IEEE Open Journal of the
Industrial Electronics Society

Received 26 November 2023; revised 10 January 2024 and 11 February 2024; accepted 22 February 2024. Date of publication 5 March 2024; date of current version 18 March 2024. The review of this article was arranged by Associate Editor C.-M. Chituc.

Digital Object Identifier 10.1109/OJIES.2024.3373232

A Scalable Real-Time SDN-Based MQTT Framework for Industrial Applications

E. SHAHRI [1] (Student Member, IEEE), P. PEDREIRAS [1], AND L. ALMEIDA [2] (Senior Member, IEEE)

[1]DETI/IT, University of Aveiro, 3810-193 Aveiro, Portugal
[2]CISTER-FEUP, University of Porto, 4099-002 Porto, Portugal

CORRESPONDING AUTHOR: E. SHAHRI (e-mail: ehsan.shahri@ua.pt)

This work was supported in part by the Portuguese national funds through FCT/MCTES (PIDDAC), in part by the European Community funds under Grant IT-UIDB/50008/2020-UIDP/50008/2020 and Grant CISTER-UIDB/04234/2020, and in part by the FCT scholarship under Grant PD/BD/137388/2018.

ABSTRACT The increasing prominence of concepts such as Smart Production and Industrial Internet of Things (IIoT) within the context of Industry 4.0 has introduced a new set of requirements for the engineering of industrial systems, including support for dynamic environments, timeliness guarantees, support for heterogeneity, interoperability and reliability. These requirements are further exacerbated at the network level by the notable rise in the number and variety of devices involved. To stay competitive in this ever-changing industrial landscape while boosting productivity, it is vital to meet those requirements, combining established protocols with emerging technologies. Software-Defined Networking (SDN) is the forefront traffic management paradigm that offers flexibility for complex industrial networks, enabling efficient resource allocation and dynamic reconfiguration. Message Queuing Telemetry Transport (MQTT) is a low-overhead protocol of the application layer that is gaining popularity in the scope of the IoT and IIoT. However, its Quality-of-Service (QoS) policies do not support timeliness requirements. This article presents a framework that seamlessly integrates SDN and MQTT, enhancing network management flexibility while satisfying real-time requirements found in industrial environments. It leverages the User Properties of MQTTv5 to allow specifying real-time requirements. MQTT traffic is intercepted by a Network Manager that extracts real-time information and instructs an SDN controller to deploy corresponding network reservations. MQTT traffic across multiple edge networks is propagated by selected brokers using multicasting. Extensive experiments validate the proposed approach, demonstrating its superiority over MQTT and Direct Multicast-MQTT (DM-MQTT) DM-MQTT in latency reduction. A response time analysis, validated experimentally, emphasizes robust performance across metrics.

INDEX TERMS Edge computing, industrial Internet of Things (IIoT), Internet-of-Things (IoT), message queuing telemetry transport (MQTT), real-time (RT) communication, software-defined networking (SDN).

I. INTRODUCTION

In recent years, the rapid progress in technology has ushered in a transformative era known as Industry 4.0 [1], which seamlessly integrates physical entities, human interaction, and intelligent machines into a sophisticated information network. This evolution represents a shift toward a digitally driven industrial landscape, contributing to the increasing popularity of the Internet of Things (IoT). The widespread adoption of the IoT has permeated a wide array of application domains, including smart grids [2], medical systems [3], wearable devices [4], agriculture [5], industrial automation [6], [7], among others.

The diversity in these applications inevitably introduces an increased complexity in requirements and demands for interoperability. Notably, the industrial Internet of Things (IIoT) domain imposes stringent prerequisites for real-time (RT) performance and reliability [8], necessitating a robust computing and communication infrastructure. However, these stringent requirements often clash with the capabilities of existing technologies and protocols. On the one hand, general-purpose data

networks, such as Ethernet [9], and comprehensive frameworks for network management, such as software-defined networking (SDN) [10], prioritize throughput and resource utilization optimization, often lacking support for deterministic RT services. On the other hand, industrial networks have traditionally prioritized determinism and timeliness attributes, such as low latency and jitter, at the expense of throughput and flexibility.

Recently, the authors addressed this gap showing the effectiveness of integrating SDN network management in tandem with communication protocols, such as message queuing telemetry transport (MQTT) [11], facilitating deterministic RT data transmissions among heterogeneous IIoT end-nodes while preserving flexibility and reliability. This has been achieved through three main contributions as follows.

1) Developing an SDN-based resource management framework along with proposing a set of RT extensions to MQTT, referred to as RT-MQTT [12], [13]. This architecture empowers applications to specify RT MQTT requirements, which are then translated into SDN/OpenFlow (OF)-enforced network reservations to establish RT channels.
2) Evaluating RT-MQTT system predictability and schedulability through response time analysis [14], [15].
3) Enhancing the scalability of RT-MQTT with a new architecture, known as multicast real-time MQTT (MRT-MQTT) [16], that leverages multicast-based connectivity across multiple edge networks that follow the RT-MQTT architecture.

This article extends this body of research in several significant ways as follows.

1) First, it revisits and clarifies the entire system formal model.
2) Second, it introduces a response time analysis for the MRT-MQTT architecture, leveraging both the holistic approach (HA) [17] and the trajectory approach (TA) [18] to establish the analyzability of this architecture.
3) Finally, it evaluates the performance of MRT-MQTT through a comprehensive set of case-study experiments that underscore the robustness of this architectural framework.

It should be remarked that MRT-MQTT [16] subsumes RT-MQTT [13]. Particularly, MRT-MQTT reduces to RT-MQTT when applied to a single edge network with a single MQTT broker. This article provides the reference work to this framework, introducing RT-MQTT and MRT-MQTT for completeness and then delving in the specific contributions referred previously, around the analysis of the framework.

Both RT-MQTT and MRT-MQTT are based on commercial off-the-shelf protocols, software, components and services, relying on prescribed extension mechanisms to implement the additional features. This way, RT services can be deployed in existing systems as an add-on, without impacting existing applications and end-nodes, as the additional software modules are transparent to traffic not related with the RT extensions. Furthermore, the use of multicasting in MRT-MQTT adds support to edge-networks, efficiently handling increased data volumes and thus providing horizontal scalability.

The rest of this article is organized as follows. Section II provides background on MQTT, SDN, multicast routing schemes and schedulability analysis methods. Section III offers an overview of the most relevant state-of-the-art research within the scope of this article. Section IV introduces RT-MQTT that adds RT extensions to MQTT, followed by the unveiling of an extended RT MQTT system designed for large networks (MRT-MQTT) in Section V. Section VI presents a set of schedulability analyses, evaluating both the RT-MQTT and MRT-MQTT architectures. A comprehensive assessment of the proposed architectural approaches is presented in Section VII. Finally, Section VIII concludes this article.

II. BACKGROUND

This section briefly introduces the features of MQTT, SDN, data multicasting techniques, and schedulability analysis that are relevant to this article. The section ends with a short discussion on the implications of security aspects.

A. MESSAGE QUEUING TELEMETRY TRANSPORT

MQTT [11] is a popular protocol designed specifically for efficient communication between machines, particularly those with limited resources. Its simplicity, scalability, small footprint, and effective publisher–subscriber model are key reasons for its widespread adoption. MQTT typically operates over TCP/IP networks, which inherently provide reliable, ordered, and bidirectional data transfer. In addition, MQTT offers three quality-of-service (QoS) levels to control message delivery guarantees. QoS 0 (send once) is the most basic level, with messages sent without confirmation or guarantees of delivery, depending only on the underlying TCP protocol for reliability. QoS 1 (deliver at least once) ensures at least one delivery, but duplicates may occur due to retransmissions. Finally, QoS 2 (deliver exactly once) guarantees that each message is received exactly once by its receiver. MQTT latest version, V5.0 [19], introduces *user properties* that are a valuable extension mechanism. This feature allows for the inclusion of user-defined information using UTF-8 key/value pairs within MQTT messages. This extension mechanism is adopted in this work to convey the topics RT requirements set by end-nodes.

B. SOFTWARE-DEFINED NETWORKING

SDN [10] stands as a pivotal technology, instrumental in the efficient management of complex and dynamically evolving networks. The core concept behind SDN is the separation of the data plane, where SDN switches handle packet forwarding, from the control plane, overseen by a logically centralized controller. This logic separation enables the creation of networks that are both flexible and easy to manage. The centralized controller establishes communication with network switches through the southbound interface, thereby

gaining a comprehensive understanding of the network status, including device inventory, device attributes, network topology, and resource utilization.

The OF protocol [20] serves as the primary SDN southbound interface protocol. It enables OF-controllers to dynamically configure and oversee OF-switches, instructing them on how to process incoming data packets. OF-switches, in turn, consist of flow tables housing prioritized flow rules with filters to identify, and actions to process, packets. A successful match triggers the specified action, while unmatched packets follow a default action, including forwarding to the controller, routing to a group table, or dropping. Group tables serve similar functions to flow tables but with a more limited scope of instructions. OF is standardized by the open networking foundation.

C. DATA MULTICAST TECHNIQUES

Data multicast techniques efficiently deliver data from one sender to a group of selected receivers within a network, making it an intermediate approach between unicast (one-to-one) and broadcast (one-to-all) communication. This approach is valuable for the simultaneous distribution of identical data to multiple recipients, being especially beneficial to save network bandwidth.

In this context, protocol independent multicast sparse mode (PIM-SM) [21] is a branch of the PIM multicast routing protocol [22] designed for one-to-many communication scenarios. In this protocol, a single data packet is transmitted from a source and efficiently received by multiple recipients through a shared multicast tree. PIM-SM is particularly well-suited for situations with limited bandwidth, as it optimizes network usage by delivering traffic only to requested receivers. This protocol operates within two distinct domains: the *source* domain and the *receiver* domain. Within these domains, routers take on various roles, including the designated router (DR), rendezvous point (RP), and bootstrap router (BSR). These roles collaborate to facilitate the distribution of multicast traffic to IP multicast receivers. The DR serves as an intermediary, forwarding multicast traffic from unicast sources to the RP, which is responsible for disseminating the requested multicast traffic to the multicast receivers. The BSR plays a pivotal role in informing all routers within the PIM-SM domain about the currently assigned RP for each known multicast group, either through manual configuration or dynamic election via the BSR process.

The multicast source discovery protocol (MSDP) [23] complements PIM-SM by enabling interdomain multicast connectivity. MSDP enables routers across different network domains to share information about active multicast sources. This is achieved through a network of connected MSDP peers, each maintaining a local database of learned sources from other peers. This information is then used to build a comprehensive multicast distribution tree that seamlessly spans across domain boundaries. To ensure loop-free forwarding of multicast packets, MSDP leverages the reverse path forwarding mechanism, which verifies the direction of incoming multicast data by comparing the source IP address with the internal routing table. If the source is reachable through the interface where the traffic arrived, it is forwarded onwards. In addition, MSDP allows for granular control by enabling the use of access control lists to filter unwanted multicast traffic. By combining the strengths of PIM-SM with MSDP, network operators can create efficient and scalable multicast services capable of reaching a large user base across diverse network domains.

D. SCHEDULABILITY ANALYSIS METHODS

Timely and predictable execution of actions are two fundamental requirements of RT systems. Accurate predictions of timing behavior are essential for guaranteeing that actions are successfully completed within their designated deadlines, making schedulability analysis of paramount relevance [24]. Worst-case response-time (WCRT) analysis applied to RT multihop networks can be categorized into three main groups: HA, TA, and network calculus (NC).

The HA [17], which is tailored for distributed systems, focuses on determining the minimum and maximum response times for both the tasks and the messages they generate. It takes a conservative approach, considering the worst-case scenario on every node (even if unlikely to occur), leading to simple implementation and fast execution. However, this can lead to overestimation of actual response times.

The TA [18], on the other hand, offers more accurate results. It calculates the latest possible starting time for a message at its final destination and then retraces its path backwards, identifying factors that impact its latency at each previous node. While more precise, TA is computationally expensive and complex to implement, particularly in dynamic routing environments. Recent advancements have optimized TA by considering the impact of multiple message streams sharing the same link, leading to tighter estimations of end-to-end response times [25], [26].

The NC [27] operates by characterizing network elements and incoming flows using service curves and arrival curves, respectively. With this information, NC facilitates the calculation of the maximum delay each flow may encounter at network elements, the maximum queue sizes, and corresponding departure curves. While NC provides deterministic results, it necessitates determining arrival and service curves, often in the form of conservative bounds that introduce some level of pessimism. Despite the existence of techniques to reduce this pessimism [28], NC is generally more pessimistic than TA and of similar complexity when applied to individual flows, thus we use TA in this work. Moreover, we also use HA since it is response-time oriented similarly to TA and offers a tradeoff between accuracy and simplicity.

E. ESSENTIALS AND ISSUES IN NETWORK SECURITY

The increasing interconnectivity of devices in the realm of industrial automation, spurred by the advent of paradigms, such as IIoT and Industry 4.0, underscores the paramount importance of security in industrial network deployments.

This pervasive connectivity significantly broadens the attack surface, heightening security risks across multiple facets, from data integrity to equipment and process control.

Within factory automation, time-sensitive (TS) processes, such as cell control and synchronization, are commonplace. These processes often demand RT protocols, typically over physical cabled media, especially when stringent timing requirements are in play. However, when extending these RT processes to broader contexts, such as production lines or groups of cells, security considerations become increasingly relevant. The incorporation of security mechanisms, though crucial, can introduce overhead that must be carefully managed to avoid conflicts with the stringent timing requirements aforementioned.

Historically, inner segments of factory networks have been considered relatively safe from physical security threats. Consequently, security mechanisms were often omitted from these segments, with a focus on fortifying security at network boundaries, such as routers or gateways. However, the evolving trend of tightly integrating industrial networks with the broader Internet necessitates the inclusion of security mechanisms, even if this entails some tradeoffs in timeliness and predictability. As a result, application-layer messaging middleware protocols, such as MQTT and AMQP often incorporate security-related protocols, with SSL/TLS being the prevalent standard for secure communication. Nevertheless, RT extensions or adaptations of these messaging systems often introduce an intermediary node, such as a network manager or controller, which requires access to clear-text traffic for control purposes. This introduces a challenge, commonly referred to as the man-in-the-middle scenario, as it is inherently incompatible with SSL/TLS. Numerous mechanisms have been proposed to reconcile this apparent incompatibility, drawing parallels with network monitoring systems on the Internet, which often require clear-text access to payloads, including deep packet inspection beyond packet metadata. Such a class of approaches can be adapted and incorporated into the framework presented in this article, allowing nodes with management functionality to have clear-text access to MQTT traffic. Thus, although security is left out of this article, there are security mechanisms that can be applied to the proposed framework and which we will consider in future work.

III. RELATED WORK

This section reviews the literature on two primary areas as follows. i) Exploring the RT performance enhancement from information technology (IT) and operational technology (OT) convergence in the industrial landscape. ii) Leveraging edge computing to improve RT communication.

A. TECHNOLOGIES FOR IT AND OT CONVERGENCE

In this section, we examine three pivotal domains, namely MQTT RT performance, utilization of SDN to enhance RT services, and enhancement of RT communication through protocol integration. We aim at highlighting the most pertinent contributions and cutting-edge research that have emerged over the years.

1) ON THE TIMELINESS OF MQTT

While MQTT eventually became one of the most widely adopted application-layer protocols for IoT and IIoT, its native QoS mechanisms lack RT guarantees. This limitation has prompted an extensive research, focusing in distinct aspects. For example, broker-centric approaches address the timing behavior within the MQTT broker itself, like Tachibana et al. [29] priority control with registration, prioritized data exchange, and release phases. The process starts with the registration phase, in which end-nodes communicate their QoS requirements to the broker and ends with the release phase, which closes the application connection. While the connection is active, the sending time and rate of end-nodes is controlled by the priority broker. While demonstrating reduced latency and improved delivery success, these solutions remain confined to the broker, not encompassing the entire network. Another class of approaches are message-centric, i.e., introduce priority fields within the MQTT message header, offering varying levels of importance. Kim et al. [30] used a two-bit field for four priority levels (no priority to urgent), while p-MQTT [31] employ a three-component architecture with classification, virtual queues, and control for emergency events. While effective, these solutions primarily rely on software modifications within the MQTT protocol and lack explicit control over network behavior, limiting their RT performance potential. Other works explore tailoring MQTT services for specific applications to achieve bounded latency [32], [33], [34]. However, these approaches primarily focus on software-level adjustments within MQTT itself, lacking direct control over network-related aspects that influence RT performance. Therefore, a key challenge remains in developing solutions that go beyond software-centric approaches and delve into network-level control mechanisms to address RT communication requirements within the MQTT framework for both IoT and IIoT applications.

2) ENHANCING RT SERVICES THROUGH SDN

SDN has emerged as a promising technology for enabling RT communications in the industrial domain, with the scientific literature reporting several contributions that explore its potential in this context.

Egilmez et al. [35] proposed the OpenQoS framework, which leverages an OF controller to manage multimedia delivery with guaranteed end-to-end latency. It differentiates between regular data and multimedia traffic, routing the latter with specified maximum delay constraints. HiQoS (Yan et al.'s work [36]) builds upon the OpenQoS framework by incorporating multipath routing. In addition, it utilizes the queuing mechanisms of SDN switches to offer varying bandwidth guarantees on each chosen path. Tomovic et al. [37] adopted an approach similar to the IntServ model, offering hard QoS guarantees for specific flows. Bandwidth reservation

and admission control mechanisms ensure these guarantees are met. Dwarakanathan et al. and Sharma et al. ([38], [39]) proposed a two-queue QoS framework, segregating high-priority traffic from best-effort traffic. Bandwidth reservation is used for the high-priority queue, ensuring stable performance. Guck et al. and Kumar et al. ([40], [41]) presented SDN-based frameworks that also employ flow segregation and bandwidth reservations to achieve bounded end-to-end delays. The PrioSDN-resource manager resource management framework, by Leonardi et al. [42], combines SDN and network virtualization to manage varying workload and flow priorities. It utilizes admission control and a priority-based runtime bandwidth distribution mechanism to allocate resources efficiently.

This review provides a brief summary of the diverse approaches explored in the literature for utilizing SDN to support RT communication in the industrial domain. Each framework offers unique strengths and addresses specific challenges, showcasing the potential of SDN in this critical area.

3) RT COMMUNICATION WITH PROTOCOL INTEGRATION

Prior research has explored enhancing either the RT performance of MQTT or the capabilities of SDN. However, a crucial gap remains in providing integrated solutions that bridge the gap between application-level RT requirements and effective network reservations. Recent scientific contributions have started to address this challenge, although each offers distinct approaches with specific strengths and limitations.

DM-MQTT (Park et al.'s work [43]) exhibits some overlap with the architecture presented in this article through its integration of MQTT and SDN. However, its primary objective is to minimize bandwidth usage in large-scale IoT deployments by implementing a multicast mechanism that bypasses the MQTT broker. While this offers potential benefits, it does not address essential aspects, such as traffic prioritization, admission control, or RT guarantees. RT-MQTT (see Section IV) takes a different approach, targeting more confined settings, such as factory automation and focusing on providing dynamic and deterministic RT channels with traffic prioritization. By addressing timeliness requirements effectively, it presents a complementary approach to DM-MQTT.

RT extensions for MQTT-SN (Fontes et al.'s work [44]) build upon MQTT-SN, a sensor network-specific version of MQTT, to enable online specification of RT requirements at all levels. This approach offers enhanced flexibility in resource management through the assignment of attributes to topics and network interface configuration based on importance. However, it relies on specific MQTT-SN mechanisms not present in the current MQTT V5.0 standard, limiting its broader applicability.

These diverse contributions collectively highlight the ongoing efforts to bridge the gap between application-level RT needs and network capabilities. While each approach offers unique advantages and addresses specific challenges, further research is necessary to develop comprehensive solutions that are broadly applicable across diverse scenarios and compatible with evolving standards. This continued exploration has the potential to unlock the full potential of RT communication in various domains, including the IoT and industrial automation.

B. ENHANCING TIMELINESS USING EDGE COMPUTING

The increasing demand for RT communication in IoT and IIoT applications has increased the interest in leveraging edge computing. This approach strategically distributes computing resources to the network edge, enabling efficient and timely data processing, particularly for latency-sensitive applications. Two recent advancements, MI-SDN and DM-MQTT, show the advantages of combining MQTT and SDN to enhance RT performance in edge environments.

Tamri et al. [45] proposed MI-SDN, which utilizes a multicast-based SDN solution designed for efficient MQTT data exchange across multiple edge networks. This architecture employs a master–slave broker structure. Slave brokers manage device groups within each edge network, while a central master broker, residing within the SDN controller, aggregates information from them. The master broker then leverages the SDN controller's capabilities to disseminate MQTT data via multicast across edge networks. This approach effectively reduces transmission delays compared to traditional MQTT, making it suitable for RT applications requiring low latency. DM-MQTT, introduced by Park et al. [43] and already mentioned in the previous section, takes a different approach. It integrates MQTT and SDN to minimize latency for TS data transfer between edge networks. Unlike our proposed MRT-MQTT approach (presented in Section V), DM-MQTT utilizes a redundant master broker solely for data collection from edge networks. This information is then used by the SDN controller to determine optimal data paths, resulting in reduced transmission delay and improved bandwidth utilization in IoT deployments. While DM-MQTT introduces a multicast mechanism to bypass brokers within edge networks, its primary focus lies on minimizing latency and bandwidth consumption, without explicitly addressing traffic prioritization or providing RT guarantees.

In summary, both MI-SDN and DM-MQTT showcase the potential of edge computing to enhance RT communication through integrated MQTT and SDN architectures. While each approach offers distinct advantages, MI-SDN emphasizes efficient data dissemination across edge networks, while DM-MQTT prioritizes minimizing latency and bandwidth consumption.

Other edge-relevant technologies have been used in IIoT settings, such as DDS [46] or more recently Zenoh [47]. While DDS has explicit support for a myriad of QoS parameters, including some related to RT, Zenoh offers enhanced timeliness via low overhead. Streaming technologies have also been used, such as IGMP leveraging IP-level multicasting, and more recently, Kafka [48], both supporting multimedia industrial use cases. None of these offer RT guarantees, though.

Moreover, MQTT is incomparably more prevalent in IoT and IIoT settings and for this reason, as explained in the "Introduction" section, it is the focus of this work. Carrying out an extensive comparison among all edge technologies applicable to the industrial domain is beyond the scope of this article and is left for future work.

C. NOVELTY OF THIS ARTICLE

From the above-mentioned discussion, we can conclude that both RT-MQTT and MRT-MQTT frameworks, addressed in this article, advance the state-of-the-art in several aspects, including the following.

1) Providing RT extensions to MQTT.
2) Integrating MQTT with SDN providing dynamic RT communication channels.
3) Offering WCRT analyses.

This article extends previous work in the scope of RT-MQTT and MRT-MQTT frameworks in three main directions. First, it revisits the existing system formal model to make it more concise and amenable to the development of schedulability tests for MRT-MQTT. Then, it introduces two response time analyses, namely HA and TA that are both response-time oriented and present different compromises between complexity and tightness, and shows their adaptation so they can be applied to MRT-MQTT. Finally, this article presents a comprehensive set of experiments that aim at verifying the correctness of the implementation, validating the schedulability tests, and also evaluating the performance gains with respect to DM-MQTT, which is the protocol that more closely approaches MRT-MQTT.

IV. RT-MQTT FRAMEWORK

The aim of RT-MQTT [12], [13] is to provide RT guarantees to TS MQTT flows, while keeping full compliance with the standard. To this end, the RT requirements associated with a particular TS topic are conveyed within the *user properties* of MQTT messages. This information is captured and processed by a network manager module, which subsequently takes appropriate actions to reserve RT communication channels for these flows. In this article, we implement the network manager using SDN/OF technology to establish and enforce network reservations. However, other networking technologies capable of providing bandwidth reservations and traffic prioritization, such as IEEE TSN [49] or HaRTES [50], could potentially be employed for the same purpose.

A. RT-MQTT REFERENCE ARCHITECTURE

Fig. 1 illustrates the architecture of the RT-MQTT system, which consists of the following.

1) *OF-switches:* network devices that route data based on OF rules.
2) *OF-controller:* central entity that manages the OF-switches and configures their flow tables.
3) *IoT devices (MQTT clients):* devices that generate and consume data using the MQTT protocol.

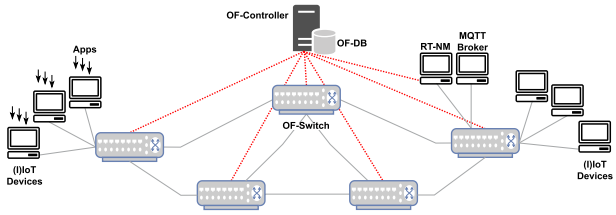

FIGURE 1. High-level RT-MQTT system architecture.

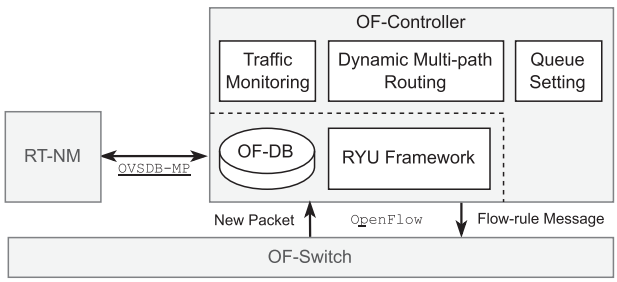

FIGURE 2. Proposed OF-controller architecture.

4) *MQTT broker:* node that acts as a central hub for MQTT communication, relaying messages between clients.
5) *Real-time network manager (RT-NM):* key component that resides between clients and the broker, acting as a gatekeeper for RT requests.

The RT-NM plays a crucial role in enabling RT communication. It intercepts all MQTT messages sent to the broker. If a message contains RT requirements encoded in the *user properties* field, the RT-NM extracts these attributes, processes them, and updates the OpenFlow database (OF-DB) within the OF-controller. Based on this information, the OF-controller dynamically modifies the flow tables of the OF-switches, creating dedicated channels that guarantee specific RT performance criteria for the associated data flow.

The centralized OF-controller, as illustrated in Fig. 2, comprises four main components: a traffic monitoring module, a dynamic multipath routing module, a queue configuration module, and the OF-DB, which is responsible for managing the TS flows' RT requirements. This controller is based on the RYU framework [51], an open-source software platform specifically designed for simplifying the development of SDN applications.

Currently, OF-switches are instances of the *Open vSwitch (OVS)* soft switch. OVS is a virtual switch purpose-built for SDN environments, operating through the open vSwitch database management protocol (OVSDB-MP) [52], as illustrated in Fig. 3. OVS provides adaptable and programmable switching capabilities suitable for virtualized and cloud-based settings. It can be deployed on physical servers and virtualized platforms, including hypervisors and container systems. OVS empowers network administrators to define and manage virtual networks, connect virtual machines and containers, and exercise control over network traffic flow within and between these virtualized environments.

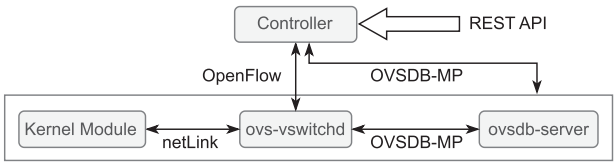

FIGURE 3. OVS architecture [53].

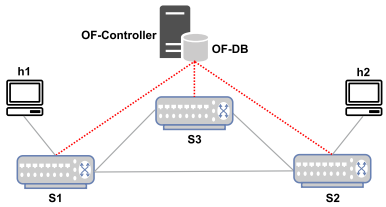

FIGURE 4. Architecture example showcasing the path selection algorithm.

B. RESERVATION MECHANISM

The OF protocol grants the central OF-controller with comprehensive network oversight thanks to its direct connections with all OF-switches. This enables the controller to gather detailed topological information, including the identification of all OF-switches, their connected ports, and the links that interconnect them. In addition, OF-switches are configured to transmit *PacketIn* messages to the controller whenever packets encounter unknown flow rules. These messages contain essential information, such as the source and destination IP addresses, along with the originating switch port, allowing the controller to maintain a comprehensive RT view of the network for subsequent routing decisions.

This work adopts the depth-first search (DFS) algorithm [54] for RT routing. Key factors such as transmission delay and maximum available bandwidth along potential paths between source and destination nodes are combined into a weighted function to determine link costs. The choice of DFS stems from its memory efficiency and its ability to deliver low jitter and round-trip time (RTT) for the optimized path.

As a recursive algorithm, DFS starts at the network's root node and systematically explores all possible paths until completion. These paths are maintained in a stack, organized in descending order based on the minimum distance between nodes. This structure ensures that the shortest path, occupying the final position, is readily identified. Since DFS itself does not incorporate path weight considerations, we leverage (1), derived from [55], to compute the minimum path distance using the open shortest path first (OSPF) technique ([56]). The widespread adoption and maturity of OSPF in interior gateway protocols and large enterprise networks make it a compelling choice. Its additional strengths include load balancing through equal-cost routes, unlimited hop count capabilities, and swift convergence, solidifying its suitability for our RT routing goals

$$0 \leq \mathrm{bw}(\mathcal{L}) < 10$$
$$\mathrm{bw}(\mathcal{L}) = \left(1 - \frac{\mathrm{pw}(\mathcal{L})}{\sum_{i=0}^{i<n} \mathrm{pw}(i)}\right) \times 10. \quad (1)$$

In (1), $\mathcal{L}$ represents a path composed of a set of links crossed by data flows, bw represents the bucket weight and pw the path weight, while n represents the total number of possible paths. For illustration, consider the example depicted in Fig. 4, in which host `h1` wants to publish data to host `h2`.

TABLE 1. Shortest Path Selection Algorithm

Source host (h1) / Destination host (h2)		
All paths	pw	bw
$\mathcal{L}1$	$\mathrm{pw1} = (h1 - s1) + (s1 - s2) + (s2, h2) = 3$	$\mathrm{bw1} = 5.8$
$\mathcal{L}2$	$\mathrm{pw2} = (h1 - s1) + (s1 - s3) + (s3 - s2) + (s2 - h2) = 4$	$\mathrm{bw2} = 4.3$

Each available path ($\mathcal{L}_i$) from `h1` to `h2` has a duplet $l = \langle x, y \rangle$ that represents a link l between node/switch x and node/switch y, as follows.

$\mathcal{L}1 = \{\langle h1, s1\rangle, \langle s1, s2\rangle, \langle s2, h2\rangle\}$.
$\mathcal{L}2 = \{\langle h1, s1\rangle, \langle s1, s3\rangle, \langle s3, s2\rangle, \langle s2, h2\rangle\}$.

Table 1 shows the result of OSPF for this example, where it can be seen that the shorter path (lower pw) is assigned to a higher bucket weight.

Upon identifying the shortest path, the controller dynamically configures the flow tables of all OF-switches constituting the path. This ensures consistent forwarding of subsequent packets belonging to the same data stream within the data plane.

It is typically challenging or even impossible to establish exclusive paths for each TS MQTT flow, therefore these packets will potentially share network links with other TS and nontime-sensitive (NTS) MQTT flows, as well as with other generic data sources. Consequently, without specific mechanisms in place, the potentially unpredictable behavior of NTS flows and other data sources could impact the timeliness of TS MQTT flows. To tackle this issue, MQTT clients involved in TS flows, both publishers and subscribers, convey their RT requirements via the MQTT *user properties* field. RT-MQTT employs a subset of attributes commonly found in RT systems, including

$$F_i^{\mathrm{TS}} = \{C_i, P_i, T_i, D_i, \mathrm{BW}_i\} \quad (2)$$

where:

i : flow index;
C_i : message transmission time (including overheads);
P_i : flow priority;
T_i : period or minimum inter-arrival time between two successive publish messages (by the publisher);
D_i : message deadline, with $D_i \leq T_i$;
BW_i : maximum link bandwidth use.

These RT attributes are placed in the variable header of the MQTT message, following the message structure of MQTT V5.0. The structure of the message is depicted in Fig. 5.

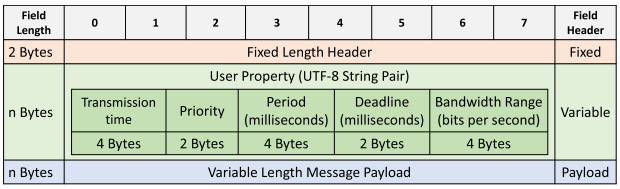

FIGURE 5. **Message structure used in RT-MQTT.**

The RT-NM module functions as a transparent intermediary, intercepting and analyzing all communications exchanged between the MQTT broker and its connected clients. This comprehensive analysis enables the RT-NM to extract and store critical RT requirements associated with each individual message flow, denoted as F_i^{TS}. This extracted information is then compiled and maintained within the system real-time requirements table (SRT), a dedicated component residing within the OF-DB. In addition to the RT requirements, the SRT table also stores both the source (publishers) and destination (subscribers) addresses associated with each message flow. These critical addresses are acquired by the RT-NM module through its interception of the standard MQTT messages exchanged during the initial connection establishment phase, ensuring a complete and accurate representation of the communication network's RT demands. Formally, the SRT is defined as follows:

$$\mathrm{SRT} = \left\{(F_i^{\mathrm{TS}}, \mathrm{SRC}_i, \mathrm{DST}_{i,k}, \mathcal{L}_\rangle, n_i) \;\; i = 1 \ldots N\right\} \quad (3)$$

where:

SRC_i : source node address;
$\mathrm{DST}_{i,k}$: set of k destination node address;
$\mathcal{L}_i$: set of n_i links crossed by flow F_i^{TS};
n_i : number of links that F_i^{TS} crosses, i.e. $n_i = |\mathcal{L}_i|$;
N : number of TS flows.

The information within the SRT, together with the network topology data collected by the OF-controller, is used to configure RT channels for the links along the path, including both the segments from the publisher to the broker, and from the broker to the subscriber(s). This fundamental architecture can be augmented with additional control services if necessary. For instance, adjusting the RT attributes of a specific flow might be restricted to a subset of trusted nodes, with requests from other sources being ignored. Similarly, resource allocation for each flow may be confined within predefined limits. These topics are out of the scope of this work and are left to possible future work.

The RT-NM module serves as the cornerstone of the RT extension functionality. This component acts as a transparent intermediary, intercepting all communication exchanged between MQTT clients and the broker. By intercepting these messages, the RT-NM module gains access to RT attributes associated with TS traffic flows. As illustrated in Fig. 6, when the RT-NM module receives a message from an MQTT client, it performs a thorough inspection of the message content

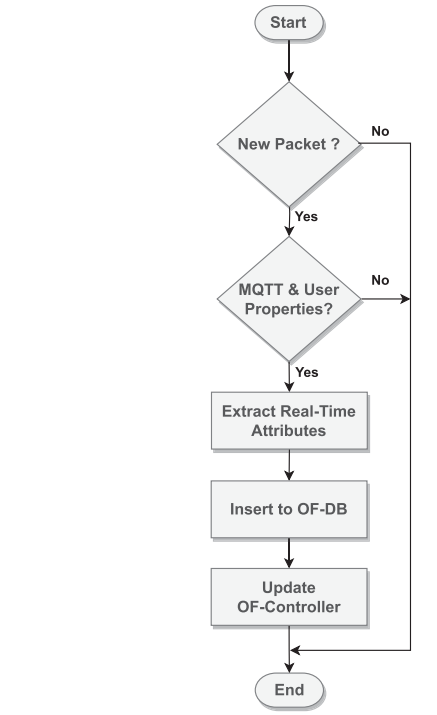

FIGURE 6. **RT-NM operation flow diagram.**

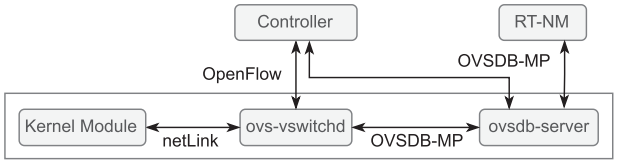

FIGURE 7. **Network configuration process.**

to identify the presence of an RT reservation request. Upon successful identification, the module extracts the relevant RT information and incorporates it into the OF-DB.

As depicted in Fig. 7, the RT-NM transmits RT information to the OVSDB-server using the OVSDB-MP [52]. Upon receiving an update from the RT-NM, the server processes the request and commits the pertinent data to a designated shared memory space, and then the OF-DB is updated accordingly. When updates occur in the OF-DB, the ovs-vswitchd daemon, responsible for managing and controlling OVS switches, retrieves RT information. It collaborates with the OF-controller to analyze the registered RT attributes, subsequently updating the flow tables of the OF-switch and establishing the necessary data paths. In addition, the ovs-vswitchd daemon communicates with the kernel module of the respective node through netLink (a tool to transfer information between the kernel and user-space processes), to carry out the actions corresponding to the processing of each received packet.

The RT information associated with a topic is dynamic and subject to continuous modification by connected MQTT clients. Clients possess the ability to register or update RT

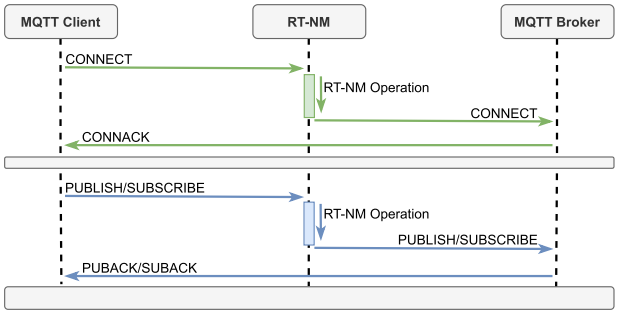

FIGURE 8. Message set up sequence diagram in RT-MQTT.

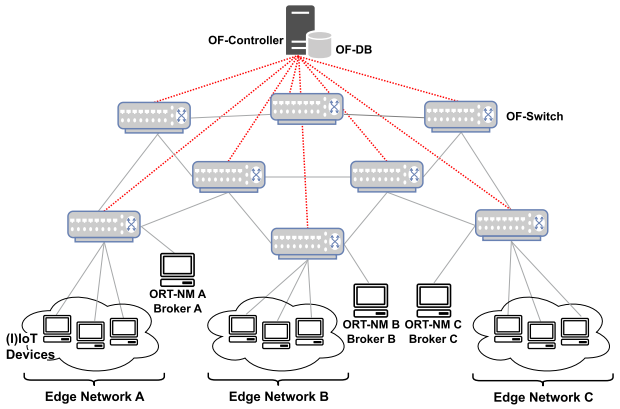

FIGURE 9. High-level MRT-MQTT system architecture in edge networks.

attributes directly within the OF-DB for a specific topic. This attribute modification can occur during the initial connection phase through the CONNECT message or can be realized dynamically at a later stage, such as when a client publishes data via the PUBLISH message. Conversely, subscribers can also specify their own RT requirements during the connection or subscription process using the SUBSCRIBE message. Fig. 8 presents the sequence diagram for these actions, including connecting, publishing, and subscribing. The RT-NM has the role of receiving and validating the messages' attributes. Once validated, the RT-NM executes the necessary network configurations to fulfill the specified requirements. Finally, the broker acknowledges the successful operation by sending a confirmation message back to the client. A technical description of RT-MQTT services is publicly available.[1]

V. EXTENDED FRAMEWORK FOR EDGE NETWORKS

When multiple end-nodes require a specific MQTT topic, the broker generates one distinct unicast transaction per subscriber [43]. This replication of data transmissions significantly amplifies overall network traffic, especially in networks with many devices, such as edge networks. These networks follow a distributed computing paradigm that places computation and data storage in close proximity to the network edge with the goal of minimizing latency and increasing performance by reducing RTT and network traffic [57]. In such scenarios, whenever devices attached to different edge networks subscribe to the same topic, network congestion escalates, ultimately leading to predictability and timeliness issues, which are crucial considerations in IIoT applications.

To tackle this challenge, we extend RT-MQTT to combine RT services and multicast capabilities, thus allowing RT and scalable communication in IIoT edge networks, resulting in a framework called MRT-MQTT [16]. In this framework applications running on edge networks can also define their RT requirements explicitly, as in RT-MQTT. In turn, the SDN/OF resource manager reads these requirements and establishes the corresponding RT channels within and between edge networks. MRT-MQTT also prioritizes TS data flows over NTS ones, providing RT guarantees to TS data. However, contrasting to its predecessor, MRT-MQTT employs multicast routing between edge networks. This option reduces the load in the network and leads to lower latency and jitter of RT data communications while improving scalability.

A. MRT-MQTT REFERENCE ARCHITECTURE

Fig. 9 shows the high-level architecture of MRT-MQTT, which encompasses a network of OF-switches interconnected to a central OF-controller. This controller (see Fig. 2) is extended with an additional module for managing PIM-SM multicast routing in conjunction with MSDP. Furthermore, the design incorporates multiple edge networks that leverage MRT-MQTT, several (I)IoT devices as MQTT clients, and a pair of MQTT broker plus an optimized real-time network manager (ORT-NM) per edge network. Each ORT-NM is responsible for collecting the corresponding edge information, sending it for analysis to the OF-controller and establishing data transmission paths between edge networks. This network manager is strategically positioned between the MQTT clients and the broker in each network, with a preference for co-deployment on the same node as the broker for reducing the communication overhead.

MRT-MQTT extends the flow set definition inherited from the RT-MQTT protocol by introducing the incorporation of the flow topic (FT_i) and MQTT QoS level (Q_i) within the RT attributes, defined as follows:

$$F_i^{\mathrm{TS}} = \{C_i, P_i, T_i, D_i, \mathrm{BW}_i, \mathrm{FT}_i, Q_i\}\,. \tag{4}$$

All messages exchanged between the broker and clients in each edge network are intercepted by the corresponding ORT-NM module that extracts the RT requirements associated with each TS flow, and communicates this information to the OF-controller, which stores it within the SRT. Consequently, the SRT is extended as follows, to include the addresses of the corresponding broker nodes (BA_i):

$$\mathrm{SRT} = (F_i^{\mathrm{TS}}, \mathrm{SRC}_i, \mathrm{DST}_{i,k}, \mathrm{BA}_i, \mathcal{L}_i, n_i)\ i = 1\ldots N. \tag{5}$$

[1] [Online]. Available: https://new-rt-mqtt-extension-api.readthedocs.io/en/latest/

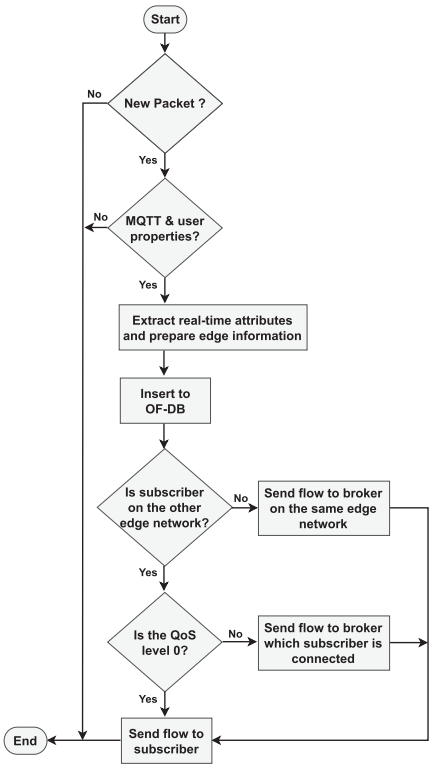

FIGURE 10. ORT-NM operation flow diagram in the proposed system.

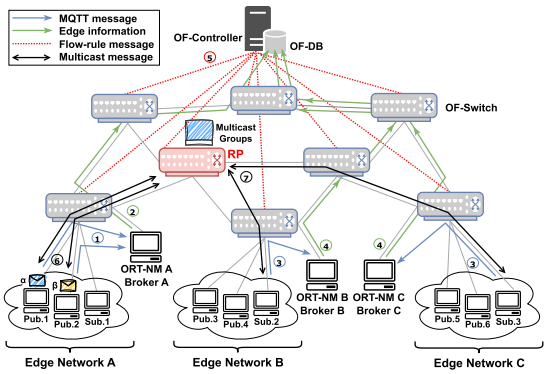

FIGURE 11. Data flow operation of MRT-MQTT in edge networks for QoS 0.

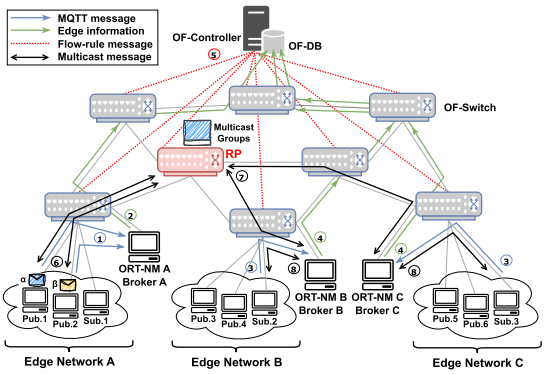

FIGURE 12. Data flow operation of MRT-MQTT in edge networks for QoS 1 and 2.

The edge information contained within the SRT, when combined with the topological insights gathered by the OF-controller, is instrumental in establishing RT channels for all connections traversing MQTT components across multiple edge networks. When a single network is involved, then RT-MQTT can be employed. When subscribers are located in different edge networks, then MRT-MQTT uses the ORT-NM nodes to leverage multicast groups and routes using OVSDB-MP [52] to communicate with the OF-DB. The ORT-NM closely mirrors the operation of the RT-NM, extending it with multicast routing.

Fig. 10 provides a visual representation of ORT-NM's operational workflow. It starts when an MQTT client publishes on a topic instantiated in its broker. The local ORT-NM intercepts the respective message and searches in its contents for RT attributes. If the search is successful, the relevant information is extracted and the OF-DB updated. Then, the OF-controller processes this information and creates paths according to the subscribers location and specified QoS level.

Fig. 11 illustrates the scenario where MQTT subscribers in edge networks B and C (designated as Sub.2 and Sub.3) subscribe to topics α and β. The publishers of these topics are Pub.1 and Pub.2, respectively, both in edge network A but each with a different priority level. Initially, the ORT-NM in edge network A gathers data from Pub.1 and Pub.2 and forwards this information to the OF-controller. Then, the controller generates an SRT containing the publishers RT requirements, their addresses as well as that of the broker in edge network A. When MQTT clients in edge networks B and C subscribe to topics α and β through their respective brokers, the associated ORT-NMs extract the subscribers RT requirements, their addresses and those of the respective edge brokers. Once compiled, this data is also sent to the OF-controller that updates the SRT accordingly and determines communication paths between edge network A and edge networks B and C. Considering that subscribers are spread across multiple edge networks, the OF-controller creates one multicast group per topic maintaining their respective priority levels for data reception. For each multicast group a path tree is defined and the OF-switches configured accordingly. The actual paths for topics α and β is contingent on the QoS level specified by the subscribers. If the QoS level is 0, the data are transmitted directly from the publishers in edge network A to the subscribers in edge networks B and C (see Fig. 11).

Conversely, for QoS levels 1 and 2, which may trigger message retransmissions, the message destination is set to the brokers in edge networks B and C (see Fig. 12). These brokers join the respective multicast groups to obtain the data, which they relay to the subscribers in their respective networks. In cases where nodes within edge network A also subscribe to topics α or β, such intraedge network communication is facilitated by the local broker, adhering to the RT requirements defined by RT-MQTT.

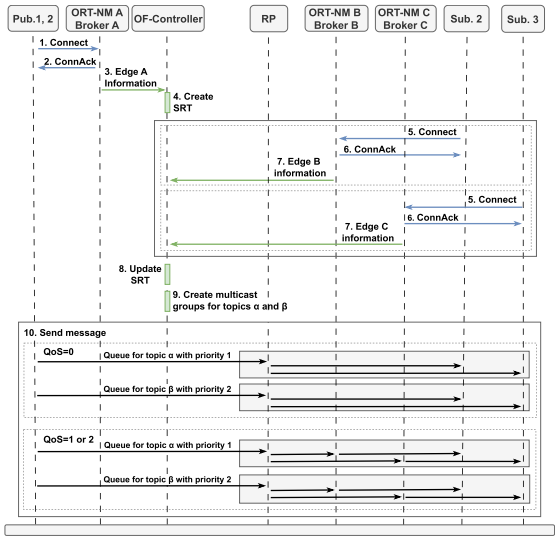

FIGURE 13. Message set up sequence diagram.

Fig. 13 demonstrates the data transfer process from edge network A to edge networks B and C, using MRT-MQTT for QoS levels 0 and 1/2.

B. MULTICAST MECHANISM

As previously mentioned, the MRT-MQTT protocol uses multicast to efficiently distribute data to subscribers residing in different edge networks. This multicast traffic delivery mechanism is based on the PIM-SM protocol and encompasses three key phases. First, the protocol collects edge information from the publisher side. Second, it collects corresponding edge information from the subscribers' side. The information collected in both of these phases is registered in the SRT alongside the group table, effectively linking clients subscribing to the same topics. Finally, the protocol sets a multicast tree for each topic, encompassing the respective publisher and its subscribers. The root of this tree is a designated OF-switch that will act as an RP (see Section II-C), receiving the publisher multicast traffic and forwarding it to the subscribers. The RP designation is done by the OF-controller considering the shortest average delay between each candidate RPs and subscribers [58]. The OF-controller monitors the multicast tree continually, dynamically adapting it as needed to maintain an optimal multicast traffic delivery.

The distribution of MQTT messages from a publisher within edge network A to subscribers in edge networks B and C (such as Sub.2 and Sub.3) employing QoS 0 (see Fig. 11) involves the assignment of a multicast group IP address derived from the topic. The publisher initiates individual TCP connections with each subscriber, sending a request for their participation in the multicast group. Subsequently, each subscriber requests their respective local edge OF-switch to join the multicast group. After joining the multicast group, subscribers transmit an acknowledgment message to the publisher through their TCP connections, indicating their preparedness to receive data. Subsequently, the publisher can start sending data to the multicast group address on the RP, knowing that all subscribers are ready to receive it. Using MSDP, the RP maintains the multicast distribution tree and routes multicast group messages to the subscribers. Upon receiving data, subscribers acknowledge it to the publisher via their individual TCP connections. This process can occur concurrently for multiple publishers (such as Pub.1 and Pub.2 in this setup), each allocated its own multicast group determined by the associated topic and its priority level. When the topic QoS level is configured as 1 or 2 (see Fig. 12) the procedure remains unchanged, albeit with a small difference: the publisher modifies the destination addresses to communicate with the brokers of the target edge networks B and C, rather than the actual subscribers. After the brokers successfully receive the multicast data and acknowledge it to the publisher, the subscribers in edge networks B and C subscribe to the topics in their respective brokers, receiving the respective data.

VI. SCHEDULABILITY ANALYSIS

Although RT computing systems have found extensive use in industrial domains [59], [60], analyzing the worst-case timing behavior of communication protocols can be a challenging endeavor. This section firstly refines the prior analytical approach for TS flows within the RT-MQTT architecture [15], with a focus on enhancing system analyzability and predictability. Then it introduces the adaptation of HA and TA to the MRT-MQTT architecture, enabling to derive end-to-end timeliness guarantees for TS flows in edge networks. The choice of TA was motivated by its tightness, but due to its increased complexity in implementation and validation, especially in the context of unconstrained routing schemes where flows can cross multiple times, a simpler, yet more pessimistic, HA-based analysis was also included. The results obtained through both approaches are subsequently validated, through experimental methods, and their performance is compared (see Section VII).

A. MESSAGE MODEL

Considering the message flow set, or for simplicity, the message set as defined in (2), we consider single-packet messages, which is a common approach in RT applications. It is assumed that client nodes are equipped with an operating system that supports RT capabilities. This allows them to concurrently generate both TS and NTS traffic including standard MQTT messages and general background traffic.

In MQTT, data exchanges are primarily conducted through unicast transactions, where a single data stream is transmitted directly from one sender to one receiver [43]. This unicast approach characterizes the analysis of the RT-MQTT architecture, focusing on individual, point-to-point data streams. When addressing scenarios with multiple subscribers, the framework extends this model, applying a similar unicast-based analysis to each separate broker-to-subscriber connection. In contrast, the MRT-MQTT architecture introduces a

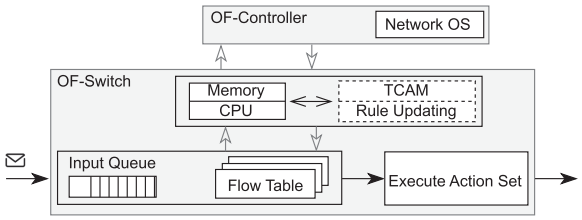

FIGURE 14. SD in an OF-switch [62].

hybrid approach, initially employing unicast for the transmission of data from the source to an RP node, and then shifting to multicast for broader distribution. Here, the unicast stream, upon reaching an RP node, undergoes replication for efficient multicasting to multiple subscribers across edge networks. This dual-phase approach necessitates a more comprehensive analysis in MRT-MQTT, encompassing both individual and group data transmission methods, thereby enhancing network efficiency and scalability.

B. SCHEDULING MODEL

Due to the inherent characteristics and capabilities of current OF-switches, both the RT-MQTT and MRT-MQTT architectures use nonpreemptive fixed-priority scheduling with first-in first-out (FIFO) within each priority level. Based on the priority level of a specific message m_i, we define the following message subsets.

1) $\text{hp}_i = \{\forall m_{j\,j\in[1,N]}, P_j > P_i\}$, subset of messages with priority strictly higher than that of m_i.
2) $\text{sp}_i = \{\forall m_{j\,j\in[1,N], j\neq i}, P_j = P_i\}$, subset of messages with same priority as m_i but excluding m_i.
3) $\text{lp}_i = \{\forall m_{j\,j\in[1,N]}, P_j < P_i\}$, subset of messages with priority strictly lower than that of m_i.

Upon arrival at a switch ingress port, a message m_i is pushed through the OF-pipeline and then forwarded to a designated output port queue according to its path and priority. The packet processing within the OF-pipeline at each switch incurs in a delay that we define as switching delay (SD) and it impacts all messages traversing a switch. This delay can be decomposed into three main components (see Fig. 14) as follows.

1) Time to manage the input queues (essentially memory operations).
2) Time to process the flow tables.
3) Time to execute the action set.

The actual value of SD is variable and depends on factors, such as the switching hardware, input load, the number of flow tables, and the complexity of the action set. Therefore, in practice a conservative bound for SD is determined experimentally and subsequently used in the analysis. SD is seldom constant, though, and its variability, the SD jitter, is denoted as J_i^{SD}.

After the OF-pipeline, a message arrives at an output port queue. Here, message m_i may suffer two types of delay as follows.

- *Blocking delay:* this delay occurs when a lower priority message lp_i is already in transmission, and preemption is not supported.
- *Interference delay:* this delay is caused by the presence of messages belonging to hp_i (due to priority scheduling) and to sp_i (due to FIFO ordering at each priority level).

To ascertain both the SD and jitter, network traffic is captured at both the ingress and egress points of the OF-switches using *tcpdump* [61], a widely used command-line tool favored for its efficiency and lightweight operation, and timestamps for both the sending and receiving of MQTT messages are recorded. The SD is then calculated by subtracting the send time from the receive time for each message, thereby providing a measure of the time taken for messages to traverse the network. Furthermore, switching jitter is calculated by analyzing the variations in time delays and computing the standard deviation of these observed delays to represent the bounded switching jitter.

Finally, note that this article does not delve into the message setup phase, assuming that all resources are preconfigured and the system remains in a static state. Although both RT-MQTT and MRT-MQTT accommodate dynamic updates for RT traffic, the analysis of the updating response time is outside the scope of this article.

C. RESPONSE TIME ANALYSIS FOR RT-MQTT

In RT-MQTT, the response time of an RT message can be broken down into three parts as follows.

1) The time taken to travel upstream from the publisher to the broker.
2) The processing time at the broker.
3) The time taken to travel downstream from the broker to a given subscriber.

The first and third parts of the response time are communication delays, while the second part is a computational delay. The response time analysis for the upstream and downstream segments is computed in the same way, as the scheduling and forwarding mechanisms are the same, being only necessary to set the source and destination nodes properly. For this reason, and without loss of generality, we will focus the analysis on the upstream segment. The literature reports numerous approaches to tackle the RT behavior of MQTT brokers, as evidenced in [63] and [64]. These approaches can be seamlessly integrated with the analysis presented in this article, to compute the overall delay in publisher–subscriber communication and, for this reason, will not be further considered in this article.

1) RESPONSE TIME ANALYSIS USING THE HA

HA [65] is based on the accumulation of delays and jitter on a given path $\mathcal{L}_i$ extending from the source to the destination nodes. Delay and jitter computation is carried out for each individual link and switch in the path, with the total response time being the sum of the response times for all the constituent links. The calculation of link delay includes interference,

blocking, switch delays, and the cumulative impact of jitter throughout the path. In order to maintain compatibility with standard nodes within the network, NTS traffic is intentionally left unreported within the RT-MQTT framework, because its characteristics and inherently unknown. The only possible impact of NRT on TS traffic is blocking, as NRT traffic is handled in the background, having the lowest priority level. As such, its worst-case impact can be accounted for simply via a blocking term that matches the link maximum transmission unit (MTU).

In each link, whether it is a source node or an OF-switch output, incoming messages are forwarded to prioritized queues. As previously mentioned, messages within each queue or priority level follow a FIFO discipline and are transmitted nonpreemptively. This model lends itself to analysis using a well-established method for response time without preemption in uniprocessors, known as the cumulative delays method [65]. Moreover, a crucial aspect of this analysis involves identifying the critical instant, defined as the message release pattern that leads to the worst-case interference a message can encounter. In this context, the critical instant for message m_i occurs when it is released immediately after a lower priority message with the maximum size, thus maximizing the blocking delay. Simultaneously, it is released immediately after all messages of the same priority and concurrently with all higher priority messages, considering their maximum release jitter to ensure maximum interference.

Nonpreemptive transmissions can suffer from the push-through effect, a phenomenon in which a specific instance of a message m_i is delayed by its own previous instance. Consequently, the WCRT for m_i at a given output port may manifest in instances beyond its initial release during a critical instant, within the so-called **occupied period**. The number of instances following the critical instant that requires examination to determine the WCRT can be computed as $Q_i = \lceil (w_i + J_i)/T_i \rceil$ where J_i is the release jitter, and w_i corresponds to the length of the level-i busy period. This period can be computed with a fixed-point iterative method as outlined in (6). A possible initial value is given by $w_i^0 = B_i + C_i$, where B_i signifies the blocking delay resulting from NTS and lp_i messages, and concludes when $w_i^{n+1} = w_i^n$

$$w_i^{n+1} = B_i + \sum_{j \in \mathrm{hp}_i \cup \mathrm{sp}_i} \left\lceil \frac{w_i^n + J_j}{T_j} \right\rceil .C_j. \tag{6}$$

The level-i occupied period starts at the critical instant and ends at the onset of the subsequent level-i **idle period**. This occupied period arises when the queues for priorities P_i and above at the output port have been completely drained. Determining the worst-case delay for each of the Q_i message instances occurring within the level-i occupied period requires an individual computation to account for the load introduced by the preceding q instances. Equation (7) provides an upper bound for these delays, denoted as $v_i(q)$. It can be solved iteratively using a fixed-point iteration approach, potentially

Algorithm 1: WCRT Calculation: HA for RT-MQTT.

Input: Γ

Output: RT_i^{Total}

```
/* A. Compute delays & jitter at source node:      */
RT_i^Total = UnicastResponseTimeCalc(i, 1)
J_i^acc = J_1 + J_{i,1}^QP
k = 2
/* B. Compute delays & jitter for each switch:     */
for k from 2 to n_i do
    SD_{i,k} = SwitchingDelayCalc(i, k)
    J_i^acc += J_{i,k}^SD
    RT_{i,k} = UnicastResponseTimeCalc(i, k)
    J_i^acc += J_{i,k}^QP
    RT_i^Total += RT_{i,k} + SD_{i,k}
end for
```

initialized with $v_i^0(q) = B_i + qC_i$. The iteration continues until one of two conditions is met: either $v_i^{n+1}(q) = v_i^n(q)$ or when $v_i^{n+1}(q) + C_i - qT_i > D_i - J_i$, at which point the deadline cannot be guaranteed

$$v_i^{n+1}(q) = B_i + qC_i + \sum_{j \in \mathrm{hp}_i \cup \mathrm{sp}_i} \left(\lfloor \frac{v_i^n(q) + J_j}{T_j} \rfloor + 1 \right) .C_j. \tag{7}$$

The WCRT for an instance that precedes q instances can be calculated using $\mathrm{RT}_i(q) = v_i(q) + C_i - qT_i$. This equation, as represented by (8), provides us with the maximum response time for message m_i

$$\mathrm{RT}_i = \max_{q=0,1\ldots Q_i-1}(v_i(q) + C_i - qT_i). \tag{8}$$

To compute the WCRT for a specific message m_i in the RT-MQTT, Algorithm 1 is utilized. This algorithm takes into account the complete path from the source to the broker, considering both the network topology and the set of messages as its inputs.

The algorithm uses two accumulators that track the accumulated response time ($\mathrm{RT}_i^{\mathrm{Total}}$) and release jitter ($J_i^{\mathrm{acc}}$) over the successive links along the message path. Initially (part A), it is processed the output link of the source node (k = 1), which involves computing the initial value for the response-time [ResponseTimeCalc(i, 1), by applying (8)], and the initial value of the accumulated output jitter (J_1 plus $J_{i,1}^{\mathrm{QP}}$). While the primary one is the maximum release jitter of tasks and the queuing effect of the interface card at the source node, the other significant element is the response time jitter. The response time jitter is calculated by subtracting the best-case response time (BCRT) from the WCRT. The BCRT represents the optimal scenario of the message fastest possible transmission time in the absence of external delays. The BCRT calculation assumes the analysis without interference from messages of higher or equal priority, no blocking delay from lower priority or non-RT traffic, and zero jitter.

Part B of the algorithm closely resembles part A but also accounts for the SD upper bound introduced by the switch under analysis and the additional jitter it may induce. This process is repeated for links from the second to the final link in the

path of m_i. The SD and jitter are determined experimentally, as discussed in Section VI-B.

2) RESPONSE TIME ANALYSIS USING TA

TA [66] entails the identification of messages within the network that result in the postponement of a specific message m_i generated at time t. These delaying messages are situated along the path $\mathcal{L}_i$ that m_i traverses from the source to the destination node. The analysis evaluates their impact on response times, considering the links visited by m_i, starting from the last link and progressing backward to the source link. At each link, the identified messages create a so-called **busy period**. The response time for message m_i emerges from the cumulative addition of these busy periods plus the overall transmission delay across the network.

Considering fixed-priority nonpreemptive scheduling, the processing of a message cannot experience any further delays once initiated. Therefore, TA [66] is employed for the recursive determination of the latest starting time of message m_i on its last visited switch last_i (9). The notation $(1+\lfloor x \rfloor)^+$ stands for $\max(0; \lfloor x \rfloor)$. In addition, within this context, the terms $\text{first}_{j,i}$ (representing the first switch visited by message m_j on path $\mathcal{L}_i$), $\text{last}_{j,i}$ (referring to the last switch visited by m_j on the same path), $\text{slow}_{j,i}$ (indicating the slowest switch visited by m_j on path $\mathcal{L}_i$), and $\text{pre}_i(\text{sw})$ (representing the switch visited immediately before the switch sw by m_i) play significant roles. In turn, $S\text{max}_i^{\text{sw}}$ denotes the maximum time required by m_i to travel from its source switch to switch sw. On the other hand, $S\text{min}_j^{\text{sw}}$ (in relation to m_j) represents the minimum time for the journey from its source switch to switch sw, while $S\text{max}_j^{\text{sw}}$ indicates the maximum travel time for the same journey. Finally, $A_{j,i}$ can be defined as $S\text{max}_j^{\text{first}_{j,i}} - M_i^{\text{first}_{j,i}} + J_j$. The term $M_i^{\text{first}_{j,i}}$ is used to denote the minimum time necessary for a message to traverse the path from the source of m_i to switch $\text{first}j, i$, as defined in (10)

$$W_{i,t}^{\text{last}_i} = \sum_{j \in \text{hp}_i} \left(1 + \left\lfloor \frac{W_{i,t}^{\text{last}_{j,i}} - S\text{min}_j^{\text{last}_{j,i}} + A_{j,i}}{T_j} \right\rfloor\right)^+ .\text{SD}_j^{\text{slow}_{j,i}} + \sum_{j \in \text{sp}_i} \left(1 + \left\lfloor \frac{t + S\text{max}_i^{\text{first}_{j,i}} - S\text{min}_j^{\text{first}_{j,i}} + A_{j,i}}{T_j} \right\rfloor\right)^+ .\text{SD}_j^{\text{slow}_{j,i}} + \sum_{\substack{\text{sw} \in \mathcal{L}_i \\ \text{sw} \neq \text{slow}_i}} \max_{j \in \text{hp}_i \cup \text{sp}_i} \{\text{SD}_j^{\text{sw}}\} + (|\mathcal{L}_i|.C_i) + B_i^{\text{TA}} - \text{SD}_i^{\text{last}_i} \tag{9}$$

$$M_i^{\text{first}_{j,i}} = \sum_{\text{sw}=\text{first}_i}^{\text{pre}_i(\text{first}_{j,i})} \left(\min_{j \in \text{hp}_i \cup \text{sp}_i} \{\text{SD}_j^{\text{sw}}\} + C_i \right). \tag{10}$$

Algorithm 2: WCRT Calculation: TA for RT-MQTT.

Input: Γ
Output: RT_i^{Total}

```
/* A. Initialize the iteration:                    */
sw = 1
for sw = first_j to pre(first_{j,i}) do
    Smax_j^{first_{j,i}}(1) += SD_j^{sw} + C_i
end for
/* B. Compute latest starting time & response time:
   */
while true do
    sw += 1
    for m_j ∈ (hp_i ∪ sp_i) do
        for sw ∈ L_i do
            if (sw = last_i) or (m_j cross m_i such that
              sw = last_{j,i} or sw = pre_i(first_{j,i}) then
                W_{i,t}^{sw} = LatestStartTimeCalc(sw, i)
                if (m_j cross m_i such that sw = pre_i(first_{j,i}))
                  then
                    Smax_i^{first_{j,i}}(sw + 1) =
                      W_i^{sw}(t) + SD_i^{sw} − t + C_i
                    if Smax_i^{first_{j,i}}(sw + 1) =
                      Smax_i^{first_{j,i}}(sw) then
                        break
                if sw = last_i then
                    RT_i^{Total} = ResponseTimeCalc(i)
                    if RT_i^{Total} > D_i then
                        break
            end if
        end for
    end for
end while
```

In $W_{i,t}^{\text{last}_i}$, the first two terms account for the maximum interference experienced by message m_i due to packets with higher and same priority, respectively. The third term corresponds to the SD of messages with same or equal priority at each switch along path $\mathcal{L}_i$, except the slowest one. The fourth term indicates the maximum transmission delay on links that m_i traverses within the path $\mathcal{L}_i$. The term B_i^{TA} denotes the blocking delay due to nonpreemption, as defined by

$$B_i^{\text{TA}} = \left(\max_{j \in \text{lp}_i} \{\text{SD}_j^{\text{first}_i}\} - 1 \right)^+ + \sum_{\substack{\text{sw} \in \mathcal{L}_i \\ \text{sw} \neq \text{first}_i}} \times \left(\left(\max_{\substack{j \in \text{lp}_i \\ \text{sw}=\text{first}_{j,i}}} \{\text{SD}_j^{\text{sw}}\} - 1 \right) ; \left(\max_{\substack{j \in \text{lp}_i \\ \text{sw} \neq \text{first}_{j,i}}} \{\text{SD}_j^{\text{sw}}\} - \text{SD}_i^{\text{sw}-1} \right) \right)^+ . \tag{11}$$

The final term in the equation is computed considering the start time of message m_i transmission at the output port last_i. Consequently, (12) provides the WCRT for m_i given by TA

$$\text{RT}_i = W_{i,t}^{\text{last}_i} + \text{SD}_i^{\text{last}_i} - t. \tag{12}$$

Algorithm 2 outlines the method for determining the WCRT of a message m_i in the RT-MQTT framework, employing the TA. This calculation takes into account the complete path from the source to the broker, and it is conducted based on key inputs such as the network topology and the set of messages.

The algorithm comprises two distinct parts. In the first part (A), the computation is centered around determining the value of $Smax_j^{first_{j,i}}(sw)$ (specifically for sw = 1) for any message m_j within the set $(hp_i \cup sp_i)$ that traverses m_i. This value serves as the initialization step for the subsequent iterative process aimed at calculating the overall response time for the entire route (RT_i^{Total}). The second part (B) marks the beginning of the iterative computation for the WCRT of m_i. Within this phase, the algorithm considers all messages m_j from the set $(hp_i \cup sp_i)$ to identify scenarios that could lead to the most unfavorable conditions for m_i. It involves the determination of the latest starting time [LatestStartTimeCalc(sw, i) as described in (9)] at the last switch visited and subsequently calculates the WCRT for m_i [ResponseTimeCalc(i) in accordance with (12)]. The termination condition for the algorithm is met when there exists a message for which the end-to-end response time surpasses its predefined end-to-end deadline.

D. RESPONSE TIME ANALYSIS FOR MRT-MQTT

The analysis of MRT-MQTT aligns with that of RT-MQTT, as detailed in Section VI-C, encompassing scenarios from publisher to subscriber(s) or publisher to broker(s), depending on whether brokers are involved. This approach mirrors the methodology used in RT-MQTT, similarly omitting the broker processing time in the analysis. A key distinction arises, however, in the nature of the data streams. Initially, these streams are unicast from the source to the RP switch. Beyond this point, the unicast streams are replicated and transition into multicast streams directed toward the destination nodes. These nodes can be either subscribers or brokers on the subscription side, depending on the QoS level. This transformation enables the efficient handling of multiple, simultaneous subscriber requests, marking a key difference in the data handling between RT-MQTT and MRT-MQTT systems.

In multicast scenarios, the functions and responsibilities of RP switches encompass critical tasks, such as establishing and overseeing multicast groups, duplicating messages, and keeping track of multicast group memberships along with the distribution trees. These RP switches serve as pivotal branching points where these additional operations lead to increased latency. Therefore, this extra time incurred should be factored into the overall SD. Consequently, the SD uniquely associated with the RP switches for a message m_i is represented as SD_i^{RP}. Furthermore, the implementation of multicasting mechanisms such as join/prune messaging for multicast group management introduces variability in packet delivery times, which contributes to additional jitter J_{mc}. This specific delay and jitter are also empirically determined, as discussed in Section VI-B.

1) RESPONSE TIME ANALYSIS USING THE HA

In MRT-MQTT, the previously conducted HA analyses are adopted to calculate delays and jitter for a path $\mathcal{L}_i$ from source to destination. This involves summing response times of individual links and switches, considering factors, such as interference, blocking, and cumulative jitter impact. NTS traffic still is modeled by the MTU.

Considering the branching effects of the multicast tree, the critical instant still accounts for the release of messages with lower, equal, or higher priority, each factoring in its maximum release jitter. Accordingly, the number of instances following the critical instant is calculated by $Q_i = \lceil (w_i + J_i + J_{mc})/T_i \rceil$, and the length of the level-i busy period can be extended as follows:

$$w_i^{n+1} = B_i + \sum_{j \in hp_i \cup sp_i} \left\lceil \frac{w_i^n + J_j + J_{mc}}{T_j} \right\rceil .C_j. \tag{13}$$

This period is determined using a fixed-point iteration akin to the previous analysis, starting with $w_i^0 = B_i + C_i$ and ending when $w_i^{n+1} = w_i^n$.

To encapsulate multicast effects, (14) is also adjusted to calculate an upper bound for the accumulated delays $v_i(q)$. The resolution proceeds via a fixed-point iteration approach, commencing with an initial value $v_i^0(q) = B_i + qC_i$. The process halts when either $v_i^{n+1}(q) = v_i^n(q)$ or when $v_i^{n+1}(q) + C_i - qT_i > D_i - (J_i + J_{mc})$, indicating that the deadline assurance is untenable

$$v_i^{n+1}(q) = B_i + qC_i + \sum_{j \in hp_i \cup sp_i} \left(\lfloor \frac{v_i^n(q) + J_j + J_{mc}}{T_j} \rfloor + 1 \right) .C_i. \tag{14}$$

Consequently, the WCRT for an instance is $RT_i(q) = v_i(q) + C_i - qT_i$ and the WCRT for m_i can now be repeated as follows:

$$RT_i = \max_{q=0,1\ldots Q_i-1}(v_i(q) + C_i - qT_i). \tag{15}$$

Algorithm 3 presents a method for calculating the WCRT of a message m_i in MRT-MQTT, from source to destination. It takes the network topology and message set as inputs for this computation.

The algorithm has three parts. The first two parts calculate the response times from a source node to the RP node which considers unicast streams, while the third part calculates the response times of multicast streams originating from the RP node to multiple destinations. The first part, labeled A, mirrors part A of Algorithm 1. The second part (B) is similar to the first one, involving the bounded SD and the additional jitter it may cause. This part is repeated from the second to the last link that m_i crosses to reach the RP node, with D_{up} representing the depth of the unicast path. The SD and jitter are measured experimentally, as discussed in Section VI-B.

The last part of the Algorithm (C) focuses on calculating delays and jitters for switches that multicast message m_i from the RP node to the intended destination nodes. For each path $l_i \subsetneq \mathcal{L}_i$ from the RP to each specific destination, the Algorithm determines the cumulative delay and jitter across every link in that path. The process begins right after the

Algorithm 3: WCRT Calculation: HA for MRT-MQTT.

Input: Γ
Output: RT_i^{Total}

```
/* A. Compute delays and jitter at source node:   */
RT_i^Total = UnicastResponseTimeCalc(i, 1)
J_i^acc = J_1 + J_{i,1}^QP
/* B. Compute delays & jitter for switches to RP:
   */
for k from 2 to D_up do
    SD_{i,k} = SwitchingDelayCalc(i, k)
    J_i^acc += J_{i,k}^SD
    RT_{i,k} = UnicastResponseTimeCalc(i, k)
    J_i^acc += J_{i,k}^QP
    RT_i^Total += RT_{i,k} + SD_{i,k}
end for
/* C. Compute delays & jitter for switches from RP:
   */
foreach l_i do
    for k = D_ur + 1 to n_i do
        if link k connects to an RP then
            SD_{i,k} = SwitchingDelayCalc(i, k)
            J_i^acc += J_{i,k}^SD
            RT_{i,k} = MulticastResponseTimeCalc(i, k)
            J_i^acc += J_{i,k}^QP
            RT_i^Total += RT_{i,k} + SD_{i,k}
            foreach clustered path l_j starting from RP after link k do
                for k' = 1 to length(l_j) do
                    SD_{i,k'}^RP = SwitchingDelayCalc(i, k')
                    J_i^acc += J_{i,k'}^{SD RP}
                    RT_{i,k'} =
                      MulticastResponseTimeCalc(i, k')
                    J_i^acc += J_{i,k'}^QP
                    RT_i^Total += RT_{i,k'} + SD_{i,k'}^RP
        else
            SD_{i,k} = SwitchingDelayCalc(i, k)
            J_i^acc += J_{i,k}^SD
            RT_{i,k} = MulticastResponseTimeCalc(i, k)
            J_i^acc += J_{i,k}^QP
            RT_i^Total += RT_{i,k} + SD_{i,k}
        end if
    end for
end foreach
```

RP and calculates the SD $\mathrm{SD}^{\mathrm{RP}}$, and then the response time for each multicasting link [MulticastResponseTimeCalc(i, 1) according to (15)], subsequently updating the accumulated jitter and total response time. Once the computations for a path are complete, the total response time $\mathrm{RT}_i^{\mathrm{Total}}$ for that specific path (l_i) is returned for later use. This procedure repeats for all routes leading from the RP to their respective destination nodes.

2) RESPONSE TIME ANALYSIS USING THE TA

In MRT-MQTT, the TA analysis conducted on RT-MQTT is also adaptable for identifying network messages that delay a specific message m_i, by tracking its path $\mathcal{L}_i$ from the source to the destination.

As described before, adapting the TA response time analysis for multicast scenarios involves incorporating additional delay and jitter specific to multicast communication. Accordingly, the latest starting time of message m_i on its last visited switch ($W_{i,t}^{\mathrm{last}_i}$) defined in (9) can be adopted as follows:

$$
\begin{aligned}
&W_{i,t}^{\mathrm{last}_i} \\
&= \sum_{j\in \mathrm{hp}_i} \left(1 + \left\lfloor \frac{W_{i,t}^{\mathrm{last}_{j,i}} - S\mathrm{min}_j^{\mathrm{last}_{j,i}} + A_{j,i}}{T_j} \right\rfloor\right)^+ \cdot \mathrm{SD}_{j,i}^{\mathrm{mpt}} \\
&+ \sum_{j\in \mathrm{sp}_i} \left(1 + \left\lfloor \frac{t + S\mathrm{max}_i^{\mathrm{first}_{j,i}} - S\mathrm{min}_j^{\mathrm{first}_{j,i}} + A_{j,i}}{T_j} \right\rfloor\right)^+ \cdot \mathrm{SD}_{j,i}^{\mathrm{mpt}} \\
&+ \sum_{\substack{\mathrm{slow},\mathrm{RP}\in\mathcal{L}_i \\ \mathrm{slow}\neq\mathrm{RP}}} \min_{j\in \mathrm{hp}_i\cup \mathrm{sp}_i} \{\mathrm{SD}_j^{\mathrm{slow}_{j,i}}, \mathrm{SD}_{j,i}^{\mathrm{RP}}\} \\
&+ (|\mathcal{L}_i|.C_i) + B_i^{\mathrm{TA}} - \Delta\mathrm{SD}_i^{\mathrm{last}_i}
\end{aligned} \tag{16}
$$

where $A_{j,i}$ and $M_i^{\mathrm{first}_{j,i}}$ are defined as follows:

$$
A_{j,i} = S\mathrm{max}_j^{\mathrm{first}_{j,i}} - M_i^{\mathrm{first}_{j,i}} + (J_j + J_j^{\mathrm{mc}}) \tag{17}
$$

$$
M_i^{\mathrm{first}_{j,i}} = \sum_{\mathrm{sw}=\mathrm{first}_i}^{\mathrm{pre}_i(\mathrm{first}_{j,i})} \left(\min_{j\in \mathrm{hp}_i\cup \mathrm{sp}_i} \{\Delta\mathrm{SD}_j^{\mathrm{sw}}\} + C_i\right). \tag{18}
$$

The $\Delta\mathrm{SD}^{\mathrm{sw}}$ is defined in (19) to adjust the delay for a switch based on whether it is the RP or not. If it is the RP ($\delta^{\mathrm{RP}} = 1$), then the RP-specific delay is included. If not ($\delta^{\mathrm{RP}} = 0$), the SD is equal to $\mathrm{SD}_i^{\mathrm{sw}}$

$$
\Delta\mathrm{SD}^{\mathrm{sw}} = \delta^{\mathrm{RP}} \cdot \mathrm{SD}_i^{\mathrm{RP}} + (1 - \delta^{\mathrm{RP}}) \cdot \mathrm{SD}_i^{\mathrm{sw}}. \tag{19}
$$

The $\mathrm{SD}_{j,i}^{\mathrm{mpt}}$ denotes the switch passed message m_j in path $\mathcal{L}_i$ that takes the maximum processing time by compressing the slowest switch and RP switch, as follows:

$$
\mathrm{SD}_{j,i}^{\mathrm{mpt}} = \max_{j\in \mathrm{hp}_i|\mathrm{sp}_i} \left\{\mathrm{SD}_j^{\mathrm{slow}_{j,i}}, \mathrm{SD}_{j,i}^{\mathrm{RP}}\right\}. \tag{20}
$$

The blocking delay B_i^{TA} directly attributed to the nonpreemptive effect is determined as follows:

$$
\begin{aligned}
B_i^{\mathrm{TA}} = &\left(\max_{j\in \mathrm{lp}_i} \{\Delta\mathrm{SD}_j^{\mathrm{first}_i}\} - 1\right)^+ \\
&+ \sum_{\substack{\mathrm{sw}\in\mathcal{L}_i \\ \mathrm{sw}\neq\mathrm{first}_i}} \left(\left(\max_{\substack{j\in \mathrm{lp}_i \\ \mathrm{sw}=\mathrm{first}_{j,i}}} \{\Delta\mathrm{SD}_j^{\mathrm{sw}}\} - 1\right); \left(\max_{\substack{j\in \mathrm{lp}_i \\ \mathrm{sw}\neq\mathrm{first}_{j,i}}} \{\Delta\mathrm{SD}_j^{\mathrm{sw}}\}\right.\right. \\
&\left.\left. - \Delta\mathrm{SD}_i^{\mathrm{sw}-1}\right)\right)^+.
\end{aligned} \tag{21}
$$

The final term of $W_{i,t}^{\mathrm{last}_i}$ is deducted, accounting for the start time of message transmission for m_i at the output port last_i.

Consequently, (22) provides the WCRT for m_i through the use of TA

$$
\mathrm{RT}_i = W_{i,t}^{\mathrm{last}_i} + \Delta\mathrm{SD}_i^{\mathrm{last}_i} - t. \tag{22}
$$

Algorithm 4:. WCRT Calculation: TA for MRT-MQTT.

Input: Γ
Output: RT_i^{Total}

```
/* A. Initialize the iteration:                    */
sw = 1
for sw = first_j to pre(first_{j,i}) do
    if δ^RP = 1 then
        ΔSD_j^sw = SD_j^RP
    else
        ΔSD_j^sw = SD_i^sw
    Smax_j^{first_{j,i}}(1) += ΔSD_j^sw + C_i

/* B. Compute latest starting time & response time:
   */
while true do
    sw += 1
    for m_j ∈ (hp_i ∪ sp_i) do
        for sw ∈ L_i do
            if (sw = last_i) or (m_j cross m_i such that
              sw = last_{j,i} or sw = pre_i(first_{j,i}) then
                W_{i,t}^sw = LatestStartTimeCalc(sw, i)
                if (m_j cross m_i such that sw = pre_i(first_{j,i}))
                  then
                    if δ^RP = 1 then
                        ΔSD_i^sw = SD_i^RP
                    else
                        ΔSD_i^sw = SD_i^sw
                    Smax_i^{first_{j,i}}(sw + 1) =
                      W_i^sw(t) + ΔSD_i^sw − t + C_i
                    if Smax_i^{first_{j,i}}(sw + 1) =
                      Smax_i^{first_{j,i}}(sw) then
                        break
            if sw = last_i then
                RT_i^Total = ResponseTimeCalc(i)
                if RT_i^Total > D_i then
                    break
            end if
        end for
    end for
end while
```

Algorithm 4 outlines the process for determining the WCRT of m_i in MRT-MQTT, from source to destination using the TA, based on network topology and message set inputs.

This algorithm closely resembles Algorithm 2, with the primary distinction being its assessment of whether a switch is an RP to determine the bounded SD ($\Delta \mathrm{SD}_i^{\mathrm{sw}}$). In part (A), it computes $S\mathrm{max}_j^{\mathrm{first}j,i}(\mathrm{sw}=1)$, considering the RP status. Subsequently, part (B) begins an iterative process to calculate the WCRT for m_i, as defined in (22). This involves determining the latest starting time using LatestStartTimeCalc(sw, i), as specified in (16).

VII. VERIFICATION AND PERFORMANCE ASSESSMENT

This section details an empirical study conducted through the Mininet emulation framework, encompassing various scenarios differentiated by their complexity and load levels. The primary objective is to validate the accuracy and evaluate the relative performance of the analyses proposed for the MRT-MQTT protocol. The experimentally obtained values are subsequently juxtaposed against the analytically computed response times, derived using both HA and TA. Furthermore, the conducted experiments entail a thorough comparison of the MRT-MQTT protocol with its counterparts, namely standard MQTT and DM-MQTT as the most comparable architecture, highlighting the performance advantages of MRT-MQTT. The analysis for the foundational RT-MQTT protocol has been previously proposed and validated in [13] and [15]. For the sake of consistency, the analysis and experiments focus on the upstream path delay, only. This delay spans from the moment of writing to the respective socket on the publisher side to the corresponding reception at the broker. A similar analysis can be directly applied to the downstream path delay.

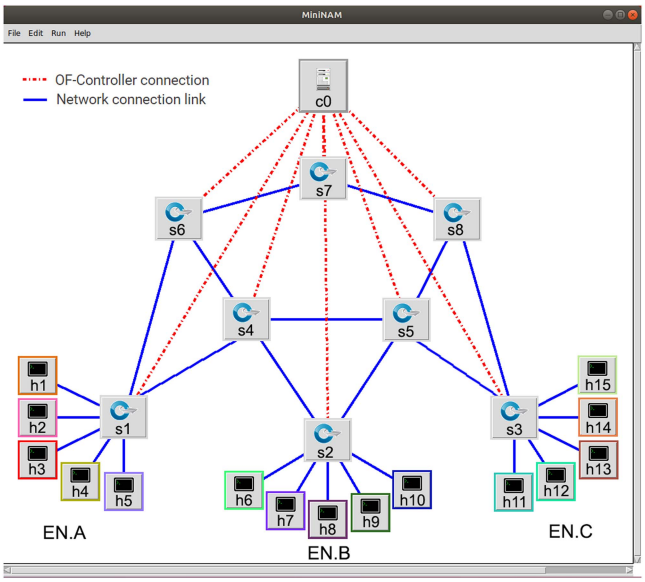

FIGURE 15. Network topology used in MiniNet.

A. EMULATION SETUP

The experimental results were obtained using the Mininet virtual network emulator, version 2.3.0d6,[2] in conjunction with the Eclipse Mosquitto [67] (v2.0.10) and the Eclipse Paho MQTT library for establishing MQTT clients and brokers. The Mininet framework was deployed on a laptop equipped with a 4.9 GHz Intel Core i7 processor and 16 GB of RAM. In addition, the RYU OF-controller[3] served as the SDN controller, running on the same laptop.

The impact of retransmissions on RT traffic response time falls beyond the scope of this article, so the analyses and emulation assume no transmission errors. Nevertheless, to emulate realistic scenarios, in which often it is important to have delivery guarantees, the experiments were designed with the QoS for all MQTT messages set to 1 (delivery at least once).

The experimental setup involved a network topology, as depicted in Fig. 15. This topology comprises eight OF-switches (labeled `s1` to `s8`), connected to three edge networks (`EN.A`,

[2][Online]. Available: http://mininet.org/
[3][Online]. Available: https://ryu-sdn.org/

EN.B, and EN.C), and includes an OF-controller (c0). The setup features nine nodes (h1 to h3, h6 to h8, and h11 to h13) in each edge network, designated as MQTT publishers. These nodes periodically publish messages on distinct topics. Specifically, nodes h3, h8, and h13 are assigned to publish normal MQTT traffic (NTS), while the others handle TS traffic. Each edge network contains one subscriber located at nodes h4, h9, and h14, respectively, to subscribe to these publications. The broker and the ORT-NM are hosted on nodes h5, h10, and h15 within each edge network. To emulate diverse data exchanges typical of industrial scenarios, various applications are used: the distributed Internet traffic generator (D-ITG)[4] generates TCP packets from node h3; VLC media player sends audio/video from node h8; and vsftpd manages file transfer protocol[5] from node h13, all to the corresponding receiver nodes in all broker locations. The bandwidth for non-MQTT traffic sources was limited to 10 Mbit/s for D-ITG, 6 Mbit/s for VLC, and 4 Mbit/s for vsftpd. This setup reflects common industrial application settings, with the link speed set at 100 Mbit/s.

In this topology, three load levels are established, namely Low, Medium, and High. Under the Low load level, each publisher node functions as a gateway and comprises 20 publishers, transmitting MQTT messages on the same topic. For the Medium and High load levels, the number of publishers per node increases to 40 and 60, respectively. MQTT publications are asynchronous, occurring at nominal periods randomly generated within the range of [15 20]ms, affected by a release jitter also randomly generated and comprised in the interval [0 0.4]ms, leading to varying interference patterns. The publications used a single Ethernet packet of maximum size (1500 bytes), resulting in a transmission time of approximately 123 μs.

In the designed scenario, the experiments involve three edge networks (EN.A, EN.B, and EN.C). Each network publishes two unique, TS topics, which are then requested by all subscribers across these three networks, generating intra- and internetwork traffic. The RT-MQTT protocol supports data transmission within edge networks, and in this case the ORT-NM performs essentially as an RT-NM. Conversely, communications involving subscribers located in different edge networks are managed through multicast groups and routes orchestrated by the MRT-MQTT protocol, with the extended services provided by ORT-NM being used.

To keep the amount of results manageable, and without loss of generality, only edge traffic between edge network A and edge networks B and C is included in the evaluation. The notation $\mathcal{L}_{XY}$ is used to denote the routes taken by topics published in edge network X to reach brokers in edge network Y, while hEN.A denotes the publisher nodes in EN.A, $h1$ and $h2$, each publishing a distinct topic, α and β, respectively, distributed to all brokers, specifically $h5$, $h10$, and $h15$. Therefore, messages under evaluation transverse the following paths.

- $\mathcal{L}_{AA} = \{\langle hEN.A, s1\rangle, \langle s1, h5\rangle\}$.
- $\mathcal{L}_{AB} = \{\langle hEN.A, s1\rangle, \langle s1, s4\rangle, \langle s4, s2\rangle, \langle s2, h10\rangle\}$.
- $\mathcal{L}_{AC} = \{\langle hEN.A, s1\rangle, \langle s1, s4\rangle, \langle s4, s5\rangle, \langle s5, s3\rangle, \langle s3, h15\rangle\}$.

To ensure timely handling of various topics, TS ones are evenly distributed across two distinct classes in each edge network, with each class assigned a specific deadline and priorities determined based on the deadline monotonic policy, as shown in Table 2.

To guarantee the comparability of results, the same configuration is employed for both emulation and analysis. The *tcpdump* tool is utilized for measuring jitter and SDs, following the methodology described in Section VI-B. To reduce the impact of platform-induced perturbations, timestamps are temporarily stored in memory until all simulation runs are complete.

TABLE 2. Topic Priority Distribution

	EN.A Pubs	EN.B Pubs	EN.C Pubs	P_i	D_i
Class 1	h1	h6	h11	1	20
Class 2	h2	h7	h12	2	15

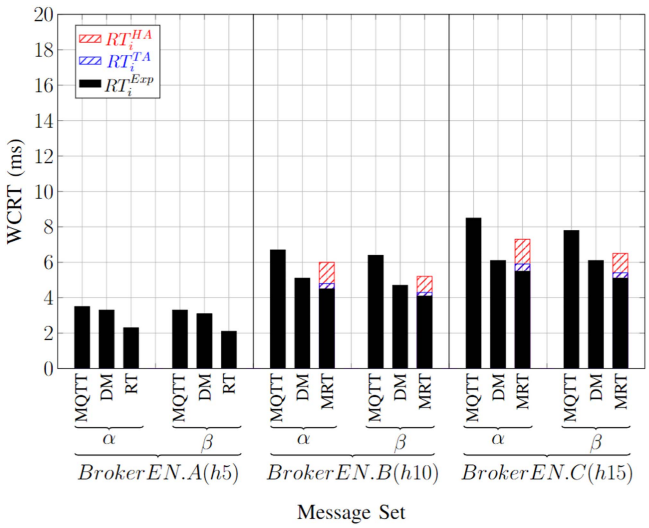

FIGURE 16. Analytical RT_i^{HA} and RT_i^{TA} versus observed RT_i^{Exp} WCRT(s) obtained using MQTT, DM-MQTT, and MRT-MQTT in load level Low.

B. EXPERIMENTAL RESULTS

The evaluation considered three load levels {Low, Medium, High} as previously described. For each load were executed 1000 iterations, with publishers dispatching a minimum of 100 messages in each iteration. For every iteration, data were collected and analyzed to ascertain the WCRT of all messages. In addition, the WCRT for each configuration in MRT-MQTT was calculated using both HA and TA, presented in Algorithms 3 and 4, respectively.

Figs. 16–18 display the WCRT distribution for TS MQTT flows transmitted from the publishers of EN.A (i.e., node h1

[4][Online]. Available: http://traffic.comics.unina.it/software/ITG/

[5][Online]. Available: https://linuxconfig.org/how-to-setup-and-use-ftp-server-in-ubuntu-linux

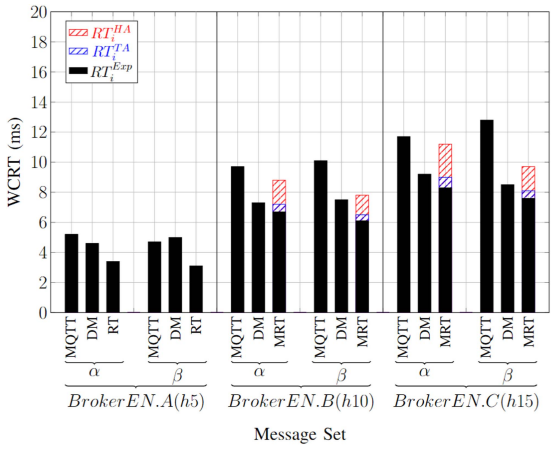

FIGURE 17. Analytical RT_i^{HA} and RT_i^{TA} versus observed RT_i^{Exp} WCRT(s) obtained using MQTT, DM-MQTT, and MRT-MQTT in load level Medium.

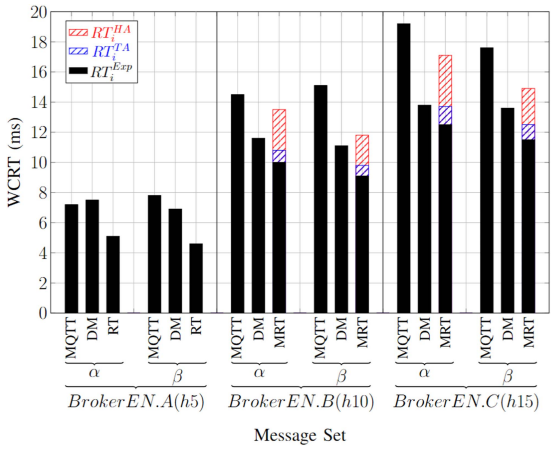

FIGURE 18. Analytical RT_i^{HA} and RT_i^{TA} versus observed RT_i^{Exp} WCRT(s) obtained using MQTT, DM-MQTT, and MRT-MQTT in load level High.

and h2 associated with topics α and β, respectively) to subscribers in all edge networks A, B, and C (i.e., nodes h5, h10, and h15, respectively), using standard MQTT, DM-MQTT, and MRT-MQTT, all in QoS level of 1 and at the three load levels. When the publishers and subscribers are all in the same network, namely `EN.A`, then MRT-MQTT reduces to RT-MQTT and the corresponding analyses apply. We include this case as a baseline to show the impact of scaling the dispersion of publishers and subscribers across multiple edge networks. For this reason, we use the new analyses in the new cases, only, i.e., with publishers and subscribers in different edge networks, for which the prior RT-MQTT analyses do not apply.

The bar charts clearly show how multicasting techniques reduce response times for MRT-MQTT and DM-MQTT with respect to standard MQTT, with MRT-MQTT exhibiting superior performance relatively to DM-MQTT for TS traffic, as expected, since it provides traffic prioritization. The effect of traffic prioritization provided by MRT-MQTT is also noticeable with β topic messages, which have higher priority than α topic messages, showing a lower latency.

For interedge traffic, DM-MQTT reserves bandwidth, which reduces response times with respect to standard MQTT. However, inside a single edge network, DM-MQTT lacks such bandwidth reservations, performing similarly to standard MQTT. Fig. 18 illustrates this point in line with our expectation, showing comparable WCRT for messages of topics α and β in both standard MQTT and DM-MQTT, without one dominating the other, when sent via path $\mathcal{L}_{AA}$ (i.e., directed to $h5$—single edge network case). For standard MQTT, the measured WCRT distribution does not reveal any consistent difference between the topics α and β, as expected, since the network treats all flows equally, regardless of their time sensitivity. This is observed across all load and distribution levels.

Regarding the WCRT analyses of MRT-MQTT when there are multiple edge networks involved, Figs. 16–18 reveal that the WCRT values computed using TA and HA are higher than those measured experimentally, aligning with expectations. TA (RT_i^{TA}) more closely approximates the experimental values (RT_i^{Exp}) than HA (RT_i^{HA}), as expected, too. The maximum deviation for TA is at most +7.3%, +8.5%, and +9.7% for the low, medium, and high load levels, respectively, while for HA, it reaches +32.7%, +34.9%, and +36.8%. As expected, the degree of pessimism in HA, compared to TA, increases with longer path lengths or lower priority, since HA allows for more improbable interference scenarios. For example, Fig. 18 shows a higher WCRT for messages of topic α compared to those of topic β, sent through the same path, due to the lower priority of α. This observation confirms the expected trend that HA's pessimism escalates for lower priority messages. Moreover, messages with the same priority traversing path $\mathcal{L}_{AC}$ (i.e., directed to $h15$) exhibit higher response times than those on path $\mathcal{L}_{AB}$ (i.e., directed to $h10$), as the former path includes an additional hop.

As seen previously, TA attains tighter schedulability bounds than HA, but this improvement is obtained at the expense of higher complexity. To confirm and quantify the impact in terms of processing time overhead, we measured the analyses average execution time over 200 iterations for each of the load scenarios. The results for HA were 262, 308, and 345 ms while for TA we achieved 964, 1140, and 1346 ms for low, medium, and high load levels, respectively. These values confirm our expectations, with HA showing an overall average execution time of just 26.5% of that taken by TA. Furthermore, the cyclomatic complexity (CC) of both algorithms was evaluated using the Lizard checking tool [68], being obtained a CC number of 4.6 for HA and 11.5 for TA, further confirming the higher complexity of TA.

Overall, these results validate our claims. First, using multicasting reduces the WCRT when publishers and subscribers are distributed across multiple edge networks. This is visible comparing MRT-MQTT with MQTT in the three load levels

and in the two cases where publishers and subscribers are placed in different edge networks (broker in h10 and in h15).

Second, using traffic prioritization MRT-MQTT is capable of segregating TS traffic and providing adequate timing QoS. This is visible in the reduction of WCRT associated to both topics α and β when compared to DM-MQTT and MQTT. It is also visible in the lower WCRT associated to α (higher priority) compared to the higher WCRT of β (lower priority).

Finally, the WCRT of MRT-MQTT is analyzable with RT approaches as opposed to DM-MQTT and MQTT. In particular, we showed how two existing analyses, namely HA and TA, can be adapted to MRT-MQTT. This is visible in the safe (higher) WCRT estimates that they generate, which are consistently above but close to the observed WCRT values in all applicable cases. This result validates the main contribution of this article, which is the adaptation of the said analysis to the MRT-MQTT framework. Moreover, the analyses based on HA and TA offer a tradeoff of accuracy against complexity. This is visible in the WCRT estimates provided by both analyses and their complexity measurements.

C. LIMITATIONS OF THE FRAMEWORK

As referred earlier in this article, MRT-MQTT deals exclusively with communication, leaving out the delays associated to the MQTT broker operation. For end-to-end operation and analysis, we need to integrate such delays. Fortunately, as explained in the related work section, there is a myriad of works on the MQTT broker RT behavior, most of which applicable to our system. Adding this component to enforce end-to-end RT behavior is left for future work.

VIII. CONCLUSION

MQTT is becoming increasingly popular in IoT and IIoT applications, however, its applicability is limited due to its inadequate support for timeliness requirements, which are crucial in RT industrial settings. To address this shortcoming, the authors have shown that combining SDN network management with the MQTT communication protocol presents a viable solution for fulfilling the stringent RT requirements of industrial applications. The authors have developed a framework named RT-MQTT that effectively integrates SDN and MQTT. This integration not only increases the flexibility of network management but also meets the RT demands prevalent in industrial environments by introducing a set of RT extensions to MQTT. Moreover, the framework is further enhanced with multicast-based connectivity, termed MRT-MQTT, paving the way for optimized network performance, greater scalability, and improved resource efficiency. For completeness, this article includes a summary of both RT-MQTT and MRT-MQTT and then introduces a novel schedulability analysis for the TS traffic in MRT-MQTT using both TA- and HA-based approaches, extended to encompass edge computing scenarios. While TA was selected for its precision, the inclusion of the simpler, yet more conservative, HA-based analysis acknowledges TA's complexity of implementation and validation. We believe this is the first WCRT analysis provided for an MQTT-based protocol. Comprehensive emulation experiments validate the effectiveness of the proposed architecture and its key attributes, as well as the proposed timing analysis. Future research will consider as follows.

1) Adding broker-backed multicasting to enhance efficiency.
2) Assessing end-to-end response times including broker processing time.
3) Comparing the whole architecture with emerging IoT technologies.
4) Exploring security features within this framework.

REFERENCES

[1] M. Xu et al., "The fourth industrial revolution: Opportunities and challenges," *Int. J. Financial Res.*, vol. 9, no. 2, pp. 90–95, 2018.

[2] Q. Wang and Y. G. Wang, "Research on power Internet of Things architecture for smart grid demand," in *Proc. IEEE 2nd Conf. Energy Internet Energy Syst. Integration*, 2018, pp. 1–9.

[3] S. Siyang, S. Lokavee, and T. Kerdcharoen, "The development of IoT-based non-obstructive monitoring system for human's sleep monitoring," in *Proc. IEEE Int. Conf. Consum. Electron.-Taiwan*, 2019, pp. 1–2.

[4] A. J. Jara, "Wearable internet: Powering personal devices with the Internet of Things capabilities," in *Proc. Int. Conf. Identification, Inf. Knowl. Internet Things*, 2014, pp. 7–12.

[5] L. Zhang, I. K. Dabipi, and W. L. Brown Jr, "Internet of things applications for agriculture," in *Internet Things A - Z: Technol. Appl.*, pp. 507–528, 2018.

[6] Y. J. Kwon and D. H. Kim, "IoT-based defect predictive manufacturing systems," in *Proc. IEEE Int. Conf. Inf. Commun. Technol. Convergence*, 2017, pp. 1067–1069.

[7] A. Massaro, G. Mastandrea, L. D'Oriano, G. R. Rana, N. Savino, and A. Galiano, "Systems for an intelligent application of automated processes in industry: A case study from "PMI IoT industry 4.0" project," in *Proc. IEEE Int. Workshop Metrol. Ind. IoT*, 2020, pp. 21–26.

[8] R. Atmoko, R. Riantini, and M. Hasin, "IoT real time data acquisition using MQTT protocol," in *Proc. J. Phys.: Conf. Ser.*, 2017, vol. 853, Art. no. 012003.

[9] J. Sommer et al., "Ethernet–a survey on its fields of application," *IEEE Commun. Surv. Tut.*, vol. 12, no. 2, pp. 263–284, Apr.–Jun., 2010.

[10] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, Jan.–Mar., 2015.

[11] O. Standard, "MQTT version 5.0," *Retrieved June*, vol. 22, 2019, Art. no. 2020.

[12] E. Shahri, P. Pedreiras, and L. Almeida, "Enhancing MQTT with real-time and reliable communication services," in *Proc. IEEE 19th Int. Conf. Ind. Informat.*, 2021, pp. 1–6.

[13] E. Shahri, P. Pedreiras, and L. Almeida, "Extending MQTT with real-time communication services based on SDN," in *Proc. Sensor Appl. Ind. Automat.*, 2022, pp. 1–6.

[14] E. Shahri, P. Pedreiras, and L. Almeida, "Response time analysis for RT-MQTT protocol grounded on SDN," in *Proc. EMSIG Workshop Next Gener. Real-Time Embedded Syst.*, Jan. 2023, pp. 5:1–5:15. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2023/17736

[15] E. Shahri, P. Pedreiras, and L. Almeida, "End-to-end response time analysis for RT-MQTT: Trajectory approach versus holistic approach," in *Proc. IEEE 19th Int. Conf. Factory Commun. Syst.*, 2023, pp. 1–8.

[16] E. Shahri, P. Pedreiras, L. Almeida, and J. Sousa, "Scalable SDN-based MQTT real-time communications for edge networks," in *Proc. IEEE 28th Int. Conf. Emerg. Technol. Factory Automat.*, 2023, pp. 1–8.

[17] J. J. Gutiérrez, J. C. Palencia, and M. Gonzalez Harbour, "Holistic schedulability analysis for multipacket messages in AFDX networks," *Real-Time Syst.*, vol. 50, no. 2, pp. 230–269, 2014.

[18] S. Martin and P. Minet, "Worst case end-to-end response times of flows scheduled with FP/FIFO," in *Proc. IEEE Int. Conf. Syst. Int. Conf. Mobile Commun. Learn. Technol.*, 2006, pp. 54–60.

[19] O. Standard, "MQTT version 5.0," 2019. Accessed: Mar. 12, 2024. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[20] "OpenFlow." Accessed: Mar. 12, 2024. [Online]. Available: https://opennetworking.org/sdn-resources/customer-case-studies/openflow/

[21] Y.-D. Lin, Y.-C. Lai, H.-Y. Teng, C.-C. Liao, and Y.-C. Kao, "Scalable multicasting with multiple shared trees in software defined networking," *J. Netw. Comput. Appl.*, vol. 78, pp. 125–133, 2017.

[22] S. Deering et al., "Protocol independent multicast (PIM): Motivation and architecture," *Work Prog.*, 1995. Accessed: Mar. 12, 2024.

[23] B. Fenner and D. Meyer, "Multicast source discovery protocol (MSDP)," Tech. Rep. 3618, 2003.

[24] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, "Fixed priority pre-emptive scheduling: An historical perspective," *Real-Time Syst.*, vol. 8, pp. 173–198, 1995.

[25] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach," *IEEE Trans. Ind. Inform.*, vol. 6, no. 4, pp. 521–533, Nov. 2010.

[26] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Applying trajectory approach with static priority queuing for improving the use of available AFDX resources," *Real-Time Syst.*, vol. 48, no. 1, pp. 101–133, 2012.

[27] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Germany: Springer, 2001.

[28] Q. Xu and X. Yang, "Performance analysis on transmission estimation for avionics real-time system using optimized network calculus," *Int. J. Aeronautical Space Sci.*, vol. 20, no. 2, pp. 506–517, 2019.

[29] T. Tachibana, T. Furuichi, and H. Mineno, "Implementing and evaluating priority control mechanism for heterogeneous remote monitoring IoT system," in *Proc. Adjunct Proc. 13th Int. Conf. Mobile Ubiquitous Syst.: Comput. Netw. Serv.*, 2016, pp. 239–244.

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

[31] Y.-S. Kim, H.-H. Lee, J.-H. Kwon, Y. S. Kim, and E.-J. Kim, "Message queue telemetry transport broker with priority support for emergency events in Internet of Things," *Sensors Mater.*, vol. 30, no. 8, pp. 1715–1721, 2018.

[32] A. N. Rosli, R. Mohamad, Y. W. M. Yusof, S. Shahbudin, and F. Y. A. Rahman, "Implementation of MQTT and LoRaWAN system for real-time environmental monitoring application," in *Proc. IEEE 10th Symp. Comput. Appl. Ind. Electron.*, 2020, pp. 287–291.

[33] H. T. Yew, M. F. Ng, S. Z. Ping, S. K. Chung, A. Chekima, and J. A. Dargham, "IoT based real-time remote patient monitoring system," in *Proc. IEEE 16th Int. Colloq. Signal Process. Appl.*, 2020, pp. 176–179.

[34] M. Zambrano 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. 149190–149198, 2020.

[35] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Proc. IEEE Asia Pacific Signal Inf. Process. Assoc. Annu. Summit Conf.*, 2012, pp. 1–8.

[36] J. Yan, H. Zhang, Q. Shuai, B. Liu, and X. Guo, "HiQoS: An SDN-based multipath QoS solution," *China Commun.*, vol. 12, no. 5, pp. 123–133, 2015.

[37] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for Qos provisioning," in *Proc. IEEE 22nd Telecommun. Forum*, 2014, pp. 111–114.

[38] S. Dwarakanathan, L. Bass, and L. Zhu, "Cloud application ha using SDN to ensure QoS," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, 2015, pp. 1003–1007.

[39] S. Sharma et al., "Implementing quality of service for the software defined networking enabled future internet," in *Proc. IEEE 3rd Eur. Workshop Softw. Defined Netw.*, 2014, pp. 49–54.

[40] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time quality of service with software defined networking," in *Proc. IEEE 3rd Int. Conf. Cloud Netw.*, 2014, pp. 70–76.

[41] R. Kumar et al., "End-to-end network delay guarantees for real-time systems using SDN," in *Proc. IEEE Real-Time Syst. Symp.*, 2017, pp. 231–242.

[42] L. Leonardi, L. L. Bello, and S. Aglianó, "Priority-based bandwidth management in virtualized software-defined networks," in *Electronics*, vol. 9, no. 6, 2020, Art. no. 1009.

[43] J.-H. Park, H.-S. Kim, and W.-T. Kim, "Dm-MQTT: An efficient MQTT based on SDN multicast for massive IoT communications," *Sensors*, vol. 18, no. 9, 2018, Art. no. 3071.

[44] F. Fontes, B. Rocha, A. Mota, P. Pedreiras, and V. Silva, "Extending MQTT-SN with real-time communication services," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2020, vol. 1, pp. 1–4.

[45] R. Tamri et al., "The MI-SDN system to manage MQTT data in an interoperable IoT wireless network," *Turkish J. Comput. Math. Educ.*, vol. 12, no. 5, pp. 1031–1036, 2021.

[46] J. Yang, K. Sandström, T. Nolte, and M. Behnam, "Data distribution service for industrial automation," in *Proc. IEEE 17th Int. Conf. Emerg. Technol. Factory Automat.*, 2012, pp. 1–8.

[47] "The zero overhead, pub/sub, store, query, and compute protocol." Accessed: Mar. 12, 2024. [Online]. Available: https://zenoh.io/

[48] N. Garg, *Apache Kafka*. Birmingham, U.K.: Packt Publishing, 2013.

[49] "IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks," in *IEEE Std. 1722-2016 (Revision of IEEE Std 1722-2011)*, pp. 1–233, 2016. doi: 10.1109/IEEESTD.2016.7782716.

[50] L. Moutinho, P. Pedreiras, and L. Almeida, "A real-time software defined networking framework for next-generation industrial networks," *IEEE Access*, vol. 7, pp. 164468–164479, 2019.

[51] "What's ryu?." Accessed: Mar. 12, 2024. [Online]. Available: https://ryu-sdn.org/

[52] "ovsdb." Accessed: Mar. 12, 2024. [Online]. Available: https://docs.openvswitch.org/en/latest/ref/ovsdb.7/

[53] "Open vSwitch advanced features," 2016. Accessed: Mar. 12, 2024. [Online]. Available: http://docs.openvswitch.org/en/latest/tutorials/ovs-advanced/

[54] B. Awerbuch, "A new distributed depth-first-search algorithm," *Inf. Process. Lett.*, vol. 20, no. 3, pp. 147–150, 1985.

[55] M. Hua and J. Pei, "Probabilistic path queries in road networks: Traffic uncertainty aware path selection," in *Proc. 13th Int. Conf. Extending Database Technol.*, 2010, pp. 347–358.

[56] A. Verma and N. Bhardwaj, "A review on routing information protocol (RIP) and open shortest path first (OSPF) routing protocol," *Int. J. Future Gener. Commun. Netw.*, vol. 9, no. 4, pp. 161–170, 2016.

[57] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[58] H.-s. Kim, S. Yun, H. Kim, and W.-T. Kim, "A novel SDN multicast for large-scale IoT environments," in *Proc. IEEE Int. Conf. Inf. Commun. Technol. Convergence*, 2017, pp. 823–828.

[59] Y. J. Kwon and D. H. Kim, "IoT-based defect predictive manufacturing systems," in *Proc. IEEE Int. Conf. Inf. Commun. Technol. Convergence*, 2017, pp. 1067–1069, doi: 10.1109/ICTC.8190856.

[60] A. Massaro, G. Mastandrea, L. D'Oriano, G. R. Rana, N. Savino, and A. Galiano, "Systems for an intelligent application of automated processes in industry: A case study from "PMI IoT industry 4.0" project," in *Proc. IEEE Int. Workshop Metrol. Ind. 4.0 IoT*, 2020, pp. 21–26, doi: 10.1109/MetroInd4.0IoT48571.9138231.

[61] "Tcpdump & libpcap." Accessed: Mar. 12, 2024. [Online]. Available: https://www.tcpdump.org/

[62] T. Zhang and B. Liu, "Exposing end-to-end delay in software-defined networking," *Int. J. Reconfigurable Comput.*, vol. 2019, 2019, pp. 1–12.

[63] H. M. T. Tachibana and T. Furuichi, "Implementing and evaluating priority control mechanism for heterogeneous remote monitoring IoT system," in *Proc. Adjunct Proc. 13th Int. Conf. Mobile Ubiquitous Syst.: Comput. Netw. Serv.*, Hiroshima, Japan, 2016, pp. 239–244.

[64] C. O. Seongjin Kim, "A study on method for message processing by priority in MQTT broker," *JKIICE- J. Korea Inst. Inf. Commun. Eng.*, Jul. 2017.

[65] R. I. Davis and A. Burns, "Response time upper bounds for fixed priority real-time systems," in *Proc. IEEE Real-Time Syst. Symp.*, 2008, pp. 407–418.

[66] S. Martin, P. Minet, and L. George, "The trajectory approach for the end-to-end response times with non-preemptive FP/EDF," in *Proc. Int. Conf. Softw. Eng. Res. Appl.*, Springer, 2006, pp. 229–247.

[67] R. A. Light, "Mosquitto: Server and client implementation of the MQTT protocol," *J. Open Source Softw.*, vol. 2, no. 13, 2017, pp. 265–266.

[68] T. Yin, "Lizard, a simple code complexity analyser," 2024. Accessed: Mar. 12, 2024. [Online]. Available: https://github.com/terryyin/lizard