DOI: 10.2478/sbeef-2023-0008

PERFORMANCE ANALYSIS OF MQTT OVER WEBSOCKET FOR IOT APPLICATIONS

[1]B.A. ENACHE, [1]C.K. BANICA, [2]ANA GEROGE BOGDAN
[1]Department of Measurements, Apparatus and Static Converters, Electrical Engineering Faculty, University POLITEHNICA of Bucharest
[2]Business Consulting House SRL, Bucharest
E-mail: bogdan.enache2207@upb.ro

Abstract. ***With the growing popularity of the Internet of Things (IoT), the development of applications harnessing this technology is becoming increasingly widespread and accessible. Moreover, high-performance, and real-time applications necessitate the use of a low-latency protocol. Given these factors, this study aims to analyse Message Queue Telemetry Transport (MQTT) protocol used in combination with WebSocket for IoT applications. The comparison employs experimental tests used to determine data transfer capabilities, latency, and other performance metrics.***

Keywords: IoT, MQTT, WebSocket, application protocol, metrics.

1. INTRODUCTION

The proliferation of IoT technologies across a multitude of domains signifies the ubiquity of this remarkable digital revolution. It integrates a vast array of intelligent devices, each equipped with embedded capabilities that enable seamless communication not only amongst each other but also with more sophisticated applications through the conduit of the internet [1-2]. The seamless operation of the IoT ecosystem is attributable to a comprehensive suite of protocols organised on several layers as follows:

- Application: WebSocket, CoAP, MQTT, XMPP;
- Transport: TCP, UDP;
- Network: IPv4, IPv6, LoWPAN;
- Data-link: IEEE 802.3, IEEE 802.11, IEEE 802.15, IEEE 802.16.

Within the intricate and diverse landscape of the IoT, it is impossible for a single protocol to fulfil all conceivable requirements. Consequently, an extensive array of protocols has been developed to cater to the varying prerequisites of different end-user applications. The Advanced Message Queuing Protocol (AMQP) proves advantageous for applications necessitating rapid message transactions. When applications must operate within constrained resource environments, protocols such as MQTT or CoAP become the appropriate choices. Furthermore, XMPP is suitable for applications demanding immediate messaging. Therefore, the protocol selection is contingent on the specific business requirements of the target system [3].

This paper is focused only on the Application Layer, which is the component interfacing directly with the end user. From this layer, the WebSocket and MQTT are some of the well-known protocols and have been in a real competition for IoT-based applications ever since MQTT was developed. The WebSocket Protocol was developed as a response to the limitations of HTTP, which inherently struggles to facilitate bidirectional communication between web applications and servers without employing cumbersome workarounds. This protocol, detailed in RFC 6455 [4-5], successfully enables two-way communication in web applications. On the other hand, the MQTT protocol exhibits a set of features explicitly designed to address the unique demands of IoT solutions. The two protocols were used separately based on the devices, applications, and environments used for most of their existence [6]. Due to security concerns, many IT infrastructures have been dedicated to safeguarding HTTP connections while blocking unwarranted TCP traffic in the last few years. Consequently, many organisations display reluctance in opening the standard ports for MQTT (1883 and 8883), which are essential for allowing MQTT applications to interface with intra-organizational infrastructure [3], [7–10]. In this context, the WebSocket protocol paves the way for MQTT communications to leverage pre-existing HTTP facilities, such as TCP port 80, firewalls, and proxies. The paper's main contributions are to analyse the performance of newly developed MQTT client libraries that support WebSockets and do not run in a web browser. Sections 2 and 3 present the standard MQTT and WebSocket protocols in detail, each with its advantages and disadvantages. Section 4 presents the performance of libraries that use WebSockets to tunnel MQTT are presented and compared to a classic MQTT over TCP. In the last part of the paper, discussions are presented, and a Conclusion is drawn.

2. MQTT PROTOCOL

MQTT was developed with the intent to replace HTTP with a more lightweight protocol. Conceived by IBM and subsequently standardised by OASIS in 2013 is designed to minimise bandwidth requirements [7].

In MQTT communication, there are two parties - clients acting as publishers or subscribers and a server or broker acting as a middleman – Figure 1. Publishers send messages to the broker, who then determines the recipients of these messages. Subscribers, or clients, register with a broker/server for specific topics. Upon receiving messages related to these topics, the server distributes them to the subscribers who have registered

for the topics. Thus, clients receive updates on their registered topics from the broker. To function as a broker, a device must have the MQTT broker library installed, and to act as a client, it should have the MQTT client library installed.

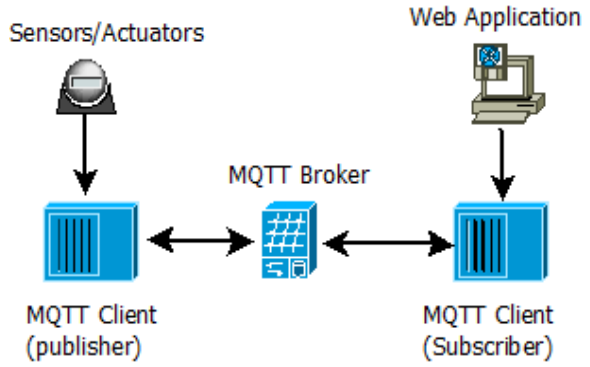

Figure 1. MQTT standard implementation.

Given its publish-subscribe communication style, MQTT is particularly suitable for environments with low bandwidth and resource constraints. As it operates over TCP, MQTT tends to be more reliable than protocols running on UDP. It offers three alternatives to accomplish messaging Quality of Service (QoS) [2]:

- At Most One Delivery - In this case, messages are delivered based on the network's best effort; an acknowledgement is not required, marking this the least level of QoS.
- At Least One Delivery - With this level, each message is sent at least once, potentially resulting in the existence of duplicate messages, and an acknowledgement message is required.
- Exactly One Delivery - This highest level of QoS necessitates an additional protocol to ensure that each message is delivered once and only once.

Unlike other protocols, MQTT's small message header makes it a preferred choice for IoT applications, requiring less power for operation. It utilises a many-to-many communication protocol, transmitting messages to multiple clients who subscribe to the broker.

The minimum size of the MQTT message header is 2 bytes. The message format of MQTT is represented in Table 1.

Table 1. MQTT message format

Message type	DUP	QoS level (0-2)	RET
Remaining length (1-4 bytes)			
Variable length header (optional)			
Variable length payload (optional)			

It includes a Message type (4 bits) and three flags: DUP, QoS Levels, and RET. The first layer is 1 byte, while the second comprises the message length, ranging from 0-127. If the most significant bit of the message type is set, it signifies that the header length can vary between 2 to 5 bytes. The DUP flag indicates if the message is duplicated, provided the QoS is level 1 or 2. The 2-bit QoS level specifies whether the connection uses QoS levels 0, 1, or 2. RET indicates retain; if the flag is set (i.e., the flag is 1), the last received message is retained, and when a new client subscribes, the message is sent to the new subscriber.

Beyond ensuring reliable packet delivery, MQTT offers a myriad of features, such as: support for multicast communication (one-to-many messaging), and the ability to establish communication channels between remote devices. However, the protocol's most significant attribute is its capacity to minimize network traffic by reducing transport overhead and protocol exchanges. It also includes a notification mechanism to alert if any abnormal situations occur.

3. WEBSOCKET PROTOCOL

WebSocket is a web-based protocol that operates on a single TCP channel and enables full-duplex communications. Unlike the MQTT protocol, a WebSocket session initiates without resorting to publish/subscribe or request/response approaches instead, it relies on constructing a handshake from the client to the server to commence communication. Once a WebSocket session is established, it fosters an asynchronous full-duplex connection between the client and the server. This session persists until both client and server decide to terminate it, signifying the cessation of their communication needs.

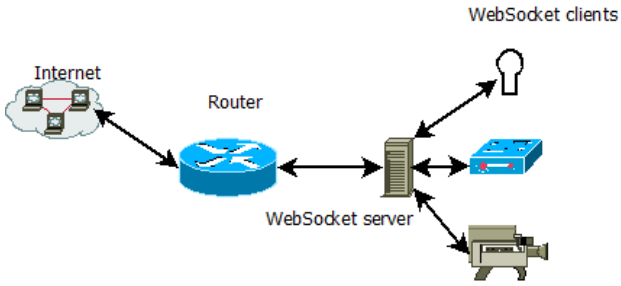

Figure 2. WebSocket standard implementation.

The WebSocket protocol structures its data into frames, with each frame beginning with a 2-byte header, followed by the payload. This header carries information pertaining to the header and payload, as depicted in Table 2.

Table 2. WebSocket packet format

F	000	Op	M	Len
1 bit	3 bits	4 bits	1 bit	7 bits

The 'F' set to 1 signifies a complete, self-contained packet, with the final packet always featuring 'F' as 1. The 'Op' field specifies the data format, which could be binary, or UTF-8 encoded. The initial packet has the 'Op' field set to 1, while subsequent packets will have 'Op' set to 0. 'M' denotes whether the data is masked using a 32-bit OR operation. 'Len' indicates the length of the payload, which can be encoded in three different ways.

WebSocket provides a level of security akin to the security model employed by web browsers. Since it operates over TCP, this protocol is beneficial for applications reliant on browsers that require interaction and communication with remote hosts. WebSocket uses the same security model as HTTP. When utilised securely (known as WebSocket Secure or WSS), it employs the Transport Layer Security (TLS) protocol for encryption, similar to HTTPS [11]. Consequently, WebSocket benefits from the extensive security features and ongoing developments in HTTP and web browser security.

4. MQTT OVER WEBSOCKET FOR IOT APPLICATIONS

The development of libraries to facilitate MQTT communication over WebSocket was driven primarily by the need for integration with web technologies. Initially, these libraries were designed to run in web browsers - Figure 3, and were primarily written in JavaScript, i.e. Eclipse Paho JavaScript Client, Mosquitto JavaScript Library etc.

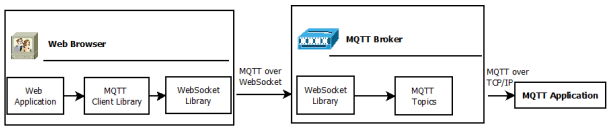

Figure 3. Web browser based MQTT over WebSocket.

While MQTT client libraries that run over WebSockets offer many advantages, they often depend on browser capabilities and behavior, which can vary among different web browsers and their versions. This can lead to inconsistent behaviors or compatibility issues. Also, WebSockets use the same security model as HTTP, and browser-based libraries may have implications for the security of the IoT application. Browsers can be vulnerable to cross-site scripting (XSS) attacks, which could compromise MQTT communication [1].

To overcome these problems, a new generation of libraries based on various languages, such as Python, Java, and C++, was produced. They provide MQTT traffic to pass through firewalls without requiring to change the rules and offer encrypted communication, ensuring the data being transferred between the client and the broker is secure [9]. The most common ones are:

- Eclipse Paho Python Client
- Eclipse Paho C Client
- Eclipse Paho Java Client
- MQTT.js
- HiveMQ MQTT Client

Only the Eclipse Paho C library will be analysed in this paper, but the findings can be easily extended to all the others [3], [6], [11–13].

The proposed methodology compares the library metrics to a standard MQTT over TCP/IP. There will be four main scenarios regarding payload size, the QoS, the use of TLS and latency. The contained results are presented in Tables 3 – 5.

Table 3. Payload and QoS analysis [8]

No. of messages	Payload (bytes)	QoS	MQTT TCP/IP (msgs/s)	MQTT WebSock (msgs/s)
10 000	256	0	6631	5708
10 000	256	1	1591	1306
10 000	256	2	1344	1095
10 000	100 000	0	1119	705
10 000	100 000	1	1060	676
10 000	100 000	0	1042	660

Table 4. Latency analysis [13]

Payload (bytes)	MQTT TCP/IP Latency (ms)	MQTT WebSock Latency (ms)
5	283.35	284.70
10	280.02	279.78
50	286.62	289.57
100	279.05	282.01
500	270.93	270.77
1000	267.43	270.73

Table 5. Security analysis [8]

	MQTT TCP (bytes)	MQTT WeSock (bytes)
Establish connection	512	1161
Subscription request	291	299
Disconnect	334	418
256b QoS1 publishe	545	555

5. DISCUSSIONS

In concordance with the proposed methodology, all the measurements were performed by [8] and [13] and involved transmitting messages from the client to the broker, waiting for an acknowledgement (ACK) of receipt for each message before proceeding to send the next one.

Table 3 outlines the performance impact of using WebSocket connections instead of plain TCP connections for MQTT communication. It's noted that as the payload size increases, the relative difference between TCP and WebSocket connections also grows. This growth results from the additional computational overhead associated with WebSocket connections. Specifically, WebSocket necessitates masking payloads at the sender's end and unmasking them at the receiver's end. This requirement results in extra processing overhead that increases with payload size.

The latency of the MQTT over WebSocket – Table 4, is, in most cases, higher than that of MQTT over TCP/IP, mostly due to the extra processing steps, such as masking and unmasking of payloads. Although this increase is not significantly large, varying in the range of 0.05 – 1.24%.

When comparing the total data used in MQTT over TCP and MQTT over WebSockets without TLS – Table 5, it becomes clear that WebSockets consistently uses more data, which is largely due to the protocol's intrinsic overhead. This extra data use can lead to increased

latency and higher bandwidth requirements, which should be considered when designing IoT applications.

While MQTT over WebSockets offers the benefits of easy firewall traversal and compatibility with web technologies, it also comes with increased data usage, which may impact overall performance, particularly in terms of latency and bandwidth usage. It is crucial to consider these factors when selecting the most suitable protocol for specific IoT applications and environments.

6. CONCLUSIONS

The choice between MQTT over WebSocket and MQTT over TCP/IP for IoT applications depends on each application's specific requirements and constraints. MQTT over TCP/IP is lean and efficient, ideal for resource-limited environments and networks with open TCP ports. On the other hand, although MQTT over WebSocket introduces additional overhead, it offers compatibility with web technologies and better firewall traversal, making it attractive for certain IoT scenarios, particularly those involving interaction with web applications or strict firewall rules. The selection between these protocols should be made after considering factors like network environment, resources, payload size, security, and web compatibility and by conducting realistic benchmark tests mainly due to the very close scores obtained by both protocols for latency and data used.

7. ACKNOWLEDGEMENT

The work presented in this paper is part of the project Innovative Smart Digital Platform (ISDP) SMIS code 2014+ 142654.

8. REFERENCES

[1] M. B. Yassein, M. Q. Shatnawi, and D. Al-zoubi, “Application layer protocols for the Internet of Things: A survey,” in 2016 International Conference on Engineering & MIS (ICEMIS), Agadir, Morocco: IEEE, Sep. 2016, pp. 1–4.

[2] Y. Sueda, M. Sato, and K. Hasuike, “Evaluation of Message Protocols for IoT,” in *2019 IEEE International Conference on Big Data, Cloud Computing, Data Science & Engineering (BCD)*, Honolulu, HI, USA: IEEE, May 2019, pp. 172–175.

[3] V. Sarafov, “Comparison of IoT Data Protocol Overhead,” 2018.

[4] N. Imtiaz Jaya and Md. F. Hossain, “A Prototype Air Flow Control System for Home Automation Using MQTT Over Websocket in AWS IoT Core,” in 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Zhengzhou, China: IEEE, Oct. 2018, pp. 111–1116.

[5] P. Sharma, I. Pandey, and P. M. Pradhan, “Hardware Implementation and Comparison of IoT Data Protocol for Home Automation Application,” in 2022 IEEE Delhi Section Conference (DELCON), New Delhi, India: IEEE, Feb. 2022, pp. 1–6.

[6] A. Zainudin, M. F. Syaifudin, and N. Syahroni, “Design and Implementation of Node Gateway with MQTT and CoAP Protocol for IoT Applications,” in 2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE), Yogyakarta, Indonesia: IEEE, Nov. 2019, pp. 155–159.

[7] P. Gupta and I. O. Prabha. M, “A Survey of Application Layer Protocols for Internet of Things,” in 2021 International Conference on Communication information and Computing Technology (ICCICT), Mumbai, India: IEEE, Jun. 2021, pp. 1–6.

[8] I. Craggs, “Understanding the Differences between MQTT and WebSockets for IoT,” 2022. https://www.hivemq.com/article/understanding-the-differences-between-mqtt-and-websockets-for-iot/ (accessed Jun. 08, 2023).

[9] S. D. Grigorescu, G. C. Seritan, B. A. Enache, F. C. Argatu, and F. C. Adochiei, “Open Source Architecture for Iot Based SCADA System for Smart Home,” Sci. Bull. Electr. Eng. Fac., vol. 20, no. 1, pp. 33–36, Apr. 2020.

[10] G. Seriţan, R. Porumb, C. Cepişcă, and S. Grigorescu, “Integration of Dispersed Power Generation,” Electr. Distrib. Intell. Solut. Electr. Transm. Distrib. Netw., pp. 27–61, 2016.

[11] G. M. B. Oliveira *et al.*, “Comparison Between MQTT and WebSocket Protocols for IoT Applications Using ESP8266,” in 2018 Workshop on Metrology for Industry 4.0 and IoT, Brescia: IEEE, Apr. 2018, pp. 236–241.

[12] R. Porumb, G. Seritan, “Integration of Advanced Technologies for Efficient Operation of Smart Grids,” Green Energy Adv., p. 95-104, 2019.

[13] D. R. C. Silva, G. M. B. Oliveira, I. Silva, P. Ferrari, and E. Sisinni, “Latency evaluation for MQTT and WebSocket Protocols: an Industry 4.0 perspective,” in 2018 IEEE Symposium on Computers and Communications (ISCC), Natal: IEEE, Jun. 2018, pp. 01233–01238.