з інформацією растрової карти і дозволяє пристрою входити в режим енергозбереження до моменту, коли він повинен прокидатися і отримувати інформацію від маяка.

Інша функція енергозбереження називається **цільовим часом пробудження** (*Target Wake Time, TWT*) і призначена для STA, які рідко передають або отримують дані. Це поширено при розгортанні IoT для прийому таких даних, як дані від давача температур, наприклад. STA і асоційована з нею AP будуть обговорювати узгодження TWT, після чого STA перейде в стан сну,

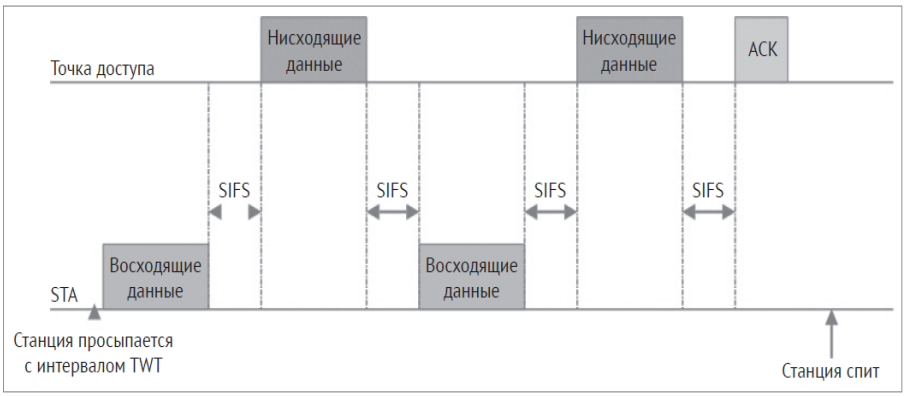


Рисунок 6.10 Speed Frame Exchange 802.11ah: приклад цільового часу пробудження (TWT), який використовується для запуску передачі STA. SIFS є розрив між передачею AP і STA. Між парами даних не використовуються ACK. Тільки один ACK в кінці передачі відправляється до того, як STA повернеться у сплячий режим

поки не спрацює сигнал TWT. Процесс неявних ACK називається *Speed Frame Exchange*, як показано на рис. 6.10.

6.2 Порівняння персональних бездротових IP та не-IP мереж

Використання стандарту зв'язку на основі IP значно спрощує дизайн і дозволяє швидко і легко масштабувати сенсорну мережу при необхідності.

Масштабування має вирішальне значення для розгортання IoT, які охоплюють тисячі або мільйони вузлів. Використання транспорту на основі IP дозволяє працювати за допомогою простих інструментів. Bluetooth (та мережі 6LoWPAN і Thread) демонструють стандарти, які можуть застосовуватися до не IP-протоколів, таких як 802.15.1 та 802.15.4. Обидва протоколи дозволяють адресувати IPv6 і mesh-мережу в масивні мережі IoT. 802.11 є важливим і надзвичайно успішним протоколом, який складає основу WLAN, але також може діяти в пристроях IoT і давачах з використанням 802.11ah або транспортних системах з використанням 802.11p. Табл. 6.2 містить порівняння традиційного протоколу, відмінного від IP, з протоколом IP. В основному, різниця буде в потужності, швидкості і максимальній відстані зв'язку.

Таблиця 6.2 Порівняння традиційного протоколу, відмінного від IP, з протоколом IP

	802.15.1	**802.11ah**
IP	не IP	IP
Максимальна міжвузлова відстань	100 м	1000 м
Мережева топологія	повністю чарункова	ієрархічна з одним переходом
Використання каналів	ISM 2,4 ГГц тільки з DSSS	нижче 1 ГГц ISM з різними схемами кодування модуляції. Смуга пропускання каналу: 1, 2, 4, 8, 16 МГц
Пропускна здатність	150 Кбіт/с…2 Мб/с	150 Кбіт/с…347 Мб/с
Затримка	гарна	краща (в 2 рази краще, ніж у 802.15.1)
Енергетична ефективність	найкраща (17 мДж/пакет)	добра (63 мДж/пакет)
Енергозберігання	механізми сну-пробудження в кадрах	безліч структур даних для керування та тонкого налагоджування потужності на різних рівнях
Розмір мережі	до 65000 вузлів	8192 STA до однієї AP

Контрольні питання

1. Порівняти характеристики стандартів 802.11n, 802.11ah та 802.11p. Охарактеризувати області застосування.
2. Охарактеризувати застосування наступних топологій Wi-Fi мереж: Ad-Hoc, BSS та ESS.
3. Які задачі вирішуються шифруванням та аутентифікацією у сенсорній мережі Wi-Fi?
4. Які класи загроз мережевої інформаційної безпеки ви знаєте? Охарактеризувати їх.
5. Які види шифрування для Wi-Fi мереж ви знаєте? Їх характеристики.
6. Які різновиди загроз існують у Wi-Fi мережах? Що таке чужинець?
7. Що таке опосередковані загрози в мережі?
8. Які особливості функціонування Wi-Fi мереж слід враховувати при аналізі загроз?
9. Які існують методи обмеження доступу у Wi-Fi мережах?
10. Які ви знаєте методи аутентифікації у мережах Wi-Fi? Охарактеризувати їх.
11. Які ви знаєте методи шифрування у мережах WiFi? Охарактеризувати їх.
12. Для чого використовується і як працює технологія стандарту 802.11p? ЇЇ особливості і відмінності від 802.11b/n.
13. Яка топологія використовується у стандарті 802.11p?
14. Для чого використовується і як працює технологія стандарту 802.11ah? ЇЇ особливості і відмінності від 802.11b/n.
15. Яка топологія використовується у стандарті 802.11ah? Можливість організації чарункової мережі на базі цього стандарту.

7 СЕНСОРНІ БЕЗДРОТОВІ СИСТЕМИ І ПРОТОКОЛИ ДАЛЕКОГО ЗВ'ЯЗКУ LPWAN

До цієї пори ми обговорювали принцип роботи **персональних мереж (WPAN)** і **локальних мереж (WLAN)**. У таких системах передачі даних давачі підключаються до локальної мережі, але не обов'язково до мережі Інтернет або до інших систем. Необхідно пам'ятати, що екосистема інтернету речей включає в себе давачі, виконавчі пристрої, камери, вбудовані інтелектуальні пристрої, транспортні засоби та роботів, які можуть знаходитися в найвіддаленіших місцях. Для передачі даних на великі відстані необхідно звертатися вже до **глобальної обчислювальної мережі (ГОМ)**.

У цій темі буде охоплено основні пристрої і топології мереж *LPWAN (англ. Low-power Wide-area Network — енергоефективна мережа далекого радіусу дії),* включаючи стільникову **4G LTE (NB-IoT)** та пропрієтарну систему зв'язку, таку як **LoRa** [8][19]. Слід мати на увазі, що далека передача даних - це зазвичай послуга, що отримується по підписці у постачальника таких послуг (наприклад, оператора мобільного зв'язку), який має в своєму розпорядженні вишки стільникового зв'язку і надає відповідну вдосконалену інфраструктуру. Відмінність попередніх архітектур WPAN і WLAN полягає в тому, що вони існують в рамках пристрою, який клієнт або розробник виробляє або перепродає. Підписка або угода про рівень надання послуги (*service-level agreement, SLA*) має інший вплив на архітектуру і обмеження системи, в якому повинен розбиратися архітектор.

7.1 Розвиток стільникових мереж

Найпоширеніший формат обміну даними - шляхом стільникового радіозв'язку, де особливе місце займає стільникова передача даних. Так як пристрої мобільного зв'язку існували задовго до винаходу технології стільникового зв'язку, вони мали обмежену зону дії, загальний діапазон частот і працювали переважно за принципом двостороннього радіозв'язку. Американська корпорація Bell Labs (Лабораторії Белла) працювала над створенням експериментальних версій технології мобільного зв'язку ще в 1940-х рр. (Служба

мобільного зв'язку) і в 1950-х рр. (Удосконалена служба мобільного зв'язку), але вони мали незначний успіх. У той час ще не існувало універсальних стандартів для мобільного телефонного зв'язку. Тільки коли концепція розвитку мобільного зв'язку була розроблена Дугласом Рингом і Реєм Янгом в 1947 р і реалізована Річардом Френкелем, Джоелом Енгелем і Філіпом Портером в офісі Bell Labs в 1960-х рр., можна було говорити про більш широке й ефективне використання мобільних технологій. Концепція «міжстільникового хендовера» була створена і реалізована Амосом Е. Джоелом - молодшим, також в офісі компанії Bell Labs. Вона мала на увазі переключення абонента до нової базової станції (стільнику) при переміщенні мобільного пристрою. Усі ці технології у 70-і роки минулого століття склали докупи і створили першу систему стільникових телефонів, перший дзвінок по такому телефону зробив Мартін Купер, співробітник компанії Motorola, 3 квітня 1973 року.

Слід мати на увазі, що намагання зробити систему мобільного радіозв'язку провадилися і в Радянському Союзі ще с кінця 50-х років минулого століття. Результатом цієї розробки стала система «Алтай» - радянська і російська система повністю автоматичного мобільного зв'язку. Робота над дуплексною системою автоматичного мобільного зв'язку почалася в 1958 році у Воронезькому НДІ зв'язку (ВНИИС). Були створені абонентські станції і базові станції для зв'язку з ними. Абонентські станції встановлювалися в автомобілях вищого радянського та партійного керівництва, для військових та служби безпеки. Антенні системи були розроблені у московському Державному Спеціалізованому Проектному Інституті (ГСПИ). Також брали участь підприємства Ленінграда, Білорусії і Молдавії. У 1963 році в Москві була запущена перша система «Алтай» (рис. 7.1). Спочатку в системі використовувався частотний діапазон 150 МГц, пізніше, в 1970-х р.р., коли «Алтай» був встановлений і успішно працював в 114 містах СРСР, були виділені нові радіоканали в діапазоні 330 МГц - що дозволяло забезпечувати чималу дальність і одночасно обслуговувати більше абонентів.

Рисунок 7.1 Мобільне обладнання системи "Алтай", що монтувалося в автомобіль "Волга"

У якості ідеальної стільникової моделі, компанією Bell Labs була взята модель, в якій стільникові комірки представлені шестикутними ділянками з оптимальним розміщенням, так як показано на малюнку нижче.

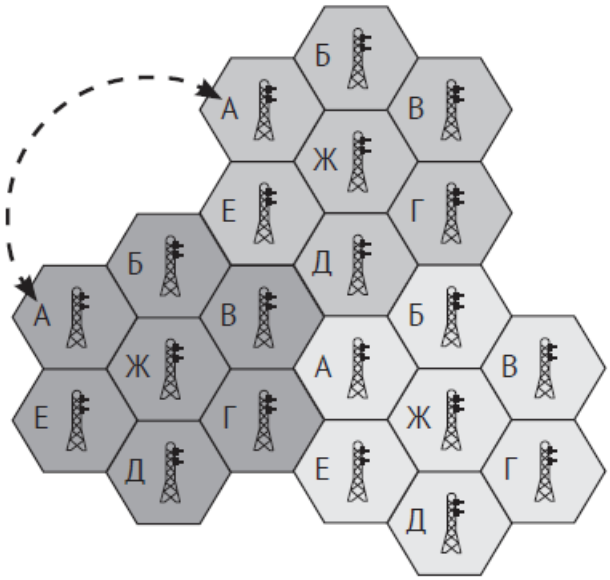


Рисунок 7.2 Організація стільникової мережі. Шестикутна модель розташування стільників гарантує такий поділ частот, при якому уникають перетину на одній і тій самій частоті найближчі сусіди. В одному стільнику (шестикутнику) не може бути перетину двох однакових частот, як показано на прикладі стільника А в двох різних регіонах, що вже допускає повторне використання однієї частоти

Орган, який зараз займається установкою стандартів в світі стільникового зв'язку, це організація з назвою 3GPP (акронім від «Партнерський проект 3-го покоління»), яка складається з 7 телекомунікаційних компаній (також відомих як «Партнерські організації») з усього світу, які впливають на роботу і розвиток

технологій стільникового зв'язку. Була створена у 1998 р у співпраці з канадською компанією Nortel Networks і американською телекомунікаційною компанією AT&T Wireless і випустила перший стандарт в 2000 р. Партнерські організації і Market Representatives працюють в Японії, Америці, Китаї, Європі, Індії та Кореї. Загальна мета групи - визнання вже встановлених стандартів і специфікацій для Глобальної системи мобільного зв'язку (GSM) в процесі створення 3G специфікацій для стільникового зв'язку. Обов'язки 3GPP виконуються трьома Групами технічних специфікацій (TSG) і шістьма Робочими групами (WG). Групи збираються кілька разів на рік у різних регіонах. Головним завданням релізів 3GPP є зробити спадну і висхідну сумісність пристроїв можливою.

На Україні у даний час отримує розвиток технологія NB-LTE, яка вже підтримується усіма мобільними національними операторами (Vodafone, Kyivstar, Life cell) і призначена для роботи IoT пристроїв. З січня 2020 р. технологія NB-LTE буде офіційно запущена у комерційну експлуатацію оператором Vodafon у містах Київ, Харків, а на протязі 2020 р. – по всій території України.

7.2 Технології стільникової мережі NB-LTE

7.2.1 Категорії абонентського обладнання

У релізі 8 3GPP (найперший LTE Advanced) було 5 категорій абонентського обладнання, кожна з різною швидкістю передачі даних і архітектурою MIMO (табл. 7.2). Категорії дозволили 3GPP розрізняти між собою еволюції LTE. Після 8-го релізу було додано більше категорій. У категоріях поєднуються можливості прийому і передачі даних, як зазначено організацією 3GPP. Як правило, кінцевий коритстувач отримує стільникове радіо або чіпсет, на яких вказано, яку категорію вони підтримують. Якщо обладнання користувача підтримує певну категорію, стільникова система (eNodeB, про яку йтиметься далі) також повинна підтримувати цю категорію.

Частиною процесу асоціації між пристроєм стільникового зв'язку та інфраструктурою є обмін інформацією про можливості, такі як категорія.

Таблиця 7.1 Категорії абонентського обладнання 3GPP

Релиз 3GPP	Категория	Максимальная скорость передачи данных при нисходящей связи (Мб/с) L1	Максимальная скорость передачи данных при восходящей связи (Мб/с) L1	Максимальное количество каналов нисходящей передачи данных MIMO
8	5	299,6	75,4	4
8	4	150,8	51	2
8	3	102	51	2
8	2	51	25,5	2
8	1	10,3	5,2	1
10	8	2998,60	1497,80	8
10	7	301,5	102	2 или 4
10	6	301,5	51	2 или 4
11	9	452,2	51	2 или 4
11	12	603	102	2 или 4
11	11	603	51	2 или 4
11	10	452,2	102	2 или 4
12	16	979	неизвестно	2 или 4
12	15	750	226	2 или 4
12	14	3,917	9,585	8
12	13	391,7	150,8	2 или 4
12	0	1	1	1
13	NB1	0,68	1	1
13	M1	1	1	1
13	19	1566	неизвестно	2,4 или 8
13	18	1174	неизвестно	2,4 или 8
13	17	25,065	неизвестно	8

Зверніть увагу на категорії M1 і NB1 в релізі 13. Тут організація 3GPP значно зменшила швидкість передачі даних до 1 Мб/с і менше. Це категорії, призначені для пристроїв з функціоналом інтернета речей, яким достатньо низької швидкості передачі даних, так як з'єднання відбувається за допомогою посилання коротких сигналів.

LTE-13 використовує дуплекс з частотним поділом (FDD) - в конфігурації FDD базова станція (eNodeB) і абонентський пристрій (UE). Використовує два діапазони частот для прийому і передачі даних: діапазон передачі даних варіюється від 777 до 787 МГц, а діапазон прийому даних від 746 до 756 МГц (в Україні на даний момент підтримується тільки діапазон 1800 МГц). Передача і прийом даних можуть відбуватися одночасно.

7.2.2 Характеристики NB-LTE

Технологія Cat-NB (LTE), також відома як NB-IoT, NB1 або IoT з вузьким діапазоном, є протоколом LPWAN, що керується 3GPP, починаючи з 13 релізу.

Cat-NB працює в ліцензованому спектрі (800 чи 1800 МГц). Метою розробки цього протоколу стало намагання подальшого зменшення використовуваної потужності абонентського пристрою (що збільшує до 10 років час роботи від одної батареї), розширення покриття (на +20 дБ) і зниження вартості абонентського обладнання (5 доларів США за абонентський модуль). Cat-NB заснований на *Evolved Packet System (EPS)* та оптимізован для *Cellular Internet of Things (CIoT)* (сотового IoT). Оскільки канали набагато вужчі, ніж навіть в попередньому протоколі Cat-M1, вартість і потужність можуть бути помітно зменшені завдяки більш простим конструкціям абонентського пристрою.

Особливості протоколу NB при порівнянні з попереднім M1:

- дуже вузька смуга пропускання каналу – у випадку Cat-M1, в якому ширина каналу становить максимум до 1,4 МГц, Cat-NB зменшує його до 180 кГц з тих же причин (знизити витрати і потужність);
- відсутній VoLTE - оскільки канал настільки вузький, що у нього немає можливості для передачі голосового або відео трафіку;
- немає мобільності (це відноситься до 13 релізу. 14 реліз підтримує мобільність зі швидкостями до 20 миль/год) - Cat-NB не підтримує передачу обслуговування в іншу соту і повинен залишатися пов'язаним з одною і тою самою сотою (хоча і може переміщуватися всередині стільника!), або залишатися нерухомим. Це відмінно підходить для більшості захищених і закріплених давачів IoT. Це унеможливлює передачу обслуговування іншому стільнику чи іншій мережі.

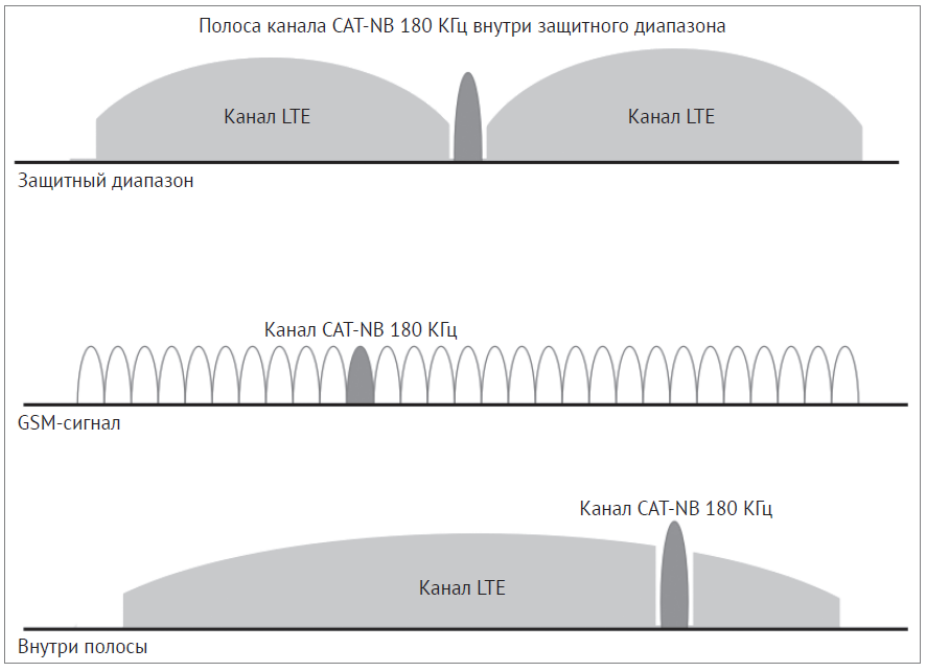


Рисунок 7.3 Варіанти розгортання Cat-NB в якості захисного діапазону, всередині GSM-сигналу або всередині смуги з LTE

Незалежно від цих істотних відмінностей Cat-NB заснований на мультиплексуванні низхідної і висхідної лінії зв'язку, і використовує один і той самий інтервал піднесучої частоти і тривалість символу.

Так як ширина каналу дуже мала (всього 180 кГц), це дає можливість розмістити сигнал Cat-NB всередині більшого LTE-каналу, чи замінити ним канал GSM або навіть існувати в захисному каналі разом зі звичайним сигналом LTE (див. **Ошибка! Источник ссылки не найден.**). Це забезпечує потрібну гнучкість в розгортанні Cat-NB в існуючих LTE, WCDMA і GSM мережах. Опція GSM є найпростішою і вже існуючою на ринку. Деякі частини існуючого трафіку GSM можуть бути розміщені в мережі WCDMA або LTE. Це звільняє провайдеру GSM місце для трафіку IoT. Варіант **In-band** (всередині смуги пропускання LTE) забезпечує величезний спектр для використання, оскільки смуги LTE є набагато ширшими, ніж смуга 180 кГц. Це дозволяє на практиці розгортати до 200 000 пристроїв у кожному стільнику. У такій конфігурації базова стільникова станція буде мультиплексувати дані LTE з трафіком Cat-NB. Це цілком можливо, так як архітектура Cat-NB є автономною мережею і прекрасно взаємодіє з існуючою інфраструктурою LTE. Нарешті, використання Cat-NB у полосі захисної смуги LTE є унікальною і новою концепцією. Оскільки архітектура повторно використовує одну і ту саму конструкцію піднесучих частот 15 кГц, її можна забезпечити за допомогою існуючої інфраструктури мережі.

Так як MCL сигналу Cat-NB (максимальна втрата зв'язку) становить 164 дБ, LTE-NB забезпечує глибоке покриття в підвалах, тунелях, сільських районах

і відкритих середовищах (це не зовсім так. Практика тестування засвідчує, що у закритих приміщеннях чи у підвалах прийом з великою вірогідністю може бути відсутнім). Підвищення MCL на 20 дБ у порівнянні зі стандартним LTE (144 дБ) – зразу дає збільшення площі покриття в 7 разів. Доступна швидкість передачі даних є функцією SNR і ширини каналу, як ми бачили в теоремі Шеннона-Гартлі. Для зв'язку по висхідній лінії Cat-NB призначить кожному абонентському обладнанню UE одну або більше піднесучих 15 кГц в блоці ресурсів 180 кГц. Cat-NB має можливість зменшити ширину одної піднесучої до 3,75 кГц, що дозволяє іншим пристроїв розділяти той самий діапазон. Однак необхідно ретельно перевіряти наяність перешкод між піднесучими шириною 3,75 кГц і 15 кГц.

Саме ці особливості зробили даний протокол основним для стільникових операторів при виборі стандарту організації роботи мереж IoT на території України.

7.2.3 Керування енергозбереженням у пристроях IoT NB LTE

Потужність має вирішальне значення для надійного функціонування периферійних пристроїв IoT. Самим значним фактором зниження потужності Cat-NB є зміна потужності передачі. Як уже згадувалося, організація 3GPP знизила потужність передачі з 23 дБ до 20 дБ. Це зниження потужності не обов'язково означає зменшення покриття. Стільникові вежі будуть ретранслювати пакети від шести до восьми разів. Це робиться для забезпечення надійного прийому в особливо проблемних областях. Радіостанції Cat-NB можуть відключати прийом, як тільки вони отримають пакет без помилок.

Іншою функцією енергозбереження є режим розширеного переривчастого прийому (eDRX), який дозволяє використовувати період очікування 10,24 с між циклами пошукового виклику. Це значно знижує потужність і дозволяє абонентському пристрою спати протягом програмованої кількості гіперкадров (HF) по 10,24 с кожен. Пристрій може увійти в цей розширений режим очікування на час до 40 хвилин. Це дозволяє пристрою у режимі радіо використовувати струм холостої ходи всього до 15 мкА.

Додаткові можливості по зниженню потужності в абонентському пристрої включають:

- **режим PSM**. У традиційному LTE-пристрої модем увесь час залишається підключеним до мережі, споживаючи енергію, незалежно від активності пристрою, чи коли пристрій не діє або спить. Щоб уникнути перевищення енергоспоживання пристрій може самостійно відключатися від мережі, але це може приводити до повторного підключення та пошуку протягом декількох хвилин. PSM дозволяє модему увійти в дуже глибокий стан сну, коли він взагалі відключений, але може швидко при цьому прокидатися. Модем робить це, періодично виконуючи оновлення зони відстеження (TAU) і залишаючись доступним через пейджинг протягом програмованого періоду часу. По суті, пристрій IoT може входити в період бездіяльності на цілий рік і прокидатися один раз на день/місяць/рік, щоб передавати дані з давача, увесь час при цьому, залишаючись на зв'язку;
- **функція оптимізації контексту** контрольної панелі користувача і управління. У звичайних LTE-системах новий RRC-контекст повинен створюватися кожен раз, коли абонентський пристрій прокидається з режиму сну. Це споживає значну частину потужності пристрою, особливо коли просто потрібно відправити обмежений обсяг даних. Використовуючи оптимізацію, контекст контрольної панелі зберігається і не вимагає повторної передачі;
- **стиснення заголовків пакетів** TCP або UDP;
- **скорочення часу синхронізації** після тривалих періодів сну.

7.3 Пропрієтарна мережа LoRa

LoRa (*LongRang*) - це пропріетарна (приватна) технологія і її користувачі мають переваги використання неліцензійного спектра частот. Сам термін LoRa - це назва фізичного рівня для протоколу IoT малої потужності і на великі відстані, в той час як термін LoRaWAN означає назву протоколу рівня доступу MAC.

Архітектура була спочатку розроблена компанією Cycleo у Франції, але потім була придбана Semtech Corporation (французьким виробником електроніки змішаних сигналів) в 2012 р. за 5 мільйонів доларів готівкою. Альянс LoRa був

сформований в березні 2015 року. Альянс є стандартизуючим органом для специфікації і технології LoRaWAN. Туди також входить організація процесу дотримання і сертифікації для забезпечення сумісності і відповідності стандарту. Альянс підтримується IBM, Cisco і ще більш ніж 160 іншими учасниками.

LoRaWAN вже працює в Європі з розгортанням на мережах KPN, Proximus, Orange, Bouygues, Senet, Tata і Swisscom. В Україні впровадженням мереж на основі LoRa займається компанія IMC. У м. Києві в 2018 році почалося спорудження опорної мережі на основі технології LoRa. У Харкові тим самим вже кілька років займається компанія LOTHINGS.

Один шлюз LoRaWAN може покрити значну площу. Бельгія площею 30500 км^2 повністю покрита сім'ю шлюзами LoRaWAN. Типова відстань дії одного шлюзу від 2 до 5 км в міських районах і до 15 км в приміських районах. Таке зниження вартості інфраструктури вигідно відрізняє LoRa від 4G LTE з набагато меншими стільниками.

7.3.1 Фізичний рівень LoRa

LoRa являє собою фізичний рівень мережі LoRaWAN. Він керує модуляцією, потужністю, приймачем і передаючими радіостанціями, а також формує сигнали.

Архітектура заснована на наступних діапазонах ISM без ліцензування:

- 915 МГц - в США з обмеженнями потужності, але без обмеження робочого циклу;
- 868 МГц - в Європі з 1% -ним і 10% -ним робочим циклом;
- 433 МГц - в Азії.

Метод модуляції, що використаний в LoRa, є похідним від **Chirp Spread Spectrum (CSS)**. CSS балансує швидкість передачі даних з чутливістю приймача в смузі фіксованого каналу. CSS був вперше використаний в 1940-х рр. для військового довгохвильового зв'язку з використанням модульованих імпульсів чірпа (скрекотання) для кодування даних і був визнаний особливо стійким до перешкод, ефектів Доплера і багатопроменевого розповсюдження. Чірпи - це синусоїдальні хвилі, частота яких з часом збільшується або зменшується. Оскільки вони використовують весь канал для зв'язку, вони відносно надійні в

плані перешкод. Ми можемо уявити чирп-сигнали зі збільшенням або зменшенням частоти, як звук поклика кита. Бітрейт у LoRa є функцією швидкості чірпа і швидкості передачі символів. Тому бітрейт (біт/с) у LoRa може змінюватися від 0,3 до 5 Кбіт/с і визначається як:

$$R_b = S * \frac{1}{\left[\frac{2^S}{B}\right]}, \quad (7.1)$$

де R_b – бітрейт канала LoRa (б/с);

S – коефіцієнт розширення;

B – смуга пропускання (Гц).

Як показало використання такої форми модуляції у військовій сфері вона допускає малу потужність сигналу на прийомі для великих відстаней. Дані кодуються з використанням збільшення або зменшення частоти, і кілька передач можуть вестися одночасно з різною швидкістю передачі даних на одній і тій самій частоті. CSS дозволяє отримувати сигнали на рівні 19,4 дБ нижче рівня шуму, використовуючи пряму корекцію помилок FEC. Смуга каналу також поділяється на кілька піддіапазонів. LoRa використовує канали зі смугою 125 кГц, виділяє шість каналів 125 кГц і використовує стрибкоподібний псевдовипадковий перехід між каналами під час передачі. Кожен кадр буде передаватися з певним коефіцієнтом розширення. Чим більше коефіцієнт розширення, тим повільніше іде передача, але тим більше збільшується можлива відстань між приймачем та передавачем. Кадри в LoRa є ортогональними, що означає, що кілька кадрів можуть відправлятися одночасно на одній несучій частоті, якщо кожен відправляється з іншим коефіцієнтом розширення. Всього є шість різних коефіцієнтів розширення (від SF = 7 до SF = 12).

Типовий пакет LoRa містить преамбулу, заголовок і корисні дані від 51 до 222 байт.

Мережі LoRa мають потужну функцію, звану **Adaptive Data Rate (ADR)**. По суті, ця функція дозволяє динамічно масштабувати ємність мережі, грунтуючись на щільності вузлів і інфраструктурі. ADR керується сервером управління мережею в хмарі. Вузли, близькі до базової станції, можуть мати більш високу швидкість передачі даних через високу достовірність сигналу. Вузли, що знаходяться в безпосередній близькості, можуть швидко передати

дані, звільнити свою смугу пропускання і швидко увійти у стан сну в порівнянні з віддаленими вузлами, які передають з меншою швидкістю.

Таблиця 7.2 Властивості висхідного і низхідного каналу зв'язку

Властивість	Висхідний канал	Нисхідний канал
Модуляція	CSS	CSS
MCL сигналу (максимальна втрата зв'язку), дБ	156	164
Швидкість передачі (адаптивна), Кб/с	0,3…5,0	0,3…5,0
Розмір повідомлення (корисні дані), байт	0…250	0…250
Тривалість передачі повідомлення, с	0,04…1,2	0,02…0,06
Енергія, витрачена на передачу одного повідомлення, мкAh	E_{tx} = 1,2 с * 32 мА = 11 мкAh при максимальній чутливості прийому E_{tx} = 40 мс * 32 мА = 0,36 мкAh при мінімальній чутливості прийому	E_{tx} = 160 мс * 11 mA = 0,5 мкAh

7.3.2 Рівень MAC LoRaWAN

LoRaWAN представляє MAC рівень, який знаходиться поверх LoRaPHY. LoRaWAN є відкритим MAC протоколом, в той час як PHY закритий. Існує три протоколи MAC, які є частиною рівня каналу передачі даних і відповідають трьом режимам роботи абонентського пристрою. Всі три режима балансують затримки і використання енергії. Клас А є найкращим для зменшення енергоспоживання при максимальній затримці. Клас В знаходиться між класом А і класом С. Клас С має мінімальну затримку, але найвищий рівень використання енергії абонентським пристроєм.

Пристрої класу А - це давачі і абонентські пристрої (кінцеві точки) на батареях. Всі кінцеві точки, які з'єднуються з мережею LoRaWAN, спочатку асоціюються як клас А з можливістю зміни класу під час роботи. Клас А оптимізує живлення, встановлюючи різні затримки прийому під час передачі. Кінцева точка починає з відправки пакета даних в шлюз. Після передачі пристрій

переходить в стан очікування до закінчення часу очікування таймера прийому. Коли час таймера закінчується, кінцева точка прокидається і відкриває слот прийому після чого очікує передачу протягом певного періоду часу, а потім знову переходить в стан очікування. Пристрій знову прокинеться після закінчення іншого таймера. Це означає, що весь низхідний зв'язок відбувається протягом короткого періоду часу після того, як пристрій відправляє пакет по висхідному каналу. Період висхідного зв'язку може бути досить довгим.

Пристрої класу B балансують енергозбереження і затримку. Цей тип пристрою покладається на сигнал маяка, що відправляється шлюзом через рівні проміжки часу. Маяк синхронізує всі кінцеві точки в мережі і транслюється в мережу. Якщо отримано сигнал маяка, він створює слот для перевірки, який є коротким вікном прийому. Під час цих періодів пристрій може відправляти і отримувати повідомлення. Будь-який інший час пристрій знаходиться в стані очікування. По суті, це сеанс, ініційований шлюзом, і заснований на методі зв'язку слотами.

Пристрої класу C використовують найбільше енергії, але мають найкоротшу затримку. Ці пристрої відкривають два вікна прийому класу A, а також вікно прийому з постійно ввімкненим живленням. Пристрої класу C зазвичай підключаються у якості виконавчих пристроїв або можуть являти собою зовнішні пристрої, що підключаються до мережі додатково у випадкові моменти. При передачі по низхідному каналу в них взагалі немає затримки. Пристрої класу C не можуть реалізовувати клас B.

Таблиця 7.3 Стек протоколів LoRa і LoRaWAN. Порівняння зі стандартною моделлю OSI

<table>
<tr><th colspan="3">Стек протокола LoRa/LoRaWAN</th><th>Модель OSI</th></tr>
<tr><td colspan="3">Прикладний рівень</td><td>7. Прикладний рівень</td></tr>
<tr><td colspan="3">Шар LoRaWAN</td><td rowspan="2">2. Канальний рівень</td></tr>
<tr><td>Клас A (базовий)</td><td>Клас B (базовий)</td><td>Клас C (безперервний)</td></tr>
<tr><td colspan="3">Модуляція Lora PHY</td><td rowspan="4">1. Фізичний рівень</td></tr>
<tr><td colspan="3">Lora PHY регіональний діапазон ISM</td></tr>
<tr><td>Lora PHY
EU діапазон 868 МГц</td><td>Lora PHY
Азія діапазон 433 МГц</td><td>Lora PHY
US діапазон 915 МГц</td></tr>
</table>

Для безпеки LoRaWAN шифрує дані з використанням моделі AES128. Одна з відмінностей безпеки LoRaWAN від інших мереж - LoRaWAN відокремлює аутентифікацію і шифрування. Аутентифікація використовує один ключ (*NwkSKey*), а призначені для користувача дані - окремий ключ (*AppSKey*). Щоб під'єднатися до мережі LoRa, пристрої відправлять запит JOIN. Шлюз відповість адресою пристрою і маркером аутентифікації. Ключ додатків і мережевого сеансу буде отримано під час процедури JOIN. Цей процес називається **Over The Air Activation (OTAA)**. В якості альтернативи пристрій на основі LoRa може використовувати активацію за допомогою персоналізації. У цьому випадку постачальник / оператор LoRaWAN попередньо розподіляє 32-розрядні мережеві і сеансові ключі, і клієнт повинен замовити план підключення і відповідний набір ключів. Ключі будуть замовлятися у виробника кінцевого пристрою з ключами, вже вбудованими в пристрій.

LoRaWAN - це асинхронний протокол на основі першої бездротової мережі ALOHA. Чистий протокол ALOHA був спочатку розроблений в Гавайському університеті в 1968 році як форма зв'язку з множинним доступом до тих пір, поки ще не існували такі технології, як CSMA. У ALOHA клієнти можуть передавати повідомлення, не знаючи, чи знаходяться ще й інші клієнти одночасно в процесі передачі. Немає ніяких застережень або методів мультиплексування. Основою з'єднання є хаб (або шлюз в разі LoRaWAN), який негайно ретранслює отримані пакети. Якщо абонентська точка виявляє, що один з її пакетів не був підтверджений зі сторони шлюза, вона буде чекати, а потім повторно передасть пакет. У LoRaWAN колізії виникають тільки в тому випадку, якщо при передачах використовуються одні й ті ж канали та один і той самий коефіцієнт розширення.

7.3.3 Топологія LoRaWAN

LoRaWAN заснований на топології зірки. В цьому відношенні можна сказати, що він підтримує зірку топології зірок. Модель LoRaWAN може використовуватися не як одна модель з хабом (шлюзом), а як кілька хабів. Вузол може бути пов'язаний з декількома шлюзами.

Критичним компонентом LoRaWAN, який відрізняє його від більшості протоколів, є той факт, що дані користувача будуть переноситися з кінцевого вузла на шлюз по протоколу LoRaWAN. У цей момент шлюз LoRaWAN відправить пакет з будь-якого транспортного об'єкту (наприклад, 4G LTE, Ethernet, Wi-Fi) на виділену мережеву послугу LoRaWAN в хмарі. Це унікально, оскільки більшість інших WAN архітектур не здійснюють будь-який контроль над даними клієнтів в той час, коли дані залишають свою мережу до місця призначення в інтернеті.

Топологія самої мережі LoRaWAN представлена на малюнку нижче:

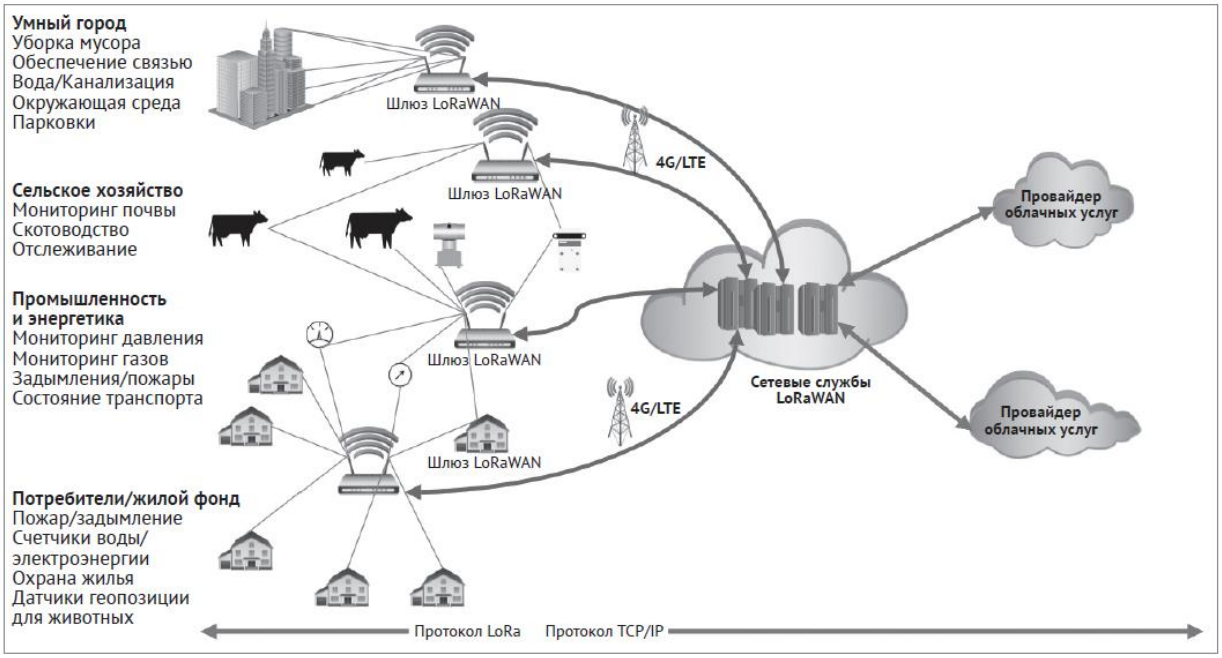


Рисунок 7.4 Мережева топологія LoRaWAN. LoRaWAN побудований на топології зірки зі шлюзами, що діють в якості концентраторів, а також агентами зв'язку в порівнянні з традиційними IP-мережами для адміністрування LoRaWAN в хмарі. Кілька вузлів можуть зв'язуватися з декількома шлюзами LoRaWAN

Мережева служба LoRaWAN (**мережевий сервер**) має правила і логіку для обслуговування необхідних верхніх рівнів мережевого стека. Побічним ефектом цієї архітектури є те, що передача обслуговування від одного шлюзу до іншого не потрібна. Якщо вузол мобільний і переміщається від антени до антени, мережева служба буде захоплювати кілька ідентичних пакетів з різних шляхів. Ця хмарна мережева служба дозволяє системам LoRaWAN вибирати кращий маршрут і джерело інформації, якщо абонентський вузол пов'язаний з декількома шлюзами. Обов'язки мережевої служби включають:

- ідентифікацію і видалення повторного пакету;

- послуги безпеки;
- маршрутизацію;
- повідомлення про підтвердження.

Крім того, LPWAN-системи, такі як LoRaWAN, матимуть у 5-10 разів менше базових станцій для аналогічного покриття мережі 4G. Всі базові станції прослуховують один і той же набір частот, тому вони логічно є однією дуже великою базовою станцією. Це, звичайно ж, підтверджує твердження про те, що системи LPWAN можуть мати більш низьку вартість, ніж традиційна стільникова мережа.

7.3.4 Недоліки LoRaWAN

У LoRaWAN є прогалини в архітектурі та протоколах, які вимагають ретельного розгляду від архітектора IoT під час вибору технічного рішення:

- деякі функції, загальні для мереж LTE, просто не призначені для архітектури LoRaWAN. LoRaWAN не є повним мережевим стеком в моделі OSI. Він не має спільних рис мережевого рівня, рівня сеансу і транспортного рівня: роумінгу, пакетування, механізмів повтору, QoS й роз'єднання. При необхідності розробник і інтегратор можуть додати ці послуги;
- LoRaWAN спирається на хмарний мережевий інтерфейс. Тому при використанні LoRaWAN необхідна платна хмарна підписка;
- постачальник чіпів Semtech є єдиним джерелом технології, хоча були оголошені партнерські відносини з STMicroelectronics;
- LoRaWAN заснований на протоколі ALOHA. ALOHA ускладнює перевірку і підтвердження пакетів, а також сприяє підвищенню частоти помилок більш ніж на 50%;
- можливості низхідного каналу зв'язку, як і раніше обмежені. LoRaWAN є принципово одностороннім протоколом. У деяких випадках для використання цього цілком досить, у інших випадках це вимагає гнучкості і додаткового алгоритмування;
- LoRaWAN має високу затримку і не може працювати в режимі реального часу;

- немає оновлень прошивки OTA (Online Trust Alliance) - OTA- це всесвітня некомерційна організація, яка розробляє кращі рішення для IoT-безпеки;
- мобільними і рухомими об'єктами складніше управляти в LoRaWAN. Повідомлення довжиною від 40 до 60 байтів може мати час передачі від 1 до 1,5 с. Це зразу виключає управління транспортними засобами. Базова станція також вимагає незмінності лінійного інваріантного часу (LTI), що означає, що радіохвиля не повинна змінювати час проходження від абонентського пристрою до шлюза і навпаки.

7.4 Висновки та порівняння стільникової та пропрієтарної мережі

Незважаючи на деяку спільність між типами технологій зв'язку NB та LoRa на великі відстані, вони також націлені на різні варіанти використання і різні сегменти. Архітектор IoT повинен правильно вибирати, яку саме систему слід прийняти. Як і інші компоненти системи IoT, LPWAN важко змінити після розгортання. Міркування щодо вибору правильного LPWAN включають наступні питання:

- яку швидкість передачі даних необхідно використовувати для розгортання IoT?
- чи може рішення бути масштабованим з тою самою LPWAN по регіонах? Чи є у всіх потрібних місцях належне охоплення/покриття або потрібно його побудувати?
- яка дальність передачі потрібна?
- чи існує будь-яка вимога латентності для цього рішення IoT? Чи може рішення працювати з дуже високими (декількома секундами) затримками?
- чи абонентські точки IoT живляться від батарей і яка буде вартість їх обслуговування?
- які існують обмеження витрат на абонентські точки?

Нижче наведена порівняльна таблиця LPWAN технологій.

Таблиця 7.4 Порівняльна таблиця LPWAN технологій

Спецификация	Cat-0 (LTEM) релиз 12	Cat-1 релиз 8	Cat-M1 релиз 13	Cat-NB релиз 13	LoRa/LoRaWAN	Sigfox
Полосы ISM	Нет	Нет	Нет	Нет	Да	Да
Общая полоса пропускания	20 МГц	20 МГц	1,4 МГц	180 КГц	125 КГц	100 КГц
Пиковая нагрузка на нисходящем направлении	1 Мб/с	10 Мб/с	1 Мб/с или 375 Кб/с	200 Кб/с	0,3–5 Кб/с	100 б/с
Пиковая нагрузка на восходящем направлении	1 Мб/с	5 Мб/с	1 Мб/с или 375 Кб/с	200 Кб/с	5 Кб/с до 5 Кб/с адаптивно	600 б/с
Дальность	LTE	LTE	~4x Cat-1	~7xCat-1	5 км в городе, 15 км в пригороде	До 50 км
Maximum Coupling Loss (MCL)	142,7 дБ	142,7 дБ	155,7 дБ	164 дБ	165 дБ	168 дБ
Энергопотребление в спящем режиме	Низкое	Высокое: ~2 мА	Очень низкое: ~15 мкА	Очень низкое: ~15 мкА	Экстремально низкое: 1,5 мкА	Экстремально низкое: 1,5 мкА
Конфигурация дуплекса	Полу/полный	Полный	Полу/полный	Полный	Полный	Полный
Антенны (MIMO)	1	2 MIMO	1	1	1	1
Задержка	50–100 мс	50–100 мс	10–15 мс	1,6–10 мс	500 мс – 2 с	До 60 с
Мощность передачи (UE)	23 дБ	23 дБ	20 дБ	23 дБ	14 дБ	14 дБ
Сложность разработки	50% от Cat-1	Сложно	25% от Cat-1	10% от Cat-1	Несложно	Несложно
Стоимость (относительная)	~ $15	~$30	~$10	~$5	~$15	~$3
Мобильность	Мобильный	Мобильный	Мобильный	Ограниченно	Мобильный	Ограниченно

Контрольні питання

1. Організація стільникової мережі зв'язку. Намалювати і пояснити.
2. Які категорії абонентського обладнання для використання з NB-LTE ви знаєте? Короко охарактеризувати їх. Які категорії використовують українські мобільні оператори?
3. У чому полягає різниця між категоріями M1 та NB 13 релізу? Яка категорія використовується в Україні?
4. Які ви знаєте характеристики 13 категорії NB-LTE, що дозволена до використання в Україні?
5. Особливості протоколу NB при порівнянні з M1.
6. Навести варіанти розгортання Cat-NB: в якості захисного діапазону, всередині GSM-сигналу або всередині смуги з LTE.
7. Чи можливо використання Cat-NB 13 релізу у якості мобільного протоколу?
8. Особливості керування енергозбереженням у пристроях IoT NB-LTE.
9. Історія створення та сьогодення пропрієтарної технології далекої дії LoRa.
10. В яких діапазонах працю LoRa? Які з цих діапазонів дозволені до застосування в Україні?

11. Що таке метод модуляції CSS (чірп-модуляція)? Охарактеризувати його можливості.
12. Яка швидкість передачі цифрового сигналу у технології LoRa? Що таке коефіцієнт розширення? Як швидкість передачі залежить від коефіцієнту розширення та смуги пропускання каналу?
13. Яка довжина пакету даних у технології LoRa? Що таке ADR?
14. Властивості висхідного і низхідного каналу зв'язку для LoRa.
15. Охарактеризувати термінальні LoRa пристрої класів A, B і C.
16. До якого класу протоколів, синхронних чи асинхронних, відноситься протокол LoRaWAN?
17. Що називається OTAA процесом в LoRa?
18. Описати мережову топологію LoRa. Що таке мережовий сервер? Його функції?
19. Недоліки сенсорної мережі на основі протокола LoRaWAN.

8 ГРУПОВЕ МЕРЕЖОВЕ ОБЛАДНАННЯ ДЛЯ IIoT

Існує два методи формування мережі IIoT:

- крайові (граничні) давачі і пристрої самостійно забезпечують **прямий шлях до хмари**. Це означає, що ці вузли і давачі граничного рівня матимуть достатньо ресурсів, апаратних засобів, програмного забезпечення і угод про рівень обслуговування для прямої передачі даних через WAN до хмари;
- крайові давачі **утворюють групи і кластери навколо шлюзів і маршрутизаторів,** які вже і забезпечують перетворення протоколів та можливості обробки на границі / в тумані і будуть керувати безпекою й аутентифікацією між давачами і глобальною мережею.

Перша модель є складною і дорогою, оскільки використовує потужні і дорогі давачі / перетворювачі / пристрої граничного рівня. Більш логічним буде другий варіант. Роль граничного маршрутизатора або шлюзу для давачів / пристроїв включає до себе усі мережеві можливості, які надають сучасні маршрутизатори, такі як маршрутизація пакетів, переадресація портів, туннелювання, безпека і резервування. Принципи маршрутизації і ретрансляції TCP/IP вивчалися під час бакалаврату і тут повторюватися не будуть. У цій темі буде розглянуто роль і необхідність маршрутизаторів граничного рівня, а також наведено поради та рекомендації щодо функцій таких маршрутизаторів, які слід враховувати при розгортанні масового рішення IIoT [3].

8.1 Функції шлюза

У нашому курсі ми попередньо розглянули кілька типів бездротового зв'язку, кожен з яких вимагає власної технології та протоколів передачі сигналу, а також механізм для перетворення трафіку однієї технології в іншу при підключенні до Інтернету, чи іншої зовнішньої мережі. Це найважливіша роль граничного шлюзу IIoT. Незалежно від того, чи являє собою сенсорна мережа Bluetooth-мережу, що вимагає граничного вузла типу моста, або базовий вузол eNodeB LTE мережі, - ролі аналогічні. Шлюз перетворює і пересилає дані між двома технологічно різними мережами.

Найчастіше у якості шлюзу використовують маршрутизатор. Сам по собі маршрутизатор здійснює направлення і керування трафіком між подібними мережами. В граничній архітектурі IIoT може одночасно існувати суміш декількох видів сенсорних мереж, наприклад mesh-Bluetooth, 802.11ah та NB-LTE. Трафік від усіх цих мереж повинен безперешкодно, підтримуючи функції безпеки, проходити в обох напрямках в хмару Інтернет, чи до спеціалізованої мережі.

Часто граничний шлюз також є центральним контролером PAN. Це означає, що всі функції управління мережею PAN, забезпечення безпеки, аутентифікація і підключення нових вузлів, даних щодо управління та пристроїв управління живленням відносяться до відповідальності граничного шлюзу.

У даний час все більше і більше розповсюджуються рішення, коли промисловий шлюз чи маршрутизатор вбудовуються у корпус для монтування на DIN-рейку і окрім виконання функцій маршрутизації може керувати і групою ПЛК. Прикладом таких рішень є рішення eWON компанії **Klinkman**.

8.2 Різновиди маршрутизації, що використовуються на граничних пристроях

Основна функція маршрутизатора - це забезпечення з'єднань між сегментами мережі. Маршрутизація вважається функцією третього рівня стандартної моделі OSI, оскільки вона використовує рівень IP-адресації для управління передачею пакетів. По суті, всі маршрутизатори покладаються на таблицю маршрутизації для управління потоком даних. Таблиця маршрутизації використовується для пошуку найкращого шляху доставки пакета відповідній IP-адресі призначення.

Існує кілька перевірених алгоритмів, використовуваних для ефективної маршрутизації. Одним з типів маршрутизації є динамічна маршрутизація, де алгоритми реагують на зміни в мережі і топології. При такій маршрутизації інформація про стан мережі поширюється протоколом маршрутизації з часом або при виникненні кожного оновлення мережі. Прикладами динамічної маршрутизації є маршрутизація вектора відстані і маршрутизація стану каналу. В якості альтернативи використовується і статична маршрутизація, яка важлива і корисна для невеликих мереж, яким потрібні певні шляхи, вже налаштовані між

маршрутизаторами. Статичні маршрути неадаптивні, тому тут немає необхідності сканувати топологію або оновлювати метрики мережі.

Алгоритми маршрутизації вбудовуються у маршрутизатор і поділяються на такі види:

- **маршрутизація по найкоротшому шляху** - побудова графа, що представляє маршрутизатори в мережі. Дуга між вузлами є відомий зв'язок або з'єднання між ними. Алгоритм просто знаходить найкоротший шлях з будь-якого вихідного джерела до будь-якого пункту призначення;
- **лавинна маршрутизація** - кожен пакет повторюється і транслюється кожним маршрутизатором в кожну кінцеву точку його зв'язків. Це генерує величезну кількість дублюючих пакетів і вимагає, щоб в заголовку пакета був встановлений лічильник переходів для того, щоб гарантувати той факт, що пакети мають обмежений час життя. Альтернативою є вибіркова розсилка, яка наповнює мережу тільки в основному напрямку пункту призначення. Лавинні мережі є основою mesh-мереж Bluetooth;
- **маршрутизація на основі потоку** - перевіряє поточні потоки мережі для визначення шляху. Для будь-якого даного з'єднання, якщо відомі пропускна здатність і середній потік, обчислюється середня затримка пакета по цьому з'єднанню. Цей алгоритм знаходить шлях, який має мінімальне середнє значення затримки;
- **маршрутизація на основі вектора відстані** - таблиця маршрутизатора містить відому відстань до кожного пункту призначення. Таблиці оновлюються сусідніми маршрутизаторами. Таблиця містить запис для кожного маршрутизатора в підмережі. Кожен запис містить кращий маршрут/шлях і приблизну очікувану відстань до пункту призначення. Відстань може бути метрикою кількості переходів, затримки або довжини черги;
- **маршрутизація за станом каналу** - маршрутизатор спочатку виявляє всіх своїх сусідів за допомогою спеціального пакета HELLO. Маршрутизатор вимірює затримку до кожного зі своїх сусідів шляхом відправки пакета ECHO. Ця інформація про топологію і час потім розповсюджується серед

маршрутизаторів в підмережі. Повна топологія будується і публікується всіма маршрутизаторами;

- **ієрархічна маршрутизація** - маршрутизатори розділені на регіони ієрархічної топології. Кожен маршрутизатор підтримує знання тільки про свій регіон, але не про всю мережу. Ієрархічна маршрутизація є ефективним засобом управління розміром таблиці маршрутизації і ресурсами в пристроях з ресурсними обмеженнями;
- **широкомовна маршрутизація** - кожен пакет містить список адрес призначення. Широкомовний маршрутизатор досліджує адреси і визначає набір вихідних ліній для передачі пакета. Маршрутизатор буде генерувати новий пакет для кожної вихідної лінії і включати до нього тільки адреси, необхідні для передачі по цій лінії;
- **групова (багатоадресна) маршрутизація** - мережа розділена на чітко визначені групи. Додаток може відправляти пакет на всю групу, а не на окрему адресу чи широкомовно.

Типові граничні маршрутизатори повинні підтримувати такі протоколи маршрутизації, як **Border Gateway Protocol (BGP)**, **Open Shortest Path First (OSPF)**, **Routing Information Protocol (RIP)** і **RIPng**. Архітектор, який використовує граничні маршрутизатори в реальній роботі, повинен знати про обмеження і вартість використання певного протоколу маршрутизації в порівнянні з іншими, особливо якщо з'єднання між маршрутизаторами є WAN-з'єднанням з обмеженням даних:

BGP - BGP-4 є стандартом для протоколів інтернет-доменної маршрутизації і описаний в RFC 1771; він використовується більшістю інтернет-провайдерів. BGP - це алгоритм динамічної маршрутизації на основі вектора відстані, він сповіщає про повні шляхи в повідомленнях оновлень маршрутизації. Якщо таблиці маршрутизації є великими, це буде вимагати значної пропускної здатності. При використання BGP маршрутизатор відправляє 19-байтове повідомлення *keepalive* кожні 60 с для підтримки з'єднання. BGP може бути слабким протоколом маршрутизації для топології mesh-мережі, оскільки BGP підтримує з'єднання з сусідами. Недоліком BGP також є зростання таблиць

маршрутизації у великих топологіях. BGP є унікальним, оскільки він єдиний з протоколів маршрутизації використовує інформацію з TCP-пакетів;

OSPF - цей протокол описаний в RFC 2328, він забезпечує переваги масштабування мережі і збіжності. Інтернет-мережа і корпоративні мережі інтенсивно використовують OSPF. OSPF - це алгоритм стану з'єднання, який підтримує IPv4 та IPv6 (RFC 5340) і працює з IP-пакетами. Перевага полягає в виявленні за секунди динамічних змін з'єднань і реагуванні на них;

RIP - друга версія RIP являє собою алгоритм маршрутизації на основі вектора відстані, і заснована на підрахунку числа переходів (хопів) між вузлами з використанням протоколу внутрішнього шлюзу. Спочатку він був заснований на алгоритмі Беллмана-Форда, тепер він підтримує підмережі динамічно змінюваного розміру, долаючи обмеження вихідної версії. Можливі петлі в таблиці маршрутизації обмежуються в ньому завданням максимально-допустимої кількості переходів для пакета. RIP працює по UDP і підтримує тільки трафік IPv4. RIP має більш тривалий час збіжності, ніж такі протоколи, як OSPF, але його легко адмініструвати для топологій невеликого граничного маршрутизатора. Проте, збіжність для RIP з усього кількома маршрутизаторами може зайняти кілька хвилин;

RIPng - RIPng означає RIP наступного покоління (RFC 2080). Він дозволяє підтримувати трафік IPv6 і IPsec для аутентифікації.

Типова таблиця маршрутизації, яка використовується у виробничому середовищі таким маршрутизатором, як Cradlepoint IBR900, виглядає так, як на рис. 8.1 нижче.

Цей приклад містить **три** таблиці маршрутизації: *wan, main і local*. Кожна таблиця містить конкретні шляхи маршрутизації, характерні саме для цього інтерфейсу:

Destination - повна або часткова IP-адреса призначення пакета. Якщо таблиця містить IP, вона буде посилатися на решту запису, щоб дозволити інтерфейс для маршрутизації на часткову адресу; для чого може бути задана з префіксом /. Цей префікс вказує фіксовані позиції бітів адреси для дозволу. Наприклад, /24 в 192.168.1.0/24 вказує, що старші 24 біта 192.168.1 є

фіксованими, а молодші 8 біт можуть мати будь-яку адресу в підмережі 192.168.1. *;

```
[administrator@IBR900-e11: /]$ route
Table: wan
Destination        Gateway     Device UID       Flags    Metric   Type
default            96.19.152.1  wan             onlink   0        unicast

Table: main
Destination        Gateway     Device UID       Flags    Metric   Type
96.19.152.0/21     *             wan                     0        unicast
172.86.160.0/20    *             iface:pertino0          0        unicast
172.86.160.0/20    None          None                    256      blackhole
192.168.1.0/24     *             primarylan              0        unicast
2001:470:813b::/48 *             *iface:pertino0         256      unicast
fe80::/64          *             lan                     256      unicast

Table: local
Destination       Gateway     Device UID       Flags    Metric   Type
96.19.152.0        *             wan                     0        broadcast
96.19.153.13       *             wan                     0        local
96.19.159.255      *             wan                     0        broadcast
127.0.0.0          *             *iface:lo               0        broadcast
.
.
.
```

Рисунок 8.1 Типова таблиця маршрутизації для промислового роутера Cradlepoint IBR900

Gateway - це інтерфейс для направлення пакетів, що відповідає знайденому адресату. У попередньому випадку шлюз вказаний як 96.19.152.1, а пункт призначення - default. Це означає, що вихідна WAN-адреса 96.19.152.1 буде використовуватися для всіх адрес призначення. Це по суті, пересилання IP;

DeviceUID - це буквено-цифровий ідентифікатор інтерфейсу для відправки даних. Наприклад, будь-яке місце призначення в підмережі 172.86.160.0/20 направить пакети в інтерфейс під назвою *iface:pertino0*. Часто це поле буде виражатися цифровою IP-адресою, а не символічним посиланням;

Flags - використовується для діагностики і вказівки стану маршруту;

Metric - це відстань до місця призначення, зазвичай підраховується за кількістю переходів;

Type - можуть використовуватися кілька типів маршрутів:

- *unicast*: маршрут є реальним шляхом до місця призначення;

- *unreachable*: місце призначення недоступне. Пакети відкидаються, і генерується повідомлення ICMP, яке вказує, що хост недоступний. Місцевий відправник отримує помилку EHOSTUNREACH;

- *blackhole*: одержувачі недосяжні. На відміну від типів «*prohibit*», пакети будуть тихо відкинуті. Локальні відправники отримують помилку EINVAL;

- *prohibit*: одержувачі недосяжні. Пакети будуть відкидатися і генеруються ICMP-повідомлення. Локальні відправники отримують помилку EACCES;

- *local*: одержувачем є цей хост. Пакети завертаються назад і доставляються локально;

- *broadcast*: пакети будуть передаватися як широкомовні через інтерфейс до всіх одержувачів;

- *throw*: спеціальний маршрут управління, щоб насильно відкинути пакети і згенерувати повідомлення ICMP про недоступність.

У наведеному вище прикладі слід зазначити, що адреси IPv6 змішуються з адресами IPv4. Наприклад, 2001:470:813b::/48 в основній таблиці – є адресою IPv6 с /48-бітною підмережею.

8.3 Відмовостійкість і керування по окремому каналу (позасмугове керування)

Відмовостійкість критична для деяких граничних маршрутизаторів IoT, особливо для застосувань, що працюють з транспортом і по догляду за пацієнтами. Відмовостійкість, як випливає з назви, - це переключення з одного інтерфейсу WAN на інший, коли основний канал втрачений. Втрата WAN, наприклад, може бути пов'язана з втратою стільникового зв'язку в тунелі. Компанії-перевізнику з автопарком рефрижераторів може знадобитися гарантоване підключення, при тому що стан послуги стільникового зв'язку змінюється по всій країні. Переключення у момент зникнення основного зв'язку від одного стільникового оператора до іншого з використанням декількох SIM-карт може допомогти пом'якшити і згладити перепідключення. Іншим варіантом може бути використання клієнтського Wi-Fi в якості основного інтерфейсу WAN для внутрішнього моніторингу стану пацієнта, але перепідключення до стільникового WAN заради відмовостійкості, якщо сигнал Wi-Fi втрачений. Відмовостійкість повинна бути безболісною і автоматичною без втрати пакетів або помітного впливу на затримку даних.

Керування по окремому каналу (*Out-of-Band Management (OOBM)*) також має розглядатися для пристроїв IoT. OOBM корисний для забезпечення

відмовостійкості, коли для управління обладнанням необхідний виділений і ізольований канал. При використанні OOBM якщо первинні системи вийшли з ладу, пошкоджені, чи фунціонують нештатно, лишається можливість керувати і перевіряти обладнання віддалено через додатковий окремий канал, що не перетинається з основним. У IoT це може бути корисно для ситуацій, що вимагають гарантованого часу безвідмовної роботи і дистанційного керування, таких як моніторинг нафти і газу, або промислова автоматизація. Добре функціонуюча система OOBM не повинна мати ніякого відношення до контрольованої нею системи. Типові вимоги до OOBM, у якості якої використовують віртуальні тунелі або SSH, вимагають, щоб пристрій було можливо як перезавантажувати, так і керувати ним в умовах втрати основного каналу.

Таким чином приходимо до того, що OOBM повинен бути додатковим і ізольованим від системи, як це показано на рис. 8.2 нижче.

Рисунок 8.2 Приклад конфігурації для позасмугового керування

8.4 Використання VLAN

VLAN функціонує, як будь-яка інша фізична локальна мережа, але дозволяє групувати комп'ютери і інші пристрої, навіть якщо вони фізично не прив'язані до одного мережового комутатора. Поділ відбувається на рівні DLL (другий рівень) моделі OSI. VLAN - це форма мережевої сегментації пристроїв, додатків або користувачів, хоча вони і знаходяться в одній і тій самій фізичній мережі. VLAN також може групувати хости разом, хоча вони не знаходяться на одному мережевому комутаторі, що істотно полегшує розбиття на підмережі без

використання додаткових кабелів. Стандарт IEEE 802.1Q - це стандарт, за яким побудовані VLAN. По суті, VLAN використовує ідентифікатор або тег, що складається з 12 біт кадра Ethernet. Таким чином, існує жорстке обмеження в 4096 потенційних VLAN в одній фізичній мережі.

Керований комутатор може призначити порт для безпосередньої прив'язки до певної VLAN. Оскільки VLAN працює на другому рівні OSI, трафік може тунелюватись через третій рівень, що дозволяє географічно розділеним VLAN використовувати загальну топологію (Рис 8.3).

На малюнку показана корпоративна точка продажу (POS) і система VoIP, яка фактично ізольована від набору пристроїв IoT, а також гостьового Wi-Fi. Це робиться за допомогою адресації VLAN, хоча система використовує одну і ту ж фізичну мережу. Тут ми припускаємо, що це розумне розгортання Інтернету, де всі граничні пристрої та давачі IoT працюють з використанням стека IP і адресуються через локальну мережу. Дизайн VLAN особливо корисний в світі IIoT. Ізолювання пристроїв IIoT від інших корпоративних підмереж є типовим сценарієм. Але VLAN корисні тільки при роботі з пристроями, що підтримують IP-адресацію.

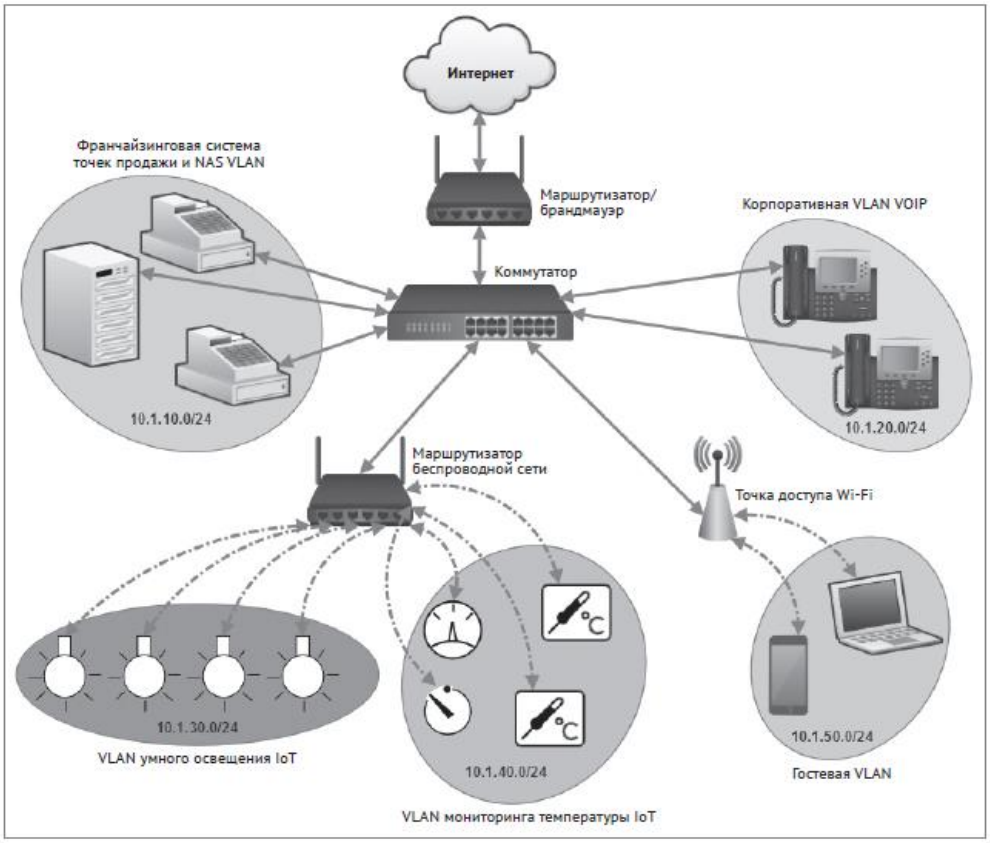


Рисунок 8.3 Приклад архітектури VLAN в сценарії філій або роздрібної торгівлі

8.5 Використання VPN-тунелей

VPN-тунелі використовуються для встановлення безпечного підключення до віддаленої мережі через публічну мережу. Наприклад, VPN-тунелі можуть використовуватися для окремого користувача для підключення через Інтернет до захищеної корпоративної мережі під час поїздок або для об'єднання двох офісних географічно розподілених мереж. Дві мережі налаштовують безпечне з'єднання через (зазвичай) незахищений Інтернет, застосовуючи протоколи шифрування VPN.

Для впровадження IIoT для передачі даних з віддалених давачів і периферійних пристроїв в корпоративну або приватну локальну мережу необхідна VPN. Як правило, корпорація буде знаходитися за брандмауером, а VPN - це єдиний спосіб переміщення даних безпосередньо на приватний локальний сервер.

Існує декілька типів VPN:

Internet Protocol Security (IPSec) VPN - традиційна технологія VPN, яка

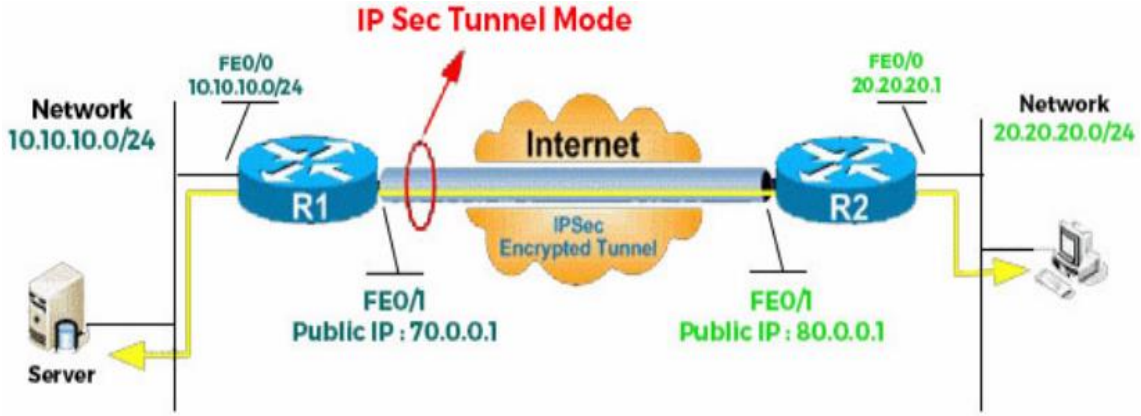


Рисунок 8.4 Ілюстрація роботи протоколу IPSec

працює на мережевому рівні стека OSI і забезпечує передачу даних через тунель між двома кінцевими точками (рис. 8.4). Являє собою набір протоколів для забезпечення захисту даних, що передаються за допомогою протоколу IP, дозволяє здійснювати підтвердження справжньості та/або шифрування IP-пакетів. IPsec також містить в собі протоколи для захищеного обміну ключами в мережі Інтернет;

OpenVPN - це вільна реалізація технології віртуальної приватної мережі

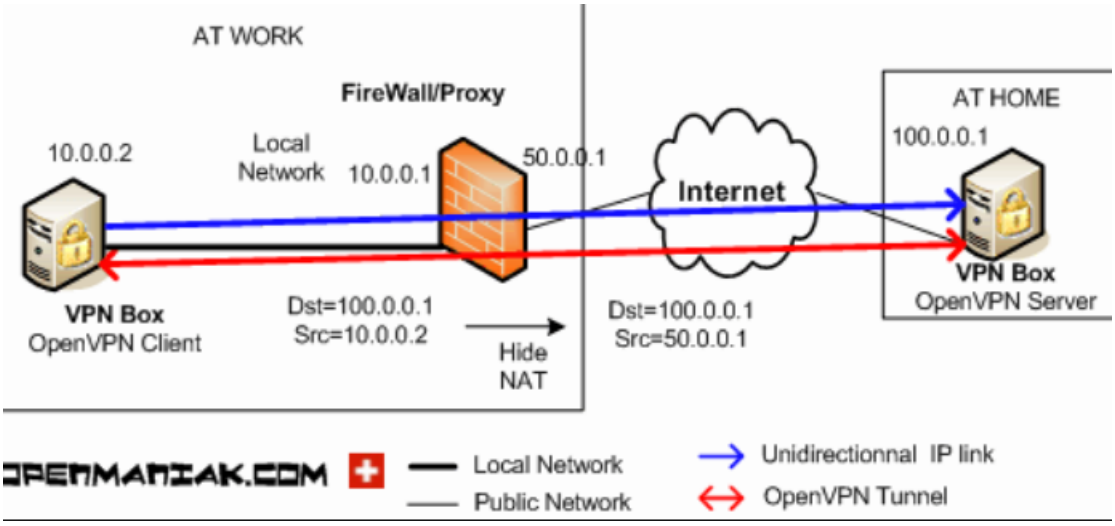


Рисунок 8.5 Ілюстрація роботи Open VPN

(VPN) з відкритим початковим кодом для створення шифрованих з'єднань між двома клієнтськими машинами або забезпечення роботи централізованого VPN-сервера для одночасної роботи декількох клієнтів. OpenVPN дозволяє встановлювати з'єднання між комп'ютерами, що перебувають за NAT-екраном, без необхідності зміни їхніх налаштувань. Підтримує з'єднання «точка-точка» і «сайт-сайт» в маршрутизованих або мостових конфігураціях. Включає спеціальний протокол безпеки, який використовує *SSL/TLS* (OpenSSL) для обміну ключами і управління шифруванням і обміну даними. Цей тип може працювати через транспорт UDP і TCP. SSL поширений в більшості додатків для браузерів, тому SSL VPN-системи можуть забезпечувати захищені тунелі не тільки для всієї мережі, а і для окремих програмових додатків;

Generic Routing Encapsulation (GRE) – розробка компанії Cisco, створює з'єднання «точка-точка» між кінцевими точками через тунель, аналогічний тунелю VPN, але інкапсулює його корисне навантаження (рис. 8.6). Протокол обгортає внутрішній пакет у зовнішній пакет. Це дозволяє передавати дані корисного навантаження через інші IP-маршрутизатори та тунелі незміненими. Крім того, тунелі GRE можуть транспортувати IPv6 і багатоадресні передачі;

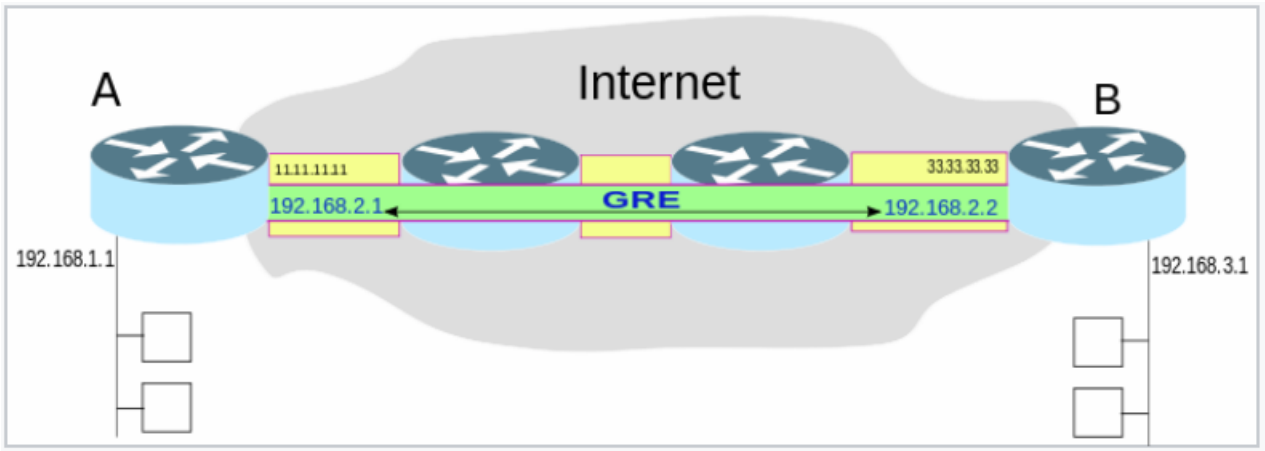


Рисунок 8.6 Приклад роботи GRE-тунелю. Між двома маршрутизаторами A і B знаходиться кілька маршрутизаторів, тунель дозволяє забезпечити зв'язок між сегментами мережі 192.168.1.0/24 і 192.168.3.0/24 так, як якщо б маршрутизатори A і B були з'єднані прямим лінком.

Layer 2 Tunneling Protocol (L2TP) - створює з'єднання між двома приватними мережами через дейтаграми UDP, типово використовується для VPN, або як частина служб доставки провайдерами. У протоколі немає вбудованої безпеки або шифрування, і для цього часто використовується IPsec.

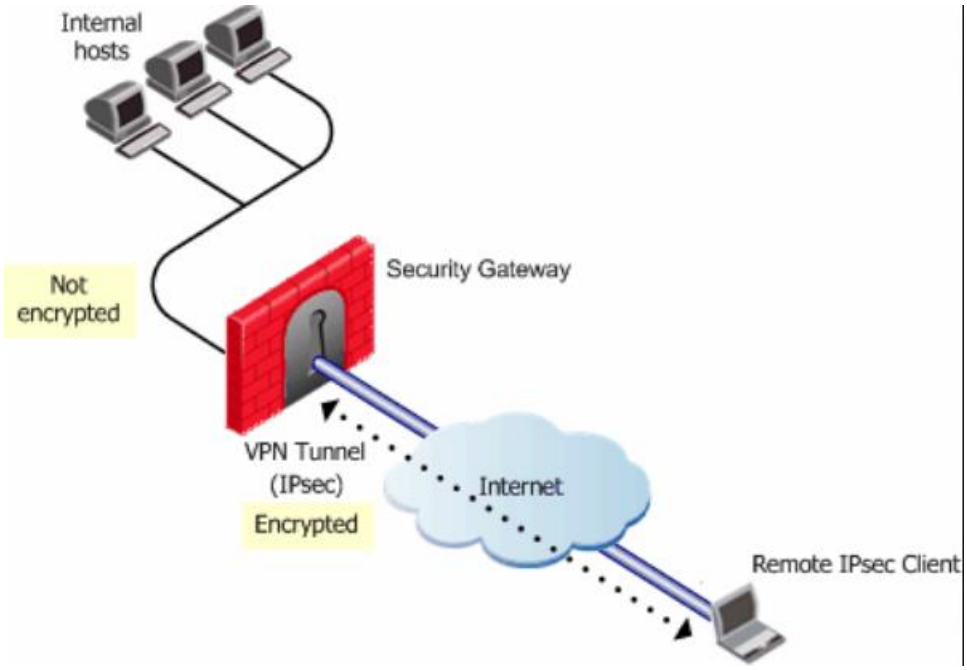


Рисунок 8.7 Робота протоколу L2TP

8.6 Управління швидкістю трафіка (шейпер) і QoS

Функції управління швидкістю трафіку і якості послуг (QoS) корисні при розгортанні, яке вимагає гарантованого рівня обслуговування при роботі з перевантаженнями або змінними навантаженнями на мережу. Наприклад, в разі використання IIoT при змішуванні живих відеопотоків і публічного Wi-Fi, відеопотоки можуть вимагати пріоретізації і гарантованого рівня якості,

особливо для громадської безпеки або спостереження. Залишені, як є, дані що надходять від глобальної мережі до граничного маршрутизатора будуть обслуговуватися за принципом «першим прийшов - першим пішов»:

функції QoS - дозволяють адміністратору призначати рівні пріоритету для заданої IP-адреси, або певного порту, розміщеного на маршрутизаторі. Функції QoS керують тільки висхідним каналом зв'язку. Вони особливо корисні в тих випадках, коли висхідний канал зв'язку має набагато меншу пропускну здатність, ніж нисхідний. Як правило, споживчий широкосмуговий доступ матиме на висхідній лінії щось біля 5 Мбіт/с, а на низхідній – 100 Мбіт/. QoS, при цьому, в змозі забезпечити спосіб балансування навантаження на обмежену висхідну лінію. QoS не призначає жорсткі ліміти і не сегментує з'єднання, як це робить управління швидкістю трафіку;

функції управління швидкістю трафіку - управління швидкістю трафіку - це статична форма зумовлення смуги пропускання. Наприклад, уся смуга 15 Мбіт/с може бути розділена на менші сегменти по 5 Мбіт/с. Ці сегменти повинні бути призначені заздалегідь. Як правило, це зайві витрати, оскільки виділені частини смуги пропускання можуть не звільнятися в разі потреби;

динамічне управління швидкістю і пріоритет пакетів - сучасні маршрутизатори підтримують атрибути динамічного управління швидкістю. Це дозволяє адміністратору динамічно призначати правила сегментації смуги пропускання для вхідного і вихідного трафіку. Він також може управляти чутливими до затримок пакетами (наприклад, відео або призначеним для користувача інтерфейсом) для додатків реального часу. Динамічне управління швидкістю і пріоритет пакетів дозволяють створювати правила на основі типу даних або програми, а не тільки IP-адреси або порту.

8.7 Функції безпеки

Граничний маршрутизатор або шлюз виконує ще одну важливу задачу, забезпечуючи рівень безпеки між WAN (Інтернетом) і базовими пристроями PAN/IIoT. Багатьом пристроям не вистачає необхідних ресурсів, пам'яті і обчислювальної потужності для забезпечення власної надійної безпеки. Незалежно від того, чи створює архітектор свій власний шлюзовий сервіс або

користується готовим, для захисту компонентів IIoT слід враховувати наступні рекомендації.

Захист за допомогою брандмауера - є базовою формою безпеки. Існують дві основні форми брандмауерів для телекомунікацій. Перший - це **мережевий брандмауер**, який фільтрує і управляє потоком інформації з однієї мережі в іншу. Другий - це **брандмауер на базі хоста**, який локально захищає додатки і служби на цій машині. У разі граничних маршрутизаторів IIoT ми фокусуємося на мережевих брандмауерах. За замовчуванням брандмауер запобігає проникненню певних типів мережевого трафіку в зону, захищену брандмауером, але будь-який трафік, що виходить з цієї зони, може йти назовні. Брандмауер знайде і ізолює інформацію на основі пакетів, станів чи програмних застосувань в залежності від складності брандмауера. Як правило, зони створюються з мережних інтерфейсів з правилами, призначеними для управління потоком трафіку між ними. Прикладом може служити граничний маршрутизатор з гостьовою зоною Wi-Fi і корпоративною приватної зоною.

Пакетний брандмауер може ізолювати і придушувати певний трафік на основі IP-адреси джерела або одержувача, портів, MAC-адрес, IP-протоколів та іншої інформації, що міститься в заголовку пакета. Брандмауер працює на четвертому (мережовому) рівні стека OSI. Він збирає і об'єднує пакети, шукаючи шаблони і інформацію про стан, таку як нові з'єднання і існуючі з'єднання. Фільтрація додатків виконується складніше, так як вона може здійснювати пошук за певними мережевими потоками програмних додатків, включаючи FTP-трафік або HTTP-дані.

Брандмауер також може використовувати демілітаризовану зону (DMZ). DMZ - це просто логічна зона на одному з портів брандмауера. DMZ — технологія забезпечення захисту інформаційного периметра організації, при якій сервери, що відповідають на запити з зовнішньої мережі, перебувають в особливому сегменті мережі (який і називається DMZ) і обмежені в доступі до основних сегментів мережі (таких, як корпоративна чи промислова мережа) за допомогою міжмережевого екрану (файрвола) з метою мінімізування збитків при зломі одного із загальнодоступних сервісів, які знаходяться в DMZ. Хост у зоні DMZ ефективно не захищений брандмауером в тому сенсі, що будь-який

комп'ютер в Інтернеті може віддалено отримати доступ до нього за його IP-адресою. Типові види використання включають запуск спільного веб-сервера і обмін файлами. Хост DMZ зазвичай задається прямою IP-адресою.

Переадресація портів - це концепція, що дозволяє відкривати певні порти за брандмауером. Для кількох пристроїв IIoT необхідний відкритий порт для надання сервісів, які контролюються хмарними компонентами. Знову ж, можливо побудувати правило, яке дозволяє за вказаною IP-адресою в захищеній зоні брандмауера мати відкритий порт.

Як DMZ, так і переадресація портів відкривають порти і інтерфейси у взагалі-то захищеній мережі. При цьому слід проявляти обережність, щоб це дійсно відповідало намірам архітектора. При масовому розгортанні IoT з безліччю граничних маршрутизаторів загальне правило для відкриття порту може бути корисно для одного пункту, але для іншого буде істотним ризиком для безпеки. Крім того, по закінченні розгортання слід періодично проводити аудит DMZ і відкритих портів, оскільки мережева топологія і конфігурації змінюються з часом. Слід мати на увазі, що погано налаштований DMZ в будь-який момент може привести до відкритої діри в мережевій безпеці.

8.8 Метрики і аналітика

У багатьох випадках граничні пристрої IIoT будуть предметом угод рівня обслуговування (SLA) або обмеженнями для передачі даних, як у випадку 4G LTE. В інших ситуаціях граничний маршрутизатор або шлюз являє собою безліч інших мереж PAN і пристроїв IIoT. Він повинен служити центральним (але місцевим) органом по нагляду за станом мережі / mesh-мережі PAN. Метрики і аналітика, існуючі на ньому, корисні для збору і вирішення проблем з підключенням і вартістю, особливо в міру збільшення масштабів IIoT. Типові метрики і збірки повинні включати наступне:

- аналітика часу безперебійної роботи WAN - історичні тенденції, рівні обслуговування;
- використання даних - вхід, вихід, інтегрована інформація для кожного клієнта, для кожної програми;

- смуга пропускання - випадковий або запланований аналіз смуги пропускання на вході і виході;
- стан PAN - пропускна здатність, аномальний трафік, реорганізація mesh-мережі;
- цілісність сигналу - сила сигналу;
- місце розташування - GPS-координати, переміщення, зміна місцеположення;
- контроль доступу - підключення клієнтів, вхід адміністратора в систему, зміни конфігурації маршрутизатора, успіх / невдача аутентифікації PAN;
- відмовостійкість - кількість, час і тривалість подій відмови.

Ці типи метрик повинні збиратися і контролюватися за розкладом автоматично. Крім того, просунуті маршрутизатори повинні мати можливість створювати правила і попередження, коли на границі виникають певні події або фіксується аномальна поведінка.

8.9 Обробка на границі

Граничні маршрутизатори здатні надавати обчислювальні ресурси близько до того місця, де генеруються дані. Це важлива характеристика в граничних маршрутизаторах, особливо для рішень IIoT. Оскільки більше рішень IIoT побудовано в віддалених і різноманітних місцях, наявність можливості локальних обчислень важлива для багатьох випадків використання.

Обчислення на границі може впливати на дані користувача за допомогою таких методів, як:

- фільтрація і агрегація;
- зміна даних;
- аналіз безпеки і виявлення вторгнень;
- ключовий менеджмент;
- процесори правил движків / подій;
- кешування і зберігання.

Граничні маршрутизатори будуть відрізнятися за розміром і обчислювальною потужністю. Це не сервери центрів обробки даних або стоєчне

обладнання. Як правило, характеристики граничного маршрутизатора будуть такими, як показано в наступній таблиці:

Таблиця 8.1 Приклади граничних маршрутизаторів

Свойство	Потребительский маршрутизатор	Граничный маршрутизатор среднего уровня	Туманный узел высокого уровня
Марка	Apple Airport Extreme	Advantech WISE-3310	HP Edgeline EL20 Gateway
MSRP	~ $199	~ $547	~ $1400
SOC/процессор Mfr.	Broadcom 53019	Freescale i.MX6	Intel 4300U
Тип и скорость ЦПУ	2-ядерный ARM A9 @ 1 GHz	ARMA9 @ 1 GHz	2-ядерный Intel Core i5 @ 1,9GHz
RAM	512 MB DDR3	1 GB DDR3	8 GB DDR3
Хранение	32 MB последовательная флэш	4 GB eMMC	64 GB SSD (опционно SATA)

**MSRP - рекомендована виробником роздрібна ціна*

Доступні ресурси повинні використовуватися спільно з основними функціями маршрутизатора як мережевого агента, а також з тим, що користувач розробляє як програмне забезпечення для границі або туману.

Граничному пристрою необхідно управляти і забезпечувати безпеку розгорнутих рішень замовника на границі, так як у багатьох ситуаціях пристрій буде знаходитися дуже далеко від програміста або адміністратора. Границю необхідно враховувати в тому ж контексті, що і хмарне програмне забезпечення і сервіси. Маршрутизатор, що пропонує послуги граничних обчислень, надасть API або інтерфейс SDK для використання ресурсів на пристрої та розгортання програм.

Контрольні питання

1. Роль і завдання шлюза у сенсорній мережі.
2. Що таке граничний маршрутизатор? Чи може він бути одночасно і шлюзом?
3. Що таке лавинна маршрутизація та маршрутизація по найкоротшому шляху?
4. Чим відрізняється маршрутизація на основі потоку від маршрутизації на основі вектора відстані?
5. Порівняти маршрутизацію за станом каналу з ієрархічною маршрутизацією.

6. Чим відрізняється широкомовна маршрутизація від групової маршрутизації?
7. Які протоколи маршрутизації ви знаєте? Охарактеризуйте кожен.
8. Перерахуйте інформацію, яка міститься в типовій таблиці маршрутизації.
9. Як забезпечити відмовостійкість каналів і що таке керування по окремому каналу?
10. Що таке VLAN і як застосовується ця технологія при роботі з віддаленим обладнанням?
11. Що таке VPN тунель, для чого він потрібен? Як його застосовують для роботи з віддаленим обладнанням?
12. Як працює IPSec? Навести схему.
13. Як працює OpenVPN? Навести схему
14. Що таке GRE? Навести схему
15. Як працює L2TP? Навести схему
16. Як керують швидкістю трафіка і пріоритетом пакетів в мережі? Що таке QoS?
17. Які функції у брандмауера сенсорної мережі? Їх форми?
18. Для чого потрібен DMZ? Що таке переадресація портів?
19. Для чого потрібні метрика та аналітика на граничному маршрутизаторі?
20. Які параметри повинна збирати і оброблювати типова метрика і аналітика на граничному маршрутизаторі?
21. Що таке "обробка на границі" і які типові функції вона включає у себе?

9 IoT ПРОТОКОЛИ ОБМІНУ ДАНИМИ

9.1 Для чого потрібні протоколи IoT

Стандартні протоколи прикладного рівня передачі даних - це інструменти, які пов'язують і інкапсулюють необроблені дані від давача до клієнта і форматують дані для хмари. Сучасні IoT-рішення будуються на базі протоколів передачі даних (наприклад, *MQTT, AMQP, HTTP, CoAP* і інші) і забезпечують взаємодію кінцевих пристроїв з хмарними сервісами (наприклад, *Azure IoT Suite, Amazon Web Services IoT*). Одне з основних відмінностей між системою IoT і системою M2M полягає в тому, що M2M може зв'язуватися через WAN з виділеним сервером або системою без будь-якого інкапсульованого протоколу. За визначенням, IoT заснований на зв'язку між кінцевими точками і службами, причому Інтернет - є спільною мережевою основою.

На сьогоднішній день існує декілька протоколів прикладного рівня (*Application layer*), використовуваних при створенні IoT-сервісів:

CoAP,

DTLS,

Eddystone,

HTTP,

iBeacon,

MQTT,

MQTT-SN

PJON,

STOMP,

Websocket,

XMPP.

Незважаючи на їх різноманітність, на практиці розробники частіше застосовують протоколи MQTT, HTTP, Websocket і CoAP [3][4]. Крім того, їх підтримують основні провайдери хмарних сервісів в своїх рішеннях (Amazon, Microsoft, IBM, Google). Статистика використання протоколів підтверджується наступною діаграмою (рис. 9.1):

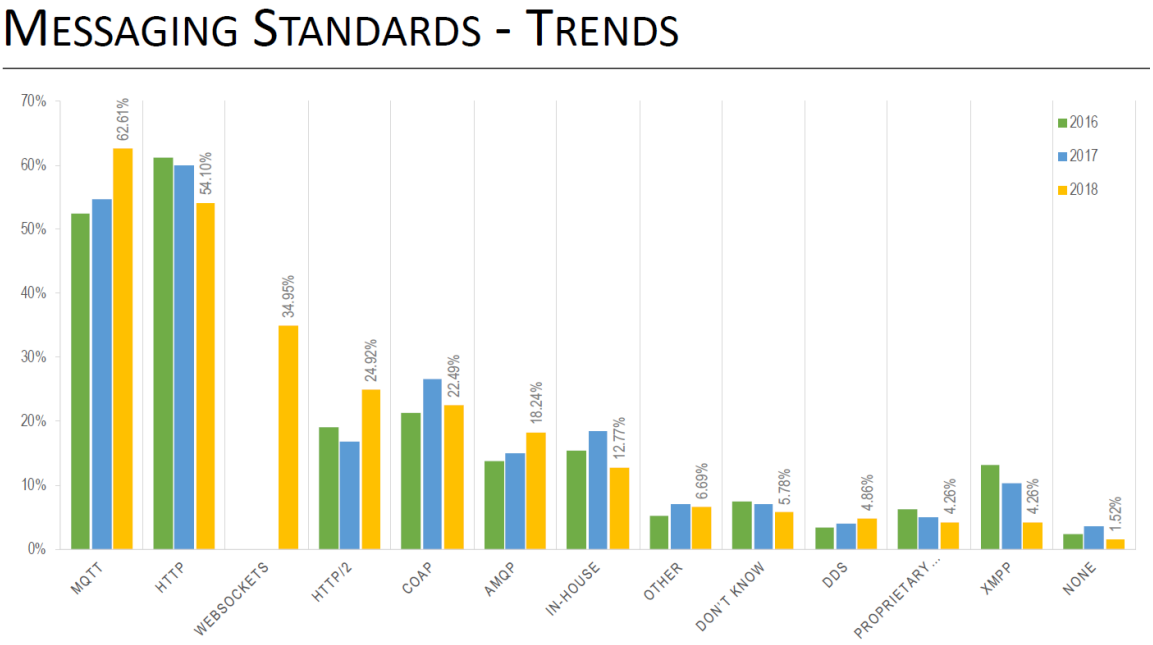


Рисунок 9.1 Дослідження IoT Developer Survey 2018, проведено Eclipse IoT Working Group (підрозділ Eclipse Foundation) спільно з AGILE IoT, IEEE і Open Mobile Alliance.

Природне запитання: для чого потрібні ще які-небудь протоколи за межами HTTP для передачі даних по глобальній мережі? Протокол HTTP і без того вже майже 30 років забезпечує значні можливості та має весь необхідний набір послуг для інтернет-зв'язку. Річ у тому, що протокол HTTP був розроблений і сконструйований для загального використання у моделях клієнт/сервер універсального типу. Пристрої ж IoT можуть бути дуже обмеженими у ресурсах, віддаленими і обмеженими по смузі пропускання (тобто, по швидкості передачі). Тому для управління безліччю пристроїв в різних мережевих топологіях, таких як mesh-мережі, ієрархічні та змішані мережі, необхідні більш ефективні, безпечні та масштабовані протоколи.

Протоколи, що будуть вивчатися у цьому розділі, є реалізацією **Message Orientated Middleware** - проміжного програмного забезпечення, орієнтованого на повідомлення (MOM). Основна ідея MOM полягає в тому, що зв'язок між двома пристроями відбувається з використанням **повідомлень**, а саме, **розподілених черг повідомлень**. MOM відправляє повідомлення від однієї програми в просторі користувача іншій. Одні пристрої виробляють дані для додавання в чергу, в той час як інші пристрої споживають дані, що знаходяться в черзі. Керує чергою окрема програма на сервері, що називається **брокер**. У цьому випадку виробники і споживачі повідомлень називаються відповідно

публікаторами та **підписниками** та мають відповідні зв'язки з брокером. Реалізація MOM з використанням черг використовується і для підвищення стійкості у процесі їх роботи. Дані можуть продовжувати зберігатися в чергах, навіть якщо відмовить сервер.

Альтернативою реалізації MOM є протокол *RESTful*. У моделі RESTful сервер володіє статком ресурсу, але стан не передається в повідомленні від клієнта на сервер. RESTful використовує HTTP-методи, такі як GET, PUT, POST і DELETE для розміщення запитів за універсальним ідентифікатором ресурсу (URI) (див. Рис. 9.2). У цій архітектурі не потрібен брокер або посередник. Оскільки вони засновані на стеку HTTP, вони користуються більшістю вже існуючих сервісів, таких як безпечний протокол HTTPS. Проекти RESTful типові для клієнт-серверних архітектур. Клієнти ініціюють доступ до ресурсів через синхронні шаблони запиту-відповіді.

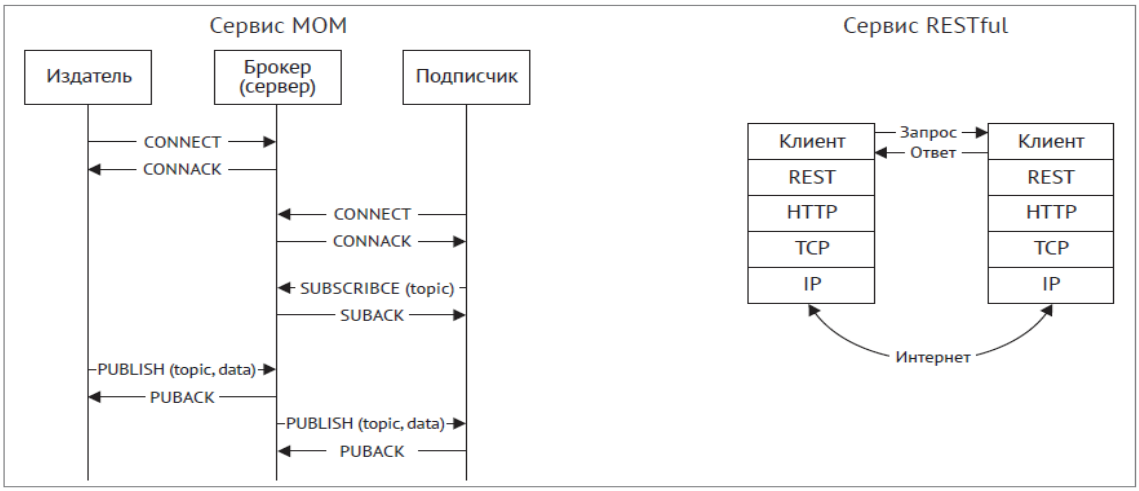


Рисунок 9.2 Порівняння MOM з RESTful

Малюнок нижче (рис. 9.3) ілюструє деякі масштабні взаємодії, необхідні для повної реалізації IoT з протоколами типу MOM чи RESTful.

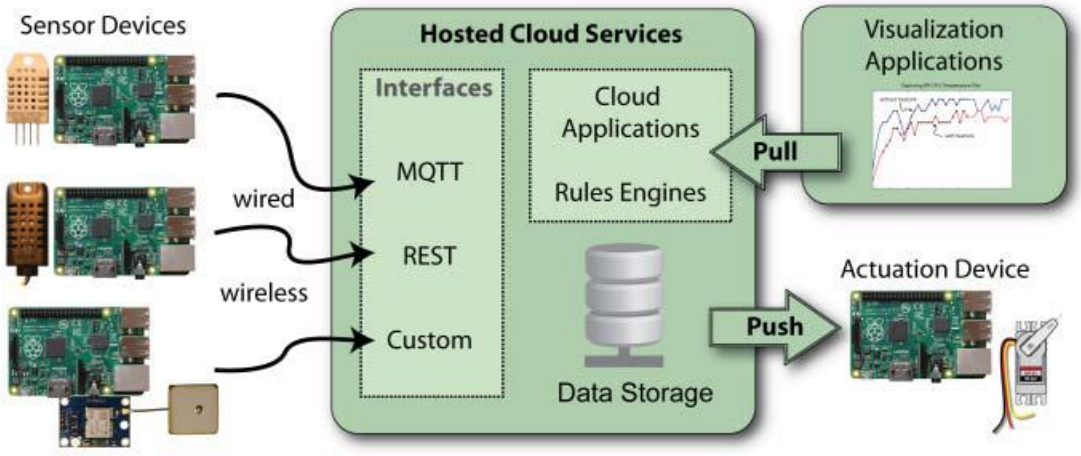


Рисунок 9.3 Типове рішення сучасної IIoT архітектури

9.2 Протокол MQTT

9.2.1 Історія створення та сьогодення

Протокол **Message Queue Telemetry Transport** (MQTT, телеметричний транспорт черги повідомлень) веде свою історію від технології IBM Websphere Message Queue, що була вперше впроваджена в 1993 р. для вирішення проблем в незалежних і неконкурентних розподілених системах для забезпечення захищеного зв'язку. Похідний протокол від Web Sphere Message Queue було створено Енді Стенфордом-Кларком і Арлен Ніппер в IBM в 1999 р. для вирішення конкретних проблем, пов'язаних з підключенням віддалених нафто- і газопроводів через супутниковий зв'язок. Саме цей протокол і став відомий як MQTT. Вимоги до створення цього транспортного протоколу, що базується на IP, були такі:

- він повинен бути простим в реалізації;
- він повинен забезпечувати задану якість обслуговування;
- він повинен бути дуже легким і ефективним з точки зору пропускної здатності;
- він повинен бути крос-платформенним;
- він повинен постійно відстежувати сеанс зв'язку;
- він повинен вирішувати проблеми безпеки.

На сайті протоколу MQTT приведено характеристику протоколу: "*MQTT - це MQ Telemetry Transport, який являє собою простий і легкий протокол обміну повідомленнями, призначений для обмежених пристроїв і мереж з низькою пропускною здатністю, з високою затримкою або ненадійністю. Розроблено на принципах мінімізації пропускної здатності мережі та вимог до ресурсів пристроїв, намагаючись в той же час забезпечити надійність і деяку ступінь впевненості в доставці. Ці принципи також роблять цей протокол ідеальним для появи світу підключених пристроїв типу «машина-машина» (M2M) або «інтернет речей», а також для мобільних застосувань, де вкрай важливі пропускна здатність і заряд батареї*".

MQTT був внутрішнім і пропрієтарним протоколом для IBM протягом багатьох років, поки не був випущений у версії 3.1 в 2010 р. в якості безкоштовного продукту. У 2013 р. MQTT був стандартизований і прийнятий в консорціум OASIS (глобальний некомерційний консорціум, який займається розробкою, конвергенцією і ухваленням відкритих стандартів в рамках міжнародного інформаційного співтовариства). У 2014 р. OASIS опублікував його публічно як версію MQTT 3.1.1. На квітень 2019 р. опублікована вже версія 5.0 протоколу. Нова версія включає до себе:

- *покращення звітів про помилки* - зокрема, код відповідей було додано до квитанцій на публікації (*PUBACK/PUBREC*). MQTT виник для використання з давачами вздовж нафтопроводу - якщо публікації давача не можуть бути передані, то давач просто не буде виконувати ніяких дій. Однак у випадку використання MQTT для IIoT, програмне застосування, наприклад на смартфоні, може захотіти попередити користувача, якщо дані не передаються успішно. Код відповіді тепер в змозі надати інформацію користувачу про причини не успішної передачі;
- *спільні підписки* - якщо швидкість передачі повідомлень по передплаті висока, можна використовувати спільні підписки для завантаження повідомлень декільком клієнтам відразу;
- *властивості повідомлення* – добавлені додаткові метадані в заголовку повідомлення. Вони використовуються для реалізації інших функцій, а також дозволяють визначити користувачу власні (нестандартизовані) властивості, наприклад, щоб допомогти в шифруванні повідомлень, повідомляючи одержувачу, яку клавішу використовувати для розшифрування вмісту повідомлення;
- *термін дії повідомлення* - можливість скасувати повідомлення, якщо вони не можуть бути доставлені протягом визначеного користувачем періоду часу;
- *закінчення сеансу* - якщо клієнт не підключається протягом визначеного користувачем періоду часу, деякі стани (наприклад,

підписки і буферізовані повідомлення) можуть бути відкинуті без необхідності очищення.

У даний час MQTT являється також і стандартом ISO (ISO/IECPRF 20922).

9.2.2 Логіка функціонування

Публікація/підписка, також відома як *pub/sub*, є способом відокремити клієнта, що передає повідомлення від іншого клієнта, який отримує повідомлення. На відміну від традиційної моделі клієнт-сервер, клієнти не повинні бути обізнані про будь-які фізичні ідентифікатори свого візаві, на зразок IP-адреси або порту (рис.9.4).

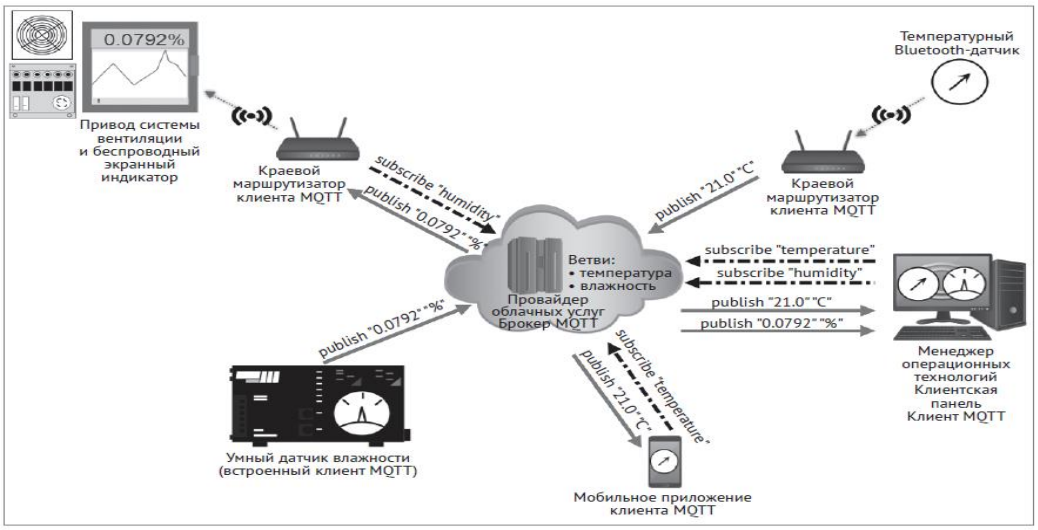


Рисунок 9.4 Модель і топологія публікація/підписка MQTT.

Клієнт, що передає повідомлення, називається **публікатором (видавцем)**; клієнт, який отримує повідомлення, називається **підписником (передплатником)**. У центрі між ними знаходиться брокер MQTT, який несе відповідальність за з'єднання клієнтів і фільтрацію даних. Такі фільтри забезпечують:

- фільтрацію по темам - за задумом, клієнти підписуються на **теми (топіки)** і певні гілки в них і не отримують даних більше, ніж хочуть. Кожне опубліковане повідомлення повинно містити тему, і брокер несе відповідальність за повторну передачу цього повідомлення передплатникам або ігнорування його;
- фільтрацію по вмісту - брокери мають можливість перевіряти і фільтрувати опубліковані дані. Таким чином, будь-які дані, які не

зашифровані, можуть управлятися брокером до того, як зберегти їх або передати іншим клієнтам;

- фільтрацію за типом - клієнт, що прослуховує потік даних, на які він підписаний, може також застосовувати свої власні фільтри. Вхідні дані можуть аналізуватися, і в залежності від цього потік даних обробляється далі або ігнорується.

У MQTT може бути багато публікаторів і підписників.

Одне із застережень моделі обчислень публікатор/підписник полягає в тому, що як публікатор, так і підписник повинні знати тему гілки і формат даних перед початком передачі.

MQTT успішно відокремлює публікаторів від підписників. Оскільки брокер є керівним органом між публікаторами і підписниками, немає необхідності безпосередньо ідентифікувати публікатора і підписника на основі фізичних даних (таких як IP-адреса). Це дуже корисно при розгортанні IoT, оскільки фізичний ідентифікатор може бути невідомим або загальним. MQTT і інші моделі pub/sub також є часонезалежними. Це означає, що повідомлення, яке опубліковане одним клієнтом, підписник може прочитати і відповісти на нього в будь-який час. Абонент (підписник) може перебувати в місці з дуже низьким енергоспоживанням / обмеженою пропускною здатністю (наприклад, LoRaWAN) і відповісти на повідомлення через кілька хвилин або годин. Через відсутність фізичних і часових відносин моделі pub/sub дуже добре підходять для підвищення продуктивності.

Керовані хмарно брокери MQTT зазвичай можуть обробляти мільйони повідомлень на годину і підтримувати десятки тисяч публікаторів.

MQTT не залежить від формату даних. Корисне навантаження може містити будь-який тип даних, тому і публікатори, і підписники повинні розуміти і погоджувати одне з одним формат даних. Можна надсилати текстові повідомлення, дані зображення, звукові дані, зашифровані дані, двійкові дані, об'єкти JSON або практично будь-яку іншу структуру в корисному навантаженні пакета. Однак текстові і двійкові дані JSON є найбільш поширеними типами даних корисного навантаження.

Максимально допустимий розмір пакета в MQTT становить 256 МБ, що дозволяє отримати надзвичайно велике корисне навантаження. Однак **слід звернути увагу, що максимальний розмір корисного навантаження даних залежить ще від хмари і брокера**. Наприклад, IBM Watson дозволяє обробляти дані розміром лише до 128 КБ, а Google підтримує 256 КБ. З іншого боку, опубліковане повідомлення може включати корисне навантаження і нульової довжини. Поле корисного навантаження не є обов'язковим. Архітектору IIoT доцільно звіряти відповідність розмірів корисного навантаження з можливостями потрібного хмарного провайдера. Недотримання цієї вимоги призведе до помилок і відмови у доступі до хмарного брокера.

9.2.3 Особливості протоколу

Сама назва протоколу MQTT є неточною. У протоколі немає черги повідомлень. Хоча можна відправляти повідомлення і в чергу, це необов'язково і часто не робиться. MQTT заснований на TCP (рис. 9.5) і тому є деяка гарантія того, що пакет передається надійно.

MQTT також є асиметричним протоколом, тоді як HTTP - це симетричний протокол. Скажімо, вузол A повинен зв'язуватися з вузлом B. Асиметричний протокол між A і B вимагає, щоб протокол використовувала тільки одна сторона (A), і тим самим вся інформація, необхідна для повторного складання пакетів, повинна міститися в заголовку фрагментації, надісланому від A. В асиметричних

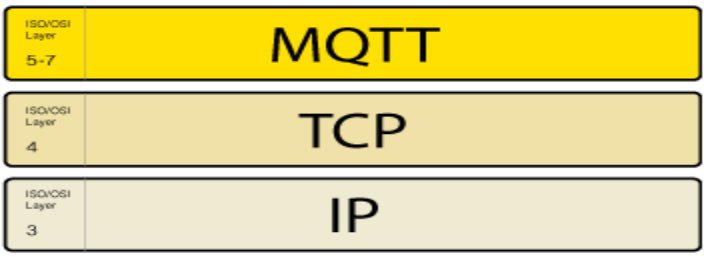


Рисунок 9.5 Протокол MQTT працює на прикладному рівні поверх TCP/IP

системах є один ведучий і один ведений (*FTP* - класичний приклад). У симетричному протоколі ведучий/ведений може встановлюватися по черзі як на A, так і на B. A або B можуть приймати на себе роль ведучого або веденого (тут прикладом може служити *telnet*). У MQTT ролі різні, що якраз і має сенс в топології давачів/хмар.

MQTT може зберігати повідомлення у брокера необмежено довго. Цей режим роботи керується відповідним прапорцем при нормальній передачі повідомлення. Збережене у брокера повідомлення відправляється будь-якому клієнту, який підписаний на цей топік. Повідомлення негайно відправляється до цього нового клієнта. Це дозволяє новому клієнтові отримати статус або сигнал з топіка, на який він недавно підписався, без очікування. Клієнт, що підписується на топік, може очікувати години або навіть дні, перш ніж публікатор опублікує нові дані.

TCP/IP-порт 1883 зарезервований для протоколу MQTT, а порт 8883 зарезервований для нього ж з використанням безпечного протоколу SSL. Додатково до SSL, MQTT підтримує в транзакції логін та пароль.

9.2.3.1 Запобігання наслідків від розриву з'єднання

MQTT визначає додатковий об'єкт під назвою **Last Will and Testament, LWT** (Остання воля і заповіт). LWT - це повідомлення, яке вказує клієнт під час фази підключення. LWT містить «*Last Will*» топік, QoS і фактичне повідомлення. Якщо клієнт неправильно відключається від брокерського з'єднання (наприклад, по тайм-ауту *keep-alive*, помилка введення/виведення або закриває сеанс без відключення), тоді брокер зобов'язаний транслювати повідомлення LWT всім іншим, підписаним на цю тему клієнтам.

Незважаючи на те, що MQTT заснований на надійному протоколі TCP, з'єднання все ще можуть обриватися, особливо в разі бездротових давачів. Пристрій може втратити живлення, втратити сигнал або може бути просто польова поломка, і сеанс перейде в напіввідкритий стан. Тоді сервер буде вважати, що з'єднання як і раніше надійно і очікувати дані. Щоб вийти з цього напіввідкритого стану, MQTT використовує метод *keep-alive*. Використовуючи цей метод, як брокер MQTT, так і клієнт мають гарантію того, що з'єднання залишається працездатним, навіть якщо протягом деякого часу не було передачі. Клієнт відправляє пакет *PINGREQ* брокеру, який, в свою чергу, підтверджує повідомлення за допомогою *PINGRESP*. Таймер встановлений на стороні клієнта і брокера. Якщо повідомлення не було передано жодним з них протягом заданого

проміжку часу, повинен бути відправлений пакет *keep-alive*. Як *PINGREQ*, так і повідомлення *keep-alive*, скинуть таймер *keep-alive*.

У разі, якщо *keep-alive* ми отримали і час таймера закінчується, брокер закриє з'єднання і відправить LWT-пакет всім клієнтам. Клієнт може в якийсь момент пізніше спробувати знову підключитися. У цьому випадку брокер закриває напіввідкрите з'єднання і відкриває нове з'єднання з клієнтом. Максимальний час очікування - 18 годин, 12 хвилин і 15 с. Установка внутрішнього стану *keep-alive* на 0 призведе до відключення функції *keep-alive*. Таймер керується клієнтом і може динамічно змінюватися для адекватного відображення режимів сну або зміни рівня сигналу.

9.2.3.2 Зниження накладних витрат при перепідключенні

У той час як, параметр *keep-alive* допомагає боротися з порушеннями каналу зв'язку клієнт-брокер, повторне встановлення всіх підписок клієнта і параметрів QoS може привести до непотрібних накладних витрат на знову підключеному з'єднанні. Щоб зменшити ці додаткові витрати, MQTT дозволяє підтримувати так звані **постійні з'єднання**. Постійне з'єднання зберігає на стороні брокера наступне:

- всі підписки клієнта;
- всі повідомлення QoS, які не були підтверджені клієнтом;
- всі нові повідомлення QoS, пропущені клієнтом.

Параметр *client_id* посилається на цю інформацію для унікальної ідентифікації клієнтів. Клієнт може запитувати постійне з'єднання, проте брокер може відхилити запит і примусово перезапустити "чистий" сеанс з початку. При з'єднанні брокером використовується прапорець *cleanSession* для дозволу або заборони постійних з'єднань. Клієнт може визначити, чи ввімкнено збереження постійного з'єднання за допомогою повідомлення *CONNACK*. *Постійні сеанси повинні використовуватися для клієнтів, які повинні отримувати всі повідомлення, навіть коли немає зв'язку. Вони не повинні використовуватися в ситуаціях, коли клієнт тільки публікує (записує) дані в топіки.*

9.2.3.3 Використання QoS

У протоколі MQTT є три рівня якості обслуговування:

Рисунок 9.6 QoS-0. Незавірена передача

QoS-0 (незавірена передача) - це мінімальний рівень QoS (рис. 9.6). Це аналогічно моделі «вистрілив і забув», докладно описаної в інших бездротових протоколах. Це найефективніший процес доставки повідомлень, без підтвердження одержувачем і без повторної передачі повідомлення відправником;

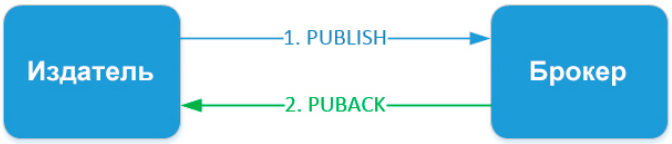


Рисунок 9.7 QoS-1. Гарантована передача

QoS-1 (гарантована передача) - цей режим гарантує доставку повідомлення одержувачу хоча б один раз (рис. 9.7). Повідомлення може бути доставлено кілька разів, і одержувач при отриманні повідомлення повинен відправити назад відправнику підтвердження з відповіддю *PUBACK*;

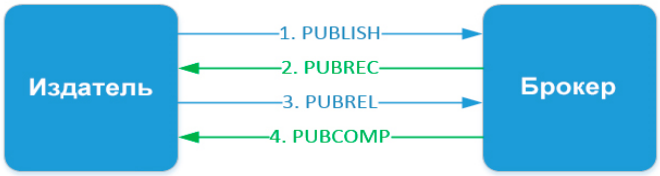


Рисунок 9.8 Гарантований сервіс

QoS-2 (гарантований сервіс для програмових застосувань, рис. 9.8) - це найвищий рівень QoS, який переконується в доставці і інформує відправника і одержувача, що повідомлення було передано правильно. Цей режим генерує більше трафіку через багатокроковий **хендшейкінг** між відправником і одержувачем. Якщо одержувач отримує повідомлення з рівнем обслуговування QoS-2, він відповість відправнику повідомленням *PUBREC*. Це підтверджуюче повідомлення, і відправник відповість одержувачу повідомленням *PUBREL*. *PUBREL* дозволяє одержувачеві безпечно відкинути будь-які повторні передачі свого підтверджуючого повідомлення. Потім *PUBREL* підтверджується одержувачем за допомогою *PUBCOMP*. Поки повідомлення *PUBCOMP* не буде відправлено, одержувач буде кешувати оригінальне повідомлення для забезпечення безпеки.

QoS в MQTT визначається і контролюється відправником, і у кожного відправника може бути своя політика. Типові випадки використання:

- QoS-0 слід використовувати, коли повідомлення не потрібно зберігати в черзі. QoS-0 найкраще підходить для бездротового підключення або коли система сильно обмежена у пропускній здатності;
- QoS-1 слід використовувати за замовчуванням. QoS-1 працює набагато швидше, ніж QoS-2, і значно знижує вартість передачі;
- QoS-2 - для критично важливих застосувань. Крім того, для випадків, коли повторна передача дубльованого повідомлення може призвести до помилок.

Таблиця 9.1 Вплив вибору QoS на тривалість функціонування батарейного MQTT клієнта

3G

QoS	% Battery / Hour	Messages / Hour	% Battery / Message *
QoS 0	15.12	614400	0.000025
QoS 1	16.87	23938	0.000705
QoS 2	17.66	15489	0.001140

Wifi

QoS	% Battery / Hour	Messages / Hour	% Battery / Message *
QoS 0	2.02	368640	0.000005
QoS 1	0.93	26144	0.000036
QoS 2	0.89	13704	0.000065

На таблицях 9.1 показано, як в залежності від вибору QoS буде залежати тривалість роботи батарейного MQTT клієнта. Таблиці зроблено на практичному прикладі з використанням смартфону Android та 3G і Wi-Fi:

9.2.4 Структура пакета MQTT

Пакет MQTT знаходиться поверх рівня TCP мережевого стека в моделі OSI. Пакет складається з фіксованого заголовка довжиною від 2 до 5 байт (один байт відводиться під селектор типу пакета, і від 1 до 4 байт займає значення довжини пакета у байтах) який завжди повинен бути присутнім, заголовка зі змінним розміром (необов'язково) і закінчується корисним навантаженням (теж необов'язково) (малюнок нижче).

На пакетному рівні протокол простий і відрізняється порівняно невеликим «перевантаженням» службовою інформацією. Це важлива властивість, що

обумовлює незаперечні і доведені переваги MQTT. Загальну структуру всіх пакетів можна відобразити на наступному рисунку 9.9:

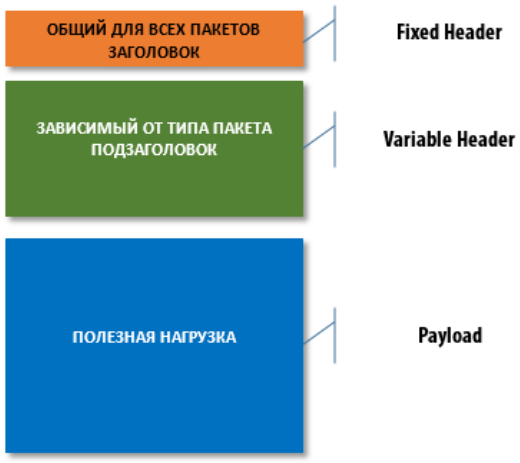


Рисунок 9.9 Загальна структура пакета MQTT

9.2.4.1 Фіксований заголовок

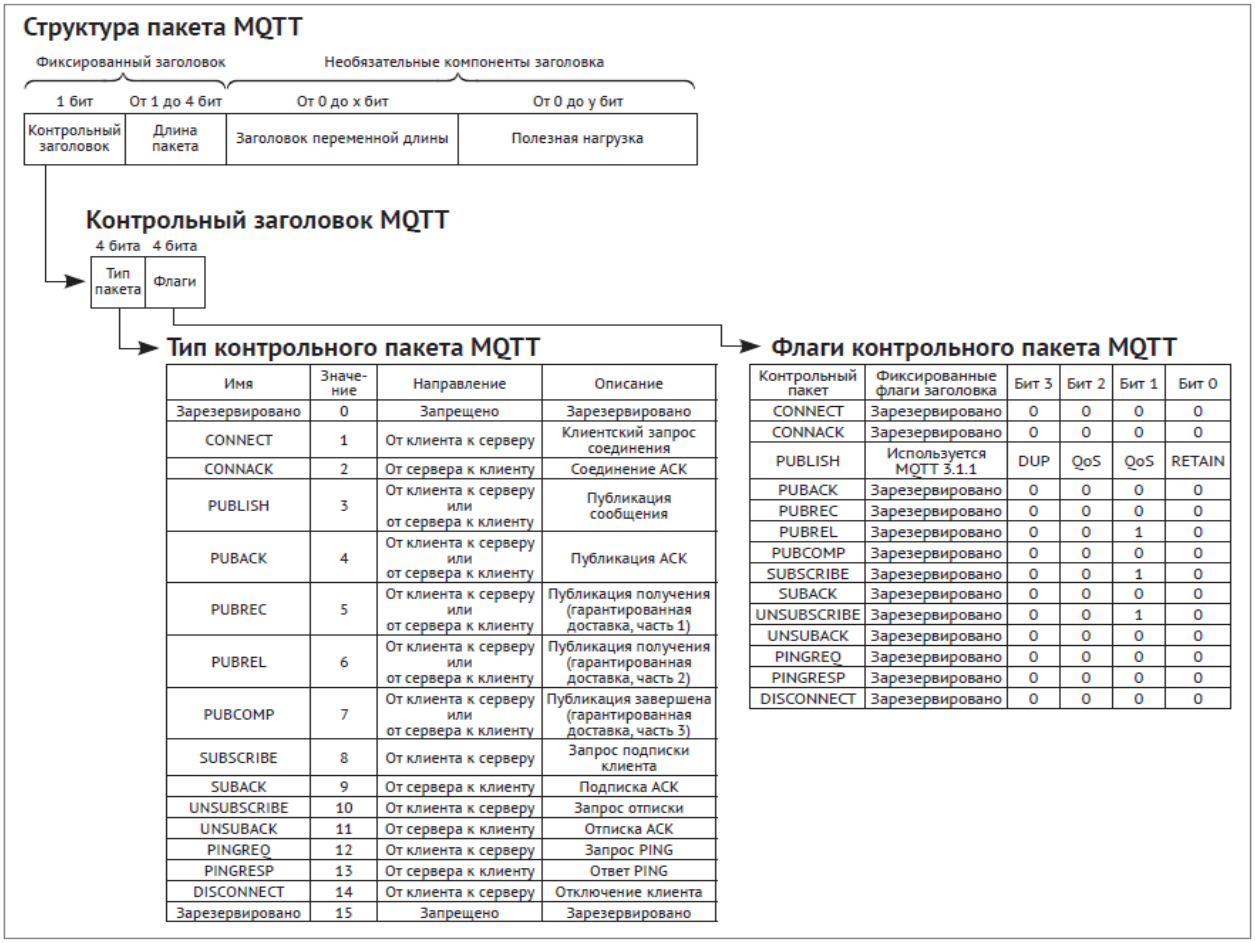
Загальний для всіх пакетів заголовок (Fixed Header), незважаючи на слово «fixed» в стандартній назві, має змінну довжину від двох до п'яти байтів (рис. 9.10).

Найперший байт - фактично селектор типу пакета (всього поточна версія стандарту описує 14 типів пакетів) і, за сумісництвом, контейнер для керуючих прапорців всього одного ключового типу пакета - *Publish*. Цей байт в старшому ніблі (тетраді) містить код типу пакета, в молодшому ніблі - прапорці для пакетів *Publish* і, іноді встановлені біти для деяких типів пакетів.

Другий і наступні (до чотирьох сумарно) байти містять "хитро" закодовану довжину (в байтах) всіх інших елементів пакету. Механізм кодування довжини заснований на наступному принципі - якщо старший біт байта встановлений в одиницю, 7 бітів цього байта записуються в результуюче 32-бітове (точніше, 28-бітове) слово в 7-бітову позицію, що відповідає номеру байта в послідовності. Отже, повна довжина максимального MQTT-пакета, що відповідає стандарту, буде дорівнює 5 (байтів Fixed Header) + значення 28-бітного слова з усіма одиничними бітами, або 2^{28}-1, що в сумі дає приблизно 255 мегабайтів сумарної довжини пакета.

Я не випадково звертаю увагу на цю ніби як незначну деталь стандарту. Тому що є реальність, з її реальними обчислювачами. І в цій реальності з одного

боку є MQTT-клієнт, який може виконуватися малим вбудовуваним обчислювачем (мова йде не про мікроконтролер, а про малі одноплатні машини масштабу Raspberry Pi і менше) і, з іншого боку, є MQTT-сервер, що обслуговує масу клієнтів. Через ***pub-симетричність*** *протоколу інжектовані в загальну систему пакети великого розміру можуть одночасно створити жахливе навантаження на сервер і фактично знищити тонкі MQTT-клієнти. Тому, хоч стандарт і не встановлює ніяких додаткових обмежень на розмір пакета, при реалізації що клієнта, що сервера розумно обмежитися максимально можливими двома байтами кодування довжини пакета, іншими словами - в серверній реалізації декодера пакетів MQTT строго перевіряти старший біт третього байта загального для всіх пакетів заголовка (якщо цей байт, звичайно, є) будь-якого MQTT-пакета, і якщо в ньому з'являється одиниця, негайно закривати мережеве з'єднання з клієнтом, від якого надійшов* ***pub-пакет*** *такого розміру. І, само собою, «бити на сполох». Так, це порушення стандарту, але воно обмежує можливу максимальну довжину пакетів приблизно 16 кілобайтами, що для реальних M2M і IIoT застосувань більш ніж достатньо, навіть для неспішних оновлень великого обсягу firmware (передачею фрагментів), і рятує систему від потенційних небезпек (не обов'язково умисних).*



Имя	Значение	Направление	Описание
Зарезервировано	0	Запрещено	Зарезервировано
CONNECT	1	От клиента к серверу	Клиентский запрос соединения
CONNACK	2	От сервера к клиенту	Соединение ACK
PUBLISH	3	От клиента к серверу или от сервера к клиенту	Публикация сообщения
PUBACK	4	От клиента к серверу или от сервера к клиенту	Публикация ACK
PUBREC	5	От клиента к серверу или от сервера к клиенту	Публикация получения (гарантированная доставка, часть 1)
PUBREL	6	От клиента к серверу или от сервера к клиенту	Публикация получения (гарантированная доставка, часть 2)
PUBCOMP	7	От клиента к серверу или от сервера к клиенту	Публикация завершена (гарантированная доставка, часть 3)
SUBSCRIBE	8	От клиента к серверу	Запрос подписки клиента
SUBACK	9	От сервера к клиенту	Подписка ACK
UNSUBSCRIBE	10	От клиента к серверу	Запрос отписки
UNSUBACK	11	От сервера к клиенту	Отписка ACK
PINGREQ	12	От клиента к серверу	Запрос PING
PINGRESP	13	От сервера к клиенту	Ответ PING
DISCONNECT	14	От клиента к серверу	Отключение клиента
Зарезервировано	15	Запрещено	Зарезервировано

Контрольный пакет	Фиксированные флаги заголовка	Бит 3	Бит 2	Бит 1	Бит 0
CONNECT	Зарезервировано	0	0	0	0
CONNACK	Зарезервировано	0	0	0	0
PUBLISH	Используется MQTT 3.1.1	DUP	QoS	QoS	RETAIN
PUBACK	Зарезервировано	0	0	0	0
PUBREC	Зарезервировано	0	0	0	0
PUBREL	Зарезервировано	0	0	1	0
PUBCOMP	Зарезервировано	0	0	0	0
SUBSCRIBE	Зарезервировано	0	0	1	0
SUBACK	Зарезервировано	0	0	0	0
UNSUBSCRIBE	Зарезервировано	0	0	1	0
UNSUBACK	Зарезервировано	0	0	0	0
PINGREQ	Зарезервировано	0	0	0	0
PINGRESP	Зарезервировано	0	0	0	0
DISCONNECT	Зарезервировано	0	0	0	0

Рисунок 9.10 Структура пакетів MQTT

DUP - прапор дубліката. Встановлюється, коли клієнт або MQTT брокер здійснює повторну відправку пакета (використовується в типах *PUBLISH*, *SUBSCRIBE*, *UNSUBSCRIBE*, *PUBREL*). При встановленому прапорі змінний заголовок повинен містити *Message ID* (ідентифікатор повідомлення).

QoS - якість обслуговування (0,1,2).

RETAIN - при публікації даних з встановленим прапором retain, брокер збереже його. При наступній підписці на цей топік брокер негайно відправить повідомлення з цим прапором. Використовується тільки в повідомленнях з типом *PUBLISH*.

9.2.4.2 Змінний заголовок

Змінний заголовок міститься лише в деяких пакетах. У ньому розміщуються наступні дані (рис. 9.10):

- **Packet identifier** - ідентифікатор пакета, присутній у всіх типах повідомлень, крім: *CONNECT*, *CONNACK*, *PUBLISH* (з QoS <1), *PINGREQ*, *PINGRESP*, *DISCONNECT*;
- **Protocol name** - назва протоколу (тільки в повідомленнях типу *CONNECT*);

- **Protocol version** - версія протоколу (тільки в повідомленнях типу *CONNECT*);
- **Connect flags** - прапори вказують на поведінку клієнта при підключенні (див. рис. 9.11)

Bit	7	6	5	4	3	2	1	0
Byte 8	User name	Password	Will Retain	Will QoS		Will Flag	Clean Session	Reserved

Рисунок 9.11 Змінний заголовок MQTT повідомлення

Значення окремих полів змінного заголовка:

- **User name** - при наявності цього прапорця в «навантаженнях» (даних) має бути вказано ім'я користувача (використовується для аутентифікації клієнта);
- **Password** - при наявності цього прапорця в «навантаженнях» (даних) повинен бути вказаний пароль (використовується для аутентифікації клієнта);
- **Will Retain** - при установці в 1, брокер зберігає у себе Will Message;
- **Will QoS**- якість обслуговування для Will Message, при встановленому прапорці *Will Flag*, встановлення в 1 прапорів *Will QoS* і *Will retain* є обов'язковим;
- **Will Flag** - при встановленому прапорі, після того, як клієнт відключиться від брокера без відправлення команди DISCONNECT (у випадках непередбачуваного обриву зв'язку і т.і.), брокер сповістить про це всіх підключених до нього клієнтів через так званий *Will Message*;
- **Clean Session** - очистити сесію. При встановленому «0» брокер збереже сесію, всі підписки клієнта, а також передасть йому всі повідомлення з QoS1 і QoS2, які були отримані брокером під час відключення клієнта, при його наступному підключенні. Відповідно при встановленій «1», під час наступного з'єднання клієнту буде необхідно заново підписуватися на топіки;

- **Session Present** - застосовується в повідомленні з типом CONNACK. Якщо брокер приймає підключення з *Clean Session* = 1 він повинен встановити «0» біт *Session Present* (SP). Якщо брокер приймає

підключення з *Clean Session* = 0, то значення біта SP залежить від того, чи зберігав брокер раніше сесію з цим клієнтом (якщо так, то в SP виставляється 1 і навпаки). Тобто цей параметр дозволяє клієнту визначити чи була збережена брокером попередня сесія;

- **Connect Return code** - якщо брокер з якихось причин не може прийняти правильно сформований CONNECT пакет від клієнта, то в другому байті CONNACK пакета він повинен встановити відповідне значення з числа зазначених у таблиці 9.2:

Таблиця 9.2 Перелік кодів повернення

Значення	Код повернення	Опис
0	0x00 Connection Accepted	Підключення прийнято
1	0x01 Connection Refused, unacceptable protocol version	Брокер не підтримує версію протокола, що використовується клієнтом
2	0x02 Connection Refused, identifier rejected	Client ID клієнта, що підключається, відсутній у списку дозволених
3	0x03 Connection	З'єднання встановлено, але MQTT сервіс недоступний

Значення	Код повернення	Опис
	Refused, Server unavailable	
4	0x04 Connection Refused, bad user name or password	Не вірний логін, чи пароль
5	0x05 Connection Refused, not authorized	Підключення заборонено
6…255		зарезервовано

- **Topic** – назва топіка.

9.2.4.3 Дані "корисного навантаження"

Зміст і формат даних, що передаються у полі корисного навантаження в MQTT повідомленнях, визначено в http://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/annotated.html. Розмір даних корисного навантаження може бути обчислений шляхом вирахування з *Довжини пакету* довжини змінного та фіксованого заголовків.

9.2.5 Різновиди комунікаційних повідомлень

Повний опис формату комунікаційних повідомлень протоколу MQTT знаходиться на сайті. Тут ми розглянемо лише декілька типів формату повідомлень.

З'єднання з використанням MQTT починається з того, що клієнт відправляє повідомлення *CONNECT* брокеру. Тільки клієнт може ініціювати сеанс, і жоден клієнт не може напряму зв'язатися з іншим клієнтом. У відповідь на повідомлення *CONNECT* брокер завжди буде відсилати *CONNACK* і код статусу.Після встановлення з'єднання залишається у робочому стані до роз'єднання.

Тип *CONNECT* (клієнт до сервера): типове повідомлення *CONNECT* буде містити наступні дані (для ініціації сесії потрібно тільки поле *clientID*!), показані в таблиці 9.3 нижче.

Таблиця 9.3 Поля в повідомленні *CONNECT*

Поле	Необхідність	Опис
clientID	Обов'язкове	Ідентифікує клієнта на сервері. Кожен клієнт повинен мати унікальний ідентифікатор клієнта. Він може становити від 1 до 23 байтів UTF-8
cleanSession	Необов'язкове	0: сервер повинен відновити зв'язок з клієнтом. Клієнт і сервер повинні зберігати стан сеансу після відключення. 1: клієнт і сервер повинні скасувати попередній сеанс і почати новий
username	Необов'язкове	Логін, який використовується сервером для аутентифікації
password	Необов'язкове	Двійковий пароль довжиною від 0 до 65536 байтів з префіксом 2 байта
lastWillTopic	Необов'язкове	Ім'я топіка для публікації повідомлення ***Last Will***
lastWillQos	Необов'язкове	2 біта із зазначенням рівня QoS при публікації повідомлення ***Last Will***
lastWillMessage	Необов'язкове	Визначає зміст повідомлення ***Last Will***

Поле	Необхідність	Опис
lastWillRetain	Необов'язкове	Вказує, чи зберігається повідомлення ***Last Will*** після публікації (0/1)
keepAlive	Необов'язкове	Інтервал часу в секундах. Клієнт відповідає за відправку повідомлення або пакета *PINGREQ* до закінчення таймера *keepAlive*. Сервер відключиться від клієнта через 1,5 значення часу зазначеного інтервалу в режимі *keep-alive*. Значення 0 відключить механізм *keepAlive*

Коди повернення *CONNECT* (від сервера до клієнта): брокер буде відповідати на повідомлення *CONNECT* повідомленням *PUBACK* з кодом відповіді, що наведена в Таблиця 9.2. Архітектор повинен розуміти, що не всі вимоги з'єднання можуть бути схвалені брокером. Тому слід перевіряти коди відповіді перед тим, як продовжувати.

Тип *PUBLISH* (клієнт до сервера): публікація повідомлення в топіку. Кожне повідомлення містить поля (див. таблицю 9.4).

Таблиця 9.4 Формат *PUBLISH* (клієнт до сервера)

Поле	Необхідність	Опис
packetID	Обов'язкове	Унікально ідентифікує пакет в змінному заголовку. У зоні відповідальності клієнтської бібліотеки. Завжди встановлений на нуль (0) для QoS-0
topicName	Обов'язкове	Назва топіка для публікації (наприклад, Київ/Бажана 3/квартира 199/температура)
qos	Обов'язкове	Значення QoS повідомлення рівня 0, 1 або 2
retainFlag	Обов'язкове	Вказує, чи зберігається повідомлення після публікації (0/1)
payLoad	Необов'язкове	Корисне навантаження повідомлення в будь-якому форматі

Поле	Необхідність	Опис
dupFlag	Обов'язкове	Повідомлення є дублікатом і послано вдруге

Тип *SUBSCRIBE* (від клієнта до сервера): корисне навантаження пакета передплати включає в себе щонайменше одну пару кодованих UTF-8 *topicID* і рівні *QoS*. У цьому повідомленні у полі корисного навантаження може бути зазначено кілька *topicID*, щоб позбавити клієнта від кількох передач. Поля повідомлення наведені в таблиці 9.5 нижче.

Таблиця 9.5 Формат *SUBSCRIBE* (клієнт до сервера)

Поле	Необхідність	Опис
packetID	Обов'язкове	Унікально ідентифікує пакет в змінному заголовку. Відповідальність клієнтської бібліотеки
topic1	Обов'язкове	Назва першого топіка, на який виконується підписка
qos1	Обов'язкове	Значення QoS першого топіка (0/1/2)
……	Необов'язкові	……
topicN	Необов'язкове	Назва *N*-ого топіка, на який виконується підписка
qosN	Необов'язкове	Значення QoS *N*-ого топіка (0/1/2)

Символи шаблону можуть використовуватися для підписки на кілька тем в одному повідомленні. Для цих прикладів темою буде повний шлях *"{місто} / {вулиця} / {квартира} / {температура, вологість}"*.

+ Підстановлювальний шаблон одного рівня - замінює один рівень в імені рядка топіка. Наприклад, *Київ / Бажана3 / + / температура* замінить рівень дому на Бажана 3 в Києві температурами по усім квартирам;

*** Багаторівневий шаблон** - замінює кілька рівнів, а не один. Він завжди є останнім символом в назві теми. Наприклад, *Київ / Бажана3 / квартира 199 / **

буде відповідати вологості і температурі за адресою м. Київ, пр. Бажана 3, кв. 199.

$ Спеціальні теми - це спеціальний статистичний режим для брокерів MQTT. Клієнти не можуть публікувати в темах $. На даний момент офіційного стандарту для використання немає. Одна з моделей використовує $ SYS наступним чином: *$ SYS / broker / clients / connected*.

9.2.6 Приклад використання протоколу MQTT при роботі з Arduino

Детальний опис клієнта MQTT, що використовується для контролерів Arduino, наведено на сайті Arduino Client for MQTT. Звідти можна переписати собі потрібну бібліотеку під C++ *PubSubClient.cpp* із файлом заголовка *PubSubClient.h*, та детальний опис API-інтерфейса для скетча Arduino. Ця бібліотека забезпечує клієнта для простої публікації / підписки повідомлень на сервер, який підтримує MQTT. Обмеження бібліотеки:

- використовується версія MQTT 3.1.1;
- клієнт може публікувати лише повідомлення QoS 0, а підписатися на QoS 0 або QoS 1;
- максимальний розмір повідомлення за замовчуванням (включаючи заголовок) становить 128 байт. Цю величину можна змінити через макрос *MQTT_MAX_PACKET_SIZE* у файлі заголовку *PubSubClient.h*. Більш довгі повідомлення можна також надсилати методом *publish_P();*
- інтервал зберігання повідомлення за замовчуванням встановлений на 15 секунд. Це можна змінити через макрос *MQTT_KEEPALIVE* у файлі *PubSubClient.h*;
- клієнт за замовчуванням використовує версію MQTT 3.1.1. Версію можна змінити на використання MQTT 3.1, змінивши значення макросу *MQTT_VERSION* у *PubSubClient.h*.

Приклад використання протоколу MQTT з цієї бібліотеки при підключенні цифрового давача температури до хмарного середовища наведено у статті

Создание датчика температуры для облачной среды с помощью Arduino Uno и IBM IoT Foundation.

Приклад використання WiFi контролера ESP8266 разом з MQTT на прикладі плати ESP-12F наведено у статті Как заварить чай по MQTT или доступная умная розетка с контролем температуры и тока.

9.3 Організація обміну з використанням MQTT

9.3.1 Ініціалізація з'єднання

Повідомлення з командою MQTT CONNECT розпочинає з'єднання. Для встановлення з'єднання клієнт надсилає брокеру це повідомлення. Якщо повідомлення CONNECT оформлено невірно (відповідно до специфікації MQTT) або проходить занадто багато часу між відкриттям мережевого сокета та відправленням повідомлення про підключення, брокер завершує з'єднання. Така поведінка стримує зловмисних клієнтів, які можуть уповільнити роботу брокера. Доброзичливий клієнт MQTT 3 надсилає повідомлення про підключення із таким вмістом (серед іншого):

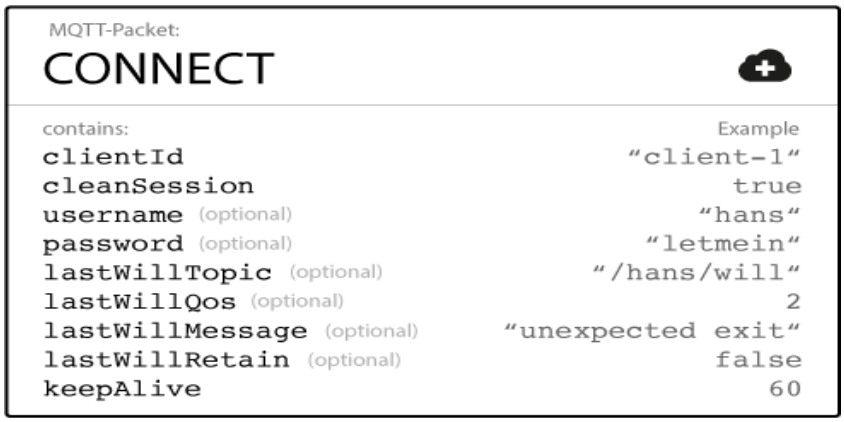


Рисунок 9.12 Приклад командного повідомлення CONNECT

Розглянемо більш детально наступні параметри повідомлення:

ClientId

Ідентифікатор клієнта (*ClientId*) ідентифікує кожного клієнта MQTT, який підключається до брокера MQTT. Брокер використовує *ClientId* для ідентифікації клієнта та поточного стану клієнта, тому цей ідентифікатор повинен бути унікальним для кожного клієнта та брокера. У MQTT 3.1.1

дозволено надсилати порожній *ClientId*, якщо клієнту не потрібен стан, який повинен утримувати брокер. Порожній *ClientId* призводить до з'єднання без будь-якого стану. У цьому випадку прапорець чистого сеансу *Clean Session* повинен бути встановлений в 1, інакше брокер відхилить підключення.

Clean Session

Цей прапорець повідомляє брокеру, чи хоче клієнт встановити постійний сеанс чи ні. У постійному сеансі (*CleanSession* = 0) брокер зберігає всі підписки для клієнта та всі пропущені повідомлення для клієнта, який підписався з QoS 1 або 2. Якщо сеанс не є постійним (*CleanSession* = 1), брокер нічого не зберігає для клієнта і видаляє всю інформацію з будь-якого попереднього постійного сеансу.

Username/password

MQTT може надсилати ім'я користувача та пароль для аутентифікації та авторизації клієнта. Однак, якщо ця інформація не зашифрована або не хешована (наприклад, за допомогою TLS), пароль надсилається простим текстом. Для промислових застосувань настійно рекомендується використовувати імена користувачів та паролі разом із безпечним транспортом. Такі брокери, як HiveMQ, можуть аутентифікувати клієнтів за допомогою SSL-сертифіката, тому ім’я користувача та пароль тут стають не потрібні.

Will Message

Текст повідомлення останньої волі. Є частиною функції останньої волі та заповіту (LWT) в MQTT. Це повідомлення сповіщає інших клієнтів, коли клієнт невдало відключається. Коли клієнт підключається до брокера, він може надати брокеру останню волю у вигляді повідомлення MQTT та теми в повідомленні CONNECT. Якщо клієнт невдало відключається, брокер надсилає повідомлення LWT від імені клієнта.

keepAlive

keepAlive - це інтервал часу в секундах, який клієнт вказує та повідомляє брокеру при встановленні зв’язку. Цей інтервал визначає найдовший проміжок часу, який брокер і клієнт можуть пережити, не надсилаючи повідомлення. Навіть при відсутності нової інформації клієнт зобов'язується надсилати брокеру регулярні повідомлення запиту PING не пізніше, ніж через час, встановлений в

keepAlive. Брокер також відповідає повідомленням PING. Цей метод дозволяє обом сторонам визначити, чи доступна інша.

В основному, це вся інформація, яка потрібна для підключення до брокера MQTT від клієнта MQTT 3.1.1. В окремих бібліотеках часто є додаткові параметри, які можна налаштувати. Наприклад, спосіб збереження повідомлень у черзі в конкретній реалізації.

Коли брокер отримує повідомлення CONNECT, він зобов'язаний відповісти повідомленням CONNACK. Повідомлення CONNACK містить два поля даних:

- session present flag;
- connect return code.

Session present flag

Флаг наявності стану попередньої сесії повідомляє клієнту, чи зберігся у брокера постійний сеанс, що відбувся за попередньою взаємодією з клієнтом. Коли клієнт підключається з *Clean Session*, встановленому на 1, прапор присутності сеансу завжди є помилковим, оскільки сеанс недоступний. Якщо клієнт підключається з *Clean Session*, встановлену на 0, є дві можливості: якщо інформація про сеанс доступна для *ClientId* і брокер зберіг інформацію про сеанс, *Session present flag* встановлений в 1. В іншому випадку, якщо брокер не має жодної інформації про попередній сеанс для *ClientId, Session present flag* ,elt встановлений в 0. Цей прапор було додано в MQTT 3.1.1, щоб допомогти клієнтам визначити, чи потрібно їм підписуватися на теми, чи теми все ще зберігаються в постійному сеансі.

Connect return code

Другий прапор у повідомленні CONNACK - це прапор підтвердження підключення. Цей прапорець містить ціле число - код повернення, який повідомляє клієнту, чи була спроба з'єднання успішною чи ні.

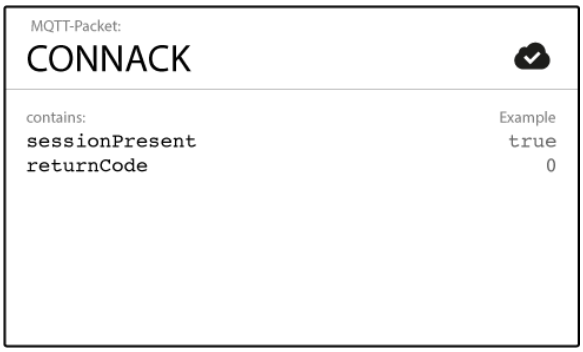


Рисунок 9.13 Приклад відповіді брокера повідомленням CONNACK

Відповідні коди повідомлень наведено у Таблиця 9.2.

9.3.2 Публікація повідомлень, підписка та відписка

Клієнт MQTT може публікувати повідомлення, як тільки підключається до брокера. MQTT використовує тематичну фільтрацію повідомлень про брокера. Кожне повідомлення повинно містити тему, за допомогою якої брокер може переслати повідомлення зацікавленим клієнтам. Як правило, кожне повідомлення має корисне навантаження, яке містить дані для передачі в байтовому форматі. MQTT є інформаційно-агностичним. Приклад використання клієнта визначає структуру корисного навантаження. Клієнт-відправник (видавець) вирішує, чи бажає він надіслати двійкові дані, текстові дані або навіть повноцінний XML або JSON.

Повідомлення PUBLISH у MQTT має кілька атрибутів, які зараз буде детально обговорено:

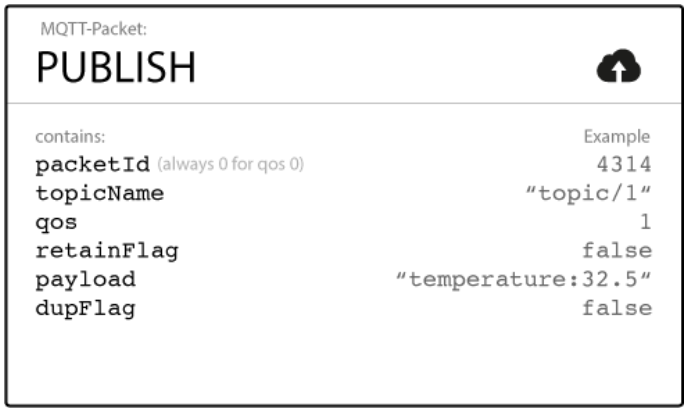


Рисунок 9.14 Атрибути в повідомлені PUBLISH

topicName - це простий рядок, який ієрархічно структурований косими рисками (слешем) як роздільниками. Наприклад, “myhome/вітальня/температура” або “Німеччина/Мюнхен/Жовтневий фестиваль/люди”.

QoS – це число вказує на рівень якості обслуговування (QoS) повідомлення. Існує три рівні: 0, 1 і 2. Рівень обслуговування визначає, яку гарантію надається повідомленню для передбачуваного отримання одержувачем (клієнтом або брокером).

retainFlag - цей прапор визначає, чи зберігається повідомлення брокером як останнє відоме значення для вказаної теми. Коли новий клієнт підписується на тему, він отримує саме це останнє повідомлення, яке зберігається за цією темою.

payload - це фактичний зміст повідомлення. MQTT є інформаційно-агностичним. Можна надсилати зображення, текст у будь-якому кодуванні, зашифровані дані та практично всі дані у двійковій формі.

Packet Identifier - ідентифікатор пакета однозначно ідентифікує повідомлення, яке надходить від клієнта до брокера. Ідентифікатор пакета актуальний лише для рівнів QoS, що перевищують нуль. Клієнтська бібліотека та / або брокер відповідають за встановлення цього внутрішнього ідентифікатора MQTT.

dupFlag - прапор вказує на те, що повідомлення є дублікатом, і його було повторно надіслано, оскільки призначений одержувач (клієнт або брокер) не отримав попереднє повідомлення. Це стосується лише якості обслуговування, що перевищує 0. Зазвичай механізм повторного надсилання / дублювання обробляється клієнтською бібліотекою MQTT або брокером як деталь реалізації.

Коли клієнт відправляє повідомлення брокеру MQTT для публікації, брокер читає повідомлення, підтверджує повідомлення (відповідно до рівня QoS) та обробляє повідомлення . Обробка брокером включає всі наперед встановлені визначення, на які клієнти попередньо підписані в темі, і подальші надсилання їм повідомлення.

Клієнта, який спочатку публікує повідомлення (тобто, публікатор), стурбований лише доставкою повідомлення PUBLISH брокеру. З моменту отримання брокером PUBLISH повідомлення, вже брокер несе відповідальність за доставку повідомлення всім підписникам. Публікатор не отримує від брокера жодного відгуку про те, чи цікавить когось опубліковане ним повідомлення, чи ні, або скільки підписників отримало повідомлення від брокера.

Публікація повідомлення не має сенсу, якщо його ніхто ніколи не отримує. Іншими словами, якщо немає клієнтів, які підписуються на теми повідомлень. Щоб отримувати повідомлення на цікаві теми, клієнт-підписник надсилає повідомлення SUBSCRIBE брокеру MQTT. Це повідомлення про підписку дуже просте, воно містить унікальний ідентифікатор пакета та список підписок:

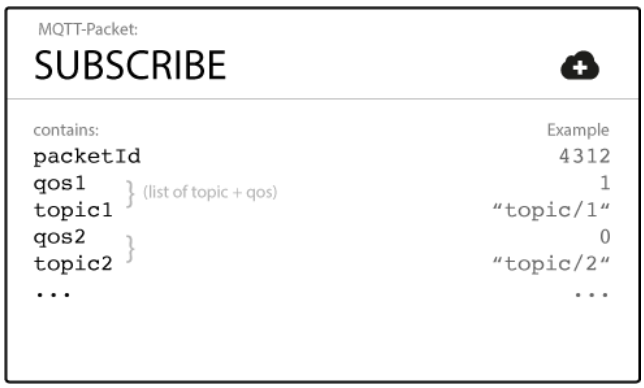


Рисунок 9.15 Приклад повідомлення SUBSCRIBE

Packet Identifier - ідентифікатор пакета однозначно ідентифікує повідомлення, яке надходить від клієнта до брокера. Клієнтська бібліотека та / або брокер відповідає за встановлення цього внутрішнього ідентифікатора MQTT.

Список підписок - повідомлення SUBSCRIBE може містити кілька підписок для клієнта. Кожна підписка складається з теми та рівня QoS. Тема в повідомленні про підписку може містити символи шаблону, що дозволяють підписатися на декілька тем, а не на певну тему. Якщо підписки накладаються на одного клієнта, брокер доставляє повідомлення з найвищим рівнем якості обслуговування для цієї теми.

Для підтвердження кожної підписки брокер надсилає клієнту повідомлення про підтвердження SUBACK. Це повідомлення містить ідентифікатор пакета оригінального повідомлення SUBSCRIBE (для чіткої ідентифікації повідомлення) та список кодів повернення:

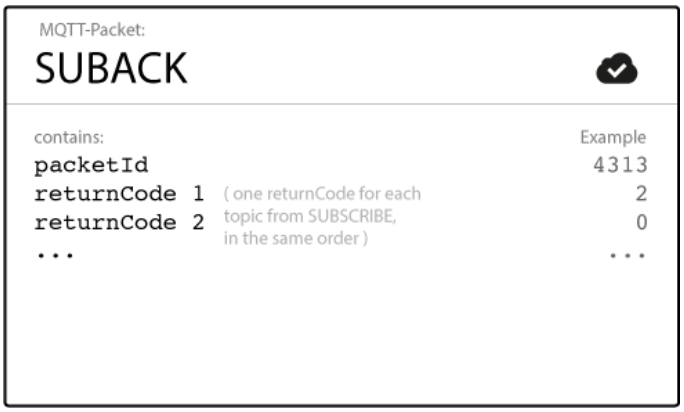


Рисунок 9.16 Вигляд повідомлення SUBACK

Брокер надсилає по одному коду повернення для кожної теми / пари QoS, яку він отримав в повідомленні SUBSCRIBE. Наприклад, якщо повідомлення SUBSCRIBE має п’ять підписок, повідомлення SUBACK буде містити п’ять кодів повернення. Код повернення розпізнає кожну тему та відображає рівень якості обслуговування, який надається брокером. Якщо брокер відмовляється від підписки, повідомлення SUBACK створює код повернення помилки для цієї конкретної теми. Наприклад, якщо у клієнта недостатній дозвіл на підписку на тему або тема неправильно сформована.

Після того, як клієнт успішно відправляє повідомлення SUBSCRIBE та отримує повідомлення SUBACK, він отримує кожне опубліковане повідомлення, яке відповідає темі в підписках, що містилися в повідомленні SUBSCRIBE.

Двійником SUBSCRIBE повідомлення є UNSUBSCRIBE повідомлення. Це повідомлення видаляє наявні підписки клієнта на брокера. Повідомлення UNSUBSCRIBE подібне до повідомлення SUBSCRIBE і має ідентифікатор пакета та список тем:

Рисунок 9.17 Повідомлення UNSUBSCRIBE

packetID - Ідентифікатор пакета однозначно ідентифікує повідомлення, яке надходить від підписника до брокера. Клієнтська бібліотека та / або брокер відповідає за встановлення цього внутрішнього ідентифікатора MQTT.

Список тем - може містити кілька тем, з яких клієнт хоче скасувати підписку. Потрібно лише надіслати тему (без QoS). Брокер скасовує підписку на тему, незалежно від рівня якості обслуговування, на який клієнт спочатку був підписан.

Щоб підтвердити відмову від підписки, брокер надсилає клієнту повідомлення про підтвердження відписки UNSUBACK . Це повідомлення містить лише ідентифікатор пакета оригінального повідомлення UNSUBSCRIBE (для чіткої ідентифікації повідомлення):

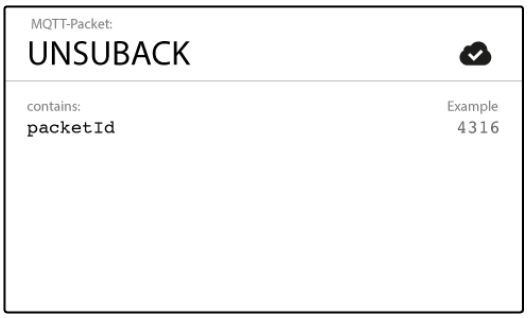


Рисунок 9.18 Повідомлення UNSUBACK

packetID однозначно ідентифікує повідомлення. Як уже зазначалося, це повторюється той самий ідентифікатор пакета, який містився в повідомленні UNSUBSCRIBE. Отримавши UNSUBACK від брокера, клієнт з цього моменту може вже не обробляти видалені підписки.

9.3.3 Шаблони і топіки в MQTT

Термін *топік* MQTT стосується рядка UTF-8, який використовує брокер для фільтрації повідомлень для кожного підключеного клієнта. Топік складається з одного або кількох рівнів топіків. Кожен рівень топіку відокремлений від іншого слешем (роздільник рівня):

місто / вулиця / квартира / температура, вологість

У порівнянні з чергою повідомлень, топіки MQTT дуже легкі. Клієнту не потрібно створювати бажаний топік перед тим, як його опублікувати або підписатися. Брокер приймає кожний дійсний топік без попередньої ініціалізації. Ось ще кілька прикладів топіків:

myhome / нижній поверх / вітальня / температура

США / Каліфорнія / Сан-Франциско / Силіконова долина

5ff4a2ce-e485-40f4-826c-b1a5d81be9b6 / статус

Німеччина / Баварія / автомобіль / 2382340923453 / широта

Зверніть увагу, що кожен топік повинна містити принаймні 1 символ і що рядок топіку дозволяє порожні пробіли. Топік чутливий до регістру. Наприклад, *myhome / temperature* та *MyHome / Temperature* - це два різні топіка. Крім того, лише слеш попереду назви є допустимим.

Коли клієнт підписується на топік, він може підписатися на точне найменування топіка опублікованого повідомлення або може використовувати символи шаблону, щоб підписатися на кілька топіків одночасно. Знак шаблону можна використовувати лише для підписки на топіки, а не для публікації повідомлення. Існує два різні типи шаблонів: однорівневі та багаторівневі.

Однорівневий: **+**

Як випливає з назви, однорівневий шаблон замінює один рівень топіка. Символ плюс представляє однорівневий шаблон у топіку. Наприклад,

Київ / Бажана3 / + / температура

замінить рівень дому на Бажана 3 в Києві температурами по усім квартирам. Будь-який топік відповідає топіку з однорівневим символом підстановки, якщо він містить довільний рядок замість символу підстановки. Наприклад, підписка на *myhome / groundfloor / + / temperature* може дати такі результати:

Рисунок 9.19 Приклад підписки з шаблоном *myhome/groundfloor/+/temperature*

Багаторівневий: **#**

Багаторівневий знак шаблона охоплює багато рівнів топіків. Символ хешу представляє багаторівневий знак шаблона у топіку. Щоб брокер визначив, які теми збігаються, багаторівневий знак шаблона повинен бути розміщений як останній символ у топіку, після слешу:

Рисунок 9.20 Приклад підписки з шаблоном *myhome/groundfloor/#*

Коли клієнт підписується на топік з багаторівневим символом шаблону, він отримує всі повідомлення топіку, який починається з шаблону перед символом підстановки, незалежно від того, наскільки довгий чи глибокий топік. Якщо вказати в імені топіка лише тему багаторівневого шаблону (#), клієнт буде отримувати всі повідомлення, які надсилаються брокеру MQTT. Якщо ви очікуєте високої пропускної здатності, підписка лише на багаторівневий знак шаблону є антикорисною.

Топіки, що починаються з $

Як правило, топіки MQTT можна називати як завгодно. Однак є один виняток: імена топіків, які починаються з символу $, мають інше призначення. Ці топики не є частиною передплати, коли підписуються на багаторівневий знак шаблону у якості теми (#). Теми *$-символ* зарезервовані для внутрішньої статистики брокера MQTT. Клієнти не можуть публікувати повідомлення на ці теми. На даний момент офіційної стандартизації таких тем не існує. Зазвичай *$SYS/* використовується для всієї наведеної нижче інформації, але залежить від реалізації брокера. Одна пропозиція щодо *$SYS-моніків* існує у *wiki MQTT GitHub* . Ось кілька прикладів:

$SYS/broker/clients/connected

$SYS/broker/clients/disconnected

$SYS/broker/clients/total

$SYS/broker/messages/sent

$SYS/broker/uptime

Кращі практики у використанні імен топіків та символів шаблону:

Ніколи не використовуйте слеш перед іменем

Застосування символ слешу першим в імені топіку дозволено в MQTT. Наприклад, */myhome/цокольний поверх/вітальня*. Однак початковий слеш вводить непотрібний рівень топіку з нульовим символом спереду. Нуль не дає ніякої вигоди і часто призводить до плутанини.

Ніколи не використовуйте пробіли в темі топіка

Простір - природний ворог кожного програміста. Коли справи йдуть не так, як слід, пробіли значно ускладнюють читання та налагоджування топіків. Як і у випадку переднього слешу, те, що щось дозволено, не означає, що це слід використовувати. UTF-8 має багато різних типів пробілів, таких незвичайних символів слід уникати.

Нехай тема топіку буде короткою та лаконічною

Кожна тема обов'язково включається в кожне повідомлення, в якому вона використовується. Зробіть свої теми максимально короткими та стислими. Що стосується невеликих пристроїв, кожен байт має значення і довжина теми має велике значення.

Використовуйте лише символи ASCII, уникайте недрукованих символів

Оскільки символи UTF-8, що не належать до ASCII, часто відображаються неправильно, дуже важко знайти помилки друку або проблеми, пов’язані з набором символів. Якщо це абсолютно не потрібно, рекомендується уникати використання символів, що не є символами ASCII, у назві теми.

Вбудуйте в тему унікальний ідентифікатор або ідентифікатор клієнта

Дуже корисно буде включити до теми топіку унікальний ідентифікатор клієнта-публікатора. Унікальний ідентифікатор у темі допомагає визначити, хто надіслав повідомлення. Вбудований ідентифікатор можна використовувати для забезпечення авторизації. Публікувати в цьому топіку дозволено лише клієнту, який має той самий ідентифікатор клієнта, що і ідентифікатор у темі. Наприклад, клієнту з ідентифікатором *client1* дозволено публікувати в *client1/status* , але заборонено публікувати в *client2/status*.

Не підписуйтесь на #

Іноді потрібно підписатися на всі повідомлення, які передаються через брокера. Наприклад, зберігати всі повідомлення у базі даних. Не підписуйтесь на всі повідомлення брокера, використовуючи клієнт MQTT та підписуючись на багаторівневий шаблон. Часто клієнт, що передплачує, не в змозі обробити навантаження повідомлень, що виникає в результаті цього методу (особливо якщо у вас велика пропускна здатність). Загальна рекомендація полягає у впровадженні розширення в брокері MQTT. Наприклад, за допомогою системи додаткових плагінів брокера HiveMQ можливо підключити поведінку HiveMQ і додати асинхронну процедуру для обробки кожного вхідного повідомлення та збереження його в базі даних.

Не забувайте про розширюваність дерева тем

Теми - це гнучка концепція, і немає необхідності їх попередньо розподіляти будь-яким способом. Однак і публікатор, і підписник повинні бути в курсі теми. Важливо продумати, як можна розширити теми, щоб дозволити нові функції або продукти. Наприклад, якщо рішення для об'єкту управління у майбутньому додає нові давачі, має бути можливо додати їх до існуючого дерева тем, не змінюючи всієї ієрархії тем.

Використовуйте імена конкретних тем, а не загальні

Коли ви називаєте теми, не називайте їх так само, як і при використанні черги. Максимально диференціюйте свої теми. Наприклад, якщо у вашій вітальні є три давачі, створіть теми для *myhome/вітальня/температура*, *myhome/вітальня /яскравість* та *myhome/вітальня/вологість*. Не надсилайте всі дані через *myhome/вітальня*. Використання однієї теми для всіх повідомлень є прикладом анти-шаблона. Конкретне іменування також дозволяє використовувати інші функції MQTT, такі як збережені повідомлення, тощо.

9.3.4 Використання QoS

Рівень якості обслуговування (QoS) - це угода між відправником та одержувачем повідомлення, яка визначає гарантію доставки конкретного повідомлення. Як ми вже знаємо, у MQTT є три рівні QoS, від 0 до 2.

Коли ми говоримо про QoS в MQTT, слід враховувати дві сторони доставки повідомлень:

- *доставка повідомлення від клієнта-видавця до брокера;*
- *доставка повідомлень від брокера клієнту, який підписався.*

Ми розглянемо дві сторони доставки повідомлень окремо, оскільки між ними є незначні відмінності. Клієнт, який публікує повідомлення брокеру, визначає QoS повідомлення, коли він надсилає повідомлення брокеру. Брокер передає це повідомлення клієнтам-підписникам, використовуючи рівень QoS, який кожен підписуваний клієнт визначає під час підписки. Якщо клієнт, який підписався, визначає нижчий рівень якості обслуговування, ніж клієнт-видавець, брокер передає підписнику повідомлення з нижчою якістю обслуговування.

Чому якість обслуговування важлива? QoS - це ключова особливість протоколу MQTT. QoS надає клієнту можливість вибрати рівень сервісу, який відповідає надійності його мережі та логіці застосування. Оскільки MQTT керує повторною передачею повідомлень і гарантує доставку (навіть коли базовий транспорт не є надійним), QoS значно полегшує спілкування в ненадійних мережах. Розглянемо детальніше різновиди і використання QoS. Відповідні малюнки дивитися на рис. 9.6 – 9.8.

QoS 0 - не більше одного разу

Мінімальний рівень якості обслуговування дорівнює нулю. Цей рівень обслуговування гарантує найкращі зусилля. Гарантії доставки немає. Одержувач не підтверджує отримання повідомлення, і повідомлення не зберігається та не передається повторно відправником. Рівень QoS 0 часто називають "вистрілив і забув" і забезпечує ту саму гарантію, що і базовий протокол TCP.

QoS 1 - принаймні один раз

QoS рівень 1 гарантує, що повідомлення принаймні один раз доставляється одержувачу. Відправник зберігає повідомлення, поки не отримає від одержувача пакет PUBACK, який підтверджує отримання повідомлення. Повідомлення можна надсилати або доставляти кілька разів.

Відправник використовує ідентифікатор пакета в кожному пакеті, щоб зіставити пакет PUBLISH з відповідним пакетом PUBACK. Якщо відправник не отримує пакет PUBACK за розумний проміжок часу, відправник повторно

відправляє пакет PUBLISH. Коли одержувач отримує повідомлення з QoS 1, він може негайно його обробити. Наприклад, якщо одержувач є брокером, брокер зразу надсилає повідомлення всім підписаним клієнтам, а потім відповідає пакетом PUBACK публікатору.

Якщо клієнт-публікатор надсилає повідомлення знову, він встановлює прапор дубліката (DUP). У QoS 1 цей прапор DUP використовується лише для внутрішніх цілей і не обробляється брокером або клієнтом. Одержувач повідомлення надсилає PUBACK, незалежно від прапора DUP.

QoS 2 - рівно один раз

QoS 2 - це найвищий рівень сервісу в MQTT. Цей рівень гарантує, що кожне повідомлення отримується призначеними одержувачами лише один раз. QoS 2 - це найбезпечніший і найповільніший рівень якості обслуговування. Гарантія забезпечується щонайменше двома потоками запитів / відповідей ("рукопотискання" у чотири фази) між відправником та одержувачем. Відправник і одержувач використовують ідентифікатор пакета оригінального повідомлення PUBLISH для координації доставки повідомлення.

Коли одержувач отримує пакет QoS 2 PUBLISH від відправника, він відповідно обробляє повідомлення публікації та відповідає відправнику пакетом PUBREC, який підтверджує пакет PUBLISH . Якщо відправник не отримує від одержувача пакет PUBREC, він знову надсилає пакет PUBLISH із позначкою дубліката (DUP), поки не отримає підтвердження.

Як тільки відправник отримує від приймача пакет PUBREC, відправник може безпечно відкинути початковий пакет PUBLISH. Відправник зберігає пакет PUBREC у приймача і відповідає пакетом PUBREL .

Після того, як одержувач отримує пакет PUBREL, він може відкинути всі збережені стани і відповісти пакетом PUBCOMP (те саме стосується і відправника, який отримує PUBCOMP). Поки приймач не завершить обробку і не відправить пакет PUBCOMP назад відправнику, приймач зберігає посилання на ідентифікатор пакета оригінального пакета PUBLISH. Цей крок важливий, щоб уникнути обробки повідомлення вдруге. Після того, як відправник отримує пакет PUBCOMP, ідентифікатор пакета опублікованого повідомлення стає доступним для повторного використання.

Коли потік QoS 2 завершено, обидві сторони впевнені, що повідомлення доставлено, а відправник має підтвердження доставки.

Якщо пакет загубиться в дорозі, відправник відповідає за повторну передачу повідомлення протягом розумного періоду часу. Це однаково вірно, якщо відправник є клієнтом MQTT або брокером MQTT . Одержувач несе відповідальність за відповідь на кожне командне повідомлення на його адресу відповідно.

Деякі аспекти якості обслуговування на перший погляд не дуже очевидні. Ось кілька речей, про які слід пам’ятати, використовуючи QoS:

Пониження якості обслуговування

Як ми вже згадували, визначення QoS та рівні між клієнтом, який надсилає (публікує) повідомлення, та клієнтом, який отримує повідомлення, - це дві різні речі. QoS цих двох взаємодій також можуть бути різними. Клієнт, який надсилає брокеру повідомлення PUBLISH, визначає QoS повідомлення. Однак, коли брокер доставляє повідомлення одержувачам (абонентам), брокер використовує QoS, який отримувач (абонент) визначив для себе під час підписки. Наприклад, клієнт A - це відправник повідомлення. Клієнт B є одержувачем повідомлення. Якщо клієнт B підписується на брокера з QoS 1, а клієнт A відправляє повідомлення брокеру з QoS 2, брокер доставляє повідомлення клієнту B (одержувач / абонент) з QoS 1. Повідомлення може бути доставлено більше одного разу клієнту B, оскільки QoS 1 гарантує доставку повідомлення принаймні один раз і не перешкоджає багаторазовій доставці одного і того ж повідомлення.

Ідентифікатори пакетів унікальні для кожного клієнта

Ідентифікатор пакета, який MQTT використовує для QoS 1 та QoS 2, є унікальним між конкретним клієнтом та брокером у межах взаємодії. Цей ідентифікатор не є унікальним для всіх клієнтів. Після завершення потоку ідентифікатор пакета стане доступним для повторного використання. Це повторне використання є причиною того, чому ідентифікатор пакета не повинен перевищувати 65535. Нереально, що клієнт може надіслати більше цієї кількості повідомлень, не завершивши взаємодії.

Слід використовувати QoS 0, коли…

У вас є повністю або в основному стабільний зв’язок між відправником та одержувачем. Класичним варіантом використання QoS 0 є підключення тестового клієнта або інтерфейсної програми до посередника MQTT через дротове з'єднання.

Ви не проти, якщо зрідка втрачається кілька повідомлень. Втрата деяких повідомлень може бути прийнятною, якщо дані не так важливі або коли дані надсилаються через короткі проміжки часу.

Вам не потрібні черги повідомлень. Повідомлення ставляться в чергу лише для відключених клієнтів, якщо вони мають QoS 1 або 2 і постійний сеанс.

Слід використовувати QoS 1, коли ...

Вам потрібно отримати кожне повідомлення, і ваш варіант використання може обробляти дублікати. Рівень якості 1 є найбільш часто використовуваним рівнем обслуговування, оскільки він гарантує, що повідомлення надходить принаймні один раз, але дозволяє здійснювати багаторазові доставки. Звичайно, використовуваний програмовий додаток повинен допускати дублікати та мати можливість їх відповідно обробляти.

Ви не можете нести накладні витрати на QoS 2. QoS 1 доставляє повідомлення набагато швидше, ніж QoS 2.

Використовуйте QoS 2, коли ...

Важливо, щоб ваша програма отримувала всі повідомлення рівно один раз. Це часто трапляється, якщо дублікат доставки може завдати шкоди користувачам програми або клієнтам, які підписалися. Слід пам’ятати про накладні витрати та про те, що взаємодія QoS 2 займає більше часу.

Черги повідомлень QoS 1 та 2

Усі повідомлення, надіслані з QoS 1 і 2, ставляться в чергу для офлайн-клієнтів, поки клієнт знову не стане доступним. Однак така черга можлива лише *за умови постійного сеансу клієнта*.

9.3.5 Постійна сессія та черга повідомлень

Щоб отримувати повідомлення від брокера MQTT , клієнт підключається до брокера і створює підписки на теми, які його цікавлять. Якщо зв’язок між

клієнтом і посередником перервано під час нестійкого сеансу, ці теми втрачаються, і клієнту потрібно знову підписатися на повторне підключення. Повторна підписка кожного разу, коли з’єднання переривається, є тягарем для клієнтів з обмеженими ресурсами. Щоб уникнути цієї проблеми, клієнт може вимагати постійного сеансу, коли він підключається до брокера. Постійні сеанси зберігають у брокера всю необхідну для клієнта інформацію. *ClientId*, що клієнт надає , коли він встановлює з'єднання з брокером ідентифікує сеанс.

Що зберігається у постійному сеансі?

Під час постійного сеансу брокер зберігає таку інформацію (навіть якщо клієнт у режимі офлайн):

- наявність сеансу (навіть якщо підписки відсутні);
- всі підписки клієнта;
- усі повідомлення з QoS 1 або 2, які клієнт ще не підтвердив;
- усі нові повідомлення QoS 1 або 2, які клієнт пропустив у режимі офлайн;
- усі повідомлення QoS 2, отримані від клієнта, які ще не повністю підтверджені.

Коли клієнт повторно підключається, ця інформація стає доступною для нього негайно.

Як розпочати або закінчити постійний сеанс?

Коли клієнт підключається до брокера, він може вимагати постійного сеансу. Клієнт використовує прапор *cleanSession*, щоб повідомити брокеру, який сеанс йому потрібен:

- коли прапорець *cleanSession* встановлено на *true*, клієнт не хоче постійного сеансу. Якщо клієнт від'єднується з будь-якої причини, вся інформація та повідомлення, які потрапляють у чергу з попереднього постійного сеансу, втрачаються;
- коли прапорець *cleanSession* встановлено на *false*, брокер створює постійний сеанс для клієнта. Вся інформація та повідомлення зберігаються до наступного разу, коли клієнт запитує чистий сеанс. Якщо для прапорця *cleanSession* встановлено *false*, і брокер вже має сеанс, доступний для

клієнта, він використовує наявний сеанс і доставляє клієнту повідомлення, що стояли в черзі.

Як клієнт дізнається, чи вже зберігається сеанс?

Починаючи з MQTT 3.1.1, повідомлення CONNACK від брокера містить *session present flag*. Цей прапорець повідомляє клієнту, чи раніше встановлений сеанс все ще доступний у брокера.

Постійний сеанс на стороні клієнта

Подібно брокеру, кожен клієнт MQTT також повинен зберігати постійний сеанс. Коли клієнт просить сервер зберігати дані сеансу, клієнт відповідає за збереження наступної інформації:

- усі повідомлення в потоці QoS 1 або 2, які ще не підтверджені брокером;
- усі повідомлення QoS 2, отримані від брокера, які ще не повністю підтверджені.

Ось декілька рекомендацій, які можуть допомогти вирішити, коли використовувати постійний сеанс чи чистий сеанс:

Постійна сесія

Клієнт повинен отримувати всі повідомлення з певної теми, навіть якщо вона не в мережі:

- потрібно, щоб брокер розміщував повідомлення в черзі для клієнта та доставляв їх, як тільки клієнт знову з'явиться в мережі;
- клієнт має обмежені ресурси. Потрібно, щоб брокер зберігав інформацію про підписку клієнта і швидко відновлював перерваний зв'язок;
- клієнт повинен відновити всі публікації повідомлень QoS 1 і 2 після повторного підключення.

Чистий сеанс

- клієнту потрібно лише публікувати повідомлення на теми, клієнту не потрібно підписуватися на теми. Не потрібно, щоб брокер зберігав інформацію про сеанси або повторював передачу повідомлень QoS 1 та 2;
- клієнту не потрібно отримувати повідомлення, які він пропускає в автономному режимі.

Як довго брокер зберігає повідомлення?

Люди часто запитують, як довго брокер зберігає сесію? Проста відповідь: брокер зберігає сесію, доки клієнти не повернуться в Інтернет і не отримають повідомлення. Однак що трапиться, якщо клієнт тривалий час не повертається в Інтернет? Зазвичай обмеження пам’яті операційної системи є основним обмеженням для зберігання повідомлень. Стандартної відповіді на цей сценарій немає. Правильне рішення залежить від конкретного випадку використання.

9.3.6 Робота зі збереженими у брокера повідомленнями

У MQTT клієнт, який публікує повідомлення, не має гарантії того, що клієнт, що підписався, насправді отримає повідомлення. Клієнт-публікатор може лише переконатися, що повідомлення надійно доставляється брокеру. В основному, те саме стосується і клієнта-підписника. Клієнту, який підключається до тем та підписується на них, не гарантується, що клієнт-публікатор опублікує повідомлення в одній із тем, що цікавлять. Публікатор може надіслати нове повідомлення в одній із тем, на які клієнт підписався, через кілька секунд, хвилин або годин (якщо не днів). Поки не буде опубліковано наступне повідомлення, клієнт, що підписаний, знаходиться в повній темряві щодо поточного стану теми. Це ситуація, коли збережені повідомлення вступають у гру.

Збережене повідомлення - це звичайне повідомлення MQTT із прапором *retain*, встановленим у значення *true*. Брокер зберігає останнє збережене повідомлення та відповідний йому QoS для цього топіка. Кожен клієнт, який підписався на шаблон теми, що відповідає темі збереженого повідомлення, отримує збережене повідомлення відразу після підписки. Брокер зберігає лише одне збережене повідомлення для кожної теми.

Якщо підписник включає символи підстановки в шаблон теми, на який підписується, він отримує збережене повідомлення, навіть якщо тема збереженого повідомлення не відповідає точно. Ось приклад: Клієнт А публікує збережене повідомлення для *myhome/вітальня/температура*. Дещо пізніше клієнт В підписується на *myhome/#*. Тоді клієнт В отримує збережене повідомлення у *myhome/вітальня/температура* безпосередньо після підписки на *myhome/#*. Клієнт В (клієнт, що підписується) може бачити, що повідомлення є

збереженим повідомленням, оскільки брокер надсилає збережені повідомлення із прапором *retain*, встановленим на *true*. Клієнт вже сам може вирішити, як він хоче обробити збережені повідомлення.

Збережені повідомлення допомагають знов підписаним клієнтам отримувати оновлення стану відразу після підписки на тему. Збережене повідомлення виключає очікування підписником публікації нового повідомлення публікатором у топіку. Іншими словами, збережене повідомлення у топіку - це останні відомі корисні дані теми. Слід мати на увазі, що збережене повідомлення може і не бути останнім значенням даних (оскільки після нього публікатором можуть передаватися повідомлення у топік без встановленого прапору *retain*), але воно повинно бути останнім повідомленням із прапором *retain*, встановленим у значення *true*.

Важливо розуміти, що збережене повідомлення не має нічого спільного з постійними сесіями . Як тільки збережене повідомлення зберігається брокером, існує лише один спосіб його видалення.

Надіслати збережене повідомлення

З точки зору розробника, надсилання збереженого повідомлення досить просте і прямолінійне: просто встановлюється для прапора *retain* повідомлення PUBLISH MQTT значення *true*. Це забезпечує клієнтська бібліотека MQTT (простий спосіб встановити цей прапор).

Видалення збереженого повідомлення

Існує також дуже простий спосіб видалення збереженого повідомлення теми: слід надіслати нове збережене повідомлення з нульовим байтом корисного навантаження у топік, де потрібно видалити попереднє збережене повідомлення. Брокер видаляє збережене повідомлення, а нові абоненти більше не отримують збережене повідомлення для цієї теми. Часто навіть видаляти не потрібно, оскільки кожне нове повідомлення, яке зберігається, замінює попереднє.

Чому і коли слід використовувати збережені повідомлення?

Збережене повідомлення має сенс, коли необхідно, щоб нещодавно підключені абоненти отримували повідомлення негайно (не чекаючи, поки публікатор надішле наступне повідомлення). Це надзвичайно корисно для оновлення статусу компонентів або пристроїв за окремими темами. Наприклад,

статус *device1* знаходиться в топіку *myhome/devices/device1/status*. Коли використовуються збережені повідомлення, нові абоненти теми отримують статус (онлайн / офлайн) пристрою відразу після підписки. Те саме стосується клієнтів, які надсилають дані з інтервалами, температуру, координати GPS та інші дані. Без збережених повідомлень нові абоненти залишаються без даних між інтервалами публікації. Використання збережених повідомлень допомагає негайно надати останнє хороше значення для клієнта, що підключається.

9.3.7 Використання останньої волі і заповіту LWT

Оскільки MQTT часто використовується у сценаріях, що включають ненадійні мережі, розумно припустити, що деякі клієнти MQTT у цих сценаріях час від часу стануть випадково відключатися. Неправильне роз'єднання може статися через втрату з'єднання, розряжений акумулятор або багато інших причин. Своєчасна інформація про те, як саме відключився клієнт: правильно (із повідомленням MQTT DISCONNECT) чи ні (без повідомлення про відключення), допомагає правильно побудувати подальші дії. Функція LWT якраз і надає можливість клієнтам реагувати на неправильні роз'єднання відповідним чином.

Функція LWT використовується, щоб повідомляти інших клієнтів про неправильно відключений клієнт. Кожен клієнт може вказати свою останню волю у тексті відповідного поля, коли він підключається до брокера. Останнє повідомлення - це звичайне повідомлення MQTT з темою, прапором *lastWillRetaine*, QoS та корисним навантаженням. Брокер зберігає це повідомлення, доки не виявить, що клієнт невдало відключився. У відповідь на невдале відключення брокер надсилає повідомлення останньої волі всім підписаним клієнтам топіка з останньою волею. Якщо клієнт правильно відключається за допомогою повідомлення DISCONNECT, брокер відкидає збережене повідомлення LWT.

Рисунок 9.21 Повідомлення DISCONNECT

LWT допомагає реалізовувати різні стратегії, коли з'єднання клієнта перестає функціонувати (або принаймні може інформувати інших зацікавлених клієнтів про стан офлайн).

Як визначається повідомлення LWT для клієнта?

Клієнти можуть вказати повідомлення LWT у повідомленні CONNECT, яке ініціює з'єднання між клієнтом та брокером, так як наведено у 9.3.2.

Коли брокер надсилає повідомлення LWT?

Відповідно до специфікації MQTT 3.1.1 , брокер повинен поширити LWT клієнта в таких ситуаціях:

- брокер виявляє помилку введення/виведення або збій мережі;
- клієнт не може спілкуватися протягом визначеного періоду *Keep Alive*;
- клієнт не надсилає пакет DISCONNECT до того, як закриє мережеве з'єднання;
- брокер закриває підключення до мережі через помилку протоколу.

Коли слід використовувати LWT?

LWT - чудовий спосіб повідомити інших підписаних клієнтів про несподівану втрату з'єднання іншого клієнта. У реальних сценаріях LWT часто поєднується із збереженими повідомленнями для зберігання стану клієнта за певною темою. Наприклад, *client1* спочатку надсилає брокеру повідомлення CONNECT з *lastWillMessage*, в якому корисним навантаженням є “Offline”, прапором *lastWillRetain* встановлено значення *true* і *lastWillTopic* встановлено *client1/status*. Далі клієнт надсилає повідомлення PUBLIC з корисним навантаженням «Online» та прапором *retain*, встановленим як *true* для тієї ж теми (*client1/status*). Поки *client1* залишається на зв'язку, нещодавно підписані клієнти на тему *client1/status* отримують “Online” збережене повідомлення. Якщо *client1* несподівано відключається, брокер публікує повідомлення LWT з корисним

навантаженням "Offline" як нове збережене повідомлення. Клієнти, які підписалися на тему, поки *client1* перебуває в автономному режимі, отримують LWT збережене повідомлення брокера "Offline". Такий шаблон збережених повідомлень інформує інших клієнтів про поточний стан *client1* у певному топіку.

9.3.8 Сердцебиття і поглинання клієнта

Проблема напіввідкритих TCP-з'єднань

MQTT базується на протоколі управління передачею TCP . Цей протокол забезпечує передачу пакетів через Інтернет *надійним, упорядкованим та з перевіркою помилок* засобом. Тим не менше, час від часу передача між сторонами, що спілкуються, може виходити з синхронізації, наприклад, якщо одна зі сторін аварійно завершує роботу або має помилки передачі. У TCP цей стан неповного з'єднання називається напіввідкритим з'єднанням. Важливо пам’ятати, що при цьому одна сторона спілкування продовжує функціонувати і не отримує повідомлення про збій іншої сторони. Сторона, яка все ще залишається на зв’язку, продовжує намагатися надсилати повідомлення та чекає підтвердження. Як зазначає Енді Стенфорд-Кларк (винахідник протоколу MQTT), проблема з напіввідкритими з'єднаннями зростає в мобільних мережах:

"Хоча теоретично TCP/IP сповіщає вас про розрив сокета, на практиці, особливо щодо таких речей, як мобільні та супутникові лінії зв'язку, які часто "підробляють" TCP в ефірі і передають пакети з кінця в кінець, цілком можливо для TCP сесії перетворення у "чорну діру", тобто вона, здається, все ще відкрита, але насправді просто скидає все, що ви їй пишете, "на підлогу".

MQTT Keep Alive

MQTT включає функцію *KeepAlive*, яка забезпечує обхідний шлях для вирішення проблеми напіввідкритих з'єднань (або принаймні дає можливість оцінити, чи з'єднання все ще остається відкритим).

KeepAlive гарантує, що зв’язок між брокером і клієнтом все ще залишається відкритим і що брокер, і клієнт знають про зв’язок. Коли клієнт встановлює підключення до брокера, клієнт повідомляє брокеру проміжок часу в секундах.

Цей інтервал визначає максимальний проміжок часу, протягом якого брокер і клієнт можуть не спілкуватися між собою.

Специфікація MQTT говорить наступне :

"Keep Alive ..." - це максимальний інтервал часу, який дозволено пройти між моментом закінчення Клієнтом передачі одного Пакету Керування, і моментом, в який він починає відправляти наступний. Клієнт несе відповідальність за те, щоб інтервал між відправленими пакетами керування не перевищував значення Keep Alive. За відсутності відправки будь-яких інших пакетів керування Клієнт ПОВИНЕН надіслати пакет PINGREQ."

Поки повідомленнями обмінюються часто і інтервал *KeepAlive* не перевищується, немає необхідності надсилати додаткове повідомлення, щоб встановити, чи зв’язок все ще відкритий.

Якщо клієнт не надсилає повідомлення протягом періоду *KeepAlive*, він повинен відправити пакет PINGREQ брокеру, щоб підтвердити, що він доступний, і переконатися, що брокер також все ще доступний.

Брокер повинен відключити клієнта, який не надсилає повідомлення або пакет PINGREQ за час, що у півтора рази перевищує *KeepAlive*. Також клієнт повинен розірвати з'єднання, якщо він не отримає відповідь від брокера за розумний проміжок часу.

Відслідковування сердцебиття

Давайте уважніше розглянемо повідомлення про сердцебиття. Функція *KeepAlive* використовує два пакети:

Рисунок 9.22 Пакет PINGREQ

PINGREQ надсилається клієнтом і вказує брокеру, що клієнт ще живий. Якщо клієнт не надсилає будь-який інший тип пакетів (наприклад, пакет PUBLISH або SUBSCRIBE), клієнт повинен відправити пакет PINGREQ брокеру. Клієнт може надіслати пакет PINGREQ у будь-який час, коли хоче

підтвердити, що мережеве з'єднання все ще живе. Пакет PINGREQ не містить корисного навантаження.

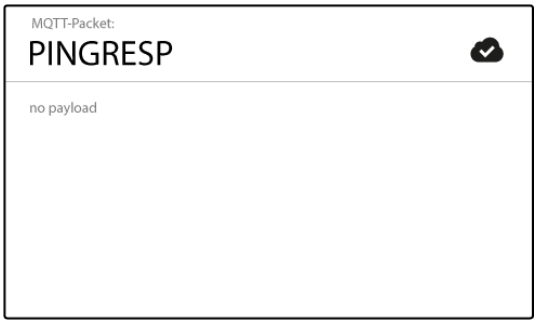


Рисунок 9.23 Пакет PINGRESP

Коли брокер отримує пакет PINGREQ, брокер повинен відповісти пакетом PINGRESP, щоб показати клієнту, що він все ще доступний. Пакет PINGRESP також не містить корисного навантаження.

Слід знати:

- якщо брокер не отримує PINGREQ або будь-який інший пакет від клієнта, брокер розриває з'єднання і відправляє LWT (якщо клієнт вказав LWT);
- клієнт MQTT несе відповідальність за встановлення відповідного значення *KeepAlive*. Наприклад, клієнт може відрегулювати інтервал сердцебиття відповідно до поточної сили сигналу;
- максимальний інтервал *KeepAlive* - 18 годин 12 хвилин 15 секунд;
- якщо KeepAlive встановлено в 0, механізм сердцебиття вимикається.

Поглинання клієнта

Зазвичай відключений клієнт намагається відновити зв’язок. Іноді брокер все ще має напіввідкрите з'єднання для клієнта. У MQTT, якщо брокер виявляє напіввідкрите з'єднання, він виконує "поглинання клієнта". Брокер закриває попереднє з’єднання з цим клієнтом (що визначається ідентифікатором клієнта) та встановлює новий зв’язок з клієнтом. Така поведінка гарантує, що напіввідкрите з'єднання не заважає відключеному клієнту відновити з'єднання.

9.4 Використання WEB-сокетів

WebSocket - протокол зв'язку поверх TCP-з'єднання, призначений для обміну повідомленнями між браузером і веб-сервером в режимі реального часу. Прикладний програмний інтерфейс WebSocket був стандартизований W3C, крім того протокол WebSocket стандартизований IETF як RFC 6455.

WebSocket розроблений для втілення у веб-браузерах і веб-серверах, але він може бути використаний для будь-якого клієнтського або серверного додатка. Протокол WebSocket - це незалежний протокол, заснований на протоколі TCP. Технологія дозволяє створювати інтерактивне з'єднання між клієнтом (браузером) та сервером для обміну повідомленнями в режимі реального часу. Веб-сокети, на відміну від HTTP, дозволяють працювати з двонаправленим потоком даних, що робить цю технологію придатною для використання в IIoT.

9.4.1 Як працює HTTP

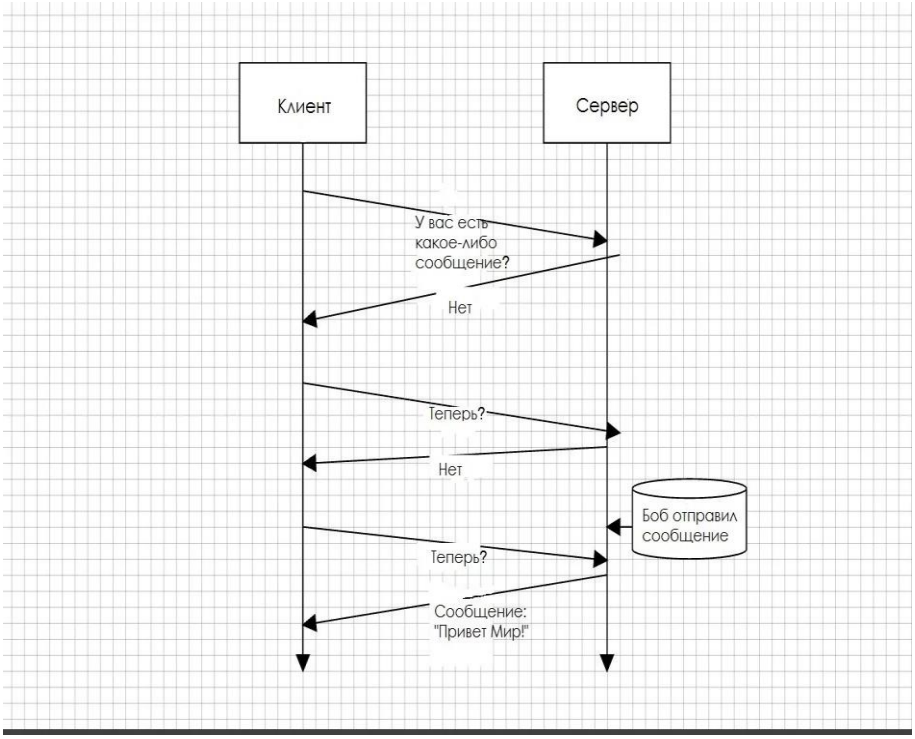


Рисунок 9.24 Схема обміну повідомленнями по HTTP

Ви напевно знаєте, що таке HTTP (або HTTPS), оскільки зустрічаєтеся з цим протоколом кожен день у своєму браузері. Браузер постійно запитує у сервера, чи є для нього нові повідомлення, і отримує їх (Рисунок 9.24).

Ви також можете знати, що HTTP дозволяє використовувати різні типи запитів, такі як POST, GET або PUT, кожен з яких має своє призначення.

9.4.2 Як працюють WEB-сокети

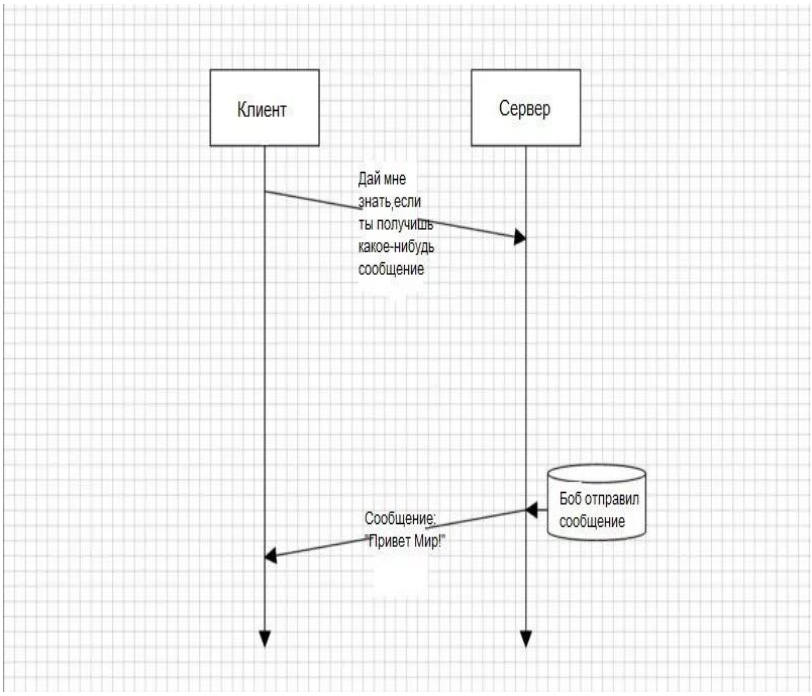


Рисунок 9.25 Схема обміну повідомленнями при використанні WEB-сокетів

Веб-сокетам для відповіді не потрібні наші повторювані запити. Досить виконати один запит і чекати відгуку (Рисунок 9.25). Можна просто слухати сервер, який буде відправляти нам повідомлення по мірі готовності.

Веб-сокети можна використовувати, якщо ми розробляємо:

- додатки реального часу;
- чат-додатки;
- IoT-додатки.

WebSocket - це керований подіями повнодуплексний асинхронний канал зв'язку для наших веб-додатків. Він має можливість надавати нам дані в режимі реального часу, які раніше (при використанні HTTP) ми отримували за допомогою програмного полінга. Основною перевагою є зниження потреб в ресурсах як на клієнті, так і (що більш важливо) на сервері.

Хоча WebSocket використовує HTTP в якості нижнього транспортного механізму, обмін даними не припиняється після отримання відповіді клієнтом. Використовуючи API WebSocket, стає можливим звільнитися від обмежень типового циклу HTTP-запиту / відповіді. Це також означає, що поки з'єднання залишається відкритим, клієнт і сервер можуть вільно відправляти повідомлення асинхронно, не опитуючи кожен раз один одного по новому.

9.4.3 Робота з WEB-сокетами

Для роботи з WEB-сокетами нам знадобиться WEB-сервер. У якості такого часто використовують *Nginx*, чи прості сервера, написані на Сі, типу *gwsocket*.

9.4.3.1 Відкриття WEB-сокета

Конструктору для WebSocket потрібна URL-адреса для ініціювання з'єднання з сервером. За замовчуванням, якщо після хоста не вказаний порт, він підключиться через порт 80 (порт HTTP) або порт 443 (порт HTTPS). Щоб відкрити WEB-сокет з'єднання, нам потрібно створити об'єкт *new WebSocket*, вказавши в *url-адресі* спеціальний протокол ***ws***:

```
socket = new WebSocket ("ws://testserver.info");
```

Також існує протокол *wss://*, який використовує шифрування. Це як HTTPS для веб-сокетів. Протокол *wss://* не тільки використовує шифрування, але і володіє підвищеною надійністю. Дані, що передаються по *ws://* не зашифровані, доступні для будь-якого посередника.Протокол *wss://* - це WebSocket поверх TLS (так само, як HTTPS - це HTTP поверх TLS), безпечний транспортний рівень шифрує дані від відправника і розшифровує на стороні одержувача. Пакети даних передаються в зашифрованому вигляді через проксі, які не можуть бачити, що всередині, і завжди пропускають їх.

Як тільки об'єкт WebSocket створений, ми повинні слухати його події. Їх всього 4:

- *open* - з'єднання встановлено,
- *message* - отримані дані,
- *error* - помилка,
- *close* - з'єднання закрито.

9.4.3.2 Передача даних

А якщо ми хочемо відправити що-небудь, то виклик *socket.send(data)* зробить це. Потік даних в WebSocket складається з «фреймів» (RFC 6455),

фрагментів даних, які можуть бути відправлені будь-якою стороною, і які можуть бути наступних видів:

- «текстові кадри» - містять текстові дані, які сторони надсилають один одному;
- «бінарні фрейми» - містять бінарні дані, які сторони надсилають один одному;
- «пінг-понг фрейми» використовується для перевірки з'єднання; відправляються з сервера, клієнт (браузер) реагує на них автоматично;
- також є «фрейм закриття з'єднання» і деякі інші службові фрейми.

Фрейми можуть бути різної довжини і передаватися або цілком, або фрагментами (кілька фреймів). Вони можуть мати наступний вигляд:

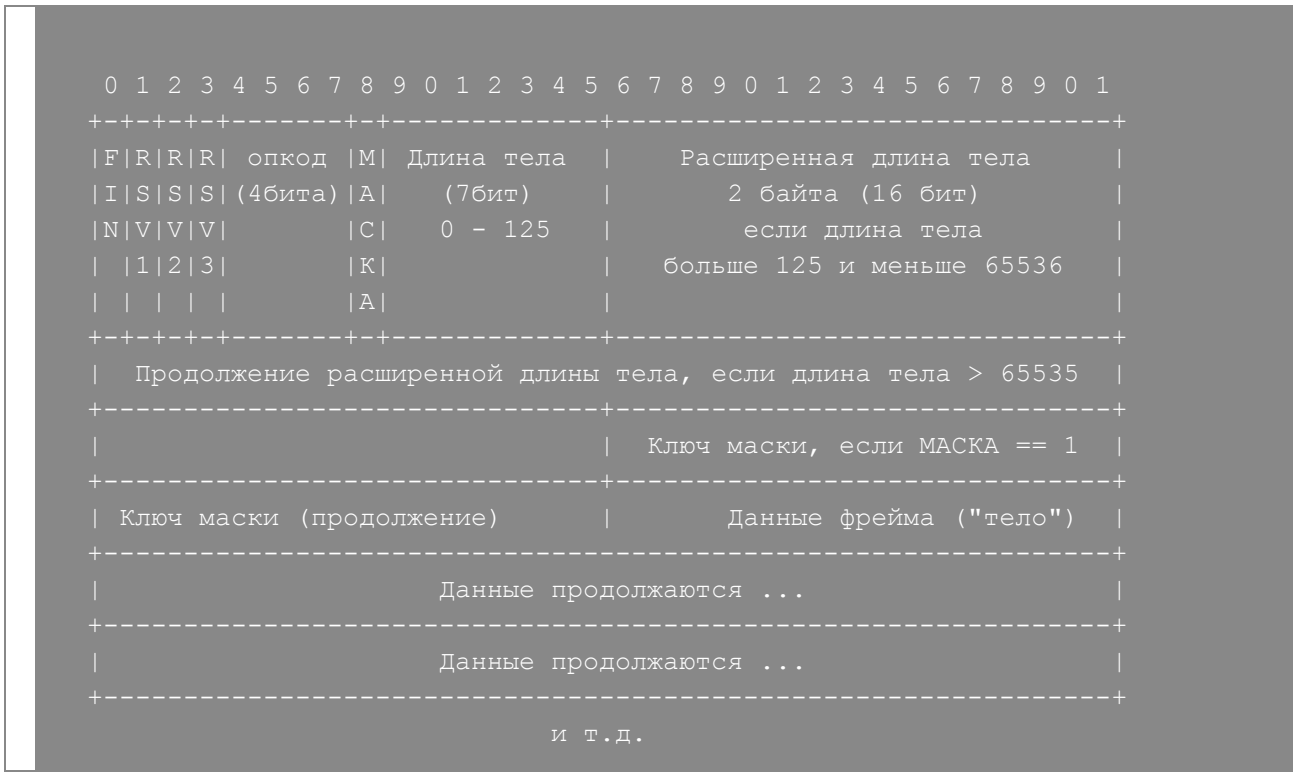


Рисунок 9.26. Приклад фреймів з даними у WEBSocket

Метод *WebSocket .send()* може відправляти і текстові і бінарні дані.

Виклик *socket.send (body)* приймає *body* у вигляді рядка або будь-якому бінарному форматі включаючи *Blob, ArrayBuffer* і інші. Додаткових налаштувань не потрібно, просто відправляємо в будь-якому форматі.

При отриманні даних, текст завжди надходить у вигляді рядка. А для бінарних даних ми можемо вибрати один з двох форматів: *Blob* або *ArrayBuffer*.

Це задається властивістю *socket.bufferType*, за замовчуванням воно дорівнює "*blob*", так що бінарні дані надходять у вигляді *Blob*-об'єктів. *Blob* - це високорівневий бінарний об'єкт, він безпосередньо інтегрується з *<a>*, *<img>* і іншими тегами HTML, так що це цілком зручне значення за замовчуванням. Але для обробки даних, якщо потрібно мати доступ до окремих байтів, ми можемо змінити його на *ArrayBuffer*:

```
socket.bufferType = "arraybuffer";
socket.onmessage = event;
// event.data є рядком (якщо текст) чи arraybuffer (якщо двійкові дані)
```

9.4.3.3 Врахування обмеження швидкості каналу

Уявимо, що наш додаток генерує багато даних для відправки. Але у користувача повільне з'єднання, можливо, він в інтернеті з мобільного телефону і не з міста. Ми можемо викликати *socket.send (data)* знову і знову. Але дані буде буферизованно в пам'яті і відправлятися лише з тією швидкістю, яку дозволяє мережа. За допомогою властивості *socket.bufferedAmount* можливо побачити кількість ще не відправлених байт буферизованих даних на поточний момент, які мають бути надіслані по мережі.

```
// кожні 100 мс перевіряти сокет і відправляти наступні дані лише тоді,
// коли всі поточні вже відіслані
setInterval(100) {
  if (socket.bufferedAmount == 0)
    socket.send(moreData());
  return;
  }
```

9.4.3.4 Прийом пакетів

Подія *onmessage* - це вухо клієнта до сторони сервера. Щоразу, коли сервер надсилає якісь дані, подія *onmessage* буде запущена. Повідомлення можуть містити звичайний текст, зображення або двійкові дані. Від вас залежить, як інтерпретувати та візуалізувати ці дані:

```
socket.onmessage = function (event) {
```

```
console.log("Дані отримано!");
}
```

Перевірка типів даних досить проста. Ось як ми можемо відобразити відповідь сервера рядком:

```
socket.onmessage = function (event) {
if (typeof event.data == "string") {
        // якщо сервер послав рядок, то відобразити його.
        label = document.getElementById("status-label");
        label.innerHTML = event.data;
        }
}
```

9.4.3.5 Закриття підключення

Зазвичай, коли сторона хоче закрити з'єднання (для цього клієнт і сервер мають рівні права), вони відправляють «фрейм закриття з'єднання» з кодом закриття і вказують причину у вигляді тексту. Метод для цього:

```
socket.close([code], [reason]);
```

тут *code* - спеціальний WebSocket-код закриття (не обов'язковий).

reason - рядок з описом причини закриття (не обов'язковий).

Потім протилежна сторона в обробнику події *onclose* отримає і код *code* і причину *reason*, наприклад:

```
// сторона, що закриває з'єднання
socket.close (1000, "робота закінчена");

// протилежна сторона:
socket.onclose = function (event) {
// event.code === 1000
// event.reason === "робота закінчена"
// event.wasClean === true (закрито чисто)
```

};

Слід пам'ятати, що *code* - це не будь-яке число, а спеціальний код закриття WebSocket. Найбільш поширені значення:

- 1000 - за замовчуванням, нормальне закриття,
- 1006 - неможливо встановити такий код вручну, вказує, що з'єднання було втрачено (немає фрейма закриття).

Є й інші коди:

- 1001 - сторона відключилася, наприклад сервер вимкнений або користувач покинув сторінку,
- 1009 - повідомлення занадто велике для обробки,
- 1011 - непередбачена помилка на сервері,
- …і т.ін.

Повний список кодів знаходиться в RFC 6455, §7.4.1.

9.4.3.6 Перевірка стану з'єднання

Щоб отримати стан з'єднання, існує додаткова властивість *socket.readyState* з наступними значеннями:

0 - «CONNECTING»: з'єднання ще не встановлено,

1 - «OPEN»: іде обмін даними,

2 - «CLOSING»: з'єднання закривається,

3 - «CLOSED»: з'єднання закрито.

9.5 Порівняння деяких існуючих протоколів обміну даними

Modbus. Відкритий комунікаційний протокол для обміну даними між мережевими пристроями, заснований на архітектурі *ведучий-ведений* (*master-slave*) Обмін даними являє собою транзакції, що складаються із запитів і відповідей. Перевагами протоколу Modbus є відкритість і масовість використання (безліч виробників випускають різні типи пристроїв, давачів, що підтримують даний протокол). Завданнями протоколу Modbus є контроль і управління на локальних мережах, читання і запис даних в регістри зберігання, доступ до файлів, діагностика стану локальних пристроїв.

MQTT. Мережевий протокол для обміну повідомленнями в мережах з низькою пропускною спроможністю між пристроями, що реалізують модель *master-slave*. Працює поверх TCP / IP, який забезпечує простий і надійний потік даних. Основною метою MQTT є віддалений моніторинг даних, що збираються з великої кількості пристроїв, і їх телеметрія в IT інфраструктурі. Оскільки дані надаються для IT-інфраструктури, вся система має можливість легко транспортувати дані в корпоративні технології, такі як ActiveMQ і ESBS. Протокол націлений на велику мережу невеликих пристроїв, які необхідно контролювати або управляти ними з хмари. Також він призначений для «багатоадресної передачі» даних для багатьох клієнтів. MQTT надзвичайно простий, пропонує декілька варіантів управління. Прикладом роботи є моніторинг нафтопроводу на наявність витоків або вандалізму. Це інформація від тисяч давачів, яка повинні бути сконцентрована в одному місці для аналізу. Коли система виявляє проблему, вона може вжити заходів, щоб виправити цю проблему.

CoAP (*Constrained Application Protocol*). Веб-протокол передачі даних для використання в обмежених вузлах і мережах Інтернету речей. На відміну від HTTP CoAP на транспортному рівні використовує протокол UDP, клієнт і сервер взаємодіють без встановлення з'єднання. Основними перевагами використання CoAP для IoT є простота, низькі витрати пам'яті і енергії живлення.

Websocket. За оцінками, Websocket забезпечує зниження затримок в три рази у порівнянні з напівдуплексним HTTP-опитуванням (полінгом). Websocket не призначений для пристроїв з обмеженими ресурсами, оскільки попередні протоколи та його клієнт-серверна архітектура не зовсім підходять для застосунків IoT. Однак він призначений для обміну в реальному часі, він безпечний (**wss**), він мінімізує поточні витрати і, використовуючи WAMP, може забезпечити ефективні системи обміну повідомленнями. Таким чином, він може конкурувати з будь-яким іншим протоколом, що працює через TCP.

Контрольні питання

1. Які функції виконують протоколи прикладного рівня передачі в IoT?

2. Поясніть терміни *публікатор, передплатник, брокер, топік* при обміні повідомленнями в IoT.
3. Коли і ким було створено протокол MQTT? Для яких цілей? Розкажіть історію стандартів для цього протоколу.
4. Дайте характеристику протоколу MQTT. Його можливості. Що таке корисне навантаження?
5. Навести модель і топологію публікації/підписки у протоколі MQTT. Розповісти логіку функціонування пристроїв з таким протоколом.
6. Які міри прийнято у протоколі MQTT для запобігання наслідків від передчасного розриву з'єднання?
7. Які міри прийнято у протоколі MQTT для зниження накладних витрат при перепідключенні?
8. Як у протоколі MQTT реалізовано використання QoS? Які рівні якості обслуговування існують, коли застосовуються?
9. Як впливає вибір рівня якості QoS на використання енергії батареї в давачах?
10. З яких частин складається структура пакету MQTT? Розповісти про призначення цих частин.
11. Навести структуру фіксованого заголовка пакета MQTT та розповісти про призначення полів у ньому.
12. Як саме кодується поточна довжина пакету MQTT у фіксованому заголовку пакета?
13. Призначення заголовку змінної довжини у пакеті MQTT. Які дані можуть розміщуватися в ньому?
14. Що таке дані корисного навантаження у пакеті MQTT? Звідки слід брати їх зміст та формат?
15. Розповісти про різновиди комунікаційних повідомлень в MQTT. Надати їх коротку характеристику.
16. Для чого використовуються символи шаблону при передплаті на повідомлення? Які шаблони ви знаєте?
17. Що таке WEB-сокет і коли він використовується?
18. Описати, як працює обмін повідомленями в IoT з використанням WEB-сокетів.

19. Як виконати відкриття WEB-сокета?
20. Як здійснюється передача даних з використанням WEB-сокетів? Як враховувати обмеження швидкості каналу зв'язку?
21. Як здійснюється прийом пакетів при використанні WEB-сокетів?
22. Як закрити підключення з використанням WEB-сокетів?
23. Як перевірити поточний стан з'єднання з використанням WEB-сокетів?

10 ХМАРНІ, ГРАНИЧНІ ТА ТУМАННІ ОБЧИСЛЕННЯ

10.1 Історія хмарних обчислень та їх необхідність для IIoT

Без хмарних технологій ринок IIoT-пристроїв не існував би. По суті, десятки тисяч кінцевих пристроїв, які були історично немислимі і не пов'язані одне з одним, повинні були б бути самокерованими, не маючи можливості обмінюватися даними або збирати дані. Тисячі невеликих вбудованих систем самі по собі не додають цінності для клієнтів. Значення IIoT - в даних, які він виробляє, не в одній кінцевій точці, але в тисячах і мільйонах кінцевих точок. Хмара (рис. 10.1) забезпечує можливість мати прості давачі, камери, перемикачі, маяки і виконавчі механізми, які розмовляють спільною мовою один з одним. Хмара - спільний знаменник для "валюти даних".

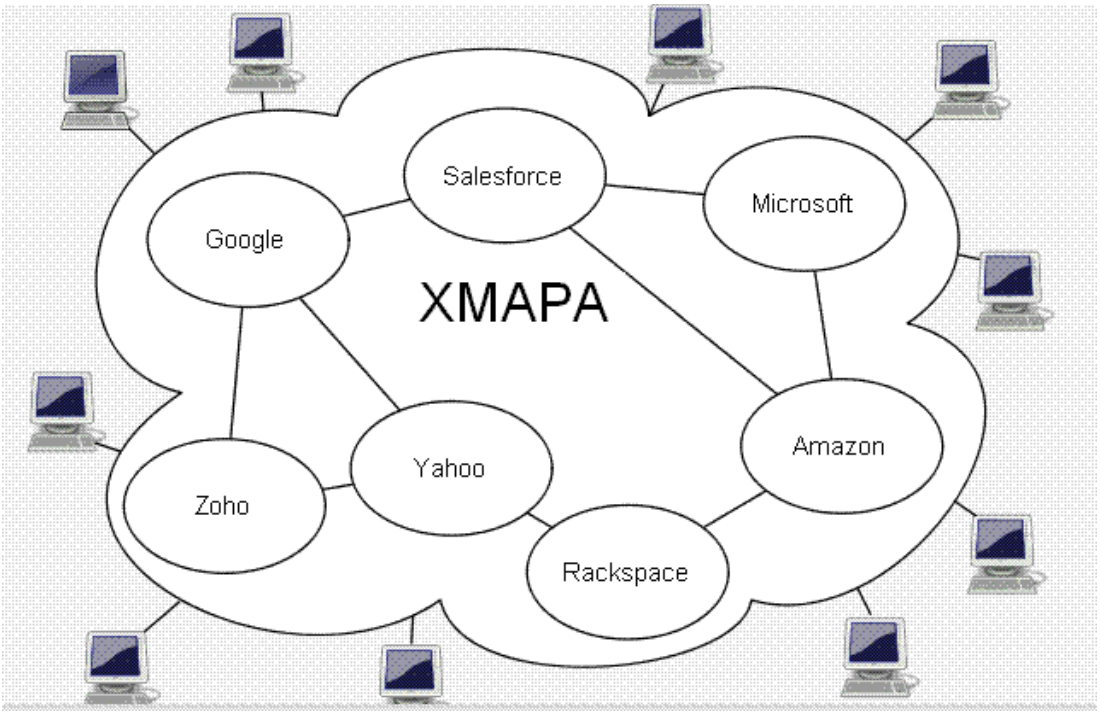


Рисунок 10.1 Діаграма, що дає уяву про хмару

Поняття *хмара* відноситься до інфраструктури обчислювальних служб, які зазвичай необхідні відповідно до застосування. Набір ресурсів (обчислень, мереж, сховищ і пов'язаних з ними програмних сервісів) може динамічно масштабуватися в бік збільшення або зменшення в залежності від середнього навантаження і якості обслуговування. Хмари, як правило, являють собою великі *центри обробки даних (ЦОД)*, які надають клієнтам послуги, орієнтовані на зовнішнього споживача, і пропонують відповідні моделі оплати за їх

використання. Ці ЦОД створюють ілюзію єдиного хмарного ресурсу, в той час як насправді може бути використано багато географічно розподілених ресурсів. Це дає користувачеві відчуття незалежності від місця розташування. Ресурси є *еластичними* (що означає потрібну ступінь масштабованості), а пропоновані на них сервіси створюють постійний дохід для провайдера хмарного ресурсу. Сервіси, які працюють у хмарі, відрізняються від традиційного програмного забезпечення своєю архітектурою та реалізацією. Хмарні застосування можуть розроблятися і розвиватися швидше, і в меншій мірі залежати від мінливості середовища у клієнта. Таким чином, розгортання хмар відбувається дуже швидко.

Концепція хмарних обчислень з'явилася ще в 1960 році, коли американський учений, фахівець з теорії ЕОМ Джон Маккарті (John McCarthy) висловив припущення, що коли-небудь комп'ютерні обчислення стануть надаватися подібно комунальним послугам (public utility).

Розповсюдження мереж з високою потужністю, низька вартість комп'ютерів і пристроїв зберігання даних, а також широке впровадження віртуалізації, сервіс-орієнтованої архітектури привели до величезного зростання хмарних обчислень. Кінцеві користувачі можуть не перейматися роботою обладнання технологічної інфраструктури «в хмарі», яка їх підтримує. Аналогією обчислювальних «хмар» зі звичного життя можуть служити електростанції. Хоча домовласник може купити електрогенератор і піклуватися про його справність самостійно, більшість людей воліє отримувати енергію від централізованих постачальників.

Майже всі сучасні характеристики хмарних обчислень, порівняння їх з електроенергетикою та використання приватних, публічних та громадських моделей були представлені Дугласом Паркгілом (Douglas Parkhill) в книзі «The Challenge of the Computer Utility» (Виклик комп'ютерних послуг), в 1966 році. Згідно інших джерел, хмарні обчислення беруть початок з 1950-х років, коли вчений Герб Ґрош (Herb Grosch) стверджував, що весь світ буде працювати на терміналах, якими керують близько 15 великих ЦОД.

Сам термін «хмара» походить з телефонії, тому що телекомунікаційні компанії, які до 1990-х років пропонували в основному виділені лінії передачі

«точка-точка», почали пропонувати віртуальні приватні мережі (VPN), з порівняною якістю обслуговування, але при набагато менших витратах. Перемикаючи трафік для оптимального використання каналів вони мали змогу ефективніше використовувати мережу. Символ хмари був використаний для позначення розмежування між користувачем і постачальником.

Вперше поняття сьогоднішньої хмари виникло в компанії *Compaq* в середині 1990-х років. В ньому технологічні футуристи передбачали обчислювальну модель, яка переводила обчислення з окремих комп'ютерів в мережу і на хости. По суті, це було основою хмарних обчислень, але тільки з появою деяких інших технологій хмарні обчислення стали економічно вигідними для галузі. Телекомунікаційна індустрія з виділеними каналами зв'язку традиційно була побудована на двоточковій системі зв'язку. Створення віртуальних приватних мереж дозволило забезпечити безпечний і контрольований доступ до кластерів і створити приватні та публічні хмарні гібриди.

Першим же кроком до втілення хмарних обчислень можна вважати появу ASP (*Application service provider - провайдери послуг доступу до програмних застосувань*) у другій половині 1990х років. ASP можна вважати одними із перших SaaS сервісів. Пальма першості належить сервісу електроної пошти від компанії *Hotmail*. Але відстуність на той час широкосмугових каналів інтернет та технологій віртуалізації стали на перепоні: за відсутності швидких та стабільних каналів інтернет-користувачі не могли отримати якісні послуги, а без технологій віртуалізації неможливо було ефективно та гнучно розподіляти ресурси та масштабувати сервіси. Також слід зазначити що лавиноподібний ріст користувачів інтернет, що сформували попит на послуги SaaS, відбувся лише у 2000-х роках, тому можна лише на пальцях рук порахувати ASP провайдерів, що дожили до наших днів, серед них найбільш відомий - Salesforce.

Ключову роль в розвитку хмарних обчислень зіграв *Amazon*, модернізувавши свої ЦОД, які, як і більшість комп'ютерних мереж, одночасно використовують лише 10% своєї потужності, заради забезпечення надійності при стрибках навантаження. Дізнавшись, що нова хмарна архітектура забезпечує значне внутрішнє підвищення ефективності, Amazon почав нові дослідження в

галузі розвитку продуктів для забезпечення хмарних обчислень для зовнішніх клієнтів, і запустив *Amazon Web Service (AWS)* на основі розподілених обчислень в 2006 році.

10.2 Моделі хмарних сервісів

Хмарні провайдери зазвичай підтримують цілий ряд продуктів «Все як сервіс» (*Everything as a Service - XaaS*). Тобто, послуга програмного забезпечення з оплатою за використання. Сервіс включає до себе **службу мережі** (*Networking as a Service - NaaS*), **інфраструктуру як послугу** (*Infrastructure as a Service - IaaS*), **платформу як послугу** (*Platform as a Service - PaaS*) і **програмне забезпечення як послугу** (*Software as a Service - SaaS*). Кожна модель надає все більше і більше хмарних сервісів від провайдера[5][6][8].

Проілюструємо ці моделі на прикладі приготування новорічного салату олів'є. Що потрібно, щоб приготувати і подати новорічний олів'є? Ось попередній план:

- місце збіговиська;
- інгредієнти: докторська ковбаса, солоні огірки, зелений горошок, варені картоплю, моркву і яйця, майонез, сіль (можливо, авторські секретні інгредієнти);
- змішати інгредієнти в потрібній пропорції;
- додати майонез;
- сервірувати стіл;
- подати шампанське і мандарини;
- розкласти олів'є по тарілках.

У моделі NaaS ми отримуємо тільки місце, де пройде захід та оборудки для нього (столи та посуд). Усім іншим будемо займатися самі.

У моделі IaaS ми вже отримуємо готові інгредієнти (так, якщо б ми купили їх в магазині): зварені і нарізані. Нам залишається змішати їх в потрібній пропорції, додати майонез, сервірувати стіл і подати готовий салат з шампанським і мандаринами.

У моделі PaaS, можна сказати, що ми купуємо вже готовий олів'є у відділі кулінарії. Нам залишається лише підготувати стіл з шампанським та мандаринами і подати готовий салат.

Модель SaaS в нашій аналогії – це вже ресторан. Ми нічого не готуємо, а тільки споживаємо. Офіціант запрошує нас за красиво сервірований стіл, де нам залишається лише насолоджуватися шампанським, мандаринами і бажаним новорічним салатом від шеф-кухаря.

Повертаючись від смачного до земного ці чотири моделі можна зобразити на одному рис. 10.2 нижче:

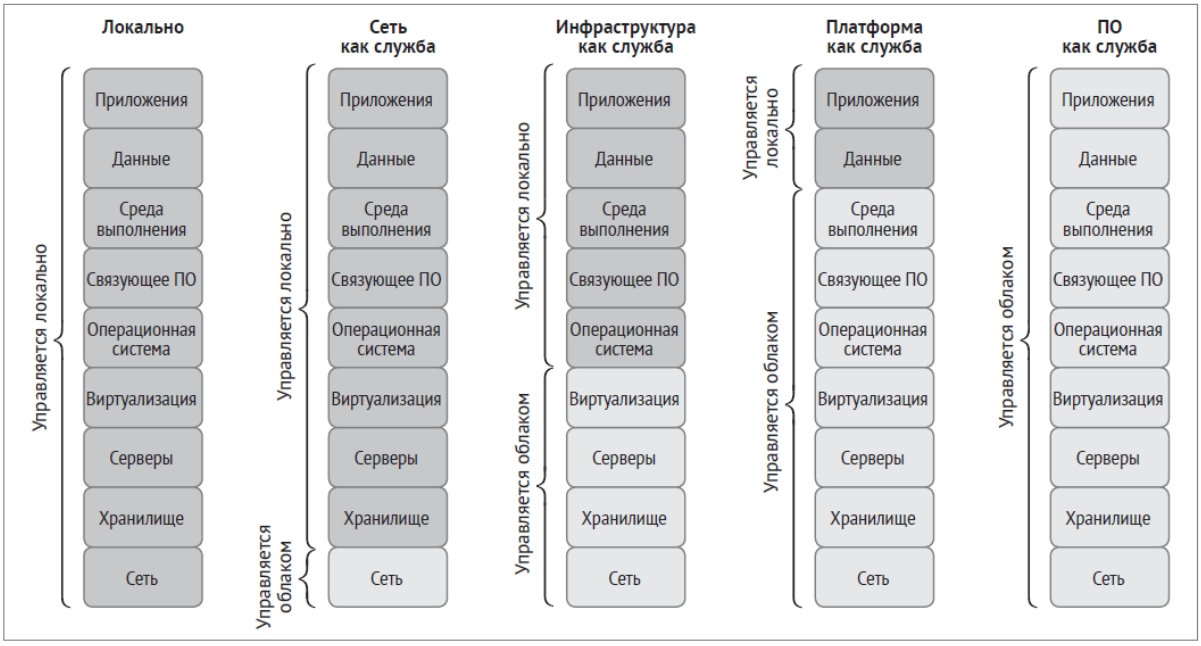


Рисунок 10.2 Моделі хмарних архітектур

Для NaaS характерні такі сервіси, як **мережева взаємодія**, **ПЗ, що визначає мережу** (*Software-Defined Networking - SDN*) і **периметр, що визначається мережевим ПЗ** (*Software-Defined Perimeters - SDP*). Ці продукти керуються вже не користувачем, а хмарою і організованими механізмами для забезпечення роботи та безпеки мереж підприємства користувача. Замість того, щоб створювати самому глобальну інфраструктуру і виділяти капітал для підтримки корпоративних комунікацій, при створенні віртуальної мережі може використовуватися хмарний підхід. Це дозволяє мережі оптимально масштабувати ресурси в бік збільшення або зменшення в залежності від потреб, а нові мережеві якості можуть бути при потребі додатково придбані і швидко розгорнуті.

IaaS була початкової концепцією хмарних сервісів. У цій моделі провайдер створює масштабовані апаратні служби в хмарі і надає модифікацію програмних фреймворків для створення клієнтських віртуальних машин. Це забезпечує максимальну гнучкість при розгортанні, але вимагає більших зусиль з боку клієнта.

PaaS використовує базове устаткування і програмні засоби нижнього рівня, що надаються хмарою. В такому випадку кінцевий користувач просто використовує апаратне забезпечення ЦОД, операційну систему, проміжне ПЗ і різні бази даних провайдера для розміщення свого приватного програмного застосунку або сервісів. Проміжне ПЗ може складатися з систем баз даних. При побудові багатьох галузей промисловості було використано обладнання хмарних постачальників, наприклад для *Swedbank, Trek Bicycles* і *Toshiba*. Прикладами публічних постачальників PaaS є *IBM Watson, Google App Engine* і *Microsoft Azure*. Різниця між PaaS і IaaS полягає в тому, що ви не тільки отримуєте переваги масштабованості і зменшуєте операційні витрати з хмарною інфраструктурою, але у вас також є перевірене проміжне ПЗ і операційні системи від провайдера. Це такі системи, як *Docker*, де програмне забезпечення розгортається в контейнерах. Якщо ваш програмний додаток розгортається в межах границь, що надаються постачальником інфраструктури, ви можете очікувати більш швидкий вихід на ринок, оскільки більшість компонентів, ОС і проміжного програмного забезпечення гарантовано будуть доступні.

SaaS є основою хмарних обчислень. У провайдера зазвичай є пропоновані програмні додатки або послуги, які пропонуються кінцевим користувачам за допомогою таких клієнтів, як мобільні пристрої, тонкі клієнти або фреймворки в інших хмарах. З точки зору користувача, віртуальний SaaS фактично працює на клієнті користувача. Ця абстракція програмного забезпечення дозволила галузі домогтися значного зростання в хмарному сервісі. Сервіси SaaS працюють для таких прикладних застосунків, як *Google Apps, Salesforce* і *Microsoft Office 365*.

10.3 Моделі розгортання хмарного середовища

Існують три різні **моделі розгортання** хмар, які зазвичай і використовуються: *приватна хмара, хмара загального користування* та *гібридна*

хмара. Незалежно від моделі, фреймворки хмар повинні забезпечувати динамічну масштабованість, швидкість розробки і розгортання, а також роботу у локальному місці незалежно від його віддаленості.

Приватні хмари також мають на увазі керовані компоненти за запитом. Сучасні корпоративні системи, як правило, використовують гібридну архітектуру для забезпечення безпеки критично важливих застосунків у приватній частині і використовують публічну частину для підключення, простоти і швидкості розгортання.

10.3.1 Приватна хмара

Рисунок 10.3 Приклад приватної хмари на колокейшн у ЦОД

Приватна хмара (англ. *private cloud*) - це хмарна інфраструктура (рис. 10.3), яка призначена для використання виключно однією організацією, що включає декілька користувачів (наприклад, підрозділів). Приватна хмара може перебувати у власності, керуванні та експлуатації як самої організації, так і третьої сторони (чи деякої їх комбінації), наприклад колокейшн у ЦОД. Така хмара може фізично знаходитись як в, так і поза юрисдикцією власника.

Тут немає концепції спільного використання ресурсів або об'єднання за межами власної інфраструктури власника. У приміщеннях власника спільне використання та розпорядження ресурсами є загальними. Приватне хмара існує по ряду причин, включаючи безпеку та впевненість в якості. Тобто, приватна хмара дає гарантію, що інформація обробляється виключно системами, керованими клієнтом. Однак, щоб вважатися хмарою, повинні існувати деякі

аспекти хмарних сервісів, такі як віртуалізація і балансування навантаження. Приватна хмара може бути локальною або може бути спеціалізованою в обладнанні, що надається третьою стороною для використання.

10.3.2 Публічна хмара

Публічна хмара (англ. *public cloud*) - це хмарна інфраструктура, яка призначена для вільного використання широким загалом. Публічна хмара може перебувати у власності, керуванні та експлуатації комерційних, академічних (освітніх та наукових) або державних організацій (чи будь-якої їх комбінації). Публічна хмара перебуває в юрисдикції постачальника хмарних послуг.

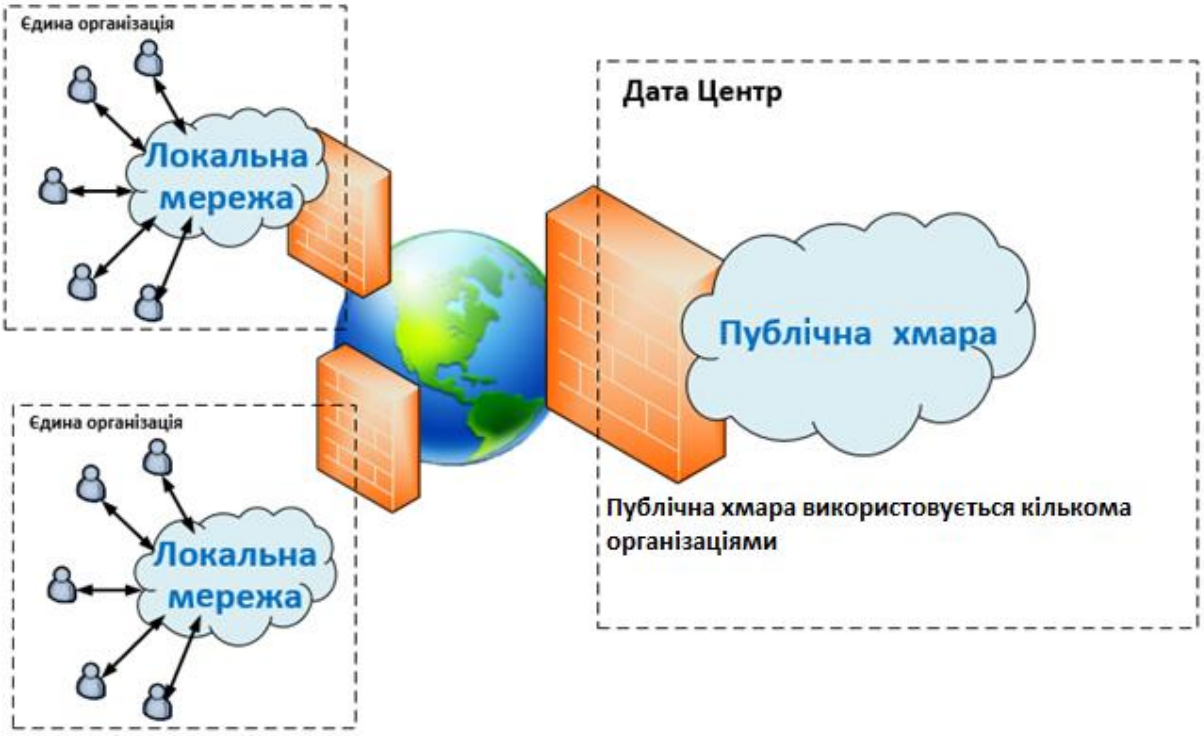


Рисунок 10.4 Приклад публічної хмари

Тут інфраструктура надається на вимогу для безлічі клієнтів і застосувань. Інфраструктура являє собою набір ресурсів, які будь-яка людина може використовувати в будь-який час в рамках своїх угод про рівень обслуговування. Перевага тут у тому, що хмарні ЦОД дозволяють забезпечити безпрецедентну масштабованість для багатьох клієнтів, які обмежені тільки своїми фінансовими можливостями.

10.3.3 Гібридна хмара

Гібридна хмара (англ. *hybrid cloud*) - це хмарна інфраструктура, що складається з двох або більше різних хмарних інфраструктур (приватних,

громадських або публічних), які залишаються унікальними сутностями, але з'єднанні між собою стандартизованими або приватними технологіями, що уможливлюють переносимість даних та прикладних програм (наприклад, використання ресурсів публічної хмари для балансування навантаження між хмарами). Приклад гібридної хмари наведено на рис. 10.5 нижче.

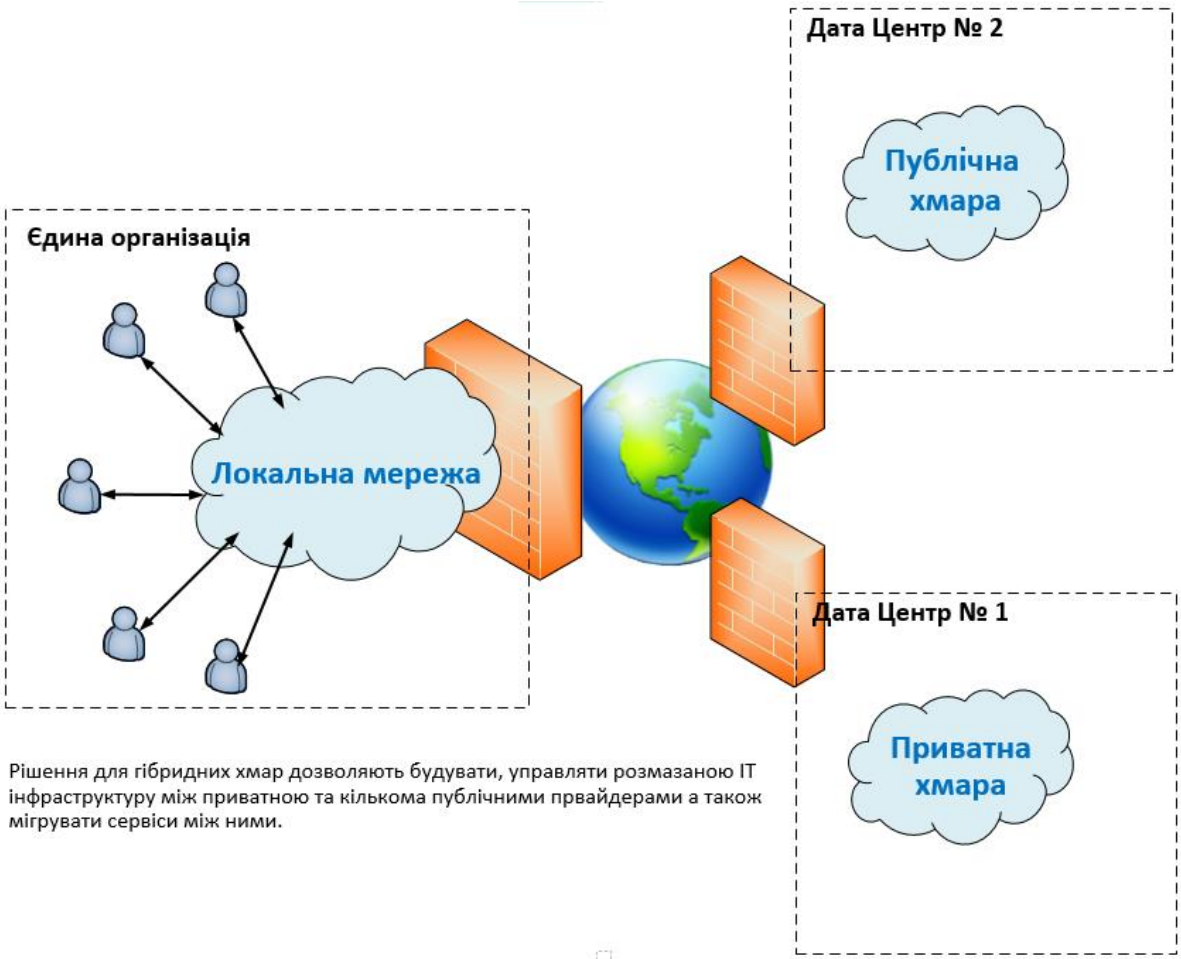


Рисунок 10.5 Приклад гібридної хмари

Організації вважають за краще вибирати гібридну модель, якщо є дані, які потребують унікального підходу, а інтерфейс дозволяє використовувати хмару. Іншим варіантом використання є необхідність використання додаткових хмарних областей для компенсації умов, коли кількість накопичених даних переповнює хмару приватної корпорації в цілому. В цьому випадку публічна хмара буде використовуватися як балансувальник навантаження до тих пір, поки накопичення даних і їх використання не повернуться в обмежений простір приватної хмари. Цей варіант використання називається *хмарним вибухом* і відноситься до використання хмар в якості умовних ресурсів.

10.4 Публічний хмарний сервіс IBM Cloud (Watson)

10.4.1 Що таке IBM Cloud

Хмарна платформа *IBM Cloud* об'єднує платформу як послугу (PaaS) з інфраструктурою як послуга (IaaS) для забезпечення використання користувачем. Платформа масштабує та підтримує як невеликі команди та організації, так і великі підприємства. Платформа вже розгорнена глобально через ЦОД, розміщених по всьому світу. Платформа *IBM Cloud* складається з декількох компонентів, які працюють разом, щоб забезпечити користувачеві послідовну та надійну роботу з хмарами:

- каталог, що складається з сотень пропозицій IBM Cloud;
- надійна консоль, яка служить користувачеві вікном для створення, перегляду, керування його хмарними ресурсами;
- компонент управління ідентифікацією та доступом, який надійно аутентифікує користувачів для обох служб платформи та керує послідовним доступом до ресурсів IBM Cloud;
- механізм пошуку та маркування для фільтрації та ідентифікації ресурсів користувача;
- система керування обліковим записом і платіжною системою, яка забезпечує використання планів ціноутворення та захист від шахрайства з кредитними картками.

В *IBM Cloud* більше не потрібно робити великі інвестиції в апаратне забезпечення, щоб перевірити чи запустити нову програму. Замість цього сама хмара керує всім для користувача, а оплата стягується лише за те, що використовується. Хмарне серверне середовище є базою рівня інфраструктури користувача. Користувач може вибрати один варіант або їх комбінацію для більш складних середовищ.

Користувач може використовувати різні варіанти розміщення власних додатків, що надає йому стільки контролю над інфраструктурою, скільки він забажає, або скільки йому буде потрібно. Програмний додаток можна запускати будь-яким із наведених нижче способів:

- як функцію без сервера;

- як додаток Cloud Foundry;
- як контейнер Docker на кластері Kubernetes;
- як віртуальну машину;
- на високопродуктивних серверах BMS.

Bare Metal Servers (BMS) є фізичними серверами, кожен з яких призначено лише для окремого клієнта і його не розділено з іншими. Користувач керує майже всім, від хоста-сервера до оперативної пам'яті і пристроїв зберігання даних. Ці сервери використовуються з робочими навантаженнями, які вимагають обчислювальної потужності протягом тривалого часу, наприклад, на протязі кількох місяців.

10.4.2 Деякі сервіси IBM Cloud, які призначені для використання в автоматизації

Internet of Things Platform

Сервіс дозволяє поєднати світ давачів, актуаторів і контролерів з великими даними і аналітикою. У поєднанні з платформою IBM Cloud, сервіс *IoT Platform* надає простий, але потужний масштабований доступ до пристроїв і їх даних. Розробник має можливість швидко створювати програми, панелі візуалізації, програми для мобільних пристроїв та програми, які можуть постачати окремі департаменти підприємства даними IoT.

IoT Platform дозволяє розробникам:

- безпечно підключати давачі та актуатори до відповідних програм моніторінгу і керування;
- збирати та керувати переглядом часових даних;
- організувати візуальний збір потоків даних для обробки за допомогою Node-Red та Watson;
- керувати з'єднаннями та підписками за допомогою високомасштабованого сервісу.

Node-RED Starter

Інструмент для візуального програмування потоком даних, розроблений працівниками компанії IBM для поєднання різноманітних пристроїв, API та онлайн-сервісів як складових частин Інтернету речей.

Node-RED дає змогу працювати з браузерним редактором потоків даних як окремими вузлами з різним функціоналом, що уможливлює створення JavaScript-функцій. Причому можна використовувати як базові вузли, якими одразу забезпечений *Node-RED*, так і встановлювати вузли з додатковим функціоналом з репозиторію NPM або ж навіть створити свій власний вузол з унікальним функціоналом. Програми, або ж їхні частини, розроблені за допомогою *Node-RED*, можуть бути збережені та поширені для вільного використання. Саме середовище побудовано на основі *Node.js*. Потоки, створені за допомогою *Node-RED*, зберігаються у вигляді JSON. Починаючи з MQTT версії 0.14, вузли можна налаштовувати для TLS-з'єднання.

10.4.3 IBM Cloud архітектура (публічна хмара)

IBM Cloude - це середовище для розробки додатків і використання служб, які надають користувачу готові до використання функції (рис. 10.6). Компанія IBM також надає середовище для розміщення артефактів застосувань (додатків), які виконуються на сервері додатків, наприклад, профіль *IBM WebSphere Application Server Liberty*.

Використовуючи глобальну інфраструктуру *IBM SoftLayer*, *IBM Cloude* розгортає віртуальні контейнери, в яких розміщують кожну розгорнуту програму. У цьому середовищі програма може використовувати готові послуги (включаючи служби третіх осіб), щоб полегшити збірку програми.

Розробник може взаємодіяти з інфраструктурою *Watson* за допомогою інтерфейсу на основі браузера. Можна також використовувати інтерфейс командного рядка *Cloud Foundry*, званий **cf**, для розгортання веб-додатків.

Клієнтами можуть бути мобільні програми, зовнішні програми, програми, створені на IBM Cloud, або розробники, які використовують веб-браузер. Всі вони взаємодіють з додатками, розміщеними на IBM Cloud. Клієнти

використовують REST або HTTP API для маршрутизації запитів через IBM Cloud до одного з екземплярів програми або до композитної послуги.

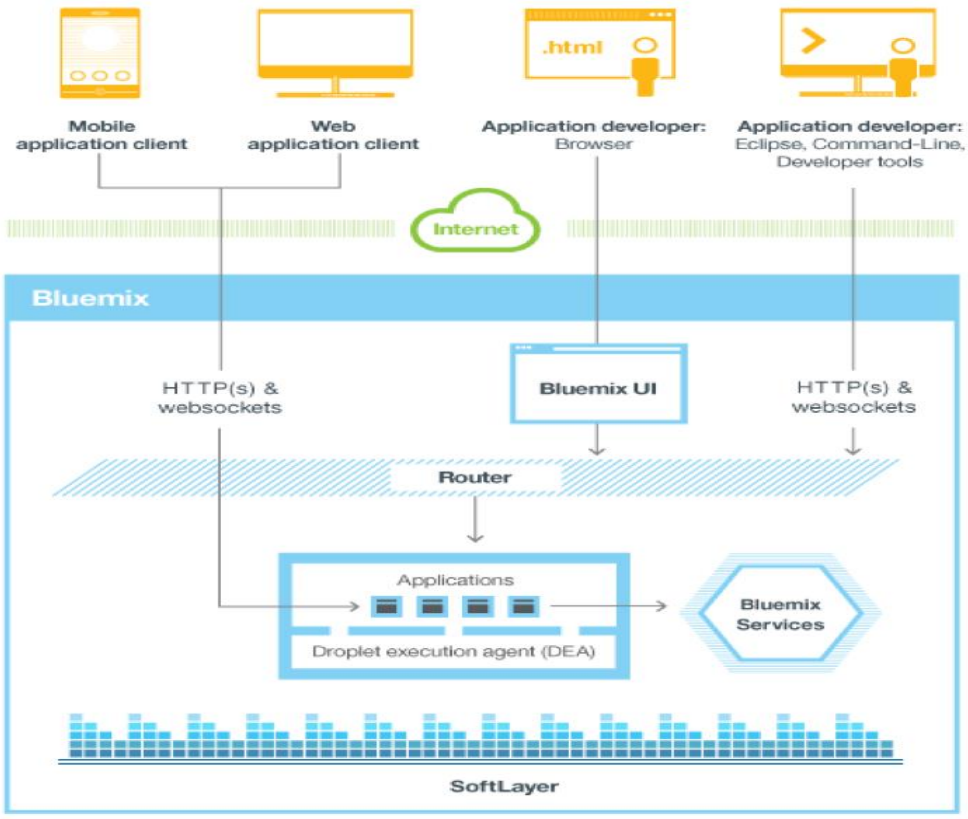


Рисунок 10.6 IBM Cloud архітектура

10.4.4 Безпека в IBM Cloud

Платформа IBM Cloud має багаторівневий контроль безпеки як мережі, так і інфраструктури. Компанія IBM також надає набір служб безпеки, які можуть використовуватися розробниками програм для забезпечення мобільних і веб-додатків. Ці елементи об'єднуються, щоб зробити IBM Cloud платформою для безпечної розробки програмних додатків.

IBM Cloud забезпечує готовність безпеки, дотримуючись політик безпеки. Ці політики включають такі практики, як сканування вихідного коду, динамічне сканування, моделювання загроз і тестування на проникнення. IBM Cloud слідкує за процесом реагування на інциденти безпеки продуктів IBM (PSIRT) для керування цими інцидентами.

IBM Cloud використовує хмарну послугу IBM SoftLayer та інфраструктуру як послугу (IaaS) і повністю використовує архітектуру безпеки. SoftLayer IaaS надає додаткам користувача кілька рівнів захисту, що перекривають один

одного. IBM Cloud надає можливості захисту на рівні PaaS в різних категоріях: платформа, дані та програмні додатки.

10.4.4.1 Безпека на рівні платформи

Хмара забезпечує функціональну, інфраструктурну, операційну та фізичну безпеку (через IBM SoftLayer) для основної платформи.

Середовище IBM Cloud на SoftLayer відповідає стандартам безпеки інформаційних технологій IBM (IT), включаючи такі, які відповідають або навіть і перевищують галузеві стандарти:

- мережа, шифрування даних і контроль доступу;
- списки керування доступом до додатків (ACL), дозволи та тестування на проникнення;
- ідентифікація, аутентифікація та авторизація;
- захист інформації та даних;
- цілісність та доступність послуг;
- уразливість і управління виправленнями;
- відмова в обслуговуванні та виявлення систематичних атак;
- відповідь на інциденти безпеки.

10.4.4.2 Функціональна безпека

IBM Cloud надає різні функціональні можливості безпеки, включаючи аутентифікацію користувачів, авторизацію доступу, аудит критичних операцій і захист даних.

Аутентифікація (користувача)

Розробники програмних додатків аутентифікуються у хмарі за допомогою веб-ідентифікації IBM. Для гібридної хмари, аутентифікація через *LDAP* підтримується за замовчуванням. За бажанням може бути встановлена аутентифікація за допомогою веб-ідентифікації IBM.

Авторизація (надання різноманітних прав)

Хмара використовує механізми *Cloud Foundry* для забезпечення того, щоб кожен розробник програми мав доступ лише до створених ним додатків і служб. Авторизація для служб хмари базується на *OAuth*. Доступ до всіх внутрішніх кінцевих точок платформи IBM Cloud обмежений зовнішніми користувачами.

Аудит

Журнали аудиту створюються для всіх успішних і невдалих спроб доступу розробників програмних додатків. Журнали аудиту також створюються для привілейованого доступу до систем *Linux*, які розміщують контейнери, звідки запускаються програми IBM Cloud.

Захист даних

Весь трафік хмари проходить через пристрої *IBM WebSphere DataPower SOA*, які забезпечують функції зворотного проксі, припинення SSL та балансування навантаження. Дозволені наступні методи HTTP: DELETE, GET, HEAD, OPTIONS, POST, PUT, TRACE. Якщо процес HTTP неактивний, він "засипає" через дві хвилини.

Забезпечення практики розвитку

Періодичні сканування вразливостей безпеки виконуються на різних компонентах хмари за допомогою динамічного аналізатора *IBM Security AppScan* і пропозицій статичного аналізатора. Моделювання загроз і тестування проникнення виконуються для виявлення та усунення будь-яких потенційних уразливостей. Крім того, розробники програм можуть використовувати службу *AppScan Dynamic Analyzer* для захисту своїх веб-застосувань, які розгортаються у хмарі.

10.4.4.3 Безпека інфраструктури

В архітектурі передбачено декілька компонентів для безпеки та ізоляції інфраструктури:

Сегрегація середовища

Зміни, доповнення і саме середовище IBM Cloud відокремлені один від одного для поліпшення стабільності та безпеки прикладних програм.

Брандмауери

У брандмауерів є обмеження доступу до хмарної мережі.

Захист від вторгнень

Хмара забезпечує захист від вторгнення для виявлення загроз, щоб їх можна було вирішити. Політики захисту від вторгнення включені на брандмауерах.

Безпечне управління контейнерами додатків

Кожен програмний застосунок в хмарі є ізольованим і виконується у власному контейнері, який має певні обмеження ресурсів для процесора, пам'яті та диска.

Забезпечення безпеки операційної системи

Адміністратори IBM регулярно виконують посилення захисту мережі та операційної системи за допомогою таких інструментів, як *IBM Endpoint Manager*. Крім того, для забезпечення цілісності та доступності впроваджуються процедури керування змінами та резервного копіювання та відновлення.

10.4.4.4 Операційна безпека

Хмара забезпечує надійне робоче середовище безпеки з такими елементами керування:

Сканування вразливості

Використовується інструменти сканування вразливостей *Tenable Network Security*, *Nessus*, для виявлення будь-яких проблем з мережевими та хост-конфігураціями, для своєчасного вирішиння проблеми.

Автоматизоване керування виправленнями

Адміністратори гарантують, що виправлення для операційних систем застосовуються з відповідною частотою. Автоматизовані виправлення включені за допомогою *IBM Endpoint Manager*.

Консолідація та аналіз журналу аудиту

Використовуються інструменти *IBM QRadar Security Intelligence* для консолідації журналів *Linux* для відстеження привілейованого доступу до систем *Linux*. Компанія також використовує систему безпеки IBM QRadar та управління подіями (SIEM) для моніторингу успішних і невдалих спроб входу розробників програмних застосунків.

Керування доступом користувачів

У рамках IBM Cloud дотримуються керівних принципів «Розділення обов'язків», щоб призначити користувачам гранульовані привілеї доступу та забезпечити їм тільки доступ, необхідний для виконання своїх завдань відповідно до принципу найменшого привілею. У виділеному середовищі хмари призначені адміністратори можуть керувати ролями та дозволами для користувачів своєї організації за допомогою консолі адміністратора.

10.5 Обмеження хмарних архітектур для IIoT

Провайдер хмарних сервісів знаходиться за межами граничного пристрою IIoT і керує глобальною мережею так, як це показано на рис. 10.7 нижче. Реакція в реальному часі має вирішальне значення в багатьох застосуваннях IIoT і змушує пересувати обробку ближче до кінцевого пристрою.

У міру наближення до давача ми входимо в область, де діють жорсткі вимоги виконання в реальному часі. Ці системи, як правило, являють собою вбудовані системи або мікроконтролери з затримкою, призначені для реального часу. Наприклад, відеокамера чутлива до частоти кадрів (як правило, 30 або 60 кадрів в секунду) і повинна виконувати ряд послідовних завдань в конвеєрі потоку даних (позбавлення від мозаїки, позначення, баланс білого і гамма-регулювання, відображення гами, масштабування і стиснення). Швидкість

даних, що проходять через конвеєр відеозображення (відео 1080p з використанням 8 біт на канал зі швидкістю 60 кадрів в секунду) складає приблизно 1,5 ГБ/с. Кожен кадр повинен проходити через цей конвеєр в режимі реального часу, тому більшість процесорів сигналів відеозображення використовують для цих перетворень контролери.

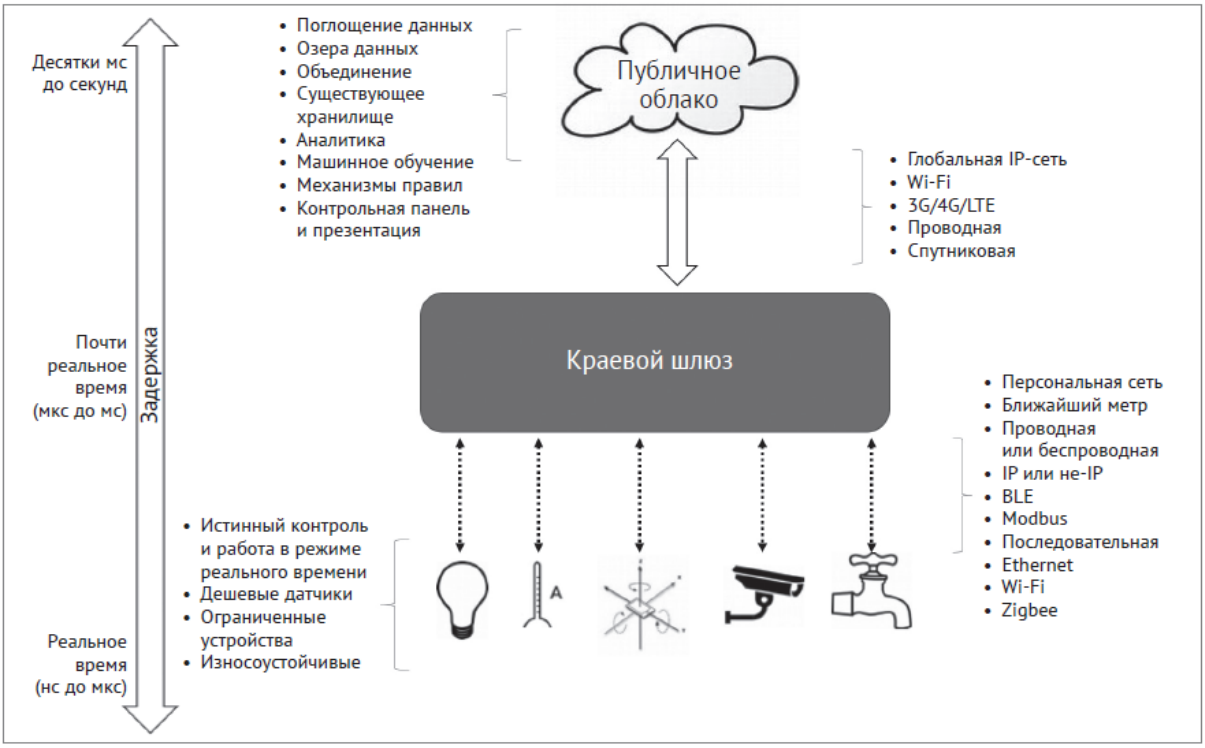


Рисунок 10.7 Ефекти затримок у хмарі

Якщо ми перемістимося вгору по стеку, шлюз буде мати кращий час відгуку і зазвичай реагує за одиниці мілісекунд. Коефіцієнт стробування в часі відгуку - це латентність WPAN (затримка між сигналом і реакцією на нього) і навантаження на шлюз. Як згадувалося раніше в темі про WPAN, більшість WPAN, таких як BLE, не мають постійної структури і залежать від кількості пристроїв BLE під шлюзом, інтервалів сканування, інтервалів оголошень, тощо. Інтервали з'єднання BLE можуть досягати 7,5 мс, але можуть варіюватися в залежності від того, як клієнт налаштовує інтервали оголошень, щоб звести до мінімуму споживання енергії. Сигнали Wi-Fi зазвичай мають затримку 1,5 мс. Затримка такого рівня вимагає фізичного інтерфейсу з PAN. Не можна очікувати, що при передачі необроблених пакетів BLE в хмару, буде швидкодія майже в реальному часі.

Компонент обробки в хмарі додасть ще одну ступінь затримки в WAN. Маршрут між шлюзом і провайдером хмари може пролягати декілька шляхами в

залежності від розташування ЦОД і шлюза. Хмарні провайдери зазвичай надають набір регіональних центрів обробки даних для нормалізації трафіку. Для того, щоб уявити істинний вплив провайдера хмари на латентність, необхідно відстежувати затримку пінга на протязі тижнів, або місяців і по регіонах.

Таблиця 10.1 Затримка пінга по регіонах (для AWS)

Регион	Задержка
US East (Virginia)	80 мс
US East (Ohio)	87 мс
US West (California)	48 мс
US West (Oregon)	39 мс
Canada (Central)	75 мс
Europe (Ireland)	147 мс
Europe (London)	140 мс
Europe (Frankfurt)	152 мс
Asia Pacific (Mumbai)	307 мс
Asia Pacific (Seoul)	192 мс
Asia Pacific (Singapore)	306 мс
Asia Pacific (Sydney)	205 мс
Asia Pacific (Tokyo)	149 мс
South America (São Paulo)	334 мс
China (Beijing)	436 мс
GovCloud (US)	39 мс

Вичерпний аналіз часу затримки можливо отримати при використанні сервісу *CL Audit*. Існують і інші інструменти для аналізу латентності у багатьох провайдерів хмарного сервісу.

Як правило, затримки у хмарах будуть складати десятки, якщо не сотні мілісекунд, без урахування накладних витрат на обробку даних. Це повинно бути прийняте до уваги при створенні хмарної архітектури для IIoT. Використання граничних обчислень допускає відповіді до 10 мс, а також вони мають повторюваність і детермінованість. Хмарні рішення можуть мати мінливий час відгуку, а також час затримки на порядок більше, ніж при використанні граничних пристроїв. Архітектор повинен враховувати, де розгорнути частину рішення на основі розгляду цих двох ефектів. Провайдерів хмарних обчислень також слід вибирати на основі моделей розгортання ЦОД. Якщо рішення IIoT розгортається у всьому світі або, можливо, буде розширюватися для охоплення

декількох регіонів, хмарний сервіс повинен мати ЦОД, розташовані в географічно близьких областях, щоб якомога більше зменшити час відгуку.

10.6 Граничні обчислення

10.6.1 Де знаходиться границя

Тема **граничних (крайових, edge) обчислень** фокусується на функціональних і практичних поглядах *Industrial Internet Reference Architecture* (*IIRA*) *Консорціуму промислового Інтернету* (*Industrial Internet Consortium - IIC*)

На рис. 10.8 показано багаторівневу структуру граничних обчислень з *наскрізними функціями (crosscating functions)*, які корисні при розгортанні визначених в IIRA структур. **Наскрізна функція підприємства** - це функція, яка повністю або частково включає до себе діяльність, виконувану структурними підрозділами організації, що мають різну функціональну та адміністративну підпорядкованість. До наскрізних функцій належать управління даними, взаємодія, координація, аналітика и безпека.

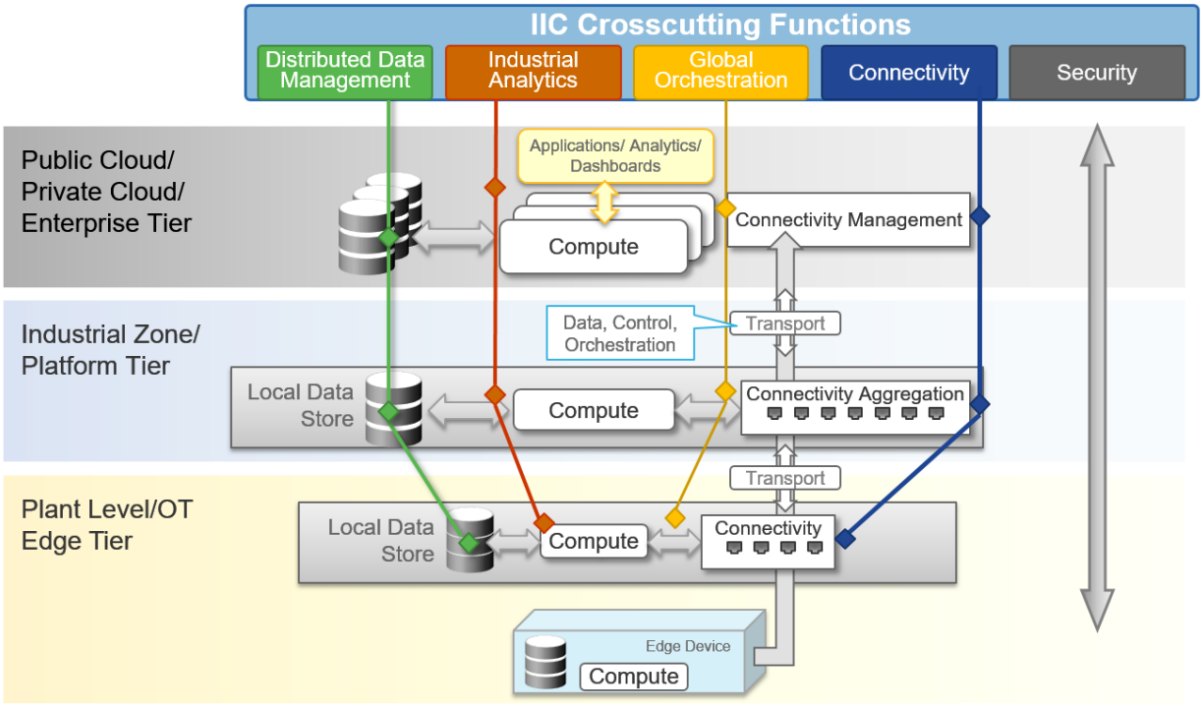


Рисунок 10.8 Наскрізні функції в архітектурах концепції граничних обчислень

Границя визначається скоріше як логічний рівень, а не певний фізичний поділ, тому вона відкрита для інтерпретації того, «де» вона знаходиться. Погляди зі сторони бізнесу та використання надають загальні уявлення, в той час як

бачення з функціональної точки зору та реалізації стосуються технічних аспектів.

З погляду бізнесу, місце розташування границі залежить від бізнес-проблеми або ключових завдань, які необхідно вирішити. ***Ключовими завданнями*** *є кількісно визначені високотехнологічні бізнес-результати, що очікуються від результуючої системи*. Існує сукупність фундаментальних рішень для використання IIoT, і *границя* рухається по цьому спектру в залежності від вирішуваної проблеми, про що свідчать наступні приклади з типових промислових задач.

Приклад 1: захист обладнання від перегріву

У цьому випадку термопара вимірює температуру в насосі. Насос зі здатністю до граничних обчислень може виконувати базову аналітику для визначення виходу температури за допустимі межі та зупиняти насос практично миттєво. У такому випадку відсутня затримка прийняття рішень і немає необхідності підключення до мережі для виконання цієї функції. Хоча зв’язок може використовуватися для інформування про ситуацію. Цінність інформації про температуру з плином часу швидко спадає, оскільки затримка реакції може призвести до пошкодження обладнання. В цьому випадку границя знаходиться на рівні пристрою, так як вона може досягти ключової мети навіть якщо зв'язок з системами і мережами більш високого рівня перерваний.

Приклад 2: моніторинг продуктивності заводів або виробничих ліній

Продуктивність обладнання і виробничих ліній часто виражається через показники ефективності, такі як **загальна технічна ефективність** (*Overall Equipment Effectiveness, OEE*). Аналіз значень від кількох давачів на ділянці обладнання може відбуватися практично в реальному часі на локальному шлюзі, забезпечуючи оперативний персонал чи системи даними щодо динаміки параметра OEE. В цьому випадку для забезпечення елементарної дієздатності необхідно отримувати інформацію з кількох технологічних джерел. Час отримання інформації грає важливу роль, оскільки затримки реакції в очікуванні рішень з хмари можуть призвести до значних втрат. Наявність такої проблеми вказує, що границя має знаходиться усередині заводу.

Приклад 3: оптимізація ланцюга постачання для локації чи заводу двічі на день

Оптимізація процесів системи постачання для локальних об'єктів, заводів чи, наприклад, нафтових родовищ для застосування алгоритмів оптимізації і аналітики потребує даних від кількох джерел через короткі проміжки часу. Ці алгоритми адаптуватимуть плани постачання в таких бізнес-системах, як SCM або ERP. Для простої працездатності потрібен зв’язок на локальному чи заводському рівні з рішеннями, що були прийняті протягом останніх годин. Додаткова інформація за межами периметра заводу може бути корисною, але не обов'язковою для ефективної оптимізації. У цьому випадку границя знаходиться на периметрі заводу, фабрики або локального об'єкта.

Приклад 4: прогнозування відмов обладнання і планування превентивних заходів

Для прогнозування відмов електричних погружних насосів (Electric Submersible Pump, ESP) моделі машинного навчання потребують даних від кількох платформ, що знаходяться за границею системи. Аналітичні моделі складні і потребують значного обсягу даних для навчання та перенавчання моделей. Вони також потребують регулярних потоків даних від працюючих насосів для визначення залишку терміну експлуатації кожної одиниці. Дані окремих ESP необхідно аналізувати регулярно, але знецінення інформації відбувається набагато повільніше, ніж в інших випадках, тому рішення можуть бути щоденними або щотижневими. Обчислення зазвичай виконується на рівні підприємства з використанням загальнодоступного або приватного хмарного сховища. Тим самим, границя знаходиться у хмарі.

Границя може знаходитися будь-де вздовж графіку «дані-час» (див. рис. 10.9 нижче), що демонструють наведені приклади.

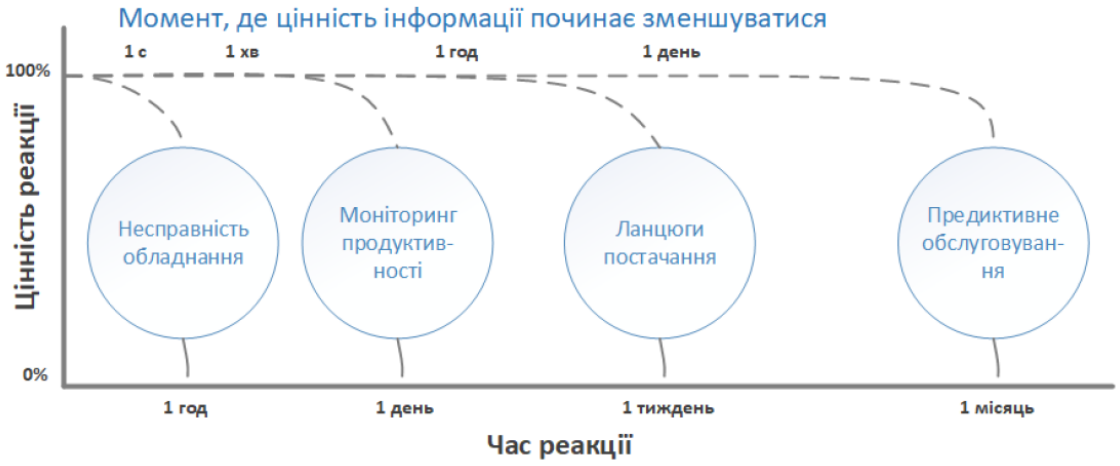


Рисунок 10.9 Графік "час-цінність" інформації від граничних пристроїв

Це і є «місцем», де дані давачів використовуються для досягнення конкретної ключової мети або для вирішення конкретної бізнес-проблеми

10.6.2 Що дає використання граничних обчислень

Рисунок 10.10 Переваги граничних обчислень

Периферійні обчислення - децентралізована обчислювальна інфраструктура, в якій обчислювальні ресурси та програмні служби можуть знаходитися вздовж усього шляху від джерела даних до хмари. Тобто обчислювальні потреби можуть бути задоволені "на границі", де дані збираються, або де користувач виконує певні дії. Перевагами периферійних обчислень є (рис. 10.10):

- покращена продуктивність;
- дотримання вимог про конфіденційність і безпека даних;

- менші експлуатаційні витрати.

Покращення продуктивності

Граничні обчислення - це не тільки спосіб збирання даних для передачі в хмару, а також обробка, аналіз і миттєві дії над зібраними даними на границі, що і робить їх важливими для оптимізації промислових даних.

Наприклад, у вітроенергетиці, якщо змінюється швидкість або напрямок вітру, локальне програмне забезпечення може аналізувати ці дані в режимі реального часу і налаштовувати окремі турбіни для оптимізації загального виробництва вітрової електростанції. У хмару надсилаються тільки узагальнені дані, що знижує вимоги до пропускної здатності каналу зв’язку і зменшує тривалість передачі даних.

Крім того, турбіни генерують терабайти даних. Відправлення цих даних до хмари для виконання складної аналітики є технологічно досяжним, але невиправдано дорогим для щоденного виконання. За допомогою периферійних обчислень кінцевий користувач може збирати потокові дані з турбіни і використовувати їх в режимі реального часу для попередження незапланованих простоїв і подовження терміну служби обладнання.

Проблема передачі великої кількості даних у реальному часі з мінімальними витратами від віддалених промислових об’єктів може бути пом'якшена шляхом використанням інтелектуальних пристроїв на границі мережі - на рівні заводу або на польовому рівні. Обчислення на периферійних пристроях наближають можливості аналітики ближче до устаткування та забезпечують менш вартісний варіант оптимізації.

Дотримання конфіденційності і безпеки даних

Публічна хмара має довгий перелік питань до конфіденційності, нормативно-правового регулювання та дотримання встановлених вимог, пов'язаних з секретними або конфіденційними даними. На сьогоднішній день, постачальники послуг можуть гарантувати приватний доступ і керування ним, однак за рахунок громіздкості, високої вартості, низької гнучкості та складності управління такими рішеннями. Граничні обчислення дозволяють підприємствам працювати незалежно, використовуючи публічну/приватну хмару разом з

локальними обчисленнями, що розміщується в цій області, регіоні, домені або в необхідних межах локальної безпеки.

Зменшення експлуатаційної вартості

Підключення, перенесення даних, пропускна спроможность та затримка обчислень в хмарних технологіях коштують дорого. Граничні обчислення вирішують ці проблеми за рахунок зниження вимог до пропускної здатності і затримок каналів зв'язку.

Якщо, наприклад, для буріння нафтових і газових свердловин в Нігерії потрібні розрахунки для прогнозування швидкості зниження видобутку нафти, альтернативними варіантами є створення власних ЦОД (враховуючи відповідні обмеження за вартістю і масштабуванням) або використання послуг провайдерів хмарних технологій (де найближчий центр обробки даних може знаходитися аж за 8000 кілометрів) зі значними грошовими витратами і ненадійним обслуговуванням. Завдяки граничним обчисленням кінцевий користувач може локально обробляти дані в режимі реального часу за невелику частину вартості публічної хмари, зберігаючи при цьому гнучкість, яку забезпечує хмарна інфраструктура.

Граничні обчислення утворюють елемент, необхідний для обробки великих обсягів створених в IIoT даних, що прямують від пристрою до хмари. Обробка даних ближче до місця, де вони генеруються, і в межах часу відгуку, який вимагають локальні застосування, вирішує проблеми швидкого зростання обсягу даних. Граничні обчислення зменшують час реакції на події, за рахунок виключення передачі даних в хмару і назад для проведення аналітики. Це дозволяє уникнути потреби нарощування пропускної здатності мережі, усуваючи необхідність передачі гігабайтів даних у хмару. Це також захищає конфіденційні дані IIoT, аналізуючи їх локально в приватній мережі.

10.6.3Топологія граничних обчислень

Рисунок 10.11 ілюструє кілька прикладів реалізації граничних обчислень на основі різних поглядів бізнесу. Приклади розглядаються зліва направо, по мірі ускладнення граничного рівня і об’єднання декількох системних функцій .

Обчислювальний рівень переміщується вгору по ієрархії, об'єднуючи можливості обробки та дані знизу.

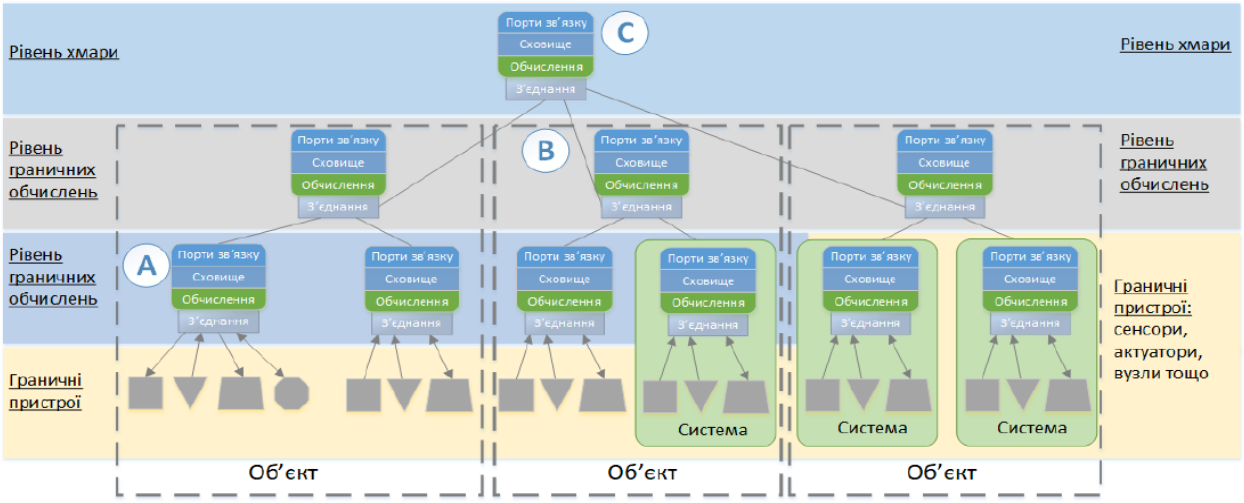


Рисунок 10.11 Ілюстрація різновидів топології граничних обчислень

Кілька наведених на малюнку варіантів означають, що існує багаторівневий взаємозв'язок границь і хмар. Там, де це можливо, в цифровізації завжди буде використовуватись взаємодоповнюваність границі і хмари, де швидкі і локалізовані обчислення відбуватимуться на периферії, а глобальні обчислення, розробки моделі, менеджмент і безпека можуть скористатися «мудрістю хмари».

Розгляд різновидів топології граничних обчислень почнемо з прикладу простого регулятора температури (див. «А» зліва на рис. 10.11).

Проблема, яку необхідно вирішити, - це моніторинг температури та керування конкретним пристроєм або зоною. У цьому випадку периферійними пристроями будуть термопари, що передають дані про температуру, та елемент, що забезпечує нагрівання або охолодження. Граничним же обчислювальним пристроєм буде сам регулятор температури, який виконує алгоритм керування і здійснює регулювання.

Якщо мета полягає в підтримці температури на декількох пристроях або ділянках, то граничним пристроєм стає сам регулятор температури (будь-то окремі компоненти або автономні системи), а граничний обчислювальний рівень стає системою, що координує управління, як правило це ПЛК або SCADA система ("B" на рис10.11).

Якщо комерційна мета полягає у моніторингу та управлінні кількома географічно віддаленими об'єктами, то границя кожного окремого об'єкта повідомляє про свій стан обчислювальному рівні в хмарі ("С" на рис. 10.11).

10.6.4 Модель граничних обчислень в IIoT

Граничні обчислення є вертикальними в комплексному розумінні, від пристрою до хмари, і горизонтальними з погляду підсистем IIoT. Нова модель обчислень повністю розподілена і може підтримувати широкий спектр взаємодій і комунікаційних підходів, включаючи:

- однорангові мережі, наприклад, зони безпеки, які повідомляють про наявність об'єктів в межах цих зон;
- співпрацю граничних пристроїв, наприклад, самоорганізацію транспортних засобів, які пересуваються разом, або вітрових турбін у віддалених місцях;
- розподілені запити за даними, що зберігаються в пристроях, хмарі та в будь-якому місці між ними;
- розподілене керування даними, визначення того, де, які дані, і як довго повинні зберігатися;
- управління даними, включаючи якість, зручність використання, конфіденційність та безпеку даних.

10.6.5 Архітектура та вимоги до граничних обчислень

На рис. 10.12 показані логічні об'єкти, що знаходяться в хмарі або на краю та з'єднані через WAN. Коли вперше були введені хмари, існувала тенденція «перемістити все в хмару», але через затримки мережі і вартість передачі великої кількості даних, завдання, які більш пов’язані з логікою, залишалися на периферії. Завдяки поліпшенню обчислювальної потужності та розширенню можливостей пристроїв, кількість завдань, що виконуються на границі, продовжуватиме зростати.

Рисунок 10.12 Схема логічної архітектури граничних обчислень

Загальні вимоги до граничних обчислень.

Зв'язок: граничні пристрої повинні продовжувати функціонувати навіть за умови, що передача даних може буде тимчасово перервана.

Можливості граничних пристроїв: граничні пристрої повинні підтримувати можливості периферійних обчислень: зв'язок, локальні обчислення і локальні сховища.

Функціональність граничного пристрою: граничним пристроям можна надавати різні специфічні характеристики залежно від потреб різних галузей промисловості: компактні розміри, низьке енергоспоживання, антивібрація, електромагнітне екранування та водонепроникність, тощо.

10.6.6 Приклади використання граничних обчислень

Для полегшення обговорення границь та необхідних засобів для впровадження периферійних обчислень у кожному прикладі використання додаються пункти «Основні вимоги», «Границі» та «Граничні пристрої». «Основні вимоги» не відносяться до стандартів, тому вони не є нормативними. «Границі» - це суб'єктивне бачення щодо варіантів використання, що не має на меті бути еталонним, бо, як правило, межі кожної служби або програми змінюються в залежності від рішень, використаних розробниками системи.

Подібним чином, термін «Граничні пристрої», що використовується в прикладах, насправді не обмежується наведеним переліком.

10.6.6.1 Забезпечення безпеки персоналу

Мета: використовувати багатофункціональні детектори газу для моніторингу впливу шкідливих газів під час робочої зміни. Створити профілі впливу в режимі реального часу, використовуючи дані давачів, і скоригувати графік роботи або послідовність виконання робіт, щоб попередити проблеми зі здоров'ям.

Опис використання: безпека в шкідливому або небезпечному для життя середовищі, наприклад видобуток в шахтах чи промислова хімія. Є й інші питання, пов'язані з виробництвом, - стан використовуваних інструментів, стан конвеєра або транспортних засобів, що застосовуються для переміщення руди, і обсягів руди, виробленої в обліковий період тощо.

Основні вимоги:

1. (Моніторинг навколишнього середовища) необхідно реалізувати можливості виявлення отруйних газів, визначення та контролю температури навколишнього середовища, керування освітленням;

2. (Персональний моніторинг) особиста система моніторингу основних показників стану організму повинна бути встановлена локально і дані будуть відправлені до центральної станції моніторингу в операційному центрі.

Границі: робоче місце (наприклад, в офісі експлуатації шахти або операційному центрі).

Граничні пристрої: особиста система моніторингу основних показників стану організму (температура тіла, частота серцевих скорочень і кров'яний тиск, рівень CO_2), персональний трекер, моніторинг навколишнього середовища (температура навколишнього середовища, рівень CO, виявлення небезпечного газу, освітлення), інструментів, вантажних автомобілів і автоматично керованих транспортних засобів, конвеєрних стрічок і вагових пристроїв.

10.6.6.2 Предиктивне обслуговування підключених ліфтів

Мета: За допомогою застосувань, встановлених на приєднаних до мережі ліфтах, оператори і технічні працівники здатні виконувати профілактичне обслуговування ліфтів на основі даних, які надаються на границі.

Опис використання: Оператори підключених ліфтів, спираючись на прогностичні функції граничних пристроїв для обслуговування їхніх систем, підвищують надійність і зменшують час простою ліфтів. Експлуатаційні витрати цих систем значно знижується в той же час, як ефективність може бути значно покращена.

Підключений ліфт використовує багато давачів для збору даних про шум, вібрацію, температуру тощо. Поточний стан ліфта може бути оцінений на основі аналізу отриманих даних. З ліфтами, підключеними до граничних обчислювальних пристроїв, звідки отримані дані завантажуються в хмару, оператори ліфтів можуть отримати поточний статус всіх ліфтів. Технічні працівники потім можуть виконувати попереджувальне технічне обслуговування з використанням даних граничних обчислень та даних у хмарі, для вибіркової перевірки і обслуговування ліфта, які, швидше за все, будуть виходити з ладу відповідно до даних аналітики. Предиктивне обслуговування підвищує ефективність роботи обладнання, знижуючи витрати на технічне обслуговування за рахунок цілеспрямованого запобігання збоїв та уникнення незапланованих простоїв.

Основні вимоги:

1. Граничні пристрої пропонують відкриті API, які дозволяють третім особам розробляти програмні додатки, що встановлюються на них.

2. Для підтримки цілодобового моніторингу, граничні пристрої підтримують оновлення свого програмного забезпечення.

Границя: центр експлуатації ліфта або сам ліфт

Граничні пристрої: Інфрачервоні давачі, давачі ваги, диму, вібрації, шумові, камери та інтерфейси.

10.6.7 Периферійні обчислення та промислова аналітика

Аналітика, в широкому розумінні, визначається як дисципліна, що перетворює дані в інформацію та комерційну цінність шляхом систематичного аналізу. Промислова аналітика - це використання аналітики в системах IIoT.

Розширена аналітика лежить в основі переходу на новий рівень в обробці інформації і, що стосується машин, процесів та структур даних, передбачає нову цінну інформацію та знання для оптимізації прийняття рішень і забезпечення інтелектуальних операцій, що впливають на бізнес - результати та соціальні цінності. Ці нові уявлення та знання можна застосовувати на будь-якому рівні у будь-якій галузі, якщо можливо зібрати відповідні дані та коректно провести їхній аналіз. Існує навіть думка, що дані - це нове "чорне золото". Якщо це так, то аналітика даних - це новий двигун, який стимулює IIoT трансформації.

Аналітична обробка даних може класифікуватися за багатьма різними ознаками, залежно від того, де вона виконується, який період часу для відповіді потрібен, і які функціональні можливості вона намагається досягти. Один аналітичний потік може включати до себе граничну аналітику для попереднього визначення сутності даних і негайного спрацювання, та аналіз в хмарі, який поповнює останніми даними з границі *історичні big data сховища* і повертає результати назад до границі для подальшого їх застосування.

Технологія та еволюція промислової аналітики

Досягнення IT та OT можливостей, таких як обчислювальна потужність, пропускна здатність каналів зв'язку, низька латентність, можливості програмного забезпечення та сенсорна технологія усунули технологічні обмеження та дозволили розгорнути аналітику через всю систему IIoT. Наприклад, дивлячись на граничний рівень системи, можливості обробки на границі в поєднанні з низькою затримкою комунікації дозволили працювати алгоритмам в реальному часі, підтримуючи моделі, які генерують цінну інформацію та керуючий вплив на систему. Аналогічно, далі розглядаючи рівень хмари, те, що було колись непрактичним - виконувати потокову аналітику величезних об’ємів даних - зараз стає можливим завдяки *Big Data* і комунікаціям з високою пропускною спроможністю. Ці ж досягнення також дозволили

розподілити аналітичну обробку даних так, щоб зникла потреба в централізованих обчисленнях і з’явилась можливість впровадження їх в екосистему IIoT.

Де повинна виконуватися аналітика?

Більшість рішень впровадження промислової аналітики використовують гібридний підхід, де аналітики працюють на всіх рівнях від границь до хмари, і кожен її рівень відповідає конкретній бізнес-меті. Економічна вигода виникає через зменшення кількості даних, які надсилаються та зберігаються в хмарі. Периферійна аналітика зменшує витрати на зберігання і обробку малоцінних і часто повторюваних даних.

Шуми, які накладаються на дані, негативно впливають на аналітичні моделі. Для того, щоб уникнути проблеми використання *Big Data* з шумами, периферійна аналітика повинна відфільтрувати дані до відправлення в хмару. А це вже робота для розумних цифрових систем фільтрації на основі DSP.

10.6.8 Безпека граничних обчислень

Безпека є важливим фактором для граничних обчислень. Більша кількість компонентів і каналів зв'язку створюють більший потенціал для *векторів атаки*. Необхідні інновації для контролю, управління та забезпечення безпеки глобально розподілених систем і попередження неминучих зломів. *Industrial Internet Security Framework (IISF)* документує узагальнену структуру безпеки. У реалізації периферійних обчислень:

- безпека повинна бути вбудована в кожен пристрій і рівень архітектури;
- комп'ютерні та мережні кінцеві точки повинні контролюватися та управлятися;
- необхідно застосувати останні патчі;
- атаки повинні бути ізольованими;
- пошкоджені компоненти повинні бути здатними до відновлення.

10.6.9 Оркестрування (координація) граничних обчислень

Централізований характер хмарних обчислень забезпечує простий доступ до масштабованого та адаптивного сховища спільних фізичних або віртуальних ресурсів. В той же час стає не так просто забезпечити подібну масштабованість та адаптивність для граничних обчислень:

- граничні обчислювальні ресурси можуть бути на окремих ізольованих частинах системи, де вони не можуть взаємодіяти з основною системою при координації обчислень;
- доступ до граничних обчислювальних ресурсів може бути складним або дорогим;
- граничні обчислення, що ведуться в захищених зонах можуть бути недоступними для розширення,
- технічним працівникам для виконання роботи може бути складно дістатися до граничних пристроїв.

З цими обмеженнями підхід змінюється на протилежний. Розуміння як і “вбудованих”, так і мережевих ресурсів має вирішальне значення для використання правильного програмного забезпечення в правильному місці. Після впровадження системи необхідно забезпечити її інструментами для управління, моніторингу та захисту впродовж всього життєвого циклу. Якщо доступні об’єми ресурсів змінюються, для програмного забезпечення може виникнути потреба урізання або перерозподілу функціональності, обмеження використання пам’яті, скорочення баз даних та записів. Також необхідно прогнозувати динаміку використання для вирішення проблем до їх виникнення.

Завдання для розробників і адміністраторів полягає в тому, щоб зрозуміти не тільки фізичні вимоги до прикладного рівня (розрахунки входів, виходів, підключення тощо), а також вимоги до безпеки та обробки даних, і як ці вимоги змінюються в залежності від типу центрального процесора та операційної системи. Можуть знадобитися галузеві стандартні розрахунки та показники.

Оркестрування та керування інфраструктурою на границі створює проблеми, відмінні від хмарних, значною мірою завдяки:

Гетерогенності ділянок пристроїв і програмових застосувань: на границі не розглядається можливість однорідності пристроїв або апаратних чи програмних платформ. Система управління інфраструктурою повинна бути достатньо гнучкою, щоб керувати безліччю пристроїв й ефективно використовувати їх ресурси. Для забезпечення деякого рівня однорідності на границі можна використовувати як технології віртуалізації, так і контейнеризації. Крім того, пристрої можуть вести себе по-різному в залежності від області застосування. Рішення координації для розумного заводу, де використовуються статичні вузли та надійна мережа, буде мати інші характеристики від координації системи логістики або розумного міста, де транспортні засоби постійно рухаються і якість зв'язку постійно змінюється.

Використанню різноманітних технологій зв'язку та комунікації: шлюзи IIoT повинні обробляти декілька варіантів підключення з використанням різних протоколів. Координатор повинен знати про наявні рішення для гарантії зв'язку між розгорнутими функціями та програмовими додатками.

Відмінності в можливостях, вимогах і обмеженнях: вищий рівень однорідності і практично нескінченна доступність ресурсів у хмарі полегшує процес координації. І навпаки, на границі можна побачити ширший діапазон вимог до послуг, можливостей пристроїв і обмежень. Наприклад, пристрої на периферії мають різні давачі, приводи, операційну систему реального часу або промислові мережі. Тому координатор повинен мати *повну інформацію* про можливості інфраструктури. Також необхідно заздалегідь знати обмеження на відповідних вузлах (наприклад, пропускну здатність, ємність батареї живлення, потужність процесора, пам'ять). Під час координації сервісна служба повинна вміти описувати свої вимоги, що будуть перевірятися на відповідність доступним можливостям і обмеженням, знайденим в опису інфраструктури.

З огляду на це, **координатори можуть працювати як вертикально, так і горизонтально**. Вертикальні координатори обробляють служби в **конкретній області**, в той час коли горизонтальні керують **службами в різних областях**, що забезпечує інтеграцію між ними. Прикладом може бути розумний завод, що співпрацює з логістичною компанією, і кожен має свою власну систему

координації. В такому випадку горизонтальна система координації співставляє служби, які охоплюють різні області (наприклад, сервіс, який регулює пропускну здатність виробничої лінії на основі поточного розташування необхідних запасів).

Оркестрування - це важлива частина граничних обчислень, що створює певну платформу для підтримки IT та OT-операцій в IIoT. Можливість координації при впровадженні нових сервісів і програмних застосунків надає границі можливість бути програмованою та постачати послуги, необхідні її споживачам. Намагаючись забезпечити потрібну якість обслуговування в хмарі, необхідно забезпечити її присутність і в периферійних рішеннях.

10.7 Туманні обчислення

10.7.1 Що таке туманні обчислення

Як неважко здогадатися, **туман** - це, як і «хмара» деяка зв'язана розподілена обчислювальна потужність. Застосуємо диференційний підхід до хмари і покладемо, що замість одного дискретного вузла хмари у складі: процесор, ОЗУ, ПЗУ, пристрої введення / виведення ми маємо скалярне поле (розподіл в обсязі щільності) обчислювальної потужності, оперативної і постійної пам'яті, а також векторне поле потоків даних. Комп'ютери стають меншими. Комп'ютери стають дешевшими. Про розмір і найголовніше - споживання енергії, взагалі, мова не йде. Зараз щільність обчислювальних пристроїв настільки висока, що впору застосовувати до неї статистичні методи.

Разом із здешевленням комп'ютерів подешевшали і системи зв'язку. Блютус є майже скрізь і не факт, що вже завтра він не з'явиться у ваших черевиках. І ось саме зараз не залишилося перешкод для того, щоб всі ці маленькі слабкі обчислювальні машинки об'єдналися в один великий бузковий обчислювальний туман!

В основі *туману* лежить *крапля* - чіп мікроконтролера з пам'яттю і інтерфейсом для передачі даних на борту, і чіп бездротового зв'язку типу чарункової мережі (сенсорна мережа). Живлення *крапля* отримує від маленької батарейки, якої, проте, вистачить на пару років роботи з регулярними перервами на сон. До *краплі* можуть бути підключені пристрої введення (давачі всіх типів,

від температури і напруги до положення в просторі і рівня ультрафіолетового випромінювання) і виведення (світлодіоди, ЖК і LED індикатори, сухі контакти, виконуючі механізми, тощо).

«І ось коли ми в двох кроках від купи казкових скарбів ...» - як співав герой відомого мультмюзікла, залишається найцікавіше - в цій мережі може зберігається і оброблятися інформація, *безпосередньо не пов'язана з цими самими давачами*. Очевидно, що для більшості завдань продуктивності сучасних мікроконтролерів більш ніж достатньо і ми отримуємо *поле надлишкової обчислювальної потужності*. А грошей, патронів і обчислювальної потужності, як відомо, зайвих ніколи не буває. Програмна основа в першу чергу, ймовірно, буде включати в себе *гіпервізор*, здатний консолідувати потужність *туманної мережі* і представляти їх як одну багатопроцесорну систему.

Для забезпечення більш продуктивної передачі даних, в тумані будуть створюватися додаткові тунелі - *краплі* можуть мати високошвидкісні інтерфейси і забезпечувати кращу зв'язність туману. Не мине й 10 років як *обчислювальний туман* охопить весь ареал проживання *homo sapiens* і з комерційних приватних хмар обчислення підуть в туман, який не має власника. Програми будуть паразитувати на обчислювальному тумані і конкурувати між собою. Це буде екосистема, на порядок більше жива, ніж Інтернет сьогодні.

В ідеалі, в еру *туманних обчислень* комп'ютери-вузли знаходяться буквально всюди - під ногами, в повітрі, на вулиці... Вони настільки мініатюрні і дешеві, що їх можна носити з собою кілограмами. У наш час це швидше за все буде якесь програмне середовище, консолідуюче ресурси безлічі віртуалізованних *крапель*, яке дозволяє на такій *паралельній віртуальній машині* виконуватися програмам, написаним під крос-платформні середовища - платформна залежність в такому оточенні буде убивча. Швидше за все, мова буде йти про Java, CLR, Python, JavaScript...

Таким чином, *туманні обчислення* - це еволюційне розширення *хмарних обчислень на краю*. Тепер, підсумовуючи, можна дати визначення того, що ж таке *туманні обчислення*, або *fog computing?* **Це обчислення, які засновані на розподіленій інфраструктурі з негарантованою доступністю**. **Топологічно -**

це чарункова (mesh) мережа з динамічною маршрутизацією, вузлами якої є порівняно однорідні по обчислювальній потужності комп'ютери (рис. 10.13).

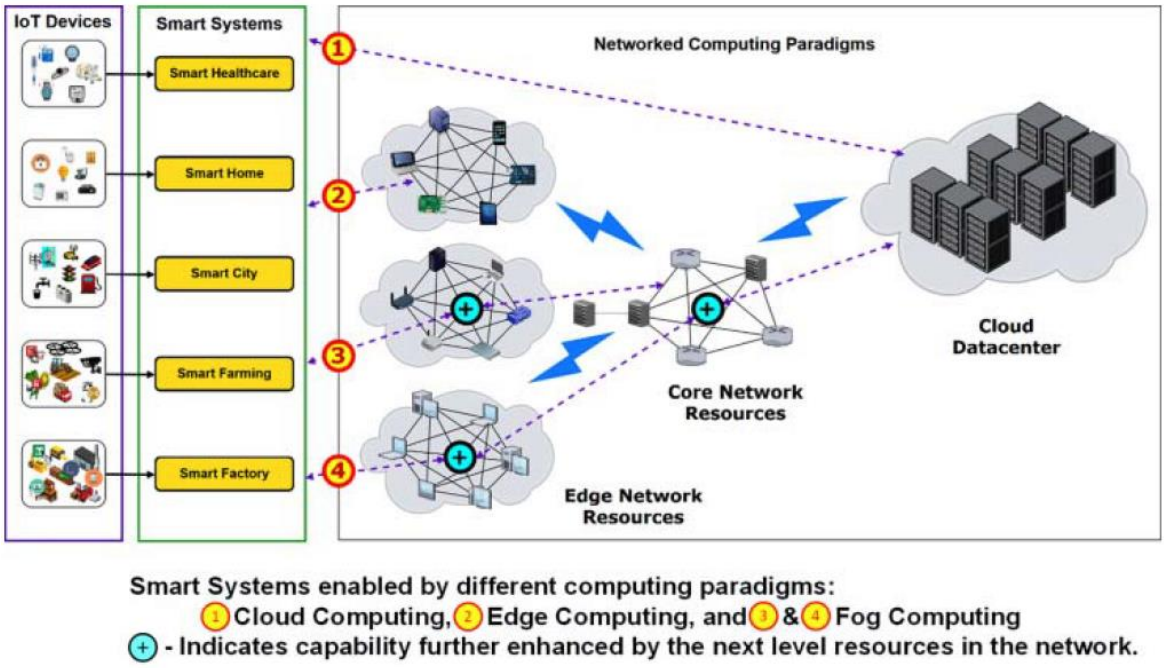


Рисунок 10.13 IoT-застосунки та середовища з підтримкою обчислювальних парадигм

Тут можна відзначити, що нам не бачити повноцінних *туманних ПК* і *туманних мереж* поки не будуть виконані наступні вимоги:

- наявність високоефективних (в Вт/MIPS) SoC поширених архітектур (ARM, x86, AMD64) з мінімальною обв'язкою по екстремально низькій ціні ($2 … $10);
- стандартизація та поширення консолідуючих гіпервізорів;
- повсюдне поширення IPv6 і швидкісних інтерфейсів - як кабельних, так і бездротових;
- наявність дешевих і ємних джерел автономного живлення (як мінімум на порядок більш ємних, ніж сучасні літій-іонні акумулятори).

10.7.2 Порівняння туманних, граничних і хмарних обчислень

Ми вже визначили граничні обчислення як переміщення обробки до того місця, де генеруються дані. У разі IIoT граничним пристроєм може бути сам давач з невеликим мікро контролером або вбудованою системою, здатною до WAN-зв'язку. В інших випадках границя буде являти собою шлюз в мережевій архітектурі. Гранична обробка також зазвичай згадується в контексті машина-

машина, де існує щільна кореляція між границею (клієнтом) і сервером, розташованим в іншому місці. Граничні обчислення існують, як зазначено, для усунення проблем із затримкою при передачі і непотрібним завантаженням каналу зв'язку, а також для додавання таких сервісів, за допомогою яких можливо змінювати алгоритм роботи вузла і підвищувати рівень його безпеки, шляхом зближення з джерелом даних. У граничного пристрою може бути зв'язок з хмарним сервісом ціною затримки повідомлень; він не бере активної участі в хмарній інфраструктурі.

Туманні обчислення трохи відрізняються від парадигми граничних обчислень. У туманних обчисленнях в першу чергу розділяються API інфраструктури та стандарти зв'язку з іншими туманними вузлами і/або покриваючою хмарною службою. Туманні вузли є розширеннями хмари, тоді як граничні пристрої можуть використовувати або не використовувати хмару. Іншим ключовим принципом туманних обчислень є те, що туман може існувати в ієрархічних шарах. Туманні обчислення також можуть балансувати навантаження і керувати даними зі сходу на захід і з півночі на південь, щоб допомогти в балансуванні ресурсів. З визначення хмари і її сервісів, які розглядалися в попередніх розділах, можна думати про ці туманні вузли (краплі) як просто про ще одну (хоча і менш потужну) інфраструктуру в гібридній хмарі.

10.7.3 Архітектура туманих обчислень

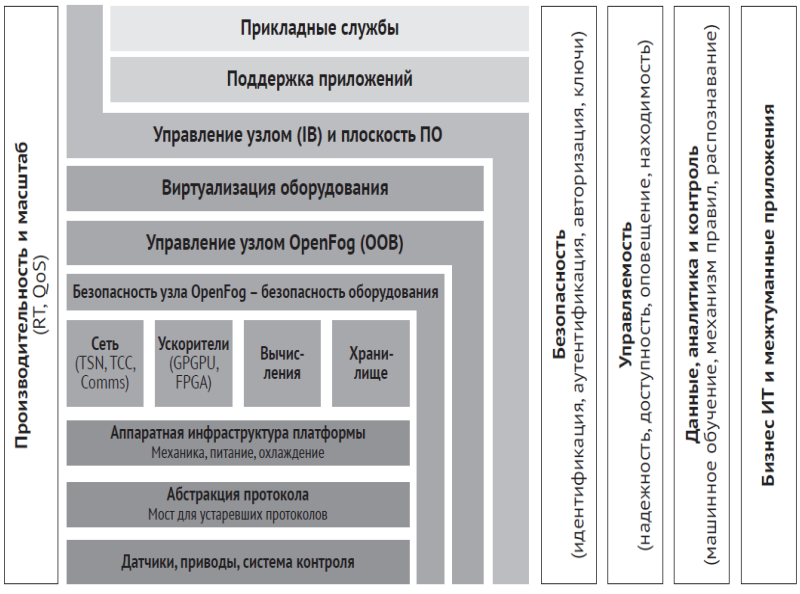


Рисунок 10.14 Архітектура OpenFog RA

Архітектура туманих обчислень визначається документом *OpenFog Consortium Reference Architecture*. Цей документ розроблено консорціумом *OpenFog*, який є некомерційною галузевої групою, зайнятої визначенням стандартів сумісності з туманними обчисленнями. Хоча він не є органом стандартизації, але

впливає на напрямок діяльності інших організацій за допомогою взаємодії і впливу у галузі. Архітектура *OpenFog* - це модель, яка допомагає архітекторам і бізнес-лідерам в створенні устаткування, програмного забезпечення і придбання інфраструктури для туманних обчислень. *OpenFog* розуміє переваги хмарних рішень і бажання розширити цей рівень обчислень, зберігання, мереж і масштабованості до граничного рівня, не жертвуючи затримками і пропускною спроможністю.

OpenFog Reference Architecture є багатошаровою: від граничних давачів і виконавчих механізмів внизу, до програмних додатків нагорі (див. рис. 10.14).

10.7.3.1 Прикладні сервіси

Роль сервісного рівня полягає в наданні *прозорого доступу* до спеціальних послуг, необхідних для виконання завдань. Це включає в себе надання коннекторів для інших служб, пакетів для аналітики даних, при необхідності надання користувацького інтерфейсу і надання широкого спектру усіх основних послуг.

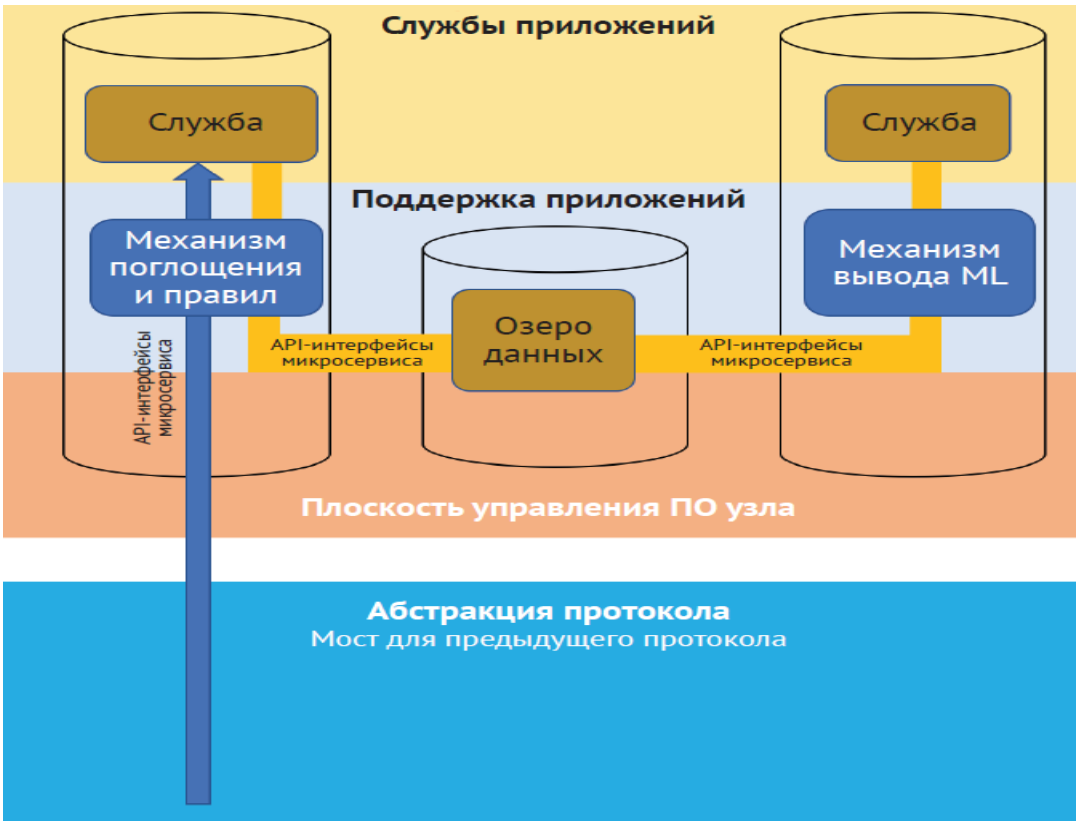


Рисунок 10.15 Приклад програми OpenFog. Тут можуть бути розгорнуті кілька контейнерів, кожен з яких надає різні послуги та функції підтримки

Коннектори на прикладному рівні з'єднують служби з рівнем підтримки. Рівень абстракції протоколу забезпечує шлях для спілкування конектора

безпосередньо з давачем / актуатором. Кожна послуга повинна розглядатися як мікросервіс в контейнері. Консорціум *OpenFog* виступає за метод розгортання контейнерів в якості основного методу розгортання програмного забезпечення на границі. Це має сенс, коли ми розглядаємо граничні пристрої як розширення хмари. Приклад розгортання контейнера може виглядати наступним чином (див. рис. 10.15).

Кожен циліндр на малюнку являє собою окремий контейнер, який можна розгорнути і керувати ним окремо. Потім кожен сервіс надає API для взаємодії контейнерів і шарів.

10.7.3.2 Підтримка програмних застосувань

Це компонент інфраструктури, який допомогає зібрати остаточне рішення для клієнтів. Цей рівень може залежати від того, як він був розгорнутий (наприклад, як контейнер). Підтримка здійснюється в багатьох формах, у тому числі:

- управління програмним застосунком (ідентифікація образу, перевірка образу, встановлення образу, аутентифікація);
- засоби журналізації;
- реєстрація компонентів і сервісів;
- рушії виконання (контейнери, віртуальні машини);
- мови виконання (*Node.js, Java, Python*);
- сервери додатків (*Tomcat*);
- шини повідомлень (*RabbitMQ*);
- бази даних та архіви (*SQL, NoSQL, Cassandra*);
- аналітичні фреймворки (*Spark*);
- служби безпеки;
- веб-сервери (*Apache*);
- інструменти аналітики (*Spark, Drool*).

OpenFog пропонує, щоб усі ці служби були упаковані у вигляді контейнерів, так як показано на рис. 10.15. Еталонна архітектура не є строгим керівництвом, і архітектор повинен вибрати правильний рівень підтримки, яка може бути задіяна лише на обмеженому граничному пристрої. Наприклад,

обробка та ресурси можуть допускати лише прості правила і забороняти що-небудь на зразок потокового процесора, не кажучи вже про рекуррентні нейронні мережі.

10.7.3.3 Управління вузлом і базове ПЗ

Це відноситься до управління *In-Band (IB)* і визначає, як *туманний вузол* (капля туману) спілкується з іншими вузлами в своїй області. Вузли також керуються через цей інтерфейс для оновлень, статусу і розгортання. Базове ПЗ може включати в себе операційну систему вузла, призначені для користувача драйвери і прошивку, протоколи зв'язку і управління, управління файловою системою, програмне забезпечення для віртуалізації і контейнеризацію для мікросервісів.

Цей рівень стека програмного забезпечення стосується майже будь-якого іншого шару в *OpenFog Reference Architecture*. Типовими особливостями базового ПЗ є:

- **виявлення сервісу** - дозволяє ситуативні моделі, довірчі відносини між хмарами;
- **виявлення вузла** - дозволяє додавати нові туманні вузли і приєднувати їх до кластерів, подібних хмарним кластерам;
- **управління станом** - дозволяє різні обчислювальні моделі для обчислень із збереженням стану і без збереження для багатьох вузлів;
- **управління Pub / Sub** - дозволяє переносити дані, а не витягувати їх. Крім того, дозволяє використовувати рівні абстракції при розробці програмного забезпечення.

Еталонна архітектура *OpenFog* повинна забезпечувати рівні розгортання. Тобто, туманна архітектура не просто обмежена хмарою, підключеною до туманного шлюзу, з'єднаному з декількома давачами. Насправді існує безліч топологій, що залежать від масштабу, пропускної здатності, навантаження і економічних факторів, які можуть бути розроблені. Еталонна архітектура повинна підходити для безлічі топологій, так само, як реальна хмара може динамічно масштабуватися і балансувати навантаження на підставі попиту.

10.7.3.4 Віртуалізація обладнання

Як і звичайні хмарні системи, OpenFog визначає апаратне забезпечення як рівень віртуалізації. Програмні застосунки не повинні мати прив'язки до певного набору обладнання. Тут система повинна балансувати навантаження через туман і переміщатися по ресурсам або додавати ресурси в міру необхідності. Всі апаратні компоненти віртуалізовані на цьому рівні, включаючи обчислення, мережу і сховище.

10.7.3.5 Безпека вузла OpenFog

Консорціум визначає цей рівень як частину безпеки обладнання стека. Туманні вузли вищого рівня повинні мати можливість контролювати туманні вузли нижчого рівня як частину ієрархії в топології. Однорангові вузли повинні мати можливість спостерігати за своїми сусідами на сході-заході (по горизонталі). Цей шар також виконує наступні функції:

- криптування;
- монітори спостереження і фізичної безпеки;
- інспекції та моніторингу пакетів (зі сходу на захід (по горизонталі) і з півночі на південь (по вертикалі)).

10.7.3.6 Мережа

Це перший компонент апаратного рівня. Компонент являє собою модуль вертикального та горизонтального зв'язку. Мережевий рівень обізнаний про туманну топологію і маршрутизацію. Він виконує фізичну функцію маршрутизації до інших вузлів. Це велика відмінність від традиційної хмарної мережі, яка віртуалізує всі свої внутрішні інтерфейси.

Вимоги до мережевого компоненту:

- стійкість, якщо лінія зв'язку відмовляє. По суті, може знадобитися зрозуміти, як перебудувати мережу, щоб потік даних не переривався;
- мережевий рівень також є місцем для передачі даних і їх перепакування від не IP-давачів до IP-протоколів. Прикладами цього є Bluetooth, Z-Wave і проводові давачі;
- обробка відмов в з'єднанні;

- можливість коммунікування з багатьма мережами (Wi-Fi, проводова, 5G);
- забезпечення типової мережевої інфраструктури, необхідної при розгортанні підприємства (безпека, маршрутизація, тощо).

10.7.3.7 Прискорювачі

Іншим аспектом *OpenFog*, чим він відрізняється від хмарних схем, є поняття послуг прискорювача. Прискорювачі тут звичайно надаються у вигляді графічних процесорів і навіть *FPGA* для надання послуг для обробки зображень, машинного навчання, комп'ютерного зору і сприйняття, обробки сигналів і шифрування / дешифрування. *OpenFog* передбачає, що туманні вузли можуть забезпечуватися ресурсами і розподілені по мірі необхідності. В ієрархії туману можна змусити ферму вузлів другого або третього рівня забезпечити динамічно додаткові обчислювальні засоби в міру необхідності.

Ми можемо змусити працювати інші форми прискорення в тумані, наприклад:

- вузли, призначені для масового зберігання в разі, якщо необхідно створити велике озеро даних;
- вузли, які включають в себе альтернативні лінії зв'язку, такі як супутникова радіостанція у випадку аварії, коли не працює весь наземний зв'язок.

10.7.3.8 Обчислення

Це основа служби управління обчислювальними ресурсами туману. Її мета - визначити і врахувати обчислювальні ресурси на основі попиту. Вона також несе відповідальність за управління системним гіпервізором і віртуальними машинами, або може управляти контейнерами. Масштабування на вимогу є невід'ємною її частиною. Основні функції:

- виконання завдань;
- моніторинг і забезпечення ресурсів;
- балансування навантаження;
- запити можливостей.

10.7.3.9 Зберігання

Середовище зберігання в архітектурі підтримує інтерфейс низького рівня до туманного сховища. Нові типи зберігання, такі як *озера даних*, можуть знадобитися на границі для жорсткого аналізу у реальному часі. Шар зберігання також буде керувати всіма традиційними типами пристроїв зберігання, такими як:

- масиви віртуальної пам'яті;
- змінні диски;
- flash-накопичувачі;
- RAID-масиви;
- шифрування даних.

10.7.3.10 Інфраструктура апаратної платформи

Рівень інфраструктури - це не стільки фактичний рівень між програмним і апаратним забезпеченням, скільки фізична і механічна структура туманного вузла. Оскільки туманні пристрої будуть знаходитися в суворих і віддалених місцях, вони повинні бути міцними і стійкими, а також автономними. *OpenFog* визначає випадки, які необхідно враховувати при розгортанні туману, в тому числі:

- розміри, потужність і вагові характеристики;
- системи охолодження;
- механічне кріплення і утримання;
- механізм обслуговування;
- стійкість до фізичного впливу і звітність.

10.7.3.11 Абстракція протоколу

Рівень абстракції протоколу пов'язує самі нижні елементи системи IIoT (давачі) з іншими шарами туманного вузла, іншими туманними вузлами і хмарою. *OpenFog* підтримує модель абстракції для ідентифікації та зв'язку з сенсорним пристроєм через шар абстракції протоколу. Абстрагуючи інтерфейс до давачів і периферійних пристроїв, гетерогенну суміш давачів/актуаторів можна розгорнути на одному туманному вузлі, наприклад, аналогові пристрої,

які проходять через цифро-аналогові перетворювачі, а також цифрові давачі. Навіть інтерфейси до давачів можуть бути індивідуалізовані, наприклад, *Bluetooth* для температурних пристроїв в транспортних засобах може підключатися до інших давачів двигуна; давач з інтерфейсом *SPI* на різних автомобільних електроприладах і давачі, що підключені через інтерфейс *GPIO* (наприклад, давачі відкриття дверей і давачі руху). Абстрагуючи інтерфейс, верхні шари стека програмного забезпечення можуть звертатися до таких розрізнених пристроїв за допомогою стандартизованого підходу.

10.7.3.12 Давачі, приводи і системи управління

Це нижній рівень туманного стека IIoT: давачі і граничні пристрої. Ці пристрої можуть бути розумними, німими, дротовими, бездротовими, розташовуватися ближче, далі, тощо. Асоціація, однак, полягає в тому, що вони взаємодіють певним чином з вузлом туману, а туманний вузол відповідає за надання, захист і керування даним давачем.

10.7.4 Різновиди туманної топології

10.7.4.1 Визначальні чинники

Туманна мережа може бути дуже простою, такою як граничний маршрутизатор з підтримкою туману, що з'єднує давачі з хмарним сервісом. Вона також може ускладнюватися до багаторівневої туманної ієрархії з різним ступенем здатності обробки і ролями на кожному рівні, одночасно перерозподіляючи навантаження обробки, коли і де це необхідно (зі сходу на захід і з півночі на південь). Визначальні чинники моделі засновані на наступному:

зменшення обсягу даних до хмари- наприклад, система збирає неструктуровані відеодані з тисяч давачів або камер, агрегує їх і шукає конкретні події в режимі реального часу. Якщо це так, то скорочення набору даних буде значним, так як тисячі камер будуть щодня виробляти сотні гігабайт інформації, а туманним вузлам потрібно буде перевести великі обсяги даних в прості форми: *«так», «Немає», «небезпека», «токени безпеки події»* (приклад на рис.10.17);

кількість граничних пристроїв - якщо система IIoT являє із себе всього лише один давач, то набір даних малий і може не виправдовувати використання граничного туманного вузла. Однак, якщо число давачів зростає або, в ще гіршому випадку, зростання кількості давачів не передбачувано і динамічно у часі, то туманна топологія може зажадати динамічного горизонтального масштабування. Як приклад, можна розглянути стадіон з використанням Bluetooth-маяка. Оскільки аудиторія на ньому може зростати / зменшуватися у часі на певних майданчиках, система повинна мати можливість масштабуватися нелінійно. В інших випадках стадіон може займати лише невелику площу, і йому потрібні тільки обмежені ресурси обробки та підключення (приклад на рис. 10.18);

можливості туманного вузла - в залежності від топології і вартості деякі вузли можуть краще підходити для підключення до *WPAN*, тоді як інші вузли в ієрархії можуть мати додаткові можливості обчислень для машинного навчання, розпізнавання образів або обробки зображень. Прикладом можуть бути граничні туманні вузли, які управляють безпечною мережею *Bluetooth* і мають спеціальне обладнання для аварійних ситуацій або безпеки *WPAN*. Вище цього рівня може існувати ще один туманний вузол, який буде мати додаткову оперативну пам'ять і графічний процесор для підтримки потоків необроблених даних з шлюзів *WPAN* (приклад на рис. 10.20);

надійність системи - архітекторові може знадобитися розгляд різновидів відмов у моделі IIoT. Якщо один крайовий туманний вузол дав збій, інший міг би зайняти його місце для виконання якої-небудь дії або сервісу. Цей випадок стає важливий в життєво-критичних випадках або при роботі в реальному часі. Таким же чином додаткові туманні вузли можуть підключатися на вимогу; надлишкові вузли можуть знадобитися в ситуаціях забезпечення відмовостійкості. У разі відсутності таких надлишкових вузлів деяка обробка може спільно здійснюватися сусідніми вузлами за рахунок використання системних ресурсів і збільшення затримки, але система все ще буде продовжувати функціонувати. Крайовий варіант використання - це той, де сусідні вузли діють як сторожові пси один для одного. У разі збою туманного вузла або збою зв'язку з вузлом сторожовий таймер сигналізує про подію збою

для хмари і може виконувати локальні критичні дії. Хорошим прикладом є випадок, коли туманний вузол дає збої при контролі трафіку на шосе; сусідній вузол може побачити відмову краплі туману, попередити хмару про подію і подати сигнал на щит керування трафіком шосе, щоб зменшити швидкість транспортного потоку.

10.7.4.2 Проста туманна топологія

Найпростішим рішенням для туману є блок обробки на границі (шлюз, тонкі клієнти, маршрутизатор), розташований поруч з матрицею давачів. Тут туманний вузол може використовуватися в якості шлюзу для *мережі* або *mesh-мережі WPAN* і обмінюватися даними з хостом, як на рис. 10.16 нижче:

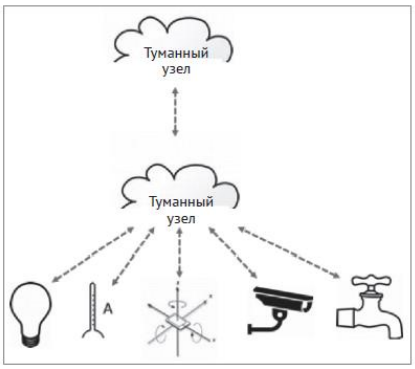


Рисунок 10.16 Гранично-туманний пристрій управляє масивом давачів і може взаємодіяти з іншим туманним вузлом на основі M2M

10.7.4.3 Туман у хмарну топологію

Наступна базова туманна топологія включає хмару як батьківську туманну мережу. Туманний вузол в цьому випадку буде збирати дані, захищати периметр і виконувати обробку, необхідну для зв'язку з хмарою. Цю модель відокремлює від моделі граничних обчислень те, що сервісні і програмні рівні туманного вузла поділяють відповідні обчислення з хмарним фреймворком, як на рис. 10.17 нижче:

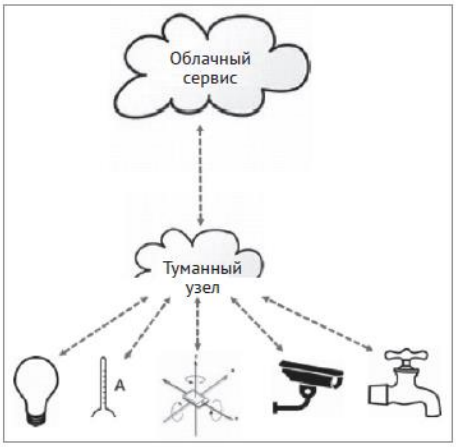


Рисунок 10.17 Тут туманний вузол встановлює відносини з хмарою

10.7.4.4 Декілька туманних вузлів з однією основною хмарою

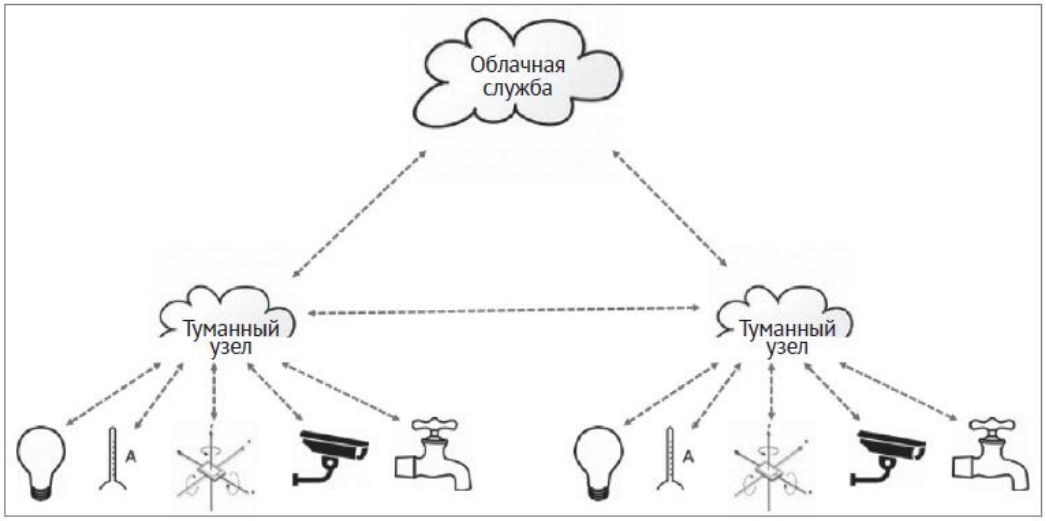


Рисунок 10.18 Декілька туманних вузлів з однією хмарою

Модель на рис. 10.18 використовує кілька туманних вузлів, відповідальних за послуги і обробку на границі, і кожен з них підключається до свого набору давачів. Батьківська хмара забезпечує кожен туманний вузол так, якщо б він був єдиним вузлом. Кожен вузол має унікальний ідентифікатор, щоб забезпечити унікальний набір сервісів на основі свого розташування. Наприклад, кожен туманний вузол може перебувати в іншому місці для роздрібної торгової мережі. Туманні вузли також можуть пов'язувати і передавати дані зі сходу на захід між граничними вузлами. Прикладом можуть служити умови холодного зберігання, коли необхідно підтримувати і управляти вентиляторами і морозильниками, щоб запобігти псуванню продуктів харчування. У роздрібного продавця може бути кілька вентиляторів в різних місцях, всі вони управляються єдиною хмарної службою, але працюють з різними туманними вузлами на границі (рис. 10.18).

10.7.4.5 Декілька туманних вузлів з декількома хмарними провайдерами

Наступна модель розширює топологію п. 10.7.4.4, додаючи можливість надійно і конфіденційно спілкуватися з декількома хмарними провайдерами одночасно декільком туманним вузлам. У цій моделі можуть бути розгорнуті кілька батьківських хмар. Наприклад, *розумні міста* можуть існувати у численних географічних районах, і вони можуть охоплюватися *різними міськдержадміністраціями (МДА)*. Кожна *МДА* може вибрати свого хмарного провайдера, порівнявши його з іншими, але всі *МДА* управляються з використанням одного затвердженого та бюджетного виробника камер і давачів. У цьому випадку виробник камер і давачів матиме один екземпляр хмари, що співіснує з декількома *МДА*. Туманні вузли повинні мати можливість доставляти дані декільком провайдерам хмарних обчислень, як на рис. 10.19 нижче:

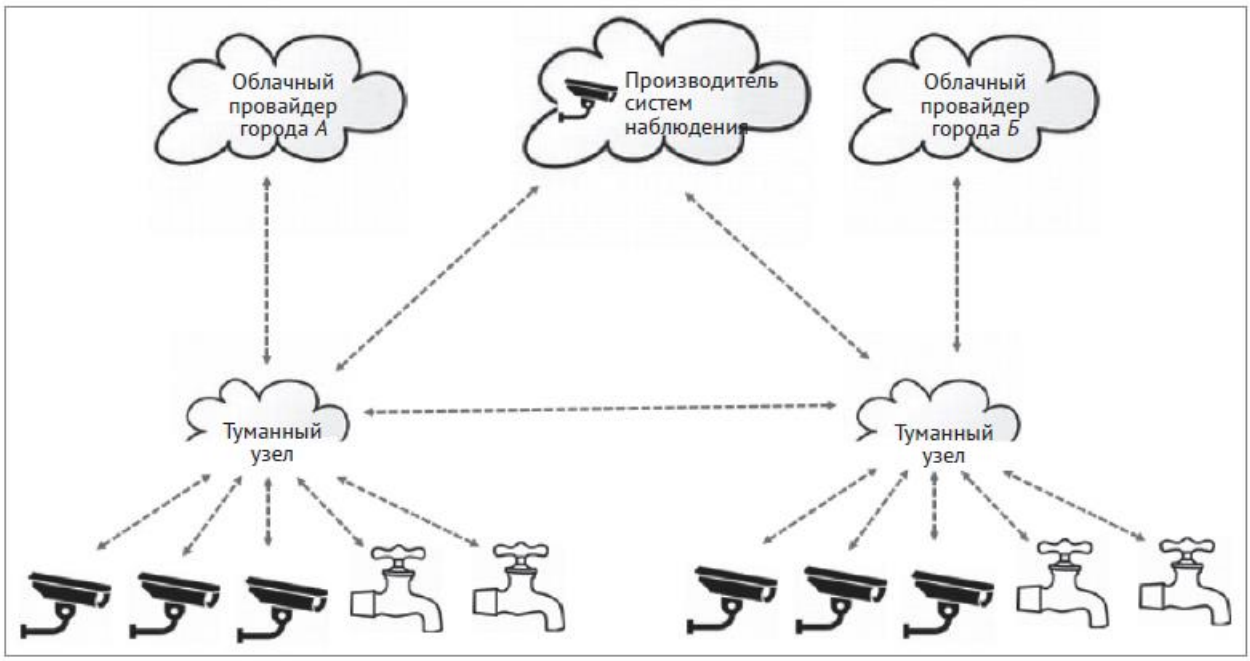


Рисунок 10.19 Кілька туманних вузлів з кількома хмарними провайдерами. Хмари можуть являти собою суміш публічних і приватних хмар

10.7.4.6 Багаторівнева туманна топологія

Туманні вузли також не потребують обов'язкового індивідуального з'єднання, що зв'язує давачі з хмарами *один до одного*. Туманні вузли можуть бути поміщені в стек, багаторівневими або навіть законсервованими до тих пір, поки не знадобляться. Ієрархія рівнів туманних вузлів відносно одне одного може здаватися суперечливою, якщо ми намагаємося зменшити затримки, але,

як згадувалося раніше, вузли можуть бути спеціалізованими. Наприклад, вузли, розташовані ближче до давачів, можуть надавати послуги в режимі реального часу або мати обмеження за вартістю, що вимагають від них мінімального обсягу даних для зберігання і обчислення. Рівні над ними можуть надавати обчислювальні ресурси, необхідні для агрегованого зберігання, машинного навчання або розпізнавання зображень з використанням додаткових запам'ятовуючих пристроїв, або графічних процесорів. Наступний приклад ілюструє таке використання на прикладі реалізації освітлення міста (див. рис. 10.20).

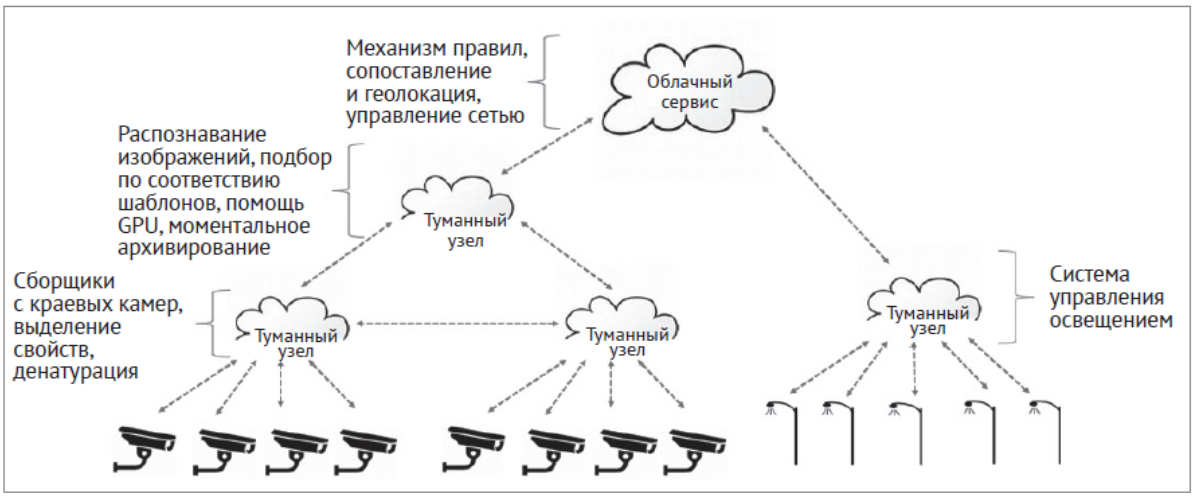


Рисунок 10.20 Багаторівнева туманна топологія. Туманні вузли складаються в стек в ієрархії рівнів, щоб забезпечити додаткові послуги або абстракції

Тут кілька камер відчувають рух пішоходів і транспортний трафік; туманні вузли, які мають найтісніший контакт з камерами, виконують агрегацію і виявлення ознак руху і передають ці дані вгору до наступного рівня. Батьківський туманний вузол отримує ці дані і виконує необхідне розпізнавання зображень за допомогою алгоритму глибокого навчання. Якщо буде спостерігатися цікава подія (наприклад, пішохідна прогулянка вночі вздовж шляху), подія буде відправлена до хмари. Компонент хмари реєструє подію і сигналізує вуличним ліхтарям поблизу пішохода, щоб вони збільшили освітленість. Це буде продовжуватися до тих пір, поки туманні вузли бачать рух пішохода. Кінцевою метою є загальна економія енергії, щоб кожна вулична лампочка не працювала на повну потужність весь свій час:

10.8 Висновки

Аналіз даних і отримання значущих висновків з давачів є метою IIoT. Коли ми масштабуємося до тисяч, мільйонів і потенційно мільярдів об'єктів, які передають і приймають потокові дані без зупинок, ми повинні впроваджувати передові інструменти для отримання, зберігання, передачі, аналізу та прогнозування значень з цього моря даних. Хмарні обчислення - один з елементів, що дозволяють використовувати цю послугу у вигляді кластерів, що масштабують обладнання та програмне забезпечення. Туманні обчислення наближають хмарну обробку ближче до границі для вирішення проблем із затримкою, безпекою та витратами на зв'язок. Обидві технології працюють разом, щоб забезпечити роботу аналітичних пакетів у вигляді рушіїв правил для складних агентів обробки подій. Вибір моделі хмарних провайдерів, фреймворків, туманних вузлів і модулів аналітики є важливим завданням, і безліч матеріалів присвячено глибокому розгляду семантики програмування і створення цих сервісів. Архітектор повинен зрозуміти топологію і кінцеву мету системи, щоб побудувати власну структуру, яка задовольняє сьогоднішні потреби і масштабується в майбутньому.

Хмара, безумовно, може містити ряд аналітичних функцій, однак ми повинні бути готові зрозуміти, що якась частина аналізу повинна виконуватися на границі, ближче до джерела даних (давача) або, якщо це має сенс, у хмарі (використовуючи довгострокові історичні дані).

Контрольні питання

1. Історія розвитку хмарних обчислень. Коли, як і завдяки чому виник термін *хмара*?
2. Перерахувати і охарактеризувати існуючі моделі хмарних сервісів.
3. Які типи моделей розгортання хмарного середовища існують на сьогодні? В яких випадках їх слід застосовувати?
4. Описати модель приватної хмари з наведенням прикладу.

5. Описати модель публічної хмари з наведенням прикладу.
6. Описати модель гібридної хмари з наведенням прикладу.
7. Що таке IBM Cloud та які сервіси для IoT вона має?
8. Для чого застосовується інструмент Node Red?
9. Як забезпечується безпека в IBM Cloude? Якими засобами?
10. Що таке безпека на рівні платформи, функціональна безпека та безпека інфраструктури?
11. Які існують обмеження хмарних архітектур для IoT?
12. Пояснити поняття границі, граничних обчислень та їх відмінність від хмарних обчислень. Що дає використання граничних обчислень при порівнянні з хмарними?
13. Навести і пояснити різновиди топології граничних обчислень.
14. Які висуваються вимоги до граничних обчислень?
15. Як забезпечується безпека граничних обчислень?
16. Що таке оркестрування в граничних обчисленнях?
17. Дати визначення терміну *туманні обчислення*. У чому полягає їх відмінність від граничних?
18. Дати загальне порівняння хмарних, граничних та туманних обчислень.
19. Розповісти про архітектуру туманних обчислень, яку було розроблено консорціумом OpenFog.
20. Які різновиди туманної топології ви знаєте? Які чинники є визначальними для неї?
21. Що таке проста туманна топологія?
22. Що таке туман у хмарну топологію?
23. Описати топологію декілька туманних вузлів з однією хмарою.
24. Описати топологію декілька туманних вузлів з декількома різними хмарами
25. Що таке багаторівнева туманна топологія?

11 ВИКОРИСТАННЯ ТЕХНОЛОГІЙ *BIG DATE* В IIoT

11.1 Що таке *Big Date*

При використанні IIoT технологій в системі може одночасно працювати велика кількість давачів (тисячі, десятки та сотні тисяч), що передають величезну кількість технологічної інформації у реальному часі. Такий величезний обсяг даних, неможливо зберігати і обробляти за допомогою традиційного підходу з використанням класичних технологій БД. Оскільки ці дані можуть містити цінну інформацію, її потрібно встигати обробляти у режимі реального часу. В подальшому, цю цінну інформацію можна використовувати для керування різноманітними процесами, проведення прогностичних аналізів, а також для маркетингових і багатьох інших цілей. Якщо ми будемо використовувати традиційний підхід, ми не зможемо виконати це завдання протягом жорстко заданого часу, оскільки потужності зберігання та обробки не будуть достатніми для цих типів завдань.

Загальноприйняте визначення терміну ***Big Date*** *(великі дані*) виглядає таким чином: *Великі дані (англ. Big data) - серія підходів, інструментів і методів обробки структурованих і неструктурованих даних величезних обсягів і значного різноманіття для отримання кінцевих результатів, які в змозі бути сприйнятими людиною, ефективних в умовах потокової обробки, розподілу по численних вузлах обчислювальної мережі, і є альтернативою традиційним системам управління базами даних і рішень класу Business Intelligence.* Характерною рисою *Big Data* є феноменальне прискорення нагромадження даних та їх ускладнення у процесі роботи.

Ви коли-небудь замислювалися, як YouTube пропонує нам відео, які нас можуть зацікавити? Як Google обслуговує наші запити інформації, і пропонує купу посилань на сайти, які нас можуть зацікавити? Ці компанії зберігають всю діяльність, яку ми виконуємо у своєму браузері, і використовують її для кращого передбачення подальшої поведінки користувача, а також для його вигоди в процесі отримання доходу. Існує багато прикладів такого типу поведінки, і з кожним днем процес наростає, оскільки все більше компаній усвідомлюють силу даних.

Тепер, коли ми маємо певне розуміння того, що є великими даними, ми обговоримо їх різні характеристики.

11.2 Характеристики *Big Date*

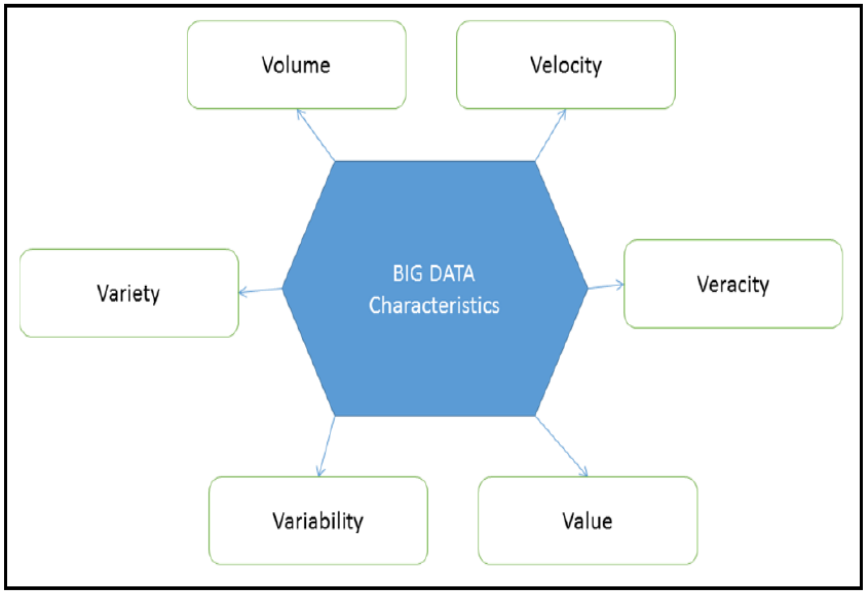


Рисунок 11.1 Характеристики, що описують Big Date

У 2001 році Дуг Лейні вперше представив те, що стало відомо як **три V великих даних**, щоб описати деякі характеристики, які роблять великі дані відмінними від інших даних. Ці три V є **обсягом** (volume), **швидкістю надходження** (velocity) і **різноманітністю** (variety) (показано на рис. 11.1).

На даний час , ці три V стали **шістьма V великих даних** (як на малюнку). Як видно, до характеристик добавлено ще **достовірність** (veracity), **мінливість** (variability) і **цінність** (value). В майбутньому ця кількість характеристик може ще збільшитися.

В таблиці нижче наведено різний розмір пам'яті комп'ютера, щоб дати уявлення про переходи між різними одиницями. Це дозволить нам зрозуміти розмір даних у наступних прикладах:

Таблиця 11.1 Одиниці виміру пам'яті

1 біт	двійкова одиниця
8 біт	1 байт
1024 байт	1 KB (кілобайт)
1024 KB	1 MB (мегабайт)
1024 MB	1 GB (гігабайт)
1024 GB	1 TB (терабайт)
1024 TB	1 PB (петабайт)
1024 PB	1 EB (ексабайт)
1024 EB	1 ZB (зеттабайт)
1024 ZB	1 YB (йоттабайт)

1024 YB	1 бронтобайт
1024 бронтобайт	1 геопбайт

11.2.1 Обсяги даних

У попередні роки дані компанії стосувалися лише даних, створених їх співробітниками. Тепер, коли з кожним днем збільшується використання все нових і нових технологій, це не тільки дані, створені працівниками, а й дані, створені машинами, що використовуються компанією та її клієнтами. Крім того, з розвитком соціальних медіа та інших інтернет-ресурсів користувачі розміщують та завантажують багато додаткового вмісту - відео, фотографії, твіти, тощо. Достатньо просто уявити населення планети 7 мільярдів, і майже 6 мільярдів з них мають стільникові телефони (смартфони). Сам смартфон містить десятки давачів, таких як акселерометр, освітленності, декілька відеокамер, тощо, які генерують дані для кожної події, яку вони збирають і аналізують.

Коли ми говоримо про обсяг у великому контексті даних, це велика кількість даних, яка є **масовою** по відношенню до системи обробки, і яка не може бути зібрана, збережена і оброблена з використанням традиційних підходів. Це дані, які вже зібрані, і *потокові дані*, які постійно генеруються.

Якщо взяти за приклад *Facebook* (на початок 2018 року), то вони мали 2 мільярди активних користувачів. Відповідно до статистики, наданої *Facebook,* щодня 600 ТБ даних потрапляють до бази даних *Facebook*. Наведений нижче графік відображає дані, які існували в світі в попередні роки, поточну ситуацію та прогнозну ситуацію для майбутнього (рис. 11.2):

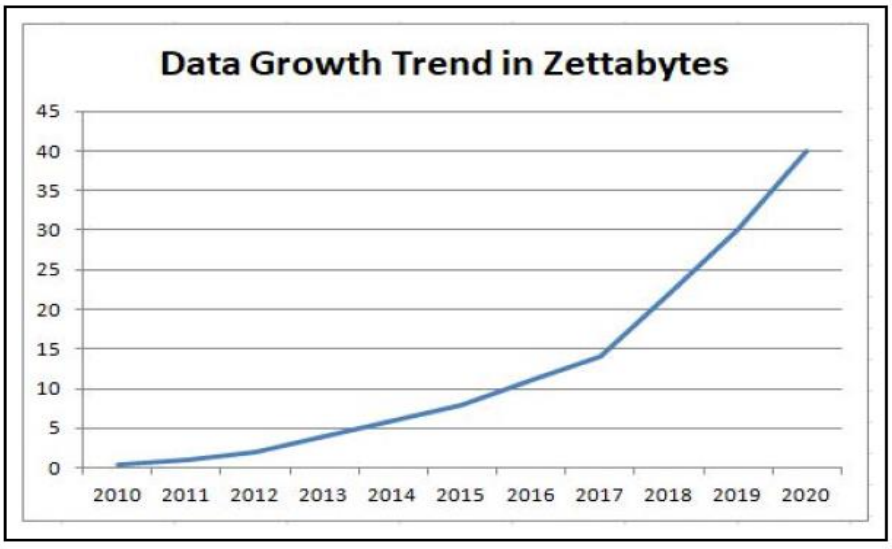


Рисунок 11.2 Тенденція зростання даних у світі в зетабайтах

Візьмемо ще один приклад реактивного літака. Статистика показує, що він генерує 10 ТБ даних за кожну годину польоту. А тепер уявимо тисячі рейсів авіакомпанії кожен день на протязі року. Кількість генерованих даних може досягати багатьох петабайт кожен день. Одне з досліджень показує, що до 2020 року в світі було створено 40 зетабайт даних.

Не так давно генерування такої масивної кількості даних вважалося проблемою, оскільки вартість зберігання була дуже високою. Але тепер, коли вартість зберігання зменшується, це вже не проблема. Крім того, такі рішення, як *Hadoop* і різні алгоритми, що допомагають в прийомі і обробці цієї масивної кількості даних роблять її навіть швидкою.

11.2.2 Швидкість надходження (обробки) даних

Швидкість - це оцінка темпу, з яким генеруються дані, або стрімкість надходження даних. Простішими словами, ми можемо назвати їх даними у русі, або **потоковими даними**. Уявіть, яку кількість даних отримують щодня на *Facebook*, *YouTube* або на будь-якому іншому сайті соціальної мережі. Вони повинні зберігати їх, обробляти, а пізніше і якось їх відновлювати (рис. 11.3).

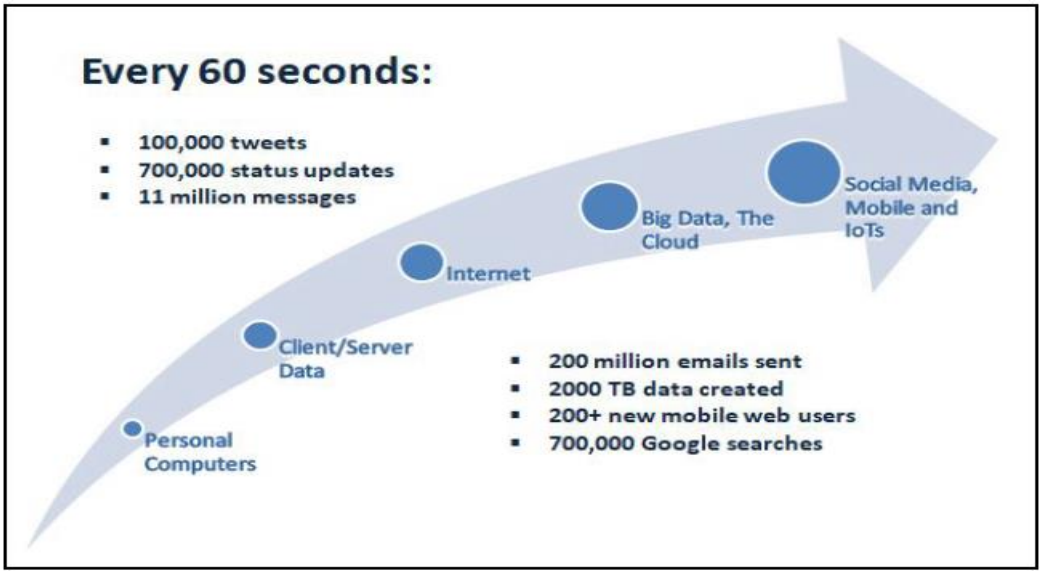


Рисунок 11.3 Швидкість генерації даних

Нижче наведено кілька прикладів того, як швидко збільшуються дані:

- на кожній торговельній сесії Нью-Йоркська фондова біржа фіксує 1 Тб даних;
- кожну хвилину на YouTube завантажується 120 годин відео;

- дані, що генеруються сучасними автомобілями: походять від майже 100 давачів для моніторингу кожного елемента, від палива і тиску з урахуванням навколишньої обстановки біля автомобіля;
- 200 мільйонів повідомлень надсилаються щохвилини.

Іншим виміром швидкості обробки є період часу, протягом якого дані будуть мати сенс і будуть цінними. Чи будуть вони старіти і втрачати цінність з часом, або вони будуть постійно цінними? Цей аналіз також дуже важливий, тому що якщо дані старіють і втрачають цінність з плином часу, то можливо, їх використання пізніше може ввести в оману.

11.2.3 Різноманіття даних

Рисунок 11.4 Різноманіття даних

Ця характеристика відноситься до класифікації даних. Це можуть бути структуровані або неструктуровані дані. Структуровані дані звісно є більш інформативними, і мають заздалегідь визначену схему, або модель даних з усіма попередньо визначеними стовпцями (**атрибутами**), типами даних і так далі, тоді як неструктуровані дані не мають жодної з цих характеристик. До неструктурованих можна віднести довгий список таких даних, як документи, електронні листи, текстові повідомлення в соціальних мережах, відео, фотографії, аудіо, графіки, **усі типи машинно-генерованих даних з давачів, пристроїв, міток RFID, машинних журналів, сигнали GPS**, і багато іншого.

Це все приклади створення неструктурованих даних, які необхідно обробити або для кращого використання, або для отримання прибутку самій компанії.

11.2.4 Достовірність даних

Ця характеристика стосується невизначеності даних через низьку якість самих даних або через їх зашумленість. Це типова людська поведінка - не довіряти наданій інформації, кожен третій керівник бізнесу не довіряє інформації, яку він використовує при прийнятті рішень.

Достовірність випливає з невизначеності даних. Візьмемо, наприклад, яблука, де ми повинні вирішити, чи є вони якісними. Можливо, деякі з них мають середню або погану якість. Після того, як ми почнемо перевіряти їх у величезних кількостях, можливо, наше рішення буде базуватися на стані більшості, і на цій основі ми зробимо припущення щодо решти, тому що якщо ми станемо перевіряти кожне яблуко, то вся партія яблук може втратити свіжість. Наступна діаграма (рис. 11.5) ілюструє цей приклад:

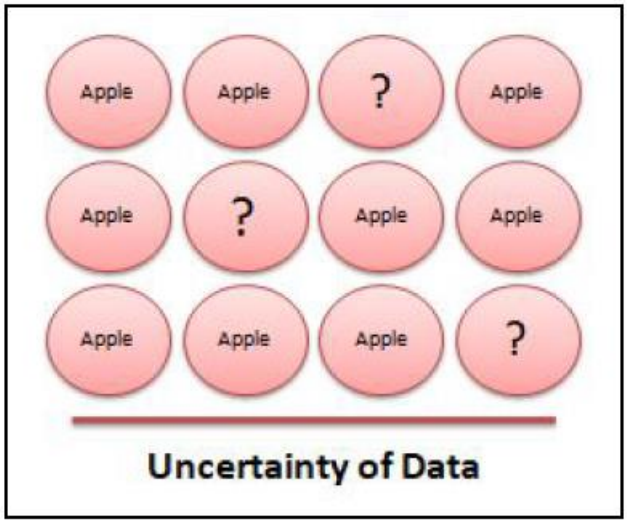


Рисунок 11.5 Достовірність: ілюстрація невизначеності на прикладі з яблуками

Основна проблема полягає в тому, що ми не маємо часу для повного очищення потокових даних з ціллю повного усунення невизначеності. Такі дані, що описують події і генеруються машинами та давачами, дуже швидко втрачають свою цінність, і якщо ми станемо чекати повного очищення та обробки, ці дані можуть втратити значення. Таким чином, ми змушені обробляти їх так, як є, з урахуванням невизначеності.

Достовірність стосується невизначеності та степені довіри до даних, але коли ми використовуємо її з точки зору контексту великих даних, можливо, доведеться перевизначити дані, яким є сенс довіряти. Довіра до даних впливає на цінність та вплив остаточних рішень, які приймає керівництво.

11.2.5 Мінливість даних

Ця характеристика великих даних походить від відсутності узгодженості або фіксованості моделей даних. Вона відрізняється від різноманітності. Візьмемо приклад магазину, що торгує смаколиками. Вони можуть мати безліч

різних смаків. Тепер, якщо ми їмо один і той же смаколик кожен день, і при цьому виявляємо, що він відрізняється за смаком кожного разу, це мінливість. Те ж саме відноситься і до даних - якщо зміст і розуміння даних постійно змінюється, це матиме величезний вплив на наш аналіз і спроби виявити закономірності.

11.2.6 Цінність даних

Це найбільш важливий вектор з точки зору великих даних, але не сильно пов'язаний з самими великими даними – цінність однаково справедлива і для малих даних. Після вирішення всіх інших питань: обсягу, швидкості обробки, різноманітності, достовірності, що вимагає багато часу, зусиль і ресурсів, тепер прийшов час вирішити, чи варто зберігати ці дані в самій інфраструктурі, або в окремому приміщенні, або навіть у хмарі. Одним з аспектів цінності є те, що ми повинні зберігати величезну кількість історичних даних, перш ніж зможемо використати їх для того, щоб нарешті надати цінну інформацію на їх основі. Раніше зберігання такого обсягу даних було неможливим через необхідність величезних витрат, але тепер технологія зберігання та пошуку стала набагато дешевшою. Ми хочемо бути впевненими, що наша організація отримує користь від зберігання такої кількості даних. Подальший аналіз даних має виконуватися і з урахуванням етичних міркувань.

Тепер, коли ми визначили і зрозуміли всі шість V великих даних, настав час розширити нашу сферу розуміння і з'ясувати, що робити з даними, що мають такі характеристики. Компанії все ще можуть вважати, що їхні традиційні системи є достатніми для даних, які мають такі характеристики, але якщо вони залишаються під цим впливом, вони можуть втратити в довгостроковій перспективі. Після розуміння важливості даних та їх характеристик, основний фокус повинен бути спрямовано на те, як їх зберігати, як обробляти, які саме дані зберігати, як швидко очікується результат аналізу і так далі. Різні рішення для обробки даних цього типу, кожен зі своїми власними плюсами і мінусами, доступні на ринку, а нові розробляються постійно. Архітектор даних повинен

пам'ятати ключові моменти у прийнятті рішень, які в кінцевому підсумку змусять прийняти одне з них і залишити інші.

11.3 Термінологія, що використовується в *BIG DATA*

Великі дані (Big Data) - дані, які є великими за обсягом по відношенню до системи обробки, з безліччю структурованих і неструктурованих даних, що містять різні шаблони даних для аналізу.

Пакетна обробка (batch process) - процес аналізу великих наборів даних, який, як правило, заплановано і виконується тоді, коли не виконуються інші процеси. Як правило, це ідеально підходить для роботи, яка не залежить від часу і працює на дуже великих наборах даних. Як тільки процес закінчиться виконанням завдання, він поверне результати, зазвичай як вихідний файл або як запис бази даних.

BI (Business intelligence)— це збирання, зберігання і аналіз даних що утворюються при діяльності організації. Метою BI є підтримка прийняття кращих управлінських рішень.

Кластерні обчислення - це практика поєднання ресурсів декількох недорогих технічних засобів (обчислювачів) та управління їхніми спільними можливостями обробки та зберігання інформації для виконання різних завдань. Їх виконання вимагає програмного рівня для обробки зв'язку між різними окремими вузлами з метою ефективного управління та координації виконання призначеної для них роботи.

Сховище данних (Data warehouse) - велике сховище структурованих даних для цілей аналізу та звітності. До нього входять лише данні, які вже очищені, з певною схемою і добре інтегровані у власні джерела. Використовуються та згадуються в контексті традиційних систем, таких як BI.

Озеро даних (data lake) - подібне до сховища даних, для зберігання великих наборів даних, але разом з структурованими містить і неструктуровані дані. Це часто використовуваний термін у контексті великих даних, що зберігають таку інформацію, як історичні дані, блоги, публікації, відео, фотографії та багато іншого.

Data mining - інтелектуальний аналіз даних. Це процес перетворення великого масиву даних в більш зрозумілий і наочний формат. Є широким терміном для практики пошуку прихованих шаблонів у великих наборах даних.

ETL (extract, transform, and load - вилучення, перетворення та завантаження) - в основному стосується традиційних систем, таких як BI, які беруть необроблені дані та обробляють їх для аналітичних та звітних цілей. В основному термін пов'язано зі сховищами даних, але характеристики цього процесу також зустрічаються і у потоках прийому великих систем даних.

Hadoop - це програмна платформа з відкритим кодом для обробки дуже великих наборів даних у розподіленому середовищі для зберігання та об'єднання обчислювальної потужності дешевих апаратних ресурсів. Вона розроблена для легкого масштабування від декількох до тисяч серверів. Вона допомогає оброблювати локально збережені дані у загальній паралельній обробці. Вона включає до себе різні модулі - *Hadoop Distributed File System (HDFS), Hadoop MapReduce* і *Hadoop YARN* (*Yet Another Resource Negotiator* - посередник ресурсів).

Обчислення в пам'яті (in-memory computing)- стратегія, яка передбачає перенесення робочих наборів даних повністю в загальну пам'ять кластера замість читання з жорсткого диска, щоб скоротити час обробки, не використовуючи операції вводу-виводу. Проміжні розрахунки теж не записуються на диск і замість цього зберігаються в пам'яті. Це фундаментальна ідея таких проектів, як Apache Spark. Через це, він має величезні переваги в швидкості I/O над такими системами як MapReduce.

Машинне навчання (machine learning, ML) - дослідження, що передбачає розробку системи, яка може навчатися, не будучи явно запрограмованою. Вона може мати можливість коригувати і вдосконалювати себе на основі даних, що надходять до неї. Вона передбачає реалізацію алгоритмів прогнозування та статистики, які можуть постійно коригувати правильну поведінку та розуміння тим більш, чим більше даних проходить через систему.

MapReduce - це фреймворк для обробки будь-якого завдання в розподіленому середовищі. Він працює за принципом *Master/Slave*, подібним до HDFS. Він включає розміщення набору завдань на різних вузлах, що доступні в

кластерному обчислювальному середовищі, і виробляє проміжні результати. Потім він перетасовує результати для вирівнювання наборів даних, а потім зменшує їх, створюючи єдине значення для кожного набору.

NoSQL - забезпечує механізм для зберігання і вилучення даних, які не знаходяться в табличній формі (тобто такі, що зберігаються в системах реляційних баз даних). Її також називають не-SQL, або нереляційною базою даних. Вона добре підходить для великих даних, оскільки в основному використовується з неструктурованими даними і може працювати в розподілених середовищах.

Потокова обробка (stream processing)- практика обробки окремих елементів даних при їх переміщенні через систему. Це допомогає аналізувати дані в режимі реального часу, коли вони надходять у систему, тим самим спрощуючи виконання операцій, які чутливі до часу.

11.4 Платформа *Apach Hadoop*

Apache Hadoop - це програмна платформа з відкритим кодом, яка обробляє дуже великі розподілені набори даних у середовищі з багатьох обчислювачів, які в основному побудовані на дешевих апаратних платформах. Вона з'явилася завдяки публікації системи *Google File System*, опублікованій у жовтні 2003 року. Інший документ *Google MapReduce* розглядає спрощення обробки даних у великих кластерах.

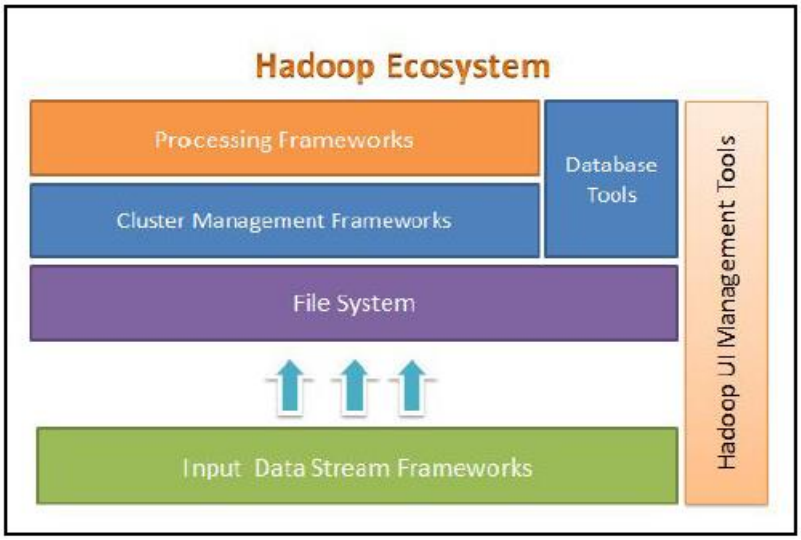


Рисунок 11.6 Архітектура Hadoop

Спершу виник дуже добре масштабований і розширюваний проект з відкритим кодом *Apache Nutch*, в якому було реалізовано об'єкт *MapReduce* і розподілену файлову систему на основі дослідницької роботи *Google*. Ці об'єкти пізніше були оголошені як підпроект під назвою *Apache Hadoop*.

Apache Hadoop розроблений для простого масштабування від декількох до тисяч серверів. Це допомогає компаніям паралельно обробляти локально збережені дані. Одна з переваг *Hadoop* полягає в тому, що він обробляє відмови на рівні програмного забезпечення. Рис. 11.6 ілюструє загальну архітектуру екосистеми *Hadoop* і де в ній знаходяться різні фреймворки.

В екосистемі *Hadoop*, *Apache Hadoop* забезпечує структуру для шару файлової системи, шару керування кластером і шару обробки. Це залишає можливість для інших проектів і фреймворків працювати разом з екосистемою *Hadoop*, А також, розроблювати власні фреймворки для будь-якого з шарів, доступних в системі.

Apache Hadoop складається з чотирьох основних модулів. Ці модулі - *Hadoop Distributed File System* (шар файлової системи), *Hadoop MapReduce* (який працює як з управлінням кластером, так і як шар обробки даних), *Yet Another Resource Negotiator* (YARN, шар управління кластером) і *Hadoop Common*.

11.5 Розподілена файлова система *HDFS*

11.5.1 Базова архітектура *HDFS*

Hadoop Distributed File System (HDFS) - це файлова система, подібна до *Windows* (FAT або NTFS) і *Linux* (swap, ext3, тощо), Спеціально розроблена для розподілених систем, які потребують обробки великих наборів з високою доступністю. Вона оптимізована для роботи з безліччю вузлів і розповсюдження даних між ними в кластерному обчислювальному середовищі для забезпечення максимальної доступності даних. Основна відмінність між *HDFS* та іншими розподіленими файловими системами полягає в тому, що *HDFS* призначений для роботи на недорогих серверних засобах і є дуже стійким до відмов. Базова архітектура показана на рис. 11.7.

HDFS має два основні компоненти, які в основному працюють в концепції *master/slave*. Ці два компоненти називаються *NameNode*, який є головним вузлом, і *DataNode*, який є підлеглим вузлом.

NameNode - це ключовий вузол для всієї файлової системи. Він відповідає за поширення даних через різні *DataNode* на основі певної конфігурації. Він містить і керує метаданими блоків даних, доступних для різних вузлів даних.

Метадані - це інформація, пов'язана з файлами, що зберігаються в системі, наприклад, ім'я файлу, розмір файлу, кількість блоків, на які було розділено файл, інформація про те, де блоки файла зберігаються в *DataNodes*, тощо.

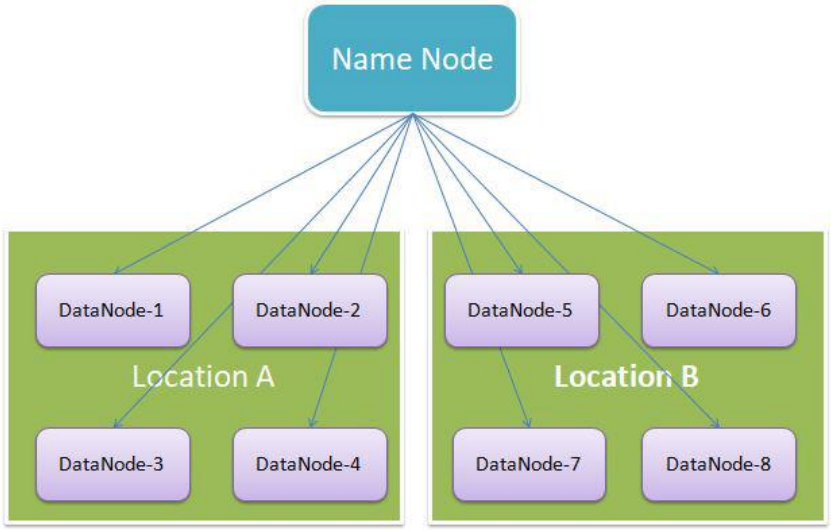


Рисунок 11.7 Базова архітектура HDFS

Розглянемо, що може статися, якщо ми втратимо ці метадані. Чи зможемо ми все-таки отримати всю інформацію та файли, які зберігаються в *DataNodes*? Відповідь: ні. Це пояснюється тим, що ми не знаємо, який блок даних належить файлу, і навіть не знаємо, скільки блоків даних входить в даний файл. Без цієї інформації метаданих вся система *HDFS* розвалиться; тому *NameNode* вважається єдиною точкою відмови.

Щоб подолати цю проблему, рекомендується саме для *NameNode* мати надійне серверне обладнання. Якщо ми станемо використовувати звичайні офісні комп'ютери для роботи *NameNode*, вірогідність відмов збільшиться. Але архітектура *HDFS* дозволяє використовувати звичайні комп'ютери під *DataNode*, тому що ми реплікуємо дані на три або більше вузлів даних. Якщо будь-який вузол даних виходить з ладу, ми все одно зможемо витягти інформацію з іншого вузла даних (як мінімум, з двох). Однак у випадку *NameNode* інформація не реплікується ніде - тому вихід з ладу вузла *NameNode* призведе до відмови усієї системи.

Існують певні конфігураційні моменти, про які ми повинні піклуватися під час налаштування *HDFS*, наприклад, скільки разів необхідно реплікувати дані, визначити розмір блоку даних, тощо. Розмір блоку за замовчуванням у *HDFS*

становить 64 МБ. Цей розмір може бути змінено на 128 МБ, 256 МБ і так далі. Як правило, в інших файлових системах розмір блоку даних є відносно низьким - 4 КБ або 8 КБ, але *HDFS* призначений для обробки великих наборів даних. Якщо ми зменшимо розмір блоку даних до декількох кілобайт або мегабайт, інформація у метаданих збільшиться, що, у свою чергу, збільшує навантаження на *NameNode* під час отримання інформації про файл. Одночасно збільшуються накладні витрати. Іншими словами, коли нам потрібно виконати завдання або програму, потрібно буде більше пересилань даних, мережевої активності та обробки.

DataNode відповідає за фрагмент великого набору даних, зазвичай відомий як блок даних. Керування локальним сховищем виконує *NameNode*. Конфігурація *HDFS* за замовчуванням копіює кожен блок даних у три різні вузли *DataNode*, один на одному сервері і дві копії на інших. Це призводить до 100% доступності даних тому, що навіть якщо одна копія даних стає непрацездатною, інші дві копії будуть доступні. Слід мати на увазі, що хоча *HDFS* і має вбудований механізм відмовостійкості, він не зможе передбачити людські помилки, такі як копіювання або видалення даних. Як профілактика цьому, слід використовувати додаткові резервні копії в іншому місці.

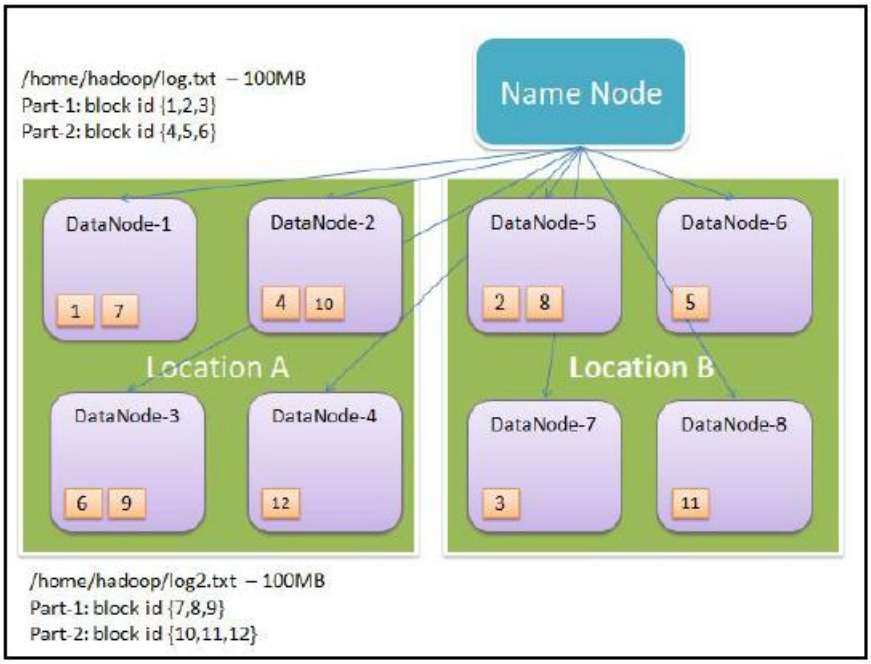


Рисунок 11.8 Приклад реплікації файлів в HDFS на декілька вузлів

Інформація метаданих використовується для зберігання інформації про файл для легкого пошуку та обробки. Ілюстрація механізму зберігання даних в різних *DataNodes*, показана на рис. 11.8 вище.

На рис. 11.8 два файли, *log1.txt* і *log2.txt*, зберігаються в *HDFS*. Вони обидва мають розмір в 100 МБ. Тепер, оскільки розмір блоку за замовчуванням в *HDFS* становить 64 МБ, ці файли будуть розділені на дві частини, і, як визначено за замовчуванням, кожен блок файлу даних буде репліковано на три вузли *DataNode*. *NameNode* призначить ідентифікатор *BlockID* кожному блоку даних. Ця інформація метаданих буде зберігатися в самому *NameNode*, а фактичний блок даних буде перенаправлено в *DataNodes* для подальшого зберігання.

Крім самого управління метаданими, *NameNode* також відповідає за звіти про стан кожного *DataNode*. Це робиться за допомогою технології *heartbeat* (серцебиття). *DataNode* посилає пакети *heartbeat* через регулярний інтервал часу, щоб повідомити *NameNode*, що він в робочому стані. Час інтервалу за замовчуванням - 3 секунди. Якщо будь-який вузол даних не може відправити *серцебиття, NameNode* буде ще чекати до 10 пропущених пакетів *heartbeat*, тобто 30 секунд, перш ніж згенерувати попередження. *NameNode* генеруватиме попередження, щоб замінити і позначити цей *DataNode* як перебуваючий в стані відмови, поки він не буде замінений або відлагоджений. Якщо в майбутньому буде здійснено спробу звернутися до блоку даних на несправному вузлі, *NameNode* вкаже на його наступну доступну копію на іншому вузлі.

NameNode також полегшує механізм балансування навантаження. Таким чином, якщо будь-який *DataNode* вже перевантажено, *NameNode* призначить завдання наступному доступному *DataNode*, який містить необхідні блоки даних. Це допоможе швидше обробляти великий набір даних і використовувати максимальну пропускну здатність доступних серверів.

11.5.2 Виконання запису даних в HDFS

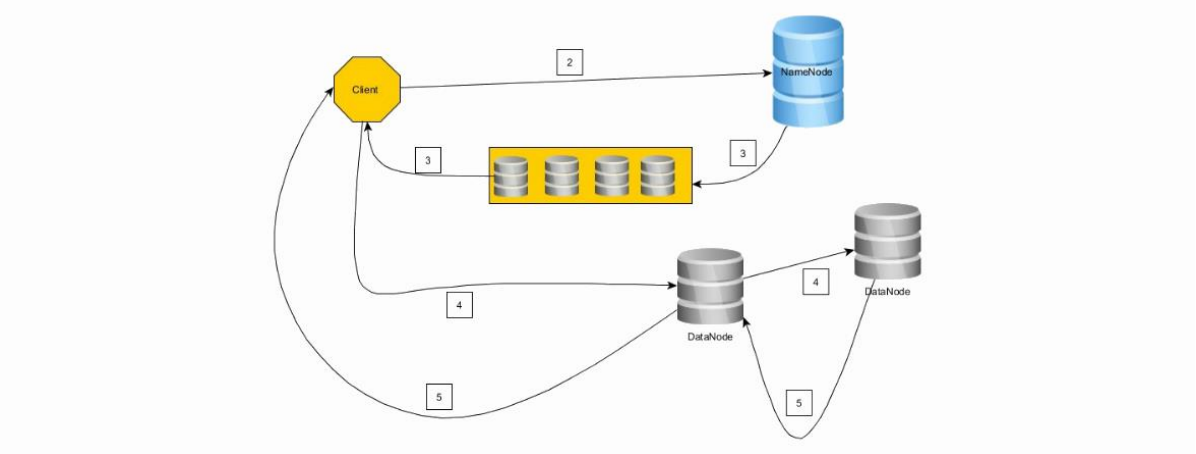


Рисунок 11.9 Виконання запису даних в HDFS

1. Клієнт розрізає файл на ланцюжки блокового розміру.

2. Клієнт з'єднується з *NameNode* і запитує операцію запису, надсилаючи кількість блоків і необхідний рівень реплікації.

3. *NameNode* відповідає ланцюжком з *DataNode*.

4. Клієнт з'єднується з першим вузлом ланцюжка (якщо не вийшло з першої, з другої, тощо спроб. Не вийшло зовсім - відкат). Клієнт робить запис першого блоку на перший вузол, перший вузол далі самостійно реплікує його на другий і далі по ланцюжку.

5. Завершивши цикл запису, в зворотному порядку (4 -> 3, 3 -> 2, 2 -> 1, 1 -> клієнту) надсилаються повідомлення про успішний запис.

6. Як тільки клієнт отримає підтвердження успішного запису блоку, він сповіщає *NameNode* про запис блоку, потім отримує ланцюжок *DataNode* для запису другого блоку, тощо.

Клієнт продовжує записувати блоки, поки не запише успішно кожен блок хоча б на один вузол.

11.6 Hadoop MapReduce

Hadoop надає фреймворк *MapReduce* для полегшення виконання програм і паралельної обробки. Рисунок 11.10 ілюструє, як фреймворк *MapReduce* вписується в архітектуру *Hadoop*. Цей фреймворк в основному має у своєму складі дві частини: *Map* і *Reduce*. Процес *Map* переважно складається з отримання інформації із збережених даних, застосування необхідного алгоритму і генерування результату у вигляді пар ключ-значення. Процес *Reduce* використовується для узагальнення інформації, зібраної та об'єднаної в пари під час процесу *Map*.

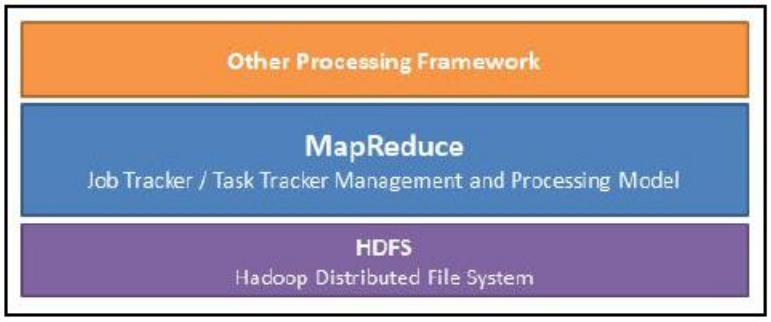


Рисунок 11.10 Місце MapReduce в архітектурі Hadoop

11.6.1 *Job Tracker* та *Task Tracker*

Подібно до *HDFS*, у виконанні будь-якої програми беруть участь два основні процеси, які також працюють за концепцією *Master/Slave*. Це *Job Tracker* (відстеження завдань) і *Task Tracker* (відстеження потоків). *Job Tracker* - це основний процес, який отримує від клієнта запити на виконання будь-якої програми. Потім він отримує інформацію про метадані від *NameNode* і призначає їх відповідним *Task Tracker* для подальшого виконання. Рисунок 11.11 ілюструє загальну архітектуру *MapReduce*.

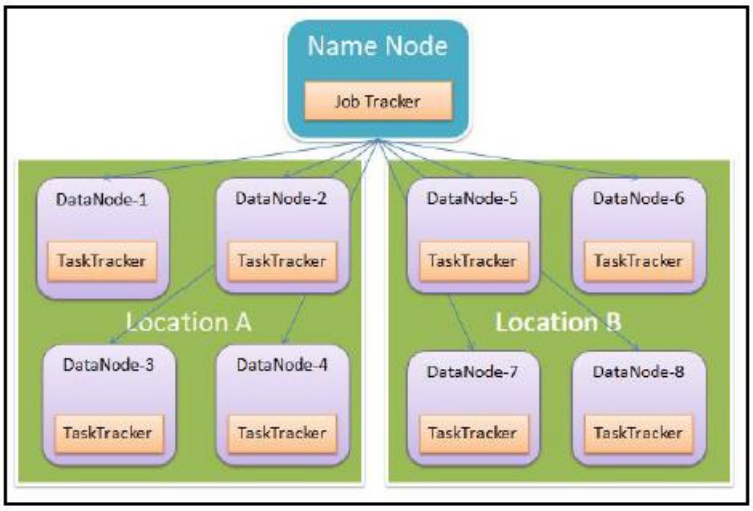


Рисунок 11.11 Архітектура MapReduce

Процес *Job Tracker* є критичним процесом, оскільки він управляє інформацією про всі виконання програми в повному кластері *Hadoop*. Ось чому цей процес знаходиться в *NameNode*. *Task Trackers* керуються *Job Tracker* і отримують від нього регулярні оновлення для виконання. Як частина конфігурації за замовчуванням, *Task Tracker* відправляє пакети

серцебиття кожні 3 секунди, і *Job Tracker* буде чекати до 30 секунд (10 пакетів), перш ніж позначити цей *Task Tracker* як неробочий. У такому разі це завдання буде перепризначено *Task Tracker* іншого вузла даних, який містить ті ж дані.

Job Tracker також відповідає за балансування навантаження між різними *DataNode*. Наприклад, якщо будь-який вузол даних вже виконує якусь програму обробки даних і з'являється новий запит від іншої програми для доступу до цих даних, *Job Tracker* призначить новий *Task Tracker* реплікованому вузлу даних, щоб уникнути перевантаження вузла і отримати максимальний ефект від паралельної обробки.

11.6.2 Виконання потоку *MapReduce*

У традиційній системі, коли клієнтська програма виконується, вона отримує дані, які потім будуть доступні в системі клієнта для подальшої обробки. У системі *Hadoop* програма-обробник буде сама відправлена до потрібних їй даних, так що якщо дані знаходяться в декількох *DataNode*, копії програми будуть відправлені до кожного вузла, що робить виконання всього процесу паралельним. Саме тут і вбудовується виконання фреймворку *MapReduce*, і саме він визначає, як буде виконуватися програма-обробник, який буде потік даних, і як саме буде генеруватися результат.

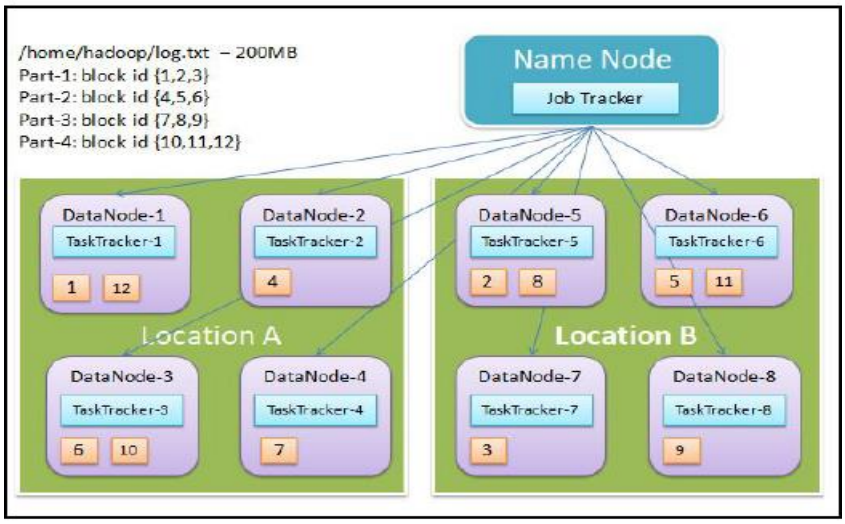


Рисунок 11.12 Зберігання файлу log.txt розміром 200 МБ на вузлах DataNode

Job Tracker призначає завдання *Task Tracker*, перш ніж розпочнеться обробка на вузлах даних. Припустимо, у нас є файл з ім'ям *log.txt*, розмір якого становить 200 МБ (рис. 11.12). Відповідно до конфігурації за замовчуванням, він буде розділений на чотири блоки даних; три з них становитимуть 64 МБ, а четвертий - 8 МБ. Рис. 11.11 ілюструє обробку цього файлу на різних *DataNode*.

Одиниця роботи *job* - набір *map* (паралельна обробка даних) і *reduce* (об'єднання результату роботи *map*) завдань. *Map*-завдання виконують mapper'и, *reduce* - reducer'и. Job складається мінімум з одного mapper'а, reducer'и опційні.

11.6.2.1 Mapper

На цій стадії дані попередньо обробляються за допомогою функції *map()*, яку визначає сам користувач. Робота цієї стадії полягає в передобробці і фільтрації даних. Функція *map()* застосовується до кожного вхідного запису з даними і видає на виході безліч пар <*ключ-значення*>. Під *безліч* розуміється той факт, що функція може видати тільки одну пару, може не видати нічого, а може видати кілька пар <*ключ-значення*>. Що буде знаходитися в ключі і в значенні ключа - вирішувати користувачу, але ключ - дуже важлива річ, так як дані з одним ключем в майбутньому потраплять в один екземпляр функції *reduce*.

Mapper - це процес генерування результатів у формі пар <*ключ-значення*> на основі наданих вхідних даних. Він приймає вхідні пари у вигляді <ключ, значення> і виводить їх так само, але дані вже можуть відрізнятися іншим типом даних.

Кількість *Mapper* залежить від кількості **Вхідних Розділів** (*Input Split),* задіяних у збереженні даних. Припустимо, що є чотири вхідних розбиття - у процесі обробки будуть задіяні чотири маппери. На рис. 11.13 показано, що відбувається у функції *Mapper*:

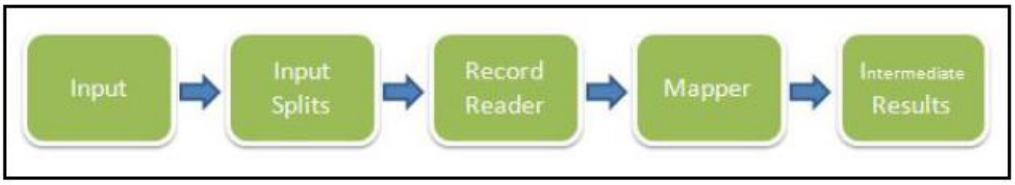


Рисунок 11.13 Послідовність роботи Mapper

Засіб *Record Reader* відповідає за перетворення даних у пари *ключ-значення* для *Mapper*. Коли програма виконується, *Mappers* буде працювати на *DataNode-1,…, DataNode-8* (див. рис. 11.12), щоб скористатися перевагою паралельної обробки; це відбувається тому, що незалежні *DataNode* містять різні блоки даних одного і того ж файлу. Потім *Mapper* формує результат своєї роботи у вигляді пар *ключ-значення*. Цей результат називається **Проміжним результатом** (*Intermediate Results*).

11.6.2.2 Тасування та Сортування

Між етапами "*Map*" та "*Reduce*" є етап, який називається *Shuffle* (тасування). Це проміжний процес агрегації результатів після фази *Mapper* для функції *Reduce*. Всі *Mappers* були виконані незалежно на різних *DataNode* і були отримані результати у вигляді пар ключових значень. Крок *Shuffle* не буде чекати, поки *Mappers* повністю закінчать свою роботу і починають обробляти вже наявні дані від *map()* ще у той час, коли вони ще створюються. Незважаючи на те, що *Mapper* ще не завершив свою роботу, *Shuffle* почне трансформувати дані.

Shuffle працює непомітно для користувача. На цій стадії результати роботи функції *map()* «розбираються по кошиках» - причому кожен кошик відповідає одному ключу виведення стадії *map*. Надалі ці кошики послужать входом для *Reduce*.

Пара *ключ-значення проміжного результату* буде відсортована перед передачею на *Reduce*. Це сортування виконується тільки для ключів, а не для їх значень. Далі відсортовані значення будуть передані в *Reduce* як є. Рисунок 11.14 ілюструє цей процес:

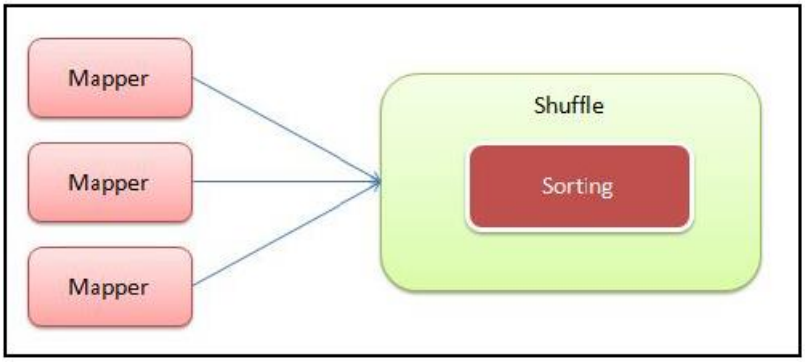


Рисунок 11.14 Робота тасування та сортування ключів

11.6.2.3 Reducer

Кожен кошик із значеннями, сформованими на стадії *Shuffle*, потрапляє на вхід функції *Reduce()*. Функція *Reduce* також задається користувачем і обчислює фінальний результат для окремого кошика. Безліч всіх значень, повернутих функцією *Reduce()*, і буде являти собою фінальний результат *MapReduce*-завдання. *Reduce* в основному виконує функцію агрегації пар *ключ-значення* для різних обчислень залежно від мети програми. Агрегація може бути будь-якого

типу, включаючи підсумовування, множення і так далі, як показано на наступному малюнку:

Рисунок 11.15 Агрегація за допомогою Reduce

Нарешті, результат виконання програми записується до файлової системи HDFS. Варто зазначити, що кількість *Reducer* налаштовується в системі *Hadoop*; за замовчуванням є один.

11.7 Компонента *YARN*

11.7.1 Призначення та архітектура *YARN*

В *MapReduce* ми бачимо службу, яка називається *Job Tracker*. Засіб відстеження завдань виконує такі дії:

- отримує інформацію від *NameNode*, щоб з'ясувати, де знаходяться дані, які будуть потрібні клієнтській програмі;
- координує з відповідними трекерами призначення завдань;
- контролює життєвий цикл роботи *Task Tracker*.

У звичайному сценарії цілком прийнятним є те, що одна машина може виконувати усі ці дії. Але у нашому випадку усе це відбувається у середовищі великого кластеру, де працюють тисячі обчислювачів. Це означає, що *Job Tracker* буде виконуватися одночасно сотні разів, що стане тягарем для *NameNode*. Такий спосіб роботи призведе до зменшення продуктивності, оскільки нові завдання будуть постійно надходити до єдиної інтерфейсної точки (*NameNode*) в *Hadoop*.

Іншою проблемою в потоковій системі *Hadoop* є розподіл ресурсів. Припустимо, що у кластері є чотири машини. Дві з них мають 8 ГБ оперативної пам'яті, а дві інші мають вже по 16 ГБ оперативної пам'яті з рівною кількістю пам'яті. Коли *Job Tracker* призначає завдання трекерам процесів, кожний *Task Tracker* відповідає лише за обробку цього завдання в межах власних ресурсів машини. Варто також зазначити, що в цьому сценарії всі машини будуть

розглядатися однаково, а *Task Tracker* буде виконувати завдання однаковим чином, незалежно від того, має він 8 ГБ оперативної пам'яті чи 16 ГБ.

Щоб подолати ці архітектурні проблеми, в *Hadoop* було розроблено нову модель обробки, яка називається ***YARN*** (*Yet Another Resource Negotiator, ще один розподілювач ресурсів*) для заміни моделей *Job Tracker* і *Task Tracker*. Його також називають *Hadoop 2*. Рисунок 11.16 ілюструє архітектуру *Hadoop* з включеним *YARN*:

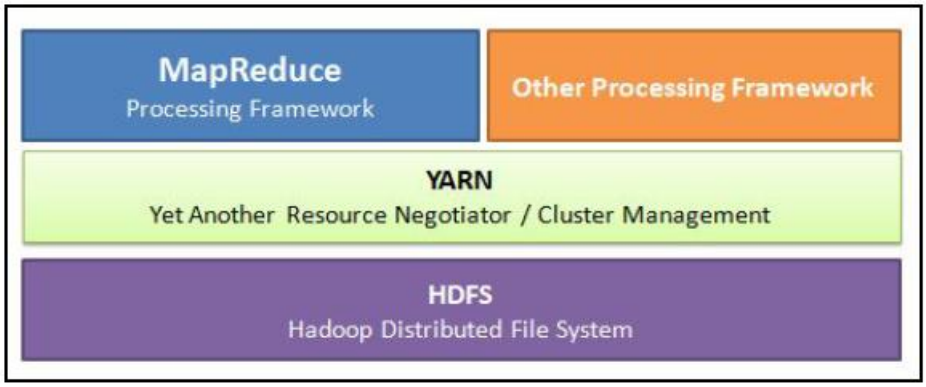


Рисунок 11.16 Місце YARN в архітектурі Hadoop

YARN розподіляє функціонал *Job Tracker* на декілька функціональних різновидів. Він працює з наступними чотирма новими послугами:

- **менеджер ресурсів** (*Resource Manager*);
- **менеджер вузлів** (*Node Manager*);
- **контейнер** (*Container*);
- **менеджер програм** (*Application Manager*).

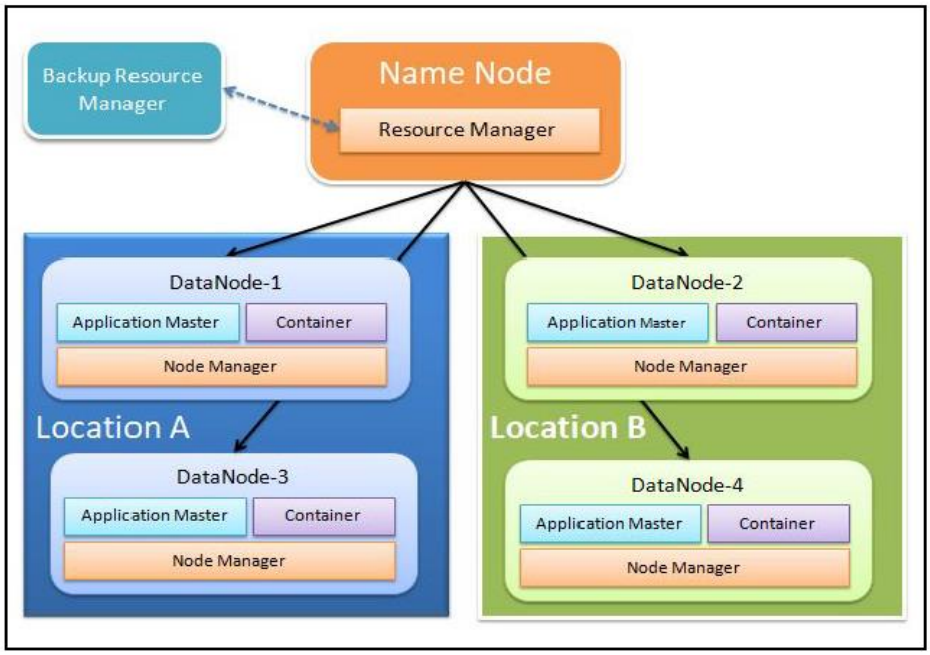


Рисунок 11.17 Іллюстрація моделі роботи YARN і розміщення його функціоналу у кластері

Рисунок 11.17 вище ілюструє модель роботи *YARN* і вказує місця, де буде виконуватися його функціонал:

11.7.2 Менеджер ресурсів

Менеджер ресурсів має дещо обмежену функціональність у порівнянні з *Job Tracker*. У попередній моделі *Job Tracker* відповідав за всю діяльність по плануванню процесів та їх статус у кластері, тоді як менеджер ресурсів відповідає лише за наступне:

- присвоєння ідентифікатора новому завданню, запущеному клієнтом;
- ініціювання нової служби демонів, що називається *менеджер програм* на доступному *менеджері вузлів* і призначення завдання цьому менеджеру програм для подальшого виконання і обробки;
- управління та призначення ресурсів кластера у вигляді контейнерів.

У *Hadoop v1* ми мали лише один трекер завдань. У разі будь-якої помилки в роботі *Job Tracker* виконання всього кластеру завершувалось. Тепер, з *Hadoop v2*, присутній *резервний менеджер ресурсів*, який може замістити оригінал у випадку будь-якої помилки.

11.7.3 Менеджер вузлів

Менеджер вузлів - це службовий демон, який працює на всіх вузлах даних. Це означає, що кожен вузол даних має власний менеджер вузлів. Він відповідає за моніторинг ресурсів свого вузла, таких як пам'ять, процесор і місце зберігання даних; звітує менеджеру ресурсів про те, скільки ресурсів вузла ще доступно і скільки було вже використано.

11.7.4 Контейнер

Контейнер - це частина використання процесора та оперативної пам'яті вузла, що була призначена менеджером ресурсів даному *Data Node* у кластері. Керування контейнерами здійснюється менеджером вузлів. Контейнер - це місце, де користувацька програма знаходиться і виконує свої завдання, такі як *Map* і *Reduce*.

11.7.5 Менеджер програм

Після того, як клієнт запросить виконання нового завдання у менеджера ресурсів, той зв'яжеться з будь-яким з доступних менеджерів вузлів і ініціює на вузлі виконання служби *менеджер програм*. Менеджер програм зв'яжеться з *NameNode* для отримання від нього метаданих, необхідних для виконання цього завдання. Потім буде створено по одній задачі для кожної розбивки файла вхідних даних.

Менеджер програм також вирішить, скільки контейнерів буде потрібно для виконання самого завдання. Якщо завдання невелике, менеджер програм виконує їх на своїй власній *JVM* або контейнері, що допомагає зменшити накладні витрати на запит і виділення окремого контейнера. Ці невеликі завдання називаються ***Uber Tasks***.

Менеджер програм припинить свою роботу після завершення завдання, призначеного для нього.

11.8 Висновки

Основна цінність IIoT-систем полягає не в архівації мільйонів подій, згенерованих давачами, а *в їх інтерпретації та прийнятті відповідних рішень*. Від світу, в якому мільярди суттєвостей з'єднуються і взаємодіють одна з одною і з хмарними платформами, захоплює дух, але *нас в першу чергу цікавлять дані, їх вміст, то, що в них не потрапило, і закономірності, які з них можна вивести*. Це та область інтернету речей, що відноситься до науки про дані та їх аналіз, є, напевно, найбільш цінною для клієнтів.

В інтернеті речей аналізується така інформація:

- структуровані дані (SQL-сховище, БД) передбачуваного формату;
- неструктуровані дані (необроблене відео або сигнали) високого ступеня випадковості і варіативності;
- частково структуровані дані (Twitter-потоки) з деяким ступенем випадковості і варіативності.

Іноді дані необхідно інтерпретувати і аналізувати в режимі реального часу у вигляді потоку, а іноді вони архівуються до часових БД з подальшим поглибленим аналізом у хмарі. Це стадія споживання даних. У деяких випадках

інформацію доводиться зіставляти з іншими джерелами потокових даних, а іноді вона просто записується і скидається в так зване *озеро даних*, на зразок *Hadoop*.

Далі йде етап *переміщення даних*. Такі системи обміну повідомленнями, як *Kafka*, направляють дані в поточний або пакетний процесор (або в обидва відразу). Дані обробляються у вигляді безперервного потоку. Цей процес зазвичай має певні обмеження і високу продуктивність, тому що інформація обробляється у пам'яті. У зв'язку з цим обробка даних повинна відбуватися не повільніше, ніж їх надходження. І якщо в хмарі можна розраховувати на роботу в режимі, близькому до реального часу, то промислові пристрої або безпілотні автомобілі не дають жорстких гарантій щодо продуктивності.

З іншого боку, пакетна обробка добре підходить для великих обсягів даних, особливо при зіставленні показань давачів з раніше збереженої інформацією.

Наступний етап може включати в себе прогнозування і повернення відповіді на запит. Дані можуть бути виведені на панель керування, записані в журнал або повернуті назад граничному пристрою для вжиття певних заходів при вирішенні локальних проблем.

Контрольні питання

1. Дати визначення Big Data. Пояснити кожну складову.
2. Назвати шість характеристик Big Data. Пояснити призначення кожної.
3. Пояснити терміни *пакетна обробка, потокова обробка*. Що це за обробка та які відмінності між ними? Призначення однієї і другої?
4. Пояснити терміни *Business intelligence, Data Mining, machine learning* Що це за обробки та які відмінності між ними? Коли їх застосовують?
5. Пояснити терміни *Data warehouse, data lake*. Що це за об'єкти та які відмінності між ними? Призначення одного і другого?
6. Чим відрізняється NoSQL БД від реляційної? Призначення однієї і другої?
7. Розповісти про склад і призначення платформи Hadoop. Архітектура платформи.

8. Що таке HDFS? Її призначення та відмінність від інших подібних систем.
9. Якою є базова архітектура HDFS? Показати на прикладі роботу і схему зберігання даних.
10. Що таке NameNode та DataNode? Їх призначення та функціонування.
11. Показати на прикладі, як виконується запис даних в HDFS.
12. Призначення і склад фреймворка MapReduce. Процеси Job Tracker та Task Tracker.
13. Як виконується поток MapReduce? Проілюструвати його роботу.
14. Що таке Mapper? Його призначення, як він функціонує?
15. Що таке етап тасування даних?
16. Що таке Reducer& Його призначення та функціонування.
17. Призначення та архітектура YARN компоненти. Чи може платформа Hadoop функціонувати без неї?
18. Призначення і функціонування менеджерів ресурсів та вузлів.
19. Поняття контейнеру в YARN та призначення менеджера програм.

12 АНАЛІТИКА ДАНИХ У ХМАРНИХ І ТУМАННИХ ПЛАТФОРМАХ

12.1 Різновиди форм хмарної аналітики верхнього рівня

На рис. 12.1 нижче показаний типовий шлях надходження даних від давача до панелі керування. Дані проходять через кілька проміжних ланок (вузли WPAN, широкосмуговий канал, хмарне сховище у вигляді озера даних і т.ін.). При виборі архітектури для побудови хмарного аналітичного рішення слід враховувати ефект масштабування. Рішення, прийняті на ранніх етапах проектування, можуть згодитися для десятка IoT-вузлів і одного хмарного кластера, але, коли число кінцевих IoT-пристроїв зросте до тисяч, а система почне охоплювати кілька географічних зон, масштабування може виявитися неефективним.

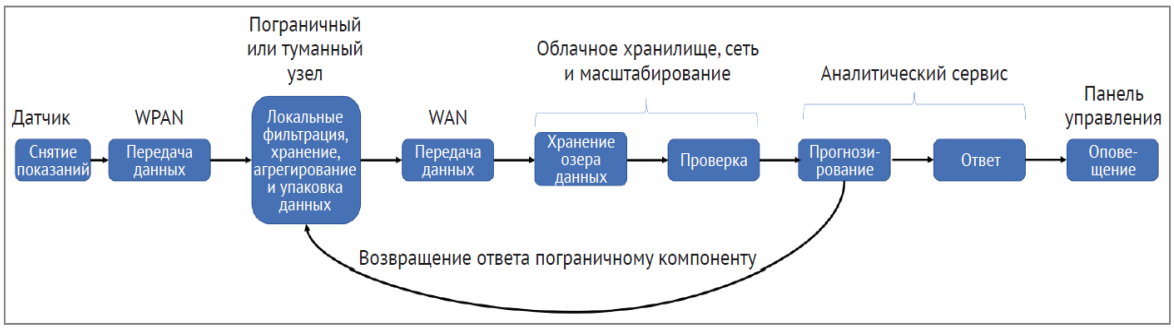


Рисунок 12.1 Типова схема передачі даних від давача до хмари

Аналітична (прогнозно-реагуюча) частина хмарної системи може приймати наступні форми:

- *система правил дії* - просто визначається дія і повертається результат;
- *потокова обробка* - тут події, такі як показники давачів, передаються потоковому процесору. Шлях обробки представляє собою граф з операторами обробки в якості вузлів; при цьому дані переходять від одного оператора до іншого. Кожен вузол містить код для певного етапу обробки та посилання на наступний вузол в графі. Такий граф можна реплікувати і виконувати паралельно в кластері, тому він легко масштабується на сотні комп'ютерів;

- *обробка складних подій* - ця частина системи заснована на мові запитів високого рівня подібної SQL і займається обробкою подій. Вона оптимізована під мінімальні затримки;
- *Lambda-архітектура* - ця модель намагається збалансувати пропускну здатність системи і рівень затримок, одночасно виконуючи пакетну обробку і паралельні обчислення з величезними наборами даних.

В IoT дані одночасно можуть надходити з мільйонів вузлів; це відбувається безперервно і асинхронно, з різними помилками, проблемами форматування та розбіжностями у часі. Для прикладу, у Києві встановлено близько 140 000 вуличних ліхтарів. Уявимо собі, що кожен ліхтар є розумним - тобто, він регулює свою яскравість в залежності від руху навколо себе, і, якщо поблизу нікого немає, він залишається затемненим і економить електроенергію (на це йде 2 байта у повідомленні). Кожен ліхтар може відстежувати сусідні ліхтарі, які вимагають ремонту (ще 1 байт). Крім того, він вимірює температуру (1 байт) і вологість (1 байт), допомагаючи генерувати локальний прогноз погоди. Нарешті, дані, з якими він працює, містять його ідентифікатор і часовий тег (8 байт). У штатному режимі ліхтарі генерують, в цілому, 140 000 повідомлень в секунду, хоча в години пік і на свята (а також через велике скупчення людей і завдяки наявності туристичних пам'яток) цей показник може досягати 180 000. Припустимо, наш хмарний сервіс здатний обробляти 140 000 повідомлень в секунду - тобто, відставання може досягати 40 000 подій в секунду. Якщо година пік дійсно затягнеться рівно на 1 годину, за цей час у нас накопичиться 144 000 000 необроблених подій. Щоб дати системі шанс надолужити згаяне, нам доведеться збільшити обчислювальну потужність кластера або зменшити вхідний потік. Якщо покласти, що кожну секунду в періоди низької активності генерується лише 20% нормального потоку повідомлень (30 000 повідомлень), хмарному кластеру знадобиться 0,36 години = 144 000 000 / (140000-30000) і 1872 МБ пам'яті (144 мільйона подій по 13 байт кожна), щоб нівелювати відставання.

Таким чином, продуктивність хмарної системи можна виразити наступними співвідношеннями:

$$\text{Відставання} = \begin{cases} 0, \text{якщо } R_{event} \leq C, \\ R_{event} - C, \text{якщо } R_{event} > C \end{cases}, \quad (12.1)$$

$$M_{backlog} = \text{Відставання} * M_1, \quad (12.2)$$

$$T_c = \frac{(\text{Відставання} * T_{\text{ЧНН}})}{C}, \quad (12.3)$$

де: *C – продуктивність роботи кластеру, подій/с;*

R_{event} - частота надходження подій, подій/с;

$T_{ЧНН}$ - тривалість часу найбільшого навантаження, с;

T_c – потрібний час на усунення відставання, с;

$M_{backlog}$ – об'єм відстаючіх повідомлень, байт;

M_1 – розмір одного повідомлення, байт.

12.2 Система правил дії

Система правил дії- це програмний компонент, який виконує якісь дії у відповідь на події. Наприклад, якщо вологість в кімнаті перевищить 50%, мешканцю відправляється SMS. Цей механізм також називають системою управління бізнес-правилами (англ. *Business Rule Management System*, або BRMS).

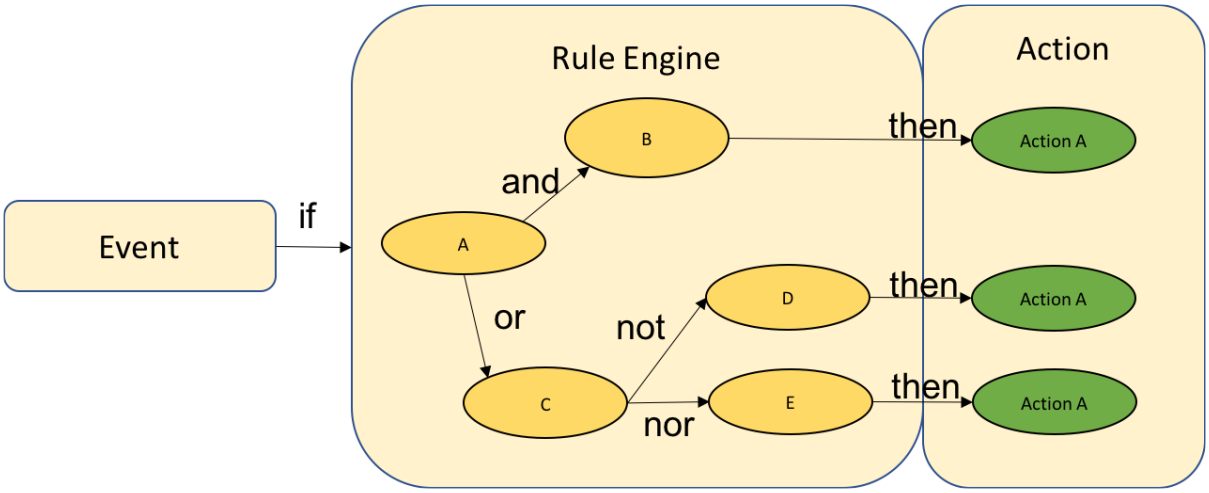


Рисунок 12.2 Приклад простої системи правил дії

Деякі системи правил мають поняття стану. Це означає, що вони можуть зберігати історію подій і виконувати різні дії в залежності від їх порядку,

кількості або характеру їх надходження. Системи, що не мають стану, перевіряють тільки поточну подію, як показано на рис. 12.2 вище.

У нашому прикладі роль системи правил виконуватиме *Drools*. *Drools* - це BRMS, розроблена компанією Red Hat і поширювана під ліцензією Apache 2.0. Її промислова версія називається JBoss Enterprise. Всі свої об'єкти *Drools* зберігає в робочій пам'яті; це такий набір подій, надісланих давачами, які слід порівняти для роботи заданого правила. *Drools* підтримує два види зчеплення: пряме і зворотне. Зчеплення - це метод логічного виведення, запозичений з теорії ігор.

При прямому зчепленні дані приймаються до тих пір, поки не буде сформовано ланцюжок правил. Останній може являти собою послідовність операторів *if / then*, як на малюнку вище. Пряме зчеплення безперервно шукає дані, що задовольняють одному з маршрутів *if / then*, щоб завершити дію. Зворотне зчеплення рухається в протилежному напрямку: від дії до даних, а не навпаки. Нижче представлений псевдокод, що демонструє роботу простої системи правил дії:

```
Smoke Sensor = Smoke Detected
Heat Sensor = Heat Detected

if (Smoke_Sensor == Smoke_Detected) && (Heat_Sensor == Heat_Detected)
        then Fire
if (Smoke_Sensor == !Smoke_Detected) && (Heat_Sensor == Heat_Detected)
        then Furnace_On
if (Smoke_Sensor == Smoke_Detected) && (Heat_Sensor == !Heat_Detected)
        then Smoking
if (Fire)
        then Alarm
if (Furnace_On)
        then Log_Temperature
if (Smoking)
        then SMS_No_Smoking_Allowed
```

Припустимо, що:

- Smoke_Sensor (давач задимленості) не спрацював;
- Heat_Sensor (давач тепла) спрацював.

Пряме зчеплення зупиниться на другій умові, роблячи висновок про те, що температура записується в журнал.

Зворотне зчеплення намагається довести, що опалення включено. Для цього воно рухається в протилежному напрямку:

1) чи можемо ми довести, що температура записується в журнал? Погляньте на наступний код:

if (Furnace_On) then Log_Temperature

2) так як температура записується в журнал, умова (*Furnace_On*) стає новою метою:

if (Smoke_Sensor == !Smoke_Detected) && (Heat_Sensor == Heat_Detected) then

Furnace_On

3) ми вже довели, що опалення включено, тому нова умова складається з двох частин: *Smoke_Sensor* і *Heat_Sensor*. Система правил розбиває його на дві мети:

Smoke_Sensor off
Heat_Sensor on

4) тепер система правил намагається досягти обох цілей. Під час цього процесу завершується робота логічного виведення.

Пряме зчеплення дозволяє реагувати на нові дані по мірі їх надходження, що може привести до запуску нового процесу логічного виведення.

Drools підтримує створення дуже складних і детальних правил, для зберігання яких може навіть знадобитися окрема база даних. Семантика мови

передбачає шаблони, входження в діапазон, підрахунок кількості спрацьовувань правила, зіставлення типів і роботу з колекціями об'єктів.

12.3 Потокова обробка

IoT-пристрій зазвичай пов'язаний з якимось давачем або іншим пристроєм, який вимірює або відстежує умови в реальному світі. Це робиться асинхронно по відношенню до решти працюючих технологій інтернету речей. Тобто, давач намагається передавати дані незалежно від того, чи слухає його хмарний або туманний вузол. Навіть якщо більша частина інформації від нього виявляється надлишковою, в якийсь момент може статися важлива подія. У зв'язку з цим дані передаються у вигляді потоків.

Потік інформації, що передається з давача в хмару, сприймається як:

- постійний і нескінченний;
- асинхронний;
- неструктурований або структурований;
- максимально близький до режиму реального часу.

Раніше, у п. 10.5, ми вже обговорювали тимчасові затримки, властиві хмарам. Ми також дізналися, що туманні обчислення допомагають вирішити цю проблему. Але і без наявності туманних вузлів робляться зусилля для оптимізації хмарної архітектури, щоб забезпечити підтримку режиму реального часу в інтернеті речей. Для цього хмара має вміти працювати з безперервним потоком даних. Альтернативою є *пакетна обробка*. Більшість апаратних платформ використовують одні і ті ж методи роботи з потоками, переміщаючи дані з одного блоку в інший; при цьому на кожному етапі їх надходження активізується наступна функція. Важливим елементом зниження загальної затримки є правильне використання сховища даних і правильно налаштований доступ до файлової системи.

У зв'язку з цим більшість поточних фреймворків виконують свої операції в пам'яті, намагаючись повністю виключити тимчасове зберігання інформації у файловій системі.

До того ж, потоки даних рідко бувають ідеальними. У ситуації, коли сотні тисяч давачів асинхронно передають свої показання, деякі дані будуть загублені

(при втраті зв'язку з давачем), мати некоректний формат (помилка при передачі) або надходити не в тому порядку (якщо хмара приймає інформацію з кількох джерел). Поточна система повинна як мінімум:

- мати здатність до масштабування при зростанні і сплесках кількості подій;
- надавати можливість публікації / підписки в своєму API-інтерфейсі;
- прагнути до мінімізації затримок;
- забезпечувати масштабування правил обробки;
- підтримувати озера і сховища даних.

Apache пропонує кілька відкритих проектів (з ліцензією *Apache 2*), які повинні допомогти в побудові архітектури обробки потоків. Один з них, *Apache Spark*, обробляє дані у вигляді невеликих пакетів. Це вкрай зручно, коли обсяг пам'яті в хмарному кластері обмежений (наприклад, менше 1 ТБ). Фреймворк *Spark* виконує свої операції в пам'яті, що знижує затримки і залежність від файлової системи (як уже згадувалося раніше). Пакетна обробка даних також добре підходить при роботі з моделями машинного навчання, про які ми поговоримо пізніше. Підтримка пакетних даних реалізована в декількох моделях, таких як *згорткова нейромережа* (англ. *Convolutional Neural Network*). Ще однією альтернативою від *Apache* є проект *Storm*, який намагається максимально наблизити хмарні обчислення до режиму реального часу. На відміну від *Spark* він має низькорівневий API-інтерфейс і обробляє дані у вигляді великих наборів подій, не розбиваючи їх на пакети. Це дозволяє домогтися низької затримки (частки секунди). *Storm* потребує більшого обсягу пам'яті хмарного кластера.

Щоб надавати дані вищевказаним фреймворкам для обробки потоків, можна використовувати *Apache Kafka*. *Kafka* використовується для утворення черги *MQTT* повідомлень, яка приймає повідомлення від різних IoT-давачів і клієнтів, і потім передає їх в *Spark* або *Storm*. Сам *MQTT* не займається буферизацією, тому, якщо з хмарою взаємодіють тисячі клієнтів, нам знадобиться якась система, що буде накопичувати повідомлення з вхідного потоку. Це дозволить *Kafka* масштабуватися на вимогу в залежності від навантаження (ще одна важлива якість хмари) і належним чином реагувати на

сплески подій. *Kafka* підтримує потоки на швидкості 100 000 повідомлень в секунду. Фреймворк може передавати дані зразу у пам'ять, навіть не зберігаючи їх; однак, як правило, цього не роблять - необроблені дані від давачів завжди намагаються зберегти в максимально "сирій формі", наскільки це можливо, з одночасною обробкою потоків від усіх інших давачів.

Коли планується розгортання IoT з тисяч чи мільйонів давачів та кінцевих вузлів, хмарне середовище буде використовувати озеро даних. *Озеро даних* є по суті величезним сховищем, де зберігаються неочищені нефільтровані дані з багатьох джерел. Озера даних - це плоскі файлові системи. На відміну від звичайних файлових систем така файлова система є повністю плоскою і не підтримує ієрархію, тома, каталоги, файли та папки. Для структуризації вмісту до кожного запису додається елемент метаданих (тег). Класична модель озера даних - це *Apache Hadoop*, і майже всі постачальники послуг у хмарі використовують певну форму озера даних для своїх послуг.

Зберігання озерних даних є особливо корисним в IoT, оскільки воно буде зберігати будь-які форми даних, будь то структуровані чи неструктуровані. Озеро даних також передбачає, що всі дані є цінними та зберігатимуться постійно. Ця величезна маса даних є оптимальною для аналітичних движків. Якість роботи цих алгоритмів прямо залежить від кількості використаних ними даних.

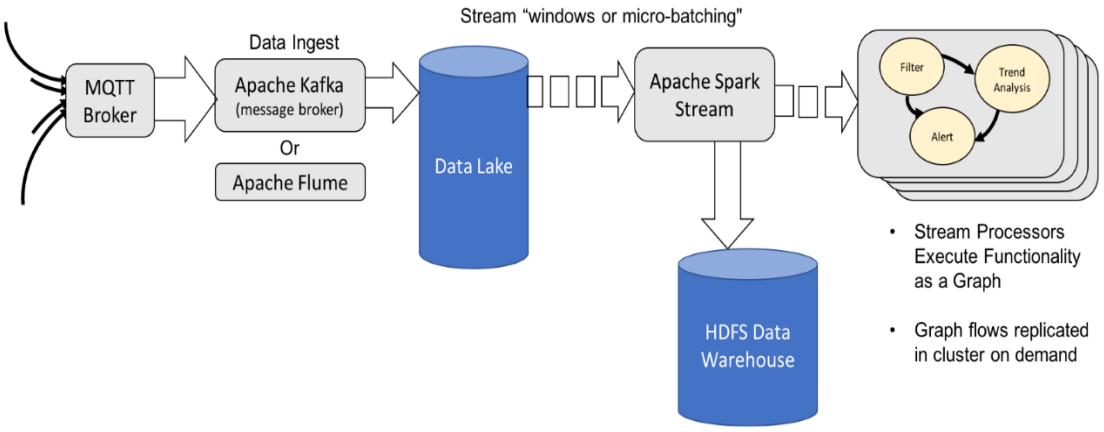


Рисунок 12.3 Принцип подачі даних в хмарне сховище. Spark грає роль потокового сервісу

На рис. 12.3 представлена концептуальна архітектура на основі традиційної пакетної і потокової обробки. Озеро даних наповнюється за допомогою екземпляра *Kafka*. Хмара *Kafka* надає пакетний інтерфейс до *Spark* і передає дані до сховища.

Компоненти використовують стандартні з'єднання, тому топологію даної архітектури можна модифікувати декількома способами.

12.4 Обробка складних подій

Обробка складних подій (англ. *Complex event processing*, або *CEP*) - це ще одна аналітична система, яка використовується для розпізнавання шаблонів у вхідних даних. У 1990-х р.р. її почали використовувати в дискретно-подієвому моделюванні і торгівлі волатильністю на фондовому ринку. За своєю природою вона дозволяє аналізувати живі потоки даних в режимі реального часу. Сотні і тисячі подій, що надходять в систему, фільтруються і очищаються, перетворюючись в події вищого рівня - більш абстрактні, ніж початкові показники давачів. Системи *CEP* забезпечують більш швидкий аналіз навіть у порівнянні з поточними процесорами, які можуть обробляти події в межах декількох мілісекунд. Але, на відміну від *Spark*, системи *CEP* мають меншу надмірність і меншу здатність до динамічного масштабування.

Системи *CEP* використовують SQL-подібні запити, однак задані шаблони або правила шукаються не в базі даних, а у вхідному потоці. *CEP* складається з кортежу (окремого елемента даних з часовим тегом) і використовує різні аналітичні шаблони, подібні до того, що описаний в п. 12.2, які добре поєднуються з ковзаючими вікнами подій. За своєю семантикою мова запитів схожа на SQL, але працює значно швидше звичайних баз даних, тому всі правила і дані зберігаються в пам'яті (зазвичай всередині багатогігабайтної БД). Крім того, дані повинні поставлятися сучасною поточною системою повідомлень, такою як *Kafka*.

Системи *CEP* підтримують такі операції, як ковзаючі вікна, з'єднання і виявлення послідовностей. Крім того, як і системи правил, вони можуть працювати як з прямим, так і з зворотним зчепленням. Промисловим стандартом у цій галузі є система *Apache WSO2 CEP*, яка в поєднанні з *Apache Storm* здатна обробляти більше 1 мільйона подій в секунду, не використовуючи при цьому постійне сховище. В системі *WSO2* використовуються SQL-запити, але вона також підтримує скрипти на мовах *JavaScript* і *Scala*. Ще однією її перевагою є

можливість розширення функцій за допомогою пакета під назвою *Siddhi*, який надає такі послуги:

- геолокація;
- обробка природної мови;
- машинне навчання;
- кореляція і регресія часових рядів;
- математичні операції;
- робота з рядками і *RegEx*.

Дані сприймаються як окремі події, що дозволяє застосовувати складні правила до мільйонів повідомлень, що надходять одночасно.

Розробник повинен розуміти, в яких ситуаціях слід застосовувати системи правил і *CEP*. Якщо умова оперує простими показаннями, такими як діапазони температур, система позбавлена стану і може бути реалізована на основі простого механізму *BRMS*. Якщо ж система зберігає часові дані або ряд станів, слід використовувати *CEP*.

12.5 *Lambda-архітектура*

Lambda-архітектура намагається знайти компроміс між затримками і пропускною здатністю (рис. 12.4). По суті, вона поєднує в собі пакетну і потокову обробку. За аналогією із загальною хмарної топологією *OpenStack* та інших хмарних фреймворків *Lambda* зберігає споживані дані в репозиторії, доступному тільки для читання. Така топологія складається з трьох рівнів:

- *пакетний рівень* - зазвичай заснований на кластерах *Hadoop*. Обробка в ньому відбувається набагато повільніше, ніж на поточному рівні. Приносячи в жертву затримки, він максимізує пропускну здатність і точність;
- *потоковий рівень* - це потік даних, що проходять через пам'ять в режимі реального часу. Дані можуть губитися, містити помилки і приходити не в тому порядку. *Apache Spark* надає відмінну систему обробки потоків;
- *сервісний рівень* - тут відбувається збереження, аналіз і візуалізація пакетних і поточних результатів. Зазвичай тут застосовуються такі

компоненти як *Druid* (надає засоби об'єднання пакетного і поточного рівнів), *Apache Cassandra* (масштабована база даних) і *Apache Hive* (довгострокове сховище даних).

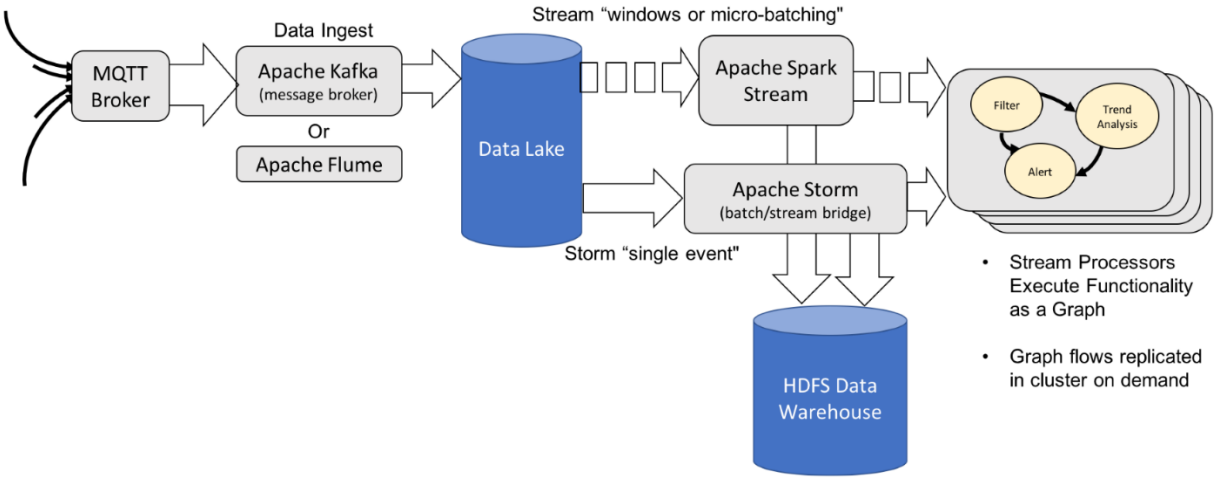


Рисунок 12.4 Складнощі Lambda-архітектури. Тут пакетний рівень розміщує дані в сховищі *HDFS*, а потоковий рівень передає свої результати безпосередньо системі аналізу в режимі реального часу, використовуючи *Spark*

*Lambda-архіт*ектури за своєю природою мають підвищену складність, якщо порівнювати їх з іншими аналітичними системами. Це гібридний підхід, для успішної реалізації якого потрібні додаткові ресурси.

Контрольні питання

1. Що таке Data Science? Її визначення та функції.
2. Які різновиди форм хмарної аналітики вам відомі?
3. Як розрахувати продуктивність хмарної системи?
4. Описати роботу системи правил дії, як різновида хмарної аналітики.
5. Що таке потокова обробка? Її функціонування та вимоги до неї.
6. Охарактеризувати два програмних потокових пакета: Spark та Kafka. В яких випадках який слід використовувати?
7. Як працює аналітична система на основі обробки складних подій. Де застосується?
8. Що таке Lambda-архітектура, її топологія та функціонування?

НАЧАЛЬНО-МЕТОДИЧНІ МАТЕРІАЛИ

Базові

1. Molloy D. Exploring Raspberry Pi. Interfacing to the Real World with Embedded Linux. – Indianapolis: John Wiley & Sons, Inc., 2016. – p. 722
2. Membrey P., Hows D. Learn Raspberry Pi 2 with Linux and Windows 10. - New York: Springer APress, 2015. – p. 319
3. Lea P. IoT and Edge Computing for Architects, 2nd Edition. - BIRMINGHAM – MUMBAI: Packt Publishing Ltd, 2020. – p. 633
4. Culic I. Commercial and Industrial Internet of Things Applications with the Raspberry Pi. Prototyping IoT Solutions. - New York: Springer APress, 2020. – p. 304
5. Using Node-RED to build the internet of things. Workshop
6. Пупена О. Довідник Node-RED з елементами опису Java Script, JSON, JSONata. – Київ: Самвидав, 2019. – 146 с.
7. Петин В. Arduino и Raspberry Pi в проектах Internet of Things, 2-е изд. – СПб: БХВ Петербург, 2019. – 429 с.

Допоміжні

8. Buyya R., Dastjerdi A. Internet of Things. Principles and Paradigms. – Cambridge: Elsevier Inc., 2016. – p. 348
9. Bell Ch. Beginning Sensor Networks with Arduino and Rasberry Pi. – New York: Apress, 2013. – p. 407
10. Негус К. Ubuntu и Debian Linux для продвинутых. Более 1000 незаменимых команд, 2-е изд. — СПб.: Питер, 2014. — 384 с.
11. Херцог Р., Ма Р. Настольная книга администратора Debian. - Лицензия: Creative Commons Attribution-ShareAlike, 2016. – 522 с.
12. Zarrelli G. Mastering Bash. - Packt Publishing, 2017. – p. 546
13. Prinz P., Crawford T. C in a Nutshell. The Definitive Reference, 2nd Edition. - Sebastopol: O'Reilly Media, Inc, 2015. – p. 2016
14. Гриффитс А. GCC. Полное руководство. – М. – С.-Пб. – К.: Диасофт, 2004. – 609 с.
15. Posch M. Mastering C++ Multithreading. – Birmingham: Packt Publishing, 2017. – p. 552.
16. Карлсон Д. Eclipse. – М.: ЛОРИ, 2013. – 353 с.
17. Allan A., etc. Make. Bluetooth. Bluetooth LE Projects with Arduino, Raspberry Pi, and Smartphones. - San Francisco: Maker Media, 2016. – p. 257
18. Waher P. Mastering Internet of Things. Design and create your own IoT applications using Raspberry Pi 3. - BIRMINGHAM – MUMBAI: Packt Publishing Ltd, 2018. – p. 398

19. Kamani S. Full Stack Web development with Raspberry Pi 3. - BIRMINGHAM – MUMBAI: Packt Publishing Ltd, 2017. – p. 233
20. Шварц М. Интернет вещей с ESP8266. - СПб: БХВ Петербург, 2018. – 192 с.