Edge Based MQTT Broker Architecture for Geographical IoT Applications

Ryo Kawaguchi and Masaki Bandai
Graduate School of Science and Technology, Sophia University
7-1 Kioi-cho, Chiyoda-ku, Tokyo, 102-8554 Japan
Email: r-kawaguchi-m4w@eagle.sophia.ac.jp

***Abstract*—Message queue telemetry transport (MQTT) is a simple and lightweight application protocol that has attracted attention in Internet of Things (IoT) and M2M communications. Because the MQTT standard defines only one broker in a system, the broker is not only a single point of failure but also not suitable for edge-based IoT applications. In fact, some existing brokers work in cooperation with multiple brokers by broadcasting publish or subscription messages, and these brokers increase system availability. Thus, in situations where a large number of clients connect and subscriptions change frequently, the load on the broker increases enormously. In this paper, we propose an MQTT broker architecture that improves system availability without broadcasting client information such as publish messages or subscriptions. According to numerical calculations, the proposed method significantly reduces the number of shared messages between brokers.**

***Index Terms*—IoT, MQTT, Edge computing**

I. Introduction

In recent years, as the cost of sensor devices for data collection continues to fall, Internet of Things (IoT) devices have spread rapidly. In 2030, 1,230 billion IoT devices will be connected to the Internet [1]. One of the promising applications of IoT is location-based services that provide location-specific information, such as environmental and traffic data, from numerous geographically placed sensors. The transmitted data are linked to a location. Location-based data have four requirements: 1) Ability to accommodate frequently changing interests of mobile users, 2) High system availability, 3) Small latency, 4) Tolerance for high network traffic.

Message queue telemetry transport (MQTT) is used as a lightweight application-layer protocol for IoT devices [2] [3]. MQTT is a publish/subscribe-type protocol in which publishers send a message to subscribers via an intermediate server called a broker. Each published message has one topic, which clients can use to subscribe to a broker. In the MQTT standard, because only one broker is defined in a system, this broker can be a single point of failure. Therefore, introducing multiple brokers in a system is a natural solution for improving system availability. Also, because the system has only one broker, the system architecture must be cloud-based. In particular, when MQTT is used for location-based services, the following issues will arise:

1) Although much of the data are used nearby [4], concentrating all the data in the cloud causes unnecessary network traffic and high latency.
2) If a broker breaks down, a geographically wide range of systems or applications may be affected.

There are some implementations that work cooperatively with multiple brokers. In such implementations, multiple brokers work as a single logical broker [5] [6] [7] [8]. The brokers broadcast publish messages or subscriptions to other brokers. However, the network traffic for sharing information between brokers increases when subscriptions frequently change in location-based services. Another option is a centralized architecture in which the management server manages topics of the entire system, as presented in [8]. However, because the management server is a single point of failure, essential system availability is not guaranteed. It is necessary to develop a system that considers both the frequency of subscription changes and system availability simultaneously.

In this paper, we propose an edge-based MQTT broker architecture for geographical IoT applications. The proposed method introduces a geographical topic structure for handling location-based data. In addition, the method distributes multiple brokers among edge networks to reduce latency from publishers to subscribers. Moreover, brokers form clusters to improve system availability. We implement a prototype of the proposed methods on Amazon Web Services (AWS). Furthermore, we numerically compare the number of messages shared by broker to confirm the effectiveness of the proposed system.

II. Related Work

We introduce two types of implementation for sharing information from clients among brokers. The first type shares publish messages among all brokers. In JoramMQ v1-1, "Clustered broker" broadcasts all publish messages to all other brokers [5]. Because all messages are gathered by all brokers, subscribers can obtain any message regardless of which broker they are connected to. The second type, for example, HiveMQ, shares subscription messages among all brokers [6]. A subscription is the information about who has subscribed to each topic. All brokers know all subscriptions, so each broker knows which broker to send publish messages to. These implementations have high system availability. However, in cases where the number of clients is large and subscriptions change frequently, a huge number of messages are delivered to all brokers, which is not appropriate from the viewpoint of broker load and network bandwidth.

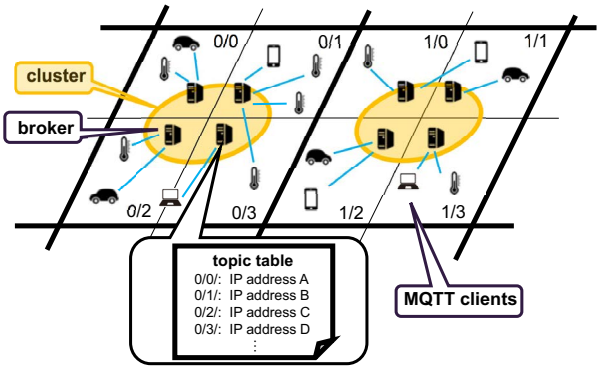

Fig. 1. System architecture.

Banno et al. proposed nodes between brokers to link multiple heterogeneous brokers [7]. The nodes share publish messages or subscriptions, but the method essentially causes the same problem as the implementation that shares information among multiple brokers. Park et al. used a software defined network (SDN) to implement a multiple-edge broker method that makes the entire system work logically as one broker [8]. However, because of the centralized control by the SDN controller, the essential availability cannot be considered. In addition, we previously proposed a geographically distributed broker system for linking heterogeneous brokers without sharing information [9]. In this system, a group of topics can be assigned to each broker, but if a broker fails, none of the topic groups for which this broker is responsible can used.

III. Proposed Method

In this paper, we propose a multi-broker architecture that guarantees availability without sharing client information between brokers. The proposed method has three components:

1) Geographically hierarchical topics and a topic table,
2) Edge-based brokers, and
3) Broker clusters.

Figure 1 shows the system architecture. The target geographical area is divided into hierarchies, and brokers with their own copy of the topic table are assigned at a level. Moreover, multiple brokers form a cluster. Each area is named by Z-ordering, such as 0/1, and each area contains a broker. The MQTT clients are IoT devices or applications that use location-based data.

A. Geographically hierarchical topics and topic table

In the proposed method, Z-ordering [10] is introduced to include location-based data in the general MQTT topic notation. MQTT topics have a hierarchical structure separated by "/," and subscribers can use wildcards, such as single-level "+" and multilevel "#." Thus, subscribers who use these wildcards can effectively subscribe to topics in specific geographical areas.

In the proposed method, the topic notation of location-based data is as follows:

(geographical representation based on Z-ordering) / (unlimited part)

Z-ordering is a method that can map two-dimensional information, such as (x, y), to hierarchical one-dimensional information. First, a geographical square area is equally divided into four areas. Each division is labeled as 0, 1, 2, or 3 in the same order in which the letter Z is written. By repeating this operation recursively k times for each divided area, the first square is divided into 4^k areas, and k numbers are obtained. In the general notation of MQTT, a hierarchy is created by separating these numbers with a slash ("/"). Z-ordering is used in the proposed method because it is compatible with location-based data and MQTT.

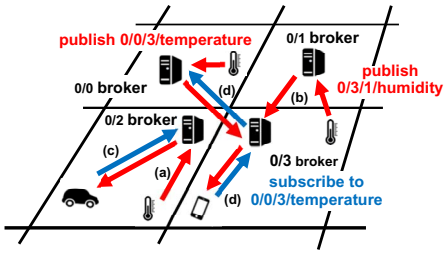

Fig. 2. Broker actions.

Publishers of location-based data embed the location information for the data in a topic and publish it in a manner such as "1/0/2/1/2/temperature." This allows a client to use wildcards to subscribe to a geographically wide range of topics and to know the location of the data, for example, "1/0/2/+/+/temperature." The topic table describes a group of topics, such as "1/0/," paired with the IP address of all brokers responsible for the topic. All brokers have the same topic table.

B. Edge broker

Each broker is placed in edge nodes, and Figure 2 shows how they act within this structure.

(a) If a topic that the broker is responsible for is published, the broker forwards it to the subscribers to which it is connected.
(b) If a topic that the broker is NOT responsible for is published, the broker looks at the topic table and publishes it to the broker that is responsible for the topic.
(c) If a client subscribes to a topic for which the broker is responsible, the broker forwards published messages to the subscriber.
(d) If a client subscribes to a topic for which the broker is NOT responsible for, the broker looks at the topic table and forwards the subscription to the broker that is responsible for the topic by proxy.

Thus, brokers publish messages forwarded to them from other brokers by proxy subscription to the subscribers to which they are connected. If a change message for the topic table is sent from other brokers, the broker updates its own topic table.

Generally, clients do not need to be aware of the brokers to which they are connected. Each client connects to the nearest broker in the network. Usually, the broker responsible broker for a topic for publish or subscribe messages is the same as the nearest broker. If the nearest broker is not responsible for the topic, the broker works as a proxy client.

C. Broker clusters

A cluster formed from multiple brokers a unit by which brokers can check each other's availability and move their topics.

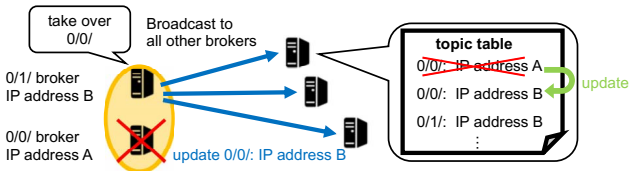

Fig. 3. Broker action when a broker becomes unavailable.

All clusters are formed with brokers that are geographically close. The parameters l_c and l_b represent the levels of the hierarchy where clusters and brokers are placed, respectively. If the number of divisions by Z-ordering is k, l_c and l_b satisfy the following inequality:

$$1 \leq l_c < l_b \leq k, \qquad l_c, l_b, k \in \mathbb{N}.$$

The size of l_c and l_b depends on the system scale or the number of clients in the system.

Each broker periodically confirms the availability of other brokers in the cluster, for example, by exchanging "hello packets" with other brokers in the same cluster. If a broker in the cluster is not available, the geographically closest broker in the same cluster or the broker with the least number of connected clients can temporarily take on the unavailable broker's topic. Figure 3 shows the action of brokers when a broker becomes unavailable. When a topic migration occurs, the broker accepting the topic broadcasts the changes to the topic table to all brokers in the system.

In the proposed method, each broker is associated with a group of topics. All topics have geographic information embedded by Z-ordering, and the broker that is located near the topic's geographical area is responsible for that topic. As a result, clients that are geographically close to each other are more likely to be connected to the same broker, and latency among clients is reduced with high probability. In addition, each broker can uniquely identify any broker responsible for a topic by referencing the topic table. Therefore, it is possible for brokers instead of clients to transmit publish and subscribe messages without sharing client information of between brokers. In addition, the broker responsible for a topic can be changed by updating all copies of the topic table. By migrating the topics of non-available brokers, this system can provide high availability.

IV. Prototype Implementation

We implement a prototype of the proposed method. We perform the broker implementation on an AWS EC2 t2.micro instance with $k = 4$, $l_b = 2$, and $l_c = 1$. Brokers are implemented using mosquitto [11] and Paho MQTT Python Client. The mosquitto-client, which is a library of MQTT clients, is used to check the system operation.

Figure 4 shows a photograph of the implemented system. This image shows an experiment when a broker receives a subscription for a topic that it is not responsible for. The top four images are the terminals when the broker program is run when connected to a broker with an SSH client. The terminal at the bottom left of the screen is in our local area network publishing to a 1/2/ broker. The terminal at the bottom right of the screen is in same local area network and subscribes topic

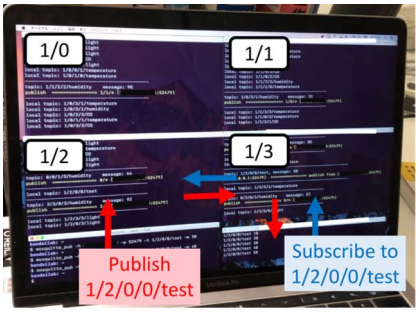

Fig. 4. Photograph of the implemented system.

"1/2/0/0/test" to the 1/3/ broker. In this case, the 1/3/ broker subscribed to the 1/2/ broker with topic 1/2/0/0/test. Because the message of "1/2/0/0/test" is published to 1/2/broker, the 1/2/ broker sent it to the 1/3/ broker. After that, the 1/3/ broker published this message to the subscriber by proxy. We confirmed that the publish message sent from the bottom left terminal arrived at the bottom right terminal.

V. Performance Evaluation

The performance evaluation compares the proposed method with the following two systems.

System 1) Broadcast published messages to all brokers
System 2) Broadcast subscription messages to all brokers.

The purpose of this paper is to improve system availability and to deal with frequent subscription changes and a huge number of clients. The system availability of the proposed method is the same as the other methods. Because systems 1 and 2 share all publish or subscription messages between brokers, the system will continue to work if any broker failure. The proposed method provides the capability for when the broker becomes unavailable, after which the broker responsible for the failed broker's topic will change immediately. Therefore, we only compare the cost of the number of messages sent and received by each broker per second.

A. Theoretical derivation

We derive the theoretical number of transmitted and received messages of the existing implementations and the proposed system. MQTT control packets, such as CONNECT, are not considered. We assume the cost of each publish event to be 1 and the cost of each subscribe event is also 1. System 1 sends the publish message to all brokers, so the cost is 1 for each message. System 2 sends the subscription message to all brokers, so the cost is again 1 for each. The cost of publish and subscribe by proxy between brokers of the proposed method is also assumed to be 1. We assume that the broker level is l_b, and there are N_{pub} publishers and N_{sub} subscribers connect to each broker. Each publisher publishes a message every T_{pub}. Each subscriber sends a subscription message every T_{sub}. A subscriber subscribes to a topic from other areas with a probability of p_{other}.

1) System 1: In system 1, the number of published messages received by each broker per unit time and the number of messages due to subscription change received are:

$$Publish : \frac{4^{l_b} N_{pub}}{T_{pub}}, \qquad Subscription : \frac{N_{sub}}{T_{sub}}.$$

In addition, the number of messages that the broker transmits per unit time is as follows:

$$Publish : \frac{N_{sub}}{T_{pub}} + \frac{(4^{l_b} - 1) N_{pub}}{T_{pub}}, \quad Subscription : 0.$$

2) System 2: In system 2, the number of published messages received by each broker per unit time and the number of messages due to subscription changes received are:

$$Publish : \frac{N_{pub}}{T_{pub}} + \frac{p_{other} N_{sub}}{T_{pub}}, \; Subscription : \frac{4^{l_b} N_{sub}}{T_{sub}}.$$

In addition, the number of messages that the broker transmits per unit time is as follows:

$$Publish : \frac{(1 + p_{other}) N_{sub}}{T_{pub}}, \; Subscription : \frac{(4^{l_b} - 1) N_{sub}}{T_{sub}}.$$

3) Proposed method: Geographical proximity and network proximity are different. Therefore, the broker to which a publishing client connects is not necessarily the broker in charge of publisher's topic. In such a case, proxy publish occurs between brokers in the proposed method. Therefore, let p_{miss_pub} be the probability that a broker is in the same geographical area but is connected to a broker in another area by publishers.

In the proposed method, the number of published messages received by each broker per unit time and the number of messages due to subscription change received are:

$$Publish : \frac{(1 + p_{miss\_pub}) N_{pub}}{T_{pub}} + \frac{p_{other} N_{sub}}{T_{pub}},$$
$$Subscription : \frac{(1 + p_{other}) N_{sub}}{T_{sub}}.$$

In addition, the number of messages that the broker transmits per unit time is as follows:

$$Publish : \frac{(1 + p_{other}) N_{sub}}{T_{pub}} + \frac{p_{miss\_pub} N_{pub}}{T_{pub}},$$
$$Subscription : \frac{N_{sub} p_{other}}{T_{sub}}.$$

B. Numerical results

We calculate the number of messages dealt with by each broker by using these theoretical formulas. We assume that $l_b = 3$ (in other words, there are 4^3 brokers), $N_{pub} = 500$, $N_{sub} = 10,000$, $T_{pub} = 10$ s, $T_{sub} = 150$ s, and $p_{other} = 0.3$, $p_{miss\_pub} = 0.4$. Figure 5 shows the results for the number of transmitted and received messages. Because system 1 broadcasts the publish message and system 2 broadcasts the subscription, each has a large value. For both message types in the proposed method, because broadcasting is reduced, none of the values are too large. The bottom of Figure 5 shows the sum of the number of transmitted and received messages. Because broadcasting is reduced, the total number of messages is smallest for the proposed method. Figure 6 shows that the total number of messages affected by changes in T_{sub}. The proposed method can significantly reduce the messages when subscriptions change frequently.

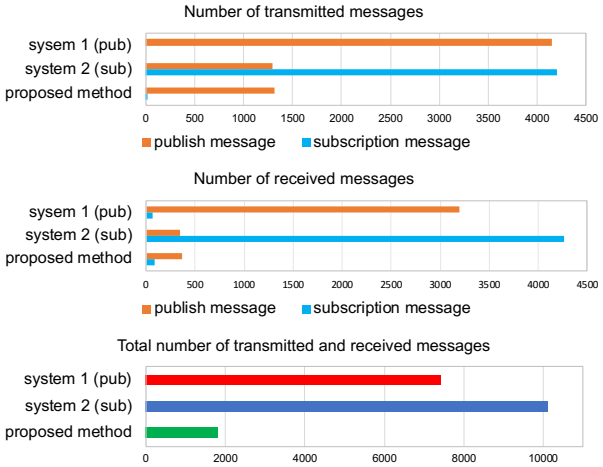

Fig. 5. Number of transmitted and received messages.

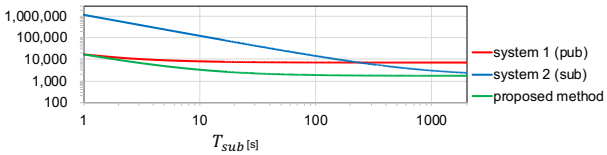

Fig. 6. Total number of messages affected by changes in T_{sub}.

VI. Conclusion

In this paper, we proposed an MQTT architecture that employs multiple brokers without sharing client information between brokers. We implemented a prototype on AWS EC2 and confirmed its operation. The numerical evaluation confirmed that the proposed method significantly reduces message traffic between brokers compared with existing implementations.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 19K11931.

References

[1] IHS Markit, Number of Connected IoT Devices Will Surge to 125 Billion by 2030. https://technology.ihs.com/596542/number-of-connected-iot-devices-will-surge-to-125-billion-by-2030-ihs-markit-says

[2] OASIS, MQTT Version 3.1.1 Plus Errata 01. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[3] N. Naik, "Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP," In Proc. *IEEE ISSE2017*, 2017.

[4] Y. Teranishi, R. Banno, T. Akiyama, "Scalable and Locality-Aware Distributed Topic-based Pub/Sub Messaging for IoT," In Proc. *IEEE GLOBECOM2015*, 2015.

[5] JoramMQ, http://www.scalagent.com/IMG/pdf/JoramMQ_MQTT_white_paper.pdf

[6] HiveMQ, https://www.hivemq.com/

[7] R. Banno, J. Sun, M. Fujita, S. Takeuchi, K. Shudo, "Dissemination of Edge-Heavy Data on Heterogeneous MQTT Brokers," In Proc. *IEEE CloudNet2017*, 2017.

[8] J. Park, H. Kim, W. Kim, "DM-MQTT: An Efficient MQTT Based on SDN Multicast for Massive IoT Communications," In Proc. *Sensors Journal*, 2018.

[9] R. Kawaguchi, M. Bandai, "A Distributed MQTT Broker System for Location-based IoT Applications," In Proc. *IEEE ICCE2019*, 2019.

[10] G.M. Morton, "A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing," *Morton1966*, 1966.

[11] Eclipse Mosquitto, https://mosquitto.org/