MQTT-CB: Cloud Based Intelligent MQTT Protocol

Muhammed Raşit Erol[1(✉)], Tuğçe Bilen[2], Mehmet Özdem[2], and Berk Canberk[1,3]

[1] Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey
{erolm15,b.canberk}@itu.edu.tr

[2] Innovation and Product and Service Development Directorate, Türk Telekom, Istanbul, Turkey
{tugce.bilen,mehmet.ozdem}@turktelekom.com.tr

[3] School of Computing, Engineering and the Build Environment, Edinburgh Napier University, Edinburgh, UK
b.canberk@napier.ac.uk

Abstract. The MQTT protocol, which stands for Message Queuing Telemetry Transport, is widely recognized within the IoT community as one of the most frequently utilized communication protocols. Conventional MQTT protocols described in the literature could improve their capacity to support distributed environments and scalability. In this manner, MQTT-ST is an advanced MQTT protocol that provides bridging capabilities within a distributed environment, making it a preferred option for IoT systems. In this study, we introduce a new, intelligent, scalable, and distributed MQTT-ST-based protocol named MQTT-CB. Our approach leverages containers to improve portability, and our protocol is designed with a cloud-based architecture that streamlines deployment. The primary contribution of our research is the integration of intelligent capabilities into the MQTT-ST protocol, utilizing an LSTM (Long Short-Term Memory) network, which is the leading deep learning model. Specifically, our protocol employs predictive algorithms to foresee retransmitted packets, dynamically adjusts the number of brokers in real-time, and reduces the number of brokers when clients are inactive. Our experiments demonstrate that our protocol significantly outperforms conventional MQTT-ST protocol regarding latency between subscribers and publishers. Furthermore, our protocol adapts seamlessly to changes in the publication rate. In summary, we present a cloud-based intelligent MQTT protocol that offers significant advantages over traditional MQTT-ST protocol.

Keywords: MQTT · MQTT-ST · IoT · LSTM · Cloud · Container · AI

1 Introduction

The MQTT protocol is commonly utilized within the IoT (Internet of Things) ecosystem, as noted in [1]. MQTT utilizes a publish/subscribe-based

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2023
Published by Springer Nature Switzerland AG 2023. All Rights Reserved
N.-S. Vo and H.-A. Tran (Eds.): INISCOM 2023, LNICST 531, pp. 236–254, 2023.
https://doi.org/10.1007/978-3-031-47359-3_18

communication model, which enables efficient message transfer between clients and servers. This lightweight and efficient protocol is especially suitable for IoT devices with limited resources. The publish/subscribe model eliminates the need for direct communication between devices, which reduces the burden on individual devices and enables effective communication with a broker [2]. Due to its dependability and adaptability, MQTT is extensively employed in IoT implementations like smart cities, intelligent devices, and transportation systems, as referenced in [3].

MQTT is well-suited for devices with limited resources; however, it experiences a single point of failure with its brokers, as mentioned in [4]. If the broker becomes unavailable, the entire network may become disconnected, and devices may not be able to exchange messages. Therefore, the centralized broker is a significant drawback for MQTT. Additionally, managing the number of devices and messages can present difficulties in terms of scaling the quantity of brokers, as stated in [5]. Careful consideration of these and other factors is vital to ensure the success of a large-scale MQTT deployment.

Several MQTT protocols [6–8] support distributed brokers and use multiple brokers to provide redundancy. However, even with multiple brokers, the network may still have a fault issue, depending on the configuration and deployment strategy. Clustering or load balancing technologies [9] can be used further to reduce the danger of a single point of failure, but this may add additional complexity to the system.

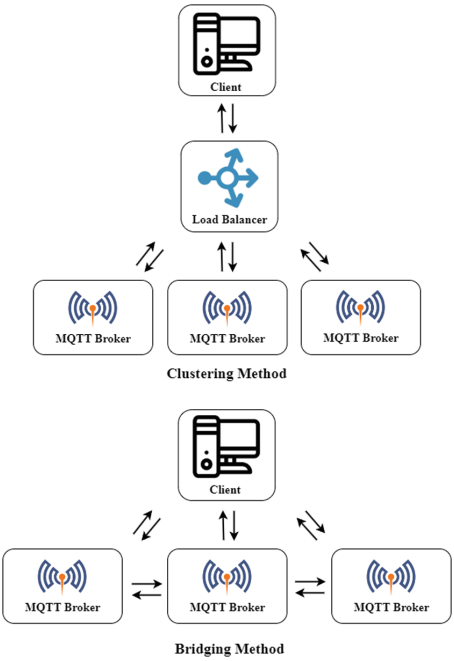

Fig. 1. Clustering and bridging methods.

The MQTT-ST project, an open-source MQTT protocol based on the widely used Mosquito implementation [10], supports distribution, as noted in [11]. Unlike several other MQTT protocols, such as [12–14], MQTT-ST does not sustain broker clustering but uses bridging to connect multiple brokers. Clustering refers to the collaboration of several brokers to establish a unified broker, whereas bridging enables the interchange of messages between brokers. Figure 1 illustrates the clustering and bridging approaches. According to [15], the bridging method is more effective than clustering regarding network traffic and broker resource usage. Consequently, MQTT-ST is well-suited for utilization in resource-limited and cost-conscious IoT settings.

The MQTT-ST protocol is a tree-based, distributed protocol that replicates messages across all brokers and can react to failures. It is an upgraded version of the Mosquitto source code and is designed for efficient message distribution, making it suitable for scaling the number of brokers. However, MQTT-ST is not a dynamic protocol, which means that it cannot increase or decrease the number of brokers in real-time, leading to resource consumption when IoT publishers are idle. Our proposed MQTT-CB protocol offers a dynamic, intelligent, cloud-based solution built on MQTT-ST to address this limitation.

MQTT-CB protocol can adapt to environmental conditions and operate optimally regarding resource consumption and network traffic overhead. It utilizes the LSTM model to predict retransmitted packets and adjusts the number of brokers accordingly. This makes the MQTT-CB protocol dynamic and intelligent, unlike other MQTT protocols. Additionally, MQTT-CB runs on Docker containers, making it cloud-based and highly portable compared to MQTT-ST and other MQTT protocols. Overall, the proposed MQTT-CB protocol offers a more efficient, adaptable, and portable solution for message distribution in IoT environments.

In summary, this paper's primary contributions are as follows:

- We propose a new MQTT-ST-based protocol that is intelligent and scalable in terms of broker numbers. While MQTT-ST is an open-source MQTT protocol, it lacks dynamic scalability by default. However, the proposed MQTT-CB protocol provides this feature by predicting the following retransmitted packets and adjusting the number of brokers in the future.
- The MQTT-CB protocol is a cloud-based MQTT protocol that can be easily deployed to the cloud without any burden. This makes it more flexible and portable in terms of deployment.
- We design the MQTT-CB protocol to run on Docker containers and orchestrate other container-based brokers using Docker CLI. This makes it more environment dependent and reusable in terms of development and simulation.
- We create a new system model with four modules to monitor MQTT packets and predict future retransmitted packets. This model is a layered architecture, making it a robust structure.
- We can improve the proposed system model in the future with more complicated deep learning methods to predict with higher accuracy and speed.

The remaining sections of this paper are organized as follows: Sect. 2 covers the related works. Section 3 describes the network architecture, while Sect. 4 outlines the system model. The simulation environment is detailed in Sect. 5, and we present our proposed model's performance evaluation in Sect. 6. Lastly, Sect. 7 concludes our work by summarizing its accomplishments.

2 Related Work

In literature, several works on distributed MQTT protocols employ bridging techniques [16,17]. The Mosquitto MQTT protocol has a Java-based implementation called D-MQTT, as cited in [16]. D-MQTT utilizes bridging and provides a distributed MQTT protocol that outperforms the Mosquito MQTT protocol regarding publication traffic exchanged between the brokers. Another MQTT bridge-based protocol, EMMA [17], is a QoS-aware MQTT protocol that enables edge computing. This work shows that communication between closely located devices has low latency and offers low-cost message transferring.

Additionally, there are several container-based MQTT implementations in the literature [18–20]. The work in [18] presents a microservice-based MQTT protocol in an edge computing environment with containers, where multiple brokers perform better in terms of throughput and multiple nodes work better than a single broker. Another work on container-based MQTT protocols, [19], provides dynamic MQTT deployment with containers, increasing resource utilization in CPU and memory. Moreover, [20] is a container-based MQTT protocol implements fog service as multiple containers. Their experimental results show that migrating container-based fog services performs better than no migration in terms of QoS.

Cloud-based MQTT protocols are also present in the literature [21,22]. FogMQ [21] provides a cloud-based MQTT protocol that uses bridging and offers self-deployment auto-migration in the cloud, achieving low latency in their work. Another cloud-based MQTT approach, [22], reduces processing message time in burst mode compared to the default algorithm in MQTT brokers.

Several works have also been on prediction using RNN models [23,24]. In [23], the authors forecast QoS in terms of loss rate, speed of the link, throughput, and RTT concerning user locations using several deep learning models with RNNs, achieving prediction accuracy above 90% in QoS status. Another work, [24], predicts network traffic in the future using time series network data. In terms of precision, their suggested model surpasses conventional RNN models.

However, these works do not provide an intelligent MQTT broker to adjust broker numbers in real-time. Our proposed model can be deployed in a cloud environment, making it more reachable and robust in terms of connectivity. Furthermore, our container-based model makes our work more portable. Additionally, adjusting broker numbers reduces resource consumption in terms of CPU power and memory.

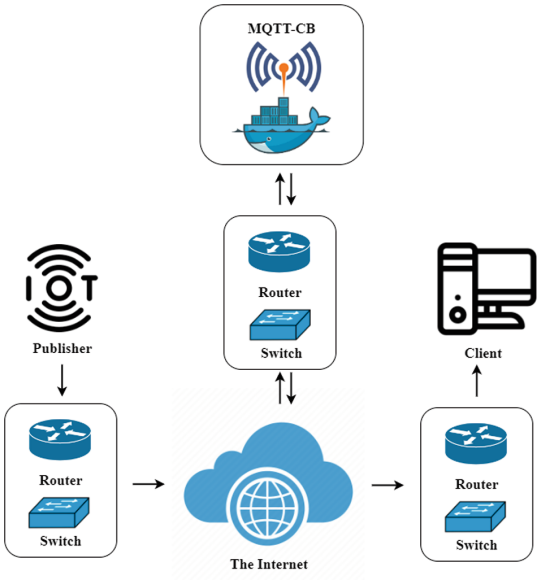

Fig. 2. Network Architecture.

3 Network Architecture

Our network architecture is founded on MQTT protocol and comprises three primary components: publishers and subscribers, serving as clients, and brokers. The brokers are responsible for collecting data from the publishers in the form of topics and subsequently delivering them to the subscribers. As a result, the brokers facilitate the data transfer between publishers and subscribers, starting from publishers to subscribers.

Each component, including brokers, publishers, and subscribers, situate on a local network that consists of a switch and a router. Therefore, our network topology encompasses three local networks, namely LN1, LN2, and LN3. LN1 comprises publishers, while LN2 consists of subscribers. The brokers are placed in LN3. The data flow between LN1 and LN3 is unidirectional, starting from LN1 to LN3. Conversely, the data flow between LN2 and LN3 is bidirectional. Additionally, we parameterize the delay between the local networks.

Each local network connects to the Internet through its router. Furthermore, each switch in the local network distributes packets among its connected devices. We also parameterize the number of publishers and subscribers. Furthermore, our model dynamically decides the number of brokers in real-time. We illustrate our network topology in Fig. 2.

4 System Model

The system model presented in this study illustrates our proposed MQTT-CB protocol, a virtual and software-based protocol implemented on the broker's side. Clients perceive the brokers as a single MQTT-CB broker, while the existing system may consist of multiple brokers orchestrated using Docker CLI.

The proposed system model comprises four primary modules, namely the Monitoring Module, Classification Module, AI Module, and MQTT-CB Controller Module, each of which performs a unique task independently of the other modules. This design approach results in a more robust system. The layered system model is depicted in Fig. 3.

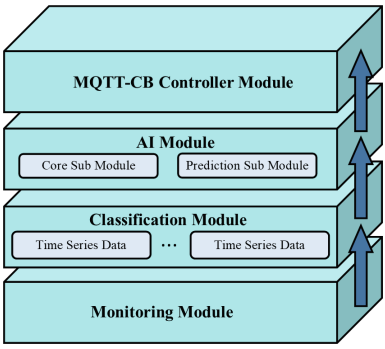

Fig. 3. The layered system model.

4.1 Monitoring Module

The Monitoring Module monitors each client and stores the resulting network traffic data in a database. This module ensures that monitoring is performed for each broker and that time-series data is persisted in a database. To achieve this, we employ the Tshark tool [25], a Linux-based command-line version of Wireshark, which enables us to monitor and analyze network traffic in a software-based manner. This approach provides us with the required capabilities for network monitoring while ensuring the system's scalability and flexibility. Furthermore, we utilize SQLite [26] as the database engine to store the monitored data. SQLite is a lightweight, fast, and stable database engine that is well-suited for our model's requirements.

4.2 Classification Module

The Classification Module of our proposed MQTT-CB protocol is tasked with classifying the monitored data that is stored in the database. This module is

implemented using Python 3 programming language and reads the database that contains the monitored data. Upon reading the database, the module reduces the network data every five minutes into a single time series data. The time series data consists of the average delay, the average retransmitted packets and the total packet number. Once the classification process is complete, the classified data is transformed into a shape compatible with the AI module's requirements.

4.3 AI Module

The AI Module is designed to learn from the classified time series data and generate predictions about future network traffic patterns. An LSTM model is employed within the module to forecast the number of retransmitted packets within the upcoming five-minute periods. Deep learning techniques are utilized in this module to provide a more intelligent and scalable MQTT protocol for the IoT domain. By leveraging the power of artificial neural networks, the proposed model can identify complex patterns and make accurate predictions about network traffic, resulting in a more efficient and reliable communication system for IoT devices.

We have a scalable MQTT protocol in terms of broker numbers. To do that, we have to predict upcoming retransmitted packets in the future. The problem is that the number of retransmitted packets are dependent sequences, complicating the prediction. However, Recurrent Neural Network (RNN) solves this problem according to [27]. In that work, they define RNN as an artificial neural network (ANN) that uses time series to make predictions. Hence, the RNN model can predict retransmitted packets in the future. However, it is hard to train RNNs to solve issues that need learning long-term time series because the gradient of the loss function decreases fast when time passes, according to [28]. Thus, we prefer to use a more sophisticated RNN implementation called LSTM in terms of memory usage in our AI Module.

The LSTM is also suitable for time series data, and its main advantage over the RNN is that it can maintain information in its memory for a more extended period, as explained in [29]. LSTM models work like RNNs in that they can remember previous information and use it to process current inputs. However, due to the vanishing gradient situation, RNNs need to retain long-term dependencies. On the other hand, the LSTM model is explicitly designed to avoid long-term dependence problems. As noted in [30], LSTM networks construct on the capabilities of RNNs to expand memory and can serve as the core blocks for an RNN's layers. Therefore, the LSTM model is more suitable for making predictions regarding memory usage and performance, as indicated by [29,30].

In our LSTM model, we employ an autoregressive approach similar to that described in [31]. An autoregressive network is a neural network that generates the value of one sample by relying on previously generated samples. This approach involves making one prediction at a time and feeding the output back into the model. The mechanics of the autoregressive LSTM are illustrated in Fig. 4.

In summary, the AI Module incorporates an LSTM model to make predictions about retransmitted packets in the future, treating them as time series data. The

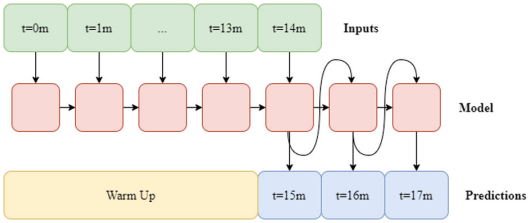

Fig. 4. Auto Regressive LSTM Model.

module comprises two sub-modules: the Core and Prediction Sub Modules, which facilitate these operations together.

Core Sub Module. The Core Sub Module plays a crucial role in our system by compiling and re-generating the LSTM model at pre-defined periods. However, since there is no input for the number of retransmitted packets when the model is run for the first time, it operates at a reduced capacity during the warm-up phase, which lasts for the first three hours after the module is booted up. During this time, the Monitoring Module collects the required network data while a single MQTT-CB broker is used. After the warm-up phase, the Core Sub Module generates the first LSTM model and makes its initial prediction.

After the first prediction, the Core Sub Module compiles and fits the LSTM model with new time series data every five minutes. The user can modify the period configuration as needed, initially set to five minutes in the default settings. The module performs this recompilation operation on the first day at five-minute intervals. After the first day, the periods are changed to one day, and the module recompiles the LSTM model daily.

The Core Sub Module of the AI Module generates predictions of the number of retransmitted packets for three-time steps ahead, which corresponds to fifteen minutes into the future. The predicted average retransmission number is then forwarded to the Prediction Sub Module for further processing.

Prediction Sub Module. The Prediction Sub Module predicts the following retransmitted packet number fifteen minutes later. We choose this value to give time to MQTT-CB Controller to run new brokers or stop the existing ones. The Prediction Sub Module makes its decisions as follows. If the percentage of predicted retransmitted packets fifteen minutes later is higher than ten percent, then the LSTM model decides that create a new broker. We calculate the retransmission percentage RP_t as in the Eq. 1.

$$R_t/P_t = RP_t(\%) \tag{1}$$

where, R_t refers to the number of retransmitted packets in a specific time slot t, while P_t denotes the total number of packets transmitted in the same time slot.

The Prediction Sub Module also checks the percentage of retransmitted packets in periods. Suppose the retransmitted packets are below one percent of the total number of packets in that period, which happens three times. In that case, the module passes the decision of shutting down a broker to the MQTT-CB Controller.

4.4 MQTT-CB Controller Module

The MQTT-CB Controller Module serves as an orchestrator for all MQTT-ST brokers. It is a Python 3-based software that manages the creation and deletion of brokers in response to predictions made by the AI Module. The MQTT-CB Controller leverages the Docker CLI to perform these operations. This module is a master, with all brokers operating as its slaves. The MQTT-CB Controller monitors the brokers' health and automatically creates new brokers if any go down. In summary, the MQTT-CB Controller Module is responsible for creating, deleting, and managing the brokers in order to maintain the overall health of the MQTT-ST system.

5 Simulation Environment

To simulate our MQTT-CB and MQTT-ST models in our experiments, we utilized the BORDER framework [15]. This framework places each client and broker in separate Docker containers, providing an isolated environment for each container using Docker Engine, container-based technology that provides a command line interface (CLI) to control or interact with each Docker container. The framework also relies on the Containernet network emulator, developed on Mininet, providing a variety of network components in the Docker containers for simulation. This emulator is container-based, providing an isolated environment in terms of computer resources and enabling a resource-independent test environment. Additionally, the Timing Compensated High-Speed Optical Link (TCLink) was used between each local network, which provides adjustable link configurations in terms of delay and bandwidth, thus providing a configurable network environment with Docker containers.

We have two simulation environments for our evaluations, one for MQTT-ST and one for our model MQTT-CB, inspired by the BORDER framework. In contrast to the BORDER framework, we provide a test environment where the number of brokers is dynamically changeable. We simulate MQTT-ST without changing the number of clients and brokers because it is not a dynamic MQTT protocol adapting to changing environmental conditions. We conduct these tests in the first simulation environment. The second test environment simulates our model MQTT-CB, a software-based upper layer on all brokers, as shown in Fig. 2. MQTT-CB has developed using Python 3, orchestrating the brokers using Docker CLI.

In our experiments, the MQTT-CB controller module is responsible for deciding whether a new broker is required. Based on the decision made by the controller, a new broker is created and launched using Docker CLI. In addition,

Table 1. Simulation Parameters.

Parameter	Value
MQTT-ST broker number	2
MQTT-CB broker number	starts with 2
RAM limit of brokers	2 GB
Number of CPU for each broker	1
Delay between router	10 ms
Delay between router and switch	0 ms
Delay between switch and client	0 ms
Subscriber number	200
Publisher number	10
Packet rate	5 msg/s
Packet size	20 bytes
Quality of Service (QoS)	1

the MQTT-CB controller can delete existing brokers, as determined by the prediction model. As a result, unlike MQTT-ST, the simulation environment for MQTT-CB is dynamic, with varying numbers of clients and brokers. The QoS (Quality of Service) level of the MQTT protocol is set to one, which is referred to at least once. This QoS level ensures that a packet is delivered to the receiver at least once, with the sender storing the message until it gets an ACK packet from the recipient. A packet can be sent or delivered multiple times, resulting in packet retransmission in the MQTT protocol. We provide a detailed overview of the simulation parameters in Table 1, which are the default values we adjust during the experiments to evaluate the protocol's behavior. Performance evaluation section contains additional information regarding these adjustments.

6 Performance Evaluation

In this study, we employ three types of evaluation strategies. Firstly, we conduct a real-time comparison of MQTT-ST and MQTT-CB to observe the running mechanism of our model and the delay between the subscribers and publishers. Secondly, we calculate the percentage of retransmitted packets in real-time to evaluate the prediction accuracy of our model. Lastly, we compare the overall delay of the two protocols over a certain amount of time.

We use a pre-trained model warmed up for three hours for all evaluations to collect necessary network data. As a result, all-time representations in the evaluation figures start after the initial three-hour running period. In all figures, we define the delay as the passing time between the publishers and subscribers.

We conduct three experiments with different publication rates using the evaluation mentioned earlier strategies. In the first experiment, we keep the publication rate constant for MQTT-ST and MQTT-CB, as specified in Table 1. In the

second experiment, we increase the publication rate after a while and observe the behavior of our model. Lastly, in the third experiment, we evaluate the performance of our model as the publication rate decreased in real-time.

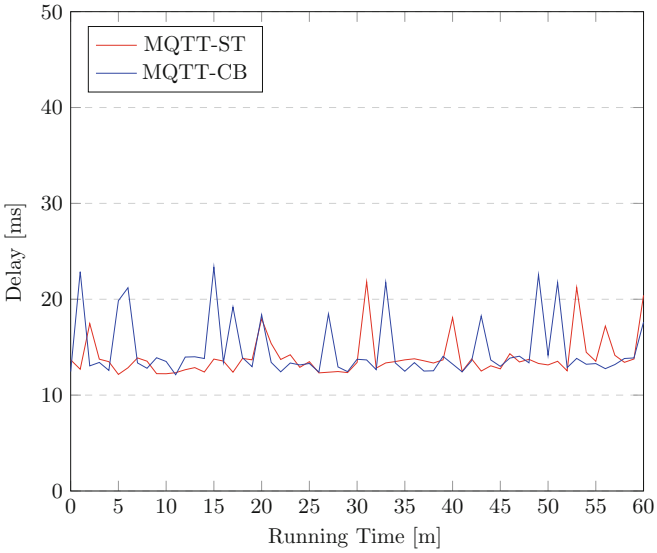

Fig. 5. Ex-1: The average delay of packets by constant 5 msg/s publication rate.

In the first experiment, we simulate MQTT-ST and MQTT-CB protocols with the same set of parameters listed in Table 1. We maintain constant values for all parameters to observe how our AI model performs in real-time. We introduced a burst of packets at a publication rate of 5 messages per second into the network. As shown in Fig. 5, both MQTT-ST and MQTT-CB behave similarly, as MQTT-CB is built on MQTT-ST and operates in the same way in a stable environment. The performance of our model can be seen in the prediction plot in Fig. 6. Moreover, the overall average delay of both protocols is approximately the same, as depicted in the plot in Fig. 7.

In the second experiment, we conduct simulations using the same parameters as in Table 1. All parameters were kept constant, except for the publication rate, which is varied. We initially fed the network with packets at a rate of 5 msg/s in bursting mode. After ten minutes, we increase the publication rate to 50 msg/s. In Zone-2, we observe that the delay for both protocols increased. Figure 8 depicts this increase in delay. At the end of Zone-2, the Prediction Module determines that the number of brokers needed to be increased by one. The reason for this can be seen in Fig. 9, which shows that the prediction percentage of retransmitted packets at the end of Zone-3 is higher than 10%. Thus, the MQTT-CB Controller creates a new broker at the end of Zone-2 because the model predicted according to values 15 min later, corresponding to the end of

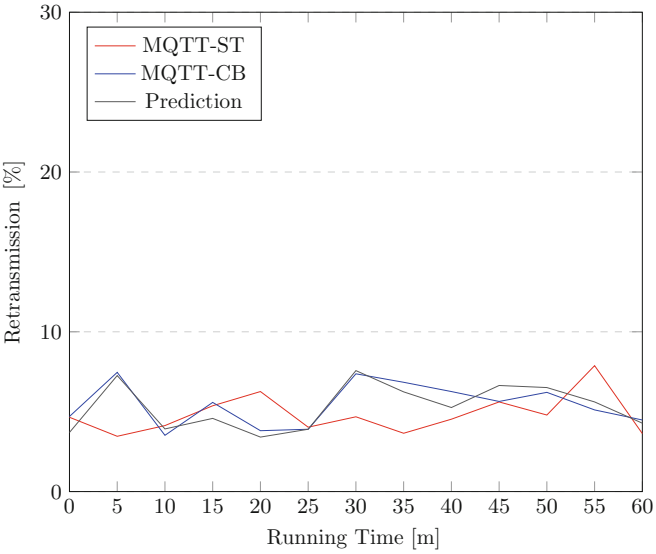

Fig. 6. Ex-1: The retransmission percentage by constant 5 msg/s publication rate.

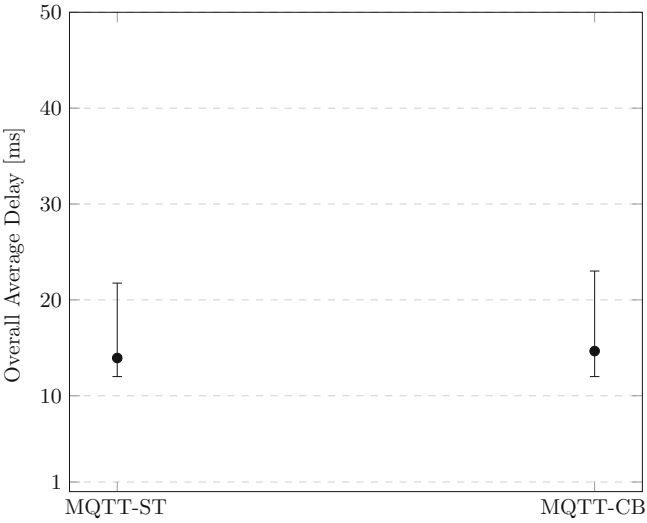

Fig. 7. Ex-1: The overall average delay of packets by constant 5 msg/s publication rate.

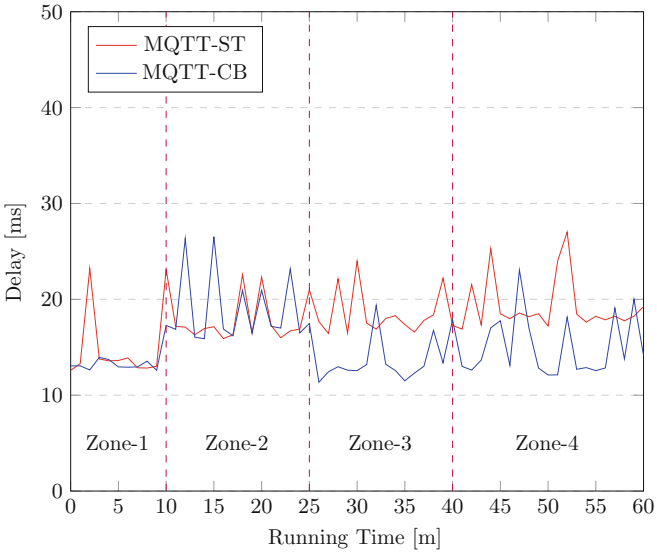

Fig. 8. Ex-2: The average delay of packets by increasing publication rate.

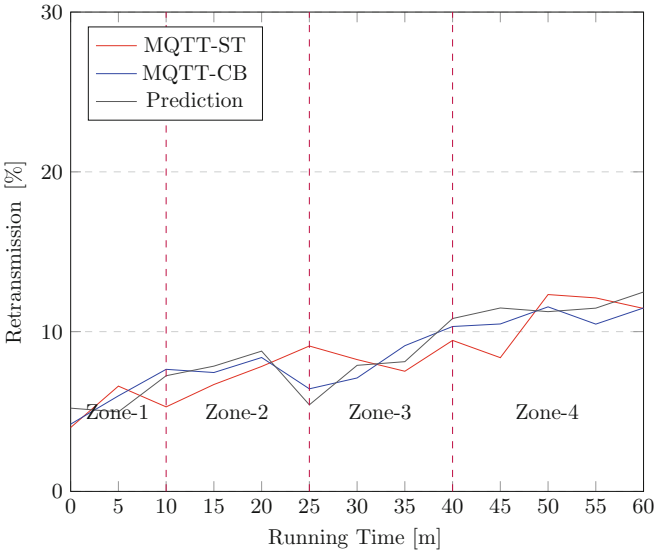

Fig. 9. Ex-2: The retransmission percentage by increasing publication rate.

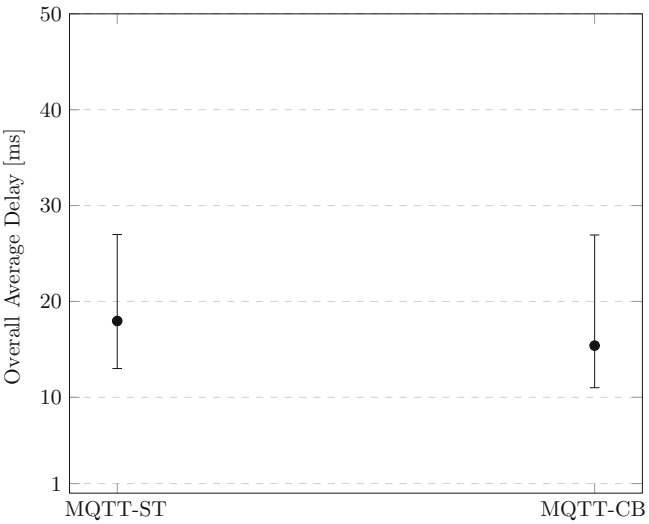

Fig. 10. Ex-2: The overall average delay of packets by increasing publication rate.

Zone-3. As a result, the delay for MQTT-CB decreases by approximately 3ms due to the increased number of brokers in Fig. 8. However, the retransmission rate for MQTT-ST and MQTT-CB is similar after Zone-2. Nevertheless, we focus on the delay for performance evaluation; hence, we use Fig. 9 for prediction. Moreover, the overall average delay of MQTT-CB is less than MQTT-ST, as illustrated in Fig. 10, although it has an extensive range.

In the third experiment, we conduct the simulation with the same parameters listed in Table 1. We decrease the publication rate in this experiment, feeding the network with packets in bursting mode at a rate of 5 msg/s until ten minutes have elapsed, after which we decrease the publication rate to 2 msg/s. In Zone-2, we observe that the delay for both protocols decreased. As shown in Fig. 11, at the end of Zone-3, the Prediction Module decides to decrease the number of brokers by one. This decision is based on the percentage of retransmitted packets predicted by the model, which was less than one percent at the end of Zone-2, and remained low until the end of Zone-3 in Fig. 12. This results in the MQTT-CB Controller deleting a broker at the end of Zone-3, which increases the delay of the MQTT-CB by about 3ms. Additionally, we can see in Fig. 13 that the overall average delay of MQTT-CB is higher than MQTT-ST. This is because we decrease the number of brokers, which impacts the delay. However, this approach helps decrease the resource consumption of brokers and clients without compromising performance in terms of delay. Here we assume the main reason for power consumption is the number of brokers, not the retransmission rate.

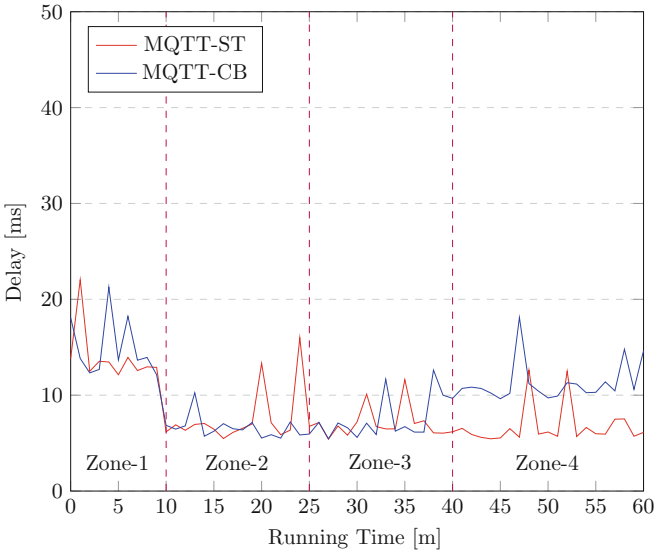

Fig. 11. Ex-3: The average delay of packets by decreasing publication rate.

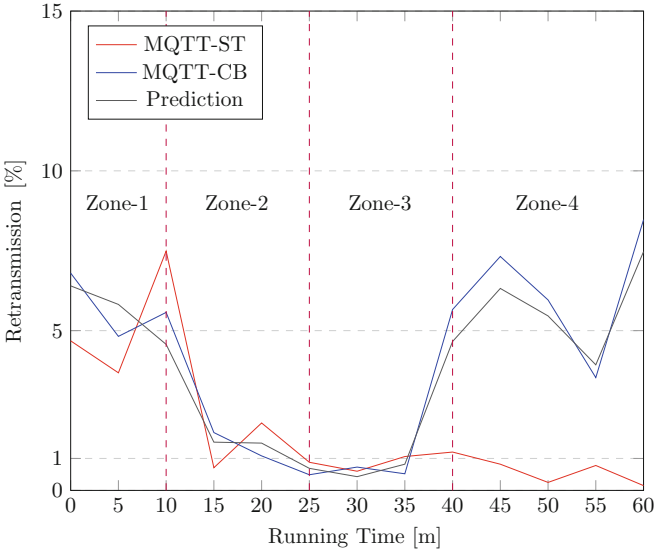

Fig. 12. Ex-3: The retransmission percentage by decreasing publication rate.

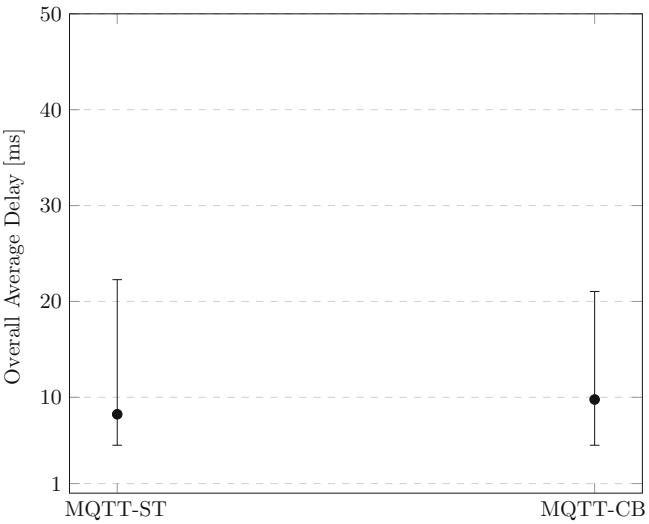

Fig. 13. Ex-3: The overall average delay of packets decreasing publication rate.

7 Conclusion

The MQTT protocol is widely used in the IoT industry. However, traditional MQTT protocols in literature are not designed for distributed environments and lack scalability. In this study, we propose a new MQTT-ST-based protocol that is intelligent, scalable, and distributed. To enhance the portability of our protocol, we utilize containers, and our cloud-based architecture facilitates easy deployment. Our primary contribution is adding intelligence to the MQTT-ST protocol. Our protocol can predict retransmitted packets in the future and dynamically adjust the number of brokers in real-time. Furthermore, it can decrease the number of brokers when clients are idle. Our experiments demonstrate that our protocol outperforms the MQTT-ST protocol regarding the delay between subscribers and publishers. Additionally, our protocol adapts well to changing environments regarding publication rates. Thus, we present an intelligent cloud-based MQTT protocol.

In future work, we aim to improve the accuracy of our prediction model and eliminate the warm-up time without affecting predictions. Also, we can evaluate our model with more brokers for all experiments, especially for resource consumption evaluation. Additionally, we can apply this intelligent model to other IoT protocols.

Acknowledgements. Muhammed Raşit Erol has received support from The Scientific and Technical Research Council of Turkey (TUBITAK) 2210A-National Scholarship Programme for MSc Students for this paper.

References

1. Mishra, B., Kertesz, A.: The use of MQTT in M2M and IoT systems: a survey. IEEE Access **8**, 201071–201086 (2020). https://doi.org/10.1109/ACCESS.2020.3035849
2. Naik, N.: Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria 2017, pp. 1–7 (2017). https://doi.org/10.1109/SysEng.2017.8088251
3. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of Things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutorials **17**(4), 2347–2376 (2015). https://doi.org/10.1109/COMST.2015.2444095
4. Johnsen, F.T., Landmark, L., Hauge, M., Larsen, E., Kure, Ø.: Publish/subscribe versus a content-based approach for information dissemination. In: MILCOM 2018–2018 IEEE Military Communications Conference (MILCOM), Los Angeles, CA, USA, pp. 1–9 (2018). https://doi.org/10.1109/MILCOM.2018.8599786
5. Ronzani, D., Palazzi, C.E., Manzoni, P.: Bringing MQTT brokers to the edge: a preliminary evaluation. In: IEEE 19th Annual Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA 2022, pp. 695–698 (2022). https://doi.org/10.1109/CCNC49033.2022.9700526
6. Koziolek, H., Grüner, S., Rückert, J.: A comparison of MQTT brokers for distributed IoT edge computing. In: Jansen, A., Malavolta, I., Muccini, H., Ozkaya, I., Zimmermann, O. (eds.) ECSA 2020. LNCS, vol. 12292, pp. 352–368. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58923-3_23
7. Banno, R., Sun, J., Fujita, M., Takeuchi, S., Shudo, K.: Dissemination of edge-heavy data on heterogeneous MQTT brokers. In: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), Prague, Czech Republic, pp. 1–7 (2017). https://doi.org/10.1109/CloudNet.2017.8071523
8. Banno, R., Ohsawa, K., Kitagawa, Y., Takada, T., Yoshizawa, T.: Measuring performance of MQTT v5.0 brokers with MQTTLoader. In: 2021 IEEE 18th Annual Consumer Communications and Networking Conference (CCNC), pp. 1–2 (2021)
9. Jutadhamakorn, P., Pillavas, T., Visoottiviseth, V., Takano, R., Haga, J., Kobayashi, D.: A scalable and low-cost MQTT broker clustering system. In: 2017 2nd International Conference on Information Technology (INCIT), Nakhonpathom, Thailand, 2017, pp. 1–5 (2017). https://doi.org/10.1109/INCIT.2017.8257870
10. Bender, M., Kirdan, E., Pahl, M.-O., Carle, G.: Open-Source MQTT Evaluation. In: IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA 2021, pp. 1–4 (2021). https://doi.org/10.1109/CCNC49032.2021.9369499
11. Longo, E., Redondi, A.E., Cesana, M., Arcia-Moret, A., Manzoni, P.: MQTT-ST: a spanning tree protocol for distributed MQTT brokers. In: ICC 2020–2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, pp. 1–6 (2020). https://doi.org/10.1109/ICC40277.2020.9149046
12. Emitter: Distributed Publish-Subscribe Platform. https://github.com/emitter-io/emitter. Accessed 12 Aug 2022
13. Downloading and Installing RabbitMQ. https://www.rabbitmq.com/download.html. Accessed 14 Aug 2022

14. VerneMQ: A Distributed MQTT Broker. https://github.com/vernemq/vernemq. Accessed 14 Aug 2022
15. Longo, E., Redondi, A.E.C., Cesana, M., Manzoni, P.: BORDER: a benchmarking framework for distributed MQTT brokers. IEEE Internet Things J. **9**(18), 17728–17740 (2022). https://doi.org/10.1109/JIOT.2022.3155872
16. Staglianò, L., Longo, E., Redondi, A.E.: D-MQTT: design and implementation of a pub/sub broker for distributed environments. In: 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), Barcelona, Spain, pp. 1–6 (2021). https://doi.org/10.1109/COINS51742.2021.9524110
17. Rausch, T., Nastic, S., Dustdar, S.: EMMA: distributed QoS-aware MQTT middleware for edge computing applications. In: 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, USA, 2018, pp. 191–197 (2018). https://doi.org/10.1109/IC2E.2018.00043
18. Thean, Z.Y., Voon Yap, V., Teh, P.C.: Container-based MQTT broker cluster for edge computing. In: 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE). Kedah, Malaysia, pp. 1–6 (2019). https://doi.org/10.1109/ICRAIE47735.2019.9037775
19. Bellavista, P., Foschini, L., Ghiselli, N., Reale, A.: MQTT-based middleware for container support in fog computing environments. In: IEEE Symposium on Computers and Communications (ISCC). Barcelona, Spain, pp. 1–7 (2019). https://doi.org/10.1109/ISCC47284.2019.8969615
20. Puliafito, C., Virdis, A., Mingozzi, E.: The impact of container migration on fog services as perceived by mobile things. In: 2020 IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Italy, pp. 9–16 (2020). https://doi.org/10.1109/SMARTCOMP50058.2020.00022
21. Abdelwahab, S., Hamdaoui, B.: FogMQ: A Message Broker System for Enabling Distributed, Internet-Scale IoT Applications over Heterogeneous Cloud Platforms (2016). ArXiv:1610.00620
22. Matic, M., Antic, M., Istvan, P.A.P.P., Ivanovic, S.: Optimization of MQTT communication between microservices in the IoT cloud. In: 2021 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2021, pp. 1–3 (2021). https://doi.org/10.1109/ICCE50685.2021.9427602
23. Ak, E., Canberk, B.: Forecasting quality of service for next-generation data-driven WiFi6 campus networks. IEEE Trans. Netw. Serv. Manage. **18**(4), 4744–4755 (2021). https://doi.org/10.1109/TNSM.2021.3108766
24. Madan, R., Mangipudi, P.S.: Predicting computer network traffic: a time series forecasting approach using DWT, ARIMA and RNN. In: 2018 Eleventh International Conference on Contemporary Computing (IC3), Noida, India, pp. 1–5 (2018). https://doi.org/10.1109/IC3.2018.8530608
25. tshark(1). (n.d.). https://www.wireshark.org/docs/man-pages/tshark.html Accessed 17 Jan 2022
26. SQLite Home Page. (n.d.). https://sqlite.org/index.html Accessed 15 Jan 2022
27. Mou, L., Ghamisi, P., Zhu, X.X.: Deep recurrent neural networks for hyperspectral image classification. IEEE Trans. Geosci. Remote Sens. **55**(7), 3639–3655 (2017). https://doi.org/10.1109/TGRS.2016.2636241
28. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS 2014 Workshop on Deep Learning (2014)
29. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735

30. Karim, F., Majumdar, S., Darabi, H., Chen, S.: LSTM fully convolutional networks for time series classification. IEEE Access **6**, 1662–1669 (2018). https://doi.org/10.1109/ACCESS.2017.2779939
31. Fu, R., Zhang, Z., Li, L.: Using LSTM and GRU neural network methods for traffic flow prediction. In: 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), Wuhan, China 2016, pp. 324–328 (2016). https://doi.org/10.1109/YAC.2016.7804912