Dissemination of Edge-Heavy Data on Heterogeneous MQTT Brokers

Ryohei Banno*†, Jingyu Sun*, Masahiro Fujita‡, Susumu Takeuchi* and Kazuyuki Shudo†
* NTT Network Innovation Laboratories, Tokyo 1808585, Japan, Email: banno.ryohei@lab.ntt.co.jp
† Tokyo Institute of Technology, Tokyo 1528550, Japan
‡ Kyoto Sangyo University, Kyoto 6038555, Japan

***Abstract*—MQTT is one of the promising protocols for exchanging IoT data. IoT data have a characteristic called "edge-heavy" which means that things at the network edge generate a massive volume of data with high locality of utilization. For dissemination of such edge-heavy data, an architecture in which multiple MQTT brokers placed at the network edges cooperate with each other is quite effective. This edge-based architecture makes latency lower, as well as reducing consumption of cloud resources. However, under this kind of architecture, heterogeneity could be a vital issue, i.e., an appropriate product of MQTT broker could vary according to the different environment of each network edge. In this paper, we propose Interworking Layer of Distributed MQTT brokers (ILDM), which enables arbitrary kinds of MQTT brokers to cooperate with each other. We provide two basic cooperation algorithms, including the way to furnish MQTT-specific functions such as QoS and Retain. To evaluate the feasibility of ILDM, we also formulate a benchmark method which can be used for both a single broker and multiple brokers. Experimental results show that the throughput of five brokers running together by ILDM is improved 4.3 times at maximum than that of single broker.**

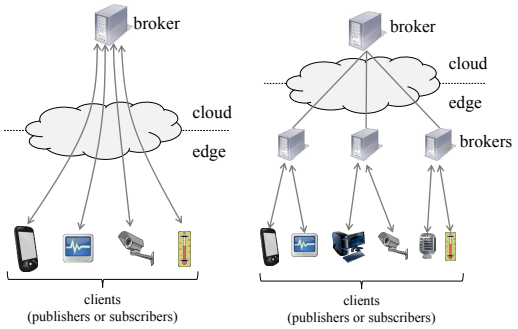
***Keywords*—Publish-subscribe, Distributed information systems, Edge computing, Multicast algorithms.**

I. Introduction

MQTT has attracted much academic and industrial interest in recent years as one of the key technologies of IoT services [1]. It is a protocol of topic-based pub/sub, in which messages are exchanged through logical channels called "topics". MQTT uses a server called "broker" to manage topics and mediate between publishers and subscribers. This paradigm provides decoupling between clients, e.g., each publisher has no concern with the location of subscribers that will receive its message [2].

Although typical IoT systems deploy an MQTT broker in the cloud [3] as shown in Figure 1(a), this centralized architecture can cause some issues due to the following characteristics of IoT data:

- A massive volume of data is generated at the network edge, rather than in the cloud.
- Many of the data have high locality of utilization; a data generated in an area is often utilized in the same area.
- Data are often utilized for event-driven services, so that the capability of exchanging data in real-time is indispensable.



(a) Cloud-based (b) Edge-based

Fig. 1. Example of architectures for disseminating edge-heavy data.

Such characteristics are called "edge-heavy" [4]. The problems of managing edge-heavy data with the above cloud-based architecture are resource consumption and latency. That is, heavy load is concentrated with oppressing cloud resources such as network bandwidth, and required time from a publisher to a subscriber tends to be long due to the distance between devices and cloud data centers.

Cooperation of multiple MQTT brokers placed at the edges is a solution to the problems. In this architecture, shown in Figure 1(b), the cloud broker does not need to communicate with all clients directly, so that the consumption of cloud resources is reduced and consequently overall throughput is increased. Furthermore, it makes latency lower for locally consumed data, because the edge brokers are closer to IoT devices. There could be several variations of this architecture, such as cascaded edge brokers and cooperation without the cloud broker.

Assuming such edge-based architecture, there is an issue of heterogeneity of brokers. Namely, an appropriate product of a broker is different according to an environment of each network edge. There are many choices: open source or proprietary, software or embedded appliance, difference in supported OSs or functional features, and so on. To construct the edge-based architecture, such heterogeneous brokers have to cooperate with each other. Even though some of existing products have functions of cooperation between multiple brokers, e.g., "bridge" of Mosquitto [5] and "cluster" of HiveMQ [6], there is no interoperability between different products because any

cooperation protocols are not standardized in the latest version 3.1.1 of MQTT specification [7].

In this paper, we propose Interworking Layer of Distributed MQTT brokers (ILDM), which enables arbitrary kinds of brokers to cooperate with each other. ILDM provides APIs which facilitate rapid development of variety of cooperation algorithms. We also propose two basic algorithms using the APIs.

In addition to the heterogeneity, there is another issue of benchmark. To evaluate and determine an appropriate architecture, benchmark method which can be used for both a single broker and multiple brokers is needed. For this matter, we formulate a benchmark method which ensures that error ratios of resulted performance are not more than 5 percent.

The contributions of this paper are threefold:

- First, we give a fundamental idea of ILDM with two basic cooperation algorithms.
- Second, we provide a practical method for benchmark of MQTT broker/brokers.
- Third, we show the feasibility of ILDM-based cooperation through experiments.

The rest of this paper is organized as follows. Section II illustrates a fundamental idea of ILDM-based cooperation. Section III describes two basic algorithms of cooperation using ILDM. In section IV, we introduce a practical method for benchmark of MQTT broker/brokers. Section V discusses the results of experiments to confirm the feasibility of ILDM-based cooperation. Section VI explains related studies. Finally, we summarize and conclude this paper in Section VII.

II. MQTT AND ILDM

A. Overview of MQTT

MQTT [7] is a protocol of topic-based pub/sub, standardized by OASIS. It is known for lightweight design such as a minimum of two bytes header size. As we stated before, a broker manages topics and mediates between clients. Below is an example flow of using MQTT.

1) A client X sends CONNECT message to a broker. This establishes a connection between X and the broker.
2) X sends SUBSCRIBE message to the broker. This message informs the topics of interest of X to the broker.
3) Another client sends PUBLISH message to the broker, with specifying a topic. If the topic is included in the above topics of interest, this message is forwarded to X by the broker.
4) X sends DISCONNECT message to the broker, to terminate the connection.

MQTT provides several useful functions for clients, such as "QoS", "Retain" and "Will".

QoS provides capability of configuring the level of delivery confirmation. A client and a broker try to confirm the delivery of a PUBLISH message and resend it if needed, according to the QoS level. Three levels are defined: "At most once delivery", "at least once delivery", and "exactly once delivery".

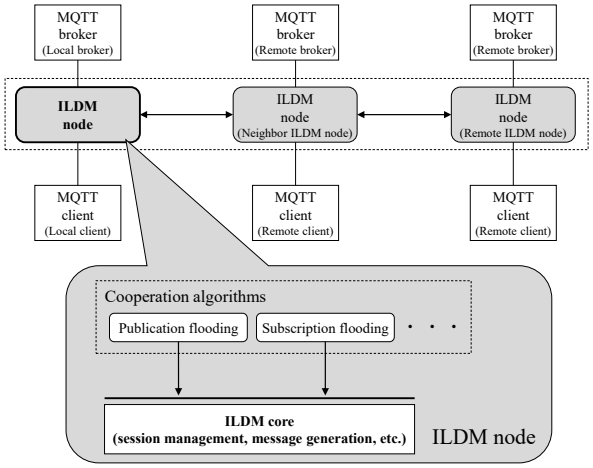


Fig. 2. Interworking Layer of Distributed MQTT brokers.

Retain is for delivering a latest message in the past to a new subscriber. A PUBLISH message has a flag of Retain. If the flag is set to $true$, a broker stores the message until a new PUBLISH message whose Retain flag is $true$ of the same topic arrives. This stored message will be forwarded to new subscribers of the topic.

Will enables to inform unexpected close of a connection. CONNECT message has a flag of Will. If the Will Flag is set to $true$, a broker stores a Will message and Will topic which are also included in the CONNECT message. The Will message will be published from the broker, when it detects the connection with the client which has sent the CONNECT message is unexpectedly closed.

B. Fundamentals of ILDM

In this paper, we propose Interworking Layer of Distributed MQTT brokers (ILDM). ILDM-based cooperation is composed by multiple brokers and ILDM nodes. An ILDM node is arranged between a broker and clients as shown in Figure 2. As well as relaying MQTT clients and a broker as if it were a proxy, an ILDM node can connect with other ILDM nodes so that multiple and arbitrary kinds of brokers can communicate with each other via ILDM nodes.

Regarding an ILDM node, we assume the following notations: **local client** denotes a client which directly connects with the ILDM node, **local broker** denotes a broker which directly connects with the ILDM node, **remote ILDM node** denotes one of the other ILDM nodes included in the whole cluster, **neighbor ILDM node** denotes one of the remote ILDM nodes which directly connects with the ILDM node, **remote client** denotes a client which connects with a remote ILDM node, **remote broker** denotes a broker which connects with a remote ILDM node.

As there can be a variety of cooperation algorithms, a core component of an ILDM node provides APIs so that algorithm developers can implement their algorithms easily. For example, there is a callback API:

```
void mqttMessageArrived(SessionInfo session, MsgType
    type, byte[] message)
```

MsgType indicates the type of MQTT messages such as CONNECT, CONNACK and PUBLISH. This API enables to execute arbitrary processes when an ILDM node receives a corresponding type of an MQTT message.

Such APIs are useful for not only cooperation, but also various data processing, e.g., validation of data format.

III. Cooperation algorithms

In this section, we propose two basic cooperation algorithms: Publication Flooding (PF) and Subscription Flooding (SF). These algorithms suppose ILDM nodes are connected in a tree structure which does not include closed paths.

A. PF-based cooperation

PF is a method to share each PUBLISH message among all brokers via ILDM nodes.

Each ILDM node relays a SUBSCRIBE message received from a local client to its local broker. Regarding a PUBLISH message, an ILDM node does not only relay, but also transfers to its neighbor ILDM nodes. ILDM nodes, which receive the transferred PUBLISH message, send it to their own local broker. They further transfer the message to their neighbor ILDM nodes, if exist. Eventually, all connected brokers receive the PUBLISH message and forward it to their local clients subscribing to the corresponding topic.

Figure 3 shows an example. There are five sets of a broker and an ILDM node: B_1 and I_1 to B_5 and I_5. There are also three clients: C_1 to C_3. We consider the following three steps.

1) Step 1: C_1 subscribes to a topic t.
2) Step 2: C_2 subscribes to the same topic t.
3) Step 3: C_3 publishes to the same topic t.

Dotted arrows represent the flow of SUBSCRIBE messages, while solid arrows are the PUBLISH messages'.

When I_2 and I_3 receive a SUBSCRIBE message from C_1 and C_2, they just relay it to their local broker. As well as being relayed alike, a PUBLISH message from C_3 is transferred by I_5 to I_3, and spreaded to all ILDM nodes in a chain reaction.

B. SF-based cooperation

Unlike PF, the basic idea of SF is to share subscription information among ILDM nodes.

When an ILDM node receives a SUBSCRIBE message, it informs the subscription information, e.g., topic name and QoS level to its neighbor ILDM nodes, as well as relays the message to its local broker. We call this operation "inter-subscribe", because it is as if it were the subscribe operation of the MQTT protocol between two ILDM nodes. For example, when an ILDM node X informs the information to another ILDM node Y, it means that "ILDM node X inter-subscribes against ILDM node Y".

When an ILDM node is about to inter-subscribe against a neighbor ILDM node, it checks overlapping with existing subscriptions. If an overlap is judged to be present, the ILDM node will not inter-subscribe redundantly. That is, inter-subscribe operations between ILDM nodes are to share only the difference from existing subscriptions.

Regarding a PUBLISH message, as well as relaying to the local broker, an ILDM node transfers it to neighbor ILDM nodes which have inter-subscribed to the topic of the message. ILDM nodes which receive the transferred PUBLISH message send it to their own local broker. They further transfer the message to their neighbor ILDM nodes which have inter-subscribed to the topic. Eventually, all brokers which have local clients subscribing to the topic receive the PUBLISH message and forward it to corresponding subscribers.

Figure 4 shows an example. The topology and scenario is same as Figure 3. When I_2 receives a SUBSCRIBE message from C_1, it does not only relay the message to its local broker, but also inter-subscribes against I_1, I_3 and I_4. I_3 further inter-subscribes against I_5. In the next step, I_3 receives a SUBSCRIBE message from C_2 and subsequently inter-subscribes against I_2. I_3 does not inter-subscribe against I_5, because I_3 has already inter-subscribed in the first step. Similarly, I_2 does not inter-subscribe against I_1 and I_4.

A PUBLISH message from C_3 is transferred by I_5 to I_3, because I_3 has inter-subscribed to the topic t against I_5. I_3 also transfers the message to I_2, and finally C_1 and C_2 receive the message.

C. Furnishing MQTT-specific functions

As we described in Section II-A, MQTT has some specific functions. We show the way to enable clients to use these functions transparently over multiple brokers with PF and SF method.

1) QoS: Both in PF and SF, An ILDM node relays QoS-related messages such as PUBACK so that QoS level configuration is available between a local broker and local clients. Further, we can apply the idea of QoS control to transferring a PUBLISH message between adjacent ILDM nodes. This enables distributed brokers to adjust a tradeoff of reliability and performance.

2) Retain: PF method can provide Retain function without adding special processes, because each broker receives all PUBLISH messages and stores them if they have retain-flag being set to $true$. In case of SF method, when an ILDM node receives a PUBLISH message with retain-flag set to $true$, it needs to transfer the message to adjacent ILDM nodes even though the topic is not inter-subscribed. This makes sure each broker can send out an appropriate retained message when a new subscriber of the corresponding topic comes.

3) Will: As described above, an ILDM node transfers a PUBLISH message when it receives it from local clients, not from a local broker. Hence, a will-message which comes from a local broker is not transferred to the neighbor ILDM nodes within the basic procedures of PF and SF methods.

According to the MQTT specification of version 3.1.1, a PUBLISH message from a broker does not have any information to know whether it is an will-message or not. Conse-

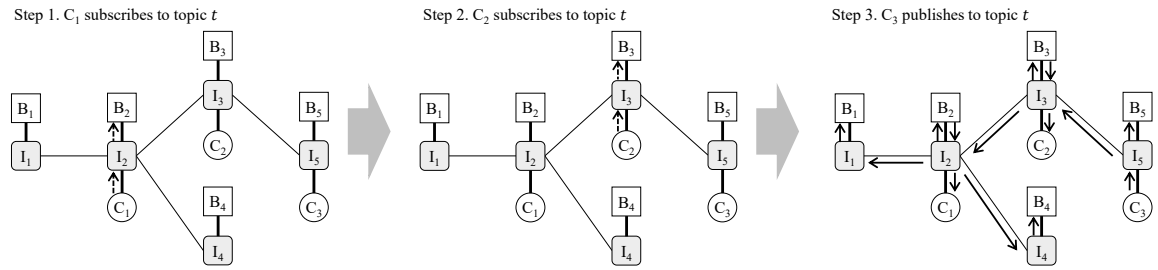


Fig. 3. Example of PF-based cooperation.

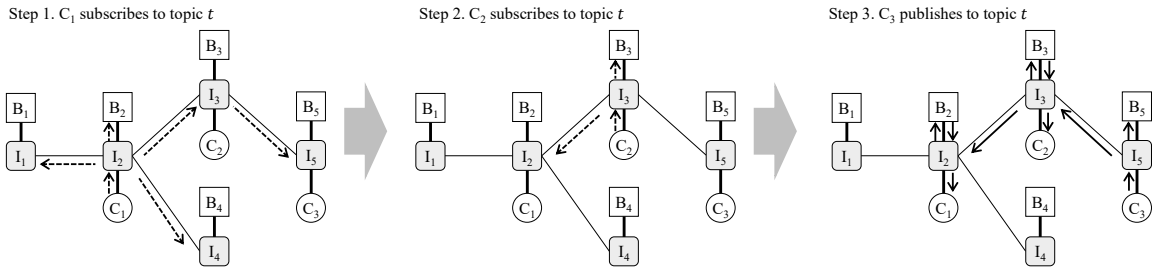


Fig. 4. Example of SF-based cooperation.

quently, an ILDM node itself needs to store a will-message and a will-topic internally when it receives a CONNECT message.

When an ILDM node detects the unexpected closing of a network connection with a local client or the local broker, and if the will-flag of the connection is set to $true$, it sends out the corresponding will-message to its neighbor ILDM nodes. The ILDM node needs not to send the will-message to local clients, because the local broker sends it. In case of SF method, sending will-messages to neighbor ILDM nodes is executed only if the will-topics are inter-subscribed.

D. Qualitative comparison

In PF method, each broker receives all PUBLISH messages regardless of the presence of corresponding subscribers. This means that the total number of ingress messages on each broker is basically same as the case of a single broker. Therefore, the effect of load distribution mainly depends on a dispersion condition of subscribers. The more scattered the subscribers are, the more effective this method is.

In SF method, a PUBLISH message is delivered to brokers which have subscribers of the same topic as the PUBLISH message. Brokers, which do not have such subscribers and are not in the paths of delivering the message, do not receive it. Hence this method is effective when publishers and subscribers of a same topic are convergently placed on a small sub-tree.

IV. Benchmark method

To verify the effects of ILDM, we formulate a benchmark method which can be applied for both a single broker and multiple brokers.

A. Performance indexes

We consider the following four viewpoints as performance indexes of MQTT brokers.

ingress throughput
: Number of messages brokers receive from publishers per unit time.

egress throughput
: Number of messages brokers send out to subscribers per unit time.

latency
: Required time since a publisher sends a message until a subscriber receives it.

loss rate
: Ratio of the number of missed messages to the number of messages which subscribers should receive.

We also define the limit of performance as follows.

Definition 1. *If measured throughput satisfies the following restriction, the performance is under the limit.*

$$\frac{\{\text{egress throughput}\}}{\{\text{ingress throughput}\}\times\{\text{sp-ratio}\}} \geq 0.99$$

where
$\text{sp-ratio} = \frac{\sum_i P(t_i)*S(t_i)}{\sum_i P(t_i)}$,
$P(t_i) = \{\text{number of publishers of } i\text{th topic}\}$,
$S(t_i) = \{\text{number of subscribers of } i\text{th topic}\}$

This is based on the idea that if egress throughput is less than ingress throughput multiplied by sp-ratio, the number of pending messages in the brokers is monotonically increasing.

In other words, this definition represents the limit of allowable continuous load.

B. Benchmark procedure

To find the maximum performance satisfying the restriction defined in Definition 1, we introduce a new benchmark method. This method tries to find the point of the very limit by varying the interval of PUBLISH messages. It is conducted along with the following steps.

Step 1: Conduct measurement repeatedly with doubling the interval of sending a PUBLISH message. For example: 1ms, 2ms, 4ms, 8ms, 16ms, ...

Step 2: From the results of Step 1, find the minimum interval satisfying the restriction stated in Definition 1.

Step 3: Divide the segment between the minimum interval and the smaller interval next to the minimum interval into 20. For example, if the minimum interval is 4ms, we divide the segment between 4ms and 2ms like as : 2ms, 2.1ms, 2.2ms, ..., 3.9ms, 4ms.

Step 4: Conduct measurement for each interval calculated in Step 3.

Step 5: From the results of Step 4, find the minimum interval satisfying the restriction in Definition 1.

Finally, the result by using the minimum interval clarified in Step 5 indicates the limit of performance.

This result is ensured that the error ratio is not more than five percent. In other words, the throughput resulted by using the smaller interval next to the minimum interval is at most 1.05 times larger than using the minimum interval. We can prove it as follows:

Proof. We assume that x is the minimum interval in Step 2. Therefore $x/2$ is the smaller interval next to the minimum interval. Here the stepping width calculated in Step 3 is $(x - x/2)/20 = x/40$. We denote the width as y. The error ratio is at least $x/(x - y) - 1$ and at most $(x/2 + y)/(x/2) - 1$. Consequently, the highest error ratio is 0.05. □

V. Evaluation

We implemented ILDM with PF and SF methods in Java, based on the MQTT version 3.1.1 specification. We also implemented a load testing tool by using a client library of SurgeMQ [8] known for its high performance so that the tool can send/receive PUBLISH messages with high frequency.

Regarding hardware environments, we prepared two types of servers described in Table I for installing brokers and ILDM nodes. All machines used in experiments were connected by using a non-blocking L2 switch.

We conducted some experiments by using the benchmark method described in Section IV. The aim of the experiments is to confirm the feasibility of ILDM mainly from the viewpoint of improvement of throughput, which is carried from reducing consumption of resources with the edge-based architecture.

In each experiment, we ran the load testing tool for 80 seconds. The performance indexes stated previously were calculated by excluding the first and last 10 seconds, i.e., substantial measurement time was 60 seconds. QoS level was set to 0, and the size of payload of each PUBLISH message was 32 bytes.

TABLE I
Spec of servers

	Type S_1	Type S_2
Processor	Atom C2750 (8 core, 2.4 GHz)	Xeon E5-2690V3 (12 core, 2.6 GHz) × 2
Memory	16 GB	256 GB
OS	Ubuntu 14.04	Ubuntu 14.04
NW	1 GbE	10 GbE

The configuration of topics and clients are as follows:

- There are five topics for measuring throughput and loss rate: from topic1 to topic5.
- These topics have 10 publishers and 10 subscribers respectively, thus the sp-ratio is 10.
- There also be topic6 with a publisher and a subscriber for measuring latency.
- This publisher sends a PUBLISH message for each one second.

For convenience, we denote the clients of topic1 to topic5 by **t-clients**, and the clients of topic6 by **l-clients**. In case of multiple brokers, we used additional l-clients. We describe about this later.

We calculated the average of ingress/egress throughput and latency in the measurement time of 60 seconds, and found the limit of performance by using the benchmark method.

A. Evaluation of single brokers

As preliminary experiments, we evaluated the performance of open-source MQTT brokers alone. The aim is to get a reference of choosing a broker in evaluation of ILDM, as well as providing knowledge of performance characteristics of well-known MQTT brokers.

We used the following four brokers: Mosquitto 1.4.5, Moquette 0.8 [9], RabbitMQ 3.6.0 [10], and ActiveMQ 5.13.3 [11]. We measured the performance by changing the types of servers, S_1 and S_2, on which we ran the brokers.

Figure 5(a) and 5(b) shows the results of throughput. As the benchmark method indicates, egress throughputs are almost equal to ingress throughputs multiplied by the sp-ratio 10.

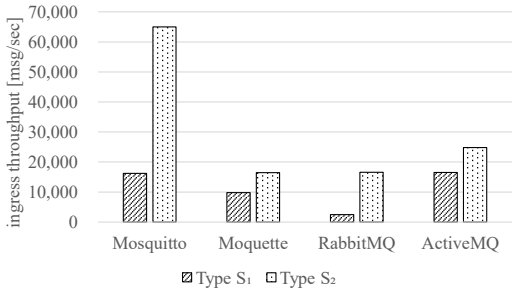
When using type S_1 server, the performance of ActiveMQ and Mosquitto are the tops. ActiveMQ is slightly larger, but almost even. Regarding type S_2 server, Mosquitto is the largest and its egress throughput reaches over $600,000$.

Figure 5(c) shows the result of latency. As for latency, the shorter the better. In case of type S_1 server, Mosquitto has the shortest latency. On the other hand, using type S_2 server, every broker has approximately less than 1 millisecond latency. ActiveMQ is the best, but the difference is quite small.

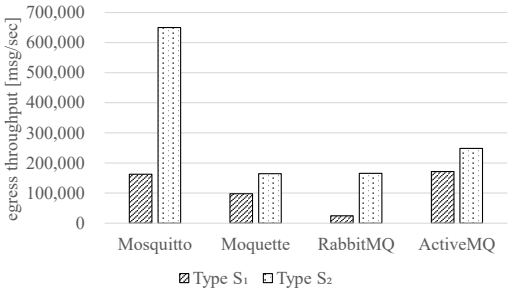
In these measurements, the loss rate was zero for all patterns.

B. Evaluation of ILDM-based cooperation

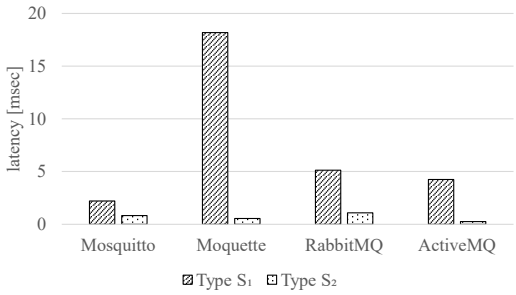
We evaluated the performance of ILDM-based cooperation. Although the principal feature of ILDM is the capability



(a) Ingress throughput

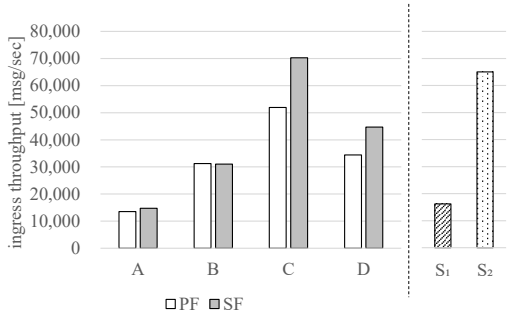


(b) Egress throughput

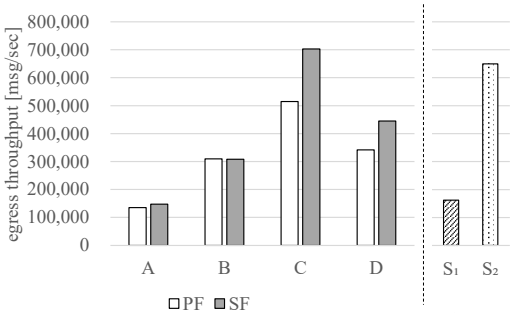


(c) Latency

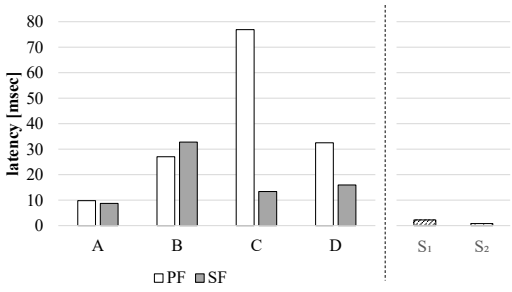
Fig. 5. Evaluation of single brokers.



(a) Ingress throughput



(b) Egress throughput



(c) Latency

Fig. 6. Evaluation of ILDM-based cooperation.

TABLE II
PATTERNS OF MEASURMENTS

Pattern	Description
A	Using one broker with one ILDM node.
B	Using 5 brokers with ILDM. t-clients are placed with no locality.
C	Using 5 brokers with ILDM. t-clients are placed with high locality.
D	Using 5 brokers with ILDM. t-clients are placed with low locality.

of connecting heterogeneous brokers, we used one kind of broker to clarify the performance characteristics of ILDM itself. We chose Mosquitto because it indicated relatively better performance among the four brokers in Section V-A.

Table II states the patterns of measurements. In these patterns, pairs of a broker and an ILDM node are placed on one or five S_1 servers.

Pattern B, C, and D use 5 pairs of a broker and an ILDM node connected in a row. Each ILDM node has the same number of local t-clients, i.e., 20 t-clients. These patterns have a difference of locality of those 100 t-clients. The number of t-clients placed on each ILDM node is as follows:

pattern B: two publishers and two subscribers for every five topics.
pattern C: 10 publishers and 10 subscribers of a topic.
pattern D: Eight publishers and eight subscribers of a topic, one publisher and one subscriber of a different topic, one publisher and one subscriber of another different topic.

In pattern B, C, and D, each of five ILDM nodes has a l-client as a subscriber of topic6. Only one ILDM node placed at the end of the list of the five ILDM nodes has one more l-client as a publisher of topic6. Hence, five data of latency are obtained every second in the measurement time of 60 seconds.

Figure 6(a) and 6(b) shows the results of throughput. "PF" and "SF" in the legend denote the cooperation algorithms. The results of single mosquitto broker are depicted again for

comparison, on the right side of the graphs. Same as results in section V-A, egress throughputs are almost equal to ingress throughputs multiplied by the sp-ratio 10.

By comparing patterns A and S_1, we can see that the overhead of using an ILDM node was suppressed to approximately 10 percent. Results of pattern B, C, and D indicate that ILDM-based cooperation can provide better throughput compared with using a single broker. Especially in pattern C with SF method, the throughput overtook the case of using a single broker on type S_2 server. Since the spec of type S_2 is quite higher than type S_1, this is an impressive result.

It can be said that locality of placing clients affects the performance, by comparing patterns B, C and D. High locality made throughput larger, especially with SF method. This is due to the characteristic of SF method described in Section III-D. Considering edge-heavy data, having high locality of utilization, such tendency could be effective.

Figure 6(c) shows the result of latency. Here also the results of single mosquitto broker are depicted again for comparison. Basically the patterns using multiple brokers are inferior, because a PUBLISH message is forwarded with multi-hop until it arrives at corresponding subscribers.

Patterns A shows approximately 10 msec. Although this is larger than S_1, the result is considered not to impair the effect of reducing latency in the edge-based architecture, since RTT between IoT devices and data centeres could be over 100 msec if it across different countries.

In pattern B, both cases of PF and SF seem to have the same load of throughput. Therefore, the latency of SF method is a little longer due to its complicated processing compared to PF method probably. On the other hand, pattern C and D show that latency of PF method is longer than that of SF method. The reason for this is considered that more redundant messages are propagated in PF method compared to SF method. Pattern C is the case with the highest throughput, so that brokers and ILDM nodes running with PF method tend to be busy and take much time for handling PUBLISH messages.

In these measurements, the loss rate was zero for all patterns.

VI. Related study

Dynomite [12] makes existing non distributed data stores, e.g., Redis and Memcached, into a distributed data store. The aim is to provide high availability and resiliency on storage engines which do not have those functionalities. BondFlow [13] proposes a system enables encapsulated web services to interconnect. These are similar to ILDM from the viewpoint of modularizing functionality of interwork, while the target is different from ILDM.

There are existing methods of topic-based pub/sub with mesh-based topologies [14] [15], unlike that PF and SF suppose a tree structure which does not include closed paths. PIQT [16], which is based on PIAX [17], is one of the implementations using such mesh-based methods. We can obtain better characteristics like scalability and reliability by implementing these methods in ILDM, though such methods may occur additional overheads e.g., maintaining complex routing tables.

As far as we know, there are no existing proposals of connecting heterogeneous MQTT brokers.

VII. Conclusion

In this paper, we proposed a novel mechanism called ILDM which enables heterogeneous MQTT brokers to cooperate with each other. The APIs provided by ILDM enable to develop a variety of cooperation algorithms easily. Two basic algorithms, PF and SF, and a practical benchmark method for MQTT brokers were also presented. The benchmark method ensures that the error ratios of performance are no more than 5 percent.

We evaluated the feasibility of ILDM with the benchmark method. By connecting five brokers via ILDM, the throughput increases approximately two to four times than using a Mosquitto broker alone. This result indicates that the architecture based on edge brokers is useful for reducing consumption of cloud resources.

Our future work includes the following:

- Evaluate with practical environments, including the difference of latency between the cloud and the edge.
- Confirm characteristics of performance with combinations of different kinds of brokers.
- Develop more scalable cooperation algorithms with consideration for fault tolerance.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[3] A. Popov, A. Proletarsky, S. Belov, and A. Sorokin, "Fast prototyping of the internet of things solutions with ibm bluemix," in *Hawaii International Conference on System Sciences*, 2017, pp. 1064–1072.

[4] D. Okanohara, S. Hido, N. Kubota, Y. Unno, and H. Maruyama, "Krill: an architecture for edge heavy data," in *Third Workshop on Architectures and Systems for Big Data*, 2013.

[5] Mosquitto, "mosquitto.org (accessed 2017-05-25)."

[6] HiveMQ, "www.hivemq.com (accessed 2017-05-25)."

[7] MQTT, "mqtt.org (accessed 2017-05-25)."

[8] SurgeMQ, "github.com/influxdata/surgemq (accessed 2017-05-25)."

[9] Moquette, "github.com/andsel/moquette (accessed 2017-05-25)."

[10] RabbitMQ, "www.rabbitmq.com (accessed 2017-05-25)."

[11] ActiveMQ, "activemq.apache.org (accessed 2017-05-25)."

[12] Dynomite, "github.com/Netflix/dynomite (accessed 2017-05-25)."

[13] J. Balasooriya, M. Padhye, S. K. Prasad, and S. B. Navathe, "Bondflow: A system for distributed coordination of workflows over web services," in *IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 121a–121a.

[14] R. Banno, S. Takeuchi, M. Takemoto, T. Kawano, T. Kambayashi, and M. Matsuo, "Designing overlay networks for handling exhaust data in a distributed topic-based pub/sub architecture," *Journal of Information Processing*, vol. 23, no. 2, pp. 105–116, 2015.

[15] Y. Teranishi, R. Banno, and T. Akiyama, "Scalable and locality-aware distributed topic-based pub / sub messaging for iot," in *IEEE Global Communications Conference*, 2015, pp. 1–7.

[16] P. distributed pub/sub broker, "piqt.org (accessed 2017-05-25)."

[17] Y. Teranishi, "Piax: Toward a framework for sensor overlay network," in *IEEE Consumer Communications and Networking Conference*, 2009, pp. 1–5.