MQTT Protocol for the IoT – Review Paper

Technical Report · May 2021

DOI: 10.13140/RG.2.2.26065.10088

1 author:

Mohammad Faiz Usmani
Frankfurt University of Applied Sciences
3 PUBLICATIONS **3** CITATIONS

SEE PROFILE

MQTT Protocol for the IoT

Mohammad Faiz Usmani
High Integrity Systems
Frankfurt University of Applied Sciences
Hessen, Germany
Matriculation number: 1323197
mohammad.usmani@stud.fra-uas.de

***Abstract*—The Internet of Things (IoT) is modernizing the world by connecting devices with sensors, actuators, and softwares with each other over the internet. For the exchange of messages, there are several protocols being used in IoT like MQTT, CoAP, HTTP, etc. This paper explains the concepts and advantages in detail of the lightweight, reliable, and power-efficient protocol 'MQTT' working on the publish/subscribe model of data exchange. Then it will dive into some real-world use cases of how MQTT is used by established corporations at a large scale. It will also critically analyze MQTT's performance with similar protocols based on various factors.**

1. Introduction

The Internet of Things (IoT) is a network of physical devices embedded with sensors, software, actuators, and other technologies connected over the internet with systems and other similar devices and constantly sharing data with each other. As these devices operate over the internet, reliable and efficient communication is the key for the successful functioning of these devices. For that reason, there were protocols laid out by the Standard Organizations for the IoT devices to send and receive data over the internet. Every IoT device must adhere to the set protocol for successful operation. But these devices (mostly consisting of sensors and actuators) came with their own set of challenges. They are often placed in such extreme and isolated areas where constant human intervention is not possible. In addition to that, there is limited internet bandwidth present at these places and popular protocols like Hypertext Transfer Protocol (HTTP) would not be ideal for constant communication. For example, Pressure sensors placed in Ocean beds operating on batteries require a lightweight protocol because they cannot use HTTP for communication as the packet size would be large requiring more bandwidth consequently resulting in more power consumption and faster battery drainage. Moreover, various industry-specific protocols posed a problem of not having smooth interoperability between devices of various domains. These challenges and the need for a standard lightweight messaging protocol between Machine-to-Machine communication lead to the introduction of MQTT.

2. MQTT

MQTT stands for Message Queuing Telemetry Transport. Designed and developed by Arlon Nipper and Andy-Stanford-Clark in 1999 for connecting Oil Pipeline data measurement and transfer systems via satellite. MQTT was a proprietary protocol initially that became royalty-free in 2010 and finally in 2014 was declared as the standard protocol by the Organization for the Advancement of Structured Information Standards (OASIS) [1]. Its core architecture consists of two components- Client and Broker. The Centralized Server known as Broker is connected to various devices which are Clients and transfer messages to and fro from them based on the publish-subscribe model. It can work over any network protocol providing bi-directional and lossless communication but in most cases, it is used by devices operating over (Transmission Control Protocol/Internet Protocol) TCP/IP. Looking over various layers of TCP/IP and the protocols categorized in different layers based on their working, MQTT is an application layer protocol. In the TCP/IP model, application layer is the topmost abstraction layer providing interfaces and protocols for the exchange of information between various devices.

2.1. Advantages of MQTT

The attributes that made MQTT hugely popular are- open source, being lightweight, faster transmission speed, less power consumption, event-oriented architecture using publish/subscribe model, easier to use and implement in both client and server with simple commands, reliability due to Quality of Service (QoS) levels, multicasting, better congestion control mechanisms, built-in support between client and broker for loss of connection (persistence of messages), and ability to connect a large number of devices to the broker without them being aware of each other [2].

3. Concepts of MQTT

This section explains the various terminologies and functionalities, it also aims at understanding how some of the advantages mentioned in Section 2.1 are implemented in MQTT.

3.1. Publish Subscribe Model

In the Publish-Subscribe model, the client is decoupled with various other clients present in the same ecosystem, meaning it does not know about the existence of other clients. It can subscribe to a relevant topic and receive messages pertaining to it, also it can publish messages to a certain topic so that other clients can subscribe and consume the information. This decoupled architecture is achieved by introducing a message broker in between which acts as a middleman between various clients as all the information passes through it [3].

3.2. Client

In MQTT there are two kinds of clients- a Publisher, and a Subscriber. Depending on the need a client can be a publisher, a subscriber, or both subscriber and publisher at a time. There are various client libraries along with the set of supported features available in various programming languages. These client libraries enable a device to become an MQTT client. Meaning a device operating over a TCP/IP connection and having an MQTT client library can publish/-subscribe to messages via a central server. For example, an Android phone, a Macbook, Arduino board, etc.

3.3. Broker

Sometimes also referred to as Server, a Broker acts as a mediator between various clients connected to it. The connection between clients and broker is established over TCP and Broker can handle a large number of messages coming in from clients asynchronously. The main tasks of a broker are as follows- receiving messages from the client, filtering and storing them, broadcasting relevant messages to the subscribed clients, managing connection states including credentials and certificates of the connected clients, maintaining persistence sessions so that a client can easily reconnect after going offline, removing bad and vulnerable client connections, retaining messages so that they can be sent to the new subscribers. Any machine whether it is on-premise or on the cloud with OS like Windows, Linux, MacOS installed in it can act as MQTT brokers. There are various free, open-source, and paid broker services available. Mosquitto, IBM Websphere MQ Telemetry, HiveMQ are some of the popular broker services.

3.4. Topics and Wildcards

Broadly, a topic can be said as the categorization of the messages into various subject areas. Topics are important in MQTT because the messages published are distributed into various groups and the topic name uniquely identifies each message group. A client can subscribe to multiple groups to receive their messages. The topic is represented by a Unicode Transformation Format 8-bit (UTF-8) string. For example, '/temperature' denotes a topic. Any level of subtopics can be created using the '/' character and are typically termed as 'topic trees'. Example, '/temperature/Celsius' A string is a set of characters that identifies a topic of a publish-subscribe model [3]. Wildcard characters can be used here to subscribe to multiple topics at once. The wildcards can be multilevel(#) or singlelevel(+). For example, subscribers of '/temperature/#' will receive messages of topics '/temperature/Fahrenheit' and '/temperature/Celsius' and '/temperature/Celsius/Negative' while subscribers of '/temperature/+' will receive messages from '/temperature/Fahrenheit' and '/temperature/Celsius' and not from '/temperature/Celsius/Negative'.

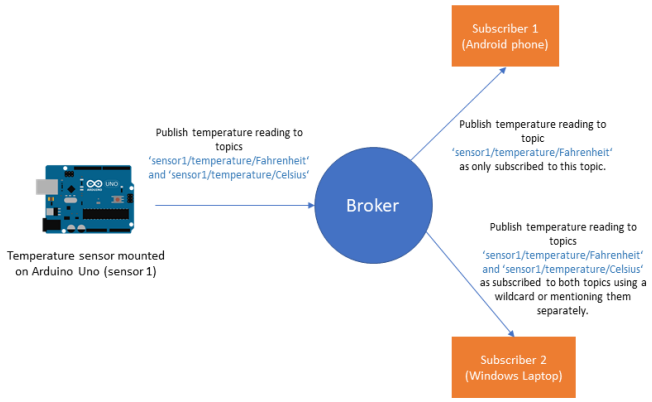

Figure 1. MQTT Publish Subscribe Scenario

3.5. QoS levels

There are three Quality of Service (QoS) levels defined by MQTT for message delivery. They are QoS 0: at most once, QoS 1: at least once, and QoS 2: exactly once. With each increase in the level denoting the increase in effort by the server to make sure that the messages are delivered. Higher QoS levels contribute to reliability in message delivery but on the downside, it may consume more bandwidth and lead to message delays. In QoS level 0, the Broker does not send any acknowledgement after receiving the message and the message is sent at most once. It is called 'Fire and Forget' because there is no guarantee of message delivery, it is usually used when there is trust in the network connectivity and speed as it ensures the best delivery. In QoS level 1, a message can be sent multiple times, hence the naming 'atleast once'. The broker sends an acknowledgement after receiving the message. QoS level 2 uses a four-way handshake mechanism to ensure that the message is delivered exactly once, it is the safest but the slowest method and used in cases when the system can not afford to have duplicate messages [3].

3.6. Clean Session and Wills

Optional to use, A 'clean session flag' is set by the client whenever it connects to the broker. All of the client's subscriptions are deleted when it disconnects from the server if the value of the flag is true [3]. If the value is false, the client's subscriptions are persisted even if it disconnects, so the messages with higher QoS levels are stored and delivered in the case of re-connection. The client can also tell the server that it has a 'will', implying that in case of an

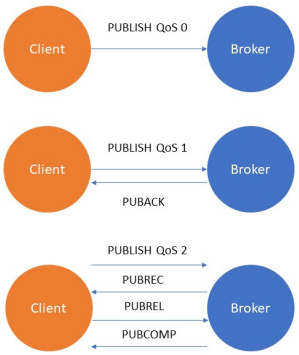

Figure 2. QoS levels in MQTT

unforeseen disconnection a message would be published by it to a specific topic(s) [3]. This can be helpful for sending alert messages when an important device disconnects from the network due to a bad connection.

3.7. Connection Process between Client and Broker

1) When a client wants to connect to the broker. It sends out the CONNECT packet with a payload that holds all the required information to start the connection and authenticate/authorize itself.
2) The Broker checks the CONNECT packet and performs the authentication, if everything is correct it will acknowledge the connection by sending back the CONNACK control packet to the client. The Broker will automatically close the connection if the CONNECT control packet is invalid [4].
3) The Client can close the connection by sending the DISCONNECT control packet to the Broker.

The structure of the CONNECT control packet is as follows:

- ClientId: A unique identifier used by the broker to identify clients. Each client sends its unique client id to the broker, if it is empty the id is generated by the broker depending on the value of the 'CleanSession' field.
- CleanSession: As described in Section 3.6 this boolean field describes what will happen to the state of the connection if the client disconnects.
- UserName: If there is a need to provide a username by the client for authentication, the UserName flag is set to true and the corresponding value is entered in the UserName field.
- Password: If there is a need to provide a password by the client for authentication, the Password flag is set to true and the corresponding value is entered in the Password field.
- ProtocolLevel: Indicates the MQTT version being used. For example, version 3.1.1.
- KeepAlive: A time field expressed in seconds. If the value of this is greater than 0, the client must send a Control Packet to the broker within the time specified. If it does not have anything to send, it must send a PINGREQ control packet to tell the broker that the connection is alive, the server will acknowledge it and send back the PINGRESP response packet [5]. If there is an absence of packets, the server will close the connection. This whole functionality of Keep Alive is turned off if the value of the KeepAlive field is 0.
- Will, WillQoS,WillRetain,WillTopic, WillMessage [5]: If the Will flag is set to 1, it will enable the functionality mentioned in the Section 3.6. The QoS level for the last message is specified in the WillQoS flag. Should the message be retained by the broker is set through the WillRetain flag and the actual message with its topic are specified by the WillMessage, WillTopic flags respectively.

The CONNACK control packet sent by the server contains the following fields:

- SessionPresent: If the value in the CleanSession flag set by the client was 1, the value in SessionPresent would be 0 and vice versa. The value 1 in this field indicates that the Broker is dealing through a persisted session with a Client.
- ReturnCode: If a connection is acknowledged by the Broker after authorization and authentication, its value would be 0. In case of errors, the value would range from 1 to 5, with the digits relating to the following errors: 1- Broker does not support the MQTT version requested by the client, 2- Specified ClientId was rejected, 3- MQTT service is not available, 4- Username or Password values are invalid, 5- Authorization failed.

3.8. Packet Format of MQTT

The packet structure of MQTT is byte-based and contains three parts: fixed header, variable header, and the payload [6].

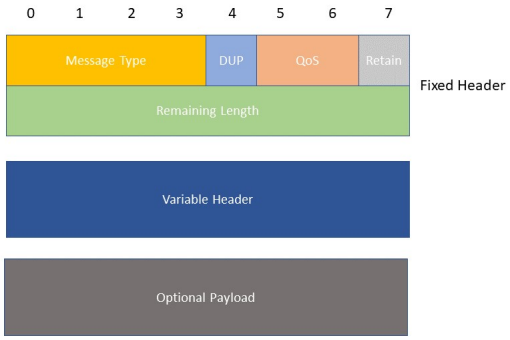

Figure 3. Packet Structure of MQTT

- The first four bytes in the fixed header section indicate the type of message. For example, CONNECT, PUBLISH, SUBSCRIBE message.
- The next 1 byte is a flag that indicates if the message is duplicate or not.
- The next 2 bytes are used to designate the QoS level.
- The last 1 byte is the RETAIN flag indicating the broker to keep the message for future use.

- The Remaining Length field indicates the length of the next subsequent variable sections.
- The Variable Header section contains the additional information based on the message type. For CONNECT packet it contains all the additional fields mentioned in Section 3.7 like flags for Username, Password indicating if their values are present in the payload. And for the PUBLISH packet, it has: topic name, the most and least significant bytes of the length of the topic string along with message ID.
- The payload section contains usually the data to be published in the Broker or the name of the topics to be subscribed from the Broker.

4. Some Real World Use Cases

4.1. Celikler Holding's Power Plant Monitoring [7]

Celikler Holding a big corporation based in Turkey dealing in the sectors of energy, construction, etc. used MODBUS (lightweight communication protocol) for monitoring power generation of the power plant and viewing the status by connecting individual phones to the control dashboard. Because of the absence of built-in security features and even basic authentication mechanisms, they were unable to take it directly to the internet. Because of this employees had to be present in the vicinity of the power plant to view the data, and this was a major drawback. The company needed a protocol as lightweight as MODBUS along with good security features. The company started using MQTTRoute (A ready to deploy MQTT broker) which has device-level authentication and maintains secure connections using TLS/SSL along with MODBUS MQTT Gateway to connect with the machines sending the data in the MODBUS format and translate it into the MQTT format. Thanks to this, the employees can view the data of the power plant through dashboards over the internet and are not needed to be always present near the power plant.

4.2. Enabling IoT in Deutsche Bahn Germany [8]

DB Systel which is Deutsche Bahn's Information and Technology partner wanted to make a switch from solutions based on communicating via machine-to-machine (M2M) Short-Message-Service (SMS) to IoT technologies. Their main point of consideration was the protocol should be open source so that it can be easily connected with other solutions also making it easier to collaborate with other organizations. After careful considerations, the company decided to go for a mix of MQTT and HTTP. DB decided to use several open-source MQTT libraries of Eclipse Paho for MQTT client support. The broker which is deployed is IBM MessageSight. The long-distance trains of DB use a mix of Eclipse Paho, MQTT, and a Java client for real-time communication. The information that is being sent consists of location, diagnostic check, and delay. This data after getting received by IBM MessageSight in the DB's control center is immediately pushed back to the monitors situated inside the train. The location data is sent every 10 seconds through an Eclipse Paho Client. This data is also used for business cases such as predictive maintenance, forecasts, etc. Operations of escalators and elevators at the platform are also monitored by sensors, in which around 3000 devices send a total of 10 MQTT messages per second to the central broker. The information that is sent includes the running status of the escalator, door opening/closing, and power consumption. A new project known as "Variable Vehicle IT" was deployed in ICE trains using Eclipse Mosquitto- a lightweight and popular MQTT broker to separate the tight coupling of non-safety system-related features (like some entertainment, customer features) with the safety-related ones (interlocking, train control, etc.) so that the process of changing or upgrading non-safety features is fast and not resource-intensive. To enable this a compute IaaS is created on the trains for sending messages within the train as well as the control center.

5. Comparison with Other Protocols

In present times where the technology is evolving at a fast pace, it is important to understand where MQTT stands amongst other protocols being used in IoT and being used for similar use cases as MQTT. This section will look into some of the works published towards this goal.

5.1. Relative Analysis of Protocols

Mr. Naik [9] analyzed relatively MQTT, CoAP, AMQP, and HTTP protocols then jotted down their strengths and limitations. This comparison used the empirical evidence present in the existing literature and static components.

- In terms of message size and overhead, the HTTP has the highest values because it was originally made for the web. Although MQTT is lightweight and has an individual requirement of 2-byte header size per message but HTTP, AMQP, and MQTT run on TCP which is why they also additionally have the connection overheads associated with TCP which overall increases their size. As CoAP works on UDP so it does not have connection overheads making it the least in terms of size.
- Similar trend can be observed in the power consumption and resource requirement. MQTT consumes slightly more power than CoAP and takes the second-lowest place as pointed out by various studies mentioned in [9].
- When it comes to bandwidth and latency, the results are similar to the first two. TCP plays a major role in it as it does not fully take advantage of the available network bandwidth to improve latency at the start of the connection, it starts slowly and gradually gets better in each round trip. That is why MQTT is again beaten by CoAP in these factors.
- When it comes to Reliability vs Interoperability because of the presence of three QoS levels MQTT has the highest reliability and lowest interoperability because it only supports publish/subscribe model of communication. CoAP does not have explicit features to ensure reliability but it uses confirmable and non-confirmable messages for the same and in terms of

interoperability, it is limited to devices supporting UDP. AMQP also has two QoS levels and supports JSON to serialize data, that is why it is somewhere in the middle. While HTTP has the lowest reliability because it relies on default features of HTTP and does not have its own while having the highest interoperability because the devices only need an HTTP stack for message exchanges.

- When it comes to Security vs Provisioning, MQTT ranks the lowest. It depends on TLS/SSL features for security, apart from that it has minimal authentication which uses username and password, also it does not offer any extra services providing additional functionalities. While other protocols have multiple and advanced security mechanisms whilst also provisioning several extra features. For example, IPsec and DTLS for authentication in CoAP along with observer support, resource discovery, etc.
- Usage vs Standardization shows that MQTT is the highest used protocol for IoT devices but it is still not a global standard, while HTTP is the web standard globally but not used by IoT devices.

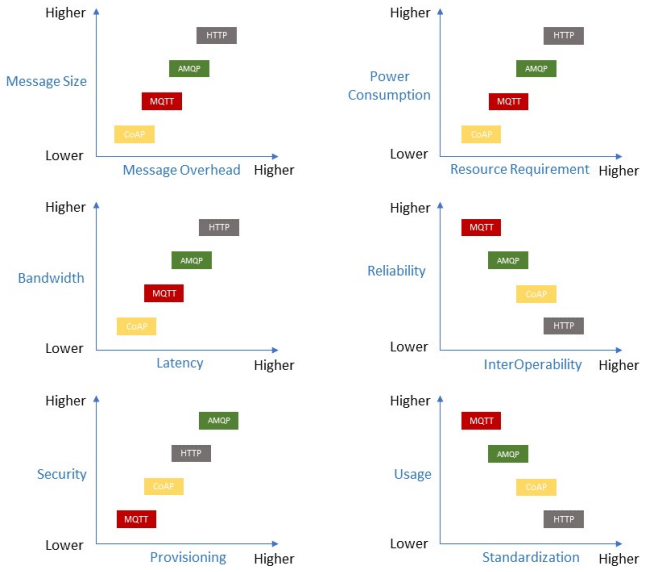

Figure 4. MQTT vs CoAP vs AMQP vs HTTP

5.2. Comparison of Protocols over ethernet

OPC UA, DDS, ROS, and MQTT were compared against each other in an ethernet-based testing environment in the research published by Profanter, Tekat, Dorafeev, Rickert, and Knoll [10]. During the experiment, while sending out payloads of different sizes it was found out that the MQTT sends out the smallest amount of data during the initialization of connection. Also, it has the smallest overhead while sending out the messages. To evaluate performances in various scenarios third-party tools were used to generate high network traffic and CPU usage. It was concluded that OPC UA and DDS are the fastest and deliver high performance. Also, it was noted that protocols operating on TCP like MQTT are not that much affected by high network traffic. It was also found out that as the CPU usage increase on the server-side MQTT messages required more time to be processed and were affected the most when the CPU load was increased.

6. Conclusion

As seen in the previous sections MQTT is a protocol that is widely used by IoT devices. It is lightweight, power-efficient, reliable, and decouples the clients which are communicating with each other making communication possible of various devices belonging to different domains. The provision of a central broker also makes it easier to manage clients. MQTT also developed its trust amongst industry experts that says a lot about its capabilities. In the future, it may come up with some extra services and advanced security mechanisms which are right now its only major limitations.

References

[1] Steve, "Beginners Guide To The MQTT Protocol," 2016, Accessed on: May 20, 2021. [Online]. Available: http://www.steves-internet-guide.com/mqtt/

[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, pp. 1–5. [Online]. Available: https://doi.org/10.1109/scvt.2013.6735994

[3] V. LampKin, W.T. Leong, L. Olivera, S Rawat, N. Subrahmanyam and R. Xiang, *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*, first ed. U.S.A: ibm.com/redbooks, Sep. 2012, pp. 12–28, Accessed on: May 20, 2021. [Online]. Available: https://www.redbooks.ibm.com/redbooks/pdfs/sg248054.pdf

[4] K. Grgic, I. Speh and I. Hedi, "A web-based iot solution for monitoring data using mqtt protocol," in *2016 International Conference on Smart Systems and Technologies (SST)*. IEEE, Oct. 2016. [Online]. Available: https://doi.org/10.1109/SST.2016.7765668

[5] G.C. Hillar, *MQTT Essentials - A Lightweight IoT Protocol*, first ed. Birmingham, U.K.: Packt Publishing, Apr. 2017, pp. 8–20, Accessed on: May 20, 2021. [Online]. Available: https://www.packtpub.com/product/mqtt-essentials-a-lightweight-iot-protocol/9781787287815

[6] M. Calabretta, R. Pecori and L. Veltri, "A token-based protocol for securing mqtt communications," in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, Sep. 2018. [Online]. Available: https://doi.org/10.23919/SOFTCOM.2018.8555834

[7] Hema, "MQTT Implementation on Celikler Holding's Power Plant Monitoring," Sep. 2020, Accessed on: May 20, 2021. [Online]. Available: https://www.bevywise.com/blog/iot-success-stories-mqtt-broker-celikler-holding/

[8] Eclipse IoT, "Deploying IoT on Germany's DB Railway System," May 2017, Accessed on: May 20, 2021. [Online]. Available: https://iot.eclipse.org/community/resources/case-studies/iot-on-railway-systems-db/

[9] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, Oct. 2017. [Online]. Available: https://doi.org/10.1109/SysEng.2017.8088251

[10] S. Profanter, A Tekat, K. Dorafeev, M. Rickert and A. Knoll, "Opc ua versus ros, dds, and mqtt: Performance evaluation of industry 4.0 protocols," in *2019 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2019. [Online]. Available: https://doi.org/10.1109/ICIT.2019.8755050