Journal of Information Technology and Computer Science
Volume 6, Number 3, December 2021, pp. 288-296
Journal Homepage: www.jitecs.ub.ac.id

Internet of Things (IoT) Cloud Platform Using Message Queue Telemetry Transport (MQTT) End-to-Cloud Architecture

Fariz Andri Bakhtiar*[1],Moh. Wildan Habibi[2],Adhitya Bhawiyuga[3],Achmad Basuki[4]

[1,2,3,4]Faculty of Computer Science, Universitas Brawijaya, Indonesia
{[1]fariz,[3]bhawiyuga,[4]abazh}@ub.ac.id,[2]mohwildanhabibi@gmail.com
*Corresponding Author

Received 04 May 2021; accepted 28 December 2021

Abstract. IoT devices are constrained in computation and storage, therefore cannot store all long-term obtained data or perform complex computations. Shifting those jobs to cloud platform are feasible, yet rising heterogeneity and security issues. This study proposes an IoT cloud platform to facilitate communication among heterogeneous devices and the cloud while ensuring devices' validity. It uses publish/subscribe paradigm with an end-to-cloud architecture and HTTP-based auth server. The proposed system has successfully addressed heterogeneity and security issues. Performance tests conclude that the fewer publishers publish data simultaneously, the smaller the delay. Moreover, the system performs better at up to 250 publishers as the average delay is under 1000 ms, compared to 500 publishers that has average delay above 1000 ms. On its scalability, in 250-concurrent-publishers experiment, the system affords 191 publishers responded in under one second with 100% success rate. In 500-concurrent-publishers one, 187 responded in under one second with 99% rate.

1 Introduction

Internet of Things (IoT) refers to networks connecting various physical devices through different protocols [1]. IoT enables many events to be monitored and controlled remotely through the internet. The application of IoT in various fields could have connected each other via internet, hence making activities easier and more efficient.

The "things" in IoT are represented by interconnected devices that have certain identity, attributes, and characteristics. An IoT device commonly uses mini-computer having sensors to get some data. However, IoT devices have some limitations in computation and storage due to constrained components [2]. IoT device itself could not store all data obtained over the years of acquisition. They also do not have adequate resources to operate complex computations.

It is said [3] that "cloud" is a group of computing resources objects that can be configured and accessed from anywhere and have resources that can be quickly added or subtracted in an easy way. The study states that cloud virtually has unlimited storage and computing capabilities. Therefore, shifting IoT device's storage and computation processes to other system such as cloud computing platform can be workable solution.

Integrating cloud into IoT architecture could rise some issues, mainly heterogeneity and security [2]. A use of communication standard is needed to make sure that all IoT devices can communicate with the cloud. To ensure that the system is secure from spoofing threats, a device identification/validation mechanism is required.

The MQ Telemetry Transport (MQTT) protocol has the ability to work at environments that come up with low supply of power and small bandwidth [4]. It also offers reliability of the messages exchanged among nodes. Those features make it suitable to support an IoT system with diverse nodes.

As mentioned earlier, this study is intended to build an IoT cloud platform to facilitate communication among heterogeneous IoT devices and the cloud while ensuring validity of the devices involved. MQTT protocol is put to use in order to tackle the heterogeneousness issue. For ascertaining that the data would come from legitimate IoT devices, a technique of device management, authentication and authorization is necessary.

2 Proposed System

2.1 System Overview

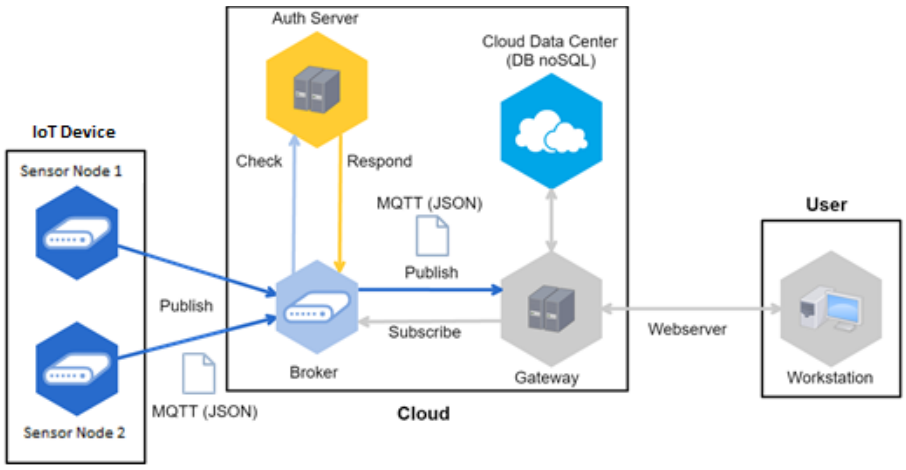

Fig. 1. The system's data communication flow.

In general, the proposed IoT cloud platform system uses MQTT Publish/Subscribe communication paradigm with an end-to-cloud architecture. The "end" is represented by an IoT device (sensor node) in the field which acts as a publisher sending its obtained data. On the other hand, the "cloud" in question is an IoT cloud platform system consisting of a subscriber, database, broker, and auth server, as depicted in Fig. 1. The cloud acts as storage and the recipient of data sent by sensor nodes. Publisher and subscriber scripts are developed using Python with the help of the paho-mqtt module.

The MQTT protocol uses the broker as a center for exchanging information between publishers and subscribers. The publisher, in the form of the sensor node, sends to the broker the sensor data information that has been obtained along with "topic" initialization. The system utilizes the topic as "destination address" which is

used by the publisher to send its data to. The data sent to a particular topic through the broker will be accessible by any subscriber knowing what topic is currently being used.

To be able to get messages from the publishers, a subscriber must make requests or subscribe to the same topics as the ones used by the publishers. Subsequently, the broker will reply the subscriber with information by first comparing the identity of the same topic between publisher and subscriber. If they match, the broker will then forward to the subscriber the data obtained from the publisher.

2.2 Publisher-Handling Workflow

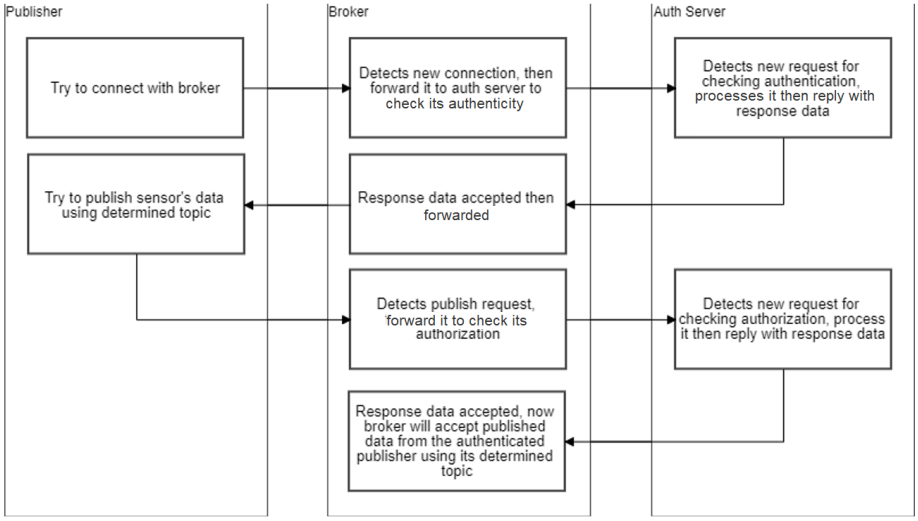

Fig. 2. The system's publisher-handling workflow.

Publishers are handled by the system as represented in Fig. 2. As the sensor node needs to publish the data, it is required to build a connection to the broker first. The broker then detects a connection request from it, and the connection data will be forwarded to auth server for authenticity checking afterwards.

When the auth server accepts the authentication request, it will then be processed, and the result will be sent back as the response which will be forwarded by the broker to the publisher. If the received response at publisher indicates that it is approved to connect to the broker, then subsequently the publisher will publish the data on the determined topic.

A request to publish data will then be sent by the authenticated publisher to the broker, which for the second time will get the published data off to the auth server, but this time for authorization check. The authorization check results will proceed as a response from the auth server to the broker.

If the response received by the broker is "positive", then the broker will be entitled to temporarily store the published data so that when any subscriber subscribes to the same topic, that data will be forwarded to the subscriber. If the response is "negative", the broker will not save the published data and will not accordingly forward it to the subscriber, even though the same topic is subscribed.

2.3 Subscriber-Handling Workflow

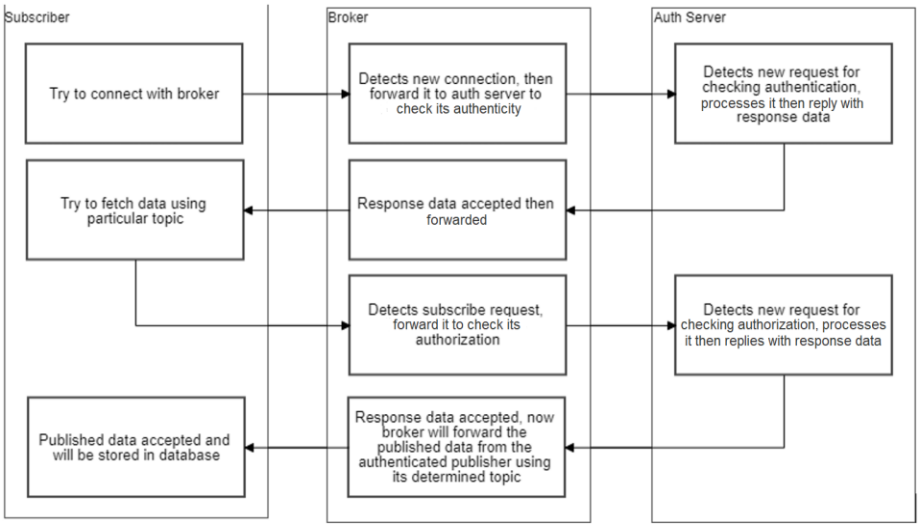

Fig. 3. The system's subscriber-handling workflow.

The system handles subscribers in a way described in Fig. 3. The workflow starts when a subscriber builds a connection required to subscribe a specific topic. Connection request from subscriber will be detected to be forwarded by broker to the auth server to check its authenticity.

After accepting the authentication request, auth server will immediately process and returns it to the broker as a response. Broker forwards the response back to the subscriber that sent the request at the first place. Auth server's approval will be the signal for subscriber to try to fetch the data it needs using certain topic.

When the broker detects a subscription request came from an authenticated subscriber, it will then forward it to the auth server for checking its authorization. The result from auth server will be returned as a response to the broker. If the initial request by the subscriber is authorized, the broker may finally forward published data from authenticated publishers to any subscriber that subscribes to the same topic.

2.4 Database Design Scheme

The system's data processing starts from the process of sending data that has been successfully authenticated and authorized. After this stage, the data will be received on the subscriber side. The subscriber checks the data payload scheme. If it matches the database schema, the data will be stored in the MongoDB-based database.

The database uses MongoDB, which is a document-oriented database system. In this database, data is represented in BSON (Binary JSON) documents. There are three collections used in this database, namely User collection, Nodes collection, and Subscriptions collection. Each has a field used as reference in filling in the collection. The contents of a collection are documents.

User collection is useful for storing user-related data, or users who have rights to contribute to the system. Nodes collection is effectively used to store data related to microcontrollers utilized as sensor nodes. There is one collection, namely Sensors collection, that is a little different from other ones as it is a collection inside another collection. It is worthwhile in storing data related to sensors owned or used by the microcontroller. Subscriptions collection is functional for storing data that has been sent by sensor nodes. The entire collections are related to one another through the "_id" field used by each collection, as depicted in Fig. 4.

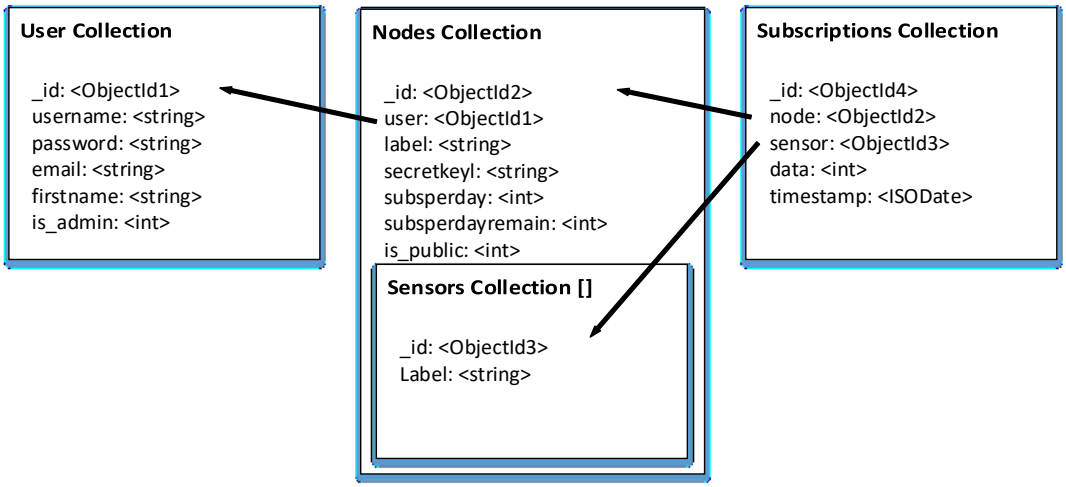

Fig. 4. MongoDB database scheme used by the system.

2.5 Auth Server Design

To address the needs for mechanisms of device management, authentication, and authorization, a Hypertext Transfer Protocol (HTTP)-based Auth Server is required. It is formulated to always monitor every connection that tries to connect to the broker, as per broker's demands. It serves three kinds of requests, namely authentication, superuser, and authorization. Authentication aims to ensure the validity of the IoT device that sends data. Superuser verifies the validity of subscribers. Meanwhile, authorization's objective is to make certain the validity of the topics used by IoT devices when sending data.

As portrayed in Fig. 5, when Auth Server is running, it will always be in a state of ready to use if there is no active request from the broker for any examination. If there is a request, the follow-up of each request will be nonidentical in the process, but all of them will end up responding to the broker.

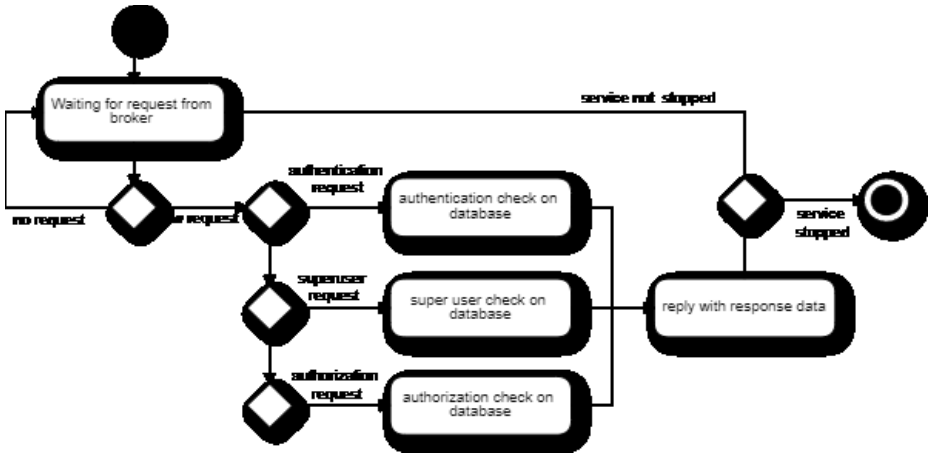

Fig. 5. The system's Auth Server design.

2.6 Broker Design

Broker's line of work is to accept published data from publishers using specific topics, then forward them to subscribers who subscribe to the very topics. Fig. 6 illustrates the algorithm of the broker as an MQTT Server. It starts with running the MQTT Broker function. When there is any latest data published by publisher, the broker will receive and store it based on its topic. At the time of a subscriber subscribes to a topic identical with one used by any publisher, published data that is already at the broker will be forwarded to the subscriber.

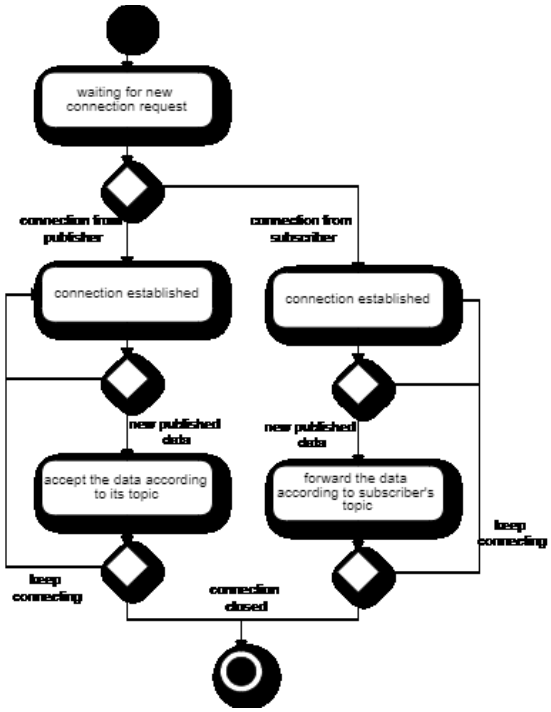

Fig. 6. The system's broker design.

2.7 Subscriber Design

Subscriber's job is to receive data that has been checked by the broker and store it to its database afterwards. Fig. 7 shows the subscriber's algorithm. A subscriber starts its work with a connection attempt to the broker. The broker will always check every attempted connection. Once connected, the subscriber will be ready to receive published data. When there is published data received, the subscriber will check whether the payload data have met the parameters in the publisher design. If so, the subscriber will then match the database used and store the published data in its database.

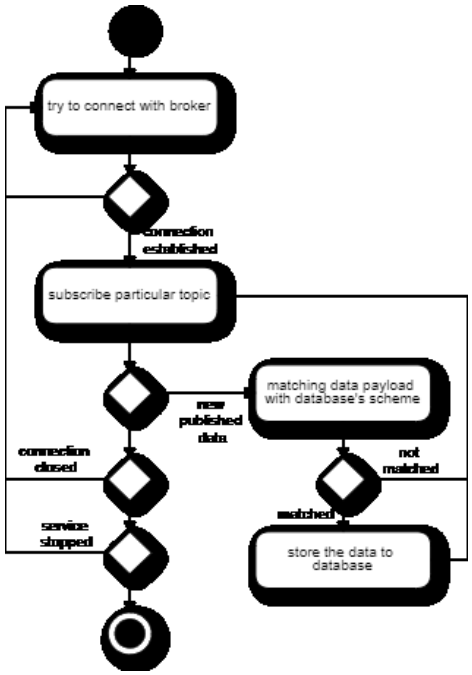

Fig. 7. The system's subscriber design.

3 Experiment

The experiment was divided into two parts i.e. delay time experiment and the proposed system's scalability performance. Table 1 shows experiment results of delay that occured while the data was being sent from publisher until received by subscriber without being inputted to the database.

Table 1. Results of average delay time occurred on data delivery from publisher to subscriber with as many publishers as 100, 250, and 500.

Delay					
Publisher	**Avg. Delay (ms)**	**Publisher**	**Avg. Delay (ms)**	**Publisher**	**Avg. Delay (ms)**
100	56.730	250	173.380	500	450.370
100	91.810	250	1021.920	500	5786.104

Delay					
100	112.700	250	810.332	500	3495.172
100	87.080	250	668.544	500	3243.882

The results in Table 1 convey that the delay time occured is directly proportional to the number of publishers who published data. The delay time occurred in the system when using 100 publishers and when using 250 publishers was always under 1000 ms. Whereas in experiments using number of publishers as many as 500, the delay time was above 1000 ms, so that it can be said to have worse results compared to one with less than 500 publishers.

It can be concluded that the system can handle the number of publishers less than 500 publishers at one time. Table 2 shows the results of experiments conducted to find out the number of publishers the system can handle per second. The experiments were carried out using 100, 250, and 500 publisher variations that had been accessed at the same time.

Table 2. Results of experiments getting the number of publishers the system can handle per second with the number of publisher variations accessed simultaneously as many as 100, 250, and 500.

Scalability			
Publisher	**Responded**	**Success Rate (%)**	**Responded < 1s**
100	100	100%	100
100	100	100%	100
100	100	100%	100
100	100	100%	100
250	250	100%	250
250	250	100%	156
250	250	100%	167
250	250	100%	191
500	500	100%	451
500	493	99%	1
500	495	99%	109
500	496	99%	187

The results reveal that the system is able to handle up to 500 publishers concurrently, with a success rate of 99%. In addition, the test results show that in system testing where 500 publishers were employed, the average number of publishers handled in under 1 second is 187 publishers.

4 Conclusion

The proposed system of MQTT-based publish/subscribe IoT cloud platform has successfully addressed heterogeneity and security issues in integrating IoT and cloud by implementing IoT device management, authentication, and authorization. Its performance test results conclude that the sending delay time is directly proportional to the number of connected publishers. It means that the fewer publishers publish data simultaneously, the smaller the delay. Moreover, the system performs better at up to 250 publishers as the average delay time is less than 1000 ms, compared to 500 publishers that has average delay time above 1000 ms. When scalability is on the

table, the system can have 191 publishers responded in less than one second in a 250-concurrent-publishers window with 100% success rate, and 187 publishers responded in less than one second in a 500-concurrent-publishers window with 99% success rate. Based on experiments, the system can deal with 0 to 250 publishers simultaneously to get better performance results.

References

1. Guoqiang, S., Yanming, C., Chao, Z. & Yanxu, Z.: Design and Implementation of a Smart IoT Gateway. In: 2013 IEEE International Confrence on Green Computing and Communications and IEEE Cyber, Physical and Social Computing, pp. 720–723 (2013)
2. Botta, A., Donato, W. d., Persico, V. & Pescape, A.: Integration of Cloud Computing and Internet of Things: A Survey. In: Future Generation Computer Systems, pp. 684-700 (2016)
3. Zhang, Q., Cheng, L. & Boutaba, R.: Cloud computing: state-of-the-art and research challenges. In: J Internet Serv Appl, Volume I, pp. 7-18 (2010)
4. OASIS Open, https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf (2019)