EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

Managing Trade-off Between Subscription Load and Latency in Vehicular Edge Platform

Takumu Takada[1,*], Ryohei Banno[1,†]

[1]Kogakuin University, Tokyo, Japan

Abstract

Adding connectivity to vehicles is attracting much attention toward developing smarter vehicles such as autonomous cars. To obtain and utilize real-time information from vehicles, techniques that combine the concept of edge computing and publish/subscribe messaging model have been proposed. However, there is an issue that the increase in the number of edge servers imposes a heavy load upon a subscriber for managing connections to them. To address this issue, we propose an edge-based platform with the functionality of adjusting the number of connections to edge servers. Experimental results clarify the trade-off characteristic between subscription load and latency.

Received on 11 January 2022; accepted on 21 April 2022; published on 29 April 2022

Keywords: Edge computing, Publish/Subscribe, MQTT, IoT, Vehicular platform

doi:10.4108/eetiot.v8i28.708

1. Introduction

Recently many studies have been carried out for realizing smarter vehicles e.g., autonomous cars and traffic congestion control. Such vehicular functionality requires gathering real-time information like the speed, position, and surrounding environment from vehicles over a wide area.

To this end, there are existing studies that combine the concept of edge computing and publish/subscribe messaging model [1, 2]. Edge computing [3] enables to reduce latency and obtain better load distribution, whereas Publish/subscribe messaging model [4] allows clients to exchange messages in a loosely coupled manner. By running distributed brokers on edge servers, vehicles can publish their data to applications through the brokers. For example, a vehicle can publish traffic congestion information around its current location. For the use in a cloud application like generating a wide-area traffic information map, the application can gather data via the brokers with avoiding load concentration on a single broker. An edge broker also lowers latency in an edge application like information sharing among neighboring vehicles.

*Present affiliation: Hino Motors, Ltd.
†Corresponding author. Email: banno@computer.org

However, there is a problem that the increase in the number of edge brokers imposes heavy load upon a subscriber for managing connections to them. The existing studies assume that each subscriber maintains a connection to every edge broker. Accordingly, a large number of edge brokers involves a significant load on a subscriber to manage connections to them. To address this issue, we propose an edge-based vehicular platform with functionality of adjusting the number of connections to edge brokers.

2. Related work

Aoyama et al. [1] proposed a concept of double edge architecture for connected vehicles that fuses edge computing and publish/subscribe messaging model. Brokers are placed at the edge hierarchically, i.e., out-car edge and in-car edge. Subscribers in the cloud connect to those brokers so that they can receive vehicular information over a wide area. UPub [2] also has an architecture combining edge computing and publish/subscribe messaging model, while it assumes not only vehicles but general mobile devices. Mobile applications behave as publishers whereas cloudlets placed at the fog behave as subscribers. In these existing studies, each subscriber needs to maintain connections to every edge broker. Therefore, the increase in

the number of brokers causes an enlarged load on subscribers.

One possible approach to suppress an increase in the load is clustering the brokers and making each subscriber connects to one of them, as proposed in [5]. However, it assumes that each subscriber connects to only one broker node. This impose an increase in latency when a subscriber receives data originated in a publisher that connects to a different broker from the subscriber.

VerneMQ [6] is also capable of clustering. A client node can connect to arbitrary multiple brokers and receive data originated in a publisher that connects to those brokers without forwarding among brokers[1]. However, multiple connections can cause redundant traffic among brokers. For example, if subscriber S_1 connects to two brokers B_1 and B_2 and subscriber S_2 connects to broker B_1, data on broker B_3 can be forwarded to both B_1 and B_2 though B_3 essentially can deliver all subscribers by forwarding it to B_1. This is because the connection used to deliver the data to S_1 is randomly chosen.

From these, maintaining connections to every edge broker could increase the load of subscribers, whereas using less connections imposes an increase in latency unless allowing redundant traffic among brokers. To enable to adjust the trade-off between the load and latency without redundant traffic, we propose an edge-based platform in which a subscriber can selectively connect to an arbitrary number of brokers. This enables each application subscribing to data to connect to multiple brokers according to its requirements. For example, if an application wants to obtain data with low latency from a specific area, it can directly connect to brokers taking charge of the area. If the application wants to lower the load of maintaining connections, it can reduce the number of connections while keeping the ability to obtain data from all brokers.

3. Proposed method

We assume that brokers are placed at the edge over a wide area and applications in the cloud behave as subscribers. Vehicles can be either publishers or subscribers, though how to determine the broker to connect to is out of the scope of this paper.

Figure 1 shows the architecture of the proposed method. Different from the existing studies [1, 2] as depicted on the left, the proposed method depicted on the right enables a subscriber to connect to one or more arbitrary edge brokers. Data from a publisher connecting to a broker that is not connected to the subscriber is delivered via the brokers that are connected to it, as shown by a red arrow.

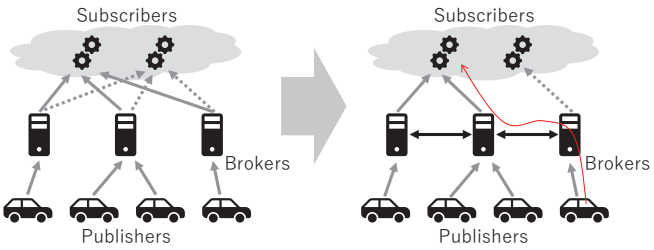

Figure 1. Architecture of the proposed method

In the proposed method, each broker connects to all other brokers. When a subscriber subscribes to a topic on a broker, the broker shares the subscription information among the brokers.

3.1. Behavior of brokers

Hereafter, we refer to a client, i.e., a publisher or a subscriber, directly connected to a broker as a local client of the broker.

Each broker has a subscription table consisting of client information (client identifier, IP address, etc.), a list of brokers that the client connects to, and topics that the client subscribes to. This table is synchronously shared among brokers.

When a broker forwards data to another broker, it contains information about the destination clients[2] for avoiding the delivery of duplicate data.

When a broker receives data from its local publisher, it immediately delivers the data to its local subscribers. When it receives data from another broker, it delivers the data to its local subscribers contained in the destination information of the data. It also forwards the data to other brokers according to the following procedure. At first, it extracts information from its subscription table by excluding its local subscribers. Then, it determines a set of brokers to be forwarded the data by using the extracted information. This determination is made so that the number of brokers in the set is minimized while satisfying the following condition: all subscribers subscribing to the topic of the data are connected to at least one of the brokers in the set. Finally, the broker forwards the data to the brokers in the set. Upon forwarding, destination information is attached for each of the brokers such that the number of destination clients becomes as equal as possible. By these, we can avoid redundant traffic among brokers like in VerneMQ as mentioned in Section 2.

[1] This is realized by using shared subscription and configuring the policy of shared subscription to "prefer_local".

[2] Since each broker has the exact client information in its subscription table, there is a possibility to compress the destination information by techniques like Bloom Filter, though it needs to consider handling false positives.

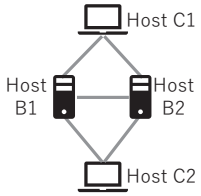

Figure 2. Basic topology of the experiments

4. Evaluation

To clarify the necessity of the proposed method, we conducted experiments for measuring the load of a subscriber and the latency from a publisher to the subscriber.

In the following experiments, we used the MQTT protocol that is one of the best-known protocols for publish/subscribe messaging. We used Mosquitto 1.6.12 [7] for brokers and MQTTLoader 0.7.3 [8, 9] for clients. The publisher sends messages of 100 bytes payload size at 2 milliseconds intervals for 60 seconds. QoS level of both the publisher and the subscriber is set to 0.

Figure 2 shows the basic topology of the experiments. We used two hosts for brokers (B1 and B2) and one or two hosts for clients (C1 and C2). For forwarding messages between the two brokers, we used the bridge function of Mosquitto.

4.1. Subscription load

We conducted an experiment for measuring the load of the subscriber. In this experiment, C1 is the subscriber and C2 is the publisher. The spec of B1, B2, and C1 is as follows: Celeron N3350 CPU, 4GB memory, Ubuntu 20.04 OS. The spec of C2 is as follows: Core i5-10400 CPU, 32GB memory, Windows 10 OS.

We compared the following two cases:

- **Two connections:** C1 connects to both B1 and B2. C2 connects to both B1 and B2 and sends out messages to them equally.
- **One connection:** C1 connects to B1. C2 connects to B2 and sends out all messages to it.

Figure 3 shows the result. Note that we removed the first five seconds and last five seconds from the 60 seconds of measurement time as ramp-up/ramp-down time. From the result, the load of the subscriber is higher when it connects to two brokers than when it connects to one broker. The average CPU usages are 25.83% and 16.30% respectively.

4.2. Latency

We conducted an experiment for measuring the latency between the publisher and the subscriber. In this experiment, C1 behaves as both the subscriber and the publisher to calculate the latency accurately. The spec of B1, B2 is as follows: Celeron N3350 CPU, 4GB memory, Ubuntu 20.04 OS. The spec of C2 is as follows: Core i5-10400 CPU, 32GB memory, Windows 10 OS.

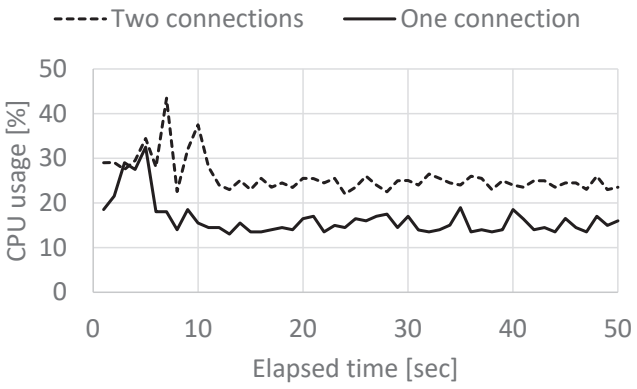

Figure 3. Subscriber load

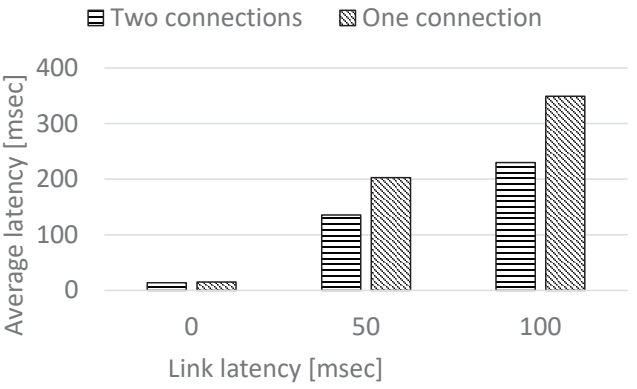

Figure 4. Latency

We compared the following two cases:

- **Two connections:** The subscriber in C1 connects to both B1 and B2. The publisher in C1 connects to both B1 and B2 and sends out messages to them equally.
- **One connection:** The subscriber in C1 connects to B1. The publisher in C1 connects to B2 and sends out all messages to it.

Each measurement was conducted three times and the average latency was calculated.

For every link among the hosts (B1, B2, and C1), we set three patterns of link latency by using the Traffic Control (tc) utility of Linux to reproduce the case that those hosts are placed at a distance; 0 millisecond, 50 milliseconds, and 100 milliseconds.

Figure 4 shows the result. From the result, the latency is larger when the subscriber connects to one broker than when it connects to two brokers. The difference

becomes large especially when the link latency is set to a large value.

Analytical evaluation for latency. In addition to the experimental evaluation, we analytically evaluate the latency. We assume a Jackson network, a well-known queueing network class, with multiple brokers and one subscriber for simplicity. We call the brokers that the subscriber connects to "a connected broker", whereas those not connected to the subscriber "a non-connected broker". We also make the following assumptions.

- The number of brokers: n
- The number of connected brokers: m
- The average arrival rate of each broker: λ messages per second
- The average service rate of each broker: μ messages per second
- Published data on the non-connected brokers are equally forwarded to the connected brokers.

Hereafter, we call $\frac{m}{n}$ the "connection rate" of the subscriber. Since each broker connects to all other brokers in the proposed method, published data on the non-connected brokers are delivered to the subscriber through one connected broker.

The average waiting time from a connected broker to the subscriber is

$$\frac{1}{\mu - \lambda}, \tag{1}$$

whereas that from a non-connected broker is

$$\frac{1}{\mu - \lambda} + \frac{1}{\mu - \frac{n}{m}\lambda}. \tag{2}$$

Therefore, the entire average waiting time W is

$$W = \frac{1}{\mu - \lambda} + \frac{1}{\mu - \frac{n}{m}\lambda}(1 - \frac{m}{n}). \tag{3}$$

Figure 5 shows the impact of connection rate on latency, calculated by the above equation. We set parameters as follows: the size of each message is 10 KBytes, and the network bandwidth is 1 Gbps. Thus, the service rate of each queue is roughly 12, 207 messages per second. The figure shows that a lower connection rate enlarges the latency. Note that this analysis focuses on the brokers and does not consider the network latency since it aims at revealing the tendencies when changing the connection rate.

From these results, we can say that there is a trade-off characteristic between the load of maintaining connections to brokers and the latency from a publisher to a subscriber. The proposed method enables each subscriber to adjust this trade-off according to its situation.

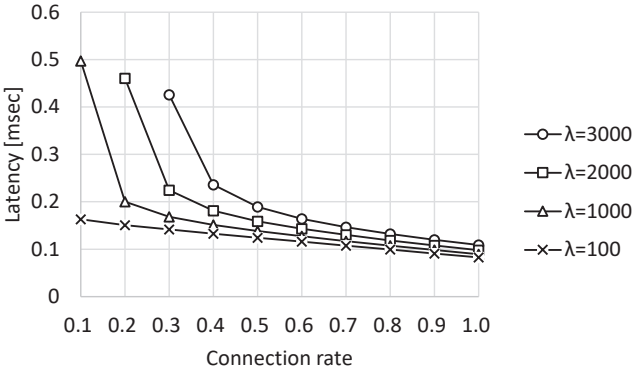

Figure 5. Connection rate impact on latency

5. Conclusion

In this paper, we proposed an edge-based vehicular platform that provides the functionality of adjusting the number of connections from a subscriber to edge brokers. Experimental results show that reducing the number of connections lowers the CPU usage but enlarges the latency. The proposed method enables each subscriber to adjust the trade-off, so that it eases to support various situations of applications. One of the drawbacks is that the application parameter increases; the operator of each application has to determine the number of brokers to connect and select broker(s) to connect. It is desirable to determine the above parameters dynamically according to the load status and topics of interest. Besides, there is also an issue that each broker is forced to manage and exchange more information compared to existing studies. As we mentioned in Section 3.1, there is a possibility to improve by using techniques like Bloom Filter. Our future work includes designing such an efficient cooperation algorithm among edge brokers.

Acknowledgement. This work was supported in part by JSPS KAKENHI Grant Numbers 19K20253 and in part by JST PRESTO Grant Number JPMJPR21P8.

References

[1] Aoyama, M. and Uno, T. (2019) A Concept and Design Method of Double Edge Computing Architecture for Connected Vehicles Software Platform and its Evaluation. In *JSAE Congress (Autumn)*: 1–6. (in Japanese).

[2] Quental, N. (2021) UPub: Enabling Mobility Management for Publish/Subscribe Systems in the Edge. In *Proc. IEEE Consumer Communications and Networking Conference*.

[3] Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016) Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* **3**(5): 637–646.

[4] Eugster, P.T., Felber, P.A., Guerraoui, R. and Kermarrec, A.M. (2003) The Many Faces of Publish/Subscribe. *ACM Computing Surveys* **35**(2): 114–131.

[5] Banno, R., Takeuchi, S., Takemoto, M., Kawano, T., Kambayashi, T. and Matsuo, M. (2015) Designing overlay networks for handling exhaust data in a distributed topic-based pub/sub architecture. *Journal of Information Processing* **23**(2): 105–116.

[6] VerneMQ, https://vernemq.com/ (accessed Apr. 26, 2022).

[7] Light, R.A. (2017) Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software* **2**(13): 265.

[8] Banno, R., Ohsawa, K., Kitagawa, Y., Takada, T. and Yoshizawa, T. (2021) Measuring Performance of MQTT v5.0 Brokers with MQTTLoader. In *Proc. IEEE Consumer Communications and Networking Conference*.

[9] MQTTLoader, https://github.com/dist-sys/mqttloader/ (accessed Apr. 26, 2022).