Akshatha *et al.*
Journal of Engineering and Applied Science (2024) 71:67
https://doi.org/10.1186/s44147-024-00400-2

Journal of Engineering and Applied Science

RESEARCH Open Access

Priority-enabled MQTT: a robust approach to emergency event messaging

P S Akshatha[1,2*], S Divyashree[3] and S M Dilip Kumar[1]

*Correspondence:
akshatha.ps@gmail.com

[1] University Visvesvaraya College of Engineering, Bengaluru, Karnataka 560 001, India
[2] Dept of AIML, New Horizon College of Engineering, Bengaluru, Karnataka 560 103, India
[3] East West Institute of Technology, Bengaluru, Karnataka 560 091, India

Abstract

This paper presents priority support in the Internet of Things to support the reliable and timely transmission of messages during emergencies. The Message Queuing Telemetry Transport protocol is a widely used IoT messaging protocol. However, it does not support the timely and fast delivery of emergency messages. In this regard, this paper proposes to classify the messages into three different queues. The RabbitMQ broker manages virtual queues based on the message type, such as First Come First Served, Critical, and Urgent. In addition, the proposed approach stores the messages in the MySQL database for further analysis. To confirm its efficacy, we compare the Urgent and Critical queues with the current First Come First Served technique in an experimental implementation. Wireshark packet analyzer is used to record packets while messages are being transmitted between clients and the broker to examine end-to-end latency, jitter, response time, and total time. The results show that the proposed approach performs better for high-priority emergency messages.

Keywords: AMQP, Internet of Things, Message Queuing Telemetry Transport, Brokers, RabbitMQ, Delay, Jitter, Response time

Introduction

IoT is one of the most essential and current technologies that has conquered all aspects of human existence. It enables the connection and communication of devices without human interaction. IoT has facilitated faster and easier access to remote applications and information [1–4]. IoT expanded the Internet's capabilities beyond the limitations of standard computers by allowing smart devices to transmit and receive data remotely from diverse settings and communication networks. Due to the heterogeneity of IoT applications in various aspects of life, many IoT protocols are chosen based on the protocol's essential function [5]. These protocols can handle message transmission between sensing and processing nodes, transmit data, collect data from sensing nodes, and gather data from sensing nodes. The MQTT protocol is most widely used in the IoT application layer [6]. It is a networking protocol for M2M and IoT. It transmits messages between devices via a publish/subscribe communication mechanism. It makes it easier to communicate with remote areas when network bandwidth is constrained.

Vehicle tracking, health status assessment, accident detection, and industrial machine monitoring are among the most generic and representative IoT applications. Rapid and

efficient message delivery is essential in many applications, especially in emergencies. The MQTT standard maintains message transmission dependability by defining three quality-of-service (QoS) levels, namely,

1. *QoS-0*: At most once,
2. *QoS-1*: At least once, and
3. *QoS-2*: Exactly once.

However, the current MQTT for IoT does not prioritize processing emergency messages. The transmission of information regarding a patient's abFCFS state for medical purposes and an emergency requiring an emergency stop of related field facilities when an abFCFSity occurs in a specific facility at a manufacturing site are representative examples that should be handled first and foremost [7–10]. We also require historical data for analytics and reporting. Most MQTT brokers do not include a built-in mechanism for saving MQTT data to a database.

This paper provides a way to overcome the presented IoT protocols' difficulties and limitations. The MQTT protocol in the proposed approach coordinates the publishing of messages between publishers based on specified priority queues, allowing emergency messages to be sent on time and reducing message latency. The proposed method can also save MQTT data from sensors to a MySQL database for later analysis and reporting.

RabbitMQ

RabbitMQ is open-source message broker software. The RabbitMQ MQTT plugin supports many MQTT clients and supports version MQTT 3.1.1. It also enables interoperability between (Advanced Message Queuing Protocol) AMQP 0-9-1, AMQP 1.0, STOMP clients, and MQTT clients. The concept of multi-tenancy is supported (https://www.rabbitmq.com/mqtt.html). It also supports various plugins and utilities, providing operational metrics, continuous integration, and integration with other enterprise systems.

Contributions

The contributions of this paper are listed below:

1. Introducing a prioritized MQTT emergency message delivery system to ensure high-priority transmission.
2. Implementing a message storage mechanism in the MySQL database, enabling comprehensive analysis of stored data.
3. Estimating and comparing the end-to-end delay between the proposed prioritized approach and the conventional FCFS approach, providing insights into latency improvements.
4. Assessing and comparing the total time required to publish messages via Urgent, Critical, and FCFS queues, highlighting the efficiency of the proposed prioritization.
5. Evaluating and comparing the mean response time of messages, offering a comprehensive understanding of the performance differences between the proposed and FCFS approaches.

6. Analyzing and comparing mean jitter at the subscriber end in both the proposed prioritized approach and the FCFS approach, contributing to a thorough examination of message delivery stability.

Paper organization

The rest of the paper is formulated as follows: The related works to improve the priority mechanism for the MQTT protocol are discussed in the "Related work" section. The "Problem statement" section discusses the problem statement and objectives. The "Proposed system" section discusses the proposed system with architecture and analytical analysis of queues. In the "Results and discussion" section, Results and Discussions for existing and proposed approaches are compared and analyzed. Finally, in the "Conclusions" section, the conclusion of the paper and future work is presented.

Related work

This section discusses related works that use MQTT communication to improve the priority feature.

The authors of [11] proposed an algorithm to prioritize messages using a multi-scanned priority message sorting algorithm, with Mosquitto broker and smart factory applications in IoT as examples. To reduce the latency of the data packets needed for time-sensitive applications, the authors of [12] described the prioritization of the traffic information gathered using sensors in a gateway for the MQTT-SN. To reduce uplink traffic load, the developers of [13] suggested the Backoff method, which applies an exponential delay factor to suspect publishers. Additionally, depending on the recently determined frequency rate, a priority scheduling algorithm is offered to classify publishers as high or low priority.

In [14], a technique for detecting out-of-order notifications on top of an existing topic-based Event Notification Service is presented. The results demonstrate that events published on various subjects are either sent in the same sequence to all subscriber clients of those topics or labeled as out-of-order. The authors of [15] suggested a novel way to enhance the capabilities of MQTT by transmitting Urgent messages first. The Mosquitto broker is modified to build a "U-Mosquitto" broker capable of processing emergency messages.

The paper [16] explores existing literature on methods for assigning priority to MQTT messages and introduces an approach for identifying and processing Urgent messages in MQTT applications. The focus is prioritizing specific messages from Critical sources in IoT environments. The work in [17] utilizes Software Defined Networking, particularly the OpenFlow protocol, to enforce temporal behavior through network reservations. Extensive emulation and implementation results demonstrate the approach's feasibility, showcasing adequate segregation and prioritization of time-sensitive MQTT traffic.

Uchida et al. [18] introduced assigning priority values to IoT data based on abFCFSity, storing them in broker node priority queues. The experiments with a prototype system demonstrate the effectiveness of the Enhanced MQTT method.

The conventional methods to enhance MQTT communication priority involve modifications to the MQTT protocol packet structure or broker configuration. Some papers

propose approaches without experimental validation. In contrast, the proposed method introduces a priority enhancement mechanism in MQTT communication without altering the standard structure. The practical approach validates the proposed method across parameters such as delay, jitter, response time, and total time taken. Results demonstrate that the proposed method responds rapidly to emergencies, showcasing its effectiveness in real-world scenarios. The approach not only preserves the integrity of the MQTT protocol but also ensures a reliable and swift response in Critical situations, addressing the limitations of existing methods.

Problem statement

The problem statement of this work is to enhance the MQTT message delivery between publisher and subscriber in emergency events by supporting priority support. The objectives of this work include:

1. To provide priority support by creating virtual queues to enhance message delivery.
2. To compare average response time and total time taken by `Critical`, `Urgent`, and `FCFS` queues.
3. To compare end-to-end delay and mean jitter of `Critical` and `FCFS` queues.

Proposed system

This section presents the proposed system to enhance MQTT message delivery in emergency events using priority support. Figure 1 shows the proposed architecture for MQTT communication. This work is implemented using an application where users can publish the messages in the three provided topics: Urgent, Critical, and FCFS. The highest priority is the Urgent queue to transmit the messages immediately. Messages in the Critical queue are less Urgent than those in the Urgent queue but still demand high reliability. Messages saved in the FCFS queue are the same as in the current MQTT standard. Figure 2 depicts the flow of the proposed architecture. The MQTT publisher client can publish messages on `Urgent`, `Critical`, and `FCFS` topics. Upon receiving messages, the RabbitMQ broker classifies the messages based on topics and assigns respective predefined priority queues. The published messages are stored in a database for further analysis. Due to the highest priority, the Urgent queue messages are published to the consumers first, followed by the Critical and then the FCFS queue.

The proposed approach has the following benefits:

1. *Delivery of MQTT messages in emergency events:* When managing emergencies in IoT and messaging apps, it is crucial to prioritize and consider some vital messages. This is fixed by establishing distinct virtual queues for various message types dependent on the importance of the message.
2. *Resolving Bottleneck problems:* Some applications may use an exceptionally high volume of messages; under these circumstances, the brokers may experience bottleneck issues. This circumstance typically occurs in publisher-subscriber-based applications when a single client subscribes to several topics (using wildcard topics). In these situations, we might observe a decline in the network's processing power and bandwidth

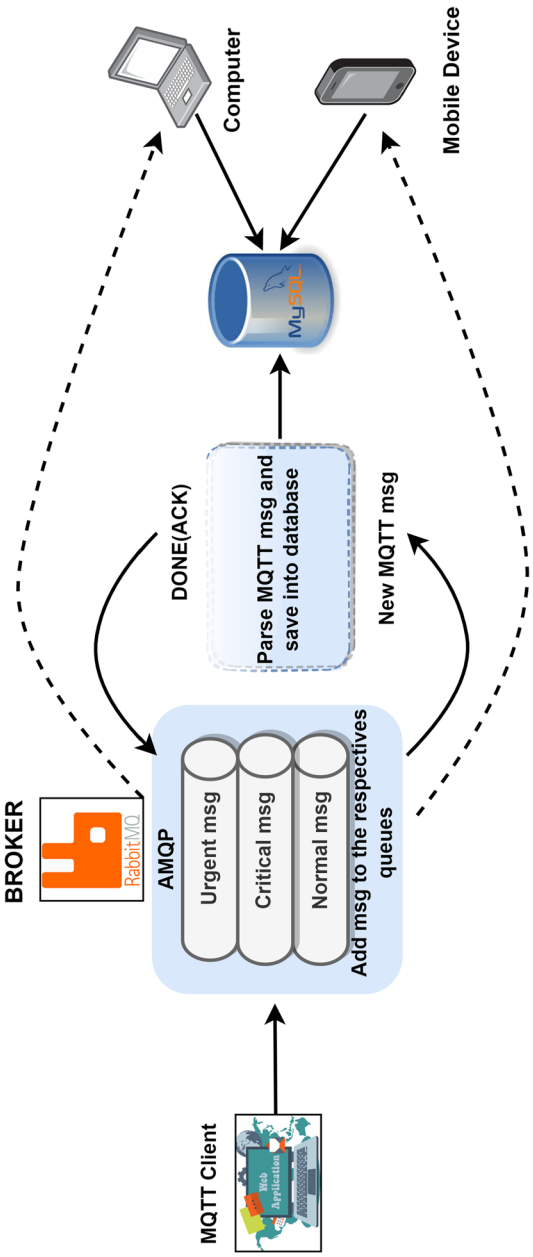

Fig. 1 Proposed architecture of MQTT communication

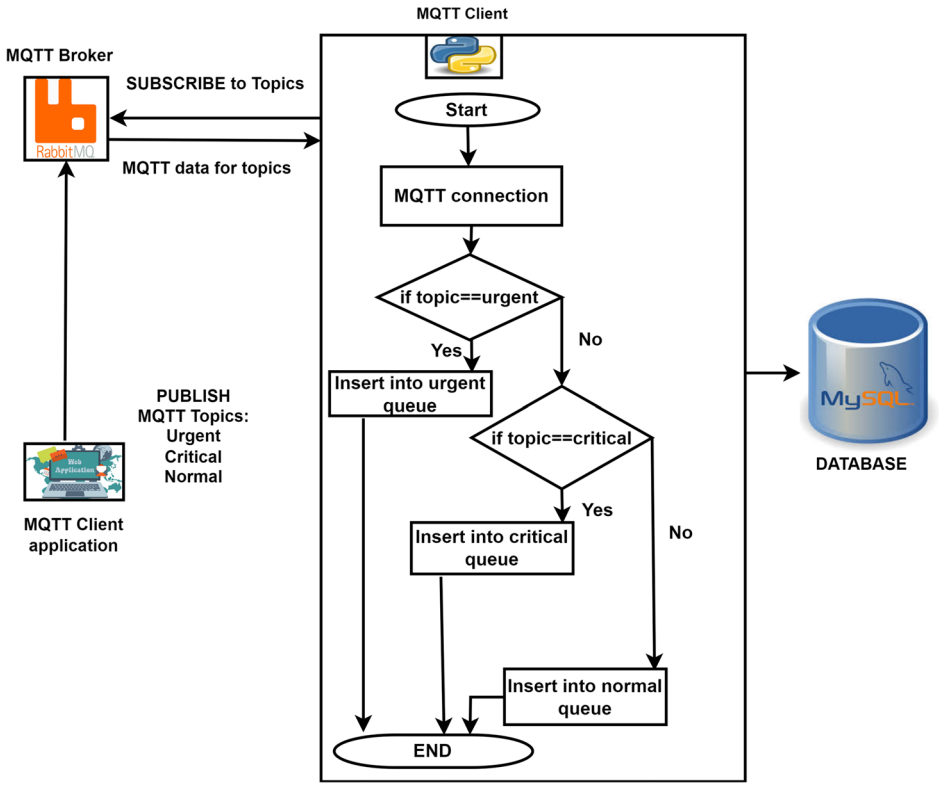

Fig. 2 Flowchart of MQTT message delivery with priority support

in the computers that host these MQTT clients. The MQTT data to the RabbitMQ broker's AMQP queue is temporarily transferred to fix this.

3. *Storing MQTT data into database:* For Sensors to publish data to their Subscribers, MQTT is an excellent protocol. However, historical data is necessary for analytics and reporting. Most MQTT brokers do not have any built-in functionality for saving MQTT data to databases. The suggested method enables the database storage of MQTT data. Figures 3 and 4 depict the messages stored in the Urgent and Critical queues for further analysis.

Analysis of queues

Consider an `M/G/1` queue where the messages are divided into three priority classes:

- Class 1 has the highest priority, and class i has the lowest priority.
- The arrival rates of different classes are $\lambda_1, ..., \lambda_i$ (Poissonian)
- The service time of different classes are $\mu_1, ..., \mu_i$.

The notations used in the analysis are listed below in Table 1. In the same method, we deduce the Pollaczek-Khinchinin mean findings for the Urgent (highest priority) queue as expressed in Eq. (1).

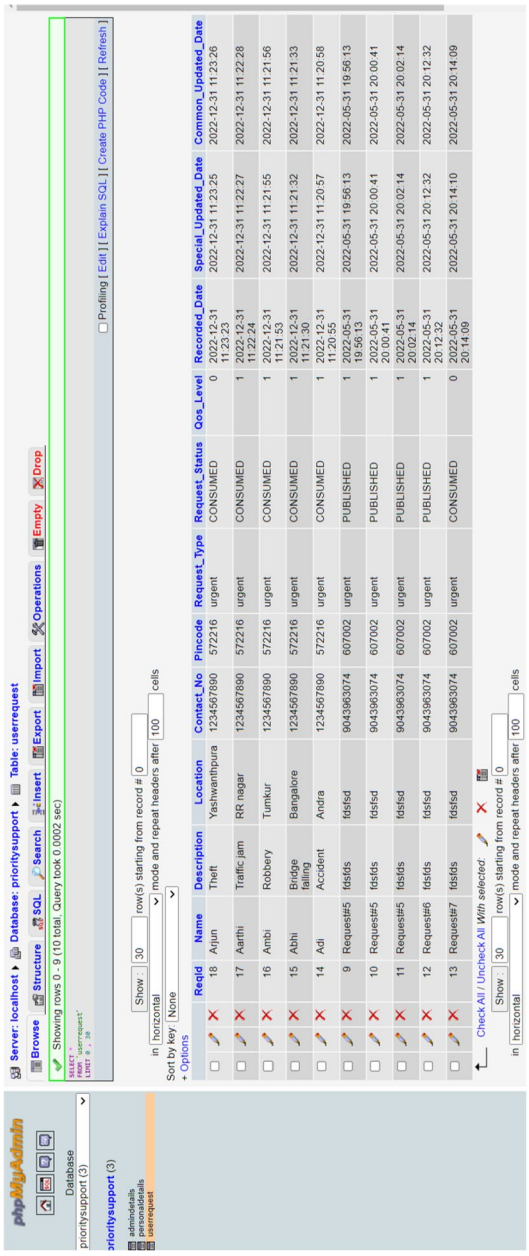

Fig. 3 Urgent queue messages stored in database

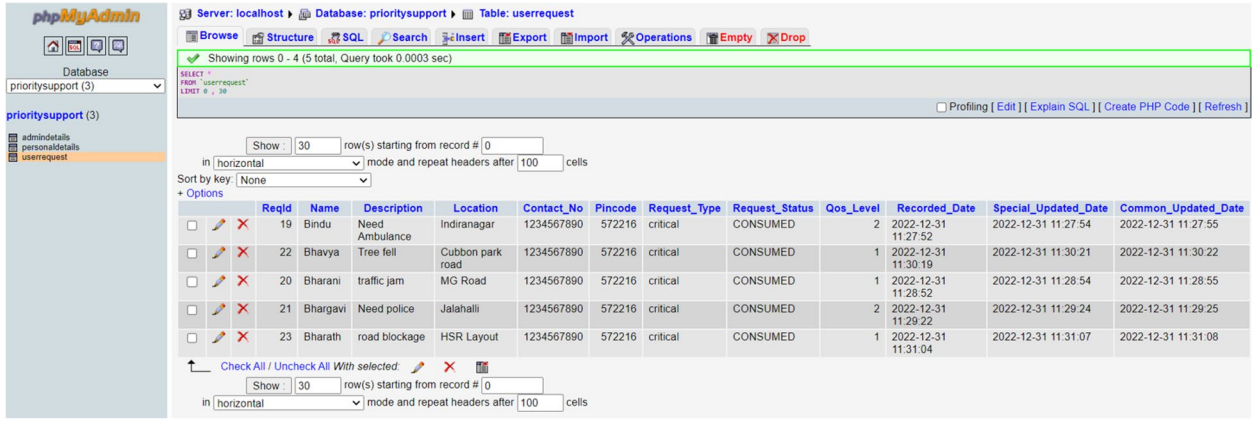

Fig. 4 Critical queue messages stored in database

Table 1 Notations

Symbol	Description
$\bar{C}_{(i)}$	Mean number of waiting queue-*i* messages in the queue.
$\bar{W}_{(i)}$	Mean waiting time of queue-*i* messages.
ρ_i	The load of queue *i*, $\rho_i = \lambda_i \bar{\mu}_i$.
$\bar{R}_i$	The mean residual service time in the broker upon arrival of messages.
u	Urgent queue
c	Critical queue

$$\bar{W}_u = \bar{R} + \bar{\mu}_u \bar{C}_u \tag{1}$$

By Little's law, we have,

$$\bar{C}_u = \lambda_u \bar{W}_u \implies \bar{W}_u = \bar{R} + \rho_u \bar{W}_u \implies \bar{W}_u = \frac{\bar{R}}{1-\rho_u} \tag{2}$$

For Critical priority class, we get

$$\bar{W}_c = \bar{R} + \bar{\mu}_u \bar{C}_u + \bar{\mu}_c \bar{C}_c + \bar{\mu}_u \lambda_u \bar{W}_c \tag{3}$$

where $\bar{\mu}_u \bar{C}_u + \bar{\mu}_c \bar{C}_c$ indicates the time needed to serve the Urgent and priority messages ahead in the queue and $\bar{\mu}_u \lambda_u \bar{W}_c$ indicates the time needed to serve those messages in higher queues that arrive during the waiting time for Critical messages. By Little's law again, we get,

$$\bar{C}_c = \lambda_c \bar{W}_c \implies \bar{W}_c = \bar{R} + \rho_u \bar{W}_u + \rho_c \bar{W}_c + \rho_u \bar{W}_c \implies \bar{W}_c = \frac{\bar{R} + \rho_u \bar{W}_u}{1-\rho_u-\rho_c} \tag{4}$$

By substituting the expression $\bar{W}_u$ in $\bar{W}_c$ formula, we get

$$\bar{W}_c = \frac{\bar{R}}{(1-\rho_u)(1-\rho_u-\rho_c)} \tag{5}$$

We obtain the overall result by applying the same strategy to lower priority classes (higher values of *i*) as in Eq. (6).

$$\bar{W}_i = \frac{\bar{R}}{(1-\rho_1-...-\rho_{(i-1)})(1-\rho_1-...-\rho_i))} \tag{6}$$

The total time in the system of queue-*i* messages is, on average, expressed in Eq. (7).

$$\bar{T}_i = \bar{W}_i + \bar{\mu}_i \tag{7}$$

The mean residual service time $\bar{R}$ in $\bar{W}_i$ can be calculated as in the Pollaczek-Khinchinin mean value formula expressed in Eq. (8):

$$\bar{R} = \frac{1}{2}\sum_{i=1}^{i} \lambda_i \bar{\mu_i} \tag{8}$$

Let us compute the average stay time T_i of $class - i$ messages. It is split into three parts.

1. The customer's own mean service time $\bar{\mu_i}$.
2. The average time it takes to serve messages in classes 1, ..., i ahead in the queue: $\frac{\bar{R_i}}{1-\rho_1-...-\rho_i}$.
3. The mean-time it takes to serve messages in higher classes $1, ..., (i-1)$ that comes while the class-i message is still in the system.

$$\sum_{n=1}^{i-1} \bar{\mu}_n \lambda_n \bar{T}_i = \sum_{n=1}^{i-1} \rho_n \bar{T}_i, k > 1 \tag{9}$$

We obtain the equations below using the Pollaczek-Khinchinin mean value formula and Kleinrock's conservation theorem.

$$\bar{T}_i = \bar{\mu}_i + \frac{\bar{R}_i}{1-\rho_1-...-\rho_i} + (\sum_{n=1}^{i-1} \rho_n)\bar{T}_i \tag{10}$$

Therefore, $\bar{T}_u = \frac{(1-\rho_u)\mu_u + \bar{R}_u}{1-\rho_u}$ and $\bar{T}_i = \frac{(1-\rho_1-...-\rho_i)\bar{\mu}_i + \bar{R}_i}{(1-\rho_1-...-\rho_{(i-1)})(1-\rho_1-...-\rho_i)}$

Experimental

Wireshark [19] is crucial in measuring the end-to-end delay between MQTT publish and AMQP received timing. The study involves the transmission of messages through three different queues with priority support and a single queue using the FCFS approach. Wireshark, a packet analyzer, captures and analyzes the packets exchanged between clients and the broker during message transmission. By inspecting the packet data, Wireshark enables the calculation of end-to-end delay for each scenario. Specifically, ten messages are published to each topic under both prioritized and FCFS conditions, allowing for a detailed examination of the impact of the proposed prioritization approach on the temporal aspects of message delivery. This explicit use of Wireshark ensures accurate and comprehensive data collection, contributing to a more detailed analysis of the experimental results.

1. Time T_1 is the mqtt publish(mqtt ping request) timing in the wireshark.
2. Time T_2 is the amqp receive(amqp connection start with ack) timing in the wireshark

Considering the Adapter for loopback traffic transmit(wireshark capture) timestamps T_1 and T_2, end-to-end delay for each message is measured as:

$$Delay = T_2 - T_1 \tag{11}$$

Results and discussion

The experimental findings of this paper are described in this section. The experimental results show the importance of separate queues for emergency events. Figure 5 depicts the total time taken via three queues. It is evident that the time taken by the

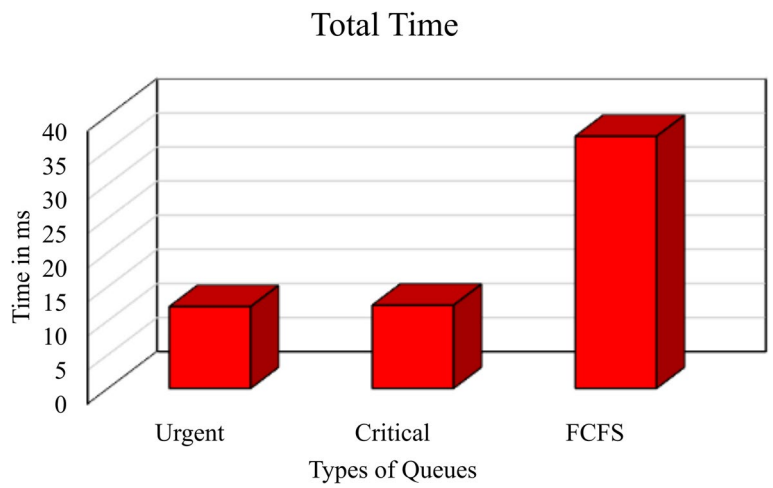

Fig. 5 Total time taken to publish messages by Urgent, Critical, and FCFS queues

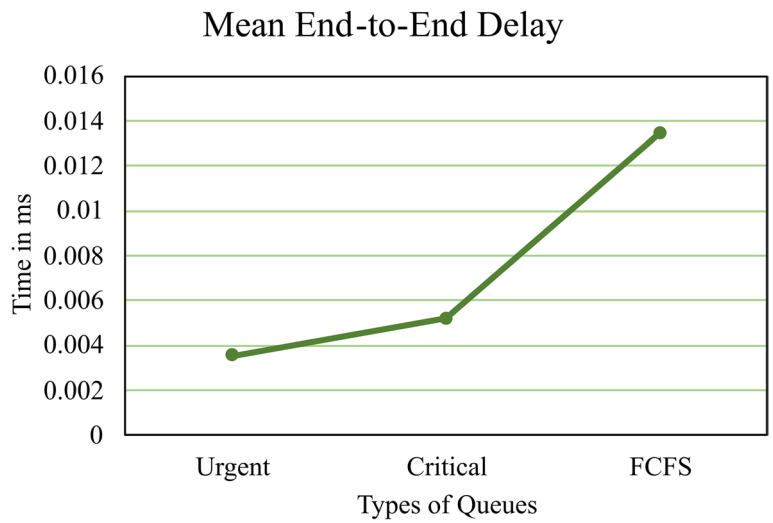

Fig. 6 Mean end-to-end delay of Urgent, Critical, and FCFS queues

FCFS approach is more. For more extended communication, it is always better to have separate queues to avoid delay at the receiver end. Figure 6 depicts the mean end-to-end delay in publishing messages through three queues. The delay is more for FCFS in comparison to Urgent and Critical queues. Figure 7 depicts the mean variation in delay in publishing messages via three queues. The mean jitter is more for the FCFS approach as bandwidth is depleted, then data packets must be reassembled at the receiver's end, adding to the amount of jitter. Figure 8 depicts the mean response time of publishing messages through three queues. FCFS requires more time to publish messages. The Urgent and Critical queues outperform the FCFS queue in all the results. This is because all messages are published via the FCFS queue, which requires more computation, bandwidth, and adding more to the jitter. Because of all these reasons, a separate queue for emergency events is necessary for extended MQTT communication.

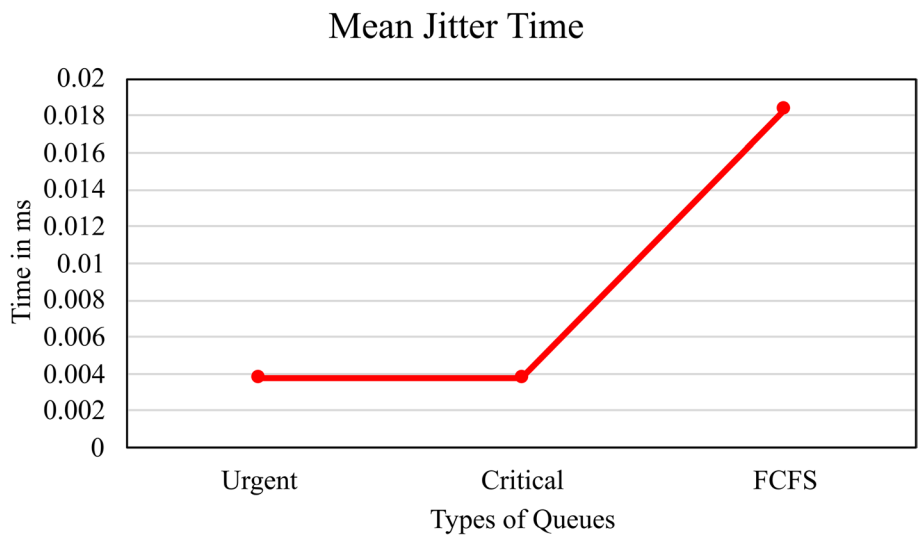

Fig. 7 Mean jitter of Urgent, Critical, and FCFS queues

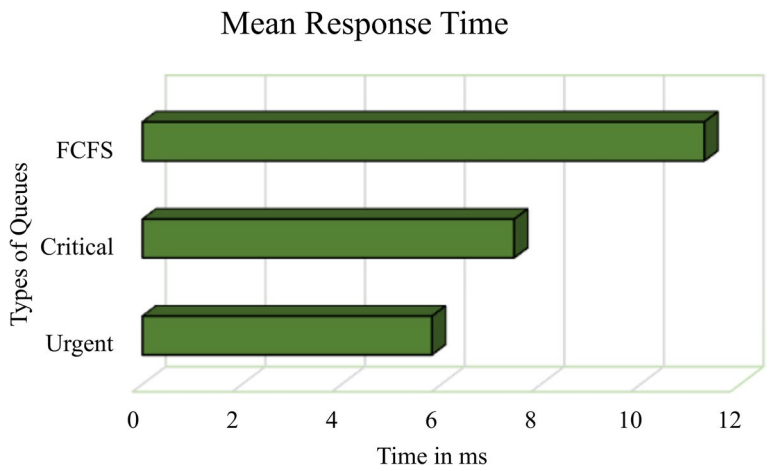

Fig. 8 Mean response time for Urgent, Critical, and FCFS messages

Conclusions

In a publish/subscribe system, prioritizing the emergency messages across topics is a challenging issue that, if not addressed, could negatively influence several application types that depend on it. This paper described a new strategy for supporting high-priority messages in IoT for fast and reliable delivery of messages without modifying the MQTT standards. The proposed method queued communications based on priority levels, such as Urgent, Critical, and FCFS. Furthermore, the proposed method saves the messages in the MySQL database for later analysis. We experimentally implemented the proposed approach using the RabbitMQ message broker, Wireshark, and Python to evaluate its performance. We compared the proposed method and the existing system based on end-to-end delay, total time, response time, and jitter. The outcome shows that the proposed approach performs better than the current FCFS approach.

Future enhancement

However, this work can be extended in the following ways:

1. To analyze the stored messages using machine learning techniques.
2. To implement a dynamic priority messaging approach.
3. To overview potential problems arising from QoS-1 and queuing messages for guaranteed delivery.

Abbreviations

AMQP	Advanced Message Queuing Protocol
FCFS	First Come First Served
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
MQTT-SN	MQTT for Sensor Networks
QoS	Quality of service

Acknowledgements
Not applicable.

Authors' contributions
P S Akshatha: conception and design of study, acquisition of data, analysis and/or interpretation of data, writing — original draft, Writing — review and editing. S Divyashree: conception and design of study, analysis and/or interpretation of data, writing — original draft. S M Dilip Kumar: writing - original draft, analysis and/or interpretation of data, writing — review and editing. All authors read and approved the final manuscript.

Authors' information
P S Akshatha: Ms. Akshatha P S received a B.E. degree in 2005 and an M.Tech degree in 2013 from Vishweswaraiah Technological University, Belgaum, and Lingaya's University, Haryana, respectively. She has been pursuing a full-time Ph.D. from Bangalore University since October 2020. All three degrees are in the field of Computer Science and Engineering. She has around 12 years of teaching experience, she has authored over 25 papers published in international journals and conferences, including one that received the Best Paper Award at an international conference. She has also contributed two book chapters to the Springer series. Currently, she is working as a Senior Assistant Professor in the Department of Artificial Intelligence and Machine Learning at New Horizon College of Engineering, Bengaluru. Her current research focuses on Computer Networks, Genetic Algorithms, Internet of Things.
S Divyashree: Ms. S Divyashree received the B. E and M. Tech degrees in 2020 and 2023 respectively in Computer Science and Engineering discipline. She is currently working as an Assistant professor in the Department of Information Science and engineering, East West Institute of Technology, Bengaluru. She is pursuing phd at Vishweswaraiah Technological University, Belgaum in Machine learning and Artificial intelligence.
Dr. S. M Dilip Kumar: Dr. S. M Dilip Kumar is presently serving as a Professor in the Department of Computer Science and Engineering, University of Visvesvaraya College of Engineering (UVCE), Bengaluru, India. He is also heading the Training and Placement Office at UVCE. Dilip Kumar has been teaching Computer Science for the past 26 years and pursuing research for 20 years. He has guided eight Ph.D candidates and six are pursuing Ph.D under his guidance. Dilip Kumar has published 132 papers in International Journals including IEEE, Elsevier, Springer, etc. and Conferences, and has received six best paper awards at International Conferences. More than 30 papers have appeared in best quartile SCI Rank Journals out of which 6 papers have appeared in Q1 Journals and the number of Google Scholar citations is 969 till date. He has published two Indian Patents. He has delivered more than 50 technical talks in National level seminars, workshops, short-term courses and faculty development programs. He has completed two research projects, both sponsored by the Science and Engineering Research Board, Department of Science and Technology (SERB-DST), Government of India in the areas of grid computing and Internet of Things. He has completed two consultancy projects in the areas of mobile governance and e-FMS sponsored by the Government of Karnataka. He has served in over 100 technical committees of the State Government, Universities and Colleges. His area of interest includes Internet of Things, Edge and Fog Computing.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 28 November 2023 Accepted: 24 February 2024
Published online: 06 March 2024

References

1. Chekired DA, Khoukhi L, Mouftah HT (2018) Industrial IoT data scheduling based on hierarchical fog computing: A key for enabling smart factory. IEEE Trans Ind Inf 14(10):4590–4602
2. Rayan A, Taloba AI, Abd El-Aziz RM, Abozeid A (2020) IoT enabled secured fog based cloud server management using task prioritization strategies. Int J Adv Res Eng Technol 11(9):01–12
3. García-Magariño I, Sendra S, Lacuesta R, Lloret J (2018) Security in vehicles with IoT by prioritization rules, vehicle certificates, and trust management. IEEE Internet Things J 6(4):5927–5934
4. Akshatha PS, Dilip Kumar SM (2023) MQTT and blockchain sharding: An approach to user-controlled data access with improved security and efficiency. Blockchain Res Appl 4(4):100158–100171
5. Ferrari P, Flammini A, Sisinni E et al (2018) Delay estimation of industrial IoT applications based on messaging protocols. IEEE Trans Instrum Meas 67(9):2188–2199
6. Akshatha PS, Dilip Kumar SM, Venugopal KR (2022) MQTT Implementations, Open Issues, and Challenges: A Detailed Comparison and Survey. Int J Sensors Wirel Commun Control 12(8):553–576
7. Safara F, Souri A, Baker T et al (2020) Prinergy: A priority-based energy-efficient routing method for IoT systems. J Supercomput 76(11):8609–8626
8. Wadhwa H, Aron R (2023) Optimized task scheduling and preemption for distributed resource management in fog-assisted IoT environment. J Supercomput 79(2):1–39
9. Zunino C, Cena G, Scanzio S, Valenzano A (2023) Adaptive Seamless Redundancy to Achieve Highly-Dependable MQTT Communication. IEEE Trans Ind Inf 20(1):984–994
10. Hintaw AJ, Manickam S, Aboalmaaly MF, Karuppayah S (2023) MQTT vulnerabilities, attack vectors and solutions in the internet of things (IoT). IETE J Res 69(6):3368–3397
11. Oh SC, Kim YG (2019) A Study on MQTT based on Priority Topic for IIoT. J Inst Internet Broadcast Commun 19(5):63–71
12. Tabinda Pathania N, Jain R, Malik N (2019) Traffic Prioritization in Message Queue Telemetry Transport Protocol. Think India J 20(30):1006–1014
13. Al Enany MO, Harb HM, Attiya A (2021) A New Back-off Algorithm with Priority Scheduling for MQTT Protocol and IoT Protocols. (IJACSA) Int J Adv Comput Sci Appl 12(11):1–10
14. Baldoni R, Bonomi S, Platania M, Querzoni L (2012) Dynamic message ordering for topic-based publish/subscribe systems. In: IEEE 26th international parallel and distributed processing symposium. IEEE, Shanghai, p 909–920. https://doi.org/10.1109/IPDPS.2012.86
15. Hwang K, Lee JM, Jung IH, Lee D-H (2019) Modification of mosquitto broker for delivery of urgent MQTT message. In: 2019 IEEE Eurasia conference on IOT, communication and engineering (ECICE). IEEE, Yunlin, p 166–167. https://doi.org/10.1109/ECICE47484.2019.8942800
16. Puthiyidam JJ, Joseph S (2022) Prioritization of MQTT Messages: A Novel Approach. Commun Comput Inf Sci 1894:40–52
17. Shahri E, Pedreiras P, Almeida L (2022) Extending MQTT with real-time communication services based on SDN. Sensors 22(9):3162–3181
18. Uchida N, Endo S, Ishida T, Yuze H, Shibata Y (2022) Enhanced MQTT Method with IoT Data Priority Controls for Scalability and Realiability on Early Landslide Warning System. In: International Symposium on Mobile Internet Security. Springer, South Korea, p 257–267. https://doi.org/10.1007/978-981-99-4430-9_19
19. Wireshark (1998) GPL-2.0-or-later. https://www.wireshark.org/. Accessed 25 Oct 2023

Publisher's Note