ISSN 2409-2665
Journal of Logistics, Informatics and Service Science
Vol. 9 (2022) No.1, pp.85-96
DOI:10.33168/LISS.2022.0107

Performance Monitoring of MQTT-based Messaging Server and System

Kitae Hwang, Jae Moon Lee and In Hwan Jung

School of Computer Engineering, Hansung University, 02876 Korea

calafk@hansung.ac.kr

Abstract. In the messaging server where many devices send and receive messages, the ability to monitor the performance of the server and characteristics of the messaging is essential. This paper decides MQTT as the messaging protocol, and analyzes and defines key parameters that show the performance of the MQTT server and the characteristics of MQTT communication like MQTT message topics. In addition, this paper built an MQTT messaging server for testing by using Mosquitto as a MQTT broker and a separate monitoring system to monitor defined key parameters. In this paper, the system for monitoring the performance of the MQTT messaging server was built into three parts: a dashboard server, a monitoring application, and a test load generator. The test load generator is configured to generate a large amount of MQTT message load using 11 Raspberry PIs. The monitoring application was developed and installed on the MQTT server computer and periodically stored the server's performance and the MQTT message-related parameters processed by Mosquitto in the dashboard server's DB. The dashboard server was developed as a web server and implemented so that the administrator can view the data stored in the DB through a web browser in real time. Through experiments that generate various loads in the test load generator, it has been confirmed that the monitoring system operates normally. The monitoring system built in this paper is expected to be a good model of the monitoring system to be built together when developing an MQTT-based messaging server.

Keywords: MQTT, Mosquitto, Performance Evaluation, MQTT Broker, Messaging Server, IoTs.

1. Introduction

The world is evolving into a hyper-connected society where objects with various built-in sensors and small devices exchange various information in real time (Choi, 2014; Jung et al., 2018). Smart farms, smart universities, and smart city connected vehicles are representative. In the case of users, mobile devices, mainly smart phones, are connected to the Internet. The core performance of the connected world of Internet-connected things depends on the communication technology and performance between them (Sharma & Towari, 2016).

MQTT (Message Queuing Telemetry Transport) is a communication protocol standardized by a private standardization organization called OASIS (Organization for the Advancement of Structured Information Standards). It is a messaging protocol optimized for mobile devices and small devices with low bandwidth, in slow and low-quality networks. Since it is designed to transmit messages stably and focuses on low power, it is evaluated to be faster and more efficient than web-based information systems (MQTT, 2018; Aichernig & Schumi, 2018; Soni & Makwana, 2017).

MQTT is a text-based message exchange protocol and consists of clients and MQTT broker. Clients are divided into the subscriber and the publisher, and instead of directly communicating with each other, the communication between them is done through the relay of an MQTT broker. The subscriber is a message receiving client, and it notifies the message to be received by registering the topic string to the message broker. The publisher is a client that sends a message. When the publisher sends a topic string and a message to an MQTT broker, the MQTT broker sends the message to all subscribers waiting for the topic instead. Clients can be subscribers or publishers, or both.

Figure 1 shows an example of an IoT messaging system based on messages such as smart city (Su et al., 2011; Kumar et al., 2020; Grgić et al., 2016). There is an MQTT Server equipped with an MQTT broker, and a number of publishers and subscribers are connected to the MQTT broker to send and receive messages. For example, a street light device is a publisher, and when the current street light is on, if it sends an "on" message with "facility/street_lamp" topic, all subscriber devices waiting for "facility/street_lamp" topic receive the "on" message. And the message information is collected on the Dashboard Server and transmitted to the smart city management center or the administrator computer.

The system that manages the MQTT-based messaging system needs to provide various statistical information such as what topic messages are being used the most, how manymessages are being used, how many clients are sending/receiving messages, and what is the ratio of subscribers and publishers in real time. Meanwhile, in the MQTT-based messaging system, the message throughput of the message server increases in proportion to the number of clients, and the CPU utilization and memory usage of the messaging server are affected accordingly

(Benchmark of MQTT servers, 2019). Therefore, when an MQTT-based IoT system is built, a dashboard system that monitors the traffic volume of MQTT messages and the performance of the MQTT server is essential. The shaded part in Figure 1 is the subsystem that collects the performance of the MQTT server, monitoring information about MQTT topics and clients, and outputs it to the administrator.

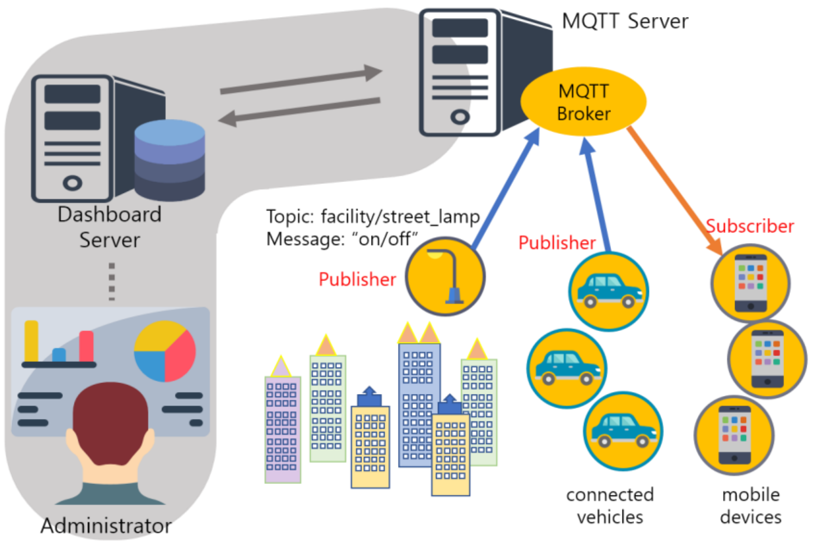


Fig. 1: MQTT Messaging System

In this paper, we define key parameters for MQTT communication such as MQTT server performance and MQTT messages and topics, and describe an example of design and implementation of a monitoring system. Mosquitto (Eclipse Mosquitto, 2018) was used as an MQTT broker, and a PC with Linux was used as an MQTT server computer and Mosquitto was run there. To emulate a large number of MQTT clients, we created dozens to hundreds of publishers and subscribers by utilizing 11 units of the single board computer, Raspberry Pi (Raspberry Pi, 2018). A web server and Database were built on the Dashboard computer to store server performance parameters and MQTT message traffic in real time, and output them to the administrator's web browser in real time.

2. Related Works

IBM MessageSight is a representative massive messaging server using MQTT. Without a doubt the IBM MessageSight product has a Web UI that acts as a dashboard for administration. Through this UI, users can monitor information about topics, subscriptions, connected clients, and performance of server computers. Information on topics includes the topic with the most published messages and the topic with the most subscribers. Information on subscription includes the subscriber with the most messages received, the subscriber with the most buffered messages, etc. The information on the client are information such as client ID and last connection date, etc. Information about the server is the server's memory usage or disk usage, etc.

Meanwhile, there are various studies on the dashboard for monitoring IoT systems based on MQTT (Postol, 2018; Zdraveski et al., 2017). There are various sites on Internet where you can easily create the dashboard web. There is also an example of an authoring tool that can simply create a dashboard that monitors the target MQTT server and IoT system using MQTT with a mouse and keyboard without programming (ThingsBoard, 2018). However, these dashboard production sites and authoring tool applications do not monitor performance related to MQTT servers or MQTT messages, but monitor each sensor value of an IoT system based on MQTT.

3. Performance Monitoring System

3.1. Monitoring system construction

The system to monitor the MQTT server is composed of 4 subsystems: Target Messaging Server, Dashboard Server, Test Load Generator, and Web Browser, as shown in Figure 2.

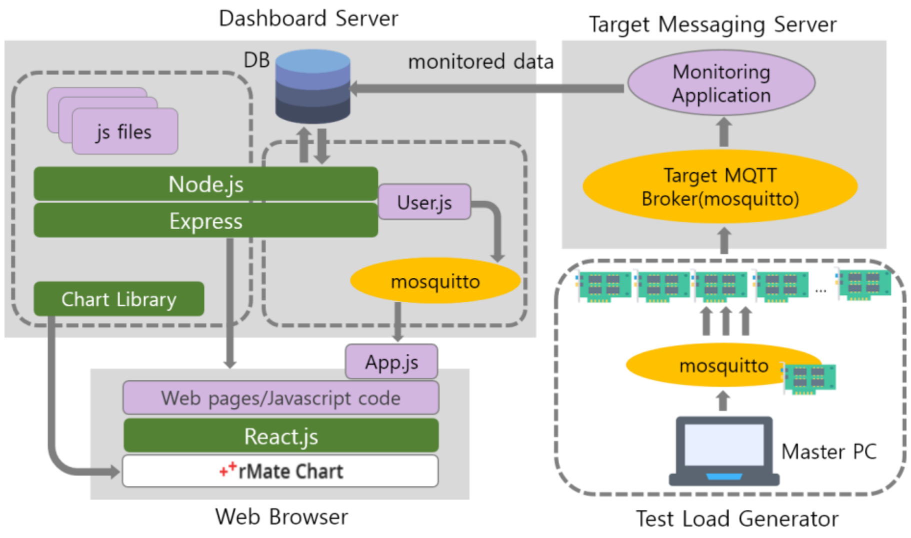


Fig. 2: System architecture

The Target Messaging Server is a monitoring target message server running a MQTT broker. Mosquitto, a target MQTT broker, is installed in Target Messaging Server. In addition, the Target Messaging Server is equipped with a Monitor

Application that monitors the messaging parameters of MQTT broker and performance parameters of the server computer and periodically transmits them to the DB of the dashboard server.

Dashboard Server is a web server system that outputs monitoring results through a web browser. It is equipped with a DB that stores data monitored from the Monitoring Application in real time and Node.js (Tilkov & Vinoski, 2010) for web services, and web pages and chart libraries, etc. The administrator can access the dashboard server through a web browser and view the monitoring results. A Mosquitto is additionally installed on Dashboard Server. User.js code of Dashboard Server publishes monitoring data stored in DB to this Mosquitto periodically in real time, and JavaScript code (App.js) running in Web Browser subscribes to this Mosquitto. It receives monitoring data in real time from the Mosquitto and outputs it in chart format in the Web Browser.

Test Load Generator is a subsystem to load a large amount of MQTT messages to the target Mosquitto installed in the Target Messaging Server. It consists of 1 Master PC and 11 Raspberry Pi single board computers, which exchange information using a separate Mosquitto installed on a Raspberry Pi. When the Master PC issues a command, 10 Raspberry Pis generate MQTT messages to the target MQTT Broker as instructed.

3.2. Monitoring Parameters

The monitoring data of this study is divided into two parts: server computer performance and monitoring parameters related to MQTT messages, and are shown in Table 1 and Table 2, respectively.

Table 1: Parameters of Server Performance

Parameter	Meaning
CPU/Core Utilization	current CPU/Core Utilization(%)
Memory Usage	current Memory Usage(%)
Network-in traffic	Network amount received(MB/s)
Network-out traffic	Network amount sent(MB/s)
CPU utilization of Process	CPU utilization used by each process(%)
Memory Usage of Process	CPU usage by each process(%)

Table 2: Parameters related to MQTT

Parameter	Meaning
N_client	# of clients connected currently
R_client	recent connected client ID

O_client	oldest connected client ID
MAX_client	Client ID with maximum message
MIN_client	Client ID with minimum message
N_RecMsgTopic	# of received messages per topic
N_SedMsgTopic	# of sending messages per topic
N_TotalMsgTopic	# of bytes of transferred message per topic

3.3. Design of DB Storing Monitoring Data

There are two DB tables that store monitored performance parameters of the server, as shown in Tables 3 and 4, and two sample data are included in each table. Table 3 is a Performance_Table that stores the server's current performance values, such as cpu utilization(%), memory usage(KB), and network I/O(KB/sec), and Table 4 is a Process_Table that stores a list of processes currently running in the Target Messaging Server including target Mosquitto, cpu usage per process(%), and memory usage(%).

Five tables between Tables 5 and 9 store parameters related to MQTT messages delivered through the target Mosquitto. Table 5 is a Connection_Info_Table table that contains connection information such as the number of currently connected clients, recently connected clients, and long-connected clients, and Table 6 is a Client_Table that stores client names and accumulated message count information. Table 7 is a Topic_Table containing information on the topic name, number of transmissions and receptions, and accumulated message size. Table 8 is a Subscription_Table that stores the names of subscribers and topics that they subscribe to and Table 9 is Publication_Table with names of publishers and topics published by them.

Table 3: Performance_Table

cpu_util (%)	cores_util (%)	memory_usage (KB)	network_in (KB/sec)	network_out (KB/sec)	date
3.48	0.66/0.44/0.21/0.11	954624	6.11	0	2020-05-15 15:34:10
4.67	0.36/0.11/0.23/0.55	755028	6.18	3.12	2020-05-15 15:34:13

Table 4: Process_Table

pid	cpu_util(%)	memory_util(%)	command
877	1.34	2.30	Mosquitto
29495	1.68	1.00	Java

3.4. Test Load Generator

Test Load Generator to monitor the performance change and the change of messaging parameters by sending a large number of MQTT messages to the Target Messaging Server is configured as shown in Figure 3. A total of 10 Raspberry Pis were deployed with clients publishing and subscribing to messages, and one other Raspberry Pi gave orders to these 10 Raspberry Pis. This Raspberry Pi is called Master Raspberry Pi.

Table 5: Connection_Info_Table

number_of_current_connections	recent_client_id	old_client_id	client_id_of_minimum_msg	client_id_of_maximum_msg	date
6134	Client55	Client82	Client155	Client82	2020-06-22 13:55:01
7436	Client62	Client102	Client155	Client100	2020-06-22 13:55:04

Table 6: Client_Table

id	number_of_messages(bytes)
Client1	1237
Client2	256

Table 7: Topic_Table

topic	message_receiving_count	message_sending_count	accumulated_msg_size
/Hansung/Creative	345	164	23426
/Hansung/Design	243	326	24321

Table 8: Subscription Table

client_id	subscription_topic
Client23	Hansung/Design
Client135	Hansung/Imagination

Table 9: Publication_Table

client_id	publication_topic
Client25	/Hansung/Creative
Client198	/Hansung/Imagination

And the administrator configures the test configuration that sends the test command to the Master Raspberry Pi using the Master PC. In Master PC, you can designate the ones to participate in the test among 10 Raspberry Pis, select whether to operate as a subscriber or a publisher, designate the number of subscribers and publishers participating in the test, and specify the topics to be used. The Master Raspberry Pi, who received the command, instructs 10 Raspberry Pis to work, and the 10 Raspberry Pis immediately connect to the target Mosquitto and send or receive messages.

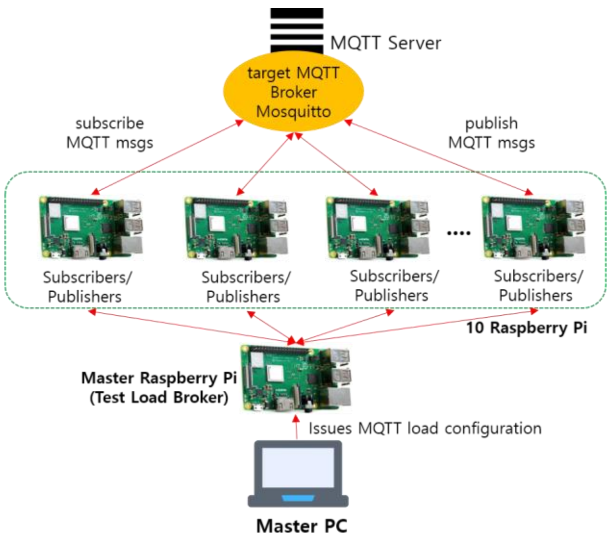


Fig. 3: Configuration of Test Load Generator

3.5. Target Messaging Server

The Monitoring Application running in Target Messaging Server is implemented to perform two functions. One is to find out the server performance in real time using the /proc file system and save it in the form of Table 3 and Table 4 in the DB of the Dashboard Server.

The other function is to monitor MQTT messages and traffic of Mosquitto, a target MQTT broker, and store them in the DB of the Dashboard Server. To this end, the Monitoring Application parsed the log information output by Mosquitto. Actually, Mosquitto prints log information to the standard output device for all MQTT messages arriving or occurring. Therefore, if the Monitoring Application parses the log information, it can analyze information about the connected clients, the topic of messages coming and going, and the size of the message.

3.6. Dashboard Server

Dashboard Server was implemented as a web server utilizing Node.js. And the chart displayed on the web browser was created using a commercial library called rMate. The DB that stores monitoring data in Dashboard Server was built using MySQL.

Mosquitto was additionally installed on the Dashboard Server to periodically transmit the data stored in the DB to the web browser. In the Dashboard Server, a separate JavaScript code periodically reads the data stored in the DB and publishes it to the Mosquitto broker, and the JavaScript code running in the web browser subscribes to the Mosquitto to receive the data published by the Javascript code of server and draws it in a chart format.

4. Performance Monitoring Result

Figure 4 shows the actual appearance of 11 Raspberry Pis built to load MQTT on Target Messaging Server. The Raspberry Pi shown at the top in Figure 4 is the Master Raspberry Pi, and the remaining 10 are actual MQTT clients, which cause the load of MQTT messages.

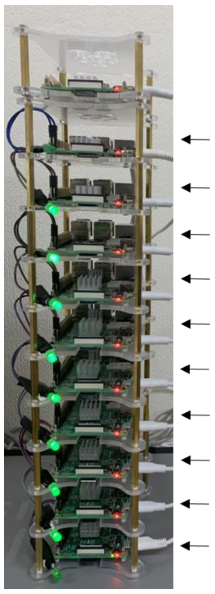
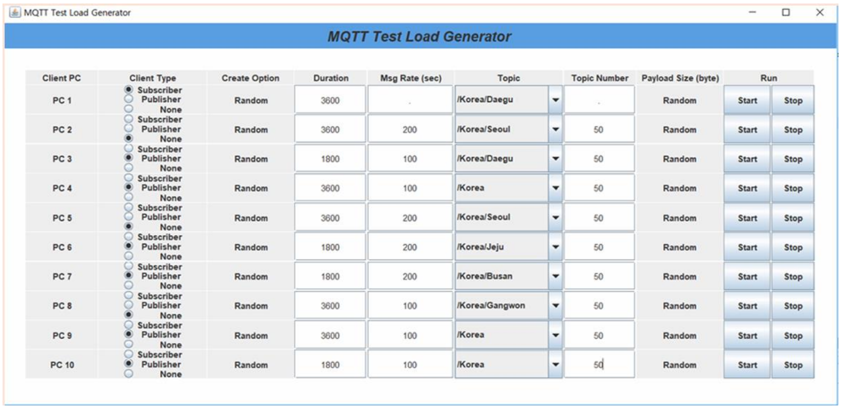


Fig. 4: Raspberry Pis constructed for Test Load

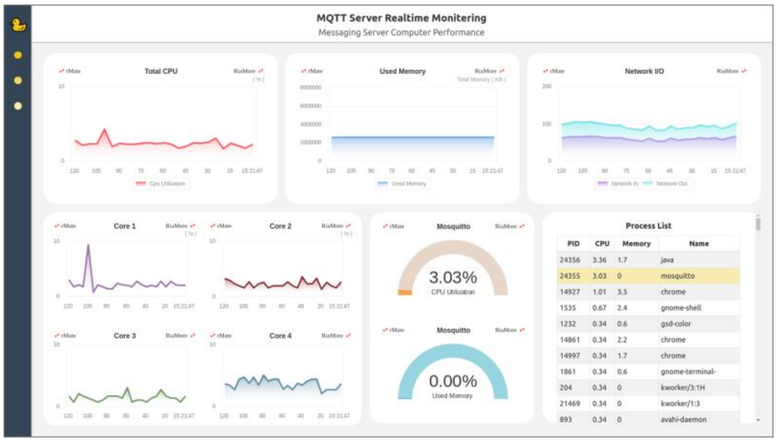
Figure 5 shows the process of designing the workload in Master PC by using the Test Load Generator Application created in this study. The user uses this application to input the load that the 10 Raspberry Pis will generate. For each Raspberry Pi, the user can specify the MQTT client type (either Subscriber or Publisher), the range of the number of clients, the duration of the client, and the topic to be used. The program installed in each Raspberry Pi receives this instruction, generates a client randomly within the maximum range. And these clients connect to the target Mosquitto of Target Messaging Server with the indicated topic and send and receive messages.

Figure 6 shows the status of a web browser that outputs the performance of the Target Messaging Server computer in real time, and Figure 7 shows the status of MQTT-related performance in real time.



Client PC	Client Type	Create Option	Duration	Msg Rate (sec)	Topic	Topic Number	Payload Size (byte)	Run	
PC 1	[✓ Subscriber] Publisher None	Random	3600	-	/Korea/Daegu	-	Random	Start	Stop
PC 2	Subscriber Publisher [✓ None]	Random	3600	200	/Korea/Seoul	50	Random	Start	Stop
PC 3	Subscriber [✓ Publisher] None	Random	1800	100	/Korea/Daegu	50	Random	Start	Stop
PC 4	Subscriber [✓ Publisher] None	Random	3600	100	/Korea	50	Random	Start	Stop
PC 5	Subscriber Publisher [✓ None]	Random	3600	200	/Korea/Seoul	50	Random	Start	Stop
PC 6	Subscriber [✓ Publisher] None	Random	1800	200	/Korea/Jeju	50	Random	Start	Stop
PC 7	Subscriber [✓ Publisher] None	Random	1800	200	/Korea/Busan	50	Random	Start	Stop
PC 8	Subscriber Publisher [✓ None]	Random	3600	100	/Korea/Gangwon	50	Random	Start	Stop
PC 9	Subscriber [✓ Publisher] None	Random	3600	100	/Korea	50	Random	Start	Stop
PC 10	Subscriber [✓ Publisher] None	Random	1800	100	/Korea	50	Random	Start	Stop

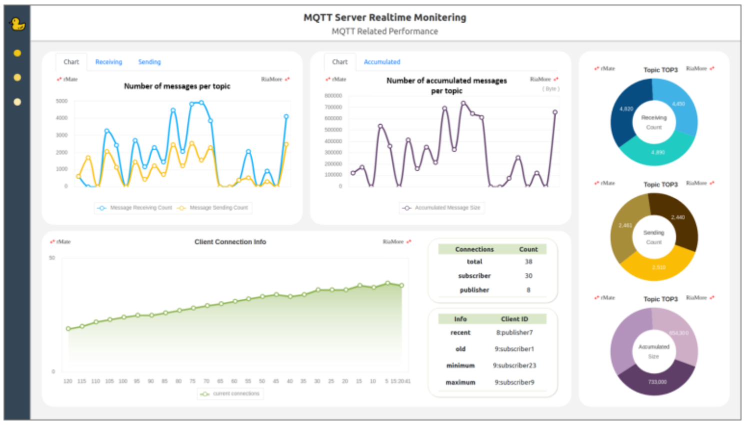
Fig. 5: Test Load Generator Application in Master PC



Process List

PID	CPU	Memory	Name
24356	3.36	1.7	java
24355	3.03	0	mosquitto
14927	1.01	3.5	chrome
1535	0.67	2.4	gnome-shell
1232	0.34	0.6	gsd-color
14861	0.34	2.2	chrome
14997	0.34	1.7	chrome
1861	0.34	0.6	gnome-terminal-
204	0.34	0	kworker/3:1H
21469	0.34	0	kworker/1:3
893	0.34	0	avahi-daemon

Fig. 6: Real time monitoring of performance of Messaging Server



Connections	Count
total	38
subscriber	30
publisher	8

Info	Client ID
recent	8:publisher7
old	9:subscriber1
minimum	9:subscriber23
maximum	9:subscriber9

Fig. 7: Real time monitoring of MQTT related traffic of Target MQTT broker

5. Conclusion

When implementing an MQTT system that processes large-scale messages, monitoring the performance of the MQTT broker and the server computer is inevitable. This paper showed the design and implementation of a system that monitors the performance of a messaging server equipped with Mosquitto as an MQTT broker. To put a large test load on the messaging server, we created a large number of MQTT clients using a total of 11 Raspberry Pis to generate a large MQTT message load. Also, we implemented an application that monitors the performance of the messaging server computer and the MQTT message load on Mosquitto on the messaging server computer, and stores them in a DB installed on the dashboard server in real time. We built a dashboard server in the form of a web server and implemented so that the administrator can view the data stored in the DB through a web browser in real time. The monitoring system built in this paper is expected to be a good model of the monitoring system to be built together when developing an MQTT-based messaging server.

Acknowledgment

This research was financially supported by Hansung University.

References

Aichernig, B.K., & Schumi, R. (2018). How Fast Is MQTT? *In International Conference on Quantative Evaluation of Systems, QEST 2018,* (pp. 35-56).

Benchmark of MQTT servers (2019). Scalagent. WEB: http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf

Choi, A.J. (2014). Internet of Things: Evolution towards a hyper-connected society. In *2014 IEEE Asian Solid-State Circuits Conference (A-SSCC)* (pp.5-8). DOI: 10.1109/ASSCC.2014.7008846.

Eclipse Mosquitto (2018). https://mosquitto.org.

Grgić, K., Špeh, I., & Heđi, I. (2016). A web-based IoT solution for monitoring data using MQTT protocol. *In International Conference on Smart Systems and Technologies (SST),* (pp. 249-253). DOI: 10.1109/SST.2016.7765668.

Jung, I.H., Lee, J.M., & Hwang, K. (2018). An MQTT based real rime LBS system for vehicles and pedestrians. *International Journal of Engineering and Technology*. *7*(3.24), 125-130. DOI: 10.14419/ijet.v7i3.24.2252

Kumar, V., Sakya, G., & Shankar, C. (2020). WSN and IoT based smart city model using the MQTT protocol. *Journal of Discrete Mathematical Sciences and Cryptography*. *22*(8). 1423-1434, DOI: 10.1080/09720529.2019.1692449.

MQTT. (2018). https://en.wikipedia.org/wiki/MQTT

Postol, A. (2018). How to Create Web Dashboards for IoT Devices. https://www.ibm.com/support/knowledgecenter/ko/SSWMAJ_2.0.0/WelcomePage/ic-homepage.html

Raspberry Pi. (2018). https://www.raspberrypi.org/products/raspberry-pi-3-model-b

Sharma, V., & Tiwari, R. (2016). A review paper on IOT & It's Smart Applications. *International Journal of Science, Engineering and Technology Research (IJSETR)*, *15*(2), 472-476.

Soni, D., & Makwana, A. (2017). A survey on mqtt: a protocol of internet of things (iot). *International Conference on Telecommunication, Power Analysis and Computing Techniques (ICTPACT-2017).*

Su, K., Li, J., & Fu, H. (2011). Smart city and the applications. *In International Conference on Electronics, Communications and Control (ICECC)*, (pp. 1028-1031). DOI: 10.1109/ICECC.2011.6066743.

ThingsBoard. (2018). http://thingsboard.io

Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing*. *14*(6), 80-83. DOI: 10.1109/MIC.2010.145.

Zdraveski, V., Mishev, K., Trajanov, D., & Kocarev, L. (2017). ISO-Standardized Smart City Platform Architecture and Dashboard. *IEEE Pervasive Computing*. 16(2), 35-43. DOI: 10.1109/MPRV.2017.31.