Application of MQTT Based Message Brokers for IoT Devices Within Smart City Solutions

Denis Selimović*, Alen Salkanović* i Mladen Tomić*
* Sveučilište u Rijeci Tehnički fakultet, Rijeka, Republika Hrvatska
mladen.tomic@riteh.hr

***Abstract* - The advent of the Internet of Things (IoT) devices and platforms opened several new services and application possibilities within Smart City solutions. Their successful implementation is reflected in operational efficiency improvements as well as the quality of the provided government services. Such solutions also provide a way to share information with the public. The solutions assume the utilization of various IoT devices to collect and analyze data. Very important decisions in the implementations of such systems are related to the technology stack and the choice of data transfer system during communication. Different communication protocols are used in the application layer, and one of them is the lightweight MQTT message protocol, with the corresponding message broker. We propose the structure of an entire Smart City system solution. MQTT protocol is used for communication, and, data is visualized on an interactive map. Benchmark tests were done to examine the performance of the used message brokers. We show that the MQTT message protocol is a suitable solution for the proposed application. Also, a dedicated message broker service is the most suitable solution for routing information messages.**

Keywords - Internet of Things; Smart City; MQTT; Broker; Mosquitto; RabbitMQ; Benchmark test

I. Introduction

The Internet of Things (IoT) infrastructure is comprised of web-connected smart gadgets making use of embedded computers, sensors and communication hardware. Examples of gadgets include smartwatches, voice control devices, lighting appliances, alarm systems, medical sensors, security systems, etc. These devices, linked through the Internet, communicate with one another and with central servers and applications to exchange information. Their primary function is to gather, transmit, and analyze data obtained from their surroundings. An example of an IoT system is illustrated in Figure 1.

The IoT incentivizes businesses to reconsider their present business models and help them enhance their business strategies [5]. The benefits of IoT systems are diverse and include an increased focus on consumers, better security measures, improved supply chain management, cost and energy savings as well as pollution control. The IoT applications saw considerable growth in real-world applications for daily human activities and are particularly common in transportation, industrial, manufacturing and utility businesses. It is often used in other sectors as well, including the infrastructure industry, home automation, agriculture, healthcare and finance [2]. Figure 2 illustrates some of the possible application domains for the IoT.

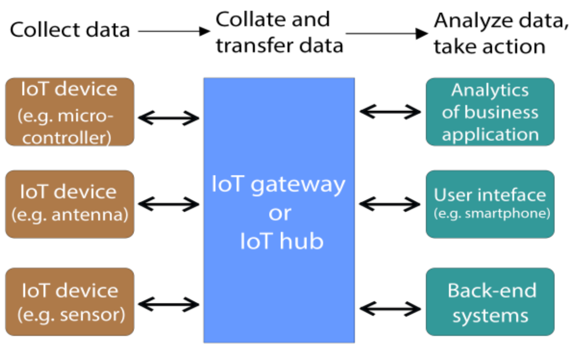

Collect data
Collate and transfer data
Analyze data, take action
IoT device (e.g. micro-controller)
IoT device (e.g. antenna)
IoT device (e.g. sensor)
IoT gateway or IoT hub
Analytics of business application
User inteface (e.g. smartphone)
Back-end systems

Figure 1. Example of an Internet of Things (IoT) system

IoT is increasingly used within technology-enabled smart city areas. This type of municipality can help residents in decreasing energy consumption. Furthermore, the system can be utilized to improve overall infrastructure and mobility as well as implement intelligent transportation systems, smart homes and/or buildings. A significant benefit of smart cities is their capacity to provide improved service delivery to citizens while using less infrastructure and resources, thus advancing municipal operations and enhancing inhabitants' quality of life. The enhancements enable new income sources and operational efficiency, resulting in cost savings for governments and individuals. The utilization of IoT infrastructure is likely to expand in terms of the number of devices and capabilities it can handle [10].

IoT devices are at the core of the IoT system. They are integrated into various systems and form a critical component of the Internet of Things. Various sensors are ubiquitous and essential to the functioning of a large number of modern organizations. There are many different types of IoT sensors available as well as respective areas of application and use cases. The sensors can deliver important data to a central location that analyzes it for specific patterns and abnormalities. This enables businesses and individuals to optimize their operations and management by responding promptly to critical changes [5]. For an IoT system to operate, the data gathered from many devices must be sent for processing. Therefore, a standard protocol is required for communication between devices. In other words, the IoT communication protocol is the key part of an IoT technology stack. Protocols and standards within the IoT infrastructure can be divided into two groups of layers: application layers include IoT dataprotocols, and physical layers define the corresponding network protocols used to connect devices over a network.

Network protocols are used over the Internet for seamless communication and data exchange. There are many network protocols, and some of the most commonly used ones are WiFi, Bluetooth, LoRaWan, Sigfox, ZigBee, Z-Wave and others. These protocols are not discussed in detail in this paper, and their selection depends on the target application requirements and implementation circumstances. On the other hand, IoT data protocols are very convenient for connecting low-power IoT devices. There are different application-layer messaging protocols for IoT and Machine-to-Machine (M2M) communication. When it comes to message protocols for IoT devices, a comparison of different protocols in terms of performance for specific applications is often mentioned. Each of the protocols has some advantages and disadvantages, but with an appropriate design and optimization, each of them can be a favourable choice for certain communication requirement.

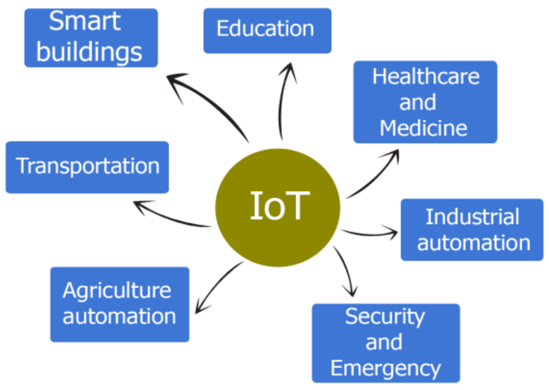


Figure 2. Some of the possible application domains for the Internet of Things

In this paper, data exchange within a Smart City solution is provided with the lightweight IoT data protocol called Message Queuing Telemetry Transport (MQTT). For the creation of a communication layer between services and applications, a discrete service called a message broker is used. Such a service can provide data processing, routing, message translation, persistence and delivery to all appropriate destinations. It is middleware that allows for a common communications infrastructure that grows and scales to meet the most demanding conditions. Many message brokers are open-source, and the paper describes and implements only some of them for performance comparison.

The paper is structured as follows. Section 2 gives a description and comparison of several application-layer messaging protocols. We outlined the advantages of using the MQTT protocol for the described application. Implemented message brokers based on the MQTT protocol are described in section 3. Section 4 provides a detailed description of the structure and design of the system for sending and receiving data from IoT devices, including a central server, device simulator and a sample web graphical user interface. Benchmark test results for different utilized message brokers are given and discussed in section 5. Finally, a conclusion is given regarding the implemented system and its results.

II. IoT Data Protocols

As mentioned earlier, IoT data protocols within application layers are used to connect low-power IoT devices. They are utilized to transfer data, i.e. messages, between IoT devices and the IoT messaging hub - a message broker. For devices to communicate, messaging protocols provide a set of rules and data formats. Many communication protocols are used according to the requirements of the target application. Each application requires a unique set of hardware, software, connection, and messaging protocols. Additionally, specific IoT systems may have additional critical requirements that demand the deployment of a specific, specialized protocol. The IoT devices may employ different messaging protocols and communication methods at different layers.

At first, a widely used Hypertext Transfer Protocol (HTTP) seemed a reasonable option for communicating with IoT devices. However, the protocol implementation is complex and needs some workarounds (e.g. long polling) that allow servers to push messages to clients. It also has significant drawbacks such as security weaknesses and device high cost and power consumption, weight issues and short battery life. Additionally, there is a problem of allowing clients to exchange data individually. Very importantly, HTTP is a heavyweight protocol due to its text-based protocol nature, and this entails the transmission of large-sized messages and correspondingly, a significant protocol overhead. It consumes far more power and memory than MQTT or any other lightweight protocol [1]. Since it is appropriate to use a lightweight protocol for communication with IoT devices, the following section provides a brief overview of several lightweight data protocols commonly used in IoT-qualified devices. The protocols may share common tendencies, but have different architectures and standards and are optimized for different applications related to IoT devices and their communication.

A. MQTT IoT Data Protocol

MQTT protocol is a straightforward and lightweight protocol for communicating between client machines through a central server machine, also known as a message broker. In other words, devices and applications communicate through a central point, and all communication routes are given through a centralized message broker.

It is a useful protocol for gathering data from extensive networks of tiny devices and storing it in a centralized place (server machine) for processing. It is also easy to implement, even in devices with low-end microcontrollers, and quickly gained popularity in the IoT world [11]. A good example is remote monitoring devices in cloud systems. Choosing such a centralized data collection and analysis may provide a satisfactory base for scalability and operational efficiency, and MQTT is optimized just for observed application. MQTT provides flexibility in communication patterns and allows for usage of devices with low power consumption [3].

MQTT protocol is simple to deploy and bandwidth-efficient, ensuring data transmission and interoperability with a high level of quality of service [5]. Even in the case of poor or unstable network connections, it can provide excellent performance in terms of the reliability of message delivery. Moreover, MQTT features additional message delivering reliability, known as a Quality of Service (QoS), in 3 levels - Level 0 for at most once delivery, Level 1 for at least once delivery and Level 2 for exactly-once delivery, which fits critical usage scenarios. Also worth noting is a good performance in terms of low battery consumption. It is ideal for IoT devices that need a longer lifespan without the need for frequent replacements. Thus, for the application in this paper, MQTT has proven to be a suitable choice.

The MQTT protocol features a publisher-subscriber messaging model, and it operates on top of the TCP/IP protocol. Messages are exchanged over TCP using a Publisher - Subscriber architecture through a central message server. The exchanged messages are classified into "topics". A device may either "publish" to a topic, i.e. provide data to the MQTT broker, or "subscribe" to a topic to acquire the data [11].

B. Alternative IoT Data Protocols

The MQTT protocol is not the only solution for communicating and sending IoT data between devices. There are many other protocols that we have collectively called alternative protocols in this chapter. They may be a preferred alternative if a target application has some specific requirements that are better addressed there than in MQTT. As mentioned above, the MQTT protocol has an optimized architecture for centralized IT infrastructure. Therefore, suitable applications of this protocol can be found in this sense. On the other hand, solutions can be found in the literature for cases when not all data are necessarily processed in one central place.

For example, the Data Distribution Service (DDS) is optimized for directly connecting sensors, devices and applications without dependence on a centralized IT infrastructure (peer-to-peer communication). This way, high reliability and fast real-time transmission of data with delivering millions of messages per second to a large number of recipients are achieved. It should be noted that data, in this case, can be sent to the central point on request, but this is not the original purpose of the protocol. Therefore, the protocol is very advantageous in cases where direct contact between devices is required. It may be useful in situations when critical decisions must be made in real-time, such as in military or industrial sectors [4]. Like the MQTT, this protocol operates on a publisher-subscriber model. Additionally, unlike MQTT, the DDS protocol enables consistent data transmission that is platform-independent [9].

It is apparent that this protocol exceeds the capabilities of the MQTT protocol, but for the application in Smart City solutions proposed in this paper, such a protocol is not required and in a way represents overkill. Another well-known application protocol used in wireless sensor networks is the Constrained Application Protocol (CoAP). It is a document transfer protocol like HTTP and designed to be easily translated to HTTP, but with some restrictions in terms of smaller HTTP packets. The main difference compared to HTTP is that it is a UDP-based protocol, which is why it is more suitable to be used in lightweight IoT devices. CoAP has typically been employed for building automation and smart energy purposes. This binary protocol facilitates communication in two modes: publish/subscribe and request/response [8]. Compared to the MQTT protocol which represents a many-to-many protocol (multiple clients), CoAP is primarily a one-to-one protocol, optimized for state transfer between client and server machines. It also supports data labelling with associated metadata which makes it easier for the client to understand the messages. MQTT does not support this way of communication, but with a standardized message format, it can achieve the same results.

Another binary TCP-based protocol that can use either a publish/subscribe model or a request/response pattern is the Advanced Message Queuing Protocol (AMQP). It is an open standard application layer protocol that is used to communicate between servers. AMQP is a safe and dependable protocol that is commonly used in banking and other industries that demand server-based analytical tools. AMQP also includes capabilities such as clustering, federation, flexible routing, persistent and durable queues, and high availability queues [6]. A cluster is a collection of one or more nodes that share users, virtual hosts, queues, exchanges, bindings, runtime settings, and other distributed state. Federation enables the exchange of messages across clusters. The features mentioned before introduce additional protocol overhead and makes AMQP more costly in terms of RAM and CPU resources than MQTT. Therefore, it is not recommended to utlize AMQP protocol for limited memory IoT devices, so its usage in the field of IoT remains fairly restricted.

We can conclude that all the listed alternative IoT data protocols are useful for certain applications, each with its advantages and disadvantages. We find out that the MQTT protocol for the proposed application within the Smart City solution represents the best fit beacuse of excellent performance in terms of the reliability of message delivery as well as low hardware requirements and good performance in terms of low battery consumption. We decided to employ the MQTT message protocol with several different message broker services. A more detailed description of the used message brokers is given below.

III. MQTT BASED MESSAGE BROKERS

Applications, systems, and services may connect and exchange data with the help of a message broker. By translating messages across formal messaging protocols, interdependent services can interact directly, regardless of the programming language or hardware used. Utilizing a message broker, the source application (producer) delivers a message to a server process that allows message translation, routing, data marshalling, and distribution to all the relevant recipients (consumers). There are two primary ways to communicate with a message broker: through Publish and Subscribe (Topics) or Point-to-Point (Queues) architectures. Broker selection is a very important step in designing such a system because it needs to process a potentially large number of simultaneously connected MQTT clients. The selection is generally data-driven and depends on the application. Scalability, integration, monitoring, security, and failure resistance also need to be considered. We will discuss three different message broker services. Common to the three brokers is that they support the MQTT data protocol. The brokers will be used in the proposed system structure and performance evaluated in the following sections.

A. MQTTnet

Messaging over the MQTT protocol may be accomplished using a publish/subscribe structure. If we want to create a custom made MQTT broker, we can use the MQTTnet .NET library that offers high-performance communication by MQTT, providing both MQTT client and MQTT server (broker). Our implementation is based on the documentation from the official MQTT site and is freely available on GitHub repository [12]. By using this solution we have much more flexibility in terms of modification to specific requirements. However, in most cases, dedicated broker services, such as those listed later in this section, provide better performance. It was to be expected as they are most often written in native programming languages as opposed to the MQTTnet library.

B. Mosquitto

A Mosquitto message broker is one of the open-source providers with implemented MQTT protocol (enabling the IoT connections). Mosquitto is a high-performance, lightweight message broker appropriate for usage on both desktops/servers and server platforms based on a single low-power board computer, like Raspberry Pi [3]. It is highly portable and also compatible with a variety of systems [7]. Mosquitto is simple to configure due to its lack of complex features, and it can manage significant application workloads with low performing CPUs and low memory requirements. Although it supports TLS and has plugins for database-based authorisation, it does have a major drawback since it does not support clustering so scalability becomes more challenging.

C. RabbitMQ

RabbitMQ is an open-source message broker that supports multiple messaging protocols. Although it is natively AMQP-based, it includes a robust plug-in ecosystem that supports MQTT, MQTT Web Sockets, HTTP REST API, STOMP, and server-to-server communications. RabbitMQ supports the usage of both the AMQP and MQTT protocols concurrently.

It is intended to be a general-purpose messaging protocol that may be used for both message-oriented middleware and peer-to-peer data transmission. One of the issues with the RabbitMQ is the MQTT support itself. Namely, with the MQTT plugin installed, RabbitMQ can act like a standalone MQTT broker. While this broker natively supports the AMQP protocol, the MQTT implementation lacks several critical capabilities, such as QoS level 2. This level guarantees that a message is received precisely once. This feature can be crucial in certain situations, for example when commands are transmitted from the IoT platform to the devices.

An example of a RabbitMQ messaging system with implemented MQTT broker is presented in Figure 3. It is a form of asynchronous service-to-service communication in which any message published to a queue or topic is received by one or more subscribers. In a RabbitMQ system environment with MQTT support, clients (subscribers) can subscribe directly to MQTT messages or it is possible to configure RabbitMQ to make MQTT data available as AMQP. Routing MQTT messages to AMQP queues is done by the direct exchange method. If we want all subscribers to receive a message, it is convenient for us to use the Publish and Subscribe method that contains topics. For example, when Publisher-Subscribe communication is used with IoT devices that send their states to a specific topic within the Smart City system, these devices are publishers, while subscriber(s) are device(s) to which these states are sent for further processing.

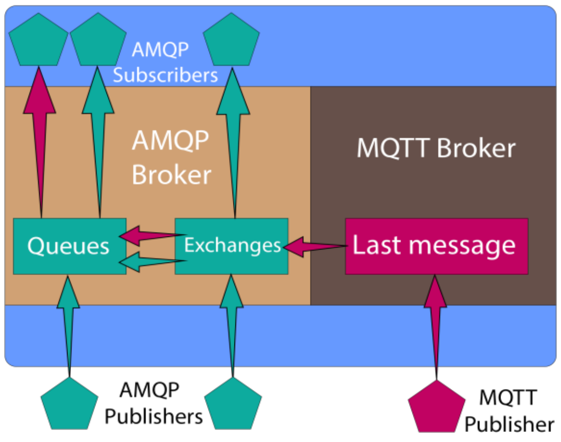


Figure 3. Data diagram of RabbitMQ message broker with MQTT plugin

IV. SMARTY CITY SYSTEM STRUCTURE

This section presents the proposed system architecture for data communication of IoT devices (clients) with the central processing and storage system (server). The system also includes a visualization of data (device status) using a graphical user interface that can be implemented as a web, mobile or desktop application. The proposed architecture is shown in Figure 4. For creating such a system, the .NET developer platform was used. It offers a wide range of libraries for implementation, so scaling and expanding the capabilities of this system is quite manageable. The entire code with instructions for setting up, building and running the system is available on the Github repository [12].

As an example, we observe IoT devices for monitoring the state of occupancy of city garbage containers, the shaft cover states on the roads and the occupancy of parking spaces. The implementation of such sensors requires IoT hardware. For testing purposes, a device simulator is designed as shown in Figure 5. According to the registered device unique id and unique topic name, the state message of that device is sent to the central message broker for processing. It is possible to send only one message, or, for testing purposes, 100 or 1000 random messages.

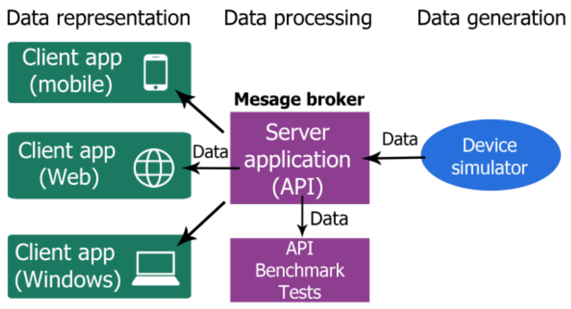
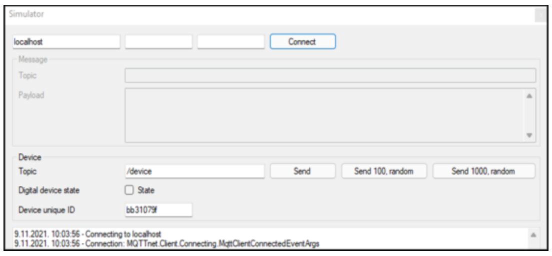


Figure 4. Proposed smart city system structure

The interactive representation of the device states is given in the web client interface as shown in Figure 6. Each unique device is stored in a database, with additional information such as latitude and longitude coordinates, device description, date of last change and current device state. We can monitor the status of all devices in a city or display filtered items on demand. The state change is visible in real-time. Container devices and parking spaces have status information (filled or unfilled), and shaft covers show the status as opened or closed. If any shaft is open or a container is full, a red mark appears on the map. The monitoring system is alerted in such cases. Since this is one of the bigger problems in a large number of cities, a smart city monitoring system can achieve better and more efficient management by utility companies. It is also possible to integrate additional alarms which are sent to specific locations, and the remediation process takes place promptly.



localhost

/device

bb31079f

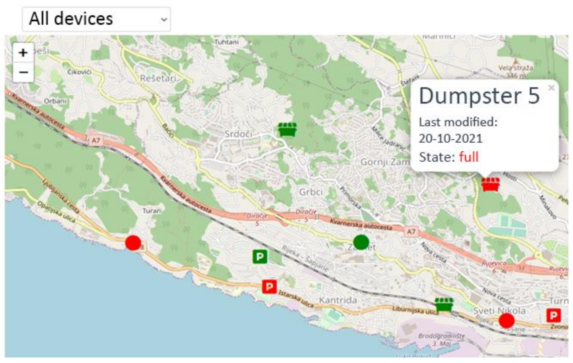
```
9.11.2021. 10:03:56 - Connecting to localhost
9.11.2021. 10:03:56 - Connection: MQTTnet.Client.Connecting.MqttClientConnectedEventArgs
```

Figure 5. IoT device simulator

V. Results & Discussion

Three message brokers have been implemented for application and comparison. One is MQTTnet (a custom made local MQTT message broker) and the other two (Mosquitto and RabbitMQ) are dedicated to native broker services. To use such dedicated services, it is necessary to prepare them according to official instructions. Setting up RabbitMQ may produce several problems depending on the operating system used, but a support page with frequently asked questions can assist in most cases. Also, the RabbitMQ message broker service had to be upgraded with an add-on for MQTT support as it was originally intended for use with AMQP. This was discussed in one of the previous sections.

The benchmark test application, as well as the implemented system, do not depend on the type of message broker used. In other words, it is possible to use any running message broker service. In process of changing the active message broker, we need to ensure that the previous broker service is stopped.



Dumpster 5

Last modified: 20-10-2021

State: full

Figure 6. Interactive web graphical user interface for monitoring devices

To examine the performance of message brokers used within the server application, benchmark tests with a different number of messages were generated. A benchmark test was used to compare descriptive statistics of different message brokers. Table 1 gives a comparative presentation of descriptive statistics obtained for all three examined message brokers after sending 10k messages. The tests were performed using an Intel Core i5-8265U CPU with 1.60 GHz and four physical cores. The operating system is Windows 10, and the tests were performed in .NET 5.0. Runtime environment. Examination time varies depending on the used broker, and averages about one minute.

TABLE 1. Comparative Representation of Descriptive Statistics for Different Message Brokers.

Broker Type	**Mean [ms]**	**Error [ms]**	**Std [ms]**	**Memory [MB]**	**Gen 0**
Mosquitto	83.34	1.30	1.78	3	875
MQTTnet	93.01	1.83	3.79	17	5500
RabbitMQ	80.40	2.21	5.90	3	875

The message processing time arithmetic mean value after sending 10k messages is given in the first column. Here, one can notice a difference in the use of dedicated native message broker services. By scaling to more messages and more customers, this difference is expected to be even more pronounced. For performance estimation and system design reasons, it is important to know how close the actual mean message delivery times will be to the benchmark mean values. For the purpose, we calculated the 99.9% confidence interval for the population mean and designated half of the confidence interval as Error, i.e. the maximum expected difference in mean delivery time, compared to the obtained benchmark values.

Mosquitto broker service provides the smallest error value which indicates that the sample mean is a more accurate reflection of the actual population mean. In addition, this message broker gives the lowest values of the standard deviation, which means it can be viewed as more reliable in terms of smaller variations in message processing time.

Table data related to allocated managed memory are given per single operation. It is given with the garbage collector (GC) Generation 0 collections per 1000 operations. The results show that the MQTTnet message broker has higher values related to memory collection. It means that MQTTnet collects memory many more times per one thousand benchmark invocations in Generation 0. This information is important when choosing the appropriate message broker. Namely, in the common language run-time, the GC serves as an automatic memory manager. The GC manages the allocation and release of memory for an application. Generation 0 gives the number of all newly constructed objects which are never examined by GC. That number should be as small as possible. Accordingly, it is better to use a dedicated message broker service (e.g. Mosquitto or RabbitMQ) in terms of memory occupation. Since the MQTTnet message broker is not a native service broker, it has been confirmed that its performance is worse than the other two native broker services. Thus, the memory allocation is significantly higher with MQTTnet brokers.

VI. Conclusion

The implementation of IoT systems is visible in the increased operational efficiency and quality of provided government services, while also facilitating information sharing with the public. Its growing popularity is especially emphasized within technology-enabled smart cities. We presented a framework for the complete smart city system and discussed the structure and architecture of the solution for transmitting and receiving data from IoT devices. Since these devices often need the least amount of memory and processing power feasible, they must communicate using suitable protocols used for messaging in conjunction with a suitable message broker. In this paper, we demonstrate that the MQTT protocol is well-suited for the proposed smart system. Performances are evaluated with three different message brokers: MQTTnet, Mosquitto and RabbitMQ and several benchmark studies. The related data is represented on an interactive map. The option to receive real-time notifications enables a user to get informed on certain events. Furthermore, message brokers might be evaluated in a variety of different scenarios including internet connection loss. Finally, it is possible to perform system scaling on more different devices as well as more messages. Additionally, client applications for smart devices running mobile operating systems might be implemented in the future.

Reference

[1] A. Petrova, “Reasons and Peculiarities of Choosing MQTT Protocol for Your IoT Devices,” Integra Sources, 2021, https://www.integrasources.com/blog/mqtt-protocol-iot-devices/

[2] P. Arivubrakan, K. Prema, “ The routing based protocol technique for enhancing the performance metrics using MQTT in the Internet of Things,” Materials Today: Proceedings, 2020, https://doi.org/10.1016/j.matpr.2020.11.070

[3] R. Balaji, A.V.R. Mayuri, N. Ramadevi, R. Anirudh Reddy, “ Advanced implementation patterns of internet of things with MQTT providers in the cutting edge communications,” Materials Today: Proceedings, 2020, https://doi.org/10.1016/j.matpr.2020.11.090

[4] D. Barnett, “MQTT and DDS: Machine to Machine Communication in IoT,” 2013, https://www.rti.com/blog/mqtt-dds-m2m-protocol-internet-of-things/

[5] Behrtech, “What is MQTT and Why You Need It in Your IoT Architecture,” 2021, https://behrtech.com/blog/mqtt-in-the-iot-architecture/

[6] CloudAMQP, “Why message queues for IoT projects? ,” 2021, https://www.cloudamqp.com/blog/why-message-queues-for-iot-projects.html

[7] Eclipse,“Eclipse Mosquitto - An open source MQTT broker,” 2021, https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

[8] Eclipse Foundation, “MQTT and CoAP, IoT Protocols,” 2014, https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

[9] C. Gregersen, “,” A Complete Guide to IoT Protocols & Standards In 2021, Nabto, 2021, https://www.nabto.com/guide-iot-protocols-standards/

[10] A. H. Hussein, “ Internet of Things (IOT): Research Challenges and Future Applications,”, International Journal of Advanced ComputerScience and Applications, 2019, , http://dx.doi.org/10.14569/IJACSA.2019.0100611

[11] Hwgroup, “ MQTT - universal protocol for cloud and IoT applications,”, 2021, https://www.hw-group.com/support/mqtt-universal-protocol-for-cloud-and-iot-applications

[12] Github Code, „RitehCityWatch“, available at https://github.com/denselGit/RitehCityWatch