Construction of An Environment of Edging System for IoT devices

Tao WANG[1] Yaokai FENG[2] Kouichi SAKURAI[2]

Abstract：In recent years, as the application of the IoT(Internet of Things) has become more and more important in people's lives, edge computing has gradually received attention due to its superiority. MQTT (Message Queuing Telemetry Transport), as an information transmission protocol in IoT, has also been widely used in edge computing. In this paper, we will introduce the MQTT and build a simple Edging system for IoT devices. There are many systems (or application software) which implement MQTT protocol communication, but because each system has advantages and disadvantages, none of them can occupy a dominant position. The specifications and implementation of systems are also different. That caused great difficulties for beginner. Our study analyzes and compares commonly used application systems. And narrate how to build a system that fits needs. Guide beginners to better choose system which fits for them. We believe that will brings assistance to the people who are interested in relevant research. This report is the first step in our research.

Keywords： Internet of Things, MQTT, Mosquitto, EMQ, Edge-computing, Information Security

1. Introduction

The IoT is an emerging communications paradigm in which devices serve as objects or “things” that have the ability to sense their environment, connect with each other, and exchange data over the Internet. IoT (Internet of Things) era has been coming up and our world will become more convenient and more efficient. According to the Cisco Visual Network Index, mobile data traffic will grow at a compound annual growth rate of 47 percent from 2016 to 2021, reaching 49 exabytes per month by 2021 [1]. And at the same time, Innovations in the area of digital things, Information Communication Technology and IPV6 (Internet protocol) are enabling rapid deployment of Internet of Things (IoT) around the globe. It is estimated that trillions of IoT devices are going to be deployed in next five years [2]. More and more companies launch their own IoT cloud platforms. Typical IoT cloud platforms include AWS IoT, IBM Watson, OneNet, etc.

```
{
    "temperature": 25,
    "humidity": 60
}
```

Figure 1 IBM Communication Protocols

1 九州大学システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
2 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University

```
{
    "datastreams": [
        {
            "id": "Temperature",
            "datapoints": [
                {
                    "value": 25
                }
            ]
        },
        {
            "id": "Humidity",
            "datapoints": [
                {
                    "value": 60
                }
            ]
        }
    ]
}
```

Figure 2 OneNet Communication Protocols

Although there are so many IoT cloud platforms, all current IoT cloud platforms have common limitations:
1) The current cloud platforms only provide data communication protocols specified by their companies. For example, IBM Watson is like this(Figure1);
But OneNet looks like this (Figure 2).
2） The data encryption methods of these cloud platforms are different. Many companies need to burn the authentication files provided by the corresponding platforms for encrypted transmission. They cannot use custom encryption algorithms.
3）If you want to DIY upper-level applications based on remote node data collected by the platform. Many platforms provide APIs for pushing node data for third-party applications, such as OneNet's data push service. However, the data information of the first push is not complete enough to obtain specific user information. Secondly, the third-party application is regarded by OneNet as a user in the platform and has no right to obtain information of other users. At the same time, the API provided by the cloud platform to provide data push services for third-party applications only represents the agreement of the cloud platform manufacturer, and is not a standard. Third-party

applications rely heavily on the cloud platform. Once the interface of the data push service changes, the third-party application will have to be modified. Finally, the node data must first be uploaded to a cloud platform and then forwarded to the server of the third-party application.

These disadvantages may be tolerable for ordinary users, but for researchers, it is necessary to expand the use rights as much as possible, so a private IoT cloud platform needs to be built. In this paper, we mainly introduce the MQTT protocol and build an edge system for IoT devices under the environment of personal computer.

2. MQTT and Our Study

2.1 The Purpose of this study

There are many systems (or application software) which implement MQTT protocol communication, but because each system has advantages and disadvantages, none of them can occupy a dominant position. The specifications and implementation of systems are also different [10]. That caused great difficulties for beginner.

Our study analyzes and compares commonly used application systems which implement MQTT protocol. And narrate how to build a system that fits needs. Guide beginners to better choose system which fits for them. We believe that will brings assistance to the people who are interested in relevant research. The goal of this article is to establish an MQTT simulation system, based on which to conduct further security analysis and attack detection. Our study analyzes and compares commonly used application system which implement MQTT protocol. And narrate how to build a system that fits needs. Guide beginners to better choose system which fits for them.

2.2 Introduction of MQTT

Internet of Things (IoT) devices must be connected to the Internet. By connecting to the Internet, devices can collaborate with each other and with back-end services. Message Queue Telemetry Transport (MQTT) protocol is an application layer protocol designed for resource-constrained devices [7].

MQTT was originally invented and developed by IBM in the late 1990s [8]. Its original purpose was to link sensors on oil pipelines with satellites. As the name suggests, it is a messaging protocol that supports asynchronous communication between parties. Asynchronous messaging protocols separate message senders from receivers in space and time, so they can scale in unreliable network environments. Although it is called message queue telemetry transmission, it has nothing to do with message queues. Instead, it uses a publish and subscribe model. At the end of 2014, it officially became an OASIS open standard and was supported (using multiple open source implementations) in some popular programming languages.

2.3 Publish and subscribe models

The MQTT protocol defines two entity types in the network: a message broker and some clients. A proxy is a server that receives all messages from clients and then routes those messages to the relevant target clients. A client is anything that can interact with a broker to send and receive messages. Clients can be IoT sensors in the field, or applications that process IoT data in the data center. It also called Publish/Subscribe (pub/sub) messaging systems [10].

1) The client connects to the proxy. It can subscribe to any message "topic" in the broker. This connection can be a simple TCP / IP connection or an encrypted TLS connection for sending sensitive messages.
2) The client publishes messages within a certain topic range by sending messages and topics to the agent.
3) The broker then forwards the message to all clients that subscribe to the topic.

Because MQTT messages are organized by topic, application developers have the flexibility to specify that certain clients can only interact with certain messages. For example (Figure 3), sensors will publish readings within the “sensor_data” topic and subscribe to the “config_change” topic. Data processing applications that save sensor data to a back-end database subscribe to the "sensor_data" topic. The management console application can receive commands from the system administrator to adjust sensor configuration, such as sensitivity and sampling frequency, and post these changes to the “config_change” topic [11].

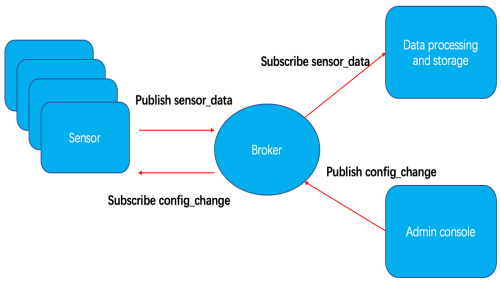

Figure 3 Framework of MQTT

2.4 MQTT main features

The MQTT protocol is a protocol designed to communicate with remote sensors and control equipment in low-bandwidth, unreliable networks. It has the following main features:

(1) Use the publish / subscribe message mode to provide one-to-many message publishing and decouple applications.
This is very similar to XMPP, but the information redundancy of MQTT is much smaller than XMPP because XMPP uses XML format text to pass data [12].
(2) Shielded message transmission.
(3) Use TCP / IP to provide network connection.
The mainstream MQTT is based on TCP connection for data push, but there is also a UDP-based version called MQTT-SN. Because these two versions are based on different connection methods, the advantages and disadvantages are naturally different.
(4) There are three types of message publishing service quality:
"At most once", message release is completely dependent on the underlying TCP / IP network. Message loss or duplication can occur. This level can be used in the following situations. Environmental sensor data. It does not matter if a reading record is lost, because there will be a second transmission in the near future. This method is mainly for general APP push. If your smart device is not connected to the network when the message is pushed, the push has not been received in the past, and it will not be received again when connected to the network.
"At least once", make sure the message arrives, but message duplication may occur.
"Only once", make sure the message arrives once. This level can be used in some demanding billing systems. In billing systems, duplicate or lost messages can lead to incorrect results. This highest quality message publishing service can also be used for the push of instant messaging apps, ensuring that users receive it only once [10].
(5) Small transmission, small overhead (fixed length header is 2 bytes), protocol exchange is minimized to reduce network traffic.
This is why the introduction said that it is very suitable for "in the field of Internet of Things, communication between sensors and servers, and information collection." It is necessary to know that the computing capacity and bandwidth of embedded devices are relatively weak. It is suitable to use this protocol to pass messages.
(6) Use the Last Will and Testament features to notify the parties concerned about the client's abnormal interruption mechanism.
Last Will: the last words mechanism, used to notify other devices under the same topic that the device that sent the last words has been disconnected.
Testament: Testament mechanism, similar in function to Last Will.

2.5 MQTT protocol packet structure

In the MQTT protocol, an MQTT data packet is composed of three parts: a fixed header, a variable header, and a payload. The MQTT packet structure is as follows:
(1) Fixed header. It exists in all MQTT data packets and indicates the packet type and the packet class identifier of the data packet.
(2) Variable header. It exists in some MQTT data packets. The type of the data packet determines whether the variable header exists and its specific content.
(3) Message body (Payload). It exists in some MQTT data packets, indicating the specific content received by the client [10].

2.6 The advantage of MQTT

For IoT developers, MQTT is the perfect balance between lightweight and network protocol flexibility. The lightweight protocol means that it can be implemented on strictly restricted device hardware or on networks with high latency or limited bandwidth. The flexibility of MQTT enables it to support multiple application scenarios for IoT devices and services [3].

At the same time, by comparing other protocols, we find that MQTT is more suitable for the Internet of Things than other existing protocols.

First, the most widely used HTTP protocol does not perform well in the Internet of Things. We assume that IoT devices are connected to Web services. By connecting to a web service, the device data is sent as an HTTP request, and updates are received from the system as an HTTP response. However, due to some characteristics of HTTP itself, many defects are generated.

HTTP is a synchronous protocol. Therefore, the client waits for a response from the server. Web browsers can meet this requirement, but at the cost of scalability. In the IoT world, synchronous communication has problems due to the large number of devices and the network is usually unreliable or has high latency. So far, asynchronous messaging protocols are more suitable for IoT applications. With asynchronous communication, the network can determine the best path and time for the sensor to send measurements and pass that data to the target device or service.
And it is one-way. The client must initiate a connection. Devices and sensors are often clients in IoT applications. That is, no device or sensor can passively receive commands from the network.
HTTP is a one-to-one protocol. The client sends a request and the server responds. Therefore, it is difficult and expensive to broadcast a message to every device on the network. But in the Internet of Things, the number of devices is extremely large. At the same time, the Iot is heterogeneous, for IoT due to multitude of heterogeneous devices, storing and managing the certificates and key exchanges for every session is cumbersome and also SSL/TLS suffers from attacks such as BEAST, CRIME, RC4, Heartbleed, etc. Thus a

scalable, lightweight and robust security mechanism is required for MQTT and its variants for deploying in IoT [5]. This leads to the fact that HTTPS cannot be applied to the IoT.

HTTP is a heavyweight protocol with many headers and rules. It is not suitable for networks with limited bandwidth.

The most commonly used messaging protocol in enterprise middleware systems is AMQP (Advanced Message Queuing Protocol) [3]. But in these high-performance environments, computing power and network latency are usually not a problem. AMQP is designed to provide reliability and interoperability for enterprise applications and has a rich feature set, but is not suitable for resource-constrained IoT applications.

In addition to AMQP, there are other popular messaging protocols. One example is Extensible Messaging and Presence Protocol (XMPP). XMPP is a peer-to-peer instant messaging (IM) protocol that focuses on features that support IM use cases, such as status and media attachments. Compared with MQTT, XMPP requires more resources on both the device and the network.

Compared with the existing protocols, MQTT has many of the advantages mentioned above, and has been applied to IoT actual systems. With the popularity of the IoT, MQTT will become more common in the future, and network attack detection and defense in MQTT-based IoT environments will more and more important, there are quite a few researchers involved in the research of MQTT security, Y. Upadhyay et.al. propose using ACL (access control list) to provide encryption method for the data and finally monitoring those data on webpage or any network devices [4]. To achieve the secure transmission of information in the MQTT system. And M. Singh et.al. propose a secure version of MQTT and MQTT-SN protocols (SMQTT and SMQTT-SN) in which security feature is augmented to the existing MQTT protocol based on Key/Ciphertext Policy-Attribute Based Encryption (KP/CP-ABE) using lightweight Elliptic Curve Cryptography [5].

3 Construction of a Simple System

3.1 Part of Broker:

Apache-Apollo: A proxy server developed on the basis of ActiveMQ that supports exchanging messages between heterogeneous and distributed software applications using several messaging mechanisms [9], and also can support multiple protocols such as STOMP, AMQP, MQTT, Openwire, SSL, and WebSockets. Apollo provides back-end management pages to facilitate developer management and debugging.

EMQ: EMQ 2.0, known as a million-level open source MQTT message server. It is based on the Erlang / OTP language platform and supports large-scale connections and distributed clusters. It is an open source MQTT message server with a publish-subscribe model.

Mosquitto: An open source message broker software that implements the MQTT v3.1 message push protocol, providing a lightweight, push / publishable message push mode [14].

Since Apache-Apollo has not been updated for a long time, in this article, we only introduce the installation of EMQ and Mosquitto in Mac.

3.2 Mosquitto installation：

Step1: Download the code from [13].

Step2: Because it is under the mac environment, you need to install brew. Open terminal and enter command (Figure 4):

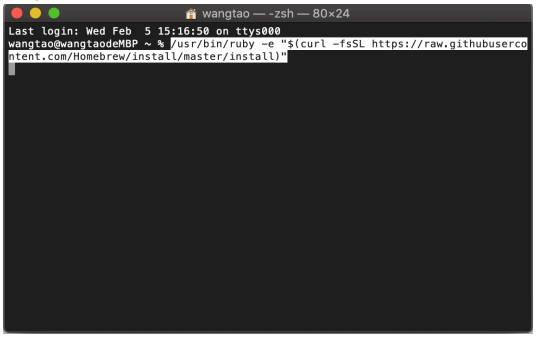

Figure 4 Download Command

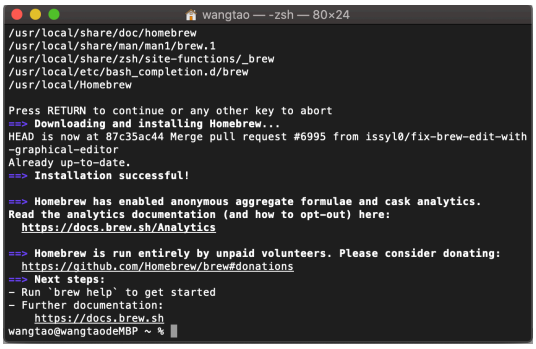

Figure 5 Installation Success

In this picture (Figure 5), installation successful, and next we can install the Mosquitto.

Step 3：

Input the commend: brew install mosquito. (Figure 6)

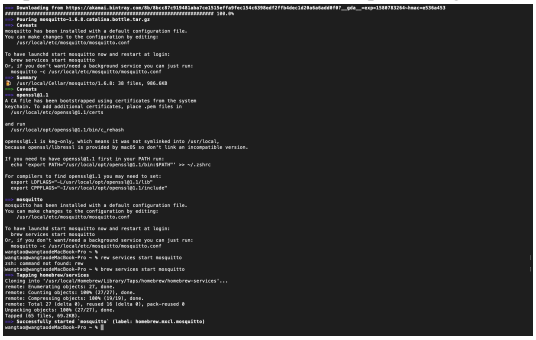

Figure 6 Install Mosquito

And the Terminal show the mosquito is installed. Now we try to verify it.

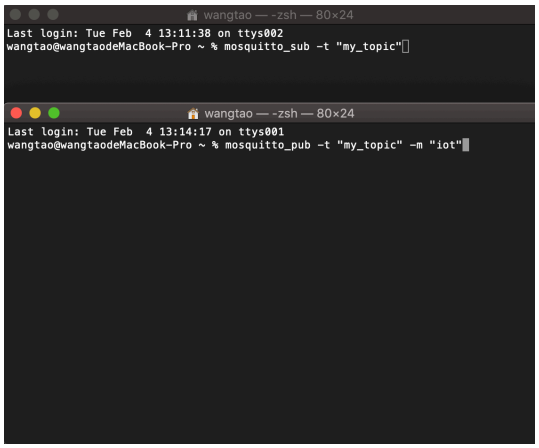

Figure 7 Publisher and Subscriber

We open two new Terminal and input the commend, make one to be a publisher and other is subscriber (Figure 7).

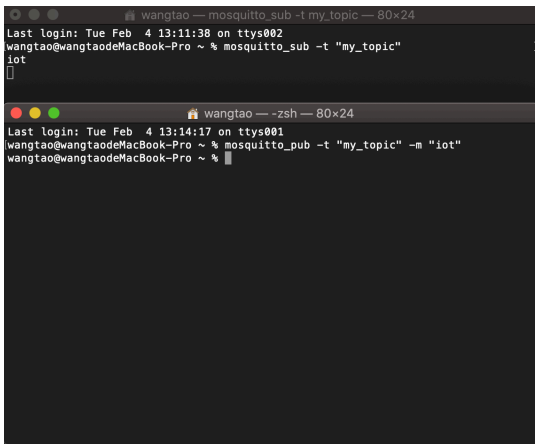

Figure 8 Results

And then we can see the result, the message is showed in subscriber (Figure 8).

3.3 EMQ installation

Step1: In the step, it also need to install the brew.
Step2: Add the tag for EMQ X [17](Figure 9).

Figure 9 Add the Tag

Step3: Input the commend: brew install emqx (Figure 10)

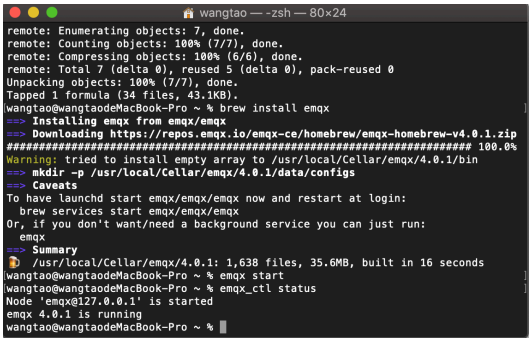

Figure 10 Start EMQ

Step3: Next input the commend: emqx start, emqx_ctl status to run emqx (Figure 10).
Step4: http://127.0.0.1:18083 to use EMQ X's web management control interface. The default login is admin and the password is public [15].

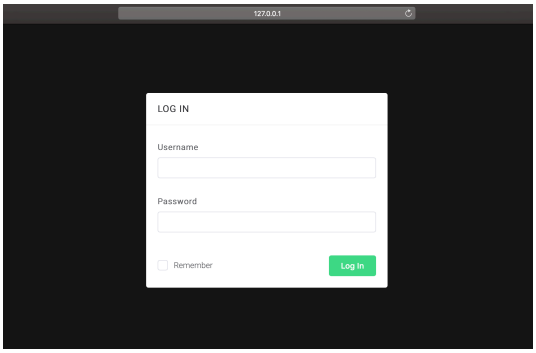

Figure 11 EMQ Register

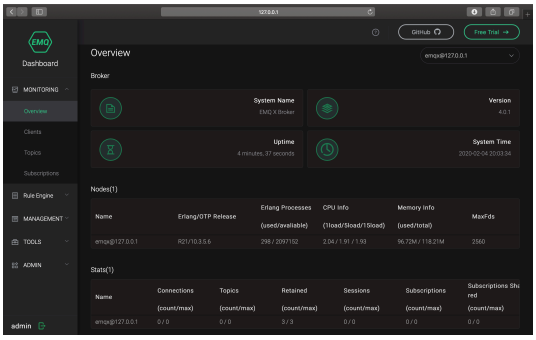

Figure 12 Interface of EMQ

And from the web, you can check how many connections there are and how many client there are. For client side, we use MQTTBox that is a cross platform application available on Chrome, Linux, MAC, Web and Windows [16] (Figure 13).

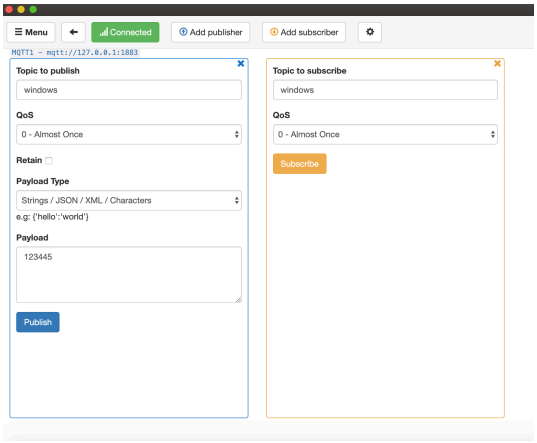

Figure 13 Interface of MQTTBox

4 Evaluation

For mosquito, that is an MQTT open source agent also supports the CoAP protocol. And have strong data processing support, you can add modules independently (for example: user authentication or server data storage). However, it's not intuitive enough to display, it not has management interface like EMQ.

For EMQ, 1. Simple deployment; 2. Support cluster deployment; 3. Comprehensive official documents; 4. Easy to get started; 5. Million-level distributed open source IoT MQTT message server.

And for EMQ, it also be used for Edge Computing [15] (Figure 14).

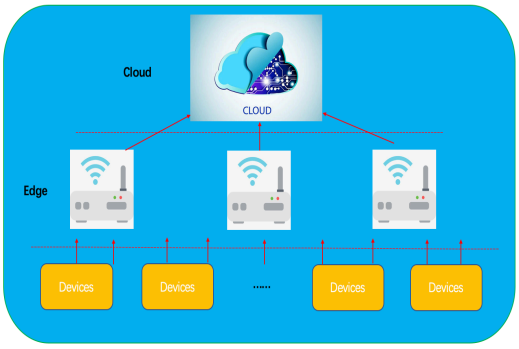

Figure 14 Framework of Edge Computing

The edge message server is the bridge between the cloud and local devices. As a message broker running on the edge computing nodes in the IoT platform system, the business rules engine is used to process most of the data locally on the edge nodes, and it can also respond the request from Device-side in real time [17].

There is not much difference in software functions between the edge platform and the cloud platform. The main difference is the way of processing data and the ability, and we can implement the edge computing platform through the bridge mode.

5 Conclusion and Future Work

This paper firstly focuses on the protocol MQTT (Message Queuing Telemetry Transport) used in the IoT (Internet of Things), which mainly includes the framework model, main features and data packet structure. The installation of two related software is also introduced. The purpose of building a simple Internet of Things on a personal computer is achieved through the installation of the software.

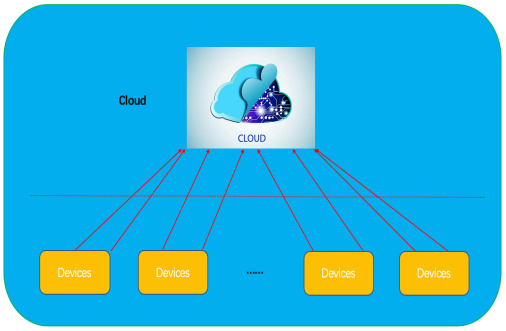

Figure 15 Framework of system-2

Regarding the future work, we plan to use the software that we discussed above and related IoT devices to build an edge computing platform with a certain scale in the laboratory (referred to as system-1, in this system we place the detection method in the platform like figure 14). As a comparison, the other system we directly place the detection method in IoT devices (referred to as system-2 like figure 15). And we plan to use DDoS and other related network attacks to attack systems 1 and 2 respectively. By comparing and analyzing the related values of memory consumption, energy consumption, time spent, and accuracy rate of each device in the systems, and through the results, we analyze the role of edge computing in IoT security.

Reference

[1]. Cisco, "Cisco Visual Networking Index (VNI)," Globle Forecast Update, pp. 1–35, 2017.

[2]. B. S. Adiga, P. Balamuralidhar, M. A. Rajan, R. Shastry, and V. L.Shivraj, "An Identity Based Encryption Using Elliptic Curve Cryptography for Secure M2M Communication," in Proceedings of the First International Conference on Security of Internet of Things, ser. SecurIT' 12. ACM, 2012, pp. 68–74.

[3]. IBM Developer homepage https://www.ibm.com/developerworks/jp/iot/library/iot-mqtt-why-good-for-iot/index.html

[4]. Y. Upadhyay, A. Borole and D. Dileepan, "MQTT based secured home automation system," 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, 2016, pp. 1-4.

[5]. M. Singh, M. A. Rajan, V. L. Shivraj and P. Balamuralidhar, "Secure MQTT for Internet of Things (IoT)," 2015 Fifth International Conference on Communication Systems and Network Technologies, Gwalior, 2015, pp. 746-751
[6]. Introduction of MQTT server https://github.com/mqtt/mqtt.github.io/wiki/servers
[7]. Z. Shelby, Sensinode, K. Hartke, C. Bormann, and U. B. TZI, "Constrained application protocol (coap) draft-ietf-core-coap-17," http://tools.ietf.org/html/draft-ietf-core-coap-17, 2013.
[8]. M. B. Yassein, M. Q. Shatnawi and D. Al-zoubi, "Application layer protocols for the Internet of Things: A survey," 2016 International Conference on Engineering & MIS (ICEMIS), Agadir, pp. 1-4, 2016.
[9]. Q. Sarhan and I. Gawdan, "Java Message Service Based Performance Comparison of Apache ActiveMQ and Apache Apollo Brokers" , sjuoz, vol. 5, no. 4, pp. 307-312, Dec. 2017.
[10]. Introducing the MQTT Protocol https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/
[11]. P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kernmarrec, "The many faces of publish/subscribe," ACM Computing Surveys, vol. 35, no. 2, pp. 114–131, June 2003.
[12]. P. Saint-Andre, "Streaming XML with Jabber/XMPP," IEEE Internet Computing pp. 82-89, Oct 2005.
[13]. Eclipse Mosquitto homepage https://mosquitto.org/
[14]. Mosquitto https://github.com/eclipse/mosquitto
[15]. EMQ homepage https://www.emqx.io
[16]. S. Ahmed, A. Topalov and N. Shakev, "A robotized wireless sensor network based on MQTT cloud computing," 2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), Donostia-San Sebastian, pp. 1-6, 2017.
[17]. EMQ' s document. https://docs.emqx.io/broker/latest/en/