A scalable IoT data collection method by shared-subscription with distributed MQTT brokers

Ryohei Banno[0000−0001−7268−4657] and Toshinori Yoshizawa†

Kogakuin University, Tokyo, JAPAN
banno@computer.org

Abstract. Internet of Things (IoT) systems like a smart factory and a smart city require collecting a massive amount of data from sensors. MQTT is a promising protocol for such uses due to its lightweight design and loose-coupling nature by publish/subscribe messaging model. On the other hand, there is an issue that a broker could be a performance bottleneck. Even though it is not the case, reception by a subscriber might not catch up with the amount of data from the broker. In this paper, we propose a scalable IoT data collection method with distributed MQTT brokers. By shared-subscription functionality, which appeared in MQTT version 5.0, the proposed method enables an application to receive a massive amount of IoT data. To evaluate the proposed method, we have developed a load testing tool named MQTTLoader. Experimental results show that the proposed method can improve the throughput compared to conventional ways.

Keywords: MQTT · IoT · Publish/subscribe · Distributed systems

1 Introduction

Internet of Things (IoT) has been developing in various application fields. For example, in Industrie 4.0, monitoring the condition of industrial robots enables preventive maintenance and reduces unexpected downtime [8]. It is made possible by collecting data such as operation time and enclosure temperature from industrial robots. Another example is a smart city, where sensors provide environmental data to realize various services, e.g., relieving traffic congestion and lowering damages caused by natural disasters.

For collecting data from IoT devices, MQTT [14] is a promising communication protocol. It provides loose-coupling nature by publish/subscribe messaging model [7], in addition to its small size of header that contributes to traffic reduction and power saving.

However, an MQTT broker or a subscriber could be a performance bottleneck. Figure 1 illustrates a typical architecture of collecting data by MQTT. A massive amount of data from publishers concentrates on the broker, and thus

† Present affiliation: VINX Corp.

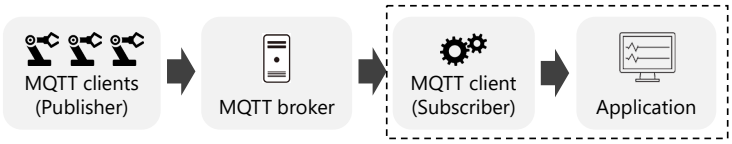


Fig. 1. Collecting IoT data by MQTT

it might not accept all published data. Even though the broker can process all the data, the subscriber might not catch up with the amount of data from the broker.

To tackle this issue, we propose a scalable IoT data collection method with distributed MQTT brokers. In the proposed method, an application can use more than one subscriber to receive data by shared-subscription functionality, which appeared in MQTT version 5.0 [16].

2 Related work

Since an MQTT broker could be a performance bottleneck, there are existing studies to extend the performance of brokers.

muMQ [17] is a high-performance MQTT broker. It efficiently exploits multi-core CPUs by an event-driven I/O mechanism and avoids the kernel overhead by DPDK. muMQ can improve the performance of a single broker, though it does not obtain horizontal scalability and requires hardware supporting DPDK.

MQTT-ST [13] and ILDM [3, 4] are methods to utilize multiple MQTT brokers by connecting them and forming a delivery tree. They are effective if the placement of publishers and subscribers has a high locality, i.e., publishers and subscribers of the same topic tend to connect to the same broker. On the other hand, considering the case to collect IoT data from all publishers as shown in Fig. 1, it is hard to obtain load distribution as well as causing high latency due to multi-hop forwarding among brokers.

Detti et al. [5] have proposed a method to reduce the traffic among brokers by making each subscriber connect to multiple brokers. It is similar to our method from the viewpoint that an application utilizes multiple connections to brokers to receive data in which it is interested. However, there is a difficulty in load distribution if a topic has a massive amount of data.

3 Scalable IoT data collection with MQTT brokers

We propose a scalable IoT data collection method with distributed MQTT brokers. Figure 2 shows an overview of the proposed method. In the proposed method, we use multiple brokers for load distribution. In addition, each application uses multiple subscribers to receive IoT data in parallel.

We assume a large number of publishers. Each publisher connects to one of the brokers. The broker to connect to is decided by Domain Name System (DNS)

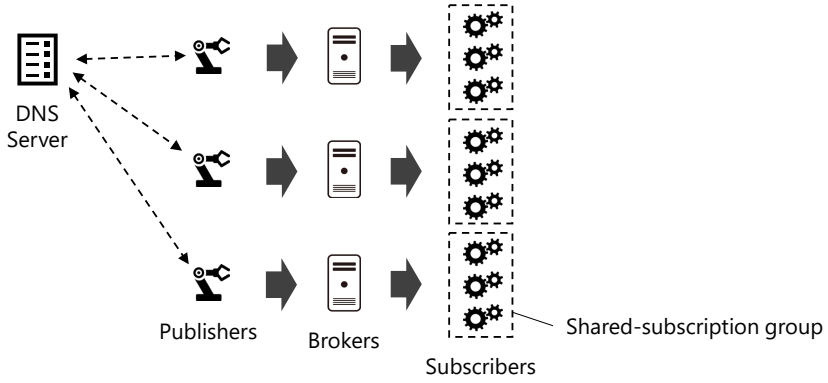


Fig. 2. Overview of proposed method

round-robin, i.e., a publisher does not need additional functionality except for specifying the broker by a domain name.

An application uses subscribers more than or equal to the number of brokers. Each subscriber connects to one of the brokers. We assume that the broker to connect to is decided as the configuration of the application by its user. Subscribers connecting to the same broker make a shared-subscription group.

Shared-subscription functionality has appeared in MQTT version 5.0 [16]. It enables to group subscribers so that they share receiving messages of a topic. That is, each message of the topic is delivered to one of the subscribers participating in the group. This could help load distribution and improve the throughput of subscribers.

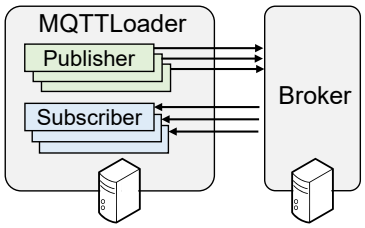
Collecting data by shared-subscription is similar to the consumer group mechanism of Apache Kafka [1,11], which is proprietary software. In contrast, MQTT is an open standard protocol by OASIS [16] and an ISO recommendation [10].

4 Load testing tool

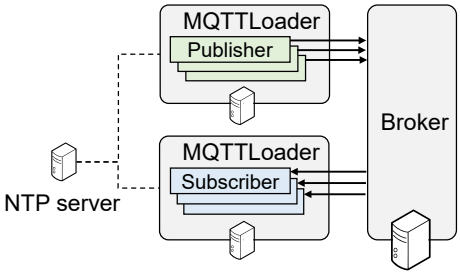
The performance of MQTT brokers holds interest from engineers and researchers since they could be bottlenecks, as we mentioned in Section 2. However, it has not been sufficiently clarified what characteristics actual MQTT brokers have and how we can appropriately measure and analyze their performance, especially for the MQTT v5.0.

To evaluate MQTT v5.0 brokers and the proposed method, we have developed a load testing tool named MQTTLoader [2, 15]. It is implemented in Java. Binary files, source codes, and documents are publicly available on GitHub [15].

Figure 3a shows an overview of MQTTLoader. It generates multiple MQTT clients (publishers and subscribers) and applies a load on a broker based on specified parameter settings. Examples of parameters are listed in Table 1. Besides the list, various parameters are provided, such as QoS level, Retain flag, payload size, publish interval, ramp-up/ramp-down time, etc.



(a) Running on single-host



(b) Running on multi-host

Fig. 3. Overview of MQTTLoader

Table 1. Example of parameters

Parameter	Description
broker	Broker address.
mqtt_version	MQTT version.
num_publishers	Number of publishers.
num_subscribers	Number of subscribers.
shared_subscription	Enable shared-subscription.
ntp	NTP server address.

We can run MQTTLoader either on a single host machine or multiple host machines like Fig. 3b. Running both publishers and subscribers on a single host may cause mutual influence, e.g., receiving load of subscribers lowers the throughput of publishers. By running publishers and subscribers separately on different hosts, we can avoid such mutual influence.

Latency is the time from sending out by a publisher to receiving by a subscriber. To calculate the latency, each message has a timestamp of sending out in its payload. The latency is calculated by the difference between this timestamp and when a subscriber receives the message. In the case of running MQTTLoader on multi-hosts, MQTTLoader obtains time information from the specified Network Time Protocol (NTP) server and uses it for the calculation to avoid the influence of the time offset among the hosts.

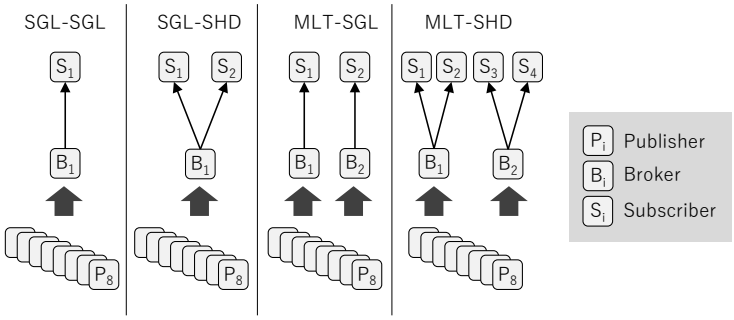


Fig. 4. Experimental configurations

Table 2. Hardware information of experimental environment

	Publisher	Broker, Subscriber
CPU	Core i9 10900K	Celeron N3350
Memory	64GB	4GB
OS	Ubuntu 20.04	Ubuntu 20.04

5 Evaluation

To evaluate the proposed method, we conducted experiments to measure the following performance characteristics.

Throughput (publishers)
Average throughput between publishers and brokers.

Throughput (subscribers)
Average throughput between brokers and subscribers.

Latency
Average time from publishers to subscribers.

We use MQTTLoader v0.7.2 for publishers and subscribers. Although there are various MQTT broker products [6, 9, 12], we choose Mosquitto (v1.6.9 − 1) since it is widely used and known for its superior performance.

We set the number of publishers 8. Each publisher sends out messages with no interval, where each message has a 600 bytes payload. Each measurement takes 60 seconds and it is conducted three times repeatedly.

Regarding brokers and subscribers, we use four cases in Fig. 4 for comparison:

SGL-SGL
Using a single broker B_1 and a single subscriber S_1.

SGL-SHD
Using a single broker B_1 and two subscribers S_1 and S_2, which consists of a shared-subscription group.

MLT-SGL
Using two brokers B_1 and B_2 and two subscribers S_1 and S_2 without shared-subscription.

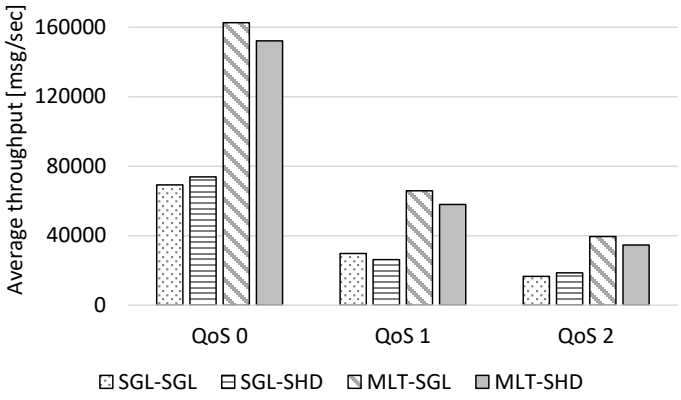


Fig. 5. Publisher throughput

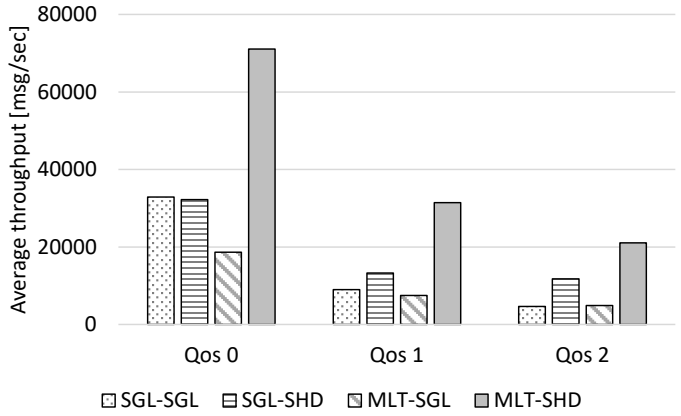


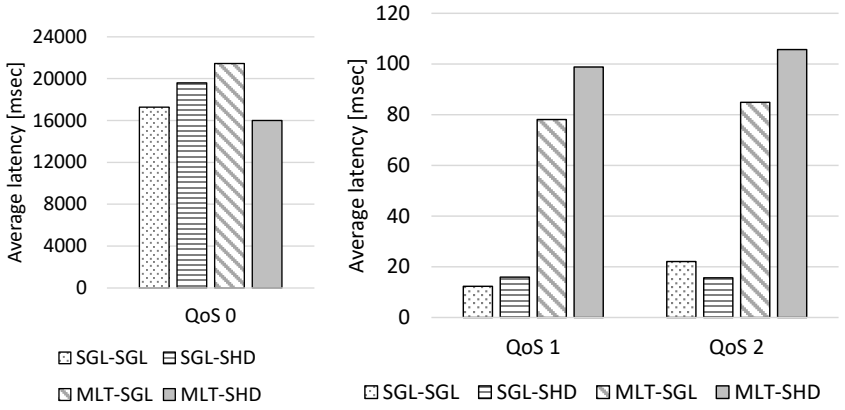
Fig. 6. Subscriber throughput

MLT-SHD
Using two brokers B_1 and B_2 and four subscribers S_1, S_2, S_3, and S_4. The pair S_1 and S_2 and the pair S_3 and S_4 form a shared-subscription group, respectively.

Table 2 shows the hardware information of the host machines. Note that the pair S_1 and S_2 and the pair S_3 and S_4 run in one host machine.

5.1 Throughput

Figure 5 and Figure 6 show the throughput of publishers and the throughput of subscribers, respectively. Overall, the former is larger than the latter. This means that the amount of published data exceeds the capability of brokers.



(a) QoS level 0 (b) QoS level 1 or 2

Fig. 7. Latency

When focusing on the throughput of subscribers, MLT-SHD, i.e., the proposed method, achieves the largest throughput. That is, using multiple brokers and multiple subscribers is effective for collecting a massive amount of IoT data.

On the other hand, in Fig. 5, MLT-SGL gets larger throughput than MLT-SHD. One possibility is that the load of processing shared-subscription inside each broker influences the receiving process of the broker.

5.2 Latency

Figure 7 shows the results of latency. Regardless of the four cases, QoS 0 traffic causes significantly long latency. In the case of QoS 1 and 2, MLT-SHD has the largest latency. It possibly appears in return for the high throughput.

Note that the latency could increase according to the elapsed time because the brokers are considered to be overloaded as mentioned above.

6 Conclusion

In this paper, we proposed a scalable IoT data collection method with distributed MQTT brokers. In the proposed method, each application uses multiple subscribers with shared-subscription functionality to receive IoT data in parallel. To evaluate the proposed method, we developed a load testing tool MQTTLoader. By the experiments with Mosquitto and MQTTLoader, we found that the proposed method can improve the throughput compared to conventional ways. On the other hand, the latency of the proposed method tends to become large. Future work includes a detailed analysis of those experimental results and improvement of the latency.

Acknowledgments

This work was supported by JSPS KAKENHI Grant No.19K20253.

References

1. Apache Kafka: `https://kafka.apache.org/` (accessed July 7, 2021)
2. Banno, R., Ohsawa, K., Kitagawa, Y., Takada, T., Yoshizawa, T.: Measuring Performance of MQTT v5.0 Brokers with MQTTLoader. In: Proc. IEEE Consumer Communications & Networking Conference. pp. 1–2 (2021)
3. Banno, R., Sun, J., Fujita, M., Takeuchi, S., Shudo, K.: Dissemination of edge-heavy data on heterogeneous MQTT brokers. In: Proc. IEEE International Conference on Cloud Networking. pp. 1–7 (2017)
4. Banno, R., Sun, J., Takeuchi, S., Shudo, K.: Interworking Layer of Distributed MQTT Brokers. IEICE Transactions on Information and Systems **E102.D**(12), 2281–2294 (2019)
5. Detti, A., Funari, L., Blefari-Melazzi, N.: Sub-linear scalability of mqtt clusters in topic-based publish-subscribe applications. IEEE Transactions on Network and Service Management **17**(3), 1954–1968 (2020)
6. EMQ X: `https://www.emqx.io/` (accessed July 7, 2021)
7. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The Many Faces of Publish/Subscribe. ACM Computing Surveys **35**(2), 114–131 (2003)
8. FANUC Corp.: Field system. `https://fanuc.co.jp/en/product/field/index.html` (accessed July 7, 2021)
9. HiveMQ: `https://www.hivemq.com/` (accessed July 7, 2021)
10. ISO Central Secretary: Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1. Standard ISO/IEC 20922:2016, International Organization for Standardization (2016)
11. Kreps, J., Narkhede, N., Rao, J.: Kafka: A distributed messaging system for log processing. In: Proc. International Workshop on Networking Meets Databases. pp. 1–7 (2011)
12. Light, R.A.: Mosquitto: server and client implementation of the MQTT protocol. Journal of Open Source Software **2**(13), 265 (2017)
13. Longo, E., Redondi, A.E., Cesana, M., Arcia-Moret, A., Manzoni, P.: Mqtt-st: a spanning tree protocol for distributed mqtt brokers. In: Proc. IEEE International Conference on Communications. pp. 1–6 (2020)
14. MQTT: `https://mqtt.org/` (accessed July 7, 2021)
15. MQTTLoader: `https://github.com/dist-sys/mqttloader` (accessed July 7, 2021)
16. OASIS Standard: MQTT Version 5.0 (2019)
17. Pipatsakulroj, W., Visoottiviseth, V., Takano, R.: mumq: A lightweight and scalable mqtt broker. In: Proc. IEEE International Symposium on Local and Metropolitan Area Networks. pp. 1–6 (2017)