BMDD: A novel approach for IoT platform (Broker-less and Microservice architecture, Decentralized identity, and Dynamic transmission messages)

Lam Thanh Tran Nguyen[1], **Ha Xuan Son***[2], **Trieu Hai Le**[3], **Huong Hoang Luong**[4], **Khanh Hong Khanh Vo**[5], **Khoi Tuan Huynh Nguyen**[6], **Anh The Nguyen**[7], **Tuan Anh Dao**[8], **and Hy Khang Vuong Nguyen**[9]

[1]**Ho Chi Minh University of Technology, Ho Chi Minh city, Vietnam**
[2]**University of Insubria, Varese, Italy**
[3]**Cantho University of Technology, Vietnam**
[4,5,6,7,8,9]**FPT University, Can Tho City, Vietnam**

Corresponding author:
Lam Thanh Tran Nguyen[1]

Email address: thanhlam.bk.2110@gmail.com

ABSTRACT

Undeniably, Internet of Things (IoT) devices are gradually getting better from time to time; and IoT-based systems play a significant role in our life. The pervasiveness of the new essential service models is expanding, such as self-driving cars, smart homes, smart cities, as well as promoting the development of some traditional fields such as agriculture, healthcare, and transportation; besides, the development of IoT devices has not shown any sign of cooling down. On the one hand, several studies are coming up with many scenarios for IoT platforms, but some critical issues related to performance, speed, power consumption, availability, security, and scalability are not yet fully resolved. On the other hand, IoT devices are manufactured and developed by different organizations and individuals; hence, there is no unified standard (uniformity of IoT devices), i.e., sending and receiving messages among them and between them and the upper layer (e.g., edge devices). To address these issues, therefore, this paper proposes an IoT Platform called BMDD (Broker-less and Microservice architecture, Decentralized identity, and Dynamic transmission messages) that has a combination of two architectural models, including broker-less and microservices, with cutting-edge technologies such as decentralized identity and dynamic message transmission. The main contributions of this article are five-fold, including i) proposing broker-less and microservice for IoT platform which can reduce single failure point of brokering architecture, easy to scale out and improve failover; ii) providing decentralized authentication mechanism which is suitable for IoT devices attribute (i.e., mobility, distributed); iii) applying the Role-Based Access Control (RBAC) model for the authorization process; iv) exploiting the gRPC protocol combined with Kafka message queue enhances transmission rates, transmission reliability, and reduces power consumption in comparison with MQTT protocol; and v) developing a dynamic message transmission mechanism that helps users communicate with any device, regardless of the manufacturer because it provides very high homogeneity.

INTRODUCTION

Nowadays, science and technology have developed significantly day by day, which leads to stricter and more demanding needs from users. The current scientific landscape can be generally described as connecting the edge-cutting techniques as well as meeting the era's demands. As the Fourth Industrial Revolution spread throughout the globe, all devices/sensors were able to connect to the wireless system (e.g., Bluetooth, Wifi, Zigbee) to increase the intra-interoperability among themselves and the different layers (e.g., devices - owners). In particular, the communication among the devices connect via a 5G/6G mobile network, the Internet for long-range transmission or wifi version 6.0, and Bluetooth version 5.0

45 for short-range connection. The increasing of transmission speed requires has promoted researches in the
46 field of signal processing and wireless transmission. To cover that, all devices have internet capability,
47 a.k.a Internet of Things (IoT). The released IoTs architecture platforms/models allows the user's device
48 connect to the Internet; thereby they are able to monitor and configure access control, enabling collecting
49 and exchanging their data. This development is the main reasons for the birth of Big Data-related fields.

50 The Big Data data analysis challenge is not only discrete and meaningless combination but helps
51 providers perceive users' behavior or habits to predict customers. As data becomes a new oil, new
52 requirements for IoT platforms are set to ensure the privacy of users' data (11). After obtaining data that
53 has undergone analysis by Big Data, Machine Learning, and AI applications, devices can "learn" from
54 the data itself and the owner's feedback in specific cases; at this point, the concept of "smart devices" is
55 introduced. IoT platforms also replace humans operating in dangerous places such as high mountains,
56 deep oceans, or even outer space (58). For Big Data or AI applications to operate effectively, a massive
57 amount of data is sufficiently required to be rapidly and accurately collected, and the IoT platform is the
58 main component responsible for this problem. (89).

59 IoT application is increasingly diverse in many areas, including smart cities, healthcare, supply chains,
60 industry, agriculture, etc. The number of IoT-connected devices worldwide is estimated to increase to
61 43 billion by 2023, three times more than in 2018 (19). Investments in IoT technology may probably
62 grow at 13.6% per year through 2022 (19). However, in any application area, IoT platforms still share
63 some of the most crucial aspects to evaluate an IoT platform: response time (27%), cost (18%) , energy
64 consumption (18%), reliability (14%), availability (14%), throughput (5%) and security (4%) (10). The
65 above five aspects are interrelated, and depending on the application, the developer will focus on achieving
66 a few specific features. Excluding the cost factor, in this article, we will evaluate the fundamental
67 factors of the IoT platform. The following paragraphs first describe the five drawbacks of the current
68 IoT architecture, then highlight our contribution for the new platform which focus on broker-less and
69 microservice architecture, decentralized identity, and dynamic transmission messages (a.k.a BMDD IoT
70 platform).

71 In terms of response time, every user wants to send or receive data as quick as possible when they
72 make a request. However, depending on the field of application, the response time of the IoT platform
73 may be distinct (37). For example, when applying IoT to the medical field, the required response time
74 should be the smallest as in emergency cases, the patient's life sometimes depends on just a few seconds.
75 On the other hand, for agricultural purposes, the response time can be flexible [1].

76 Some of IoT platform evaluation are energy consumption and reliability related to the protocol used
77 in IoT platforms. Due to their small size, low performance, and limited processing ability, IoT devices
78 often orient towards mobility (60). Therefore, multiple protocols are introduced for IoT, such as MQTT,
79 CoAP, HTTP, XMPP, AMQP, RESTful, Websocket, SMQTT, and DDS. Among them, MQTT is the
80 most popular protocol and is used in both IBM, Microsoft, and Amazon IoT Framework (30). On the
81 one hand, Bansal et al. (14) has evaluated the advantages and disadvantages of all nine protocols above
82 and concluded that the DDS protocol satisfies the best response time and reliability criteria. Indeed,
83 the MQTT protocol has three levels of QoS – 0, 1, and 2, equivalent to an increasing level of reliable
84 transmission; the higher the QoS level, the greater the reliability of the transmission. On the other hand, to
85 achieve this reliability, energy usage and response time have to trade-off. The QoS-0 level has the fastest
86 transmission rate but the lowest reliability (47), while the QoS-2 level is the opposite. According to (86),
87 the power consumption of QoS-0 level is only about 50% compared to QoS-2 level, this means the higher
88 the transmission reliability, the greater the power consumption while the transmission rate decreases.

89 Regarding availability, this is another important aspect of the IoT applications. There are two popular
90 architectures i) architecture that uses a broker (brokering) represented by the MQTT protocol ii) the
91 decentralized architecture represented by a machine-to-machine protocol, a.k.a broker-less (e.g., DDS).
92 The most noticeable weakness of the first architecture is the single failure point (88) since it requires
93 a central broker to collect and coordinate messages, which affects the system scalability. On the other
94 hand, the broker-less architecture allows devices to communicate directly with each other with extreme
95 flexibility, high communication performance, and it is easy to expand the system (14).

96 Besides, the security aspect is an essential because the IoT platform is responsible for collecting data
97 directly from user side. However, this issue has not received much attention as the current IoT devices
98 have low processing capacity, and its protocols are not fully equipped with security mechanisms. The

---

[1]https://www.emnify.com/blog/iot-connectivity-comparison-guide

MQTT protocol has many drawbacks, namely authentication, authorization, and privacy mechanisms (7). Nevertheless, there have been many articles proposing security enhancement mechanisms for MQTT (45; 44; 29) , they are mainly centralized forms of authentication that quickly lead to single failure points and are difficult to scale. Similarly, the DDS protocol also has inadequate security (14). The IoT systems, in general, are vulnerable to attacks due to user behavior. In particular, users often tend not to care about security issues, especially privacy, until damage occurs, such as loss of necessary data (81). For instance, the statistics of (78) presented a significant percentage of IoT users are not aware of where they are sharing information.

In addition, uniformity (23; 55) is also a challenge for IoT platforms. Each IoT application collects and processes a different type of data; for instance, environment-related applications collect only one float value related to temperature or humidity, while coordinate-related applications collect a group of three values, including longitude, latitude, and altitude. This point leads to building multiple different applications even though the main feature is data collection. Besides, there is a significant number of service providers and manufacturers of IoT devices, leading to inconsistencies in data collection and transmission.

From the above analysis, the paper's innovations and contributions are as follows:

Proposing a broker-less and microservices IoT Platform architecture - BMDD: broker-less architecture eliminates the reliance on a central broker for data collection and distribution of control commands. Brokering architecture is very easy to cause a single failure point when there are many users and devices connected to the IoT Platform simultaneously, which inevitably every IoT Platform has to encounter. In addition, the broker-less architecture also allows services to operate independently of each other, meaning that a service failure will not affect the entire operation of the system. Besides, BMDD also follows a modern microservice design that allows scaling out of the system efficiently and grants permission for BMDD to act as an open platform for third-party applications to integrate easily. The problem of monitoring and fixing errors is also an advantage of systems that follow the microservice design.

Proposing a decentralized authentication method with a strict model in the management of users, things, and channels: BMDD uses decentralized identifier (DID) technology based on blockchain technology for users and IoT devices. The nature of objects participating in the IoT network is highly decentralized; therefore, an IoT platform that allows relatively decentralized authentication will have significant relevance and allow the system to be scaled easily. Furthermore, authentication using DID technology allows to take advantage of the benefits of blockchain, and the device's DID token remains forever compared to Single Sign-On and Oauth protocol authentication systems where these tokens always have an expiration time and have to request a new token after a period, which is quite inconvenient. Moreover, DID allows rapid authentication, only about 0.9s - 1.1s, which is entirely suitable for actual deployment and is easily revoked via the API. In addition, BMDD provides user management service according to RBAC architecture and user organization model according to model tree, allowing to quickly isolate one or a group of users when the system is attacked. In addition, BMDD also provides APIs for device and channel management.

Proposing to apply gRPC protocol and dynamic message feature to ensure high homogeneity: The gRPC protocol is applied instead of the popular MQTT protocol because it is suitable for building a broker-less architecture. In addition, gRPC is also appropriate for all kinds of IoT applications (clarified in the background section) and provides excellent performance (83; 82). Most importantly, the gRPC protocol provides the ability to build dynamic messaging, which allows IoT applications to be undiscriminating when collecting different data types such as string, float, integer, or JSON, contributing to the uniformity of IoT platform deployments for different domains without the need to reprogram services, also does not concern IoT device manufacturers.

The rest of the paper is organized as follows:

- Section 2: existing research and related work.

- Section 3: technologies used in the paper.

- Section 4: detailed description of IoT Platform BMDD.

- Section 5: IoT platform BMDD deployment.

- Section 6: test scenarios setting, the evaluation, and discussion.

152   • Section 7: conclusion and future development direction.

RELATED WORK

Broker-less systems

Industrial Internet of Things

The Industrial Internet of Things (IIoT) is a network of sense, management, and communication devices collaborating to attain common goals while satisfying the industrial environment's requirements (57). To increase the safety and effectiveness of industrial processes, IIoT integrates several technical types of equipment (e.g., sensors, communication equipment) (87). Also, IIoT is commonly used to collect, process, and transmit data from industrial environments for both centralized (48) and decentralized (51) architectures.

Research in (61) designed a dynamic reprogramming about IIoT-applied devices based on distributed automation. This paper allowed the allocation of computational tasks on the channel based on each node's specific role and context. Some Coaty solutions' applications for collaborative IIoT were presented in the studies of (68; 67). These modules introduce augmented reality (AR) and blockchain-based application markets in an industrial circumstance. Edge devices were categorized by their available resources, the apps and the services they could access. Blockchain acts as a technology that allows application traceability validation to ensure transparency for the whole system. Besides, Chenaru et al. (17) debated the integration of wireless sensor networks with cloud systems as back-end infrastructure supporting complex monitoring. Meanwhile, the fog computing model have been used to exploit on-node computing resources to efficiently utilize the restricted radio communication channels (52). The work in (77) presented the consensus algorithms that might represent a possible solution for distributed computing. Localization models could be built on the gathered data for online inference in numerous process summaries such as energy management (76).

These proposed approach provided a real-time, deterministic, and re-programmable network behavior that the system used to leverage device capabilities and to schedule the computation workloads in the network while providing real-time guarantees. The novel runtime systems also made them possible to execute dis-persible code in existing off-the-shelf devices leveraging currently evolving technologies.

However, in the above approaches, the connections between the actors were loosely coupled, and it did not need an entity to manage the communication among them. This potential affects the entire system if a malicious user might take over a device and led to a collusion attack (92). For instance, IoT devices are very vulnerable to malicious users or hardware devices such as node detection, corruption, eavesdropping, denial of service attacks, and routing attacks such as sinkholes and wormholes (27). Therefore, a decentralized identity mechanism is essential in the current IIoT environment.

Cyber-physical systems (CPS)

Edward Lee proposed Cyber-physical systems (CPS) to link virtual space and physical reality via the combination of networks, computers, and storage in National Science Foundation 2006 (46). In particular, CPS is an automated distribution system that integrated physical devices with a communication network based on a computer infrastructure platform (90). The main focus of this protocol is on networking different devices in an IoT environment (35). It, therefore, includes a control unit that can regulate sensors and actuators, connect with the physical world, process the data obtained and swap them with other systems and/or cloud services by means of the communication interface (21).

An important feature of CPS is its ability to obtain information and services in real-time (16). Hence, CPSs have been applied to several areas, such as Healthcare (49), Smart Home (70), Transportation (54), and at most popular in Computers' Network (43; 93). In other words, CPS could be thought of as a system that sent and received data from devices over a single network (13).

However, in addition to the listed advantages, CPS needs to consider its stability, trustworthiness, effectiveness, and security in operations (18). With this in mind, one of the top priorities of CPS is to accommodate a high level of security supporting in all layers of the CPS architecture, guarding private data while also providing acute anonymity of data (6). Besides, for the reliability of the CSP, the requests to send and receive data are also very important, where very small vulnerabilities can have huge consequences (84).

Microservice

The microservices architecture addressed the traditional monolithic issues; for instance, when designing a new application is a unit consisting of many components. The application is reasonably divided into modules, and each module serves a function. The monolithic approach shows its limits as the number of functions increases and the app gets larger and larger. Indeed, in a short time, developers cannot fully understand the entire system, which leads to serious problems during construction and maintenance. Therefore, microservices architecture was introduced to fill this gap (15). This section 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.

Microservice framework

Amazon offers AWS IoT Greengrass (OSG) as a solution to migrate analytics capabilities directly to edge devices (e.g., smartphones). Essentially, it is a program that permits running AWS Lambda features locally once installed on an edge gadget. AWS Lambda allows the device to run code without provisioning or controlling servers. Microsoft Azure provides similar functionality to Azure Stream Analytics on IoT Edge (mav), enabling users to use near-real-time analytic functionalities by using Azure Stream Analytics on IoT-applied devices. Besides, Micro-Mu (mic) is an open-source microservices toolset, giving a full stack for building and managing small services. Micro-Mu supports APIs to query microservices used as gateways for external access using Transport Layer Security (TLS). Azure Stream Analytics is a complex event processing and real-time analytics engine. Like AWS IoT Greengrass, the working principle allows edge devices to locally perform Azure Stream Analytics rules. In addition, Google Cloud IoT (OPS) integrated the Apache Beam SDK (Sen), providing a rich set of session and windowing analysis primitives. It presented a united advancement pattern for defining and implementing data processing pipelines across various stream handling engines, including Apache Flink, Apache Samza, Apache Spark, etc.

For the limitation of these platforms, AWS IoT Greengrass neither provide any functionality to move Lambda calculation to and fro between cloud edge devices nor mix with exterior stream processing engines. Again, a limitation of the Azure Stream Analytics tool is that there is no mechanism for dynamically converting rules between the edge and the cloud. For the Apache Beam, this platform only supported core/cloud-appropriate extensible engines and it is not designed to support edge analytics.

Microservice application

Maia and associates (71) presented IRRISENS, which is designed based on microservices architectures used in agricultural environments to sense soil, crop, and atmospheric parameters. IRRISENS interacted with third-party cloud services to schedule irrigation and could control automatic irrigation. For their results, the proposed framework could add new devices (IoT) to the existing system. It also was possible to successfully expose the interaction model to clients' applications. Moreover, security metadata was added to the description, stating what is needed for secure access to a thing. In this case, the Security schema (JSON Web Token Bearer) to be used in order to access to specific content that exposes how to access and interact with the Smart House resources. In addition, Pratama et al. (59) introduced a smart collar based on microservices backend design for dairy cow behavior monitoring. The platform's backend is created using the Microservices architecture idea, where each progress is isolated into services independent from one to another. Each service has specific tasks conveyed through the Rest API. This platform supports five services, namely Auth, Sensor, Monitoring, Learning, Front-end services. To a greater extent, Claudio et al. (12) have implemented a specific set of Microservices for IoT Applications in the Smart City context. Specifically, they carried out a Microservices set for Node-RED, enabling the formation of a great variety of new IoT applications for smart cities, including dashboards, IoT Devices, data analytics, discovery, and the corresponding Lifecycle. It has enabled the creation of a wide range of innovative IoT Applications for Smart Cities, involving Dashboards, IoT Devices, GDPR-compliant personal storage, geographical routing, geo-utilities, data analytics. The proposed solution has been validated against a large number of IoT Applications for the cities of Firenze, Antwerp and Helsinki. Therefore, to exploit these benefits, we apply the microservice architecture to design our proposed framework in IoT platform.

Security and privacy issues

IoT device vendors desired to develop smart IoT devices based on their benefits, which had led to heterogeneous device production and possible cross-platform conflicts; meanwhile, data generated from IoT devices should be kept safe and not misused (31). These two issues created significant challenges in

providing a trusted environment in the IoT platform. It was beyond the scope of this paper to provide a detailed assessment of data security requirements. Instead, in related work, we focused on the currently popular approaches, including the access control models and data encryption, and provided their pros and cons.

Encrypted data approaches

Research in (22) proposed PICADOR, a secure topic-based pub/sub-system through the proxy re-encryption scheme. Brokers re-encrypted the publications, the authorized subscribers then restore these publications' plaintext in their systems. However, this re-encoding increases the computational cost dramatically at the broker point. The subject matter of each publication is shipped off the agent in plaintext, which could not fulfil the confidentiality assurance conditions of the participants.

A high-level study project on Content Centralized Networks (called CCNx or Named Data Network - NDN) based on the pub/sub system was introduced in (64). CCNx uses a hierarchical naming approach (39) to organize data content. Unlike the traditional MQTT system relying on end-to-end delivery, CCNx exploits a broadcast-based to convey information rather than broker-based. However, this issue might not apply to malicious publishers; for instance, if a malicious publisher uses the fake labels or re-uses another publisher's public key, the subscribers in CCNx may get the faulty publication.

To address this issue, Trinity (62) was introduced as a blockchain-based distributed pub/sub broker. Specifically, Trinity checks the immutability of transactions in the same system by integrating the broker system in the same blockchain network. Trinity also addresses the ordering problems by using a blockchain-based system. However, authentication problems can seriously affect the security of the entire system, since a malicious device (e.g, malicious code, controlled by an attacker) could replace one benign device of the users that might seriously affect the security of the whole system (80).

Lu et al. in (50) introduced a resolution to gather data while conserving the confidentiality of fog computing-enhanced IoT devices. Authors 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. 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, where only authorized objects can accessed data. (91; 74).

Besides, Anusree and Sreedhar (8) proposed a secure broker-less sub/pub system. In particular, this approach was proposed to use the Elliptic curve-based identification mark encoding algorithm; thereby, they effectively reduce the cost of encryption and decryption on the respective parties. In other words, thanks to the Elliptic curve-based encryption algorithm, the costs related to computation and communication costs no longer affect the whole system. However, the key privacy issues still existed and had not been addressed in such a mechanism (45; 44). Furthermore, complete encryption of the data generated by IoT devices is not strictly necessary since not all data is sensitive, depending on the usage context; for example, privacy-preserving in a healthcare environment (25; 26) and cash-on-delivery (33; 34).

In this article, to balance the above problems, we only apply Blockchain to build a DID model for authentication purposes. To protect the IoT data, we apply a decentralized mechanism based on a role-based access control model instead of encrypting all data.

Access control models

Sunghyuck (36) demonstrated that security threats could occur in each component of the IoT. Therefore, IoT devices and their generated data needed an access control method to protect the device from unauthorized personal device intrusion (69). Jain and Nishtha (40) introduced a service delivery and utility manager (SDUM) as an intermediate data access layer between data owners and data consumers. The data consumer sends a command to SDUM for accessing the data using a unique personal ID. Then data access floor transits the user's verification order to KMaaS (i.e., key management as a service). A one-time password (OTP) is utilized to check a correct user prior to sending information; afterwards data access layer confirms the user with an OTP code. SDUM transfer scrambled data to the corresponding user via a safe communication route. Besides, Rashid et al. (63) proposed Enhanced Role-Based Access Control (ERBAC) for securing IoT medical data in Health Care Systems in terms of its storage on public cloud systems. The model was implemented by using the Model View Controller Framework (MVC) of Microsoft (72).

310 However, the bouncing model only uses basic authentication methods based on the IDs of the data
311 owners and data consumers. This point is not suitable for IoT devices as well as problems with multi-level
312 users where the lower level received the permissions to access data from the upper ones (85). Moreover,
313 instead of dealing with security requirements such as collecting data from IoT devices, defining roles
314 for each data user, this paper focuses on data management in the Microsoft's Azure cloud. These above
315 solutions did not solve the key problems of access control in the IoT platform; it merely introduced the
316 traditional solution for data protection based on the RBAC model (73).

BACKGROUND

318 This section addresses details regarding the technologies used to build the BMDD. Besides, selected
319 technologies and current popular technologies are also compared to support the article logically.

Internet of things protocol

321 IIoT devices are small and possess low processing capacity; thus, the protocol selection for the communi-
322 cation process is crucial, determining the architecture of the entire IoT Platform, taking into account these
323 key factors: transmission speed, reliability, power consumption, scalability, and security.

MQTT Protocol

325 MQTT (Message Queuing Telemetry Transport) is a small network protocol operating under the publish-
326 subscribe mechanism according to ISO and OASIS standards [2]. The main concepts of MQTT include
327 MQTT broker, MQTT client (publisher and subscriber), MQTT topic.

328 MQTT Broker is the junction of all incoming connections from the client and is the center of systems
329 that uses the MQTT protocol. The principal responsibility of the Broker is to receive messages from
330 publishers, queue them, and then forward them to subscribers based on topics.

331 MQTT client is classified into two groups: publishers and subscribers, in which the prior is the
332 message sender to the Broker and the latter is the message receiver from the Broker. A client can act as
333 both a publisher and a subscriber on one or more specific topics.

334 The topic is a logical concept that MQTT uses to route messages sent from publisher to subscriber;
335 for example, when the publisher sends a message to topic A, only topic A subscribers can receive the
336 message. The model has the advantage that every user who subscribes to the topic receives a message
337 from the publisher. However, sensitive data is considered an enormous weakness in terms of security.

338 Another concept of MQTT is Quality of Service (QoS). The MQTT protocol has three QoS levels
339 including QoS-0, QoS-1, and QoS-2 which are defined as follows (9):

- QoS-0 (maximum one time): one packet is transfered to a destination up to once.

- QoS-1 (minimum one time): each packet is emitted to the destination at least once, expecting packet
  iteration to occur.

- QoS-2 (precisely one time): each packet is shipped to its target sole one time.

344 The transmission process of MQTT according to each QoS level[3] is shown as in Figure 1:
345 Figure 1 indicates that transmitting a message with the highest accuracy (QoS-2) using the MQTT pro-
346 tocol requires at least four steps, which reduce the transmission speed, increase the number of connections
347 as well as increase the energy usage. Although the article (44) proposed a model combination between
348 MQTT and message queue to make use of the quick transmission rate, the tiny energy consumption of
349 QoS-0, as well as ensure the reliability of QoS-2, however, there is still a weakness that the publisher does
350 not receive any notification to ensure the message reception. In addition, when the subscribers receive the
351 messages, the MQTT protocol also does not support sorting them in order (75), which can seriously affect
352 the accuracy of the data.

353 In addition, MQTT has another noticeable concept, "keep-alive"[4], which is a function of the MQTT
354 broker that allows maintaining the connection so that communication is available between the broker and
355 the client. When the keep-alive time exceeds, and the client does not send any messages to the broker,

---

[2]https://mqtt.org/
[3]https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/
[4]https://www.hivemq.com/blog/mqtt-essentials-part-10-alive-client-take-over/
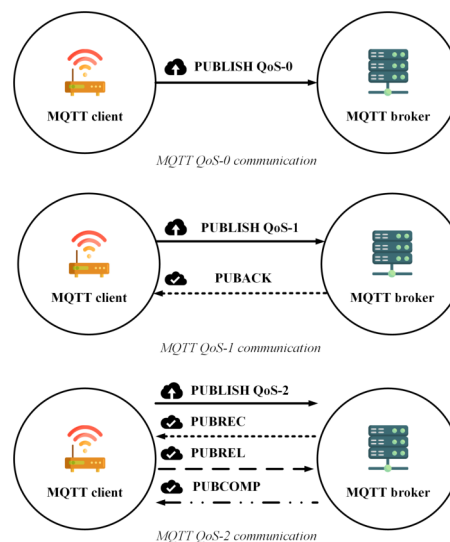
Figure 1. MQTT QoS level communication

356 the broker will disconnect from the client. At this point, the client must send a "PINGREQ" packet to
357 notify the broker that it is still operating, requesting to reconnect, then the client can send the message to
358 the broker. In a practical scenario with thousands of clients, the broker has to handle many PINGREQ
359 packets that can overload the broker. The keep-alive feature allows optional maintenance of a constant
360 connection between the client and the broker to reduce energy consumption. However, this feature also
361 has the weakness of not ensuring system availability, especially in emergencies.

362 ### gRPC protocol

363 gRPC (general-purpose Remote Procedure Calls) protocol [5] is an open-source system of superior ef-
364 fectiveness of the Remote Procedure Call (RPC) protocol improved by Google in 2016. The gRPC is
365 constructed on top of the http/2 protocol, which contains multiple advancements over earlier forms to
366 permit greater and more productive HTTP connections. The gRPC provides machine-to-machine (M2M)
367 communication. One of the most important aspects of http/2 is multiplexing, allowing us to deliver and
368 gain many packages in a sole connection in comparison with http/1.1 as illustrated in figure 2. This feature
369 is one of the main advantages that make us choose gRPC as it allows the client to open a single connection
370 and transmit a collection of different data to the server, which saves energy during data transmission. In
371 addition, gRPC carries a wide variety of programming languages and is perfectly suitable for embedded
372 devices (41).

373 The gRPC protocol provides for 4 types of machine-to-machine communication (38) as in figure 3 :

374 • **Unary**: This type is similar to conventional client-server communication. The client sends a request
375   to the server; afterwards, the server processes and transfers the outcome to the client.

376 • **Server streaming**:In this type, the server receives a request from the client, after processing, sends
377   back a stream of data to the client. The client will be notified when they reach the end of the data
378   stream. The sequence of messages of every stream is ensured to be identical between client and
379   server.

380 • **Client streaming**: Like server streaming, in this type, the client will be the side sending the data
381   stream to the server. After reading and operating the required processes, the server delivers the
382   response to the client.

383 • **Bi-directional streaming**: In this approach, data will be sent from both directions, client and
384   server, independent from each other. Also, the client and server can deal with the data stream
385   independently, meaning that when the client delivers information to the server, the server can

---

[5]https://grpc.io/
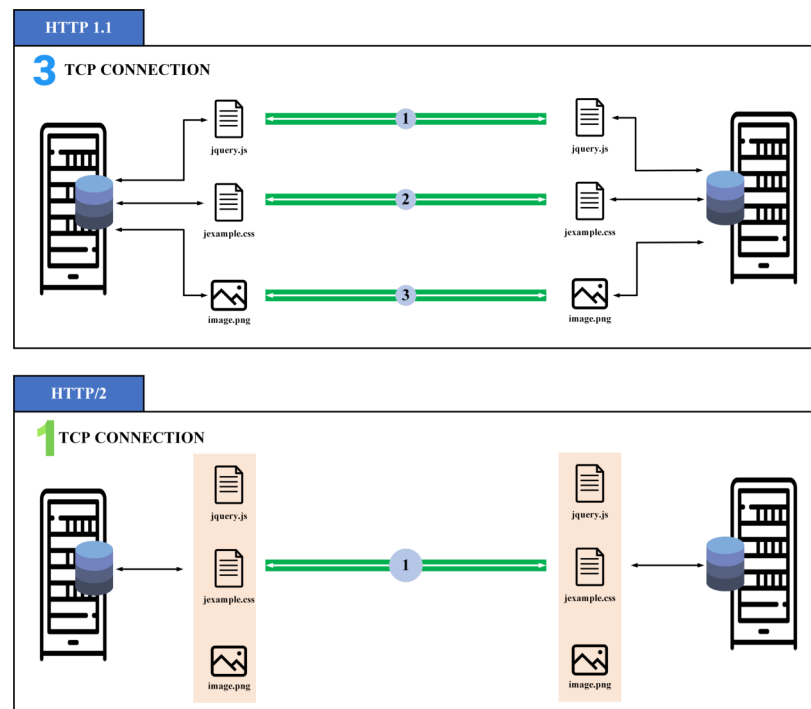
Figure 2. Overview of multiplexing in http/2 (24)

386   conduct a particular task (while still accepting the messages coming from the client) and transfer
387   the response back to the client (while the client is still conveying other messages).
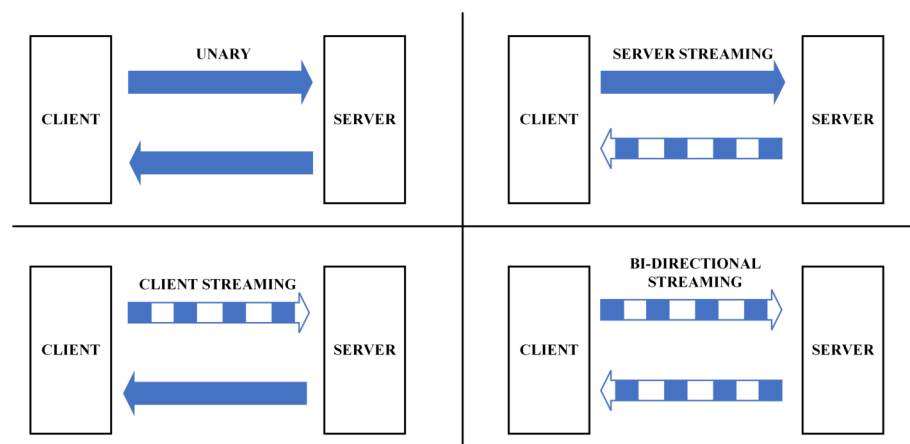
Figure 3. M2M communication in gRPC (38)

388   There are some features that MQTT does not provide. Regarding the first benefit compared to the
389   MQTT protocol, gRPC offers four types of machine-to-machine communication, which make it highly
390   appreciated due to its suitability with the diversity of IoT applications. For example, when applying
391   IoT in agriculture, the unary communication type is suitable to collect temperature data, similar to the
392   ability provided by MQTT. However, considering the application of IoT for the medical field with many
393   sensors working simultaneously, and each sensor may stream a bunch of data in a different period, the
394   client streaming communication type is a better option than MQTT. Apart from that, with the ambition
395   of building BMDD towards adaptation with many areas of IoT application, gRPC is the optimal choice
396   since it provides a variety of communication forms. Moreover, a prominent advantage of gRPC over

397 MQTT is that it guarantees message ordering in streaming data cases; both the sender and the receiver get
398 notifications when the transmission is successful, which is probably a superiority.

Dynamic message

400 The gRPC framework provides a special data type, "Any" [6]. This data type allows the client and server to
401 freely exchange messages regardless of the data form. The "Any" type also supports the four types of
402 communication methods that gRPC provides. This feature is handy in constructing BMDD because it
403 provides the ability to transfer many different types of data without developing a single type of service,
404 increasing the compatibility of BMDD. Therefore, it can be applied to many different fields using IoT
405 technology, contributing to the increment of the homogeneity of systems. For instance, a customer
406 of a health care service running on the BMDD platform often uses transmission devices provided by
407 company A. When this customer travels to another country, but the hotel only has B carrier equipment, the
408 incompatibility between devices is negligible when using BMDD. The communication process between
409 the client and the server using the "Any" data type is summarized as follows:

410   **Step 1**: build a function to send data (function **putData**) on the client side with the data type "Any":

```
func putData(valueType string, value []byte)
error {
    conn, err := grpc.Dial(address, grpc.WithInsecure())
    if err != nil {
        log.Fatalf("did not connect: %v", err)
    }
    defer conn.Close()
    client := proto.NewDataServiceClient(conn)
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*3)
    defer cancel()
    reqData := &proto.PutDataRequest{
        Data: &any.Any{Value: value},
        Type: valueType,
    }
    resp, err := client.PutData(ctx, reqData)
    if err != nil {
        log.Printf("put data error. %v", err)
        return err
    }
    if resp.Err != 0 {
        log.Printf("put data exec error. %v", resp.Desc)
        return err
    }
    return nil
}
```

436   **Step 2:** When sending data, the client attaches the type of this data. This is illustrated in function
437 **testPutData()**

```
func testPutData() {
    putData("int", []byte("1"))
    putData("float", []byte("1.1"))
    putData("bool", []byte("true"))
    putData("string", []byte("this is a string msg"))
    putData("json", []byte("{\"msg\":\"this is a json msg\",
    \"other\":1}"))
}
```

446   **Step 3**: After receiving data in form of bytes from the clients, server will base on the data type to
447 parse the byte data with the correct data type (function **parseAnyData()**)

---

[6]https://developers.google.com/protocol-buffers/docs/proto3#any

```
448  func parseAnyData(valueType string, anyData *anypb.Any) (interface{},
449  error) {
450      if anyData == nil {
451          return nil, errors.New("illegal param")
452      }
453      var err error
454      var reqData interface{}
455      switch valueType {
456      case "int":
457          reqData, err = strconv.ParseInt(string(anyData.Value), 10, 64)
458      case "float":
459          reqData, err = strconv.ParseFloat(string(anyData.Value), 10)
460      case "json":
461          err = json.Unmarshal(anyData.Value, &reqData)
462      case "bool":
463          reqData, err = strconv.ParseBool(string(anyData.Value))
464      case "string":
465          reqData = string(anyData.Value)
466      default:
467          reqData = string(anyData.Value)
468      }
469      return reqData, err
470  }
```

We also provide the source code via the github link [7], [8].

Kafka Message Queue

Kafka [9] is a dispersed texting scheme, able to transmit a huge number of messages in real-time, where the information will be deposited on the message line and on the disk to guarantee security if the receiver has not received it.

Kafka architecture consists of four principal elements: producer, consumer, topic, and partition. The producer element is the client to distribute messages into topics. Data transmitted to the topic's partition is stored on the broker. Kafka consumers can be multiple clients who subscribe to and receive messages from similar or different topics classified by group names. Data is conveyed in Kafka by many topics which can be created when passing data to varied applications is necessary. For the BMDD platform, the Kafka topic concept is equivalent to a transmission channel created and managed by each user. The Partition's function is to store the topic data, and and each topic can at least have partitions. Each Partition can store data permanently and assigns them an ID called offset. Moreover, a set of Kafka servers, known as brokers and zookeepers, is a broker management service.

Because gRPC does not support message caching, and to increase the availability of BMDD, we add a Kafka message queue. All messages exchanged between services in BMDD must go through Kafka. This method allows the service to be recovered when it fails, and messages will not be lost during downtime due to its storage on Kafka for a configurable amount of time.

Microservice and monolithic architecture

Monolithic architecture is an architecture software in which different components (such as authorization, business logic, notification module, etc.) are combined into a single program from a single platform (32). Microservice is an approach to develop an application using a set of small services, each of which runs its process independently, usually an HTTP resource API. This modern architecture allows more extensive, more complex, and scalable applications (79).

To clarify the monolithic and microservice architectures, the description of the image below illustrates the application of these two architectures in building an E-commerce application, as in figure 4 and 5.

---

[7]https://github.com/thanhlam2110/dynamic-message-client.git
[8]https://github.com/thanhlam2110/dynamic-message-server.git
[9]https://kafka.apache.org/
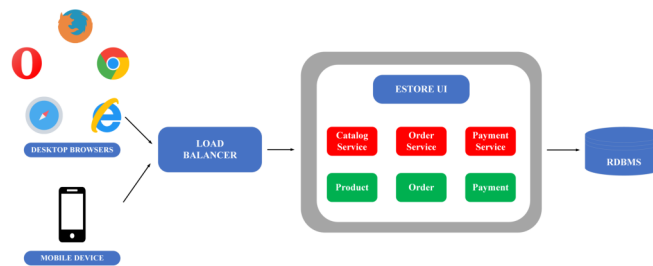
Figure 4. Monolithic flow on the example of E-commerce application (32)
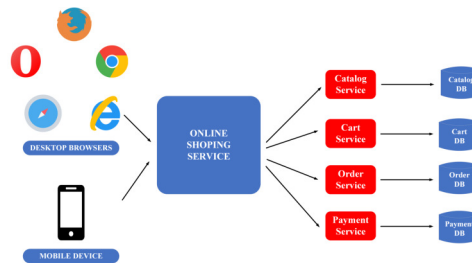
Figure 5. Microservice flow on the example of E-commerce application (32)

Monolithic architecture's advantage is its ease to be developed and deployed.. However, the complicated maintenance mechanism, low reliability (one failure can crash the entire application), availability (must be re-deployed for update), and low scalability (32) are the drawbacks of this structure.

The downside of monolithic architecture is the benefit of microservice architecture and the applications of which are simple to maintain since the modules are completely separate from each other. It also provides high scalability and high reliability as a service failure does not affect other services. The only weakness of the microservice architecture is the complex deployment (32).

This paper prioritises constructing BMDD under microservice architecture because of the advantages it provides, simplifying the expansion and new features integration processes for BMDD in the future.

Decentralized Identity

DID (Decentralized identifiers) is a novel way to identify data objects which DID's controller specifies, designed to perform the identification separating from a third party identifier. With the utilization of URIs, DID establishes the association between a subject with a document to build up reliable interactions, allowing control of the verification without testifying with a third party. Each DID material can reveal cryptographic material, confirmation techniques, or services, enabling trusted interactions to subject and prove control by DID controller.

Sidetree

Sidetree is a protocol that allows users to init, manage the global and unique PKI metadata controlled by their user. Thanks to the connection to the existing decentralized system such as Bitcoin, Ethereum, ..., Sidetree protocol also has open, public, and permissionless and other properties.

Figure 6 elucidates **the structure of Sidetree-based DID** overlay network:

- Existing decentralized networks, such as Bitcoin, Ethereum, e.g., will be connected to SideTree via DID operations. The operations of DID consist of four main functions: creating, updating, recovering, and deactivating, responsible for querying, initializing, and updating data in the decentralized system.

- Sidetree protocol is organized into nodes located between the decentralized system and the CAS network, interacting with the decentralized network via operations according to predefined rules.

- CAS (Content-addressable storage) stores and distributes operation files having an obligation to define and store DID operations. Expressly, the Core Index File declare the create, recover and deactivate operators; meanwhile, Provisional Index File contains operator updates. Besides, Core

527   Proof File and Provisional Proof File contain necessary cryptographic proofs of operators like
528   signatures, hashes, etc. . Finally, SideTree uses $COMPRESSION_ALGORITHM$ to generate a
529   Chunk Files file which contains the operation source data.
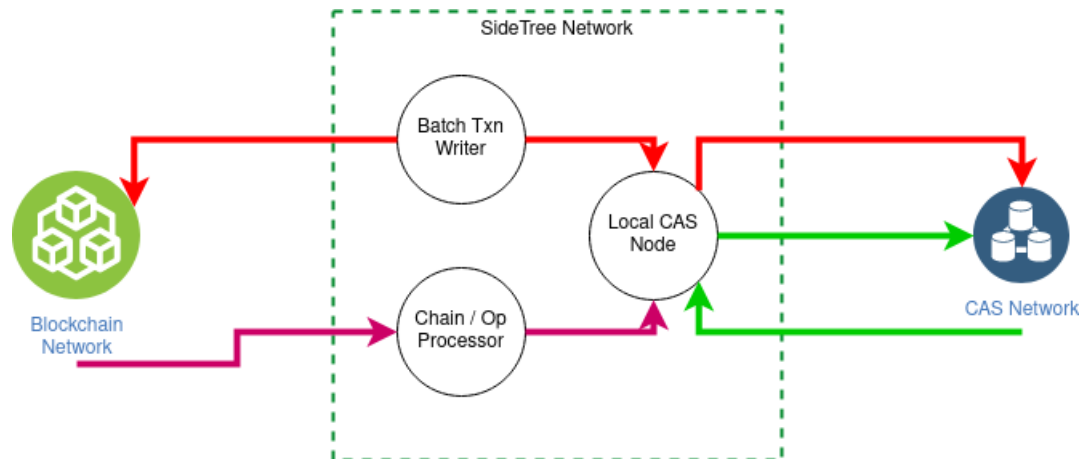
Figure 6. Sidetree-based DID overall network

530   Figure 7 elucidates the **file structures of the side tree protocol**: The protocol comprises of three file
531   constructions, which keep DID administration information and are meant to support principal functionality
532   to facilitate light node configurations, minimize forever the reserved data, and guarantee resolution
533   effectivesness of DID.
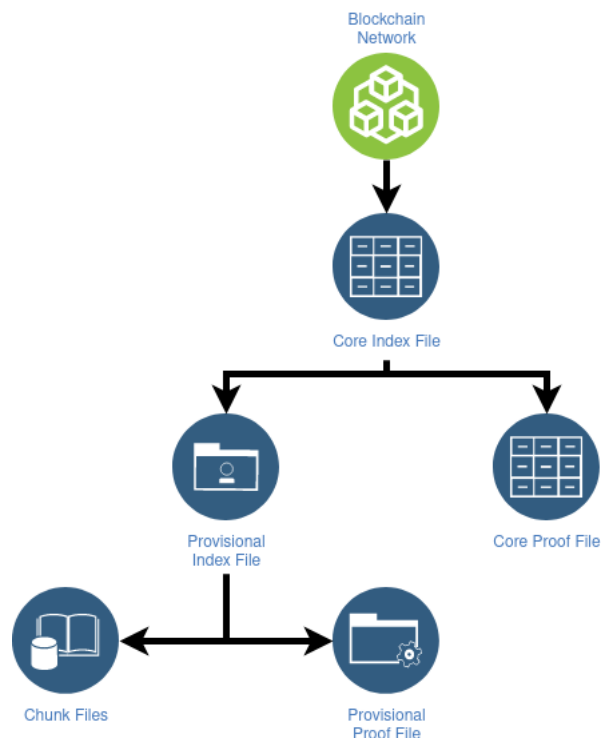
Figure 7. File Structures of Sidetree protocol

534   **DID URI Composition**: The SideTree protocol uses a unique identifier segment to identify a DID
535   method called DID Suffix; when a Sidetree-based DID is generated, the unique identifier corresponding
536   to it is also generated. These DID URIs are used to identify a corresponding method correctly and can be
537   self-certifying. DID URI can be presented in two primary forms: short-form and long-form.

538   **Short-form URI**: Short-form URI: The SideTree protocol uses a hashing algorithm to generate a
539 Short-Form DID URI from $JSON_CANONICALIZATION_SCHEME$; the short-form of the DID URI has
540 the format:

541 `did:METHOD:<did-suffix>`

542   **Long-form URI**: Long-form URI: The formation of Long-Form DID URIs is to prevent endless
543 time situations between DID's generation and anchor, propagate and process of DID operation by self-
544 certifying and self-resolving capabilities. Structurally, Sidetree Long-Form DID URIs are the Short-Form
545 DID URIs but appended with a new segment called long-form-suffix-data; the value of this segment is
546 calculated from Operation Suffix, and Operation Delta then encoded via $DATA_ENCODING_SCHEME$
547 function. Long-Form DID URI has format: did:METHOD:¡did-suffix¿:¡long-form-suffix-data¿

548 `did:METHOD:<did-suffix>:<long-form-suffix-data>`

549 ***JSON Web Signatures***
550 Generally, the authentication mechanism of DID operations is implemented based on JSON Web Signa-
551 tures and represented by two actions Signing and Verifying, consisting of four primary functions: create,
552 recover, deactivate and update. Except for the create method, the others require key materials for verifying
553 and authenticating the transactions. When a transaction is required to sign, that transaction must contain
554 two parameters *kid* and *alg*.; while the *alg* is a required parameter and cannot be null, the *kid* can be
555 declared and assigned a value or not. The format of the authentication string is:

556 `{`
557   `"kid": "did:example:example_kid_value",`
558   `"alg": "example_alg_value"`
559 `}`

560 The authentication process on DID is the same for each transaction if that transaction is required to
561 perform Verification. In the first step, the kid is extracted and traversed to find an authentication method
562 with the same id and *kid* pair. Then the found authentication method is converted to JWK to perform the
563 JWS Verification process in the final step.

564 ***Proof of Fee***
565 The mechanism of the Sidetree network is designed to resist low-cost tampering, mainly for permissionless
566 and open implementations, by leveraging public blockchains with two key characteristics: Base Fee
567 Variable and Per-Operation Fee. **Base Fee Variable** is calculated from a set of variables belonging to a
568 particular anchor system using defined functions. In addition, it can be used to set the lowest transaction
569 cost corresponding to the number of DID operations calculate the costs of other economic setups effectively.
570 **Per-Operation Fee** has the main purpose to validate the user's transaction by performing the following
571 processes sequentially:

572   1. Determine the Base Fee Variable of the transaction or block.

573   2. Use Operation Count from Anchor String to Multiply the Base Fee Variable.

574   3. Validate the transaction anchored in the anchoring system.

575   4. Analysis and execute transaction if the transaction uses required fee

576 **The Acronyms**
577 We will use the acronyms which are shown in Table 1 for the whole article.

578 # APPROACH

579 **Definition**
580 In the background and related work section, we introduced the technologies used to build the BMDD
581 platform and associated research directions. This section dedicates to user-related concepts, an essential
582 subject of BMDD.

Acronyms	Description	Acronyms	Description
IoT	Internet of Thing	RBAC	Role Based Access Control
BMDD	Broker-less and Microservice architecture, Decentralized identity, and Dynamic transmission messages	DID	Decentralized Identity
		CSS	Control Service Server
		CSC	Control Service Client
MQTT	Message Queuing Telemetry Transport	CDS	Collect Data Service Server
gRPC	general-purpose Remote Procedure Calls	CDC	Collect Data Service Client
		LB	Load Balancing
DBL	Database & Log	Org-1	Organization 1
OMS	Object Management Service	Org-2	Organization 2
DPS	Data Processing Service	User-A	User A
RTM	Realtime Messaging Service	Dr-B	Doctor B
MQ	Message Queue	Nu-C	Nurse C
QoS	Quality of Service	CCU	Concurrent User

Table 1. The Acronyms and their description

User organization

There are two main groups in BMDD: root users and normal users. The root user is the group that manages the BMDD platform, having full rights to the BMDD participants except for reading the user's data. The group of normal users use the BMDD service, divided into two subgroups, parent and child users, supposing that the user group is an enterprise and wants to organize the user rights hierarchy. The child user connects to the parent user through the "user_parentid" attribute, which means that the child user's "user_parentid" field is equivalent to the parent user's "user_name" field. Every normal user is a child user of the root user. User information is stored in NoSQL format, allowing flexibility to add additional attributes for the user apart from the specified ones. The above user management model is called the model tree and is illustrated in figure 8.
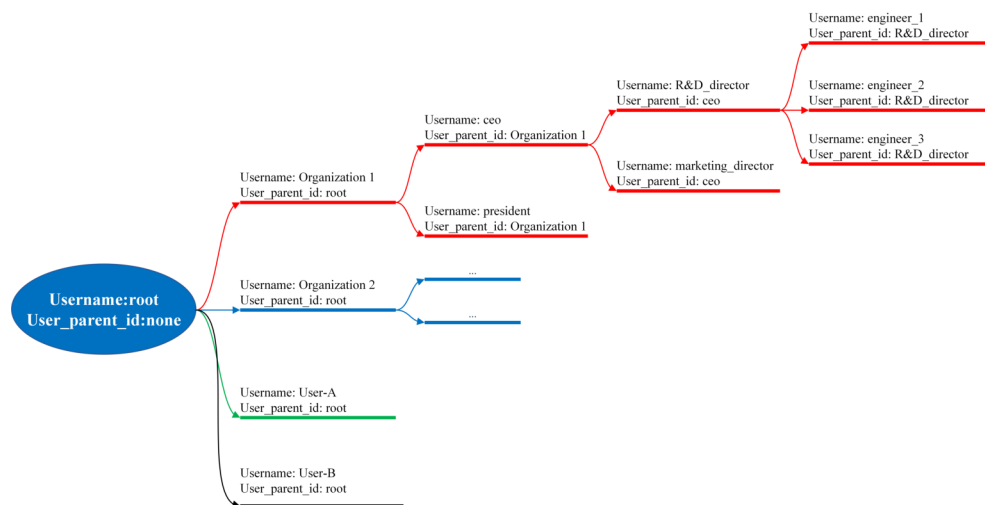
Figure 8. User organization as model tree

The parent user structure in JSON format is as follows:

```
{
    "userid":"76579b0e-e899-4fc9-a839-607c371b9585",
    "username":"organization1",
    "usermail":"representation@ organization1.vn",
    "userstatus":"ACTIVE",
    "userparentid":"root",
    "usertype":"representation"
```

601   }

602   And the child user structure in JSON format is as follows:

```
603    {
604        "userid":"37389774-98a6-4dc2-a8b1-22fc29d321ba",
605        "username":"thanhlam",
606        "usermail":"thanhlam@organization1.vn",
607        "userstatus":"ACTIVE",
608        "userparentid":"organization1",
609        "usertype":"normal"
610    }
```

611   The implemented model trees have two primary purposes. Firstly, BMDD can manage user hierarchy,
612   suitable for enterprise deployment environments and individual customers. Secondly, the users have the
613   right to build coherence and stricter management thanks to hierarchy management. For example, BMDD
614   provides an API enabling the parent user to disable all child users through the status field. When the user
615   status is "DISABLE", the users and their devices will be isolated and cannot interact with the BMDD,
616   which allows to quickly isolate a group of users when a hacker attacks the BMDD system from one user
617   point. For the root user, this is to isolate an entire organization from the BMDD. Restoring can easily be
618   done by changing the status to "ACTIVE". The API also ensures that only the high-level user can change
619   the low-level status being its child user.

620   ***User role***
621   BMDD user role involves two concepts, including user role for the organization of users following the
622   model tree (described in the user organization section) and role related to IoT service.
623   User roles related to IoT services include the following capabilities. First, the user has full authority
624   to create logical information to manage his devices and communication channels. The communication
625   channels are equivalent to the Kafka topic concept presented in the background. This information will be
626   embedded into the device and used for checking the permissions when the device interacts with BMDD.
627   This embedded process is beyond the scope of this article; thus will not discuss in detail. Besides, the user
628   can assign device information to a communication channel, allowing users to strictly manage device and
629   channel, only devices that have been authorized and assigned to a specific channel are allowed to send
630   and receive messages on that channel. This approach also helps to reduce security risks caused by the
631   careless behavior of users. Finally, the users also have control over sharing their devices with other users;
632   for instance, a patient can give their doctor permission to access the device and receive heart rate data
633   from a sensor, etc.

634   **BMDD platform proposal**
635   IoT Platform BMDD is designed according to broker-less and microservice architecture, including 3 main
636   layers: thing layer, edge layer, and cloud layer, as described in the figure 9.
637   Each layer include these services described as following:
638   The **thing layer** is the layer of physical devices (gateways) that receive information from sensors or
639   users. Each gateway uses two services, including control service client (CSC) and collect data service
640   client (CDC). The term "client" here distinguishes it from service of the same name implemented on the
641   edge layer. Every gateway only needs these two services, regardless of application to any IoT field, due to
642   the dynamic message feature that BMDD provides. In addition, CSC and CDC also operate independently
643   following the microservice architecture. The role of the CSC is to receive control commands from the
644   control service server (CSS) to control remote devices, while CDC has to send the collected data to the
645   edge layer. In practice, the **thing layer** includes the gateway and IoT devices (sensor, actuator, etc.). In
646   BMDD, the gateway is implemented on the user side and it communicates locally (LAN network) with
647   IoT devices via wifi (65) or Bluetooth (66) to collect data or control devices. For communication with the
648   Edge layer (WAN network), the gateway use gRPC protocol via CDC and CSC service.
649   The **edge layer** includes three services: control service server (CSS), collect data service server
650   (CDS), and load balancing (LB). CSS is responsible for receiving control commands from a valid user
651   and then sending this command to the CSC. The role of the CDS is to receive data sent from the CDC,
652   which is attached to the user's DID token, thing-id, and channel-id. CDS initially sends this token to
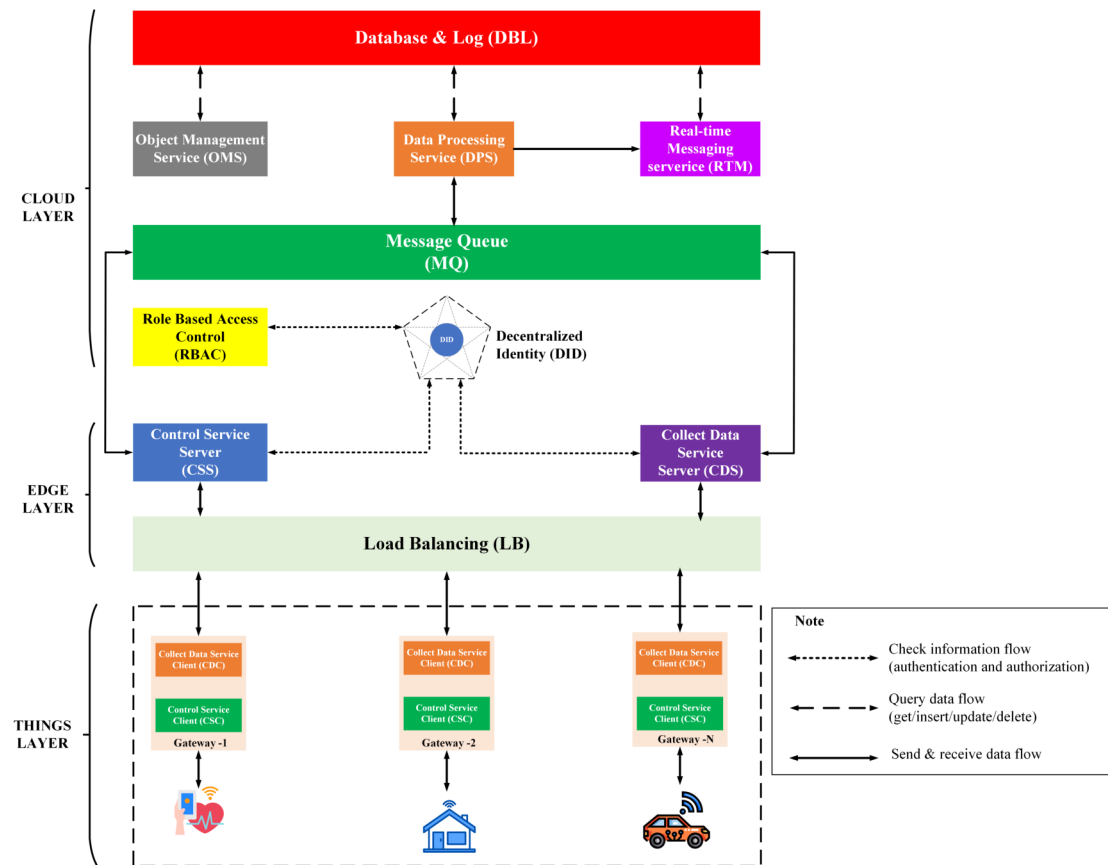
Figure 9. BMDD platform architecture

the decentralized identity service (DID) on the cloud layer for authentication, then sends thing-id and
channel-id to the role-based access control service (RBAC) for permission check. If the data passes the
authentication and authorization process, it will transfer to the message queue through CDS. LB acts as a
proxy and handles the load distribution for the system.

The **cloud layer** includes service role-based access control (RBAC), decentralized identity (DID),
message queue (MQ), object management (OMS), data processing (DPS), real-time messaging (RTM),
and database-log (DBL). DID is responsible for authenticating the user with the DID token. The role of
RBAC is to check user roles configured by the user to predefine permissions related to the data collection
or device control process. RBAC is capable of providing APIs that allow users to build their access control
policies flexibly. The implementation section will present the detailed scenario for the user role. MQ
receives data from the CDS and control command from CSS and distributes it to other services. DPS
receives data from MQ and performs advanced processing, e.g. threshold analysis. The result of the
processing may be to determine the threshold levels that should be alerted to the user. RTM handles the
process of sending notifications to users. Data processed by the DPS can be stored in the DBL. Lastly,
OMS provides APIs that allow users to create/update/delete management information of devices and
channels. It also allows upper-level users to create and manage lower-level users with a model tree.

IMPLEMENTATION

Overview

In the implementation section, we assume a BMDD-specific application scenario described as follows:
two IoT service providers using the BMDD platform: smart home service provider (Org-1) and smart
health service provider (Org-2). User-A, a customer of Org-1, decides to register to use Org-2's remote
health care service and is supervised by doctor B (Dr-B) and nurse C (Nu-C). This scenario frequently
happens in practice, particularly during the Covid-19 pandemic.

676 User-A using smart home service with Gateway device can collect information about the house such as
677 room temperature, humidity and can control home devices. Besides, the gateway also collects information
678 related to heart rate, blood pressure, etc., as health care services are also used. User-A only allows the
679 gateway to send its health-related data to Dr-B and Nu-C. At the same time, User-A only allows Nu-C to
680 view health data while Dr-B has additional rights to control healthcare devices remotely.

681 To implement the above scenario through BMDD, User-A must perform three processes, including
682 registering to use the service (1), creating management information for the devices and channels (2), and
683 sending health data (3).

684 Process 1, User-A registers to use IoT service to be granted DID token through DID service.

685 Process 2, User-A creates device management information, communication channels and then maps
686 devices on these channels through the OMS service. Subsequently, User-A creates authorization roles
687 through the RBAC service.

688 The authorization is specified as following:

689 • User-A has full access to his/her devices, including smart home devices and medical devices.
690   User-A will assign permission only to allow the gateway to send health-related information to
691   Org-B while smart home-related information is not allowed.

692 • User-A authorizes dr-B to read medical data and control medical devices.

693 • User-A authorizes nu-C to read medical data.

694 The third process is sending medical data of Org-1 from User-A to the medical staff of Org-2. This
695 process is described in the figure 10, including steps from (1) to (9).
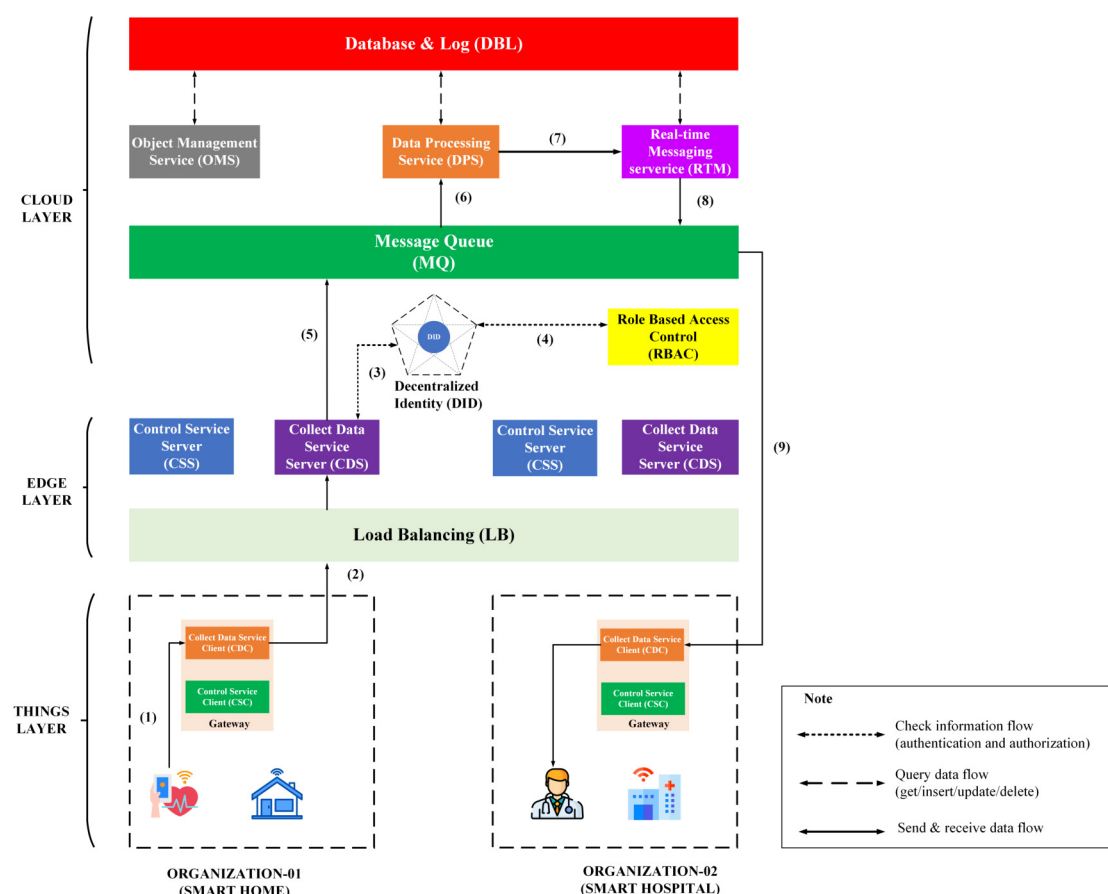
Figure 10. Overview model of data collection process through BMDD

696 The steps of the data collection process are described as following:

- Step (1): The User-A body sensor sends health data to the gateway, stored locally.

- Step (2): After a while, this data will be streamed by CDC to CDS with DID token of User-A.

- Step (3): CDS sends User-A's token to a decentralized identity (DID) for authentication. If the authentication process is successful, it can proceed next step (4); otherwise, all data will be dropped, and CDS will send an invalid token response to CDC.

- Step (4): When the user's token is authenticated, the CSC sends user_iD resolved from the user DID token to the RBAC service. The RBAC uses the user_iD to check user permissions. If permission is valid, it can continue to the next step (5). Otherwise, all data will be dropped, and CDS will send an invalid role response to CDC.

- Step (5): After the authentication and permission checks, the health data will be sent to the MQ.

- Step (6): DPS receives health data from MQ and conducts analysis.

- Step (7): Data, after analysis, will be sent to RTM and evaluated according to a predefined threshold.

- Step (8): Assuming the health data exceeds the allowable threshold, for example, User-A's average heart rate in 5 minutes is 180, then RTM will send a warning message to MQ.

- Step (9): CDC on the Org-B side receives a warning message from MQ and sends it to Dr-B and Nu-C.

Also, in this scenario, Dr-B uses only one process controlling the device. Nu-C has the right to view medical data, however for a brief presentation; we mainly present the procedures related to Dr-B as Dr-B also has additional control over the medical device. The appearance of Nu-C is to ensure that the modelling implementation scenario is more accessible to readers.

Dr-B will control emergency life support devices after receiving a warning that User-A's heart rate exceeds the threshold. The process of controlling the device through BMDD is described in the figure 11, including steps from (1) to (6).

The steps of the device control process are described as following:

- Step (1): Dr-B sends control commands to CSC.

- Step (2): CSC sends control command with Dr-B's token to CSS.

- Step (3): CSS sends Dr-B's token to DID service for authentication. If the authentication process is successful, it can move to step (4). Otherwise, the control commands will be dropped, and the CSS will send an invalid token response back to the CSC.

- Step (4): When the Dr-B's token is authenticated, the CSS service sends Dr-B_iD resolved to the RBAC service from the Dr-B DID token. The RBAC will use Dr-B_iD to check Dr-B permissions. If permission is valid, it can proceed to step (5). Otherwise, the control commands will be dropped, and the CSS will send an invalid role response back to the CSC.

- Step (5): After going through the authentication and permission check, the control command will be sent to the MQ.

- Step (6): CSC on gateway of Org-1 side receives a control command from MQ and performs device control.

BMDD Platform in detail

The section above has shown the entire exchange process between services inside BMDD in a specific scenario. This section gives the details of each process. However, because BMDD is designed according to microservice architecture, the communication process of services is enormous and complicated, so we only focus on the most important processes, including:

- Requesting and resolving DID token.
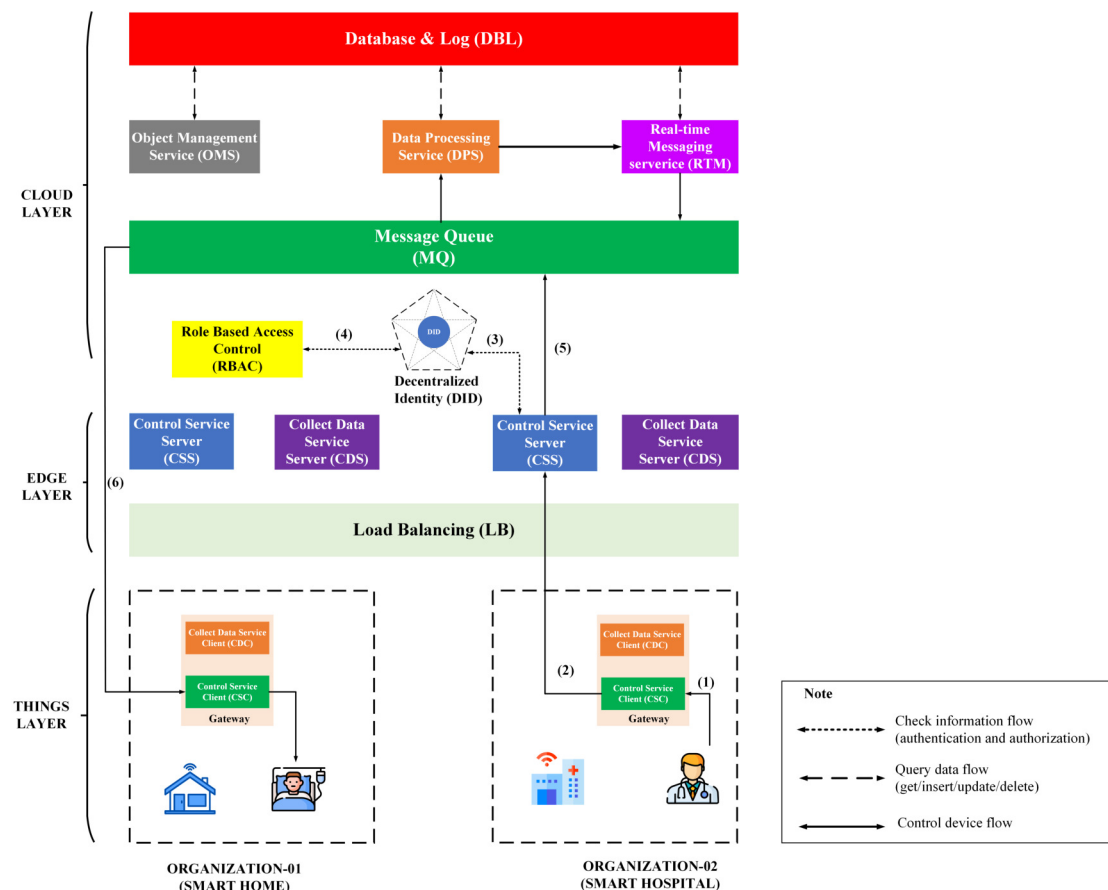
Figure 11. Overview model of the device control process through BMDD

- Creating management information of devices, communication channels, and mapping devices to channels.

- Authentication and authorization.

- Collecting data.

- Controlling device.

Detailed source code of BMDD can be found at Github link including decentralized identity service [10], collect data service [11], role-based access control and object management service [12]

Requesting and resolving DID token process

Through this process, the user can get the DID token. Before the user can send data and control the device, the DID token must be resolved to authenticate the user.

The process of requesting a DID token is presented in figure 12. The user sends his information to the DID service and receives the DID token in return.

The process of user registering has the following input and output in JSON format.

The input is

```
{
    "userName": "thanhlam",
    "serviceId": "vnptit",
```

---

[10] https://github.com/thanhlam2110/bmdd-did-service
[11] https://github.com/thanhlam2110/bmdd-collection-service
[12] https://github.com/thanhlam2110/bmdd-rbac-service
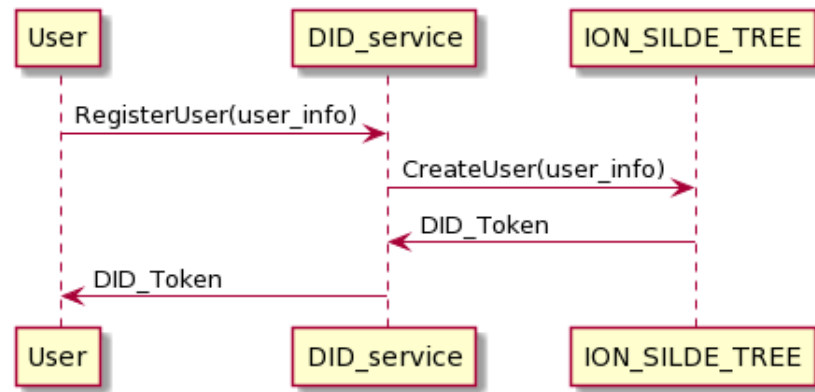
Figure 12. request DID token process

```
757        "serviceEndpoint": "vnptit"
758    }
```

And the out put is:

```
760    {
761        "longFormURI": "did:ion:EiDQx4R6z...FRaWFdETXcifX0"
762    }
```

The resolving DID token process is as the figure 13 below. The user sends DID token information to DID service. DID service will resolve the DID token and return the user information if the authentication is successful.

The process of resolving DID token has the following input and output in JSON format.

The input is:

```
768    {
769        "longFormURI": "did:ion:EiDQx4R6z...FRaWFdETXcifX0"
770    }
```

And the out put is:

```
772    {
773        "didDocument": {
774            "id": "did:ion:EiDQx4R6z...FRaWFdETXcifX0",
775            "service": [
776                {
777                    "id": "#vnptit",
778                    "type": "LinkedDomains",
779                    "serviceEndpoint": "vnptit"
780                }
781            ],
782            "authentication": [
783                "#thanhlam"
784            ]
785        }
786    }
```

Creating device, channels management information and mapping devices to channels process

Users can interact with OMS via this process to create management information of devices, channels and map devices to channels. This process helps users control their devices and communication channels. Users will clearly know where they share their data, which helps reduce security risk by user's behaviors.
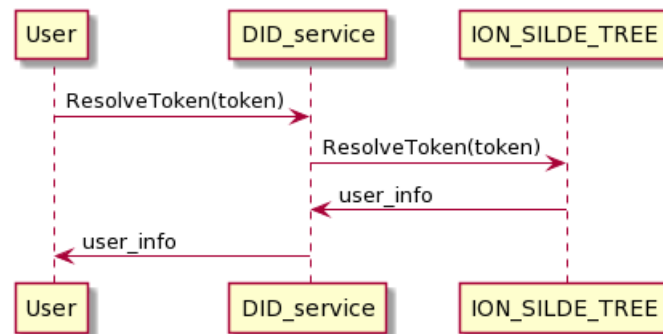
Figure 13. resolve DID token process

The process of creating *Things* management information is as the figure 14. The user sends information, including his DID token, along with device information to the OMS service. The OMS service will forward the user's DID token to the DID service for authentication. If the authentication process is successful, OMS will generate *Things* info.
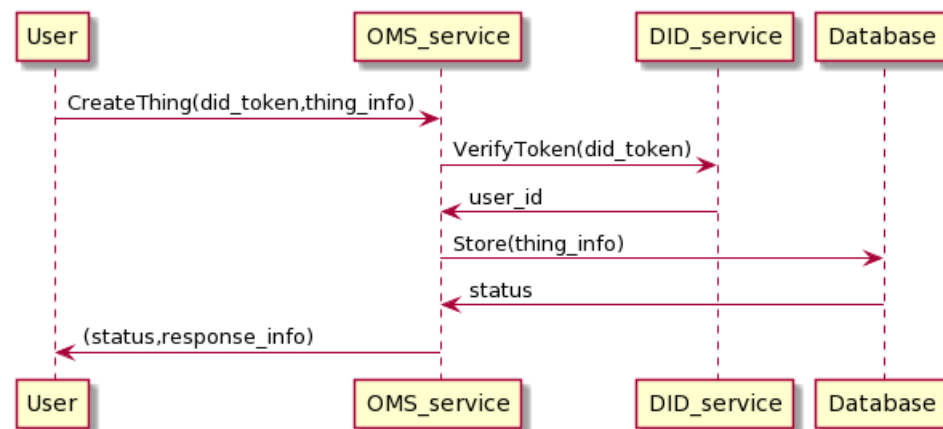
Figure 14. create thing process

The process of creating *Things* information has the following input and output in JSON format.
The input is

```
{
    "token":"user_did_token",
    "thing_name":"thing_name"
}
```

And the output is

```
{
    "thing_id":"thing_id",
    "thing_name":"thing_name",
    "thing_status":"thing_status",
    "assign_user":"user_id",
    "user_parent_id":"user_id"
}
```

OMS will automatically generate thing_id, thing_status values, where thing_id conforms to the UUID standard [13] and is a unique value. The thing_status is used for determining the device state, whose default

---

[13]https://www.ietf.org/rfc/rfc4122.txt

811 value is "ENABLE". If the status is "ENABLE", this means the Things will work properly; by contrast, it
812 can not interact with the BMDD, if the status is "DISABLE". The thing_status value can only be changed
813 by the user that owns the *Things*. Moreover, the user_id value of the user who created the *Things*, obtained
814 through the token resolution process, is equal to user_parent_id value. The "assign_user" attribute is also
815 automatically generated with the initial value of user_id. When the users assign their device to some
816 other users, this field will be added with the user_id of these assigned users, for example, when User-A
817 assigns device control to Dr-B, the field assign_user has the value as an array "assign_user":" [User_A_id,
818 Dr_B_id]".
819   The process of creating channel management information is as in the figure 15. The user sends
820 information, including his DID token, and channel information to the OMS service through which the
821 user's DID token is transferred to the DID service for authentication. If the authentication is successful,
OMS will create channel info.
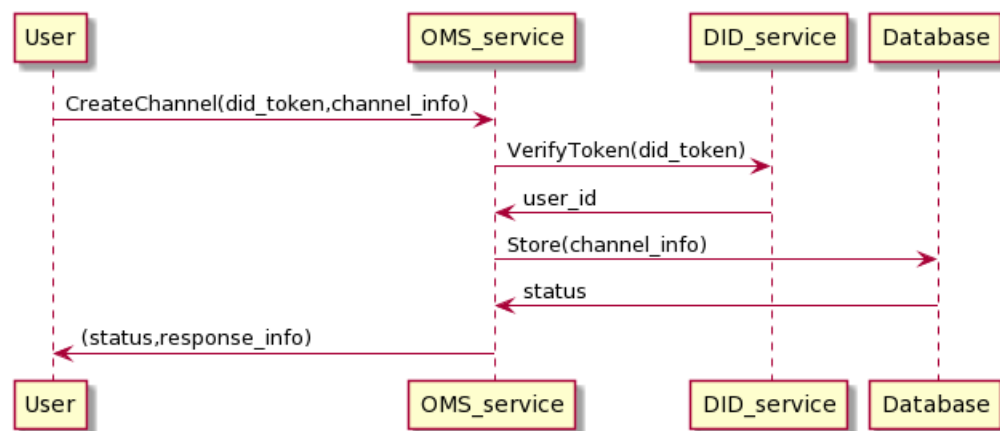
Figure 15. create channel process

822
823   The process of creating channels information has the following input and output in JSON format.
824   The input is

```
825 {
826     "token":"user_did_token",
827     "channel_name":" channel_name"
828 }
829
```

830 And the output is

```
831 {
832     "channel_name":"channel_name",
833     "channel_id":"channel_id",
834     "channel_status":"channel_status",
835     "user_parent_id":"user_id"
836 }
```

837   OMS will automatically generate channel_id, channel_status values, where channel_id conforms to the
838 UUID standard and is unique. The channel_status is used to determine the channel state, and the default
839 value is "ENABLE". If the status is "ENABLE", the channel can transport data; by contrast, when the
840 status is "DISABLE", the channel can not transmit messages to BMDD. The channel_status value can
841 only be changed by the user who owns the channel. In addition, the user_id value of the user who created
842 the channel, obtained through the token resolution process, is equal to user_parent_id value.
843   The process of mapping the *Things* into the channel is as in the figure 16.
844   The user sends information to the OMS service, including his DID token and thing_id and channel_id
845 information. The OMS service will forward the user's DID token to the DID service for authentication. If
846 the authentication is successful, OMS will map the *Things* information to the channel information.
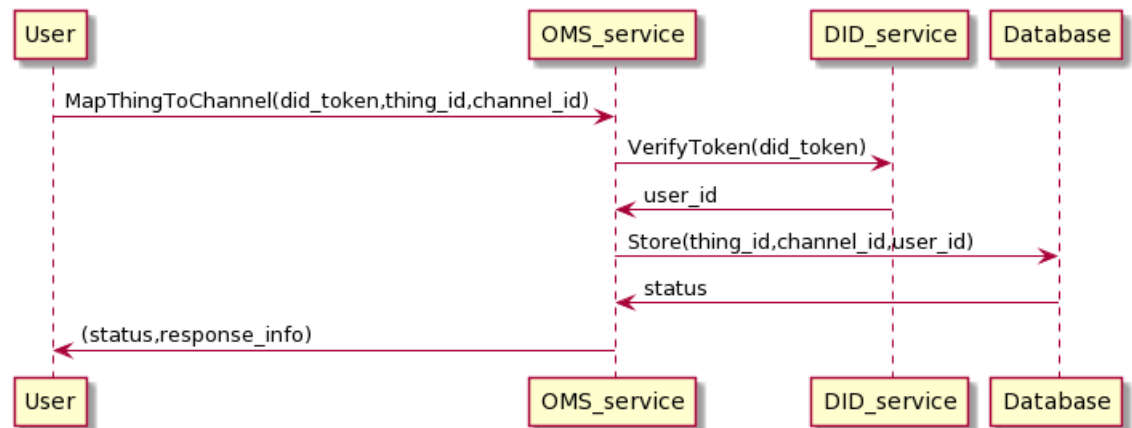
Figure 16. map thing to channel process

The process of mapping *Things* information to channels has the following input and output in JSON format.

The input is

```
{
    "token":"user_did_token",
    "thing_id":"thing_id",
    "chanel_id":"chanel_id"
}
```

And the output is

```
{
    "map_id":"map_id",
    "thing_id":"thing_id",
    "chanel_id":"chanel_id",
     "user_parent_id":"user_id"
}
```

OMS will automatically generate map_id values, which conform to the UUID standard and are unique. The user_id value of the user who maps the *Things* to the channel, obtained through the token resolution process, is equal to user_parent_id value. Note that it is possible to map multiple things on a channel.

Authentication and authorization process

The DID service is responsible for authenticating users through the DID token. The RBAC is accountable for checking the roles that the user has pre-configured through the OMS service. DID and RBAC mainly support data collection and control device processes. Therefore, we will present this process combined with two processes of collecting data (interacting with CDS) and control service (interacting with CSS).The process of interaction is as in the figure 17. Before sending data or commands to MQ, CDS and CSS must undergo authentication and check-role processes. CDS/CSS sends the user's DID token to DID authentication service; if successful, CDS/CSS will receive user_id information. Then, CDS/CSS sends information (user_id, thing_id, and channel_id) to the RBAC service to check if the user has permission to access the Things or the user has permission to the channel, or the Things is assigned to the channel. If the authentication and permission checking is successful, CDS/CSS can send data or control commands to the MQ. Otherwise, all data and control commands will be dropped.

Data collection process

The data collection is the data-sending process from CDC to CDS, going through user authentication and permission checks. Finally, data will be sent to MQ, distributed to other services, depending on the application purpose. The interaction process is as the figure 18.
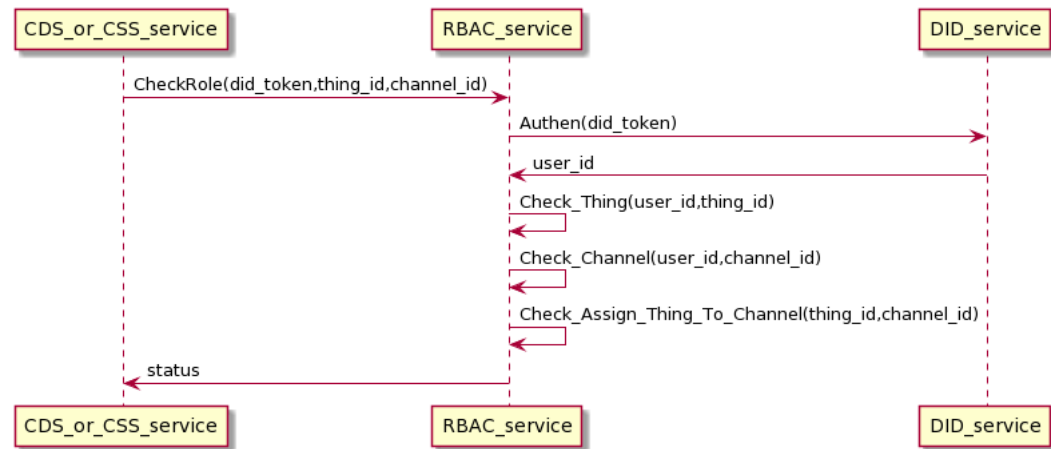
Figure 17. authentication and authorization process
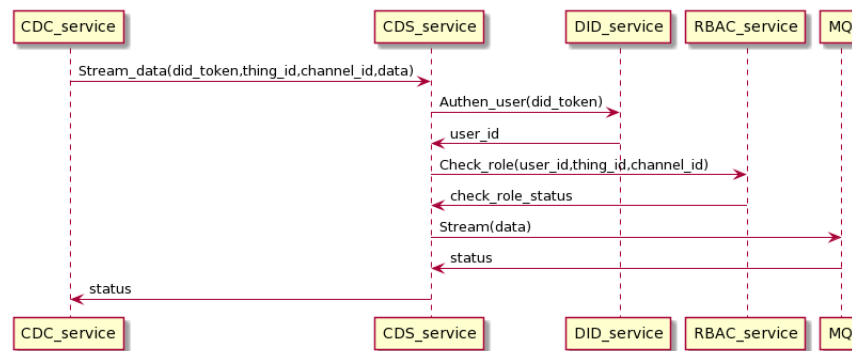
Figure 18. collect data process in detailed

CDC streams data with the users DID token, thing_id, and channel_id to CDS. CDS service receives
the data stream and interacts with the DID service and RBAC service to authenticate users and check roles.
When the authentication and permission checking is successful, the CDS will stream the data stream to
the MQ and return the status to the CDC. The data collection proceeds on the gRPC protocol.

Control device process

The control process sends control commands from CSC to CSS, going through user authentication and
permission checking. Finally, the command will be sent to the MQ, distributed to the receiving CSC to
control the device. The interaction process is as the figure 19.

The CSC sends control commands with the user's DID token, thing_id, and channel_id to CSS.
The CSS service receives control commands and interacts with the DID service and RBAC service to
authenticate users and check roles. When the authentication and permission checking is successful, the
CSS sends control commands to the MQ and returns the status to the CSC. The control device proceeds
on the gRPC protocol.

EVALUATION

After implementing BMDD, this section is for continuing to build scenarios to evaluate the performance,
load capacity, and availability of BMDD. Due to the microservice architecture-based platform of BMDD,
the services are deployed on Amazon EC2 virtual machines (VMs) in Singapore in the test scenario. Each
service is equivalent to a VM with the following hardware and software configuration as Table 2.

The above configuration is a free EC2 server configuration provided by Amazon [14]. There are

---

[14]https://aws.amazon.com/
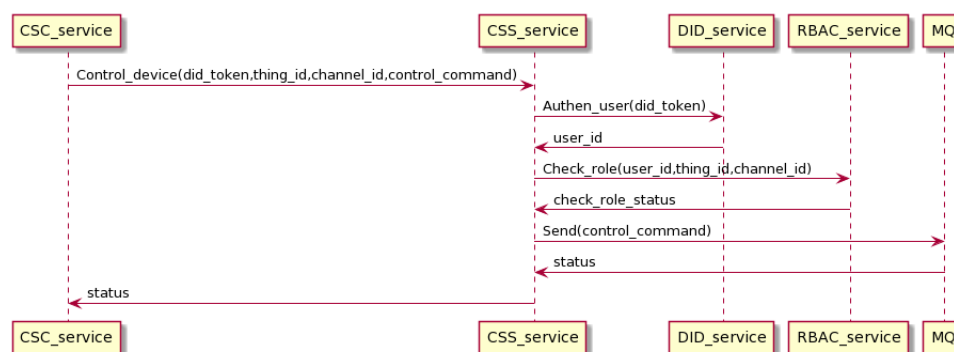
Figure 19. control device process

CPU	1 vCPU
RAM	1GB
Disk	8GB SSD
OS	Ubuntu 16.04

Table 2. VM EC2 configuration.

many famous companies have deployed their service on Amazon EC2 [15], so we chose EC2 to deploy the **cloud layer**. However, in this paper, we only use Amazon EC2 for implementing the proof-of-concept of the BMDD platform. Our purpose is to prove BMDD can be implemented, microservices can communicate with each other, and test performance. The deployments in production are heavily technical for example, using Docker [16] and Jenkins [17] pipeline for CI/CD deployment , Kubernetes [18] for microservices management , ELK [19] for logging , etc., beyond the scope of the article. These parts are completely possible because we have provided the source code of BMDD.

In addition, Table 3 indicates the hardware configuration and the software of Raspberry Pi module B applied [20] to emulate the Gateway device.

CPU	Broadcom BCM2837, ARMv8 (64bit) quad-core, 1.2 Ghz
RAM	1GB
Disk	2GB SSD
OS	Raspbian

Table 3. Configuration of Raspberry Pi module B

We use the Raspberry Pi module because of its popularity and affordable price. Raspberry Pi module is also used for the gateway in many papers (20; 51; 59). Moreover, the Raspberry Pi has built-in Wifi and Bluetooth function which is suitable for local communication (LAN network) between gateway and IoT devices. However, as described in the **BMDD platform proposal** section, the gateway includes only two gRPC services (CDC and CSC) which can be implemented by many programming languages such as C/C++, Go, Python, etc.[21]; so we can use other microcontrollers for the gateway. Apache Jmeter software was used to simulate the number of concurrent users (CCUs) interacting with the BMDD during the evaluation. Apache Jmeter [22] is an open-source, completely written in Java, utilised to test efficiency on both static assets, dynamic assets, and Web applications. It can mimic multiple virtual users, broad requests on a server, or a gathering of servers, networks, or objects to examine tolerance load tests or

---

[15]https://aws.amazon.com/ec2/customers/

[16]https://www.docker.com/

[17]https://www.jenkins.io/

[18]https://kubernetes.io/

[19]https://www.elastic.co/what-is/elk-stack

[20]https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

[21]https://grpc.io/

[22]https://jmeter.apache.org/

919 investigate response time.When using Jmeter to test the system load, at first, Jmeter creates requests
920 and transmits them to the server in accordance with a predefined operation. After taking and collecting
921 responses from the server, it presents the report data. Jmeter has numerous report factors but focuses on
922 *throughput* and *error* when operating system load testing. *Throughput* (request/s) is the request quantities
923 resolved by the server every second, and *error* (%) is the level of failed requests out of the total number of
924 requests, wherein *Error* is defined as an overloaded service condition leading to the service being idle or
925 unresponsive.

Scenario 1: The load test scenario of data collecting service

927 The first scenario aims to confirm the highest number of users and transmit data (Concurrent User -
928 CCU) to the BMDD without the system error. This situation compares the performance between two
929 architecturesystems, including broker-less systems using gRPC protocol (BMDD) and brokering systems
930 using MQTT protocol. The test model is illustrated in Figure 20.
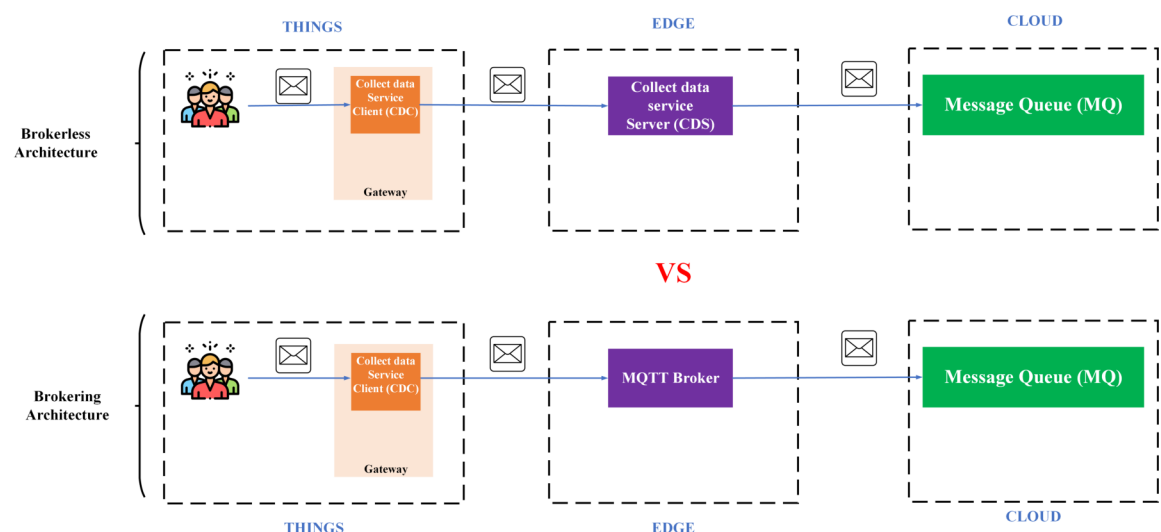
Figure 20. Model of load test collect data process

931 All edge and cloud layer services are deployed on the EC2 VM with the configuration **shown in table**
932 **2**; only the things layer components (CDC and CSC) are deployed on the Raspberry Pi module as Figure
933 21.

Figure 21. Gateway deployment on Raspberry Pi module

934 We set up a data collection model at the things layer, data transmission at the edge layer, and data
935 reception at the cloud layer. In this scenario, we assume the user is authenticated and has a valid role. We
936 simulate the number of concurrent users streaming incoming messages using Jmeter software. Then, we
937 gradually increased the number of CCUs and recorded the highest number of CCUs that did not generate
938 errors for both brokering and broker-less architectures. The test results are shown in figure 22 and figure
23 , respectively.
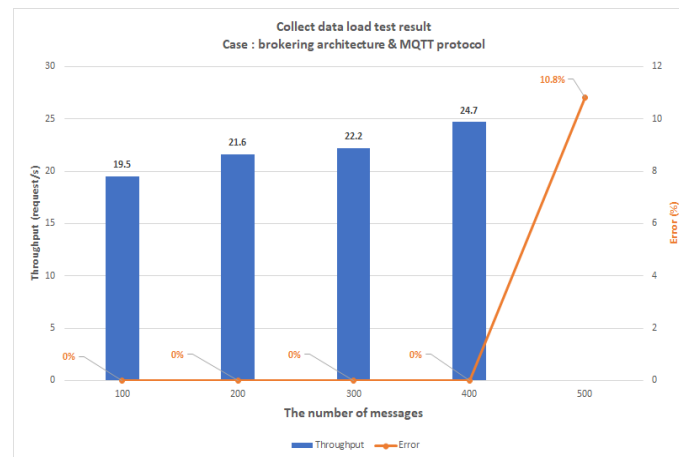
Figure 22. The collect data load test result – brokering architecture & MQTT protocol
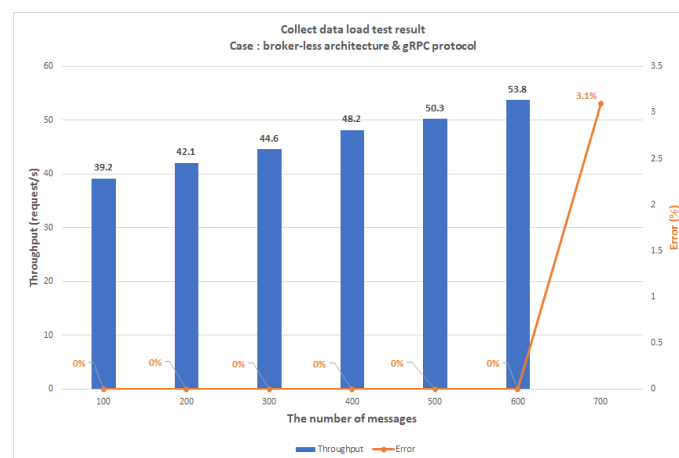
Figure 23. The collect data load test result – broker-less architecture & gRPC protocol

939
940 The test results illustrated that the broker-less architecture using the gRPC protocol provides two times
941 higher throughput than the brokering architecture using the MQTT protocol. In addition, the broker-less
942 architecture also provides better load capacity (700 CCUs) compared to brokering architecture (500
943 CCUs).

944 **Scenario 2: The speed test scenario of streaming data**
945 The purpose of this scenario is to measure the message transfer rate. We do not change the number of
946 users and use Jmeter to gradually increase the number of messages in one transmission and record the
947 Round Trip Time (RTT) from when the message is sent at CDC to when it is received in MQ.
948 In this case, we also compare the transfer rate between broker-less architecture using gRPC protocol
949 and brokering architecture using MQTT protocol. We also record the highest number of messages
950 successfully delivered in a single communication without errors. Error is defined as the percentage
951 of messages sent to the destination out of the total number of messages sent. Test model and service

952 deployment environment at layers are similar to scenario 1. The test results are shown in figure 24 and
figure 25 , respectively.
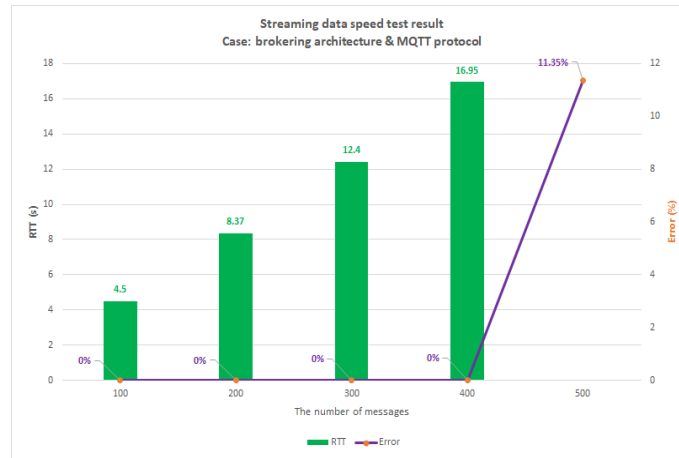
Figure 24. The streaming data speed test results – brokering architecture & MQTT protocol
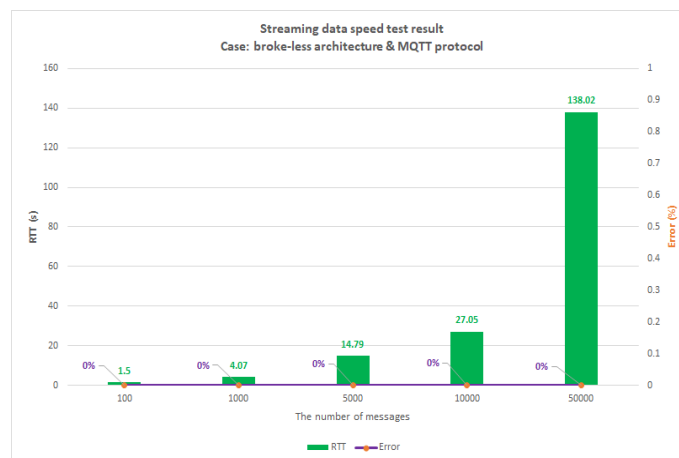
Figure 25. The streaming data speed test results – broker-less architecture & gRPC protocol

953
954   The test results illustrate that the broker-less architecture using the gRPC protocol provides powerful
955 message streaming capabilities compared to the brokering architecture using the MQTT protocol. In
956 addition, in this scenario, we also use a htop [23] software that allows monitoring processes taking place in
957 the system in real-time to evaluate CPU and RAM usage for two services CDC and CDS of broker-less
958 architecture. The test results are shown in Table 4.

Messages		No load	10000	50000	100000
CDC	CPU	0	0	0.3%	0.7%
(on Raspberry Pi)	RAM	0	0	2.5%	2.7%
CDS	CPU	0	2.3%	8.2%	10.5%
(on VM EC2)	RAM	1.7%	1.7%	1.9%	2.1%

Table 4. CPU/RAM consumption when streaming data – broker-less architecture & gRPC protocol

959   As a result, at no-load and 10000 messages, we recorded a consumption level of 0, which means
960 that the resource consumption of the services is very low (no-load) or happens for a brief time (10000

---
[23]https://htop.dev/

961 messages), so the meter almost does not record the change in resource usage. The results show that the
962 resource consumption is very low for the high load case (50000-10000 messages). The fact that *Things* is
963 not constantly streaming large amounts of data like in this test shows that applying gRPC protocol to our
964 BMDD gives very good results, suitable for devices with low hardware.

965 **Scenario 3: The speed test scenario of authentication mechanism**
966 This scenario compares user authentication speed between two authentication models, including central-
967 ized authentication using Single Sign-On (SSO) and decentralized authentication using decentralized
968 identity (DID). The test model is illustrated as in Figure 26.
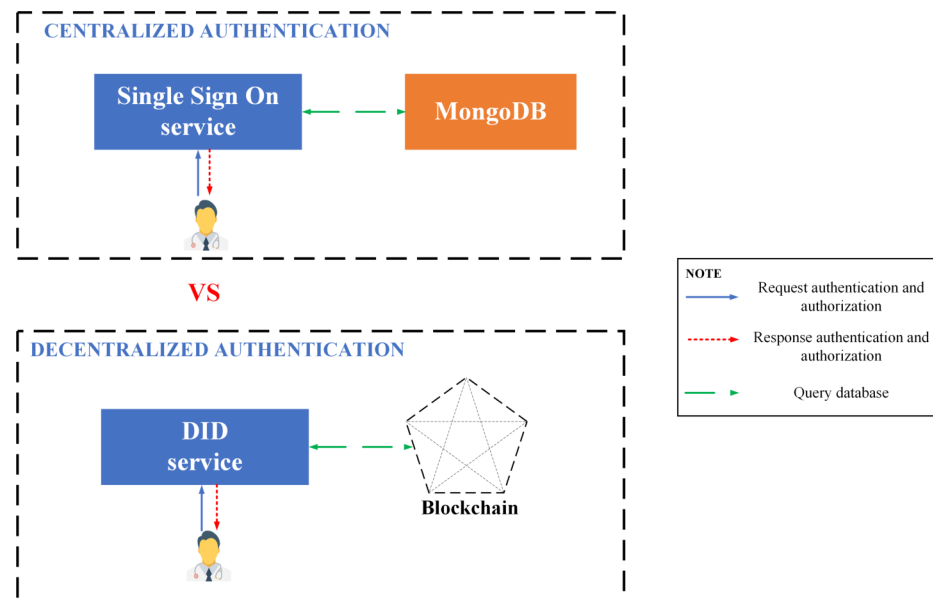
Figure 26. Model speed test authentication process

969   Both the Single Sign-On and the DID service are deployed on the EC2 VM with the configuration
970 shown in table 2. For the Single Sign-On service, we use open source CAS Apereo [24] and Oauth protocol.
971 Both centralized and decentralized authentication models go through two steps. The first step is to request
972 a token, and the user will use this token to authenticate. For centralized authentication, SSO will return
973 the Oauth token, and for decentralized authentication, DID will return DID token. The second step is
974 token validation process, also known as token resolve. In this test, we use Jmeter to simulate the number
975 of concurrent users (CCUs) requesting tokens and then resolving the received tokens. We will record the
976 highest CCU level without the system failing and two parameters throughput and error. The test results
977 are shown in the table below.

978   1. Centralized authentication model using SSO (table 5, 6)

Request token (CCU)	100	200	300	400	500
Throughput (request/s)	13.9	23.1	26.2	31.6	-
Error (%)	0%	0%	0%	0%	13.3%

Table 5. The test request token results - centralized authentication using SSO

979   2. Decentralized authentication model using DID (table 7, 8)

980 **Scenario 4: The feature test scenario of broker-less and microservice architecture**
981 The purpose of this scenario is to check the system availability. Theoretically, BMDD is designed
982 according to broker-less and microservice architecture, so when one service fails, other services still work.
983 By contrast, the entire system of brokering architecture using the MQTT protocol will stop working when
984 the MQTT broker fails. We design the test model as Figure 27 below to test in practice.

---

[24]https://www.apereo.org/projects/cas

Resolve token (CCU)	100	200	300	400	500
Throughput (request/s)	22.3	22.6	32.2	34.7	-
Error (%)	0%	0%	0%	0%	9.8%

Table 6. The test resolve token results - centralized authentication using SSO

Request token (CCU)	100	200	300	400	500	600	1000	2000
Throughput (request/s)	93.8	96.2	136.4	185.9	196	162.6	188.8	191.6
Error (%)	0%	0%	0%	0%	0%	0%	0%	0%

Table 7. The test request token results - decentralized authentication using DID

Resolve token (CCU)	100	200	300	400	500	600	1000	2000
Throughput	11.9	41.8	61.1	103.6	107.3	109.7	138.2	-
Error (%)	0%	0%	0%	0%	0%	0%	5%	-

Table 8. The test resolve token results - decentralized authentication using DID

We test the scenario where the user simultaneously sends data and controls the device. At the same time, we turned off the control service server (CSS) and the MQTT broker. As a result, BMDD still allows users to transfer data, but device control will be interrupted. Meanwhile, the entire system following the brokering architecture stops working when the MQTT broker is turned off. The results show that BMDD provides higher availability than brokering architectures.

Scenario 5: The feature test scenario of dynamic message transmission

The purpose of this script is to test dynamic message capabilities. As shown in the background, dynamic messaging allows users to communicate with any device, regardless of the manufacturer, because BMDD provides very high homogeneity; for example, the gateway only has two services: controlled service client (CSC) and collect data service client (CDC), whether applying BMDD to any field of IoT.

In the test mentioned in the first four scenarios, we have deployed the gateway on the Raspberry Pi module with two services: CSC and CDC, programmed by Go language. In addition, we found that gRPC can deploy on other modules, for example, ESP8266 [25] with many other programming languages such as C/C++, python, etc. Thus, the implementation of CSC and CDC with various programming languages and hardware modules is perfectly possible.

This ability allows us to easily provide dynamic messaging when deploying BMDD. This result brings many benefits to customers, as depicted in figure 28. A user using the smart health service deployed on the BMDD platform in Vietnam while travelling to the US can continue to use the service without interruption even though different companies supply the devices in Vietnam and the US. This trait is one of the strong points of BMDD regarding uniformity.

Scenario 6: Security analysis

For the security issue, we analysis the three aspects, namely the encrypted-format packet, authentication, authorization.

In the first issue, we collect the message that is transfer among the layers. To capture the transmitted message, we use Wireshark [26] is network packet analyzer software. Its job is to capture all network packets and then display the its content. In the encrypted message analysis, we compare the security mechanism between MQTT and gRPC protocol via wireshark tool (see figure 29).

The analyzed packet's content of MQTT and gRPC protocol is shown in figure 30 and figure 31 , respectively.

According to the test result, MQTT protocol easily gets topic information as well as message content (figure 30); for instance the MQTT topic is *"thanh-lam"* and message is *"Hello Thanh Lam"*. Whereas, the gRPC protocol provides packet encryption (figure 31). This result, therefore, proves that the gRPC protocol has a better security mechanism than the MQTT protocol.

---

[25]https://medium.com/grpc/efficient-iot-with-the-esp8266-protocol-buffers-grafana-go-and-kubernetes-a2ae214dbd29
[26]https://www.wireshark.org
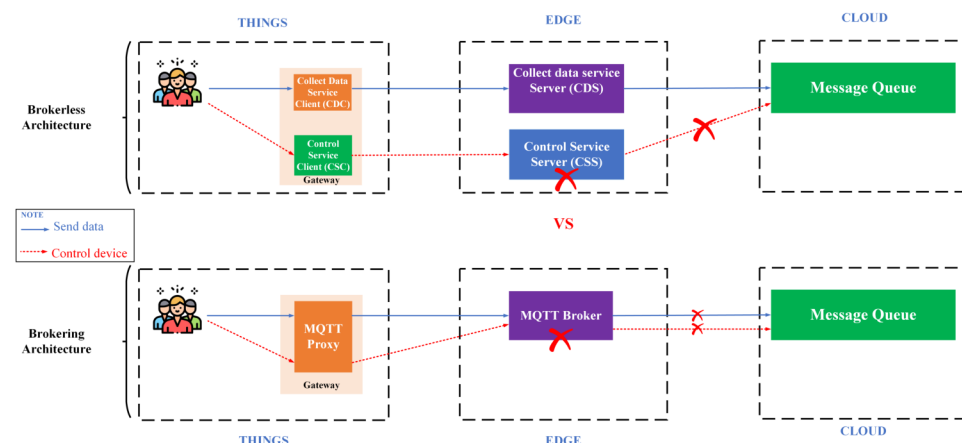
Figure 27. Broker-less and microservice scenario test model
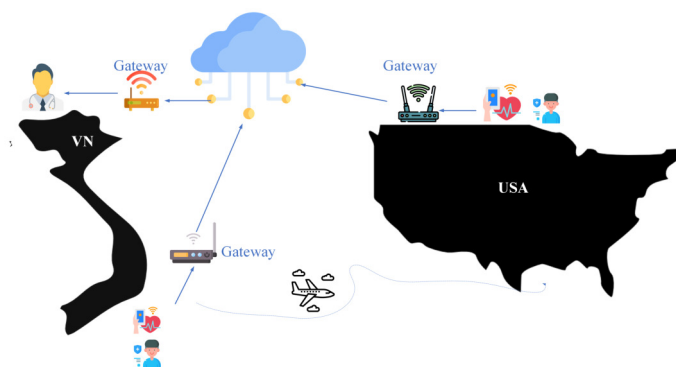
Figure 28. Dynamic transmission message scenario test model

For the second aspect, we also apply Decentralized identifiers (DID) via Microsoft ION platform to design the BMDD architecture. Thanks to ION DID, BMDD identifies data objects to perform the identification separating from a third party identifier. Due to URIs, DID establishes the association between a subject with a document to build up reliable interactions, which also allowed control of the verification without testifying with a third party (see the detail in implementation section).

For the authorization, we apply the hierrachy tree to allow the user who have a full control over their data and devices. In particular, they can grant and revoke permission to access the generated data. The permission includes insert, edit, delete and so on. Moreover, they can create an unlimited level via the hierarchy tree structure. Moreover, we also apply the role-based access control in the BMDD platform. In particular, the user can manage who can access their data via the authorized permission.

DISCUSSION

The BMDD has 5 contributions as mentioned in the **Introduction** section. Those five contributions are demonstrated through the 5 test scenarios mentioned in the **Evaluation** section.

For **test scenario 1**, the BMDD provides better load capacity than systems using MQTT (45). According to the test result (image), with the same gateway and VMs configuration, the BMDD can handle 600 CCUs without error while systems using MQTT only have a maximum of 400 CCUs. Moreover, the BMDD has a throughput 2 times higher than the system using MQTT.

For **test scenario 2**, the BMDD provides faster transmission speed and higher reliability than systems using MQTT (44). According to the test result (image), with the same gateway and VMs configuration, the BMDD can stream 50000 messages without error while the maximum of the system using MQTT is 400 messages.

For **test scenario 3**, the BMDD provides a decentralized authentication mechanism via DID ION which
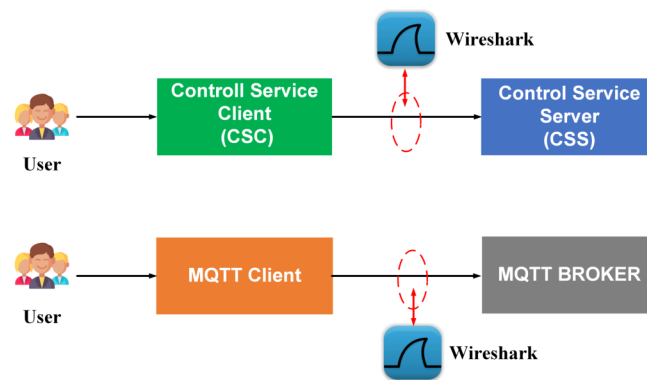
Figure 29. Capture transmission message using Wireshark
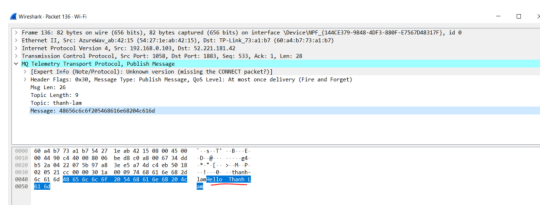
Figure 30. MQTT protocol message capture
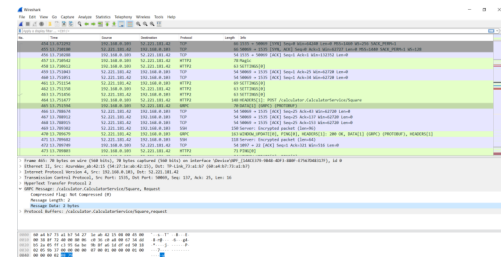
Figure 31. gRPC protocol message capture

is based on blockchain. The decentralized authentication mechanism is suitable for the natural properties of IoT devices (mobility and distribution) (56; 53). There have been many studies on authentication mechanisms for IoT before, but mainly centralized authentication (82; 28).

For **test scenario 4**, the BMDD is designed based on broker-less and microservice architecture. This allows BMDD has no single-failure point (42). In the test result, BMDD still works normally when one service is turned down while the system using MQTT depends on the central broker.

For **test scenario 5**, the BMDD provides dynamic message transmission mechanism that allows no distinction between data types so there is also no distinction between gateway manufacturers. For the best of our knowledge, this is a new feature, we have not found any IoT Platform that provides this capability.

CONCLUSION

This paper proposed the IoT platform called BMDD, a combination of broker-less and microservice architecture. These two types of architecture allow BMDD to have high availability reduce single-point failure and scalability. Moreover, BMDD provides decentralized authentication suitable for the decentralized character of IoT devices. Furthermore, thanks to the gRPC protocol benefit, BMDD have an excellent performance as well as low power consumption. Besides, this protocol supports dynamic message transmission, which brings uniformity to IoT devices. In addition, BMDD provides logical features related to authorization, devices, and channels via decentralized identity architecture. Moreover, message queues via Kafka enhance system availability to reduce packet loss when services fail. This paper also performed the six scenarios to evaluate our above contributes in the experiment section.

For future work, we plan to improve the performance, i.e., increase the and decrease the error even the massive of request token environment. The current performance can be executed in 500 CCU. For the security issues, the addition of intelligent filtering is potentially the next aim, using AI technology to limit the risk of users intentionally or being exploited to spread viruses on IoT platforms. For the traditional Kafka platform, we also plan to apply the fully homomorphic encryption (FHE) to execute the privacy-preserving computation to prevent the privacy violation behavior in the message queue. Moreover, we plan to apply the attribute-based access control (ABAC) to improve the validate process where the policy can be define in the fine-grained level.

REFERENCES

[mav] Maven. https://maven.apache.org/. Accessed: 2021-10-12.

[mic] Micro-mu. https://micro.mu/. Accessed: 2021-10-12.

[OPS] Ops4j pax url. https://ops4j1.jira.com/wiki/spaces/paxurl/overview. Accessed: 2021-10-12.

[OSG] Osgi event admin service. https://docs.osgi.org/specification/osgi.cmpn/7.0.0/service.event.html. Accessed: 2021-10-12.

[Sen] Senseioty. https://flairbit.com/senseioty-iiot-solution/. Accessed: 2021-10-12.

[6] Alguliyev, R., Imamverdiyev, Y., and Sukhostat, L. (2018). Cyber-physical systems and their security issues. *Computers in Industry*, 100:212–223.

[7] Anthraper, J. J. and Kotak, J. (2019). 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*.

[8] Anusree, P. and Sreedhar, S. (2015). A security framework for brokerless publish subscribe system using identity based signcryption. In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, pages 1–5. IEEE.

[9] Archana, E., Rajeev, A., Kuruvila, A., Narayankutty, R., and Kannimoola, J. M. (2020). A formal modeling approach for qos in mqtt protocol. In *Data Communication and Networks*, pages 39–57. Springer.

[10] Asghari, P., Rahmani, A. M., and Javadi, H. H. S. (2019). Internet of things applications: A systematic review. *Computer Networks*, 148:241–261.

[11] ATEŞ, E. C., BOSTANCI, E., and GÜZEL, M. S. (2020). Security evaluation of industry 4.0: Understanding industry 4.0 on the basis of crime, big data, internet of thing (iot) and cyber physical systems. *Güvenlik Bilimleri Dergisi*, (International Security Congress Special Issue):29–50.

[12] Badii, C., Bellini, P., Difino, A., Nesi, P., Pantaleo, G., and Paolucci, M. (2019). Microservices suite for smart city applications. *Sensors*, 19(21):4798.

[13] Bagheri, B., Yang, S., Kao, H.-A., and Lee, J. (2015). Cyber-physical systems architecture for self-aware machines in industry 4.0 environment. *IFAC-PapersOnLine*, 48(3):1622–1627.

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

[15] Bixio, L., Delzanno, G., Rebora, S., and Rulli, M. (2020). A flexible iot stream processing architecture based on microservices. *Information*, 11(12):565.

[16] Boyes, H., Hallaq, B., Cunningham, J., and Watson, T. (2018). The industrial internet of things (iiot): An analysis framework. *Computers in industry*, 101:1–12.

[17] Chenaru, O., Stamatescu, G., Stamatescu, I., and Popescu, D. (2015). Towards cloud integration for industrial wireless sensor network systems. In *2015 9th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, pages 917–922. IEEE.

[18] Cheng, G.-J., Liu, L.-T., Qiang, X.-J., and Liu, Y. (2016). Industry 4.0 development and application of intelligent manufacturing. In *2016 international conference on information system and artificial intelligence (ISAI)*, pages 407–410. IEEE.

[19] Dahlqvist, F., Patel, M., Rajko, A., and Shulman, J. (2019). Growing opportunities in the internet of things. *McKinsey, July*.

[20] Daidone, F., Carminati, B., and Ferrari, E. (2021). Blockchain-based privacy enforcement in the iot domain. *IEEE Transactions on Dependable and Secure Computing*.

[21] der Technikwissenschaften, D. A. (2011). *Cyber-Physical Systems: Driving force for innovation in mobility, health, energy and production*. Springer.

[22] Di Crescenzo, G., Burns, J., Coan, B., Schultz, J., Stanton, J., Tsang, S., and Wright, R. N. (2013). Efficient and private three-party publish/subscribe. In *International Conference on Network and System Security*, pages 278–292. Springer.

[23] Di Martino, B., Rak, M., Ficco, M., Esposito, A., Maisto, S., and Nacchia, S. (2018). Internet of things reference architectures, security and interoperability: A survey. *Internet of Things*, 1-2:99–112.

[24] Du, S. G., Lee, J. W., and Kim, K. (2018). Proposal of grpc as a new northbound api for application layer communication efficiency in sdn. In *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication*, pages 1–6.

[25] Duong-Trung, N., Son, H. X., Le, H. T., and Phan, T. T. (2020a). On components of a patient-centered healthcare system using smart contract. In *Proceedings of the 2020 4th International Conference on*

1122   *Cryptography, Security and Privacy*, page 31–35, New York, NY, USA. Association for Computing
1123   Machinery.

1124   [26]   Duong-Trung, N., Son, H. X., Le, H. T., and Phan, T. T. (2020b). Smart care: Integrating blockchain
1125   technology into the design of patient-centered healthcare systems. In *Proceedings of the 2020 4th*
1126   *International Conference on Cryptography, Security and Privacy*, ICCSP 2020, page 105–109, New
1127   York, NY, USA. Association for Computing Machinery.

1128   [27]   Fang, W., Zhang, W., Chen, W., Pan, T., Ni, Y., and Yang, Y. (2020). Trust-based attack and defense
1129   in wireless sensor networks: a survey. *Wireless Communications and Mobile Computing*, 2020.

1130   [28]   Fremantle, P. and Aziz, B. (2019). Deriving event data sharing in iot systems using formal modelling
1131   and analysis. *Internet of things*, 8:100092.

1132   [29]   Fremantle, P., Aziz, B., Kopeckỳ, J., and Scott, P. (2014). Federated identity and access management
1133   for the internet of things. In *2014 International Workshop on Secure Internet of Things*, pages 10–17.
1134   IEEE.

1135   [30]   Fuentes Carranza, J. C. and Fong, P. W. (2019). Brokering policies and execution monitors for iot
1136   middleware. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*,
1137   pages 49–60.

1138   [31]   Gheisari, M., Wang, G., and Chen, S. (2020). An edge computing-enhanced internet of things
1139   framework for privacy-preserving in smart city. *Computers & Electrical Engineering*, 81:106504.

1140   [32]   Gos, K. and Zabierowski, W. (2020). The comparison of microservice and monolithic architecture.
1141   In *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS*
1142   *Design (MEMSTECH)*, pages 150–153. IEEE.

1143   [33]   Ha, X. S., Le, H. T., Metoui, N., and Duong-Trung, N. (2020a). Dem-cod: Novel access-control-
1144   based cash on delivery mechanism for decentralized marketplace. In *2020 IEEE 19th International*
1145   *Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages
1146   71–78. IEEE.

1147   [34]   Ha, X. S., Le, T. H., Phan, T. T., Nguyen, H. H. D., Vo, H. K., and Duong-Trung, N. (2020b).
1148   Scrutinizing trust and transparency in cash on delivery systems. In *International Conference on Security,*
1149   *Privacy and Anonymity in Computation, Communication and Storage*, pages 214–227. Springer.

1150   [35]   Hermann, M., Pentek, T., and Otto, B. (2016). Design principles for industrie 4.0 scenarios. In *2016*
1151   *49th Hawaii international conference on system sciences (HICSS)*, pages 3928–3937. IEEE.

1152   [36]   Hong, S. (2020). P2p networking based internet of things (iot) sensor node authentication by
1153   blockchain. *Peer-to-Peer Networking and Applications*, 13(2):579–589.

1154   [37]   Hossein Motlagh, N., Mohammadrezaei, M., Hunt, J., and Zakeri, B. (2020). Internet of things (iot)
1155   and the energy sector. *Energies*, 13(2):494.

1156   [38]   Indrasiri, K. and Kuruppu, D. (2020). *gRPC: up and running: building cloud native applications*
1157   *with Go and Java for Docker and Kubernetes*. ” O’Reilly Media, Inc.”.

1158   [39]   Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L.
1159   (2009). Networking named content. In *Proceedings of the 5th international conference on Emerging*
1160   *networking experiments and technologies*, pages 1–12.

1161   [40]   Jain, S. K., Kesswani, N., and Agarwal, B. (2020). Security, privacy and trust: privacy preserving
1162   model for internet of things. *International Journal of Intelligent Information and Database Systems*,
1163   13(2-4):249–277.

1164   [41]   Karcher, N., Gebauer, R., Bauknecht, R., Illichmann, R., and Sander, O. (2020). Versatile configura-
1165   tion and control framework for real time data acquisition systems. *arXiv preprint arXiv:2011.00112*.

1166   [42]   Kawaguchi, R. and Bandai, M. (2020). Edge based mqtt broker architecture for geographical iot
1167   applications. In *2020 International Conference on Information Networking (ICOIN)*, pages 232–235.
1168   IEEE.

1169   [43]   Khan, M. U., Li, S., Wang, Q., and Shao, Z. (2016). Cps oriented control design for networked
1170   surveillance robots with multiple physical constraints. *IEEE Transactions on Computer-Aided Design*
1171   *of Integrated Circuits and Systems*, 35(5):778–791.

1172   [44]   Lam Nguyen Tran Thanh, The Anh Nguyen, H. K. V. H. H. L. K. N. H. T. T. D. A. H. X. S. (2021a).
1173   Toward a security iot platform with high rate transmission and low energy consumption. In *International*
1174   *Conference on Computational Science and its Applications*. Springer.

1175   [45]   Lam Nguyen Tran Thanh, The Anh Nguyen, H. K. V. H. H. L. K. N. H. T. T. D. A. H. X. S.
1176   (2021b). Toward a unique iot network via single sign-on protocol and message queue. In *International*

Conference on Computer Information Systems and Industrial Management. Springer.

[46] Lee, E. A. (2006). Cyber-physical systems-are computing foundations adequate. In *Position paper for NSF workshop on cyber-physical systems: research motivation, techniques and roadmap*, volume 2, pages 1–9. Citeseer.

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

[48] Lewandowski, T., Henze, D., Sauer, M., Nickles, J., and Bruegge, B. (2020). A software architecture to enable self-organizing, collaborative iot ressource networks. In *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 70–77. IEEE.

[49] Liu, Z., Wang, Z., Ren, Y., Feng, Q., Fan, D., and Zuo, Z. (2018). A city medical resources distribution optimization platform based on cyber physical systems (cps). In *2018 international conference on sensing, diagnostics, prognostics, and control (SDPC)*, pages 269–273. IEEE.

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

[51] Luchian, R.-A., Stamatescu, G., Stamatescu, I., Fagarasan, I., and Popescu, D. (2021). Iiot decentralized system monitoring for smart industry applications. In *2021 29th Mediterranean Conference on Control and Automation (MED)*, pages 1161–1166. IEEE.

[52] Mihai, V., Dragana, C., Stamatescu, G., Popescu, D., and Ichim, L. (2018). Wireless sensor network architecture based on fog computing. In *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 743–747. IEEE.

[53] Mishra, P., Puthal, D., Tiwary, M., and Mohanty, S. P. (2019). Software defined iot systems: Properties, state of the art, and future research. *IEEE Wireless Communications*, 26(6):64–71.

[54] Möller, D. P. and Vakilzadian, H. (2016). Cyber-physical systems in smart transportation. In *2016 IEEE international conference on electro information technology (EIT)*, pages 0776–0781. IEEE.

[55] Noura, M., Atiquzzaman, M., and Gaedke, M. (2019). Interoperability in internet of things: Taxonomies and open challenges. *Mobile Networks and Applications*, 24(3):796–809.

[56] Pahl, M.-O. and Liebald, S. (2019). A modular distributed iot service discovery. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 448–454. IEEE.

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

[58] Pillai, A. S., Chandraprasad, G. S., Khwaja, A. S., and Anpalagan, A. (2019). A service oriented iot architecture for disaster preparedness and forecasting system. *Internet of Things*, page 100076.

[59] Pratama, Y. P., Basuki, D. K., Sukaridhoto, S., Yusuf, A. A., Yulianus, H., Faruq, F., and Putra, F. B. (2019). Designing of a smart collar for dairy cow behavior monitoring with application monitoring in microservices and internet of things-based systems. In *2019 International Electronics Symposium (IES)*, pages 527–533. IEEE.

[60] Pratap, A., Gupta, R., Nadendla, V. S. S., and Das, S. K. (2019). On maximizing task throughput in iot-enabled 5g networks under latency and bandwidth constraints. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 217–224. IEEE.

[61] Quirós, G., Cao, D., and Canedo, A. (2020). Dispersed automation for industrial internet of things. *IEEE Transactions on Automation Science and Engineering*, 17(3):1176–1181.

[62] Ramachandran, G. S., Wright, K.-L., and Krishnamachari, B. (2018). Trinity: A distributed publish/subscribe broker with blockchain-based immutability. *arXiv preprint arXiv:1807.03110*.

[63] Rashid, M., Parah, S. A., Wani, A. R., and Gupta, S. K. (2020). Securing e-health iot data on cloud systems using novel extended role based access control model. In *Internet of Things (IoT)*, pages 473–489. Springer.

[64] Saadallah, B., Lahmadi, A., and Festor, O. (2012). *CCNx for Contiki: implementation details*. PhD thesis, INRIA.

[65] Sajjad, S. M., Yousaf, M., Afzal, H., and Mufti, M. R. (2020). emud: Enhanced manufacturer usage description for iot botnets prevention on home wifi routers. *IEEE Access*, 8:164200–164213.

[66] Sathyaseelan, M. P., Chakravarthi, M. K., Sathyaseelan, A. P., and Sudipta, S. (2021). Iot based covid de-escalation system using bluetooth low level energy. In *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pages 174–177. IEEE.

[67] Seitz, A., Henze, D., Miehle, D., Bruegge, B., Nickles, J., and Sauer, M. (2018a). Fog computing as enabler for blockchain-based iiot app marketplaces-a case study. In *2018 Fifth international conference on internet of things: systems, management and security*, pages 182–188. IEEE.

[68] Seitz, A., Henze, D., Nickles, J., Sauer, M., and Bruegge, B. (2018b). Augmenting the industrial internet of things with emojis. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 240–245. IEEE.

[69] Shi, W., Kumar, N., Gong, P., Chilamkurti, N., and Chang, H. (2015). On the security of a certificateless online/offline signcryption for internet of things. *Peer-to-Peer Networking and Applications*, 8(5):881–885.

[70] Shih, C.-S., Chou, J.-J., Reijers, N., and Kuo, T.-W. (2016). Designing cps/iot applications for smart buildings and cities. *IET Cyber-Physical Systems: Theory & Applications*, 1(1):3–12.

[71] Simeoni, E., Gaeta, E., García-Betances, R. I., Raggett, D., Medrano-Gil, A. M., Carvajal-Flores, D. F., Fico, G., Cabrera-Umpiérrez, M. F., and Arredondo Waldmeyer, M. T. (2021). A secure and scalable smart home gateway to bridge technology fragmentation. *Sensors*, 21(11):3587.

[72] Smith, S. (2018). Overview of asp .net core mvc. *Microsoft. Last modified January*, 7.

[73] Son, H. X. and Chen, E. (2019). Towards a fine-grained access control mechanism for privacy protection and policy conflict resolution. *International Journal of Advanced Computer Science and Applications*, 10(2).

[74] Son, H. X., Dang, T. K., and Massacci, F. (2017). 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*, pages 501–515. Springer.

[75] Soni, D. and Makwana, A. (2017). A survey on mqtt: a protocol of internet of things (iot). In *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, volume 20.

[76] Stamatescu, G., Entezari, R., Römer, K., and Saukh, O. (2019). Deep and efficient impact models for edge characterization and control of energy events. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 639–646. IEEE.

[77] Stamatescu, G., Stamatescu, I., and Popescu, D. (2017). Consensus-based data aggregation for wireless sensor networks. *Journal of Control Engineering and Applied Informatics*, 19(2):43–50.

[78] Subahi, A. and Theodorakopoulos, G. (2019). Detecting iot user behavior and sensitive information in encrypted iot-app traffic. *Sensors*, 19(21):4777.

[79] Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E., and Toulkeridis, T. (2020). From monolithic systems to microservices: A comparative study of performance. *Applied Sciences*, 10(17):5797.

[80] Tariq, M. A., Koldehofe, B., Altaweel, A., and Rothermel, K. (2010). Providing basic security mechanisms in broker-less publish/subscribe systems. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 38–49.

[81] Tawalbeh, L., Muheidat, F., Tawalbeh, M., Quwaider, M., et al. (2020). Iot privacy and security: Challenges and solutions. *Applied Sciences*, 10(12):4102.

[82] Thanh, L. N. T., Phien, N. N., Nguyen, T. A., Vo, H. K., Luong, H. H., Anh, T. D., Tuan, K. N. H., and Son, H. X. (2021a). Ioht-mba: An internet of healthcare things (ioht) platform based on microservice and brokerless architecture. *International Journal of Advanced Computer Science and Applications*, 12(7).

[83] Thanh, L. N. T., Phien, N. N., Nguyen, T. A., Vo, H. K., Luong, H. H., Anh, T. D., Tuan, K. N. H., and Son, H. X. (2021b). Sip-mba: A secure iot platform with brokerless and micro-service architecture. *International Journal of Advanced Computer Science and Applications*, 12(7).

[84] Thanh, L. N. T., Phien, N. N., Nguyen, T. A., Vo, H. K., Luong, H. H., Anh, T. D., Tuan, K. N. H., and Son, H. X. (2021c). Uip2sop: A unique iot network applying single sign-on and message queue protocol. *International Journal of Advanced Computer Science and Applications*, 12(6).

[85] Thi, Q. N. T., Dang, T. K., Van, H. L., and Son, H. X. (2017). Using json to specify privacy preserving-enabled attribute-based access control policies. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, pages 561–570. Springer.

[86] Toldinas, J., Lozinskis, B., Baranauskas, E., and Dobrovolskis, A. (2019). Mqtt quality of service versus energy consumption. In *2019 23rd International Conference Electronics*, pages 1–4. IEEE.

[87] Tran, D. L., Yu, T., and Riedl, M. (2020). Integration of iiot communication protocols in distributed control applications. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics*

1287   *Society*, pages 2201–2206. IEEE.

1288   [88] Trilles, S., González-Pérez, A., and Huerta, J. (2020). An iot platform based on microservices and
1289   serverless paradigms for smart farming purposes. *Sensors*, 20(8):2418.

1290   [89] Ullah, A. (2020). Rise of big data due to hybrid platform of cloud computing and internet of thing.
1291   *Journal of Soft Computing and Data Mining*, 1(1):46–54.

1292   [90] Wang, L., Törngren, M., and Onori, M. (2015). Current status and advancement of cyber-physical
1293   systems in manufacturing. *Journal of Manufacturing Systems*, 37:517–527.

1294   [91] Xuan, S. H., Tran, L. K., Dang, T. K., and Pham, Y. N. (2016). Rew-xac: an approach to rewriting
1295   request for elastic abac enforcement with dynamic policies. In *2016 International Conference on
1296   Advanced Computing and Applications (ACOMP)*, pages 25–31. IEEE.

1297   [92] Yassein, M. B., Shatnawi, M. Q., Aljwarneh, S., and Al-Hatmi, R. (2017). Internet of things: Survey
1298   and open issues of mqtt protocol. In *2017 international conference on engineering & MIS (ICEMIS)*,
1299   pages 1–6. Ieee.

1300   [93] ZANNI, A. (2015). Sistemas cyber-físicos e cidades inteligentes. *Developers Works. IBM COrpora-
1301   tion*.