CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Enhancing MQTT with Real-Time and Reliable Communication Services

Luís Almeida*
Ehsan Shahri
Paulo Pedreiras

*CISTER Research Centre

CISTER-TR-211001

2021/07/21

Enhancing MQTT with Real-Time and Reliable Communication Services

Luís Almeida*, Ehsan Shahri, Paulo Pedreiras

*CISTER Research Centre

Faculdade de Engenharia Universidade do Porto (FEUP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: lda@fe.up.pt, ehsan.shahri@ua.pt

https://www.cister-labs.pt

Abstract

MQTT is an application-layer protocol that eventually became popular in the Internet of Things (IoT) and Industrial IoT (IIoT) thanks to its simplicity and effective publisher-subscriber messaging model that enables its use in embedded resource-constrained devices. However, MQTT features a limited set of Quality-of-Service classes addressing exclusively message delivery, impairing its use in IIoT applications subject to timeliness requirements. This limitation of MQTT has been addressed in the literature, but with focus on the broker real-time operation, only. This paper adds to the state-of-the-art, by proposing a set of extensions to the MQTT protocol grounded on Software-Defined Networking (SDN) that enable, at the network level, attaining real-time communication services. Simulation results validate the benefits of the proposed extensions.

Enhancing MQTT with Real-Time and Reliable Communication Services

Ehsan Shahri
DETI/IT
University of Aveiro
Aveiro, Portugal
ehsan.shahri@ua.pt

Paulo Pedreiras
DETI/IT
University of Aveiro
Aveiro, Portugal
pbrp@ua.pt

Luis Almeida
CISTER-FEUP
University of Porto
Porto, Portugal
lda@fe.up.pt

***Abstract*—MQTT is an application-layer protocol that eventually became popular in the Internet of Things (IoT) and Industrial IoT (IIoT) thanks to its simplicity and effective publisher-subscriber messaging model that enables its use in embedded resource-constrained devices. However, MQTT features a limited set of Quality-of-Service classes addressing exclusively message delivery, impairing its use in IIoT applications subject to timeliness requirements. This limitation of MQTT has been addressed in the literature, but with focus on the broker real-time operation, only. This paper adds to the state-of-the-art, by proposing a set of extensions to the MQTT protocol grounded on Software-Defined Networking (SDN) that enable, at the network level, attaining real-time communication services. Simulation results validate the benefits of the proposed extensions.**

***Index Terms*—MQTT, IIoT, real-time communication, Software-Defined Networking, OpenFlow, Industry 4.0**

I. Introduction

The promise for massive and unprecedented integration of digital devices without requiring explicit human intervention is making the Internet of Things (IoT) increasingly popular. Nowadays, the IoT spans vast application domains, from smart grids [1] to industry automation [2] [3], medical systems [4], wearable devices [5] and agriculture [6], to name just a few. Within the industrial arena, the so-called Industrial IoT (IIoT) is one of the pillars of the ongoing revolution towards massive digitalization, also known as Industry 4.0.

The diversity of IoT application domains inherently brings heterogeneous requirements. While some applications, e.g. remote metering, demand essentially cheap, low-cost, low-footprint and small-size devices, and assurance that data is eventually collected, many IIoT applications add stringent requirements in terms of real-time performance and reliability [7]. As in any distributed system, the ability to satisfy these requirements depends, among other, on a proper support from the communication infrastructure, including protocols, platforms and technologies.

The Message Queuing Telemetry Transport (MQTT) [8] protocol is among the most popular application-layer protocols used in the IoT/IIoT. It is a lightweight protocol designed to allow the interchange of small amounts of data among potentially large networks composed of simple digital devices (e.g. sensors). Its popularity stems from its simplicity, low footprint, scalability and effective publisher-subscriber messaging model, which fits resource-constrained devices.

MQTT is normally used over TCP/IP networks, building on ordered, lossless and bi-directional channels. Regarding Quality-of-Service (QoS), MQTT offers three levels all of which related with delivery, missing real-time attributes entirely, which, as mentioned above, are of utmost importance in many IIoT applications. This limitation has been recognized by the scientific community in Section II. However, the contributions found in the literature have a limited scope, being essentially focused on the broker real-time performance. In this work we take a more comprehensive approach, proposing extensions to MQTT itself to enable the explicit specification of real-time requirements that can be used at all architecture components, including broker software and network. The proposed extensions allow associating commonly used real-time attributes to each topic, such as priority, deadline and periodicity. The extensions were devised to allow a flexible management of the resources, namely by allowing the online definition and modification of the real-time attributes of each topic. Moreover, the extensions were also designed to comply with the MQTT standard messages, thus allowing the co-existence of nodes with standard and enhanced (real-time) stacks.

The remainder of this paper is organized as follows. Section II briefly overviews the related work. Section III provides background on MQTT and SDN. Section IV presents the proposed MQTT real-time extensions. Section V reports simulation results that validate the desired properties. Finally, Section VI presents the main conclusions of this work and points to lines of future work.

II. Related Work

The real-time performance and reliability of MQTT have already been addressed and reported in the literature. For example, Tachibana *et al.* [9] propose a priority control mechanism for heterogeneous remote monitoring IoT systems. The architecture comprises a Broker and an Application server that interact with IoT devices. The Application server specifies the

This work is funded by the FCT/MCTES through national funds and, when applicable, co-funded by community funds under the projects UIDB/50008/2020-UIDP/50008/2020 and UIDB/04234/2020. It is also supported by Portuguese National Funds through FCT scholarship PD/BD/137388/2018.

application requirements, which are then used by the Broker to control the communication link attributes (sending rate and timing) between devices and application. The mechanism defines three phases, namely registration, QoS negotiation and data exchange. Experimental results indicate a significant latency reduction and successful sending ratio increase, both proportional to message priorities. These benefits do not always materialize, with cases in which the proposed mechanism performs worse than standard MQTT. This is caused by the (uncontrolled) mobile network infrastructure used, which exhibits bandwidth variations that the priority-based mechanism is unable to follow effectively. This observation clearly supports the argument that network-level control is a key factor for attaining real-time performance in such systems.

Kim *et al.* [10] proposed p-MQTT for IoT applications with timeliness and reliability requirements, based on prioritizing emergency events in the broker. The p-MQTT protocol has three main components: virtual queues, classification and priority control. The classification segregates the published messages according to the type field, storing them into dedicated virtual queues, namely Normal, Critical and Urgent. The priority control assures that virtual queues are processed according to their priority. The authors show the protocol ability to differentiate emergency events, reducing the latency with respect to standard MQTT.

Kim *et al.* [11] take an approach that has some similarities with the previous one, adding a priority flag field to the fixed part of the MQTT header to signal the message priority to the broker. Upon message arrivals, the broker tests the priority field, processing the incoming messages according to the corresponding priority. The priority flag field uses two bits, thus supporting four priority levels. The results show traffic segregation and handling according to priority levels, with an end-to-end delay reduction proportional to the priority.

MQTT-SN is a standardized MQTT variant designed to reduce the message payload size and remove the need for permanent connections, aiming at low-cost, battery-operated devices with limited processing, storage and communication resources. Fontes *et al* [12] extended MQTT-SN with a set of additional messages that allow associating real-time requirements to topics. The proposed extensions aim at wireless deployments supporting prioritized frame transmissions, using a prioritization scheme based on the Enhanced Distributed Channel Access (EDCA) defined in IEEE 802.11e-2005. A resource manager collects the real-time requirements and associates suitable Access Categories to messages, prioritizing their transmission. The paper reports significant improvements both in timeliness and number of retransmissions.

Other papers in the literature mention the use of MQTT in real-time applications, e.g. [13] [14] [15]. However, they focus on software aspects not supporting the specification of real-time communication requirements, which is a severe limitation for our purposes.

From this brief survey we can see that existing approaches to improve MQTT real-time performance address software aspects, only, namely broker and bridge implementations. However, requirements such as latency, jitter and delivery ratio depend on the network, too, as shown in [9]. Network control was already used in [12], but focusing on wireless sensor networks. To the best of the authors' knowledge, our proposal is the first to feature MQTT with real-time capabilities as needed for industrial applications based on wired infrastructures.

III. SDN AND MQTT BACKGROUND

SDN is a network paradigm that decouples network control from forwarding functions [16]. In SDN, switches are simple packet forwarding devices, forming the so-called data plane, while logically-centralized controllers, responsible for network management and configuration, compose the control plane. By decoupling the control and data planes, SDN networks become flexible and simple to manage, configure and operate. The OpenFlow protocol [17], standardized by the Open Networking Foundation (ONF), defines the interface between Controllers (control plane) and Switches (data plane), allowing an OpenFlow Controller to dynamically configure a set of OpenFlow switches on how to handle data packets. Each OpenFlow switch has one or more flow tables, each one comprising a set of prioritized flow entries that enable to filter packets, carry out instructions on them, keep statistical information, etc. When an OpenFlow switch receives a packet at an ingress port, it is submitted to the flow tables for processing. If the received packet matches the filter of a given flow entry, the appropriate instructions (such as send-out-port, modify-field, etc.) are performed. Packets that do not match any filter are sent to a group table or dropped. Group tables contain a subset of instructions similar to those of flow tables, with similar function. Openflow controllers are responsible for interacting with the switches, configuring the corresponding flow tables.

MQTT [18] is an application-layer protocol designed for applications where computation resources at end-nodes and bandwidth are constrained. This protocol employs a lightweight publisher-subscriber protocol, comprising a broker and clients. Clients connect to the broker and then subscribe to, or publish data on, specific topics. All data exchanges are mediated by the broker, i.e., when data sources publish data, these data are stored by the broker. In turn, the broker keeps a list of clients that are subscribers of each topic and, when it receives a new publication for a given topic, it forwards the received data to the corresponding client set. Therefore, data generation and consumption are decoupled, both in space and in time. This functionality, combined with the protocol simplicity and low footprint, are among the main reasons behind the MQTT popularity.

MQTT has three QoS levels, all associated with message delivery guarantees: QoS 0 (Once), QoS 1 (At Least Once) and QoS 2 (Only Once). QoS 0 does not have any acknowledgement mechanisms, therefore delivery is not guaranteed. On the other hand, QoS 1 and 2 have acknowledgement mechanisms, therefore message delivery is guaranteed. QoS 2 employs a four-part handshake to eliminate eventual duplicates.

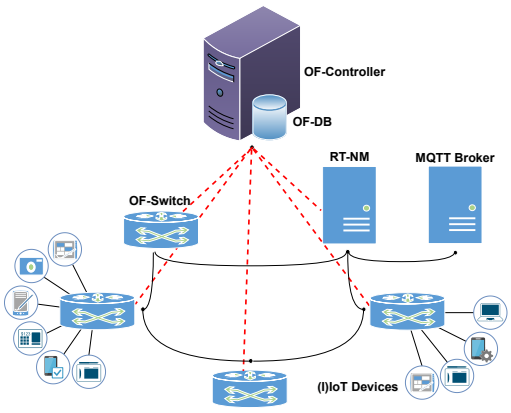

Fig. 1. The architecture of proposed RT-MQTT system.

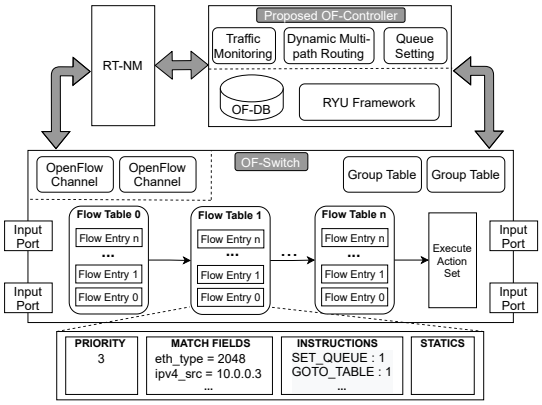

Fig. 2. The structure of proposed OF-Controller.

MQTT v5.0 [18] is the latest MQTT version and adds new features to the protocol. One of these features, particularly relevant for this work, is the extensibility mechanism granted by the so-called user properties. The user properties consist of an array of UTF-8 key/value pairs that allow adding user-defined information to MQTT messages. These user properties are present in various message types and are conveyed in the corresponding message property field. Hence, metadata associated to an unlimited number of user properties can be exchanged between publisher, broker and subscriber. This is the mechanism used in this work to allow nodes to specify the topic's real-time requirements.

IV. MQTT Real-Time Extensions

This work aims at extending the MQTT protocol to allow associating explicit real-time requirements to topics and end-nodes. The additional information is conveyed by standard MQTT messages via the user properties field and can then be used both to improve the timeliness of the execution of software components at the broker and to manage the network, creating deterministic communication channels matching the real-time requirements of the associated topics. The focus of this paper is on the network layer, since for the software components the literature already reports approaches that can be readily adapted to this framework (e.g. [11] and [10]).

A. System Architecture

The proposed Real-Time MQTT (RT-MQTT) architecture is shown in Fig. 1. This architecture is based on OpenFlow and MQTT components, comprising a centralized OpenFlow controller (OF-Controller), OpenFlow switches (OF-Switches), an MQTT broker, (I)IoT devices as the MQTT clients, and a Real-Time Network Manager (RT-NM). The centralized OF-Controller, whose structure is shown in Fig. 2, is based on the RYU [19] framework, integrating a traffic monitoring module, a dynamic multi-path routing module, a queue setting module and an OF-DataBase (OF-DB). The RT-NM, logically placed between MQTT clients and the broker, is a software layer that can be executed on the same PC as the broker and intercepts all the MQTT messages with the objective of extracting eventual real-time requirements data. These requirements are then processed and communicated to OF-Controller in order to manage the flow tables of OF-Switches and so create the real-time channels.

B. Algorithm Description

According to the OpenFlow protocol, the OF-Controller is connected to all OF-Switches, having a global view of the network. In particular, the OF-Controller can collect topological information, being aware of all OF-Switches and links between them. Moreover, OF-Switches are configured to send to the OF-Controller packets that do not match any installed flow entry (PacketIn message). The controller then checks the header fields of those packets, such as source and destination IP address, as well as the switch port at which they were received, enabling the creation of a stack of all available paths from source to destination.

To create the paths' stack it was adopted the DFS [20] algorithm due to its low memory footprint and ability to provide low jitter and round-trip time (RTT) for the optimized path. DFS is a recursive algorithm that starts at the root node of the graph and follows all possible paths till the end. DFS stores all the paths in a stack in descending order relative to the minimum distance of nodes, so it can find the shortest path, which is placed in the last position. As this algorithm does not specify the path weight, we adopted Eq. 1, based on [21], to calculate the minimum distance of a path using the Open Shortest Path First (OSPF) [22] technique. OSPF was selected because it is a widely adopted and mature protocol, commonly used in Interior Gateway Protocols and large enterprise networks, providing load balancing with equal-cost routes for the same destination, without limitation on the hop count and providing fast convergence.

$$0 \leq bw(p) < 10$$
$$bw(p) = (1 - \frac{pw(p)}{\sum_{i<n}^{i=0} pw(i)}) \times 10 \quad (1)$$

In Eq. 1, for a path `p`, `bw` is the bucket weight, `pw` is the path weight, and `n` is the total number of available paths. For

TABLE I
SHORTEST PATH SELECTION WITH MINIMUM DISTANCE

Source host (h1) / Destination host (h6)		
All Paths	**pw**	**bw**
$p1 = [2, 1]$	$pw1 = (s2 - s1) = 1$	$bw1 = 6.6$
$p2 = [2, 3, 1]$	$pw2 = (s2 - s3) + (s3 - s1) = 2$	$bw1 = 3.3$

example, consider in Fig. 5 that host `h1` wants to communicate with host `h6`. The operation of shortest path selection among multiple paths for this example is shown in Table I, where it is possible to see that the shorter path (lower `pw`) is assigned a higher bucket weight.

After finding the shortest path, the controller configures the flow tables of all OF-Switches that are part of the path accordingly, so that the following packets of the same stream are handled accordingly by the data plane. Once paths are set, MQTT packet exchanges between clients and broker are enabled altogether with other non-MQTT traffic. However, time-sensitive flows are not discriminated, thus their timeliness depends on the overall traffic load. To enable real-time guarantees, client nodes have to specify the corresponding real-time requirements via the MQTT user properties field featured by MQTT V5.0. When needed, these messages are propagated by the MQTT broker to the destination clients. From the publisher side, real-time requirements can then be (re)defined at any time. For example, they can be set initially, during the connection phase (CONNECT message) and/or updated later on, when a client publishes data (PUBLISH message). On the other hand, subscribers can specify real-time requirements also when connecting or when subscribing to a topic (SUBSCRIBE message).

The RT-NM intercepts all messages directed to the MQTT broker, thus being able to decode and assemble the requirements of time-sensitive traffic, as illustrated in Fig. 3. When the RT-NM receives an MQTT client message it inspects its content to determine if it carries a real-time reservation request. If it does, the relevant information (e.g. deadline, priority, minimum and maximum bandwidth) are extracted and registered in the OF-DB, to create/update the flow real-time parameters. These requirements are then communicated to the OF-Controller for updating the OF-Switches flow tables. As an example, the processing of a PUBLISH message is sketched in Fig. 4.

V. SIMULATION

In this section we present a set of simulation results that validate the approach and show its effectiveness in enforcing the segregation and prioritization of MQTT time-sensitive traffic.

A. Simulation Setup

The simulation is based on Mininet version 2.3.0d6[1], an instant virtual network. MQTT clients and Broker are implemented using Eclipse Mosquitto [23] (v2.0.10) and the

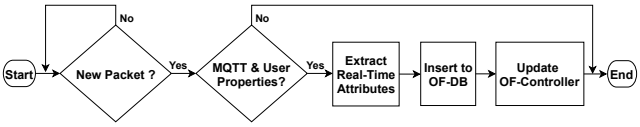

Fig. 3. RT-NM operation.

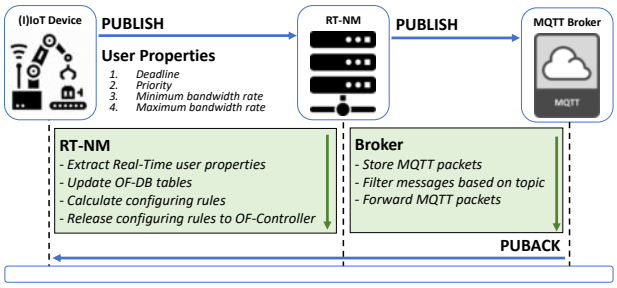

Fig. 4. Real-time attributes conveyed in a PUBLISH message.

Eclipse Paho MQTT library. The OF-Controller is based on the Ryu framework and manages the network flow entries as mentioned in Section IV. All simulations were performed on a laptop computer equipped with a 4.9 GHz Intel Core i7 processor and 16 GB of RAM. Fig. 5 shows the simulated network topology, where `s1`, `s2` and `s3` are the OF-Switches connected to the OF-Controller `c0`. The MQTT clients `h1` to `h8` play the role of (I)IoT devices. Among these, `h1`, `h2`, and `h3` publish time-sensitive MQTT packets and `h4` and `h5` publish normal (non-real-time) MQTT packets. Each publisher has a specific MQTT topic and the QoS of all MQTT messages is set to 1 (at least once) for fair comparison. It should be remarked that by using QoS 1 we are privileging reliability over timeliness, as a result of the non-deterministic TCP re-transmission mechanism. However we are also showing the prioritization and bandwidth reservations support fault-tolerance mechanism, which are important for many applications. Client `h8` is the network sink node, modeled as an MQTT subscriber that subscribes to all topics. The Mosquitto Broker is executed on `h6` and the `RT-NM` is hosted on `h7`. To emulate the heterogeneous data exchanges that are usually found in real networks, additional applications are installed on the client nodes to transmit different data packets over the network, namely dummy MQTT and TCP packets, and audio/video streams. To this end we use the Distributed Internet Traffic Generator (D-ITG)[2] to send TCP packets from `h3` to `h7`. In turn, `h5` and `h8` are configured as client and server of an audio/video stream using the VLC media player. To vary the load conditions and links saturation, `h1` is set to send MQTT dummy packets, with QoS 0, to the broker, generating a load that ranges from 20 Mbps up to 140 Mbps. The bandwidth of the links is set to 100 Mbps and clients are not synchronized. Finally, time-sensitive publishers publish packets with a nominal period of 20 ms plus a random

[1]http://mininet.org/

[2]http://traffic.comics.unina.it/software/ITG/

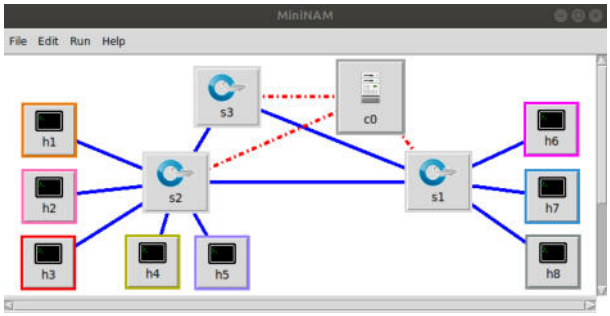

Fig. 5. Simulated network topology in Mininet.

TABLE II
SIMULATION PARAMETERS

Parameters	Value
QoS level	1
Keep alive	60 Second
MQTT packet size	120 Bytes
Traffic monitoring frequency	0.2 Hz
Time-sensitive publishing frequency	50 Hz
Maximum transportation MQTT packet	100 Packets

TABLE III
QUEUE SPECIFICATIONS

Scenario	Queue Number	Minimum Rate	Maximum Rate
First	Queue 0	10 bps	97 Mbps
	Queue 1	10 bps	3 Mbps
Second	Queue 0	10 bps	97 Mbps
	Queue 1	10 bps	1 Mbps
	Queue 2	10 bps	1 Mbps
	Queue 3	10 bps	1 Mbps

offset uniformly picked in the interval $[0, 0.4]$ ms to generate variable interference patterns during the experiments. Table. II summarizes the most relevant simulation parameters.

We use more than one queue for all data packets, defined according to user properties sent by time-sensitive publishers (`h1`, `h2`, `h3`) with appropriate bandwidth reserved by the RT-NM for each queue. Queues labeled with higher numbers (e.g. Queues $1, 2, 3$) have higher priority and are dedicated to the time-sensitive topics, while the remaining traffic is directed to Queue 0. Table. III summarizes the queues specifications.

B. Simulation Results

The simulation results that follow show packet transmission times, defined as the time that each packet takes to travel from the publisher to the subscriber through the broker (see Eq. 2, where T_p and T_r are the publishing and receiving absolute times, respectively).

$$L = (T_p - T_r) \tag{2}$$

We evaluate the proposed system in two scenarios, measuring the transmission latency of time-sensitive and normal MQTT publications. The first scenario evaluates the capability of the system to segregate time-sensitive traffic from the remaining one, thus all time-sensitive topics share the same priority level and a common reserved bandwidth. In the second scenario, we use three different priorities, one assigned to each time-sensitive stream, to assess the effectiveness of time-sensitive traffic prioritization. Note that a higher priority number corresponds to a higher priority. Fig. 6 shows the worst-case transmission latency observed for the baseline experiment without the real-time extensions. This figure clearly indicates the latency of all MQTT traffic (time-sensitive and non-time-sensitive) increases with the bandwidth utilization. These results illustrate the negative impact of high bandwidth utilization in traffic latency experienced in conventional networks.

Fig. 7 shows the worst-case transmission latency observed for the first scenario. It can be seen that the latency of time-sensitive MQTT traffic, associated with the three continuous lines, remains essentially constant, while the latency of the remaining MQTT traffic, represented by dashed lines, increases when the bandwidth utilization grows. These results show that the system can effectively segregate time-sensitive from non-time-sensitive traffic. Note that even when the system is overloaded there are no packet losses affecting the relevant MQTT streams (i.e., all MQTT topics except the dummy ones, that are sent only to generate load). This phenomenon results from the fact that these MQTT messages are transmitted with QoS level 1 (at last once), therefore TCP carries out retransmissions when necessary. Thus, in this scenario, packet losses affect only the background traffic, namely the VLC audio/video stream transmission, which uses UDP at the transport layer and sees a quality degradation, and the MQTT dummy packets sent by `h1`, which have QoS 0.

Fig. 8 illustrates the transmission latency observed in the second scenario, which aims at verifying the effect of time-sensitive flow prioritization. `h3` is assigned with the highest priority, showing the lowest transmission latency, while the highest transmission latency among the time-sensitive publishers belongs to `h1`, that has the lowest priority. As before, it is observed the transmission latency of these time-sensitive publishers is not affected by the overall bandwidth utilization, because the corresponding queues have sufficient reserved bandwidth and are assigned with higher priority than the background traffic. On the other hand, the network load affects the latency of the normal (non-real-time) MQTT publishers `h4` and `h5`, as observed in the previous experiment.

VI. CONCLUSION AND FUTURE WORK

Despite popular in (I)IoT applications, the QoS support of MQTT is rather limited. Particularly, real-time services are absent, impairing its use in applications that have timeliness requirements. The literature reports a few contributions in this area, most of them focused on the broker architecture. This work follows a different and more comprehensive approach, proposing a set of extensions designed to allow the specification and support of real-time services at the network level, enforced using SDN. The proposed extensions take advantage of the User Properties, available in MQTTv5.0

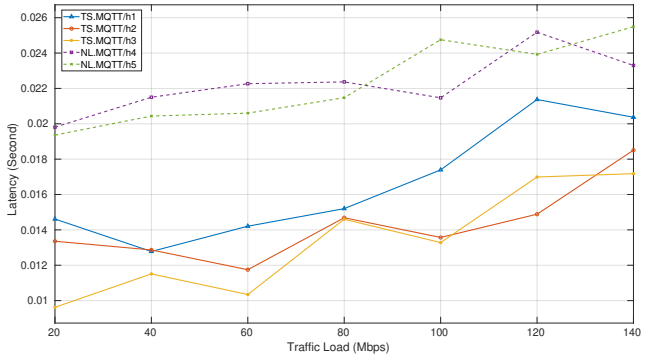

Fig. 6. Worst-case latency comparison: MQTT time-sensitive vs normal MQTT traffic without extensions.

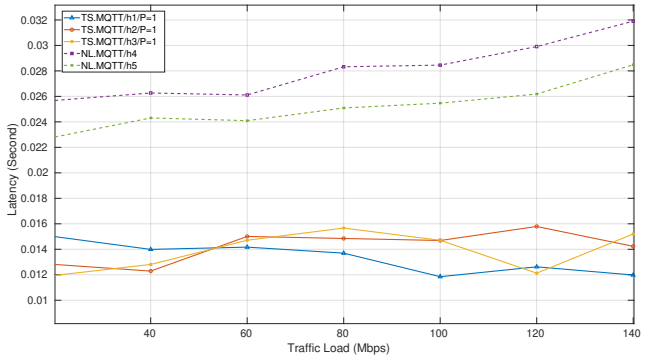

Fig. 7. Worst-case latency comparison: MQTT time-sensitive (same priority) vs normal MQTT traffic with extensions.

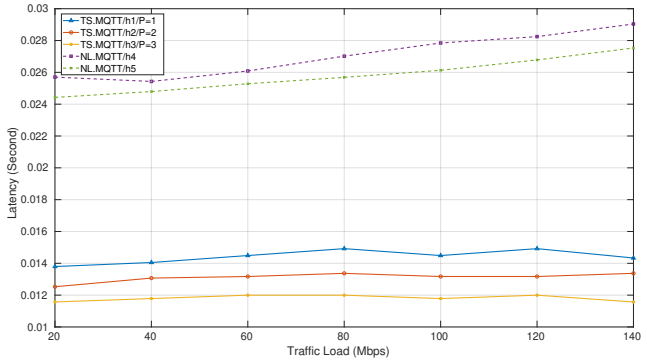

Fig. 8. Worst-case latency comparison: MQTT time-sensitive (prioritized) vs normal MQTT traffic with extensions.

that allow conveying connection-related timeliness attributes while keeping full protocol compatibility. These attributes are then used by SDN to reserve real-time channels accordingly. The properties of the proposed architecture were validated with a set of simulation experiments on Mininet, showing the capacity to segregate time-sensitive MQTT traffic and to enforce arbitrary priorities among this traffic type. Future work will validate the proposed architecture in a physical platform and assess the inherent overheads, comparing with MQTT without real-time support.

References

[1] Q. Wang and Y. G. Wang, "Research on power internet of things architecture for smart grid demand," in *2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2)*, 2018, pp. 1–9.

[2] Y. J. Kwon and D. H. Kim, "Iot-based defect predictive manufacturing systems," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, 2017, pp. 1067–1069.

[3] A. Massaro *et al.*, "Systems for an intelligent application of automated processes in industry: a case study from "pmi iot industry 4.0" project," in *2020 IEEE International Workshop on Metrology for Industry 4.0 IoT*, 2020, pp. 21–26.

[4] S. Siyang *et al.*, "The development of iot-based non-obstructive monitoring system for human's sleep monitoring," in *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, 2019, pp. 1–2.

[5] A. J. Jara, "Wearable internet: Powering personal devices with the internet of things capabilities," in *2014 International Conference on Identification, Information and Knowledge in the Internet of Things*, 2014, pp. 7–7.

[6] Q. F. Hassan, *Internet of Things Applications for Agriculture*, 2018, pp. 507–528.

[7] R. A. Atmoko *et al.*, "IoT real time data acquisition using MQTT protocol," *Journal of Physics: Conference Series*, vol. 853, p. 012003, May 2017. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/853/1/012003

[8] O. Standard, "MQTT Version 5.0."

[9] H. M. Takuma Tachibana, Tetsuo Furuichi, "Implementing and Evaluating Priority Control Mechanism for Heterogeneous Remote Monitoring IoT System," *MOBIQUITOUS '16 Adjunct Proceedings, Hiroshima, Japan*, December,01,2016.

[10] Y.-S. K. et al, "MQTT Broker with Priority Support for Emerg. Events in IoT," *Sensors and Materials*, 2018.

[11] C. O. Seongjin Kim, "A Study on Method for Message Processing by Priority in MQTT Broker," *JKIICE-Journal of the Korea Institute of Information and Communication Engineering*, Jul. 2017.

[12] F. Fontes *et al.*, "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.

[13] A. M. Zambrano V *et al.*, "Sigpro: A real-time progressive notification system using mqtt bridges and topic hierarchy for rapid location of missing persons," *IEEE Access*, vol. 8, pp. 149 190–149 198, 2020.

[14] H. T. Yew *et al.*, "Iot based real-time remote patient monitoring system," in *2020 16th IEEE International Colloquium on Signal Processing Its Applications (CSPA)*, 2020, pp. 176–179.

[15] A. N. Rosli *et al.*, "Implementation of mqtt and lorawan system for real-time environmental monitoring application," in *2020 IEEE 10th Symposium on Computer Applications Industrial Electronics (ISCAIE)*, 2020, pp. 287–291.

[16] W. Xia *et al.*, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[17] "Open networking foundation. openflow switch specification." [Online]. Available: https://opennetworking.org/sdn-resources/customer-case-studies/openflow/

[18] "Mqtt version 5.0." [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[19] "What's ryu." [Online]. Available: https://ryu-sdn.org/

[20] B. Awerbuch, "A new distributed depth-first-search algorithm," *Information Processing Letters*, vol. 20, no. 3, pp. 147–150, 1985.

[21] M. Hua and J. Pei, "Probabilistic path queries in road networks: traffic uncertainty aware path selection," in *Proceedings of the 13th International Conference on Extending Database Technology*, 2010, pp. 347–358.

[22] "Ospf version 2, rfc2328 (http://www.ietf.org/rfc/rfc2328.txt), 1998."

[23] R. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.