BMP: Toward a Broker-less and Microservice Platform for Internet of Thing

Lam Nguyen Tran Thanh[1], Khoi Le Quoc[2], The Anh Nguyen[3], Huong Hoang Luong[4], Hong Khanh Vo[5], Tuan Dao Anh[6], Hy Nguyen Vuong Khang[7], Khoi Nguyen Huynh Tuan[8], Hieu Le Van[9], Nghia Huynh Huu[10], Khoa Tran Dang[11], Khiem Huynh Gia[12]
VNPT Information Technology Company, Ho Chi Minh city, Vietnam[1]
FPT University, Can Tho City, Viet Nam[2,3,4,5,6,7,8,9,10,11,12,13]

Abstract— **The Internet of Things (IoT), currently, is one of the most interesting technology trends. IoT is the foundation and driving force for the development of other scientific fields based on its ability to connect things and the huge amount of data it collects. The IoT Platform is considered the backbone of every IoT architecture that not only allows the transfer of data between user and device but also the feed of high-level applications such as big data or deep learning. As a result, the optimal design of the IoT Platform is a very important issue, which should be carefully considered in many aspects. Although the IoT is applied in multiple domains, there are three indispensable features including (a) data collections, (b) devices and users management, and (c) remote device control. These functions usually come with some requirements, for example, security, high-speed transmission, low energy consumption, reliable data exchange, and scalable systems. In this paper, we propose the IoT Platform,called BMP (Broker-less and Microservice Platform) designed according to microservice and broker-less architecture combined with gRPC protocol to meet the requirements of the three features mentioned above. Our IoT Platform addresses five issues: (1) address the limited processing capacity of devices, (2) reduce energy consumption, (3) speed up transmission rate and enhance the accuracy of the data exchange, (4) improve security mechanisms, and (5) improve the scalability of the system. Moreover, we describe the evaluation to prove the effectiveness of the BMP (i.e., proof-of-concept) in three scenarios. Finally, a source code of the BMP is publicized on the GitHub repository to engage further reproducibility and improvement.**

Keywords—Internet of Things (IoT); gRPC; Single Sign-On; Broker-Less; Kafka; Microservice; Role-based Access Control (RBAC)

I. Introduction

The application fields of the Internet of Thing (IoT), currently, are increasingly diverse including smart cities, healthcare, supply chains, industry, agriculture. According to [1], there will be approximately 75.44 billion IoT connected devices in 2025. Iot Platform is an intermediary system that acts as an "adhesive layer" to connect devices with users. There are many architectures outlined to optimize the IoT system, including 5-layer architecture in order from low to high: Things, Connect, Collect, Learn and Do introduced in book [2]. This architecture facilitates it easy to separate each specific group of roles required of an IoT Platform. The fields of application of the current IoT are varied, it is challenging to use a specific architecture suitable for every practical application. However, we generally draw features that are indispensable for an IoT system, namely, i) data collection; ii) device and user management; and iii) remote device control. These three features are the Things, Connect and Collect layers, respectively [2].

The **Things layer** are the group of physical devices that directly collect data or perform an action based on a control command from the user. The device is usually limited in power, processing and bandwidth [3]. So, the most important requirement of the **Things layer** is balanced between processing capabilities and power consumption of devices (1).

The **Connect layer** is responsible for transmitting data. So, this layer is determined by transmission protocols which are suitable for the hardware and network processing capabilities of the devices in the **Things layer**. So, the most important requirement of the **Connect layer** is to ensure fast transmission speed and reliable transmission (2).

The **Collect layer** is responsible for gathering data from devices and users. The information can be very sensitive since it relates to user privacy, especially medical or financial IoT applications. So, the most important requirement of the **Collect layer** is a security mechanism (3).

In addition, the system scalability and resilience should be concerned as important factors in IoT platform design (4).

There are five popular IoT protocols namely Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP), Extensible Messaging and Presence Protocol (XMPP), Advanced Message Queuing Protocol (AMQP), and Message Queuing Telemetry Protocol (MQTT) [4].

Regarding to limited processing ability and power consumption (1): There are many studies comparing the advantages and disadvantages of the above-mentioned protocols [1], [5], but for the communication in constrained networks bandwidth, MQTT and CoAP are proposed to be used [6]. Furthermore, regarding energy consumption, MQTT is lower than CoAP [7], so MQTT is the most favorite protocol used by developers [8].

Regarding to transmission rate (2): MQTT is twice as fast as CoAP [9]. MQTT has three levels of Quality-of-Service (QoS) from 0 to 2. Selecting these QoS levels is a trade-off between the reliability of packet transmission (rate of loss on the link), transmission rate, and energy consumption. The transmission rate of QoS-0 is the fastest, but this has the lowest reliability [10]; whereas, the opposite is the QoS-2. On the one hand, the energy consumption of the QoS-0 level is only about 50% of the QoS-2 level [11].

Regarding to security mechanism (3): MQTT has many limitations [10]. Scenarios that attack the Integrity, Availability, and Authentication and Authorization mechanisms of MQTT are outlined in [12].

Regarding to system scalability and resilience (4): MQTT protocol is used in popular IoT frameworks including well-known companies (e.g. IBM, Amazon and Microsoft) [13]. The MQTT protocol uses a pub/sub architecture [14], with the MQTT broker at the center. The MQTT subscriber (client), connects to the broker and sends messages to topics. Brokers rely on topic to route the packet, meaning that subscribers who subscribe a topic will receive all messages sent to that topic. This easily leads to a single point failure [15] error at the central broker location. In addition, MQTT is created for transmission purposes. Therefore, MQTT broker does not provide message storage as well as guarantee the order of messages when it reaches the receiver [16]. This is also an important issue to be considered as the weakness of the system using MQTT.

Addressing the MQTT issues, the aim of this paper is to propose a new IoT platform, called BMP, built on broker-less and microservice architecture. We used gRPC protocol to build the broker-less architecture at the **Collect layer** and **Connect layer** for gateway devices. This architecture provides a peer-to-peer communication between sender and receiver in BMP instead of using a central broker to coordinate topic-based messages. Collected messages will be sent to the message queue (Kafka) which provides a caching message mechanism to reduce lost messages when system error occurs. Furthermore, the microservice architecture ensures fault tolerance, scaling horizontally, availability, and carrying capacity of the BMP [17]. Moreover, we define a management model to manage the components in the BMP including users, devices, and communication channels. The management model is built on the RBAC (role-based access control) model combined with the single sign-on and the user organization to achieve authentication and authorization. To engage further reproducibility or improvement in this topic, finally, we share the completely code solution which is publicized on the our Github.

The rest of the paper is organized as follows. In Section 2 we provide knowledge about the technology used in the paper. Section 3 discusses related work. We introduce our IoT Platform and describe the proof-of-concept in Sections 4 and 5, respectively. Section 6, we discuss our test results. Finally, we summarize the paper and provide the potential directions for future work.

II. Background

A. gRPC

The gRPC[2] is an open-source framework developed by Google for implementing RPC (Remote Procedure Calls) API via HTTP/2. The gRPC provides a new reality for RPC by making it interoperable, modern, and efficient using technologies such as Protocol buffers and HTTP/2. HTTP/1.1 protocol was born in 1997. With each request sent from the client to the server, a TCP connection is created. Connection processing must go through a three-way handshake. This takes a lot of time. In addition, the headers in each request are plain text with lots of data fields and are not compressed and the headers usually have the same data. Therefore, the header occupies a significant size and is duplicated many times in the requests, leading to consuming a lot of bandwidth. HTTP/2 protocol was born in 2015 created by Google to overcome the above problems. HTTP/2 protocol supports multiplexing so it is possible to send multiple requests in parallel on an established TCP connection. This reduces bandwidth suitable for devices with limited hardware. Protocol buffers are a popular technology for data structures developed and used in communication between google services. The developer will define the data structure in the *.proto* file. The Protoc compiler then compiles the .proto file into any language it supports. At runtime, the data is compressed and normalized to binary. The gRPC is fully compatible with embedded devices [18]. This protocol provides four communication types including unary, server-streaming, client-streaming, and bidirectional streaming [19]. In Unary, the client sends a request, then the server sends back a response. In server-streaming, the lient sends a request to the server, then the server sends back multiple responses on the same TCP connection. The order of messages for each stream is guaranteed to be the same between client and server. In client-streaming, the client sends multiple requests to the server, then the server sends back only one response to the client on the same TCP connection.In bidirectional-streaming, the client sends multiple requests to the server, then the server sends back multiple responses on the same TCP connection without waiting for response time.

[2]https://grpc.io/

B. Kafka Message Queue

Kafka[3] is a distributed messaging system. It can transfer a vast number of messages in real-time. When the receiver has not received the message, this message is still stored on the message queue and the disk. This feature allows reducing lost messages when a system error occurs. The structure of Kafka includes the following main components [20]: producer, topic, partitions, consumer, broker, and zookeeper. A producer can be any application that publishes messages to a topic.A topic is a category or feed name where the record is published. The topics are divided into different segments, which are called partitions.A consumer can be any application that subscribes to a topic and consumes messages. Kafka cluster is a set of servers, each of which is called a broker. Zookeeper used to manage and arrange brokers. Kafka-Pixy[4] is a dual API (gRPC and REST) proxy for Kafka with automatic consumer group control.In this paper, we use Kafka-pixy open source to communicate between Kafka message queue and other microservice in BMP by gRPC protocol. Detailed source code can be found at the link[5][6].

C. Oauth and Single Sign-On

Oauth version 2.0[7] stands for Open with Authentication or Authorization. OAuth was born to solve the above problem and beyond, this is an authentication method that helps applications

[3]https://kafka.apache.org/
[4]https://github.com/mailgun/kafka-pixy
[5]https://github.com/thanhlam2110/Kafka-gRPC-Producer
[6]https://github.com/thanhlam2110/Kafka-gRPC-Consumer
[7]https://oauth.net/2/

to share resources without sharing username and password information.

Single Sign-On (SSO) is a mechanism that allows users to access many websites and applications without simply logging in once. Once identified in an application A, it will also be identified in application B without repeating the login operation. This feature is suitable for IoT applications because a user can be a customer of many different IoT service providers [21]. In this paper, we use open source CAS Apereo which is modified by us to implement SSO service and Oauth protocol. Detailed source code can be found at the link[8]

D. Microservice Architecture

Microservices are an approach to developing an application using a set of small services, a service that will run its own process independently, usually an HTTP resource API. This modern architecture allows for the creation of larger, more complex, and scalable applications [22]. Applications using the microservixe architecture are very easy to maintain because the modules are completely separate from each other. It also provides high reliability since a service failure does not affect other services at all [23]. The microservice architecture is also easily extensible. The only weakness of the microservice architecture is the complex deployment process [24].

III. Related Work

A. Broker-less Architecture in IoT

Lu et al. in [25] described a solution to collect the data while meeting the confidentiality requirements via fog computing-enhanced IoT devices. They used a one-way hash string to authenticate IoT devices, then applied the Chinese Remainder Theorem to aggregate data generated by different IoT devices. They also took advantage of Uniform Paired Encryption to provide data security. Some interests of the proposed solution include i) fault tolerance, ii) solving heterogeneity problems in partial mode. Moreover, Lam et al. [26], [27] applied the broker-less architecture in their IoT platforms. However, their resolution applied a common privacy rule, differential privacy, to all IoT devices causing unintentional amounts of sensitive data to be accepted. Applying the access control model can solve this drawback [28], [29], where only authorized objects can accessed data [30], [31].

Furthermore, in the industrial environments, some research directions have applied IoT based on decentralized architecture to meet the requirements for specific system. For example, in [32], the authors focused on the varying IoT protocols in a distributed control platform by concentrating on the times and limits of the transit for different parameter options. Standardized system configuration exploits the embedded Raspberry Pi development board with open source protocol implementation for efficacy measurement, including scalability effectiveness with various data consumers in an automated network chemical. Leveraging the on-board computing resources of distributed embedded systems in the edge computing paradigm for industrial automation was argued in [33].

[8]https://github.com/thanhlam2110/cas-overlay-template

B. IoT Platform based on Microservice

The microservices architecture is introduced to address the traditional monolithic issues. For example, microservices architecture was introduced to fill this gap [34]. This article focuses on summarizing prior work that used the microservices benefits in designing their architecture rather than detailing how it works and the direction of development.

For the framework, Amazon offers AWS IoT Greengrass[1] as a solution to migrate analytics capabilities directly to edge devices (e.g., smartphones). Microsoft Azure provides similar functionality to Azure Stream Analytics on IoT Edge[2], enabling users to use near-real-time analytic functionalities by using Azure Stream Analytics on IoT-applied devices. However, these frameworks neither provide any functionality to move Lambda calculation to and fro between cloud edge devices nor mix with exterior stream processing engines.

For the application, Lam et al. [35], [36], [37] applied microservice architecture to define the healthcare and IoT-Platform applications. In addition, Maia and associates [38] presented IRRISENS, which is designed based on microservices architectures used in agricultural environments to sense soil, crop, and atmospheric parameters.

IV. BMP Architecture

The BMP is designed according to microservice architecture, and broker-less architecture including **Thing layer**, **User layer**, **Edge layer** and **Cloud layer**, as shown in Fig. 1:

The **Thing layer** is the group of sensor or physical devices.These devices are responsible for measuring environmental parameters such as humidity, temperature or patient health parameters, etc. depending on the IoT application.

In each of these devices, we implement two services i) collection data service (client) and ii) control service (client). The former is responsible for streaming data collected from the environment according to a predetermined collection data server (server) in the IoT Platform. This data streaming is only performed when the client is authenticated and checked the things' role by the Single Sign-On service. The latter receives control commands from the control service (server) through the message queue system. The **Edge layer** including gateway devices. The Edge layer includes gateway devices. In each gateway, we implement two services: *collect data service client-side (CDC)* and *control service client-side (CSC)*.The CDC receives data from devices in the **Thing layer**, then sends it to the **Cloud layer**. The CSC receives control commands from **User layer** throughout **Cloud layer**.

The **User layer** is the groups of users, with different roles, registered to use IoT services. Users can control and monitor the device state by using the control service and the collect data service, in turn. This is performed after passing at the authentication phase and verifying at the authorization phase by the Single Sign-On service and RBAC model, respectively. Besides, users can manage device information (for instance, create, delete, disable, active, or manage) their child users

[1]https://docs.osgi.org/specification/osgi.cmpn/7.0.0/service.event.html
[2]https://maven.apache.org/

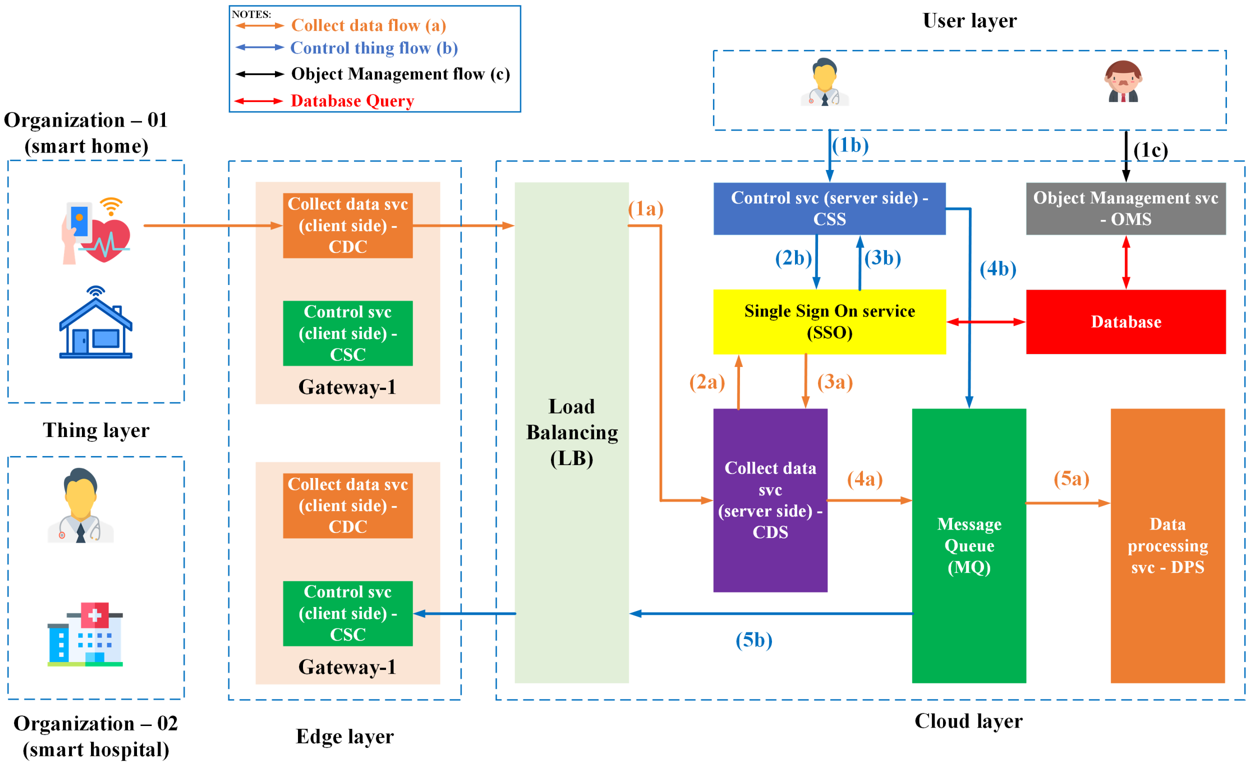

NOTES:
Collect data flow (a)
Control thing flow (b)
Object Management flow (c)
Database Query
User layer
Organization – 01 (smart home)
Thing layer
Organization – 02 (smart hospital)
Collect data svc (client side) - CDC
Control svc (client side) - CSC
Gateway-1
Edge layer
Load Balancing (LB)
(1a)
(1b)
(1c)
Control svc (server side) - CSS
Object Management svc - OMS
(2b)
(3b)
(4b)
Single Sign On service (SSO)
Database
(2a)
(3a)
Collect data svc (server side) - CDS
(4a)
Message Queue (MQ)
(5a)
Data processing svc - DPS
(5b)
Cloud layer

Fig. 1. BMP Platform Architecture.

(register, disable, active) through the Object Management service. Furthermore, users can create management information about devices and communication channels. This information is metadata and stored in a database helps users easily manage their devices and be aware clearly of where they send your messages. For example, user A can create/delete/update his device and communication channel information. Moreover, users are organized hierarchically according to a tree model so high-level users (aka *parent-user*) can manage low-level users (aka *child-user*). For example, user A (high-level) can enable or disable user B's status (low-level). If user B is disabled, user B can not interact with BMP. This feature is similar to the company model with user A as manager role and user B as staff role. In addition to the user and things layer services, the IoT platform has a Load balancing service to balance the load for the whole system. Besides, the Message Queue service is responsible for routing control packets from user to things and data collected from the things layer. Message Queue also stores messages going through the IoT Platform, ensuring that it could receive messages after recovery even the service fails. Data processing service analyzes data in-depth level.

The **Cloud layer** is the main processing part of BMP including *load balancing (LB)*, *collect data service server-side (CDS)*, *control service server-side (CSS)*, *single sign-on service (SSO)*, *message queue (MQ)*, *object management service (OMS)*, *data processing service (DPS)*, and *database (DB)*. The *LB* is responsible for distributing incoming traffic to ensure the **Cloud layer** is not overloaded. The *CDS* receives messages from CDC. These messages include Oauth access token and data collected from the **Thing layer**. The access token is verified by the *SSO*. If the access token is valid the data will be sent to the *MQ*, otherwise, the data will be discarded. The *CSS* receives messages from users in the **User layer**. These messages include an Oauth access token and control command. Similarly, the access token is verified by the *SSO*. If the access token is valid the control command will be sent to the *MQ*, otherwise, the the control command will be discarded. Thanks to gRPC's multiplexing fearture and client-stream method, we only need to establish an http/2 connection between CSC and CSS or CDC and CDS to send the access token and a bunch of data at the same time. This helps to reduce system bandwidth as well as power consumption on the gateway. The *MQ* is responsible for distributing data to the *DPS* for deeper analysis or the *DB* for storage and control commands to the CSC in the **Edge layer**.

V. Implementation

In this paper, we have implemented the services outlined in the proposal section including: single-sign on service (SSO)[9], collect data service (CDC and CDS)[10], and object management service (OMS)[11] . The model for the development and interaction between the symbolic services in Fig. 1 is as follows: (1a-5a) the collect data flow, (1b-5b) the control thing flow, and (1c) the object management thread.

The **collect data flow** details are shown in Fig. 2. The **Things layer** collects data from sensors and periodically sends them back to the gateway at the **Edge layer** (sending data periodically to help save energy). The CDC on gateway creates

[9]https://github.com/thanhlam2110/iccs-sso-service
[10]https://github.com/thanhlam2110/iccs-collect-data-svc
[11]https://github.com/thanhlam2110/iccs-object-management-svc

a single gRPC connection to both send access token, thing-id and streaming data to CDS - *process 1a* in Fig. 1. This is the multiplexing feature of the gRPC protocol, which was introduced in section 2.1. So the data sent from CDC to CDS is the combination of *access token*, *thing-id*, and *collected data from sensor*. Next, the CDS sends the *access token* and *thing-id* to the SSO service (process 2a) to verify access token and check role, the test result is returned (process 3a). If the token is authenticated and the role is verified, CDS will stream the bulk of *collected data* to MQ (process 4a), whereas the CDS will drop all data and return the error to CDC. MQ will distribute data to DPS for in-depth analysis (process 5a).

The **control thing flow** details are shown in Fig. 3. The user creates a single gRPC connection to both send the access token, things-id and control command to the CSS (process 1b). The CSS sends the *access token* and *thing-id* to the SSO service to verify access token and check role. Role-based access control integrated on SSO service uses user-id (get from check access token process) and thing-id to verify the device which the user wants to control belongs to the user or not (process 2b). Next, the check result is returned to the CSS (process 3b), if it is valid, the control command will be sent to the MQ (process 4b). The CSC on gateway subscribes to the channel (aka Kafka topic) on the MQ which is created by the user to be dedicated to receive control commands. When the MQ receives the control command, the CSC also receives it. (process 5b).

The **object management flow** provides APIs for users to interact with the database. The Single Sign On service will also use the data stored here to authenticate and verify the roles of users and things. We implement RBAC combined with Single Sign-On to provide role-based authentication and authorization for Things and Users. However, the concepts of users' roles and things' roles are somewhat different.

We define the users' role including permissions that affect their devices and child users.

About users' role on their child user, a parent user can create/delete/enable/ disable child user. For example, user A can create management information for user B who is the visitor to user A's house. In this case, user A is the *parent-user* and user B is the *child-user* of A. With management information, user B is allowed to control some devices in user A smart home. When user B leaves, user A deletes user B's information, which means user B can not interact with user A's smart home. We implement the user management by the model tree with the *child-user* has an attribute user_parent_id field equal to the *parent-user*'s username. This organization user model allows BMP to be flexibly applied to the business of implementing hierarchical user management.

About users' role on their devices, the user can create/delete management information, enable/disable, or assign/unassign (to another user) their devices. For example, user A can choose some devices that user B can not control when user B visits user A's house by unassigning these devices to user B. To do this, we provide APIs that allow user-id and thing-id mapping. Then, SSO will check this information when a user wants to control devices. In addition, OMS also provides APIs for users to create/delete communication channels management information. This channel information allows users to assign their devices to send and receive messages on it. This function helps users be aware clearly where they share data.

VI. Evaluation

The BMP design is based on microservice and broker-less architecture. To evaluate BMP, we deploy the **Cloud layer** on the Amazon EC2[10] platform with each service equivalent to a virtual machine with 1GB RAM and 1 vCPU configuration. For the gateway, we deploy CDC and CSC on the Raspberry Pi 3 model B[11] module with CPU quad-core, 1.2 GHz and 1GB RAM.

In this paper, we perform three test scenarios to evaluate BMP with aspects transmission rate, power consumption, and security mechanism.

Scenario 1: We measure the Round Trip Time (RTT) from when CDC streams *data* until it is received by the message queue. In addition, we also look at the error rate (number of messages lost per total number of messages). We assume the *data* in this test case is *data* = *access token* + *string*, with *string* = "hello number" + for (i=1; i¡=number_of_messages;i++). Measured results are shown in Table I.

The test results are quite good as we only deploy the evaluation on low-configuration servers. All messages are received in full and in order. By using gRPC which allows sending multiple messages on one connection, we achieve very fast transfer speed while still satisfy the whole process of checking and validating before streaming data as described in Section 5.

Scenario 2: We measure CPU and RAM usage to evaluate power consumption when streaming *data*. We use the htop[12] tool, a tool that allows real-time monitoring of system processes. Similarly test scenario 1, We assume the *data* in this test case is *data* = *access token* + *string*, with *string* = "hello number" + for (i=1; i¡=number_of_messages;i++).

As a result in Table II, at the no-load and 10000 messages levels, we recorded the value as 0, which means that the resource usage of the services is shallow (no-load) or takes place in a concise time (10000 messages); therefore, the measuring tool hardly records any change in resource usage. For the high load case (50000-100000 messages), the result shows that the resource consumption is very low. Furthermore, in practice, it is not always the thing that streams a large amount of data like in this scenario. This shows that our adoption of gRPC to the IoT Platform gives outstanding results, suitable for devices with low hardware.

Scenario 3: We capture the message that is transmitted between sender and receiver to analyze security risk. We use Wireshark[3] to capture the transmitted message. Wireshark is network packet analyzer software. Its job is to capture all network packets and then display the data of that packet in the most detail. We set up the test scenario to compare the security mechanism between MQTT and gRPC protocol. We use Wireshark to capture messages that are transmitted between the

[10]https://aws.amazon.com/

[11]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[12]https://vitux.com/how-to-use-htop-to-monitor-system-processes-in-ubuntu/

[3]https://www.wireshark.org/

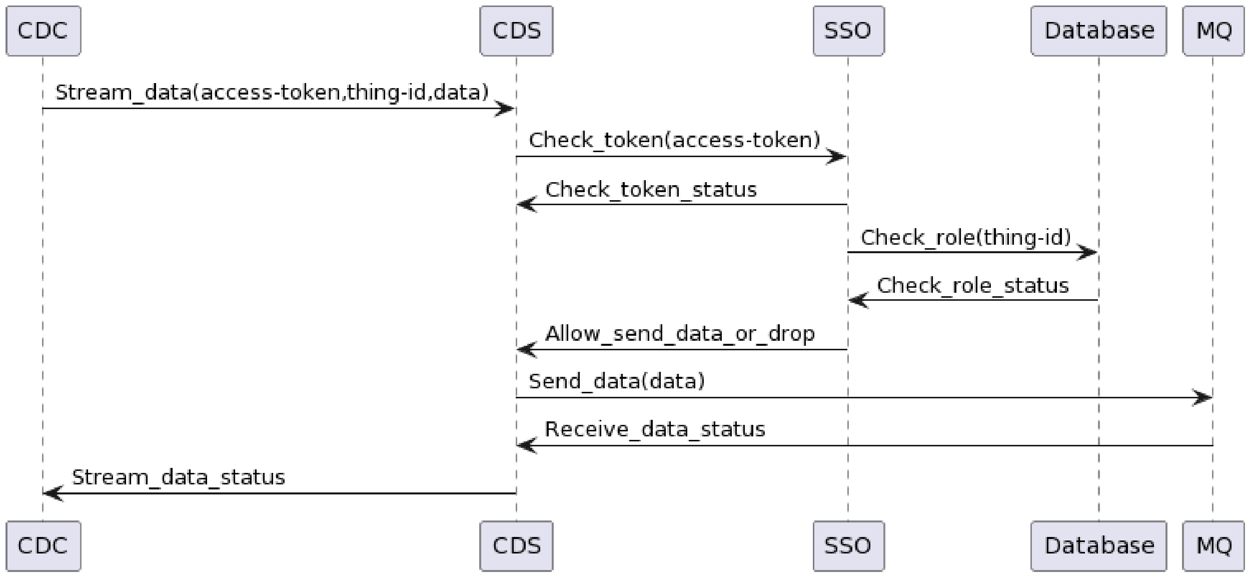


Fig. 2. The Collection Data Flow.

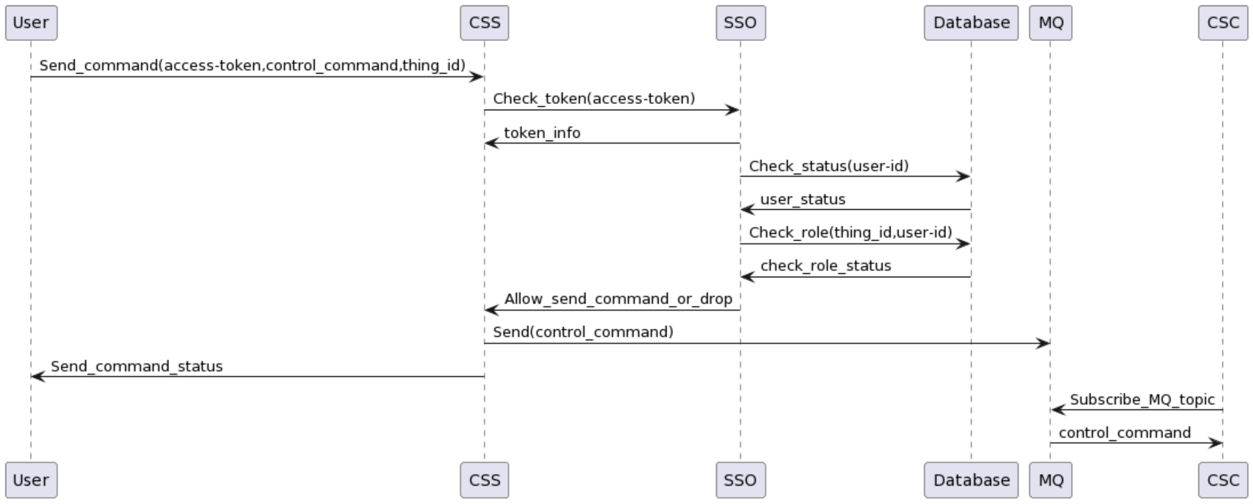


Fig. 3. The Control Flow.

TABLE I. THE RESULT OF THE FIRST SCENARIO

Number of messages	100	1000	5000	10000	50000
RTT (s)	1.5	4.07	14.79	27.05	138.02
Error (%)	0	0	0	0	0

sender and receiver and then analyze these messages. The test model is shown in Fig. 4.

The analyzed message result of MQTT and gRPC protocol is shown in Fig. 5 and Fig. 6, respectively.

According to Fig. 5 and Fig. 6, while MQTT protocol easily gets topic information as well as message content, in this case MQTT topic is *"thanh-lam"* and message is *"Hello Thanh Lam"*, gRPC protocol provides packet encryption. This result proves that the gRPC protocol has a better security mechanism than the MQTT protocol.

VII. CONCLUSION

In this paper, we propose the IoT Platform, called BMP, using the broker-less and microservice architecture. The broker-less architecture helps our IoT Platform reduce a single point of failure in comparison with brokering architecture. Furthermore, microservice architecture helps BMP easily scale out, reducing downtime when errors occur. The combination of broker-less and microservice architecture also helps 3rd parties easily integrate with BMP. Third-party developers only need to call the API and don't make major changes to the

TABLE II. THE RESULT OF THE SECOND SCENARIO

Number of messages		No load	10000	50000	100000
Raspberry Pi	CPU	0	0	0.3%	0.7%
	RAM	0	0	2.5%	2.7%
Collection data service (server)- CDS	CPU	0	0	8.2%	10.5%
	RAM	1.7%	1.7%	1.7%	1.7%

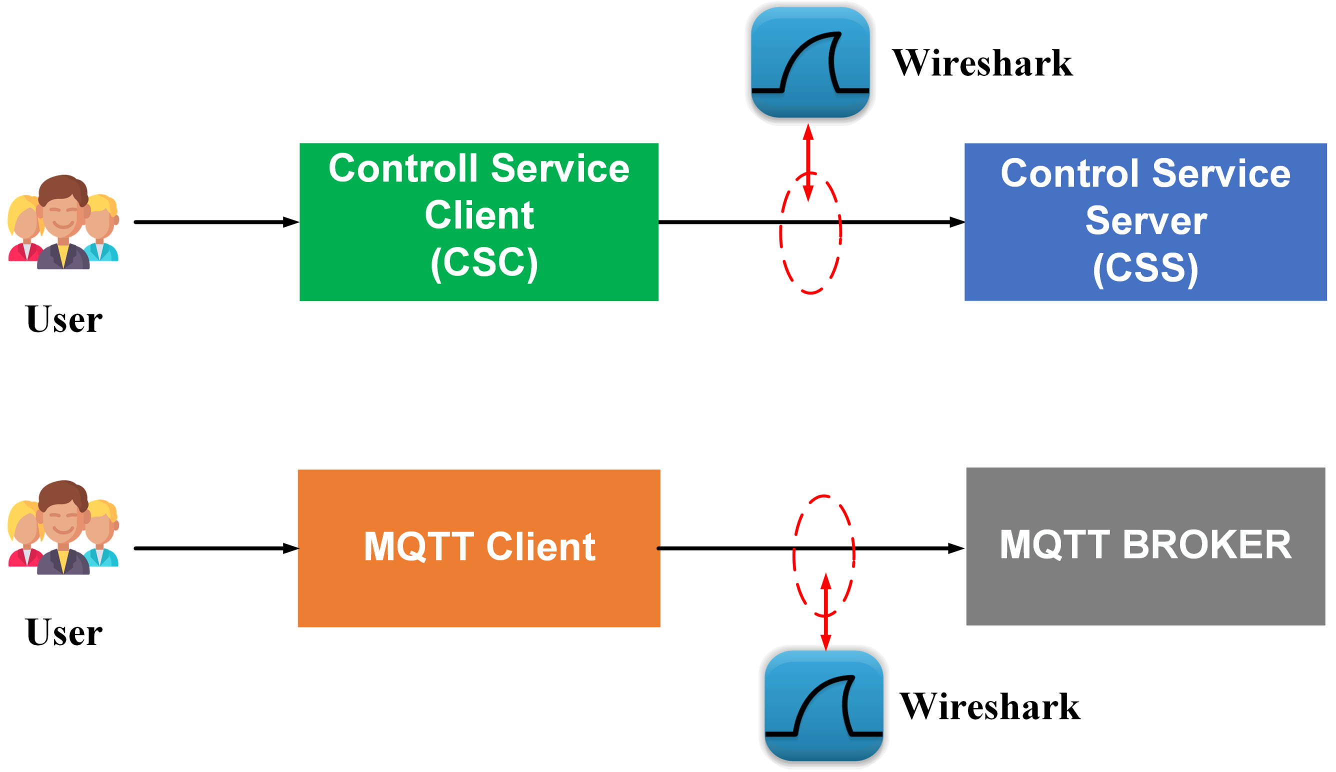
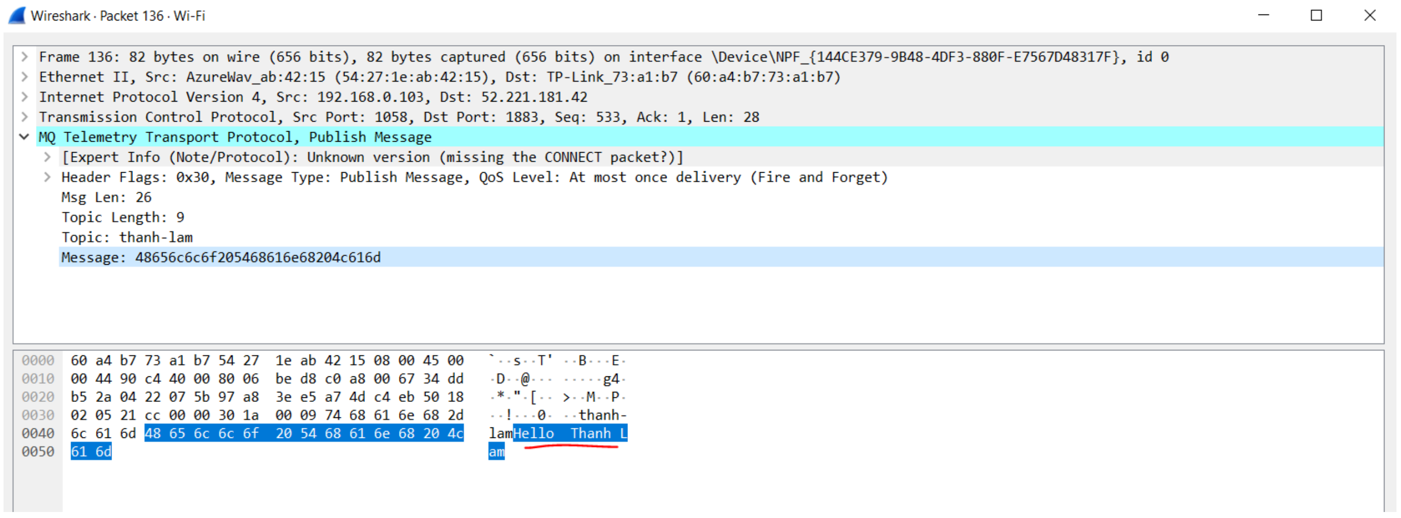


Fig. 4. Capture Transmission Message using Wireshark.

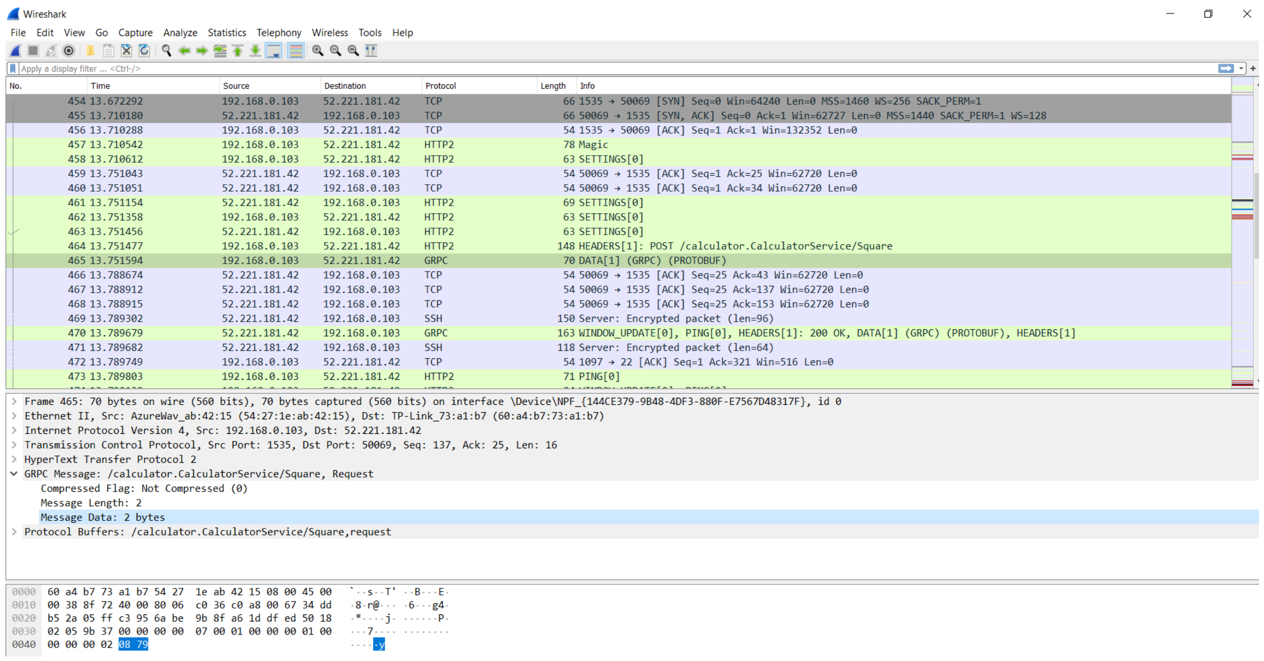


```
Wireshark · Packet 136 · Wi-Fi

> Frame 136: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface \Device\NPF_{144CE379-9B48-4DF3-880F-E7567D48317F}, id 0
> Ethernet II, Src: AzureWav_ab:42:15 (54:27:1e:ab:42:15), Dst: TP-Link_73:a1:b7 (60:a4:b7:73:a1:b7)
> Internet Protocol Version 4, Src: 192.168.0.103, Dst: 52.221.181.42
> Transmission Control Protocol, Src Port: 1058, Dst Port: 1883, Seq: 533, Ack: 1, Len: 28
v MQ Telemetry Transport Protocol, Publish Message
  > [Expert Info (Note/Protocol): Unknown version (missing the CONNECT packet?)]
  > Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    Msg Len: 26
    Topic Length: 9
    Topic: thanh-lam
    Message: 48656c6c6f205468616e68204c616d

0000  60 a4 b7 73 a1 b7 54 27  1e ab 42 15 08 00 45 00   `··s··T' ··B···E·
0010  00 44 90 c4 40 00 80 06  be d8 c0 a8 00 67 34 dd   ·D··@···  ·····g4·
0020  b5 2a 04 22 07 5b 97 a8  3e e5 a7 4d c4 eb 50 18   ·*·"·[··  >··M··P·
0030  02 05 21 cc 00 00 30 1a  00 09 74 68 61 6e 68 2d   ··!···0·  ··thanh-
0040  6c 61 6d 48 65 6c 6c 6f  20 54 68 61 6e 68 20 4c   lamHello  Thanh L
0050  61 6d                                              am
```

Fig. 5. MQTT Protocol Message without Encrypted Format.


Wireshark
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time          Source          Destination     Protocol  Length Info
  454 13.672292     192.168.0.103   52.221.181.42   TCP         66 1535 → 50069 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
  455 13.710180     52.221.181.42   192.168.0.103   TCP         66 50069 → 1535 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=1440 SACK_PERM=1 WS=128
  456 13.710288     192.168.0.103   52.221.181.42   TCP         54 1535 → 50069 [ACK] Seq=1 Ack=1 Win=132352 Len=0
  457 13.710542     192.168.0.103   52.221.181.42   HTTP2       78 Magic
  458 13.710612     192.168.0.103   52.221.181.42   HTTP2       63 SETTINGS[0]
  459 13.751043     52.221.181.42   192.168.0.103   TCP         54 50069 → 1535 [ACK] Seq=1 Ack=25 Win=62720 Len=0
  460 13.751051     52.221.181.42   192.168.0.103   TCP         54 50069 → 1535 [ACK] Seq=1 Ack=34 Win=62720 Len=0
  461 13.751154     52.221.181.42   192.168.0.103   HTTP2       69 SETTINGS[0]
  462 13.751358     192.168.0.103   52.221.181.42   HTTP2       63 SETTINGS[0]
  463 13.751456     52.221.181.42   192.168.0.103   HTTP2       63 SETTINGS[0]
  464 13.751477     192.168.0.103   52.221.181.42   HTTP2      148 HEADERS[1]: POST /calculator.CalculatorService/Square
  465 13.751594     192.168.0.103   52.221.181.42   GRPC        70 DATA[1] (GRPC) (PROTOBUF)
  466 13.788674     52.221.181.42   192.168.0.103   TCP         54 50069 → 1535 [ACK] Seq=25 Ack=43 Win=62720 Len=0
  467 13.788912     52.221.181.42   192.168.0.103   TCP         54 50069 → 1535 [ACK] Seq=25 Ack=137 Win=62720 Len=0
  468 13.788915     52.221.181.42   192.168.0.103   TCP         54 50069 → 1535 [ACK] Seq=25 Ack=153 Win=62720 Len=0
  469 13.789302     52.221.181.42   192.168.0.103   SSH        150 Server: Encrypted packet (len=96)
  470 13.789679     52.221.181.42   192.168.0.103   GRPC       163 WINDOW_UPDATE[0], PING[0], HEADERS[1]: 200 OK, DATA[1] (GRPC) (PROTOBUF), HEADERS[1]
  471 13.789682     52.221.181.42   192.168.0.103   SSH        118 Server: Encrypted packet (len=64)
  472 13.789749     192.168.0.103   52.221.181.42   TCP         54 1097 → 22 [ACK] Seq=1 Ack=321 Win=516 Len=0
  473 13.789803     192.168.0.103   52.221.181.42   HTTP2       71 PING[0]

> Frame 465: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{144CE379-9B48-4DF3-880F-E7567D48317F}, id 0
> Ethernet II, Src: AzureWav_ab:42:15 (54:27:1e:ab:42:15), Dst: TP-Link_73:a1:b7 (60:a4:b7:73:a1:b7)
> Internet Protocol Version 4, Src: 192.168.0.103, Dst: 52.221.181.42
> Transmission Control Protocol, Src Port: 1535, Dst Port: 50069, Seq: 137, Ack: 25, Len: 16
> HyperText Transfer Protocol 2
v GRPC Message: /calculator.CalculatorService/Square, Request
    Compressed Flag: Not Compressed (0)
    Message Length: 2
    Message Data: 2 bytes
> Protocol Buffers: /calculator.CalculatorService/Square,request

0000  60 a4 b7 73 a1 b7 54 27  1e ab 42 15 08 00 45 00   `··s··T' ··B···E·
0010  00 38 8f 72 40 00 80 06  c0 36 c0 a8 00 67 34 dd   ·8·r@··· ·6···g4·
0020  b5 2a 05 ff c3 95 6a be  9b 8f a6 1d df ed 50 18   ·*····j· ······P·
0030  02 05 9b 37 00 00 00 00  07 00 01 00 00 00 01 00   ···7···· ········
0040  00 00 00 02 08 79                                  ····
```

Fig. 6. gRPC Protocol Message with Encrypted Format.

BMP's architecture. Moreover, we chose gRPC as the main protocol for BMP because of its advantage in privacy, power consumption, speed, and reliable transmission in comparison with MQTT. In addition, the gRPC protocol works peer-to-peer which is suitable for broker-less implementation. BMP also provides authentication and authorization mechanisms for users, devices, and communication channels thanks to the combination of single sign-on (SSO) and role-based access control (RBAC). The BMP allows users to manage their devices and channels to aim at users clearly aware when they share data. The hierarchical organization of users as a tree model helps enhance flexibility. In the future, we will focus on the decentralized identity for IoT users and devices by applying blockchain.

REFERENCES

[1] M. Bansal *et al.*, "Application layer protocols for internet of healthcare things (ioht)," in *2020 Fourth International Conference on Inventive Systems and Control (ICISC)*. IEEE, 2020, pp. 369–376.

[2] T. Chou, *Precision-Principles, Practices and Solutions for the Internet of Things*. McGraw-Hill Education, 2017.

[3] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud computing*, vol. 3, no. 1, pp. 11–17, 2015.

[4] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, and A. Panya, "Authorization mechanism for mqtt-based internet of things," in *2016 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2016, pp. 290–295.

[5] S. Verma and M. A. Rastogi, "Iot application layer protocols: A survey."

[6] S. P. Jaikar and K. R. Iyer, "A survey of messaging protocols for iot systems," *International Journal of Advanced in Management, Technology and Engineering Sciences*, vol. 8, no. II, pp. 510–514, 2018.

[7] M. Martí, C. Garcia-Rubio, and C. Campo, "Performance evaluation of coap and mqtt_sn in an iot environment," in *Multidisciplinary Digital Publishing Institute Proceedings*, vol. 31, no. 1, 2019, p. 49.

[8] G. C. Hillar, *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd, 2017.

[9] B. H. Çorak, F. Y. Okay, M. Güzel, Ş. Murt, and S. Ozdemir, "Comparative analysis of iot communication protocols," in *2018 International symposium on networks, computers and communications (ISNCC)*. IEEE, 2018, pp. 1–6.

[10] S. Lee, H. Kim, D.-k. Hong, and H. Ju, "Correlation analysis of mqtt loss and delay according to qos level," in *The International Conference on Information Networking 2013 (ICOIN)*. IEEE, 2013, pp. 714–717.

[11] J. Toldinas, B. Lozinskis, E. Baranauskas, and A. Dobrovolskis, "Mqtt quality of service versus energy consumption," in *2019 23rd International Conference Electronics*. IEEE, 2019, pp. 1–4.

[12] J. J. Anthraper and J. Kotak, "Security, privacy and forensic concern of mqtt protocol," in *Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India*, 2019.

[13] F. Carranza *et al.*, "Brokering policies and execution monitors for iot middleware," in *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, 2019, pp. 49–60.

[14] D. Soni and A. Makwana, "A survey on mqtt: a protocol of internet of things (iot)," in *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*.

[15] P. Lv, L. Wang, H. Zhu, W. Deng, and L. Gu, "An iot-oriented privacy-preserving publish/subscribe model over blockchains," *IEEE Access*, vol. 7, pp. 41 309–41 314, 2019.

[16] H. C. Hwang *et al.*, "Design and implementation of a reliable message transmission system based on mqtt protocol in iot," *Wireless Personal Communications*, vol. 91, no. 4, pp. 1765–1777, 2016.

[17] M. Ali, S. Ali, and A. Jilani, "Architecture for microservice based system. a report," 2020.

[18] N. Karcher, R. Gebauer, R. Bauknecht, R. Illichmann, and O. Sander, "Versatile configuration and control framework for real time data acquisition systems," *arXiv preprint arXiv:2011.00112*, 2020.

[19] K. Indrasiri and D. Kuruppu, *gRPC: up and running: building cloud native applications with Go and Java for Docker and Kubernetes.* " O'Reilly Media, Inc.", 2020.

[20] G. Shapira, T. Palino, R. Sivaram, and K. Petty, *Kafka: the definitive guide.* " O'Reilly Media, Inc.", 2021.

[21] T. T. L. Nguyen, H. K. Vo, H. H. Luong, H. T. K. Nguyen, A. T. Dao, X. S. Ha *et al.*, "Toward a unique iot network via single sign-on protocol and message queue," in *International Conference on Computer*

Information Systems and Industrial Management. Springer, 2021, pp. 270–284.

[22] F. Tapia, M. Á. Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From monolithic systems to microservices: A comparative study of performance," *Applied Sciences*, vol. 10, no. 17, p. 5797, 2020.

[23] L. N. T. Thanh *et al.*, "Ioht-mba: An internet of healthcare things (ioht) platform based on microservice and brokerless architecture," *International Journal of Advanced Computer Science and Applications*, 2021.

[24] K. Gos and W. Zabierowski, "The comparison of microservice and monolithic architecture," in *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. IEEE, 2020, pp. 150–153.

[25] R. Lu, K. Heung, A. H. Lashkari, and A. A. Ghorbani, "A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced iot," *IEEE Access*, vol. 5, pp. 3302–3312, 2017.

[26] L. N. T. Thanh *et al.*, "Toward a unique iot network via single sign-on protocol and message queue," in *International Conference on Computer Information Systems and Industrial Management*. Springer, 2021.

[27] ——, "Toward a security iot platform with high rate transmission and low energy consumption," in *International Conference on Computational Science and its Applications*. Springer, 2021.

[28] H. X. Son, M. H. Nguyen, H. K. Vo *et al.*, "Toward an privacy protection based on access control model in hybrid cloud for healthcare systems," in *International Joint Conference: 12th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2019) and 10th International Conference on EUropean Transnational Education (ICEUTE 2019)*. Springer, 2019, pp. 77–86.

[29] H. X. Son and E. Chen, "Towards a fine-grained access control mechanism for privacy protection and policy conflict resolution," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 2, 2019.

[30] S. H. Xuan *et al.*, "Rew-xac: an approach to rewriting request for elastic abac enforcement with dynamic policies," in *2016 International Conference on Advanced Computing and Applications (ACOMP)*. IEEE, 2016, pp. 25–31.

[31] H. X. Son, T. K. Dang, and F. Massacci, "Rew-smt: a new approach for rewriting xacml request with dynamic big data security policies," in *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 2017, pp. 501–515.

[32] S. K. Panda, M. Majumder, L. Wisniewski, and J. Jasperneite, "Real-time industrial communication by using opc ua field level communication," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 1143–1146.

[33] D. L. Tran, T. Yu, and M. Riedl, "Integration of iiot communication protocols in distributed control applications," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2020, pp. 2201–2206.

[34] L. Bixio, G. Delzanno, S. Rebora, and M. Rulli, "A flexible iot stream processing architecture based on microservices," *Information*, vol. 11, no. 12, p. 565, 2020.

[35] L. N. T. Thanh *et al.*, "Uip2sop: A unique iot network applying single sign-on and message queue protocol," *IJACSA*, vol. 12, no. 6, 2021.

[36] ——, "Sip-mba: A secure iot platform with brokerless and microservice architecture," *International Journal of Advanced Computer Science and Applications*, 2021.

[37] N. T. T. Lam, H. X. Son, T. H. Le, T. A. Nguyen, H. K. Vo, H. H. Luong, T. D. Anh, K. N. H. Tuan, and H. V. K. Nguyen, "Bmdd: A novel approach for iot platform (broker-less and microservice architecture, decentralized identity, and dynamic transmission messages)," *International Journal of Advanced Computer Science and Applications*, 2022.

[38] E. Simeoni, E. Gaeta, R. I. García-Betances, D. Raggett, A. M. Medrano-Gil, D. F. Carvajal-Flores, G. Fico, M. F. Cabrera-Umpiérrez, and M. T. Arredondo Waldmeyer, "A secure and scalable smart home gateway to bridge technology fragmentation," *Sensors*, vol. 21, no. 11, p. 3587, 2021.