Computer Communications 220 (2024) 43–51

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

PrioMQTT: A prioritized version of the MQTT protocol ☆

Gaetano Patti *, Luca Leonardi, Giuseppe Testa, Lucia Lo Bello

Department of Electrical, Electronic and Computer Engineering, University of Catania, Catania, Italy

ARTICLE INFO

Keywords:
MQTT
Priority-based communications
Low-latency networks

ABSTRACT

MQTT is an application layer protocol that, thanks to its simplicity and low overhead, is widely used in the Internet of Things (IoT) devices typically found in home automation and consumer applications. The MQTT properties make it an interesting option also for Industrial IoT (IIoT) applications. However, MQTT is not specifically devised for IIoT applications requiring low-latency and the support for time-constrained transmissions. For this reason, this paper proposes an IIoT-enabled version of MQTT called a Prioritized MQTT (PrioMQTT) that is able to provide low latencies to time-critical messages. Unlike the standard MQTT, PrioMQTT adopts the UDP/IP stack, which is more suitable than TCP/IP for low-latency communications. Moreover, PrioMQTT introduces a mechanism to prioritize the time-critical messages over the non-time-critical ones. The combination of the UDP/IP stack and priority support in the PrioMQTT protocol is achieved while maintaining the compliance with the MQTT standard message format. As a result, PrioMQTT can be implemented on commercial-off-the-shelves (COTS) devices without hardware modifications. The paper describes the PrioMQTT protocol and investigates its performance through an assessment in a realistic industrial scenario and in comparison with the standard MQTT protocol.

1. Introduction

Industrial communications play a key role in the Industry 4.0 era. In fact, the shift to the new intelligent, connected and automated industry model requires that industrial networks coordinate a large number of devices, sensors, actuators and control systems to ensure an efficient and reliable production flow [1–3]. Given the high number of smart devices to be interconnected, an interesting option for Industry 4.0 is to try and adopt some of the solutions that are already used in Internet of Things (IoT) scenarios [4–8].

In these scenarios, the application layer protocols aim to enable communications between different types of applications and devices. Since the number of applications grows, several IoT application layer protocols, among them the ones outlined in [9] (i.e., CoAP, MQTT, AMQP, etc.), have been enhanced and modified to meet specific application needs. For example, a distributed CoAP time server for industrial domain is presented in [10], a distributed MQTT broker system that guarantees high scalability and resiliency to failures is proposed in [11], and an intelligent congestion control algorithm (iCoCoA) for CoAP is introduced in [12] to be used in dynamic environments. One interesting option for the application level of the protocol stack is the MQTT [13] standard, which is one of the most popular protocols for the IoT [14]. MQTT is a publish/subscribe protocol in which multiple publishers send messages to a node, called broker. Each published message is associated to a topic, i.e., a string used to identify and route messages between the publishers and the subscribers. The broker, in turns, forwards the messages to all the interested subscribers in the network. MQTT is widely used on devices with limited resources to allow data exchange among networks made of low-cost devices. MQTT is lightweight, efficient, and reliable. All these properties make it a very interesting candidate for the Industrial IoT (IIoT) to modernize manufacturing projects [15,16]. However, many IIoT applications exchange time-critical messages [17–19], but MQTT is not suitable for them. The reason for this limitation is twofold. First, MQTT is designed to work over the TCP transport protocol, but for time-critical

☆ The work of Gaetano Patti is funded by the Project "Soluzioni innovative per la connettività wired e wireless nell'industria 4.0 a supporto dell'efficienza e resilienza della fabbrica", within the action IV.4 – "Dottorati e contratti di ricerca su tematiche dell'innovazione del nuovo Asse IV del PON Ricerca e Innovazione 2014–2020 Istruzione e ricerca per il recupero – REACT-EU". The work of Luca Leonardi, who has contributed to the design of the PrioMQTT protocol, is funded by the European Union (NextGenerationEU), through the MUR-PNRR project SAMOTHRACE (ECS00000022). The work of Giuseppe Testa, who has contributed to the performance assessment, has been supported by MUR under the project RESTART "RESearch and innovation on future Telecommunications systems and networks, to make Italy more smart".

* Corresponding author.
E-mail addresses: gaetano.patti@unict.it (G. Patti), luca.leonardi@unict.it (L. Leonardi), giuseppe.testa@phd.unict.it (G. Testa), lobello@unict.it (L. Lo Bello).

https://doi.org/10.1016/j.comcom.2024.03.018
Received 6 January 2024; Received in revised form 4 March 2024; Accepted 21 March 2024
Available online 26 March 2024

communications that require low latency UDP is more suitable than TCP. In fact, unlike TCP, UDP does not provide any congestion control mechanism (that would make it difficult to obtain low latencies) and avoids unnecessary retransmissions. With UDP, retransmissions are not performed by the transport layer, but they are left to the upper-layer time-critical application that decides, based on the message's lifetime, if the message to be retransmitted is still temporally valid and worth of another transmission try, or if it has already expired, so there is no need for retransmitting it. The second reason why MQTT is not suitable for time-critical messages that require low latencies is that MQTT lacks prioritization, and therefore it does not enable the network to provide time-critical messages with a different service than non-time-critical ones. To enable the use of MQTT in IIoT applications requiring time-critical communications, this work proposes the Prioritized MQTT (PrioMQTT), an MQTT version that modifies the broker architecture to support message prioritization and is based on the UDP protocol. Applications that would benefit from PrioMQTT are found in Industrial Cyber–Physical Systems (ICPSs), where IoT technologies enable advanced communication solutions for interconnected machinery and collaborative robots. PrioMQTT exploits the messages' priorities to implement a priority-based schedule that reflects the different temporal constraints of the messages to be transmitted. PrioMQTT maintains the backward compatibility with the MQTT standard protocol, as it is able to support the standard MQTT clients and, at the same time, the time-critical transmissions of the PrioMQTT clients.

The main contributions of this paper are:

- The design of the PrioMQTT protocol.
- The design of the broker software architecture.
- An experimental assessment of PrioMQTT on real devices in a realistic industrial scenario.

The paper is organized as follows. Section 2 presents a comparison between PrioMQTT and related works. Section 3 provides an overview of the MQTT standard. Section 4 describes the PrioMQTT architecture and the relevant design challenges. Section 5 presents the results of an extensive assessment of the PrioMQTT protocol. Section 6 discusses the advantages and the limitations of PrioMQTT, also providing hints for future works. Finally, Section 7 summarizes and concludes the paper.

2. Related work and comparison

Several works in the literature addressed the MQTT suitability for industrial applications [20–22]. For instance, the work in [15] investigates the communication delay obtained by the MQTT protocol in IIoT applications characterized by message transmissions from field devices to a remote cloud and vice versa. The results show that MQTT obtains delays in the order from tens to hundreds of milliseconds, thus representing a suitable solution for inexpensive industrial-grade IIoT devices. However, the work in [15] does not address time-critical applications, therefore the protocol's ability to cope with the time constraints of the exchanged messages is not assessed.

The work in [16] proposes an innovative communication framework based on MQTT to obtain secure and scalable Machine-to-Machine (M2M) communications and its implementation and evaluation in a small-scale industrial testbed. The work focuses on security and scalability aspects and does not address time-critical communications. Conversely, IIoT applications typically require the network to provide time-constrained message transmissions [23–25]. As the MQTT protocol lacks the support for time-critical low-latency communications, an option is to modify the network configurations at runtime using Software-Defined Networking (SDN) [26,27]. Another option is to modify the protocol behavior to enable time constrained transmissions [28,29].

A standard extension of MQTT, called MQTT for Sensor Networks (MQTT-SN) [30], is defined by the OASIS MQTT Technical Committee. Unlike the standard MQTT protocol, which uses TCP, the MQTT-SN clients use the UDP protocol, which allows for lightweight and connectionless communications. A simplified example of MQTT-SN architecture is shown in Fig. 1. MQTT-SN requires the adoption of special nodes, called MQTT-SN gateways. The gateways act as clients for the MQTT broker and translate messages between MQTT-SN and MQTT. All the communications have to traverse the MQTT broker. The MQTT-SN protocol does not provide time-constrained communications support, as it does not define any mechanism to differentiate the transmissions according to the time constraints of the messages. Moreover, due to its architecture that requires the adoption of gateways, transmissions go through the standard MQTT network, which uses the TCP protocol. However, to prevent congestion , TCP implements some flow control mechanisms that may introduce latency due to transmission rate adjustments. In addition, TCP requires acknowledged transmissions, thus introducing additional latency due to the acknowledgment processing and the potential retransmissions. As a consequence, MQTT-SN has the same limitations of the standard MQTT protocol for IIoT applications.

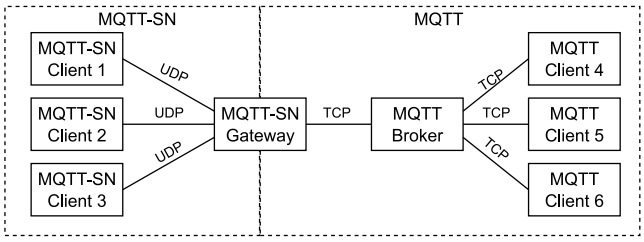

Fig. 1. Example of an MQTT-SN network.

As discussed in [15,31], MQTT is suitable for communications within the entire computing continuum, i.e., from IoT nodes to the edge and to the cloud, and vice versa. The PrioMQTT here proposed is designed to handle the time-constrained flows that are typically found at the edge level.

Several works in the literature proposed solutions to enable time-critical communications over MQTT. In the following subsections, the different approaches are classified into three categories, called (i) modified MQTT, (ii) SDN-based network management of MQTT transmissions, and (iii) hybrid solutions that combine network management (with or without SDN) with a modified version of MQTT.

2.1. Modified MQTT solutions

The approaches proposed in [32,33] address the support for message priority in the MQTT protocol. In particular, the work in [32] proposes a modified MQTT version in which the broker provides three different transmission queues for the highest priority (i.e., urgent) messages, the medium priority (critical) messages, and the non-real-time ones. Conversely, the approach in [33] uses a two-bit priority field on the MQTT message header to provide four priority levels for the messages in the broker transmission queue. Although these approaches use message prioritization on the broker, they handle a low number of priorities, i.e., at most four priorities. Conversely, PrioMQTT addresses time-critical communications using the UDP protocol for time-critical messages and encoding a 64-bit priority value in the MQTT message header to handle a number of priorities at the broker that is significantly higher than the one supported by the approaches in [32,33]. As a result, PrioMQTT makes it possible to adopt a scheduling policy that assigns priority values to messages according to their different time constrains with a thinner granularity.

The paper in [34] proposes a priority control mechanism running on the broker. The broker acts as a master that schedules and authorizes data transmission from the publishers to an application server. The approach in [34] enables prioritized communications, but it is not compliant with the MQTT protocol and does not support many-to-many communications. Conversely, the PrioMQTT here proposed not

only offers prioritized communications for time-critical messages, but it also supports many-to-many communications in the same way as the standard MQTT protocol. PrioMQTT also maintains the interoperability with the standard MQTT.

2.2. SDN-based network solutions

Some works in the literature propose the combination of MQTT and SDN to achieve time-critical communications in MQTT-based networks. For example, the work in [35] proposes RT-MQTT, which extends the MQTT protocol by enabling the explicit association of real-time requirements (i.e., priority, deadline, and periodicity) to topics and end-nodes. RT-MQTT uses the OpenFlow SDN framework to configure the network and introduces a component, called a Real-Time Network Manager (RT-NM), that runs on the broker. The RT-NM module captures all the MQTT messages with the aim of extracting their real-time requirements. These requirements are analyzed and sent to the OpenFlow Controller. The latter, in turn, manages the flow tables of the OpenFlow Switches for creating real-time communication channels that meet the real-time demands of the topics. Some analytical models to calculate the response time bounds of the RT-MQTT protocol are addressed in [36,37]. While RT-MQTT works at the network layer, the PrioMQTT protocol proposed in this paper works at the application layer. As a result, it is reasonable to assume that RT-MQTT can be easily adapted to work in combination with PrioMQTT.

Another interesting solution, proposed in [38], provides for the creation of clusters by exploiting the SDN paradigm. Each cluster is assigned to a dedicated MQTT broker, with the aim of minimizing the communication delays and supporting very low latencies. The performance evaluation presented in [38] shows that the approach significantly improves latency and network utilization compared to other existing SDN-based MQTT brokers, but it does not support any prioritization to reflect the different temporal constraints of the messages. Moreover, it requires to work on networks that use OpenFlow compliant switches. Conversely, PrioMQTT provides priority-based transmissions to cope with the messages' temporal constraints and does not require special hardware.

2.3. Hybrid solutions

The approach presented in [39] is specifically devised for the communications between different edge networks. It uses the RT-MQTT protocol [35] for the intra-edge network communications and combines OpenFlow SDN with a modified MQTT version that uses prioritized multicast for the inter-edge network communications, with the aim to minimize the delays and the network load for real-time messages. The approach in [39] allows for a significant delay reduction compared to the standard MQTT, but it only addresses edge networks that use the OpenFlow SDN framework. Consequently, RT-MQTT requires OpenFlow-compliant switches. On the contrary, PrioMQTT works at the application layer and its adoption is not bound to the underlying network technology. PrioMQTT does not require special hardware and is suitable for being implemented in low-cost industrial-grade IIoT devices.

The work in [40] presents a set of extensions for MQTT-SN that provides for associating commonly used real-time attributes (e.g., priority, periodicity, and deadline) to each topic. These extensions enable the online definition of these attributes and the configuration of the network interfaces to handle messages at the MAC layer according to the topics' priorities. This approach limits the extensions to the MQTT-SN subnetwork (Fig. 1), and therefore it does not offer support for time-critical communications to an entire network that adopts both MQTT and MQTT-SN. Conversely, the PrioMQTT protocol here proposed is a prioritized version of the standard MQTT protocol that enables message priority on all the network nodes to cope with the time constraints of the messages.

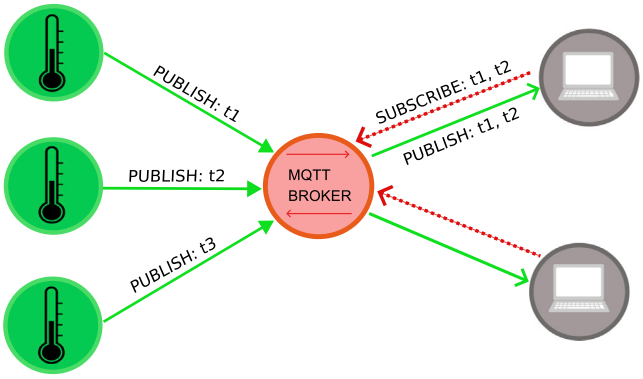

Fig. 2. Example of an MQTT network scenario.

3. MQTT overview

MQTT [41] is a simple, lightweight and bandwidth-efficient publish/subscribe application-layer protocol that is suitable for M2M communications and operates over TCP/IP.

MQTT defines two kinds of nodes, i.e., the broker and the clients. The broker is the node in charge of handling all the connections and communications between the clients. Clients establish a connection at the application layer with the broker, then they either subscribe or publish data on specific topics. All the published messages are transmitted to the broker that, in turn, forwards them to the subscribers. A topic is a string used by the MQTT protocol to filter and forward messages. According to the MQTT protocol, publishers send messages to designated topics, and subscribers can subscribe to these topics to receive the messages. A topic has a hierarchical structure, in which each level is separated by a forward slash. For instance, "home/livingroom/temperature" is a topic that represents the temperature in the living room of a home.

MQTT defines three Quality-of-Service (QoS) policies for message transmission, i.e., at least once, at most once, and exactly once, which correspond to different QoS values. The messages with QoS equal to 0 (at most once) are transmitted without acknowledgment, for the sake of low communication overhead and latency. The messages with QoS equal to 1 (at least once) have to be delivered at least once, with multiple retransmissions of the same message if the acknowledgment is lost. Finally, messages with QoS equal to 2 (exactly once) are guaranteed to be delivered exactly once. To achieve this, QoS 2 requires a four-part handshake between the publisher, the broker and each subscriber.

In MQTT the broker is the crossing point of all the communications. No direct message exchanges between publishers and subscribers are allowed, and therefore the communication between them is asynchronous. This allows for decoupling the publishers and the subscribers. In fact, the publishers do not know which nodes will consume their messages.

Fig. 2 represents a simple MQTT scenario in which three temperature sensors publish their values, each one with a specific topic ("t1", "t2" and "t3" in Fig. 2). The clients interested to the sensor temperature subscribe to these topics, e.g., the top-right client in Fig. 2 subscribes to the topics t1 and t2. This way, once the messages will have become available, the broker will send them to the relevant subscribers.

The next section presents the design of PrioMQTT, a modified version of MQTT that supports prioritized low-latency communications at the application layer, while maintaining the compatibility with the standard MQTT protocol.

4. Design

The main idea behind the design of the PrioMQTT is to remove the additional latencies of the TCP due to the flow control, the acknowledgments and the retransmissions, while providing a mechanism to differentiate the message transmission priorities so that they reflect the temporal constraints of the messages.

To this aim, multiple approaches can be adopted. For instance, a solution is to have the broker poll the publishers to authorize their transmissions, as proposed in [34]. Polling-based solutions enable transmission scheduling, but they also entail an additional network overhead due to the transmission of the polling messages. Polling is not a suitable solution for applications that require low-latency communications, because each polling message transmission introduces an additional contribution to the latencies of the messages.

The approach for prioritized time-constrained communications proposed in this work, the PrioMQTT, differently from the MQTT standard adopts the UDP protocol instead of the TCP, while maintaining the same message format defined in the MQTT specifications [41]. UDP not only avoids the burden of establishing connections, but also it does not implement congestion control, which would make it difficult to allow the timely delivery of the time-critical messages. Moreover, UDP does not perform retransmissions and this is beneficial to several industrial applications that manage by themselves the retransmission of messages, whenever needed. In fact, the applications are aware of the messages' lifetime, so they can determine if a message to be sent is still temporally valid, and therefore worth of another transmission try, or if it has already expired, so there is no need for retransmitting it.

4.1. Introducing message priority

The MQTT specification v5 [41] introduced the message property fields in the MQTT messages to encode specific attributes within each message. PrioMQTT uses the User Properties field to encode a message priority. User Properties consist of an array of UTF-8 key/value pairs, which enable the inclusion of user-specified data in MQTT messages. In PrioMQTT , priority is encoded as a string containing a numeric unsigned integer value and the lowest values correspond to the highest priority. In our implementation, the priority string is converted into a 64-bit unsigned integer, but a shorter-sized integer can be configured depending on the application needs.

The introduction of message priorities enables the support for different message scheduling policies, such as fixed priority ones or dynamic priority ones [42–44]. To allow more flexibility by design, the priority is transmitted within each published message. This leaves the publisher free to assign priorities either to topics or to each message. In the first case, the same priority is assigned to all the messages of a given topic. In the second case, the publisher assigns different priorities to the messages of a given topic. This choice has to be made at the network design time.

The broker maintains a transmission queue that contains the messages to be forwarded to the subscribers, ordered by priority. The messages with the same priority are transmitted in a First-In First-Out (FIFO) order.

4.2. Broker architecture

In the PrioMQTT protocol , the broker handles all the communications between the clients. Fig. 3 shows the main broker architecture components.

The network stack at the bottom of Fig. 3 creates and manages the network entry points for the broker. The received MQTT messages are transmitted to the *Message receiver and processor* component, which is in charge of parsing and executing the state machines of the broker to manage the MQTT connections, the subscriptions, and all the core logic of the MQTT protocol. The *Message receiver and processor* does not deal with the transport protocol adopted, as it handles standard MQTT messages only. The separation between the network stack and the PrioMQTT layer allows for the adoption of multiple transport layers working simultaneously. In addition to UDP, other choices are also possible, e.g., the Audio Video Transport Protocol (AVTP).

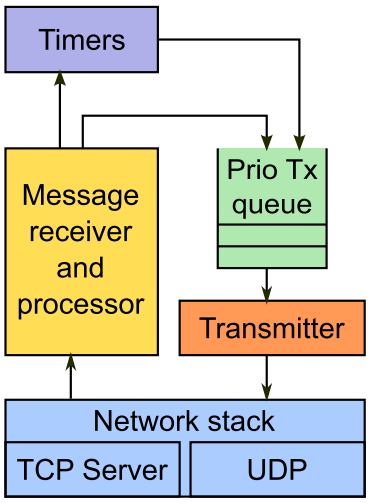

Fig. 3. The PrioMQTT broker software architecture.

In Fig. 3 the Timers module is devised to manage time-driven actions, e.g., possible retransmissions and management of the application layer connections.

The messages to be transmitted are enqueued in the *Prio Tx queue*, where messages are ordered by priority (as said before, the lower the value the higher the priority). When the *Message receiver and processor* or the *Timers* components have a message to transmit, they enqueue the message into the *Prio Tx queue*. The Prio Tx queue is implemented in software. The queue size is a configurable parameter that can be set according to the workload of the devices and the amount of memory available on the broker. If the message has to be transmitted to multiple destinations, e.g., in the case of a message with multiple subscribers, two enqueuing methods can be alternatively adopted. With the first method, *the Prio Tx queue* stores only one message and an array of addresses for the destination nodes. This choice allows for memory saving and avoids multiple queue operations, as the queue does not need to store one separate entry for each intended destination of the same message. The second method envisages that multiple references to the same message are stored in the transmission queue, each one with a different destination. This choice allows for a finer granularity in message scheduling.

In PrioMQTT, the standard MQTT traffic, i.e., the one transmitted with the TCP protocol, is considered low priority traffic, and therefore the messages transmitted using TCP obtain the lowest priority.

Finally, the role of the *Transmitter* component in Fig. 3 is to pick the messages from the queue and transmit them.

The design of PrioMQTT provides several advantages, as it enables:

- the combination of standard MQTT transmissions with time-critical constrained transmissions on the same broker, while maintaining the backward compatibility with standard MQTT client nodes;
- the adoption of multiple transport layers that work simultaneously;
- the use of different non-preemptive message transmission scheduling policies;
- the reduction of the message latency, thanks to the adoption of the UDP transport protocol for time-critical transmissions.

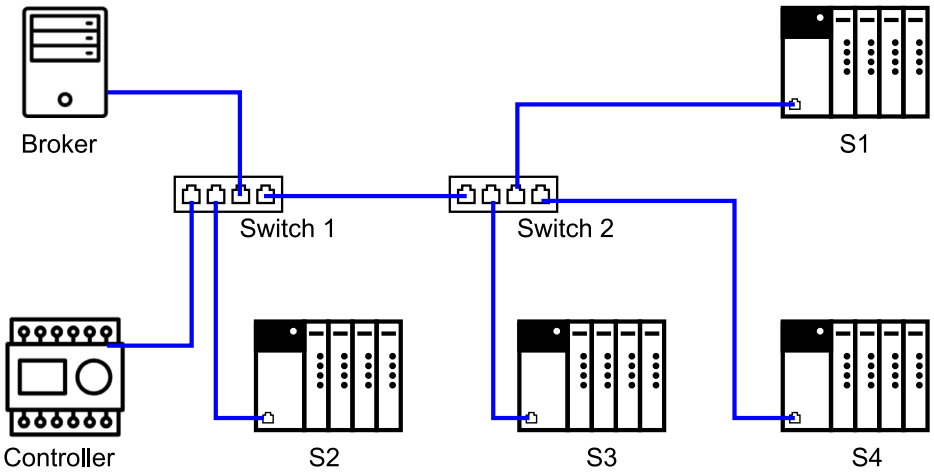

Fig. 4. The assessed scenario.

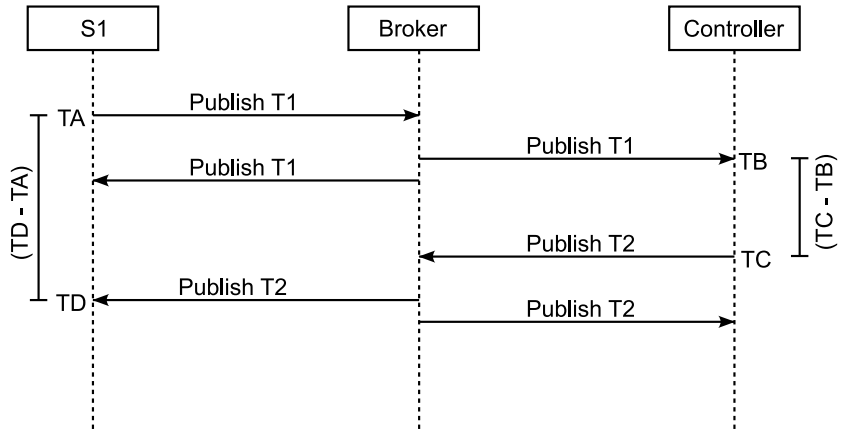

Fig. 5. Messages exchanged between the module S1 the controller.

5. Performance assessment

This section discusses the performance evaluation of the PrioMQTT protocol.

The assessed scenario, shown in Fig. 4, consists of one controller, the broker and four Input/Output (IO) modules (S1, ...,S4) that represent the sensors and actuators of a factory cell. The network is a plain Switched Ethernet working at 100 Mbps, which is a datarate typically used in industrial networks.

The broker runs on a 5 GHz 6 core Intel i7-10750H CPU, equipped with a 16 GB RAM and running the Linux Ubuntu 20.04 operating system. Each client node, i.e., the controller and the IO modules, runs on a Raspberry Pi 4 with the Raspbian 64-bit operating system.

The experimental setup is based on the one adopted in the work in [15].

Each IO module has a couple of associated topics, here called data topic and command topic, respectively. In particular, the data topic associated to an IO module is used for transmitting data to the controller, the command topic associated to the same IO module is used for receiving the commands from the controller. The controller subscribes to all the data topics. When the controller receives a data topic message from a given IO module, it replies with a message published to the command topic associated to the same IO module.

In addition, each client subscribes to the same topic it publishes, so as to obtain a feedback that the broker transmitted it.

Fig. 5 shows the message exchange between the module S1 and the controller.

The time instants TA, TB, TC and TD, shown in Fig. 5 for the node S1, are defined as follows:

- TA: time at which the S1 node publishes a message to the topic T1;
- TB: time at which the controller receives a topic T1 message from the broker;
- TC: time at which the controller publishes the reply message to the topic T2;
- TD: time at which S1 receives a topic T2 message from the broker.

The performance metrics used in the evaluation are the Round Trip Time (RTT), the average queueing time (Tq), the maximum number of elements in the broker transmission queue, the median value of elements in that queue, and the Message Loss Ratio (MLR).

The RTT is calculated, with reference to Fig. 5, as in Formula (1), where (TD-TA) is the time difference between the transmission of a topic T1 message and the reception of a topic T2 message and (TC-TB) is the controller processing time.

$$RTT = (TD - TA) - (TC - TB). \tag{1}$$

The above explained times are acquired for each S node (i.e., S1, S2, S3, S4) in the network and for each couple of dedicated topics.

The average queueing time (Tq) is the average time spent by the messages while waiting in the broker transmission queue. Finally, the Message Loss Ratio (MLR) is calculated as the percentage of lost messages over the transmitted messages.

The configuration of each topic is shown in Table 1.

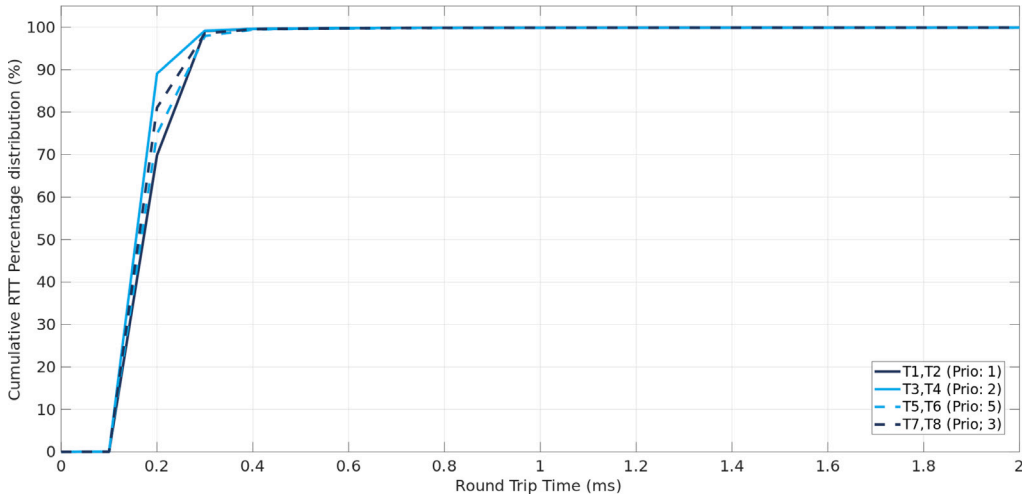

Fig. 6. RTT results with message payload size equal to 100 bytes.

Table 1
Topics' configurations.

Topic	Publisher	Subscribers	Period (ms)	Payload size (B)
T1	S1	{Controller, S1}, All	1	100, 500, 1000
T2	Controller	{Controller, S1}, All	1	100, 500, 1000
T3	S2	{Controller, S2}, All	2	100, 500, 1000
T4	Controller	{Controller, S2}, All	2	100, 500, 1000
T5	S3	{Controller, S3}, All	5	100, 500, 1000
T6	Controller	{Controller, S3}, All	5	100, 500, 1000
T7	S4	{Controller, S4}, All	3	100, 500, 1000
T8	Controller	{Controller, S4}, All	3	100, 500, 1000

In this evaluation, the PrioMQTT priorities are statically assigned to topics, i.e., all the messages of a given topic are assigned the same priority. In particular, the priorities are assigned according to the topics' transmission periods so that the shorter the period, the higher the priority.

5.1. Evaluation of PrioMQTT varying the message size

The goal of the first evaluation is to measure the impact of the prioritized transmissions on the RTTs with different message sizes. For this reason, the network scenario is assessed with three different message payload sizes, i.e., 100, 500 and 1000 bytes, that are realistic values for industrial applications.

To obtain statistically significant results, a very high number of samples was collected, i.e., from 30 000 samples for the flows with the longest period, i.e., the topics T5 and T6 in Table 1, to 150 000 samples for the flows with the shortest period, i.e., the topics T1 and T2 in Table 1.

The results in terms of cumulative RTT percentage distribution with different payload sizes are shown in the Figs. 6–8.

Fig. 6 shows that with message payload size equal to 100 bytes the cumulative RTT percentage distributions of the different topics are very similar. The reason for this result is the short message size, which entails negligible transmission times, i.e., 15 μs for each hop in this case. As a consequence, the *Transmitter* component (Fig. 3), which is in charge of picking the messages from the queue and transmitting them, is faster than the *Message receiver and processor*, and therefore the messages do not have to wait in the broker transmission queue. Similar results are obtained with message payload sizes equal to 500 bytes, as shown in Fig. 7.

The results in Fig. 7 show that the RTTs values obtained for the topic couples (T1, T2), (T5, T6), and (T7, T8) are very close, while the couple (T3, T4) obtained significantly lower RTTs, because the client S2 is one hop closer to the broker than the other clients, as it is shown in Fig. 4. From the figure it can be seen that the topic couple (T1, T2) obtained slightly lower RTTs than the other two couples at the same hop distance from the broker, i.e., (T5, T6) and (T7, T8). This is the effect of the priority, because T1 and T2 have the highest priorities. The percentage of messages that obtained RTT values lower than 0.7 ms is the 99.0% for (T1, T2), 99.6% for (T3, T4), 97.3% and 97.6% for (T7, T8) and (T5, T6), respectively.

As said before, the better results for (T4, T5) are because the client S2 associated to the topics T4 and T5 is one hop closer to the broker than the other clients. This entails RTTs that, on average, are about 94 μs lower than the those obtained by the couple (T1, T2). This a value is close to the one-hop round trip transmission time of a message, i.e., twice the transmission time, i.e., 47 μs.

Finally, Fig. 8 shows the results with message payload size equal to 1000 bytes.

The results in Fig. 8 show that the priorities used in the transmission queue of the broker improve the performance in terms of latency for the time-critical messages, by allowing the topic couples with higher priorities to obtain lower RTTs than those obtained by the topic couples with lower priorities. In fact, the larger the payloads the higher the transmission times. This entails a high probability that multiple messages will be waiting in the broker transmission queue, because in this case the *Transmitter* is slower than in the case of message payloads equal to 100 bytes. As a consequence, there is a high probability that high priority messages delay the low priority ones that are waiting in the broker transmission queue. For instance, in Fig. 8 the percentage of messages that obtained RTT values lower than 1.2 ms are the 89.6% for (T1, T2), 97.1% for (T3, T4), 74.3% for (T7, T8) and 67.5% for (T5, T6), respectively. These results confirm the benefits of introducing the priorities in PrioMQTT, as the messages with higher priorities obtained lower RTTs.

5.2. PrioMQTT broker performance assessment

This subsection addresses the broker performance in terms of average queuing time, maximum number of elements in the broker transmission queue, median value of elements in that queue, and Message Loss Ratio (MLR). The evaluation was performed in the same scenario presented at the beginning of this Section.

In the assessment, the payload size is set to 500 bytes and the transmission queue size is set to 500 elements.

In the proposed implementation the queue stores one entry for each frame to be transmitted. This means that, if a message has multiple subscribers, multiple references to the same message are stored in

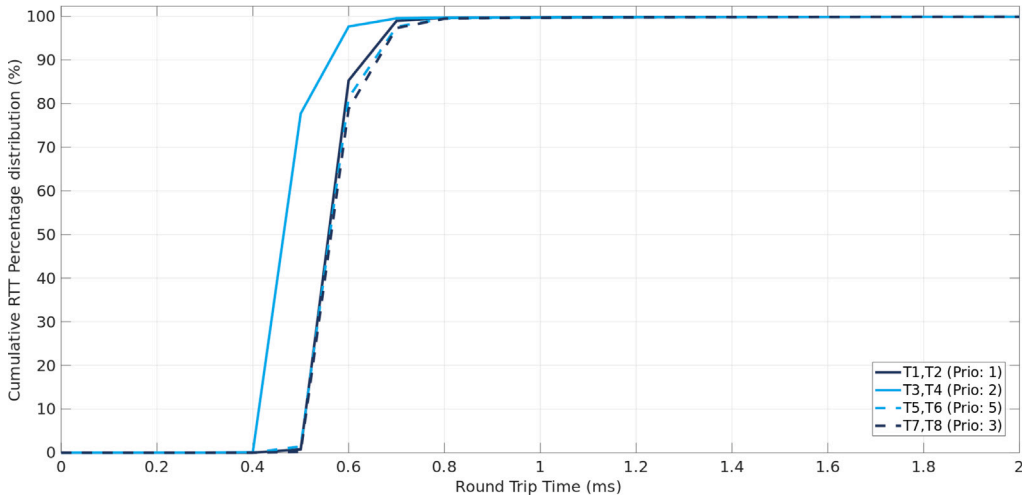

Fig. 7. RTT results with payload size equal to 500 bytes.

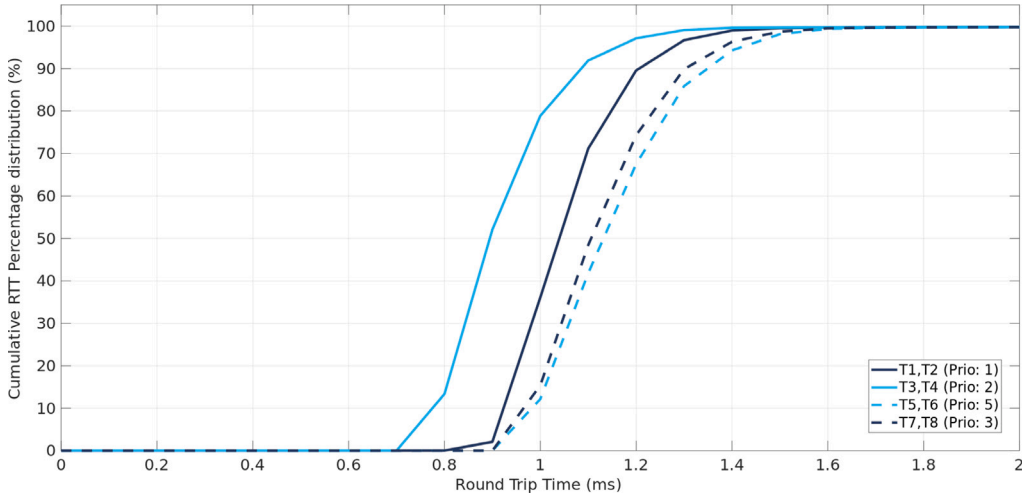

Fig. 8. RTT results with payload size equal to 1000 bytes.

the transmission queue. Measurements were performed in two cases, i.e., with two and five subscribers for each topic, respectively.

In this evaluation, to obtain statistically significant results, a very high number of samples was collected, i.e., from $72 \cdot 10^4$ samples for the flows with the longest period, i.e., the topics T5 and T6 in Table 1, to $36 \cdot 10^5$ samples for the flows with the shortest period, i.e., the topics T1 and T2 in Table 1, for a total of $11.04 \cdot 10^6$ samples.

The average queuing times obtained are shown in Fig. 9.

The results confirm the PrioMQTT ability of differentiating message transmissions according to their priorities, In fact, as shown in Fig. 9, the higher the priority the lower the average queuing time. Compared to the case with two subscribers, the case with five subscribers obtained a slightly higher queuing time because in this case the received messages are transmitted five times, thus increasing the workload on the broker.

The obtained results confirm the benefits of introducing the priorities in PrioMQTT, as the messages with higher priorities obtained lower queuing times.

The maximum number of elements in the broker transmission queue, the median value of elements in that queue, and the Message Loss Ratio (MLR), obtained in the same scenario are shown in Table 2.

The results in Table 2 show that the median number of elements in the broker's queue is lower than 7 in all cases and that the maximum number is lower than 80 in the case with the highest workload. The maximum number of frames in the queue is not an issue, because the queue stores the references (i.e., pointers in C language) to the messages. As a result, in the case of multiple subscribers to the same topic, multiple entries in the transmission queue point to the same message. Moreover, it has to be noted that the broker does not necessarily have to be a resource-limited device.

Table 2
Maximum and median number of elements in the broker transmission queue and MLR.

Number of subscribers	Queue count		MLR
	Median	Max	
2	3	33	$<10^{-7}$
5	6	79	$<10^{-7}$

In all the measurements no message loss was experienced. For this reason, as the number of transmitted messages is higher than 10^7 and all messages were correctly received, the obtained MLR is lower than 10^{-7}, which is a suitable value for industrial applications [45].

5.3. Comparison with the standard MQTT

In the second assessment, the results obtained using the PrioMQTT are compared to those obtained by the standard MQTT in the same

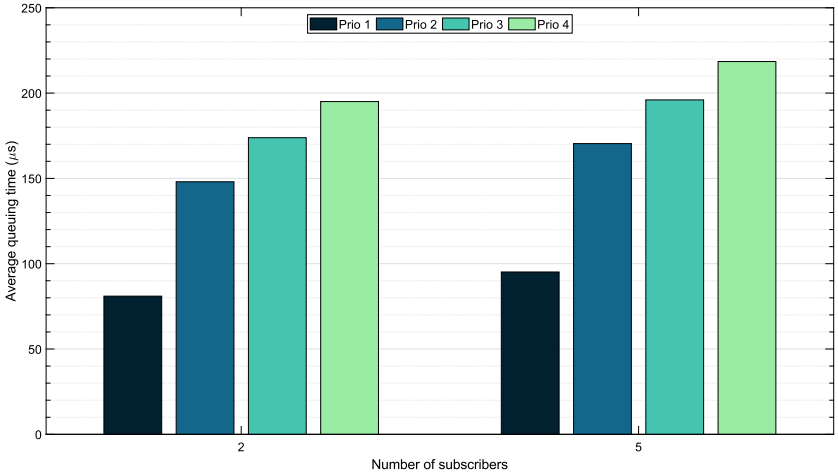

Fig. 9. Average Queuing times.

Table 3
Comparison of the average RTT between PrioMQTT and the standard MQTT protocol.

Topics	PrioMQTT			MQTT	
	Prio	Avg. RTT	Std. Dev.	Avg. RTT	Std. Dev.
T1, T2	1	1.17 ms	0.21 ms	2.40 ms	0.60 ms
T3, T4	2	1.03 ms	0.18 ms	2.94 ms	0.50 ms
T7, T8	3	1.24 ms	0.20 ms	4.03 ms	0.13 ms
T5, T6	5	1.26 ms	0.20 ms	6.04 ms	0.11 ms

scenario. For the standard MQTT the broker Eclipse Mosquitto [46,47] is adopted.

From the assessment presented above in Section 5.1 the most interesting results were obtained with the highest workload. For this reason, in this comparative evaluation with the standard MQTT, a payload size equal to 1000 bytes for all messages is used.

Table 3 shows the average RTT and the standard deviation for each couple of topics using PrioMQTT and the standard MQTT.

The results in Table 3 show that PrioMQTT in the assessed scenario outperforms the standard MQTT approach in terms of RTTs. For instance, for the topic couple (T5, T6) the PrioMQTT obtained an average RTT equal to 1.26 ms, while the standard MQTT obtained an average RTT equal to 6.04 ms.

This result is obtained thanks to the adoption of UDP protocol, which significantly reduces latencies, thus demonstrating the validity of the proposed approach.

6. Discussion and future work

The results show that PrioMQTT reduces the RTT and is able to differentiate the transmissions of messages according to their priority, thus obtaining the shortest RTTs for the messages with the highest priority. In particular, in the assessed scenario the RTTs of all messages were always lower than 1.8 ms. Moreover, a comparative evaluation between PrioMQTT and the standard MQTT was performed. In the assessed scenario, PrioMQTT obtained an RTT reduction from 51.30% to 79.14% over the standard MQTT.

The results in terms of queue utilization demonstrate that PrioMQTT does not require a large memory capacity. Moreover, PrioMQTT does not require additional data structures and maintains the standard MQTT message format. Finally, the proof-of-concept implementation described in this work proves that PrioMQTT can be implemented on low-cost devices, such as Raspberry PI.

However, PrioMQTT has some limitations. The first limitation is that it leaves to the application the task of handling possible retransmissions and out-of-order messages. Such a limitation is common in real-time applications, as the retransmissions and out-of-order messages managing policies are application-dependent.

Moreover, another limitation of PrioMQTT is that it works at the application layer only, thus the message priorities are not used by the underlying network levels. For this reason, future works will address the integration of PrioMQTT with the Time-Sensitive Networking (TSN) family of standards. For example, it will be investigated how to make the MQTT protocol exploit the Asynchronous Traffic Shaping (ATS) for time-critical transmissions.

7. Conclusions

This paper proposed the PrioMQTT protocol, which extends MQTT by introducing the support for message prioritization to enable the transmission of time-critical messages. PrioMQTT also provides time-critical messages with low latency, thanks to the adoption of the UDP protocol.

The protocol was implemented on COTS devices and the assessments were performed with different payload sizes and workloads. The results confirmed the validity of the proposed approach in reducing the RTT while properly handling the priorities of the messages.

CRediT authorship contribution statement

Gaetano Patti: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Luca Leonardi:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Giuseppe Testa:** Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Lucia Lo Bello:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

References

[1] P.K. Malik, R. Sharma, R. Singh, A. Gehlot, S.C. Satapathy, W.S. Alnumay, D. Pelusi, U. Ghosh, J. Nayak, Industrial internet of things and its applications in industry 4.0: State of the art, Comput. Commun. 166 (2021) 125–139.

[2] G. Iannizzotto, L. Lo Bello, G. Patti, Personal Protection Equipment detection system for embedded devices based on DNN and Fuzzy Logic, Expert Syst. Appl. 184 (2021) 115447, http://dx.doi.org/10.1016/j.eswa.2021.115447.

[3] A. Bonci, S. Longhi, G. Nabissi, Fault Diagnosis in a belt-drive system under non-stationary conditions. An industrial case study, in: 2021 IEEE Workshop on Electrical Machines Design, Control and Diagnosis, WEMDCD, IEEE, 2021, pp. 260–265, http://dx.doi.org/10.1109/WEMDCD51469.2021.9425680.

[4] L. Leonardi, L. Lo Bello, F. Battaglia, G. Patti, Comparative assessment of the LoRaWAN medium access control protocols for IoT: Does listen before talk perform better than ALOHA? Electronics 9 (4) (2020) http://dx.doi.org/10.3390/electronics9040553.

[5] M.B. Mollah, S. Zeadally, M.A.K. Azad, Emerging wireless technologies for Internet of Things applications: Opportunities and challenges, in: Encyclopedia of Wireless Networks, Springer, 2020, pp. 390–400.

[6] G. Iannizzotto, M. Milici, A. Nucita, L. Lo Bello, A perspective on passive human sensing with bluetooth, Sensors 22 (9) (2022) http://dx.doi.org/10.3390/s22093523.

[7] E. Oztemel, S. Gursev, Literature review of Industry 4.0 and related technologies, J. Intell. Manuf. 31 (1) (2020) 127–182.

[8] L. Leonardi, L. Lo Bello, G. Patti, LoRa support for long-range real-time inter-cluster communications over Bluetooth Low Energy industrial networks, Comput. Commun. 192 (2022) 57–65, http://dx.doi.org/10.1016/j.comcom.2022.05.026.

[9] P.K. Donta, S.N. Srirama, T. Amgoth, C.S.R. Annavarapu, Survey on recent advances in IoT application layer protocols and machine learning scope for research directions, Digit. Commun. Netw. 8 (5) (2022) 727–744.

[10] B. Konieczek, M. Rethfeldt, F. Golatowski, D. Timmermann, A distributed time server for the real-time extension of CoAP, in: 2016 IEEE 19th International Symposium on Real-Time Distributed Computing, ISORC, IEEE, 2016, pp. 84–91.

[11] E. Longo, A.E. Redondi, Design and implementation of an advanced MQTT broker for distributed pub/sub scenarios, Comput. Netw. 224 (2023) 109601.

[12] P.K. Donta, S.N. Srirama, T. Amgoth, C.S.R. Annavarapu, iCoCoA: intelligent congestion control algorithm for CoAP using deep reinforcement learning, J. Ambient Intell. Humaniz. Comput. 14 (3) (2023) 2951–2966.

[13] OASIS standard, MQTT version 5.0, 2019, Retrieved June 22, 2020.

[14] P. Colombo, E. Ferrari, E.D. Tümer, Regulating data sharing across MQTT environments, J. Netw. Comput. Appl. 174 (2021) 102907, http://dx.doi.org/10.1016/j.jnca.2020.102907.

[15] P. Ferrari, E. Sisinni, D. Brandão, M. Rocha, Evaluation of communication latency in industrial IoT applications, in: 2017 IEEE International Workshop on Measurement and Networking, M&N, 2017, pp. 1–6, http://dx.doi.org/10.1109/IWMN.2017.8078359.

[16] M. Amoretti, R. Pecori, Y. Protskaya, L. Veltri, F. Zanichelli, A scalable and secure publish/subscribe-based framework for industrial IoT, IEEE Trans. Ind. Inform. 17 (6) (2021) 3815–3825, http://dx.doi.org/10.1109/TII.2020.3017227.

[17] R. Atmoko, R. Riantini, M. Hasin, IoT real time data acquisition using MQTT protocol, J. Phys. Conf. Ser. 853 (1) (2017) 012003, http://dx.doi.org/10.1088/1742-6596/853/1/012003.

[18] G. Patti, L. Leonardi, L. Lo Bello, A novel MAC protocol for low datarate cooperative mobile robot teams, Electronics 9 (2) (2020) http://dx.doi.org/10.3390/electronics9020235.

[19] M.O. Ojo, S. Giordano, D. Adami, M. Pagano, Throughput maximizing and fair scheduling algorithms in industrial Internet of Things networks, IEEE Trans. Ind. Inform. 15 (6) (2018) 3400–3410.

[20] E.J. Sacoto Cabrera, S. Palaguachi, G.A. León-Paredes, P.L. Gallegos-Segovia, O.G. Bravo-Quezada, Industrial communication based on mqtt and modbus communication applied in a meteorological network, in: The International Conference on Advances in Emerging Trends and Technologies, Springer, 2020, pp. 29–41.

[21] H. Shi, L. Niu, J. Sun, Construction of industrial internet of things based on MQTT and OPC UA protocols, in: 2020 IEEE International Conference on Artificial Intelligence and Computer Applications, ICAICA, 2020, pp. 1263–1267, http://dx.doi.org/10.1109/ICAICA50127.2020.9182598.

[22] Z. Wang, D. Han, Y. Gong, Y. Zhao, Multi-protocol Integration and Intercommunication Technology based on OPC UA and MQTT, in: Journal of Physics: Conference Series, Vol. 2173, IOP Publishing, 2022, 012070.

[23] L. Leonardi, L. Lo Bello, G. Patti, MRT-LoRa: A multi-hop real-time communication protocol for industrial IoT applications over LoRa networks, Comput. Commun. 199 (2023) 72–86, http://dx.doi.org/10.1016/j.comcom.2022.12.013.

[24] I. Behnke, H. Austad, Real-time performance of industrial IoT communication technologies: A review, IEEE Internet Things J. (2023).

[25] F. Battaglia, M. Collotta, L. Leonardi, L. Lo Bello, G. Patti, Novel extensions to enhance scalability and reliability of the IEEE 802.15. 4-DSME Protocol, Electronics 9 (1) (2020) 126, http://dx.doi.org/10.3390/electronics9010126.

[26] M. Alam, N. Ahmed, R. Matam, M. Mukherjee, F.A. Barbhuiya, SDN-based reconfigurable edge network architecture for industrial internet of things, IEEE Internet Things J. 10 (18) (2023) 16494–16503, http://dx.doi.org/10.1109/JIOT.2023.3268375.

[27] L. Leonardi, L. Lo Bello, G. Patti, Bandwidth partitioning for Time-Sensitive Networking flows in automotive communications, IEEE Commun. Lett. 25 (10) (2021) 3258–3261, http://dx.doi.org/10.1109/LCOMM.2021.3103004.

[28] O. Dieng, R. Santos, D. Mosse, Extending LoRaWAN with real-time scheduling, in: International Conference on Ubiquitous Computing and Ambient Intelligence, Springer, 2023, pp. 114–126.

[29] G. Patti, G. Alderisi, L. Lo Bello, SchedWiFi: An innovative approach to support scheduled traffic in ad-hoc industrial IEEE 802.11 networks, in: 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation, ETFA, IEEE, 2015, pp. 1–9, http://dx.doi.org/10.1109/ETFA.2015.7301460.

[30] A. Stanford-Clark, H.L. Truong, MQTT For Sensor Networks (MQTT-SN) Protocol Specification, International Business Machines (IBM), 2013, pp. 1–28, Corporation version 1 (2).

[31] P.K. Donta, S. Dustdar, Towards intelligent data protocols for the edge, in: 2023 IEEE International Conference on Edge Computing and Communications, EDGE, IEEE, 2023, pp. 372–380.

[32] Y.-S. Kim, H.-H. Lee, J.-H. Kwon, Y.S. Kim, E.-J. Kim, Message queue telemetry transport broker with priority support for emergency events in Internet of Things, Sensors Mater. 30 (8) (2018) 1715–1721.

[33] C.S. Kim, A study on method for message processing by priority in MQTT broker, JKIICE-J. Korea Inst. Inf. Commun. Eng. (2017).

[34] T. Tachibana, T. Furuichi, H. Mineno, Implementing and evaluating priority control mechanism for heterogeneous remote monitoring IoT system, in: Adjunct Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing Networking and Services, 2016, pp. 239–244, http://dx.doi.org/10.1145/3004010.3004040.

[35] E. Shahri, P. Pedreiras, L. Almeida, Extending MQTT with real-time communication services based on SDN, Sensors 22 (9) (2022) 3162, http://dx.doi.org/10.3390/s22093162.

[36] E. Shahri, P. Pedreiras, L. Almeida, Response time analysis for rt-mqtt protocol grounded on sdn, in: Fourth Workshop on Next Generation Real-Time Embedded Systems, NG-RES 2023, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[37] E. Shahri, P. Pedreiras, L. Almeida, End-to-end response time analysis for RT-MQTT: Trajectory approach versus holistic approach, in: 2023 IEEE 19th International Conference on Factory Communication Systems, WFCS, 2023, pp. 1–8, http://dx.doi.org/10.1109/WFCS57264.2023.10144242.

[38] T. Sylla, R. Singh, L. Mendiboure, M.S. Berger, M. Berbineau, L. Dittmann, SoD-MQTT: A SDN-based real-time distributed MQTT broker, in: 2023 19th International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob, 2023, pp. 92–97, http://dx.doi.org/10.1109/WiMob58348.2023.10187779.

[39] E. Shahri, P. Pedreiras, L. Almeida, J. Sousa, Scalable SDN-based MQTT real-time communications for edge networks, in: 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation, ETFA, 2023, pp. 1–8, http://dx.doi.org/10.1109/ETFA54631.2023.10275671.

[40] F. Fontes, B. Rocha, A. Mota, P. Pedreiras, V. Silva, Extending MQTT-SN with real-time communication services, in: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Vol. 1, 2020, pp. 1–4, http://dx.doi.org/10.1109/ETFA46521.2020.9212147.

[41] MQTT version 5.0, 2023, https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf, Accessed: 2023-10-12.

[42] G. Alderisi, G. Iannizzotto, L. Lo Bello, Towards IEEE 802.1 Ethernet AVB for advanced driver assistance systems: A preliminary assessment, in: Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation, ETFA 2012, IEEE, 2012, pp. 1–4, http://dx.doi.org/10.1109/ETFA.2012.6489775.

[43] C. Xia, X. Jin, C. Xu, Y. Wang, P. Zeng, Real-time scheduling under heterogeneous routing for industrial Internet of Things, Comput. Electr. Eng. 86 (2020) 106740.

[44] G. Patti, L. Lo Bello, L. Leonardi, Deadline-aware online scheduling of TSN flows for automotive applications, IEEE Trans. Ind. Inform. 19 (4) (2022) 5774–5784, http://dx.doi.org/10.1109/TII.2022.3184069.

[45] D.D. Brandt, Industrial Automation Bit Error Rate, Tech. Rep., IEEE 802.3 10Mbps Single-Pair Ethernet Study Group – Ad Hoc - Interim Meeting, TX, USA, 2016, URL https://grouper.ieee.org/groups/802/3/10SPE/public/adhoc/brandt_082216_10SPE_01_adhoc.pdf.

[46] R.A. Light, Mosquitto: server and client implementation of the MQTT protocol, J. Open Source Softw. 2 (13) (2017) 265.

[47] A. Saleh, S. Pirttikangas, L. Lovén, Pub/sub message brokers for GenAI, 2023, arXiv preprint arXiv:2312.14647.