SDN-DMQTT: SDN-Based Platform for Re-configurable MQTT Distributed Brokers Architecture

Fatma Hmissi(✉) and Sofiane Ouni

ENSI, Cristal Laboratory, Manouba University Campus - 2010, Manouba, Tunisia
fatma.hmissi@ensi-uma.tn, sofiane.ouni@insat.rnu.tn

Abstract. The MQTT (Message Queuing Telemetry Transport) protocol is widely recommended for IoT (Internet of Things) communication. It uses a single broker for data exchange, which is typically placed in the cloud. However, this centralized approach is not effective for scaling, leading to mist and edge broker distribution. In distributed infrastructures, administrators are forced to manually reconfigure all MQTT communication to meet performance metrics. However, manual reconfiguration has several drawbacks, e.g., it is a time-consuming process and prone to error. To remedy these issues, this paper proposes combining Software-Defined Networking (SDN) and the MQTT protocol. This proposal enables dynamic reconfiguration of MQTT communication on distributed MQTT-based infrastructure. In the current paper, a real-case study of experiments on a real test-bed is carried out to justify the necessity of the proposed approach.

Keywords: IoT · SDN · MQTT · Dynamic Reconfiguration

1 Introduction

Nowadays, the Internet of Things (IoT) plays an essential role in the world. IoT aims to provide several conveniences and benefits for people's daily lives. To achieve this, connected objects interact and communicate among themselves. This kind of communication between objects (or things) requires IoT-based application layer protocols. These protocols address numerous vital issues in IoT object communication, including unreliable and complex network environments, limited capacities of storage and memory, and restricted processor capability. The focus of attention and research on IoT produce distinct IoT-based application layer protocols. Papers [1–10] survey the key IoT communication protocols for the application layer, e.g., MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), AMQP (Advanced Message Queuing Protocol), and XMPP (Extensive Messaging and Presence Protocol). However,

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved
A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 594, pp. 393–411, 2024.
https://doi.org/10.1007/978-3-031-63992-0_26

MQTT has become the choice for IoT. Recent comparative analyses show that MQTT is the preferred protocol for IoT communication because of its advantages. For example, MQTT provides better reliability than other protocols.

MQTT follows the publish-subscribe (pub/sub) architecture for IoT communication. In MQTT-based IoT communication, IoT objects are classified into the following two types: publisher and subscriber. Publishers are producers of data, and subscribers are consumers of data. MQTT employs a broker as a central point to perform and control communication between publishers and subscribers. Publishers connect to the broker and can publish messages on topics. While subscribers may subscribe to particular topics and get published messages.

A number of cloud-based MQTT brokers are available on several cloud platforms [17,18] such as Microsoft Azure IoT Suite. A cloud MQTT broker is an MQTT broker (e.g., Mosquitto [11], HiveMQ CE [13], VerneMQ [14]) placed and running on the cloud [12]. It enables many different IoT objects to communicate together via MQTT, even when they are in different physical geographic locations. However, the number of connected devices is going to increase substantially in the coming years (it is expected that there will be 29.42 billion connected objects globally by the end of 2030 [15,16]). In this trend, the quick increase of the IoT causes a large amount of data and information to be gathered by an enormous number of connected things. Hence, cloud MQTT brokers are considered major problems for MQTT communication. Existing cloud MQTT brokers receive huge amounts of data from a large number of objects and resend it to other objects in a single cloud [19]. Respectively, the employment of MQTT brokers in the cloud causes an unnecessary delay and requires high bandwidth and energy consumption during the MQTT communication process. The additional delay in MQTT communication has a negative impact on the general effectiveness of IoT applications.

To deal with this problem, many researchers have paid attention to looking for promising ways to reduce the delay of MQTT communication. The geographic distribution of MQTT brokers as close as possible to IoT objects is considered a vital solution for this problem. Several solutions, such as those in [19,30] suggested MQTT broker distribution over edge and mist levels. Nevertheless, distributing MQTT brokers over edge and mist levels introduces some side effects associated with the complexities of management (or reconfiguration) and control of MQTT communication.

To address this challenge, a potential solution is needed to control and manage communication efficiently over the distributed infrastructure of MQTT brokers. Within the current available techniques, Software-Defined Networking (SDN) [21–23] is considered the most important option for this challenge. Therefore, the SDN-DMQTT platform is built. The proposed platform is an SDN-based architecture. It hides and simplifies the complexities of MQTT communication management within the distributed physical infrastructure of MQTT brokers. SDN-DMQTT mainly focuses on the following goals:

- Build a programmable distributed MQTT-based IoT architecture capable of dealing with dynamic scalability and adaptability.

- Provide a global view of the distributed physical infrastructure of MQTT brokers.
- Control MQTT communication and brokers remotely.
- Access and reconfigure distributed brokers remotely to deal with dynamic changes in MQTT communication.
- Manage the entire MQTT communication via software from a centralized location.
- Automate certain aspects to handle demand for reconfiguration of MQTT communication against dynamic changes in the environment and/or events that are unexpected.

This paper is structured as follows: Sect. 2 depicts the current state of the art for SDN-based MQTT solutions, figuring out the novelty of the proposal. Section 3 describes the proposed Software-Defined Networking for Distributed MQTT (SDN-DMQTT) platform. Section 4 introduces a functional design of SDN-DMQTT. Section 5 presents a case study. Finally, Sect. 6 concludes this work.

2 State of Art

This section summarizes the most recent contributions found in the literature regarding the combination of the MQTT protocol and SDN technology. The presented works use SDN to enhance the MQTT protocol.

2.1 MQTT Enhancements Using SDN

Shahri et al. [24] proposed a new system called Real-Time MQTT (RT-MQTT), based on SDN, to enhance MQTT with real-time communication services. The architecture includes OpenFlow(OF)-controllerr, OpenFlow-Data Base (OF-DB), OF-switches, Real-Time-Network Manager (RT-NM), and MQTT application components. RT-MQTT uses the user properties field in standard MQTT messages to specify real-time requirements. RT-NM examines messages to identify real-time reservation requests, extracting and registering attributes in OF-DB. The OF-Controller creates deterministic communication channels and updates the flow tables of OF-Switches.

Benson et al. [27,28] proposed a SDN-based middleware system to ensure timely and reliable delivery of mission-critical data from MQTT sources to MQTT consumers in congested networks. The system includes an SDN controller, a coordinator service, SDN switches, brokers, publishers, and subscribers. The coordinator service determines network flow, while the SDN controller configures switch flows and group tables based on priority and drop rate policies. The subscriber sends a subscription packet to the desired SDN switch, while the broker forwards published data to the subscriber.

Fawwaz et al. [29] proposed an SDN-based distributed MQTT broker architecture for the Mobile Edge Computing platform for smart city applications. The

new architecture aims to avoid wasting edge computing resources and reduce network traffic and the latency of data delivery. The proposed architecture consists of dynamically placing MQTT distributed brokers to support the mobility of MQTT applications. When a device changes its location, Fawwaz et al. [29] propose to configure or create a distributed broker on edge resources (SDN switches). Edge resources are chosen in such a way that they should reduce the latency of data delivery and have many computing resources. In this proposal, brokers that are not used for any communication are completely removed from edge resources. The architecture of the proposed system includes distributed brokers (containerized on edge-enabled SDN switches) managed by an SDN controller and Distributed Broker and Edge (DBE) manager, an MQTT proxy, and a central broker. The proposed system uses MQTT proxy servers to synchronize clients and distributed brokers, providing a global view of existing distributed brokers and related topics. The DBE manager manages the table of delegated topics and distributed brokers assigned to them, enabling the proxy to deliver topics to the appropriate distributed broker container and messages to the central broker. The DBE manager also collects and manages edge device computing resources, such as Central Processing Unit (CPU), memory, and storage, using data from the SDN controller to optimize placement. The SDN controller monitors edge device status and statistics, ensuring efficient management of computing resources on the local network.

All of the above approaches show notable advancements essential to IoT MQTT-based communication, such as the timely and reliable delivery of messages, reduced delay, and effective bandwidth usage, demonstrating the practicality of SDN in an IoT MQTT-based system.

2.2 Novelty of SDN-DMQTT

The previous state-of-the-art survey shows that there is a certain need to explore the capabilities of SDN to enhance communication for MQTT-based applications. Nevertheless, the contribution of Fawwaz et al. [29] is the only work that compares directly to our proposal in the sense that it enables controllable and reconfigurable MQTT communication. However, this work includes various drawbacks. The most important ones are listed below.

First, solution of Fawwaz et al. [29] is interesting for edge broker distribution. However, Fawwaz et al. [29] were not considered mist broker distribution. In addition, Fawwaz et al. [29] have changed the MQTT publish-subscribe model to enable controllable and reconfigurable MQTT communication by introducing a proxy into the MQTT communication. Where all communication should pass through a proxy. A proxy is switch-based software that receives messages from MQTT clients and then routes them to the appropriate broker. However, this modification adds complexity to the traditional MQTT-based publish-subscribe model by introducing an intermediate layer that manages message routing between clients and brokers. Moreover, the architecture proposed by Fawwaz et al. [29] does not include an application layer or northbound Application Programming Interfaces (APIs). Hence, other applications (such as load

balancing across brokers and real-time guarantees) that require reconfiguration of MQTT-based communication are not addressed in this architecture. They are unable to communicate with the controller or provide it instructions on how to set up these communications.

Therefore, SDN-DMQTT proposed in this paper solves issues of the state-of-the-art. The proposed solution aims to allow dynamic reconfiguration of MQTT communication on mist and edge MQTT broker distribution while keeping the simplicity of the traditional MQTT-based publish-subscribe model. Also, SDN-DMQTT addresses several optimization applications. Each application has the ability to communicate with the controller and provide instructions on how to set up MQTT communications.

3 SDN-DMQTT Overview

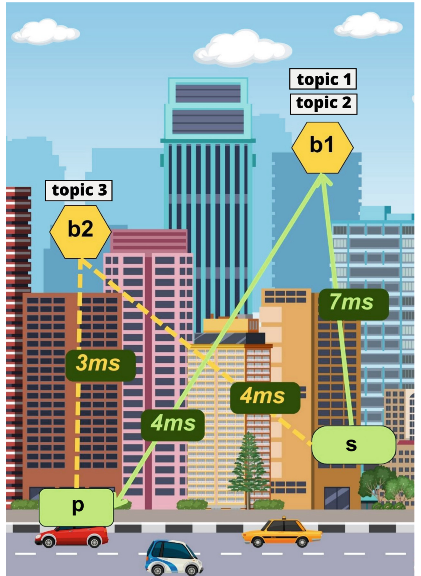
This section illustrates and describes the proposed platform. In the first place, it present the principle and aim of the proposal. On the other hand, it detail how the platform works by detailing its architecture.

3.1 Description

It is highly recommended to use the MQTT protocol for communication between IoT objects. MQTT-based communication consists of any exchange of data that takes place between IoT objects. MQTT uses a publish-subscribe pattern and a single broker for communicating data among customers based on topics of interest. Nevertheless, such a centralized approach does not scale effectively. Therefore, there is a need for mist and edge broker distribution. In these distributed infrastructures, reconfiguration of MQTT communication is required to meet a number of performance metrics, such as the requirement to extend communication to real-time.

Example. For example, considering a distribution of brokers for a smart city application, as can be seen in Fig. 1. Broker (b1) carries out communications from publisher (p) to the consumer who is interested (s) based on topic 1. Assume that the administrator requires enhanced communication between s and p by reducing the time of messages' delivery. The requirement can be addressed by minimizing the length of the pathways that the messages published by p would follow. This optimization can be accomplished by re-configuring MQTT communication, i.e., using broker b2 instead of b1.

The reconfiguration of MQTT communication is done manually by administrators, who shut down the publishers and subscribers, reconfigure their parameters, and then bring them online. In this case, subscribers are responsible for broker allocation. Obviously, this approach is very complex and has several drawbacks. This approach consumes a lot of time due to administrator interaction is prone to error. Furthermore, it is infeasible for an IoT network with billions of objects that need to communicate between them. Moreover, it is unsuitable for high-availability applications. To cope with these issues, automation and



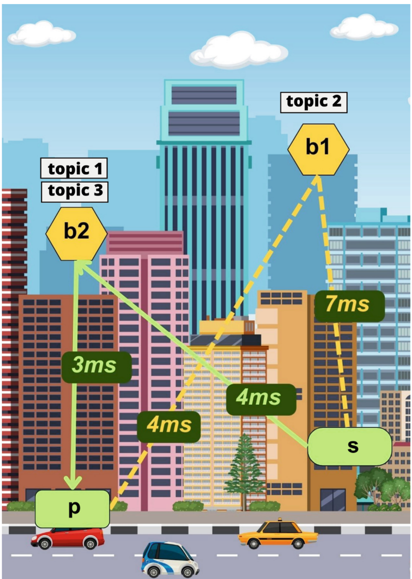


Fig. 1. An example that shows MQTT communication before and after reconfiguration

orchestration, programmability, and dynamicity, and visibility are required. To fit these characteristics, SDN-DMQTT is proposed. SDN-DMQTT leverages the capabilities of SDN technology and the MQTT protocol. The proposed approach remotely controls and reconfigures MQTT communication via a controller without requiring physical access to the MQTT components. In dynamic reconfiguration, the publishers and subscribers are connected automatically to the adjacent MQTT distributed brokers via the controller. In addition, the distributed MQTT brokers are allocated dynamically (topics are dynamically assigned to distributed MQTT brokers). The controller on the control plane dynamically controls and reconfigures MQTT communication on the infrastructure layer. MQTT applications in the application layer (e.g., monitoring of communication, load balancing, load sharing, extending communication with real-time requirements) may be managed and programmed by the controller using the Northbound API (e.g., REpresentational State Transfer (REST) API). The control layer is decoupled from the infrastructure layer, and it is logically centralized, having a global view of the whole MQTT communication. The controller guides the IoT devices to connect to their target brokers, and directs brokers to enable MQTT communication via the Southbound API (MQTT is used as a Southbound API).

3.2 Architectural Details

The proposal is an SDN-based, three-layered architecture. The proposed architecture includes an application layer, a control layer, and an infrastructure layer. Figure 2 depicts a schematic view of this architecture.

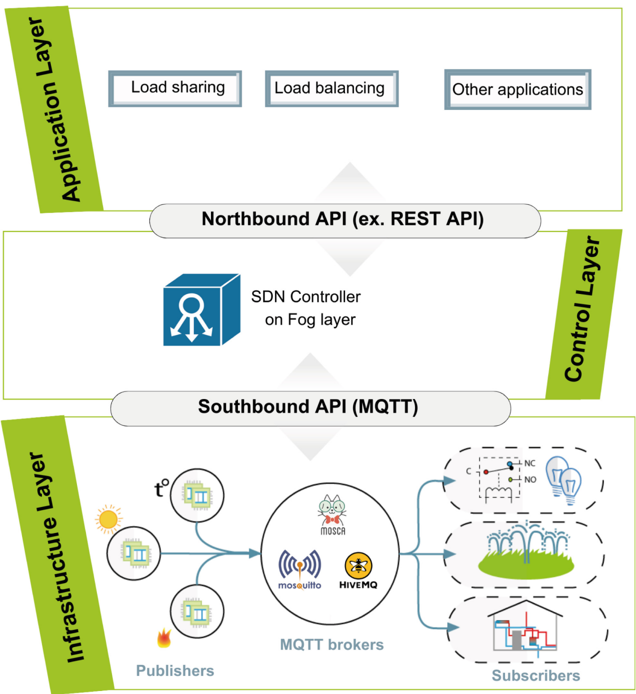


Fig. 2. Architecture of the proposed approach.

Application Layer is the highest layer in the proposed platform. Within that layer all the attributes, functions and rules of MQTT communication are defined. Applications demand the information of a distributed MQTT broker in order to react to it. These applications are able to make decisions concerning the allocation of distributed brokers based on changes, e.g., changes in the resources of brokers and the real-time location of topics. For example, when the processing time of a distributed broker increases, applications are able to change MQTT communication to a new broker dynamically. The application layer takes over all the applications like load balancing between distributed brokers, enforcement of real-time requirements, and broker monitoring (active broker on network, queue size, CPU, battery, topic location). These applications communicate with the SDN controller using the so-called northbound interface (e.g., REST). A northbound interface is an application programming interface (API) or protocol that enables an SDN controller to communicate with an application. Here, the SDN controller exposes northbound APIs, which are used by applications that can run on the same machine as the controller or on a different machine.

Control Layer is the middle layer in the proposed architecture. The control layer is responsible for controlling and reconfiguring communication over distributed MQTT infrastructures. For that purpose, it is made up of SDN controllers. An SDN controller is a central control entity in our proposal. This entity

works as the brain of SDN-DMQTT. It occurs as software on a centralized server at the fog level. The SDN controller provides two main tasks on SDN-DMQTT. First, it provides a centralized view and control of the MQTT broker distribution on the Edge and Mist layers and exchanges this information with upper-layer applications. The next task of the SDN controller is the management of MQTT communication according to the commands of the application layer.

To achieve these tasks, the SDN controller includes the following modules:

- Brokers monitoring module: the SDN controller uses this module to keep track of the distributed brokers. The controller implements this module to discover active brokers that are distributed at the edge and mist levels. Some existing works, such as in proposed a solution for brokers discovery [30]. The controller exchanges the list of discovered brokers with the top layer to give applications a global view of existing distributed brokers. In addition, it collects edge and mist device status and statistics via the SDN controller. These devices are nodes that deploy MQTT brokers. Furthermore, it collects status and statistics of the distributed MQTT broker, e.g., queue size, number of connected clients, and number of archived messages. The broker's information is used to decide about the optimization of topic placement or location. This module gets devices computing resources, such as CPU, memory, battery, and storage.
- Module of Topic Discovery: The controller provides this module to discover and retrieve topics assigned to activate MQTT brokers.
- Communication Management Module: This module allocates or frees brokers by assigning or disassociating topics to them. Therefore, it controls a handover procedure for moving topics from one broker to another. Moreover, it informs the client of the topics' locations.

Infrastructure Layer is the lowest layer in SDN-DMQTT. In the infrastructure layer, all the components of distributed MQTT are placed and connected physically: publishers, subscribers, and distributed brokers.

Publishers receive information from the controller about where to publish the data on a topic. Moreover, subscribers receive information from the controller about where to subscriber to a topic. Furthermore, the distributed brokers in the IoT network process a specific set of topics under the control of the SDN controller. In another world, distributed brokers maintain information from the SDN controller about their authorized topics. An authorized topic is used by a broker to filter published messages and forward them to their corresponding subscribers. In SDN-DMQTT, brokers are authorized only to filter and forward messages for topics assigned by the controller.

4 Functional Design

To demonstrate the feasibility of the SDN-DMQTT platform, a prototype was proposed. Figure 3 designates the proposed prototype.

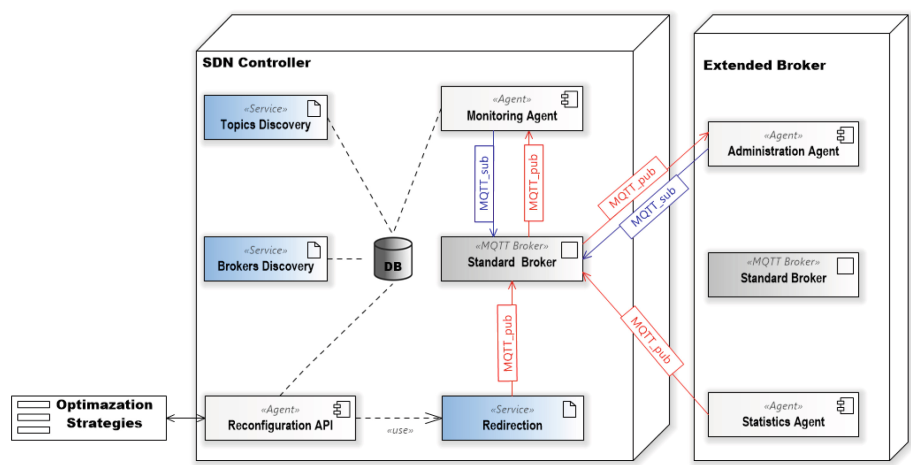


Fig. 3. Functional Design of SDN-DMQTT.

4.1 Extended Broker

It runs on the infrastructure plane, and it is responsible for delivering messages to the correct subscribers. Moreover, it is configurable by the SDN controller with which it interacts using the MQTT protocol. The components of the extended broker are shown in Fig. 3. The extended broker includes a standard MQTT broker (i.e., Mosquitto, HiveMQ, and VerneMQ) to perform the primary purpose of a broker. The standard MQTT broker operates as an intermediary, facilitating communication between publishers and subscribers. In addition, the proposed broker uses an administration agent to be notified of administration events (namely topic authorization and topic unauthorization) of the SDN controller via a standard MQTT broker. This agent configures the standard broker according to the command from the SDN controller. It is designed as an MQTT subscriber that subscribes to an administration topic in order to receive commands from an SDN controller via MQTT_sub. Furthermore, the extended broker comprises a statistics agent. This agent examines and reports on the extended broker. It is often used to track information about the CPU, memory, battery, connected clients, and number of retained messages. This agent delivers its report to the SDN controller periodically via MQTT_pub interface.

4.2 SDN Controller

The SDN controller works as the brain of SDN-DMQTT. It is in charge of discovering active brokers, controlling them, and discovering topics. Furthermore, the administrator employs it to dynamically set up the MQTT brokers and clients with the necessary communication information in an evolutionary manner. The functional design of the SDN controller is shown in Fig. 3. In particular, the designed controller is a black box that uses an MQTT-based API to interact with extended brokers. Following are the specifics. The designed SDN controller uses a simple MQTT API to monitor and manage extended brokers. Therefore, it encompasses these common tools: an MQTT broker, a monitoring agent, and a redirection service. The SDN controller uses a standard MQTT broker that acts as a go-between for the SDN controller and all the extended brokers. The monitoring agent presented in the diagram is an important component of the SDN controller used to monitor and surveil extended brokers. This agent subscribes to monitoring topics, e.g., myapi/brokerId/status, via MQTT_sub interface to receive statistics on extended brokers. The redirection service is used to move a topic from a broker, set a topic to a broker, and inform clients of topics' locations. Thus, this service publishes data on administration topics to indicate administration commands. This service is activated by the reconfiguration API. In addition, the SDN controller employs a database to store an organized collection of structured information about locations of topics and monitoring data of extended brokers and their status (is it an active endpoint or not). The SDN controller regularly discovers the active brokers on the network and the used topics via the broker and topic discovery service.

The SDN controller provides reconfiguration Application Programming Interfaces (or APIs) that administrators can use to build their own applications. An API is a set of programming instructions and standards for accessing data collected from the monitoring agent, brokers, and topic discovery services and translating administration requests to low-level language (or MQTT API). These instructions are a collection of requests for actions. In the following, a list of the API actions supported by the SDN controller and their expected input parameters is provided.

- The action named *redirectTopic(t_topic,a_broker,b_broker)* which takes the name of topic t, a reference to broker a, and a reference to broker b as parameters. This action updates the location of topic t from broker a to broker b. Therefore, it calls the service redirection to move topic t from broker a and set it to broker b. When the SDN controller successfully processes this action, the broker b can use the topic t to filter messages. However, broker a can't use it.
- The action named *setTopicLocation(t_topic,a_broker)* which has two parameters: a string that identifies a topic and a reference to a broker. This action is called to add the topic t to broker a. After the execution of this action, the broker a can use the topic t to filter messages.
- The *getBrokerTopics(a_broker)* action is passed a reference of a broker as a parameter. This action returns a list made up of the topics used to filter

messages on a_broker. If there are no topics, then the action returns an empty list.
- *getBrokerStatus(a_broker)* is an action that requires a broker reference as an input. It returns the statistics, such as remaining memory and energy, of an extended broker stored on the database.
- *getTopicLocation(t_topic)* returns the reference of the broker that filters messages using topic t. The argument may be a string that indicates the name of topic t. If the argument is an empty string, a list of all topics and their locations is returned.
- The *getBrokers()* action has no parameter as input and returns the list of all active MQTT brokers on the network.

5 Case-Study: Use of SDN-DMQTT for Load Sharing

SDN-DMQT is designed to control and reconfigure MQTT communication remotely. In this section, we provide a case study of how SDN-DMQT can be used for automatic control and reconfiguration of MQTT communication. Using SDN-DMQTT, the administrators are totally focused on the design and development of their application without being aware of how the MQTT components are controlled and managed. This section discuss a case study of many applications that can use SDN-DMQTT to optimize MQTT communication. The paragraph that follows represents the fundamental objective of this case study.

Distributed brokers placed on mist or edge nodes. However, brokers have a capacity and bandwidth limits to receive and to send messages. When the load on a broker achieves the maximum capacity of transmission medium, it will not be able to process the whole workload. Thus, delay of message communication increases when broker becomes overloaded. To solve this issue, a method known as "load sharing" is required. Load sharing aims to reduce burden of an overloaded broker. More specifically, it prevents load of a single MQTT broker of exceeding a specific threshold. However, when the broker's load reaches the threshold this method shift a particular amount of load to under loaded broker. This approach optimizes the performance of MQTT communication and availability of brokers. Load sharing is committed to reduce load of brokers to meet the delay requirements of MQTT communication. SDN-DMQTT allows administrators to automate a load sharing algorithm. This SDN-DMQTT based algorithm automatically control loads of distributed brokers and triggers the reconfiguration process of MQTT's communication when either of brokers ' loads achieve the threshold. Based on SDN-DMQTT, algorithm 1 proposes a load sharing algorithm with stable and minimal complexity and efficient load distribution influence. As defined in algorithm 1, the load sharing mechanism uses the following primitives of the controller API: getBroker(), getBrokerStatus(b), getTopics(b), getTopicLoad(t_topic,broker), and redirectTopic(t_topic,a_broker,b_broker).

Algorithm 1. A load sharing algorithm based on SDN-DMQTT

1: B : a set of active brokers
2: B_o : a set containing overloaded brokers
3: B_u : a set containing under loaded brokers
4: T : a simple threshold
5: S_b : a set of topics assigned to broker b
6: l_b : load of broker b
7: l_s : load of topic s
8: **procedure** LoadSharing(T)
9: $B \leftarrow getBroker()$
10: $B_o \leftarrow \{\varnothing\}$
11: **for all** $b \in B$ **do**
12: $l_b \leftarrow getBrokerStatus(b).load$
13: **if** $l_b > T$ **then**
14: $B_o \leftarrow B_o \cup \{b\}$
15: **end if**
16: **end for**
17: $B_u \leftarrow B_o \cap B$
18: **for all** $b \in B_o$ **do**
19: **while** $getBrokerStatus(b).load > T$ **do**
20: $S_b \leftarrow getTopics(b)$
21: $s \leftarrow S_b[1]$
22: $l_{max} \leftarrow getTopicLoad(s, b)$
23: **for** $i = 2$ **to** $|S_b|$ **do**
24: $l_s \leftarrow getTopicLoad(S_b[i], b)$
25: **if** $ls > l_{max}$ **then**
26: $s \leftarrow S_b[i]$
27: $l_{max} \leftarrow l_s$
28: **end if**
29: **end for**
30: $b_s \leftarrow b$
31: $i \leftarrow 1$
32: **while** $i <= |B_u|$ or $b_s = b$ **do**
33: $l_i \leftarrow getBrokerStatus(B_u[i]).load$
34: **if** $l_{max} + l_i < T$ **then**
35: $b_s \leftarrow B_u[i]$
36: redirectTopic(s, b, b_s)
37: **end if**
38: $i \leftarrow i + 1$
39: **end while**
40: **end while**
41: **end for**
42: **end procedure**

5.1 Experiment Test-Bed

This section describes the test bed that has been set up to conduct our experiments. For this study, a test network was performed in a local computing environment, as shown in Fig. 4.

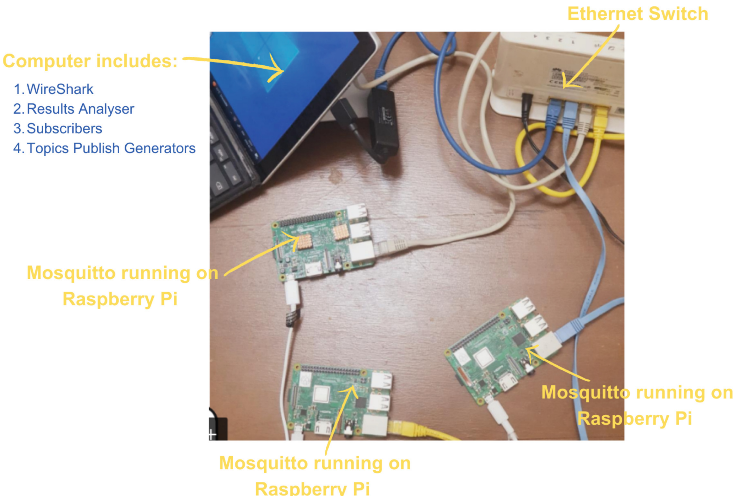


Fig. 4. Experimental environment.

As illustrated in Fig. 4, the hardware tools used in experiments are three Raspberry Pi [31], one Switch Ethernet, and one computer. For the experiments, a Raspberry Pi 3 Model B is employed. The Raspberry Pi is a tiny embedded IoT device with all the necessary features for operating as a computer. It uses less energy. On the Raspberry Pi, the Raspbian 10.13 buster [32] operating system is set up and installed for use in real-world applications. The memory is a 16-GB SD card. And the switch Ethernet is used for creating an isolated network without outside traffic and interference.

For all experiments, the following five software tools were used, as depicted in Fig. 5:

- Mosquitto is a free and open-source implementation of a broker for the MQTT protocol. It provides data to clients who are connected to it.
- Subscribers are MQTT clients. When a client subscribes to data, it means that it will receive the latest information from the broker as soon as it becomes available.
- Topics Publisher Generator is a Paho Python client. It publishes messages according to Poisson processes with a random arrival rate. In other words, this generator is typically employed to produce MQTT messages at a specific rate but is actually completely random.

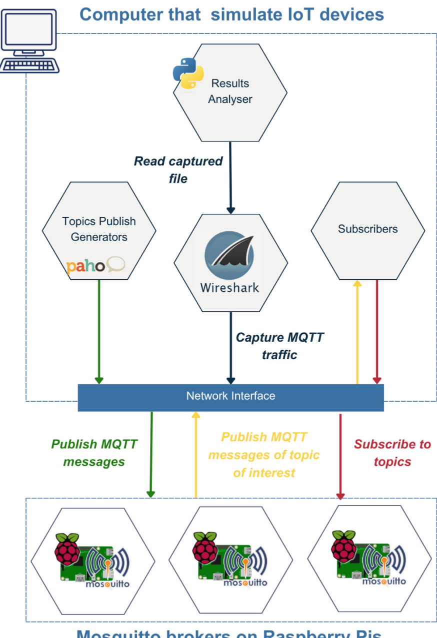


Fig. 5. Test-bed and evaluation model.

– WireShark [33] is a free, open-source packet analyzer for network maintenance, protocol development, and education. WireShark was downloaded on the computer side to capture all messages sent and received by the computer, and a special MQTT filter was configured because the focus of this expriment was only on MQTT packets.
– The Results Analyzer is a Python program. It analyzes the network scan log from a WireShark to show the delay of MQTT communication and the load of brokers.

5.2 Test Scenarios and Results

In this section, two different scenarios will be evaluated and discussed in order to understand the threshold-based load sharing mechanism. The first scenarios focus on analyzing and understanding the importance of load sharing and distribution across brokers when more than one broker's load reaches a specific threshold. The second scenario shows the benefits of the proposed mechanism. Test scenarios are given below.

Influence of Broker's Load on Delay of Communication. The aim of this test scenario is to study how the broker's load could increase the delay of MQTT communication. In the test scenario, a producer that publishes messages on a specific topic and a consumer that subscribes to the same topic are implemented. So, on the computer side, an MQTT client called "Publisher" was created using the Python programming language to publish messages on a given topic tag to one of the Mosquitto brokers on the Raspberry PI Board side. The publisher uses the Topic Publisher Generator to generate random published messages as Poisson arrivals, increasing the traffic load $\lambda \in \{400,500,1000, 2000, 3000, 4000, 5000,6000,10000\}$ messages/s. To receive data, a second MQTT client called "subscriber" was developed to subscribe to the publishing topic and receive data. In this test, the delay in actual communication was measured. For verification, WireShark packet capture software was used to capture, save, and analyze the received data (Fig. 6).

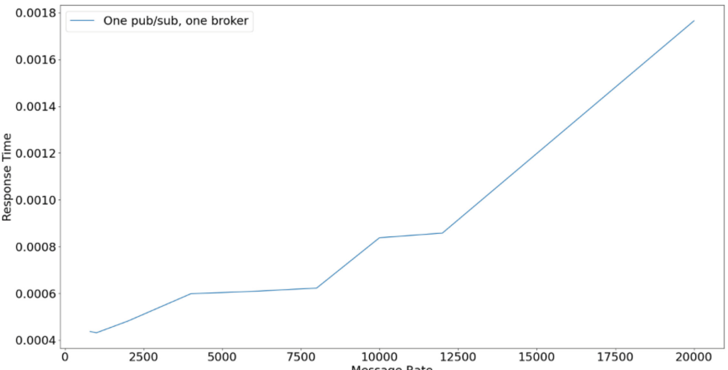


Fig. 6. Delay of 1st communication

Figure 6 represents the delay of communication against the load of the broker. Results indicate that the delay in communication increases with the broker's load. Figure 6 illustrates that with a load of up to 10,000 messages per second, an average delay of less than 0.001 s (1 millisecond) was obtained. However, an important increase in the delay of communication was noticed from the load of 12000 messages per second. From this scenario, It can be concluded that the load on a broker negatively affects the delay of MQTT communication when it reaches a specific threshold. Since, the delay gets going to increase significantly. Therefore, it's highly recommended to use a load-sharing algorithm.

Benefit of Load Sharing on Delay of Communication. This scenario aims to show that load sharing is required to reduce a broker's response time (and consequently reduce the delay of communication) when a broker is overloaded. In addition, it shows that it is highly recommended to use SDN-DMQTT for load

sharing mechanisms. As MQTT is based on a topic pattern, communication is classified per topic, and broker loads are organized into topics. Therefore, the delay of communication per topic is presented in this experiment.

This experiment focuses on justifying the benefits of using a load-sharing algorithm on the delay of each communication. Therefore, three topic-based communication channels were set up. Each communication includes a publisher that periodically publishes messages about a given topic (Top/Topic1, Top/Topic2, or Top/Topic3) as a Poisson arrival. And it includes a subscriber who is interested in receiving messages about topics.

Initially, all communication was set up to go through a single broker. Figure 7 shows that the delay increases remarkably for each communication after 19 s. This behavior is due to the significant increase in broker' load (see Fig. 8 that illustrates broker' load organized per topic.

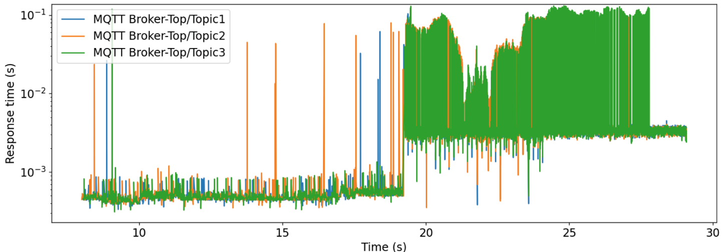


Fig. 7. Delay of communication before load sharing.

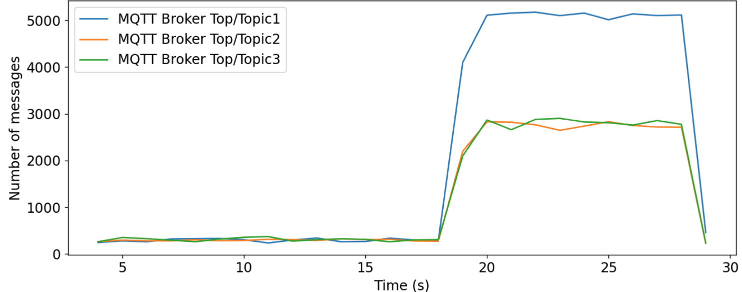
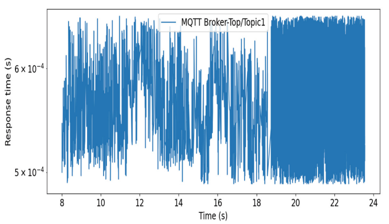


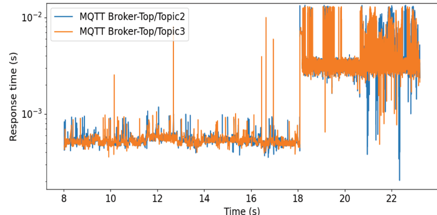
Fig. 8. Load of a broker before load sharing.

Afterwards, the delay of this communication was studied by distributing the current broker's load with another broker. Therefore, the SDN-controller will reconfigure the first communication (Top/Topic1-based communication) to go through an other broker. Comparing delay of communication that were done

without load sharing (See Fig. 7), it is obvious that delay of communication achieve better performance when load sharing is employed (See Fig. 9). Thus, in first case without load sharing, the response time delay will reach 0.1 s. However with applying the Algorithm 1 of the SDN controller, the reconfiguration will share the load between brokers. Therefore, reducing the workload per broker will lead to reduce the response time to 0,01 s for broker 2 as showed in Fig. 9 (b) and the response time to 0.001 s for the broker 1 (Fig. 9 (a)).



(a) Delay of 1st communication



(b) Delay of 2nd and 3rd communication.

Fig. 9. Delay of MQTT communication after load sharing.

6 Conclusion

MQTT broker distribution refers to the process of distributing MQTT brokers across multiple nodes in a network. This distribution enables better scalability, reliability, and load balancing of MQTT message traffic. It ensures that the MQTT brokers can handle a larger number of clients and messages by distributing the workload among multiple servers. Despite the various advantages of MQTT broker distribution, ensuring dynamic reconfiguration of communication becomes more challenging as the number of brokers and connections increases, making it crucial to propose robust mechanisms for dynamic reconfiguration. Therefore, SDN-DMQTT was proposed. SDN-DMQTT is a novel platform that leverages Software-Defined Networking (SDN) to enable seamless reconfiguration of MQTT communication within the distributed architecture of MQTT brokers. By harnessing the power of SDN, this platform offers enhanced flexibility and scalability for managing MQTT communication across multiple brokers, making it an ideal solution for large-scale IoT deployments. With SDN-DMQTT, organizations can easily adapt their MQTT communication infrastructure to meet evolving requirements and optimize communication performance.

References

1. El Alami, H., Sidna, J., Baina, A., Najid, A.: Analysis and evaluation of communication Protocols for IoT Applications. In: Proceedings of the 13th International Conference on Intelligent Systems: Theories and Applications (SITA 2020), November 2020. https://doi.org/10.1145/3419604.3419754

2. Chen, Y., Kunz, T.: Performance evaluation of IoT protocols under a constrained wireless access network. In: 2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT), Cairo, Egypt, pp. 1–7 (2016). https://doi.org/10.1109/MoWNet.2016.7496622.
3. Iqbal, F., Akhtar, S.M., Anwar, R.: A survey of application layer protocols of internet of things. Int. J. Comput. Sci. Netw. Secur. (IJCSNS) **21**(11), 301–311 (2021). http://paper.ijcsns.org/07_book/202111/20211141.pdf
4. Bayılmış, C., Ebleme, M., Çavuşoğlu, Ü., Küçük, K., Sevin, A.: A survey on communication protocols and performance evaluations for internet of things. Digit. Commun. Netw. **8**, 1094–1104 (2022). https://www.sciencedirect.com/science/article/pii/S2352864822000347
5. Naik, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: 2017 IEEE International Systems Engineering Symposium (ISSE), pp. 1–7 (2017)
6. Ouadghiri, M., Aghoutane, B., Farissi, N.: Communication model in the internet of things. Procedia Comput. Sci. **177**, 72–77 (2020). https://www.sciencedirect.com/science/article/pii/S187705092032278X, The 11th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2020)/The 10th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH 2020)/Affiliated Workshops
7. Al-Masri, E.: Investigating messaging protocols for the internet of things (IoT). IEEE Access. **8**, 94880–94911 (2020)
8. Silva, D., Carvalho, L., Soares, J., Sofia, R.: A performance analysis of internet of things networking protocols: evaluating MQTT, CoAP, OPC UA. Appl. Sci. **11** (2021). https://www.mdpi.com/2076-3417/11/11/4879
9. Donta, P., Srirama, S., Amgoth, T., Annavarapu, C.: Survey on recent advances in IoT application layer protocols and machine learning scope for research directions. Digit. Commun. Netw. **8**, 727–744 (2022). https://www.sciencedirect.com/science/article/pii/S2352864821000845
10. Jerrin Simla, A., Chakravarthy, R.: Review on application layer protocol for iot enabled agricultural intrusion detection. In: 2021 International Conference On Artificial Intelligence And Smart Systems (ICAIS), pp. 1170–1175 (2021)
11. Light, R. Mosquitto: server and client implementation of the MQTT protocol. J. Open Source Softw. **2**, 265 (2017). https://doi.org/10.21105/joss.00265
12. Mishra, B., Mishra, B., Kertesz, A.: Stress-testing MQTT brokers: a comparative analysis of performance measurements. Energies **14** (2021). https://www.mdpi.com/1996-1073/14/18/5817
13. Try hivemq HiveMQ. https://www.hivemq.com/ Accessed 17 Aug 2023
14. Company, O.: VerneMQ - A MQTT broker that is scalable, enterprise ready, and open source. (vernemq.com). https://vernemq.com/
15. Hnatyuk, K.: 120+ internet of things (IoT) statistics: market size, usage, growth & security. (MarketSplash) (2023). https://marketsplash.com/internet-of-things-statistics/#:~:text=The%20number%20of%20IoT%20devices%20is%20forecasted%20to%20reach
16. Katigbak, R.: The economy of things: the next value lever for telcos. (IBM Blog,2023,7). https://www.ibm.com/blog/the-economy-of-things-the-next-value-lever-for-telcos/
17. Hejazi, H., Rajab, H., Cinkler, T.: Lengyel, L. Survey of platforms for massive IoT, January 2018

18. Babun, L., Denney, K., Celik, Z., McDaniel, P., Uluagac, A.: A survey on IoT platforms: communication, security, and privacy perspectives. Comput. Netw. **192** 108040 (2021). https://www.sciencedirect.com/science/article/pii/S1389128621001444
19. Park, J., Kim, H., Kim, W.: DM-MQTT: an efficient MQTT based on SDN multicast for massive IoT communications. Sensors **18** (2018). https://www.mdpi.com/1424-8220/18/9/3071
20. Hmissi, F., Ouni, S.: An MQTT brokers distribution based on mist computing for real-time IoT communications, 13 July 2021. PREPRINT (Version 1). https://doi.org/10.21203/rs.3.rs-695717/v1
21. Zemrane, H., Baddi, Y., Hasbi, A.: SDN-Based Solutions to Improve IOT: Survey, October 2018
22. Sharma, R.: A review on software defined networking. Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol. 11–14 (2021)
23. Mohammed, A., Khaleefah, R., Hussein, M., Abdulateef, I.: A review software defined networking for internet of things, July 2020
24. Shahri, E., Pedreiras, P., Almeida, L.: Extending MQTT with real-time communication services based on SDN. Sensors **22** (2022). https://www.mdpi.com/1424-8220/22/9/3162
25. Almeida, L., Shahri, E., Pedreiras, P.: Enhancing MQTT with real-time and reliable communication services. In: IEEE International Conference on Industrial Informatics (INDIN), vol. **22** (2021)
26. Tamri, R., Rakrak, S.: The SDN-MQTT for an interoperable smart home. In: E3S Web Of Conferences, vol. **229** p. 01031 (2021)
27. Benson, K., et al.: FireDeX: a prioritized IoT data exchange middleware for emergency response. In: Proceedings Of The 19th International Middleware Conference, pp. 279–292 (2018). https://doi.org/10.1145/3274808.3274830
28. Scalzotto, L., et al.: An implementation experience with SDN-Enabled IoT data exchange middleware. In: Proceedings Of The 19th International Middleware Conference (Posters). pp. 21–22 (2018). https://doi.org/10.1145/3284014.3284025
29. Fawwaz, D., Chung, S., Ahn, C., Kim, W.: Optimal distributed MQTT broker and services placement for SDN-edge based smart city architecture. Sensors **22** (2022). https://www.mdpi.com/1424-8220/22/9/3431
30. Hmissi, F., Ouni, S.: TD-MQTT: transparent distributed MQTT brokers for horizontal IoT applications. In: 2022 IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), pp. 479–486 (2022)
31. Ltd, R. Buy a raspberry Pi 4 model B. (Raspberry Pi). https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
32. Debian - News - Updated Debian 10: 10.13 released. (www.debian.org), https://www.debian.org/News/2022/20220910
33. Foundation, W. Wireshark. (Wireshark.org.2016). https://www.wireshark.org/