DOI: https://doi.org/10.36910/6775-2524-0560-2021-42-30
УДК 658.8:004.73
Старовойтенко Олексій Володимирович
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

ДОТРИМАННЯ ВИМОГ ПЛАТФОРМИ ІОТ ЗА ДОПОМОГОЮ ХМАРНИХ РІШЕНЬ

Старовойтенко О. В. **Дотримання вимог платформи IoT за допомогою хмарних рішень.** Інтернет речей (IoT) надає широкий спектр програм, що забезпечують підвищену обізнаність та контроль за фізичним середовищем. Поточні системи, як правило, локально сприймають і опрацьовують фізичні явища, а потім переносяться на хмарну інфраструктуру pub / sub (публікації / підписки) для розподілу даних датчиків та контролю серед кінцевих користувачів та зовнішніх служб. Незважаючи на популярність рішень pub / sub досі незрозуміло, які функції проміжне програмне забезпечення повинно мати, щоб успішно відповідати конкретним вимогам домену IoT. Питання як велика кількість підключених пристроїв, які лише епізодично надсилають невеликі повідомлення датчиків, впливають на пропускну здатність. У цій роботі ми розглядаємо дані обмеження, аналізуючи основні вимоги платформ IoT та оцінюючи, які з цих функцій підтримуються відомими opensource (відкритими рішеннями)pub / sub. Далі ми проведемо оцінку продуктивності в загальнодоступній хмарі, використовуючи чотири популярні реалізації pub / sub: RabbitMQ (AMQP), Mosquitto (MQTT), Ejabberd (XMPP) та ZeroMQ. Ми дослідимо максимальну стійку пропускну здатність та затримку в справжніх умовах навантаження, використовуючи дані від реальних датчиків. Хоча основні функції подібні, аналізовані системи pub / sub відрізняються своїми можливостями фільтрації, семантичними гарантіями та кодуванням. Наша оцінка показує, що ці відмінності можуть мати помітний вплив на пропускну здатність та затримку хмарних платформ IoT.
Ключові слова: AMQP, pub/sub, ZeroMQ, MQTT,XMPP

Старовойтенко А. В. Соблюдение требований платформы IoT с помощью облачных решений. Интернет вещей (IoT) предоставляет широкий спектр программ, обеспечивающих повышенную осведомленность и контроль за физической средой. Текущие системы, как правило, локально воспринимают и обрабатывают физические явления, а затем переносятся на облачную инфраструктуру pub / sub (публикации / подписки) для распределения данных датчиков и контроля среди конечных пользователей и внешних служб. Несмотря на популярность решений pub / sub до сих пор непонятно, какие функции промежуточное программное обеспечение должно иметь, чтобы успешно отвечать конкретным требованиям домена IoT. Вопросы как большое количество подключенных устройств, лишь эпизодически присылают небольшие сообщения датчиков, влияют на пропускную способность. В этой работе мы рассматриваем данные ограничения, анализируя основные требования платформ IoT и оценивая, какие из этих функций поддерживаются известными opensource (открытыми решениями) pub / sub. Далее мы проведем оценку производительности в общедоступной облаке, используя четыре популярных реализации pub / sub: RabbitMQ (AMQP), Mosquitto (MQTT), Ejabberd (XMPP) и ZeroMQ. Мы исследуем максимальную устойчивую пропускную способность и задержку в настоящих условиях нагрузки, используя данные от реальных датчиков. Хотя основные функции подобные, рассматриваемые системы pub / sub отличаются своими возможностями фильтрации, семантическими гарантиями и кодированием. Наша оценка показывает, что эти различия могут иметь заметное влияние на пропускную способность и задержку облачных платформ IoT.
Ключевые слова: AMQP, pub / sub, ZeroMQ, MQTT, XMPP

Starovoitenko O. V. Compliance with the IoT Platform Requirements through Cloud Solutions. The Internet of Things (IoT) provides a wide range of programs promoting increased awareness and control of the physical environment. Current systems typically perceive and process physical phenomena locally and then transfer to the pub/sub cloud infrastructure to distribute sensor data and control by end users and external services. Despite the popularity of pub/sub solutions, it is still unclear what features middleware must have to successfully meet the specific requirements of the IoT domain. The question is how a large number of connected devices that only occasionally send small sensor messages affect bandwidth. In this paper, we consider these limitations by analysing the basic requirements of IoT platforms and evaluating which of these functions are supported by the well-known open source pub/sub solutions. Next, we will evaluate performance in the public cloud using four popular pub/sub implementations: RabbitMQ (AMQP), Mosquitto (MQTT), Ejabberd (XMPP) and ZeroMQ. We will investigate the maximum stable bandwidth and delay in real load conditions, using data from real sensors. Although the main functions are similar, the analysed pub/sub systems differ in their filtering capabilities, semantic guarantees and encoding. Our assessment shows that these differences can have a significant impact on the bandwidth and latency of IoT cloud platforms.
Key words : AMQP, pub/sub, ZeroMQ, MQTT,XMPP.

Постановка проблеми. Майбутня мережа фізичних об'єктів, Інтернет речей (IoT) забезпечує зондування навколишнього середовища в режимі реального часу і надає можливість здійснювати автономнувзаємодію як реакцію на зміни у фізичному світі. Парадигма призвела до появи різних «розумних» додатків, наприклад розумне місто, розумне підприємство та розумний дім, які, як очікується, збільшать кількість підключених пристроїв до мільярдів [9].

Завдяки своїй величезній обробній потужності, швидкій мережі та надійному сховищу, хмарні обчислення можуть забезпечити інфраструктуру для забезпечення, управліннявеликою кількістю сенсорних пристроїв. Модель гнучкого розподілу ресурсів на вимогу та витрат на оплату за час

роботи може забезпечити еластичне узгодження зростаючих вимог до комунікацій, обчислень та зберігання, пов'язаних із програмами IoT[13].

На відміну від класичних бездротових сенсорних мереж (WSN), які пристосовані і використовуються одним додатком, додаткова цінність IoT полягає в загальному використанні апаратного забезпечення датчиків неоднорідними програмами. Пристрої з обмеженими даними зчитують датчики лише один раз, і вони будуть розподілені між кількома зацікавленими програмами та службами. Масштабований рівень обміну повідомленнями на основі хмари може бути використаний для вирішення складного аспекту узгодження потоків даних сенсора та зацікавлених програм або служб та відповідного розподілу даних.

Структура обміну повідомленнями «pub / sub» дає змогу вибірково розповсюджувати повідомлення і стала добре усталеною моделлю розповсюдження даних, наприклад для даних фондового ринку або погоди. Незважаючи на популярність pub / sub рішень у відповідних застосунках, все ще відсутні дані про особливості, як pub / sub система повинна відповідати конкретним вимогам IoT, ступінь цих функцій, що надаються існуючими рішеннями. У цій роботі ми робимо три основні внески у вирішення цих проблем:

1. Ми формуємо набір вимог до систем pub / sub для хмарних IoT та аналізуємо, які доступні відкриті рішення відповідають цим вимогам;
2. Ми визначаємо три класи трафіку IoT на основі реалістичних випадків використання;
3. Ми проводимо хмарну оцінку продуктивності в загальних реалістичних умовах, використовуючи справжні файли трасування.

Аналіз останніх досліджень і публікацій. Незважаючи на популярність рішень pub / sub досі незрозуміло, які функції проміжне програмне забезпечення повинно мати, щоб успішно відповідати конкретним вимогам домену IoT. Питання як велика кількість підключених пристроїв, які лише епізодично надсилають невеликі повідомлення датчиків, впливають на пропускну здатність в нашому дослідженні ми орієнтувались на наукові праці наступних науковців Cha,M.,Rodriguez,P.,Moon,S.,Crowcroft,J., Curry, E., Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M., Hunkeler, U., Truong, H.L., Stanford-Clark, A., Locke,D., Menzel, T., Karowski, N., Happ, D., Handziski, V., Wolisz, A., Millard, P., Saint-Andre, P., Meijer, R., Saint-Andre, P.

Метою дослідження є дослідити стійку пропускну здатність та затримку в справжніх умовах навантаження, використовуючи дані від реальних датчиків та розкрити відмінності можуть мати помітний вплив на пропускну здатність та затримку хмарних платформ IoT.

Виклад основного матеріалу дослідження.

1. Pub / sub в контекстіIoT

Сьогодні декілька відомих академічних та комерційних платформ IoT мають спільну хмарну архітектуру, подібну до тієї, що зображена на рис. 1 [13]. Пристрої підключені до шлюзів, які передають дані датчиків на хмарний рівень за допомогою посередника повідомлень. Додаткові послуги, напр. зберігання, аналіз або агрегування даних, а програми, що стоять перед користувачами, підключаються до посередника, щоб отримати доступ до даних. Ми бачимо тенденцію використовувати посередника повідомлень, використовуючи шаблон pub / sub для розповсюдження даних серед кількох зацікавлених програм [22]. Система pub / sub - це проміжне програмне забезпечення, орієнтоване на повідомлення (MoM) [10], що забезпечує розподілений, асинхронний обмін данимиміждерелом повідомлень (видавець) та споживачами повідомлень (передплатник). Проміжне програмне забезпечення pub / sub пропонує три основні типи роз'єднання[12], що є придатним для широкомасштабних розгортань IoT:

1. Джерела повідомлень та споживачі роз'єднуються в часі, тобто їх не потрібно підключати одночасно;
2. Повідомлення адресовані не явно конкретному споживачеві, а символічній адресі (каналу, темі);
3. Повідомлення є асинхронними, не блокуючими.

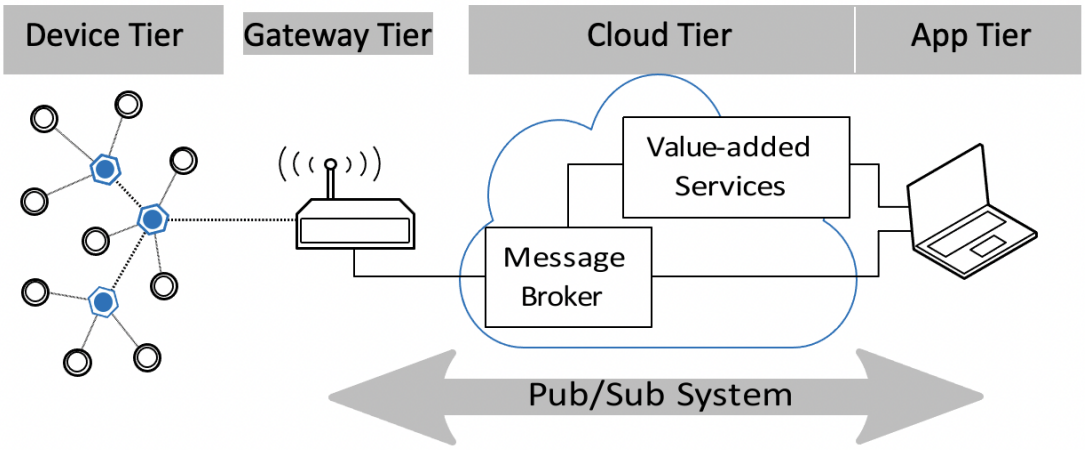


Рис. 1 Агрегування повідомлень в системі pub/sub

Основним елементом систем pub / sub є відповідність між видавцями та передплатниками, яка може базуватися на різних типах фільтрації, переважно за темою чи вмістом. Фільтрування зазвичай здійснюється декількома спеціалізованими посередниками повідомлень. У схемі, заснованій на темах, символічні адреси каналів - це теми, зазвичай у формі рядків, тобто виробники публікують, а споживачі підписуються на теми. Повідомлення доставляються лише відповідним абонентам. Теми можуть бути організовані ієрархічно, тобто тема може бути підтемою іншої теми. Тоді підписки на батьківську тему зазвичай також відповідають усім підтемам. Тематична фільтрація - це статична схема, що пропонує лише обмежену виразність. На противагу цьому, у схемі, що базується на вмісті, абоненти не мають статичного збігу на основі тем, а на основі вмісту окремих повідомлень, наприклад якщо значення досягає певного порогу, визначеного передплатником.

Подібним підходом до великомасштабних даних датчиків є обробка потоків. На відміну від повідомлень на основі pub / sub, програми обробки потоків виступають як складні безперервні запити щодо вхідних потоків даних, що генерують потоки результатів. Особливість потоків обробки потоків полягаєв перетворенні вхідного потоку, тоді як pub / sub фокусується на розподілі даних. Незважаючи на те, що ми розглядаємо потокову обробку як перспективний підхід до обробки та аналізу великих потоків даних, ми стверджуємо, що обробка потоку сама по собі не дасть змоги загально використовувати апаратне забезпечення датчиків у кількох додатках, що є однією з ключових властивостей бачення IoT.

Незважаючи на те, що pub / sub є добре встановленим та вивченим шаблоном обміну повідомленнями, використання pub / sub в хмарних налаштуваннях IoT все ще не досліджено в деталях. Існує безліч існуючих рішень без надійних даних про те, яке рішення найкраще відповідає конкретним вимогам IoT. У наступному розділі ми починаємо вирішувати ці проблеми, визначаючи довідкові сценарії для хмарних розгортань IoT.

2.1 Довідкові сценарії

Перш ніж виводити набір вимог до загальних програм IoT, спочатку ми вводимо підмножину можливих сценаріїв, визначивши в якості прикладів три еталонні сценарії в контексті розумних міст:

2.1.1 Соціальна служба погоди

Першим варіантом використання, який ми передбачаємо, є соціальна метеорологічна служба, де власники сенсорних пристроїв підключаються і діляться своїми наявними датчиками з громадськістю. Прикладом є система опалення, вентиляції та кондиціонування повітря (ОВК) в офісній будівлі, яка періодично вимірює температуру та вологість. Система може або публічно надавати власні дані, або використовувати показання з інших подібних систем, або контролювати прогноз погоди, щоб покращити час нагрівання чи охолодження, заощаджуючи енергію та покращуючи комфорт.

2.1.2 Розумний спільний доступ до автомобіля

Другий варіант використання - це система спільного використання автомобілів. Поява підключених автомобілів призвело до появи спільного використання. Спільне використання автомобілів зменшує кількість необхідних місць для паркування та, в свою чергу, зменшує кількість трафіку. У майбутньому фізичні особи можуть почати здавати свої машини в оренду подібним чином: приватні машини можуть періодично запропоновані в оренду , надаючи додаткову інформацію, таку як положення, рівень палива чи стан.

2.1.3 Моніторинг дорожнього руху

Щоб зменшити обсяг руху, іншим варіантом є відстеження дорожнього руху. Камери фотографують умови дорожнього руху та відправляють їх у хмару. Хмарний сервіс аналізує їх та оцінює стан дорожнього руху. Зацікавлені водії або навігаційні підрозділи можуть використовувати цю інформацію, щоб спланувати свій маршрут.

2.2 Вимоги до pub / sub в хмарному IoT

У цьому розділі викладено конкретні вимоги щодо Інтернету речей, які впливають на вибір відповідного проміжного програмного забезпечення pub / sub. Ми починаємо з функціональних вимог, які походять від загальних програм IoT і проілюстровані наведеними вище довідковими сценаріями:

1. Шаблон обміну повідомленнями: Усі випадки використання вимагають моніторингу показань датчиків.
2. Фільтрування: Зацікавлені сторони зазвичай хочуть отримати лише підмножину всієї інформації, наприклад датчики погоди в тому самому районі. Можливості фільтрації проміжного програмного забезпечення визначають виразність підписок, які може видавати клієнтська програма.
3. Семантика QoS: хоча втрата повідомлень даних датчика може бути допустимою в деяких налаштуваннях, інші системи можуть вимагати гарантованої доставки повідомлень, наприклад розумний автомобіль може видавати тривожні повідомлення у випадку крадіжки.
4. Топологія: У контексті хмарно-орієнтованого IoT кожен проміжний пакет pub / sub повинен підтримувати централізовану топологію, де брокер пересилає повідомлення на основі запитуваних фільтрів.
5. Формат повідомлення: Через неоднорідність апаратного забезпечення датчиків, як проілюстровано у вибраних випадках використання, складно передбачити, які саме будуть подані дані датчиків формату.

Ці вимоги будуть використані в наступному розділі для виведення таксономії систем pub / sub. На додаток до вищезазначених вимог, існують дві важливі нефункціональні вимоги до хмарної системи pub / sub:

1. Через велику кількість очікуваних пристроїв система повинна бути масштабованою;
2. Оскільки дані можуть бути терміновими, обмін повідомленнями повинен мати низьку затримку. Ми досліджуємо ці вимоги окремо під час оцінки результатів роботи в Розділі 3.

2.3 Таксономія pub / sub

Спираючись на загальні вимоги до платформ IoT, у цьому розділі ми представляємо підмножину протоколів pub / sub та надамо класифікацію відповідно до визначених вимог. Ми обмежуємось протоколами звідкрито доступна специфікація протоколу з реалізаціями з відкритим кодом, які широко використовуються. Ми обираємо протоколи AMQP, MQTT, XMPP та ZeroMQ для подальшої оцінки. Далі ми коротко описуємо ці протоколи:

AMQP: Розширений протокол черги повідомлень (AMQP) [2] розроблений як відкрита заміна власних протоколів обміну повідомленнями у галузі фінансових послуг. AMQP використовується в популярних реалізаціях, таких як RabbitMQ[22], ApacheActiveMQ [4] та ApacheApollo [5]. Версія AMQP 0-9-1 - це відкрита та безкоштовна специфікація як протоколу дротового рівня, так і моделі брокера. AMQP 1.0 нещодавно став Організацією з удосконалення стандартних стандартних інформаційних стандартів (OASIS), але включає в основному новий провідний протокол і лише

абстрактні вимоги до посередника. Більшість функціональних можливостей визначаються брокером та його поведінкою, яка не є частиною AMQP 1.0. Тому ми зосереджуємося на версії 0-91, яка визначає найбільш широко застосовувану модель брокера.

MQTT: Телеметричний транспорт черги повідомлень (MQTT) [18] - це чистий протокол pub / sub для обмежених пристроїв та мереж з низькою пропускною здатністю, високою затримкою та ненадійністю, розроблених IBM та стандартизованих OASIS. Він має різні реалізації з відкритим кодом, такі як ApacheActiveMQ [4] та клієнтська бібліотека Eclipsepaho[11] Існує спроба перенести MQTT на датчики вузлів у зменшеному варіанті MQTT -S [16]. MQTT та MQTT-S відкрито публікуються з безкоштовними ліцензіями.

XMPP: Розширюваний протокол обміну повідомленнями та присутності (XMPP) [18] бере свій початок у протоколі обміну миттєвими повідомленнями Jabber, мотивованому різноманітністю власних протоколів чату. XMPP базується на потоковій передачі XML. Основний протокол стандартизований у декількох RFC IETF та розширений протоколами розширення XMPP (XEP), включаючи обмін повідомленнями pub / sub[16]. XMPP - це протокол, який стоїть за сервером обміну миттєвими повідомленнями Ejabberd[16] та сервером Openfire[17].

ZeroMQ: ZeroMQ[15] - це бібліотека обміну повідомленнями, що пропонує API сокета з більш досконалими шаблонами обміну повідомленнями, ніж сокетиБерклі. Підтримувані шаблони включають запит / відповідь tapub / sub. Протоколом дротового рівня є ZeroMQMessageTransportProtocol (ZMTP), щодоступний за загальною публічною ліцензією GNU. ZeroMQ - це бібліотека обміну повідомленнями, що дозволяє додавати нові функції до основного протоколу. Для порівняння бібліотеки з іншими системами ми розглядаємо лише функції, включені в офіційну документацію.

В решті цього розділу ми наводимо класифікацію pub / sub протоколів щодо вимог платформи IoT на основі хмарних технологій, яка наведена в табл 1.

		AMQP	MQTT	XMPP	ZeroMQ
Messaging Pattern	Pub/Sub	X	X	X[1]	X
	Point-to-point	X		X	X
Filtering	Topic-based	X	X	X1	X
	Content-based				
QoS Semantics	At-most-once	X	X	X	X
	At-least-once	X	X		
	Exactly-once	X	X		
	Last value caching	X[2]	X	X	X
Topology	Decentralized				X
	Centralized	X	X	X	X
	Hybrid			X1	
Message Format	Payload agnostic	X	X	X	X
	Binary Encoding	X	X		X

Таблиця 1

Класифікація відкритих pub / sub протоколів проміжного програмного забезпечення

Ми використовуємо шаблони pub / sub та запит / відповідь для моніторингу сенсорного пристрою. Хоча AMQP і ZeroMQ підтримують обидва шаблони, MQTT - це чистий протокол pub / sub. Запит / відповідь все ще можливий, але він повинен використовувати спеціальні теми, щодо яких

пристроїв чи служб підписуються на запити. XMPP спочатку є протоколом чату для прямого обміну повідомленнями, який був розширений XEP-0060 для підтримки публікацій / додаткових повідомлень [16].

Розглянуті рішення надають лише тематичні pub / sub з ієрархічними темами. AMQP та MQTT підтримують узагальнюючі символи на кожному рівні теми, тоді як ZeroMQ підтримує лише відповідність префіксів.

Ми зосереджуємося на хмарних платформах IoT із централізованою топологією. Це означає, що спеціалізовані розподілені брокери в центральній точці здійснюють узгодження та доставку повідомлень. Усі розглянуті протоколи підтримують таку топологію. В умовах очікуваної кількості пристроїв буде вигідно зменшити навантаження на центральних брокерів. XMPP використовує розподілену мережу серверів XMPP як посередників. Також можливо встановити позасмугові однорангові з'єднання за допомогою XEP-0065, що відповідає гібридній топології. ZeroMQ можна використовувати без центрального брокера як P2P. Крім того, його сокетиpub / sub можуть бути використані для формування ієрархії брокерів, які вибірково пересилають публікації та підписки, дозволяючи горизонтальне масштабування.

Усі протоколи, як правило, є агностичними щодо формату повідомлення. AMQP, MQTT та ZeroMQ використовують двійкове кодування, дозволяючи будь-який формат, тоді як XMPP використовує XML, який також може транспортувати інші формати, але зазвичай ціною меншої ефективності пропускної здатності та збільшення накладних витрат на аналіз. AMQP додатково пропонує систему типів, яка відображає передані типи даних у загальні типи даних багатьох мов та платформ.

3 Хмарна оцінка продуктивності

Хоча аналізовані системи pub / sub пропонують однакові основні функції, вони відрізняються семантикою QoS, кодуванням та виразністю фільтрів, що може мати незначний вплив на продуктивність. Щоб проаналізувати нефункціональні вимоги до масштабованості та низької затримки, ми оцінюємо системи в реальних умовах. Одним із основних аспектів цих реалістичних умов є розгортання в хмарі. Іншим аспектом є реалістичне відтворення реальних джерел трафіку за допомогою великої кількості пристроїв.

3.1 Показники ефективності

Ми вибираємо показники продуктивності відповідно до нефункціональних вимог до масштабованості та низької затримки. Оскільки хмарні платформи IoT, можливо, в майбутньому повинні будуть підтримувати мільярди пристроїв [13], важливо, щоб система могла масштабуватися горизонтально. Зазвичай це робиться за допомогою кластера брокерів. Однією з можливостей буде шардинг на основі тем. Однак для роботи всієї системи найважливіше, щоб один брокер міг обробляти якомога більше датчиків і, отже, повідомлень датчиків, тобтовін пропонує найбільшу пропускну здатність повідомлень на даній фізичній або віртуальній машині. Тому ми вивчаємо пропускну здатність одного брокера, яку ми визначаємо як кількість повідомлень в секунду, яку може обробити брокер.

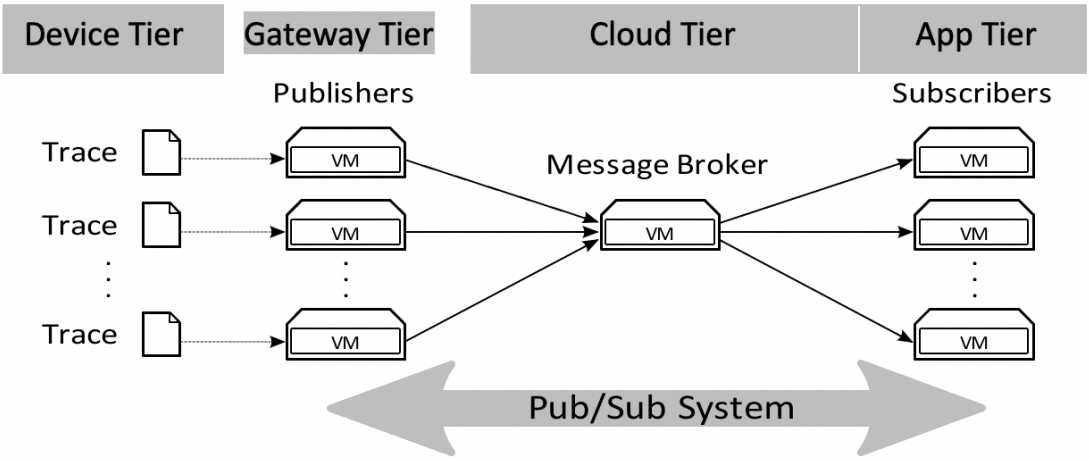


Рис. 2 Налаштування експерименту: реальні дані датчиків збираються та відтворюються в екземплярах хмари з файлів трасування; брокер розподіляє ці дані між процесами підписки, які також моделюються за допомогою хмарних віртуальних машин.

Protocol	Broker	Version	Release date	Language
AMQP	RabbitMQ	3.4.4	2015-02-11	Erlang
MQTT	mosquitto	1.4	2015-02-18	C
XMPP	ejabberd	15.02	2015-02-17	Erlang
ZeroMQ	XPUB/XSUB	-	-	C

Табл. 2Оглядвивчених систем pub / sub

Іншою важливою вимогою є низька затримка між вимірюванням значень датчика та доставкою до зацікавлених застосунків. У цьому дослідженні ми не зосереджуємося на загальній затримці, а лише на тій частині затримки, яку запроваджує система pub / sub. Таким чином, ми вивчаємо кінцеву затримку між шлюзами тазацікавлені програми, тобто між видавцями та передплатниками.

3.2 Загальні налаштування вимірювань

Далі ми представляємо налаштування вимірювань, які ми використовуємо для оцінки вибраних систем за допомогою представлених показників. Наші вимірювання спрямовані на максимально реалістичну оцінку роботи брокера обміну повідомленнями на основі хмар. Огляд нашої системи наведено на рис. 2. Оскільки брокер, як очікується, буде працювати на хмарній віртуальній машині, ми розгортаємо брокер на екземплярі в AmazonElasticComputeCloud (Amazon EC2) [1], який є одним із провідних хмарних провайдерів.

Оскільки ми не можемо розгорнути апаратне забезпечення сенсорів, що взаємодіє з брокером повідомлень, ми дотримуємося подібного підходу, як Rege та ін. [17] та відтворювати джерела трафіку на екземплярах хмар. Рівень пристрою змодельований файлами трасування, зібраними із сенсорних пристроїв реального світу. Детальніше - у наступному розділі. Файли трасування використовуються для копіювання поведінки синхронізації та розмірів повідомлень трафіку, що спостерігаються на шлюзах IoT. Шлюзи, які виконують роль видавців у системі pub / sub, а також застосунки, що підписуються, копіюються на хмарних віртуальних машинах.

Щоб уникнути відтворення одних і тих самих файлів трасування на різних шлюзах і потенційно зміщення результатів, розподіл розмірів повідомлень та часу між повідомленнями обчислюється для кожних 15-хвилинних інтервалів і відтворюється. Це зберігає інформацію про різні розміри та швидкість повідомлень протягом дня, які зазвичай співвідносяться у розумному місті, але уникає нереальних упереджень, спричинених штучним високим співвідношенням.

Ми оцінюємо протоколи, використовуючи популярні реалізації представників. Наш вибір брокерів наведено в табл. 2. Брокер ZeroMQ використовує сокети XPUB і XSUB, як зазначено в документації ZeroMQ, і написаний на мові програмування C. Брокери Erlang використовують високопродуктивний Erlang (HiPE). Оскільки ми не враховуємо безпеку, шифрування вимкнено. Брокерські та публікаційні та передплатні процеси розгортаються на екземплярах EC2, що працюють у поточному 64-бітній версії довгострокової підтримки Ubuntu (14.04) (ami-234ecc54). Апаратне забезпечення узагальнено в табл. 3. Брокери працюють на одному екземплярі m4.large з двома ядрами, як правило, IntelXeon E5-2676 v3 з 32 ГБ оперативної пам'яті. Виробники та споживачі повідомлень розміщені на екземплярах m3.medium з одним ядром та 3,75 ГБ оперативної пам'яті. Ми використовуємо 10 екземплярів шлюзу, в яких розміщується до 60 віртуальних шлюзів публікації та до 60 процесів підписки кожен. Під час експериментів контролюється навантаження центрального процесора на кожен екземпляр, тому вузькі місця на стороні клієнта не будуть розцінюватися як погана робота брокера.

Для генерації трафіку та тиражування процесів підписки ми використовуємо клієнт, написаний мовою програмування C. Застосунок публікує та підписується за допомогою єдиного інтерфейсу, який відображається на інтерфейсах окремих систем pub / sub за допомогою спеціальних плагінів.

Компонент генерації трафіку та ведення журналу залишається незмінним у кожному експерименті. Щоб уникнути побічних ефектів дискового вводу-виводу, ми буферизуємо всі результати в пам'яті, перш ніж записувати їх на диск. Ми використовуємо непостійні повідомлення з максимальною кількістю семантичних показників із фіксованим коефіцієнтом розсилки 1, тобто кожен видавець має одного передплатника.

У цій роботі ми вдаємося до використання чітко визначеного коефіцієнта розмивання як першого кроку, але працюємо над вдосконаленими моделями абонентів для подальшої роботи. По-друге, наша система спрощує вимірювання пропускної здатності та затримки, оскільки ми можемо розгортати видавців та підписників теми на одній і тій же хмарній віртуальній машині, уникаючи необхідності точної синхронізації між процесами.

Ми оцінюємо кожен протокол, використовуючи три сценарії робочого навантаження, аналогічні нашим еталонним сценаріям, які ми визначаємо нижче.

3.3 Сценарії навантаження

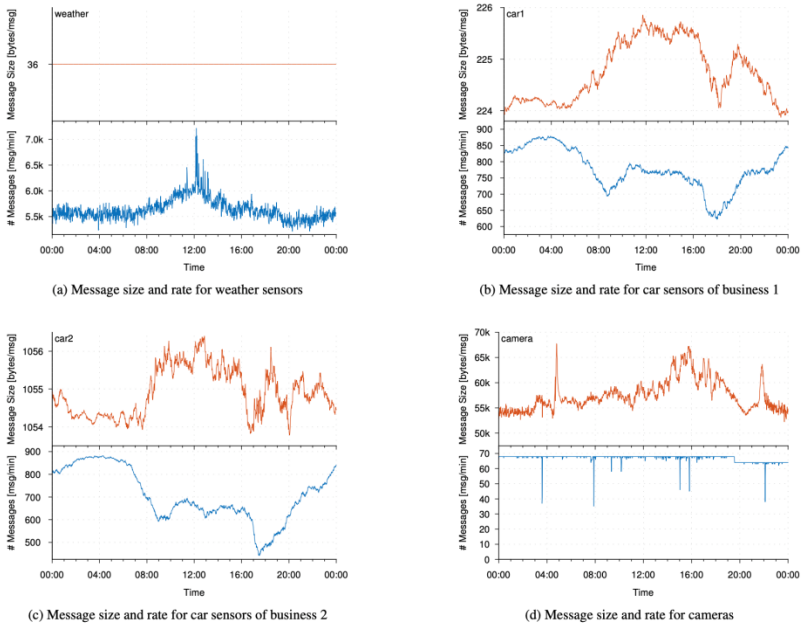
Ми узагальнюємо три еталонні сценарії в Розділі 2.1 для трьох основних класів датчиків, що виробляють чіткі схеми руху:

3.3.1 Прості датчики

Датчики погоди - це приклад того, що ми визначаємо як простий датчик, який відбирає по одній величині за раз, як правило, незалежний від інших датчиків і компактне двійкове представлення.

Тестовий стенд бездротового внутрішнього датчика (TWIST) [14] з 90 вузлами датчиків TelosRev. B [18], оснащений датчиком вологості та температури Sensirion SHT11, датчиком видимого світла Hamamatsu S1087 (від 320 нм до 730 нм), що вимірює фотосинтетично активне випромінювання (PAR)) і Hamamatsu S1087-01 (від 320 нм до 1100 нм), видимий для ІЧ-датчика світла, що вимірює загальну сонячну радіацію (TSR). Ми збираємо зразки згаданих вище датчиків на кожному вузлі датчика щосекунди, використовуючи програму TinyOS, яка відправляє дані в одному пакеті до послідовного інтерфейсу, який виставляється через порт UniversalSerialBus (USB). Процес збору використовує можливості тестового стенду TWIST: кожен вузол датчика підключений до одного супервузла, який пересилає послідовні пакети на сервер TWIST, де записується файл трасування з усіма значеннями датчика. Створюється файл трасування із загальною кількістю даних за 72 години.

Ми використовуємо ці необроблені дані датчиків для створення сигналів датчиків для публікації. Ми вирішили надіслати повідомлення з оновленими даними датчика, лише якщо перевищені розумні порогові значення, тобто якщо датчик відбирає значення, близьке до значення, відібраного раніше, шлюз не передаватиме дані повторно. Ми вважаємо, що це розумна поведінка в бездротових сенсорних мережах, де енергії недостатньо. Огляд сформованого файлу трасування наведено на рис 3а, де показано один день зразкових даних. Кількість виданих повідомлень за секунду варіюється і досягає максимуму близько обіду. Оскільки повідомлення кодуються двійково, розмір повідомлення постійний на рівні 36 байт.



(a) Message size and rate for weather sensors
(b) Message size and rate for car sensors of business 1
(c) Message size and rate for car sensors of business 2
(d) Message size and rate for cameras

Рис. 3 Часові характеристики сенсорних файлів датчиків протягом одного конкретного дня вимірювання; вказано розмір повідомлення та швидкість повідомлень за хвилину.

3.3.2 Складні датчики

На відміну від простих показників, які пов'язані, але також мають значення самі по собі, датчики автомобіля в основному мають значення лише разом. Беручи до уваги приклад спільного використання автомобілів, лише рівень палива, наприклад, не дуже значущий для потенційних споживачів без положення автомобіля. Ми визначаємо такі датчики як складні датчики, що являє собою комбінацію декількох простих датчиків, де показання датчиків згруповані, щоб утворити цілісні показники, які підключені та передаються разом як одне повідомлення. Ці повідомлення зазвичай кодуються за допомогою JSON або XML.

Дані про транспортний засіб, як складний тип даних датчиків, походять із загальнодоступних веб-сайтів двох підприємств спільного користування автомобілями [6, 11]. Подібно до того, як браузер отримує доступ до даних, щоб скласти карту доступних автомобілів, ми щохвилини отримуємо доступ до загальнодоступних даних цих служб для Берліна і розподіляємо відповідь в індивідуальних звітах про стан для кожного автомобіля. Ми реєструємо розмір цих індивідуальних показань і використовуємо їх для відтворення еквівалентних повідомлень у наших вимірах, не зберігаючи жодних фактичних даних про машини. Файли трасування містять дані за тиждень для обох компаній, що надають послуги спільного використання автомобілів.

Індивідуальні звіти про стан автомобілів використовуються як повідомлення для публікації. Обґрунтування полягає в тому, що кожен автомобіль опублікує його наявність разом із додатковою інформацією про автомобіль, такою як стан чи стан. Один день даних наведено на рис. 3b та 3c для кожного бізнесу, що займається спільним використанням автомобілів. Зміна кількості доступних автомобілів відбувається за тією ж загальною схемою і є низькою близько 9:00 та між 17:00 і 20:00, коли багато пасажирів орендують машини. Хоча компанія, що займається спільним сполученням автомобілів, зазвичай отримує звіти про стан або з фіксованим інтервалом, або за певних подій, кількість приватних автомобілів, що надаються громадянами в оренду, мабуть, має подібну модель. Розмір повідомлення досить стабільний протягом дня, але суттєво відрізняється між двома компаніями.

3.3.3 Мультимедійні датчики

Прикладами камер як датчиків IoT є зображення камер спостереження або відеопотоки, а також спостереження за дорожнім рухом за допомогою камер. Зображення дорожньої камери можна аналізувати в режимі реального часу, щоб запропонувати альтернативні шляхи до розумних навігаційних систем. Ми визначаємо клас мультимедійних датчиків, які виробляють носії, такі як

аудіо, нерухомі зображення або відеоматеріали. Ці медіапотоки кодуються як двійкові фрагменти даних.

Як приклад медіа-даних ми використовуємо загальнодоступні камери дорожнього руху міста Берлін. Подібно до автомобілів, ми отримуємо доступ до зображень з камери з фіксованим інтервалом у 30 секунд і реєструємо розмір відповіді. Файли трасування містять дані за тиждень для 34 камер.

Окремі зображення використовуються як повідомлення для публікації. Приклади даних за день показані на рис. 3d. Середній розмір повідомлення збільшується до півдня, можливо, тому, що зображення з більшою кількістю світла матимуть більше деталей, які не будуть стискатися так добре, як темні зображення. Швидкість повідомлень є досить стабільною, як очікується, приблизно два зображення на хвилину на камеру. Деякі камери, як правило, не оновлюються надійно, тому ми ігноруємо всі точні копії, пояснюючи незначні коливання.

3.4 Вимірювання пропускної здатності

Ми використовуємо класи трафіку, введені раніше, для хмарної вимірювальної кампанії, що визначає типову пропускну здатність та затримку систем pub / sub. Ми починаємо вимірювати пропускну здатність, застосовуючи все більше навантаження на брокера. Це досягається більшою кількістю пар видавців та передплатників. Для сценарію датчика погоди ми змінюємо кількість публікаційних шлюзів від 10 до 600, при цьому кожен шлюз генерує навантаження, еквівалентну 1000 датчикам, що відповідає загальній кількості до 600 000 емульованих датчиків. Для випадку спільного використання автомобілів ми використовуємо від 10 до 600 видавців, які генерують вантаж, еквівалентний до 9,6 мільйона окремих автомобілів. Ми варіюємо кількість камер від 40 до 600, де кожен видавець імітує лише одну камеру. Збільшення кількості пар видавець / передплатник є експоненціальним для точного вимірювання як дуже малих, так і дуже високих значень пропускної здатності. Експерименти складаються з генерування навантаження протягом 5 хвилин та вимірювання кількості повідомлень, які надходять до абонентів за цей час. Навантаження генерується відповідно до випадкової позиції в одному з файлів трасування. Кожна конфігурація параметрів застосовується загалом 8 разів.

На рис.4 показана виміряна пропускна здатність, коли брокер завантажений різними робочими навантаженнями, визначеними раніше. Насиченість пропускної здатності позначає індивідуальну максимальну стійку пропускну здатність протоколів за певного сценарію. Результати показані з 95% довірчим інтервалом.

На рис. 4a зображені результати для погодних датчиків. ZeroMQ має найвищу пропускну здатність - до 522 тис. повідомлень / с. Інші протоколи мають значно меншу пропускну здатність:

MQTT показує максимальну пропускну здатність, меншу за третину від пропускної здатності 134 тис. повідомлень / с. AMQP має максимальну пропускну здатність 30 тисяч повідомлень / с. Максимальний розмір XMPP в 1 тис. повідомлень / с приблизно на два порядки нижче, ніж пропускна здатність MQTT.

Рис. 4b показує результати для випадку використання автомобілів. ZeroMQ знову показує найвищу максимальну пропускну здатність до 83 тис. повідомлень / с. Примітно, що в цьому випадку використання, хоча і досі не досягає пропускної здатності ZeroMQ, розрив між протоколами зменшився: MQTT показує пропускну здатність 69 тис. повідомлень / с. AMQP має максимальну пропускну здатність 26тис. повідомлень / с. До 2 тис. повідомлень / с пропускна здатність XMPP нижча за інші протоколи, але напрочуд вища, ніж для менших повідомлень.

На рис. 4в наведені результати щодо трафіку камери. У цьому випадку AMQP має найвищу пропускну здатність з максимальною швидкістю 186 тис. повідомлень / с. ZeroMQ також показує максимальну пропускну здатність до 186 тис. повідомлень / с, що трохи нижче AMQP лише після десяткової коми. MQTT показує максимальну пропускну здатність 164 тис. повідомлень / с. XMPP показує пропускну здатність до 92 повідомлень / с, але не може впоратися з навантаженням понад 300 повідомлень / с. Якщо для інших еталонних сценаріїв ZeroMQ був безперечним лідером, то для сценарію камери протоколи показують набагато більше подібних результатів.

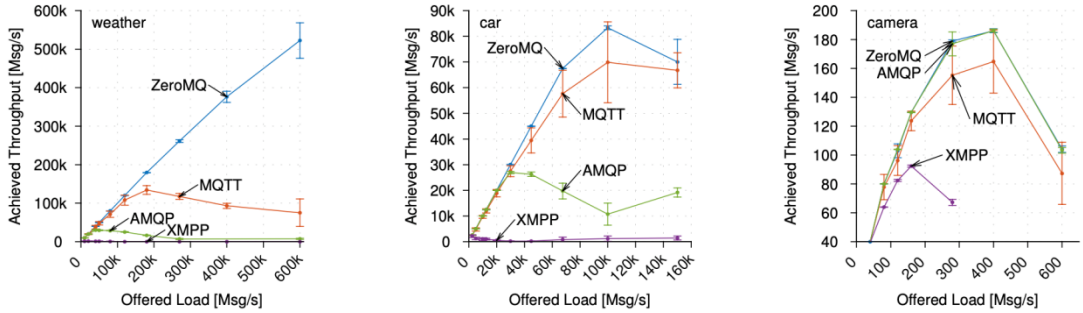
Ми пояснюємо таку поведінку наступним чином: наш брокер ZeroMQ є лише дуже простим і підтримує лише відповідність префіксів. Через це пропускна здатність значно вища, ніж інші протоколи, особливо для менших повідомлень. MQTT та AMQP пропонують більш складну фільтрацію, збільшуючи час, витрачений на кожне рішення про фільтрацію. XMPP має найбільші

накладні витрати завдяки використаному кодуванню XML, яке доводиться аналізувати клієнтам і брокеру, що призводить до подальшої обробки накладних витрат. Щодо великих повідомлень, MQTT та AMQP стають більш конкурентоспроможними, оскільки час, витрачений на прийняття рішення про фільтрацію, не залежить від розміру повідомлення, а час копіювання в пам'яті, а також мережева передача повідомлень стають домінуючими факторами часу обробки.

3.5 Вимірювання затримки

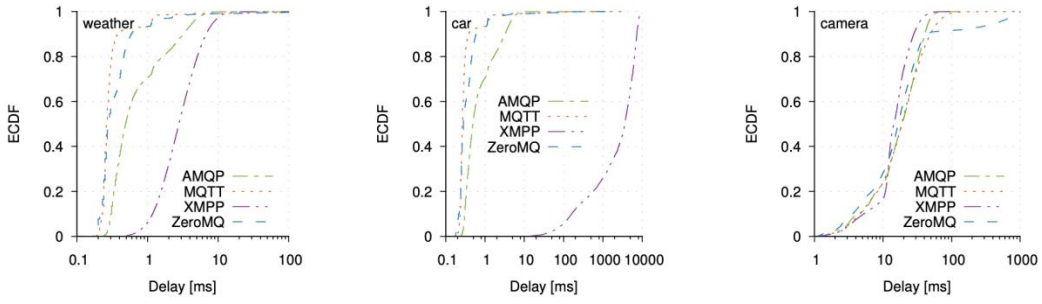
Ми також досліджуємо затримку протоколів. Оскільки затримка у насиченому середовищі була б нереально високою, ми спостерігаємо розподіл латентності протягом одного дня в цьому окремому експерименті. Ми копіюємо 40 мереж датчиків погоди розміром 200 вузлів кожна, що моделює мережу датчиків середнього розміру, таку як тестовий стенд TWIST. Ми використовуємо 100 мереж спільного використання автомобілів, подібних до тих, що записані в Берліні. Нарешті, ми запускаємо одну мережу камер, подібну до тієї, що вимірюється у Берліні з 34 камерами.

Результати представлені на рисунку 5. Окремі графіки показують емпіричну функцію розподілу затримки, введеної системою pub / sub. Графіки ілюструють, який відсоток успішно переданих повідомлень відчуває наскрізну затримку між видавцем та передплатником.



(a) Throughput under weather sensor load. (b) Throughput under car sharing load. (c) Throughput under camera load.

Рис. 4 Пропускна здатність, досягнута протоколами у трьох еталонних сценаріях.



(a) Delay under weather sensor load. (b) Delay under car sharing load. (c) Delay under camera load.

Рис. 5 Емпірична кумулятивна функція розподілу (ECDF) результуючої затримки при застосуванні різних навантажень.

Оскільки виміряна затримка має діапазон, який охоплює кілька порядків величини, ми використовуємо логарифмічну шкалу для кращого порівняння результатів.

Для сценарію датчика погоди результати на рис. 5a показують, що розподіл затримок MQTT та ZeroMQ досить подібний. Більше 90% повідомлень мають затримку менше 1 мс. AMQP показує трохи більшу затримку. Тим не менше, близько 70% виміряних значень затримки нижче 1 мс і 90% нижче 3 мс. XMPP вводить значно більшу затримку, але все-таки менше 3% повідомлень затримується більше ніж на 10 мс.

Отримані розподіли затримок, виміряні для навантаження на спільний рух автомобілів, показані на рис. 5b. Хоча повідомлень значно більше, розподіл затримок ZeroMQ, MQTT та AMQP дуже схожий на попередній сценарій. Використовуючи MQTT та ZeroMQ, більше 90% виміряних значень затримки нижче 1 мс. З AMQP приблизно 70% повідомлень мають затримку менше 1 мс і

приблизно 90% менше 3 мс. XMPP показує значно більші затримки до 10 с, що свідчить про ситуацію з перевантаженням.

Розподіл затримки для випадку використання камери наведено на рис. 5c. Загалом затримка всіх протоколів вища, ніж для датчиків з меншими розмірами повідомлень. Крім того, розподіл затримки не відрізняється настільки суттєво, як для менших розмірів повідомлень. Для всіх протоколів більше 90% затримки менше 100 мс. Хоча частіше досягається менша затримка, для ZeroMQ приблизно 10% затримки перевищує 100 мс, іноді досягаючи до 1 с.

Підсумовуючи, поведінка затримки та, отже, вибір систем pub / sub сильно залежить від різних розмірів повідомлень, що використовуються у кожному конкретному випадку використання. У той час як для невеликих повідомлень, MQTT і ZeroMQ виявляються меншими затримками, зі збільшенням розміру повідомлення розподіл затримки зближується. Додаткова затримка, введена за допомогою більш складних методів фільтрації, таких як AMQP, стає неістотною. Крім того, затримка, введена XML-кодуванням інформації заголовка в XMPP, не робить загальний час обробки значно вищим, ніж для інших протоколів у сценаріях, коли надсилаються великі повідомлення.

Висновки. У цій роботі ми провели аналіз вимог до pub / sub на хмарних платформах IoT, проаналізували основні особливості існуючих відкритих рішень та представили кількісну оцінку представників кожного протоколу за реального навантаження. Це демонструє, що, хоча жоден протокол не пропонує всіх функцій, бажаних у налаштуваннях Інтернету речей, між протоколами існують суттєві відмінності: Хоча XMPP є розширюваним відкритим протоколом, XMPP не може досягти продуктивності, що спостерігається з іншими проміжними програмами pub / sub в налаштуваннях IoT. Крім того, на стороні сервера кожен рядок данихповинна бути проаналізований для прийняття рішення про маршрутизацію. Це, мабуть, одна з причин, чому XMPP працює набагато гірше, ніж інші протоколи щодо пропускної здатності та затримки. AMQP - це повнофункціональне проміжне програмне забезпечення, орієнтоване на повідомлення, яке пропонує весь функіонал, необхідний для створення системи обміну повідомленнями з підтримкою IoT. Ефективність з точки зору пропускної здатності повідомлень та середньої затримки відстає від ZeroMQ та MQTT для навантажень з великою кількістю невеликих повідомлень. MQTT спеціально розроблений для транспортування даних, схожих на датчики, і фактично став стандартним у цій галузі. Брокер має достатньо високу пропускну здатність і низьку затримку. Найважливішою частиною, якої не вистачає в MQTT, є можливість безпосереднього контакту з підключеними клієнтами, оскільки MQTT - це суто протокол pub / sub. Наші вимірювання продуктивності свідчать про те, що ZeroMQ може досягти дуже високої пропускної здатності, зберігаючи низьку затримку, переважно незалежно від навантаження. Також привабливим є той факт, що брокер не є обов'язковим і можлива децентралізована система. Однак ZeroMQ не є повнофункціональною реалізацією брокера і менш виразною, ніж інші протоколи, оскільки підтримується лише відповідність префіксів. Отже, для розгортання ZeroMQ в налаштуваннях Інтернету речей можуть знадобитися та можуть бути додані ключові функції.

Тому наш висновок подвійний: ми рекомендуємо ZeroMQ там, де потрібне спеціально розроблене рішення, а зусилля щодо впровадження відсутніх функцій є прийнятними. Для налаштувань, де необхідне легкодоступне рішення, мирекомендуйте MQTT або AMQP, залежно від очікуваних розмірів повідомлень, оскільки існує кілька програмних рішень з відкритим кодом, які можна використовувати нестандартно.

References.

1. Amazon: Elastic Compute Cloud (EC2). URL http://aws. amazon.com/ec2
2. AMQP Working Group: Advanced message queuing protocol (2010). version 0-9-1
3. Apache Software Foundation: ActiveMQ. URL http:// activemq.apache.org/
4. Apache Software Foundation: Apollo. URL http://activemq. apache.org/apollo/
5. Apache Software Foundation: Apollo. URL http://activemq. apache.org/apollo/
6. Car2go: Berlin. https://www.car2go.com/de/berlin/
7. Carzaniga,A.,Rosenblum,D.S.,Wolf,A.L.:Achievingscalability and expressiveness in an internet-scale event notification service. In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '00), ст. 219–227. ACM, New York, NY, USA (2000). DOI 10.1145/343477.343622
8. Cha,M.,Rodriguez,P.,Moon,S.,Crowcroft,J.:Onnext-generation telco-managed p2p tv architectures. Джерело: Proceedings of the 7th International Conference on Peer-to-peer Systems (IPTPS'08), ст. 5–5. USENIX Association (2008)
9. Chui,M.,Lo¨ffler,M.,Roberts,R.:Theinternetofthings.McKinsey Quarterly 2, 1–9 (2010)
10. Curry, E.: Message-oriented middleware. In: Q.H. Mahmoud (ed.) Middleware for Communications, глава. 1, ст. 1–28. John Wiley & Sons (2005)

11. Eclipse Foundation: Paho. URL https://eclipse.org/paho/
12. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys (CSUR) 35(2), 114–131 (2003)
13. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems 29(7), 1645–1660 (2013)
14. Handziski, V., Ko ̈pke, A., Willig, A., Wolisz, A.: Twist: A scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In: Proc. of the 2nd Int. Workshop on Multi-hop Ad Hoc Networks: From Theory to Reality (REALMAN '06), ст. 63–70. Florence, Italy (2006)
15. Hintjens,P.:ZeroMQ:MessagingforManyApplications.O'Reilly (2013)
16. Hunkeler, U., Truong, H.L., Stanford-Clark, A.: MQTT-S – A publish/subscribe protocol for Wireless Sensor Networks. Джерело: 3rd Int. Conf. on Communication Systems Software and Middleware
17. Ignite Realtime: Openfire Server. URL http://www.igniterealtime.org/projects/openfire/
18. Locke,D.:MQTelemetryTransport(MQTT)V3.1ProtocolSpeci- fication. IBM developerWorks Technical Library (2010)
19. Menzel, T., Karowski, N., Happ, D., Handziski, V., Wolisz, A.: Social sensor cloud: An architecture meeting cloud-centric iot platform requirements (2014). 9th KuVS NGSDP Expert Talk on Next Generation Service Delivery Platforms
20. Millard, P., Saint-Andre, P., Meijer, R.: XEP-0060: Publish- Subscribe (2010). URL http://www.xmpp.org/extensions/ xep-0060.html. Version: 1.13
21. Moteiv Co.: Tmote sky datasheet. URL http: //www.crew- project.eu/sites/default/files/tmote- sky- datasheet.pdf
22. Pivotal Software: RabbitMQ. URL https://www.rabbitmq. com/
23. ProcessOne: ejabberd XMPP Server. URL https://www.process- one.net/en/ejabberd/
24. Rege,M.R.,Handziski,V.,Wolisz,A.:CrowdMeter:anemulation platform for performance evaluation of crowd-sensing applications. Джерело: Proc. of the 2013 ACM conf. on Pervasive and ubiquitous com- puting adjunct publication, ст. 1111–1122. Zu ̈rich, Switzerland (2013)
25. Saint-Andre, P.: Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard) (2011). URL http://www.ietf.org/rfc/rfc6120.txt Tran, P., Greenfield, P., Gorton, I.: Behavior and Performance of Message