Acceleration of MQTT-SN protocol using P4

1st Ryohei Banno
Kogakuin University
Tokyo, JAPAN
banno@computer.org

2nd Koki Osawa
Kogakuin University
Tokyo, JAPAN

***Abstract*—MQTT-SN is one of the protocols attracting attention for Internet of Things (IoT) systems. It is suitable for collecting data from sensor devices due to its lightweight protocol design and loose coupling nature derived from the publish/subscribe model. To handle enormous IoT data, the performance of MQTT-SN brokers is a significant concern. Typical approaches to improving performance are using multiple brokers for load distribution and bringing out the hardware performance. However, the former involves larger latency due to cooperation overhead among brokers, and the latter has less flexibility than generic software brokers. To address these problems, we propose PAMS, an in-network acceleration method of the MQTT-SN protocol. PAMS makes a network switch perform a part of broker functionality by using P4, a leading data plane programming language. Although PAMS introduces in-network acceleration for performance improvement, it uses a software broker for the principal functionality of MQTT-SN so that we obtain flexibility as well. From analytical and quantitative evaluation, we clarify that PAMS lowers the load of the broker and reduces latency.**

***Index Terms*—MQTT-SN, IoT, Publish/subscribe, P4, In-network processing, Software-defined networking**

I. Introduction

Internet of Things (IoT) is rapidly growing in various fields, e.g., smart factories [1] and smart warehouses [2]. A typical IoT system consists of connected devices such as sensors and collects data for subsequent utilization like monitoring devices and anomaly detection.

MQTT-SN [3], [4] and MQTT [5] are suitable protocols for such exchange of IoT data. They are based on the publish/subscribe model [6] so that they have loose coupling nature, i.e., they enable relationships among devices and applications to vary flexibly. The principle difference between MQTT-SN and MQTT is that the former relies on UDP, whereas the latter works on TCP. Besides, MQTT-SN reduces the message size compared to MQTT. Note that we call an MQTT-SN server "a broker" in this paper, whereas it is referred to as "a gateway" in the specification [4]. This is because we focus on the communication among MQTT-SN clients via a stand-alone MQTT-SN server like Mosquitto RSMB [7].

To handle increasing IoT data, the performance of MQTT-SN brokers is a significant concern. Typical approaches for improving the performance of messaging brokers are using multiple brokers for load distribution [8]–[11] and making the best possible use of hardware performance [12]. However, the former involves larger latency due to cooperation overhead among brokers, and the latter has less flexibility than generic software brokers.

To address these issues, we propose a P4-based Acceleration method for MQTT-SN brokers (PAMS). It makes a network switch perform a part of broker functionality by using P4 [13], a leading data plane programming language. PAMS uses a generic software broker for the principal functionality of MQTT-SN while it introduces the in-network acceleration so that we obtain both flexibility and better performance.

The remainder of this paper is organized as follows. Section II introduces related studies on performance improvement of messaging brokers. Section III explains PAMS, while Section IV presents the analytical and experimental evaluation. Finally we conclude this paper in Section V.

II. Related work

There are existing studies for improving the performance of brokers of MQTT-SN and MQTT.

Hunkeler et al. [3] introduce two kinds of MQTT-SN brokers: a transparent broker and an aggregating broker. The former maintains a connection to an MQTT broker for each MQTT-SN client, whereas the latter maintains one connection to an MQTT broker for all MQTT-SN clients. The aggregating broker is suitable to obtain scalability since we can construct a tree topology from multiple aggregating brokers and an MQTT broker; it helps to distribute the load of maintaining the connection of clients. Likewise, several studies intend to use multiple brokers for load distribution. MQTT-ST [8] is a method to form a spanning tree among multiple brokers. Similarly, ILDM [11] uses multiple brokers by connecting them and creating a delivery tree. These methods can improve throughput for some situations, but multi-hop forwarding among brokers may cause high latency. Detti et al. [9] have proposed a scheme that reduces inter-broker communication to mitigate such issues due to cooperation among brokers. However, it requires subscribers to have additional functionality of managing multiple sessions with brokers. That is, it could impair some strengths of MQTT, such as the inter-operability given by the standardized protocol and simplified connection management in clients derived from the publish/subscribe model.

This work was supported by JST, PRESTO Grant Number JPMJPR21P8 and by JSPS KAKENHI Grant Number 19K20253.

Another approach is to bring out the hardware performance. Pipatsakulroj et al. propose muMQ [12], which is a high-performance MQTT broker. It efficiently utilizes multicore CPUs and avoids kernel overhead by using DPDK. Although it achieves better throughput and latency, it is less flexible than generic software brokers, e.g., it requires hardware supporting DPDK and thus limits the choice of broker hardware. Besides, users cannot selectively use various software broker products in exchange for improved performance.

Unlike the existing studies, PAMS provides both flexibility and better performance almost without increasing latency. Furthermore, PAMS keeps the inter-operability given by the standardized protocol specification, i.e., it does not require clients to have additional functionality so that any implementation following the MQTT-SN specification is applicable.

P4-based in-network processing, which we adopt, has been attracting much attention recently. From this viewpoint, several studies introduce P4 into publish/subscribe messaging [14]–[16]. However, to the best of our knowledge, no studies are directed to utilize P4 to accelerate the MQTT-SN protocol.

Note that some application layer techniques also improve the performance of MQTT-SN and MQTT. Alshantout et al. propose a method using LT codes to improve the performance of the MQTT-SN protocol [17]. Such methods can be used in combination with PAMS.

III. In-network acceleration of MQTT-SN

To improve the performance of MQTT-SN and obtain flexibility, we propose an in-network acceleration method, PAMS. It uses P4 and makes a network switch perform a part of broker functionality.

A. MQTT-SN

MQTT-SN [3], [4] is an application layer protocol based on the publish/subscribe model [6]. In MQTT-SN communication, A publisher, a data sender, sends a Publish message to the broker specifying a topic. A subscriber, a data receiver, sends a Subscribe message to the broker specifying topics of interest. The broker forwards a Publish message to subscribers interested in its topic.

There are four terms that express a topic: a topic ID, pre-defined topic ID, short topic name, and topic name. A topic ID, pre-defined topic ID, and short topic name are two bytes-long. A topic ID and a pre-defined topic ID are short expressions of a topic name. The difference is that Topic IDs need a registration process in advance between publishers and the broker, whereas pre-defined topic IDs do not need such a process, i.e., their meanings are known in advance.

A publisher specifies a topic using two bytes-long topic information: a topic ID, pre-defined topic ID, or short topic name. A subscriber specifies interest by a pre-defined topic ID, short topic name, or topic name. If it specifies a topic name, the corresponding topic ID is notified by the Suback message in response to the Subscribe message. The above scheme allows exchanging data between a publisher and subscribers while suppressing message size with two bytes-long topic information.

A topic name in a Subscribe message may include wildcards, e.g., a topic "foo/bar/#" matches all topics starting with "foo/bar". The broker sends a Suback message with a specific topic ID 0x0000 in this case. Then, when the broker receives a first Publish message of a topic matching the wildcards, it sends a Register message to the subscriber to notify the topic ID. Subsequently, the broker forwards messages of the topic to the subscribers.

In MQTT-SN, a publisher can specify some parameters for each Publish message, such as the QoS level. A higher QoS level involves re-transmission as needed for reliable delivery, while QoS level 0 does not retry with the emphasis on lightweight communication. Besides, a publisher can also set Retain flag. If it is set to "true", the Publish message is stored on the broker and will be sent to a new subscriber.

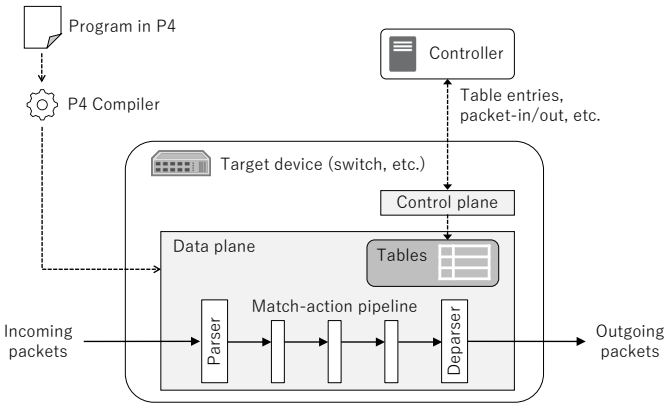

Fig. 1. Overview of P4

B. P4

P4 [13] is a programming language for the data plane of network devices. Conventional software-defined networking (SDN) techniques have a limitation in that the data plane is not programmable. Thus, new protocols and original protocols are not capable of controlling. P4 breaks through this limitation, i.e., it gives programmability to the data plane.

Figure 1 shows the overview of P4. A program written in P4 is converted to a binary specific to a target device by a P4 compiler. By deploying the binary onto the target device, users can run it with their own parser, match-action pipeline, and deparser. Same as conventional SDN like OpenFlow [18], information required for controlling packets, such as table entries, are registered through the control plane.

C. Architecture of PAMS

PAMS makes a network switch perform a part of broker functionality. We assume the switch is placed where all messages between the broker and clients pass, as shown in Figure 2. When receiving a Publish message, it checks whether existing subscribers are interested in the topic of the message.

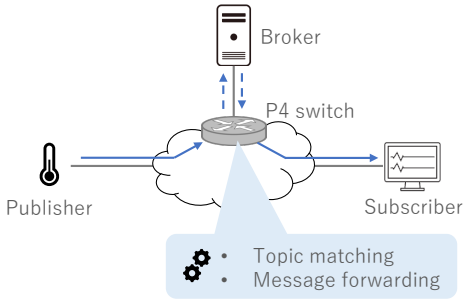

Fig. 2. Assuming architecture of PAMS

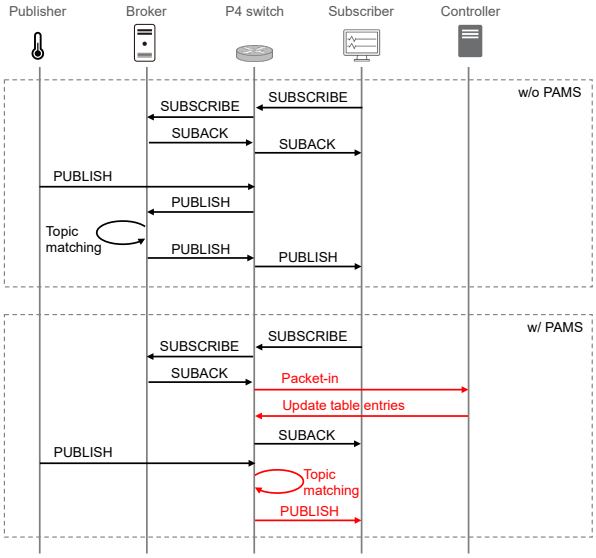

Fig. 3. Difference between with and without PAMS

If so, it forwards the message directly to the corresponding subscribers with rewriting the source information, the publisher, into the broker. Thereby the load of the broker decreases.

Figure 3 shows the basic sequence. Note that we omit some kinds of messages like Connect since they are not essential in showing the fundamental difference between with and without PAMS. In the regular sequence, i.e., without PAMS, all messages are processed by the broker. Contrarily, with PAMS, the switch processes a part of messages. We explain what kind of messages are subjected to in-network processing later. When the switch receives some kinds of MQTT-SN messages such as Suback, it sends a packet-in message to the controller. Subsequently, the controller updates table entries in the switch so that it holds correspondence between topics and subscribers. After that, if a Publish message received by the switch has a topic that the table entries include, the switch forwards the message directly to the corresponding subscribers with rewriting the source information into the broker.

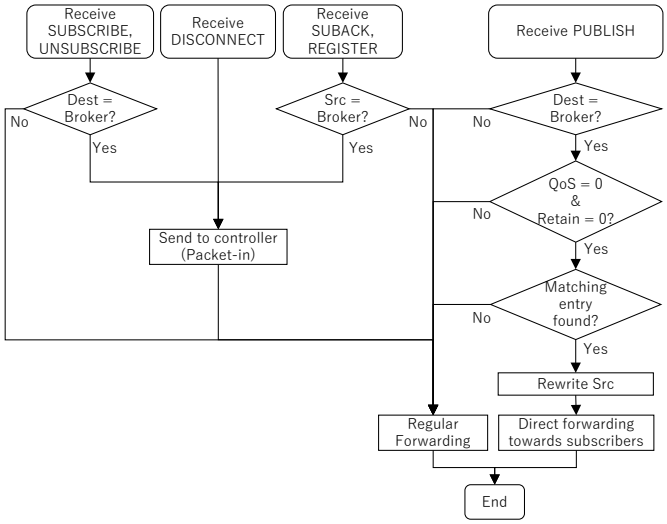

Fig. 4. Handling MQTT-SN messages in P4 switch

D. Acceleration behavior

Since the broker maintains state information for various processes like re-transmission according to the QoS level, in-network acceleration should avoid affecting it. Therefore, PAMS processes MQTT-SN messages within the switch in a manner that does not influence such state information maintained by the broker. Specifically, it targets only Publish messages with QoS level 0 and Retain flag "false".

Figure 4 shows processes in the switch. A Publish message satisfying the conditions, i.e., QoS level is 0 and Retain flag is "false", is checked if the topic in the header matches an entry in the table that maintains the correspondence between topics and subscribers. If there exists a matching entry, the switch rewrites the source information of the Publish message into the broker and then forwards it directly to the corresponding subscribers. Otherwise, it is ordinarily forwarded and handled by the broker as regularly. Besides, the switch sends a packet-in message to the controller when it receives a Subscribe, Unsubscribe, Suback, Register, and Disconnect message.

Within the controller, messages are handled according to the flow shown in Figure 5. The controller basically maintains the following three kinds of tuples:

- ⟨Short topic name, Subscribers⟩
- ⟨Topic ID, Topic name, Subscribers⟩
- ⟨Wildcard topic name, Subscribers⟩

These tuples are information of correspondence between a topic and its subscribers. Since a Publish message has a topic ID (including a pre-defined topic ID) or a short topic name as described in Section III-A, the controller generates table entries with them. Namely, the generated rules determine the switch behavior as follows; if a Publish message includes a specific topic ID or short topic name in its header, the switch rewrites the source information into the broker and forwards it to the corresponding subscribers. The above tuples

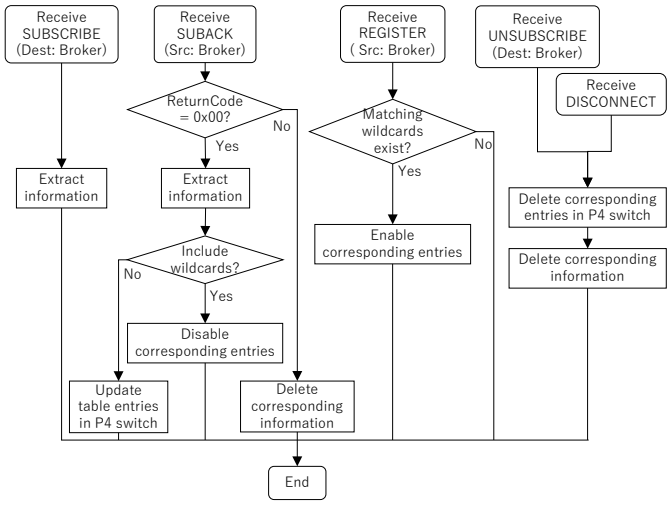

Fig. 5. Behavior of controller

are for managing such correspondence between a topic and the destination subscribers.

When the controller receives a Subscribe message, it extracts topic information T_s (short topic name, topic ID, or topic name), message ID M_s, and source information S_s (IP address and port number).

When the controller receives a Suback message, it first checks the return code. If it is not 0x00, the controller discards the information of corresponding T_s, M_s, and S_s because it means the broker rejects the subscribe request. Otherwise, the controller extracts topic information T_a with its type, message ID M_a, and source information S_a. It then creates or updates a tuple according to the type of T_a: $\langle T_a, \{S_a, ...\}\rangle$ for a short topic name, $\langle T_s, \{S_a, ...\}\rangle$ for the specific topic ID 0x0000, or $\langle T_a, T_s, \{S_a, ...\}\rangle$ for any other topic ID. T_s is a wildcard topic name in the second case, whereas it is a normal topic name in the third case. The association between T_s and T_a can be determined by the identity of message IDs M_s and M_a.

Afterward, if T_a is equal to 0x0000, i.e., the corresponding Subscriber message has a topic name with wildcards, it disables all table entries that match the wildcards. If T_a is not 0x0000, it updates table entries in the switch according to the above tuples. That is, it embeds the following forwarding rule; if the header of a Publish message has the topic information equal to T_a, then forwards the Publish message to S_a in addition to the existing subscribers.

In the above, the reason why disabling all table entries for a wildcard topic name is to allow the broker to issue a Register message. As mentioned in Section III-A, the broker sends a Register message to the subscriber to notify the topic ID. If the related table entries are enabled, Publish messages of the topic ID are processed in the switch and do not reach the broker. Hence, all table entries related to a wildcard topic name are required to be disabled. Note that this behavior does not significantly influence performance because once a new Publish message arrives at the broker, a Register message is issued that triggers enabling the table entries again.

Upon receiving a Register message from the broker, the controller checks if it is sent in response to a Subscribe message with wildcards. If so, and if there are disabled related table entries, it enables them.

When the controller detects a breakaway of a subscriber from a topic by receiving an Unsubscribe message or a Disconnect message, it deletes corresponding table entries in the switch and then deletes related information.

IV. Evaluation

To confirm the effectiveness of PAMS, we evaluate the latency and the number of packets by analyzation of a queueing network model and emulation with Mininet [19].

A. Analytical evaluation of latency

We assume a Jackson network [20], a well-known queueing network class, considering the outgoing interface in each publisher, switch, broker, and subscriber in Figure 2 has a queue.

We also make the following assumptions.

- The number of publishers: n
- The number of subscribers: 1
- The average arrival rate of each publisher: λ messages per second
- The average service rate of each node: μ messages per second
- The average acceleration rate: ω

Note that the acceleration rate indicates the ratio of Publish messages subjected to in-network processing to the total number of Publish messages.

Figure 6 shows the assuming model. We denote a publisher N_i^P, a broker N_1^B, and a switch N_1^S and N_2^S, where the former is the outgoing interface for non-acceleration and the latter for acceleration. The arrival rates of N_1^S and N_1^B are $\lambda(1-\omega)n$, and that of N_2^S is λn. Hence, the expected waiting time from a publisher to the subscriber $E[W]$ is

$$\begin{aligned} E[W] &= \omega\left\{\frac{1}{\mu-\lambda}+\frac{1}{\mu-\lambda n}\right\} \\ &\quad +(1-\omega)\left\{\frac{1}{\mu-\lambda}+\frac{1}{\mu-\lambda(1-\omega)n}\right. \\ &\quad \left.+\frac{1}{\mu-\lambda(1-\omega)n}+\frac{1}{\mu-\lambda n}\right\} \\ &= \frac{1}{\mu-\lambda}+\frac{2(1-\omega)}{\mu-\lambda(1-\omega)n}+\frac{1}{\mu-\lambda n}. \end{aligned}$$

Figure 7 shows the impact of n and ω on latency, calculated by the above equation. We set parameters as follows; λ is one message per second, and the service rate of each queue is $12,207.03125$ messages per second based on the assumption that the size of each message is 10 KBytes and the network bandwidth is 1 Gbps. Although it is a simplified model for grasping the tendency, the figure shows that PAMS could reduce the latency compared to not using it, i.e., in the $\omega = 0$ case. A larger number of publishers tends to involve

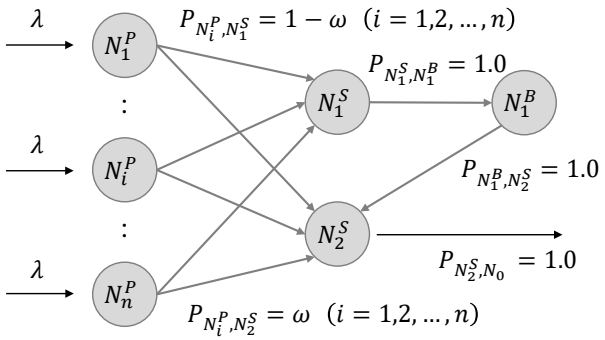

Fig. 6. Queueing network model

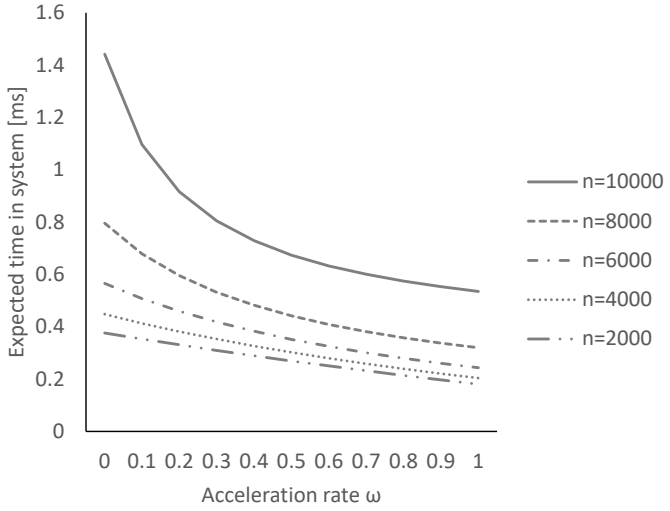

Fig. 7. Expected latency with PAMS

a more significant decrement amount. For $10,000$ publishers, the acceleration rate ω of 0.4 approximately halves the latency.

B. Experimental evaluation

We conducted experiments by emulation environment with Mininet [19]. The network topology includes a publisher, broker, subscriber, and P4 switch, as shown in Figure 8. We use BMv2 [21] for P4 switch and Mosquitto RSMB [7] for the broker. Client implementations are based on MQTT-SN Tools [22]. For the measurement of the number of packets, we use Wireshark. The Mininet environment runs on Ubuntu 16.04.7LTS on Oracle VirtualBox. The virtual machine is assigned two cores and 2GB memory on a desktop computer that has Intel Core i5-10400 CPU, 32GB memory, and Windows 10 Pro OS. Table I lists the software versions.

1) Number of packets: We measured the number of packets on each node. Specifically, we obtained the numbers of packets sent from the publisher, received by the broker, and received by the subscriber with Wireshark. The subscriber subscribes to a topic in advance. The publisher sends 100 Publish messages with the same topic at one-millisecond intervals. The packet size of each Publish message is 100 bytes, QoS level is set to 0, and Retain flag is set to false. Measurement of the number

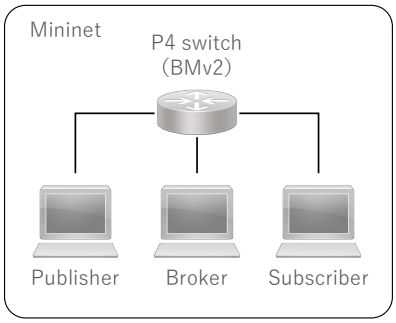

Fig. 8. Network topology in Mininet

TABLE I
SOFTWARE VERSIONS

Software	Version
Mosquitto RSMB	1.3.0.2
mqtt-sn-tools	0.0.6
BMv2	1.13.0
Mininet	2.3.0
Wireshark	2.6.10
VirtualBox	6.1.12

of packets includes messages other than Publish, like Connect and Subscribe.

Figure 9 shows the result. The numbers of packets of the publisher and subscriber are almost the same with and without PAMS. On the other hand, regarding the broker, PAMS significantly reduces the number of packets. This is because Publish messages are processed in the P4 switch and do not reach the broker.

Although the actual effectiveness depends on the ratio of in-network acceleration, it is confirmed that PAMS can lower the load of the broker while it does not influence the behavior of clients from the result.

2) Latency: We also measured the latency from the publisher to the subscriber. Each Publish message has a time stamp of packet transmission time in microseconds in its payload. We obtain the latency by calculating the difference between the above time stamp and packet reception time in the subscriber.

The publisher sends $10,000$ Publish messages with the same topic at one-millisecond intervals. The packet size of each Publish message is adjusted to 100 or 200 bytes by padding, QoS level is set to 0, and Retain flag is set to false.

Figure 10 shows the result for each packet size 100 and 200 bytes. PAMS reduces both the average and median of the latency. In PAMS, Publish messages subjected to in-network processing skip over the broker, and thus the latency could be lower. When not using PAMS, the difference between average and median values is relatively significant, i.e., the latency seems to vary widely. Since PAMS enables skipping over the process in the broker software, it is expected to suppress variations as well as reduce latency.

V. CONCLUSION

In this paper, we propose PAMS as an in-network acceleration method of the MQTT-SN protocol by P4. PAMS

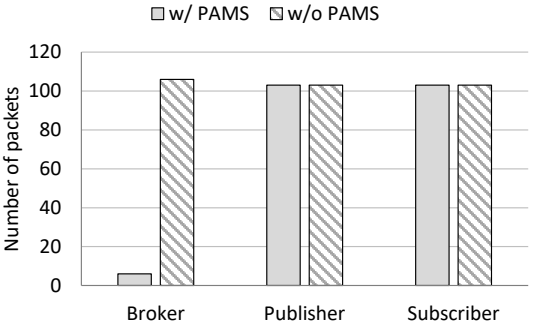

Fig. 9. The number of packets with and without PAMS

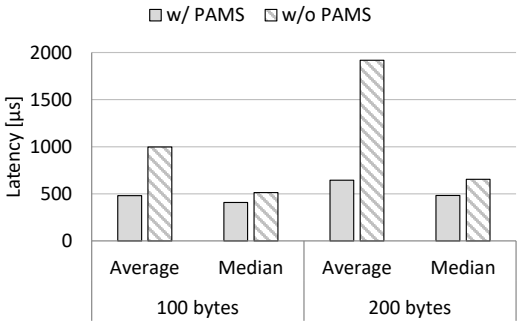

Fig. 10. Latency from publisher to subscriber

makes a network switch perform a part of broker functionality, i.e., forwarding Publish messages based on topic matching. From analytical and experimental evaluation, it is clarified that PAMS can lower the broker load by suppressing the number of Publish messages processed in the broker and reducing the latency.

Future work includes experimental evaluation with a hardware switch instead of BMv2. BMv2 is a software switch mainly for verifying purposes, so it is not easy to clarify the practical efficiency of PAMS. We plan to experiment with a hardware switch supporting P4 and clarify the actual performance for IoT data.

ACKNOWLEDGMENT

We are grateful to Tomoya Hibi for helpful discussions.

REFERENCES

[1] F. Shrouf, J. Ordieres, and G. Miragliotta, "Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm," in *Proc. IEEE International Conference on Industrial Engineering and Engineering Management*, 2014, pp. 697–701.

[2] M. van Geest, B. Tekinerdogan, and C. Catal, "Design of a reference architecture for developing smart warehouses in industry 4.0," *Computers in Industry*, vol. 124, p. 103343, 2021.

[3] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s — a publish/subscribe protocol for wireless sensor networks," in *Proc. International Conference on Communication Systems Software and Middleware and Workshops*, 2008, pp. 791–798.

[4] A. Stanford-Clark and H. L. Truong, *MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2*, IBM Corporation, 2013.

[5] MQTT, https://mqtt.org/ (accessed Aug. 24, 2022).

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

[7] RSMB: Really Small Message Broker, https://github.com/eclipse/mosquitto.rsmb (accessed Aug. 24, 2022).

[8] E. Longo, A. E. Redondi, M. Cesana, A. Arcia-Moret, and P. Manzoni, "Mqtt-st: a spanning tree protocol for distributed mqtt brokers," in *Proc. IEEE International Conference on Communications*, 2020, pp. 1–6.

[9] A. Detti, L. Funari, and N. Blefari-Melazzi, "Sub-linear scalability of mqtt clusters in topic-based publish-subscribe applications," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1954–1968, 2020.

[10] R. Banno, J. Sun, M. Fujita, S. Takeuchi, and K. Shudo, "Dissemination of edge-heavy data on heterogeneous mqtt brokers," in *Proc. IEEE International Conference on Cloud Networking*, 2017, pp. 1–7.

[11] R. Banno, J. Sun, S. Takeuchi, and K. Shudo, "Interworking layer of distributed mqtt brokers," *IEICE Transactions on Information and Systems*, vol. E102.D, no. 12, pp. 2281–2294, 2019.

[12] W. Pipatsakulroj, V. Visoottiviseth, and R. Takano, "mumq: A lightweight and scalable mqtt broker," in *Proc. IEEE International Symposium on Local and Metropolitan Area Networks*, 2017, pp. 1–6.

[13] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, p. 87–95, 2014.

[14] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, and D. Timmermann, "Realizing content-based publish/subscribe with p4," in *Proc. IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2018, pp. 1–7.

[15] R. Kundel, C. Gärtner, M. Luthra, S. Bhowmik, and B. Koldehofe, "Flexible content-based publish/subscribe over programmable data planes," in *Proc. IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–5.

[16] J. Vestin, A. Kassler, S. Laki, and G. Pongrácz, "Toward in-network event detection and filtering for publish/subscribe communication using programmable data planes," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 415–428, 2021.

[17] A. Alshantout and L. Al-Awami, "Enhancing mqtt-sn performance via fountain codes in extreme conditions," in *Proc. International Wireless Communications Mobile Computing Conference*, 2019, pp. 1184–1189.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69–74, 2008.

[19] Mininet, http://mininet.org/ (accessed Aug. 24, 2022).

[20] J. R. Jackson, "Jobshop-like queueing systems," *Management science*, vol. 10, no. 1, pp. 131–142, 1963.

[21] BEHAVIORAL MODEL (bmv2), https://github.com/p4lang/behavioral-model (accessed Aug. 24, 2022).

[22] MQTT-SN Tools, https://github.com/njh/mqtt-sn-tools (accessed Aug. 24, 2022).