Sensors and Materials, Vol. 30, No. 8 (2018) 1715–1721
MYU Tokyo

S & M 1628

Message Queue Telemetry Transport Broker with Priority Support for Emergency Events in Internet of Things

Yong-Seong Kim,[1] Hwi-Ho Lee,[2] Jung-Hyok Kwon,[2]
Yong Sin Kim,[1*] and Eui-Jik Kim[2**]

[1]School of Electrical Engineering, Korea University,
145 Anam-ro, Seongbuk-gu, Seoul 02841, South Korea

[2]Department of Convergence Software, Hallym University,
1 Hallymdaehak-gil, Chuncheon, Gangwon 24252, South Korea

(Received March 30, 2017: accepted January 22, 2018)

Keywords: emergency event, Internet of Things, MQTT, priority control, virtual queue

This paper presents a message queue telemetry transport (MQTT) broker with priority support for emergency events in the Internet of Things (IoT), which is abbreviated p-MQTT. To support the timely and reliable message delivery of emergency events, the p-MQTT classifies the published messages coming into the broker server and controls their priority according to the classification results. To this end, the p-MQTT consists of three components: virtual queue, classification, and priority control. The virtual queue stores the published messages separately according to their type, for which the p-MQTT broker server maintains three virtual queues: *Urgent*, *Critical*, and *Normal*. The classification component classifies the published messages into the three types mentioned above by checking the message type field in the published message header and stores the messages in the appropriate virtual queue. Finally, the priority control assigns a forwarding priority to each virtual queue and adjusts the quality-of-service (QoS) level of the messages within each virtual queue accordingly. To verify its effectiveness, we conduct an experimental implementation of the p-MQTT. The results show that the p-MQTT achieves better performance in emergency events than the existing MQTT.

1. Introduction

Recently, the demand for various Internet of Things (IoT) services, such as smart homes, smart grids, health care, and smart factories, has significantly increased. Most IoT services employ a number of resource-constrained devices with limited computing capabilities, limited storage capacity, and limited power, and these devices communicate with each other through low-power lossy networks (LLNs). This constrained environment can lead to high packet loss and unpredictable long delays in IoT services.

To support reliable IoT services, the organization for the advancement of structured information standards (OASIS) specifies the message queue telemetry transport (MQTT)

*Corresponding author: e-mail: shonkim@korea.ac.kr
**Corresponding author: e-mail: ejkim32@hallym.ac.kr
http://dx.doi.org/10.18494/SAM.2018.1864

protocol as the international organization for standardization (ISO) standard (i.e., ISO/IEC PRF 20922).(1) The MQTT is a lightweight publish/subscribe-based messaging protocol. In the MQTT, devices exchange messages via a broker server that distributes the published messages to interested devices based on the topic of the message. The MQTT uses very small message header sizes (i.e., 2 B) to keep message overhead small in the constrained environment of IoT services. Therefore, the MQTT is widely considered one of the essential technologies for making IoT services reliable.

Monitoring applications such as vehicle tracking, accident detection, health condition recognition, and industrial machine monitoring are considered the most general and representative IoT applications.(2) In these applications, timely and reliable message delivery is crucial, particularly for emergency events. The MQTT standard manages the reliability of message delivery by defining three quality-of-service (QoS) levels.(1) The QoS levels determine the use of acknowledgement and retransmission in message delivery; thus, the message is delivered at most once (i.e., QoS level 0), at least once (i.e., QoS level 1), or exactly once (i.e., QoS level 2), depending on the QoS level. However, the MQTT standard does not define any mechanism to support the timeliness of message delivery; thus, there could be a long delay even for the delivery of emergency events. Therefore, priority support for specific emergency-event messages and a mechanism for providing reliability in combination with priority support are needed for seamless IoT services.

In recent years, a number of studies related to timely and reliable message delivery have been conducted for IoT services. Tachibana *et al.* proposed a priority control mechanism to specify the transmission time for IoT devices taking into consideration the data type (e.g., image, text, and video) and transmission interval for the data.(3) For this, a broker server is used, and the server is responsible for managing the transmission priority of all IoT devices. This mechanism can guarantee the timeliness of message delivery by controlling the transmission priority of IoT devices. However, it has low compatibility with the existing IoT system, since it was designed without consideration for application protocol specifications such as the MQTT. Jo and Jin proposed an adaptation framework for periodic N-to-1 communication over the MQTT, which adjusts the publication period to ensure the timeliness of periodic messaging.(4) However, this framework focuses on only the requirements of periodic communication where all messages have the same priority, so it is not suitable for emergency events. This is because an emergency-event message needs to have a higher transmission priority than other messages. Al-Fuqaha *et al.* proposed an enhanced MQTT designed to allow the MQTT broker to reprioritize messages.(5) Since the MQTT broker controls the forwarding priority of published messages, the enhanced MQTT can guarantee the timeliness of message delivery in an emergency event. However, the authors do not provide details of the priority control mechanism; thus, it is very unlikely to be applied in a real environment.

In this paper, we propose an MQTT broker with priority support for emergency events for IoT monitoring applications (p-MQTT), which aims to support timely and reliable message delivery for emergency events. The p-MQTT classifies the published messages coming into the broker server and controls their priority according to the classification results. To this end, the p-MQTT consists of three components: virtual queue, classification, and priority control. The virtual queue stores the published messages separately according to the message type, for which the p-MQTT broker server maintains three virtual queues: *Urgent*, *Critical*, and *Normal*. The classification component classifies the published messages into the three types mentioned above by checking the message type field in the published message header, and stores the messages in the appropriate virtual queue. Finally, the priority control assigns a forwarding priority to each virtual queue, and adjusts the QoS level of the messages within each virtual queue accordingly. To evaluate the performance of the p-MQTT, an experimental implementation is conducted by using the open-source MQTT software Mosquitto and Paho. The results show that the p-MQTT achieves better performance compared to the existing MQTT, in terms of the latency and message loss rate.

The rest of this paper is organized as follows. In Sect. 2, the design of the p-MQTT is described in detail. Section 3 presents the results of implementation and experiment. Finally, the paper is concluded in Sect. 4.

2. Design of p-MQTT

The p-MQTT supports the timely and reliable message delivery of emergency events by assigning a forwarding priority to each virtual queue and adjusting the QoS level of the messages within each virtual queue accordingly. To this end, it consists of three components: virtual queue, classification, and priority control. In the following subsections, the architecture and operation of the p-MQTT are described in detail.

2.1 System architecture

Figure 1 shows the system architecture for the p-MQTT, where the p-MQTT broker server includes virtual queue, classification, and priority control components. The virtual queue stores the messages transmitted from the publishers separately according to the messages type. The p-MQTT broker server maintains three virtual queues: *Urgent*, *Critical*, and *Normal*. The urgent queue stores messages that should be transmitted more urgently than any other message. The critical queue stores messages that are less urgent than the urgent queue, but still require high reliability. The messages stored in the normal queue are those used in the existing MQTT standard.

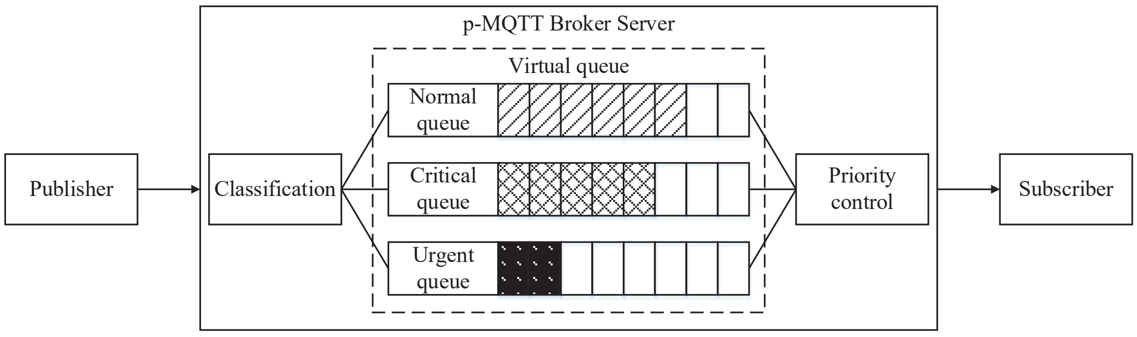


Fig. 1. System architecture for p-MQTT.

The classification component checks the message type field in the published message header, by which it classifies the messages as urgent, critical, or normal according to the value. Then, it stores the messages in the appropriate virtual queue. If the message type is 0, it classifies the message as an urgent message, and stores it in the urgent queue within the p-MQTT broker server. If the message type is 1–14, it classifies the message as a normal message and stores it in the normal queue. If the message type is 15, it classifies the message as a critical message, and stores it in the critical queue.

Finally, the priority-control component assigns a forwarding priority to each virtual queue, and adjusts the QoS level of the messages within each virtual queue accordingly. The urgent queue is assigned the highest forwarding priority; thus, the messages stored in the urgent queue are forwarded first, regardless of the presence of messages stored in other queues. In this case, the priority control component adjusts the QoS level of the message header to 0 for timely message delivery, and the message is delivered at most once without acknowledgement and retransmission.

The critical queue has a medium forwarding priority, and thus the messages in the critical queue can be forwarded only when the urgent queue is empty. In this case, the QoS level is adjusted to 2 to deliver the message exactly once, thereby ensuring a higher reliability for message delivery. Finally, the normal queue is assigned the lowest priority, and the messages in the normal queue are forwarded only when both the urgent and critical queues are empty. The QoS level of the messages remains unchanged, and the value set by the publisher also remains unchanged.

2.2 Message format

Figure 2 shows the message format for the MQTT protocol.[1] The p-MQTT uses the same message format as the existing MQTT for compatibility. In the figure, the message types are defined in 4 bits, and thus 16 message types can be defined. However, only 14 message types (i.e., 1–14) are defined in the MQTT specification, and the values 0 and 15 are "reserved". To add a message type for urgent and critical messages, we use those "reserved" values (i.e., 0 and 15). Figure 3 shows the message types for the p-MQTT in detail. In the p-MQTT, the values 0 and 15 are used for urgent and critical messages, respectively. If the value of the message type is between 1 and 14, the p-MQTT broker server identifies the message as a normal message. The value of the message type is assigned by the publisher, and the message type is identified by the p-MQTT broker server based on the value.

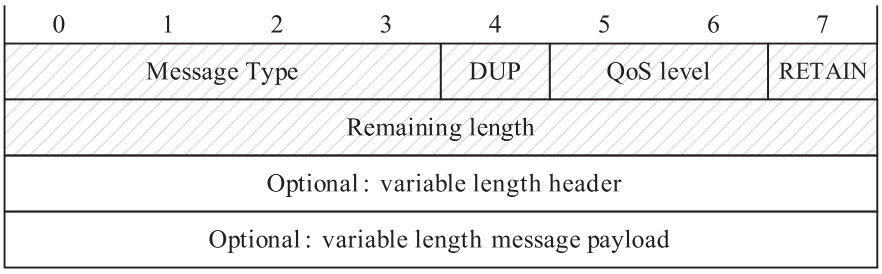
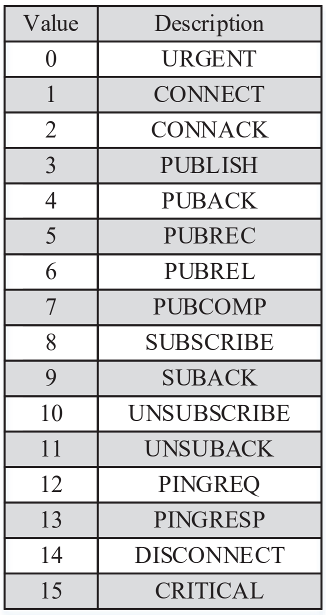

0
1
2
3
4
5
6
7
Message Type
DUP
QoS level
RETAIN
Remaining length
Optional : variable length header
Optional : variable length message payload

Fig. 2. MQTT message format.



Value	Description
0	URGENT
1	CONNECT
2	CONNACK
3	PUBLISH
4	PUBACK
5	PUBREC
6	PUBREL
7	PUBCOMP
8	SUBSCRIBE
9	SUBACK
10	UNSUBSCRIBE
11	UNSUBACK
12	PINGREQ
13	PINGRESP
14	DISCONNECT
15	CRITICAL

Fig. 3. p-MQTT message types.

3. Performance Evaluation

An experimental implementation is conducted to evaluate the performance of the p-MQTT. The p-MQTT broker server is implemented using the open-source Mosquitto software version 1.4.13 on Ubuntu version 16.04.2.(6,7) Moreover, the publisher and subscriber are implemented using open-source libraries provided by the Eclipse Paho project.(8)

In the experiment, multiple devices (i.e., publishers) are connected to the p-MQTT broker server and periodically transmit messages to it. Among these devices, only a single publisher generates urgent or critical messages that include an emergency event while the other publishers generate normal messages. In order to check the variations in latency and message loss rate when the number of devices changes, we vary the number of publishers from 100 to 1000 in the experiment. We set the total number of messages per publisher to 4; thus, each publisher transmits four messages during the experiment. In addition, each publisher generates a new message after a successful transmission until the number of generated messages reaches 4. The message size is set to 4 B. The QoS level of the published message is set to 1; thus, all messages are initially published with a QoS level of 1. However, in the case of urgent and critical messages, the QoS level is changed to 0 and 2 by the MQTT broker server. The experiment is repeated 10 times. The detailed experiment parameters are listed in Table 1.

Figure 4 shows the variations in latency for urgent messages as the number of devices increases. Overall, the latency for urgent message with the p-MQTT is on average 35.3% lower than that with the existing MQTT. This is because the urgent queue of the p-MQTT broker server has the highest forwarding priority; thus, an urgent message is delivered earlier than other messages. In the figure, the difference in latency for urgent messages between the p-MQTT and the MQTT increases as the number of devices increases. The reason for this is

Table 1
Experiment parameters.

Parameter	Value
Number of publishers	100–1000
Data rate	100 Mbps
Size of message	4 B
Total number of messages per publisher	4
QoS level of published message	1

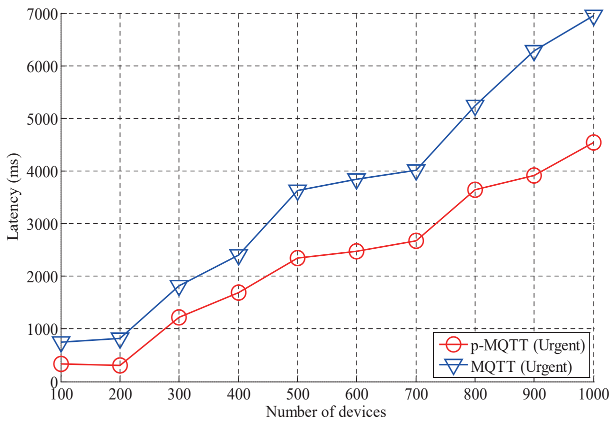


Fig. 4. (Color online) Latency for urgent messages.

that the queuing delay for urgent messages in the p-MQTT broker server is kept short because of the priority control. On the other hand, the queuing delay for urgent messages in the MQTT broker server increases exponentially as the number of transmitted messages increases.

Figure 5 shows the latency variations for critical messages as the number of devices increases. Even in the case of critical messages, the p-MQTT exhibits better latency performance than the existing MQTT owing to the priority control. However, the latency difference is less than the results in Fig. 4. The reason for this is that the critical queue has a lower forwarding priority than the urgent queue; thus, the QoS levels of the messages stored in each virtual queue are adjusted to different values. More specifically, the QoS level for an urgent message is adjusted to 0 since it only requires timely message delivery. The QoS level of a critical message is adjusted to 2 for reliable message delivery, which triggers a number of control message exchanges and retransmissions if necessary.

Figure 6 shows the message loss rate according to the message type. In the case of urgent messages, the p-MQTT shows a slightly higher message loss rate than the existing MQTT since an urgent message in the p-MQTT does not require acknowledgement owing to its QoS level. Moreover, the difference in message loss rate is not large. This is because the p-MQTT maintains a virtual queue dedicated to urgent messages, resulting in message loss reduction. With critical messages, the message loss rate of the p-MQTT is 51.8% lower than that of the existing MQTT. The reason for this is that the QoS level of a critical message in the p-MQTT is adjusted to 2 for reliable message delivery, while the existing MQTT sets the QoS level to 1 regardless of the message type.

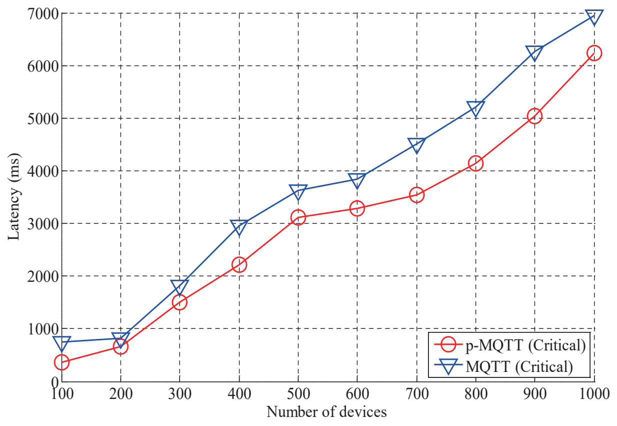


Fig. 5. (Color online) Latency for critical messages.

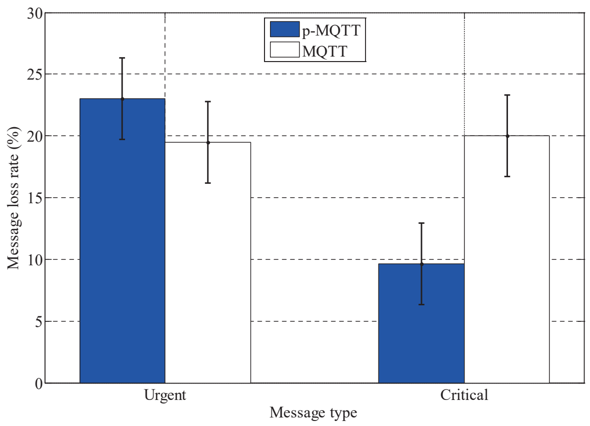


Fig. 6. (Color online) Message loss rate.

4. Conclusions

In this paper, we presented the p-MQTT, which is an MQTT broker server with priority support for emergency events in IoT services. To support the timely and reliable delivery of emergency events, the p-MQTT classifies published messages coming into the broker server and controls their forwarding priority and QoS level according to the classification results. To evaluate the performance of the p-MQTT, we conducted an experimental implementation using the open source Mosquitto and Paho, and compared the performance of the p-MQTT with that of the existing MQTT. The results show that the p-MQTT compared with the existing MQTT achieves 35.3 and 18.1% lower latencies for urgent and critical messages, respectively. Moreover, the p-MQTT achieves 51.8% lower message loss for critical messages on average.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A1B03031055). This research was also supported by the Leading Human Resource Training Program of Regional Neo Industry through the NRF funded by the Ministry of Science and ICT (NRF-2016H1D5A1910427).

References

1 ISO/IEC 20922:2016 Information Technology-Message Queuing Telemetry Transport (MQTT) v3.1.1. iso.org. International Organization for Standardization (accessed June 15, 2016).
2 P. Sethi and S. R. Sarangi: Can. J. Electr. Comput. Eng. Can. **2017** (2017) 1.
3 T. Tachibana, T. Furuichi, and H. Mineno: Proc. 13th Int. Conf. Mobile and Ubiquitous Systems: Computing Networking and Services (ACM, 2016) 239.
4 H. C. Jo and H. W. Jin: Proc. 2015 IEEE 3rd Int. Conf. Cyber-Physical Systems, Networks, and Applications (CPSNA) (IEEE, 2015) 66.
5 A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi: IEEE Commun. Mag. **53** (2015) 72.
6 Eclipse Foundation: https://mosquitto.org/ (accessed July 2017).
7 Canonical Ltd.: https://www.ubuntu.com/ (accessed July 2017).
8 Eclipse Foundation: http://www.eclipse.org/paho/ (accessed July 2017).