Developing and Evaluating MQTT Connectivity for an Industrial Controller

Selma Opačin[†], Lejla Rizvanović[*], Björn Leander[*†], Saad Mubeen[*], Aida Čaušević[*‡]
[*]*Mälardalen University*,Västerås, Sweden, {saad.mubeen, aida.causevic}@mdu.se , lrc21001@student.mdu.se
[†]*ABB AB, Process Control Platform*,Västerås, Sweden, {bjorn.leander, selma.opacin}@se.abb.com
[‡]*Alstom,* Västerås, Sweden, aida.causevic@alstomgroup.com

***Abstract*—Technical advances as well as continuously evolving business demands are reshaping the need for flexible connectivity in industrial control systems. A way to achieve such needs is by using a service-oriented approach, where a connectivity service middleware provides controller as well as protocol-specific interfaces. The Message Queuing Telemetry Transport (MQTT) protocol is a widely used protocol for device-to-device communication in the Internet of Things (IoT). However it is not commonly integrated in industrial control systems. To address this gap, this paper describes the development and implementation of a prototype of a connectivity service middleware for MQTT within an industrial private control network. The prototype implementation is done in the context of an industrial controller, and used in a simulated modular automation system. Furthermore, various deployment scenarios are evaluated with respect to response time and scalability of the connectivity service.**

I. Introduction

As part of Industry 4.0, new business requirements and technical advances lead to need of increased flexibility and heterogeneity in industrial control systems [1], [2]. The need for an industrial controller to be able to consume and produce data using different communication protocols is of increasing importance, as the industrial private control networks may comprise a plethora of different vendor devices. The network infrastructure is as well becoming increasingly heterogeneous, including, e.g., private 5G networks [3] for wireless sensing and actuation. The network-centric design strategy, as described by e.g., the Open Process Automation Standard (O-PAS) [4], requires flexible connectivity scenarios. Enabling connections between different controllers in industrial control systems is the subject of many research works [5]. The industrial control system is designed as a separate environment for controlling a single process. Over time, these systems have become increasingly integrated with outside networks [6]. In the control system development projects, numerous approaches and standards are established to facilitate connection through the communication network of controllers [7], [8].

One method of providing industrial controllers with flexibility on connectivity is by separating the protocol handling and control logic into separate services, which may be independently deployed. The connectivity service works as a middleware, providing the control service with a well-known signal interface, while implementing protocol-specific communication towards the input/output (I/O) devices. Such middleware implementations already exists for several of the traditional fieldbuses such as for Modubus [9] and Profinet [10].

The emerging Internet of Things (IoT) paradigm enables connecting devices in everyday life with different applications, meeting challenging real-time requirements [11]. Sharing, processing, and storing huge amounts of data has become manageable within IoT. Message Queuing Telemetry Transport (MQTT) [12], is a lightweight messaging protocol, suitable for constrained environments, e.g., for machine-to-machine communication within the IoT. Introducing IoT devices and technologies in the industrial setting, leads to an Industrial IoT, as a way of increasing amount of available sensor data to a low cost, increasing the amount of interconnections within an industrial system [6].

In IoT, MQTT is already a well established protocol for data exchange, but so far it has not been widely deployed in industrial settings. As IoT-devices are being adopted also in industrial systems, there is a need for traditional industrial controllers to handle MQTT data. Using the middleware design strategy as outlined above, this work aims at investigating how to implement and evaluate a connectivity service for the MQTT-protocol, using a simulated modular automation system [13], [14] as an industrial use case.

Several previous works discuss different aspects of performance measurements of the MQTT-protocol, e.g., considering the MQTT broker load handling [15], Quality of Service impact on round trip delays [16], scalability [17], energy consumption [18] and latency [11]. All of these works provide insight in limits of the MQTT protocol in certain scenarios, but neither focus specifically on implementation and evaluation of a connectivity middleware enabling MQTT in an industrial setting, which is the main target of this paper.

The **main contributions** presented in this paper are as follows:

- develop and implement a prototype of an MQTT connectivity middleware within an industrial framework;
- response-time measurements in the prototype implementation for three different deployment scenarios; and
- scalability measurements of the prototype implementation.

The remainder of this paper is organized as follows. Section II and III provides details on the system architecture and prototype implementation. Section IV describes the experimental set-up evaluation results as well as the impact, shortcomings, and potential future extensions. Section V concludes the paper.

II. Proposed System Architecture

In this section we describe the design of the middleware prototype that will provide an industrial controller with MQTT connectivity. The controller is designed for industrial processes and is highly available in control applications. The different software components required by the controller are developed as stand-alone control services, which are integrated and executed using a node manager. The node manager abstracts inter-service communications, operating system functionalities, etc., which allows for a network-oriented architecture. Development of a connectivity service for the MQTT protocol, integrated in the node manager architecture, is the selected way of creating the middleware prototype.

The MQTT protocol uses an additional component for message queuing, a broker service, leading to a system architecture as depicted in Fig. 1, in which the execution service is the existing software for executing control application logic, the MQTT connectivity is the developed middleware, adding communication capabilities for the MQTT protocol. The services both execute as a part of a node manager, which contains basic services such as handling the configurations and communication with higher layers in the system.

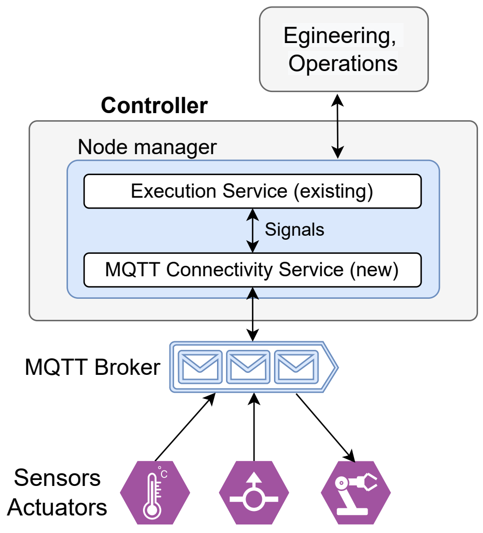


Fig. 1: System architecture.

Industrial Use Case: During development of the prototype, an industrial use case is used, which contains a distributed control system designed according to the modular automation design strategy [13]. The use case consists of numerous processing modules which can be exchanged, reconfigured, interconnected in different ways, and individually controlled. The target process for the use case is an Ice Cream Factory (simulator) as it has variability and some meaningful physical interactions and transformations in the ice cream mixture flowing through different modules. An overview of the system in the industrial use case is depicted in Fig. 2. A detailed description of the simulation environment is provided in [19]. Each of the module controllers in the system contains specific control logic handled by an execution service, and a connectivity service for handling of the simulated I/O.

The simulator is used as "software in the loop" [20] interacting with real industrial controllers. It uses a configuration file describing all simulated modules, including their physical properties, I/Os, and interconnections, to configure all instances and simulation signals. Additionally, the simulator receives data from the connectivity service via the MQTT broker and sends and writes the data to MQTT queues. All output signals are sent to the MQTT *SendQueue* which can queue messages on the MQTT broker. The clients receive the data through MQTT *ReceiveQueue* from the broker.

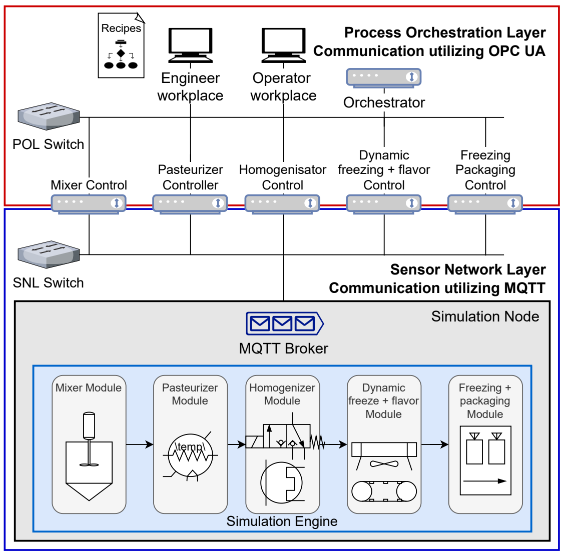


Fig. 2: Industrial Use case: Simulated ice cream factory.

III. Prototype Design and Implementation

The prototype design of the connectivity service is depicted in Fig. 1. The node manager runs the execution and connectivity services. The existing control service is also accessible via the Open Platform Communications United Architecture (OPC UA) protocol for industrial purposes. As described in Section II, the simulator performs the function of publishing and subscribing data to the MQTT broker and, as such, simulates the I/O data of sensors and actuators. The simulator consists of the data of interest published on the MQTT broker. An external input can change this data. However, it is essential to emphasize that the input data from the simulator used in this design can be replaced by the real-world input data received via one of the sensors.

The purpose of the prototype is to provide data-exchange for one module in the Ice Cream Factory, between the simulator and the execution service. The transferred data contains information about the characteristics of the module I/O signals, representing sensors and actuators. Some of these signals are input values (analog/digital), and some are output values (analog/digital). Each signal is accessible via a different MQTT topic. The root of the topic for reading data from the MQTT broker is *SimulationOut* and the root topic for writing data to the MQTT broker is *SimulationIn*. The data is published to the MQTT topics when the simulator starts its execution. For example, it is possible to access one parameter by subscribing to the topic *SimulationOut/T1/Level*. *T1* represents specified module, and *Level* represents an analog output parameter of the module *T1*. The subscribing hierarchy on the topics related to specified parameters is described in Fig. 3. As a part of the development, choosing the proper broker service to perform specified functionalities and exchange data between the simulator and the connectivity service is essential. The preferred broker is the Mosquitto[1] broker installed on devices running the simulator.

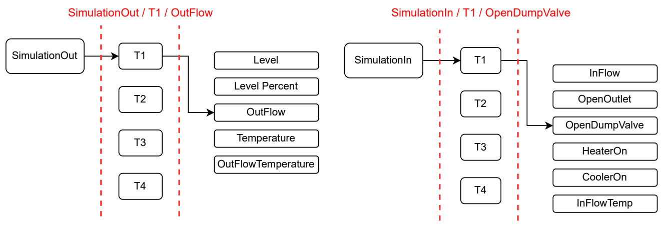


Fig. 3: Hierarchy of topics related to specified parameters.

The implementation of the connectivity service is based on changing or reading the simulator's data and storing them internally. To achieve this, functionalities such as *Read* and *Write* need to be implemented. The *Read* functionality is used to read the simulated sensors' readings while the *Write* functionality is used for writing the control signals. For this purpose, the PAHO C++ library[2] is used to easily integrate with the node manager and an existing industrial software development kit. The PAHO library provides useful functionalities that allow the connectivity service to connect and publish to the MQTT broker or subscribe and receive published messages on the topics.

An XML file is used to configure the connectivity, containing information about the broker address where the data is published, and a list of all I/O signals with their features and respective topic names. The ability to read the sensory data and write control signals has to be provided. In order to read data from the MQTT broker, it is first required to enable the connection and subscription of the formed client. When the connectivity service starts its execution, the client subscribes to all topics that are described as input signals in the configuration.

IV. Evaluation and Results

In this section, the developed prototype is experimentally evaluated. The main idea of the evaluation is to investigate and measure the developed system's response time for three different deployment scenarios, using different signal loads to assess scalability properties of the system.

A. Deployment scenarios

In the evaluations, three deployment scenarios of the connectivity service are considered within the industrial use case. The first scenario uses a single-switch setup, depicted in Fig. 4a, with the network consisting of two nodes. The simulated I/O and the MQTT broker are running on one node, and the Control Service, together with the MQTT connectivity, is running on the other, representing a typical small system set-up. In the second scenario, in Fig. 4b, the MQTT broker is hosted as a cloud service, useful, e.g., for a sensors which are geographically distributed over a large area [21]. This is a kind of worst-case deployment scenario with regards to network delay, as the number of hops between the nodes and the broker is unknown.

[1]https://mosquitto.org/
[2]https://github.com/eclipse/paho.mqtt.cpp

In the third scenario, Fig. 4c, all components run on a single node, representing the best-case with regards to network delay. This scenario represents a traditional controller-centric setup, with the I/O bus connected to the same embedded device as the controller.

B. Response time and scalability measurements

In order to measure the response time of the connectivity service prototype, a special loop-back software is developed, called the MQTT *Stimulator*, to provide a functionality representing wiring a digital output signal to a digital input signal, see Fig. 5. The Stimulator subscribes to the digital LoopBack signal, and as soon as a changed value is detected, that value is set to the SetOut signal. The Stimulator can also be configured to publish signal-data, every 100 ms, a defined set of signals are changed and published. This functionality allows to scale up the number of signals handled by the broker and connectivity.

Using these mechanisms, the time between the connectivity service writes a value to the LoopBack signal until that value is detected on the SetOut signal and can be measured, representing the round-trip from the connectivity service, through MQTT Broker, to the Stimulator and back again. For each combination of deployment scenarios and number of messages, the response time is measured 100 times. After measurement, the average response time and standard deviation for all scenarios are calculated.

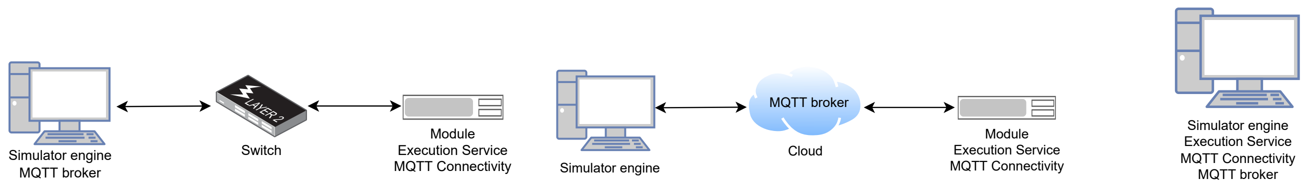
C. Results

For the first deployment scenario, Fig. 4a, response time measurement is repeated for the single-switch network, from the stimulator to the connectivity service. Analyzed numbers of subscribed signals in these experiments are 1, 100, 500, 700, and 1000, and each message is published each $100ms$. Fig. 6a summarizes the results gathered in this experiment. The horizontal axis represents the number of messages per $100ms$, and the vertical axis represents the average round-trip response time.

In the second scenario, the messages are transferred through a cloud-based MQTT broker. Analyzed number of subscribed signals in this scenario are 1, 100, 500, 600, 700, 800, and 1000. Fig. 6b shows results gathered through this experiment. In the third scenario, with all components deployed on a single node, the response time is evaluated using 1, 100, 500, 700, 800, and 1000 subscribed signals. As a part of the evaluation process, the standard deviation (σ) for each measured response time is also calculated, see Table I.

Messages/$100ms$	1	100	500	700	800	1000
σ scenario 1	28.8	31.5	22.3	28.6	N/A	69.0
σ scenario 2	50.6	47.5	132.0	153.2	4065	5054
σ scenario 3	15.8	10.6	9.8	10.4	19.9	321.6

TABLE I: Standard deviation (σ) of end-to-end average response times.



(a) Single-hop industrial private network. (b) Cloud broker. (c) Single node.

Fig. 4: Three experimental deployment scenarios.

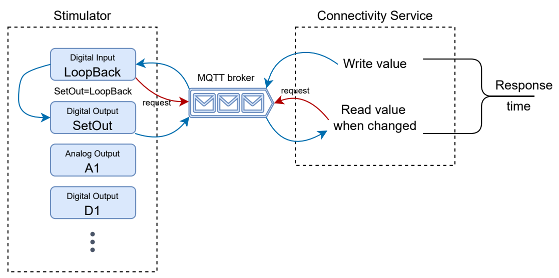


Fig. 5: The structure and idea of the Stimulator.

D. Discussion

The results from the experiments show that the prototype scales rather well up to approximately 5000 handled messages per second, until that point all three scenarios have similar average response times. For the scenario of the connectivity handling I/O for a single controller, this response time is sufficient in most cases. As expected, the different deployment scenarios exhibit different average delays due to the different network setup, and consequently control loop execution times must be adapted to these delays. The experiments do not reveal what component is causing the response time to increase when reaching approximately 8000-10000 messages per second. The fact that the single-switch scenario performs better than the single-node scenario for high-load experiments could indicate that the internal CPU load is the limiting factor.

The execution time of the *Execution* task in the connectivity service is $50ms$, and it is expected that the response time is in the range of the execution time of the connectivity service with added component delays, including the network delay.

In the scenario using a cloud-hosted broker, an average response time of between $300ms$ and $600ms$ can be acceptable, but the extreme jumps to above 5000ms for 8000 messages per second is above what would be generally acceptable for an industrial system. As mentioned, for single-controller scenarios that high load would not be required, but assuming that we have more complex system with several controllers and connectivity services using the same MQTT broker, the load on the broker could easily reach these levels. As the scale of the delays in this scenario is 50-100 times higher than for the others, the network, or combination of network and broker, is the likely bottle-neck.

Looking at the standard deviations, which is a measure of experienced jitter of the message, a similar image is painted, with a deviation of between $10ms$ and $20ms$ for the lower load experiments scenarios 1 and 3. For scenario 2, the deviation is higher, and also increases much faster, probably due to accumulated jitter in intermediate nodes.

Nevertheless, there are several limitations of the proposed prototype, which calls for further investigation as follows.

Real sensors usage: The simulator simulates signals of sensors and actuators. It would be more realistic to analyze connectivity between the controller and real I/O data from the industrial systems. Data from the industrial system can be published to the MQTT broker and transferred to the controller. For future work, complete connectivity of the controller with the I/O could be implemented.

Usage of other protocols: Messaging protocols other than MQTT could be utilized in the prototype. The prototype developed in this way could be reusable in many different industrial use cases. Therefore, more resources would need to be invested in further development in order to achieve a more generic prototype design solution.

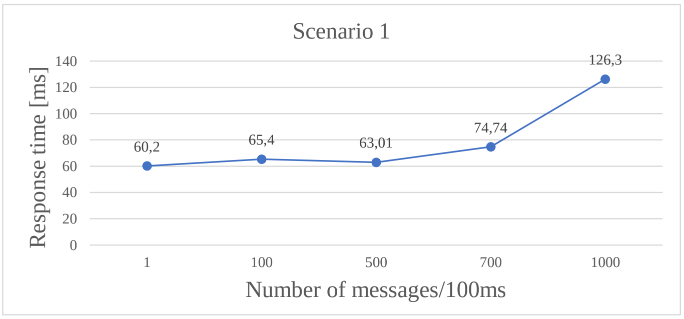
Characteristic measurements: The evaluation focused on response times and scalability. The deployment scenarios for the prototype can be evaluated using other important parameters like jitter.

Quality of Service: Another extension of the experimental evaluation could comprise analyzing how MQTT QoS affects the response times. In the developed prototype, $QoS = 1$ is used. This means, the designed prototype always has the information that the message is received from the MQTT broker. Different response times are expected with different QoS values due to the different amount of information exchanged between them.

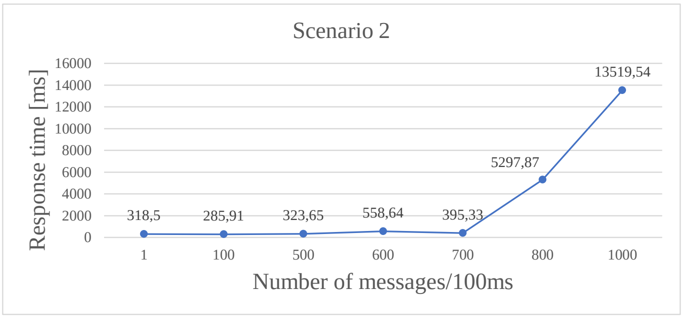
Applicability: The provided implementation and experimentation are done in the scope of a specific use case. The results are clearly limited to the implementation, but the implementation is applicable in a multitude of scenarios, not limited to that of a modular automation system.

V. Conclusions

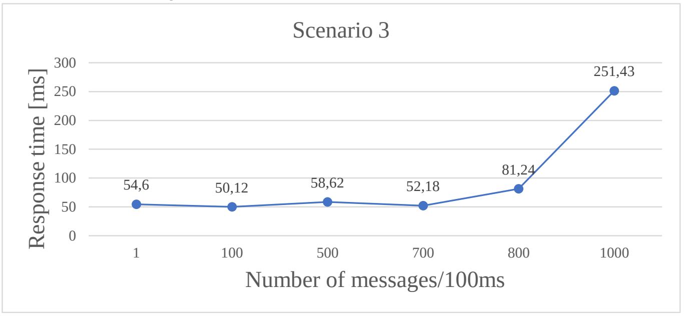
This paper presented the development and implementation of an MQTT connectivity middleware prototype. The prototype is used for connectivity between the control and simulation environments within an industrial automation framework. The prototype is evaluated using three different deployment scenarios: (i) deployment in a single node, (ii) deployment in a single-hop local network, and (iii) deployment in a cloud broker. The parameters of interest in these evaluations include response times and scalability of the connectivity service. The evaluation results show that the prototype scales well for industrial use-case scenarios with up to 5000 messages/second, with measured response times within a satisfactory range.



(a) An average end-to-end response time of the connectivity service using two nodes with a single-switch network.



(b) An average end-to-end response time of the connectivity service using two nodes with a cloud-based broker.



(c) An average end-to-end response time of the connectivity service using a single node.

Fig. 6: Evaluation results.

Acknowledgements

This work is supported by ABB AB; the industrial postgraduate school Automation Region Research Academy (ARRAY), funded by The Knowledge Foundation; and the European Union's Horizon 2020 ECSEL JU project InSecTT under grant agreement No 876038. The document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

References

[1] S. Mumtaz *et al.*, "Massive Internet of Things for Industrial Applications: Addressing Wireless IIoT Connectivity Challenges and Ecosystem Fragmentation," *IEEE Ind. Elec. Magazine*, vol. 11, no. 1, 2017.

[2] A. Kusiak, "Service manufacturing: Basic concepts and technologies," *Journal of Manufacturing Systems*, vol. 52, pp. 198–204, 2019.

[3] A. Aijaz, "Private 5G: The Future of Industrial Wireless," *IEEE Industrial Electronics Magazine*, vol. 14, no. 4, pp. 136–145, 2020.

[4] "O-PAS Standard, Version 2.0: Part 1 – Technical Architecture Overview," Open Group Preliminary Standard (P201-1), The Open Group, February 2020.

[5] B. Galloway and G. P. Hancke, "Introduction to industrial control networks," *IEEE Communications surveys and tutorials*, vol. 15, no. 2, pp. 860–880, 2012.

[6] T. John, P. De Vaere, C. Schutijser, A. Perrig, and D. Hausheer, "Linc: low-cost inter-domain connectivity for industrial systems," in *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, 2021.

[7] C. E. Pereira and P. Neumann, *Industrial Communication Protocols*, pp. 981–999. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[8] S. Vitturi, C. Zunino, and T. Sauter, "Industrial communication systems and their future challenges: Next-generation ethernet, iiot, and 5g," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 944–961, 2019.

[9] A. Swales *et al.*, "Open modbus/tcp specification," *Schneider Electric*, vol. 29, pp. 3–19, 1999.

[10] PI Organisation, "PROFINET." https://www.profibus.com. Online; Accessed: 2019-03-19.

[11] D. R. Silva, G. M. Oliveira, I. Silva, P. Ferrari, and E. Sisinni, "Latency evaluation for mqtt and websocket protocols: an industry 4.0 perspective," in *IEEE Symposium on Computers and Communications*, 2018.

[12] "MQTT Version 5.0," OASIS Standard, March 2019. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta.

[13] J. Ladiges, A. Fay, T. Holm, U. Hempen, L. Urbas, M. Obst, and T. Albers, "Integration of modular process units into process control systems," *IEEE Transactions on Industry Applications*, vol. 54, no. 2, pp. 1870–1880, 2018.

[14] M. Hoernicke *et al.*, "Automation architecture and engineering for modular process plants - Approach and industrial pilot application," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 8255–8260, 2020.

[15] R. Banno, K. Ohsawa, Y. Kitagawa, T. Takada, and T. Yoshizawa, "Measuring performance of mqtt v5.0 brokers with mqttloader," in *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–2, 2021.

[16] F. Chen, Y. Huang, J. Zhu, S. Gao, Z. Sui, and M. Duan, "Measurement and analysis of network data based on mqtt protocol," in *IEEE 20th International Conference on Communication Technology (ICCT)*, 2020.

[17] W. Pipatsakulroj, V. Visoottiviseth, and R. Takano, "mumq: A lightweight and scalable mqtt broker," in *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2017.

[18] C. Bayılmış, M. A. Ebleme, Ü. Çavuşoğlu, K. Küçük, and A. Sevin, "A survey on communication protocols and performance evaluations for Internet of Things," *Digital Communications and Networks*, vol. 8, no. 6, pp. 1094–1104, 2022.

[19] B. Leander, T. Marković, A. Čaušević, T. Lindström, H. Hansson, and S. Punnekkat, "Simulation environment for modular automation systems," in *IECON 2022–48th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1–6, IEEE, 2022.

[20] S. Demers, P. Gopalakrishnan, and L. Kant, "A generic solution to software-in-the-loop," in *MILCOM IEEE Military Communications Conference*, pp. 1–6, 2007.

[21] A. Antonić, M. Marjanović, P. Skočir, and I. P. Žarko, "Comparison of the cupus middleware and mqtt protocol for smart city services," in *2015 13th International Conference on Telecommunications (ConTEL)*, pp. 1–8, IEEE, 2015.