Performance Evaluation of IoT-Enabled Edge Computing Infrastructure for mHealth Services

Hermyson Cassiano, Kádna Camboim, David Beserra and Jean Araujo

Abstract The rapid technological advancements and growing demand for innovative healthcare solutions highlight the critical role of integrating the Internet of Things (IoT) with mobile health (mHealth) services. This study evaluates the performance of an IoT platform within a mHealth context, focusing on the MQTT protocol's effectiveness for healthcare data communication. Using an *Orange Pi Win Plus* board as the IoT platform, we simulated real-world mHealth conditions with varying workloads to assess platform resilience and scalability. Representative test scenarios were developed to simulate normal, increasing, and extreme load conditions, measuring key metrics such as CPU usage, memory consumption, throughput, and latency. Data were collected and analyzed using custom scripts to evaluate the platform's response across different Quality of Service (QoS) levels. Results indicated that the platform could effectively manage standard and moderately high demands, while performance under extreme loads highlighted areas for optimization. This study concludes that the MQTT-based IoT platform demonstrated reliable performance in the mHealth environment, providing a basis for future optimizations and scalability improvements.

Hermyson Cassiano and Kádna Camboim
Universidade Federal do Agreste de Pernambuco (UFAPE), Garanhuns, Brazil, e-mail: {hermyson.oliveira, kadna.camboim}@ufape.edu.br

David Beserra
École Pour l'Informatique et les Technologies Avancées (EPITA), Paris, France, e-mail: david.beserra@epita.fr

Jean Araujo
Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Portugal, e-mail: jean.araujo@ua.pt

1 Introduction

The Internet of Things (IoT) has emerged as a transformative technology, redefining interactions between individuals and connected devices. Its capability to facilitate remote, real-time operations and information access has sparked significant interest across multiple fields. IoT encompasses an interconnected network of physical objects capable of autonomously collecting and sharing data without direct human intervention [2].

Advancements in wireless communications have propelled IoT into an extensive network that integrates seamlessly with the Internet [3]. This connectivity enables data-capturing devices such as radio frequency identification (RFID), infrared sensors, and laser scanners to support IoT's vision [9].

The application of IoT spans various sectors, including smart cities, automated homes, logistics, intelligent transportation, and healthcare. Notably, the healthcare domain, particularly mobile health (*mHealth*), has benefited significantly from IoT. *MHealth* leverages mobile and wireless devices to enhance medical practices, facilitating communication between healthcare providers and patients and enabling real-time health status monitoring [6]. However, the integration of IoT in healthcare raises concerns related to the security and privacy of patient data.

The primary goal of this study is to evaluate the performance of an IoT platform delivering *mHealth* services using the MQTT protocol. This research focuses on a comprehensive performance assessment of MQTT in *mHealth* contexts, which has been relatively underexplored. The objective is to understand how well the platform handles load conditions and supports reliable data transmission. The findings contribute to identifying the practical implications of using MQTT for *mHealth* and recognizing any limitations.

The remainder of this paper is structured as follows: Section 2 presents some interesting related works, while Section 3 outlines the experimental methodology used. The results are presented in Section 4, and the final considerations are given in Section 5.

2 Related Works

Research on communication protocols for IoT has focused on evaluating their performance, particularly in remote health monitoring (*mHealth*). The selection of an appropriate protocol is crucial for ensuring efficient, reliable, and secure transmission of patient data. While existing studies provide valuable insights into various protocols, gaps remain in comprehensive assessments specific to *mHealth*. Next, we present some related work (RW).

Thangavel et al. [10] conducted a comparative study of MQTT and CoAP, analyzing metrics such as overhead, message size, and packet loss rate under different network conditions. This work provided an understanding of the trade-offs between these protocols in terms of network efficiency.

Silva et al. [8] examined the performance of MQTT, CoAP, and OPC UA under different testing scenarios, highlighting the suitability of each protocol for various IoT applications. However, the focus was not specifically on *mHealth* environments.

Villanueva et al. [11] explored the use of MQTT for *mHealth* by comparing ECG signals from CardiaQloud with synthesized signals generated by the ECG Waveform Generator for Matlab from PhysioNet. This study offered insights into data transmission efficiency but did not provide a comprehensive performance analysis.

Ford et al. [5] investigated the impact of a Denial-of-Service (DoS) attack on the performance of MQTT when implemented on a Raspberry Pi, emphasizing security vulnerabilities in IoT environments. Although informative, this study did not focus on *mHealth*.

Oliveira et al. [7] compared the performance of WebSocket and MQTT through qualitative and quantitative analyses, identifying key performance differences. While relevant, their work did not address *mHealth* applications.

Araujo et al. [1] used high-level models such as Petri Nets and Reliability Block Diagrams (RBD) to simulate the dependability of an *mHealth* system. However theoretical, this work lacked practical performance evaluations.

This study aims to conduct a thorough performance evaluation of MQTT within an *mHealth* context. By employing real-world testing on an *Orange Pi Win Plus* board, our research provides a practical perspective on MQTT's capabilities and limitations in *mHealth* applications, contributing novel insights to the field. Table 1 summarizes the differences between this and the presented works.

Table 1 Summary of Related Works

RW	**Protocols**	***mHealth***	**Performance Evaluation**
[10]	MQTT, CoAP	No	Yes
[8]	MQTT, CoAP, OPC UA	No	Yes
[11]	MQTT	Yes	Yes
[5]	MQTT	No	Yes
[7]	MQTT, WebSocket	No	No
[1]	No	Yes	Yes
This work	MQTT	Yes	Yes

3 Materials and Methods

To achieve the objectives outlined in this study, the following hypotheses were established:

- **Hypothesis 1**: The MQTT protocol can handle standard and moderately high user loads efficiently in an *mHealth* context.
- **Hypothesis 2**: The platform's performance will show significant latency and resource consumption increases as the user load reaches an extreme level.

- **Hypothesis 3**: Higher Quality of Service (QoS) levels will provide increased reliability but will incur higher latency and resource usage.

We employed a structured methodology to validate these hypotheses, as depicted in Figure 1. This methodology encompasses a step-by-step process for defining the functional and non-functional requirements of an IoT platform aimed at providing *mHealth* services using the MQTT protocol.

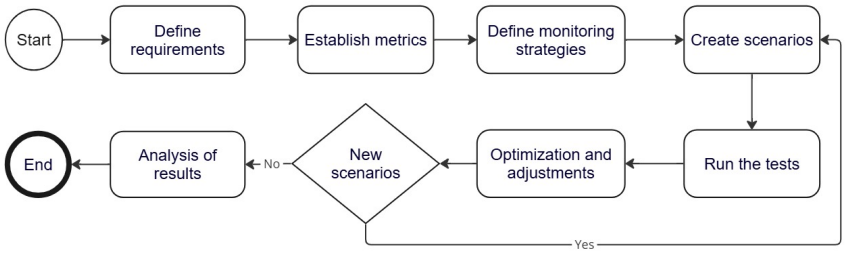


Fig. 1 Methodology flow

The process began with defining functional and non-functional requirements to establish the essential functionalities that the IoT platform must support. These functionalities include capabilities for collecting, storing, and sharing patient data in real time, enabling continuous remote patient monitoring. Additionally, non-functional requirements were specified to establish key performance attributes such as data security, scalability, and efficiency, all of which are critical for the effective implementation of *mHealth* services.

Next, we devised a monitoring strategy to assess the platform's performance under various conditions. Initial tests involved a limited number of connected users, simulating typical usage to evaluate the platform's performance under normal operating conditions. Subsequently, the number of users and their requests was gradually increased to determine the platform's saturation point. Finally, the platform was subjected to a high volume of simultaneous requests, designed to test its stability and efficiency under extreme overload conditions.

In the analysis of results phase, detailed reports were generated and retests were performed as needed. This iterative process allowed for the identification of potential areas for improvement, which could include infrastructure adjustments, configuration changes, or code optimizations. Implementing these improvements incrementally enabled a step-by-step assessment of their impact on the platform's performance.

This structured approach provided a comprehensive evaluation of the platform's ability to meet established requirements and handle different usage scenarios, ensuring alignment with research objectives and hypotheses.

3.1 Testbed Environment

For the experiment's development, both hardware and software were selected. The hardware used included an Orange Pi Win Plus IoT board, employed as the broker server, along with two notebooks, one acting as the publisher and the other as the subscriber. The Orange Pi Win Plus board is a compact and affordable device equipped with a quad-core Allwinner A64 processor, 2GB of RAM, and 64GB of internal storage.

Additionally, it features HDMI, USB 2.0, and USB 3.0 ports, along with IEEE 802.11 b/g/n Wi-Fi connectivity, making it ideal for IoT. The notebooks used were an Acer running Ubuntu 20.04 and a Lenovo running Linux Mint 19, both equipped with Intel Core i5 processors, 8 GB and 16 GB of RAM, respectively, as well as 1TB of storage.

The software used included Mosquitto, an MQTT message broker server, and JMeter, a performance testing tool. Mosquitto was employed as the MQTT broker server, while JMeter was used to simulate request loads and evaluate the performance of the IoT platform.

Mosquitto is an open-source software available for free download. It is compatible with a wide range of operating systems, including Linux, Windows, among others. Mosquitto is used as a message broker server responsible for implementing the MQTT protocol. It is a powerful tool widely used in a variety of IoT applications.

JMeter is also an open-source software capable of performing performance testing, load testing, and stress testing on systems. Conducting tests on systems is crucial to ensure their functionality, helping identify flaws and areas for improvement. Among the various types of tests applicable to a system, performance testing stands out, aiming to evaluate response capacity, reliability, throughput, interoperability, and scalability under a specific workload [4]. In this experiment, several metrics were employed based on theoretical foundations. However, the metrics that proved particularly relevant for this study were latency, throughput, scalability concerning increasing connected users, and resource efficiency (CPU and memory). These metrics provided valuable data for the analysis conducted in the experiment.

4 Results

This section presents a detailed analysis of the results obtained during the experiments, providing insights into the performance of the IoT platform under different conditions. The analysis is discussed concerning the objectives and hypotheses set out in the methodology. These insights contribute to understanding the platform's performance and advancing knowledge in mobile health (mHealth) and IoT systems.

4.1 CPU Usage Analysis

CPU performance is crucial for evaluating the platform's ability to manage concurrent user requests efficiently. In Scenario #1, where 400 users were simulated, CPU usage remained below 30% for most of the experiment (see Figures 2(a), 2(b) and 2(c)). This consistent performance aligns with Hypothesis 1, which suggested that the MQTT protocol would handle standard loads effectively. An isolated spike reaching approximately 60% was noted in the QoS 2 setup after 100 minutes (Figure 2(b)). This spike may be attributed to specific processing requirements or temporary system activity, demonstrating that while occasional peaks can occur, the system maintains stability under typical loads.

In Scenario #2 (800 users), CPU usage displayed more frequent peaks but remained below 30%, with a notable spike of 55% in QoS 0 after 130 minutes (see Figures 2(d), 2(e) and 2(f)). This result continues to support Hypothesis 1 by showing that the platform can handle moderate loads efficiently. It also indicates that even as the number of users doubled, the platform adapted without a significant decrease in performance.

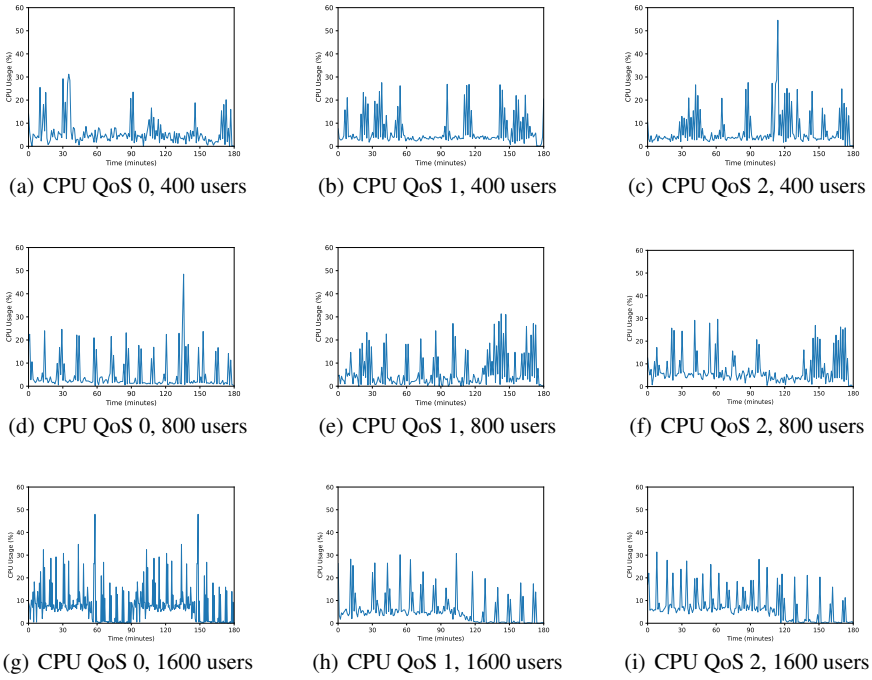
Scenario #3 (1600 users) revealed more complex behavior. CPU usage started between 15% and 20% but gradually increased to approach the 50% threshold (see Figures 2(g), 2(h) and 2(i)). Notably, a spike in QoS 0 around 170 minutes indicated additional processing needed to manage packet losses. This supports Hypothesis 2, which anticipated higher CPU resource usage as user loads increased. The overall trend suggests that while the platform is capable of scaling up, optimization is necessary for extreme load conditions.

4.2 Memory Usage Analysis

Memory management is essential in IoT platforms, especially in resource- constrained environments like mHealth. Scenario #1 (400 users) showed stable memory consumption between 700MB and 800MB (Figures 3(a), 3(b) and 3(c)), supporting Hypothesis 1. This result indicates that the platform managed memory efficiently, maintaining steady performance under standard load conditions.

In Scenario #2 (800 users), a different pattern emerged. While QoS 0 showed a peak of up to 1100MB (Figure 3(d)), QoS 1 and QoS 2 maintained stability between 700MB and 800MB (Figures 3(e) and 3(f)). The increase in QoS 0 could be linked to the absence of message delivery guarantees, which leads to resource allocation for handling discarded messages and potential packet loss.

Scenario #3 highlighted memory usage challenges as user loads reached 1600. In QoS 0, memory consumption fluctuated between 800MB and 1400MB (Figure 3(g)), reflecting higher demands on system resources as anticipated in Hypothesis 2. QoS 1 ranged between 700MB and 1100MB, while QoS 2 remained more stable at 700MB to 1000MB (Figures 3(h) and 3(i)). The observed variations underline the



(a) CPU QoS 0, 400 users (b) CPU QoS 1, 400 users (c) CPU QoS 2, 400 users

(d) CPU QoS 0, 800 users (e) CPU QoS 1, 800 users (f) CPU QoS 2, 800 users

(g) CPU QoS 0, 1600 users (h) CPU QoS 1, 1600 users (i) CPU QoS 2, 1600 users

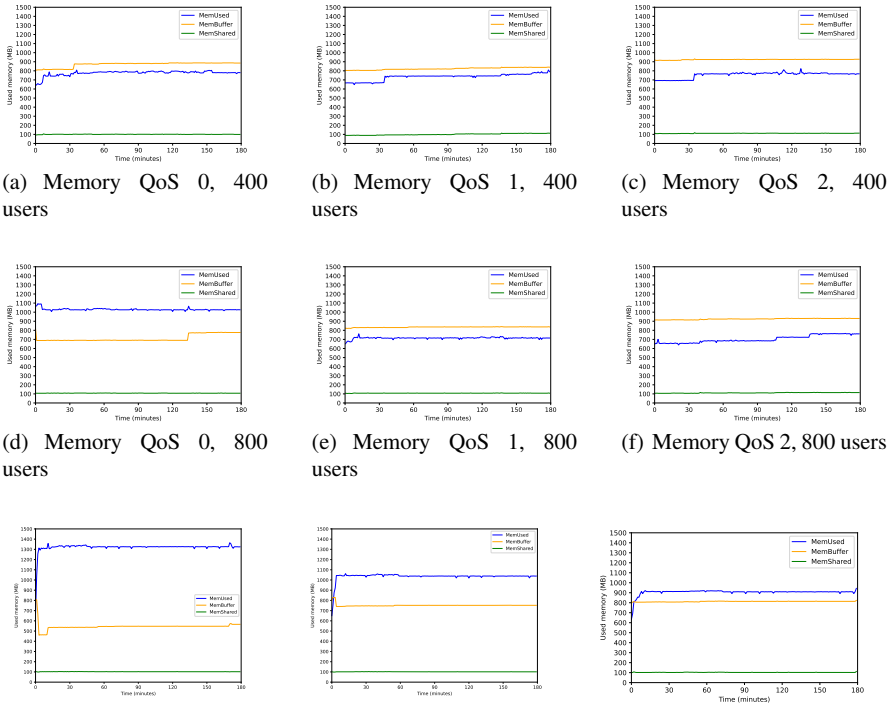
Fig. 2 Broker CPU Usage Across Different Scenarios and QoS Levels

impact of different QoS settings on memory usage, emphasizing the importance of resource management for high-load scenarios.

4.3 Throughput Analysis

Throughput, measured in bytes transmitted over time, is a critical performance indicator for evaluating data delivery in real-time IoT applications. Scenario #1 demonstrated relatively stable byte consumption, with QoS 0 ranging between 2.5MB and 5MB (Figure 4(a)), while QoS 1 showed greater variability, oscillating between 1MB and 6MB (Figure 4(b)). QoS 2 remained between 3.5MB and 6MB (Figure 4(c)), supporting Hypothesis 3 that higher QoS levels influence data throughput.

In Scenario #2 (800 users), byte consumption showed significant fluctuations, especially in QoS 1 and 2. QoS 1 reached peaks of up to 10MB (Figure 4(e)), indicating the impact of intermediate delivery guarantees. QoS 2 displayed erratic behavior, stabilizing after initial spikes (Figure 4(f)). This behavior aligns with Hypothesis 3, which suggested that higher QoS levels would increase resource demands.



(a) Memory QoS 0, 400 users (b) Memory QoS 1, 400 users (c) Memory QoS 2, 400 users

(d) Memory QoS 0, 800 users (e) Memory QoS 1, 800 users (f) Memory QoS 2, 800 users

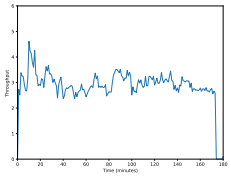
(g) Memory QoS 0, 1600 users (h) Memory QoS 1, 1600 users (i) Memory QoS 2, 1600 users

Fig. 3 Broker Memory Usage Across Different Scenarios and QoS Levels

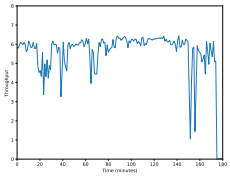
Scenario #3 revealed pronounced byte rate oscillations in all QoS levels, with initial spikes up to 14MB followed by stabilization (Figures 4(g), 4(h) and 4(i)). This indicates that high user loads and intensive data transmission challenge the platform's throughput, supporting Hypothesis 2 on the increased strain under heavy workloads.

4.4 Latency Analysis

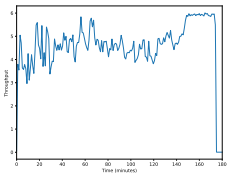
Latency is critical for real-time monitoring in mHealth. Scenario #1 showed low latency in QoS 0, staying below 1 millisecond (Figure 5(a)). QoS 1 and 2 showed more pronounced spikes, reaching up to 5 milliseconds and fluctuating between 500 and 1750 milliseconds, respectively (Figures 5(b) and 5(c)). These findings partially confirm Hypothesis 3, showing that higher QoS levels introduce latency.



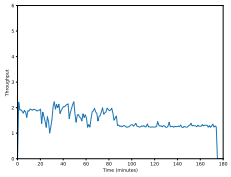
(a) Throughput QoS 0, 400 users



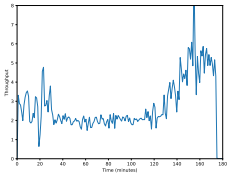
(b) Throughput QoS 1, 400 users



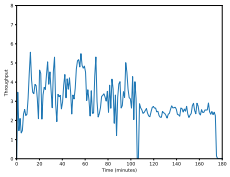
(c) Throughput QoS 2, 400 users



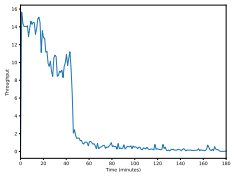
(d) Throughput QoS 0, 800 users



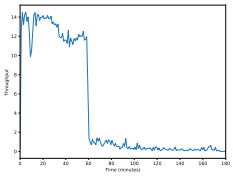
(e) Throughput QoS 1, 800 users



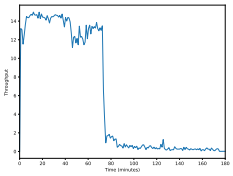
(f) Throughput QoS 2, 800 users



(g) Throughput QoS 0, 1600 users



(h) Throughput QoS 1, 1600 users

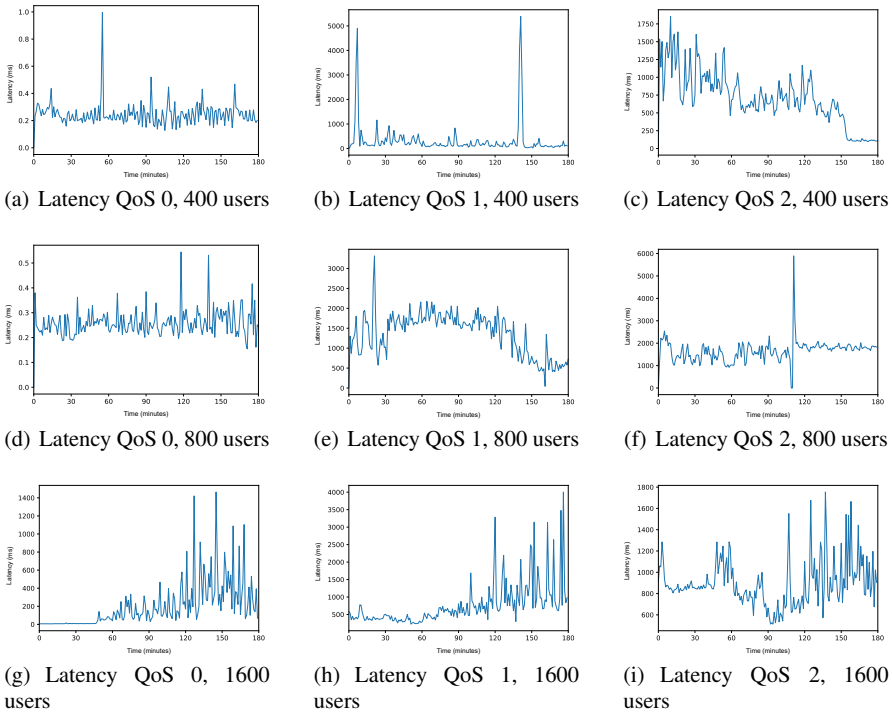


(i) Throughput QoS 2, 1600 users

Fig. 4 Broker Throughput Across Different Scenarios and QoS Levels

Scenario #2 (800 users) presented latency variations, with QoS 0 remaining stable under 1 millisecond (Figure 5(d)). In QoS 1 and 2, peaks reached up to 4000 milliseconds (Figures 5(e) and 5(f)), suggesting that higher loads challenge the platform's response time.

In Scenario #3, latency increased significantly, particularly in QoS 1 and 2 (Figures 5(h) and 5(i)). QoS 0 showed less consistent behavior, with spikes observed at 1400 milliseconds. This aligns with Hypothesis 2, which indicates that latency increases with user load and the QoS level, confirming that while QoS ensures reliability, it also introduces performance trade-offs.



(a) Latency QoS 0, 400 users (b) Latency QoS 1, 400 users (c) Latency QoS 2, 400 users

(d) Latency QoS 0, 800 users (e) Latency QoS 1, 800 users (f) Latency QoS 2, 800 users

(g) Latency QoS 0, 1600 users (h) Latency QoS 1, 1600 users (i) Latency QoS 2, 1600 users

Fig. 5 Broker Latency Across Different Scenarios and QoS Levels

4.5 DoE Analysis

The Design of Experiments (DoE) analysis is a statistical approach to planning, executing, and analyzing experiments to efficiently obtain relevant information. For the DoE analysis, we used data from all variables analyzed in the experiment.

The first step taken was to generate the Pareto chart, as shown in Figure 6, which identifies which factor shows a significant effect. We can be observed that the scenario is the factor that has the greatest impact. The Pareto chart confirmed that the Scenario, CPU usage and QoS significantly affect latency.

Based on this initial analysis, it becomes necessary to investigate the main effects. Therefore, we identified that the Scenario, CPU, and QoS are the factors that have a significant impact on average latency, with latency tending to increase as these resources decrease, as shown in Figure 7.

Complementing the previous analyses, another analysis was conducted. The interaction plot (Figure 8) highlighted interactions between CPU, memory, and QoS, indicating that higher resource levels contribute to lower latency, supporting Hypothesis 3. In the first column, representing the interactions between QoS and other metrics. As the scenario increases, the average latency decreases, indicating the ab-

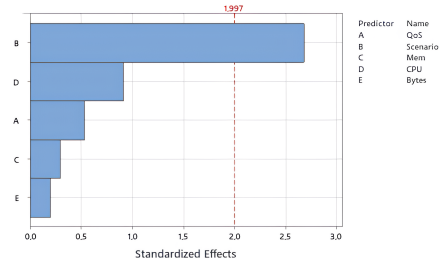


Fig. 6 Standardized effects Pareto Chart - Latency

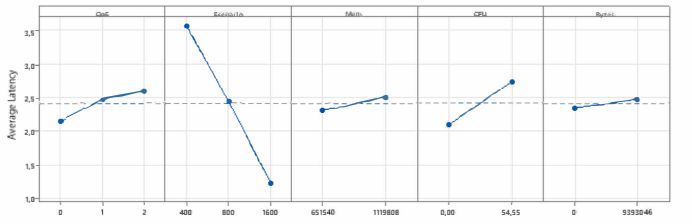


Fig. 7 Main Effects Chart

sence of interaction between the factors and metrics. In the third column, between memory (Mem) and CPU, a possible interaction is suggested. This occurs because the average latency appears to be lower when memory and CPU are at their highest levels, while the average latency seems to be higher when memory and CPU are at their lowest levels.

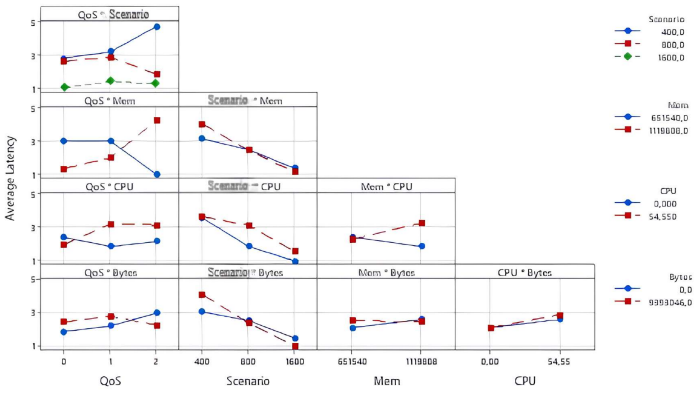


Fig. 8 Interaction Chart for Latency

5 Final Remarks

The main objective of this study was to perform a performance evaluation of an IoT platform designed for the mobile health (mHealth) environment. To achieve this goal, we simulated stress scenarios using a suitable platform for such simulations. The intention was to ensure efficiency in response and scalability of the IoT platform in delivering mHealth services.

Throughout the study, we observed the importance of evaluating the platform's performance under different workload conditions. The experiments provided valuable insights into how the platform behaves under different levels of demand, highlighting strengths and identifying possible areas for improvement. This finding underscores the importance of appropriately choosing parameters, such as quality of service (QoS) level, in implementing IoT systems for the healthcare sector. Processing capacity and effective message management are important aspects to ensure operational efficiency and system reliability, especially when considering a significant number of simultaneous users.

This study has significantly contributed to advancing knowledge in the field of IoT applied to mHealth using the MQTT protocol. The practical application of this study can guide future developments and optimizations in similar systems, providing clear improvements in the quality of service (QoS) offered. Consequently, the conclusions drawn from this study will contribute both to the selected IoT infrastructure and the adopted MQTT protocol, serving as viable options for offering mHealth services.

Concluding the analysis of the IoT system performance in the mHealth environment using the MQTT protocol in various scenarios, significant conclusions can be drawn that directly impact the efficiency and scalability of this system. Among the scenarios studied, the use of QoS 1 with 400 users stands out, demonstrating the best performance, with consistently low latency of less than 1000 milliseconds, i.e., below 1 second throughout the experiment.

Despite the achievements obtained, it is pertinent to note that this study is not without limitations. The adoption of a single IoT infrastructure, along with the simulation of test scenarios in a controlled environment, is one of these limitations. In this context, an important suggestion is to conduct tests in real environments, considering various IoT infrastructures and varied usage scenarios.

References

1. Araujo, J., Silva, B., Oliveira, D., Maciel, P.: Dependability evaluation of a mhealth system using a mobile cloud infrastructure. In: 2014 IEEE international conference on systems, man, and cybernetics (SMC), pp. 1348–1353. IEEE (2014)
2. Ashton, K.: That 'internet of things' thing. RFID journal **22**(7), 97–114 (2009)
3. Atzori, L., Iera, A., Morabito, G., et al.: The internet of things: A survey. Computer networks **54**(15), 2787–2805 (2010)
4. Erinle, B.: Performance testing with JMeter 2.9. Packt Publishing Ltd (2013)

5. Ford, T.N., Gamess, E., Ogden, C.: Performance evaluation of different raspberry pi models as mqttservers and clients. International Journal of Computer Networks and Communications **14**(2) (2022)
6. Kumar, D., Gubbi, J., Yan, B., Palaniswami, M.: Motor recovery monitoring in post acute stroke patients using wireless accelerometer and cross-correlation. IEEE Access pp. 6703–6706 (2013)
7. Oliveira, G.M.B., Costa, D.C.M., Cavalcanti, R.J.B.V.M., Oliveira, J.P.P., Silva, D.R.C., Nogueira, M.B., Rodrigues, M.C.: Comparison between mqtt and websocket protocols for iot applications using esp8266. In: 2018 Workshop on Metrology for Industry 4.0 and IoT, pp. 236–241 (2018). DOI 10.1109/METROI4.2018.8428348
8. Silva, D., Carvalho, L.I., Soares, J., Sofia, R.C.: A performance analysis of internet of things networking protocols: Evaluating mqtt, coap, opc ua. Applied Sciences **11**(11), 4879 (2021)
9. Stankovic, J.A.: Research directions for the internet of things. IEEE internet of things journal **1**(1), 3–9 (2014)
10. Thangavel, D., Ma, X., Valera, A., Tan, H.X., Tan, C.K.Y.: 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), pp. 1–6. IEEE (2014)
11. Villanueva-Miranda, I., Nazeran, H., Martinek, R., et al.: Cardiaqloud: A remote ecg monitoring system using cloud services for ehealth and mhealth applications. In: 2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom), pp. 1–6 (2018)