MQTT and IoT

Safir Mohammad Mustak Shaikh
High Integrity Systems
Frankfurt University of Applied Sciences
Hessen, Germany
Matriculation number: 1322554
safir.shaikh@stud.fra-uas.de

***Abstract*—Internet of Things (IoT), also known as Web of Things (WoT) is an ever-emerging technology which aims at digitalizing the world by connecting different devices over the internet. To achieve this, IoT makes use of several standard protocols for different scenarios such as CoAP, XMPP, MQTT and AMQP. This paper covers MQTT protocol and a real-world example of Deutsche Bahn (German Railways) that makes use of MQTT. MQTT not only considers technical aspects, but also safety-critical and safety-related approaches. MQTT was designed to run on low-end and battery-operated sensor/actuator devices and operate over bandwidth-constraint WSNs like ZigBee based networks. MQTT is many-to-many IoT protocol for transferring messages between clients through a common broker with support to Persistence running over TCP. It is extremely light-weight, flexible to grow and shrink, publish-subscribe communication mechanism for the devices that are decoupled and built with small code footprint and minimal bandwidth.**

***Keywords*—** MQTT, IoT, Publish-Subcribe

1. Introduction

IoT is a network of objects like devices, machines, people embedded with sensors, actuators, software's for communication over the internet. Since all the devices are connected over internet, communication plays a vital role here and it is implemented according to the protocol. A communication protocol defines a set of rules, syntax, synchronization and semantics that the communicating parties must agree on. These protocols are published by several standards organizations like IETF, IEEE, ISO, ITU-T to maintain consistency and universality for transmitting messages. Everything was working well until some of the constraints and issues/challenges like interoperability were identified. In most of the IoT use cases, the sensors and actuators are placed at dangerous locations like in between deep sea or in the middle of a forest where it's difficult to reach very often and not affordable as well. But in case the sensor runs on battery power, then these locations need to be visited very often to change the batteries. Millions of such devices exist at such places and cannot be managed easily. Moreover, there is limited internet/network bandwidth at such places and traditional protocols like XMPP (Extensible Messaging and Presence Protocol) and HTTP (Hypertext Transfer Protocol) consume more bandwidth and electricity/battery power due to large payloads. And since the payloads are large sized, they take much time to exchange information. Also, it was a challenge for devices from different domains to communicate smoothly over the internet using traditional industrial communication protocols. Including above, other issues like security and reliability gave rise to the need of a communication protocol to be specifically used in IoT applications which will work efficiently in such areas by consuming less power and optimizing less network bandwidth, providing security and reliability as well according the severity of the application. And then MQTT was adopted as the Standard for IoT Messaging by OASIS (Organization for the Advancement of Structured Information Standards) which overcame all the drawbacks discussed above and delivered exceptional results and is currently being used in large organizations as a standard. MQTT is an application layer protocol that runs over TCP (Transmission Control Protocol). The message size is limited in MQTT (maximum 260 MB) and therefore it is a light-weight protocol and since it carries smaller messages, it does not consume much power and transmits messages much faster. It connects devices from different domains easily over the internet since one of the aims of MQTT was enhancing Machine-to-Machine communication.

2. MQTT

MQTT (Former Acronym: MQ Telemetry Transport/Message Queuing Telemetry Transport) is an open OASIS and ISO/IEC 20922 [1] standard communication protocol. It is a light weight and publish-subscribe network protocol to transport messages between devices that runs over TCP/IP most of the times, however, it supports any network protocol providing lossless, bi-directional and ordered connections. It was first introduced in 1999 by Andrew James Stanford-Clark and Arlen Nipper. This version was bandwidth-efficient and lightweight used to monitor an oil pipeline through desert via satellite. The primary focus of MQTT was Embedded Systems and later it changed to Internet of Things. Moreover, it utilizes low battery power and this is how it made itself perfect solution for IoT applications.

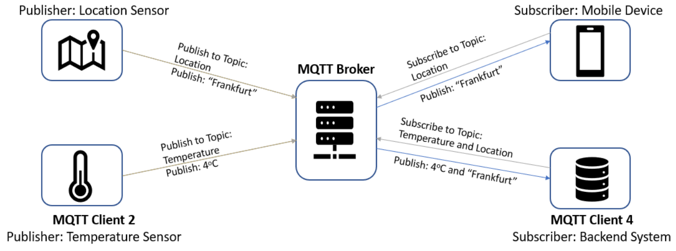
MQTT Client 1
Publisher: Location Sensor
MQTT Client 3
Subscriber: Mobile Device
Publish to Topic: Location
Publish: "Frankfurt"
MQTT Broker
Subscribe to Topic: Location
Publish: "Frankfurt"
Publish to Topic: Temperature
Publish: 4ºC
Subscribe to Topic: Temperature and Location
Publish: 4ºC and "Frankfurt"
MQTT Client 2
Publisher: Temperature Sensor
MQTT Client 4
Subscriber: Backend System

Figure 1. MQTT Publish-Subscribe Architecture

MQTT protocol possesses 2 components as depicted in its architecture, MQTT Client and Broker.

2.1. Client

In MQTT, two kinds of clients exist, Publisher and Subscriber. MQTT client can be a publisher, a subscriber or both publisher and subscriber at a time. The publisher publishes messages whereas the subscriber receives messages. The messages are exchanged in the form of packets.

2.2. Client Connection

Initially, the client sends 3 messages to the broker, “BIRTH”, “DEATH” and “LAST WILL AND TESTAMENT”, when it connects for the first time along with “Keep Alive Time”. To connect to the broker, each client first needs to send CONNECT packet to the broker. The broker then sends the CONNACK packet to the client for acknowledgement. This is just one-time activity and unlike other protocols, client does not need to establish the connection every time it connects. Once the connection is established, publisher client can send data through PUBLISH packet to the broker and subscriber client can subscribe to a topic by sending SUBSCRIBE packet to the broker. The clients publish only when there is a change in the value. When connection is established, broker forwards BIRTH message under set topic to let all other subscribed clients know that there is a new device in the system. When client ends its session with broker, broker forwards DEATH message and informs all the subscribers; while the client is still connected. Broker regularly checks with the client using the 2-byte Heartbeat packet and if it doesn’t hear a heartbeat for specified length of time (Keep Alive Time), it sends the last WILL message to all the subscribers.

2.3. Publish-Subscribe

In MQTT, Communication is carried out using an easy-to-implement Publish/Subscribe mechanism. Any MQTT client can send (Publish) messages to other MQTT client(s) and MQTT client can be any device. The client can also subscribe to its Topic of interest and receive messages whenever a new message is published on the subscribed topic. Message is the information exchanged between the devices which can be either a command or data.

2.4. Topic

Topics hold special value in this protocol since they are used to uniquely identify different message groups. They are the way to register interest for incoming messages and the way to specify about published messages. Topic is represented by a ‘UTF-8 String’ and separated by a ‘/’ (Slash), which indicates the level of topic, similar to how a directory structure is represented in a file system. Moreover, they are case-sensitive. Topic is the key to filter out messages, for example, to distinguish temperature sensors in 2 rooms, 2 different topics can be used, room1/temperature and room2/temperature, to distinguish between temperature and humidity of 1 room, topic used would be, room1/temperature and room1/humidity.

2.5. Broker

Broker is the key element of this protocol and plays vital role during the communication. It is responsible for receiving all messages, filtering the messages and broadcasting the messages to subscribed clients. Broker is nothing but a software running on a computer and the computer can be any device from Raspberry PI to PC to Server. The software can be running on premise or in the cloud and it can be self-built or hosted by 3rd party. It is available as either open source or proprietary with additional features. HiveMQ and Mosquitto are most widely used open-source broker systems.
Broker uses transport layer protocol for establishing connection with clients. It is responsible for eliminating vulnerable and insecure client connections, managing and tracking all client connection states including security credentials and certificates and it serves all these functionalities without compromising security and without putting strain even on large networks like cellular and satellite networks. Also, there is no limit on the number of incoming and outgoing messages at/from broker since the load of the clients can be shared across multiple servers onsite or cloud or both. And most of the broker software and clients have an automatic handover to redundant and backup broker if the primary broker goes down. It requires only one port to be open, i.e. “Secure Port 8883”. Broker supports both

1) Standard MQTT message and topics
2) MQTT compliant specifications e.g. Sparkplug specification

and it does it at same server at the same time with same levels of security. It doesn’t support long term storage and doesn’t keep historical log of messages, however, to do that, a database client, server etc. can be configured. But it provides a service “Retained Messages” using which newly joined clients fetch the last known value as it stores the most recent value [2]. Since clients go on and offline, it keeps track of all current sessions and can even store session information for long time. These are called “Persistent Sessions” that reduce the time to establish the connection again between client and broker and make MQTT strong network model.

Consider an example illustrated below, a lamp present in your Home Office is specified as home/office/lamp. The MQTT client ESP 8266 has subscribed to the topic ‘home/office/lamp’ and The MQTT client Node-RED publishes a message “ON” to the same topic. Then, the Broker Mosquitto receives the message and sends it to the subscribed client. As soon as the subscribed client gets this message, it turns the lamp ON.

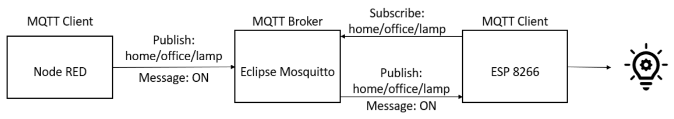

Figure 2. MQTT Scenario in IoT

2.6. MQTT 3.1.1 Standard Packet Structure

As covered above, Communication in MQTT takes place in the form of packets. The figure shown below illustrates the standard MQTT packet structure. The structure varies according to the message type. In MQTT, there exist 3 possible packet formats,

1) with Fixed Header,
2) with Fixed Header + Variable Header and
3) with Fixed Header + Variable Header + Payload.

The fixed header consists of 2 fields, 1 byte Control Header and 1-4 bytes Packet Length. The Control header field contains all commands and responses and is in turn divided into 2 fields, Packet Type and Control Flags, 4 bits each. For the Control Flags field, 16 flag options are available but only few options are used practically and most of them are used by Publish message. For example, the duplicate flag (DUP) allows clients to republish a message, the Quality of Service flag (QoS) indicates QoS level 0, 1 or 2 and the RETAIN flag is used to retain the packet at broker. The MSB of each byte in Remaining Length field is used as a Continuation

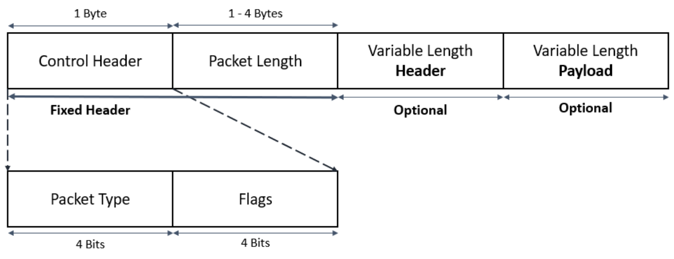

Figure 3. MQTT Packet Structure

flag. The variable length header field is used to carry additional control information such as CONNECT message control code [3].

2.7. Communication Type

MQTT protocol is not just limited to One-To-One communication and it also supports One-To-Many and Many-To-One modes of communication. There is no direct connection between the clients but only one Out-Bound connection from the client to the broker. Every MQTT client is connected to and via the broker. Broker handles all these scenarios (1-1, 1-M, M-1) easily using the topics to manage the messages. Each client can both produce and consume data by both publishing and subscribing, hence, MQTT is a bi-directional protocol which makes it powerful for both sharing data as well as managing and controlling devices. Broker is the central key to this powerful solution.

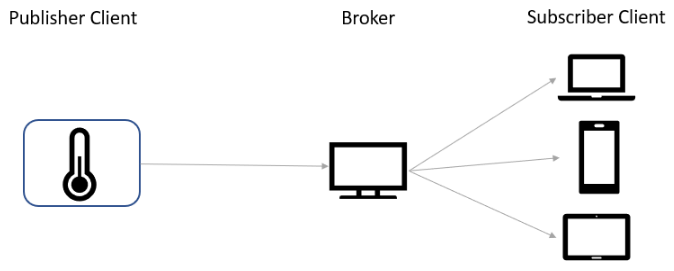

Figure 4. One-To-Many Structure in MQTT

As depicted in the figure above, MQTT can establish One-To-Many connection in which, one node sends messages to multiple nodes. Data producer sends just 1 message and broker will send many messages, 1 to each subscriber. In this way, one device can control thousands of devices with just one command.

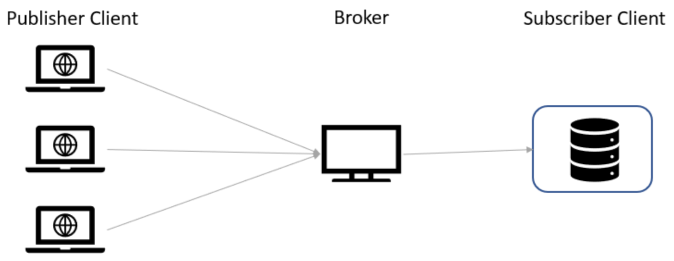

Figure 5. Many-To-One Structure in MQTT

Moreover, MQTT can also build a network in which multiple systems publish data to one system like Database or Server by making it Many-To-One mode of communication. In this scenario, multiple MQTT clients publish messages whereas there exist only 1 subscriber client. In this way, it can easily scale to any level without affecting any parameter of the network.

2.8. MQTT Features

Apart from already covered services MQTT provides all the features which are key requirements in IoT, therefore, it is best suited for IoT. The clients are small in size, require minimum resources to run and can easily run on small microcontrollers. Moreover, they utilize low network bandwidth due to small message headers. Using MQTT, it has become easy broadcasting between groups of things since MQTT allows communication from cloud to devices and devices to cloud. One of the reasons behind its success is that it can scale to connect millions of IoT devices. It covers almost all fields/application areas, and thus, it provides reliable message delivery using its own defined QoS levels for different use cases. The 3 levels, 'QoS 0' (Fire and Forget), 'QoS 1' and 'QoS 2' depict "Delivery at most once", "At least once" and "Exactly once" respectively. QoS level is chosen by the clients based on network reliability and application logic. Since MQTT offers guaranteed delivery and retransmission of messages, communication even in unreliable networks has become efficient since many IoT devices are connected over unreliable cellular networks. During transmission, MQTT encrypts messages using TLS and makes use of modern authentication techniques like OAuth. The figure shown below illustrates the structure of communication of QoS levels.

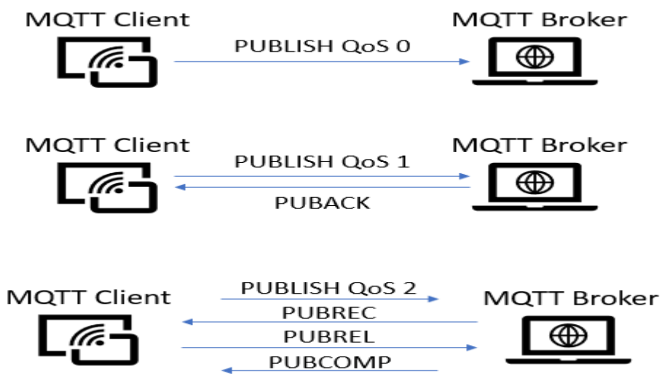

Figure 6. QoS0, QoS1 and QoS2 in MQTT

QoS2 makes use of 4-way handshake to acknowledge the message and it is used in Safety-Critical and Mission-Critical systems since it doesn't allow duplicate messages. For example, a single duplicate message received (even unintentionally/by mistake) in an Artificial Pacemaker may lead to patient's death/severe damage since the heart would beat faster/slower than the expected rate.
It has irreplaceable features as compared to traditional protocols. In traditional systems, since devices and systems are tightly coupled to each other with abundant connections and require many open ports, a change to even one element requires a lot of man hours and to go into many other systems whereas MQTT is decoupled, more secure and each client is unaware of the IP addresses and the domains that the other clients operate on. Comparing MQTT with HTTP, MQTT leaves HTTP behind when it comes to sending many messages by reusing single connection in order to improve average response time and amount of data transferred per message. Comparing it with another IoT protocol, CoAP (Constrained Aplication Protocol), CoAP is 1-1 protocol that runs over UDP and is not event based protocol since it is a state transfer model.

2.9. MQTT in Action

Due to its architecture and the services it provides, it is widely used in plethora of application areas. The major areas include

1) In Automotive industry, HiveMQ is used for reliable connection in applications like BMW Car Sharing since Network Reliability is the main issue for Service Reliability [4].

2) In Logistics business, real time monitoring is the basic requirement. Also, the way goods are being transported is changing and now-a-days it is done with the help of autonomous drones. Recently, head of Platform and Flight Safety at Matternet has confirmed that HiveMQ reduces the transportation time by almost 75% using Matternet [4].

3) In Manufacturing field, MQTT is implemented in many powerplants, like Celikler Holding's Power Plant Monitoring and Data Transfer. MQTT overcame many drawbacks in traditional approach like Absence of Probity, Secrecy and Authentication [5].

4) IoT is most widely used in Smart Home Applications and MQTT has achieved great success in many usecases. For example, IBM Telemetry use case in Home Energy monitoring and control, Home Patient monitoring and the eFon Technology's Smart Home Security System.

5) Consumer products such as smart kitchen appliances by CASO

6) Oil & Gas

3. Case Study: Deploying IoT on Germany's DB Railway System [6]

Deutsche Bahn (DB) is one of the biggest transportation companies in the world and DB Systel is its internal ICT partner. Their aim of digitalizing the system made them switch from M2M (Machine-to-Machine) solution to IoT technology since M2M systems like SMS was very expensive to run and maintain. Since they had many choices, after thorough evaluation they finalised to use combination of HTTP and MQTT as the communication standard for the trains and their monitoring systems. They opted for MQTT's open source libraries provided by Eclipse Paho. Alexander Schmitt, Chief Architect at DB Systel, also confirmed the effective use and advantages of open source technology in such a large, state owned organisation. It's been 5 years and they are feasibly using services from Eclipse. Since Eclipse Equinox and EclipseLink were already present in DB trains, adding Eclipse Paho (to provide MQTT Client Support) to the technology stack was comfortable and simple. And IBM MessageSight acted as MQTT broker at the backend. Here are some of the use cases:

1) **Long Distance Trains**
The long-distance trains in Germany use a mix of Eclipse Paho, MQTT and Java Client for communication. The train sends real-time information like location, estimated delay. Using this data, the control room updates connection details and pushes the details to DB navigation applications and on the monitors present inside the train. Approximately 400 long-distance trains send update their location details every 10 seconds to IBM MessageSight server present in the DB data center. In addition to this, regional and cargo traffic locomotives send their location data every 10 seconds which is used for predictive maintenance, forecasting etc.

2) **Dynamic Text Displays**
DB has installed smaller LED text displays at the stations with usually a smaller number of passengers and the displays utilize less power and are custom designed version of Eclipse Paho. There are such 6500 devices installed across Germany and each edge device receives 25 messages per second.

3) **Escalators & Elevators**
All elevators and escalators at DB train stations are monitored using multiple sensors that report their operations like proper working, power consumption, door opening/closing etc. In this scenario, about 3000 devices send 10 messages per second to the central monitoring system.

4) **Eclipse Mosquitto on DB ICE Trains**
Earlier, before adopting this technology, DB faced some issues in implementing software changes on trains, tightly coupled systems and bundled software and hardware etc. And the most important challenge was to differentiate between non-safety systems and safety related systems in the train. As a solution to this problem, DB started a project in 2017 called "Variable Vehicle IT" that used IaaS on the trains. A new light-weight MQTT broker, Eclipse Mosquitto, was used in this project. This implementation surpassed the count of messages transmitted using Eclipse Paho.
In the near future, more "Things" will be used so that DB would be equipped with more agile and better applications keeping safety measures in place.

4. Limitations

1) As MQTT is a light-weight protocol due to its small message size, it is not possible to send messages with heavy payload size. The message size varies from 1 byte to 260 Megabytes.

2) As compared to HTTP, there is a little overhead involved during connection establishment when the client joins the network for the first time.

5. Conclusion

MQTT is widely accepted as a standard communication protocol which provides application centric benefits that perfectly fit in IoT use cases and devices. Features like reliable communication, light-weight message size, highly secure publish-subscribe, less power consumption, network bandwidth optimization are the reasons behind its success. In the future, MQTT may provide advanced TCP features like Flow Control to become the best protocol.

Acknowledgment

This research was possible under the guidance and direction of Prof. Dr. Matthias Wagner and tutor Fatima Butt. This paper is the final report for the course 'Seminar: Internet of Things' taught in the Master's programme High Integrity Systems (HIS).

References

[1] International Organization for Standardization, "ISO/IEC 20922:2016 Information technology – MQ Telemetry Transport (MQTT) v3.1.1," June 2016. [Online]. Available: iso.org

[2] N. D. Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*. IEEE, Nov. 2013. [Online]. Available: https://doi.org/10.1109/scvt.2013.6735994

[3] Andrew Banks and Rahul Gupta, "MQTT Version 3.1.1," October 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[4] HIVEMQ, "HIVEMQ Case Studies," 2012. [Online]. Available: https://www.hivemq.com/case-studies/

[5] Hema, "MQTT Implementation on Celikler Holding's Power Plant Monitoring," 2020. [Online]. Available: https://www.bevywise.com/blog/iot-success-stories-mqtt-broker-celikler-holding

[6] Eclipse Foundation, "Case Study: Deploying IoT On Germany's DB Railway System — IoT Development Made Simple - Eclipse IoT," 2020. [Online]. Available: https://iot.eclipse.org/community/resources/case-studies/iot-on-railway-systems-db/