СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

імені ВОЛОДИМИРА ДАЛЯ

(м. Сєвєродонецьк)

Факультет Інформаційних технологій та електроніки
(повне найменування факультету)

Кафедра Програмування та математики
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

освітньо-кваліфікаційного рівня магістр
(бакалавр, спеціаліст, магістр)

напряму підготовки 122 – Комп'ютерні науки та інформаційні технології
(шифр і назва напряму підготовки)

спеціальності Інформатика
(шифр і назва спеціальності)

на тему Розробка аналітичної системи «Туманних обчислень» для пристроїв з обмеженою кількістю ресурсів Інтернету Речей

Виконав: студент групи ІНФ-16дм

Гамов О.В. …….

(прізвище, та ініціали) (підпис)

Керівник Фесенко Т.М. …….

(прізвище та ініціали) (підпис)

Завідувач кафедри Лифар В. О.. …….

(прізвище та ініціали) (підпис)

Рецензент Кучма Ю.В …….

(прізвище та ініціали) (підпис)

Сєвєродонецьк - 2018

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

імені ВОЛОДИМИРА ДАЛЯ

(м. Сєвєродонецьк)

Факультет Інформаційних технологій та електроніки

Кафедра Програмування та математики
Освітньо-кваліфікаційний рівень магістр

(бакалавр, спеціаліст, магістр)

Спеціальність Інформатика

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____
В.О.Лифар
“____” _____ 2017
року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Гамову Олексію Валерійовичу

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) Розробка аналітичної системи «Туманних обчислень» для пристроїв з обмеженою кількістю ресурсів Інтернету Речей

керівник проекту (роботи) Фесенко Т.М доц.

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “ 8” листопада 2017 року №

2. Строк подання студентом проекту (роботи) 15.01.2018

3. Вихідні дані до проекту (роботи)_____Матеріали науково-дослідної практики

_____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Актуальність теми і постановка завдань дослідження. Аналіз стану та проблем Інтернету Речей, вітчизняних та зарубіжних розробок та досліджень;

2) Огляд та вибір програмних інструментів, програмник компонентів та бібліотек для розробки системи;

3) Програмна реалізація аналітичної системи «Туманних обчислень» для пристроїв з обмеженою кількістю ресурсів Інтернету Речей;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Дата видачі завдання_____8.11.2017_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Актуальність теми і постановка завдань дослідження	09.11.17-15.11.17	
2	Аналіз стану, проблем, розробок та досліджень	15.11.17-11.12.17	
3	Дослідження, аналіз та вибір програмних компонентів та бібліотек	11.12.17-20.12.17	
4	Проектування архітектури системи	20.12.17-01.01.18	
5	Програмна реалізація системи	01.01.18-12.01.18	
6	Оформлення пояснювальної записки	12.01.18-15.01.18	

Студент _____ ____Гамов О.В._____

(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ ____Фесенко Т.М_____

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту: 81 **с.,** 37 **рис.,** 3 **табл.,** 36 **посилань.**

Предмет дослідження: аналітична система «Туманних обчислень» для пристроїв з обмеженою кількістю ресурсів Інтернету Речей

Мета роботи: розробка аналітичної системи «Туманних обчислень» для пристроїв з обмеженою кількістю ресурсів Інтернету Речей..

Метою даної роботи є розробка аналітичної системи туманних обчислень Інтернету Речей і реалізація її у вигляді програмного рішення. Це дасть можливість істотно зменшити кількість трафіку в умовах сильнозашумлених і/або дорогих каналів зв'язку таких як супутниковий зв'язок, а так само застосовувати дане рішення практично для будь-яких аналітичних задач не володіючи знаннями в області програмування, що істотно здешевить впровадження аналітики в процес.

Існуючі аналітичні системи є корисними, однак не вирішують проблем з трафіком і вимагають участі висококваліфікованих інженерів і адаптації даних рішень під вимоги користувача за допомогою програмування. Дані фактори збільшують час впровадження аналітики і істотно збільшують її вартість.

Наявність можливості візуального програмування системи покращують користувальницький досвід і дозволять легше створювати аналітичні запити, особливо на початку освоєння системи.

Застосування асинхронного архітектури заснованої на подіях дозволяє більш раціонально використовувати апаратні ресурси, що є критичним в системах з сильно обмежених кількістю ресурсів.

Використання стандартизованих протоколів верхніх рівнів служить для безшовного з’єднання в гетерогенних мережах. Використання такого проколу як MQTT дозволяє скоротити трафік і вирішити проблеми адресації, якості обслуговування та деякі інші транспортні проблеми на верхньому рівні стека протоколів. Також був доданий додатковий протокол адресації, заснований на протоколі MQTT, який дозволяє адресувати підгрупи в групах що додає системі адресації ще більшої гнучкості.

Застосування динамічних аналітичних моделей обчислення дозволяє зробити процес зміни аналітики більш легким в порівнянні з стандартним механізмом оновлень, таким як оновлення по повітрю.

Була застосована, проаналізована і вдосконалена парадигма реактивного програмування, яка застосовується для побудови максимально гнучкої асінхронної архітектури системи.

ТУМАННІ ОБЧИСЛЕННЯ, АНАЛІТИЧНА СИСТЕМА, ІНТЕРНЕТ РЕЧЕЙ, MQTT,, IoT, МЕЖЕВА АНАЛІТИКА, РОЗПОДІЛЕНА АНАЛІТИКА, ХМАРНІ ОБЧИСЛЕННЯ.

Умови отримання дипломного проекту
93400 м. Сєвєродонецьк , вул. Донецька, 41.

ABSTRACT

Explanatory note to the diploma project: 81 sec., 37 Fig., Table 3., 36 links.

Subject of study: «fog» analytics system for restricted devices of Internet of Things.

Objective: Developing «fog» analytics system for restricted devices of Internet of Things.

This work is intended to develop «fog» analytics system for restricted devices of Internet of Things and implement it in a form of software. It gives an opportunity to reduce amount of traffic significantly when it is a noisy or/and an expensive channel is used, such as satellite connection, to transmit data, as well as it allows to apply the developed solution almost for any analytic purposes without knowledge in programming field, what makes analytics integration much cheaper.

Existing analytics systems are useful, but don't solve the problems of traffic amount and require qualified engineers to be involved as well as adopting these solutions using programming skills to fit user's requirements.

The presence of a visual programming ability of the system improves user experience and allows to build analytics queries easier, especially on early stages of the system mastering.

Applying of event-based asynchronous architecture makes the system to use hardware resources rationally and efficiently.

Using of standard protocols on high levels serves for seamless connection in heterogeneous networks. Applying such protocol as MQTT brings the ability to reduce traffic amount, to solve addressing problems, quality of service problems and some transport problems on the high stack protocols level. Also there was added an additional addressing protocol, based on MQTT, which allows to address subgroups inside of groups what makes addressing system more flexible.

Dynamic analytics models usage allows to make the process of analytics changes more easier comparing to a standard update mechanism, such as update over the air.

There was used, analysed and improved the reactive programming paradigm, which is used to build flexible asynchronous system architecture.

FOG COMPUTING, ANALYTICS SYSTEM, INTERNET OF THINGS, MQTT, IoT, EDGE ANALYTICS, DISTRIBUTED ANALYTICS, CLOUD COMPUTING.

For manuscript contact

93400 Ukraine, Luhansk reg., Severodonetsk,Donets'ka, 41

Зміст

ВСТУП

У сучасному світі комп’ютерних технологій обчислювальні потужності ЕОМ досягли неймовірних показників, системи обробки інформації розвинені як ніколи, з’явилась нова течія у сфері обробки інформації - Big Data. Системи обробки даних що створюються добре маштабуються і здатні обробляти неймовірно великі обсяги даних за допомогою кластеризації і розподілу навантаження. Однак у таких систем є фундаментальні проблеми, які залишилися або не вирішеними або вирішуються для кожної конкретної системи окремо, витрачаючи все більше і більше ресурсів для вирішення одних і тих же проблем.

По-перше - проблема надання даних для обробки. Часто джерела інформації знаходяться за багато сотень або десятків тисяч кілометрів від центрів обробки даних. У таких віддалених точок, в багатьох випадках, немає зв'язку з центрами обробки даних або є зв'язок через ненадійний, високозашумленний і дорогий канал, такий як супутниковий зв'язок. При таких обставинах пересилати великі обсяги даних дорого або може просто бути не можливо з огляду на низьку пропускну здатність каналу передачі даних.

По-друге - недетермінований час передачі даних. В системах, де потрібна миттєва реакція, недетермінований час передачі може зіграти фатальну роль у стабільності системи (наприклад у системах реального часу). У таких системах рішення про корекцію поведінки системи на підставі даних цієї системи має прийматися локально.

По-третє - розробка таких систем дуже важка і вимагає значних витрат як з боку часу, так і з боку ресурсів (матеріальних, людських, тощо). Для розробки такої системи потрібен підрозділ співробітників з високою технічною кваліфікацією.

Рішенням всіх перерахованих проблем є розробка системи з гібридним підходом, який називається «Туманні обчислення». Первинна обробка даних дозволяє моментально повідомляти керуючий програмі про наявність будь-

яких проблем, скоротити дорогий трафік і знизити навантаження на центри обробки даних, а також само, система будучи максимально гнучкою і простою дозволить користувачам які мають мінімальні навички роботи з комп'ютером розгортати функціональні системи обробки даних і автоматизувати керування своїми процесами з мінімальними витратами ресурсів.

Таким чином, тема дипломної роботи є актуальною. так як вирішує перераховані вище проблеми.

Об'єкт дослідження – моделі аналітичних систем Інтернету Речей.

Предмет дослідження – аналітична система «Туманних обчислень» для пристроїв з обмеженою кількістю ресурсів Інтернету Речей.

Мета дослідження – розробка аналітичної системи «Туманних обчислень» для пристроїв з обмеженою кількістю ресурсів Інтернету Речей.

Задачі дослідження:

- аналіз сучасного стану розвитку Інтернету Речей та існуючих аналітичних рішень;
- вибір технологій та програмних засобів для розробки аналітичної системи туманних обчислень;
- розробка аналітичної системи для Інтернету Речей, що відповідає вимогам пристроїв з сильно обмеженою кількістю ресурсів.

Методи дослідження – загальні методи теорії інформації (при моделюванні системи); класичні методи математичного аналізу та лінійної алгебри (при створенні аналітичних компоннтів системи); чисельні методи для розв'язання задач на ЕОМ.

Практичне значення отриманих результатів укладено в отриманой системі що дозволяє вирішувати широкий спект аналітичних проблем не прибігаючі до написання кода, що економить час, грощі та інщі ресурси.!

1 ПЕРШИЙ РОЗДІЛ

СТАН ПИТАННЯ. МЕТА І ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Основні поняття концепції Інтернету Речей

Архітектура оригінального Інтернету була створена задовго до того, як мільярди простих пристроїв, таких як датчики і прилади, стали розглядатися як потенційні учасники комунікацій[1]. Популяризація цих простих пристроїв створює величезні проблеми для нинішньої мережевої парадигми з точки зору кількості пристроїв, безпрецедентних вимог до недорогої зв'язності, неможливості управління великими і різноманітними типами обладнання, проблем аналізу і коригування поведінки різних видів систем. В даному розділі розглядається парадигма Інтернету Речей, проблематика цієї парадигми, поточний стан проблем, вирішення і дослідження ведуться в цій галузі.

Інтернет Речей (англ. Internet of Things, IoT) - концепція обчислювальної мережі фізичних предметів («речей»), оснащених вбудованими технологіями для взаємодії один з одним і / або з зовнішнім середовищем [2]. Вперше термін «Інтернет Речей» ввів Кевін Ештон в 1999 році: «Я міг помилятися, але я впевнений, що фраза «Інтернет Речей» почала своє життя як назва презентації, яку я зробив в Procter & Gamble (P & G) в 1999 році. Зв'язування RFID в P & G ланцюжка, які постачають дані - тодішня промовиста тема в Інтернеті, була не просто хорошим способом привернути увагу керівників, а способом підвести їх до важливого прозріння, яке часто неправильно розуміється»[3]. Розвиток Інтернету Речей дав поштовх до розвитку таких технологій як IPv6, M2M Networks, Cloud Computing і розповсюдженню бездротових мереж.

Інтернет Речей - це конвергенція підключення людей, Речей, даних і процесів, які будуть змінювати людське життя, бізнес і все навколо. Cisco прогнозує ринок прибутку в розмірі 19 трильйонів доларів США для IoT і оцінює, що буде 50 мільярдів розумних об'єктів підключених до Інтернету до

2020 року [4]. Парадигма Cloud computing є важливою сходинкою еволюції, яка з'явилася в 2005 році. Додатки, такі як Google Apps, Facebook, Twitter і Flicker використовують цю технологію і широко використовуються в нашому повсякденному житті. Ефективні механізми обробки даних і масштабовані інфраструктури Google File System [5], MapReduce [6], Apache Hadoop [7], Apache Spark [8], підтримують і використовують хмарні сервіси.

1.2 Загальні проблеми сучасного Інтернету Речей

Оскільки Інтернет Речей продовжує розвиватися, оцінюється подальший потенціал в поєднанні з відповідними технологічними підходами і концепціями таких як Хмарні обчислення, майбутній Інтернет, великі дані, робототехніка і семантичні технології. Ідея, звичайно, не нова як така, але стає очевидною, оскільки ці пов'язані поняття почали об'єднує в синергію. Проте, Інтернет Речей все ще дозріває, зокрема, через кількість факторів, які обмежують повну експлуатацію IoT. Серед них:

- Немає чіткого підходу до використання унікальних ідентифікаторів і нумерації простору для різних видів постійних і непостійних об'єктів в глобальному масштабі;
- Відсутність прискореного використання і подальшої розробки еталонних архітектур IoT як, наприклад, еталонна модель архітектури (ARM) проекту IoT-A.
- Менш швидке просування в семантичної сумісності для обміну даних датчиків в гетерогенних середовищах.
- Труднощі в розробці чіткого підходу до впровадження інновацій, довіри і володіння даними в IoT в той же час поважаючи безпеку і конфіденційність в складному середовищі.
- Труднощі в розвитку бізнесу, який охоплює весь потенціал Інтернету Речей.

- Відсутність великомасштабних умов тестування і навчання, які полегшили б експерименти зі складними сенсорними мережами і стимулювали інновації за допомогою рефлексії і досвіду.

- Тільки частково розгорнуті інтерфейси в світлі зростаючих обсягів даних і потреби в контекстно-інтегрованому уявленні.

- Практичні аспекти, такі як істотні тарифи на роумінг для географічних застосувань датчиків великого діапазону і відсутність технічної доступності миттєвого і надійного з'єднання з мережею.

Парадигма Інтернету Речей розглядає наявність в середовищі безлічі речей / об'єктів, які через бездротові та дротові з'єднання і унікальні схеми адресації можуть взаємодіяти один з одним і співпрацювати з іншими речами / об'єктами для створення нових додатків / послуг і досягати спільних цілей. У цьому контексті дослідження і проблеми розвитку для створення розумного світу величезні. Світ, де реальні, цифрові і віртуальні об'єкти сходяться для створення інтелектуальних середовищ які виробляють енергію, транспорт, міста і багато інших галузей більш розумними.

Мета - дозволити підключати речі в будь-який час, в будь-якому місці, з чим завгодно і ким завгодно, ідеально використовуючи будь-який шлях / мережу і будь-який сервіс. Інтернет Речей - це нова революція в Інтернеті. Об'єкти роблять себя впізнавані, приймають Контекст-залежні рішення, завдяки тому, що вони можуть ідентифікуватися в середовищі. Вони можуть отримати доступ до інформації, яка була зібрана іншими об'єктами, або вони можуть бути компонентами складних сервісів.

Використання в Інтернеті Речей таких технологій як сенсорні мережі, RFID, M2M, мобільний інтернет, семантичні дані, семантичний пошук, IPv6 і т. д. можуть бути згруповані за трьома категоріями [9]:

1. технології, які дозволяють «речам» збирати контекстуальную інформацію;

2. технології, які дозволяють «речам» обробляти контекстуальную інформацію;

3. технології для підвищення безпеки і конфіденційності.

Перші дві категорії можуть спільно розумітися як функціональні будівельні блоки, які потребують створення «інтелекту», в «речі», які дійсно є особливостями IoT. Третя категорія не є функціональною, а скоріше вимога де-факто, без якого проникнення у IoT стало б набагато простіше. З Інтернетом Речей все, що нас оточує пов'язано між собою. Інтернет Речей набагато більше, ніж M2M зв'язок, бездротові сенсорні мережі, 2G / 3G / 4G, RFID і т. Д. Це технології є тільки частиною Інтернету Речей, які роблять застосування Інтернету Речей можливим. Базові компоненти Інтернету Речей показані на рисунку 1.1.

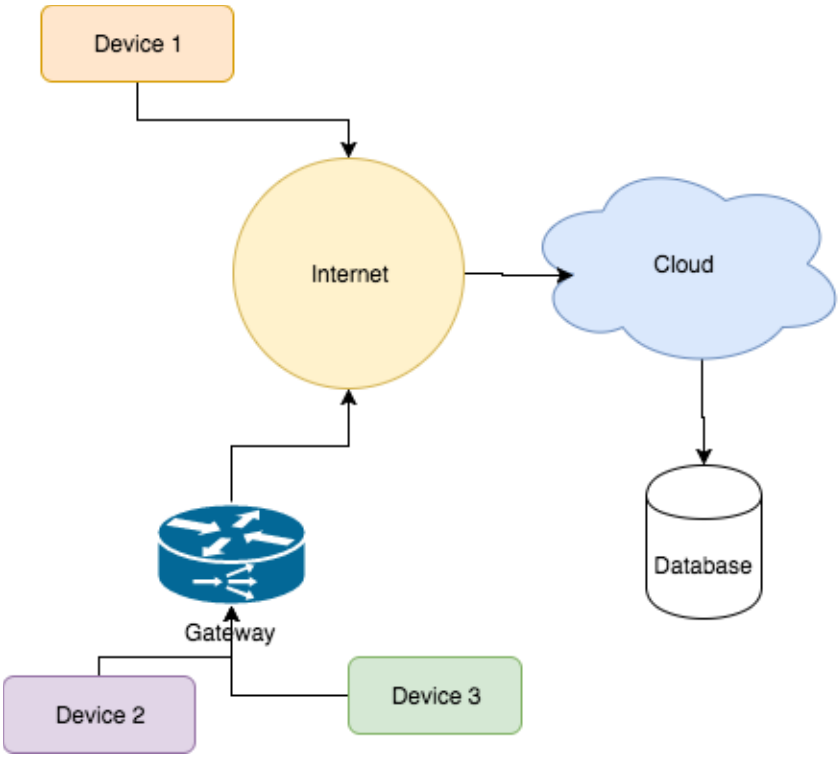

Рисунок 1.1 - Базові компоненти Інтернету Речей

Як видно, з Рисунку 1, Device 1 під'єднан безпосередньо до Cloud, а Device 2 і Device 3 через Gateway. У свою чергу Gateway є теж учасником Інтернету Речей, в зв'язку з чим виникає проблема адресації пристроїв в мережі

ІВ, так як Gateway повинен бути прозорий для пристроїв Device 1 і Device 2. Однак пристрої в мережі Інтернету Речей не повинні завжди бути фізичними. Прикладом таких не фізичних пристроїв може бути віртуалізація і віртуальні машини, або, узагальнюючи в термінах Інтернету Речей - віртуальні сенсори.

1.3 Проблема гетерогенності

Віртуальний датчик можна розглядати як продукт просторової, тимчасової і/або матеріальної трансформації вихідного або іншого віртуального або фізичного датчика, що виробляє дані з необхідною інформацією про походження даних і про проведене перетворення над даними від сенсорів-джерел. Віртуальні датчики і виконавчі механізми - це абстракція програмування, яка спрощує розробку децентралізованих додатків [10].

Дані, отримані за допомогою набору датчиків, можуть збиратися, оброблятися у відповідності з функцією агрегації, що надається додатком, а потім сприйматися як свідчення одного віртуального датчика. По суті, віртуальний виконавчий механізм забезпечує єдину точку входу для розподілу команд на набір реальних вузлів виконавчих механізмів. Потік інформації між реальними пристроями та віртуальними датчиками або приводами представлений на схемі (Рис. 1.2).

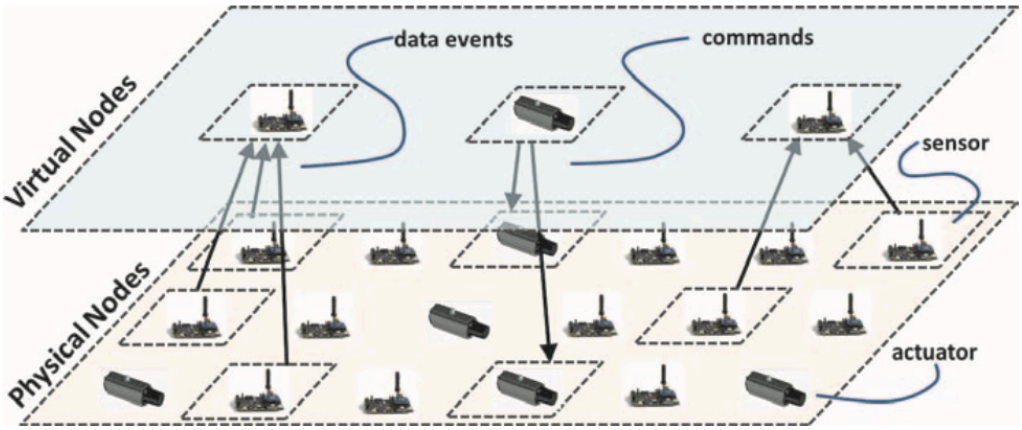

Рисунок 1.2 - Віртуальні датчики

Віртуальний датчик поводиться точно так само, як справжній датчик, що надає дані часової послідовності з певної географічної області з новими певними тематичними концепціями або спостереженнями, які можуть не мати реальні датчики. Віртуальний сенсор може не володіти фізичними властивостями фізичного датчика, такими як інформація про виробника або акумуляторі, але має інші властивості, такі як: хто створив його; які методи використовуються і на яких вхідних датчиках він заснований. Так само віртуальні сенсори можуть бути організував на різних шарах (рівнях) в мережі Інтернету Речей (Рис. 1.3).

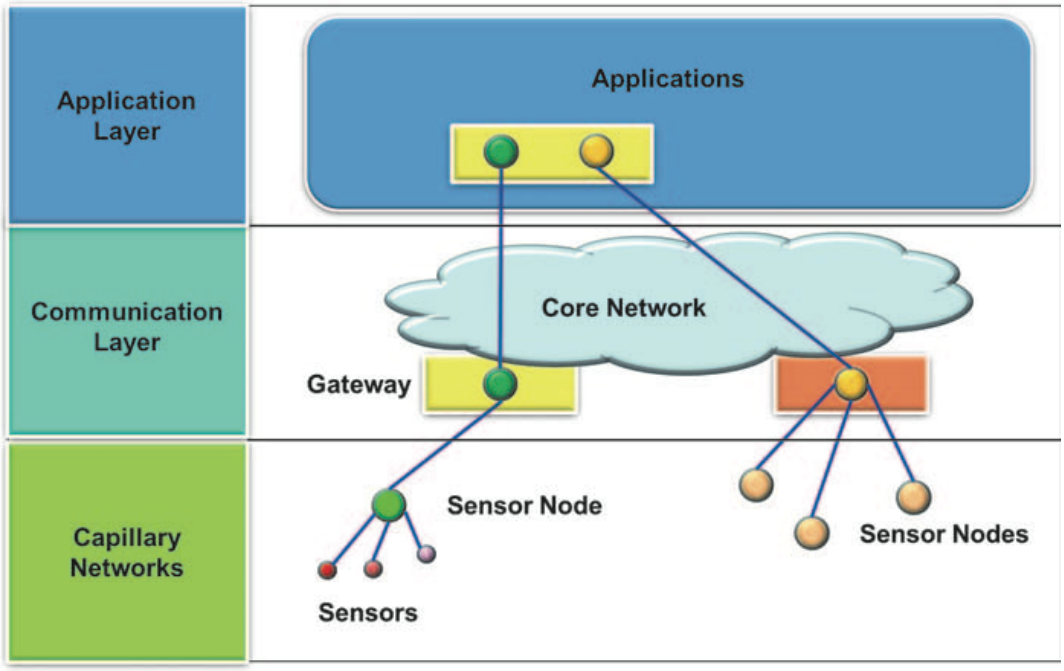

Рисунок 1.3 - Рівні організації віртуальних сенсорів

На найнижчому рівні знаходяться ті сенсори, які пов'язані з більш простою локальною обробкою даних (наприклад, в вузлі який безпосередньо з'єднаний з сенсорами) і на найвищому рівні - абстрактна комбінація різних датчиків на рівні додатку (включаючи призначені для користувача віртуальні датчики). Таким чином, до створення віртуальних сенсорів можна підійти з двома різними ступенями складності:

- Поєднання обмеженого числа пов'язаних датчиків для отримання нових віртуальних даних (зазвичай це робиться на вузлі до якого приєднані фізичні датчики або на рівні шлюзу).

- Комплексний процес отримання віртуальної інформації з величезного простору даних що отримуються (як правило, на рівні додатків).

Крім того, важливо також враховувати, що через тимчасового розміру даних датчика велика частина обробки, необхідної для розробки віртуальних датчиків, тісно пов'язана з концепцією події, як визначено в ISO 19136, «дія, яке відбувається в момент або через інтервал часу», а також з обробкою подій «створення, видалення, читання і редагування, а також реагування на події та їх подання»[11].

На даний момент існують дві проблеми які потрібно вирішити у контексті віртуальних датчиків:

- Безшовна інтеграція і интероперабільність «реальних» і «віртуальних» датчиків. Це означає, що віртуальні датчики повинні не відрізнятися від реальних з точки зору додатків зовнішнього або високого рівня, але і для інших датчиків або системних модулів, якщо це необхідно. Таким чином, віртуальні датчики можуть подаватися як вхідні датчики для нових віртуальних, що робить гнучкість і потужність цього підходу практично необмеженою.

- Підтримка (вхідних) гетерогенних датчиків. Віртуальний датчик повинен, бути здатний обробляти вхідні датчики зовсім іншого характеру. Це призводить до дуже потужного механізму реалізації складних логік. Інтеграція датчиків, що вимірюють різні дані, може допомогти в реалізації евристик або модулів прийняття рішень на основі штучного інтелекту, здатних обробляти аспекти, які не є однорідними (а не просто статистичні функції над однорідними фігурами). Це також включає автоматичну обробку або перетворення різних одиниць або шкал для вхідних датчиків, що вимірюють один і той же аспект.

1.4 Проблема автономності

Захоплюючі досягнення в області технологій принесли більш складні та великомасштабні комп'ютерні та комунікаційні системи. Автономні розрахунки [12], створені біологічними системами, були запропоновані як виклик, що дозволяє системам самостійно керувати складністю, використовуючи моделі та політики, надані людиною. Мета полягає в тому, щоб забезпечити деяку групу самостійно «*» властивостей системи, де * може бути адаптацією, організацією, оптимізацією, конфігурацією, захистом, відновленням, виявленням, описом і. т. д. Інтернет Речей буде експоненціально збільшувати масштаб і складність існуючих обчислювальних та комунікаційних систем. Таким чином, автономія - обов'язкова властивість для систем IoT. Однак у все ще відсутні дослідження того, як адаптувати існуючі автономні розрахунки до конкретних характеристик IP, таких як висока динамічність та розподіл, в реальному часі, в умовах обмеження ресурсів і середовища з втратами.

Варто відмитити наступні властивості таких систем:

Самоадаптація. У дуже динамічному контексті IoT, від фізичного рівня до рівня додатків, самоадаптація є істотною властивістю, яке дозволяє вузлам, а також службам, які використовують їх, своєчасно реагувати на безперервні зміни контексту відповідно, наприклад, з бізнес-політикою або заданої моделлю поведінки. Системи Інтернету Речей повинні бути здатні міркувати автономно і приймати рішення які адаптуються самі. Когнітивні радіоприймачі, фізичні та канальні рівні, мережеві протоколи що самоорганізуються, автоматичне обслуговування виявлення і (пере) прив'язки на прикладному рівні є важливими компонентами систем Інтернету Речей що адаптуються самостійно.

Самоорганізація. У системах IP дуже часто існують не резидентні вузли. Мережа повинна мати можливість реорганізуватися під поточну топологію. Енергоефективні протоколи маршрутизації що самоорганізуються, мають важливе значення в додатках IoT, щоб забезпечити безперешкодний об-

мін даними всередині сильно гетерогенних мереж. Через велику кількість вузлів, краще розглядати рішення без центральної контрольної точки, наприклад з кластеризацією. При роботі над самоорганізацією також дуже важливо розглянути споживання енергії вузлами і підійти до рішень, які максимізують термін служби системи і ефективність комунікацій в рамках цієї системи

Самооптимізація. Оптимальне використання обмежених ресурсів (таких як пам'ять, пропускна здатність, процесор, і, що найважливіше, потужності) пристроїв IoT необхідно для стійкого і довгострокового розгортання Інтернету Речей. З огляду на деяку оптимізацію на високому рівні з точки зору продуктивності, енергоспоживання або якості обслуговування, сама система повинна вжити необхідних заходів для досягнення своїх цілей.

Самоконфігурація. Системи Інтернету речей потенційно складаються з тисяч вузлів і пристроїв, таких як як датчики і приводи. Тому конфігурація системи дуже складна і її важко встановити вручну. Система Інтернету речей повинна забезпечувати можливість дистанційного конфігурування, а так же бути здатною автоматично налаштовувати необхідні параметри в залежності від контексту. Контекст складається з налаштування параметрів пристрою і мережі, наприклад, установка / видалення / оновлення програмного забезпечення або настройка параметрів продуктивності

Самозахист. Завдяки своєї бездротової і всюдисущої природі Інтернету-речей буде вразливий для численних шкідливих атак. Оскільки Інтернет Речей тісно пов'язаний з фізичним світом, атаки будуть, наприклад, спрямовані на управління фізичної середовищем або отримання приватних даних. Інтернет Речей повинен автономно налаштовуватися на різні рівні безпеки та конфіденційності, не впливаючи на якість обслуговування і простоту експлуатації.

Самовідновлення. Метою цієї властивості є виявлення і діагностика проблем, які можуть виникнути і негайно спробувати виправити їх автономним чином. IoT системи повинні постійно відслідковувати стан своїх вузлів і виявляти коли вони ведуть себе не так як очікується. Потім система може ви-

конувати дії для усунення проблем, що виникають, наприклад реконфигурирование вузлів або перезавантаження, або оновлення програмного забезпечення.

Самоідентифікація. Речі і ресурси (датчики і приводи) повинні бути здатні описати їх характеристики і можливості виразним чином, для того що б можна було з ними взаємодіяти. Необхідні протоколи повинні бути визначені, можливо, на семантичному рівні. Існуючі мови слід перебудувати, щоб знайти компроміс між виразністю, відповідністю і розміром описів. Самооцінка є фундаментальним властивістю для впровадження plug and play ресурсів і пристроїв[13].

Самовиявлення. Разом з самоідентифікацією функція самовиявлення грає важливу роль роль для успішних розгортання IoT. IoT-пристрої / служби повинні бути динамічно відкриті і використовуватися іншими в безшовній і прозорій формі. Тільки потужні і виразні протоколи виявлення пристроїв і сервісів (разом з протоколами опису) дозволить системі IoT бути повністю динамічною.

Самоенергозабезпечення. Самозабезпечення енергією є надзвичайно важливим (і дуже специфічним для Інтернету Речей) для реалізації і розгортання стійких рішень IoT. Методи отримання енергії (сонячна, теплова, вібрація і т. д.) Повинні бути краще батарей, які необхідно регулярно замінювати, і які негативно впливають на навколишнє середовище.

1.5 Проблема безпеки

З концепцією «Інтернет Речей» пов’язані проблеми з безпекою, які визначено в «Дорожній карті стратегічних досліджень та інновацій IERC 2010». Хоча в IoT існує ряд конкретних проблем безпеки, конфіденційності та довіри, всі вони мають ряд поперечних не функціональних вимог:

- Легкі і симетричні рішення;
- Підтримка пристроїв з обмеженими ресурсами;

- Масштабованість до мільярдів пристроїв / транзакцій.

Рішенням необхідно буде вирішити питання про об'єднання / адміністративну співпрацю:

- Неоднорідність і множинність пристроїв і платформ;

- Інтуїтивно зрозумілі рішення, плавно інтегровані в реальний світ.

Оскільки додатки та служби Інтернет Речей будуть масштабироваться за кількома адміністративним доменами та користуватися перевагами кількох режимів володіння, існує потреба в структурі довіри, щоб користувачі системи могли бути впевнені в тому, що інформація і послуги якими користувачі обмінюються є справжніми. Механізм довіри повинен взаємодіяти з людьми і машинами тобто забезпечувати достатню надійність що б люди могли довіряти такому рішенню, а машини могли користуватися рішенням без обмежень. Процес розробки сервісу для забезпечення довіри повининен відповідати наступним вимогам:

- Легка інфраструктура відкритих ключів (PKI) в якості основи для управління довірою.

- Встановлення легких функцій управління ключами і шифрування матеріалів з використанням мінімальних ресурсів зв'язку і обробки, як це пов'язано з обмеженими ресурсами багатьох пристроїв Інтернету Речей.

- Якість інформації є обов'язковою вимогою для багатьох систем на основі Інтернету Речей, де метадані можуть використовуватися для оцінки достовірності даних Інтернету Речей.

- Децентралізовані і самоконфіруючиєся системи в якості альтернативи PKI для встановлення довіри, наприклад. федерація ідентичності, точка-точка.

- Нові методи оцінки довіри до людей, пристроїв і даним, крім систем репутації. Одним із прикладів є Trust Negotiation. Trust Negotiation - це механізм, що дозволяє двом сторонам автоматично погоджувати на основі ланцюжка довірчих політик мінімальний рівень довіри, необхідний для надання доступу до сервісу або до частини інформації

- Гарантійні заходи, для довірених платформ, включаючи апаратні засоби, програмне забезпечення, протоколи і.т.д

- Контроль доступу для запобігання порушень даних. Одним із прикладів є Контроль Використання, який є процесом забезпечення правильного використання певної інформації згідно з визначеною політикою після надання доступу до інформації.

Оскільки Інтернет Речей стає ключовим елементом Майбутнього Інтернету і критичної національної / міжнародної інфраструктури, необхідність забезпечення адекватної безпеки інфраструктури Інтернет Речей стає все більш важливою. Великомасштабні програми та послуги, засновані на Інтернеті Речей, стають все більш уразливими перед збоєм від псування або крадіжки інформації.

Атаки DoS / DDOS вже добре вивчені для поточного Інтернету, але Інтернет Речей також сприйнятливий до таких атак і потребує спеціальних методів і механізмів для забезпечення того, щоб транспортні, енергетичні, міські інфраструктури не могли бути відключені або підірвані. Виявлення атак в цілому, і відновлення / відображення таких атак потрібно для того що б справлятися зі специфічними для Інтернет Речей погрозами, такими як компроментування вузла, атаки за допомогою шкідливого коду. Необхідно розробити інструменти / методи підвищення обізнаності про кібернетичну ситуації, щоб можна було контролювати моніторинг інфраструктури на основі Інтернет Речей. Потрібні інновації, щоб дозволити операторам адаптувати захист Інтернет Речей протягом життєвого циклу системи і допомагати операторам приймати найбільш підходящі захисні заходи під час атак.

Інтернет Речей вимагає різноманітного контролю доступу та пов'язаних схем обліку для підтримки різних моделей авторизації і використання, які потрібні користувачам. Неоднорідність і різноманітність пристроїв / шлюзів, що вимагають контролю доступу, потребують розробки нових полегшених схем. ІВ повинен самостійно обробляти практично всі режими роботи, не покла-

ючись на контроль людини. Нові методи і підходи, наприклад. від машинного навчання, повинні привести до самообслуговування Інтернету Речей.

1.6 Існуючі аналітичні рішення Інтернету Речей

IBM Watson Analytics. «Watson Analytics - це інструмент для аналізу та візуалізації інтелектуальних даних, за допомогою якого ви можете швидко знаходити шаблони та значення у ваших даних - все це самостійно. Завдяки керованим виявленням даних, автоматизованій аналітиці прогнозування та когнітивним можливостям, таким як діалог натурального мови, ви можете взаємодіяти з даними розмовно, щоб отримати відповіді, які ви розумієте. Незалежно від того, чи потрібно швидко помітити тренд, або у вас є команда, яка повинна візуалізувати дані звіту на інформаційній панелі, Watson Analytics вас охоплює.»[14].

Watson прославився після перемоги над Jeopardy в 2011 році, де машина використовувала свій інтелект, щоб допомогти лікарям діагностувати рак легенів. За допомогою машинного навчання, Watson здатний обробляти величезні обсяги даних і відповідати на питання, що цікавлять. Здібності платформи аналізувати природні мови дозволяють Watson робити досить глибокі висновки.

Програмне забезпечення Watson тепер є основою багатьох сторонніх додатків, таких як Wayblazer, рішення для консьєржа подорожей; Sell Smart, мобільний додаток для роздрібного продажу; OrangeHRM, рішення для рекрутерів з відкритим вихідним кодом. Основні функції Watson безкоштовні.

Платформа IBM Watson - це потужне серверне рішення для віддаленої аналітики. Вона відмінно підходить при стабільному і не дорогому інтернет з’єднанні, для рішень які не потребують миттєвої реакції. Платформа являє собою хмарне рішення для аналітики за допомогою машинного навчання і візуалізації даних, а так само SDK (Software Developoment Kit) для надання API (Application Programming Interface) до хмари із мінімальними можливостями

аналітики на переферії - простими арифметичними операціями. Також використання SDK потребує володіння однією з мов програмування, що означає необхідність персоналу з високою кваліфікацією для використання данної платформи. Інтерфейс платформи зображений на рисунку 1.4.

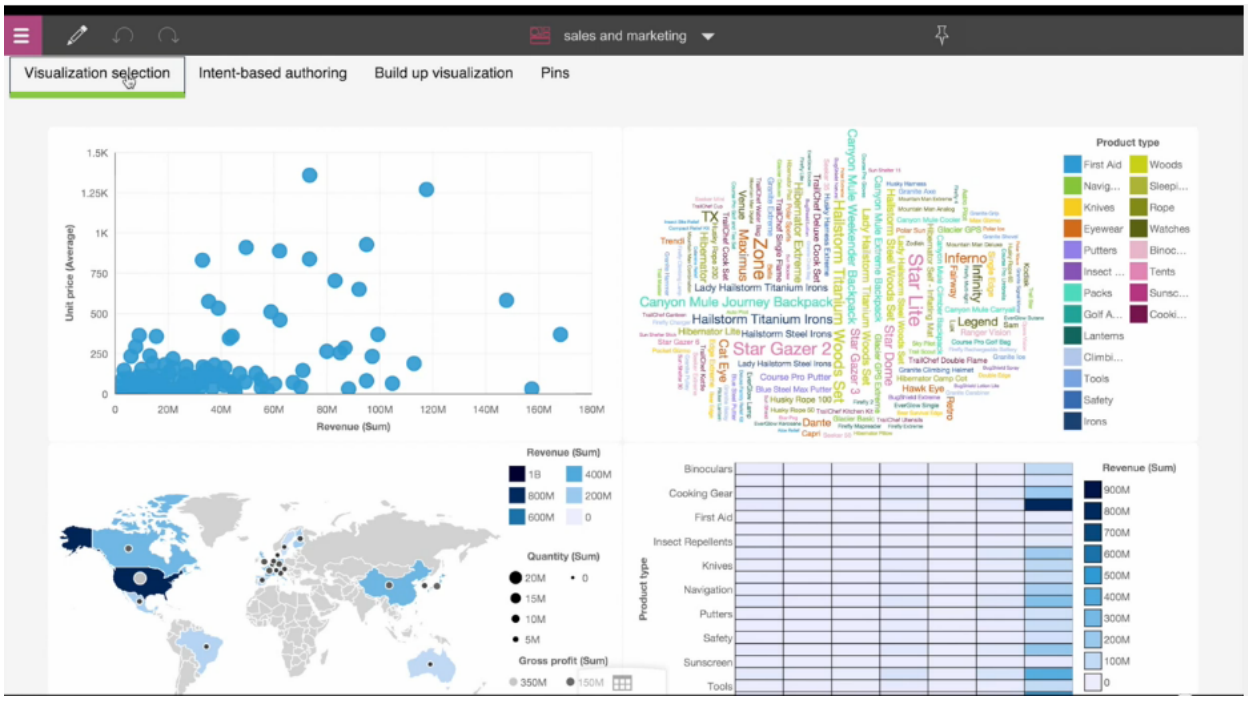

Рисунок 1.4 - IBM Watson Analytics

Amazon Web Services. Amazon Web Services (AWS) являє собою безпечну платформу хмарних сервісів[15], яка надає обчислювальні потужності, доступ до сховищ, баз даних, послуг доставки контенту і іншим функціональним можливостям, що допомагає в масштабуванні і розвитку бізнесу. AWS являє собою платформу для хмарних обчислень. Вся аналітика відбувається на стороні сервера.

AWS являє собою широкий набір інфраструктурних сервісів, таких як надання обчислювальних потужностей, різних варіантів зберігання даних, мережевих рішень і баз даних, пропонованих як послуги: у міру необхідності, з доступністю протягом декількох секунд.

Одними з наданих платформою серевісов є аналітичні сервіси. Для отримання на основі даних важливої аналітичної інформації, придатної до практичного використання, існує широкий спектр технологій, що забезпечують ефективну і економічну роботу з даними з можливістю масштабування. AWS пропонує повний набір сервісів для кожного етапу обробки даних в ході їх аналізу, включаючи зберігання даних, бізнес-аналітику, пакетну обробку, потокову обробку, машинне навчання і оркестрації робочих процесів. Потужність і гнучкість цих сервісів поєднується з простотою використання, тому вони дозволяють легко і швидко перетворити необроблені дані в робочу інформацію. Графічний інтерфейс користувача зображений на рисунку (Рис. 1.5).

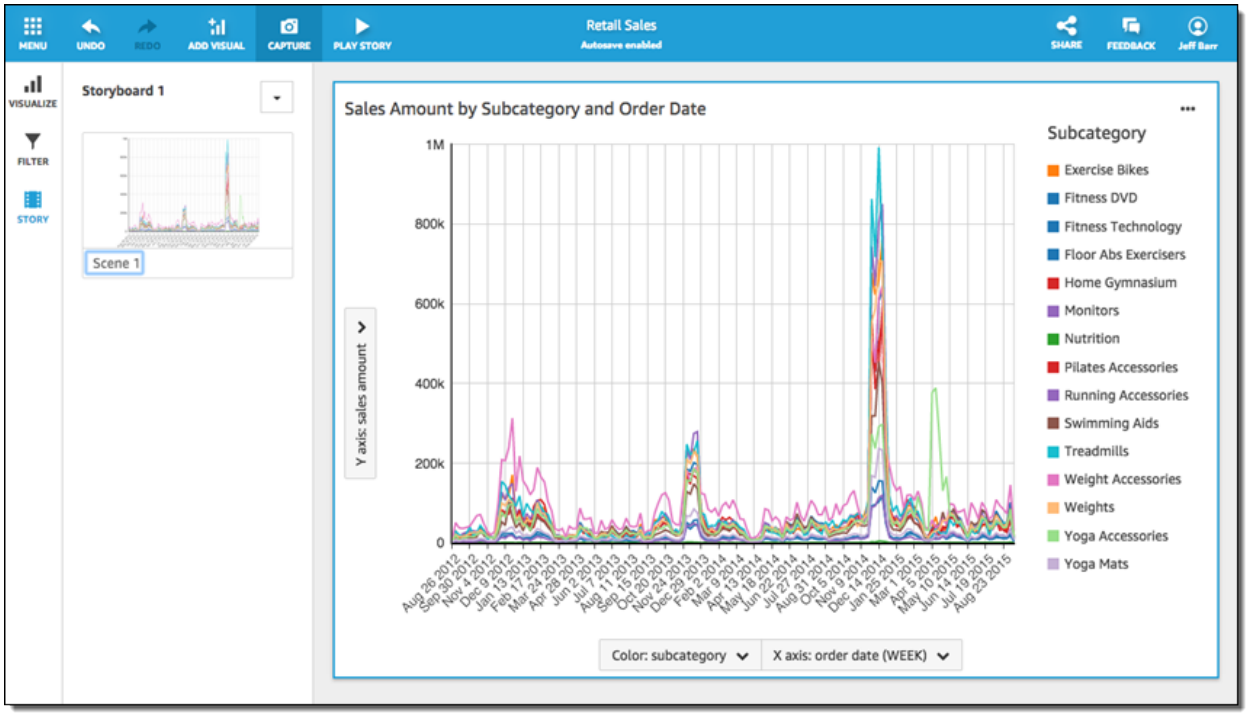

Рисунок 1.5 - Amazon Web Services

Таким чином, беручі до уваги все сказане вище ми бачимо що:

1. Додаткам Інтернету Речей може бути потрібен мінімальний час відгуку, передавати конфіденційні дані і обмінюватися величезними обсягами

інформації, створюючи серйозні навантаження на мережу. Парадигма «Хмарних» обчислень не відповідає таким вимогам.

2. Існуючі аналітичні рішення не дозволяють вести складні аналітичні обчислення без написання великої кодової бази.

3. Головними питаннями є визначення стеку протоколів, створення мінімальної модельної мови конфігурування аналітичної системи яка б дозволяла проводити максимально складні аналітичні обчислення, а також питання автономності пристроїв з обмеженою кількістю ресурсів.

2 ДРУГИЙ РОЗДІЛ

ВИБІР ТЕХНОЛОГІЙ ТА ПРОГРАМНИХ ЗАСОБІВ

Під час розробки комп'ютерної системи вижним питань є вибір технологій які будуть використовуватися для побудови цієї системи. Набір технологій визначається вимогами до продукту (функціональними і не функціональними) а так же наявними обмеженнями.

Функціональне вимоги визначає функцію системи або її компонентів. Функція описується як набір входів, поведінки і виходів [16].

Нефункціональне вимоги - це вимоги, які визначають критерії, які можуть використовуватися для оцінки функціонування системи в цілому, а не конкретну поведінку системи [17].

Таким чином необхідно скласти список функціональних і не функціональних вимог:

Функціональні вимоги:

- Система повинна надавати можливість створювати аналітичні моделі будь-якої складності;
- Система повинна надавати можливість обчислень розподілених у временені;
- Система повинна надавати воможность динамічно додавати джерела даних;
- Система повинна надавати можливість отримувати дані ззовні за допомогою API [18];
- Система повинна відправляти результати аналітики в хмару;
- Система повинна шифрувати дані при використання незахищеного каналу зв’язку;
- Система повинна восстанавлі після втрати зв'язку з збоїв;

Нефункціональні вимоги:

- Система повинна бути простою для розгортання;
- Система повинна споживати мінімальну кількість системних ресур-

сів;

- Система повинна бути максимально швидкою і гнучкою;

- Система повинна бути проста для використання користувачами не володіють високими технічними знаннями;

- Система повинна бути надійною в контексті безпеки конфіденційних даних;

- Система повинна пересилати мінімальну кількість інформації, заощаджуючи мережевий трафік;

Даний список є початковою точкою у виборі стека технологій.

2.1 Стек протоколів

Однією з істотних проблем в IoT є те, як об'єднати «речі» в інтероперабельно режимі, приймаючи до уваги енергетичні обмеження, пам'ятаючи про те, що комунікація є найбільш енергоємною завданням на пристроях.

Протягом останнього десятиліття були випущені RF-рішення для широкого спектру додатків в Інтернеті Речей, що викликано необхідністю інтеграції та низького енергоспоживання. Різні коммитет по стандартизації запропонували кілька різних технологій для зв'язку з низьким електроспоживанням. Найбільш поширені:

• Стандарт IEEE802.15.4 з низьким рівнем енергоспоживання, низькою складністю, низьким і середнім рівнем зв'язку по лінії зв'язку і фізичними рівнями для пристроїв з обмеженими ресурсами.

• Bluetooth low energy (BluetoothLE) - це технологія з низьким рівнем потужності технології Bluetooth, яка в 15 разів ефективніше Bluetooth.

• Технологія Ultra-Wide Bandwidth (UWB) - це нова технологія в області ІВ, яка передає сигнали в набагато більшому діапазоні частот, ніж звичайні системи. UWB, на додаток до своїх комунікаційних можливостей, він може забезпечити високоточне ранжування пристроїв в додатках Інтрернета Речей.

RFID / NFC пропонує різні стандарти, що пропонують менш гнучкі рішення. Карти проксімітірованія можуть зчитуватися тільки з менш 10 см і слідують стандарту ISO 14443, а також є основою стандарту NFC. RFID-Мітки мітки місць, призначені для ідентифікації об'єктів, мають відстань читання, яке може досягати 7-8 метрів.

Проте, інтерфейсні архітектури залишалися традиційними, і в даний час попит в цій області так само нужни інновації. Що стосується мети наднизького споживання, то суперрегенератори виявилися дуже енергоефективними архітектурою, використовуваними для Wake-Up приймачів. Він і залишаються активними при дуже низькому споживанні енергії і можуть генерувати сигнал, щоб розбудити повний / стандартний приймач. У цій області потрібно стандартизація, оскільки сьогодні існують тільки запатентовані рішення, оскільки фактичний приріст на загальному ринку повинен бути значним.

З іншого боку, можна передбачити зниження енергоспоживання RF-приймача з цільової потужністю менше 5 мВт, щоб забезпечити дуже малий форм-фактор і тривалий термін служби батареї. Дійсно, націленість на значення нижче 1 мВт потім забезпечить підтримку систем збору енергії, що забезпечують автономну радіочастотну зв'язок. На додаток до цього вдосконалення також слід передбачити більш легкі протоколи зв'язку, так як часте вимога синхронізації вимагає частої активації RF-зв'язку, що, таким чином, призводить до перевищення енергоспоживання.

Слід також враховувати, що недавні досягнення в області технології КМОП за межами 90 нм, навіть з 65 нм вузлами, призводять до нових парадигм в галузі радіозв'язку. Додатки, що вимагають радіочастотної зв'язку, ростуть так само швидко, як Інтернет Речей, і зараз економічно вигідно пропонувати це рішення для як одне з частин більш широкого рішення. Дане рішення вже використовується для мікроконтролерів в які вже можуть бути вбудовані ZigBee або Bluetooth RF і дане рішення буде розширюватися що б відповідати

вимогам інших додатків, що працюють з сенсорами виробляють великі обсяги даних.

Поступово портативні RF-архітектури спрощують додавання RF-рішення до існуючих пристроїв. Це призведе до високого використання RF-блоками цифрових блоків і обмеження іспользуванія аналогових, таких як пасивні / індуктивні елементів, які споживають кремній, оскільки їх рідко можна переносити з однієї технології на іншу. Проте, потрібно така ж продуктивність, що і архітектурі приймача, щоб ефективно оцифровувати сигнал в ланцюзі приймача або передавача. В цьому напрямку вирішення вибірки смуг пропускання є багатообіцяючими, оскільки сигнал квантів на набагато більш низькій частоті, ніж в рішеннях Nyquist, що пов'язано з глибокою Субдіскретізацією[130]. Тому споживання значно знижується в порівнянні з більш традиційними процесами ранній стадії, де частота дискретизації набагато нижче.

Безперервне квантування також розглядалося як рішення для високої інтеграції та простий переносимості. Це квантування на ранній стадії, але без дискретизації. Отже, немає доданого споживання через тактового сигналу. Ці два рішення є явними кандидатами в еволюції, на шляху до подальших цифровим і портативним RF-рішень.

Очікується, що пристрої з кабельним харчуванням не стануть життєздатним варіантом пристроїв Інтернету Речей, оскільки вони складні і дорогі для розгортання. У багатьох сценаріях розгортання Інтернету Речей, заміна батарей в пристроях недоцільна або дуже дорога. Як наслідок, для великомасштабного і автономного ІВ слід враховувати альтернативні джерела енергії з використанням енергії навколишнього повітря.

Цільова група з розробки інтернет (IETF) розробила альтернативні протоколи для зв'язку між пристроями ІВ з використанням Інтернета Речей, оскільки Інтернет Речей є гнучким і надійним стандартом [19, 20]. Альянс IPSO опублікував різні документи, що описують альтернативні протоколи і

стандарти для рівнів стека Інтернету Речей і додатковий рівень адаптації, який використовується для зв'язку [20-23] між «розумними» об'єктами.

Фізичний рівень і рівень MAC (IEEE 802.15.4)

Протокол IEEE 802.15.4 призначений для забезпечення зв'язку між компактними і недорогими нізкомощнимі вбудованими пристроями, яким потрібен тривалий термін служби батареї. Він визначає стандарти і протоколи для фізичного і MAC рівня стека IP. Він підтримує зв'язок з низьким енергоспоживанням, а також забезпечує низьку вартість і зв'язок на коротких Відстань. У випадках обмежених ресурсів нам потрібен невеликий розмір кадру, низька пропускна здатність і низька потужність передачі. Передача вимагає дуже невеликої потужності (максимум один милливатт), що становить лише один відсоток від того, що використовується в 802.11 або стільникових мережах. Це обмежує діапазон зв'язку. Через обмежену діапазону пристрою повинні працювати спільно, щоб забезпечити маршрутизацію декількох хостів на великі відстані. В результаті розмір пакета обмежений тільки 127 байтами, а швидкість зв'язку обмежена до 250 кбіт / с. Схема кодування в IEEE 802.15.4 має надмірність, що робить комунікацію надійної, дозволяє виявляти втрати і дозволяє повторно передавати втрачені пакети. Протокол також підтримує короткі 16-бітові адреси посилань для зменшення розміру заголовка, накладні витрати на зв'язок і вимоги до пам'яті [24].

Рівень адаптації

IPv6 вважається кращим протоколом для зв'язку в домені Інтернету Речей через його масштабованості і стабільності. Такі громіздкі IP-протоколи з самого початку не вважалися придатними для зв'язку в сценаріях з нізкомощнимі бездротовими каналами, такими як IEEE 802.15.4. 6LoWPAN, акронім для IPv6 в мережах з низьким рівнем потужності бездротової мережі, є дуже популярним стандартом для бездротового зв'язку. Він забезпечує зв'язок з використанням протоколу IPv6 по протоколу IEEE 802.15.4. Цей стандарт визначає рівень адаптації між рівнем зв'язку 802.15.4 і транспортним рівнем. Пристрої 6LoWPAN можуть взаємодіяти з усіма іншими пристроями на базі

IP в Інтернеті. Вибір IPv6 пов'язаний з великим простором адресації, доступним в IPv6. Мережі 6LoWPAN підключаються до Інтернету через шлюз (802.11 або Ethernet), який також підтримує протоколи для конвертації між IPv4 і IPv6, оскільки сьогоднішній розгорнутий Інтернет в основному будується на основі IPv4. Заголовки IPv6 мало малі, щоб входити в невеликій 127-байтовий MTU стандарту 802.15.4. Отже, стиснення і фрагментація пакетів для перенесення тільки важливої інформації - це оптимізація, яку виконує шар адаптації. Зокрема, рівень адаптації виконує наступні три оптимізації, щоб зменшити витрати на зв'язок:

Стиснення заголовка. 6loWPAN визначає стиснення заголовка пакетів IPv6 для зменшення накладних витрат IPv6. Деякі з полів видаляються, тому що вони можуть бути отримані з інформації про рівні каналу або можуть використовуватися спільно через пакети.

Фрагментація. Мінімальний розмір MTU (максимальна одиниця передачі) IPv6 становить 1280 байт. З іншого боку, максимальний розмір кадру в IEEE 802.15.4 становить 127 байтів. Тому нам необхідно фрагментировать пакет IPv6. Це здійснюється адаптаційним рівнем.

Пересилання рівня соеденения. 6LoWPAN також підтримує маршрутизацію, яка виконується на рівні каналу, використовуючи короткі адреси рівня соеденения, замість мережевого рівня. Ця функція може використовуватися для зв'язку в мережі 6LoWPAN.

Мережевий рівень

Мережевий рівень відповідає за маршрутизацію пакетів, отриманих від транспортного рівня. Робоча група IETF Routing over Low Power and Lossy Networks (ROLL) розробила протокол маршрутизації (RPL) для мереж з низьким енергоспоживанням і втратами (LLN).

Для таких мереж RPL є відкритий протокол маршрутизації, заснований на векторах відстані. У ньому описується, як орієнтований спрямований ациклічний граф (DODAG), який будується з вузлами після обміну векторами відстані. Набір обмежень і цільова функція використовуються для побудови

графіка з найкращим шляхом [23]. Цільова функція і обмеження можуть відрізнятися в залежності від вимог. Наприклад, обмеження можуть полягати в тому, щоб уникати вузлів на батарейках або надавати перевагу зашифровані посилання. Цільова функція може бути спрямована на мінімізацію затримки або очікуваної кількості пакетів, які необхідно відправити.

Створення графа починається з кореневого вузла. Корінь починає відправляти повідомлення сусіднім вузлам, які потім обробляють повідомлення і вирішують, приєднатися чи ні, в залежності від обмежень і цільової функції. Згодом вони направляють повідомлення своїм сусідам. Таким чином, повідомлення переміститься до листя і формується граф. Тепер все вузли в графі можуть відправляти пакети вгору до кореня. Алгоритм маршрутизації точка-точка може бути реалізований в такий спосіб: Пакети відправляються загальним предкам, з яких вони переміщається вниз (в сторону листя), щоб дістатися до місця призначення.

Щоб керувати вимогами вузлів до памяті, вузли класифікуються на «зберігаючі» і «не зберігаючі» в залежності від їх здатності зберігати інформацію про маршрутизації. Коли вузли знаходяться в «Не зберігає» режимі і будують спадний шлях, інформація про маршрут приєднується до вхідного повідомленням і перенаправляється далі до кореня. Корінь отримує весь шлях в повідомленні і відправляє пакет даних разом з повідомленням шляху до кінцевого одержувача. Але тут є компроміс, тому що не «зберігаючі» вузли потребують більшої потужності і пропускної здатності для відправки додаткової інформації про маршрут, оскільки у них немає пам'яті для зберігання таблиць маршрутизації.

Транспортинй рівень

TCP не є хорошим варіантом для спілкування в умовах низької потужності, оскільки у нього великі накладні витрати через те, що це протокол, орієнтований на з'єднання. Тому UDP є кращим, оскільки він є протоколом без встановлення з'єднання і має низькі накладні витрати.

Рівень додатків

Рівень додатки відповідає за форматування і представлення даних. Рівень додатки в Інтернеті зазвичай заснований на HTTP. Проте, HTTP не підходить для середовища з обмеженими ресурсами, оскільки він досить змістовний і, отже, несе великі службові дані для синтаксичного аналізу. Було розроблено багато альтернативних протоколів для середовищ Інтернету Речей, таких як CoAP (протокол обмежених додатків) і MQTT (Transport Telemetry Transport Message Queue Transport).

Constrained Application Protocol: CoAP можна розглядати як альтернативу HTTP. Він використовується в більшості додатків Інтернету Речей [25]. Типова схема комунікацій з використанням CoAP зображена на схемі (Рис. 2.1).

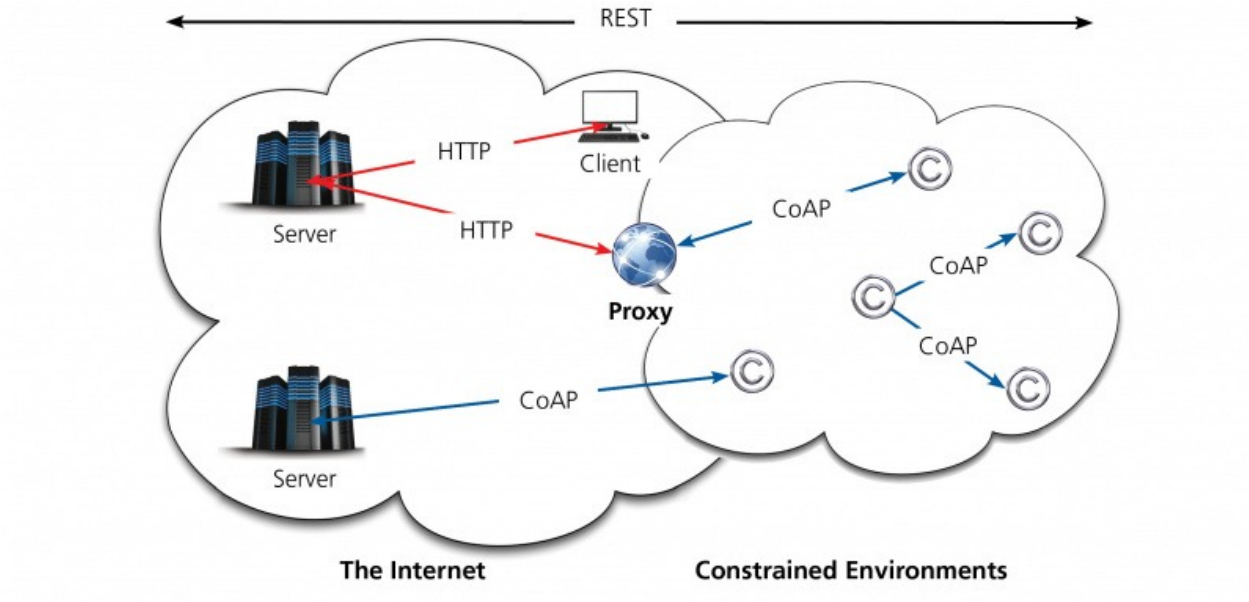

Рисунок 2.1 - CoAP

На відміну від HTTP, він включає оптимізацію для середовища з обмеженим доступом [19]. Він використовує формат даних EXI (Efficient XML Interchanges), який є бінарним форматом даних і набагато більш ефективний з точки зору простору в порівнянні зі звичайним текстовим HTML / XML. Інші

підтримувані функції - вбудоване в стиснення заголовків, виявлення ресурсів, Автом, асинхронний обмін повідомленнями, управління перевантаженням і підтримку багатоадресних повідомлень. У CoAP є чотири типи повідомлень: непідтверджений, який підтверджується, скидання (nack) і підтвердження. Для надійної передачі по UDP використовуються підтверджуються повідомлення. Відповідь може бути скомпонований в самому підтвердженні (piggybacking). Крім того, він використовує DTLS (Datagram Transport Layer Security) для забезпечення безпеки.

Таким чином COAP є протоколом між транспортним та рівнем додатків, який надає транспорті послуги додаткам,

Message Queue Telemetry Transport: MQTT - це протокол публікації / підписки, який працює через TCP. Він був розроблений IBM [27] в першу чергу як клієнт / серверний протокол. Клієнти є видавцями / передплатниками, а сервер виступає в ролі брокера, до якого клієнти підключаються через TCP. Клієнти можуть публікувати або підписуватися на тему. Це повідомлення відбувається через брокера, чия робота полягає в координації підписки, а також аутентифікації клієнта для забезпечення безпеки. MQTT - це легкий протокол, який робить його придатним для додатків Інтернету Речей. Але через те, що він працює через TCP, він не може використовуватися з усіма типами додатків Інтернету Речей. Крім того, він використовує текст для імен тим, що збільшує його накладні витрати. Схема комунікацій з використанням MQTT зображена на рисунку 2.2.

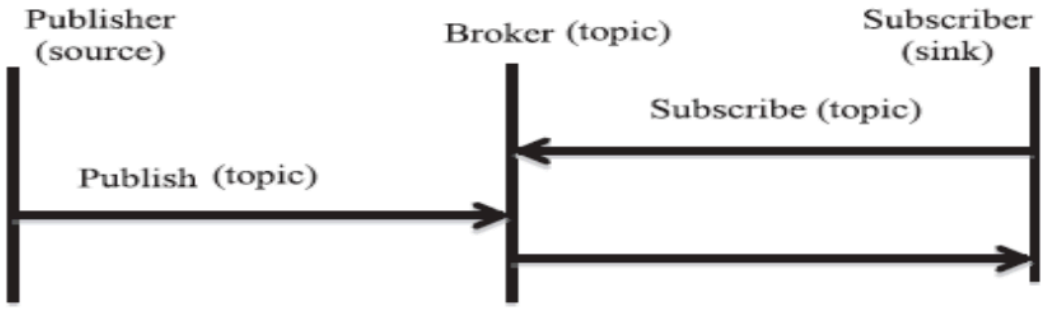

Рисунок 2.2 - MQTT

COAP - це протокол рівня додатка в структурі IOT. Це спеціалізований протокол веб-передачі для вузлів і мереж з обмеженими ресурсами в IoT. Протокол передачі даних, який визначається CAOP специфікацією, заснований на REpresentational State Transfer (REST) поверх функцій HTTP. Він призначений для додатків «машина-машина» (M2M). Його основними рисами є відкритий стандарт IETF, компактний 4-байтовий заголовок, підтримка UDP, SMS, сильний захист DTLS, асинхронна підписка і вбудоване виявлення.

REST забезпечує обмін даними без зберігання стану між клієнтами і серверами по HTTP. REST використовує HTTP методи get, post, put і delete. REST дозволяє клієнтам і серверам надавати і споживати веб-сервіси, протокол простого доступу до об'єктів (SOAP), але більш простим способом використовуючи уніфіковані ідентифікатори ресурсів (URI) як іменники і HTTP-методи отримання, публікації, розміщення і видалення у вигляді дієслів, використовуючи стандартні Internet Media Types для передачі даних. REST не вимагає XML для обміну повідомленнями. CoAP пропонує ряд функцій, яких не вистачає HTTP, таких як вбудоване виявлення ресурсів, підтримка багатоадресної передачі Інтернету Речей, вбудована push-модель і асинхронний обмін повідомленнями. Деякі за харастеристик протоколів Інтернету Речей наведено у таблиці 2.1.

Таблиця 2.1

Application Protocol	**COAP**	**MQTT**	**XMPP**	**AMQP**	**DDS**	**HTTP**
REST	Yes	No	No	No	No	Yes
Transport layer Protocol	UDP	TCP	TCP	TCP	UDP	TCP
Publish/subscribe	Yes	Yes	Yes	Yes	Yes	No

Таким чином, транспортні протоколи Інтернету Речей вирішують більшість з питать з якими зустрічаються розробники, у тому числі питання гетерогенності мереж, безпеки та адресації. З огляду на перераховані протоколи, для поставленої задачі найбільш фунціональним є MQTT, тому основна ко-

мунікацій буде побудована на цьому протоколі.

2.2 Мова програмування та система збирання

Мова програмування для системи вибирається з обмежень і вимог, зокрема описаних на початку розділу, а так само виходячи з оцінки того як він повлеяет на швидкість і якість розробки. Таким чином, виходячи з того що система повинна бути максимально маленька і швидка, а так само споживати мінімальну кількість ресурсів, має сенс розглядати компільовані мови. З популярних мов загального призначення варто виділити два - C і C++.

C - це загальноприйнятна, імперативна мова програмування, підтримує структуроване програмування, область застосування лексичної змінної та рекурсію, а система статичного типу запобігає багатьом непередбачуваним операціям. За конструкцією C передбачені конструкції, які ефективно транслюються в типові машинні вказівки, і тому C використовується у програмах, які раніше писалися на асемблері, включаючи операційні системи, а також різноманітне прикладне програмне забезпечення для комп'ютерів, починаючи від суперкомп'ютерів до вбудованих системах[28].

C++ - це загальноприйнятна мова програмування на основі мови програмування C, як описано в ISO-IEC 9899: 1999 (далі - стандарт C). На додаток до засобів, наданих мовою C, C++ надає додаткові типи даних, класи, шаблони, винятки, простору імен, перевантаження оператора, перевантаження назви функцій, посилання, оператори керування вільною пам’ятью та додаткові бібліотечні засоби[29].

C++ в порівнянні з C надає великі функціональні можливості, додаткові структури даних та методи обробки помилок, проте програма на C ++ займає куди більше місця і для використання, наприклад, механизма виключень вимагає можливість виділити пам'ять в куче, яка може не бути (тому що помилка виникла за причиною нестачі пам'яті). З іншого боку відмовитись від виключень і використовувати мову C++ видається недоцільним, тому що

більшість наданих контейнерів вимагають підтримки виключень, а відмова від контейнерів призведе до того, що буде використано менше 30% мовних можливостей. Таким чином, вигідніше буде використовувати мову С самого початку. Ще важливим фактором є можливість запускати систему на більшості архітектур. Так як мова С була створена досить давно і є де-факто стандартом написання операційних систем, мова компілятора існує практично для кожної операційної системи та / або для кожної архітектури, чого не можна сказати про С++. Даний факт приводить нас до того, що написавши систему на С ми можемо інтегрувати її в ядро для підвищення продуктивності системи (зменшення вартості перемикання контексту).

Таким чином, вибравши мову програмування, ми можемо вибрати систему збирання. Для даної мови найпоширенішою системою збирання є система CNU Make. Утиліта make визначає, які частини великої програми потрібно перекомпілювати, і видає команди для перекомпіляції. Типовий make файл зображений на рисунку 2.3.

```
edit : main.o kbd.o command.o display.o \
       insert.o search.o files.o utils.o
        cc -o edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o

main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
```

Рисунок 2.3 - Приклад make файла

У лівому стовпчику перераховані різні цілі. Праворуч від знака двокрапка перераховані вхідні дані для того що б зібрати ціль. Даний make файл демонтсрірует скільки потрібно написати коду що б зібрати фінальну ціль edit. До того ж для цієї цілі використовується мала кількість вихідних файлів. Якщо поглянути на make файли для збірки ядра Linux [30], можна побачити досить довгі і заплутані make файли в яких важко розібратися. Тому останнім часом все частіше застосовується генератор make файлів, система збирання CMake.

CMake[31] - це міжплатформенна сімейство інструментів, призначених для побудови, тестування та пакетного програмного забезпечення з відкитим вихідним кодом. CMake використовується для керування процесом компіляції програм, використовуючи прості незалежні конфігураційні файли платформи та компілятори, а також генерування власних файлів make і робочих середовищ, які можуть бути використані у вибраній середовищі компілятора. Набір інструментів CMake був створений Kitware у відповідь на необхідність створення потужного, міжплатформенного середовища для створення проектів з відкритим кодом, таких як ITK та VTK. Інтеграція з CMake в даний час є у більшості відомих IDE. Типовий CMake файл представлений на рисунку 2.4.

```
cmake_minimum_required(VERSION 3.2)
project(acompiler CXX)

  add_executable(acompiler
        src/lexical_analyzer/lexical_analyzer.cpp
        src/syntax_analyzer/syntax_analyzer.cpp
        src/main.cpp)
    target_compile_options(acompiler PUBLIC
        "-Wall;-Wpedantic;-Werror")

set_property(TARGET acompiler PROPERTY CXX_STANDARD 14)
set_property(TARGET acompiler PROPERTY CMAKE_CXX_EXTENSIONS OFF)
set_property(TARGET acompiler PROPERTY CMAKE_CXX_STANDARD_REQUIRED ON)
```

Рисунок 2.4 - Приклад CMake файла

Як можна побачити, даний код являтся інтуїтивно зрозумілим, а так же легко розширюваним. Кожна ціль компіляції має спеціальну команду, яка однозначно ідентифікує ціль. Важливою властивістю CMake є здатність генерувати кросплатформенних make файли. Таким чином написана однажни система складання може збирати проект для різних операційних систем. Ще одна важлива властивість даного генератора - підтримка кросскомпіляції. Дана властивість досягається за допомогою поддежкі спеціальних файлів конфігурації, так званих тулчейнов (toolchains). Так виглядає типовий toolchain файл (Рис. 2.5).

```
set(CMAKE_SYSTEM_NAME Linux)

set(CMAKE_C_COMPILER arm-linux-gnueabihf-gcc)
set(CMAKE_CXX_COMPILER arm-linux-gnueabihf-g++)

#Set target environment directory
set(CMAKE_FIND_ROOT_PATH  $ENV{INSTALLDIR}/arm-linux-gnueabihf/libc/usr)
```

Рисунок 2.5 - Приклад toolchain файла

В даному файлі встановлюються необхідні компілятри для цієї ціль і необхідні бібліотеки.

Таким чином, CMake є кроссплатформенной системою збирання, яка дозволяє створювати складні рішення.

2.3 Цільова операційна система

Важливим аспектом є вибір цільової операційної системи, так як саме інструменти цільової ОС будуть базисом для написання системи. Різні операційні системи надають різний набір методів (бібліотек) для взаємодії з буфером мережевої карти, таймерами, введенням-виведенням і.т.д. Так, напри-

клад, операційні системи сімейств Windows надають досить різнорідний API для взаємодії, а сімейство UNIX-подібних систем намагаються відповідати (хоч і не в повній мірі) стандарту POSIX.

Так само вибір цільової ОС визначає те, наскільки багато ресурсів буде у пристроїв, таким чином обмежуючи кількість потенційних пристроїв на яких буде запускатися система.

Доступність операційної системи так само відіграє важливу роль. В даний час наберать тенденцію використання операційних систем з відкритим вихідним кодом, так як вони піддаються конфігурації і кастомізації. Так само вони є безкоштовними і часто ліцензія дозволяє використовувати такі ОС для комерційних рішень.

Однією з таких систем з відкритим вихідним кодом є Linux. Дана операційна система була розроблена Лінусом Торвальдсом у відповідь на відмову Ендрю Таненбаума додати більше компонентів в ядро MINIX. Дана система є однією з найбільш використовуваних систем для пристроїв з обмеженими ресурсами, в той же час надаючи повноцінну функціональність операційної системи. Архітектура ядра даної ОС монолітна. Linux здатний працювати на більшості архітектур, таких як ARM, X86, PowerPC, MIPS і.т.д. В даний час більшість виробників мікроконтролерів випускають свої дистрибутиви на основі ядра Linux. Одним з яскравих прикладів є Raspbian [32] для Raspberry PI.

Так само дана операційна система здатна працювати при наявності всього декількох мегабайт оперативної пам'яті, що робить її дуже привабливою і, можливо, найбільш використовуваною операційною системою в світі мікроконтролерів з об'ємом пам’яті більше 1мб.

Дистрибутиви Windows є досить громіздкими, що не стандартизованими і, що не менш важливо, платними. Ядро даної ОС неможливо розширити (крім як встановити драйвер), а вихідний код є недоступним. Це робить дану ОС неприйнятною для використання і побудови рішень на мікроконтролерах.

WXWorks є ОС реального часу. Базовим обмеженням є закритість даної ОС, однак вона була розроблена для вбудованих пристроїв, тому вона є підходящим кандидатом на роль цельової ОС.

Так як система що розробляється повинна працювати на максимально великій кількості пристроїв, цільової ОС буде Linux. Однак WXWorks є ОС, яка багато в чому сумісна з Linux, так що можлива розробка кількох платформо-залежних реалізацій модулів і портирование розроблюваної системи.

2.4 Сторонні бібліотеки

Для того щоб прискорити розробку часто використовуються сторонні (допоміжні бібліотеки). Сторонній компонент програмного забезпечення являє собою багаторазовий програмний компонент, розроблений як для вільного розподілу, так і для продажу іншим суб'єктом, відмінним від початкового постачальника платформи розробки. Ринок сторонніх бібліотек забезпечення процвітає, оскільки багато програмістів вважають, що компонентний орієнтований розвиток покращує ефективність та якість розробки користувацьких додатків. Загальне стороннє програмне забезпечення включає в себе макроси, боти та програмне забезпечення / скрипти, які запускаються як надбудова для популярного програмного забезпечення.

Так як в якості транспортного протоколу був обраний MQTT, необхідна бібліотека для серіалізації / десеріалізациї повідомлень які передаються між сервером і клієнтом. Існує безліч реалізацій даного протоколу, однак через обмеження по пам'яті варто розглядати тільки бібліотеки розроблені для пристроїв з малою кількістю ресурсів. Серед таких реалізацій Paho-embedded-c, та WolfMQTTT. WolfMQTT є бібліотекою, яка підходить для пристроїв без операційної системи, так як в бібліотеці велика увага приділяється платформної частини, яку потрібно імплементувати. Однак у в нашому випадку, в якості платформи є Linux, отже у нас є додатковий рівень абстракції над платформою, що спрощує розробку.

Paho-embedded-c. Проект Paho був створений для забезпечення масштабованих та відкритих джерел реалізації стандартних протоколів обміну повідомленнями, спрямованих на існуючі та нові програми для M2M та IoT.

Paho відображає притаманні фізичні та економічні обмеження підключення пристрою. Цілі включають ефективний рівень відокремлення між пристроями та додатками, розроблені з метою збереження відкритих ринків та сприяння швидкому зростанню масштабованого проміжного програмного забезпечення. Paho реалізував клієнтські версії MQTT для публікації / підписки для використання на вбудованих платформах.

Репозиторій містить 3 бібліотеки: MQTTPacket, MQTTClient та MQTT-Client-C. Перша - це бібліотека досить низького рівня на C, яка реалізує синтаксичний аналіз пакетів MQTT та серіалізацію. MQTTClient - бібліотека C++, яка реалізує протокол MQTT за допомогою бібліотеки MQTTPacket. MQTTClient-C реалізує протокол MQTT на мові C. Крім того, MQTTClient-C містить реалізацію рівня платформи для Linux, FreeRTOS і TI CC3200. Рівень платформи включає мережеві та таймерні інтерфейси.

Розмір статичних бібліотек на платформі X86_64:

libpaho-embedded-c-client.a:

- Загальний розмір: 10970 байт;
- .text - 4702 байтів;
- .rodata - 288 байт.

libpaho-embedded-c-packet.a:

- Загальний розмір: 41338 байт;
- .text - 14437 байт;
- .bss - 8 байт;
- .data - 120 байт;
- .rodata - 912 байт.

libpaho-platform-linux.a:

- Загальний розмір: 4886 байт;
- .text - 1561 байт.

Дана бібліотека підтримує стандарти MQTT 3.1 і MQTT 3.1.1. Система збирання написана на make, тому її доведеться переписати для більш безшовної інтерграції з системою.

WolfMQTT. Розміри цієї бібліотики дуже маленькі, порівляно навіть з paho-embedded-c.

libwolfmqtt.a:

- Загальний розмір: 15010 байт;
- .text - 14250 байт;
- .data - 752 байт;
- .bss - 8 байт.

Але ця бібліотека не може бути використана у комерційному продукті, тому що вона розповсюджується під ліцензією GPLv2, що регламентує відритість похідного коду продукта де використовується дана бібліотека. Навідміно відь цього, paho-embedded-c має ліцензію Eclipse, яка дозволяє використовувати дану бібліотеку у комерційних цілях за умови збереження ліцензії, тому для розробки буде використана саме ця бібліотека.

Тому як Інтернет є ненадійним і небезпечним в сенсі кібербезпеки, слід використовувати додаткові механізми забезпечують аутентифікації, авторизацію і безпеку даних. Одним з таких механізмів є Transport Layer Security Protocol (TLS)[33].

Основна мета протоколу TLS полягає у забезпеченні конфіденційності та цілісності даних між двома програмами що обмінюються даними. Протокол складається з двох шарів: протоколу TLS Record і TLS Handshake Protocol. На найнижчому рівні, розміщеному на вершині деякого надійного транспортного протоколу (наприклад, TCP), розташованTLS Record Protocol. TLS Record Protocol забезпечує безпеку з'єднання, що має дві основні властивості:

- З'єднання приватне. Симметрична криптографія використовується для шифрування даних (наприклад, AES та ін.). Ключі для цього симетричного шифрування формуються однозначно для кожного з'єднання і засновані на секретному узгодженні за допомогою іншого протокола (наприклад, TLS Handshake Protocol). Record Protocol також може використовуватися без шифрування.

- З'єднання надійне. Транспортне повідомлення включає в себе перевірку цілісності повідомлення за допомогою ключового MAC. Захищені хеш-функції (наприклад, SHA-1 тощо) використовуються для обчислень MAC. Record Protocol може працювати без MAC, але зазвичай використовується лише в цьому режимі, а інший протокол використовує Record Protocol як транспорт для узгодження параметрів безпеки.

TLS Record Protocol використовується для інкапсуляції різних протоколів вищого рівня. Один такий інкапсульований протокол, TLS Handshake Protocol, дозволяє серверу та клієнту автентифікувати один одного та вести переговори з алгоритмом шифрування та криптографічними ключами, перш ніж протокол програми передає або отримує свій перший байт даних. TLS Handshake Protocol забезпечує безпеку з'єднання, що має три основні властивості:

- Ідентифікацію пірів можна перевірити за допомогою криптографії асиметричного або відкритого ключа (наприклад, RSA, DSA та ін.). Ця автентифікація може бути необов'язковою, але, як правило, потрібна принаймні одному з пірів.

- Переговори про загальний секрет є безпечними: конфіденція, що піддається обговоренню, недоступна для підслухувачів, і для будь-якого автентифікованого з'єднання - таємниця не може бути отримана навіть зловмисником, який може розмістити себе посередині з'єднання.

- Переговори є надійними: атакувач не може змінити комунікацію без виявлення цього факту сторонами комунікації.

Ще однією з переваг TLS є незалежність протоколу додатків. Протоколи високого рівня можуть прозоро наповнювати протокол TLS. Стандарт TLS, однак, не вказує, як протоколи додають безпеку за допомогою TLS; рішення про те, як ініціювати рукостискання TLS, і про те, як інтерпретувати обмінні сертифікати автентифікації, залишаються на розсуд розробників та розробників протоколів, які працюють над TLS. Використання TLS для забезпечення безпеки комунікації з MQTT брокером є загальноприйнятим підходом, рекомендованим в стандарті в секції «Security». Ім'я сервісу IANA - secure mqtt, стандартний порт - 8883.

Для інтеграції TLS необхідно вибрати допоміжну бібліотеку. Існує кілька варіантів, однак орієнтування на максимальну продуктивність і мінімальний розмір робить цей вибір меньшим. Таким чином, досить велика бібліотека OpenSSL, розмір якої:

- libcrypto.a - 5,1 Мб;
- libssl.a - 772 Кб.

становить майже шість мегабайт, що, набагато більше самого додатка. Такі накладні витрати є неприпустимими. У той же час існує бібліотека, розроблена спеціально для вбудованих пристроїв - MbedTLS. Її розмір:

- libmbedcrypto.a - 616 Кб;
- libmbedtls.a - 312 Кб;
- libmbedx509.a - 123 Кб.

Весь функціонал займає близько одного мегабайта, що майже в шість разів менше ніж OpenSSL.

MbedTLS є бібліотекою з відкритим вихідним кодом і комерційною бібліотекою SSL, ліцензованою компанією ARM Limited. MbedTLS раніше називалась PolarSSL, яка розпочалася як офіційне продовження ще одного проекту, названого XySSL. Так же MbedTLS поддерживает различные варианты систем сборки, такие как make, cmake, msvs, что делает ее максимально портируемой. Множество функционала возможно выключить во время компиляции при помощи соответсвующих деректив системы сборки и препро-

цессора. Это делает использование библиотеки более «дешевым» в отношении используемой памяти.

2.5 Система контролю версій

Системи контролю версій - це категорія програмних засобів, які допомагають команді що розробляє програмне забезпечення керувати та слідкувати за змінами вихідного коду протягом всього періоду розробки. Програмне забезпечення для контролю версії відстежує кожну модифікацію коду в базі даних спеціального виду. Якщо сталася помилка, розробники можуть повернути годинник і порівняти попередні версії коду, щоб легше знайти та усунути помилку, мінімізучи зусилля всіх членів команди.

Майже для всіх програмних проектів вихідний код є дорогим активом, вартість якого має бути захищеною. Для більшості команд що розробляють програмне забезпечення, вихідний код є сховищем знань та розуміння проблемної області, яку розробники зібрали та вдосконалили, доклавши зусилля. Контроль версій захищає вихідний код як від катастрофи, так і випадкової людської помилки та непередбачених наслідків.

Розробники програмного забезпечення, що працюють у командах, постійно пишуть новий вихідний код та змінюють існуючий вихідний код. Код для компонента проекту або програмного забезпечення, як правило, організований у структуру директорій або "дерево файлів". Один розробник у команді може працювати над новою функцією, а інший розробник виправляє не пов'язану помилку, змінивши код, кожен розробник може внести зміни в декілька частин дерева файлів.

Контроль версій допомагає командам вирішити ці проблеми, відстежуючи кожну індивідуальну зміну кожним учасником та допомагаючи запобігти конфліктуванню одночасної роботи. Зміни, внесені в одній частині програмного забезпечення, можуть бути несумісними з тими, зробленими іншим розробником, який працює одночасно. Ця проблема повинна бути виявлена та

вирішена належним чином, не блокувавши роботу іншої команди. Крім того, у всіх розробках програмного забезпечення будь-яка зміна може вводити нові помилки самостійно, і нове програмне забезпечення не може бути довірено до його тестування. Тому тестування та розробка продовжуються разом, поки не буде готова нова версія.

Якісне програмне забезпечення для керування версіями підтримує бажаний робочий процес розробника, не накладаючи жодного конкретного способу роботи. В ідеалі система контролю версій також працює на будь-якій платформі, а не диктує, яку саме операційну систему або тулчейн розробники повинні використовувати. Великі системи керування версіями полегшують плавний і безперервний потік змін до коду, а не розривний і незграбний механізм блокування файлів - надання зеленого світла одному розробнику за рахунок блокування прогресу інших.

Команди розробників, які не використовують жодної форми керування версією, часто зіштовхуються з такими проблемами, як не знаючи, які зміни є остаточно фінальним варіантом або які зміни повинні бути залишені із двох не пов’язаним між собою модифікацій файлів, які потім повинні бути ретельно розкриті та перероблені. Контроль версій - це вихід із цих проблем.

Програмне забезпечення для контролю версій є невід'ємною частиною повсякденного професійного досвіду команди сучасного програмного забезпечення. Індивідуальні розробники програмного забезпечення, які звикли працювати з функціональною системою контролю версій у своїй команді, зазвичай визнають, що управління версіями є цінним інструментом навіть у невеликих проектах.

Системи керування версіями (VCS) помітно поліпшилися за останні кілька десятиліть, а деякі - краще, ніж інші. VCS іноді називають інструментами SCM (Source Code Management) або RCS (система управління версіями). Незалежно від того, як вони називаються, або яка система використовується, основними перевагами, які слід очікувати від керування версіями, є наступні:

- Повна довгострокова історія змін кожного файлу. Це означає кожну

зміну, зроблену багатьма людьми протягом багатьох років. Зміни включають створення та видалення файлів, а також редагування їх вмісту. Різні інструменти VCS відрізняються від того, наскільки добре вони обробляють перейменування та переміщення файлів. Ця історія також повинна включати автора, дату та письмові зауваження з метою кожної зміни. Маючи повну історію, можна повернутися до попередніх версій, що допомогає у аналізі основних причин помилок, і це має вирішальне значення для вирішення проблем у старих версіях програмного забезпечення;

- Розгалуження та злиття. Наявність членів команди одночасно працюючими над різними задачами, зрозуміло вимагає використовування системи контролю версій, але навіть особи, що працюють самостійно, можуть отримати користь від можливості працювати на незалежних потоках змін. Створення "гілки" в інструментах VCS тримає декілька потоків роботи незалежними один від одного, а також забезпечує можливості об'єднати цю роботу разом, дозволяючи розробникам перевіряти, що зміни в кожній галузі не конфліктують. Багато програмних колективів застосовують практику розгалуження для кожної функції або, можливо, розгалуження для кожного комміту або обидва. Існує безліч різних робочих процесів, котрі можуть вибрати команда, коли вони вирішують, як використовувати розгалуження та об'єднання об'єктів у VCS.

- Відстеження. Можливість відстежувати кожну зміну, внесену в програмне забезпечення, та підключити її до програмного забезпечення для управління проектами та відстеження помилок, такими як Jira, а також можливість коментувати зміни повідомленнями, що описує цілі та наміри зміни, що може допомогти не тільки при аналізу причин проблем. Маючи анотовану історію коду під рукою, коли ви читаєте код, намагаючись зрозуміти, що вона робить і чому вона розроблена, розробники зможуть дозволити правильним та гармонійним змінам, які узгоджуються з передбаченим довгостроковим дизайном. системи. Це може бути особливо важливо для ефективної роботи зі спадковим кодом і має вирішальне значення для того,

щоб дозволити розробникам оцінити майбутню роботу з будь-якою точністю.

Один з найпопулярніших інструментів VCS, що використовуються сьогодні, називається Git. Git - розподілений VCS, категорія, відома як DVCS.. Як і більшість популярних VCS-систем, доступних сьогодні, Git є безкоштовним та з відкритим вихідним кодом.

Є багато вдалих практик які використовуються у різноманитних проектах. Однією з таких практик є стратегія описана Винсентом Дриссеном[34] (Рис. 2.6).

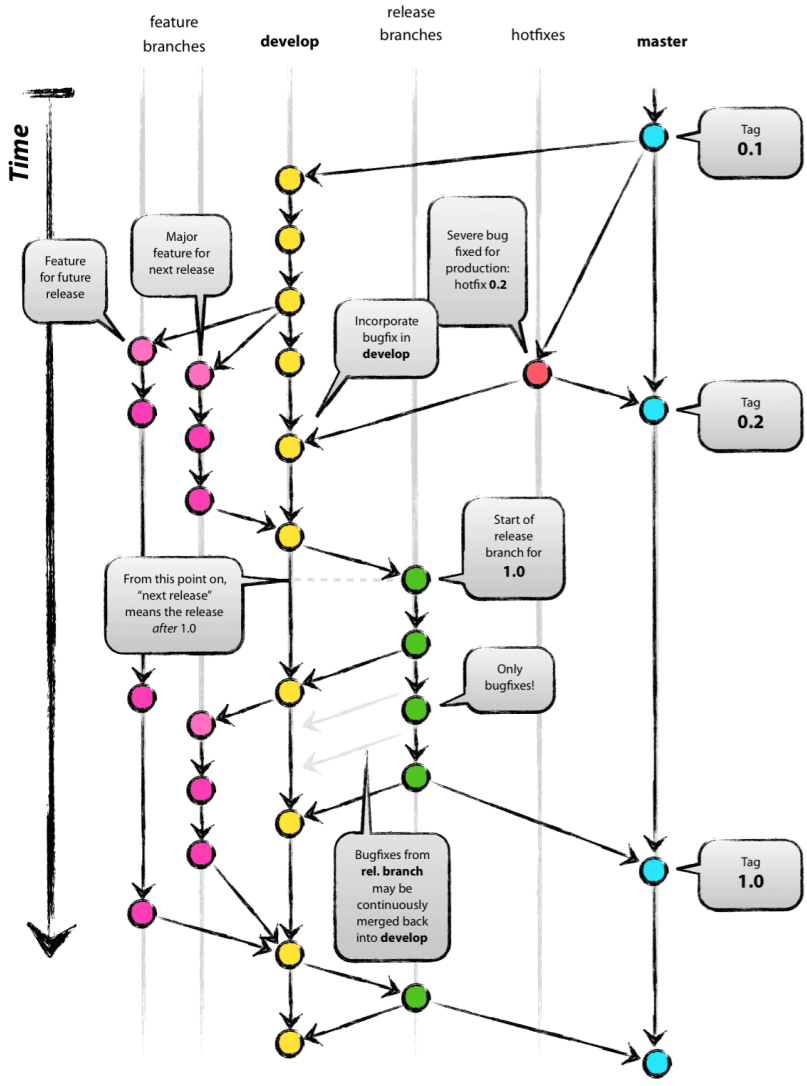

Рисунок 2.6 - A successful Git branching model

Багато розподілених ресурсів для розміщення коду розроблюють та рекомендують свій процес роботи з кодом. Так, наприклад Github рекомендує використовувати Github Flow, але це не є обмеженням та розробники програмного забезпечання можуть сами обирати чи використовувати їм Github flow чи будь-якій інший спосіб слідкування за похідним кодом. Github flow зображен на рисунку 2.7.

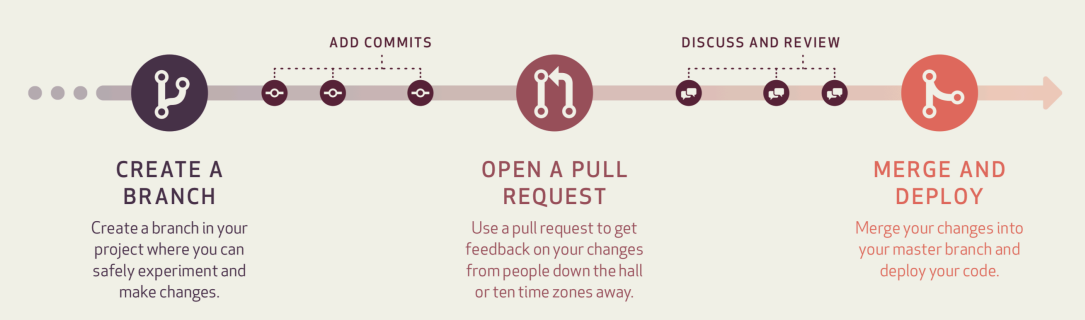

Рисунок 2.7 - Github flow

Незважаючи на те, що можна розробляти програмне забезпечення без використання будь-якої системи керування версіями, це робить проект об'єктом величезного ризику. Отже, найбільш популною системою контролю версій на теперішній час є Git. Він буде використовуватись при розробці проекту.

3 ТРЕТІЙ РОЗДІЛ

РОЗРОБКА АНАЛІТИЧНОЇ СИСТЕМИ «ТУМАННИХ ОБЧИСЛЕНЬ» ІНТЕРНЕТУ РЕЧЕЙ

3.1 Розробка сценаріїв комунікацій і схем комунікацій

Таким чином, визначившись з вимогами і списком програмних засобів та технологій, які будуть використовуватися в процесі розробки системи, можна приступити до розробки архітектури.

Для початку необхідно визначити механізми і схеми коммуніцій. Базова схема комунікацій відрізняється не значною мірою від стандартної схеми «Хмарний обчислень», вона зображена на рисунку 3.1:

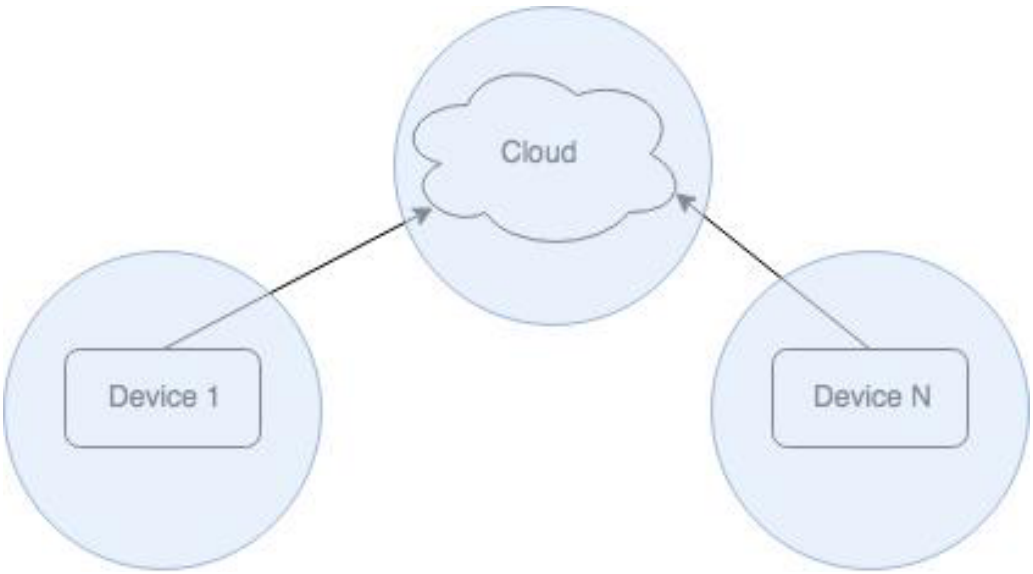

Рис 3.1 - Стандартна схема комунікацій

Синіми напівпрозорими колами позначені зони де можуть відбуватися обчислення. Стрілками позначений потік даних. Даний потік є стандартним для «Хмарних» обчислень. Дані обробляються або на пристроях на яких вони

виникають і / або відправляються в хмару для (подальшої) обробки. Даних підхід є загальноприйнятим але не є достатньо гнучким. Наприклад в тому випадку якщо сервер не може обчислити що-небудь або не може провести обробку даних певним чином через відсутність необхідного контексту. В даному випадку розумним і єдино можливим варіантом буде покласти це завдання на той вузол, який володіє необхідною інформацією. Таким чином, потік даних тепер виглядає наступним чином (Рис. 3.2):

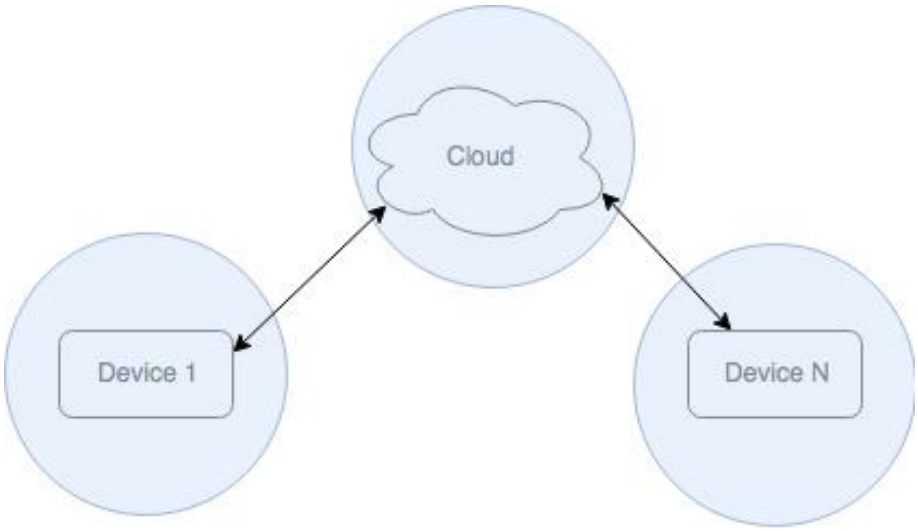

Рисунок 3.2 - Поток даних «Cloud-Device-Cloud»

Тепер дані йдуть в двох напрямках. Однак схожа ситуація може виникнути у будь-якого вузла мережі, а не тільки у сервера. Таким чином кожному з вузлів мережі потрібно мати можливість передавати дані іншим вузлам мережі для того що б забезпечити контекст для розрахунків. Таким чином ми маємо справу зі зв'язком все з усіма. Однак на даному етапі виникає питання: якщо дані генеруються одним (фізичним) пристроєм, а аналітика проводиться на іншому, то якого з пристроїв належать дані і як їх ідентифікувати? Якщо мова йде про два пристроя, то зберегти інформацію про ці пристрої не буде складним завданням, але коли коли таких пристроїв в теорії може бути безліч, виникає скрутне становище, тому що протягом усього ланцюга повинна бути

збережена метаінформація про кожного з пристроїв в ланцюжку, що вносить додаткові накладні витрати в протокол передачі даних. Дана ситуація показана на рисунку 3.3.

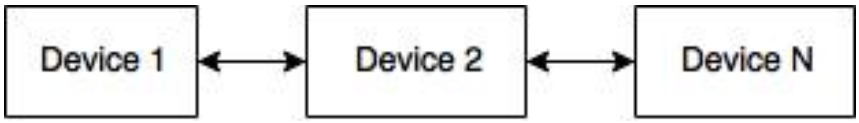

Рисунок 3.3 - Ланцюг обміну даних

На даній схемі видно що якщо Device 1 відсилає дані Device 2, а той в свою чергу відсилає дані далі по ланцюжку, в кінцевому підсумку дані приходять Device N, який відсилає дані до «Хмари», то інформація про джерело і всіх проміжних ланках повинна бути збережена і передана до «Хмари» разом із даними. Найпростішим рішенням даної проблеми є обмеження ієрархії. Можна дозволити всього один рівень, тоді буде потрібно зберігати інформацію всього про один пристрій, а в гнучкості архітектури втрат не буде. Дана схема зображена на рисунку 3.4.

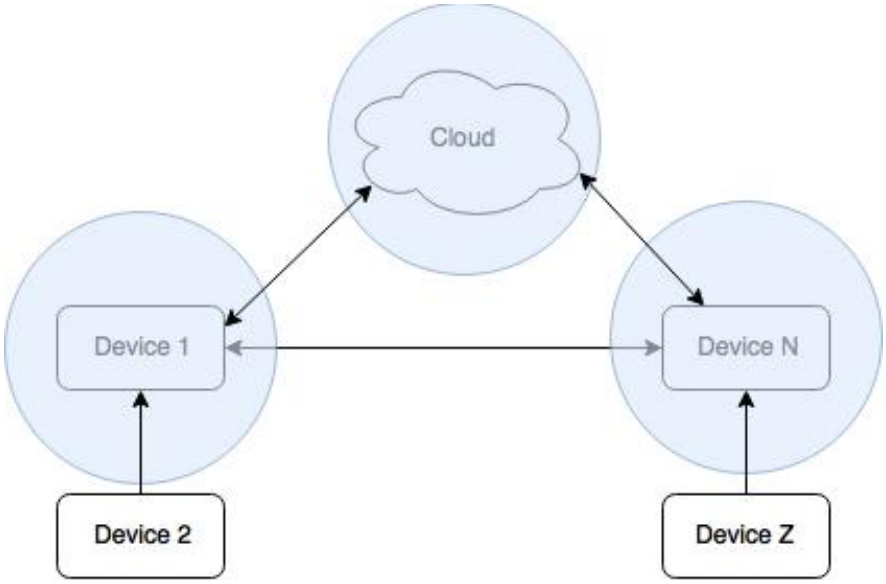

Рисунок 3.4 - Двохрівнева коммунікаційна модель

Таким чином, після того як пристрої що мають зв’язок безпосередньо з «Хмарою» відішлють дані на сервер, «Хмара» знову може надіслати дані до одного з вузлів для подальших обчислень, а вузол що прийняв дані знову відправить дані в «Хмару». Дана процедура може відбуватися мнократно в залежності від кількості пристроїв що приймають участь в аналітиці. Це допомагає уникнути додаткових рівнів ієрархій і як наслідок - скорочення накладних витрат на передачу метаінформації про вузли що приймали участь в аналітиці.

Маємо максимально гнучку схему аналітики. Приклади застосування:

- Борт літака, є чотири турбіни реактивної тяги, кожна з котрих виступає окремим пристроєм. Кожна з турбін пов'язана з бортовим центром управління (Другим аналітичним вузлом в ієрархії). Турбіни не знають про існування один одного. У кожної з турбін є ряд показників, таких як витрата палива, температура. Використовуючи дані з усіх турбін можливо обчислити середню витрату і середню робочу температуру по даному класу турбін. На підставі цих даних може бути запущена аналітика. Маємо: Кожна турбіна відсилає середня витрата палива і середню температуру за період часу T і відсилає дані в вищестоящий аналітичний вузол. Аналітичний вузол вважає устредненние показники по даному літаку і відправляє дані в «Хмара». «Хмара» порівнює дані з показниками за всіма турбін даної моделі встановлених на інших літальних апаратах і може виявити проблему або знайти аномалію в даних. Таким чином ми бачимо трирівневу схему аналітики.

- Маршрутизатор аналізує сигнал, кількість втрачених і відправлених пакетів. Як тільки виявлено зростання кількості втрачених пакетів він відправляє дані про цю подію в «Хмару». Сервер може провести аналіткику по всіх пристроях з даної прошивкою і проалізіровать виникає дана проблема на подібних моделях і / або на даній версії програмного забезпечення, таким чином локалізуючи проблему. Дана схема предствляет собою дворівневу модель аналітики: перший рівень - налітіка на маршрутизаторі, другий - аналітика в «Хмарі».

3.2 Встановлення програмного забезпечення та програмних компонентів

Для розробки системи необхідно встановити відповідне програмне забезпечення. Розробка буде вестися на операційній системі OS X High Sierra, тому для установки основних програмних стредств буде використаний пакетний менеджер home brew. Встановимо компілятор gcc, текстовий редактор emacs, систему збирання CMake і систему контролю версій git.

Для цього відкриємо термінал і виконаємо команду brew install git cmake gcc emacs (Рис. 3.5).

Рисунок 3.5 - Встановлення програм

Після того як команда буде виконана, перевіримо чи програмні стредства видно системі, таким чином переконавшись в тому що установка була виконана коректно. Для цього виконаємо в емуляторі терміналу команду which git cmake gcc emacs (Рис. 3.6).

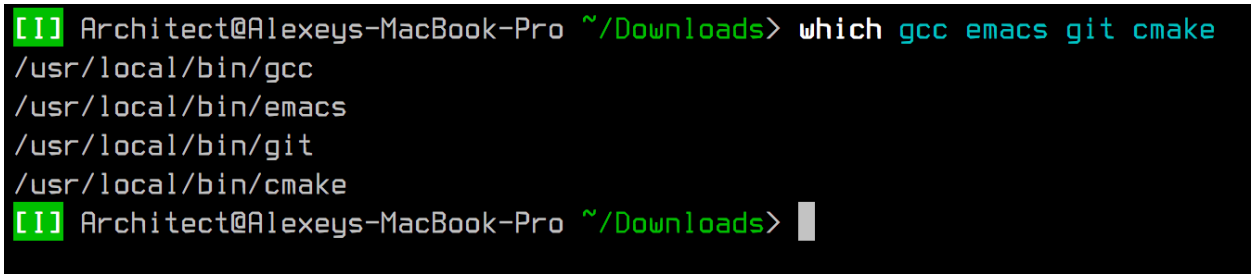

Рисунок 3.6 - Перевірка програмних засобів

Бачимо що все програмні стредства видно системі, таким чином ми встановили всі необхідні для роботи програми. Далі заватажимо всі необхідні бібліотеки, які ми визначили в другому розділі даної роботи. Всі необхідні бібліотеки мають відкритий вихідний код, який можна знайти на ресурсі github. Всі бібліотеки будуть поставлятися разом з вихідним кодом самої системи, так як вони є досить маленькі і це дозволить адаптувати і змінювати вихідний код бібліотек в разі потреби.

Для того що б завантажити всі необхідні бібліотеки скористаємося раніше встановленої утилітою git. Для початку зайдемо на сайт github.com і скопіюємо посилання для завантаження з потрібних репозиторіїв. Приклад для paho-embedded-c показаний на рисунку (Рис. 3.7).

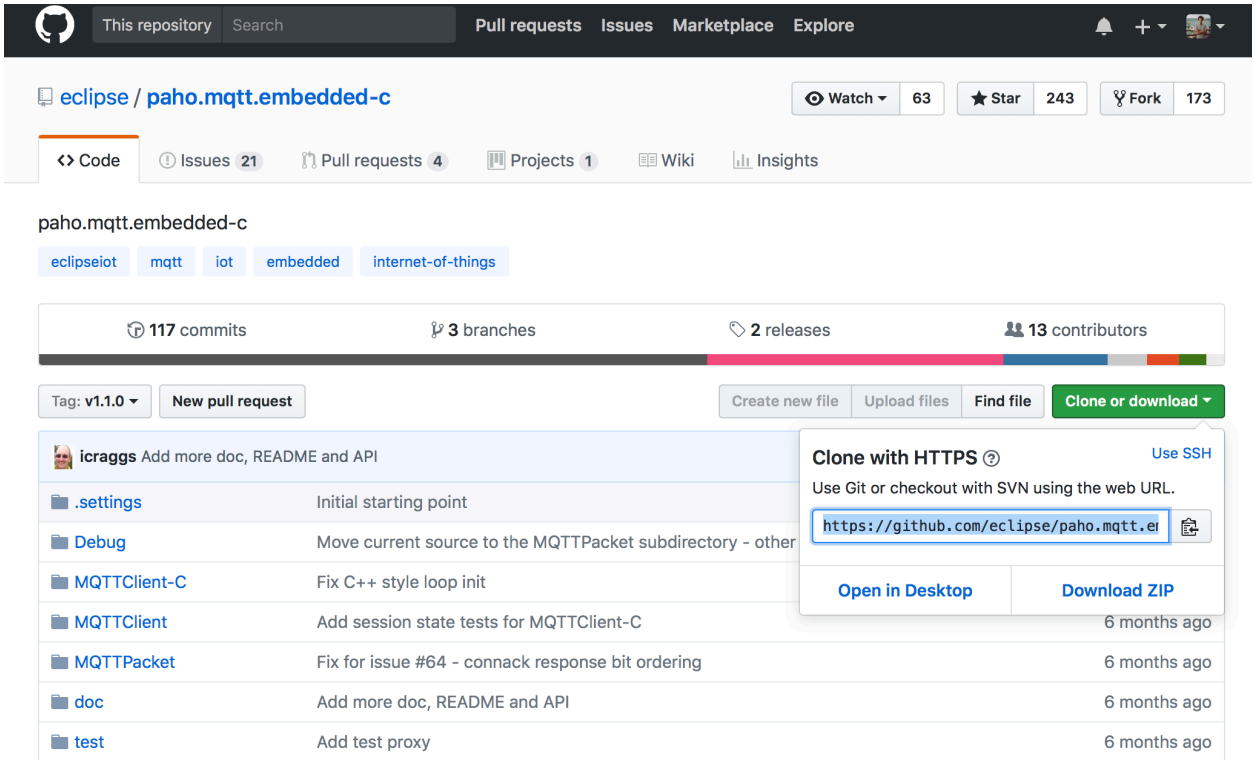

Рисунок 3.7 - Сторінка paho-embedded-c

Натиснемо кнопку «Clone or download». Відкрилося вікно в якому є посилання, скопіюємо дане посилання. Посилання яке було тільки що скопійо-

вано використовується для завантаження вихідного коду бібліотеки. Ревізія коду визначається номером комітів, який відображений на рисунку в лівому верхньому кутку (v.1.1.0) поряд з кнопкою «New pull request».

Щоб завантажити код, необхідно передати дане посилання на вхід програмі git (Рис. 3.8).

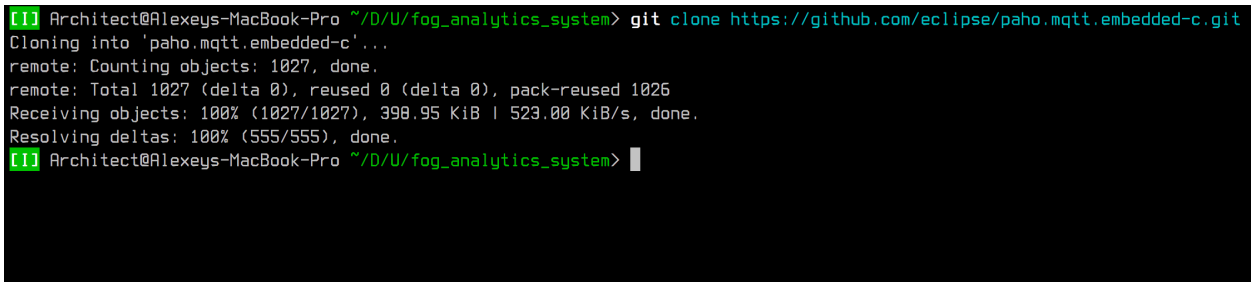

Рисунок 3.8 - Завантаження paho-embedded-c

Таким самим чином завантажемо решту бібліотек.

3.3 Розробка центральної системи обробки та управління подіями

Кожна програма має систему управління життєвим циклом програми, в якій обробляються або генеруються нові події. Як архітектура даної системи управління життєвим циклом так і події в ній відрізняються від системи до системи. Наприклад існують архітектури які в яких події приходять із зовні (ядро операційної системи відповідає за передачу сигналу про будь-яку подію за допомогою сигналу програмі) або ж програма сама опитує всі можливі джерела подій (Поллинг).

Так само система управління подіями може працювати синхронно або ж асинхронно. Асинхронна і синхронна обробка відрізняються досить сильно. У порівнянні з синхронною архітектурою, асинхронна, як правило, дає досить великий приріст в продуктивності. Зокрема всі операції введення-виведення в більшості операційних систем є асинхронними і їх обробка відбувається за виникнення переривань. Дана концепція дозволяє використовувати

ресурси процесора і комп'ютера в цілому в той час як виконується, наприклад передача або прийом даних. Так як операції введення-виведення на кілька порядків повільніше ніж, наприклад, обробка даних (обчислення за допомогою процесора, графічної карти і т.д), планувальники в операційних системах, як правило, вважають за краще давати процесам обмеженим операціями введення-виведення максимальний пріоритет, тому як вони будуть заблоковані в очікуванні завершення операції введення-виведення довше ніж використовувати процесорний час, таким чином процесор звільнитися швидше і інші процеси, ограниченине процесорним часом отримають свій квант в ремени швидше. Якби процеси обмежені операціями введення-виведення запускалися останніми, то інші процеси отримували б значно більше процесорного часу ніж перші..

Існує безліч функцій в різниї операційних системах для реалізації асинхронного введення-виведення на багатьох рівнях. Насправді, одна з основних функцій всіх, крім самих елементарних операційних систем, полягає в тому, щоб виконати хоча б деяку форму базового асинхронного введення-виведення, хоча це може бути не особливо очевидним для програміста. У найпростішому програмному рішенні стан апаратного пристрою періодично перевіряється, щоб визначити, чи готове пристрій до наступної операції. (Наприклад, була побудована операційна система CP/M. В її семантиці системних викликів не була потрібна більш складна структура введення-виведення, хоча більшість реалізацій були більш складними і, отже, більш ефективними.) Прямий доступ до пам'яті (DMA) може значно підвищити ефективність системи на основі опитування, а апаратні переривання можуть повністю виключити необхідність опитування. Багатозадачні операційні системи можуть використовувати функціональність, що забезпечується апаратними перериваннями, одночасно приховуючи складність обробки переривань від користувача. Буферизація була однією з перших форм багатозадачності, призначених для використання асинхронного введення-виведення. Нарешті, багатопотокові і явні асинхронні API-інтерфейси введення-виведення в призначених для

користувача процесах можуть додатково використовувати асинхронний ввід-висновок за рахунок додаткової складності програмного забезпечення.

Як згадувалося, механізми і функції варіюються від операційній системі до операційної системи, але в цілому існують два підходи: цикл опитування і сигнали.

Сигнали. Операції введення-виведення відбуваються асинхронно, а коли вони завершуються, генерується сигнал (переривання). Як і в низкорівневому програмуванні ядра, ресурси, доступні для безпечного використання в обробнику сигналу, обмежені, і основний потік процесу може бути перерваний майже в будь-якій точці, що призводить до неузгодженим структурам даних, як це видно оброблювачу сигналів. Оброблювач сигналів зазвичай не може самостійно проводити подальше асинхронний введеня-виведення. Хоча сигнали є простою системою для реалізації всередині операційної системи, вони привносять додаткову складність в реалізацію прикладної програми, пов'язану з написанням коду для обробки переривань операційної системи. У найгіршому сценарії кожен синхронний зв'язок буде перервано сигналом і функції синхронних операцій при таких переривання повинні викликатися повторно в таких випадках.

Полінг. Використовується Unix-подібних системах, і майже всі, включаючи стек протоколів TCP/IP, будується поверх даного рішення. Цикл опитування, використовує системний виклик select для сплячого режиму до тих пір, поки на файловому дескрипторі не виконуватиметься умова (наприклад, коли дані стануть доступні для читання), відбувається тайм-аут або приймається сигнал (наприклад, коли дочірній процес завершується). Перевіряючи параметри виклику select що повертаються, цикл виявляє, який дескриптор був змінений і виконує відповідний код. Часто для простоти використання цикл вибору реалізується як цикл подій, можливо, використовуючи функції зворотного виклику; даний спосіб особливо добре підходить до програмування заснованого на подіях.

Хоча цей метод є надійним і відносно ефективним, він багато в чому

залежить від парадигми Unix, що «все є файлом»; будь-які блокоючі операції введення-виведення, що не використовують файловий дескриптор, блокують процес. Цикл вибору також залежить від можливості задіяти всі операції введення-виведення централізовано; бібліотеки, які здійснюють свої власні операції введення-виведення, є особливо проблематичними в цьому відношенні. Додатковою потенційною проблемою є те, що операції вибору і введення-виведення, як і раніше досить розв'язані, що результат вибору може бути не правдивим: якщо два процеси читають з одного дескриптора файлу (можливо, через погану архітектуру), вибір може вказувати на доступність читання даних, які вже були прочитані іншим процесом до моменту, коли був здійснений системний виклик, що призводить до блокування; якщо два процеси записуються в один дескриптор файлу, перевірка може вказувати на безпосередню можливість запису, але запис все одно може блокуватися, оскільки буфер був заповнений іншим процесом за проміжний період або через занадто великі обсягі даних на запис.

Цикл вибору не досягає максимальної ефективності системи, наприклад, за допомогою методу черг завершення, оскільки семантика виклику витрачається певний обсяг часу на виклик і обхід масиву подій. Інші асинхронні методи можуть бути помітно більш ефективними в таких випадках. Деякі операційний системи надають системні виклики з кращим масштабуванням; наприклад, epoll в Linux (який заповнює масив вибору повернення тільки тими джерелами подій, на яких відбулася подія), kqueue у FreeBSD і MacOS і порти подій (and /dev/poll) в Solaris.

Таким чином модуль життєвого циклу системи визначається можливостями операційної системи. Так як цільово операційною системою був обраний Linux, варто використовувати epoll. Як уже згадувалося, дана функціональність дозволяє досягти оптимальної продуктивності при побудові систем заснованих на подіях.

Epoll виконує аналогічне завдання моніторингу декількох файлових дескрипторів, щоб побачити, чи можливе використання введення-виведення для

будь-якого з них. API-інтерфейс epoll може використовуватися або як інтерфейс спрацьовують по подіям або рівню і добре масштабується для великої кількості спостережуваних файлових дескрипторів.

Обговоривши механізм, можна розглянути деталі реалізації. Так як система повинна бути максимально гнучкою, події повинні очікуватися не тільки із зовні, а й повинна бути можливість генерувати події (можливо, за таймером). Це дозволить зчитувати дані із заданою частотою. Наприклад датчик температури може видавати значення раз в секунду, але для того що б не витрачати просто трафік, значення можуть надсилатися раз в хвилину, якщо ми знаємо що швидкість зміни температури така що дозволяє зчитувати зчитувати значення з певною частотою. Це можна реалізувати, наприклад, зберігаючи тільки останнім значення з датчика до тих пір поки спрацює таймер. Як тільки таймер спрацює, останнє значення може бути відправлений у «Хмара». Дана схема зображена на діаграмі (Рис. 3.9).

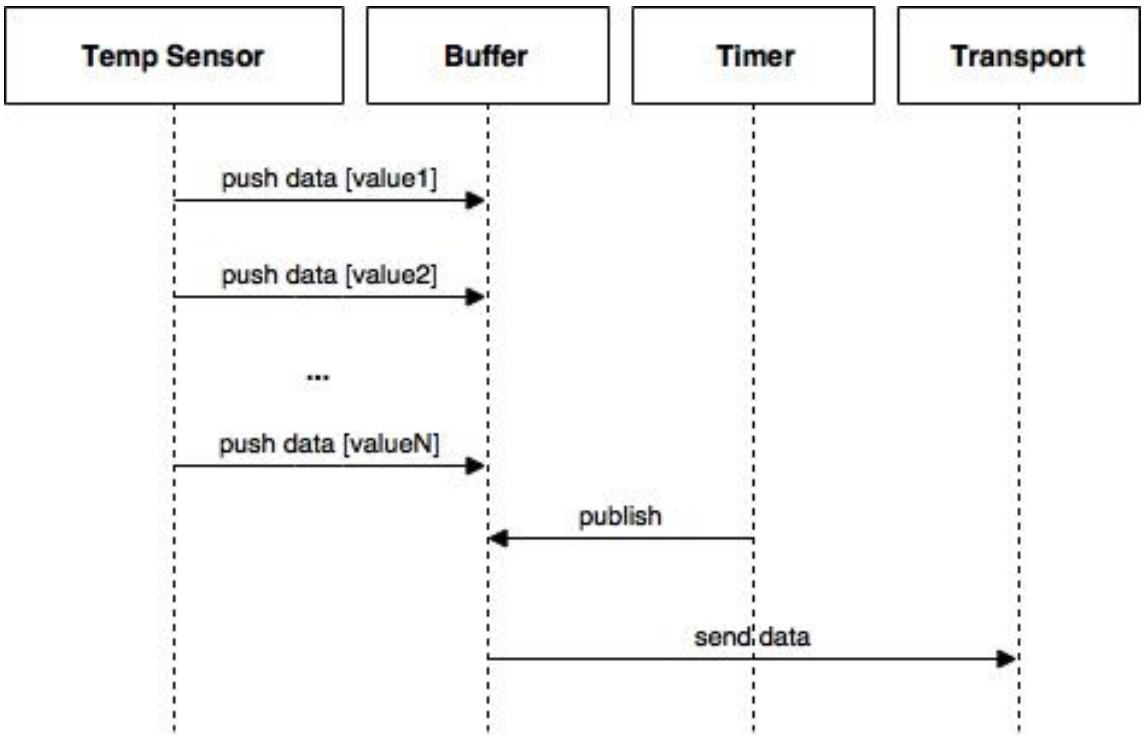

Рисунок 3.9 - Діаграмма життєвого циклу системи

Temp Sensor поміщає дані в буффер Buffer. По закінченню деякого заданого інтервалу часу спрацьовує таймер Timer і дані відсилаються в «Хмару».

При розробці алгоритму уравленія подіями слід враховувати те, що за браком зовнішніх подій або спрацьовувань таймерів програма повинна знаходиться в стані блокування - у такий спосіб не споживати процесорний час. Алгоритм зображений на блок-схемі (Рис. 3.10).

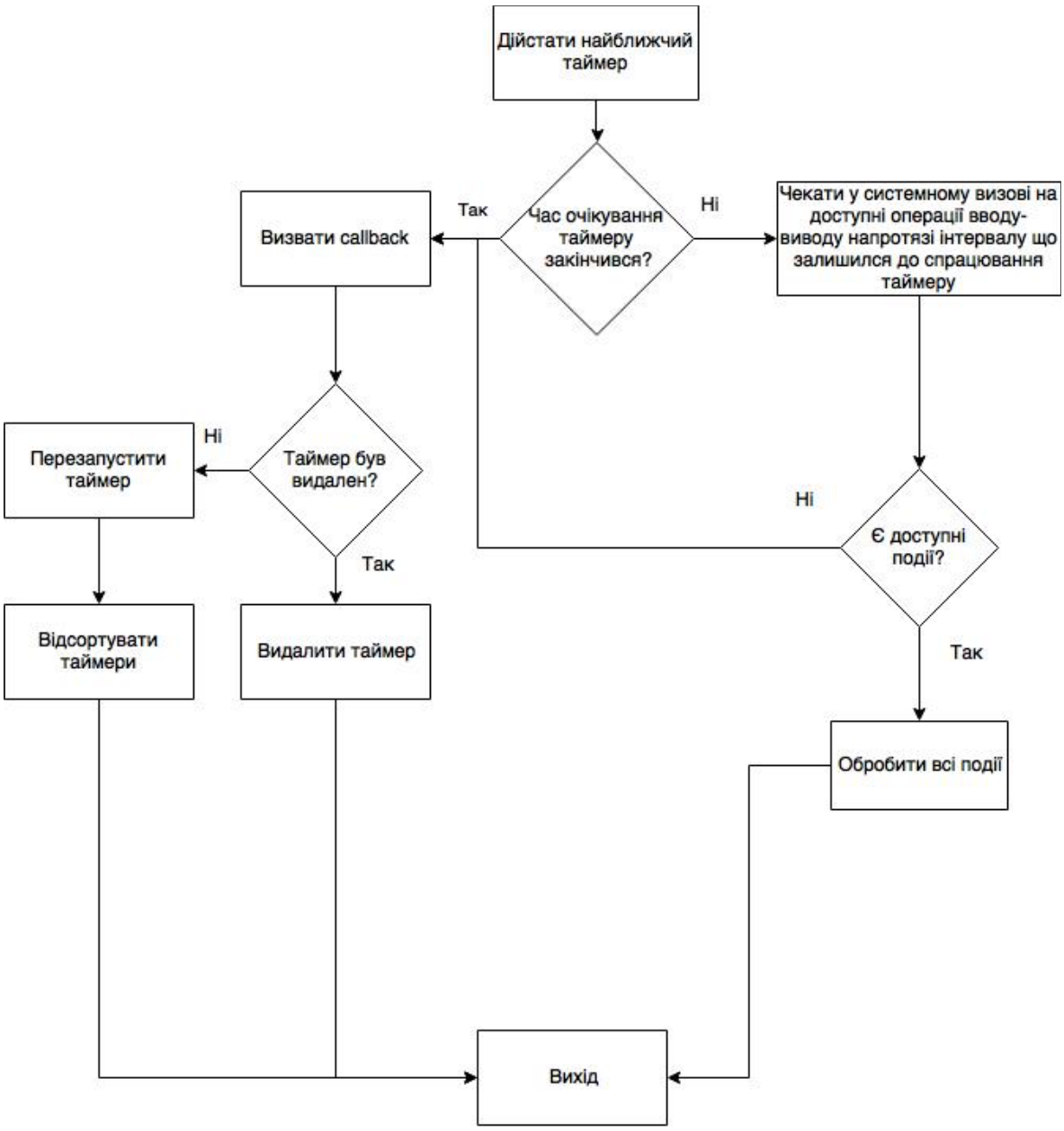

Рисунок 3.10 - Алгоритм життєвого циклу системи

Як видно зі схеми, всі таймери повинні бути відсортовані в порядку закінчення інтервалу часу. Це необхідно для того що б не пропустити час спрацьовування таймера, таким чином дістаючи таймер час якого закінчиться першим. Якщо час минув, то можна викликати оброблювач прив'язаний до таймера. В іншому ж випадку ми очікуємо, що залишився інтервал часу в системному виклику, чекаючи появи подій від epoll. Якщо система повідомляє про наявність події, відбувається обробка подій і вихід з функції. В іншому випадку ми прочекали залишився інтервал часу таймера, а це означає що таймер закінчився і можна викликати оброблювач таймеру.

Для даного модуля створимо окрему директорію, яку назвемо loop. Дана директорія буде містити два файли самого модуля управління подіями loop.c і loop.h, а так же інтерфейс event_observer.h, реалізації якого буде забезпечувати взаємодію з операційною системою (epoll/kqueue і.т.д). Концептуально в термінах ООП діаграма класів виглядає наступним чином (Рис. 3.11).

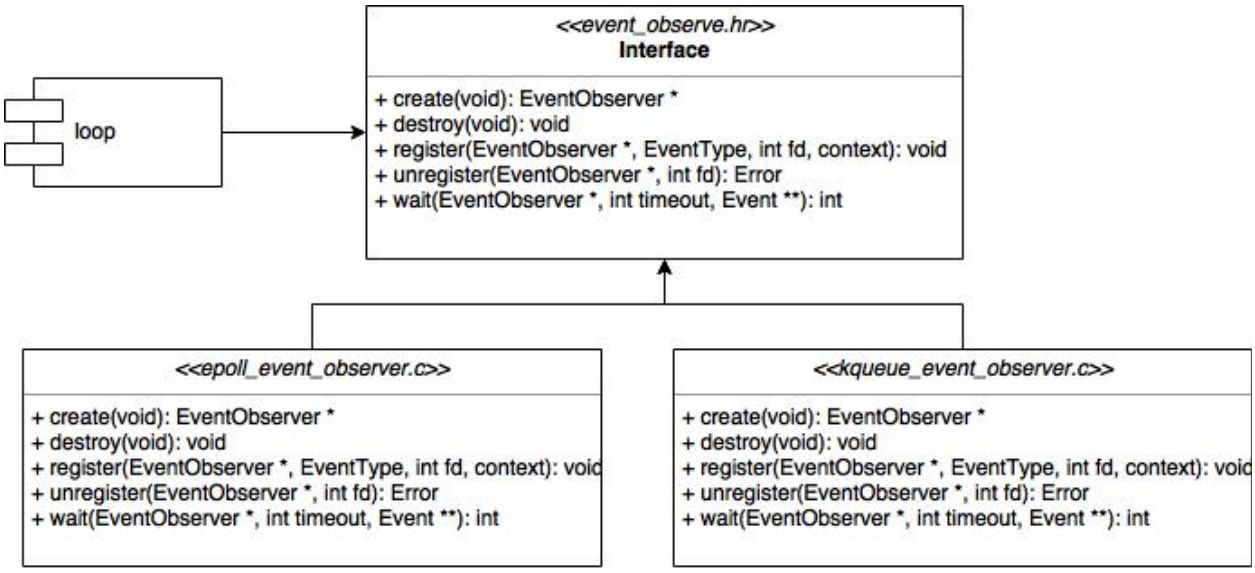

Рисунок 3.11 - UML діаграма відносин

Дана архітектура допомагає абстрагуватися від конкретної операційної

системи, таким чином дозволяючи легше перенести цю систему на інші операційні системи якщо це знадобиться. Наприклад одна з реалізацій інтерфейсу event_observable.h може використовувати цикл select. Завдяки такой реалізації систему можна буде використовувати на операційній системі VxWorks.

Модуль loop використовує функцію wait () для того що б отримати масив подій. Кожна одиниця масиву подій містить інформацію про подію (вхідна, вихідна подія або джерело було закрито), інформацію про джерело і користувацькі дані необхідні для обробки події. Таким чином ми абстрагувалися від парадигми Unix-подібних систем «Все є файл». Маємо модуль (Рис. 3.12).

```
[I] Architect@Alexeys-MacBook-Pro ~/D/w/r/r/a/src> tree loop/
loop/
├── epoll_event_observer.c
├── event_observer.h
├── kqueue_event_observer.c
├── loop.c
└── loop.h

0 directories, 5 files
```

Рисунок 3.12 - Модуль Loop

Необхідна реалізація буде обрана на етапі компіляції.

3.4 Розробка модуля аналітики

Другим важливим компонентом є модуль аналітики, так як даний модуль відповідає за програмовану бізнес-логіку. Тобто даний модуль буде виконувати аналітику згідно заданої схеми. Однак для того що б спроектувати модуль, необхідно визначити його функціональність.

Велика частина аналітики будується на арифметичних операторах або стандартних арифметичних функціях. Кожен з операторів і функцій характеризується арністю і типом вихідного значення, або множиною до якої нале-

жить результат (натуральні числа, раціональні, цілі і т.д). До операторів відносяться такі арифметичні дії як «додавання», «віднімання», «ділення» і «множення». До функцій відносяться такі арифметичні функції як «корінь з числа», «абсолютна величина», «середнє значення» і т.д.

При розробці необхідно виходити з принципу реалізації мінімальної функціональності, яка б дозволяла створювати складні аналітичні вирази і додавати складніший функціонал в міру необхідності, якщо для його реалізіції за допомогою простих виразів потрібно значне зусилля. Таким чином для реалізації найпростіших арифметичних дій, необхідно реалізувати чотири арифметичних оператора (Табл. 3.1).

Таблиця 3.1

Оператор	Арність
Додавання	2
Множення	2
Віднімання	2
Ділення	2

Перераховані вище оператори хоч і дозволяють зробити практично все, однак багато речей, такі як обчислення кореня квадратного, робляться за допомогою даних операторів досить не тривіальним способом. Має сенс розширити дану функциональність і надати користувачеві можливість використовувати такого роду функції «із коробки». Список базових функцій представлений в таблиці 3.2.

Таблиця 3.2

Функція	Арність
Зведення в ступінь	2
Корінь квардратний	1
Абсолютне значення	1
Середнє значення	∞

Так з таблиці ми бачимо що функції відрізняються арністю, це означає що ні всі оператори можуть бути комбіновані між собою. Наприклад функції «Зведення в ступінь» потрібно два аргумента, це означає що вона може приймати на вхід результати виконання двох операторів/функцій або результат виконання оператора/ функції за умови що в якості результату виступає масив що містить два числа.

Однак виникає кілька питань: Що робити якщо, наприклад, значення для бінарного оператора повинні бути отримані від одного і того ж оператора який повертає тільки одне значення? Що робити, якщо потрібно порахувати середнє значення по на певній кількості чисел? На даному етапі варто задуматися про архітектуру і програмної реалізації даних операторів, це дозволить більш глобально усвідомити проблему і вибрати найбільш вдале рішення.

Рассмотрим простое арифметическое выражение:

$$a + b - c * 2 \tag{3.1}$$

Якщо виразити цей вислів у вигляді блок схеми, можемо побачити що алгоритм обчислення даного вираження схожий на конвеєр (Рис. 3.13).

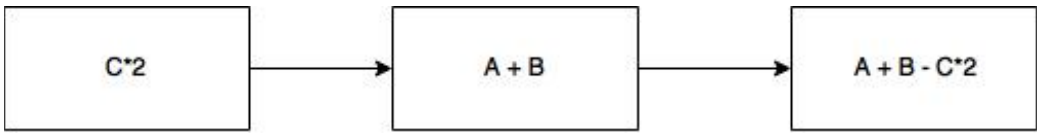

Рисунок 3.13 - Блок-схема алгоритму обчислення a+b-c*2

На даній схемі можна побачити що результати підвиразу C * 2 і A + B присутні в третьому блоці. Якщо розглядати операнди кожного підвиразу як вхідні дані для блоку обчислень, то виходить деревоподібна структура: на вхід першого блоку надходять два аргемента - C і 2. На вхід другого - A і B.

На вхід третього результати попередніх двох блоків. Дана властивість більш наочно можна побачити на діаграмі (Рис. 3.14).

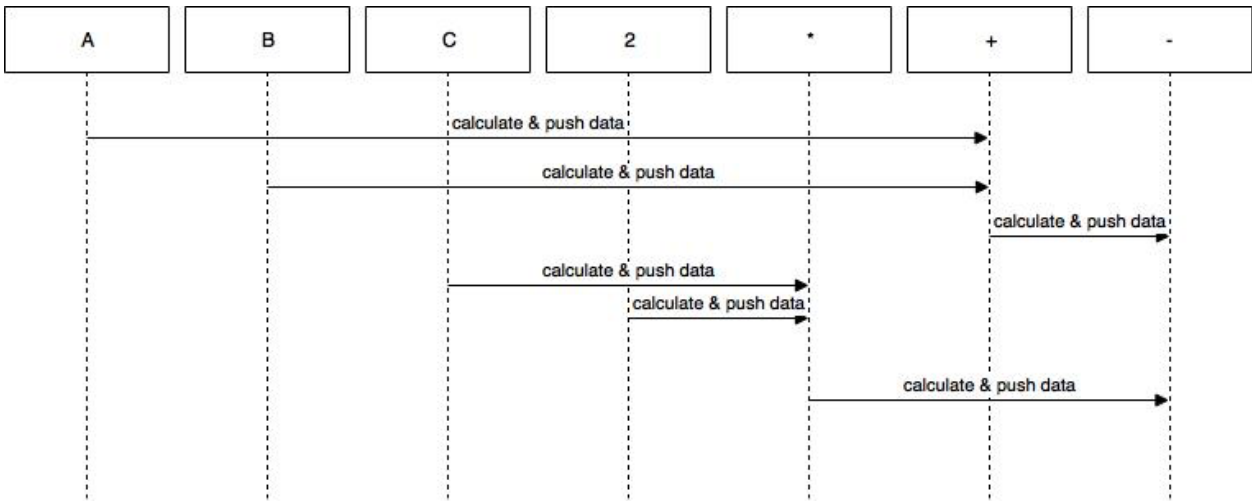

Рисунок 3.14 - Діаграма обчислення a+b-c*2

Як можна побачити маємо деревоподібну структуру. Є джерела даних, за котрими необхідно «спостерігати». Звернувшись до зібрання кращих архітектурних рішень, можна зрозуміти що основою для даної архітектури найкраще підходять два паттерна, причому один з них будується на іншому - observer і composite [35].

Паттерн observer - це паттерн проектування програмного забезпечення, в якому об'єкт, який називається предметом, зберігає список своїх об’єктів, які називають спостерігачами, і повідомляє їм автоматично про будь-які зміни стану, зазвичай за допомогою одного з їхніх методів.

Він в основному використовується для реалізації розподілених систем обробки подій, в програмному забезпеченні орієтованого на обробку подій. Так само даний патерн проектування вирішує проблему перевикористання елементів. Один спостерігач може мати безліч підпісників. Таким чином, наприклад, в виразі (1) блок «+», якщо це потрібно, міг би передавати результат своєї арифметичної операції більш ніж одному блоку. Це дозволяє будувати

довгі ланцюжки, кожен елемент в яких є одночасно і спостерігачем і ресурсом. Це дозволяє будувати довгі структури обчислень практично будь-якої складності. UML диаграмма для данного паттерна выглядит следующим образом (Рис. 3.15).

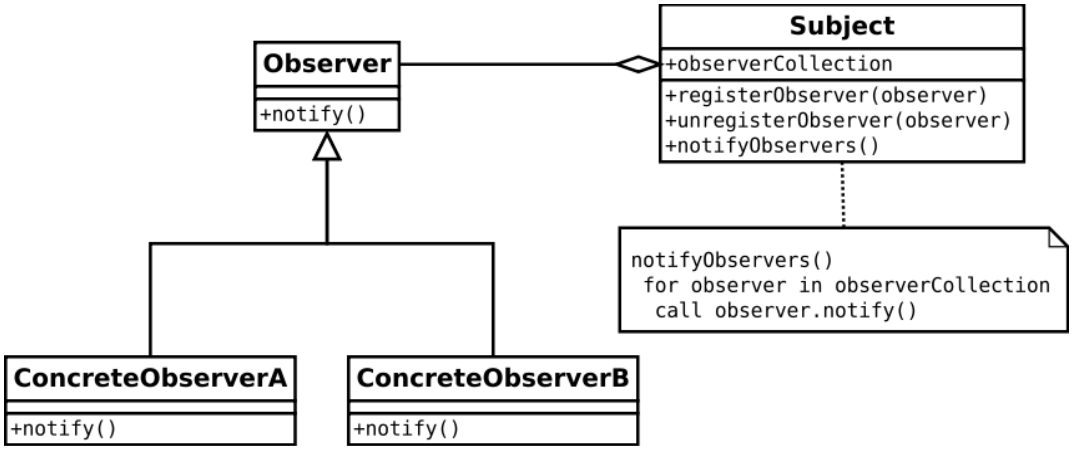

Рисунок 3.15 - Типова UML діаграма для паттерну Observer

Єдина відповідальність перед суб'єктом полягає в тому, щоб вести список спостерігачів та повідомляти їх про зміни стану, викликаючі їх функції обратного виклику.

Відповідальність спостерігачів полягає в тому, щоб підписатися (та відписатися від реєстрації) на цікаві їм зміни стану якогось об’єкта (отримувати повідомлення про зміни в стані ресурсу) та оновлювати свій стан (синхронізувати їх стан із станом суб'єкта).

Тепер розглянемо наступний арифметичний вираз:

$$x^2 - b * (x^2 - 1) \tag{3.2}$$

Якщо проаналізувати цей вислів, то можна побачити що в ньому є повторяющ підвираз «x^2», обчисливши яке можна перевикористати. Якщо подати цей вираз в деревовидному вигляді, де кожен блок є передплатником / спо-

стерігачем і відповідальність за певну операцію, то вийде наступна структура (Рис. 3.16).

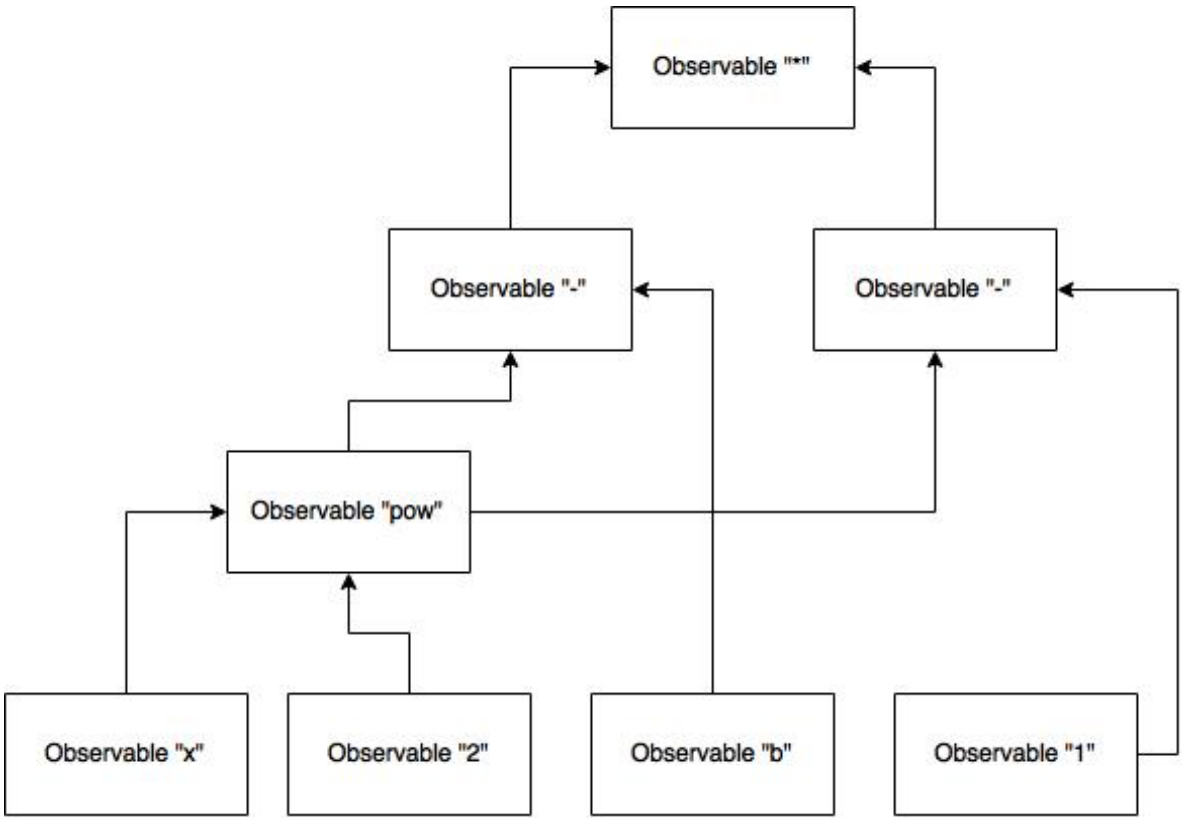

Рисунок 3.16 - Дерево з ланцюгів observables/subscribers

Як видно з схеми, вираховується один раз і належить до входу двох блоків. Це допомагає зменшити кількість обчислень і підвищити продуктивність системи в цілому. Переиспользование больших и сложных блоков в этом способе значительно понизит затраты на вычисления и память, так как каждый промежуточный результат нужен где-то хранить.

Побудова таких ланцюгів призводить нас до так званого "реактивного" программування, яке широко застосовується в фреймворку React. У обчислювальному процесі реактивного програмування - це асинхронна парадигма програмування, що стосується потоків даних і поширення змін. Це означає, що можна легко виразити поточні потоки даних статичними (наприклад, масивами) або динамічними (наприклад, емітерами подій) за допомогою використовуваних мов програмування, і що існує виявлена залежність у відповід-

ному виконанні моделі, що полегшує автоматичне розповсюдження змін, пов'язаних з потоком даних [36].

Після того, як проблема жорсткості була вирішена варто звернути увагу на механізми синхронізації даних. Точна синхронізація добитися практично не можливо, тому що різні сенсори мають різну частоту, а в тому випадку, коли дані приходять, скажім, через мережевий інтерфейс, час частота таких даних є дуже недетермінованим значенням. До того ж, що робити, якщо джерело даних віддає дані дуже часто, а аналізувати такі дані потрібно набагато рідше? Наприклад, датчик температури повідомляє своє значення раз в секунду, а потрібно визначити середнє значення за хвилину?

В такому випадку слід використовувати такі блоки, які не пропускають далі себе дані до тих пір, поки не накопичуть достатньо даних для обчислень. Наприклад дані можуть зберігатися в буффері до тих пір, поки не накопичуються дані за хвилину, потім оператор «Середнє» обчислить середнє значення за буфером і передаст дані наступному в ланцюжку.

Для цих цілей введемо поняття «Вікно». Вікно - це концепція, що описує структуру даних, що містить деяку кількість даних, в залежності від розміру вікна. У реактивному програмуванні існує декілька типів вікон:

- за часом;
- за кількістю даних.

Так само кожне з даних вікон може бути «ковзним» або спрацьовувати за таймером. Тепер між будь-якими ланками в ланцюжку може стояти вікно. Ланки обмінюються між собою даними за допомогою передачі буферу. Кожен з ланок або «проштовхує» дані подпсчікам чи ні.

Для того щоб створювати більш складні рішення, можна ввести ще кілька операторів, таких як «більше», «менше», «дорівнює», «і», «або».

Отже, на даному етапі можна виконувати прості арифметичні та логічні операції на проміжках часу. Однак не вирішене питання синхронізації даних у часі між обчисленнями. Якщо, наприклад, є два вікна, то в одному вікні

можуть бути старі дані, а в другому нові. Виконувати арифметичні операції між даними сильно розподіленими в часі є ідею що не має сенсу.

Для вирішення даної проблеми слід ввести ще одну модель даних, яка є протилежною по відношенню до поточної. PULL-модель буде працювати для синхронізації даних. Коли відбувається подія (дані йдуть по ланцюгу), оператори що мають декілька входів в разі потреби опитуюь всіх на кого віно підписани, для того що б синхронізувати дані, крім того від кого дані прийшли (емітента). Це означає що ланка що отримала дані може спитати у решти ланцюга чи є ще актульні дані. Дане опитування викликає перерахунок вікон. Якщо дані які знаходяться в вікнах застаріли - вони видаляються з вікон, а ті що чи не застаріли повертаються по ланцюжку вгору. Таким чином операції виконуються завжди над актуальнмі даними. Принцип синхронизіції за допомогою PULL-моделі показаний на рисунку 3.17.

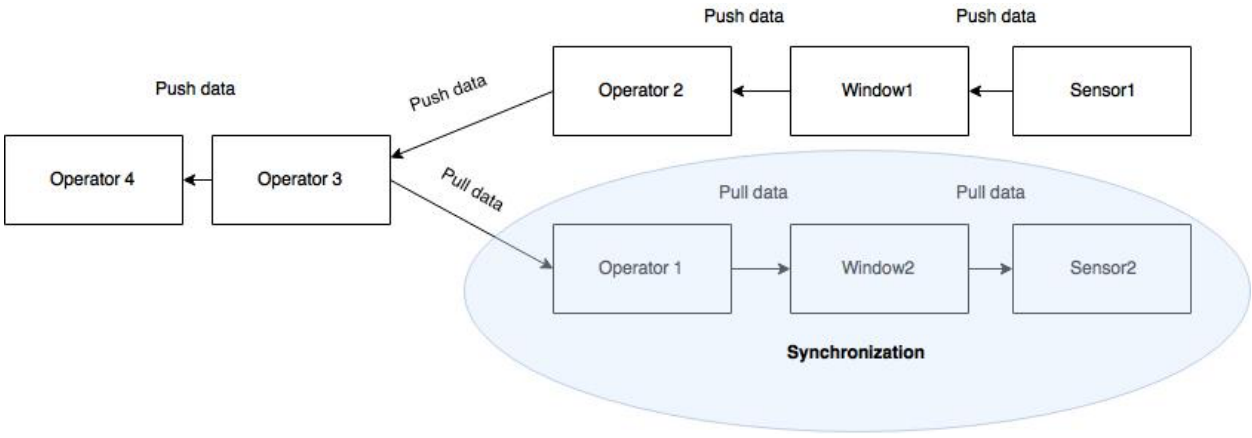

Рисунок 3.17 - Синхронізація часу з PULL-моделью

На рисунку виникає подія та дані починають рухатися по ланцюгу від Sensor1 до Operator 3. Коли дані досягають Operator 3, він в свою чергу отримує актуальні дані з іншої гілки (гілка відзначена на схемі блакитним колом). Запит синхронізації (PULL-запит даних) доходить до самого кінця. Якщо у Sensor2 є актуальні дані, він їх віддає, якщо немає - то Window2 теж синхронизує свої дані и віддає акнульні якщо вони є. Ця операція повторюється далі

по ланцюжку до тих пір поки одна з ланок не поверне актуальних даних або управління не повернеться до емітента PULL-запита.

Тепер, розібравшись із синхронізацією, можна сфокусувати увагу на більш складної аналітиці. До сих пір можна проісхводіть прості арифметичні і логічні операції, проте більш складна аналітика вимагає більш високорівневих можливостей. Для початку слід визначити кілька реальних способів іспользованія системи аналітики. Візьмемо для прикладу галузь виробництва мережевих роутерів. Відомо що при ARP-flooding зростає кількість ARP-пакетів в мережі. Значить що система в такому випадку повинна вміти виявляти зростання числа пакетів за певний проміжок часу. Зростання кількості пакетів можна змінювати в процентах або ж в одиницях. Наприклад, зростання кількості ARP-пакетів на 30% за останні 30 секунд означає що найімовірніше відбувається атака. Зростання кількості на 30% за 30 секунд є патерном шуканим патерном. А виявлення такого умови назівется збігом патерну. Даний патерн зображений на рисунку 3.18.

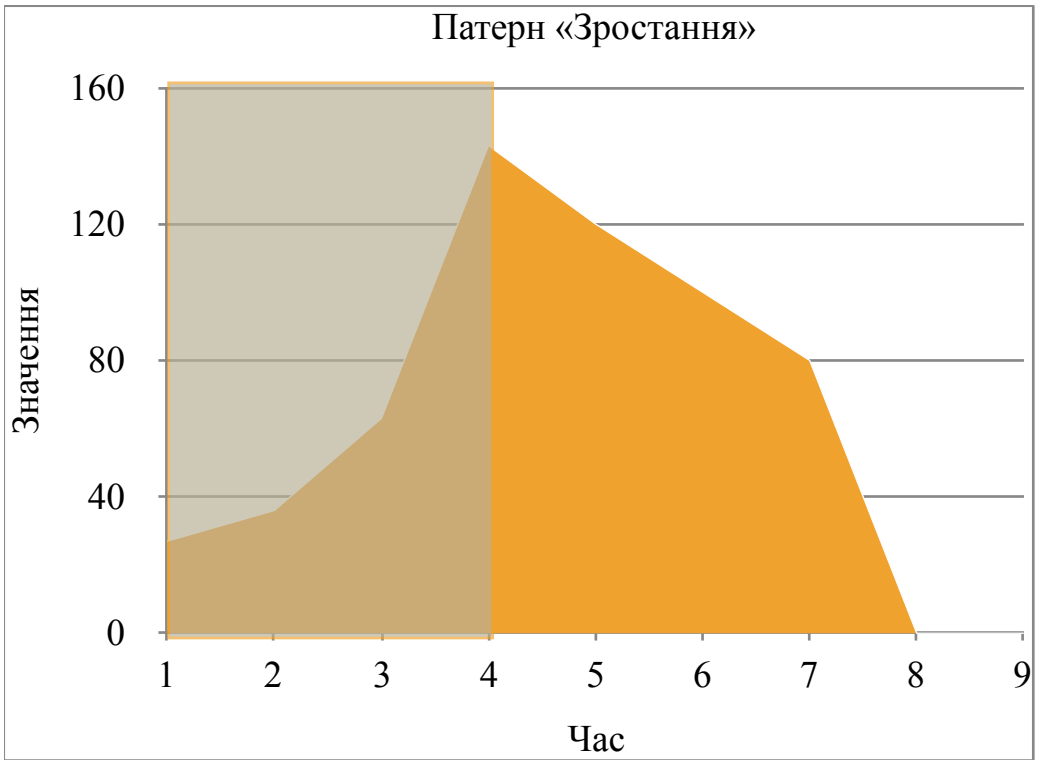

Рисунок 3.18 - Патерн «Зростання»

На діаграмі напівпрозорою заливкою відзначено збіг патерну «Зростання». Таким чином можна візуалізувати результати роботи аналітики. Даний патерн є одним з найбільш використовуваних так само, як і його протилежність - патерн «Зменшення».

Зростання/зменшення. Дані патерни використовуються для того що б виявляти тимчасові збільшити чи зменшити показник на відрізку часу для виявлення закономірностей і поліпшення роботи систем на основі аналізу виведених закономірностей. Приклад використання: Підвищення рівня втрати пакетів при використанні бездротового маршрутизатора.

Рівень/діапазон. Використовуються для виявлення перевищення/падіння нижче рівня допустимих значень і реагування на цю подію. Патерн «Діапазон» зображений на рисунку 3.19.

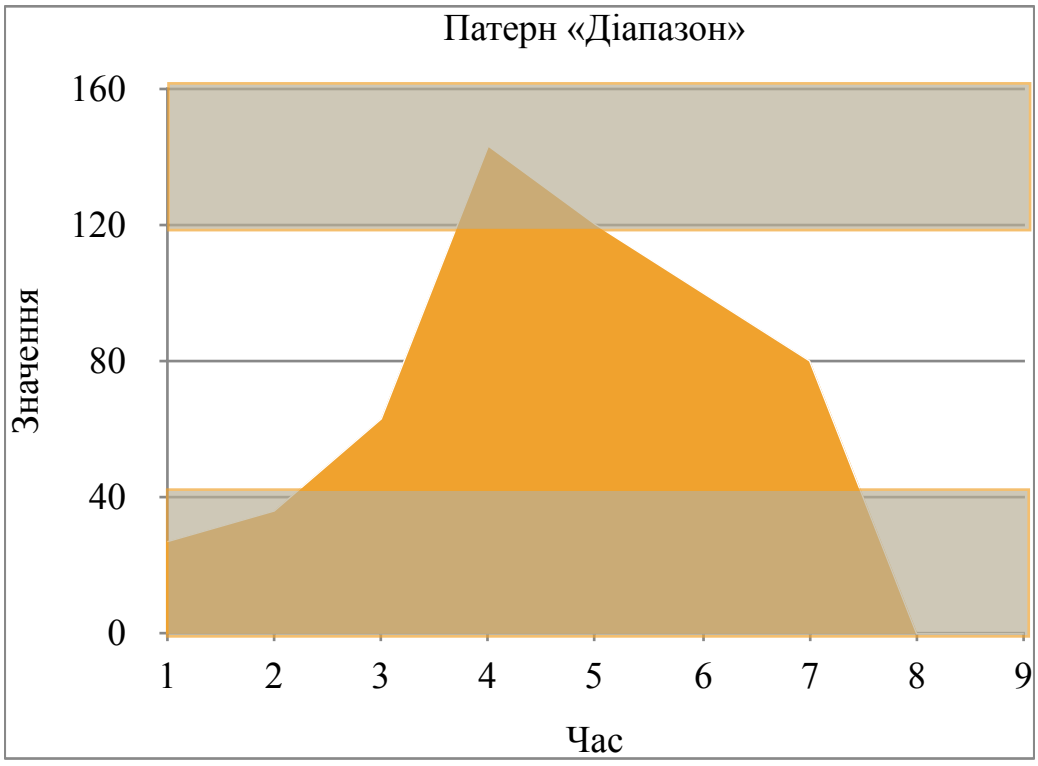

Рисунок 3.19 - Патерн «Діапазон»

Збіг паттерна відзначено напівпрозорим зеленим кольором. Якщо значення вийшло за допустимий діапазон (на рисунку [40-120]), це означає що

збіг паттерна знайдено.

Патерни реалізуються аналогічно операторам та функціям у вигляді ланок ланцюжка. Це допомагає тримати архітектуру однорідною. Так само патерни можуть бути комбіновані. Наприклад можна одночасно шукати підвищення температури і падіння тиску на часовому інтервалі (Рис. 3.20).

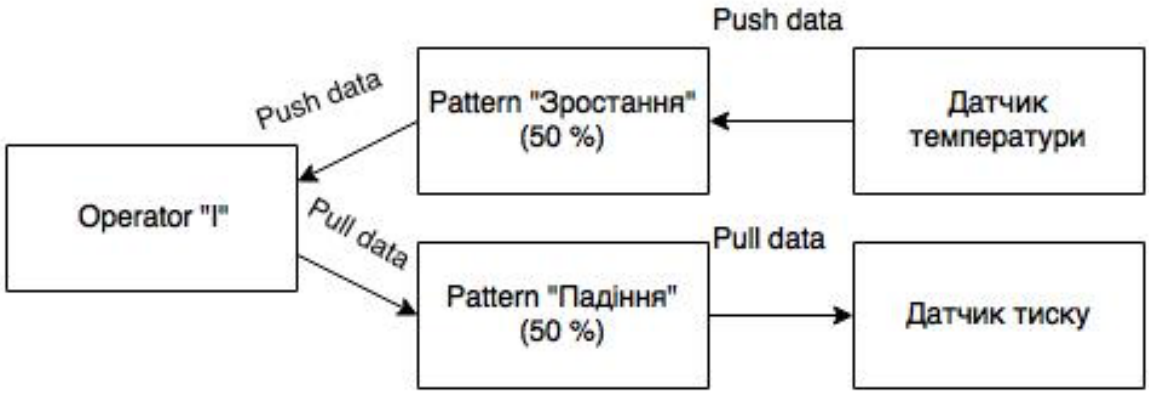

Рисунок 3.20 - Комбінування патернів

Комбінування патернів відбувається за допомогою додаткових операторів - «і», «або». На схемі зображено комбінування патернів за допомогою додаткового оператора «і», який пропускає через себе дані тільки в разі коли обидва патерни були знайдені. Як завжди для ініціювання події використовується PUSH-модель даних, а для синхронізації - PULL-модель.

3.5 Розробка системи адресації

Важливою особливістю системи є здатність адресувати пристрої. Це дозволяє виконувати аналітику даних на певному вузлі або навіть декількох вузлах. Аналогічно протоколам мережевого і транспортного рівня, важливий моментом в адресації являетс можливість адресувати групи пристроїв або всі пристрої. Частина цієї функціональності в рамках розроблюваної системи

покладається на транспортний протокол MQTT. В якості адреси пристрою використовується тема для публікацій, яка прослуховується тими вузлами, які належать певній групі. Наприклад як частина теми може використовуватися ідентифікатор групи. Приклад такої теми зображений на рисунку 3.21.

```
/edge_analytics/organization_z/group_one
```

Рисунок 3.21 - Приклад теми для публікації повідомлень

Після третього «слеша» йде ім'я групи «group_one», яке служить індикатором групи в назві теми. Всі члени цієї групи і тільки вони підписані на дану тему, тому тільки вони отримають повідомлення опубліковані на дану тему. Таким чином можливо створювати статичні групи.

Коли мова йде про декілька десятків пристроїв, даний механізм працює, проте коли кількість пристроїв досягає кілька мільйонів, додавати пристрої в групу стає досить ресурсномісткої завданням. Тому необхідний спосіб автоматично групувати пристрої по какмі-небудь ознаками.

Даний функціонал називається автоматичною угрупованням по атрибутам пристроїв. Атрибути пристроїв - деяка статична інформація про пристрій, яка не змінюється з моменту підключення пристрою до системи. Якщо дані властивості були б динамічними, то адресувати пристрої було б набагато складніше, тому що доводилося б перевіряти в кожен момент часу належить даний пристрій до групи, що само по собі ресурсномістка процедура при наявності великої кількості пристроїв.

Процес групування виглядає наступним чином: після того як пристрій завантажується, воно зчитує свою конфігурацію, таким образу дізнаючись про те де взяти інформацію про ство аттрибутах, потім зчитує атрибути і відправляє ці метадані на сервер. Далі «Хмара» може створити аналітичну модель і відправити на спеціальну створену MQTT тему, яку слухають всі пристрої. Аналітична модель містить не тільки дані про те як будувати аналітич-

ні обчислення, але і бінарний предикат який пристрою можуть застосувати до своїх Атрибути для того що б зрозуміти чи варто Цей пристрій виконувати аналітичну модель. В даному контексті бінарний предикат - це деякий вираз на основі якого створюється невеликий аналітичний ланцюжок, подібно аналітичним ланцюжках які обговорювалися в попередньому підрозділі. Метою даної ланцюжка видати результат «брехня» або «істина». Якщо результат «істина», то даний вузол повинен виконати аналітичну модель, в іншому випадку - він ігнорує цей запит.

Даний функціонал добре маштабіруется і дозволяє адресувати мільйони пристроїв.

3.6 Збирання, запуск та огляд системи

Зберемо систему. Для цього створимо каталог для збирання проекту, виконаємо конфігурацію збірки і потім зберемо проект. Результат конфігурації збірки показаний на рисунку 3.22.

```
[I] Architect@Alexeys-MacBook-Pro ~/D/w/r/r/build> cmake -DCMAKE_BUILD_TYPE=Release -DWITH_TESTS=OFF ..
-- The C compiler identification is AppleClang 9.0.0.9000038
-- The CXX compiler identification is AppleClang 9.0.0.9000038
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found PkgConfig: /usr/local/bin/pkg-config (found version "0.29.1")
-- Build type: Release
-- Configuring done
-- Generating done
```

Рисунок 3.22 - Конфігурація збирання

Як видно з рисунка, конфігурація пройшло успішно. та в директорії для збирання було сгенеровано відповідні файли. Далі слід виконати команду

make що б зібрати проект (Рис. 3.23).

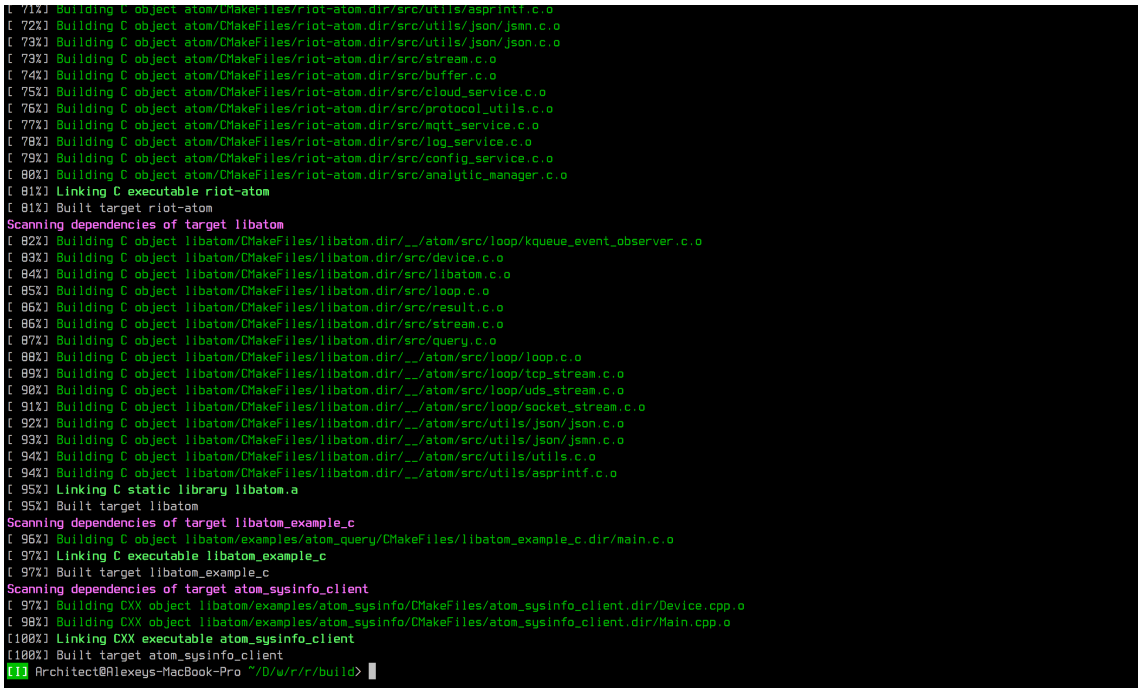

Рисунок 3.23 - Збирання проекту

Проект зібрався успішно, це означає що можна запустити систему (Рис. 3. 24).

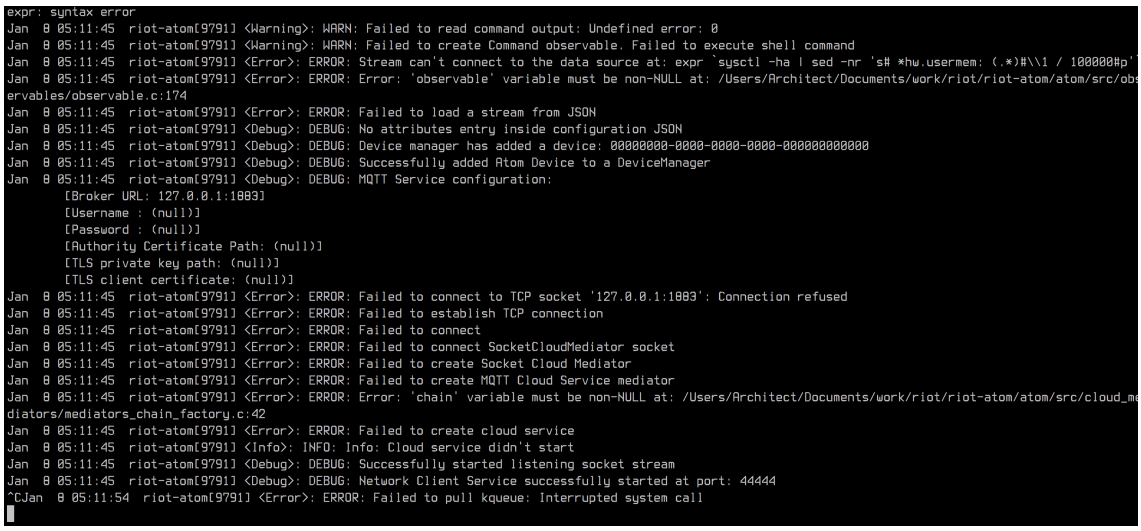

Рисунок 3.24 - Результат запуску проекту

Проект запустився, в емуляторі терміналу видно безліч повідомлень від системи логування. Логування відбувається в стандартний потік syslog, який починає існування коли операційна система запускається. Таким чином можна буде побачити повідомлення про помилки, в разі якщо розроблена система раптово впаде.

Графічний інтерфайс «Хмари» за допомогою якого користувач може спостерігати за результатами аналітики зображений на рисунку 3.25.

Рисунок 3.25 - Графічний інтерфейс «Хмари»

Використовуючи даний інтерфейс, користувач може спостерігати за аналітикою яка відбувається на ралічних вузлах даної аналітичної системи, переглядати списки пристроїв який на даний момент знаходяться в підключеному стані, фільтрувати інформацію по групам, переглядати історичні дані, створювати і видаляти аналітичні запити.

ВИСНОВКИ

Метою даної роботи є розробка аналітичної системи туманних обчислень Інтернету Речей і реалізація її у вигляді програмного рішення. Це дасть можливість істотно зменшити кількість трафіку в умовах сильнозашумлених і/або дорогих каналів зв'язку таких як супутниковий зв'язок, а так само застосовувати дане рішення практично для будь-яких аналітичних задач не володіючи знаннями в області програмування, що істотно здешевить впровадження аналітики в процес.

Існуючі аналітичні системи є корисними, однак не вирішують проблем з трафіком і вимагають участі висококваліфікованих інженерів і адаптації даних рішень під вимоги користувача за допомогою програмування. Дані фактори збільшують час впровадження аналітики і істотно збільшують її вартість.

Наявність можливості візуального програмування системи покращують користувальницький досвід і дозволяють легше створювати аналітичні запити, особливо на початку освоєння системи.

Застосування асинхронного архітектури заснованої на подіях дозволяє більш раціонально використовувати апаратні ресурси, що є критичним в системах з сильно обмежених кількістю ресурсів.

Використання стандартизованих протоколів верхніх рівнів служить для безшовного з’єднання в гетерогенних мережах. Використання такого проколу як MQTT дозволяє скоротити трафік і вирішити проблеми адресації, якості обслуговування та деякі інші транспортні проблеми на верхньому рівні стека протоколів. Також був доданий додатковий протокол адресації, заснований на протоколі MQTT, який дозволяє адресувати підгрупи в групах що додає системі адресації ще більшої гнучкості.

Застосування динамічних аналітичних моделей обчислення дозволяє зробити процес зміни аналітики більш легким в порівнянні з стандартним механізмом оновлень, таким як оновлення по повітрю.

Була застосована, проаналізована і вдосконалена парадигма реактивного програмування, яка застосовується для побудови максимально гнучкої асінхронної архітектури системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Francis daCosta, Francis, Henderson, Byron. Rethinking the Internet of Things: A Scalable Approach to Connecting Everything. – Санта-Клара, Каліфорнія, 2013.: Santa-Clara, California, Apress – 23 с.
2. Internet of Things URL https://www.gartner.com/it-glossary/internet-of-things/
3. Kevin Ashton That 'Internet of Things' Thing URL http://www.rfidjournal.com/articles/view?4986
4. Vandana C.P , Dr. Ajeet A. Chikkamannur IOT future in Edge Computing URL https://dx.doi.org/10.22161/ijaers/3.12.2
5. J. Gubbi, R. Buyya, S.Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” Future Gener. Comput. Syst., № 8(29), P. 1645–1660. 2013.
6. Z. Sheng и др., “A survey on the IETF protocol suite for the Internet of Things: Standards, challenges, and opportunities,” IEEE Wireless Commun., № 6(20), P. 91–98, 2013, P. 65–76
7. Z. Yang и др., “Study and application on the architecture and key technologies for IOT,” in Proc. ICMT, 2011, P. 747–751
8. K. Ashton, “That Internet of Things thing,” RFiD J., № 7(22), P. 97–114, 2009.
9. Naur, P., and Randell, B. (Eds.), “Software Engineering,” Garmisch, Germany 1968, Brussels, Scientific Affairs Division, NATO.
10. Logical Neighborhoods, Virtual Sensors and Actuators URL http:// logicalneigh- bor.sourceforge.net/vs.html
11. K. M. Chandy and W. R. Schulte, “What is Event Driven Architecture (EDA) and Why Does it Matter?” URL at http://complexevents.com/?p=212, (accessed on: 25.02.2008).
12. IBM, An architectural blueprint for autonomic computing, URL https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
13. Plug and Play URL https://ru.wikipedia.org/wiki/Plug_and_Play

14. IBM Watsons URL https://www.ibm.com/ms-en/marketplace/watson-analytics
15. Amazon Web Services URL https://aws.amazon.com/
16. Functional requirement URL https://en.wikipedia.org/wiki/Functional_requirement
17. Non-functional requirement URL https://en.wikipedia.org/wiki/Non-functional_requirement
18. Application Programming Interface URL https://en.wikipedia.org/wiki/Application_programming_interface
19. Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A survey on the ietf protocol suite for the internet of things: standards, challenges, and opportunities," IEEE Wireless Communications, № 6 (20), P. 91–98, 2013.
20. J.P.VasseurandA.Dunkels,"Ipforsmartobjects," URL http://dunkels.com/adam/dunkels08ipso.pdf
21. D. Culler and S. Chakrabarti, "6lowpan: incorporating IEEE 802.15. 4 into the IP architecture, IPSO Alliance," URL http://www.ipso-alliance.org/wp-content/media/6lowpan.pdf, 2009.
22. J. Vasseur, N. Agarwal, J. Hui, Z. Shelby, P. Bertrand, and C. Chauvenet, "Rpl: the ip routing protocol designed for low power and lossy networks," Internet Protocol for Smart Objects (IPSO) Alliance 36, 2011.
23. J. P. Vasseur, C. P. Bertrand, B. Aboussouan et al., "A survey of several low power link layers for IP smart objects," URL http://docplayer.net/17860080-A-survey-of-several-low-power-link-layers-for-ip-smart-objects.html
24. W. Hui and D. E. Culler, "Extending IP to low-power, wireless personal area networks," IEEE Internet Computing, № 4 (12), P. 37–45, 2008.
25. Z. Shelby, K. Hartke, and C. Bormann, " e constrained application protocol (CoAP)," Tech. Rep., IETF, 2014.
26. Internet Engineering Task Force (IETF) DTLS URL https://tools.ietf.org/html/rfc6347

27. D. Locke, “MQ telemetry transport (MQTT) v3. 1 protocol specification,” URL http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html
28. C Programming Language URL https://en.wikipedia.org/wiki/C_(programming_language)
29. Programming Languages ISO/IEC JTC1 SC22 WG21 N 3690 — C++ URL http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3690.pdf
30. Linux source tree URL https://github.com/torvalds/linux
31. CMake URL https://cmake.org
32. Raspbian URL https://www.raspberrypi.org/downloads/raspbian/
33. Network Working Group Transport Layer Security Protocol Version 1.2 URL https://tools.ietf.org/html/rfc5246
34. Vincent Driessen A successful Git branching model URL http://nvie.com/posts/a-successful-git-branching-model/
35. Эрих Гамма, Джон Влисидис, Ральф Джонсон, Ричард Хелм - Приемы объектно-ориентированного проектирования. Паттерны проектирования. - СПб: Питер, 2001. — 368 с.
36. Reactive Programming URL https://en.wikipedia.org/wiki/Reactive_programming