Internet of Things Network Performance: Impact of Message and Client sizes and Reliability Levels

Jiby J Puthiyidam[†] and **Shelbi Joseph**, Non-members

ABSTRACT

Consistent and timely message transmission is pivotal in the success of Internet of Things (IoT) communications. IoT applications typically involve resource-constrained devices with limited processing power, memory capacity, and battery life. These characteristics necessitate the use of specialized communication protocols for IoT applications. Message Queue Telemetry Transport (MQTT) is a widely adopted protocol within the IoT ecosystem. Ensuring the appropriate message size and reliability level is crucial for achieving successful MQTT communication. This article empirically assesses MQTT protocol performance across various Quality of Service (QoS) levels while considering different message and client sizes. Initially, we conducted a comparative analysis of message transfer performance between HTTP and MQTT protocols. Subsequently, we delved into the MQTT protocol's performance across three distinct configurations: one publisher-one subscriber, multiple publishers-one subscriber, and multiple publishers-multiple subscribers. We also examined how message and client sizes impact message transfer latency. When employing a 100-byte payload, we observed that the time delay in a network comprising 150 clients is 60.71% greater than in a network with 50 clients. Similarly, with a message size of 2500 bytes, a network with 50 clients requires 96% less time to deliver than a network with 150 clients.

Keywords: Internet of Things, MQTT Protocol, Quality of Service, IoT Networks, Message Transfer

1. INTRODUCTION

Advancements in computing and communication technology have led to a proliferation of internet-connected devices. Today, most internet traffic originates from the extensive network of devices within the Internet of Things (IoT). Over the past few decades, the IoT landscape has experienced unparalleled growth in connections, device numbers, and data transfer demands [1]. Efficient and dependable data exchange among interconnected devices play a pivotal role in the success of IoT applications. Historically, HTTP (Hyper Text Transfer Protocol) has dominated data transport in the internet domain, thanks to its achievements within the TCP/IP network [2]. However, the requirements of IoT applications differ significantly. IoT applications are comprised of resource-constrained devices characterized by limited processing capabilities, memory capacity, and battery life. Major application areas of IoT include patient monitoring systems, smart homes, smart agriculture, industrial monitoring and target tracking, to name a few [3]. Research in this field has underscored the inadequacy of HTTP for addressing the needs of remote, resource-constrained IoT devices. Consequently, this limitation has prompted us to shift our focus toward specialized protocols tailored to the unique demands of IoT applications.

Prominent communication protocols employed in IoT applications include CoAP (Constrained Application Protocol) [4], AMQP (Advanced Message Queuing Protocol) [5], MQTT (Message Queuing Telemetry Transport) [6], among others. MQTT is the most widely adopted communication protocol within the IoT ecosystem. MQTT's popularity is attributed to its lightweight nature and publish-subscribe paradigm adherence. These characteristics make MQTT well-suited for devices with constrained resources and scenarios involving suboptimal network conditions, such as low bandwidth and high latency [7]. MQTT finds extensive use in major public cloud platforms, including Amazon Web Services, Microsoft Azure, and Google Cloud Platform [8]. As of 2020, Amazon AWS commands a 40% market share, Microsoft Azure holds 31%, and Google Cloud Platform captures 26% of the public IoT and Cloud Platforms market [9].

Frequent transmission and timely delivery of messages are essential in IoT applications [10]. Reliability in message delivery in the MQTT protocol is achieved with the help of three reliability levels it offers, namely, QoS0, QoS1 and QoS2 [11]. Quality of Service (QoS) gives the client the power to choose a level of service that matches its network reliability and application logic. The choice of the proper QoS influences the performance of IoT applications. Selecting the proper QoS and the message size suitable for a particular application is a research question to be addressed. Some experiments proved that reliability is compromised at the expense of performance and vice versa. Though the data size of IoT applications is minimal, it is communicated in large quantities through

Manuscript received on February 3, 2023; revised on September 11, 2023; accepted on January 9, 2024. This paper was recommended by Associate Editor Nattapong Kitsuwan.

The authors are with School of Engineering, Cochin University of Science and Technology, Kochi, India.

[†]Corresponding author: jibyjp@cusat.ac.in

Digital Object Identifier: 10.37936/ecti-eec.2024221.252941

the network [30]. Consequently, it becomes imperative to explore methodologies for reducing network traffic and augmenting network performance for the effective execution of IoT communications.

This paper presents a comprehensive analysis of the message transfer capability of the MQTT protocol under different reliability levels and with varying message and client sizes. The particular QoS to be used for message transmission depends on application requirements. Experiments were conducted to analyse the impact of message payload size and the number of clients connected to the network. This work may help users choose appropriate QoS and message sizes for their IoT applications.

Major contributions of the paper are summarized below:

- A study on the performance of various IoT communication protocols was performed.
- Analysed the message transfer performance of HTTP and MQTT protocol for varying message sizes from 1 to 2500 bytes.
- Analysed the message delivery performance of MQTT protocol for different message and client sizes.
- The performance of MQTT protocol is analysed with configurations such as one publisher-one subscriber, multiple publishers-one subscriber, and multiple publishers-multiple subscribers.

Section 2 of this paper discusses some existing works related to the chosen field. MQTT protocol and its reliability aspects are briefed in section 3. A detailed experimental evaluation of MQTT message passing with varying reliability levels is presented in section 4, and section 5 analyses the results obtained with the experiment. Section 6 concludes our discussion.

2. RELATED WORKS

In this section, we review existing literature that examines the performance of communication protocols in the context of the Internet of Things. To begin with, we explore studies that have compared HTTP, a widely recognized internet protocol, and MQTT. Subsequently, we discuss the literature on the performance analysis of diverse IoT communication protocols.

Yokatani et al. [12] analysed HTTP and MQTT protocols' required bandwidth and resource requirements with different data sizes and topic lengths. This paper concludes that MQTT is more suitable for connecting IoT devices than HTTP. Manohar and Asir[13] analysed the data consumption pattern of HTTP and MQTT protocols and proved that HTTP protocol use more resources and is not suitable for IoT applications due to its overheads. They proved that MQTT consumes data only when it sends a payload and consumes negligible data in all other situations. Hantrakul et al.[14] found that the HTTP protocol consumes ten times more power than the MQTT protocol, but MQTT sends ten times more messages than the HTTP protocol in one hour of operation. Atmoko et al. [15] implemented real-time temperature and humidity data acquisition using HTTP and MQTT protocol in their work. The test result indicates that MQTT is an option for real-time data acquisition applications based on IoT. The study of Wukkadada et al.[16] proved that the MQTT offers higher throughput and lower power consumption than HTTP. MQTT uses less power to maintain an open connection. The work by Sasaki and Yokotani[17] considered traffic volume, access delay, and network resources as evaluation metrics. The authors claim that MQTT is superior to HTTP when applied to IoT applications.

Lee et al. [18] performed a correlation analysis of the MQTT protocol's end-to-end delay and message loss rate under different QoS levels and payloads. They suggest the use of MQTT protocol with appropriate QoS levels for better service. Upadhyay et al. [19] proved that the power consumption of the MQTT protocol is much lower than CoAP, and MQTT has a 30% faster performance. The work by Silva et al. [20] compared messaging protocols for IoT systems such as MQTT, CoAP, AMQP and HTTP. This paper claim that MQTT offers the highest level of reliability among the four thanks to the three levels of Quality of Services it supports. An ns-3-based simulation tool for evaluating the QoS in MQTT-based IoT systems is presented by Handosa et al. [21]. Luoto et al. [22] presented a work that substituted REST-based communication with MQTT communication and proved that MQTT uses considerably less CPU time and memory and thus improves energy efficiency. Sing and Baranwal [23] identified various QoS metrics related to the three major components of IoT (Things, Computing and Communication). Researchers and professionals can identify relevant and essential QoS metrics for developing models for different challenges. A communication infrastructure with two popular MQTT brokers, RabbitMq and mosquitto, was implemented by Oliveira et al. [24]. RabbitMQ has lower average latency and is better for more robust applications such as Smart Meters communicating in a Smart Grid. For applications that do not require a lot of data flow, the mosquitto broker is better. Toldinas et al. [25] proved that the energy consumption of the MQTT protocol is different at various QoS levels. This paper recommends that developers of IoT systems choose the most appropriate QoS level by considering reliability, latency and energy needed to communicate concerning the battery lifetime.

A study of the existing literature proved that HTTP, the standard internet protocol, consumes more resources. Therefore, HTTP is not recommended for IoT networks containing small devices with little processing power, limited storage and communication capabilities. Many communication protocols explicitly designed for constrained IoT networks were analysed. The literature review proved that MQTT is the most widely used communication protocol in the IoT environment due to its peculiar features, such as its lightweight nature, message reliability, small fixed message header and suitability for constrained applications.

3. MESSAGE QUEUE TELEMETRY TRANSPORT (MQTT) PROTOCOL

In 1999, MQTT was initially conceptualized by IBM's Andy Stanford Clark and Cirrus Link's Arlen Nipper to link sensors in oil pipelines to communication satellites. They focused on achieving minimal battery drain and conserving bandwidth consumption [26]. However, as the IoT experienced exponential growth and the necessity arose to facilitate connections and communications among low-powered, resource-constrained devices, MQTT emerged as a prominent and indispensable protocol in the IoT landscape. Today, MQTT stands as a well-established Machine-to-Machine (M2M) protocol, enjoying widespread adoption and support from numerous organizations, including IBM, Facebook, Eurotech, Cisco, Red Hat, M2Mi, and Amazon Web Services (AWS) [19].

The MQTT protocol involves two main components: clients and a broker, often referred to as the server. Clients are devices capable of performing actions such as publishing messages, subscribing to messages, or both [8]. MQTT clients communicate by utilizing message topics as the means of interaction. When a client takes on a subscriber role, it specifies its areas of interest by including topic information in its connection request to the broker. Following establishing a connection with the broker, a publisher client gathers data from its environment and transmits it to the broker, attaching a topic name to the message. Within the MQTT protocol, the broker is a central entity responsible for receiving messages from publishing clients, filtering them based on the associated topic name, and forwarding them to all clients subscribed to that particular topic. The communication pattern of the MQTT protocol is visually depicted in Fig. 1.

For communication purposes, the MQTT protocol utilizes a sequence of control packets. One of the significant advantages of MQTT is the 2-byte fixed header associated with messages, the lowest among IoT communication protocols. This feature produces low overhead for low bandwidth applications, making MQTT prominent among the IoT protocols. Control messages such as CONNECT, CONACK, PUBREL etc. need not carry any client data and hence require only the two byte message fixed header. The messages carrying message payload need to use more bytes for message payload. MQTT message header format is given in Fig. 2.

The MQTT message types are defined by allocating 4 bits, encompassing bits 0 to 3 within byte 1 of the message header. This allocation permits the definition of 16 distinct message types. However, the MQTT specification explicitly defines only 14 message types (ranging from 1to 14), designating the values 0 and 15 as "reserved" for potential future expansions [25]. Meanwhile, the bits occupying positions 4 to 7 of byte 1 within the message header serve as flags unique to various MQTT control packet types. The DUP flag is employed to signify that the message is a duplicate one

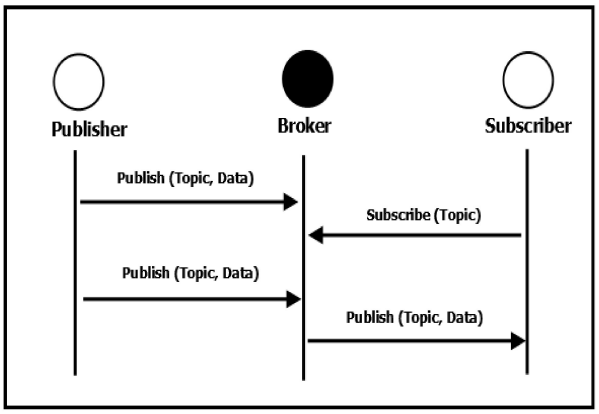

***Fig. 1**: MQTT Communication Pattern [8].*

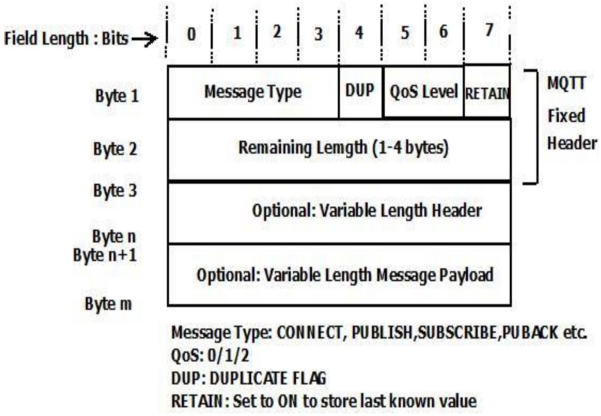

***Fig. 2**: MQTT message header format [11].*

and is re-transmitted due to the recipient (a client or broker) not acknowledging the original message. The QoS flag specifies a PUBLISH message's reliability level (QoS 0, QoS 1, or QoS 2). When the RETAIN flag is activated, the broker preserves the most recent message on a specific topic. This enables a disconnected or newly connected client to access the latest published message when it establishes a connection with the broker. The second byte of the message header specifies the optional variable header's size and the payload message's length.

3.1 MQTT Reliability

The QoS offerings provided by the MQTT protocol play a pivotal role in ensuring message reliability [18]. They simplify communication within unreliable networks and guarantee message delivery, regardless of the reliability of the underlying transport layer. QoS parameters such as Throughput and packet delay are important metrics for assessing the quality of networks [28][29]. In the MQTT protocol, the reliability level of message transfer is not end-to-end but operates between the client and the server. The subscriber client establishes the desired reliability level for receiving messages through the CONNECT command. In contrast, the publisher client specifies the reliability of the published message

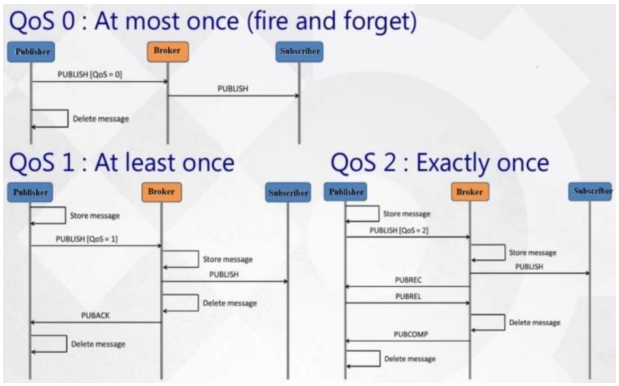

***Fig. 3**: MQTT QoS levels [27].*

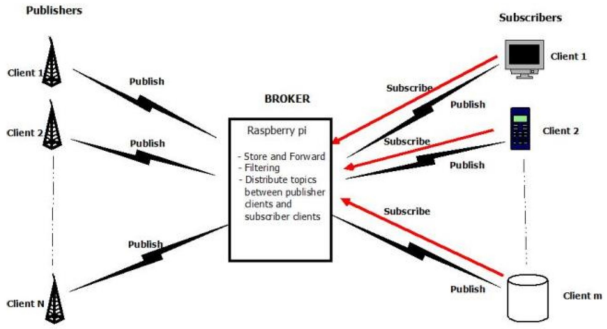

***Fig. 4**: MQTT Broker Architecture.*

***Table 1**: Technical specification of experimental setup.*

Server	
Processor	Raspberry Pi 3
RAM	1 GB
Model	Broadcom BCM2837 quadcore
Operating System	Raspbian
CPU clock frequency	1.2 GHz
Wi-Fi	802.11 b/g/n
Client	
Processor	Intel i7
RAM	8 GB
Operating System	Ubuntu 20.04
Programming Language	Python 3.7

using the PUBLISH command. Ultimately, the message is conveyed to the recipient with the lowest QoS level chosen by either the publisher or the subscriber. MQTT delineates three distinct levels of Quality of Service:

QoS 0: At most once (Fire and forget)
QoS 1: At least once (Duplicates allowed)
QoS 2: Exactly once

QoS 0 is the default reliability level MQTT protocol offers. It does not guarantee message delivery. QoS 0 is preferred when the communication network is reliable and losing messages is acceptable. For example, atmospheric pressure or temperature sensor data can publish with QoS 0. QoS 1 guarantees message delivery. Subscribers should acknowledge the receipt of messages. The message is delivered to the subscriber at least once. The message is retransmitted if there is no acknowledgement within a pre-specified period. QoS 1 is used with applications where guaranteed message delivery is essential and can handle duplicate messages. Reporting machine health data to ensure maintenance at the right time is an example of QoS 1 message delivery. QoS 2 is the highest and most expensive quality of service MQTT offers. This QoS has been used in mission-critical scenarios that neither accept the loss of messages nor duplicate messages. Example:- billing and invoicing systems where duplicate entries are not accepted. Each message should properly be delivered exactly once. The message delivery at different QoS levels by MQTT protocol is depicted in Fig. 3.

4. EXPERIMENTAL EVALUATION

4.1 Experimental Setup

We established a simulation environment to assess the reliability features of the MQTT protocol. The MQTT broker was deployed on a Raspberry Pi 3 processor with 1 GB of RAM and the Rasbian operating system. Client nodes powered by ESP32 MCUs were configured to run the MQTT clients with their respective sensors attached. To simulate many clients concurrently and intensify network load, we utilized a laptop computer supporting an Intel i7 processor and running Ubuntu 20.04 as the client operating system. Remarkably, all publishers and subscribers operated within this same laptop PC. The technical specifications of the testing environment can be found in Table 1. Python 3.7 was the programming language employed to code the simulation environment. Our experiment focused on assessing message transfer time delays from publisher clients to subscriber clients while varying the QoS levels for the same dataset. Since there was a lack of standardized datasets suitable for testing, we generated the necessary data for our testing environment. This process involved selecting appropriate message and client sizes tailored to each experiment's requirements. The architecture of the testing environment is shown in Fig. 4.

4.2 Experimental Evaluation

The experimental procedure followed in our MQTT QoS performance analysis is summarized in figure 5. Initially the performance of HTTP and MQTT were analysed in the IoT environment. After proving that MQTT is suitable for IoT applications, its performance is analysed with different reliability levels and client configurations. Client configurations such as one publisher-one subscriber, multiple publisher-one subscriber and multiple publisher-multiple subscriber were selected for analysis. The flow of experimental procedure is depicted in Fig. 5.

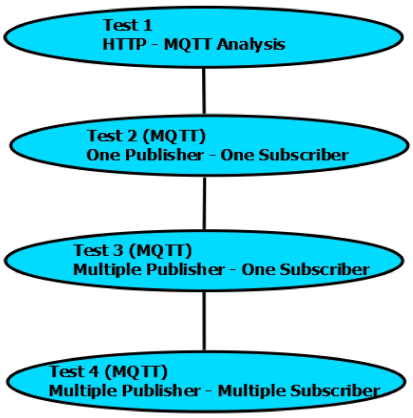

***Fig. 5**: Flow of Experimental procedure.*

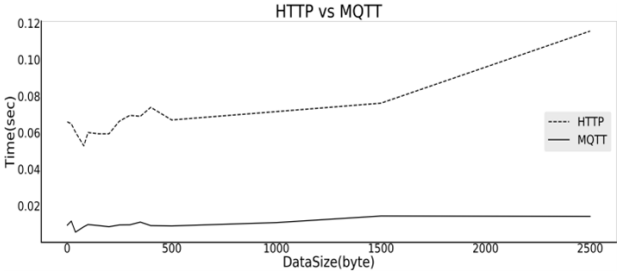

***Fig. 6**: HTTP vs MQTT performance.*

4.2.1 HTTP vs MQTT

In the initial experiment, we conducted a comparative analysis to assess the performance of HTTP and MQTT protocols regarding message transfer latency. Specifically, we examined the delivery delays for messages of varying sizes within a single-client environment. HTTP has traditionally been employed for widespread internet access, necessitating a three-way handshake with TCP each time information is accessed. In contrast, MQTT streamlines the communication session by avoiding the need to connect and disconnect each message repeatedly. In the case of HTTP, the access delay is measured when the client requests data from the server. Similarly, in MQTT, we focused on evaluating the end-to-end delay, measuring the time it takes for a publisher to transmit a message and for a subscriber to receive it through the MQTT broker. Given the absence of standardized datasets for IoT applications and the variability in benchmark values, we opted for an experimental approach. We selected a logarithmic scale of message sizes, spanning from 1 to 2500 bytes, for our evaluation. The outcomes of this experiment is graphically presented in Fig. 6.

4.2.2 MQTT (One Publisher – One Subscriber)

Subsequently, our focus shifted to evaluating the message-passing performance of the MQTT protocol while varying the levels of reliability. We manipulated the QoS settings for publishers and subscribers, all within the same data set. In the initial phase, we scrutinized message delivery delays in a scenario featuring a single publisher and subscriber. Within this setup, we assessed message sizes spanning from 1 byte to 2500 bytes. Initially, both the publisher and subscriber were configured with QoS 2. We observed variations in message transmission time at the publisher and the subsequent reception by the subscriber. We then maintained the publisher at QoS 2 while altering the subscriber's reliability level to QoS 1 and subsequently to QoS 0, monitoring the corresponding latency changes. In the subsequent phase, we designated the publisher's reliability as QoS 1. We assessed message delivery time delays using the same data set but with subscribers employing different reliability levels: QoS 2, QoS 1, and QoS 0. During the final phase, we configured the publisher at QoS 0 while systematically adjusting the subscriber's QoS values from QoS 2 to QoS 0. This phase allowed us to investigate the disparities between message sending and receiving times. The specific QoS configurations for both publishers and subscribers in each experimental round are detailed in Table 2.

***Table 2**: Publisher and Subscriber QoS Setup.*

	Publisher QoS	Subscriber QoS
Round 1	QoS 2	QoS 2
		QoS 1
		QoS 0
Round 2	QoS 1	QoS 2
		QoS 1
		QoS 0
Round 3	QoS 0	QoS 2
		QoS 1
		QoS 0

Experiments over all nine combinations of publisher-subscriber QoS levels reveal that in each round, the configurations where subscriber QoS is set to 0 yields the best result (minimum delay). The results of subscriber QoS 0 for various publisher reliability levels (QoS 0, QoS 1 and QoS 2) are plotted in Fig. 7.

4.2.3 MQTT (Multiple Publisher – One Subscriber)

In subsequent experiments, we conducted tests to evaluate message latency involving multiple publishers operating under different MQTT QoS levels. In this scenario, five publishers transmitted varying-sized messages to a single subscriber. Each publisher concurrently sent data packets ranging from 1 to 6000 bytes. As in the earlier experiment, we scrutinized the message delivery latency for various payload sizes while configuring the publishers with QoS 0 and allowing the subscriber's QoS to vary from 0 to 2. Additionally, we examined configurations with the publisher set at QoS 1 and QoS 2, while the subscriber's QoS ranged from 0 to 2. Across these tests, it is observed that configurations with subscriber QoS 0 consistently yielded the most favourable results. For a concise overview of the outcomes involving subscriber QoS 0 in conjunction with publisher QoS 0, 1, and 2, refer

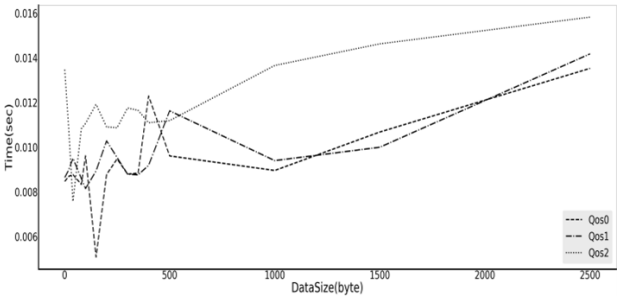

***Fig. 7**: One publisher - One subscriber performance.*

***Table 3**: Multiple publisher – Multiple subscriber testing environments.*

Publisher QoS	Subscriber QoS	Publisher – Subscriber client sizes
QoS 1	QoS 1	25 publishers – 25 subscribers
QoS 1	QoS 1	50 publishers – 100 subscribers
QoS 1	QoS 1	70 publishers – 300 subscribers

to Fig. 8, which summarizes the results.

4.2.4 MQTT (Multiple Publisher – Multiple Subscriber)

In the next phase of our study, we thoroughly examined the performance of various reliability levels within the MQTT protocol, specifically within the context of multiple publishers and subscribers scenarios. Given the absence of standardized datasets or benchmark values for IoT applications, we opted for experimental comparisons, where we selected configurations encompassing 25 publishers and 25 subscribers, 50 publishers and 100 subscribers, and 70 publishers and 300 subscribers. These configurations served as the basis for our evaluation, with message sizes ranging from 1 byte to 5000 bytes. It is important to note that the publisher and subscriber numbers were selected randomly for testing. For this experimental phase, publisher and subscriber clients were configured with a QoS level 1. This choice is justifiable in applications where the punctual delivery of every message is paramount. QoS 1 ensures message delivery while introducing only a slight increase in latency compared to QoS 0, which is generally acceptable. Table 3 provides a comprehensive overview of the configurations employed in evaluating multiple publishers and subscribers.

The results obtained are summarized in the graph in Fig. 9. The graph clearly shows that the message latency increases as the message size increases. The latency also increases with client size.

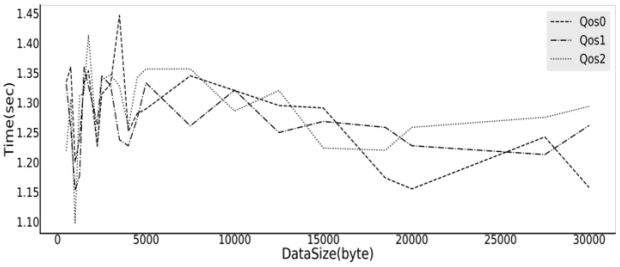

***Fig. 8**: Multiple publishers - One subscriber performance.*

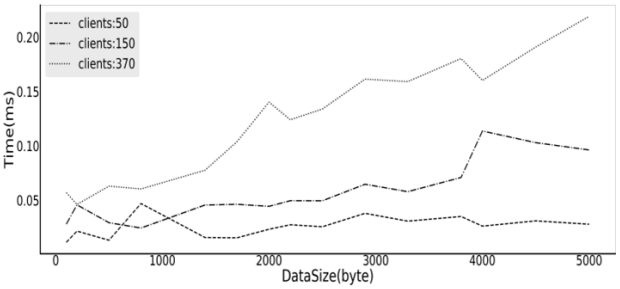

***Fig. 9**: Multiple publishers - Multiple subscriber performance.*

5. RESULTS AND DISCUSSION

The analysis of the HTTP and MQTT protocols underscores a significant disparity in message transfer delay. Specifically, it becomes evident that the HTTP protocol exhibits message transfer delays that are eight to ten times greater than those observed in the MQTT protocol, even in small message sizes and single-client environments. As message sizes expand, this latency difference between the two protocols becomes more pronounced. The performance disparity would be even more pronounced in scenarios involving multiple clients transmitting multiple messages. Therefore, our experimental analysis emphasizes the necessity of employing specialized communication protocols like MQTT within IoT environments that feature devices characterized by limited sensing, processing, storage, and network capabilities. Such protocols are crucial for ensuring efficient and reliable communication under these constraints.

The experimental analysis of the MQTT protocol's performance across different reliability levels include diverse messages and client sizes. In the context of the single publisher - single subscriber configuration analysis, it was observed that among all QoS combinations, those involving publisher QoS 0 and subscriber QoS 0, as well as publisher QoS 1 and subscriber QoS 0, exhibited the smallest time differences. This observation aligns with the theoretical premise that the reliability of message delivery is adapted to the lowest QoS level set by either the publisher or the subscriber. Similarly, a similar trend was noted when exploring the configuration of multiple publishers with a single subscriber. Combinations featuring subscriber QoS 0, such as publisher QoS 1 - subscriber QoS 0, publisher QoS 0 - subscriber QoS 0,

and publisher QoS 2 - subscriber QoS 0, outperformed all other combinations regarding message delivery latency. This consistency in results highlights the significance of subscriber QoS 0 in optimizing message delivery latency across various QoS configurations.

In multiple publishers-multiple subscriber's configurations, both the publisher and subscriber QoS were set at QoS 1. This choice was made due to its ability to ensure guaranteed message delivery, and the increase in latency associated with QoS1, compared to QoS0, was deemed acceptable in light of this guarantee. As anticipated, the message delivery time increased as the message size expanded. Notably, the percentage increase in time difference was directly proportional to the number of clients involved. Fig. 10 clearly illustrates this relationship, indicating that as the message size increased, the percentage time difference concerning client size also experienced a corresponding increase.

When utilizing a 100-byte payload, the time delay for a configuration involving 150 clients (comprising 50 publishers and 100 subscribers) is approximately 60.71% greater than the setup with 50 clients (consisting of 25 publishers and 25 subscribers)—scaling up the number of clients to 370 results in a substantial increase of 418.8%. Conversely, with a message size of 2500 bytes, a setup with 50 clients necessitates 96% less time to deliver a message compared to the scenario with 150 clients. Furthermore, a notable reduction of 432% in latency is observed when compared the 50 client setup with 370 clients. This result emphasizes the justification for setting the reliability level to QoS1 for both publishers and subscribers, as it ensures message delivery while incurring a manageable increase in latency.

Our experimental analysis demonstrates that across all MQTT reliability levels, QoS 0 consistently exhibits superior performance in terms of latency, aligning with the expected theoretical outcomes. There is only a marginal disparity between QoS 0 and QoS 1 performance. However, for critical IoT applications where guaranteed message delivery is paramount, QoS 1 is the preferred choice, making it a recommended selection for most real-world IoT applications. In the context of multiple publisher-multiple subscriber environments, it becomes apparent that as the number of clients increases, the message delivery time also experiences an increase. In a configuration involving 25 publishers and 25 subscribers, there is a slight variation in latency across different reliability levels concerning message sizes. This can be attributed to the dominant role of the initial communication channel establishment time between the publisher and the server, as well as between the server and the subscriber, which outweighs the time required for message transmission. Nevertheless, as the number of clients (publishers and subscribers) escalates, the time difference becomes proportionate to the message payload, as numerous messages are exchanged over established connections. In such scenarios, the initial setup time becomes negligible compared to the volume of data transmitted.

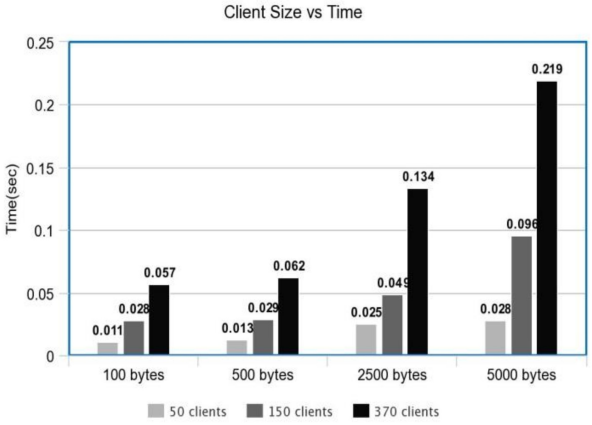

***Fig. 10**: Impact of client and message sizes.*

The findings from our experimental evaluation highlight the importance of maintaining a small message size in IoT communication. In many IoT applications, limiting the number of participating clients may not be feasible or desirable. Therefore, one practical approach to reducing data transfer latency in IoT communications is to minimize the message size. Additionally, by ensuring that message reliability is set to QoS 1, we can guarantee the timely delivery of critical messages while maintaining acceptable latency.

Our experiment results lead to the following conclusions:

- QoS 0 is recommended for applications involving constant and frequent sensor readings where occasional loss of one or more messages is acceptable. For instance, consider scenarios where atmospheric pressure or temperature data is collected at regular time intervals, and data changes are infrequent. In such cases, the loss of a few data readings may not significantly impact the application's outcome.
- QoS 1 is well-suited for applications where guaranteed message delivery is essential, and the system can handle duplicate messages. An example is the transmission of machine health data to ensure timely maintenance.
- QoS 2 messages should be limited to mission-critical applications that cannot tolerate message loss or duplication. Billing and invoicing systems, where duplicate entries are unacceptable, serve as an example where every message must be delivered precisely once.

In summary, the message size and reliability level should align with the specific requirements and priorities of the IoT application to ensure optimal performance and reliability.

The proliferation of small devices within IoT networks has led to the generation and transmission of vast volumes of data across these constrained networks. A significant portion of this data may not substantially impact the intended application. For instance, consider atmospheric temperature and pressure sensors that continuously collect data from their surroundings regularly

and relay it through a central broker. Since such data changes infrequently, the transmission often involves redundant or inconsequential information. To enhance network performance, it becomes crucial to curtail the transfer of irrelevant data that does not significantly influence the application's outcome. Moreover, not all messages are of equal importance. Some messages hold greater significance depending on their source. These critical messages warrant quick processing and delivery. For instance, messages originating from fire alarms or gas leak sensors necessitate immediate transmission to subscribers without delay, irrespective of the order in which they were generated or queued.

In our forthcoming endeavours, we plan to develop innovative techniques to identify and prevent the transfer of redundant and identical data. Additionally, we intend to design and implement methods within the MQTT broker to manage and prioritize data based on their significance effectively. These future initiatives aim to enhance the efficiency and relevance of data transmission of MQTT protocol within IoT networks.

6. CONCLUSION

MQTT has emerged as a highly successful communication protocol within the IoT landscape. The MQTT protocol achieves message delivery reliability through its three QoS levels: QoS 0, QoS 1, and QoS 2. The core objective of this research endeavour is to delve into the performance of message delivery within the MQTT communication protocol across its various reliability levels while also considering different message sizes. The insights garnered from this study can aid users in making informed choices regarding the appropriate QoS levels and message lengths tailored to their specific IoT applications. We conducted a comparative analysis between HTTP, the conventional internet protocol, and MQTT to initiate our investigation. The results confirmed MQTT's suitability for IoT applications, particularly regarding message delivery latency. Subsequently, we comprehensively explored MQTT's message delivery performance, systematically examining the effects of different reliability levels and varying data sizes. We explored diverse configurations, including single publisher-single subscriber, multiple publishers-single subscriber, and multiple publishers-multiple subscriber's scenarios. The analysis outcomes underscored the performance advantages of configurations where the subscriber operates at QoS 0 on one side, surpassing other setups. Nevertheless, the choice to set both publisher and subscriber reliability to QoS 1 is justified in scenarios where the guaranteed delivery of each message holds critical significance. QoS 1 ensures message delivery while incurring only a slight increase in latency, which remains within acceptable limits. In scenarios involving multiple publishers and multiple subscribers, the message delivery latency escalates with increases in message size and client numbers. Since limiting client size may be beyond users' control in many applications, striving for smaller message payloads is advisable to increase communication speed. Additionally, optimizing network performance can be achieved by reducing the transmission of duplicate data. In conclusion, this research provides valuable insights into optimizing MQTT's message delivery performance within IoT contexts, offering practical recommendations for enhancing communication efficiency and reliability

REFERENCES

[1] Xiao, Yanjun, Eryue Pei, Kuan Wang, Wei Zhou, and Yanchun Xiao, "Design and Research of M2M Message Transfer Mechanism of Looms for Information Transmission," *IEEE Access*, vol. 10, pp. 76136-76152, 2022.

[2] Wong, Clinton, *Http pocket reference: Hypertext transfer protocol*, "O'Reilly Media, Inc.," 2000.

[3] Khanna, Abhishek, and Sanmeet Kaur, "Internet of things (IoT), applications and challenges: a comprehensive review," *Wireless Personal Communications*, vol. 114, pp. 1687-1762, 2020.

[4] Bansal, Sharu, and Dilip Kumar, "Enhancing constrained application protocol using message options for internet of things," *Cluster Computing*, vol. 26, no. 3, pp. 1917-1934, 2023.

[5] Basavaraju, Nandeesh, Naveen Alexander, and Jochen Seitz, "Performance Evaluation of Advanced Message Queuing Protocol (AMQP): An Empirical Analysis of AMQP Online Message Brokers," In *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, 2021, pp. 1-8.

[6] Soni, Dipa, and Ashwin Makwana, "A survey on mqtt: a protocol of internet of things (iot)," In *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*, vol. 20, 2017, pp. 173-177.

[7] Dizdarević, Jasenka, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1-29, 2019.

[8] Naik, Nitin, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," In *2017 IEEE international systems engineering symposium (ISSE)*, 2017, pp. 1-7.

[9] Eclipse Foundation, IoT developer survey results, https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2020.pdf/, 2020, (Accessed 18 December 2021).

[10] Al Enany, Marwa O., Hany M. Harb, and Gamal Attiya, "A Comparative analysis of MQTT and IoT application protocols," In *2021 International Conference on Electronic Engineering (ICEEM)*, 2021, pp. 1-6.

[11] Errata OS, MQTT Version 3.1, 1 Plus Errata 01, (Accessed 12 July 2022).

[12] Yokotani, Tetsuya, and Yuya Sasaki, "Comparison

with HTTP and MQTT on required network resources for IoT," In *2016 international conference on control, electronics, renewable energy and communications (ICCEREC)*, 2016, pp. 1-6.

[13] Manohar, Hansa Lysander, and T. Reuban Gnana Asir, "Data consumption pattern of MQTT protocol for IoT applications," In *Smart Secure Systems–IoT and Analytics Perspective: Second International Conference on Intelligent Information Technologies. ICIIT 2017, Chennai, India, December 20-22, 2017, Proceedings 2*, pp. 12-22. Springer Singapore, 2018.

[14] Hantrakul, Kittikorn, Snit Sitti, and Nasi Tantitharanukul, "Parking lot guidance software based on MQTT Protocol," In *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*, 2017, pp. 75-78.

[15] Atmoko, R. A., R. Riantini, and M. K. Hasin, "IoT real time data acquisition using MQTT protocol," In *Journal of Physics: Conference Series*, vol. 853, no. 1, p. 012003, 2017.

[16] Wukkadada, Bharati, Kirti Wankhede, Ramith Nambiar, and Amala Nair, "Comparison with HTTP and MQTT in Internet of Things (IoT)," In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2018, pp. 249-253.

[17] Sasaki, Yuya, and Tetsuya Yokotani, "Performance evaluation of MQTT as a communication protocol for IoT and prototyping," *Advances in Technology Innovation*, vol. 4, no. 1, pp. 21-29, 2019.

[18] Lee, Shinho, Hyeonwoo Kim, Dong-kweon Hong, and Hongtaek Ju, "Correlation analysis of MQTT loss and delay according to QoS level," In *The International Conference on Information Networking 2013 (ICOIN)*, 2013, pp. 714-717.

[19] Upadhyay, Yuvraj, Amol Borole, and D. Dileepan, "MQTT based secured home automation system," In *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 2016, pp. 1-4.

[20] Silva, Daniel, Liliana I. Carvalho, José Soares, and Rute C. Sofia, "A performance analysis of internet of things networking protocols: Evaluating MQTT, CoAP, OPC UA," *Applied Sciences*, vol. 11, no. 11, pp. 4879, 2021.

[21] Handosa, Mohamed, Denis Gračanin, and Hicham G. Elmongui, "Performance evaluation of MQTT-based internet of things systems," In *2017 Winter simulation conference (WSC)*, 2017, pp. 4544-4545.

[22] Luoto, Antti, and Kari Systä, "Fighting network restrictions of request-response pattern with MQTT," *Iet Software*, vol. 12, no. 5, pp. 410-417, 2018.

[23] SSingh, Manisha, and Gaurav Baranwal, "Quality of service (qos) in internet of things," In *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2018, pp. 1-6.

[24] de Oliveira, Davi L., Artur F. da S. Veloso, José VV Sobral, Ricardo AL Rabêlo, Joel JPC Rodrigues, and Petar Solic, "Performance evaluation of mqtt brokers in the internet of things for smart cities," In *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, 2019, pp. 1-6.

[25] Toldinas, Jevgenijus, Borisas Lozinskis, Edgaras Baranauskas, and Algirdas Dobrovolskis, "MQTT quality of service versus energy consumption," In *2019 23rd International Conference Electronics*, 2019, pp. 1-4.

[26] Bryce, Robert, and Gautam Srivastava, "The addition of geolocation to sensor networks," In *ICSOFT*, 2018, pp. 796-802.

[27] "What sort of Quality of Service (QoS) does MQTT offer?", Devopedia for developers by developers, https://devopedia.org/mqtt#qst-ans-6. (Accessed 23 June 2022).

[28] Uthansakul, Peerapong, Patikorn Anchuen, Monthippa Uthansakul, and Arfat Ahmad Khan, "QoE-Aware self-tuning of service priority factor for resource allocation optimization in LTE networks." *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 887-900, 2019.

[29] Uthansakul, Peerapong, Patikorn Anchuen, Monthippa Uthansakul, and Arfat Ahmad Khan, "Estimating and synthesizing QoE based on QoS measurement for improving multimedia services on cellular networks using ANN method," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 389-402, 2019.

[30] Lakshmanna, Kuruva, Rajesh Kaluri, Nagaraja Gundluru, Zamil S. Alzamil, Dharmendra Singh Rajput, Arfat Ahmad Khan, Mohd Anul Haq, and Ahmed Alhussen, "A review on deep learning techniques for IoT data," *Electronics*, vol. 11, no. 10, pp. 1604, 2022.

Jiby J Puthyidam is an Assistant Professor in Computer Engineering at Govt. Model Engineering College, Ernakulam, India. He has completed his Post-Graduation (M.Tech in Computer and Information Science) from Cochin University of Science and Technology (CUSAT), Kochi, India and currently pursuing Ph.D. in School of Engineering, Cochin University of Science and Technology (CUSAT). His Research interests include Internet of Things, IoT network security, lightweight cryptography, Data Mining and Machine Learning.

Shelbi Joseph is a Professor at Division of Information Technology, School of Engineering, Cochin University of Science and Technology, Kochi, India. He has completed his Post-Graduation (M.Tech in Computer Science and Engineering) from National Institute of Technology, Tiruchirappalli, India and Ph.D. in Software Reliability from Cochin University of Science and Technology. His areas of interest include Software Engineering, Software Reliability, Open Source Software, Big Data, Data Mining, and IoT. He has several publications in National and International Journals and Conference proceedings to his credit.