Technium Vol. 28, pp.43-57 (2025)
ISSN: 2668-778X
www.techniumscience.com

Performance Analysis of MQTT Protocol in IoT Environments: Impact of Payload Size and QoS on Key Metrics

Ahmed J. Hintaw[1]
[1]Applied Medical Sciences College, University of Kerbala, Karbala, 56001, Iraq
Corresponding author: Ahmed J. Hintaw (ahmed.jameel@uokerbala.edu.iq).

Abstract

The advancement of Internet of Things (IoT) technology represents the transformative opportunity and already overcomes this limiting factor, utilizing lightweight communication protocols to minimize overhead compared to traditional internet systems. MQTT is one of the popular and efficient protocol used in IoT to exchange data. In this work, we investigate the MQTT behavior in given IoT scenarios by considering payload sizes (5 MB, 10 MB,15 MB, and 20 MB) and QoS levels as influencing factors for performance metrics such as throughput, execution time, traffic overhead, and latency. Testbed experiments on a Raspberry Pi 3 indicate that processing time scales with payload size and QoS-2 (highest reliability) introduces overhead up to an order-of-magnitude. For bigger payloads, throughput increases with QoS-0 (lowest reliability) delivering the highest throughput rates, while latency increases with both payload size and QoS level. The results provide important insight on how to efficiently improve the reliability of MQTT, as it evidences the possible trade-offs between both categories.

Keywords: Internet of Things, MQTT, Security, Performance evaluation.

1. Introduction

The interaction designs and protocols of the Internet of Things (IoT) are constantly transforming in order to meet the new challenges that have arisen as a result of IoT environments that involve a high number of various nodes that are limited in their resources [1]. Such challenges have ensured support with the heavy computing of huge amounts of information, datamining, data filtering, and classification, along with the support of wide variety concerning objects both software and hardware, in addition to various types of traffic. In this context, the Transmission Control Protocol/Internet Protocol (TCP/IP) facilitates from end to end connectivity across the IoT. Interoperability is supported by TCP/IP; however, it is constrained in the context of IoT by the necessities of massive heterogeneous networks, including the necessity to permit minimal latency (commonly, sub-second reactions) with minimal jitter. As a result, in order to satisfy the necessary application prerequisites, scenarios involving IoT depend on IP-based communicating protocols, which facilitate asynchronous communication through an intermediary or broker object. Essentially, broker-based publish-subscribe protocol stacks and designs are utilized in IoT instances in reality, establishing an abstraction over sources of data (generators) and data recipients (users)[2], [3].

Nevertheless, even though these options help with things such frequent information polling, obstacles with the consumption of resources, security, privacy, and handling mobility still exist. Besides, the fundamental connectivity concepts of TCP/IP that utilize a host-based accessibility method are the primary cause of this particular problem. Consequently, a greater understanding of the performance attributes of the many available solutions as well as the constraints of existing deployment is necessary in order to further advance communication protocols in a manner that effectively maintains the wide range of IoT systems [4]. Therefore,

the motivation behind this evaluation is the necessity of gaining a better understanding of the various data communication protocols based IoT and designs that are accessible, as well as the variations in their networking semantics and effectiveness with respect to latency and packet loss [5]. In the current setting, we propose the development of a testbed to evaluate the performance of data communication protocols, emphasizing that there may not be a universally optimal approach. This testbed aims to explore how protocol design choices influence performance under varying conditions. Understanding these design-performance relationships is the primary motivation for this work, as it seeks to provide insights into the trade-offs and considerations necessary for optimizing protocol behavior in diverse IoT scenarios.

Moreover, since the application layer determines the connection's dependability, latency, and overhead, this research attempts to provide a thorough examination of the positive and negative aspects of MQTT, the most widely used protocol for the application layer. Open-source MQTT implementations are used to investigate the protocol's modes of operation and distinctions [6]. Additionally, the recommendations within RFC7925 [7] will be adopted for protecting the data exchanges performed by connected objects, as security weaknesses in IoT communications might result in risks such message forging, manipulation, or eavesdropping. Specifically, Tschofenig et al. [7] suggested cipher packages will be evaluated in terms of increased CPU utilization and network cost in comparison to the unsecured case, in which no encryption is used. In such a limited setting, these two elements will establish the viability of the authentication and ciphering techniques, illustrating the trade-off between security and speed.

The primary objective of this research is to do a thorough investigation of the execution time, throughput, traffic overhead, and latency according to the quality of service of MQTT, evaluating their strengths and viability within device limitations. A Raspberry Pi was employed as the network nodes in this actual network circumstance, as well as a switch being used to emulate and set up the various network options.

This paper is organized as follows: Section 2 analyses the related and previous work, identifying our contributions beyond the literature. Then, Section 3 explains the data communication protocol MQTT. Section 4 draw the methodology of this work. We present an analytical study of these protocols in Section 5, and then the results of the experiment are shown, discussed and compared in Section 6. Finally, Section 7 concludes the paper and sketches our future work.

2. Related work

The relevant research that evaluates the effectiveness of the data communication protocol such as MQTT has been examined within this section. The community of scientific has been more interested in assessing the effectiveness of protocols utilized within IoT over a range of criteria and situations, as a result of the recent explosive growth in IoT node implementation. MQTT has become one of the most frequently utilized protocols for IoT data transmission. In the case of in [8], the authors evaluated MQTT and CoAP in a lossless, congestion-free network environment. According to their investigation, CoAP performed better than MQTT, especially in the context of latency.

A middleware that supports both MQTT and CoAP has been developed and put into use by the authors in [9]. They found that MQTT had a shorter delay than CoAP when the packet loss rate was lower, while a larger delay when the packet loss rate was higher. In a comparable manner, the authors of [10] assessed the NodeMCU ESP8266's MQTT and CoAP publishing capabilities. When it came to transmitting time and again they found that MQTT was more accurate whereas CoAP was faster. Nevertheless, MQTT as well as CoAP were compared in limited communication circumstances reported in the research presented in [11]. MQTT proved more energy-efficient, according to the data, especially when it came to battery utilization. Furthermore, in primitive point-to-point structures, the latency across the two protocols was consistent. Besides, the efficiency of MQTT along with CoAP on a Raspberry Pi 3 environment was evaluated by the work of [12]. According to the research

investigation, MQTT functioned better than CoAP in situations involving smart home, underscoring its applicability for particular IoT application scenarios.

Further study has broadened its focus to encompass a variety of IoT-related protocols. Mun, Dinh, and Kwon [13] evaluated CoAP, MQTT, MQTT-SN, WebSocket, and TCP by comparing parameters including memory as well as CPU consumption, power consumption, and the duration of transmission with a 50 kB payload. on their analysis of WebSocket along with MQTT with an ESP8266 prototype, the authors in [30] emphasized on memory needs over minuscule activities and message transmission delay. Moreover, Borsatti et al. [14] used latency as a major parameter to analyze MQTT at various QoS levels. They discovered comparable latency for QoS 0 and 1, but for QoS 2, they saw noticeably increased latency, up to seven times higher.

Furthermore, MQTT along with other IoT protocols, have been investigated with studies focusing on determining how security affects functionality. According to the Baranauskas [15], a Raspberry Pi 2 was employed as a broker to measure the usage of energy at various levels of QoS in order to assess the effectiveness of MQTT over TLS. The authors of [16] investigated enhancing the MQTT with different block ciphers and assessing the results. Improved lightweight protections are required. Besides, Oliveira [17] examined packets over the MQTT broker in the context of ciphering or not in order to determine the delay while utilizing the FIWARE infrastructure as a broker utilizing MQTT clients.

Two significant characteristics set this study apart from earlier research. Firstly, we consider a different payload size (ranging from 5 to 20 megabytes) for messages published over MQTT, as opposed to the 16,384 bytes that are the limit of the majority of prior research. Secondly, contrary many other research efforts, which emphasize effectiveness from the publisher to the broker, our observed transmission times examine efficiency from the publisher to the subscriber. Furthermore, our work assesses the protocol under various QoS levels while taking into account a number of performance indicators, such as execution time, traffic overhead, throughput, and latency.

3. MQTT protocol

The MQTT protocol is a lightweight Internet of Things (IoT) protocol and popular data communication protocol based on the publish–subscribe paradigm[1], [18]. It is an open standard messaging protocol that has been around for more than 20 years (OASIS Standard). It has gained relevant popularity for IoT, due to its ease of implementation, small code footprint, bandwidth efficiency, and client decoupling. However, MQTT acts as a publish/subscribe mechanism, a data communication protocol that relies on the TCP/IP [19]. significantly, conceived for domains with limited machine resources. Besides, MQTT is a perfect fit for these types of challenging circumstances. Moreover, there are three separate objects: the publisher, subscriber, and broker. While the MQTT Publisher and Subscriber act as clients, the MQTT Broker acts as the server. When a subscriber needs data on a distinct topic, it dispatches a subscription request to the MQTT Broker. Conversely, a publisher dispatches transmissions regarding its supported topics to the MQTT Broker. Upon obtaining a transmission on a certain topic from a publisher, the MQTT Broker then dispatches that message to the subscriber node who has subscribed to that specific topic. The design of the MQTT has been depicted in Figure 1.

Technium Vol. 28, pp.43-57 (2025)
ISSN: 2668-778X
www.techniumscience.com

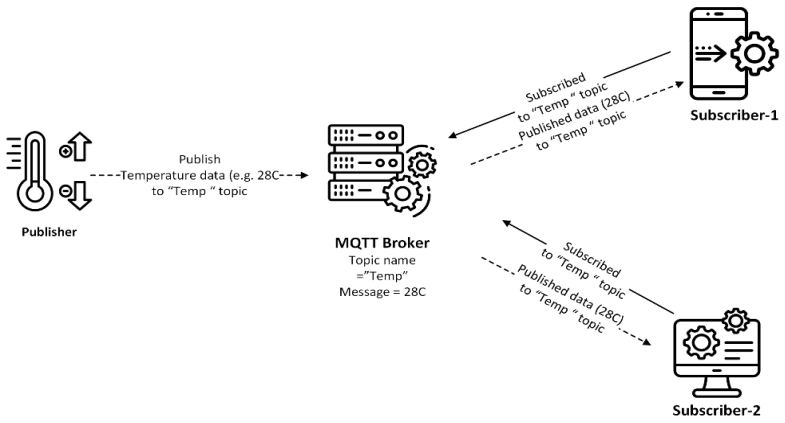

Figure 1: MQTT protocol system structure.

Likewise, three different Quality of Service (QoS) levels have been defined by the MQTT manners: QoS0, QoS1, and QoS2. QoS0 offers a minimal delivery assurance, ensuring that a message is delivered no more than once. Minimum of one delivery of the message is guaranteed by QoS1. The highest level of service, QoS2, assures that the data is transmitted exactly once every time. Different trade-offs between overhead associated with communication and delivery assurance are reflected in these QoS levels. Crucially, the approach we present works with all MQTT protocol QoS levels, offering it a flexible way to meet various communication requirements [20]. The QoS that the MQTT protocol supports is shown in Figure 2. Additionally, the MQTT protocol provides more message kinds than the majority of communication protocols, including the CONNECT message for connection, the SUBSCRIBE message for topic subscription, the PUBLISH message for topic message publication, and so on [2]. The most crucial information in the MQTT protocol, the topic of the message content, is kept in the payload field of the PUBLISH message.

Besides, three components make up the message structure of the MQTT: payload, variable header, and fixed header. Every message type requires a fixed header. The message type is implied by the first byte in the fixed header, and the length after the fixed header is implied by the next several bytes. Particular message types have a variable header that may be utilized to establish connection parameters such as connection flag, hold connection, etc. Essential information, including the topic message in a published message, is carried by the payload, which is also included in some message types.

Regarding to the context of the security, the publish message's field payload can't exceed 256 MB long. In the context of security, MQTT does come with some built-in security features, such as ciphering and entity authentication, but these are not activated by default and require further setup [21]. TLS is one of the several authentication approaches presented by the protocol. These protection services require extra resources in order to successfully protect communication among IoT nodes. The survey demonstrates that message ciphering and authentication pose the most security risks to nodes utilizing MQTT communication [22].Besides, the MQTT broker does not verify the publisher and subscriber identities, which might result in illegal and malevolent access to the endpoints. The broker's efficiency can be significantly lowered by these vulnerabilities, which can also lead to issues including manipulated messages and overloading. Further, by default, the encryption of the message is not supported by the protocol, which leads to eavesdropping and message manipulation by unauthorized parties.

Technium Vol. 28, pp.43-57 (2025)
ISSN: 2668-778X
www.techniumscience.com

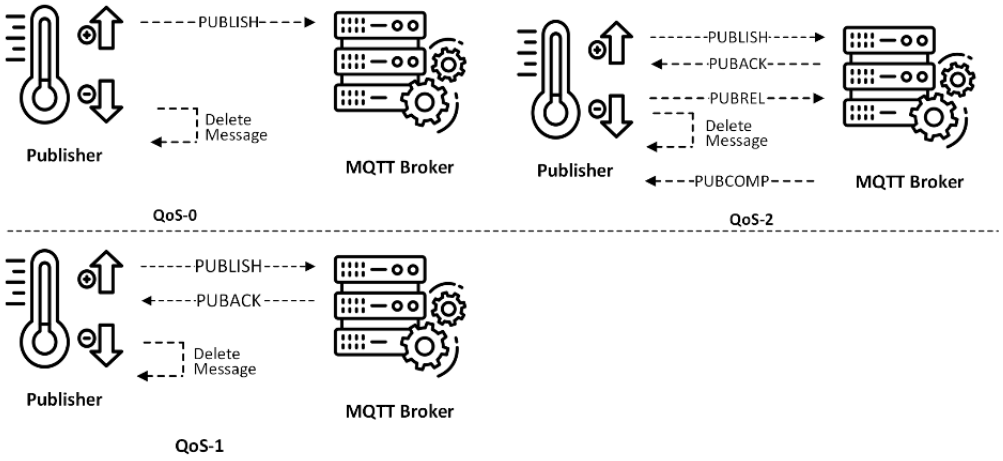

Figure 2: The reliability levels supported by the MQTT protocol.

4. Methodology

This section details the phases involved in constructing and putting into practice the scenario utilized to evaluate the MQTT protocol. The method entails establishing the measuring tools, setting up and running the equipment, implementing the network scenario, and investigating the abilities of the chosen, well-known protocol implementations (such as Mosquitto for MQTT).

4.1 Proposed Scenario

It is crucial to model realistic network settings in order to guarantee that the network protocol assessment accurately represents real-world circumstances. In order to carry out the research, an isolated network configuration was used. Figure 3 depicts the IoT protocol analysis's deployed network scenario. Table 1 lists all of the equipment in the case study along with their functions. Since MQTT required three separate devices for an individual message exchange: *Publisher*, *Subscriber*, and *Broker*. The broker unit is very imperative. Since a Raspberry Pi 3 is frequently utilized in Internet of Things, it was chosen as the MQTT clients as well as broker. To make the experiment easier, a jacket with buttons, sensors, and screens is attached to the Raspberry Pi.

Table 1 lists all the components of the network environment along with their descriptions and functions.

Element	Device	Operation system	Purpose
Switch	Lynksys EZXS88W	N/A	To isolate the whole network scenario.
Publisher	Dell Latitude E7240	Windows	To generate data and publish it to the MQTT broker
Subscriber	Raspberry Pi 3	LINUX	To subscribe to particular topic on the broker and get the subscribed information.
NetEM	Raspberry Pi 3	LINUX	To emulate the desired network conditions with NetEm.
Broker	Raspberry Pi 3	LINUX	To act and run MQTT broker as intermediate between publisher and subscriber.

Moreover, in order to replicate a network landscape similar to real-world IoT deployments, every element of the scenario had been connected through an Ethernet link switch. For simulating different network scenarios, we used the Network Emulator (NetEm), a tool that allows you to configure conditions including packet corruption, duplication, reordering, delay, and loss at the output of a network interface. NetEm works within the Linux kernel and adds losses before packets are delivered across the interface, hence a simulated machine was required for network emulation. If NetEm was deployed directly to the client or server, packets would be discarded

before capture, making them untraceable. This issue was handled by using a simulated machine as a network emulator and routing all traffic via it. Packets might thus be collected at either the client or the server, allowing for precise analysis of both sent and lost packets. In light of such considerations, the conceptual network scenario makes sure that all communication between the broker, client, and server is routed via the network emulator and guarantees isolation. Figure 10 illustrates this configuration. This was accomplished by applying network settings at the client, which included certain routes for the IP addresses of the server and broker through the IP address of the network emulator.

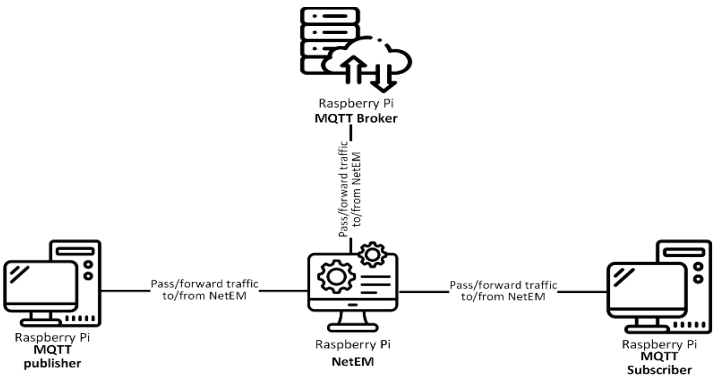

Figure 3: The conceptual network topology used to examine the MQTT protocol.

On the server and broker, comparable setups were introduced. Additionally, in order to ensure correct flow across the emulation layer, the network emulator was set up to accept traffic from the client and broker and to route any outbound traffic intended for them.

4.2 Evaluation Testbed

This study's main goal is to develop and put into practice a technique that uses an actual platform to assess MQTT's performance over TCP in Internet of Things settings. The investigation emphasizes on six performance indicators that are essential to IoT services: throughput, execution time, CPU utilization, bandwidth consumption, execution time, and traffic overhead.

However, The Eclipse Java platform served as the experiment's basis for putting this work into practice. The MQTT Publisher, Subscriber, Broker, and network emulator are the four primary parts of the prototype setup. Version 3.1.1 of the MQTT protocol was used [23]. The Eclipse Paho Java Client version 1.1.0 [24] was used to construct the MQTT Publisher and Subscriber nodes, while the open-source Mosquitto release 2.0.14 [25] was used to implement the MQTT Broker. The Broker was running on another machine, while the Publisher and Subscriber nodes were set up on other machines. Figure 4. depicts the layout of the testbed environment utilized within the experiment.

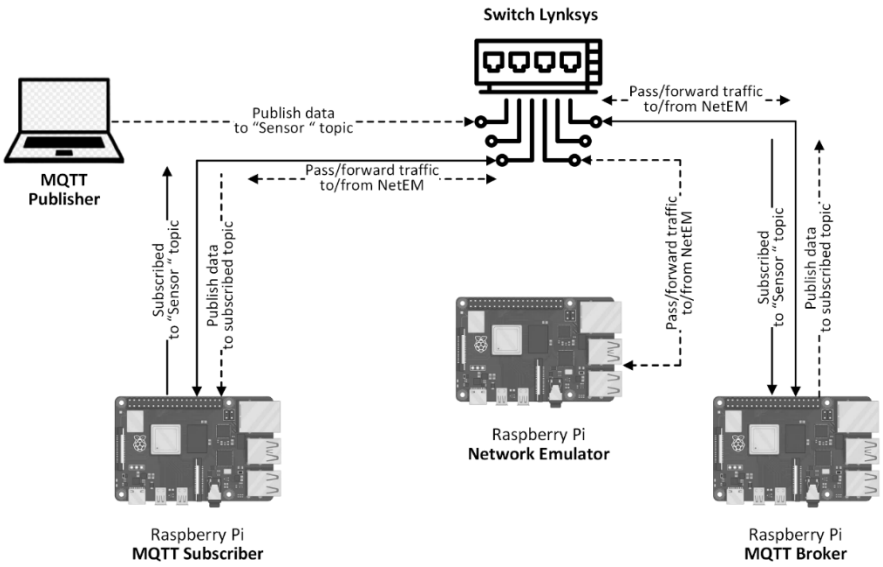

Figure 4: The layout of the testbed environment utilized within the experiment.

The testbed displays a high-level structure that highlights the essential elements as well as how they work together. Further, it mimics a typical MQTT topology, in which nodes for publishers and subscribers communicate simultaneously. Besides, the MQTT elements such as Publisher, Broker, Subscriber, as well as network emulator are the four main parts of the MQTT testbed infrastructure. Because it enables the modeling of several network situations, the network emulator is essential to this architecture. Furthermore, the broker constitutes an extremely important component in the assessment process since the MQTT protocol functions on a centralized basis, with each node largely relying on the broker.

5. Experimental results and analysis

This section exhibits outcomes from analyzing the experiment measures, with an emphasis on execution time, Traffic overhead, Throughput, and latency of the communication. Using the assessment testbed previously developed, the MQTT protocol's effectiveness has been assessed considering four significant indicators such as:

- The time taken by the publisher to create and send the message to the subscriber as an indicator of the execution time.
- The extra bytes and processing requirements added by the protocol for delivering messages as an indicator of the Traffic overhead.
- The rate at which messages can be sent and received over a network within a given time frame as an indicator of the Throughput
- The time interval between the publisher and the subscriber can be considered an indication of interaction latency.

6.1. Execution Time

In this section the execution times needed for payload sizes (5 MB, 10MB, 15 MB and 20MB) are tested on three different QoS levels (QoS-0, QoS-1 and QoS-2). Figure 5 presents the measured processing times in milliseconds, highlighting the impact of payload size and QoS level on processing efficiency.

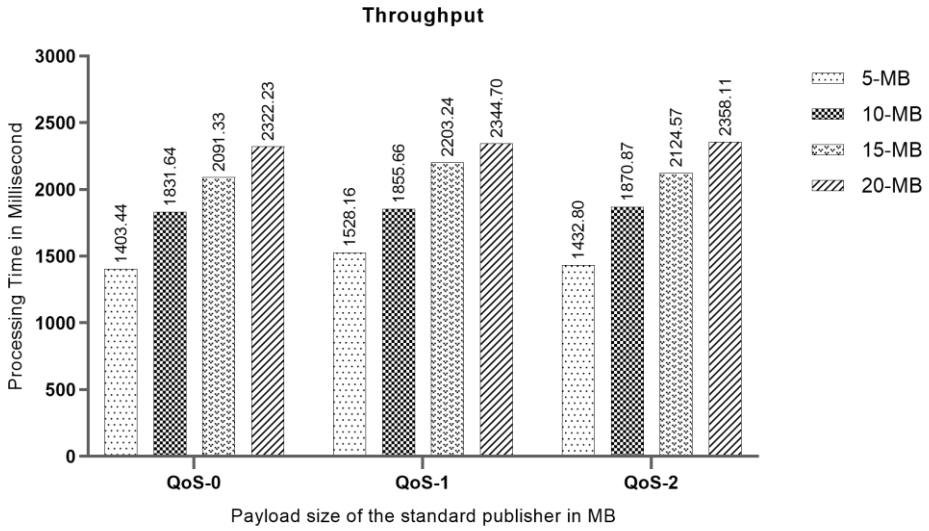

Figure 5: The effect of payload size and QoS level on processing efficiency.

The outcome shows that there is a direct proportional relationship between processing time and payload size across all the QoS [6] levels. The processing time increases linearly to the payload size from 5 MB to 20 MB. As an example, because of QoS-0 the processing time got rising from 1403.44 ms for 5MB to 2322.23 ms for 20MB We've also seen this sort of pattern in QoS-1 and QoS-2 where bigger payloads invariably result in longer processing times. Thus, there is a growing demand on system resources as the payload size increases, independent of the chosen QoS settings.

Each QoS level exhibits distinct processing characteristics. QoS-0 demonstrates the shortest processing times across all payload sizes, as it provides minimal delivery assurance. For example, at 5 MB, QoS-0 processing time is 1403.44 ms, which rises to 2322.23 ms for a 20 MB payload. The relatively low processing time of QoS-0 implies minimal overhead, making it suitable for scenarios where rapid transmission is prioritized over delivery reliability. QoS-1 introduces moderate overhead, resulting in processing times slightly higher than those observed with QoS-0. For instance, a 5 MB payload in QoS-1 needs 1528.16 ms, increasing to 2344.70 ms at 20 MB. This incremental rise in execution time can be attributed to QoS-1's message acknowledgment mechanisms, which improve delivery reliability at the cost of some processing efficiency. QoS-2 consistently records the most heightened execution times for all payload sizes, with a peak of 2358.11 ms for a 20 MB payload. This can be attributed to QoS-2's highest reliability guarantee, which guarantees exactly one message delivery through vast acknowledgment protocols. Consequently, QoS-2 is best fitted for applications that mandate high delivery reliability, even if it incurs significant processing overhead.

These observations emphasize the significance of choosing a suitable QoS level based on application-specific conditions. Applications with stringent speed requirements may satisfy the reduced overhead associated with QoS-0, whereas those prioritizing data integrity and delivery reliability may select QoS-1 or QoS-2, taking the extra execution time as a trade-off for enhanced assurance. Eventually, the selection of QoS level and payload size should be aligned with the application's functional priorities, acknowledging the inherent trade-offs between performance and reliability.

6.2. Throughput

Throughput is the rate at which data is handled or transmitted evaluated as the inverse of the time involved for finishing the entire transmission. Figure 6 demonstrates the throughput (in milliseconds) with different payload sizes (5 MB, 10 MB, 15 MB, and 20 MB) for different Quality of Service (QoS) levels (QoS-0, QoS-1, and QoS-2).

However, over most QoS levels, throughput goes up as payload size grows. As an instance, within QoS 0, throughput jumps from 0.0036 ms at 5 MB to 0.0086 ms at 20 MB. This positive association indicates higher payloads result in an increased data transfer rate, which is probably because they have lower overhead for each unit of data. QoS-0 maintains a significant throughput among the different payload sizes, notably over 20 MB, when it reaches 0.0086 ms. This significant throughput implies QoS-0 has smaller overhead, permitting more responsive transmission and making it appropriate for circumstances where speed is extremely important. QoS-1 has comparatively lower throughput amounts in comparison to QoS-0 with regard to all payload sizes. In this case, at 10 MB, QoS-1 achieves a throughput of 0.0054 ms vs 0.0072 ms over QoS-0.

Therefore, a slight decrease in throughput is probably associated with QoS-1's acknowledgment mechanism, which generates more reliability at a slight efficiency cost. QoS-2 has the lowest throughput amounts of among the three QoS levels, attributed to its increased overhead from extensive acknowledgment mechanisms that guarantee exactly-once delivery. Over 5 MB, QoS-2 obtains a throughput of 0.0035 ms, in comparison to 0.0036 ms for QoS-0 as well as 0.0033 ms for QoS-1. Besides, due to the minimal overhead, QoS-0 could have been preferable for scenarios involving high throughput, whereas QoS-1 or QoS-2, regardless their lower throughput, might be better choices for applications needing high delivery dependability. Throughput can be affected by the payload size option; irrespective of QoS levels, larger payloads frequently attain more efficient transfer of information rates.

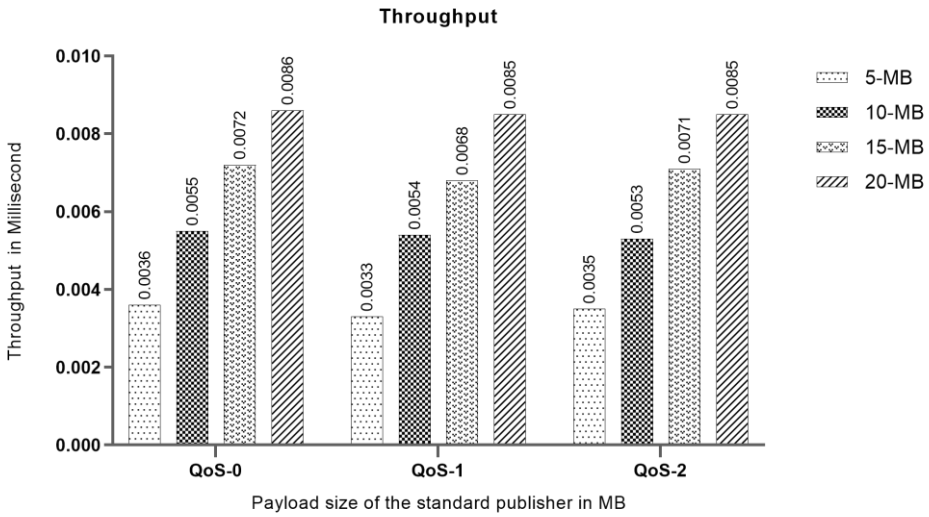

Figure 6: The impact of payload size and QoS level on processing efficiency.

6.3. Traffic Overhead

In MQTT, the extra data that needed by the protocol including message handling, control, and reliability on in addition to the content of the payload is formally referred to as the traffic overhead. MQTT headers, acknowledgement messages, keep-alive packets, and further packets utilized for different Quality of Service

(QoS) levels are all incorporated into such expenses. Optimizing network bandwidth in Internet of Things platforms where reducing data utilization is crucial necessitate a awareness of MQTT's traffic overhead. The traffic overhead % of the MQTT network's interacting messages is illustrated in Figure 7. The traffic overhead across different Quality of Service (QoS) levels (QoS-0, QoS-1, and QoS-2) can be observed in the figure for data payload sizes of 5 MB, 10 MB, 15 MB, and 20 MB.

On the other a situation QoS-0 frequently reveals a minimal traffic overhead, with payload sizes that go from 0.0147 for 5MB to 0.0037 over 20MB. Given that it is missing acknowledgment or the retransmission abilities it has a minimal overhead, hence very effective for non-critical applications where sporadic data loss is suitable. Exemplary scenarios involve telemetry or real-time data collected by sensors, in which the objective is utilizing resources that as little as possible. But QoS-1 has a moderate overhead, which ranges from 0.0206 over 5MB to 0.0052 over 20MB. Reliable message delivery is ensured by its acknowledgment scheme, thereby accounting for an increase in compared with QoS-0. Similarly, QoS-1 could be suitable for scenarios where delivery assurance is vital, such as messaging services or Internet of Things scenarios, because of its minimal increase in traffic overhead and the absence of the need for strict duplication detection. In addition, the QoS-2 has a significant traffic overhead, with 5MB reporting at 0.0374 and 20MB at 0.0094. Its extensive reliability measures, including as acknowledgment and duplicate message identification, are accountable for this substantial cost. Nevertheless, for mission-critical platforms, which include control systems or financial operations, where message integrity is crucial.

The results demonstrate clear distinctions in overhead among the QoS levels and a consistent trend of decreasing overhead as the payload size increases. Similarly, for all QoS levels, traffic overhead decreases as the payload size increases. This indicates that overhead becomes less significant in proportion to the entire data size for larger payloads. For instance, QoS-0 overhead decreases from 0.0147 for 5MB to 0.0037 for 20MB, reflecting a reduction of approximately 75%. Variations have been noticed for QoS-1 and QoS-2. In addition, the corresponding increase in overhead between QoS levels is instead consistent across payload sizes. The similarity implies that the extra processes in QoS-1 and QoS-2 scale in proportion to the data payload size.

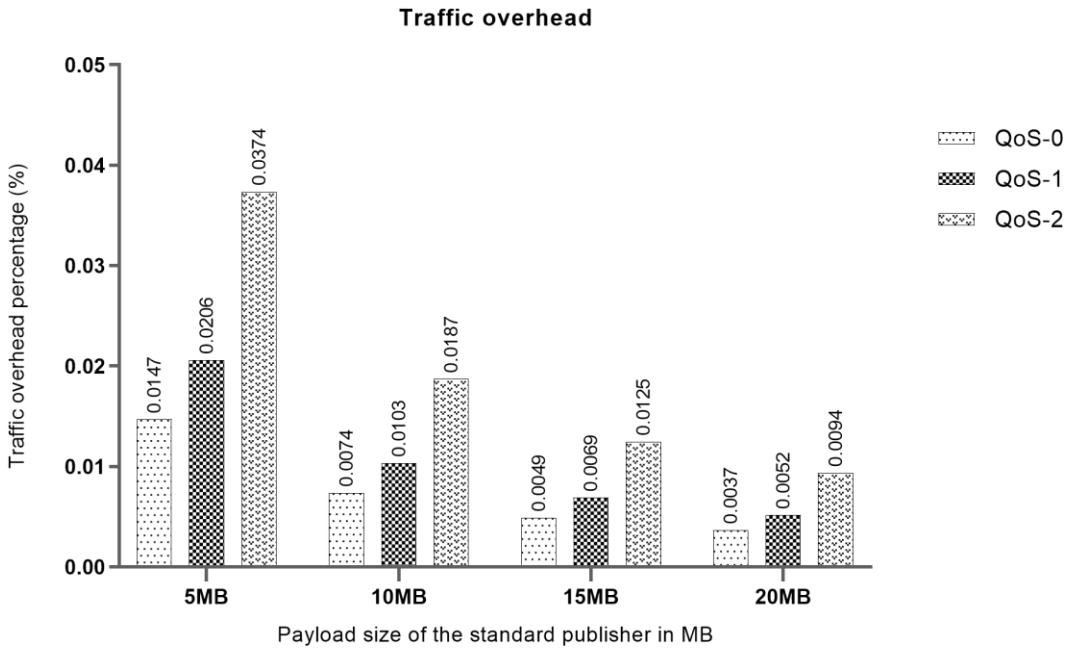

Figure 7. The traffic overhead percentage of the interacted messages in the MQTT network.

6.4. Latency

Figure 8 illustrates how latency, QoS level, and payload size relate to each other in a MQTT network. The data collected gives emphasis on how end-to-end message delay is affected by different QoS levels (QoS-0, QoS-1, and QoS-2) for payload sizes of 5 MB, 10 MB, 15 MB, and 20 MB. The investigation highlights the trade-offs between reliability and performance by exhibiting a continuous rise in delay as both payload size and QoS level increase.

However, across all payload sizes, the QoS-0 persistently exhibits the smallest latency. A 5 MB payload has a latency of 0.4066 seconds, while a 20 MB payload has a latency of 0.6555 seconds. The nonexistence of acknowledgment and resend methods results in this low latency, which cuts down on processing time. For time-sensitive scenarios, such live data streaming or real-time telemetry, where low latency is vital and sporadic data loss is tolerated, QoS-0 is appropriate. likewise the QoS-1 has a relatively small latency, which comes between 0.8918 and 7.3547 seconds for a 5MB and 20MB payload, accordingly. Given that QoS-1's acknowledgment technique enables reliable message delivery, it comes with processing delays, especially when the payload size gets bigger. Therefore, this level is sufficient for applications where latency is an insignificant consideration but delivery dependability is crucial, such event-driven IoT systems. With latency values rising from 0.9241 seconds for a 5MB payload to 7.6114 seconds for a 20MB payload, QoS-2, nevertheless has the greatest delay of any QoS level. The higher delay is a result of the extra processes involving resilient reliability checks and duplicate message identification. Mission-critical systems where message reliability and security are crucial than performance issues, such financial transactions or industrial control systems, are the greatest candidates for QoS-2.

Accordingly, for all QoS levels, latency increases as payload size grows. This trend reflects the additional time required to process larger payloads and, in the case of QoS-1 and QoS-2, the added overhead of acknowledgment and reliability mechanisms. Notably, the increase in latency is more pronounced for QoS-1 and QoS-2 compared to QoS-0 due to their additional protocol requirements. For example, Latency increases by approximately 61% from 0.4066 seconds (5MB) to 0.6555 seconds (20MB) in the context of QoS-0. Latency rises significantly by over 724% from 0.8918 seconds (5MB) to 7.3547 seconds (20MB) in case of QoS-1. Latency grows by approximately 724% from 0.9241 seconds (5MB) to 7.6114 seconds (20MB) for QoS-2. This liner increase in latency for higher QoS levels underscores the performance cost of enhanced reliability features.

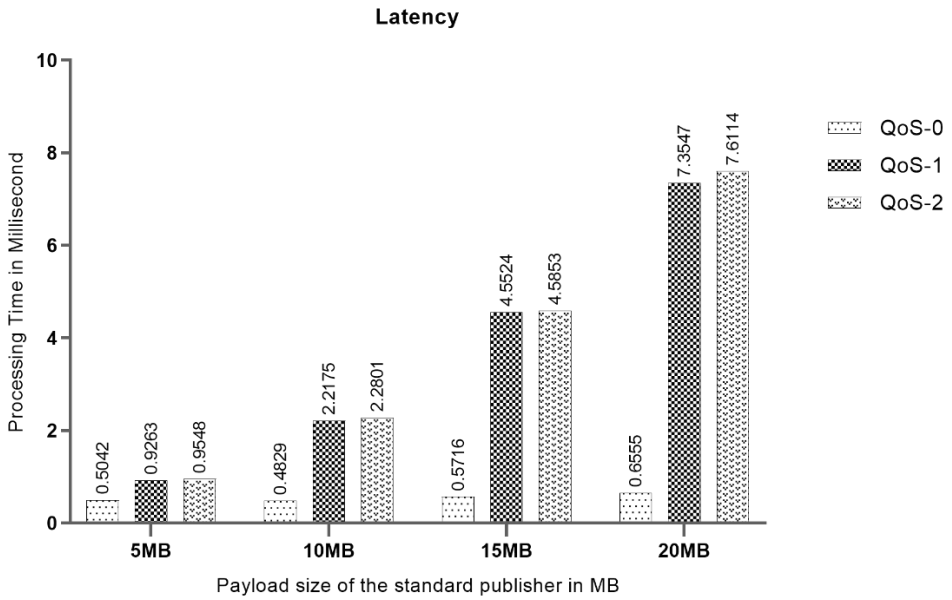

Figure 8. The impact of payload size and QoS level on latency.

Correspondingly, outcomes indicate that minimizing the size of the payload might aid in significantly reducing latency, especially during QoS-1 and QoS-2. Techniques involving payload segmentation or message frequency optimization may be able to lessen the performance effect of more extensive payloads in situations when bandwidth is unreasonable.

7. Discussion

This analysis provides practical insights into the implementation of the MQTT protocol in IoT arena, mainly focusing on the impact of varying payload sizes and QoS levels on key performance indicators such as throughput, processing time, CPU usage, traffic overhead, bandwidth consumption, and latency. The results demonstrate several significant trends in MQTT's behavior and emphasize the inherent trade-offs between efficiency and reliability that must be considered when implementing this protocol in an IoT scenario.

However, one of the most notable outcomes is the clear association between payload size and processing time. As the payload grows from 5 MB to 20 MB, processing time increases throughout all QoS levels. This tendency is expected, as bigger payloads require additional transmission resources, which will cause delays in processing. The variation in processing time across QoS levels emphasizes the importance of message reliability in resource utilization. Particularly, QoS-2, which guarantees exactly-once delivery, consistently had the greatest processing times. This is due to the extra costs associated with lengthy acknowledgment operations, which, while guaranteeing reliable delivery, grow the complexity and time of sending messages. On the other side, QoS-0, whereas offering the least reliability guarantee, has the most fast processing times, which makes it suited to instances where low latency is of greater significance than message guarantee.

Throughput, another significant performance parameter, revealed a curious pattern in which bigger payloads led to better throughput, but at different rates based on the QoS level. QoS-0 had the best throughput, indicating its low overhead, which allows for faster transmission. In comparison, QoS-2 had the lowest throughput due to the increased acknowledgment overhead as well as the necessity for reliable message delivery. This means that applications that prioritize high throughput and speed should choose QoS-0, nonetheless those that require rigorous message integrity and low packet loss might prefer QoS-1 or QoS-2, despite the trade-off in

throughput. Furthermore, QoS-0 has the lowest traffic overhead, making it suitable for non-critical applications. QoS-1 and QoS-2 come with additional overhead owing to reliability techniques.

The selection of QoS level is influenced by the need for reliability vs effective use of bandwidth. Furthermore, the chosen QoS level affects latency, which is crucial for real-time IoT applications. As implied, QoS-0 had the lowest delay, while QoS-2 had the greatest, indicating the increased amount of acknowledgment messages necessary for dependability. This finding is critical for applications that demand timely data transmission, including industrial IoT or healthcare systems, where high latency might result in performance deterioration or inability to satisfy operational demands.

Ultimately, the outcomes of this research demonstrate the significance of balancing reliability and efficiency while using MQTT in IoT systems. For use cases that require high message reliability and data integrity, QoS-1 and QoS-2 give the essential guarantees, but at the expense of additional overhead and reduced bandwidth. Conversely, QoS-0 provides significant advantages in cases where speed and low latency are more important than reliability. These findings can help developers and engineers identify the best QoS level according to the individual needs of their IoT applications.

8. Conclusion and Future Work

This research presents vital insights into the MQTT protocol's strengths and requirements, as well as efficient recommendations for IoT developers looking to maximize performance depending on their individual application requirements. The trade-offs between reliability, efficiency, and resource consumption emphasize the need of carefully selecting MQTT QoS levels to satisfy a wide range of application demands. Whereas higher QoS levels improve reliability, they are gained at the expense of increased use of resources and reduced performance. The fundamental expertise gained from this study enables developers and system designers to modify MQTT parameters to be compatible with particular application goals, such as paying priority to performance or reliability. subsequent studies could emphasize on enhancing the the MQTT protocol to handle various IoT scenarios by investigating alternative lightweight protocols, adjustable QoS techniques, or hybridization strategies that dynamically balance reliability and efficiency with respect to existing network circumstances.

References

[1] A. J. Hintaw, S. Manickam, S. Karuppayah, M. A. Aladaileh, M. F. Aboalmaaly, and S. U. A. Laghari, “A Robust Security Scheme Based on Enhanced Symmetric Algorithm for MQTT in the Internet of Things,” *IEEE Access*, vol. 11, no. March 2023, pp. 43019–43040, 2023, doi: 10.1109/ACCESS.2023.3267718.

[2] S. ul A. Laghari, W. Li, S. Manickam, P. Nanda, A. K. Al-Ani, and S. Karuppayah, “Securing MQTT Ecosystem: Exploring Vulnerabilities, Mitigations, and Future Trajectories,” *IEEE Access*, pp. 1–17, 2024, doi: 10.1109/ACCESS.2024.3412030.

[3] A. J. Hintaw, S. Manickam, S. Karuppayah, and M. F. Aboalmaaly, “A brief review on MQTT’s security issues within the internet of things (IoT),” *Journal of Communications*, vol. 14, no. 6. 2019. doi: 10.12720/jcm.14.6.463-469.

[4] L. Nastase, “Security in the Internet of Things: A Survey on Application Layer Protocols,” *Proc. - 2017 21st Int. Conf. Control Syst. Comput. CSCS 2017*, pp. 659–666, 2017, doi: 10.1109/CSCS.2017.101.

[5] 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 *Proceedings - IEEE*

Symposium on Computers and Communications, 2018, vol. 2018-June. doi: 10.1109/ISCC.2018.8538692.

[6] V. Seoane, C. Garcia-Rubio, F. Almenares, and C. Campo, "Performance evaluation of CoAP and MQTT with security support for IoT environments," *Comput. Networks*, vol. 197, p. 108338, Oct. 2021, doi: 10.1016/J.COMNET.2021.108338.

[7] H. Tschofenig and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things," *IETF*, 2016.

[8] M. Iglesias-Urkia, A. Orive, M. Barcelo, A. Moran, J. Bilbao, and A. Urbieta, "Towards a lightweight protocol for Industry 4.0: An implementation based benchmark," 2017. doi: 10.1109/ECMSM.2017.7945894.

[9] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Apr. 2014, pp. 1–6. doi: 10.1109/ISSNIP.2014.6827678.

[10] S. S. Prayogo, Y. Mukhlis, and B. K. Yakti, "The Use and Performance of MQTT and CoAP as Internet of Things Application Protocol using NodeMCU ESP8266," 2019. doi: 10.1109/ICIC47613.2019.8985850.

[11] I. Hedi, I. Špeh, and A. Šarabok, "IoT network protocols comparison for the purpose of IoT constrained networks," 2017. doi: 10.23919/MIPRO.2017.7973477.

[12] Y. Guaman, G. Ninahualpa, G. Salazar, and T. Guarda, "Comparative Performance Analysis between MQTT and CoAP Protocols for IoT with Raspberry PI 3 in IEEE 802.11 Environments," in *Iberian Conference on Information Systems and Technologies, CISTI*, 2020, vol. 2020-June. doi: 10.23919/CISTI49556.2020.9140905.

[13] D. H. Mun, M. Le Dinh, and Y. W. Kwon, "An Assessment of Internet of Things Protocols for Resource-Constrained Applications," in *Proceedings - International Computer Software and Applications Conference*, 2016, vol. 1. doi: 10.1109/COMPSAC.2016.51.

[14] D. Borsatti, W. Cerroni, F. Tonini, and C. Raffaelli, "From IoT to cloud: Applications and performance of the MQTT protocol," in *International Conference on Transparent Optical Networks*, 2020, vol. 2020-July. doi: 10.1109/ICTON51198.2020.9203167.

[15] E. Baranauskas, J. Toldinas, and B. Lozinskis, "Evaluation of the impact on energy consumption of MQTT protocol over TLS," in *CEUR Workshop Proceedings*, 2019, vol. 2470.

[16] A. Oak and R. D. Daruwala, "Assessment of Message Queue Telemetry and Transport (MQTT) protocol with Symmetric Encryption," 2018. doi: 10.1109/ICSCCC.2018.8703314.

[17] C. Thomas Oliveira, R. Moreira, F. De Oliveira Silva, R. Sanches Miani, and P. Frosi Rosa, "Improving security on IoT applications based on the FIWARE platform," in *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2018, vol. 2018-May. doi: 10.1109/AINA.2018.00104.

[18] A. J. Hintaw, S. Manickam, M. F. Aboalmaaly, and S. Karuppayah, "MQTT Vulnerabilities, Attack Vectors and Solutions in the Internet of Things (IoT)," *IETE J. Res.*, vol. 69, no. 6, pp. 3368–3397, Aug. 2023, doi: 10.1080/03772063.2021.1912651.

[19] G. Kim, S. Kang, J. Park, and K. Chung, “An MQTT-Based Context-Aware Autonomous System in oneM2M Architecture,” *IEEE Internet Things J.*, vol. 6, no. 5, 2019, doi: 10.1109/JIOT.2019.2919971.

[20] Z. Liu, T. Liang, J. Lyu, and D. Lang, “A security-enhanced scheme for MQTT protocol based on domestic cryptographic algorithm,” *Comput. Commun.*, vol. 221, pp. 1–9, May 2024, doi: 10.1016/j.comcom.2024.04.013.

[21] A. J. Hintaw, “Secure Hybrid Scheme For Securing Mqtt Protocol Based On Enhanced Symmetric Algorithm,” Universiti Sains Malaysia, 2023.

[22] C. Patel and N. Doshi, “‘a Novel MQTT Security framework in Generic IoT Model,’” in *Procedia Computer Science*, 2020, vol. 171. doi: 10.1016/j.procs.2020.04.150.

[23] O. Standard, “MQTT version 3.1. 1,” 2014.

[24] P. Client, “Eclipse Paho Client,” 2015.

[25] “Eclipse Mosquitto Broker,” 2019.