Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

АНАЛІЗ ВИКОРИСТАННЯ ПРОТОКОЛУ MQTT ПРИ ПОБУДОВІ IOT СИСТЕМ

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки» - 122

Керівник курсової роботи
асистент
Андрощук М. В.
_____
(підпис)
"____" _____ 2023 р.
Виконав студент КН-3
Цегельник Б.В.
"____" _____ 2023 р.

Київ 2023

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Викладач кафедри інформатики,
асистент
_____ Андрощук М.В.
„____” _____ 2022р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Цегельнику Богдану Володимировичу
факультету інформатики 3 курсу бакалаврської програми

Тема: Аналіз використання протоколу MQTT при побудові IoT систем
Вихідні дані: IoT система з використанням протоколу MQTT
Зміст ТЧ до курсової роботи:

Зміст
Анотація
Вступ
1 Аналіз стандарту та можливостей протоколу MQTT
2 Аналіз альтернативних протоколів
3 Розробка IoT системи
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2019 р. Керівник_____
(підпис)

Завдання отримав _____
(підпис)

Тема: Аналіз використання протоколу MQTT при побудові IoT систем

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	13.09.2022	
2.	Огляд літератури за темою роботи	16.02.2023	
3.	Написання програмного коду для проведення досліджень.	20.03.2023	
4.	Написання пояснювальної роботи.	10.04.2023	
5.	Надання роботи керівнику для перевірки	09.05.2023	
6.	Корегування роботи за результатами перевірки керівником	13.05.2023	
7.	Оформлення слайдів для доповіді.	15.05.2023	
8.	Захист роботи.	23.05.2022	

Студент Цегельник Б. В.

Керівник Андрощук М. В.

“_____” _____

Зміст

Анотація

У цій роботі розглянуті основні можливості та переваги MQTT. Проаналізовано інші популярні у Інтернеті речей протоколи, такі як HTTP та CoAP у порівняння із MQTT. Також у роботі розглянута реалізація IoT системи з використанням протоколу MQTT та платформи Home Assistant як хабу. У системі використано симуляцію фізичних пристроїв, які працюють у ній за допомогою протоколу MQTT. Детально описані процеси реалізації та налаштування системи.

Ключові слова:

Інтернет речей, IoT, протокол, аналіз протоколу, MQTT, Home Assistant.

ВСТУП

З розвитком сучасних технологій та інтернету з’являється все більше пристроїв, з якими можна так чи інакше комунікувати через мережу. Вони можуть обмінюватися інформацією між собою та виконувати потрібні сценарії поведінки, або ж просто збирати дані для моніторингу та аналізу. Мережі цих пристроїв в останньому десятилітті отримали назву Інтернет речей. Популярна сьогодні концепція розумних будинків або міст є прикладом таких мереж різних масштабів.

Для комунікації в системах Інтернету речей використовують різноманітні протоколи на всіх рівнях від прикладного (HTTP, MQTT та ін.) до фізичного (Wi-Fi, Ethernet, Bluetooth, Zigbee та ін.), які поєднуються за допомогою шлюзів. Одним із протоколів, який набуває все більшої популярності в цій сфері є MQTT. Це легкий, відкритий, простий протокол, створений спеціально для роботи на малопотужних пристроях.

Метою роботи є визначити які особливості та переваги, в порівнянні з іншими протоколами, існують в MQTT, та чому він добре підходить для застосування в системах Інтернету речей.

Робота складається з трьох розділів.

Перший розділ присвячено аналізу протоколу. Детально розглянуто його ключові особливості та принцип роботи. Наведено приклади де ці особливості будуть корисними.

В другому розділі проведено аналіз із порівнянням роботи інших популярних в Інтернеті речей протоколів, таких як HTTP та CoAP.

В третьому розділі наведено результати проектування та реалізації системи автоматизованого будинку із застосуванням MQTT.

В якості джерел для цієї роботи використано офіційний стандарт протоколу, статті від авторів популярного корпоративного брокера HiveMQ, документація до фреймворку Qt та системи Home Assistant, а також деякі наукові статті про Інтернет речей.

РОЗДІЛ 1. АНАЛІЗ СТАНДАРТУ ТА МОЖЛИВОСТЕЙ ПРОТОКОЛУ MQTT

1.1 Ключові особливості

MQTT (Message Queuing Telemetry Transport) – це мережевий протокол прикладного рівня, розрахований на роботу в умовах обмеженої швидкості та якості зв’язку на малопотужних пристроях, який працює за принципом публікації та підписки. [1]

На відміну від принципу запиту та відповіді за яким працює, наприклад HTTP, сервер не обробляє повідомлення, а лише переадресовує його іншим клієнтам. Таким чином сервер виконує тільки роль посередника, тому його ще називають брокером.

Принцип роботи протоколу базується на публікації та підписці на теми, або їх ще називають топіками. Клієнт може публікувати повідомлення на певну тему, а інші клієнти можуть підписатися на цю тему, щоб отримувати повідомлення, які там публікуються [2].

На рисунку 1.1 зображено мережу із декількох клієнтів та сервера. Можна побачити, що клієнти можуть тільки відправляти, або тільки отримувати повідомлення, або робити обидві дії. Якщо потрібно якось проаналізувати та обробити дані, цю роботу також виконує клієнт, а не сервер. Сервер не аналізує вміст повідомлень, а лише перенаправляє їх до потрібних клієнтів.

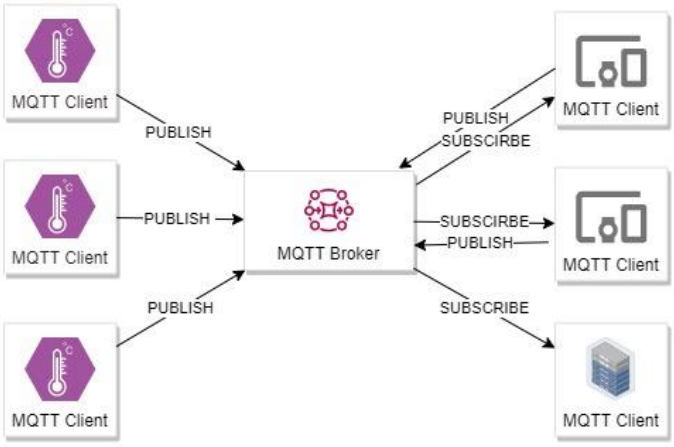

Рисунок 1.1 – Архітектура мережі в протоколі MQTT

MQTT був розроблений для використання в IoT-системах, де існують серйозні обмеження у ресурсах та пропускній здатності. Основна ідея протоколу полягала в тому, щоб забезпечити ефективну комунікацію між багатьма пристроями, використовуючи якомога менше ресурсів. Низький рівень пропускної здатності та використання ресурсів є основною перевагою цього протоколу. Це дозволяє використовувати його на пристроях з обмеженими ресурсами, таких як мікроконтролери, датчики та інші пристрої Інтернету речей.

Цього вдалося досягнути завдяки бінарності протоколу. Всі характеристики пакета, включно з видом, задаються за допомогою бітових полів мінімально потрібної довжини [1]. Це дозволяє значно зменшити розмір пакета, в порівнянні з текстовими протоколами, як, наприклад, HTTP, що особливо критично в умовах поганого зв’язку. За своєю структурою він схожий на протоколи нижчих мережевих рівнів, проте є протоколом прикладного рівня.

Іншою важливою особливістю MQTT є його можливість працювати в умовах ненадійного зв'язку, таких як мережі з високим показником втрат пакетів або низькою пропускною здатністю. MQTT підтримує механізми повторного підключення та відправлення повідомлень для забезпечення надійного доставлення. Протокол підтримує три рівні забезпечення якості обслуговування для регулювання надійності доставлення окремих пакетів, завдяки зворотному зв’язку відправника та отримувача. [3]

1.2 Структура пакетів

Як було згадано раніше, протокол MQTT є бінарним протоколом та задає характеристики пакетів за допомогою бітових полів. Пакет у цьому протоколі складається із трьох структурних частин:

1. Фіксований заголовок.
2. Змінний заголовок.
3. Корисне навантаження.

Фіксований заголовок присутній у кожному пакеті, має довжину від двох до п’яти байт та містить таку важливу інформацію, як тип та розмір пакета. Тип задається у вигляді бітового поля з чотирьох бітів, що дозволяє мати шістнадцять різних значень. Наразі два значення зарезервовано та не використовуються, а інші чотирнадцять повністю задовольняють потреби протоколу на цей момент:

- CONNECT, CONNACK, DISCONNECT – для забезпечення з’єднання із брокером.
- PUBLISH – для публікації повідомлень.
- PUBACK, PUBREC, PUBREL, PUBCOMP – для забезпечення рівнів якості обслуговування пакетів.
- SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK – для забезпечення можливостей підписки та відписки на потрібні теми.
- PINGREQ, PINGRESP – для забезпечення постійного зв’язку між клієнтом та сервером та виявлення неполадок у з’єднанні.

Ще чотири біти використовуються як прапорці та мають призначення лише в пакеті PUBLISH, для встановлення рівня забезпечення якості обслуговування, позначення повторно надісланого пакета та збереження опублікованого повідомлення. Для інших типів ці поля мають фіксовані значення та використовуються для перевірки цілісності доставленого пакета.

Далі йде значення розміру пакета, що залишився, закодоване за допомогою змінної довжини від одного до чотирьох байтів. В кожному з яких є сім значущих бітів, та один індикатор продовження. Це дозволяє надсилати пакети з максимальним розміром 256 МБ.

Змінний заголовок специфічний для кожного типу та містить додаткову інформації щодо пакету. Наприклад, у пакеті PUBLISH, змінний заголовок містить інформацію про тему, на який публікується повідомлення, а у пакеті CONNECT, він містить інформацію про параметри з'єднання клієнта з

брокером MQTT, такі як ідентифікатор клієнта, рівень якості обслуговування, тривалість сесії тощо.

Корисне навантаження є частиною пакета, де знаходиться його основний вміст. Як і змінний заголовок, воно може мати різний формат та структуру та містити текстову інформація, бінарні дані, зображення або будь-які інші дані. Наприклад, у пакеті PUBLISH, корисне навантаження містить повідомлення, яке публікується, а у SUBSCRIBE - тема, на яку клієнт бажає підписатися. Важливо зазначити, що корисне навантаження є необов'язковим полем для деяких типів, наприклад, для пакетів PINGREQ та PINGRESP, в яких це поле відсутнє. [1]

1.3 Публікація та підписка на теми

Одним з основних функціональних елементів протоколу MQTT є механізм публікації та підписки на теми. Він дозволяє обмінюватися повідомленнями між клієнтами за допомогою брокера. Клієнт може публікувати повідомлення на певну тему, вказавши її ім'я в пакеті PUBLISH.

Тема є рядком символів, який ідентифікує категорію повідомлень. Вона має деревовидну ієрархію, де кожний рівень в рядку розділяється за допомогою скісної риски. На рисунку 1.2 можна побачити приклад теми з декількома рівнями. [4]

Рисунок 1.2 – Приклад багаторівневої теми [4]

Рівні допомагають краще організувати структуру мережі, а також надають можливість зонування пристроїв та розуміння природи

повідомлень в темі. Добре організована ієрархія дозволяє клієнтам підписуватися на цілі групи тем, використовуючи символи підстановки.

Для того, щоб підписатися на тему, клієнт повинен відправити запит SUBSCRIBE із темою, на яку він бажає підписатися. Коли на брокер надходить повідомлення, він розсилає його всім клієнтам, які підписалися на відповідний топік. Клієнти можуть бути підписаними на декілька тем, зокрема за допомогою символів підстановки.

У темах MQTT існує два символи підстановки: символ решітки(«#») та символ знаку плюс(«+»). Символ «+» використовується для заміни одного рівня в темі підписки. Наприклад, якщо в теми «home/назва_кімнати/light» публікується інформація про світло у кімнатах будинку, то підписка на «home/+/light» дозволить отримувати повідомлення із усіх кімнат. Символ «#» використовується для заміни довільної кількості тем, але лише в кінці. Наприклад підписка на тему «home/kitchen/#» дозволить отримувати повідомлення про все що стосується кухні в будинку, а підписка на «home/#» - повідомлення із всього будинку. Це може бути корисно для клієнтів, які аналізують повідомлення в мережі, або для клієнтів із можливістю автоматично визначати нові пристрої в мережі.[4]

Також деякі брокери підтримують механізми авторизації, та обмеження доступу клієнтів до певних тем. Це може бути корисно при проектуванні великих ієрархій із застосуванням багатьох клієнтів.

1.4 Якість обслуговування пакетів

MQTT надає 3 рівні якості обслуговування пакетів(QoS), які застосовується при публікації повідомлень. Слід зазначити, що публікація відбувається як від клієнта до сервера, так і від сервера до клієнта за наявності підписки на тему. Тому значення якості потрібно передавати в пакетах типу

PUBLISH для надсилання даних клієнтом до сервера та SUBSCRIBE для отримання клієнтом даних від сервера. Якість обслуговування індивідуальна й незалежна для кожної підписки клієнта та окремих відправлених повідомлень. [3]

Нульовий рівень – максимум один раз. Відправник надсилає пакет PUBLISH без отримання зворотного зв'язку від отримувача. Пакет може або дійти або не дійти до отримувача, тому цей рівень не гарантує доставлення повідомлення. Його варто використовувати коли втрата повідомлення не впливає на коректну роботу системи. Наприклад сенсор може надсилати виміри з високою частою і втрата якогось із них майже ні на що не впливає. На рисунку 1.3 зображено журнал сервера при публікації та підписці з нульовим рівнем якості обслуговування.

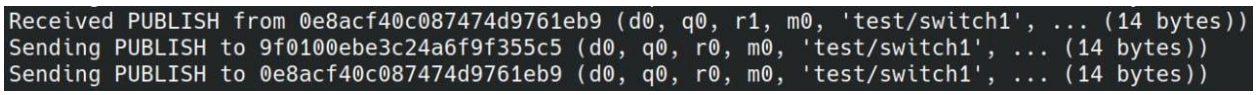

Рисунок 1.3 – Нульовий рівень якості обслуговування

Перший рівень – щонайменше один раз. Шляхом зворотного зв'язку цей рівень гарантує доставлення повідомлення, але не виключає повторне його отримання. Його варто використовувати коли потрібна впевненість в доставленні повідомлення, але його повтор ні на що не вплине. Наприклад коли перемикач надсилає повідомлення про вимкнення, то він має бути впевнений що сервер отримав його та вимкнув потрібний пристрій, але якщо повідомлення про вимкнення буде отримане сервером повторно, то це не буде критичною помилкою в системі. Відправник присвоює повідомленню унікальний ідентифікатор та надсилає пакет PUBLISH з цим ідентифікатором. Коли отримувач адресат отримує повідомлення він може його обробити та зобов'язаний надіслати у відповідь підтвердження про отримання за допомогою пакету PUBACK з тим самим ідентифікатором повідомлення. Після цього ідентифікатор звільняється та може використовуватися для нових повідомлень. Якщо через певний час відправник не отримує відповіді, то

надсилає пакет PUBLISH повторно. На рисунку 1.4 зображено журнал сервера при публікації та підписці з першим рівнем якості обслуговування.

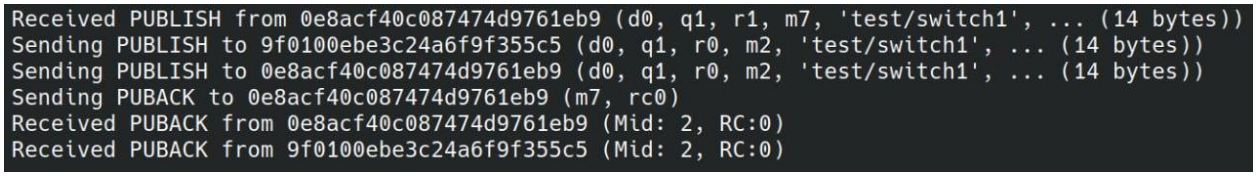

Рисунок 1.4 – Перший рівень якості обслуговування

Другий рівень – рівно один раз. Коштом подвійного «рукостискання» цей рівень забезпечує гарантію, що повідомлення буде оброблено отримувачем тільки 1 раз. Його варто тільки тоді, коли втрата або дублювання повідомлення призведе до неправильної роботи системи, оскільки він потребує у два рази більше надсилань пакетів, а тому у два рази повільніший за перший рівень якості. Одним з можливих застосувань є платіжні системи, де безпека набагато важливіша за швидкість. Реалізація першого «рукостискання» схожа на перший рівень якості, але отримувач не може обробити повідомлення після отримання, а лише повідомляє відправника про отримання за допомогою пакету PUBREC. Після підтвердження отримання відправник надсилає пакет PUBREL, отримувач може обробити повідомлення та надсилає пакет підтвердження PUBCOMP, який є завершенням сесії публікації. Тільки після того, як підтвердження було надіслано, ідентифікатор повідомлення звільняється. До того він зберігався отримувачем для уникнення повторної обробки. Якщо на якомусь з етапів за певний час не було отримано відповіді, то пакет може бути відправлено повторно. На рисунку 1.5 зображено журнал сервера при публікації та підписці з другим рівнем якості обслуговування.

```
Received PUBLISH from 0e8acf40c087474d9761eb9 (d0, q2, r1, m11, 'test/switch1', ... (14 bytes))
Sending PUBREC to 0e8acf40c087474d9761eb9 (m11, rc0)
Received PUBREL from 0e8acf40c087474d9761eb9 (Mid: 11)
Sending PUBLISH to 9f0100ebe3c24a6f9f355c5 (d0, q2, r0, m5, 'test/switch1', ... (14 bytes))
Sending PUBLISH to 0e8acf40c087474d9761eb9 (d0, q2, r0, m5, 'test/switch1', ... (14 bytes))
Sending PUBCOMP to 0e8acf40c087474d9761eb9 (m11)
Received PUBREC from 0e8acf40c087474d9761eb9 (Mid: 5)
Sending PUBREL to 0e8acf40c087474d9761eb9 (m5)
Received PUBREC from 9f0100ebe3c24a6f9f355c5 (Mid: 5)
Sending PUBREL to 9f0100ebe3c24a6f9f355c5 (m5)
Received PUBCOMP from 9f0100ebe3c24a6f9f355c5 (Mid: 5, RC:0)
Received PUBCOMP from 0e8acf40c087474d9761eb9 (Mid: 5, RC:0)
```

Рисунок 1.5– Другий рівень якості обслуговування

1.5 Збережене повідомлення для початкової ініціалізації

Коли клієнт підписується на нову тему, то нічого не знає про стан в ній доки не отримає повідомлення. Протягом цього часу публікація в тему може бути небезпечною. Але повідомлення можна чекати як декілька секунд, так і декілька годин або днів і за цей час клієнт не матиме визначеності.

Протокол MQTT дозволяє зберегти повідомлення на сервері для подальшого відправлення клієнтам, які щойно підписалися на відповідну тему. Для цього пакет PUBLISH має відповідне бітове поле в заголовку під назвою RETAIN. Кожна тема може мати тільки одне збережене повідомлення, тому коли надходить нове, то старе видаляється. Збереженим є останнє повідомлення відправлене з параметром RETAIN, а не останнє відправлене в тему. [1]

За допомогою цього механізму новий клієнт може отримати інформацію про стан системи відразу після підписки на потрібну тему. На рисунку 1.6 зображено журнал сервера де можна побачити як відразу після підписки на тему вимикач отримує повідомлення із поточним станом, що дозволяє йому синхронізуватися з іншими вимикачами або пристроями.

```
Received SUBSCRIBE from 663e2f133bb643f8b7da749
    test/switch1 (QoS 0)
663e2f133bb643f8b7da749 0 test/switch1
Sending SUBACK to 663e2f133bb643f8b7da749
Sending PUBLISH to 663e2f133bb643f8b7da749 (d0, q0, r1, m0, 'test/switch1', ... (14 bytes))
Received SUBSCRIBE from bd40859eef01426e917f512
    test/switch1 (QoS 0)
bd40859eef01426e917f512 0 test/switch1
Sending SUBACK to bd40859eef01426e917f512
Sending PUBLISH to bd40859eef01426e917f512 (d0, q0, r1, m0, 'test/switch1', ... (14 bytes))
```

Рисунок 1.6 – Відправка збереженого повідомлення після підписки

1.6 Безпека протоколу

Одним із важливих факторів при виборі протоколу є його безпека. MQTT має різні можливості забезпечення безпеки, які включають в себе шифрування, аутентифікацію та авторизацію. Також варто згадати про якісь обслуговування пакетів та моніторинг підключення до брокера, що підвищує надійність системи, та зменшує ризик втрати даних.

Шифрування використовується для захисту конфіденційної інформації, яка передається між клієнтами та брокерами. Протокол MQTT підтримує TLS/SSL (Transport Layer Security/Secure Sockets Layer), що дозволяє зашифровувати трафік між клієнтами та брокерами. Крім того, клієнти можуть застосовувати шифрування на рівні повідомлень, щоб захистити їх вміст від несанкціонованого доступу.

Одним із основних методів автентифікації, який напряму підтримується протоколом є логін та пароль. При підключені до сервера клієнт надсилає ці дані в корисному навантаженні, при цьому вказуючи про це у спеціальному полі змінного заголовка. Реалізація автентифікації та авторизації не визначена чітко стандартом [1] та залежить від можливостей брокера. Сервер сам визначає як використовувати поля заголовку та корисне навантаження, що дозволяє мати різноманітні методи автентифікації. Методи авторизації також не мають чіткого визначення і брокер сам вирішує по чому ідентифікувати клієнтів.

Загалом, MQTT має досить широкі можливості забезпечення безпеки та надійності передачі даних, проте для повної надійності системи необхідно мати комплексний підхід, включаючи не тільки забезпечення безпеки на рівні протоколу, а й на рівні IoT-пристроїв та інфраструктури.

РОЗДІЛ 2. АНАЛІЗ АЛЬТЕРНАТИВНИХ ПРОТОКОЛІВ

2.1 Популярні IoT протоколи

Існує безліч альтернативних протоколів для розробки IoT-рішень, які можуть бути ефективними залежно від специфіки конкретного проекту. Згідно із джерелом [6], найбільш популярними протоколами є MQTT, HTTP та CoAP. На рисунку 2.1 можна побачити діаграму популярності протоколів для систем Інтернету речей.

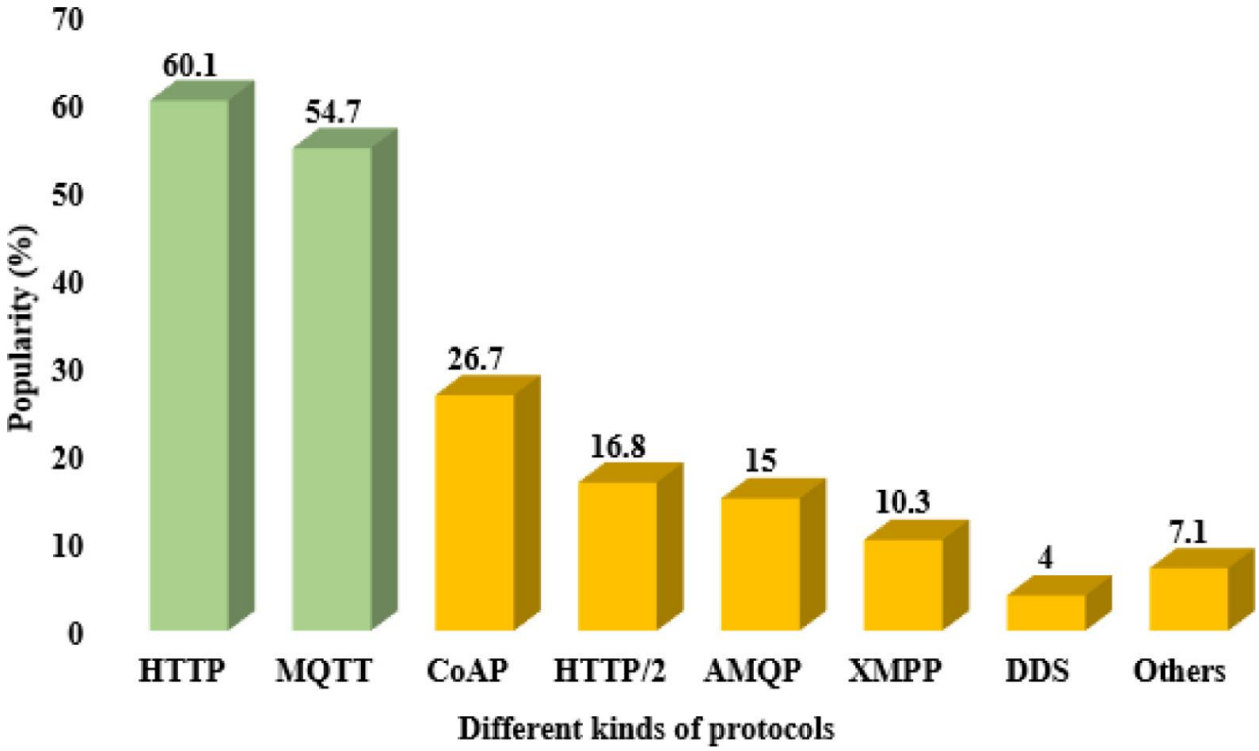

Рисунок 2.1 – популярність протоколів в IoT системах [6]

Кожен протокол має свої особливості, які можуть бути ключовими при його виборі.

HTTP залишається популярним в системах Інтернету речей через свою простоту та поширеність в інших сферах. Також дозволяє отримувати потрібні дані за запитом, що можливо зробити і за допомогою MQTT, але це буде не так зручно реалізовувати.

CoAP, як і MQTT, був розроблений для використання в IoT. Він схожий на HTTP, проте потребує менше ресурсів і використовується у системах де потрібно мати REST архутектуру. [8]

2.2 Аналіз HTTP в порівнянні з MQTT

Протокол HTTP (HyperText Transfer Protocol) був винайдений як компонент Всесвітньої павутини для передачі документів [7], проте зараз є основним протоколом для різних веб-додатків, та служить інструментом для відправки або отримання будь-яких даних.

Як і MQTT він працює поверх TCP, але за принципом запиту та відповіді від сервера, що є суттєвою відмінністю. Клієнт надсилає запит про бажання отримати дані, або ж відправляє їх до сервера, а той в свою чергу аналізує запит та повертає відповідь із певним статусом.

Відправлення запитів здійснюється на адресу під назвою URL (Uniform Resource Locator), значуща частина якої має схожу структуру до тем в MQTT. На відміну від MQTT, де теми не потрібно визначати попередньо, в HTTP це роблять для різної логіки оброблення даних, хоча можливість динамічної адреси також присутня.

Щодо безпеки, то обидва протоколи мають схожість: підтримують шифрування даних між клієнтом та сервером за допомогою SSL/TLS та механізми автентифікації і авторизації, зокрема за допомогою логіна й пароля. Проте, MQTT містить вбудований механізм підтримки підключення та гарантії доставки повідомлень, який при застосуванні HTTP потрібно реалізовувати на рівні логіки роботи пристрою.

Також відмінністю є те що MQTT вважається бінарним протоколом, в той час як HTTP символьним, де всі налаштування пакету задаються за допомогою текстових значень. Це дозволяє зробити його зрозумілішим та простішим для людини, проте додає додатковий розмір та витрати на обробку, що часто є критичним для пристроїв із низькою пропускною здатністю і частотними даними.

В таблиці 2.1 із джерела [7] можемо бачити, що для одного повідомлення витрати трафіка для HTTP менші, проте для більшої кількості вже є відчутна

різниця. І так як типовий сценарій в IoT системах – це частотні дані, як наприклад зчитування показань датчиків, то MQTT матиме в сотні разів менші витрати.

	MQTT (кількість байтів)	HTTP (кількість байтів)
Встановлення з'єднання	5572	2261
Відключення	376 (необов'язково)	0
Для кожного повідомлення	388	3285
Сума для 1 повідомлення	6336	5546
Сума для 10 повідомлень	9829	55,460
Сума для 100 повідомлень	44,748	554,600

Таблиця 2.1 – порівняння витрат трафіка для MQTT та HTTP [7]

Також можна порівняти час відгуку в цих протоколах. В зв'язку із тим, що MQTT клієнт використовує одне TCP з'єднання, час відгуку в порівнянні із HTTP набагато менший. Особливо різниця з'являється при великій кількості повідомлень, так як при кожному HTTP запиті створюється нове з'єднання, що можна побачити в таблиці 2.2. Також в зв'язку з цим, Клієнти MQTT можуть без проблем працювати без наявності зовнішньої IP адреси, а M2M передачу із застосуванням HTTP реалізувати набагато проблематичніше.

Кількість повідомлень за одне з'єднання MQTT	MQTT середній час відгуку (мс) (QoS 1)	HTTP середній час відгуку (мс)
1	113	289
100	47	289
1000	43	289

Таблиця 2.2 – порівняння часу відгуку для MQTT та HTTP [7]

Отже, MQTT та HTTP мають різні призначення та застосування, проте обидва протоколи можуть бути корисними для розробки IoT-систем. MQTT зазвичай використовується для передачі даних в режимі реального часу між багатьма пристроями, тоді як HTTP використовується для передачі даних між пристроєм та сервером даних.

2.3 Аналіз CoAP в порівнянні з MQTT

CoAP це інший протокол, який був також розроблений для використання в системах інтернету речей. Він має схожість із HTTP, бо працює за принципом запиту та відповіді до певних URL, але має менші накладні витрати для роботи на пристроях з обмеженими ресурсами. [8]

Тоді, коли MQTT має централізований сервер для перенаправлення повідомлень, у протоколі CoAP пристрій може виступати як у ролі сервера, так і у ролі клієнта. Один з пристроїв надсилає запит на інший для отримання відповіді. Тут з’являється проблема, яка відсутня в MQTT, коли пристрої не мають зовнішньої IP адреси та до них не можна відправити запит. Тому цей протокол краще використовувати в мережі де всі потрібні клієнти – доступні.

Відмінністю між протоколами також є те, що CoAP працює за допомогою UDP пакетів, які не потребують з’єднання. Також UDP не має механізму розбиття даних на декілька пакетів, що зменшує максимальний його розмір. Як можна побачити в таблиці 2.3, CoAP має менші витрати завдяки відсутності встановлення з’єднання, проте при більшому розміні пакетів можуть виникнути труднощі.

	MQTT (TCP) Bytes	CoAP (UDP) Bytes
Встановлення з’єднання	166	0
Підписка	159	0
Для кожного повідомлення	388	312+75 (запит +відповідь)
Сума для 1 повідомлення	1101	387
Сума для 10 повідомлень	8085	3870
Сума для 100 повідомлень	77925	38700

Таблиця 2.3 – порівняння витрат трафіку для MQTT та CoAP [9]

У порівнянні MQTT та CoAP виявлено, що обидва протоколи мають свої особливості та переваги для IoT-застосувань. MQTT забезпечує надійну доставку повідомлень та має простий механізм керування підпискою на теми, тоді як CoAP є більш оптимізованим для IoT-систем, що підтримують REST-архітектуру, і має менші вимоги до мережевих ресурсів. Обидва протоколи можуть бути використані разом для забезпечення комунікації між різними IoT-пристроями та сенсорами. Вибір протоколу для конкретного IoT-застосування залежить від вимог до продуктивності, безпеки, спільноти підтримки та інших факторів.

2.4 Висновок про альтернативні протоколи

В системах Інтернету речей існує різноманіття вибору протоколу до застосування. Кожен з них має свої особливості та краще вирішує конкретні задачі. В цьому розділі було оглянуто декілька з них, і можна зробити висновок, що не потрібно обмежуватися одним протоколом. Кожен з них краще підійде для певної частини і покращить її стабільність та надійність.

РОЗДІЛ 3. РОЗРОБКА IOT СИСТЕМИ

3.1 Проектування системи

Для тестування протоколу було вирішено змоделювати роботу популярної невеликої IoT системи, а саме автоматизація будинку, або «розумний дім». Так як основною метою є дослідження протоколу, замість фізичних пристроїв в системі використано програмне забезпечення, яке симулює роботу компонентів.

Всього у нашій системі передбачено 3 основні види пристроїв:

- Перемикач – може вмикати та вимикати певний електроприлад.
- Електроприлад – може бути ввімкненим або вимкненим.
- Датчик – заміряє та передає свої виміри, наприклад температуру.

В системі може існувати безліч однакових пристроїв. Їх кількість обмежується лише потужністю сервера. Але в нашій системі ми обмежимося трьома вимикачами, двома приладами та датчиком.

Два перемикачі та прилад моделюють ситуацію із світлом та декількома прохідними вимикачами в кімнаті. Ще один вимикач і прилад моделюють ситуацію увімкнення та вимкнення кондиціонера. Датчик виміряє температуру в кімнаті. Наявність датчика температури та можливості контролю стану кондиціонера дозволяє програмно створити термостат, який вмикає та вимикає кондиціонер в залежності від температури в кімнаті.

Пристрої комунікують за допомогою протоколу MQTT. Вимикачі та прилади мають зв’язок за допомогою тем. Для цього вони повинні бути підписані на одну й ту саму тему, та публікувати туди повідомлення.

В якості формату повідомлень використовується JSON. Вимикачі надсилають повідомлення «{"state":true}» або «{"state":false}» в залежності від стану. Датчик надсилає виміри у вигляді такого повідомлення: «{"data":виміри}».

В якості хабу із зручним інтерфейсом керування використано популярну операційну систему Home Assistant із веб сервером інтерфейсу та вбудованою підтримкою брокера Mosquitto та інтеграцією MQTT пристроїв. [11]

3.2 Симуляція фізичних пристроїв

Для побудови системи та тестування можливостей протоколу замість реальних фізичних пристроїв таких як датчики, електроприлади, вимикачі у будинку, використано програмне забезпечення із симуляцією зчитування показників для датчиків та увімкнення/вимкнення пристроїв.

Для програмної реалізації пристроїв обрано кросплатформовий фреймворк Qt та декларативна мова програмування QML. Він має власну бібліотеку для підтримки протоколу - Qt MQTT. В актуальній версії бібліотеки немає потрібної підтримки інтерфейсу QML, тому бібліотеку довелося модифікувати, додавши у вихідний код мовою програмування C++ потрібні директиви, такі як Q_OBJECT, Q_PROPERTY, Q_GADGET, Q_INVOCABLE. Це дозволяє використовувати C++ класи у QML. [12]

Наприклад оголошення MQTT клієнта має такий вигляд:

```
MqttClient {
    id: client
    hostname: connectionForm.hostname
    port: connectionForm.port
    username: connectionForm.username
    password: connectionForm.password
}
```

QML побудована на базі мови програмування JavaScript, тому приклад використання клієнта виглядає таким чином:

```
onToggled: {
      client.publish(topic, JSON.stringify({state: checked}),
      subscriptionForm.qos, subscriptionForm.retain)
}
```

Дані для об’єктів в даному випадку беруться із компонентів GUI та при їх зміні автоматично викликається методи-сетери для потрібних змінних.

Для кожного із трьох пристроїв створена програма із потрібним функціоналом в інтерфейсі:

- Перемикач, який дозволяє вмикати та вимикати пристрій, що відповідає певному топіку в системі. На рисунку 3.1 можна побачити його інтерфейс в різних станах (із з'єднанням та підпискою на тему та без з'єднання із сервером).

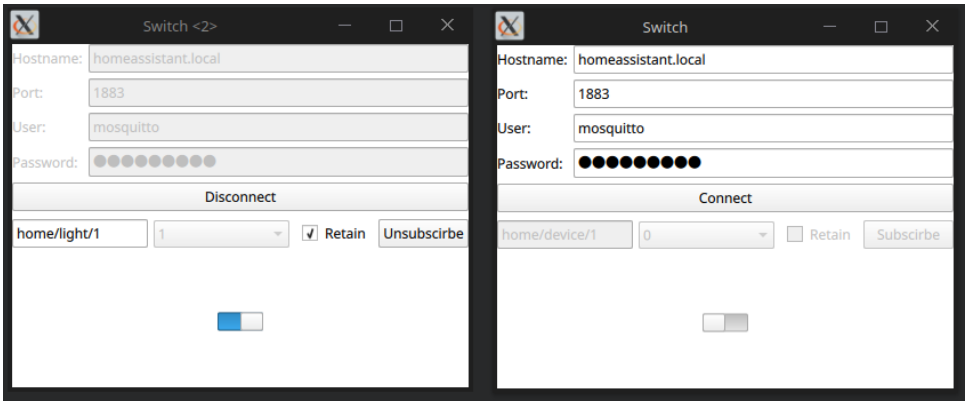

Рисунок 3.1 - інтерфейс перемикача

- Прилад, який має схожість із перемикачем, проте не публікує повідомлення, а лише показує поточний стан. На рисунку 3.2 можна побачити інтерфейс, коли він увімкнений та вимкнений відповідно.

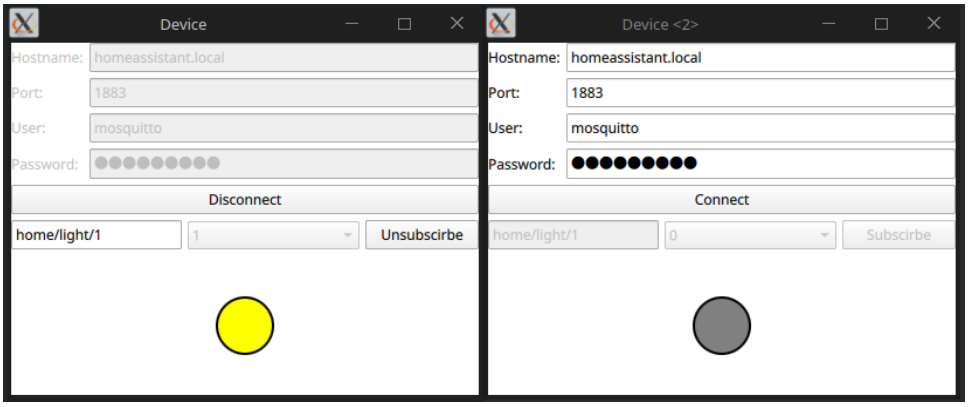

Рисунок 3.2 - інтерфейс пристрою

- Датчик, який кожну секунду випадковим чином збільшує, зменшує на одиницю або не змінює своє значення, та публікує його. На рисунку 3.3 зображено інтерфейс датчика. Для проведення тестувань також додано слайдер, який дозволяє змінити значення на потрібне.

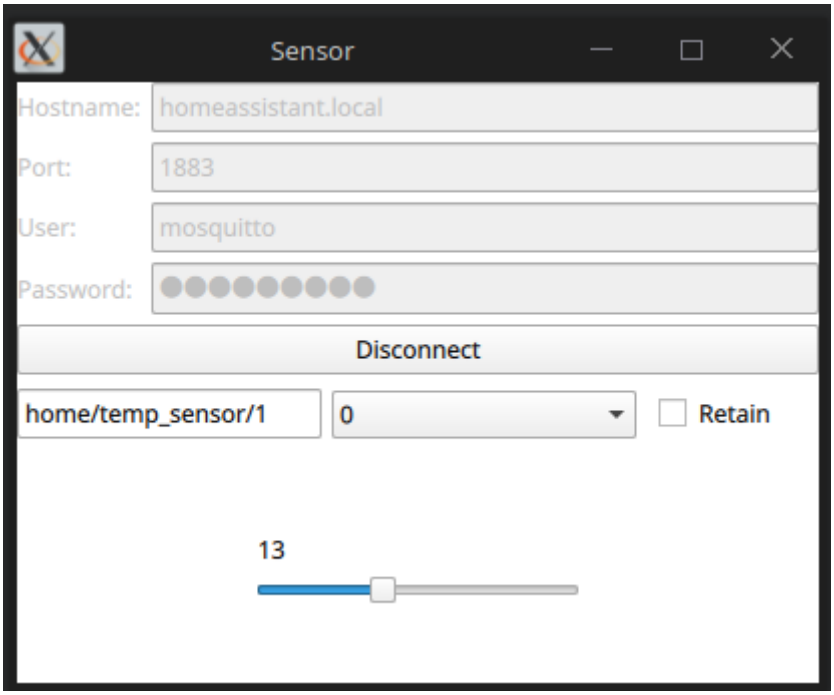

Рисунок 3.3 - інтерфейс датчика

Всі три пристрої мають схожі елементи інтерфейсу:

- Форма для підключення до брокера із можливістю введення адреси та користувача.
- Форма для налаштування публікації та підписки повідомлень. Має поле для введення теми, випадаючий список для вибору якості обслуговування пакетів, а також поле увімкнення збереження повідомлень та кнопку підписки на тему для пристроїв, де це потрібно.

Можливість введення теми, рівня забезпечення якості та збереженого повідомлення надається для тестування різних сценаріїв у використанні протоколу. Проте для нашої системи варто використовувати перший рівень із збереженим повідомленням для приладів та вимикачів, щоб упевнитися у їх синхронізованому стані, та нульовий рівень без збереженого повідомлення для датчиків, оскільки виміри проводяться часто і втрата одиничних повідомлень не є проблемою, а також нам потрібна лише актуальна інформація.

3.3 Використання Home Assistant

У якості хабу для моніторингу та керування нашою системою використано програмне забезпечення під назвою Home Assistant. Це система

автоматизації із відкритим вихідним кодом, яка має багато можливостей інтеграції різних пристроїв та дозволяє керувати ними, зокрема і із застосуванням протоколу MQTT, за допомогою зручного інтерфейсу, який зображено на рисунку 3.4.

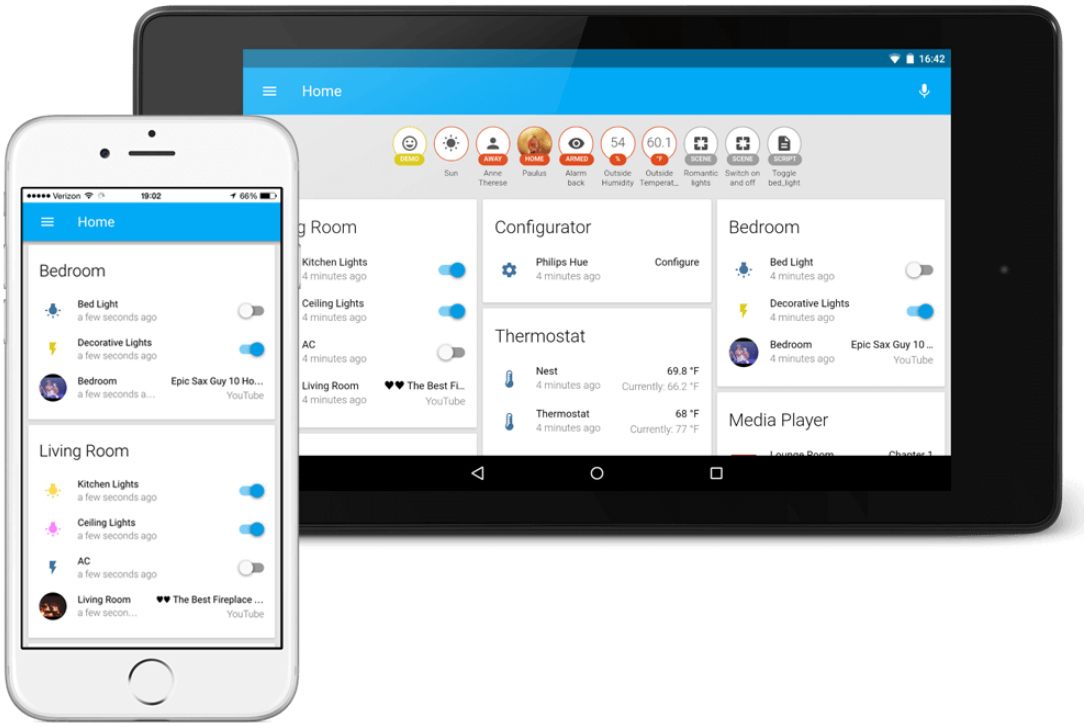

Рисунок 3.4 - інтерфейс Home Assistant [13]

Home Assistant зазвичай встановлюється локально в будинку та має декілька форм. Рекомендованою та з найбільшою кількістю можливостей є форма операційної системи. Її можна встановити на мікрокомп'ютер, наприклад Raspberry Pi, або на сервері за допомогою віртуальної машини. [14]

Одна з можливостей ОС є встановлення доповнень, серед яких є потрібний нам MQTT брокер, а саме Mosquitto. Він має хорошу інтеграцію з системою та може використовувати користувачів Home Assistant для автентифікації клієнтів, тому для його роботи достатньо встановлення за допомогою графічного інтерфейсу. Всі потрібні базові налаштування будуть виконані автоматично. Проте для автентифікації в нашій системі створено окремого користувача, щоб забезпечити обмеження в доступі до можливостей адміністратора.

Home Assistant має хорошу інтеграцію із пристроями які працюють за протоколом MQTT. Підтримується великий список різноманітних типів пристроїв: перемикач та датчик, які використовуються в нашій системі, а також кнопка, камера, регульоване світло, замок, пилосос та інші. Нові пристрої можна додавати двома способами [11]:

- Автоматичне розпізнавання. Для цього способу всі налаштування потрібно виконувати на стороні пристрою, щоб публікувати повідомлення на потрібну тему та у потрібному форматі, який розпізнається програмою.
- Ручне додавання. За допомогою конфігураційного файлу мовою YML можна додати різні види пристроїв, вказавши теми та шаблон їх повідомлень. Приклад конфігурації системи для датчика температури, який публікує дані в тему “home/temp_sensor/1” у форматі JSON, де виміри знаходяться в полі “data” так стають неактуальними через 10 секунд:

```
mqtt:
  sensor:
    - unique_id: temp_sensor1
      name: "Датчик температури"
      state_topic: "home/temp_sensor/1"
      value_template: "{{ value_json.data }}"
      unit_of_measurement: "°C"
      expire_after: 10
```

3.4 Фінальна конфігурація

В результаті проектування, в нашій системі симулюються два електроприлади(світло та кондиціонер), три перемикачі які можуть ними керувати та датчик температури. Всі ці програмовані пристрої можна побачити на рисунку 3.5. Всі вони додані до системи Home Assistant за допомогою конфігурації. Повний файл конфігурації знаходиться в додатку А.

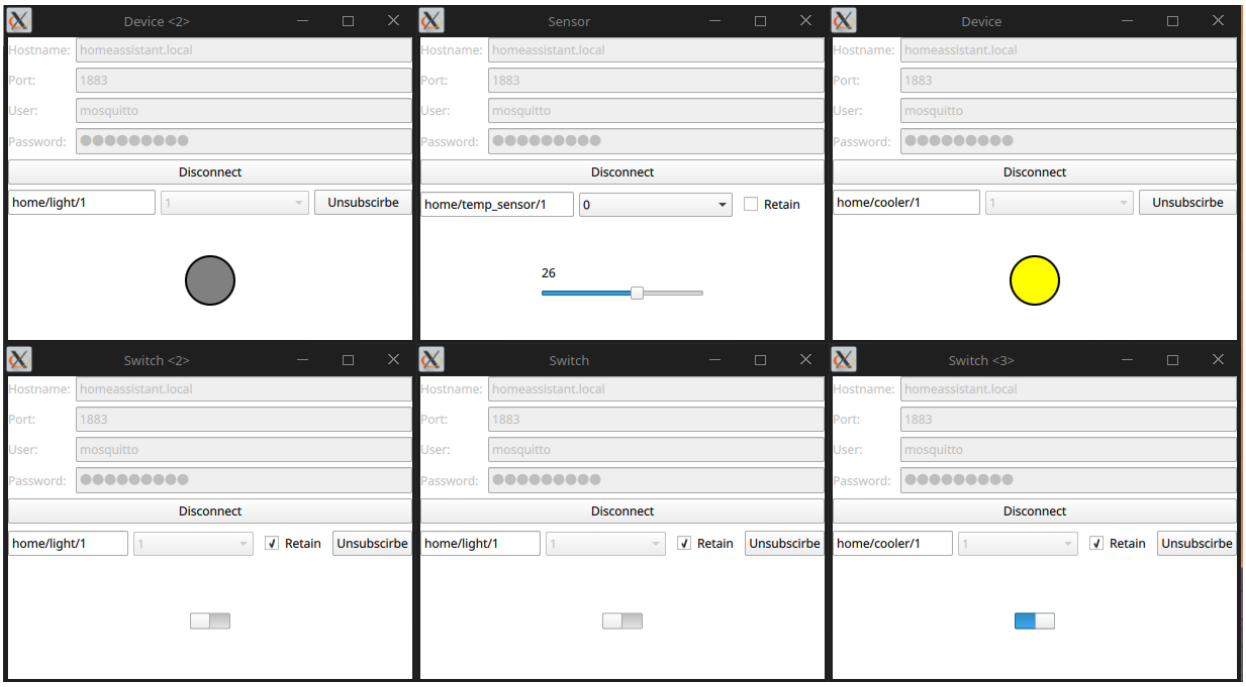

Рисунок 3.5 – пристрої побудованої IoT системи

Крім цього за допомогою датчика температури та увімкнення охолоджувача у Home Assistant створено термостат, який вмикає і вимикає кондиціонер при досягненні заданої межі. Код його конфігурації можна знайти в додатку А. В результаті інтерфейс хабу виглядає як зображено на рисунку 3.6.

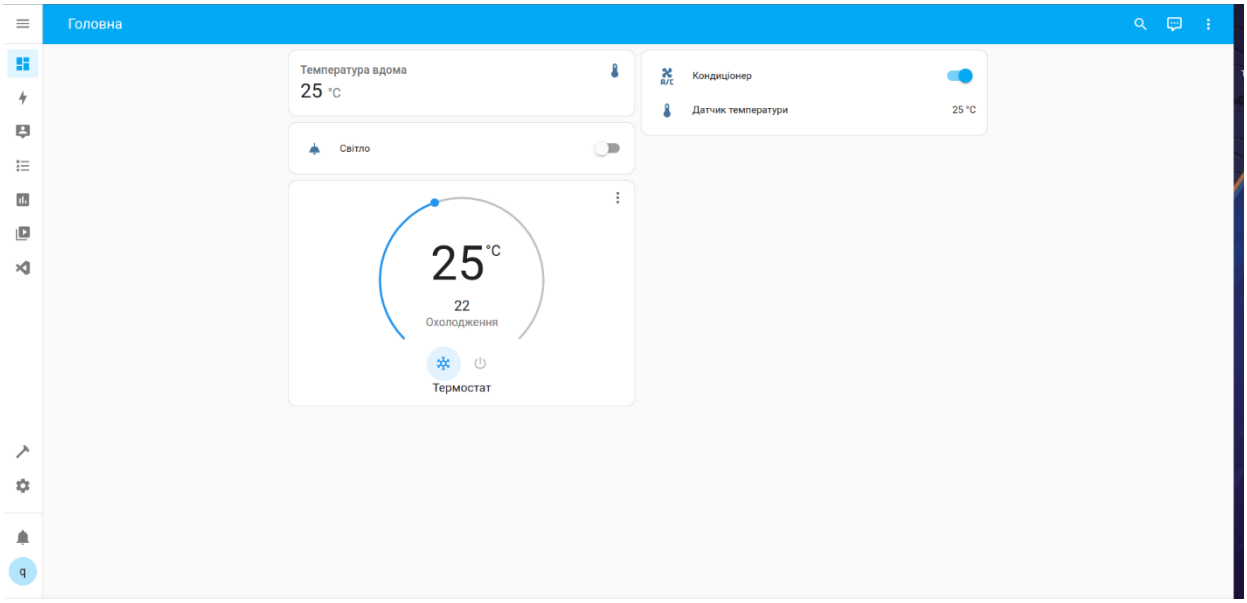

Рисунок 3.6 – інтерфейс хабу Home Assistant

Висновки

В результаті проведеного дослідження можна зробити висновок, що протокол MQTT є ефективним засобом для передачі повідомлень у мережах Інтернету речей. Основні переваги протоколу полягають у його легкості та простоті, а також високій швидкості передачі даних та низькій витраті ресурсів. При цьому, протокол MQTT дозволяє підключати до мережі IoT велику кількість пристроїв та забезпечує масштабованість системи.

У порівнянні з іншими протоколами, такими як HTTP, CoAP та інші, MQTT відрізняється відсутністю необхідності установлювати з'єднання перед кожним відправленням повідомлення. Це дозволяє зменшити навантаження на мережу та забезпечити швидку передачу даних.

Для перевірки можливостей протоколу MQTT, була побудована симуляції системи розумного будинку, яка включає в себе популярні в автоматизації будинку компоненти: перемикач та датчик із здатністю керування електроприладами та моніторингом температури. У результаті, було підтверджено ефективність використання протоколу MQTT при побудові систем IoT, а також його здатність до реалізації функцій та завдань комунікації між пристроями.

Отже, на основі проведеного дослідження можна стверджувати, що використання протоколу MQTT при побудові IoT систем є доцільним та ефективним. Його простота, швидкість та можливість масштабування системи роблять його привабливим засобом для розробки IoT-програмного забезпечення.

Список літератури

1. MQTT Version 5.0. На заміну MQTT Version 3.1.1 ; чинний від 2019-03-07. Вид. офіц. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf (дата звернення: 02.05.2023).
2. Publish & Subscribe - MQTT Essentials: Part 2. HiveMQ. URL: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/ (дата звернення: 02.05.2023).
3. MQTT Quality of Service (QoS) 0,1, & 2 – MQTT Essentials: Part 6. HiveMQ. URL: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/ (дата звернення: 03.05.2023).
4. MQTT Topics, Wildcards, & Best Practices - MQTT Essentials: Part 5. HiveMQ. URL: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/ (дата звернення: 04.05.2023).
5. MQTT Retained Messages - MQTT Essentials: Part 8. HiveMQ. URL: https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages/ (дата звернення: 04.05.2023).
6. Key communication technologies, applications, protocols and future guides for IoT-assisted smart grid systems: A review / M. O. Qays та ін. Energy Reports. 2023. Т. 9. С. 2440–2452. URL: https://doi.org/10.1016/j.egyr.2023.01.085 (дата звернення: 04.05.2023).
7. Craggs I. MQTT Vs. HTTP: Which protocol is the best for IoT or IIoT?. HiveMQ. URL: https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/ (дата звернення: 04.05.2023).
8. Azzola F. CoAP Protocol: Step-by-Step Guide - DZone. Dzone.com. URL: https://dzone.com/articles/coap-protocol-step-by-step-guide (date of access: 05.05.2023).
9. Craggs I. MQTT vs CoAP for IoT. HiveMQ. URL: https://www.hivemq.com/blog/mqtt-vs-coap-for-iot/ (дата звернення: 05.05.2023).

10. Madakam S., Ramaswamy R., Tripathi S. Internet of Things (IoT): A Literature Review. Journal of Computer and Communications. 2015. Т. 03, № 05. С. 164–173. URL: https://doi.org/10.4236/jcc.2015.35021 (дата звернення: 05.05.2023).
11. MQTT. Home Assistant. URL: https://www.home-assistant.io/integrations/mqtt/ (дата звернення: 10.04.2023).
12. Qt MQTT. Qt Documentation. URL: https://doc.qt.io/qt-6/qtmqtt-index.html (дата звернення: 10.04.2023).
13. 0.7: Better UI and improved distribution. Home Assistant. URL: https://www.home-assistant.io/blog/2015/08/31/version-7-revamped-ui-and-improved-distribution/ (дата звернення: 10.05.2023).
14. Installation. Home Assistant. URL: https://www.home-assistant.io/installation/ (дата звернення: 10.04.2023).

Додаток А

(обов’язковий)

Конфігурація системи Home Assistant

```
mqtt:
  sensor:
    - unique_id: temp_sensor1
      name: "Датчик температури"
      state_topic: "home/temp_sensor/1"
      value_template: "{{ value_json.data }}"
      unit_of_measurement: "°C"
      expire_after: 10
  switch:
    - unique_id: light1
      name: "Світло"
      state_topic: "home/light/1"
      command_topic: "home/light/1"
      value_template: "{{ value_json.state }}"
      state_on: true
      state_off: false
      payload_on: '{"state":true}'
      payload_off: '{"state":false}'
      retain: true
      qos: 1
    - unique_id: cooler1
      name: "Кондиціонер"
      state_topic: "home/cooler/1"
      command_topic: "home/cooler/1"
      value_template: "{{ value_json.state }}"
      state_on: true
      state_off: false
      payload_on: '{"state":true}'
      payload_off: '{"state":false}'
      retain: true
      qos: 1

climate:
  - unique_id: thermostat
    platform: generic_thermostat
    name: "Термостат"
    heater: "switch.konditsioner"
    ac_mode: true
    target_sensor: "sensor.datchik_temperaturi"
```

```
min_temp: 16
max_temp: 30
target_temp: 22
cold_tolerance: 0.3
hot_tolerance: 0
min_cycle_duration:
  seconds: 5
keep_alive:
  minutes: 3
initial_hvac_mode: "off"
away_temp: 16
precision: 1
```